User's Manual

Renesas Flexible Software Package (FSP) v3.5.0

User's Manual

Renesas RA Family

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Revision 2.50 Dec.3.2021

Table of Contents

Chapter 1 Introduction

1.1 Overview

This manual describes how to use the Renesas Flexible Software Package (FSP) for writing applications for the RA microcontroller series.

1.2 Using this Manual

This manual provides a wide variety of information, so it can be helpful to know where to start. Here is a short description of each main section and how they can be used.

Starting Development - Provides a step by step guide on how to use e2 studio and FSP to develop a project for RA MCUs. This is a good place to start to get up to speed quickly and efficiently.

FSP Architecture - Provides useful background material on key FSP concepts such as Modules, Stacks, and API standards. Reference this section to extend or refresh your knowledge of FSP concepts.

API Reference - Provides detailed information on each module and interface including features, API functions, configuration settings, usage notes, function prototypes and code examples. Board Support Package (BSP) related API functions are also included.

Note

Much of the information in the API Reference section is available from within the e2 studio tool via the Developer Assistance feature. The information here can be referenced for additional details on API features.

1.3 Documentation Standard

Each Modules section user guide outlines the following:

- Features: A bullet list of high level features provided by the module.
- Configuration: A description of module specific configurations available in the RA Configuration editor.
- Usage Notes: Module specific documentation and limitations.
- Examples: Example code provided to help the user get started.
- API Reference: Usage notes for each API in the module, including the function prototype and hyperlinks to the interface documentation for parameter definitions.

Each Interfaces section user guide outlines the following:

- Detailed Description: A short description and summary of the interface functionality.
- Data Structures: A list and definition of each data structure used by the interface including the structure of the pointers that define the API and are shared by all modules that implement the interface.
- Typedefs: A list and description of the typedefs used by the interface.
- Enumerations: A list and description of the enumerations used by the interface.

1.4 Introduction to FSP

1.4.1 Purpose

The Renesas Flexible Software Package (FSP) is an optimized software package designed to provide easy to use, scalable, high quality software for embedded system design. The primary goal is to provide lightweight, efficient drivers that meet common use cases in embedded systems.

1.4.2 Quality

FSP code quality is enforced by peer reviews, automated requirements-based testing, and automated static analysis.

1.4.3 Ease of Use

FSP provides uniform and intuitive APIs that are well documented. Each module is supported with detailed user documentation including example code.

1.4.4 Scalability

FSP modules can be used on any MCU in the RA family, provided the MCU has any peripherals required by the module.

1.4.5 Build Time Configurations

FSP modules also have build time configurations that can be used to optimize the size of the module for the feature set required by the application.

1.4.6 e2 studio IDE

FSP provides a host of efficiency enhancing tools for developing projects targeting the Renesas RA series of MCU devices. The e2 studio IDE provides a familiar development cockpit from which the key steps of project creation, module selection and configuration, code development, code generation, and debugging are all managed.

Chapter 2 Starting Development

2.1 Starting Development Introduction

The wealth of resources available to learn about and use e2 studio and FSP can be overwhelming on first inspection, so this section provides a Starting Development Guide with a list of the most important initial steps. Following these highly recommended first 11 steps will bring you up to speed on the development environment in record time. Even experienced developers can benefit from the use of this guide, to learn the terminology that might be unfamiliar or different from previous environments.

1. Read the section What is e2 studio?, up to but not including e2 studio Prerequisites. This will provide a description of the various windows and views to use e2 studio to create a project, add modules and threads, configure module properties, add code, and debug a project. It also describes how to use key coding 'accelerators' like Developer Assist (to drag and drop parameter populated API function calls right into your code), a context aware Autocomplete (to easily find and select from suggested enumerations, functions, types, and many other coding elements), and many other similar productivity enhancers.
2. Read the FSP Architecture, FSP Modules and FSP Stacks sections. These provide the basic background on how FSP modules and stacks are used to construct your application. Understanding their definitions and the theory behind how they combine will make it easier to develop with FSP.
3. Read a few Modules sections to see how to use API function calls, structures, enumerations, types and callbacks. These module guides provide the information you will use to implement your project code.
4. After you have a Kit and you have downloaded and installed e2 studio and FSP, you can build and debug a simple project to test your installation, tool flow, and the kit. (If you do not have a Kit or have not yet installed the development software, use the links included in the e2 studio Prerequisites for more information.) The simple Tutorial: Your First RA MCU Project - Blinky will Blink an LED on and off. Follow the instructions for importing and running this project in section Create a New Project for Blinky. It will use some of the key steps for managing projects within e2 studio and is a good way to learn the basics.
5. Once you have successfully run Blinky you have a good starting point for using FSP for more complex projects. The Using HAL Drivers Tutorial, available at Tutorial: Using HAL Drivers - Programming the WDT, shows how to create a project from scratch, using FSP API functions. Do this next.
6. Several Hands-on Quick FSP Labs are available that cover key development topics with short 15-minute Do it Yourself (DiY) activities targeting the EK-RA6M3. Topics covered include code development accelerators like Developer Assistance, Autocomplete, Help, Visual Expressions and using Example Projects. The complete list of available Quick FSP Labs can be found here: https://en-support.renesas.com/knowledgeBase/category/31087/subcategory/31090. Doing a couple of these labs provides further details on using FSP, and is also good practice. Running these labs is highly recommended.
7. The balance of the FSP Architecture sections (that is, those not called out in step 2 above) contain additional reference material that may be helpful in the future. Scan them so you know what they contain, in case you need them.
8. The balance of the e2 studio User Guide, starting with the What is a Project? section up to, but not including, Writing the Application section, provides a detailed description of each of

the key steps, windows, and entries used to create, manage, configure, build and debug a project. Much of this may be familiar after running through the tutorials and Quick Labs. However, it is important to have a good grasp of what each of the configuration tabs are used for as that is where the bulk of the project preparation work takes place prior to writing code. Skim over this section as it may help with any questions in the future.

9. Read the Writing the Application section to get a short introduction to the steps used when creating application code with FSP. It covers both RTOS-independent and RTOS-dependent applications. It also includes a short description for several of the code accelerators you should be familiar with by now. Using additional Quick FSP Labs is a good way to become familiar with the application development process and links to them are included in the appropriate places in this section. You can find the complete list of available Quick FSP Labs here: https://en-support.renesas.com/knowledgeBase/19308277.
10. Scan the Debugging the Project section to see the steps required to download and start a debug session.
11. Explore the additional material available on the following web pages and bookmark the resources that look most valuable to you:
    a. RA Landing Page: https://www.renesas.com/ra
    b. FSP Landing Page: https://www.renesas.com/fsp
    c. Example Projects on GitHub: https://github.com/renesas/ra-fsp-examples
    d. Quick FSP Labs Listing: https://en-support.renesas.com/knowledgeBase/19308277
    e. RA and FSP Knowledge Base (with articles of interest on RA and FSP): https://en-support.renesas.com/knowledgeBase/category/31087
    f. RA and FSP Renesas Rulz site (Community posted and answered questions): https://renesasrulz.com/ra/
    g. FSP Releases: https://github.com/renesas/fsp/releases
    h. FSP Documentation: https://renesas.github.io/fsp
    i. Online Technical Support: https://www.renesas.com/us/en/support/contact.html

2.2 e2 studio User Guide

2.2.1 What is e2 studio?

Renesas e2 studio is a development tool encompassing code development, build, and debug. e2 studio is based on the open-source Eclipse IDE and the associated C/C++ Development Tooling (CDT).

When developing for RA MCUs, e2 studio hosts the Renesas Flexible Software Package (FSP). FSP provides a wide range of time saving tools to simplify the selection, configuration, and management of modules and threads, to easily implement complex applications. The time saving tools available in e2 studio and FSP include the following:

- A Graphical User Interface (GUI) (see Adding Threads and Drivers) with numerous wizards for configuring and auto-generating code
- A context sensitive Autocomplete (see Tutorial: Using HAL Drivers - Programming the WDT) feature that provides intelligent options for completing a programming element
- A Developer Assistance tool for selection of and drag and drop placement of API functions directly in application code
- A Welcome Window with links to example projects, application notes and a variety of other self-help support resources
- An Information Icon from each module is provided in the graphic configuration viewer that links to specific design resources, including code 'cheat sheets' that provide useful starting points for common application implementations.

Figure 1: e2 studio Splash Screen

e2 studio organizes project work based on Perspectives, Views, Windows, Panes, and Pages (sometimes called Tabs). A window is a section of the e2 studio GUI that presents information on a key topic. Windows often use tabs to select sub-topics. For example, an editor window might have a tab available for each open file, so it is easy to switch back and forth between them. A window Pane is a section of a window. Within a window, multiple Panes can be opened and viewed simultaneously, as opposed to a tabbed window, where only individual content is displayed. A memory-display Window, for example, might have multiple Panes that allow the data to be displayed in different formats, simultaneously. A Perspective is a collection of Views and Windows typical for a specific stage of development. The default perspectives are a C/C++ Perspective, an FSP Configuration Perspective and a Debug Perspective. These provide specific Views, Windows, Tabs, and Panes tailored for the common tasks needed during the specific development stage.

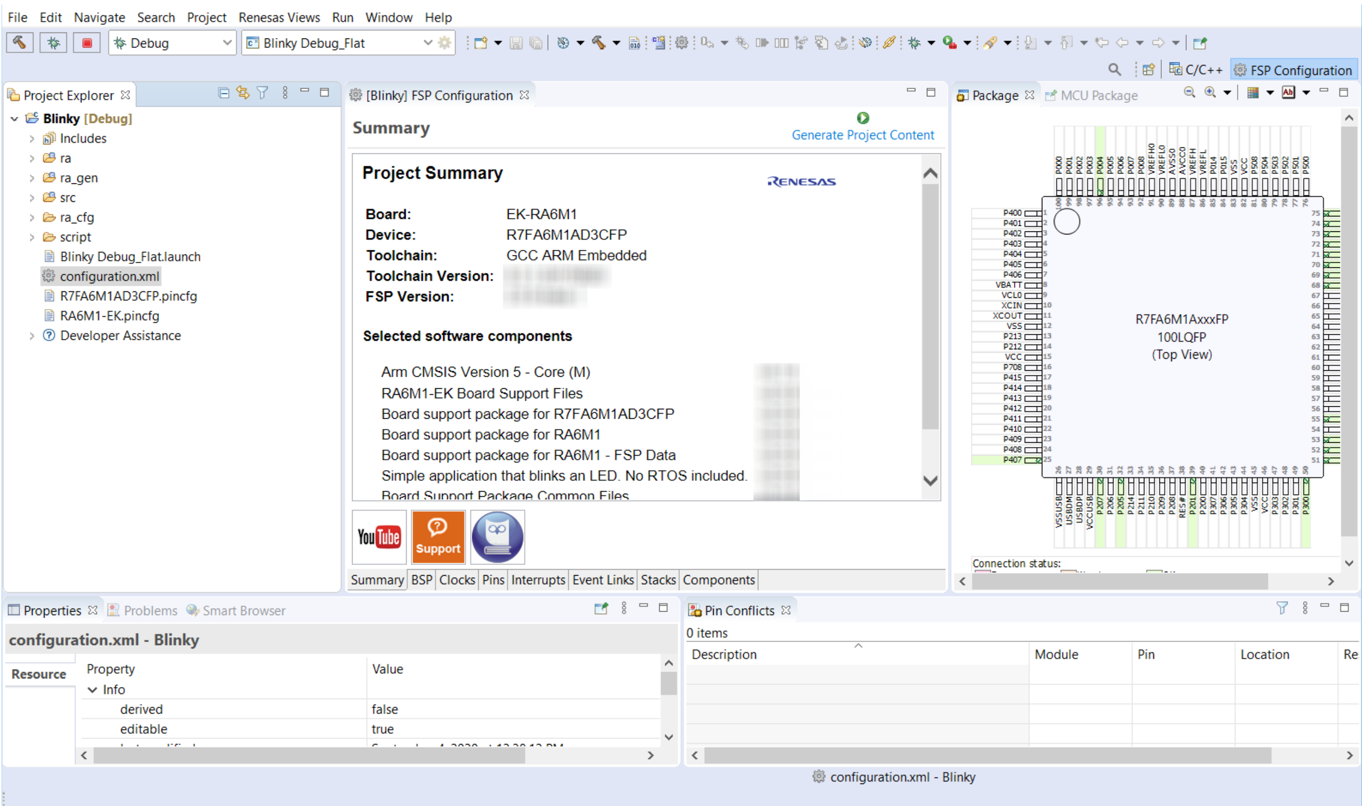


Figure 2: Default Perspective

In addition to managing project development, selecting modules, configuring them and simplifying code development, e2 studio also hosts the engine for automatically generating code based on module selections and configurations. The engine continually checks for dependencies and automatically adds any needed lower level modules to the module stack. It also identifies any lower level modules that require configuration (for example, an interrupt that needs to have a priority assigned). It also provides a guide for selecting between multiple choices or options to make it easy to complete a fully functional module stack.

The Generate Project Content function takes the selected and configured modules and automatically generates the complete and correct configuration code. The code is added to the folders visible in the **Project Explorer** window in e2 studio. The configuration.xml file in the project folder holds all the generated configuration settings. This file can be opened in the GUI-based RA Configuration editor to make further edits and changes. Once a project has been generated, you can go back and reconfigure any of the modules and settings if required using this editor.

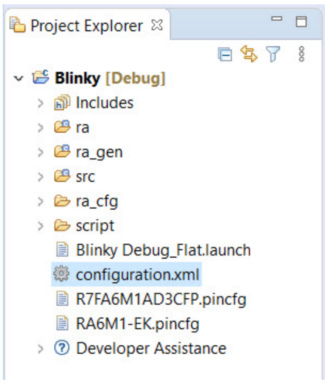


Figure 3: Project Explorer Window showing generated folders and configuration.xml file

2.2.2 e2 studio Prerequisites

2.2.2.1 Obtaining an RA MCU Kit

To develop applications with FSP, start with one of the Renesas RA MCU Evaluation Kits. The Renesas RA MCU Evaluation Kits are designed to seamlessly integrate with the e2 studio.

Ordering information, Quick Start Guides, User Manuals, and other related documents for all RA MCU Evaluation Kits are available at https://www.renesas.com/ra.

2.2.2.2 PC Requirements

The following are the minimum PC requirements to use e2 studio:

- Windows 10 with Intel i5 or i7, or AMD A10-7850K or FX
- Memory: 8-GB DDR3 or DDR4 DRAM (16-GB DDR4/2400-MHz RAM is preferred)
- Minimum 250-GB hard disk

2.2.2.3 Installing e2 studio, platform installer and the FSP package

Detailed installation instructions for the e2 studio and the FSP are available on the Renesas website https://www.renesas.com/fsp. Review the release notes for e2 studio to ensure that the e2 studio version supports the selected FSP version. The starting version of the installer includes all features of the RA MCUs.

2.2.2.4 Choosing a Toolchain

e2 studio can work with several toolchains and toolchain versions such as the GNU Arm compiler and Arm AC6. A version of the GNU Arm compiler is included in the e2 studio installer and has been verified to run with the FSP version.

2.2.2.5 Licensing

FSP licensing includes full source code, limited to Renesas hardware only.

2.2.3 What is a Project?

In e2 studio, all FSP applications are organized in RA MCU projects. Setting up an RA MCU project involves:

1. Creating a Project
2. Configuring a Project

These steps are described in detail in the next two sections. When you have existing projects already, after you launch e2 studio and select a workspace, all projects previously saved in the selected workspace are loaded and displayed in the **Project Explorer** window. Each project has an associated configuration file named configuration.xml, which is located in the project's root directory.

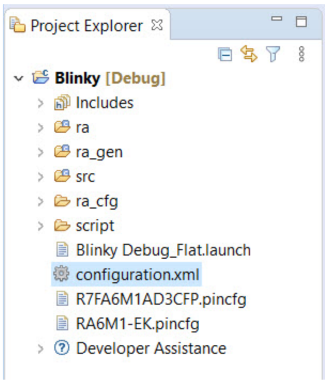


Figure 4: e2 studio Project Configuration file

Double-click on the configuration.xml file to open the RA MCU Project Editor. To edit the project configuration, make sure that the **FSP Configuration** perspective is selected in the upper right hand corner of the e2 studio window. Once selected, you can use the editor to view or modify the configuration settings associated with this project.

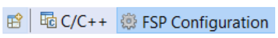


Figure 5: e2 studio FSP Configuration Perspective

Note

Whenever the RA project configuration (that is, the configuration.xml file) is saved, a verbose RA Project Report file (ra_cfg.txt) with all the project settings is generated. The format allows differences to be easily viewed using a text comparison tool. The generated file is located in the project root directory.

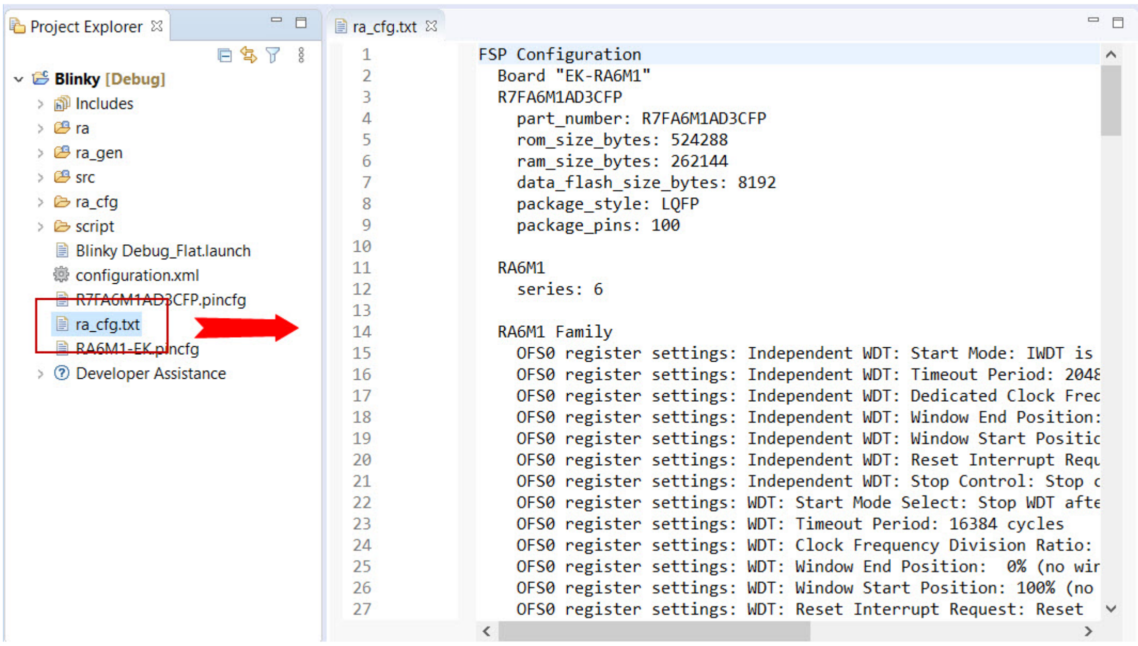


Figure 6: RA Project Report

The RA Project Editor has a number of tabs. The configuration steps and options for individual tabs are discussed in the following sections.

Note

The tabs available in the RA Project Editor depend on the e2 studio version and the layout may vary slightly, however the functionality should be easy to follow..

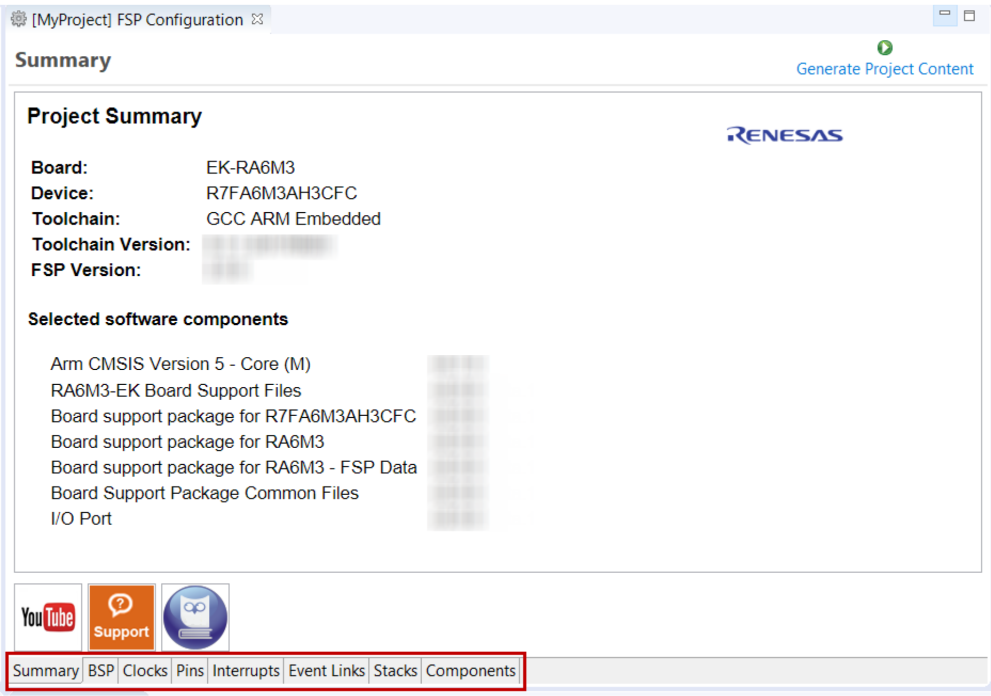


Figure 7: RA Project Editor tabs

- Click on the YouTube icon to visit the Renesas FSP playlist on YouTube
- Click on the Support icon to visit RA support pages at Renesas.com
- Click on the user manual (owl) icon to open the RA software package User's Manual

2.2.4 Creating a Project

During project creation, you specify the type of project, give it a project name and location, and configure the project settings for version, target board, whether an RTOS is included, the toolchain version, and the beginning template. This section includes easy-to-follow step-by-step instructions for all of the project creation tasks. Once you have created the project, you can move to configuring the project hardware (clocks, pins, interrupts) and the parameters of all the modules that are part of your application.

2.2.4.1 Creating a New Project

For RA MCU applications, generate a new project using the following steps:

1. Click on **File > New > RA C/C++ Project**.

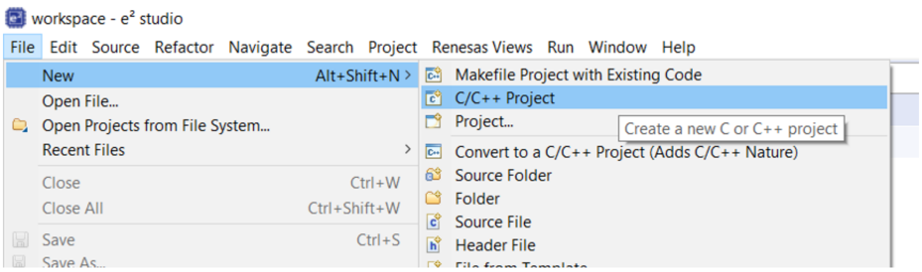


Figure 8: New RA MCU Project

Then click on the type of template for the type of project you are creating.

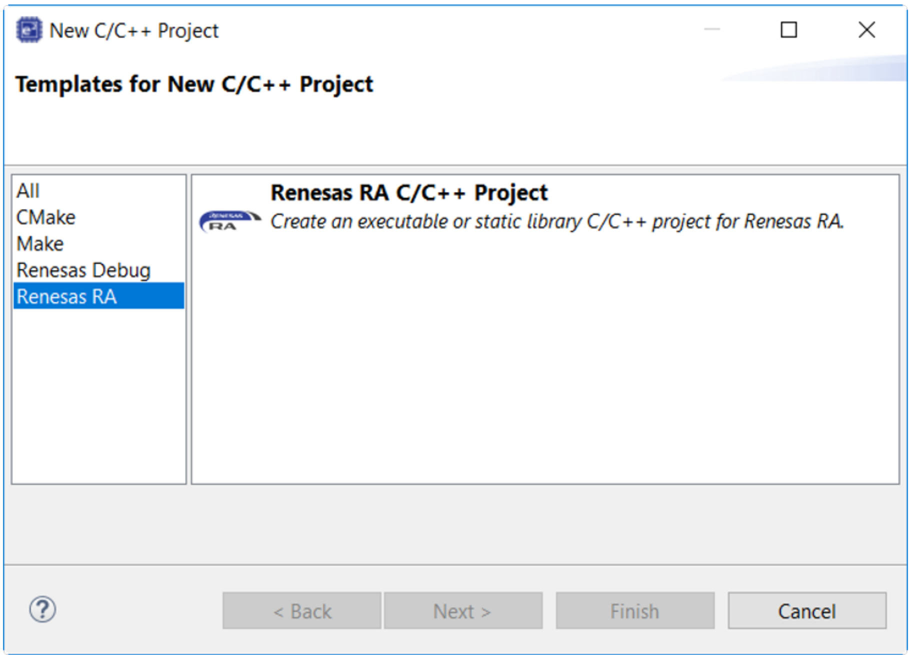


Figure 9: New Project Templates

2. Select a project name and location.

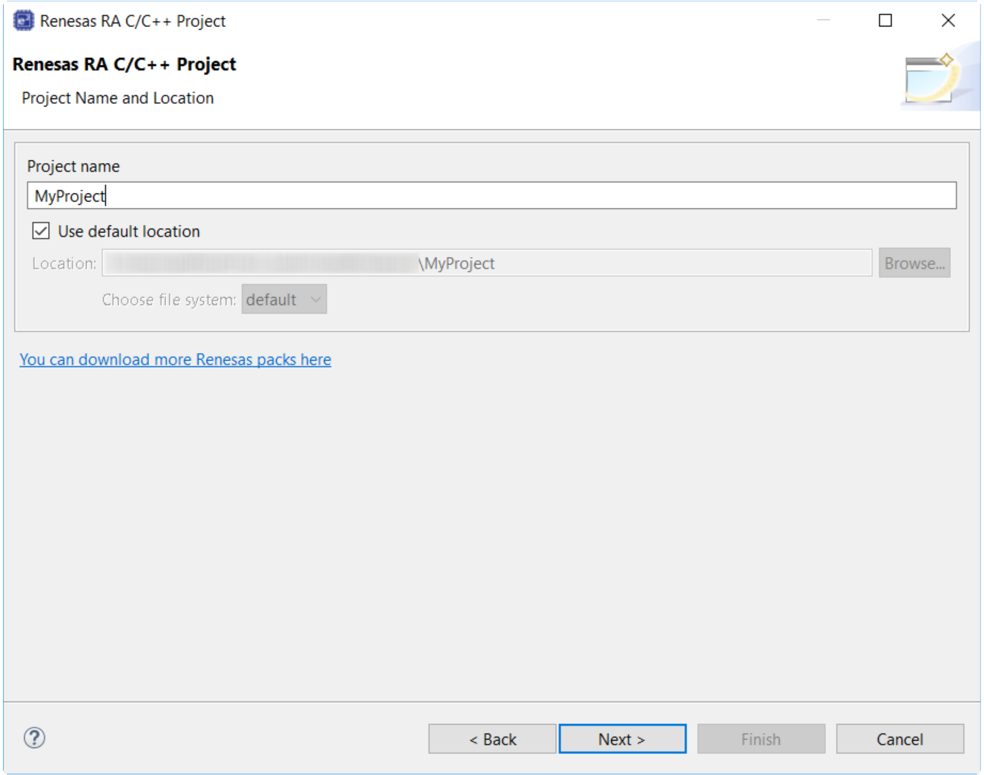


Figure 10: RA MCU Project Generator (Screen 1)

3. Click **Next**.

2.2.4.2 Selecting a Board and Toolchain

In the **Project Configuration** window select the hardware and software environment:

1. Select the **FSP version**.
2. Select the **Board** for your application. You can select an existing RA MCU Evaluation Kit or select **Custom User Board** for any of the RA MCU devices with your own BSP definition.
3. Select the **Device**. The **Device** is automatically populated based on the **Board** selection. Only change the **Device** when using the **Custom User Board (Any Device)** board selection.
4. To add threads, select **RTOS**, or **No RTOS** if an RTOS is not being used.
5. The **Toolchain** selection defaults to **GCC Arm Embedded**.
6. Select the **Toolchain version**. This should default to the installed toolchain version.
7. Select the **Debugger**. The J-Link Arm Debugger is preselected.

8. Click **Next**.

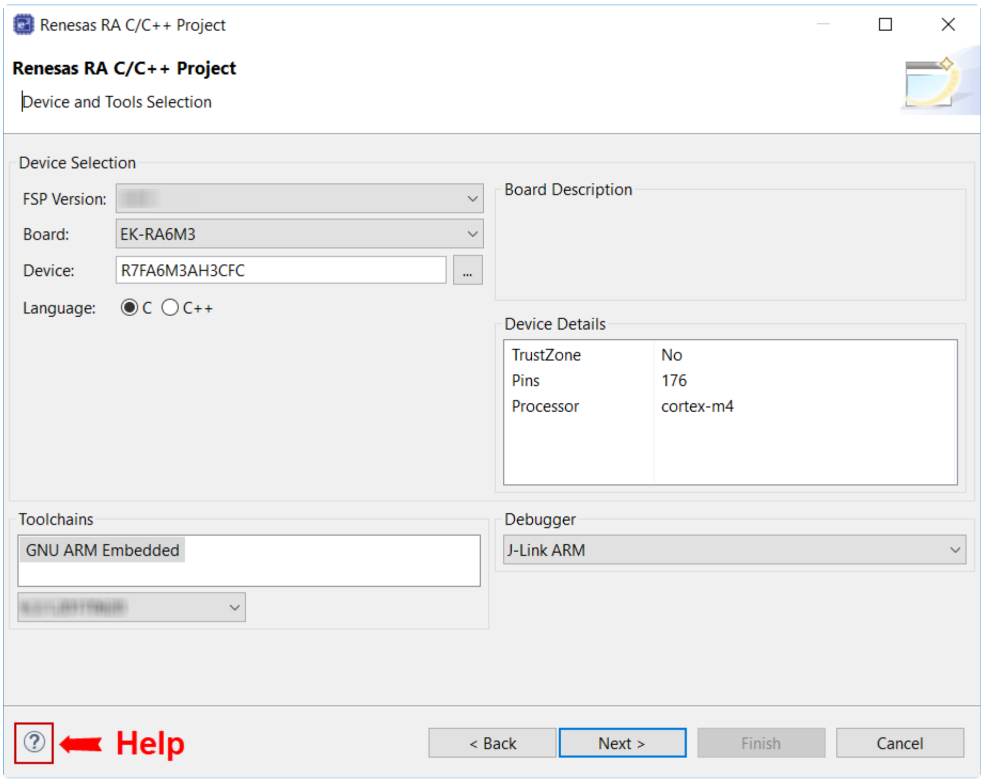


Figure 11: RA MCU Project Generator (Screen 2)

Note

Click on the ***Help*** *icon (?) for user guides, RA contents, and other documents.*

2.2.4.3 Selecting Flat or Arm® TrustZone® Project

If you selected a device or tool based on an Arm® Cortex®-M33, you next select whether to use Arm® TrustZone® in your project. For normal, non-TrustZone projects, select "Flat".

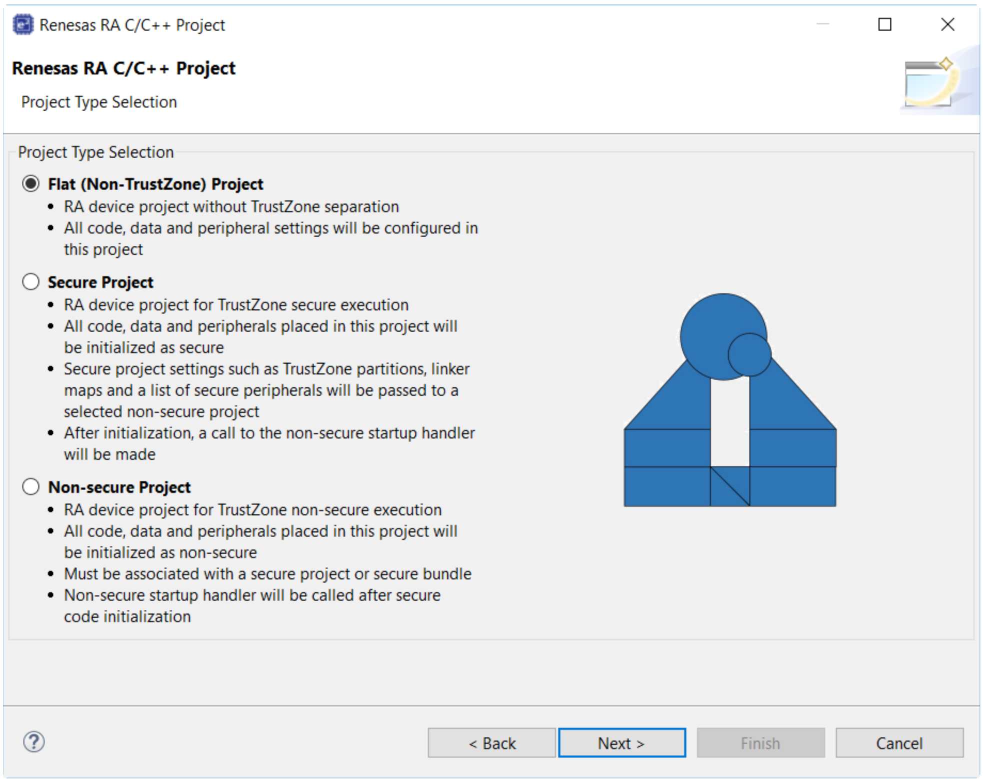


Figure 12: Flat, Secure, or Non-Secure Project

For more information on Arm® TrustZone®, see Primer: ARM® TrustZone® Project Development.

2.2.4.4 Selecting a Project Template

In the next window, select the buiild artifact and RTOS.

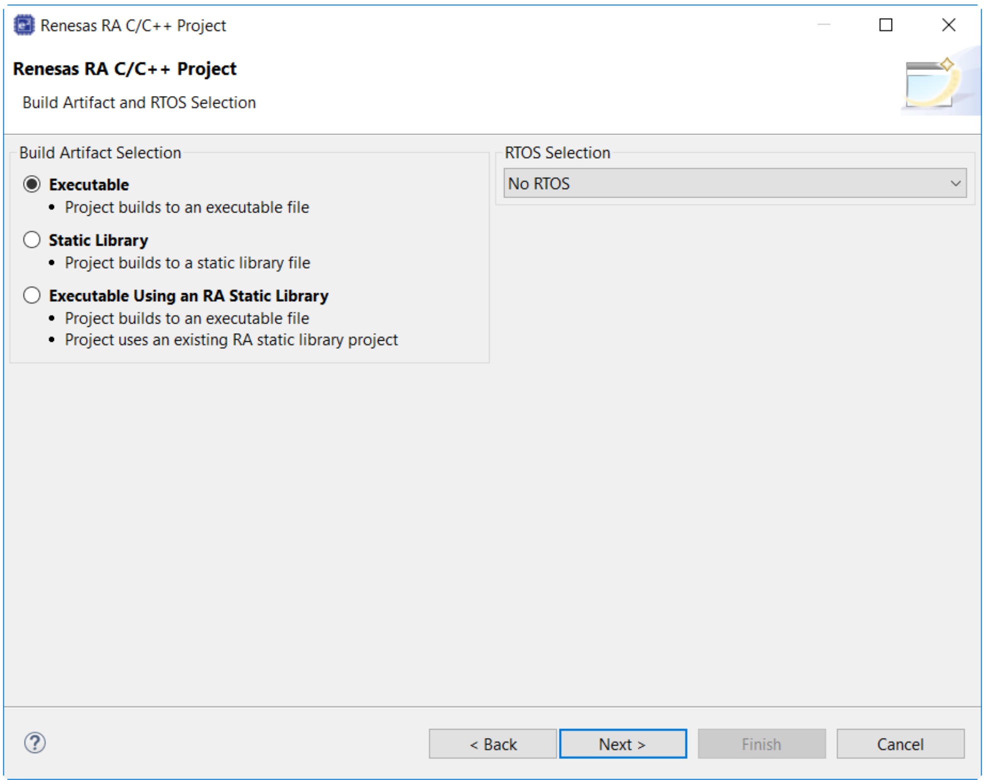


Figure 13: RA MCU Project Generator (Screen 3)

In the next window, select a project template from the list of available templates. By default, this screen shows the templates that are included in your current RA MCU pack. Once you have selected the appropriate template, click **Finish**.

Note

*If you want to develop your own application, select the basic template for your board, **Bare Metal - Minimal**.*

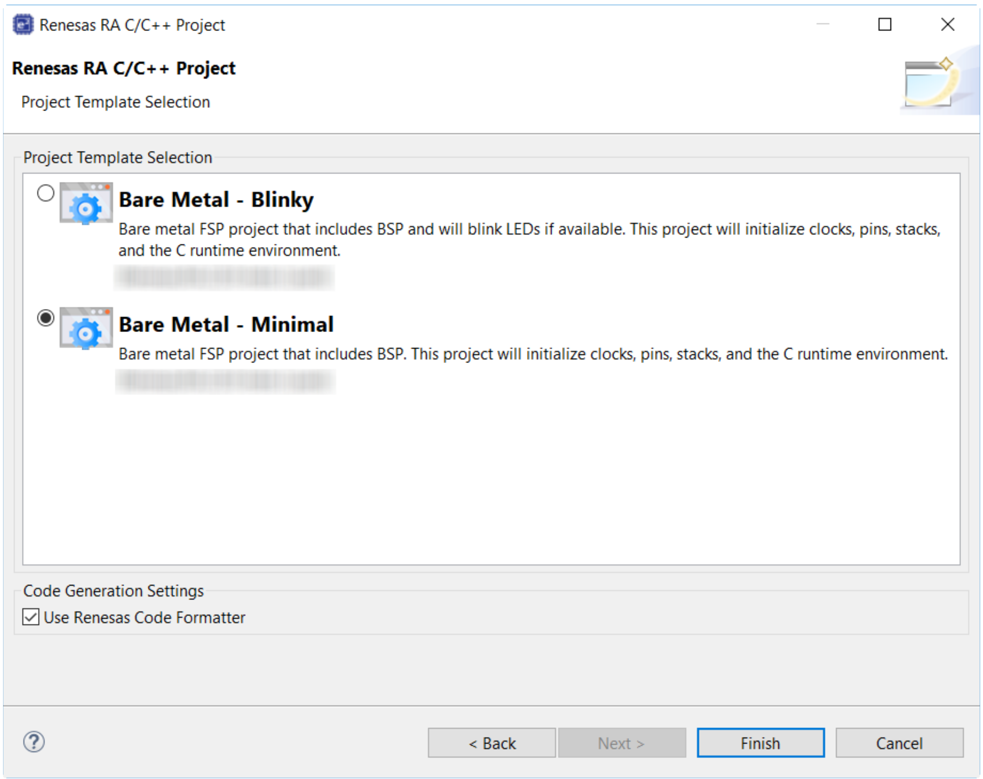


Figure 14: RA MCU Project Generator (Screen 4)

When the project is created, e2 studio displays a summary of the current project configuration in the RA MCU Project Editor.

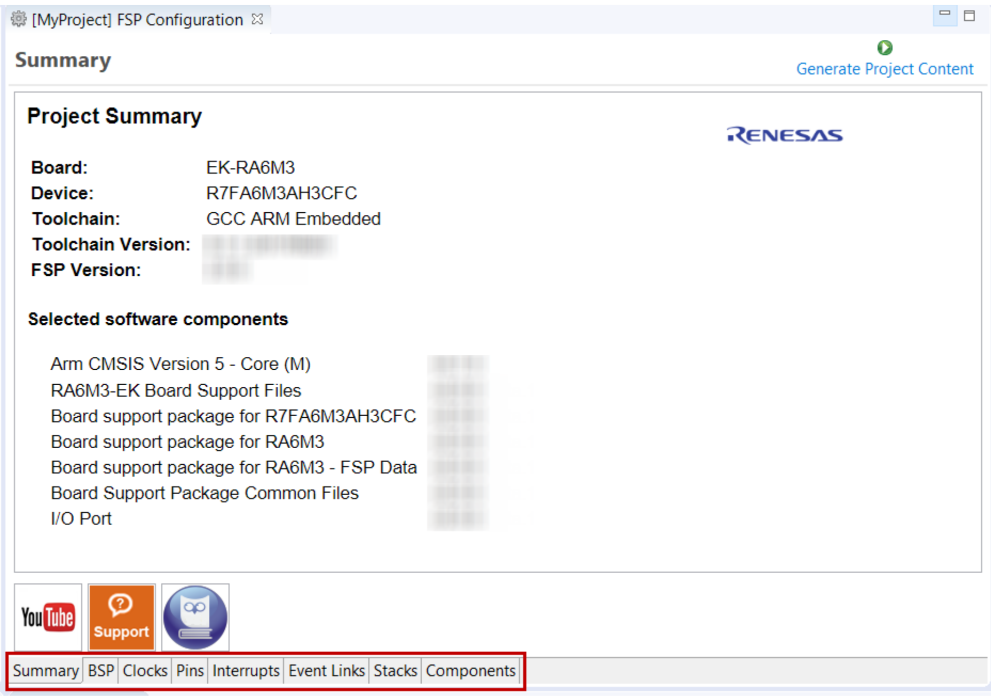


Figure 15: RA MCU Project Editor and available editor tabs

On the bottom of the RA MCU Project Editor view, you can find the tabs for configuring multiple aspects of your project:

- With the **Summary** tab, you can see all they key characteristics of the project: board, device, toolchain, and more.
- With the **BSP** tab, you can change board specific parameters from the initial project selection.
- With the **Clocks** tab, you can configure the MCU clock settings for your project.
- With the **Pins** tab, you can configure the electrical characteristics and functions of each port pin.
- With the **Interrupts** tab, you can add new user events/interrupts.
- With the **Event Links** tab, you can configure events used by the Event Link Controller.
- With the **Stacks** tab, you can add and configure FSP modules. For each module selected in this tab, the **Properties** window provides access to the configuration parameters, interrupt priorities, and pin selections.
- The **Components** tab provides an overview of the selected modules. Although you can also add drivers for specific FSP releases and application sample code here, this tab is normally only used for reference.

The functions and use of each of these tabs is explained in detail in the next section.

2.2.5 Configuring a Project

Each of the configurable elements in an FSP project can be edited using the appropriate tab in the RA Configuration editor window. Importantly, the initial configuration of the MCU after reset and before any user code is executed is set by the configuration settings in the **BSP**, **Clocks** and **Pins** tabs. When you select a project template during project creation, e2 studio configures default values that are appropriate for the associated board. You can change those default values as needed. The following sections detail the process of configuring each of the project elements for each of the associated tabs.

2.2.5.1 Summary Tab

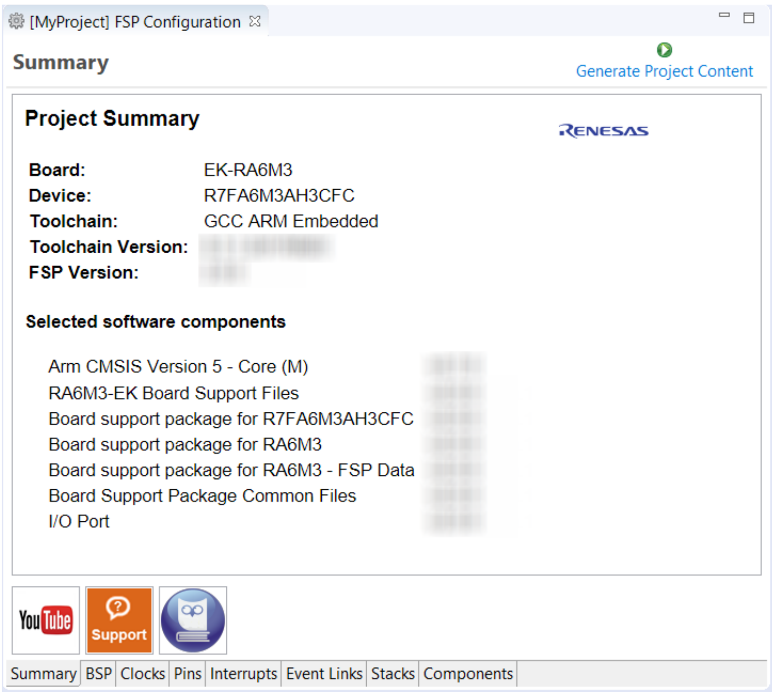


Figure 16: Configuration Summary tab

The **Summary** tab, seen in the above figure, identifies all the key elements and components of a project. It shows the target board, the device, toolchain and FSP version. Additionally, it provides a list of all the selected software components and modules used by the project. This is a more convenient summary view when compared to the **Components** tab.

The summary tab also includes handy icons with links to the Renesas YouTube channel, the Renesas support page and to the RA FSP User Manual that was downloaded during the installation process.

2.2.5.2 Configuring the BSP

The **BSP** tab shows the currently selected board (if any) and device. The Properties view is located in the lower left of the Project Configurations view as shown below.

Note

If the Properties view is not visible, click ***Window > Show View > Properties*** *in the top menu bar.*

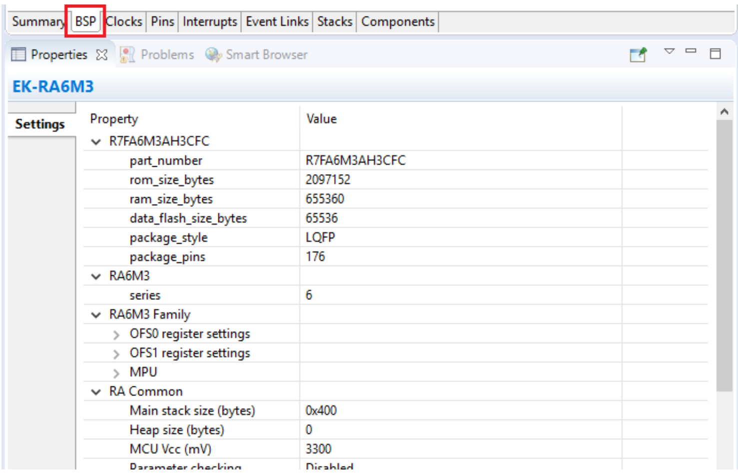


Figure 17: Configuration BSP tab

The **Properties** view shows the configurable options available for the BSP. These can be changed as required. The BSP is the FSP layer above the MCU hardware. e2 studio checks the entry fields to flag invalid entries. For example, only valid numeric values can be entered for the stack size.

When you click the **Generate Project Content** button, the BSP configuration contents are written to ra_cfg/fsp_cfg/bsp/bsp_cfg.h

This file is created if it does not already exist.

Warning

Do not edit this file as it is overwritten whenever the **Generate Project Content** button is clicked.

2.2.5.3 Configuring Clocks

The **Clocks** tab presents a graphical view of the MCU's clock tree, allowing the various clock dividers and sources to be modified. If a clock setting is invalid, the offending clock value is highlighted in red. It is still possible to generate code with this setting, but correct operation cannot be guaranteed. In the figure below, the USB clock UCLK has been changed so the resulting clock frequency is 60 MHz instead of the required 48 MHz. This parameter is colored red.

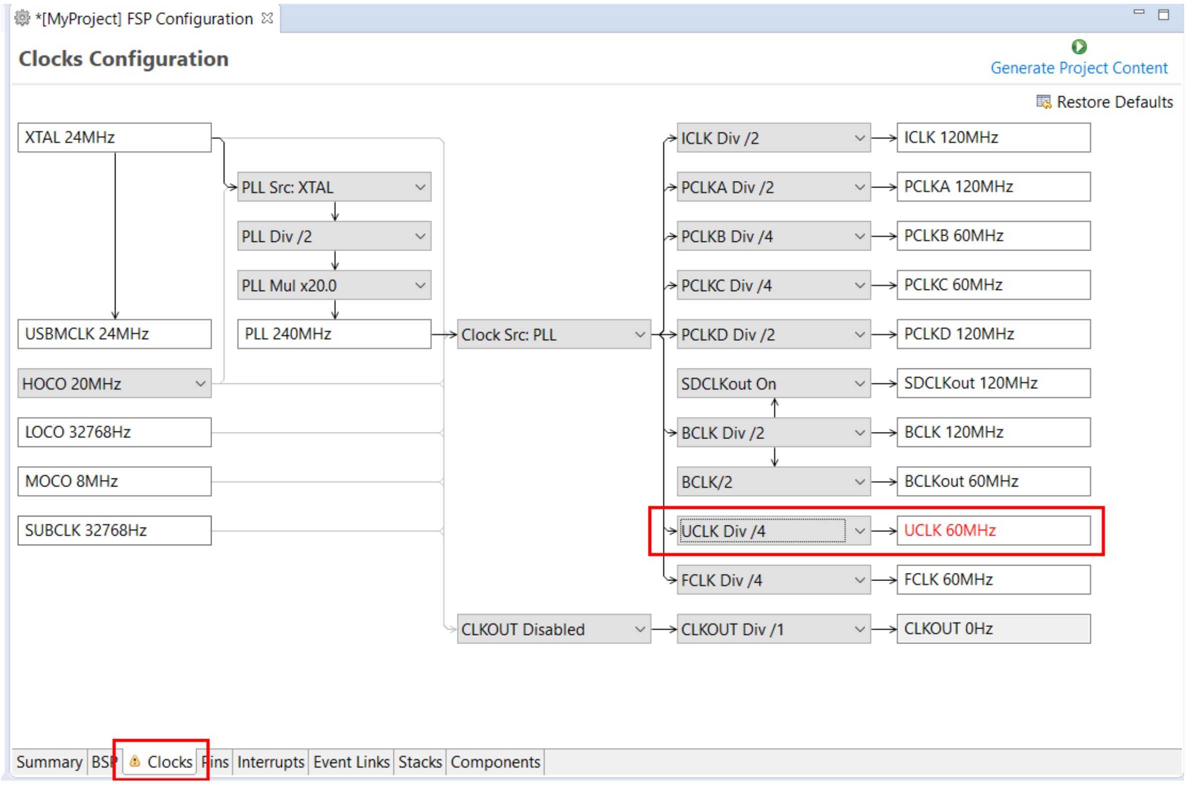


Figure 18: Configuration Clocks tab

When you click the **Generate Project Content** button, the clock configuration contents are written to: ra_gen/bsp_clock_cfg.h

This file will be created if it does not already exist.

Warning

> Do not edit this file as it is overwritten whenever the **Generate Project Content** button is clicked.

2.2.5.4 Configuring Pins

The **Pins** tab provides flexible configuration of the MCU's pins. As many pins are able to provide multiple functions, they can be configured on a peripheral basis. For example, selecting a serial channel via the SCI peripheral offers multiple options for the location of the receive and transmit pins for that module and channel. Once a pin is configured, it is shown as green in the **Package** view.

Note

> *If the **Package** view window is not open in e2 studio, select **Window** > **Show View** > **Pin Configurator** > **Package** from the top menu bar to open it.*

The **Pins** tab simplifies the configuration of large packages with highly multiplexed pins by highlighting errors and presenting the options for each pin or for each peripheral. If you selected a project template for a specific board such as the EK-RA6M3, some peripherals connected on the board are preselected.

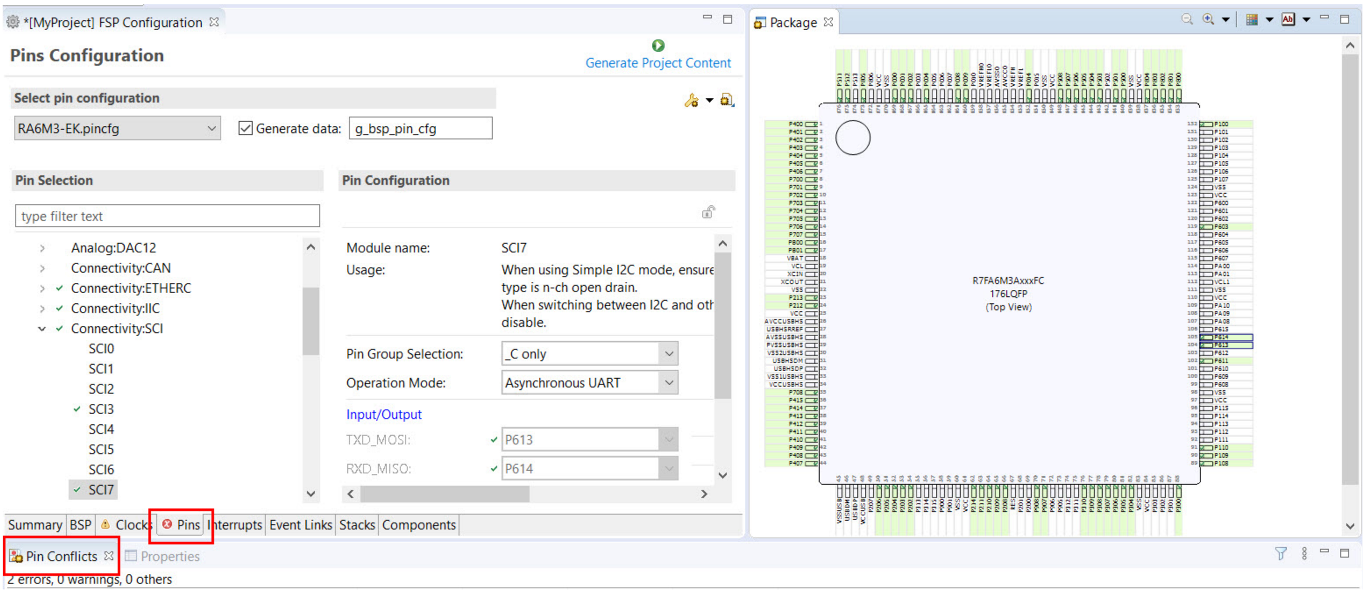


Figure 19: Pins Configuration

The pin configurator includes a built-in conflict checker, so if the same pin is allocated to another peripheral or I/O function the pin will be shown as red in the package view and also with white cross in a red square in the **Pin Selection** pane and **Pin Configuration** pane in the main **Pins** tab. The **Pin Conflicts** view provides a list of conflicts, so conflicts can be quickly identified and fixed.

In the example shown below, port P611 is already used by the CAC, and the attempt to connect this port to the Serial Communications Interface (SCI) results in a dangling connection error. To fix this error, select another port from the pin drop-down list or disable the CAC in the **Pin Selection** pane on the left side of the tab.

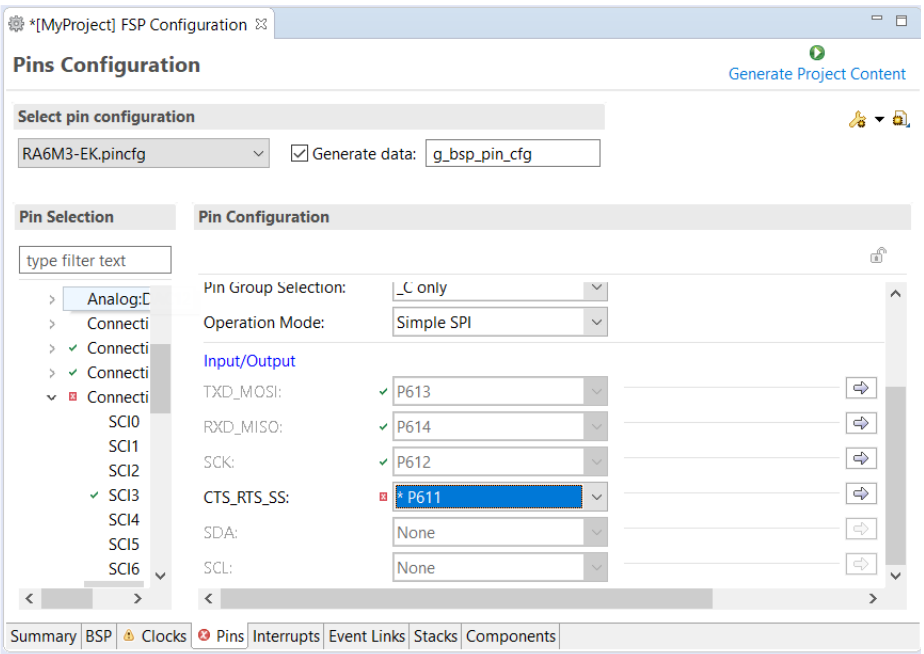


Figure 20: e2 studio Pin configurator

The pin configurator also shows a package view and the selected electrical or functional characteristics of each pin.

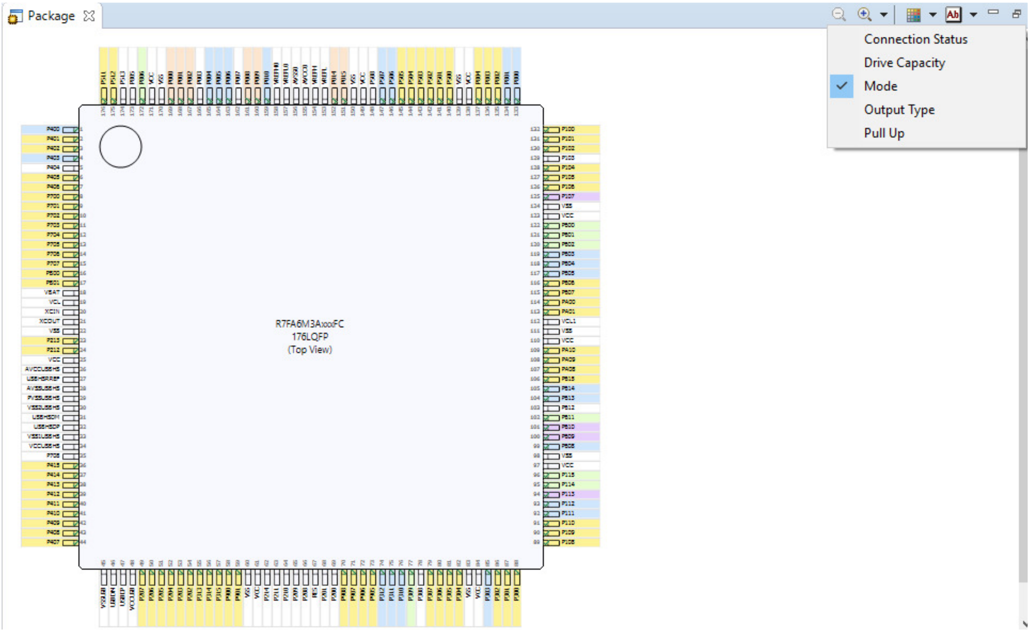


Figure 21: e2 studio Pin configurator package view

When you click the **Generate Project Content** button, the pin configuration contents are written to: ra_gen\bsp_pin_cfg.h

This file will be created if it does not already exist.

Warning

> Do not edit this file as it is overwritten whenever the **Generate Project Content** button is clicked.

To make it easy to share pinning information for your project, e2 studio exports your pin configuration settings to a csv format and copies the csv file to ra_gen/<MCU package>.csv.

2.2.5.5 Configuring Interrupts from the Stacks Tab

You can use the **Properties** view in the **Stacks** tab to enable interrupts by setting the interrupt priority. Select the driver in the **Stacks** pane to view and edit its properties.

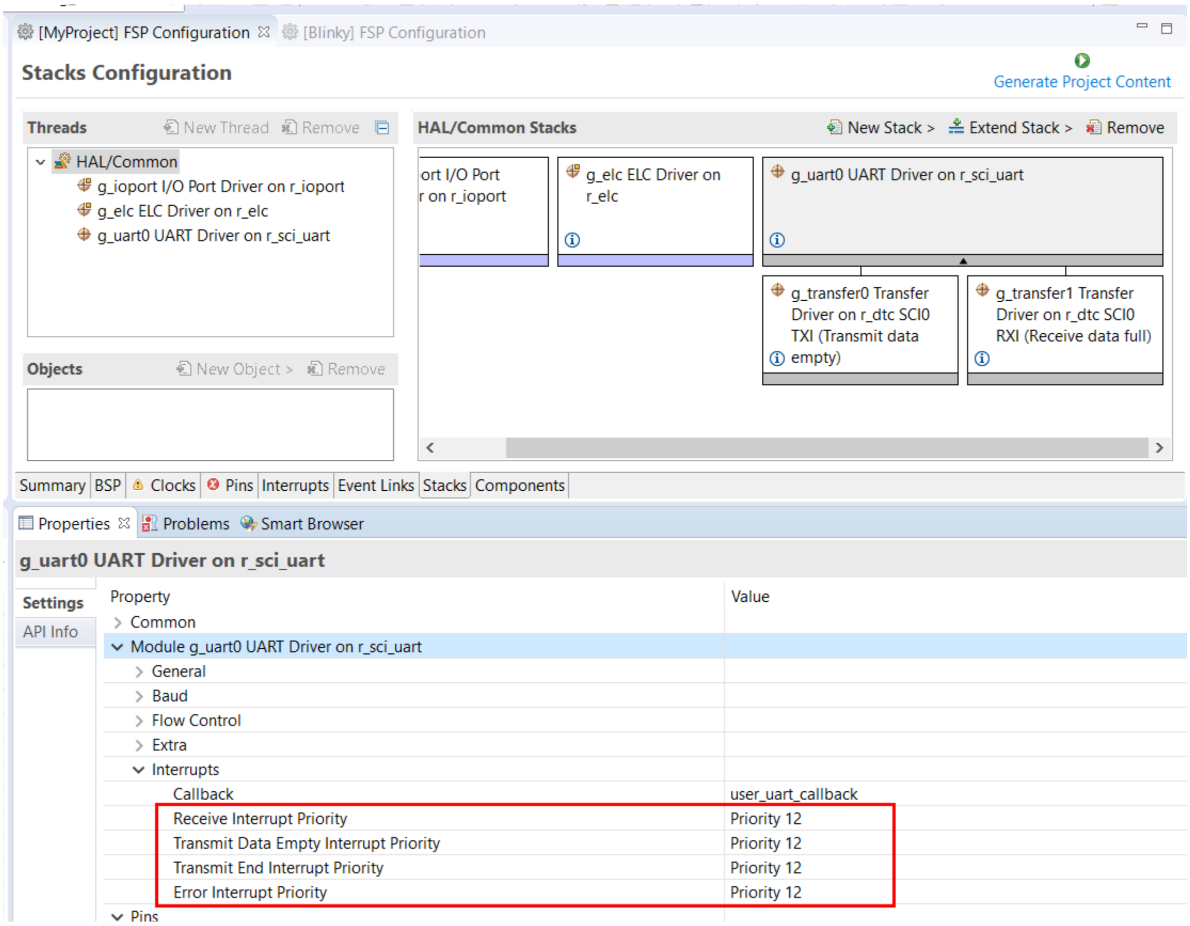


Figure 22: Configuring Interrupts in the Stacks tab

Creating Interrupts from the Interrupts Tab

On the **Interrupts** tab, the user can bypass a peripheral interrupt set by the FSP by setting a user-defined ISR. This can be done by adding a new event via the **New User Event** button.

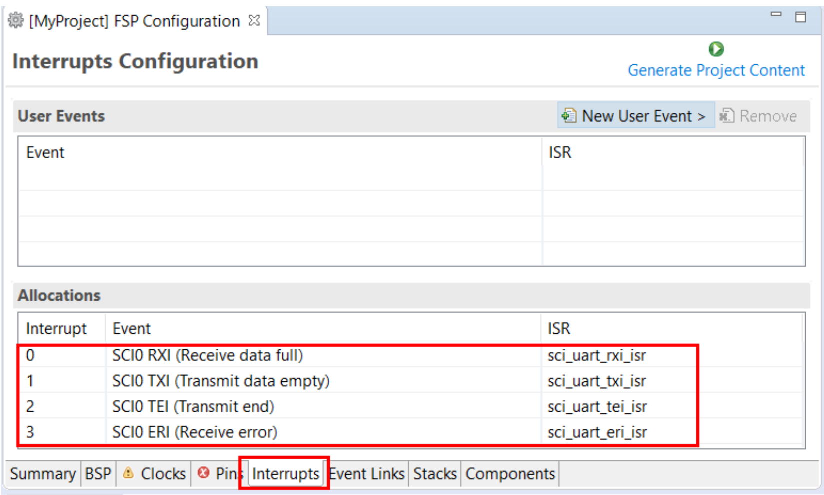


Figure 23: Configuring interrupt in Interrupt Tab

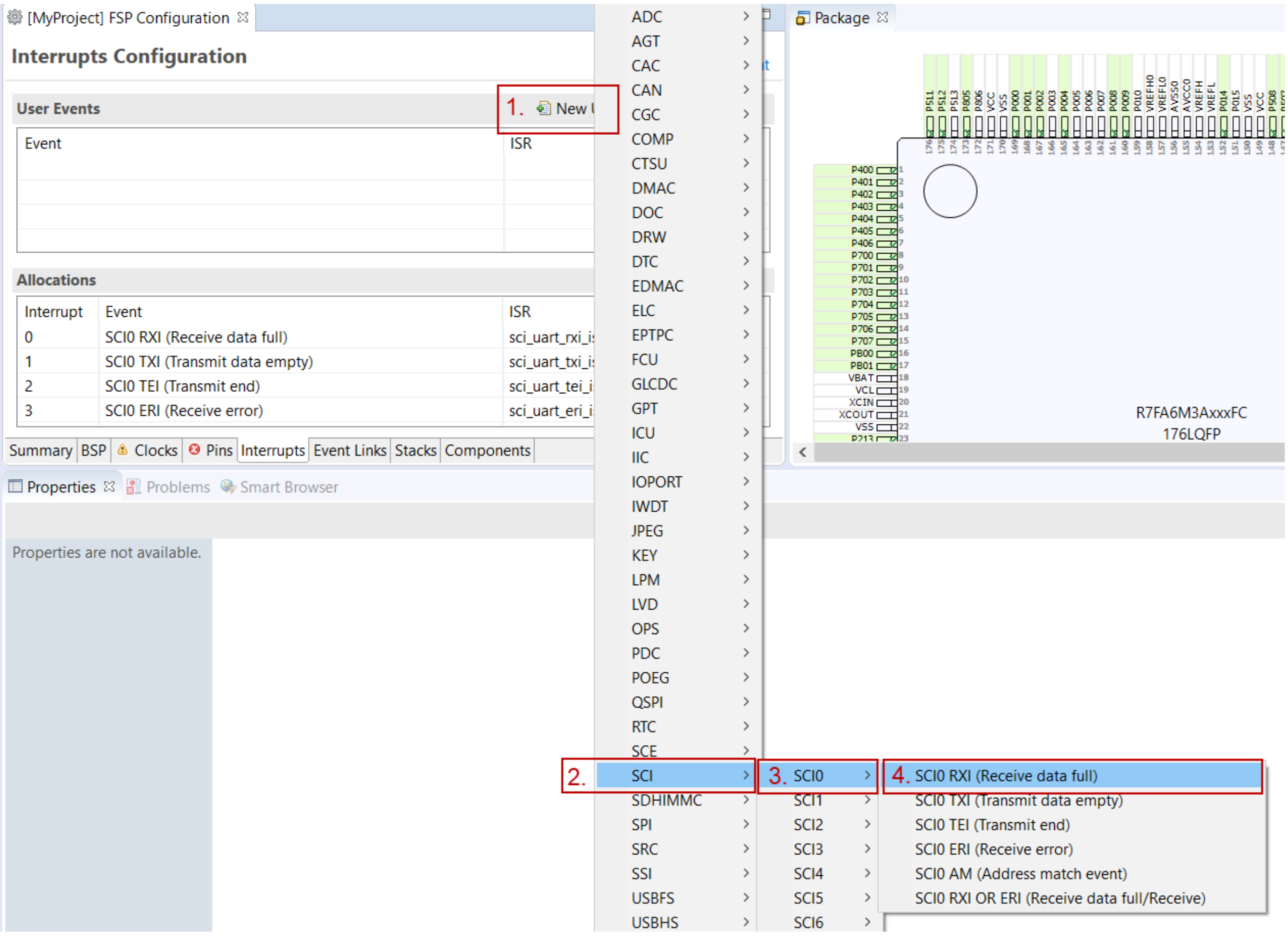


Figure 24: Adding user-defined event

Enter the name of ISR for the new user event.

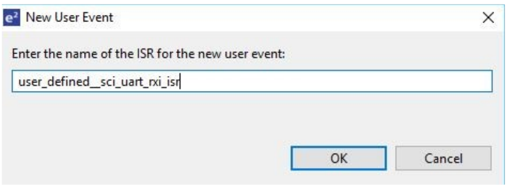


Figure 25: User-defined event ISR

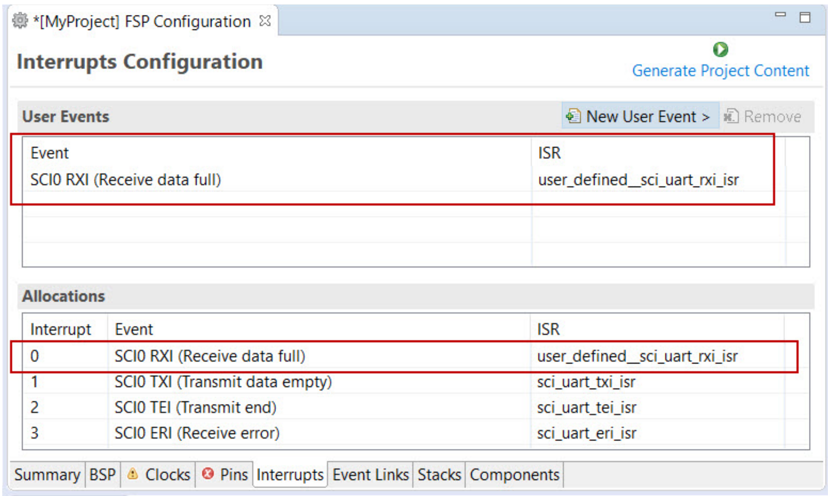


Figure 26: Using a user-defined event

2.2.5.6 Viewing Event Links

The Event Links tab can be used to view the Event Link Controller events. The events are sorted by peripheral to make it easy to find and verify them.

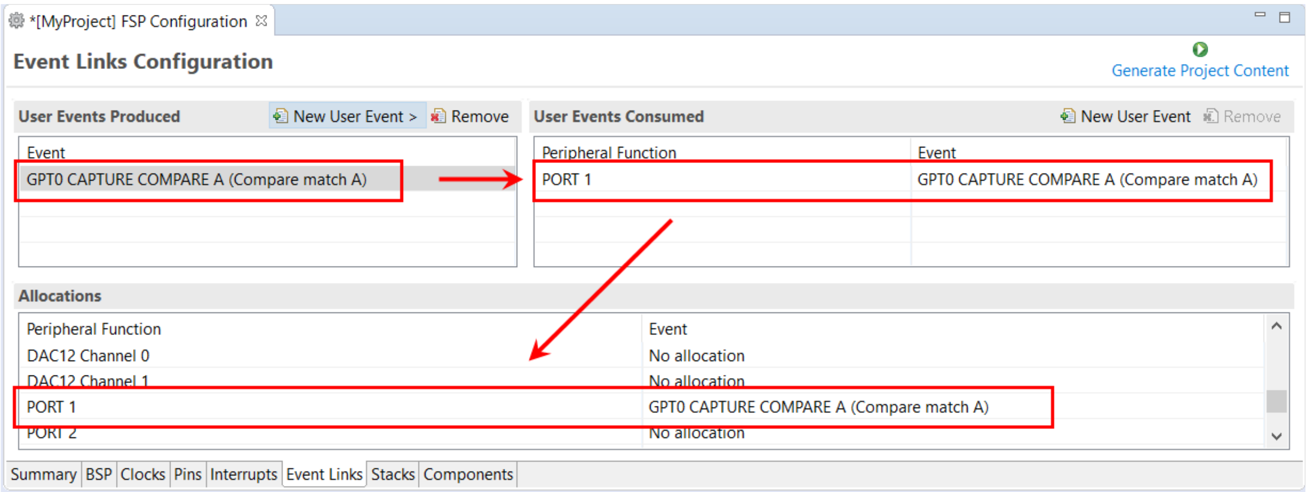


Figure 27: Viewing Event Links

Like the Interrupts tab, user-defined event sources and destinations (producers and consumers) can be defined by clicking the relevant **New User Event** button. Once a consumer is linked to a producer the link will appear in the **Allocations** section at the bottom.

Note

When selecting an ELC event to receive for a module (or when manually defining an event link), only the events that are made available by the modules configured in the project will be shown.

2.2.6 Adding Threads and Drivers

Every RTOS-based RA Project includes at least one RTOS Thread and a stack of FSP modules running in that thread. The **Stacks** tab is a graphical user interface which helps you to add the right modules to a thread and configure the properties of both the threads and the modules associated with each thread. Once you have configured the thread, e2 studio automatically generates the code reflecting your configuration choices.

For any driver, or, more generally, any module that you add to a thread, e2 studio automatically resolves all dependencies with other modules and creates the appropriate stack. This stack is displayed in the Stacks pane, which e2 studio populates with the selected modules and module options for the selected thread.

The default view of the **Stacks** tab includes a Common Thread called **HAL/Common**. This thread includes the driver for I/O control (IOPORT). The default stack is shown in the **HAL/Common Stacks** pane. The default modules added to the HAL/Common driver are special in that the FSP only requires a single instance of each, which e2 studio then includes in every user-defined thread by default.

In applications that do not use an RTOS or run outside of the RTOS, the HAL/Common thread becomes the default location where you can add additional drivers to your application.

For a detailed description on how to add and configure modules and stacks, see the following sections:

- Adding and Configuring HAL Drivers
- Adding Drivers to a Thread and Configuring the Drivers

Once you have added a module either to HAL/Common or to a new thread, you can access the driver's configuration options in the **Properties** view. If you added thread objects, you can access the objects configuration options in the **Properties** view in the same way.

You can find details about how to configure threads here: Configuring Threads

Note

Driver and module selections and configuration options are defined in the FSP pack and can therefore change when the FSP version changes.

2.2.6.1 Adding and Configuring HAL Drivers

For applications that run outside or without the RTOS, you can add additional HAL drivers to your application using the HAL/Common thread. To add drivers, follow these steps:

1. Click on the HAL/Common icon in the **Stacks** pane. The Modules pane changes to **HAL/Common Stacks**.

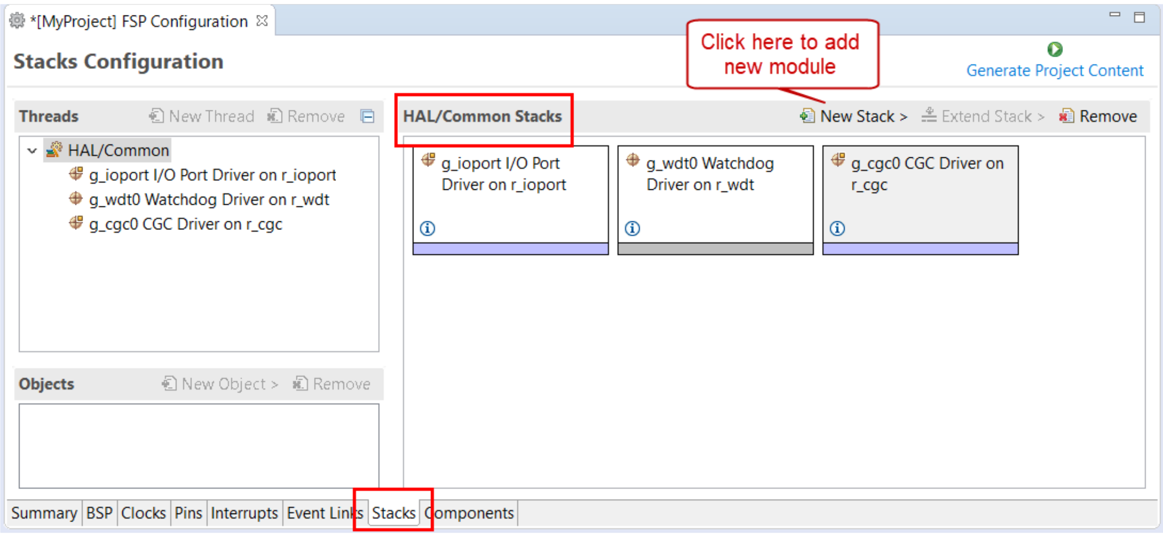


Figure 28: e2 studio Project configurator - Adding drivers

2. Click **New Stack** to see a drop-down list of HAL level drivers available in the FSP.

3. Select a driver from the menu **New Stack > Driver**.

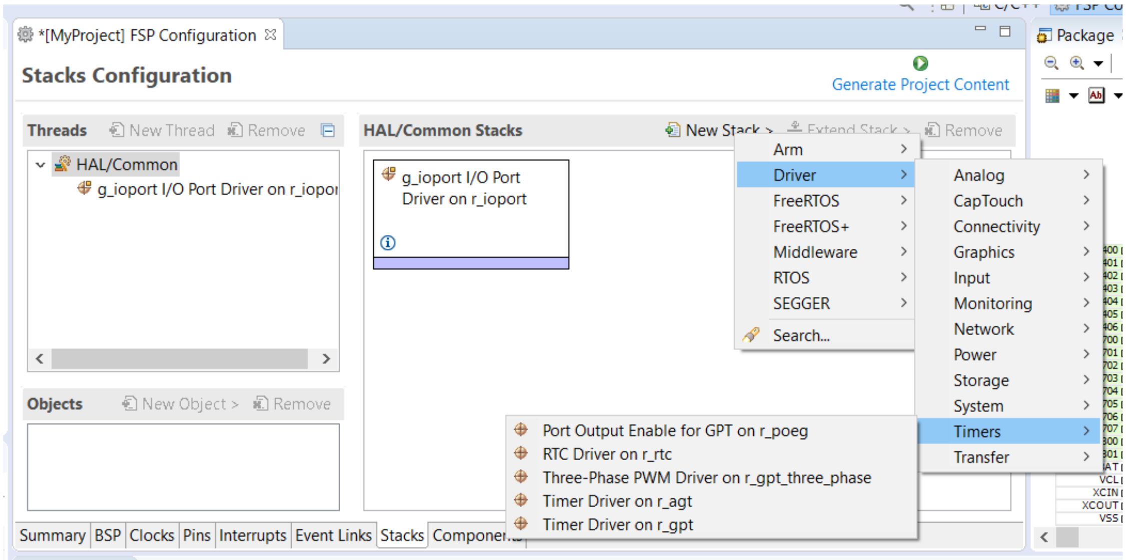


Figure 29: Select a driver

4. Select the driver module in the **HAL/Common Modules** pane and configure the driver properties in the **Properties** view.

e2 studio adds the following files when you click the **Generate Project Content** button:

- The selected driver module and its files to the ra/fsp directory
- The main() function and configuration structures and header files for your application as shown in the table below.

File	Contents	Overwritten by Generate Project Content?

ra_gen/main.c	Contains main() calling generated and user code. When called, the BSP already has Initialized the MCU.	Yes
ra_gen/hal_data.c	Configuration structures for HAL Driver only modules.	Yes
ra_gen/hal_data.h	Header file for HAL driver only modules.	Yes
src/hal_entry.c	User entry point for HAL Driver only code. Add your code here.	No

The configuration header files for all included modules are created or overwritten in this folder: ra_cfg/fsp_cfg

2.2.6.2 Adding Drivers to a Thread and Configuring the Drivers

For an application that uses the RTOS, you can add one or more threads, and for each thread at least one module that runs in the thread. You can select modules from the Driver dropdown menu. To add modules to a thread, follow these steps:

1. In the **Threads** pane, click **New Thread** to add a Thread.

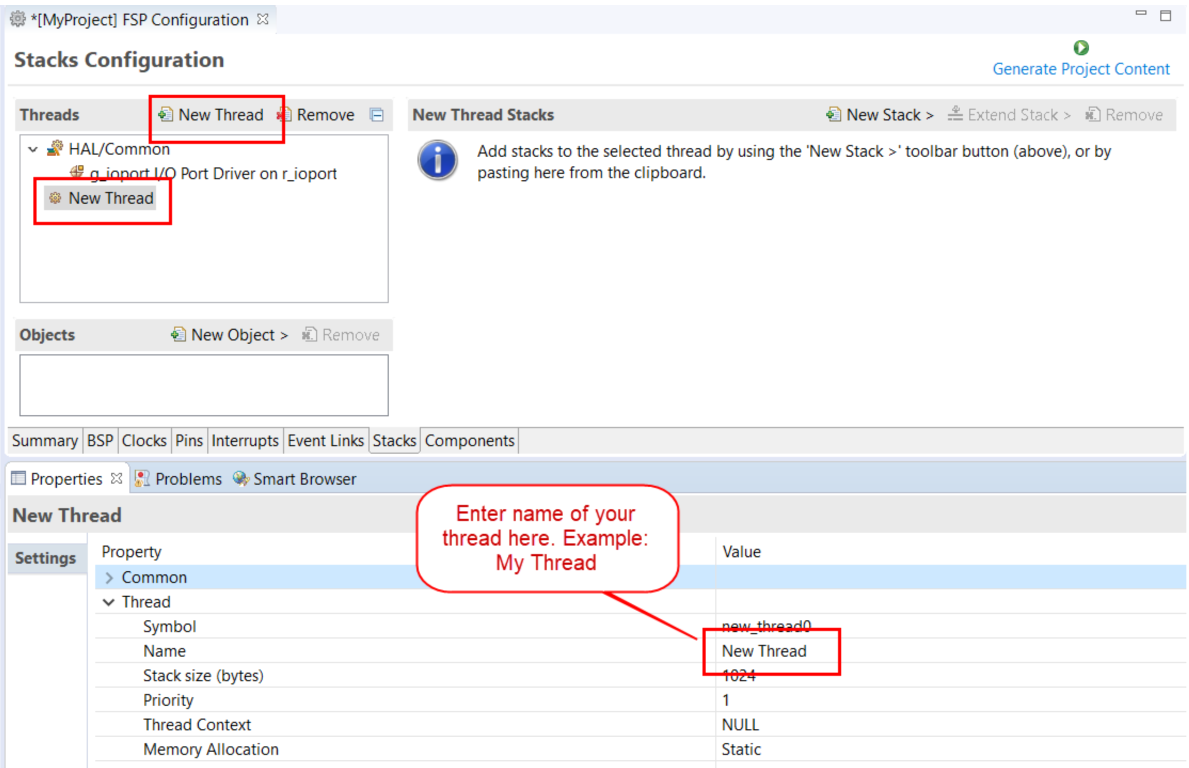


Figure 30: Adding a new RTOS Thread on the Stacks tab

2. In the **Properties** view, click on the **Name** and **Symbol** entries and enter a distinctive name and symbol for the new thread.

Note

*e2 studio updates the name of the thread stacks pane to **My Thread Stacks**.*

3. In the **My Thread Stacks** pane, click on **New Stack** to see a list of modules and drivers. HAL-level drivers can be added here.

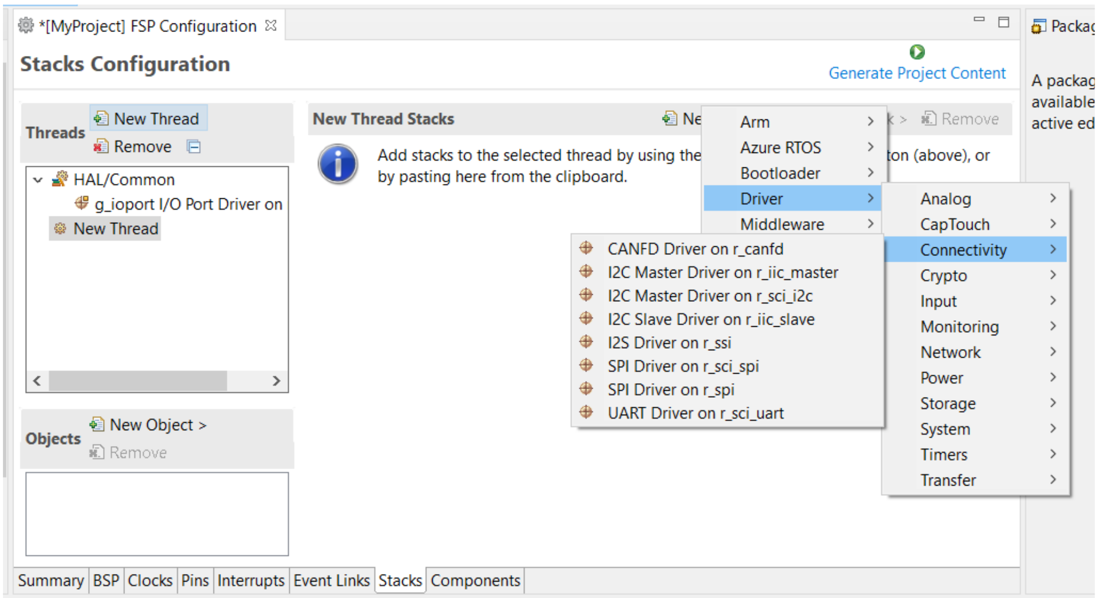


Figure 31: Adding Modules and Drivers to a thread

4. Select a module or driver from the list.

5. Click on the added driver and configure the driver as required by the application by updating the configuration parameters in the **Properties** view. To see the selected module or driver and be able to edit its properties, make sure the Thread containing the driver is highlighted in the **Threads** pane.

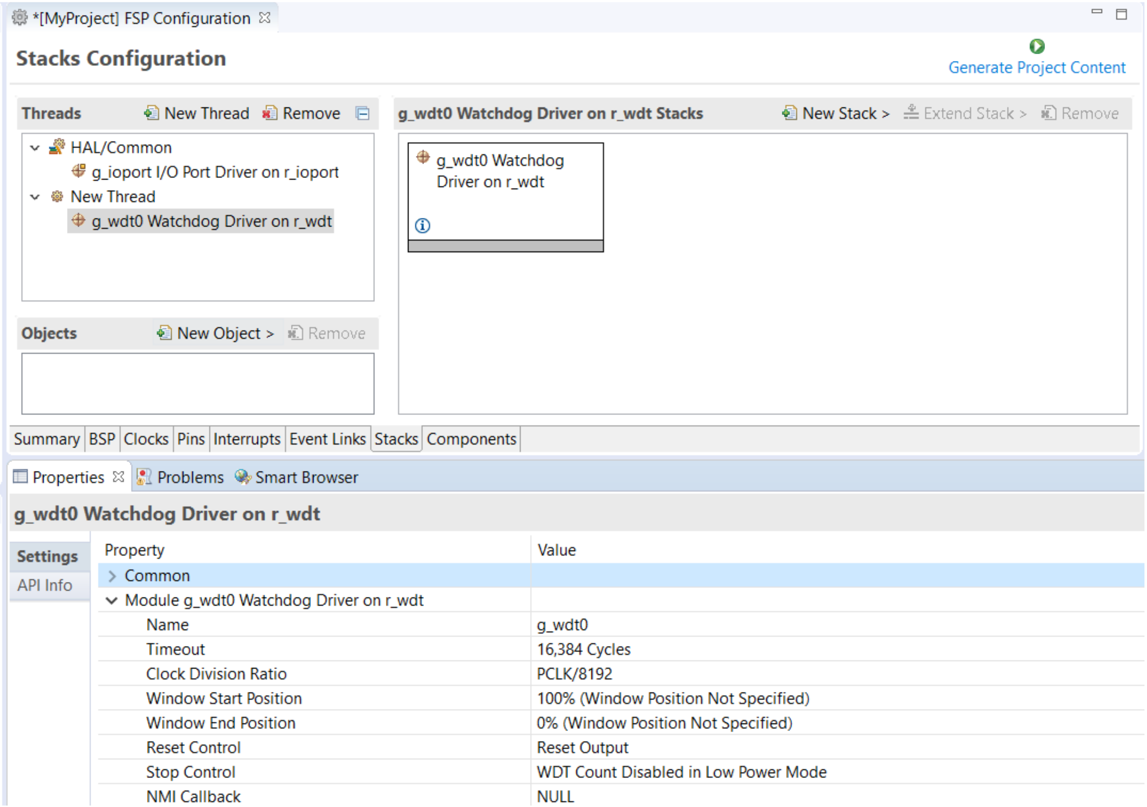


Figure 32: Configuring Module or Driver properties

6. If needed, add another thread by clicking **New Thread** in the **Threads** pane.

When you press the **Generate Project Content** button for the example above, e2 studio creates the files as shown in the following table:

File	Contents	Overwritten by Generate Project Content?
ra_gen/main.c	Contains main() calling generated and user code. When called the BSP will have initialized the MCU.	Yes
ra_gen/my_thread.c	Generated thread "my_thread" and configuration structures for modules added to this thread.	Yes
ra_gen/my_thread.h	Header file for thread "my_thread"	Yes
ra_gen/hal_data.c	Configuration structures for HAL Driver only modules.	Yes
ra_gen/hal_data.h	Header file for HAL Driver only modules.	Yes
src/hal_entry.c	User entry point for HAL Driver only code. Add your code here.	No

src/my_thread_entry.c	User entry point for thread "my_thread". Add your code here.	No

The configuration header files for all included modules and drivers are created or overwritten in the following folders: ra_cfg/fsp_cfg/<header files>

2.2.6.3 Configuring Threads

If the application uses an RTOS, the **Stacks** tab can be used to simplify the creation of RTOS threads, semaphores, mutexes, and event flags.

The components of each thread can be configured from the **Properties** view as shown below.

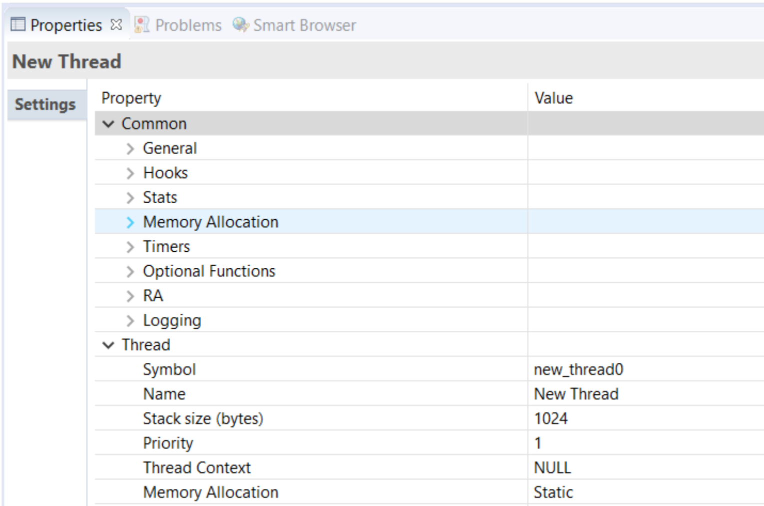


Figure 33: New Thread Properties

The **Properties** view contains settings common for all Threads (**Common**) and settings for this particular thread (**Thread**).

For this thread instance, the thread's name and properties (such as priority level or stack size) can be easily configured. e2 studio checks that the entries in the property field are valid. For example, it will verify that the field **Priority**, which requires an integer value, only contains numeric values between 0 and 9.

To add RTOS resources to a Thread, select a thread and click on **New Object** in the Thread Objects pane. The pane takes on the name of the selected thread, in this case **My Thread Objects**.

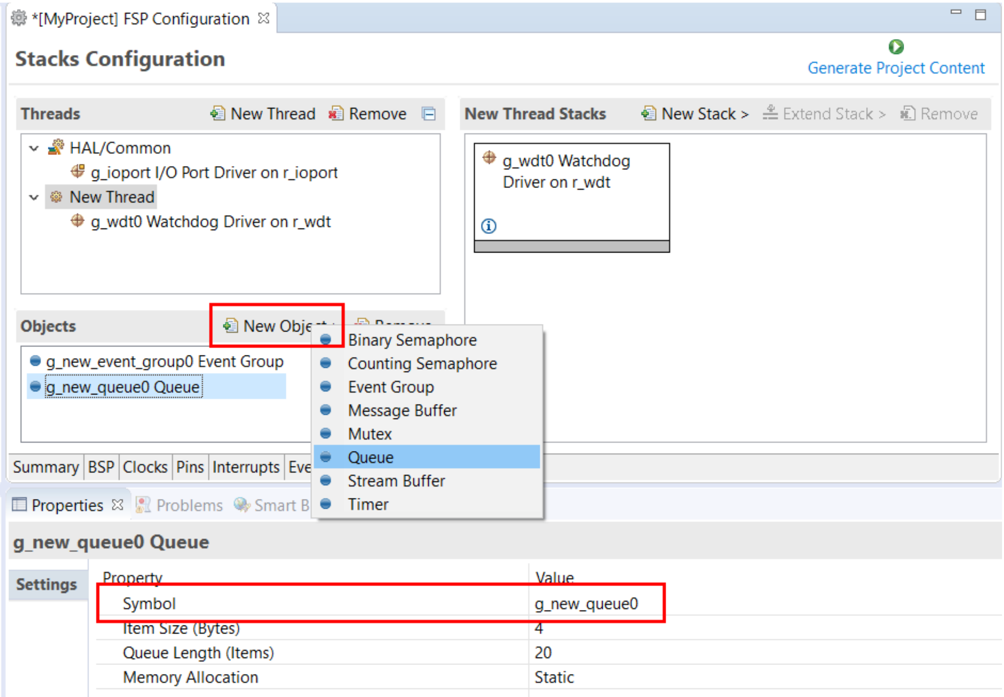


Figure 34: Configuring Thread Object Properties

Make sure to give each thread object a unique name and symbol by updating the **Name** and **Symbol** entries in the **Properties** view.

2.2.7 Reviewing and Adding Components

The **Components** tab enables the individual modules required by the application to be included or excluded. Modules common to all RA MCU projects are preselected (for example: **BSP > BSP > Board-specific BSP** and **HAL Drivers > all > r_cgc**). All modules that are necessary for the modules selected in the **Stacks** tab are included automatically. You can include or exclude additional modules by ticking the box next to the required component.

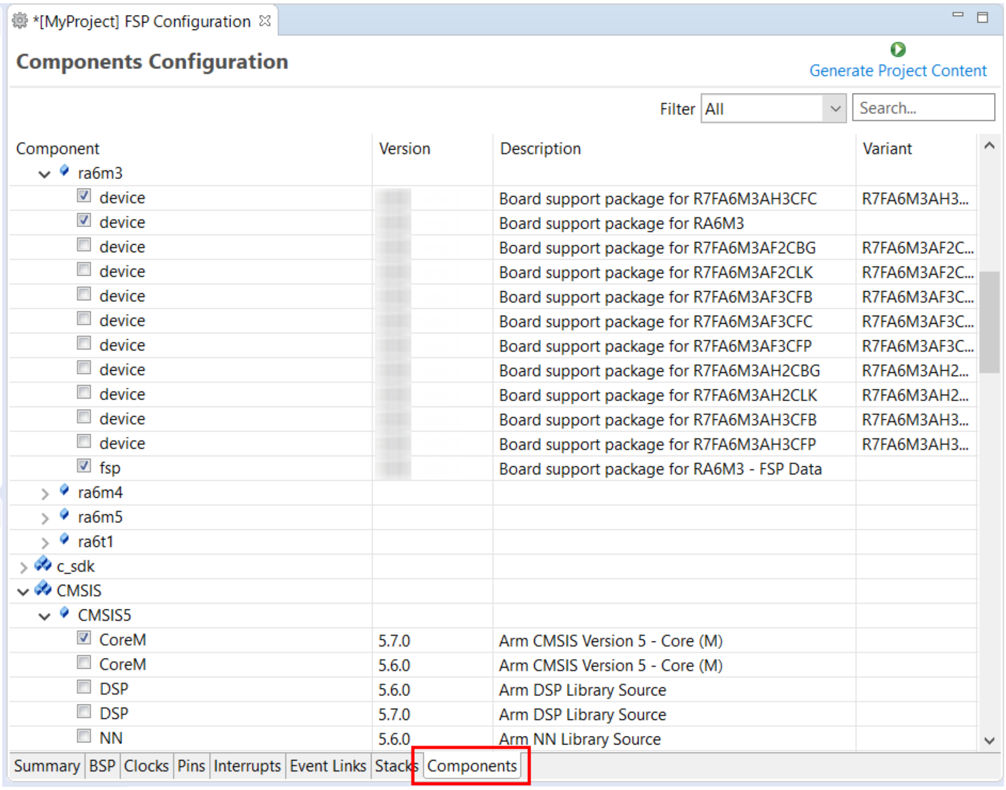


Figure 35: Components Tab

Clicking the **Generate Project Content** button copies the .c and .h files for each selected component into the following folders:

- ra/fsp/inc/api
- ra/fsp/inc/instances
- ra/fsp/src/bsp
- ra/fsp/src/<Driver_Name>

e2 studio also creates configuration files in the ra_cfg/fsp_cfg folder with configuration options set in the **Stacks** tab.

2.2.8 Writing the Application

Once you have added Modules and drivers and set their configuration parameters in the **Stacks** tab, you can add the application code that calls the Modules and drivers.

Note

To check your configuration, build the project once without errors before adding any of your own application code.

2.2.8.1 Coding Features

e2 studio provides several efficiency improving features that help write code. Review these features prior to digging into the code development step-by-step sections that follow.

Autocomplete

Autocomplete is a context aware coding accelerator that suggests possible completions for partially typed-in code elements. If you can 'guess' the first part of a macro, for example, the Autocomplete function can suggest options for completing the rest of the macro.

In the following example, a macro related to a BSP_IO setting needs to be found. After typing BSP_IO_ in a source code file, pressing Ctrl + Space opens the Autocomplete list. This list shows a selection of context aware options for completing the macro. Scroll through the window to find the desired macro (in this case BSP_IO_LEVEL_HIGH) and click on it to add it to your code.

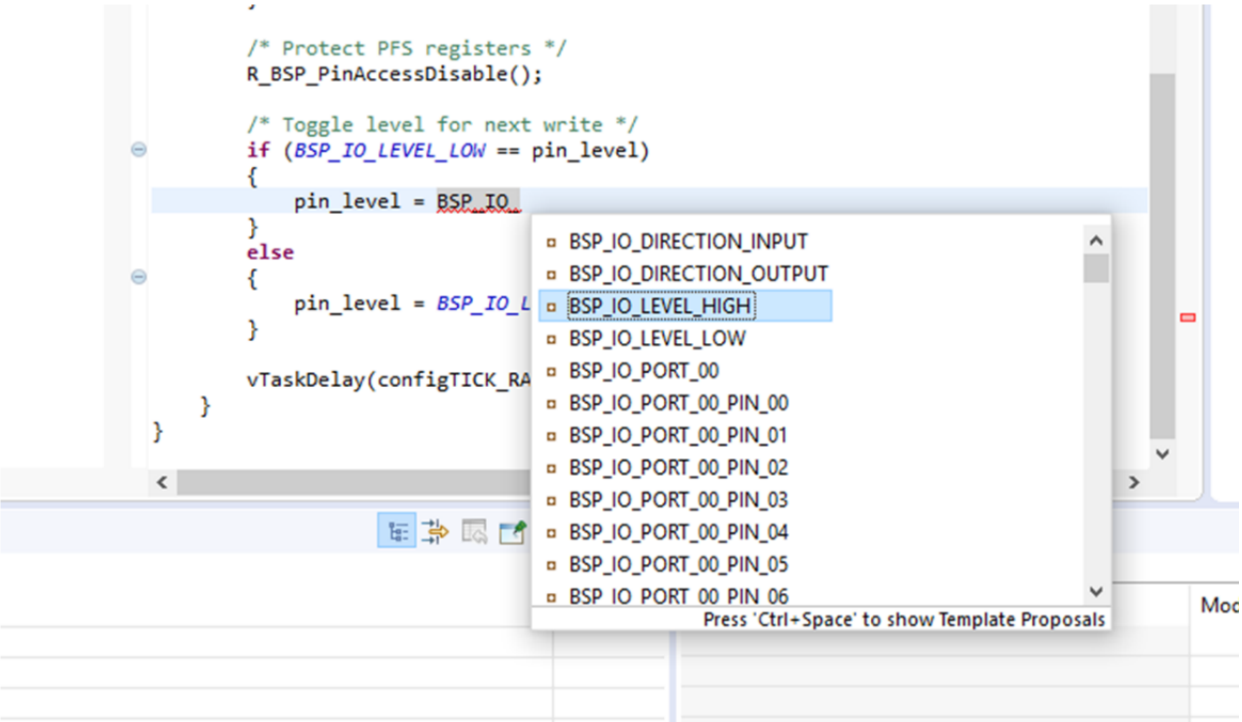


Figure 36: Autocomplete example

Other code elements can use autocomplete too. Some of the more common uses for Autocomplete include Enumerations, Types, and API functions - but try it in any situation you think the tool may have enough context to determine what you might be looking for.

For a hands-on experience using Autocomplete use the Quick FSP Labs for Creating Blinky from Scratch and Creating an RTC Blinky from Scratch. These 15-minute Do it Yourself labs take you through the step-by-step process of using Autocomplete, Developer Assistance, and the Help system.

Welcome Window

The e2 studio Welcome window displays useful information and common links to assist in development. Check out these resources to see what is available. They are updated with each release, so check back to see what has been added after a new release.

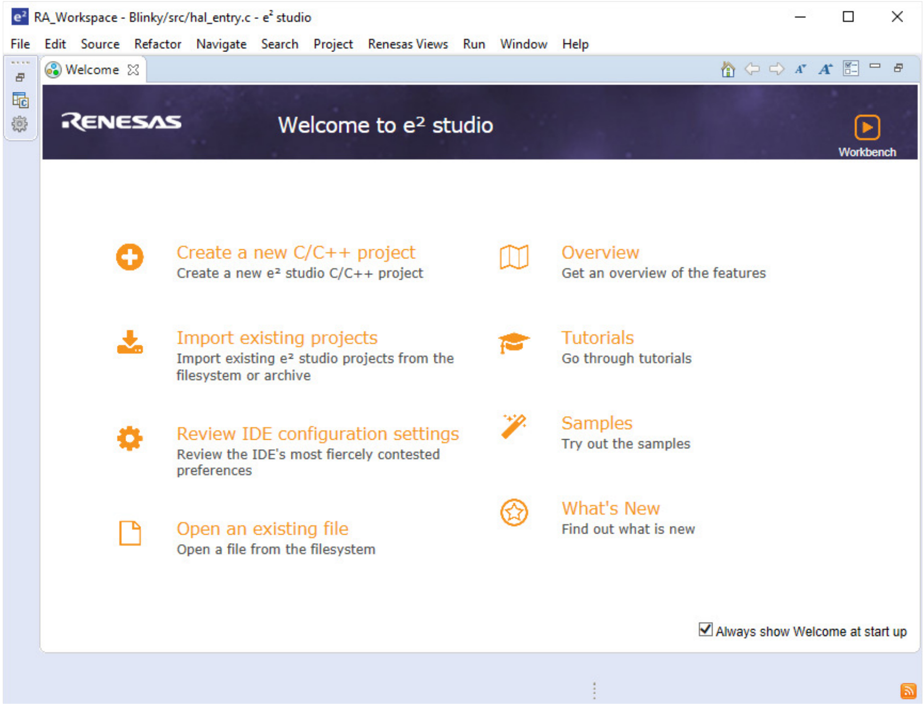


Figure 37: Welcome window

Cheat Sheets

Cheat sheets are macro driven illustrations of some common tasks. They show, step-by-step, what commands and menus are used. These will be populated with more examples on each release. Cheat Sheets are available from the **Help** menu.

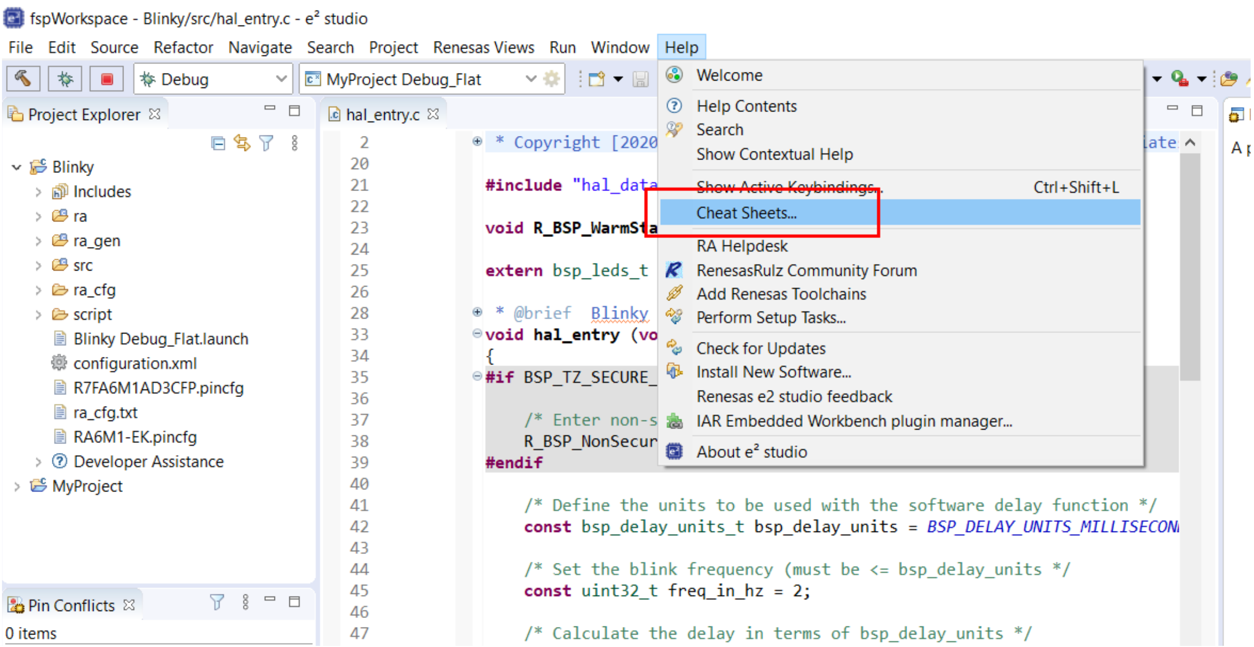


Figure 38: Cheat Sheets

Developer Assistance

FSP Developer Assistance provides developers with module and Application Programming Interface (API) reference documentation in e2 studio. After configuring the threads and software stacks for an FSP project with the RA Configuration editor, Developer Assistance quickly helps you get started writing C/C++ application code for the project using the configured stack modules.

1. Expand the project explorer to view Developer Assistance

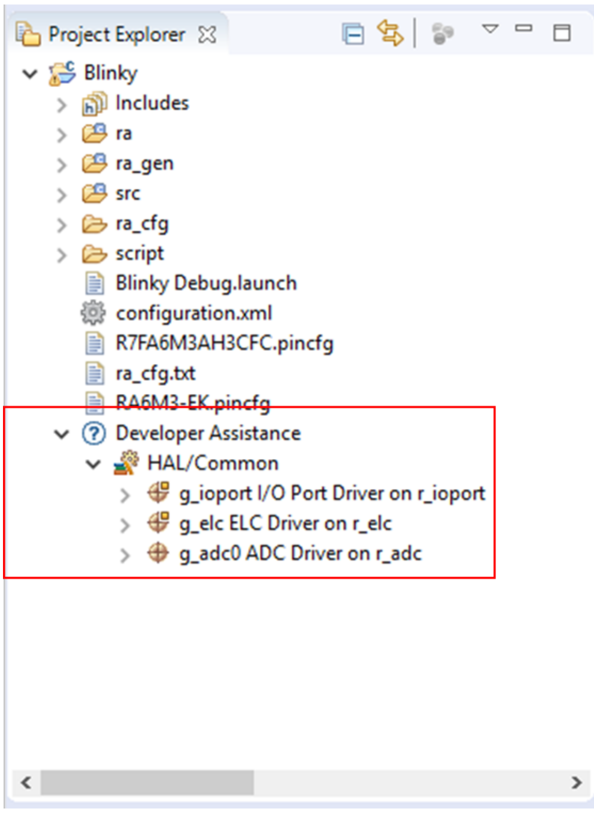


Figure 39: Developer Assistance

2. Expand a stack module to show its APIs

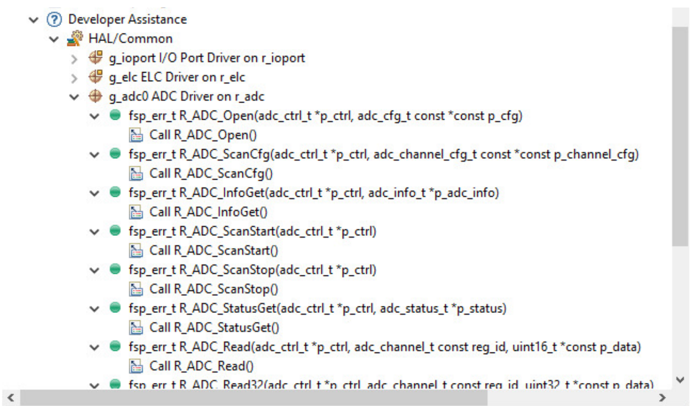


Figure 40: Developer Assistance APIs

3. Dragging and dropping an API from Develop Assistance to a source file helps to write source

code quickly.

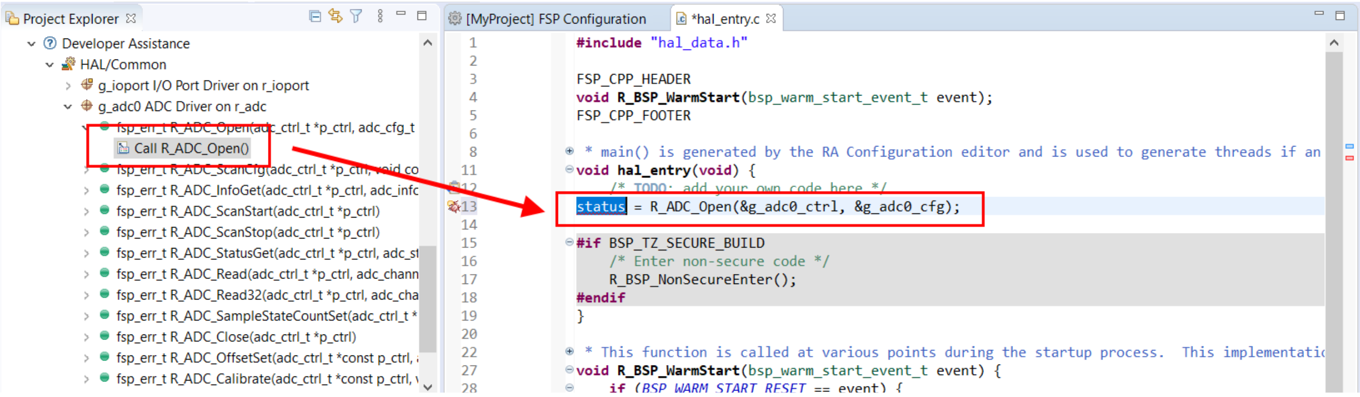


Figure 41: Dragging and Dropping an API in Developer Assistance

For a hands-on experience using Developer Assistance use the Quick FSP Labs for An Introduction to Developer Assistance, Creating Blinky from Scratch and Creating an RTC Blinky from Scratch. These 15-minute Do it Yourself labs take you through the step-by-step process of using Autocomplete, Developer Assistance, and the Help system.

Information Icon

Information icons are available on each module in the thread stack. Clicking on these icons opens a module folder on GitHub that contains additional information on the module. An example information Icon is shown below:

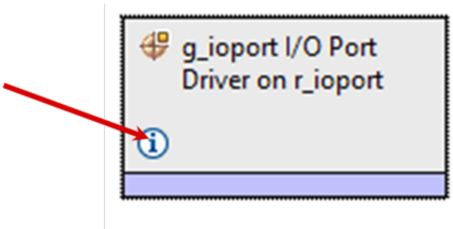


Figure 42: Information icon

IDE Help

A good source of additional information for many FSP topics is the Help system. To get to the Help system, click on **Help** and then select **Help Contents** as seen below.

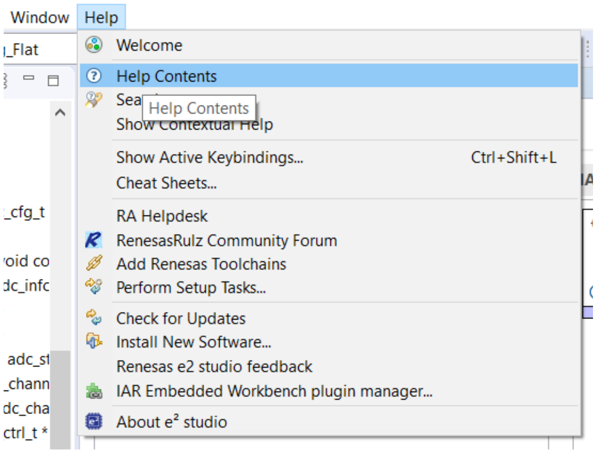


Figure 43: Opening the Help System

Once the Help system is open, select the **RA Contents** entry in the left side Guide-bar. Expand it to see the main RA Topics.

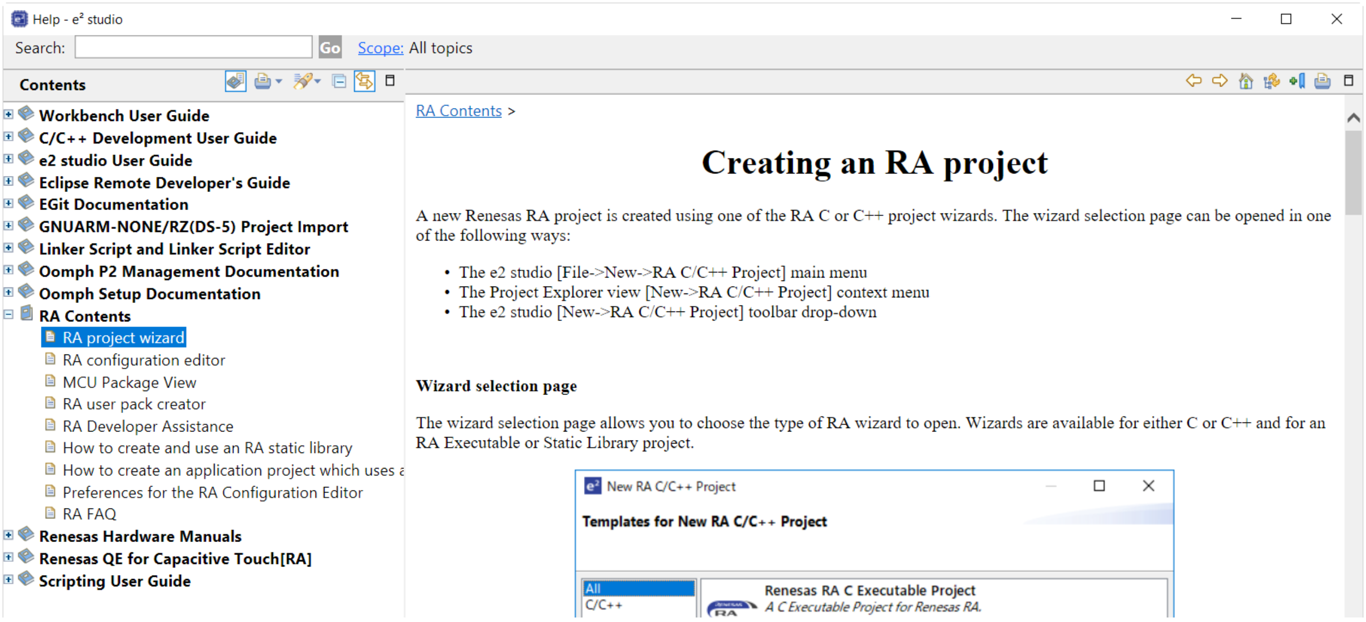


Figure 44: RA Content Help

You can also search for help topics by using the Search bar. Below is an example searching for Visual Expressions, a helpful feature in the e2 studio debugger.

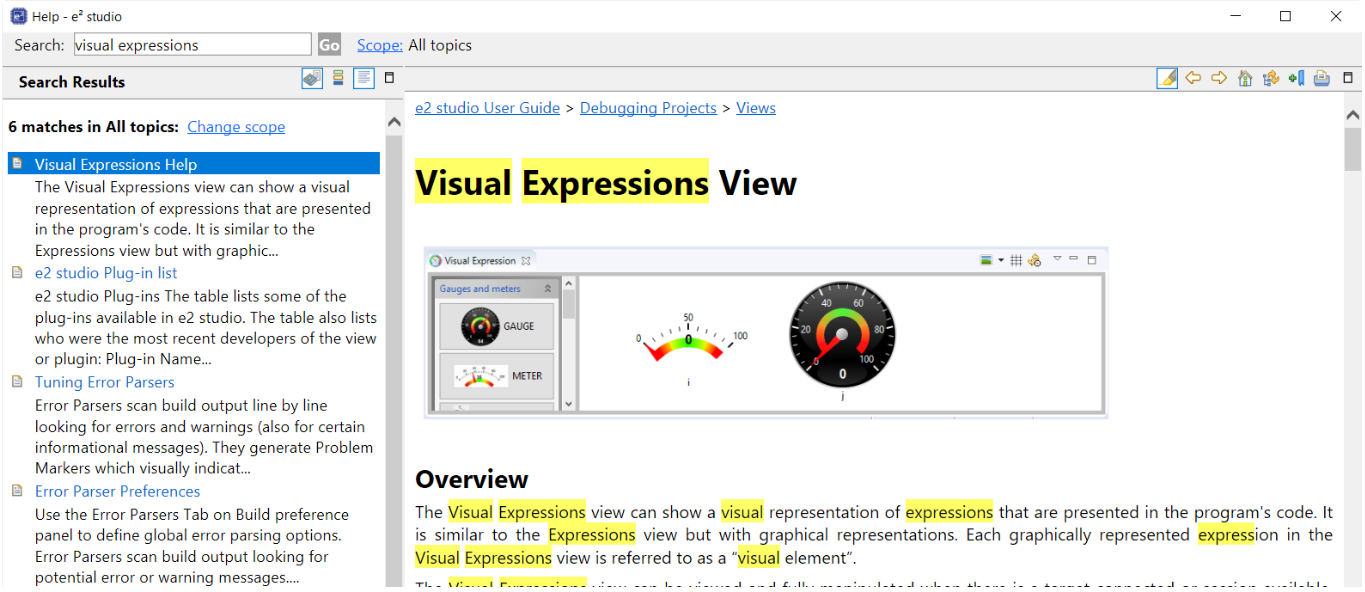


Figure 45: e2 studio Help from the Search Bar

For a hands-on experience using the Help system use the Quick FSP Labs for An Introduction to Developer Assistance, Creating Blinky from Scratch and Creating an RTC Blinky from Scratch. These 15-minute Do it Yourself labs take you through the step-by-step process of using Autocomplete, Developer Assistance, and the Help system.

2.2.8.2 HAL Modules in FSP: A Practical Description

The FSP Architecture section describes FSP stacks, modules and interfaces in significant detail, providing an understanding of the theory behind them. The following sections provides a quick and practical introduction on how to use API functions when writing code and where in the API reference sections you can find useful API related information.

Introduction to HAL Modules

In FSP, HAL module drivers provide convenient API functions that access RA processor peripheral features. Module properties are defined in the RA GUI configurator, eliminating the tedious and error prone process of setting peripheral control registers. When configuration is complete, the generator automatically creates the code needed to implement the associated API functions. API functions are the main way a developer interacts with the target processor and peripherals.

HAL Driver API Function Call Formats

HAL driver API functions all have a similar format. They all start with "R_" to indicate they are HAL related functions. Next comes the module name followed by the function and any parameters. This format is illustrated below:

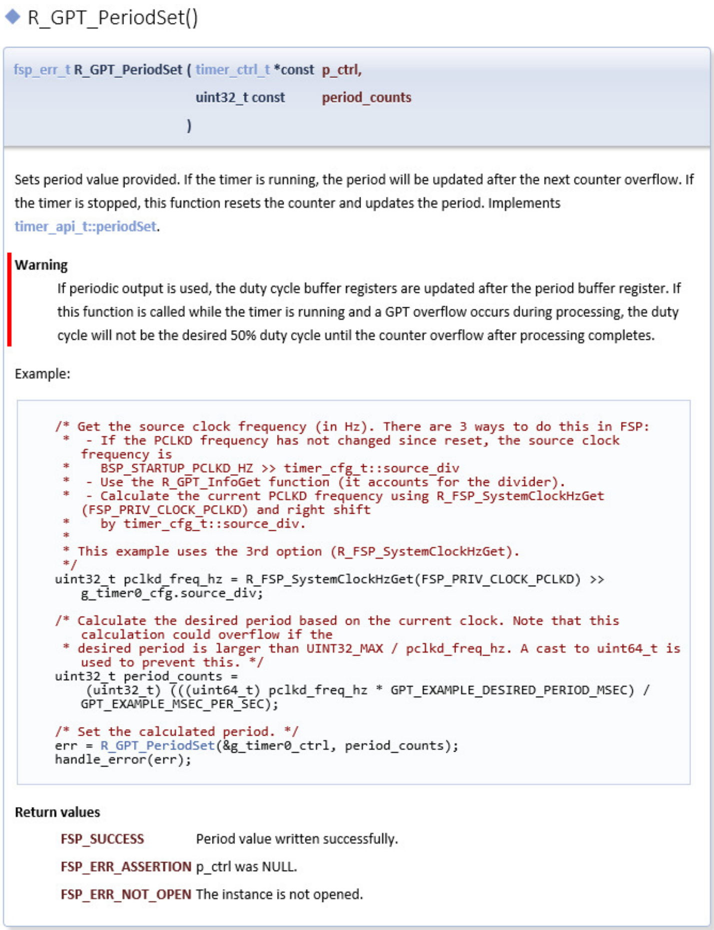
```
R_<module>_<function>(<parameters>);
```

Here are some examples:

```
status = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
status = R_GPT_Start(&g_timer0_ctrl);
status = R_GPT_PeriodSet(&g_timer0_ctrl, period);
status = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
status = R_ADC_InfoGet(&g_adc0_ctrl, &adc_info);
```

HAL Driver API Call Reference Information

Each HAL module has a useful API Reference section that includes key details on each function. The function prototype is presented first, showing the return type (usually fsp_status_t for HAL functions) and the function parameters. A short description and any warnings or notes follow the function definition. In some cases, a code snippet is included to illustrate use of the function. Finally, all possible return values are provided to assist in debugging and error management.


◆ R_GPT_PeriodSet()

```
fsp_err_t R_GPT_PeriodSet ( timer_ctrl_t *const p_ctrl,
                            uint32_t const      period_counts
                          )
```

Sets period value provided. If the timer is running, the period will be updated after the next counter overflow. If the timer is stopped, this function resets the counter and updates the period. Implements timer_api_t::periodSet.

Warning

If periodic output is used, the duty cycle buffer registers are updated after the period buffer register. If this function is called while the timer is running and a GPT overflow occurs during processing, the duty cycle will not be the desired 50% duty cycle until the counter overflow after processing completes.

Example:

```
/* Get the source clock frequency (in Hz). There are 3 ways to do this in FSP:
 *  - If the PCLKD frequency has not changed since reset, the source clock
    frequency is
 *    BSP_STARTUP_PCLKD_HZ >> timer_cfg_t::source_div
 *  - Use the R_GPT_InfoGet function (it accounts for the divider).
 *  - Calculate the current PCLKD frequency using R_FSP_SystemClockHzGet
    (FSP_PRIV_CLOCK_PCLKD) and right shift
 *    by timer_cfg_t::source_div.
 *
 * This example uses the 3rd option (R_FSP_SystemClockHzGet).
 */
uint32_t pclkd_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) >>
    g_timer0_cfg.source_div;

/* Calculate the desired period based on the current clock. Note that this
    calculation could overflow if the
 * desired period is larger than UINT32_MAX / pclkd_freq_hz. A cast to uint64_t is
    used to prevent this. */
uint32_t period_counts =
     (uint32_t) (((uint64_t) pclkd_freq_hz * GPT_EXAMPLE_DESIRED_PERIOD_MSEC) /
    GPT_EXAMPLE_MSEC_PER_SEC);

/* Set the calculated period. */
err = R_GPT_PeriodSet(&g_timer0_ctrl, period_counts);
handle_error(err);
```

Return values

FSP_SUCCESS	Period value written successfully.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

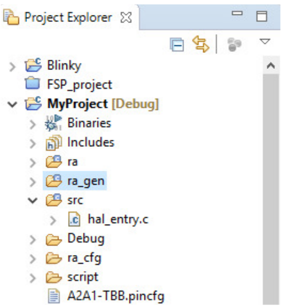
Figure 46: Module Api Reference Section Example

2.2.8.3 RTOS-Independent Applications

To write application code:

1. Add all drivers and modules in the **Stacks** tab and resolve all dependencies flagged by e2 studio such as missing interrupts or drivers.
2. Configure the drivers in the **Properties** view.
3. In the Project Configuration view, click the **Generate Project Content** button.

4. In the **Project Explorer** view, double-click on the src/hal_entry.c file to edit the source file.



Note

All configuration structures necessary for the driver to be called in the application are initialized in ra_gen/hal_data.c.

Warning

Do not modify the files in the directory ra_gen. These files are overwritten every time you push the **Generate Project Content** button.

5. Add your application code here:

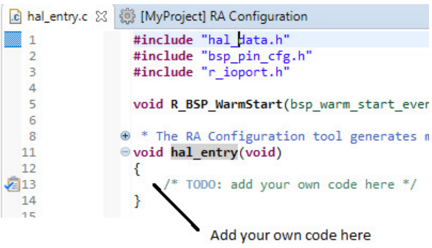


Figure 47: Adding user code to hal_entry.c

6. Build the project without errors by clicking on **Project > Build Project**.

The following tutorial shows how execute the steps above and add application code: Tutorial: Using HAL Drivers - Programming the WDT.

The WDT example is a HAL level application which does not use an RTOS. The user guides for each module also include basic application code that you can add to hal_entry.c.

2.2.8.4 RTOS Applications

To write RTOS-aware application code using RTOS, follow these steps:

1. Add a thread using the **Stacks** tab.
2. Provide a unique name for the thread in the **Properties** view for this thread.

3. Configure all drivers and resources for this thread and resolve all dependencies flagged by e2 studio such as missing interrupts or drivers.
4. Configure the thread objects.
5. Provide unique names for each thread object in the **Properties** view for each object.
6. Add more threads if needed and repeat steps 1 to 5.
7. In the **RA Project Editor**, click the **Generate Project Content** button.

8. In the **Project Explorer** view, double-click on the src/my_thread_1_entry.c file to edit the source file.

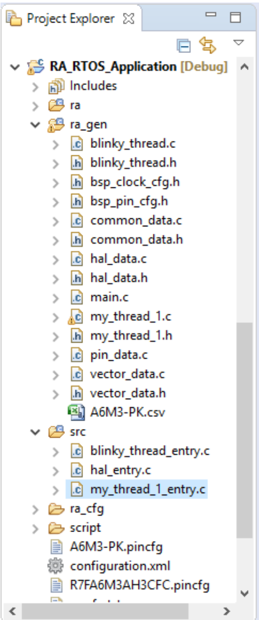


Figure 48: Generated files for an RTOS application

Note

All configuration structures necessary for the driver to be called in the application are initialized in ra_gen/my_thread_1.c and my_thread_2.c

Warning

Do not modify the files in the directory ra_gen. These files are overwritten every time you push the **Generate Project Content** button.

9. Add your application code here:

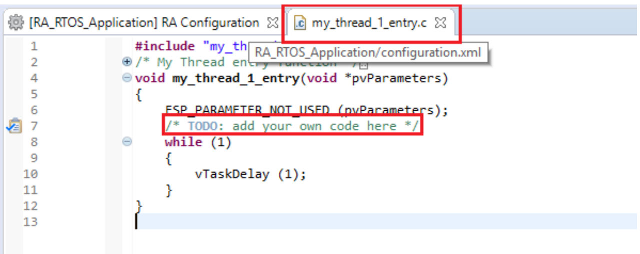


Figure 49: Adding user code to my_thread_1.entry

10. Repeat steps 1 to 9 for the next thread.

11. Build your project without errors by clicking on **Project > Build Project**.

2.2.8.5 Additional Resources for Application Development

Example Projects

A wide variety of Example Projects for FSP and RA MCUs is available on the GitHub site here: https://github.com/renesas/ra-fsp-examples. Example projects are organized by target kit so it is easy to find all the examples for your kit of choice.

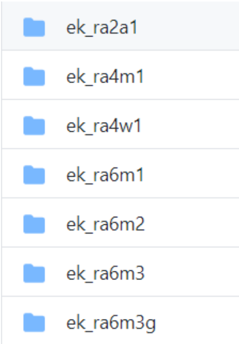


Figure 50: FSP Example Projects Organized by Kit

Projects are available as both downloadable zip files and as project source files. Typically, there is a project for each module. New example projects are being added periodically, so check back if a particular module isn't yet available.

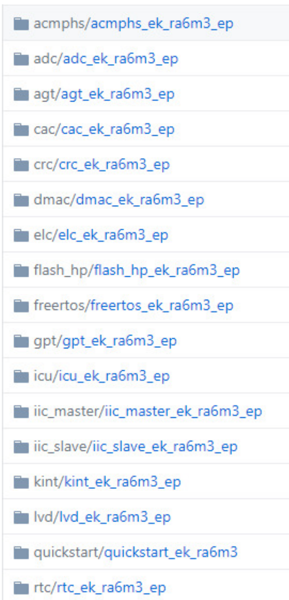


Figure 51: A Selection of Example Projects Available on GitHub

Quick Labs

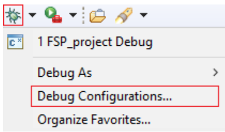
A variety of Hands-on Do It Yourself labs are available on the Renesas RA and FSP Knowledge Base. Quick FSP Labs target the EK-RA6M3 kit and typically require only 15 minutes to complete. Each lab covers a couple related development tools and techniques like Autocomplete, Developer Assistance, console I/O over RTT, and Visual Expressions, that can speed up the development process. A list of all available Quick Labs can be found here: https://en-support.renesas.com/knowledgeBase/19450948

2.2.9 Debugging the Project

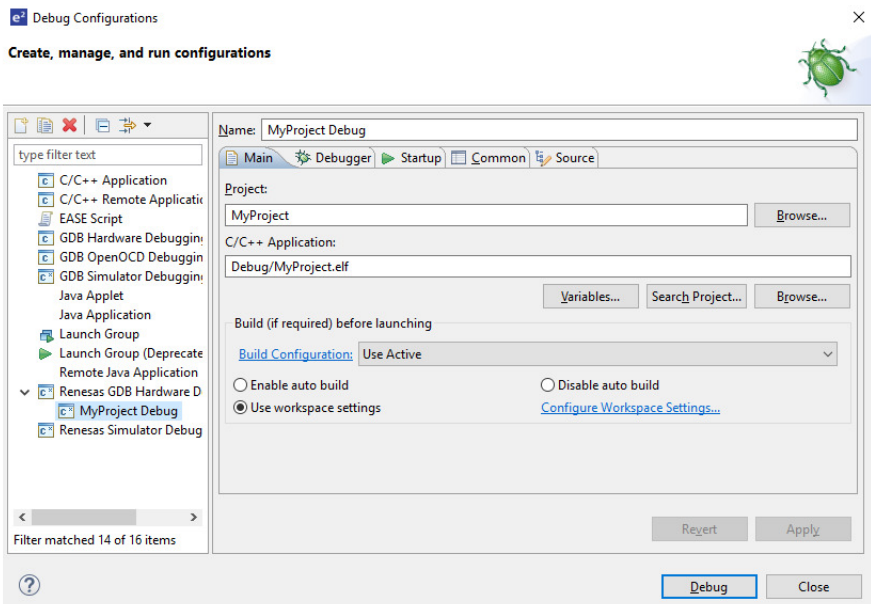
Once your project builds without errors, you can use the Debugger to download your application to the board and execute it.

To debug an application follow these steps:

1. On the drop-down list next to the debug icon, select **Debug Configurations**.



2. In the **Debug Configurations** view, click on your project listed as **MyProject Debug**.



3. Connect the board to your PC via either a standalone Segger J-Link debugger, a Segger J-Link On-Board (included on all RA EKs), or an E2 or E2 Lite and click **Debug**.

Note

For details on using J-Link and connecting the board to the PC, see the Quick Start Guide included in the RA MCU Kit.

2.2.10 Modifying Toolchain Settings

There are instances where it may be necessary to make changes to the toolchain being used (for example, to change optimization level of the compiler or add a library to the linker). Such modifications can be made from within e2 studio through the menu **Project > Properties > Settings** when the project is selected. The following screenshot shows the settings dialog for the GNU Arm toolchain. This dialog will look slightly different depending upon the toolchain being used.

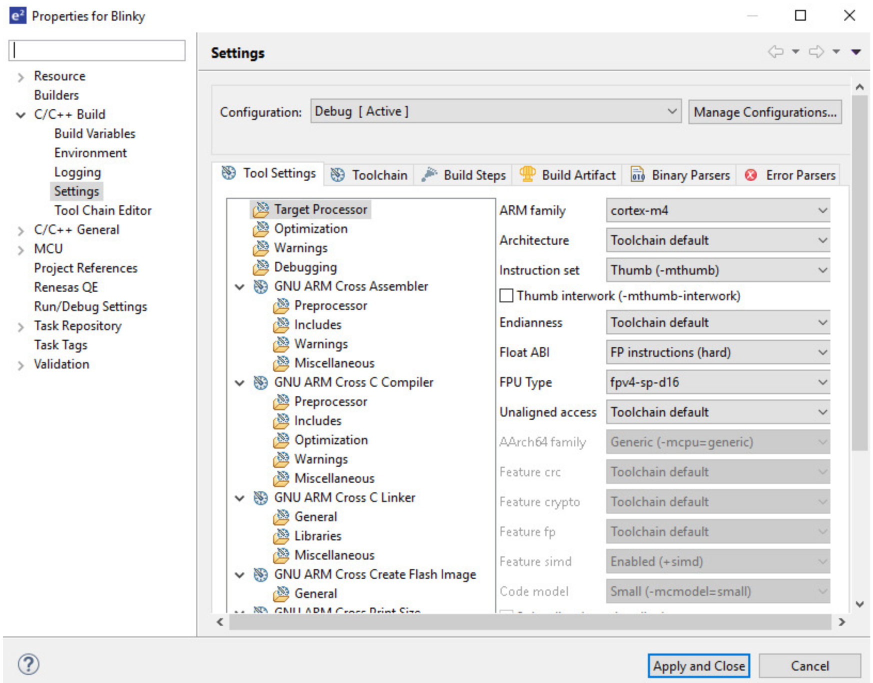


Figure 52: e2 studio Project toolchain settings

The scope for the settings is project scope which means that the settings are valid only for the project being modified.

The settings for the linker which control the location of the various memory sections are contained in a script file specific for the device being used. This script file is included in the project when it is created and is found in the script folder (for example, /script/a6m3.ld).

2.2.11 Creating RA project with ARM Compiler 6 in e2 studio

e2 studio does not include the ARM Compiler 6 (AC6) toolchain by default. Follow the steps below to integrate AC6 into e2 studio and create an AC6 RA project.

Note

> *It is assumed that the user is already familiar with RA project creation in e2 studio. e2 studio does not include ARM Compiler 6 (AC6) toolchain by default.*

Steps 1 through 8 describe the process for integrating ARM Compiler 6 into e2 studio.

1. Download, install, and configure license for the AC6 toolchain (https://developer.arm.com/tools-and-software/embedded/arm-compiler/downloads/version-6).
2. Launch e2 studio.
3. Go to **Window > Preferences > Toolchains**.

4. Click **Add**.

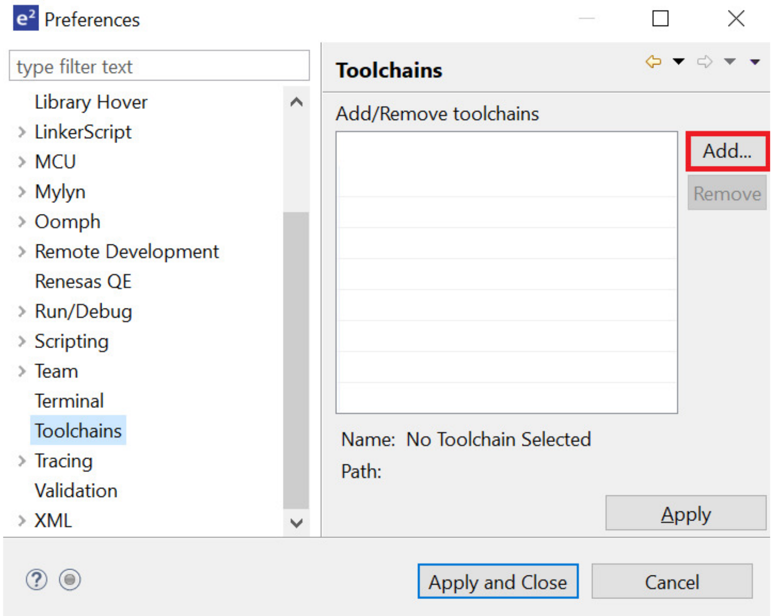


Figure 53: Add Toolchain

5. Browse to the path where AC6 toolchain is installed and select the \bin folder. Click **Next**.

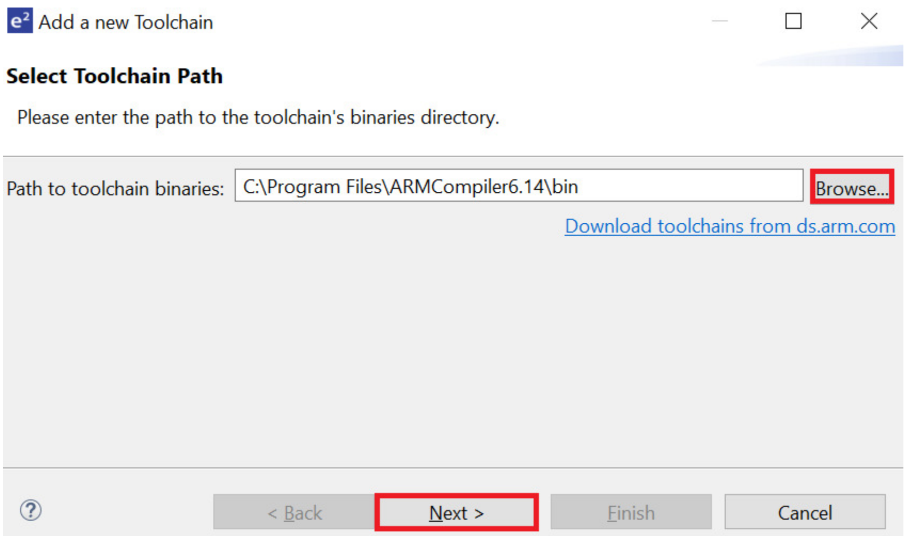


Figure 54: Browse to AC6 Compiler

6. Toolchain information in displayed. Click **Finish**.

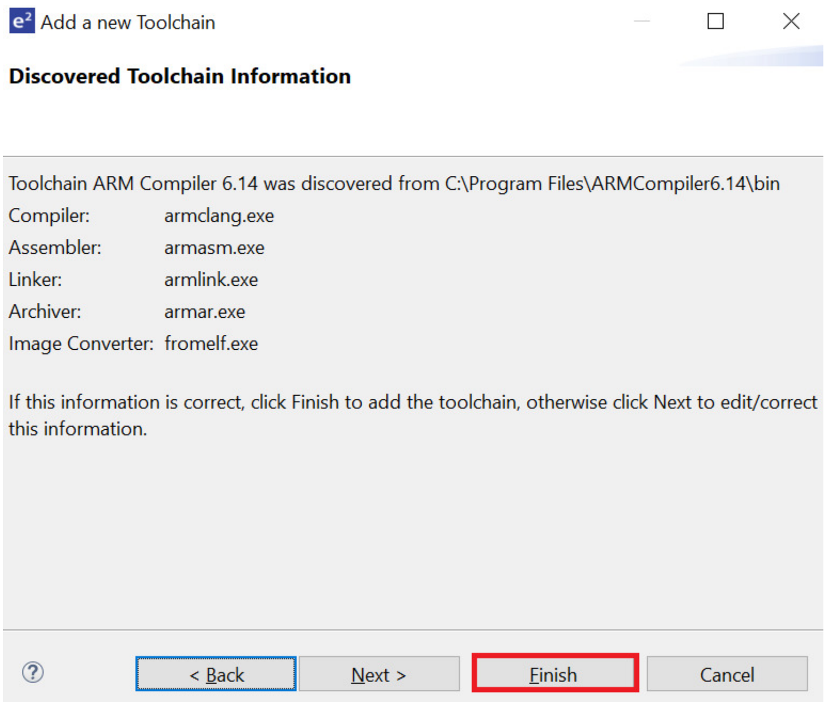


Figure 55: Toolchain Information

7. Click **Apply and Close**.

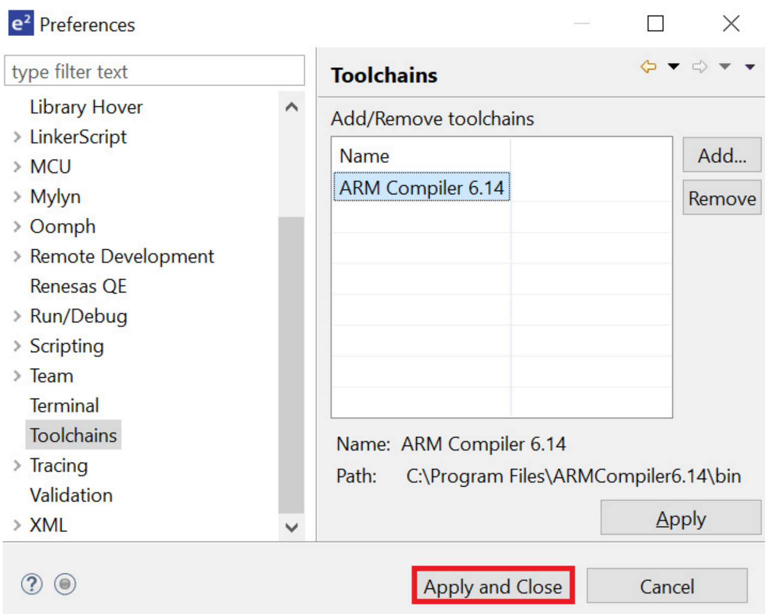


Figure 56: Apply and Close

8. Click **Restart Eclipse** when prompted.

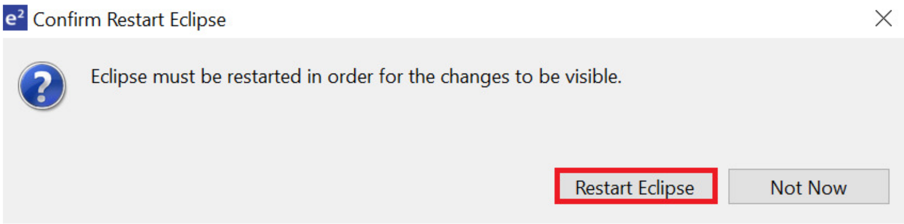


Figure 57: Restart Eclipse

9. When creating a new RA C/C++ project, select **ARM Compiler 6** included in the Toolchains section.

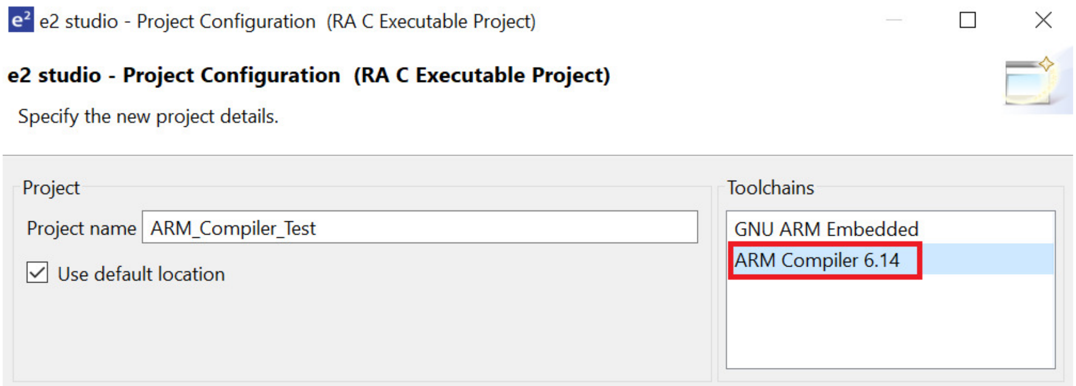


Figure 58: Select Arm Compiler

2.2.12 Importing an Existing Project into e2 studio

1. Start by opening e2 studio.
2. Open an existing Workspace to import the project and skip to step d. If the workspace doesn't exist, proceed with the following steps:

   a. At the end of e2 studio startup, you will see the Workspace Launcher Dialog box as shown in the following figure.

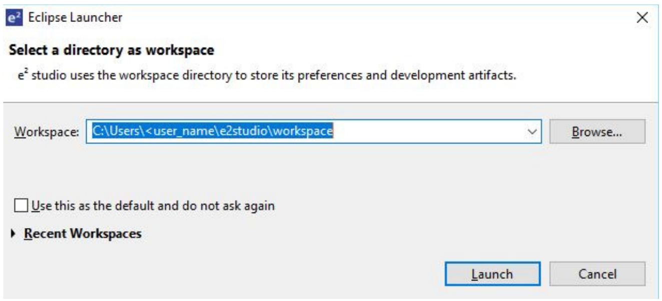


Figure 59: Workspace Launcher dialog

b. Enter a new workspace name in the Workspace Launcher Dialog as shown in the following figure. e2 studio creates a new workspace with this name.

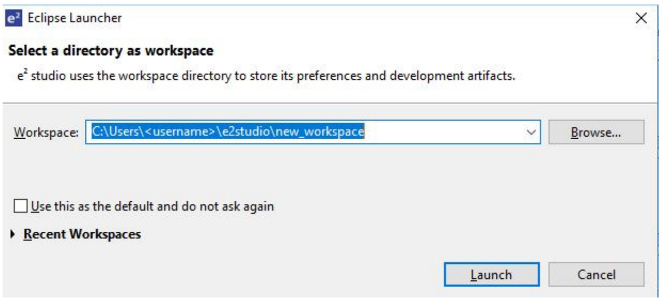


Figure 60: Workspace Launcher dialog - Select Workspace

c. Click **Launch**.

d. When the workspace is opened, you may see the Welcome Window. Click on the **Workbench** arrow button to proceed past the Welcome Screen as seen in the following figure.

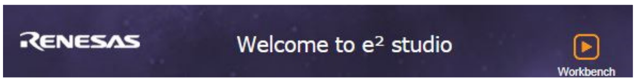


Figure 61: Workbench arrow button

3. You are now in the workspace that you want to import the project into. Click the **File** menu in the menu bar, as shown in the following figure.

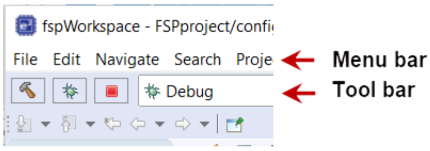


Figure 62: Menu and tool bar

4. Click **Import** on the **File** menu or in the menu bar, as shown in the following figure.

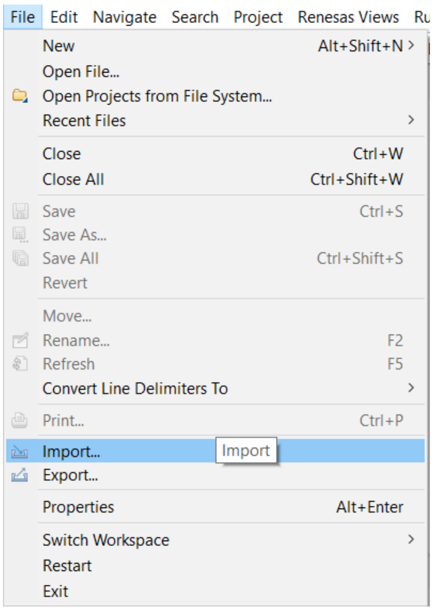


Figure 63: File drop-down menu

5. In the **Import** dialog box, as shown in the following figure, choose the **General** option, then **Existing Projects into Workspace**, to import the project into the current workspace.

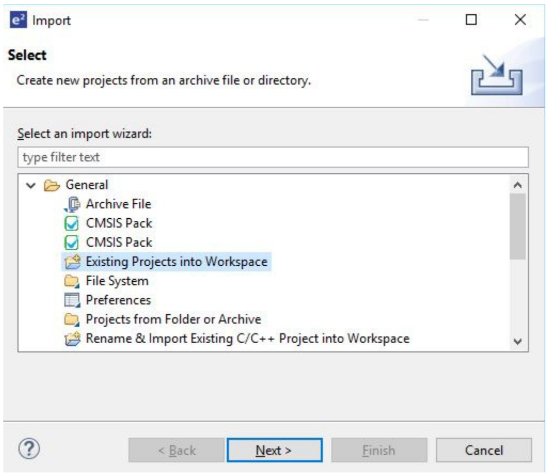


Figure 64: Project Import dialog with "Existing Projects into Workspace" option selected

6. Click **Next**.
7. To import the project, use either **Select archive file** or **Select root directory**.

    a. Click **Select archive file** as shown in the following figure.

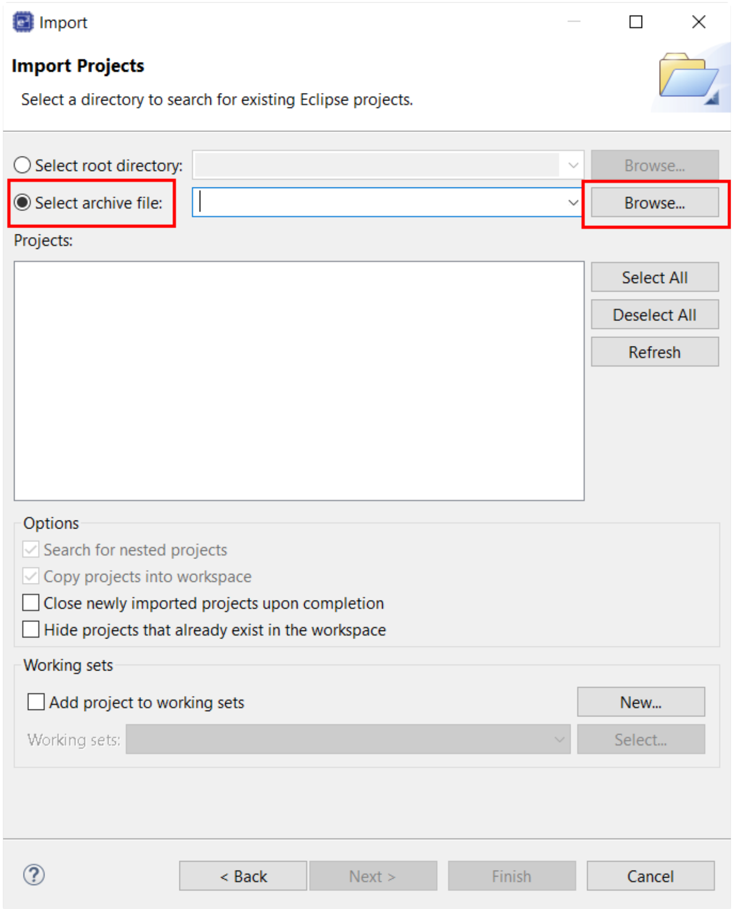


Figure 65: Import Existing Project dialog 1 - Select archive file

b. Click **Select root directory** as shown in the following figure.

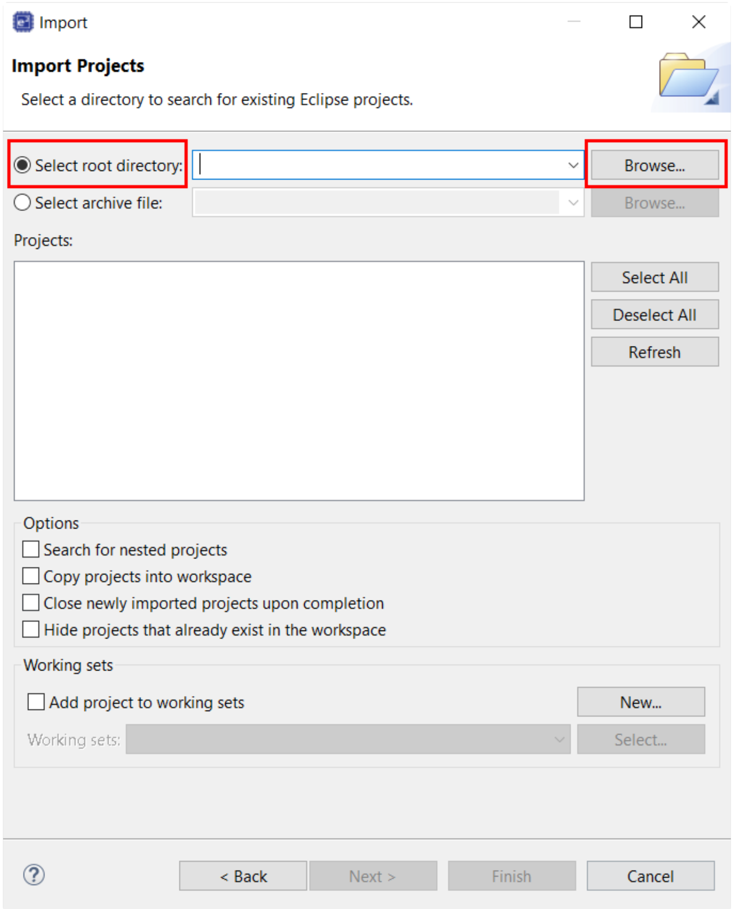


Figure 66: Import Existing Project dialog 1 - Select root directory

8. Click **Browse**.
9. For **Select archive file**, browse to the folder where the zip file for the project you want to import is located. For **Select root directory**, browse to the project folder that you want to import.
10. Select the file for import. In our example, it is CAN_HAL_MG_AP.zip or CAN_HAL_MG_AP.
11. Click **Open**.

12. Select the project to import from the list of **Projects**, as shown in the following figure.

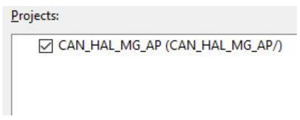


Figure 67: Import Existing Project dialog 2

13. Click **Finish** to import the project.

2.3 Tutorial: Your First RA MCU Project - Blinky

2.3.1 Tutorial Blinky

The goal of this tutorial is to quickly get acquainted with the Flexible Platform by moving through the steps of creating a simple application using e2 studio and running that application on an RA MCU board.

2.3.2 What Does Blinky Do?

The application used in this tutorial is Blinky, traditionally the first program run in a new embedded development environment.

Blinky is the "Hello World" of microcontrollers. If the LED blinks you know that:

- The toolchain is setup correctly and builds a working executable image for your chip.
- The debugger has installed with working drivers and is properly connected to the board.
- The board is powered up and its jumper and switch settings are probably correct.
- The microcontroller is alive, the clocks are running, and the memory is initialized.

The Blinky example application used in this tutorial is designed to run the same way on all boards offered by Renesas that hold the RA microcontroller. The code in Blinky is completely board independent. It does the work by calling into the BSP (board support package) for the particular board it is running on. This works because:

- Every board has at least one LED connected to a GPIO pin.
- That one LED is always labelled LED1 on the silk screen.
- Every BSP supports an API that returns a list of LEDs on a board, and their port and pin assignments.

2.3.3 Prerequisites

To follow this tutorial, you need:

- Windows based PC
- e2 studio
- Flexible Software Package
- An RA MCU board kit

2.3.4 Create a New Project for Blinky

The creation and configuration of an RA MCU project is the first step in the creation of an application. The base RA MCU pack includes a pre-written Blinky example application that is simple and works on all Renesas RA MCU boards.

Follow these steps to create an RA MCU project:

1. In e2 studio, click **File > New > C/C++ Project** and select **Renesas RA** and **Renesas RA C/C++ Project**.
2. Assign a name to this new project. Blinky is a good name to use for this tutorial.

3. Click **Next**. The **Project Configuration** window shows your selection.

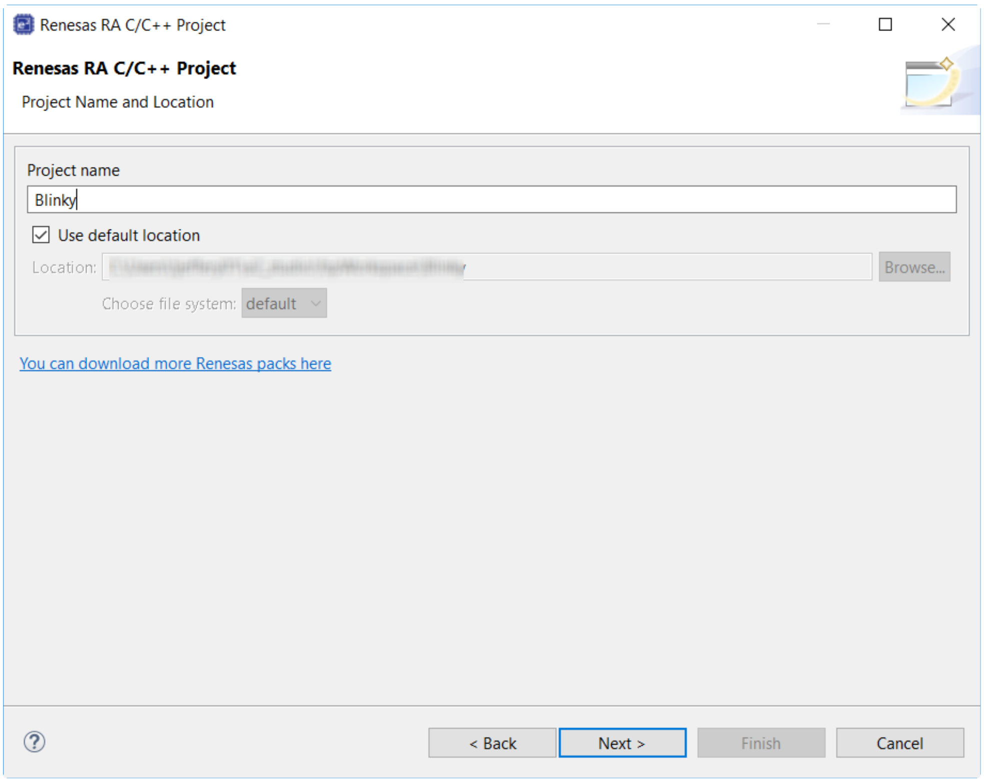


Figure 68: e2 studio Project Configuration window (part 1)

4. Select the board support package by selecting the name of your board from the **Device Selection** drop-down list and click **Next**.

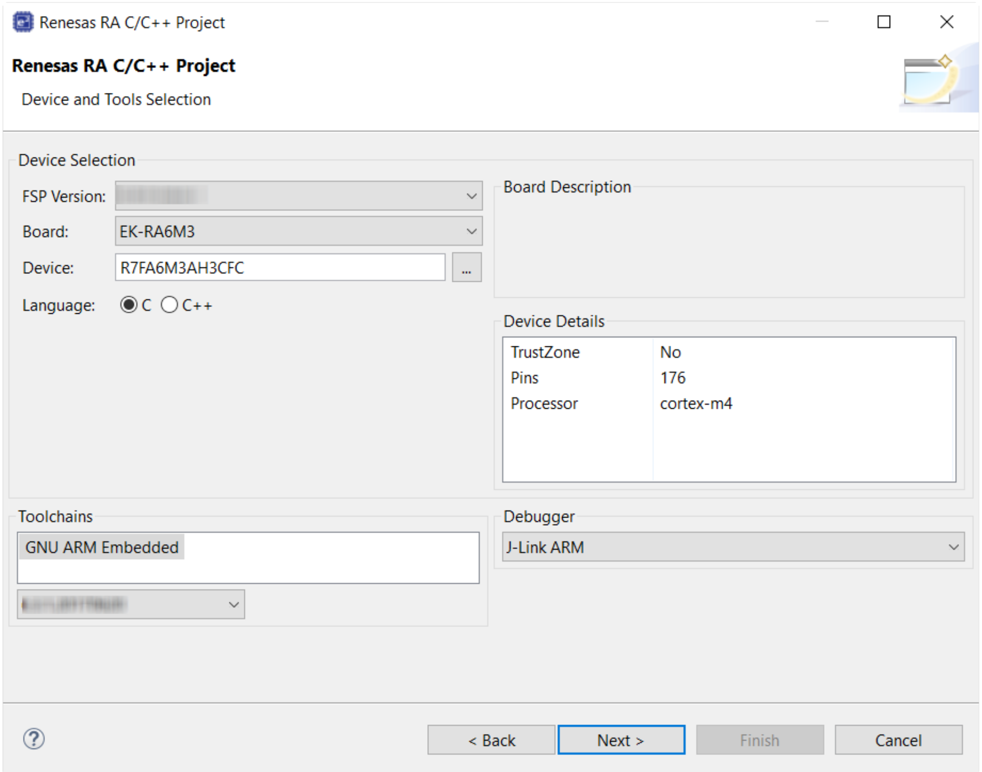


Figure 69: e2 studio Project Configuration window (part 2)

5. Select the build artifact and RTOS.

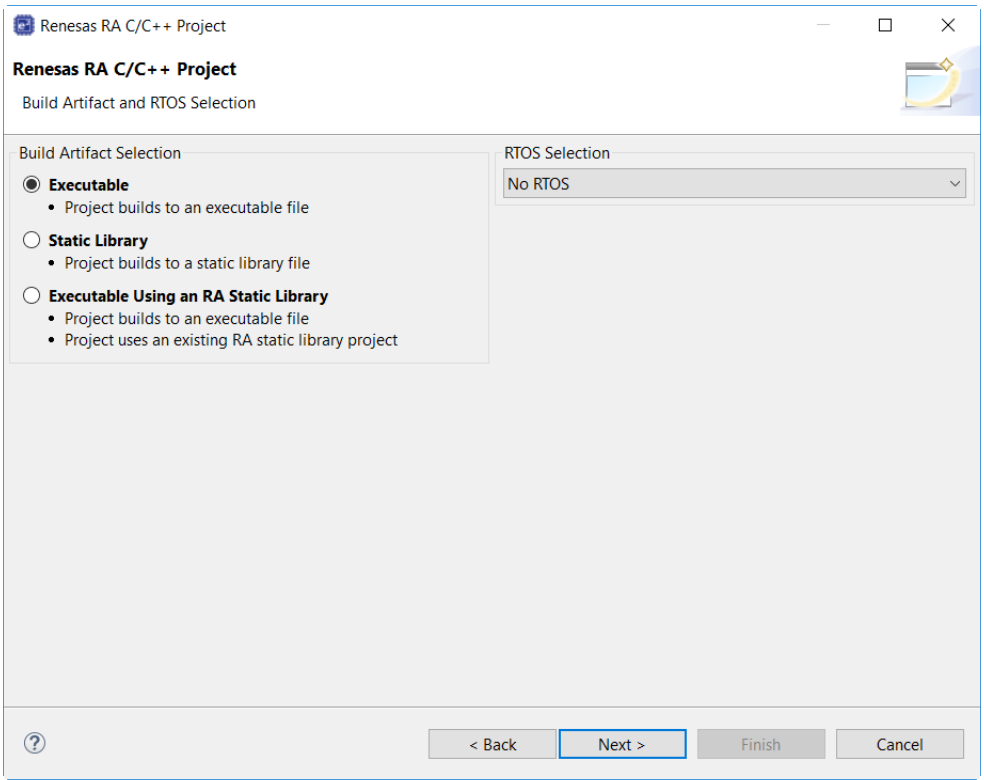


Figure 70: e2 studio Project Configuration window (part 3)

6. Select the Blinky template for your board and click **Finish**.

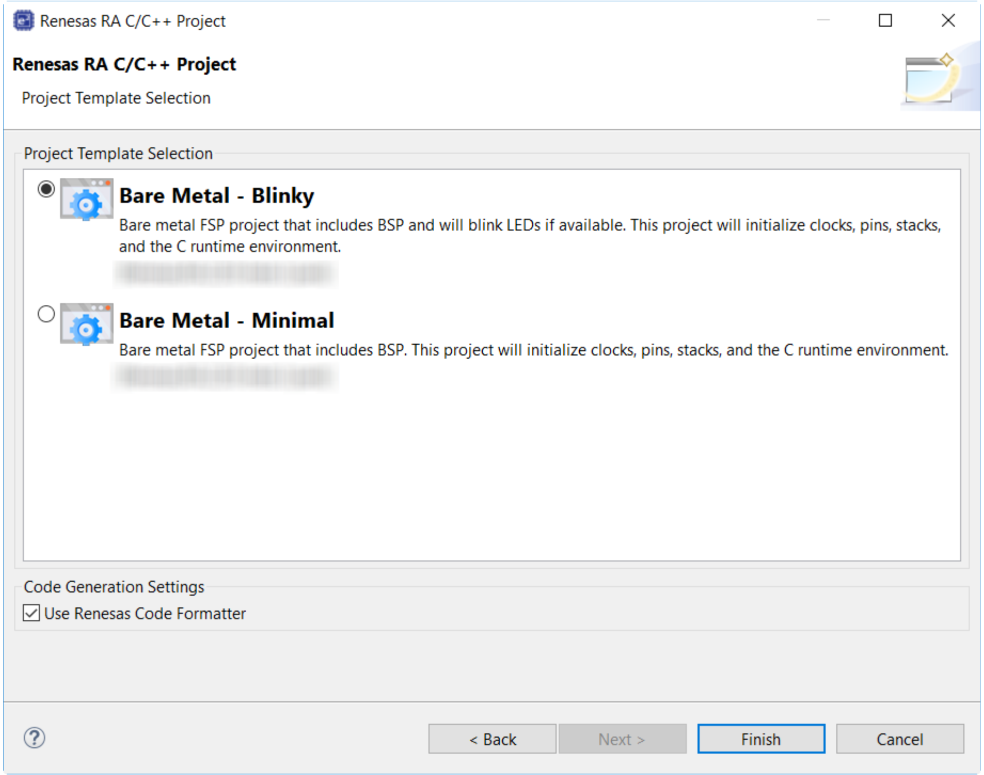


Figure 71: e2 studio Project Configuration window (part 4)

Once the project has been created, the name of the project will show up in the **Project Explorer** window of e2 studio. Now click the **Generate Project Content** button in the top right corner of the **Project Configuration** window to generate your board specific files.

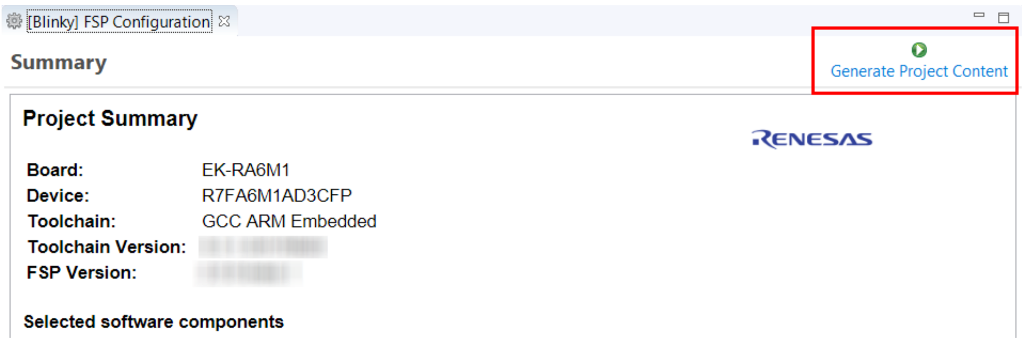


Figure 72: e2 studio Project Configuration tab

Your new project is now created, configured, and ready to build.

2.3.4.1 Details about the Blinky Configuration

The **Generate Project Content** button creates configuration header files, copies source files from templates, and generally configures the project based on the state of the **Project Configuration** screen.

For example, if you check a box next to a module in the **Components** tab and click the **Generate Project Content** button, all the files necessary for the inclusion of that module into the project will be copied or created. If that same check box is then unchecked those files will be deleted.

2.3.4.2 Configuring the Blinky Clocks

By selecting the Blinky template, the clocks are configured by e2 studio for the Blinky application. The clock configuration tab (see Configuring Clocks) shows the Blinky clock configuration. The Blinky clock configuration is stored in the BSP clock configuration file (see BSP Clock Configuration).

2.3.4.3 Configuring the Blinky Pins

By selecting the Blinky template, the GPIO pins used to toggle the LED1 are configured by e2 studio for the Blinky application. The pin configuration tab shows the pin configuration for the Blinky application (see Configuring Pins). The Blinky pin configuration is stored in the BSP configuration file (see BSP Pin Configuration).

2.3.4.4 Configuring the Parameters for Blinky Components

The Blinky project automatically selects the following HAL components in the Components tab:

- r_ioport

To see the configuration parameters for any of the components, check the **Properties** tab in the HAL window for the respective driver (see Adding and Configuring HAL Drivers).

2.3.4.5 Where is main()?

The main function is located in < project >/ra_gen/main.c. It is one of the files that are generated during the project creation stage and only contains a call to hal_entry(). For more information on generated files, see Adding and Configuring HAL Drivers.

2.3.4.6 Blinky Example Code

The blinky application is stored in the hal_entry.c file. This file is generated by e2 studio when you select the Blinky Project template and is located in the project's src/ folder.

The application performs the following steps:

1. Get the LED information for the selected board by bsp_leds_t structure.
2. Define the output level HIGH for the GPIO pins controlling the LEDs for the selected board.
3. Get the selected system clock speed and scale down the clock, so the LED toggling can be observed.
4. Toggle the LED by writing to the GPIO pin with R_BSP_PinWrite((bsp_io_port_pin_t) pin, pin_level);

2.3.5 Build the Blinky Project

Highlight the new project in the **Project Explorer** window by clicking on it and build it.

There are three ways to build a project:

1. Click on **Project** in the menu bar and select **Build Project**.
2. Click on the hammer icon.
3. Right-click on the project and select **Build Project**.

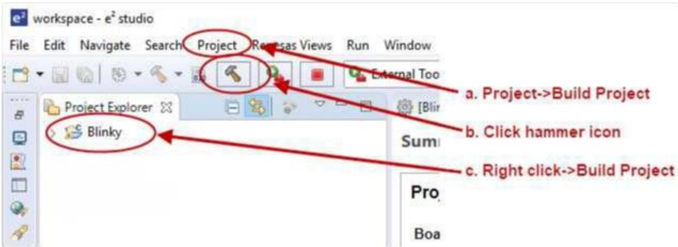


Figure 73: e2 studio Project Explorer window

Once the build is complete a message is displayed in the build **Console** window that displays the final image file name and section sizes in that image.

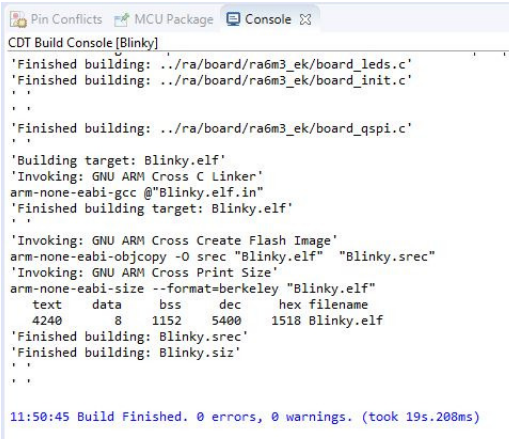


Figure 74: e2 studio Project Build console

2.3.6 Debug the Blinky Project

2.3.6.1 Debug prerequisites

To debug the project on a board, you need

- The board to be connected to e2 studio
- The debugger to be configured to talk to the board
- The application to be programmed to the microcontroller

Applications run from the internal flash of your microcontroller. To run or debug the application, the application must first be programmed to the microcontroller's flash. There are two ways to do this:

- JTAG debugger
- Built-in boot-loader via UART or USB

Some boards have an on-board JTAG debugger and others require an external JTAG debugger connected to a header on the board.

Refer to your board's user manual to learn how to connect the JTAG debugger to e2 studio.

2.3.6.2 Debug steps

To debug the Blinky application, follow these steps:

1. Configure the debugger for your project by clicking **Run > Debugger Configurations ...**

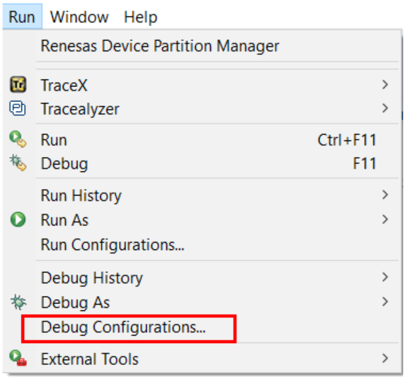


Figure 75: e2 studio Debug icon

or by selecting the drop-down menu next to the bug icon and selecting **Debugger Configurations ...**

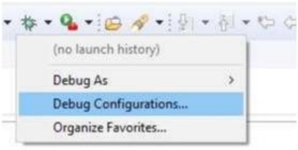


Figure 76: e2 studio Debugger Configurations selection option

2. Select your debugger configuration in the window. If it is not visible then it must be created by clicking the **New** icon in the top left corner of the window. Once selected, the **Debug Configuration** window displays the Debug configuration for your Blinky project.

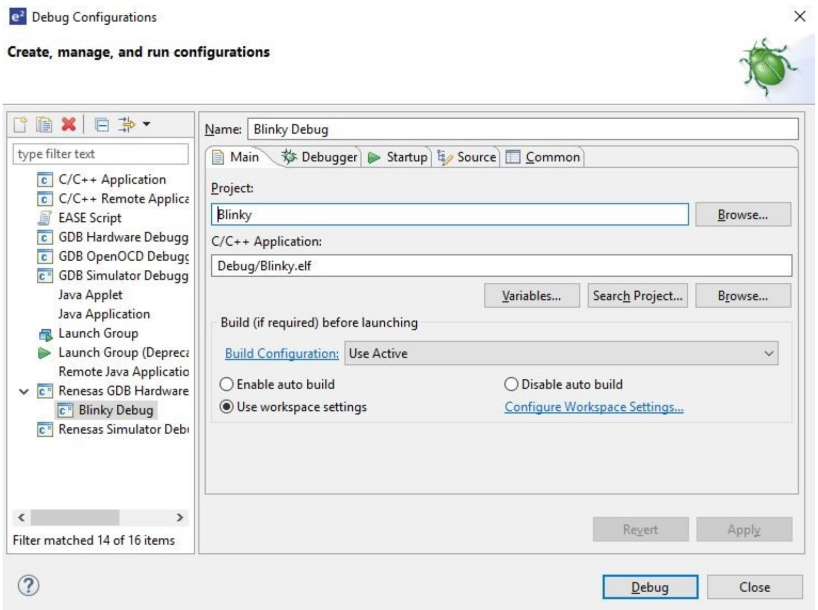
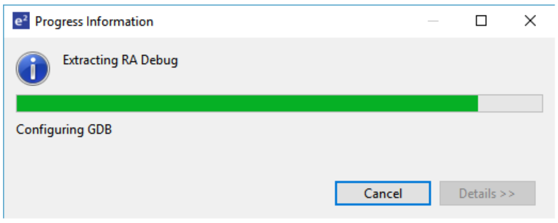


Figure 77: e2 studio Debugger Configurations window with Blinky project

3. Click **Debug** to begin debugging the application.

4. Extracting RA Debug.



2.3.6.3 Details about the Debug Process

In debug mode, e2 studio executes the following tasks:

1. Downloading the application image to the microcontroller and programming the image to the internal flash memory.
2. Setting a breakpoint at main().
3. Setting the stack pointer register to the stack.
4. Loading the program counter register with the address of the reset vector.
5. Displaying the startup code where the program counter points to.

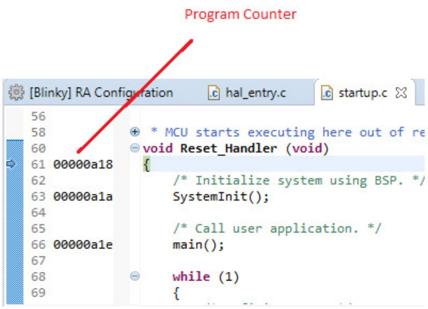


Figure 78: e2 studio Debugger memory window

2.3.7 Run the Blinky Project

While in Debug mode, click **Run > Resume** or click on the **Play** icon twice.

Figure 79: e2 studio Debugger Play icon

The LEDs on the board marked LED1, LED2, and LED3 should now be blinking.

2.4 Tutorial: Using HAL Drivers - Programming the WDT

2.4.1 Application WDT

This tutorial illustrates the creation of a simple application that uses the Watchdog Timer module to monitor program operation. The tutorial shows each step in the development process and in particular identifies the auto-generated files and project structure created when using FSP and its GUI based configurator. The level of detail provided here is more than is normally needed during development but can be helpful in explaining how FSP works behind the scenes to simplify your work.

This application makes use of the following FSP modules:

- MCU Board Support Package
- Watchdog Timer (r_wdt)
- I/O Ports (r_ioport)

2.4.2 Creating a WDT Application Using the RA MCU FSP and e2 studio

2.4.2.1 Using the FSP and e2 studio

The Flexible Software Package (FSP) from Renesas provides a complete driver library for developing RA MCU applications. The FSP provides Hardware Abstraction Layer (HAL) drivers, Board Support Package (BSP) drivers for the developer to use to create applications. The FSP is integrated into Renesas e2 studio based on eclipse providing build (editor, compiler and linker) and debug phases with an extended GNU Debug (GDB) interface.

2.4.2.2 The WDT Application

The flowchart for the WDT application is shown below.

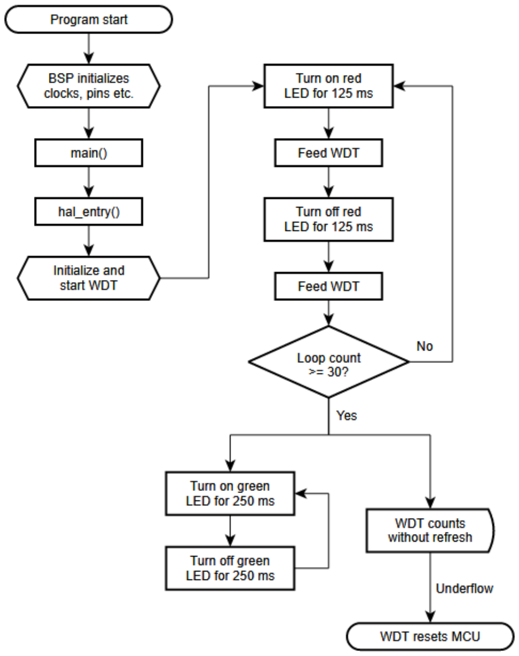


Figure 80: WDT Application flow diagram

2.4.2.3 WDT Application flow

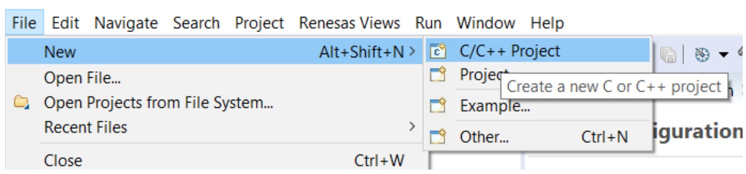
The main sections of the WDT application are:

1. The BSP initializes the clocks, pins and other elements of the MCU readying the application to run.
2. main() calls hal_entry(). The function hal_entry() is created by the FSP with a placeholder for user code. The code for the WDT is added to this function.
3. Initialize the WDT, but do not start it.
4. Start the WDT by refreshing it.
5. In the first loop the red LED flashes 30 times and refreshes the watchdog each time the LED state is changed.
6. In the second loop, the green LED flashes, but the program DOES NOT refresh the watchdog. After the watchdog timeout period the device will reset which can be observed by the red LED flashing again as the sequence repeats.

2.4.3 Creating the Project with e2 studio

Start e2 studio and choose a workspace folder in the Workspace Launcher. Configure a new RA MCU project as follows.

1. Select **File > New > RA C/C++ Project**. Then select the template for the project.



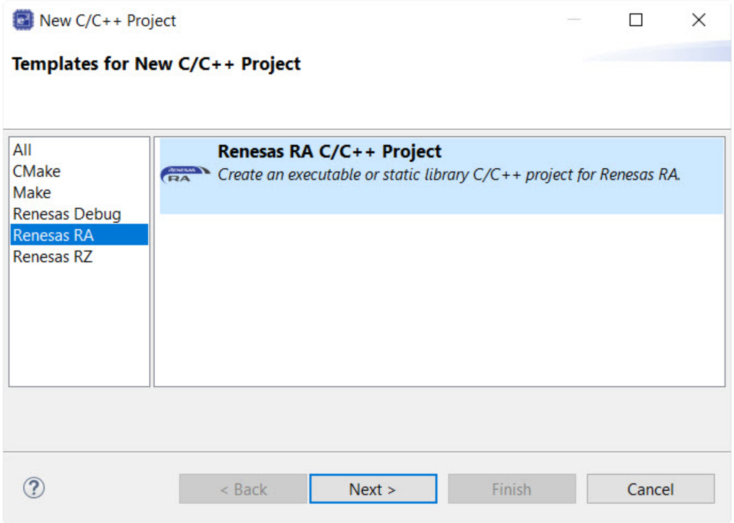


Figure 81: Creating a new project

2. In the e2 studio Project **Configuration (RA Project)** window enter a project name, for example, WDT_Application. In addition, select the toolchain. If you want to choose new locations for the project unselect **Use default location**. Click **Next**.

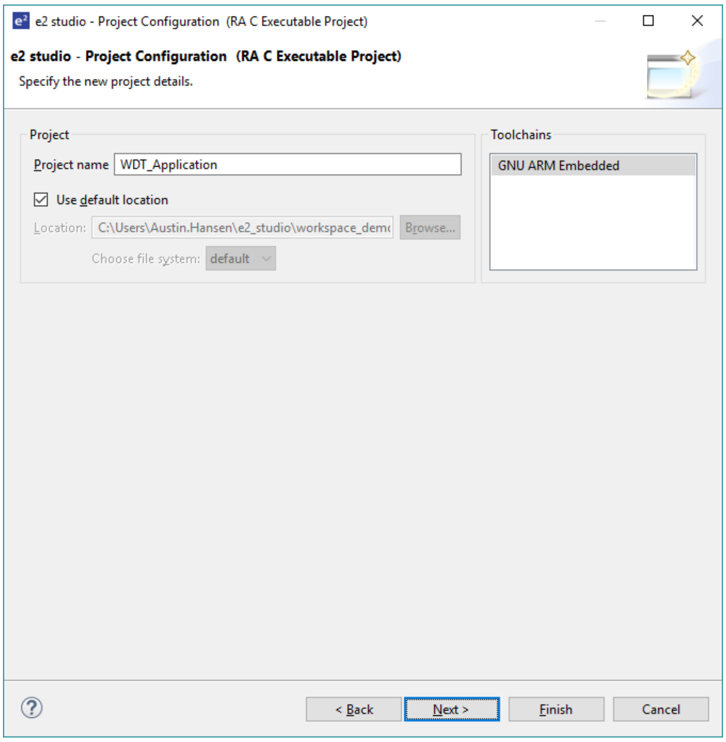


Figure 82: Project configuration (part 1)

3. This application runs on the EK-RA6M3 board. So, for the **Board** select **EK-RA6M3**.

This will automatically populate the **Device** drop-down with the correct device used on this board. Select the **Toolchain** version. Select **J-Link ARM** as the **Debugger**. Click **Next** to configure the project.

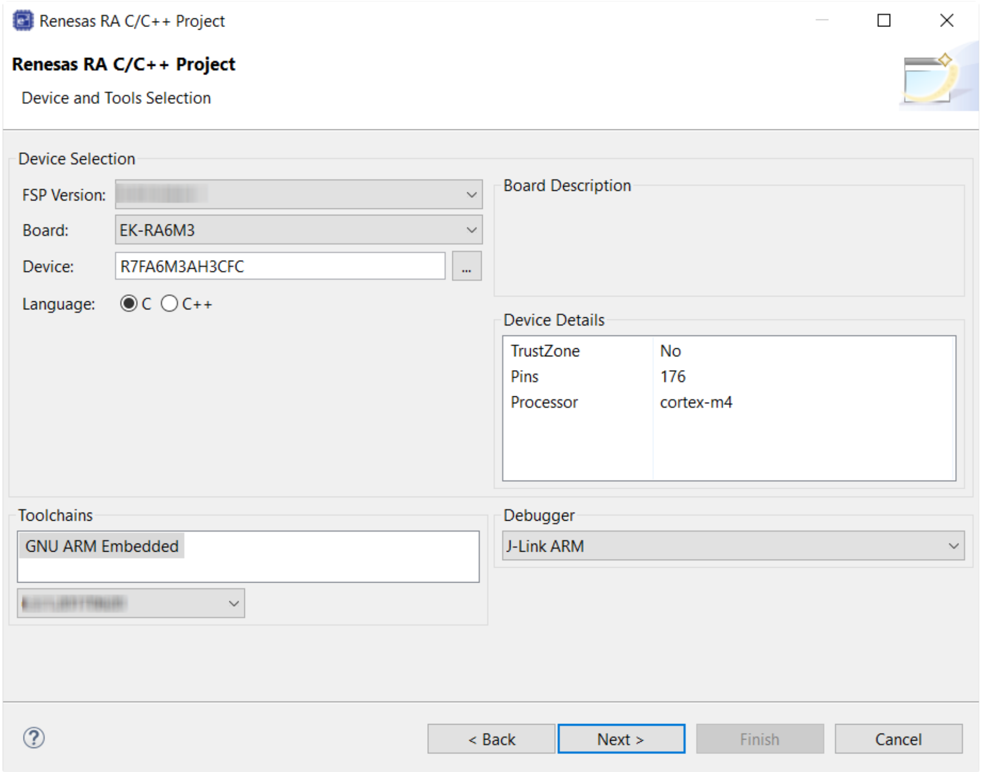


Figure 83: Project configuration (part 2)

The project template is now selected. As no RTOS is required select **Bare Metal - Blinky**.

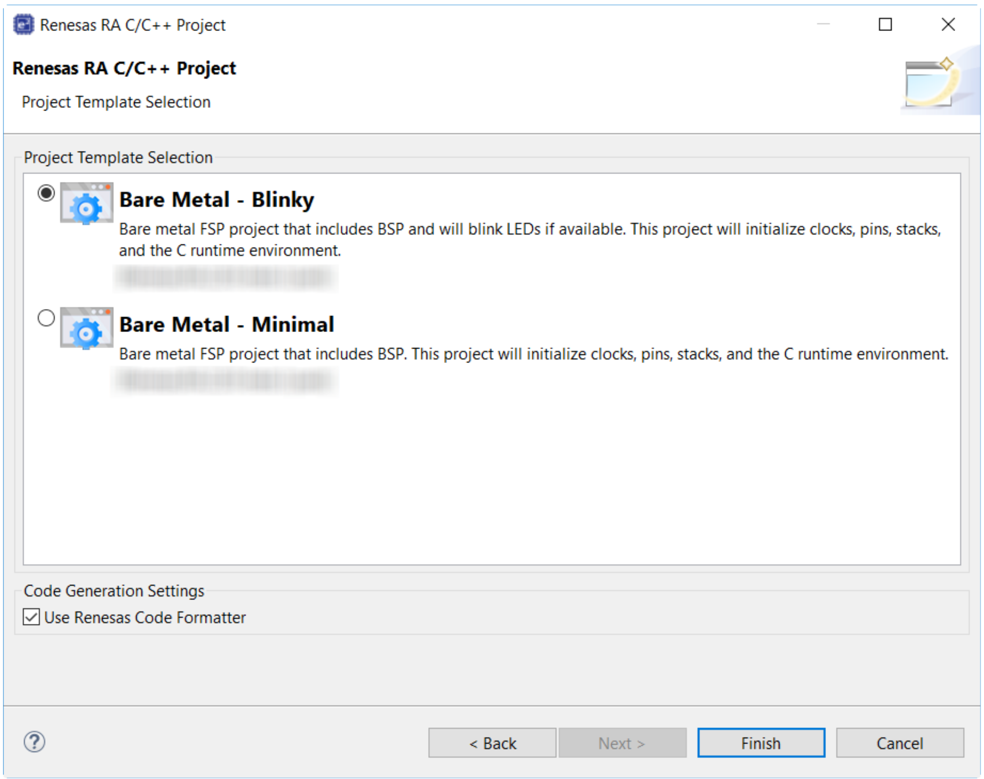


Figure 84: Project configuration (part 3)

4. Click **Finish**.

   e2 studio creates the project and opens the **Project Explorer** and **Project Configuration Settings** views with the **Summary** page showing a summary of the project configuration.

2.4.4 Configuring the Project with e2 studio

e2 studio simplifies and accelerates the project configuration process by providing a GUI interface for selecting the options to configure the project.

e2 studio offers a selection of perspectives presenting different windows to the user depending on the operation in progress. The default perspectives are **C/C++**, **FSP Configuration** and **Debug**. The perspective can be changed by selecting a new one from the buttons at the top right.

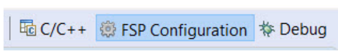


Figure 85: Selecting a perspective

The **C/C++** perspective provides a layout selected for code editing. The **FSP Configuration** perspective provides elements for configuring a RA MCU project, and the **Debug** perspective provides a view suited for debugging.

1. In order to configure the project settings ensure the **FSP Configuration** perspective is selected.

2. Ensure the **Project Configuration [WDT Application]** is open. It is already open if the Summary information is visible. To open the Project Configuration now or at any time make sure the **RA Configuration** perspective is selected and double-click on the configuration.xml file in the Project Explorer pane on the right side of e2 studio.

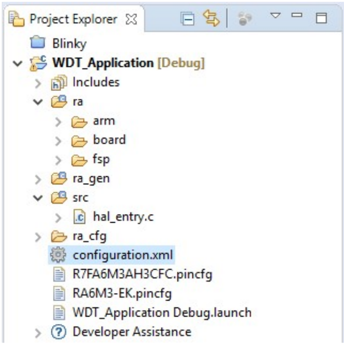


Figure 86: RA MCU Project Configuration Settings

At the base of the Project Configuration view there are several tabs for configuring the project. A project may require changes to some or all of these tabs. The tabs are shown below.

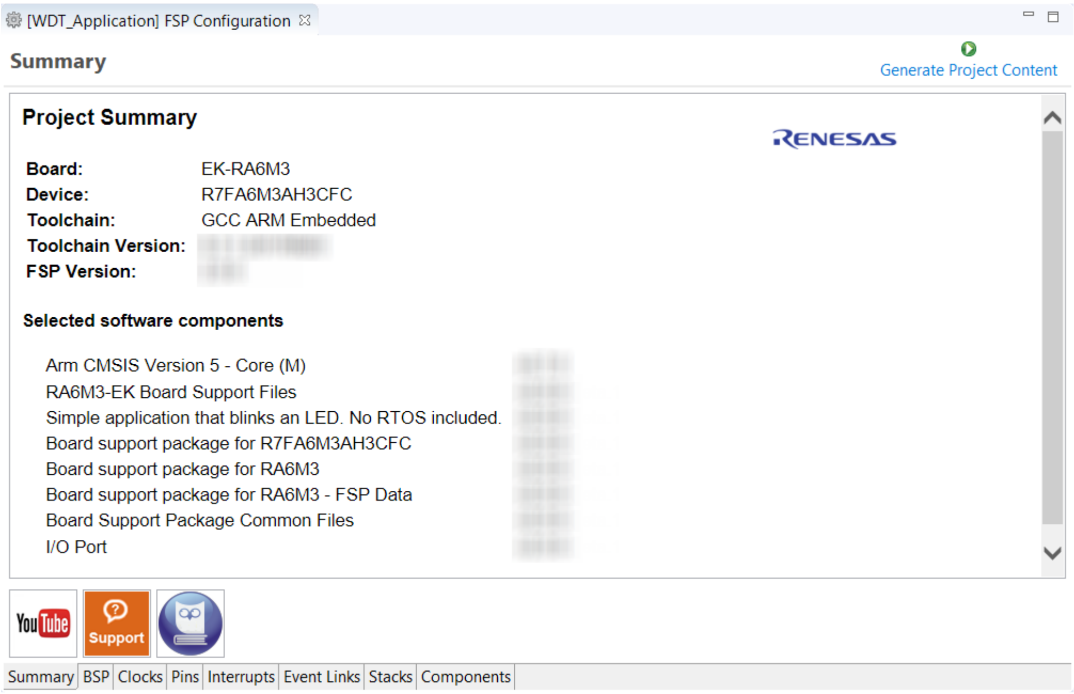


Figure 87: Project Configuration Tabs

2.4.4.1 BSP Tab

The **BSP** tab allows the Board Support Package (BSP) options to be modified from their defaults. For this particular WDT project no changes are required. However, if you want to use the WDT in auto-start mode, you can configure the settings of the OFS0 (Option Function Select Register 0) register in the **BSP** tab. See the RA Hardware User's Manual for details on the WDT autostart mode.

2.4.4.2 Clocks Tab

The **Clocks** tab presents a graphical view of the clock tree of the device. The drop-down boxes in the GUI enables configuration of the various clocks. The WDT uses PCLCKB. The default output frequency for this clock is 60 MHz. Ensure this clock is outputting this value.

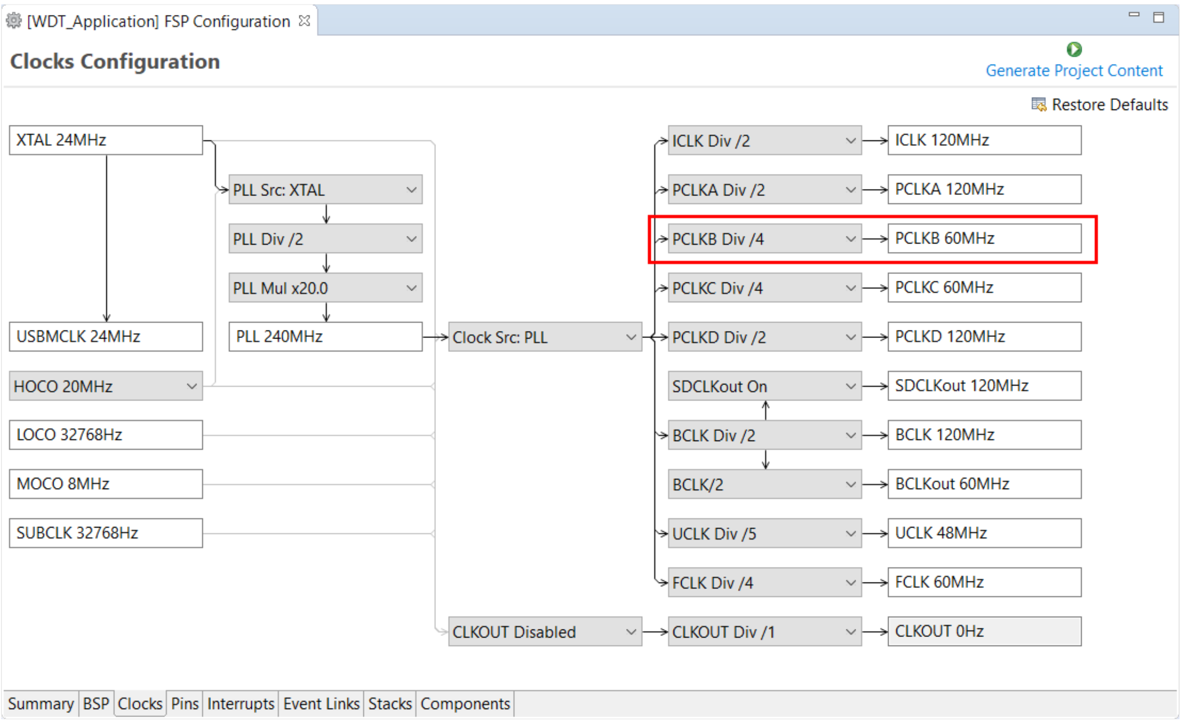


Figure 88: Clock configuration

2.4.4.3 Interrupts Tab

The **Interrupts** tab is used to add new user events or interrupts. No new interrupts or events are needed by the application, so no edits in this tab are required.

2.4.4.4 Event Links Tab

The **Event Links** tab is used to configure events used by the Event Link Controller (ELC). This project doesn't use the ELC, so no edits in this tab are required.

2.4.4.5 Pins Tab

The **Pins** tab provides a graphical tool for configuring the functionality of the pins of the device. For the WDT project no pin configuration is required. Although the project uses two LEDs connected to pins on the device, these pins are pre-configured as output GPIO pins by the BSP.

2.4.4.6 Stacks Tab

You can add any driver to the project using the **Stacks** tab. The HAL driver IO port pins are added automatically by e2 studio when the project is configured. The WDT application uses no RTOS Resources, so you only need to add the HAL WDT driver.

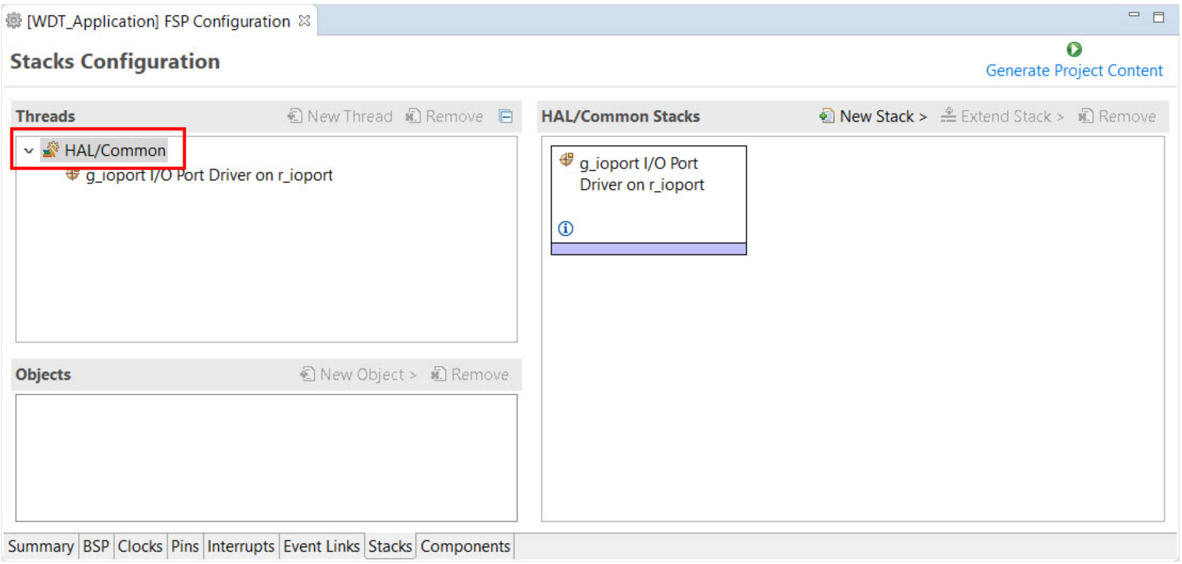


Figure 89: Stacks tab

1. Click on the **HAL/Common Panel** in the Threads Window as indicated in the figure above.

   The Stacks Panel becomes a **HAL/Common Stacks** panel and is populated with the modules preselected by e2 studio.

2. Click on **New Stack** to find a pop-up window with the available HAL level drivers.
3. Select **WATCHDOG Driver on r_wdt**.

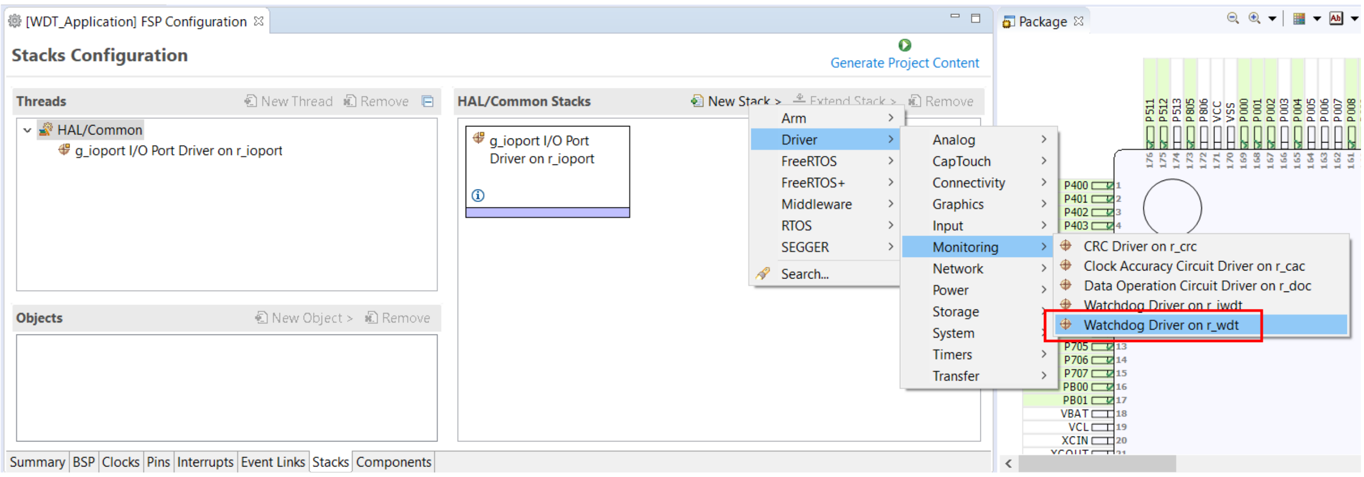


Figure 90: Module Selection

The selected HAL WDT driver is added to the **HAL/Common Stacks** Panel and the **Property** Window shows all configuration options for the selected module. The **Property** tab for the WDT should be visible at the bottom left of the screen. If it is not visible, check that the **FSP Configuration** perspective is selected.

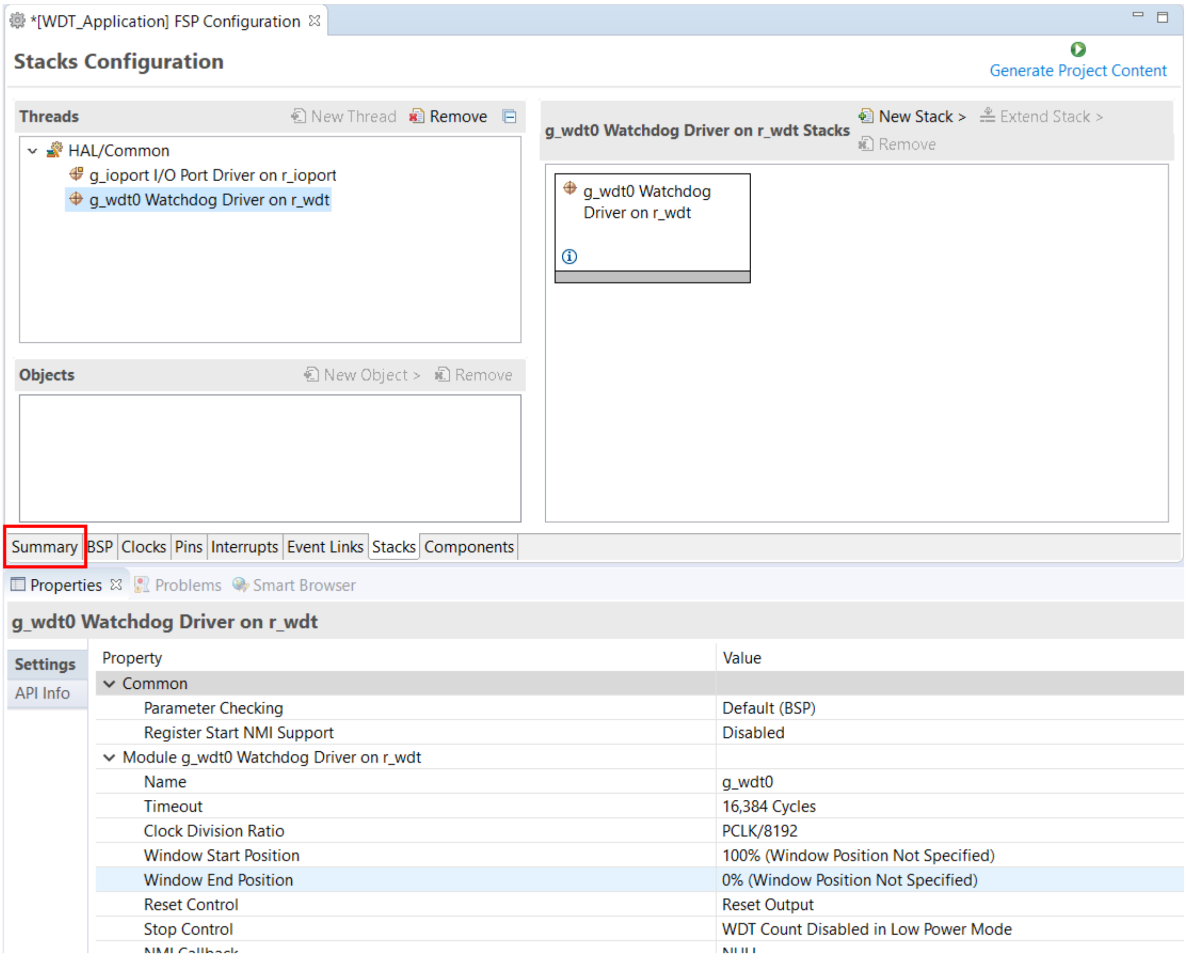


Figure 91: Module Properties

All parameters can be left with their default values.

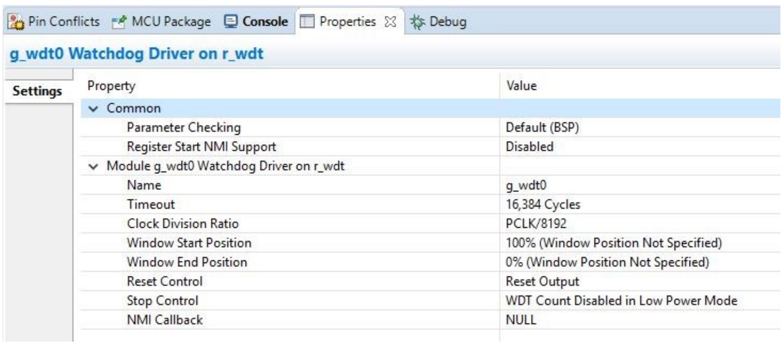


Figure 92: g_wdt WATCHDOG Driver on WDT properties

With PCLKB running at 60 MHz the WDT will reset the device 2.23 seconds after the last refresh.

WDT clock = 60 MHz / 8192 = 7.32 kHz

Cycle time = 1 / 7.324 kHz = 136.53 us

Timeout = 136.53 us x 16384 = 2.23 seconds

Save the **Project Configuration** file and click the **Generate Project Content** button in the top right corner of the **Project Configuration** pane.

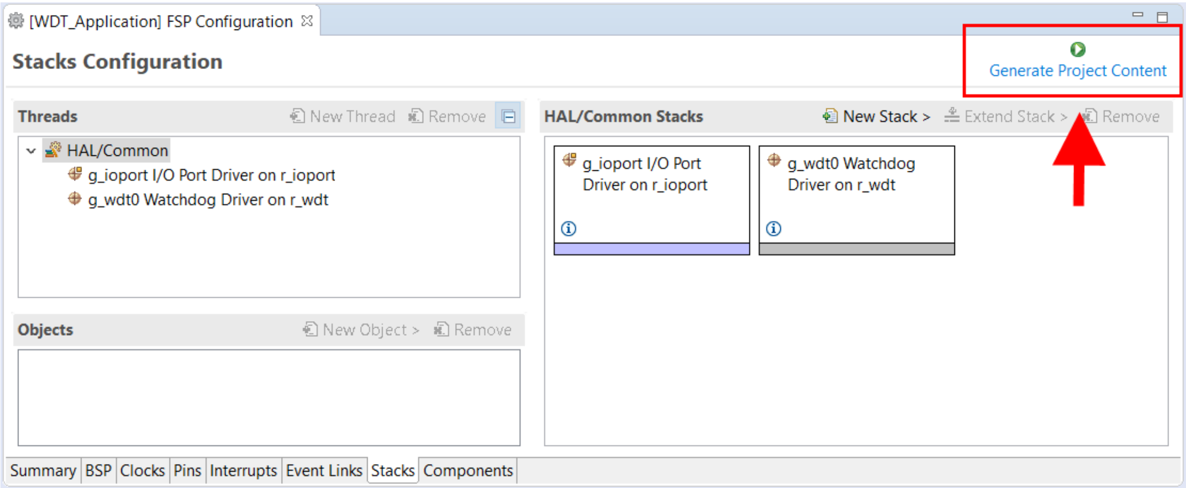


Figure 93: Generate Project Content button

e2 studio generates the project files.

2.4.4.7 Components Tab

The components tab is included for reference to see which modules are included in the project. Modules are selected automatically in the Components view after they are added in the Stacks Tab.

For the WDT project ensure that the following modules are selected:

1. HAL_Drivers -> r_ioport
2. HAL_Drivers -> r_wdt

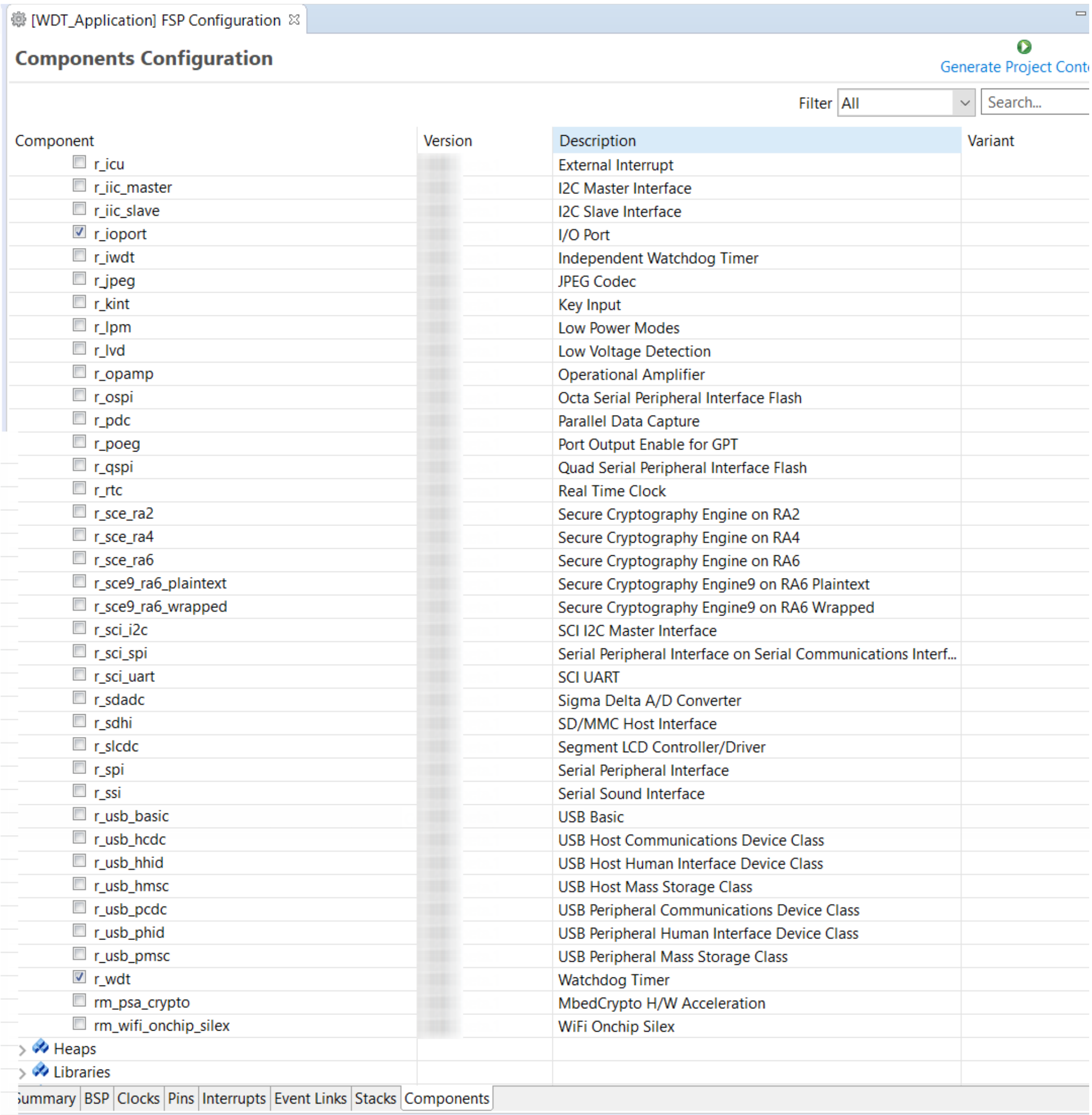


Figure 94: Component Selection

Note

The list of modules displayed in the Components tab depends on the installed FSP version.

2.4.5 WDT Generated Project Files

Clicking the Generate Project Content button performs the following tasks.

- r_wdt folder and WDT driver contents created at:

  ra/fsp/src

- r_wdt_api.h created in:

  ra/fsp/inc/api

- r_wdt.h created in:

  ra/fsp/inc/instances

The above files are the standard files for the WDT HAL module. They contain no specific project contents. They are the driver files for the WDT. Further information on the contents of these files can be found in the documentation for the WDT HAL module.

Configuration information for the WDT HAL module in the WDT project is found in:

ra_cfg/fsp_cfg/r_wdt_cfg.h

The above file's contents are based upon the **Common** settings in the **g_wdt WATCHDOG Driver on WDT Properties** pane.

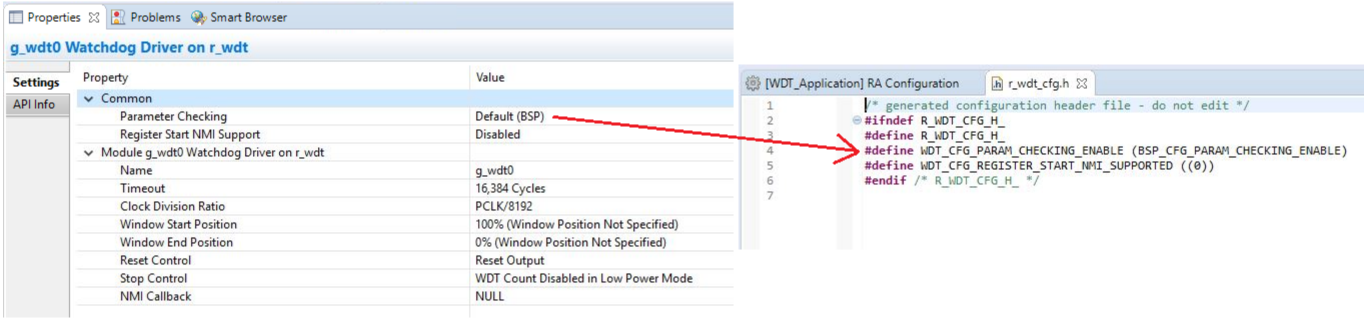


Figure 95: r_wdt_cfg.h contents

Warning

Do not edit any of these files as they are recreated every time the Generate Project Content button is clicked and so any changes will be overwritten.

The r_ioport folder is not created at ra/fsp/src as this module is required by the BSP and so already exists. It is included in the WDT project in order to include the correct header file in ra_gen/hal_data.c–see later in this document for further details. For the same reason the other IOPORT header files– ra/fsp/inc/api/r_ioport_api.handra/fsp/inc/instances/r_ioport.h–are not created as they already exist.

In addition to generating the HAL driver files for the WDT and IOPORT files e2 studio also generates files containing configuration data for the WDT and a file where user code can safely be added. These files are shown below.

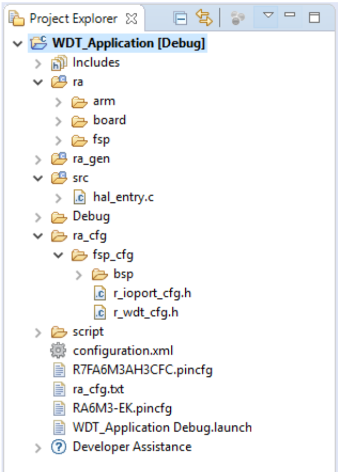


Figure 96: WDT project files

2.4.5.1 WDT hal_data.h

The contents of hal_data.h are shown below.

```
/* generated HAL header file - do not edit */
#ifndef HAL_DATA_H_
 #define HAL_DATA_H_
 #include <stdint.h>
 #include "bsp_api.h"
 #include "common_data.h"
 #include "r_wdt.h"
 #include "r_wdt_api.h"
 #ifdef __cplusplus
extern "C"
{
 #endif
extern const wdt_instance_t g_wdt0;
 #ifndef NULL
void NULL(wdt_callback_args_t * p_args);
 #endif
extern wdt_instance_ctrl_t g_wdt0_ctrl;
extern const wdt_cfg_t g_wdt0_cfg;
void hal_entry(void);
void g_hal_init(void);
```

```
 #ifdef __cplusplus
} /* extern "C" */
 #endif
#endif  /* HAL_DATA_H_ */
```

hal_data.h contains the header files required by the generated project. In addition this file includes external references to the **g_wdt0** instance structure which contains pointers to the configuration, control, api structures used for WDT HAL driver.

Warning
: This file is regenerated each time Generate Project Content is clicked and must not be edited.

2.4.5.2 WDT hal_data.c

The contents of hal_data.c are shown below.

```
/* generated HAL source file - do not edit */
#include "hal_data.h"
wdt_instance_ctrl_t g_wdt0_ctrl;
const wdt_cfg_t g_wdt0_cfg =
{
    .timeout        = WDT_TIMEOUT_16384,
    .clock_division = WDT_CLOCK_DIVISION_8192,
    .window_start    = WDT_WINDOW_START_100,
    .window_end     = WDT_WINDOW_END_0,
    .reset_control = WDT_RESET_CONTROL_RESET,
    .stop_control    = WDT_STOP_CONTROL_ENABLE,
    .p_callback     = NULL,
};
/* Instance structure to use this module. */
const wdt_instance_t g_wdt0 =
{.p_ctrl = &g_wdt0_ctrl, .p_cfg = &g_wdt0_cfg, .p_api = &g_wdt_on_wdt};
void g_hal_init (void)
{
    g_common_init();
}
```

hal_data.c contains g_wdt0_ctrl which is the control structure for this instance of the WDT HAL driver. This structure should not be initialized as this is done by the driver when it is opened.

The contents of g_wdt0_cfg are populated in this file using the **Watchdog Driver on g_wdt0** pane in the Project Configuration **Stacks** tab. If the contents of this structure do not reflect the settings made in the IDE, ensure the **Project Configuration** settings are saved before clicking the **Generate Project Content** button.

Warning

> This file is regenerated each time Generate Project Content is clicked and so should not be edited.

2.4.5.3 WDT main.c

Contains main() called by the BSP start-up code. main() calls hal_entry() which contains user developed code (see next file). Here are the contents of main.c.

```
/* generated main source file - do not edit*/

#include "hal_data.h"

int main (void)

{

    hal_entry();

 return 0;

}
```

Warning

> This file is regenerated each time Generate Project Content is clicked and so should not be edited.

2.4.5.4 WDT hal_entry.c

This file contains the function hal_entry() called from main(). User developed code should be placed in this file and function.

For the WDT project edit the contents of this file to contain the code below. This code implements the flowchart in overview section of this document.

```
#include "hal_data.h"

#include "bsp_pin_cfg.h"

#include "r_ioport.h"

#define RED_LED_NO_OF_FLASHES 30

#define RED_LED_PIN BSP_IO_PORT_01_PIN_00

#define GREEN_LED_PIN BSP_IO_PORT_04_PIN_00

#define RED_LED_DELAY_MS 125
```

```
#define GREEN_LED_DELAY_MS 250
volatile uint32_t delay_counter;
volatile uint16_t loop_counter;
void R_BSP_WarmStart(bsp_warm_start_event_t event);
/*************************************************************************************
*******************************/
void hal_entry (void)
{
 /* Allow the WDT to run when the debugger is connected */
    R_DEBUG->DBGSTOPCR_b.DBGSTOP_WDT = 0;
 /* Open the WDT */
 R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
 /* Start the WDT by refreshing it */
 R_WDT_Refresh(&g_wdt0_ctrl);
 /* Flash the red LED and feed the WDT for a few seconds */
 for (loop_counter = 0; loop_counter < RED_LED_NO_OF_FLASHES; loop_counter++)
    {
 /* Turn red LED on */
 R_IOPORT_PinWrite(&g_ioport_ctrl, RED_LED_PIN, BSP_IO_LEVEL_LOW);
 /* Delay */
 R_BSP_SoftwareDelay(RED_LED_DELAY_MS, BSP_DELAY_UNITS_MILLISECONDS);
 /* Refresh WDT */
 R_WDT_Refresh(&g_wdt0_ctrl);
 R_IOPORT_PinWrite(&g_ioport_ctrl, RED_LED_PIN, BSP_IO_LEVEL_HIGH);
 /* Delay */
 R_BSP_SoftwareDelay(RED_LED_DELAY_MS, BSP_DELAY_UNITS_MILLISECONDS);
 /* Refresh WDT */
 R_WDT_Refresh(&g_wdt0_ctrl);
    }
 /* Flash green LED but STOP feeding the WDT. WDT should reset the
  * device */
 while (1)
    {
 /* Turn green LED on */
```

```
 R_IOPORT_PinWrite(&g_ioport_ctrl, GREEN_LED_PIN, BSP_IO_LEVEL_LOW);
 /* Delay */
 R_BSP_SoftwareDelay(GREEN_LED_DELAY_MS, BSP_DELAY_UNITS_MILLISECONDS);
 /* Turn green off */
 R_IOPORT_PinWrite(&g_ioport_ctrl, GREEN_LED_PIN, BSP_IO_LEVEL_HIGH);
 /* Delay */
 R_BSP_SoftwareDelay(GREEN_LED_DELAY_MS, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
/*************************************************************************************
*******************************/
void R_BSP_WarmStart (bsp_warm_start_event_t event)
{
 if (BSP_WARM_START_RESET == event)
    {
#if BSP_FEATURE_FLASH_LP_VERSION != 0
 /* Enable reading from data flash. */
       R_FACI_LP->DFLCTL = 1U;
 /* Would normally have to wait for tDSTOP(6us) for data flash recovery. Placing the
enable here, before clock and
   * C runtime initialization, should negate the need for a delay since the
initialization will typically take more than 6us. */
#endif
    }
 if (BSP_WARM_START_POST_C == event)
    {
 /* C runtime environment and system clocks are setup. */
 /* Configure pins. */
 R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
    }
}
```

The WDT HAL driver API functions are defined in r_wdt.h. The WDT HAL driver is opened through the open API call using the instance structure defined in r_wdt_api.h:

```
/* Open the WDT */
R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
```

The first passed parameter is the pointer to the control structure g_wdt0_ctrl instantiated in hal_data.c. The second parameter is the pointer to the configuration data g_wdto_cfg instantiated in the same hal_data.c file.

The WDT is started and refreshed through the API call:

```
/* Start the WDT by refreshing it */
R_WDT_Refresh(&g_wdt0_ctrl);
```

Again the first (and only in this case) parameter passed to this API is the pointer to the control structure of this instance of the driver.

2.4.6 Building and Testing the Project

Build the project in e2 studio by clicking **Build > Build Project** or by clicking the build icon. The project should build without errors.

To debug the project

1. Connect the USB cable between the target board debug port and host PC.
2. In the **Project Explorer** pane on the left side of e2 studio, right-click on the WDT project **WDT_Application** and select **Debug As > Debug Configurations**.
3. Under **Renesas GDB Hardware Debugging** select **WDT_Application Debug** as shown below.

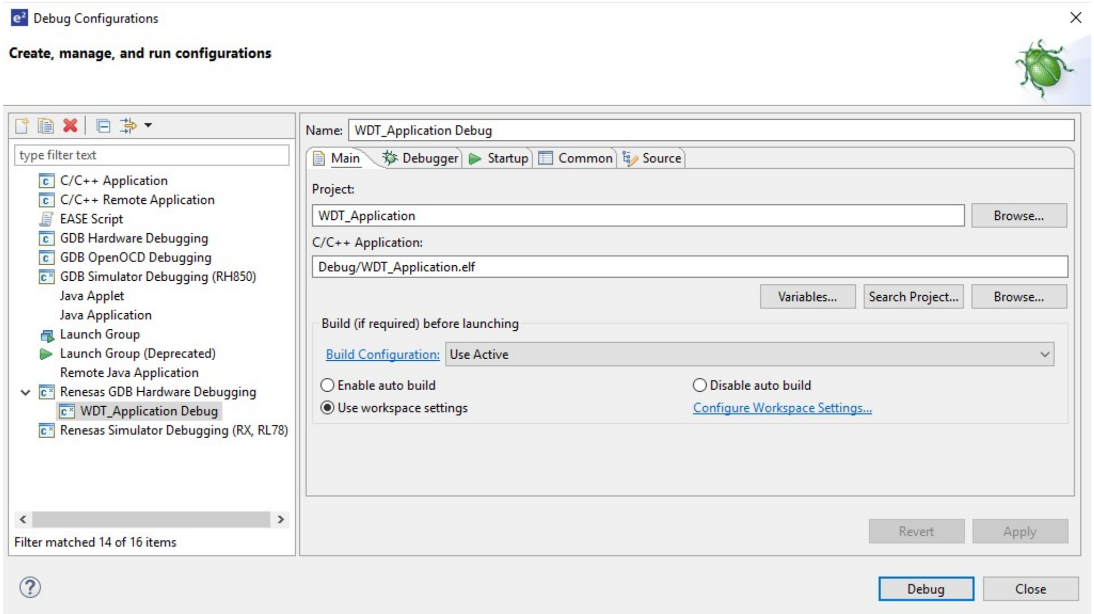
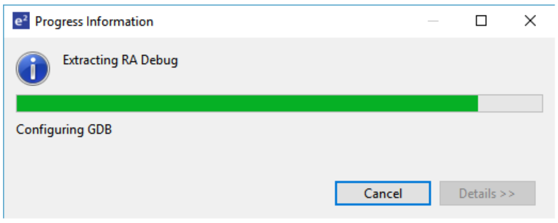


Figure 97: Debug configuration

4. Click the **Debug** button. Click Yes to the debug perspective if asked.



5. The code should run the Reset_Handler() function.
6. Resume execution via **Run > Resume**. Execution will stop in main() at the call to hal_entry().
7. Resume execution again.

The red LED should start flashing. After 30 flashes the green LED will start flashing and the red LED will stop flashing.

While the green LED is flashing the WDT will underflow and reset the device resulting in the red LED to flash again as the sequence repeats.

1. Stop the debugger in e2 studio via **Run > Terminate**.
2. Click the reset button on the target board. The LEDs begin flashing.

2.5 Primer: ARM® TrustZone® Project Development

This section will introduce the user to the tools supporting ARM® TrustZone® configuration for the RA Family of microcontrollers. It is intended to be read by development engineers implementing RA ARM® TrustZone® projects for the first time. It will introduce basic concepts followed by workflow and tooling functions designed to simplify and accelerate their first ARM® TrustZone® development. A background knowledge of e^2 studio and RA device hardware is expected.

Target Device

RA Cortex®-M33 or Cortex®-M23 devices with ARM® TrustZone® security extension.

2.5.1 Renesas Implementation of ARM® TrustZone® Technology

For brevity, ARM® TrustZone® will be abbreviated to TZ in this document.

The following section is supplied for reference only. For full details of TZ implementation, refer to Arm documentation (https://developer.arm.com/ip-products/security-ip/trustzone) and the RA6M4 device manual.

Arm TZ technology divides the MCU and therefore the application into Secure and Non-Secure partitions. Secure applications can access both Secure and Non-Secure memory and resources. Non-Secure code can access Non-Secure memory and resources as well as Secure resources through a set of so-called veneers located in the Non-Secure Callable (NSC) region. This ensures a single access point for Secure code when called from the Non-Secure partition. The MCU starts up in the Secure partition by default. The security state of the CPU can be either Secure or Non-Secure.

The MCU code flash, data flash, and SRAM are divided into Secure (S) and Non-Secure (NS) regions. Code flash and SRAM include a further region known as Non-Secure Callable (NSC). These memory security attributes are set into the non-volatile memory via SCI or USB boot mode commands when the device lifecycle is Secure Software Debug (SSD) state. The memory security attributes are loaded into the Implementation Defined Attribution Unit (IDAU) peripheral and the memory controller before application execution and cannot be updated by application code.

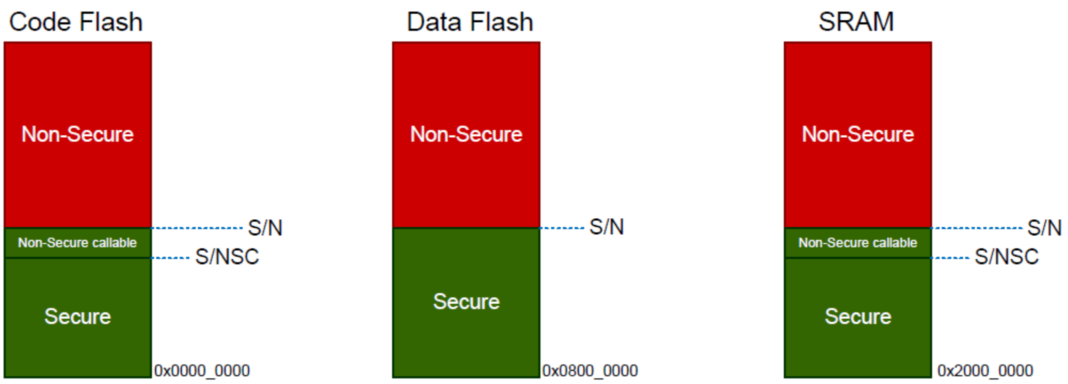


Figure 98: Secure and Non-Secure Regions

Note: All external memory accesses are considered to be Non-Secure.

Code Flash and SRAM can be divided into Secure, Non-Secure, and Non-Secure Callable. All secure memory accesses from the Non-Secure region MUST go through the Non-Secure Callable gateway and target a specific Secure Gateway (SG) assembler instruction. This forces access to Secure APIs at a fixed location and prevents calls to sub-functions and so on. Failing to target an SG instruction will generate a TZ exception.

TZ enabled compilers will manage generation of the NSC veneer automatically using CMSE extensions.

2.5.1.1 Calling from Non-Secure to Secure

A new instruction SG (Secure Gateway) has been added to the Armv8-M architecture. This MUST be the destination instruction for any branch within the Non-Secure Callable region. If an attempt is made to branch to any other instruction from the Non-Secure partition, a TZ exception will be thrown.

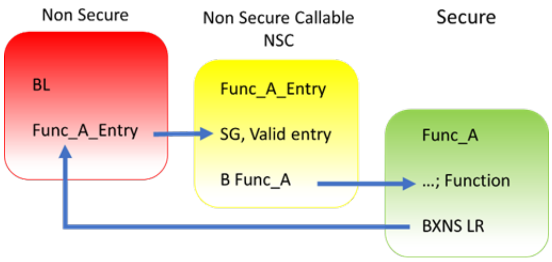


Figure 99: Calling from Non-Secure to Secure Functions

2.5.1.2 Calling from Secure to Non-Secure

Secure code uses B(L)XNS instructions to make direct calls to Non-Secure functions. While this is certainly possible, it can create a security vulnerability in the application. It is also challenging for the Secure application to determine the address of the non-secure function during build phase. From the RA Tools and FSP point view, calling directly from Secure to Non-Secure via FSP API is not supported.

Preference is for the Secure code to initialise as necessary from reset, then pass control to the Non-Secure partition. It will manage any data transfers and so forth via FSP call-backs as security checks. For example, secure data can be copied to Non-Secure RAM.

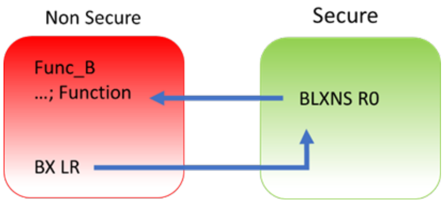


Figure 100: Calling from Secure to Non-Secure Functions

2.5.2 Workflow

ARM® TrustZone® MCU development normally consists of two projects within a workspace, Secure and Non-Secure. General project workflows are described in the following sections. The Renesas project generator also supports development with "Flat project" model with no ARM® TrustZone® awareness.

2.5.2.1 Secure Project

1. Start a new Secure project in e^2 studio.
2. Select and configure pins and drivers/stacks that need to be initialized and used in Secure

mode. This should be kept to a minimum to reduce the security attack surface.
3. Expose top of stacks as Non-Secure Callable (NSC) ***if*** they need to be accessed from Non-Secure partition. Again, this should be kept to a minimum.
4. Generate project content and write Secure code such as key handling and opening drives as needed.
5. Modify/remove any unnecessary "Guard" functions as needed to control access via NSC.
6. Build project.
7. A Non-Secure project will be needed before debugging. If necessary, prepare a "dummy" Non-Secure project or replace R_BSP_NonSecureEnter(); with while(1); in hal_entry.c.

2.5.2.2 Non-Secure Project

1. Start a new Non-Secure project.
2. If you have access to the Secure project, choose this option. However, if you only have access to a device with pre-programmed Secure code (commonly referred to as provisioned device) choose "Secure Bundle".
3. Select and configure pins and drivers/stacks that need to be initialized and used in Non-Secure mode.
4. Note that you can add NSC drivers and stacks as needed.
5. Generate project content and write Non-Secure code as needed
6. Access NSC drivers and Stacks via Guard functions.
7. Build and debug project.

2.5.2.3 Flat Project

A flat project does not technically use ARM® TrustZone® as the developer has made a decision to place the entire application in Secure partition from restart.

Notes:

- Any code placed in external memory (such as OSPI or QSPI) will be Non-Secure.
- The Ethernet EDMAC is designed to be a Non-Secure bus master so associated Ethernet RAM buffers will be placed in Non-Secure RAM. The tooling will automatically manage this.

The workflow is as follows:

1. Start a new Flat project.
2. Select and configure pins and drivers/stacks as needed.
3. Generate project content and write code as needed.
4. Build and debug project.

2.5.3 RA Project Generator (PG)

The RA project generators have been created to help users through setting up new TZ enabled projects. User will be prompted for project settings such as Project Type (Secure, Non-Secure, or Flat), compiler, RTOS and debugger. Care is needed when setting up a TZ project to ensure that the connection between Secure and Non-Secure partitions are managed correctly.

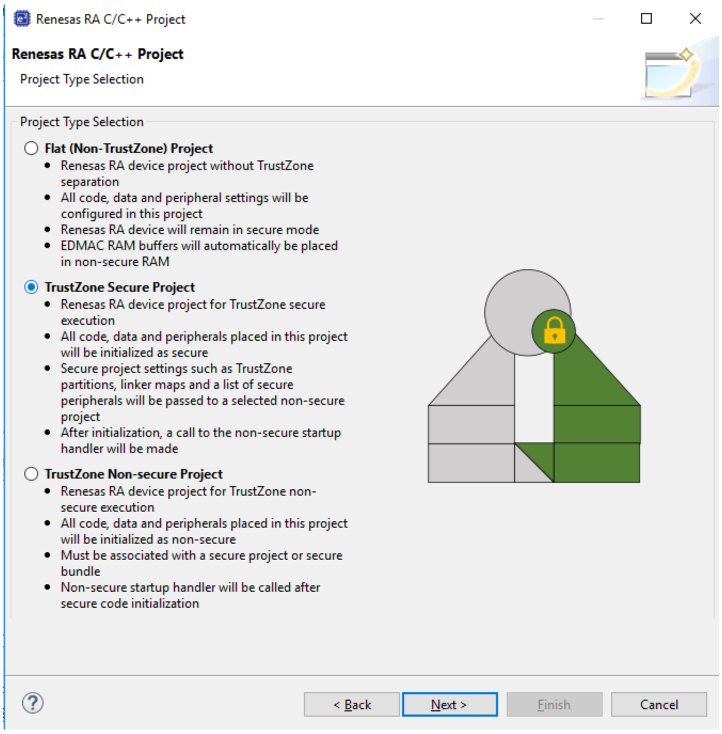


Figure 101: Secure Project (following Arm notation as green)

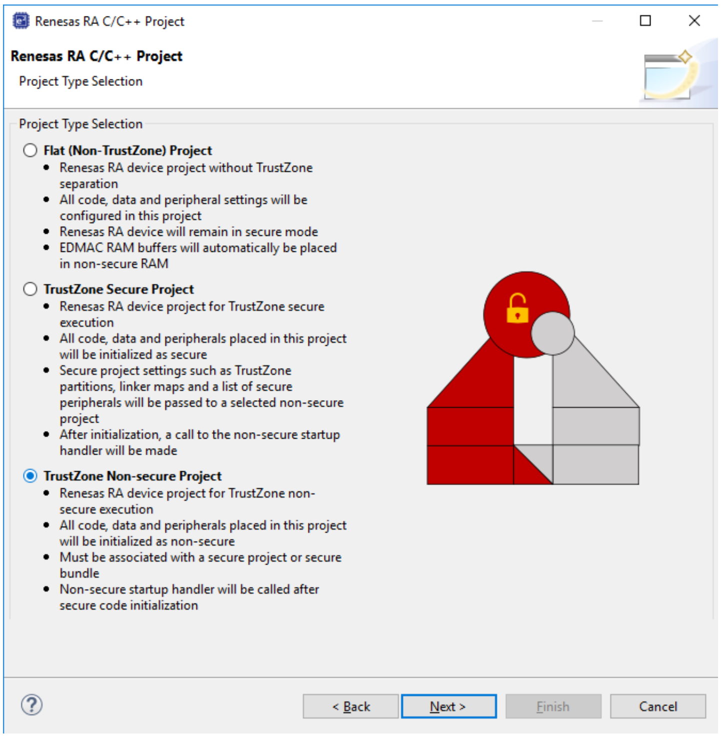


Figure 102: Non-Secure Project (following Arm notation as red)

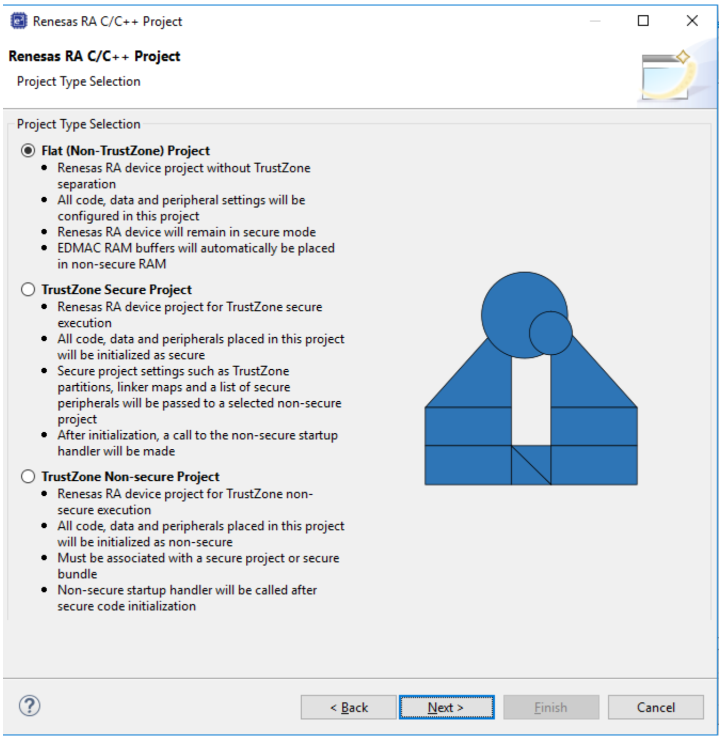


Figure 103: Flat Project

2.5.3.1 Secure Project Set Up

All code, data, and peripherals in this project will be configured as Secure using the device Peripheral Security Attribution (PSA) registers. Although it is very application specific, we recommend keeping the Secure project code as small as possible to reduce the attack surface. For example, secure key handling may be the only application code in the secure project.

Necessary values to set up the TZ memory partition (IDAU registers) will be automatically calculated after the project is built to ensure they match the code and data size, keeping the attack surface as small as possible.

Typically, ANSI C start up code (clearing of RAM, variable initialisation, etc) , clock, and secure peripheral initialisation will occur in this project.

At the end of the Secure code, a call will be made to R_BSP_NonSecureEnter(); to pass control to the Non-Secure partition.

Non-Secure Callable (NSC) "Guard" functions are added to the project and expose selected modules to Non-Secure projects. User can add application-specific access checks as needed in these functions.

Output of this project type will be an elf file that must be either pre-programmed (provisioned) into a device or referenced by a Non-Secure project (via Secure bundle *.SBD) to build a final image.

This project type will NOT typically be debugged in isolation and will normally require a Non-Secure project such as a call to a R_BSP_NonSecureEnter() to be made. This can be replaced with while(1); if needed.

2.5.3.2 RTOS Support in TZ Project

Although the RTOS kernel and user tasks will reside in the Non-Secure partition, the Secure partition needs to allocate stack space and so on. It is essential when starting a new RTOS project that the TrustZone Secure RTOS-Minimal template is selected. This will add the Arm TrustZone Context RA Port as below.

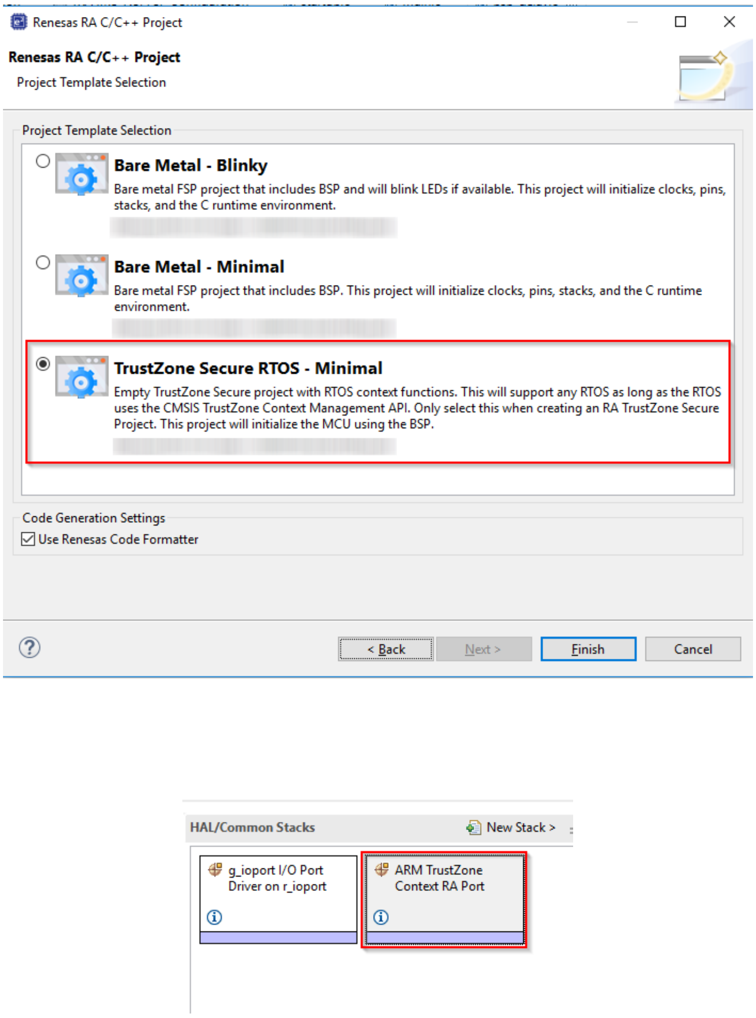


Figure 104: Secure RTOS-Minimal Template

2.5.3.3 Peripheral Security Attribution

Each peripheral can be configured to be Secure or Non-Secure. Peripherals are divided into two types.

Type-1 peripherals have one security attribute. Access to all registers is controlled by one security attribute. The Type-1 peripheral security attribute is set in the PSARx (x = B to E) register by the secure application.

Type-2 peripherals have the security attribute for each register or for each bit. Access to each register or bit field is controlled according to these security attributes. The Type-2 peripheral security attribute is set in the Security Attribution register in each module by the Secure application. For more information about the Security Attribution register, see sections in the Appropriate MCU's User's Manual for each peripheral.

Table 1. Secure and Non-Secure Peripherals

Type	Peripheral
Type 1	SCI, SPI, USBFS, CAN, IIC, SCE9, DOC, SDHI, SSIE, CTSU, CRC, CAC, TSN, ADC12, DAC12, POEG, AGT, GPT, RTC, IWDT, WDT
Type 2	System control (Resets, LVD, Clock Generation Circuit, Low Power Modes, Battery Backup Function), FLASH CACHE, SRAM controller, CPU CACHE, DMAC, DTC, ICU, MPU, BUS, Security setting, ELC, I/O ports
Always Non-Secure	CS Area Controller, QSPI, OSPI, ETHERC, EDMAC

FSP will initialise the arbitration registers during Secure project BSP start up. User code may also be written to set or clear further arbitration. However care must be taken not to undermine FSP.

2.5.3.4 Non-Secure

All code, data, and peripherals in this project will be configured as Non-Secure. This project type must be associated with a Secure project to enable access to secure code, peripherals, linker scripts and others.

2.5.3.5 Flat Project Type

All code, data, and peripherals are configured in a Secure single partition except for the EDMAC RAM buffers that will remain in the Non-Secure partition. Effectively, TZ is disabled.

2.5.3.6 Secure Connection to Non-Secure Project

When starting a new Non-Secure Project, the user will be prompted for either a Secure Project or Secure Bundle. In each case, details of the linker settings, Non-Secure Callable functions, and Secure peripherals will be read to enable the Non-Secure project setup.

Should the Secure project or bundle be rebuilt, the Non-Secure editor will detect this and prompt user to regenerate the Non-Secure project configuration.

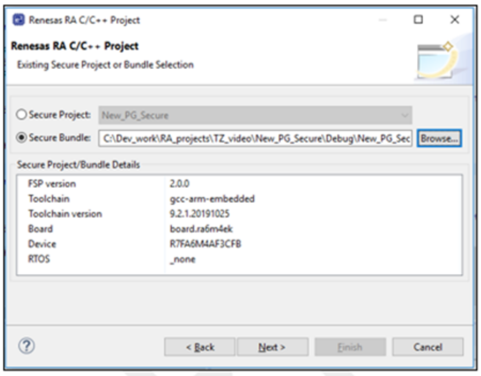


Figure 105: Secure Project or Bundle Selection

Secure Project (Combined)

A Secure project must reside in the same Workspace as the Non-Secure project and will typically be used when a design engineer has access to both the Secure and Non-Secure project sources. This is sometimes known as "Combined model".

A Secure .elf file will be referenced and included in the debug configuration for download to the target device. The development engineer will have visibility of Secure and Non-Secure project source code and configuration.

Secure Bundle (Split)

A Secure Bundle will ONLY include linker memory ranges, symbol references, and details of locked Secure peripheral configuration settings but no access to Secure source code (API header files will be included as necessary).

The Secure bundle file (*.SBD) must be supplied to the Non-Secure developer by the Secure project developer.

The development engineer will typically not have access to the Secure project or .elf file which MUST be pre-programmed or provisioned into the target MCU.

The DLM state of target device should then be switched to NSECSD (see section 6.2) before the device is provided to the non-secure developer.

This is often referred to as "Split model" where a basic security set up is developed by a Secure team and then passed to the Non-Secure team in the same facility or at a third party. The Non-Secure team has no access to the Secure source code and cannot directly access Secure peripherals, data, or APIs.

2.5.3.7 Debug Configurations

After each project type has been selected, a suitable debug configuration will be generated.

Non-Secure with Secure Project (Combined)

Both Secure and Non-Secure .elf files will be downloaded.

A debug configuration called <project name>_SSD will be generated.

Non-Secure with Secure Bundle (Split)

Only a Non-Secure elf will be downloaded. This configuration must be used with a pre-provisioned device (Secure project pre-programmed into MCU Flash).

A debug configuration called <project name>_NSECSD will be generated.

Flat Debug

A single .elf file will be downloaded.

A debug configuration called <project name>_FLAT will be generated.

2.5.4 Secure Projects

As mentioned, Secure code will be called immediately after device reset and run ANSI C start up, clock, interrupt vector table, and secure peripheral initialization before starting user code. All selected peripheral configuration settings will be automatically initialised as Secure.

2.5.4.1 Secure Clock

Device clock settings are the possible exception in that they will be initialised in the Secure project (to enable faster start up from reset) but can be set as Secure or Non-Secure as user application may need to change settings during execution (for low-power mode and so on). The Secure and Non-Secure FSP BSPs can both change the clock settings.

However, clock settings can be locked as Secure should the developer choose to do so.

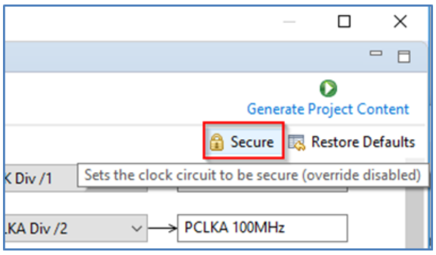


Figure 106: Secure Clock Setting

2.5.4.2 Setting Drivers as NSC

Some driver and middleware stacks in the Secure project may need to be accessed by the Non-Secure partition. To enable generation of NSC veneers, set "Non-Secure Callable" from the right-click context menu for the selected modules in the Configurator.

Note: It is only possible to "expose" top of stacks as NSC.

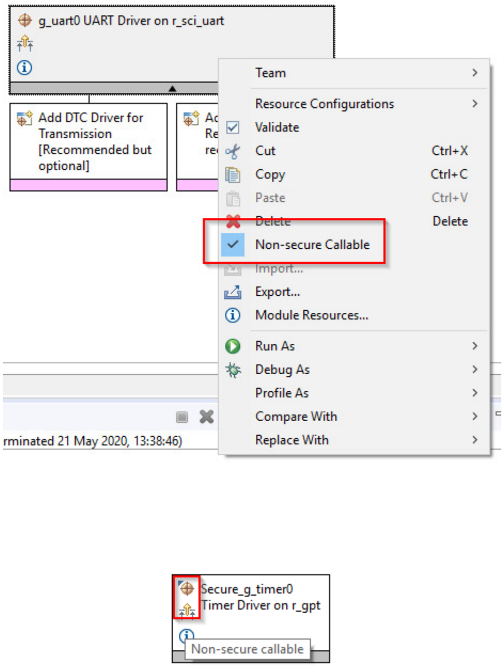


Figure 107: Generate NSC Veneers

The top of the stack will be marked with a new icon and tool tip to signify NSC access.

2.5.4.3 Guard Functions

Access to NSC drivers from a Non-Secure project is possible through the Guard APIs. FSP will automatically generate Guard functions for all the top of stack/driver APIs added to the project as Non-Secure Callable.

User can choose to add further levels of access control or delete guard function if they wish to only expose a limited range of APIs to a Non-Secure developer.

```
BSP_CMSE_NONSECURE_ENTRY fsp_err_t g_uart0_open_guard(
       uart_ctrl_t *const p_api_ctrl, uart_cfg_t const *const p_cfg) {
    /* TODO: add your own security checks here */
    FSP_PARAMETER_NOT_USED(p_api_ctrl);
    FSP_PARAMETER_NOT_USED(p_cfg);
    return R_SCI_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
}
```

For example, an SCI channel may be opened and configured for a desired baud rate by the Secure developer, but only enable the Write API to the Non-Secure developer. In which case, all but g_uart0_write_guard() could be deleted. CTRL structures are not required as they will be added on the Secure side.

For example, the call from the Non-Secure partition would be as follows:

```
err = g_uart0_open_guard(0,0);
```

2.5.5 Non-Secure projects

Configuration of the project can continue as for other RA devices, but certain resources will be locked if they have been previously set up as Secure.

The Non-Secure project will be called from the Secure project via R_BSP_NonSecureEnter();

2.5.5.1 Clock Set Up

You may recall that clocks can be set as Secure or Non-Secure. If they are set as Secure, settings will only be available to view, and user will not be able to change them. The Override button will be greyed. This is useful to preserve CGC sync with secure project by not overriding unless necessary. If it is NOT set as Secure, user can choose to override the initial Secure settings

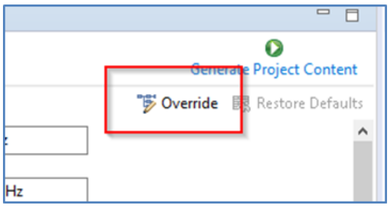


Figure 108: Clock Setting as Non-Secure

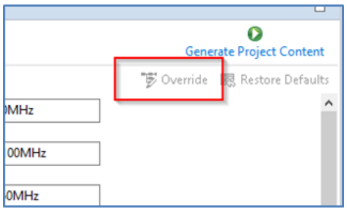
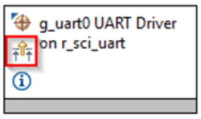


Figure 109: Clock Setting as Secure

2.5.5.2 Selecting NSC Drivers

Drivers declared as NSC in a Secure project can be selected and added to Non-Secure project and will be decorated as before.



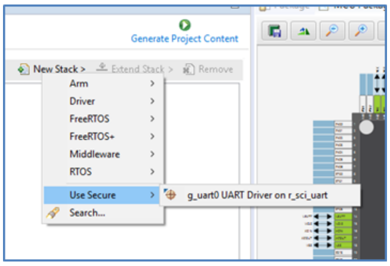


Figure 110: Selecting NSC Drivers

2.5.5.3 Locked Resources

When a NSC Secure driver is added to a Non-Secure project, the configuration settings are locked and are available for information only. A padlock is added for indication.

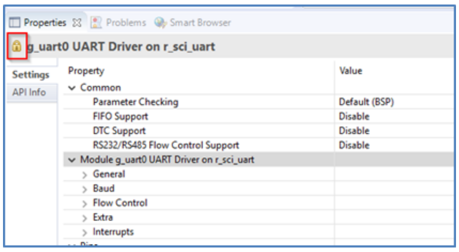


Figure 111: Locked Resources

2.5.5.4 Locked Channels

In a peripheral with multiple channels, for example, DMA, if a Non-Secure developer tries to select a channel that has already been defined as Secure, the following error message type will be displayed.

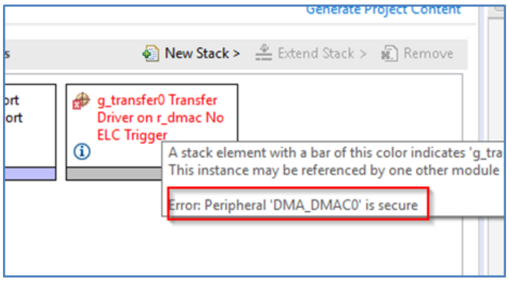


Figure 112: Error Message when Selecting a Secure Channel

2.5.6 IDAU registers

Renesas RA TZ-enabled devices include a set of registers known as Implementation Defined Attribution Unit (IDAU) that are used to set up partitions between Secure, Non-Secure Callable, and Non-Secure regions. The IDAU registers can only be programmed during MCU ***boot mode*** and NOT through the debug interfaces. Because of this, special debugger firmware has been developed to manage bringing the device up in SCI boot mode to set up the IDAU registers (automatically drives MD pin) and then switch back to debug mode as needed.

Note: Please be aware of the extra signal connection (MD pin) needed on the debug interface connector. The Renesas Evaluation Kit (EK) for your selected device is a good reference.

Pin No.	SWD	JTAG	Serial Programming using SCI
1	VCC	VCC	VCC
2	P108/SWDIO	P108/TMS	NC
4	P300/SWCLK Wired OR with MD	P300/TCK Wired OR with MD	P201/MD
6	P109/SWO/TXD9	P109/TDO/TXD9	P109/TXD9
8	P110/RXD9	P110/TDI/RXD9	P110/RXD9
9	GNDdetect	GNDdetect	GNDdetect
10	nRESET	nRESET	nRESET
12	P214/TRACECLK	P214/TRACECLK	NC
14	P211/TRACEDATA[0]	P211/TRACEDATA[0]	NC
16	P210/TRACEDATA[1]	P210/TRACEDATA[1]	NC
18	P209/TRACEDATA[2]	P209/TRACEDATA[2]	NC
20	P208/TRACEDATA[3]	P208/TRACEDATA[3]	NC
3, 5, 15,17, 19	GND	GND	GND
7	NC	NC	NC
11, 13	NC	NC	NC

The e2 studio build phase automatically extracts the IDAU partition register settings from the Secure .elf file and programs them into the device during debug connection, which can be observed in the console.

This is an important phase of TZ development as the Secure partitions should be set as small as possible to ensure that the security attack surface is as small as possible.

However, should the developer wish to make these partitions larger to accommodate, for example during field firmware updates, const or data arrays should be placed in the Secure project as needed.

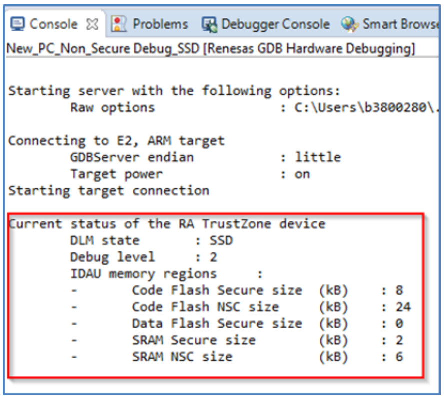
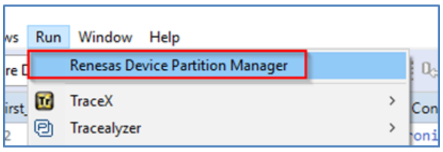


Figure 113: RA TrustZone Device Current Status

It is also possible to manually set up the partition registers through the Renesas Device Partition Manager.



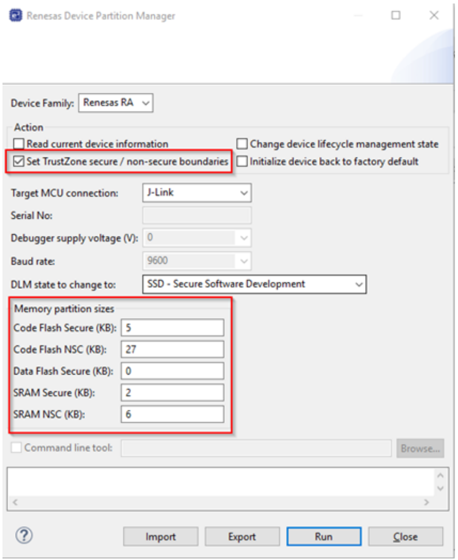


Figure 114: Renesas Device Partition Manager

2.5.6.1 SCI Boot Mode

Example of MD mode pin connection to debugger connector (from EK schematic).

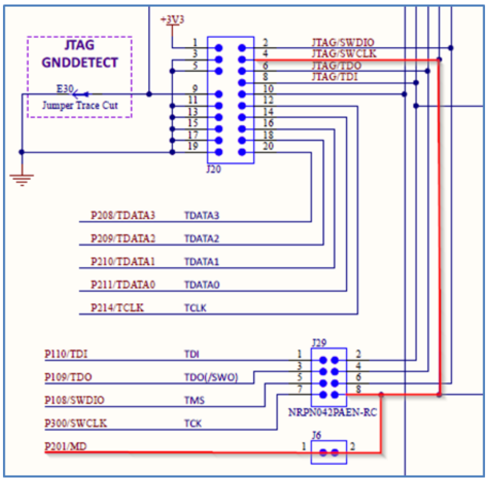


Figure 115: Example of MD Mode Pin Connection to Debugger Connector (from EK schematic

2.5.6.2 DLM States

Device lifecycle defines the current phase of the device and controls the capabilities of the debug interface, the serial programming interface and Renesas test mode. The following illustration shows the lifecycle definitions and capability in each lifecycle.

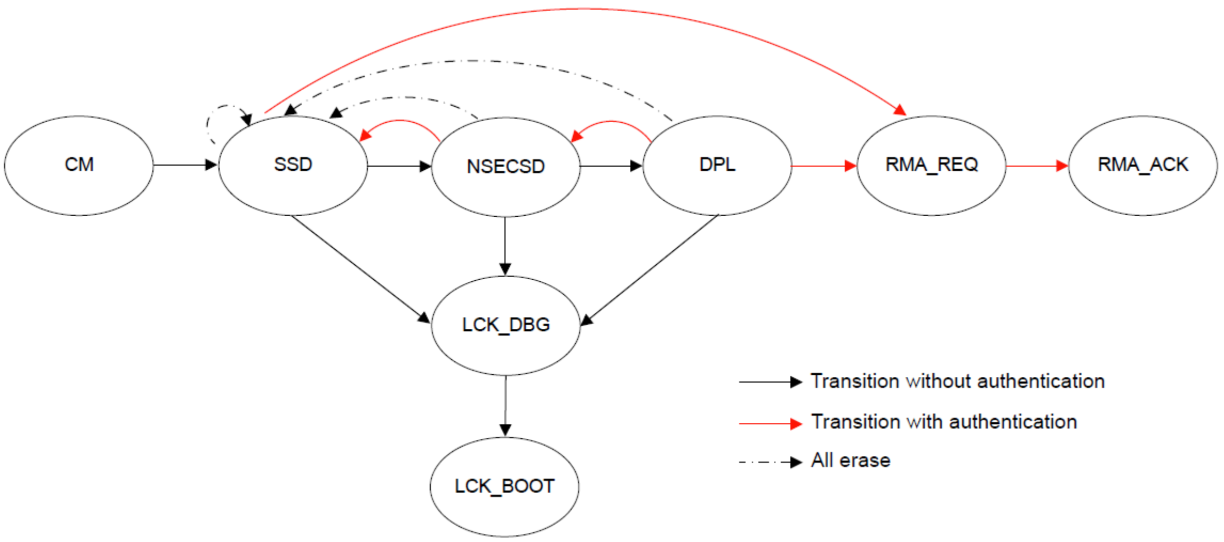


Figure 116: Lifecycle Stages

Note: All authentication key exchange and transitioning to LCK_DBG, LCK_BOOT, RMA_REQ is only managed by Renesas Flash Programmer (RFP) and NOT within e[2] studio.

Lifecycle	Definition	Debug level	Serial programming	Test mode
CM	"Chip Manufacturing" The state when the customer received the device.	DBG2	Available, cannot access code/data flash	Not available
SSD	"Secure Software Development" The secure part of application is being developed.	DBG2	Available can program/erase/read all code/data flash area	Not available
NSECSD	"Non-SECure Software Development" The non-secure part of application is being developed.	DBG1	Available can program/erase/read all code/data flash area	Not available
DPL	"DePLoyed" The device is in-field.	DBG0	Available cannot access code/data flash area	Not available
LCK_DBG	"LoCKed DeBuG" The debug interface is permanently disabled.	DBG0	Available cannot access code/data flash area	Not available
LCK_BOOT	"LoCKed BOOT interface" The debug interface and the serial programming interface are permanently disabled.	DBG0	Not available	Not available
RMA_REQ	"Return Material Authorization REQuest" Request for RMA. The customer must send the device to Renesas in this state.	DBG0	Available cannot access code/data flash area	Not available
RMA_ACK	"Return Material Authorization ACKnowledged" Failure analysis in Renesas	DBG2	Available cannot access code/data flash area	Available

Figure 117: Lifecycle Stages and Debug Levels

There are three debug access levels. The debug access level changes according to the lifecycle state.

- DBG2: The debugger connection is allowed, and no restriction to access memories and peripherals

- DBG1: The debugger connection is allowed, and restricted to access only Non-Secure memory regions and peripherals
- DBG0: The debugger connection is not allowed

Transitions for one state to another can be performed using the Renesas Flash Programmer (RFP, see section below) or using the Renesas Device Partition Manager (limited number of states possible). It is possible to secure transitions between states using authentication keys. For more information on DLM states and transitions (device specific), please refer to device user manual.

2.5.7 Debug

By default, the device will be in SSD mode and so allow access to Secure and Non-Secure partitions. In this mode both Secure and Non-Secure .elf files will be downloaded.

The current debugger status is displayed in the lower left corner and includes the DLM state (SSD or NSECSD) and current partition (Secure, Non-Secure, or Non-Secure Callable) when the debugger is stopped, for example.

Figure 118: Current Debugger Status

2.5.7.1 Non-Secure Debug

Once the device is transitioned to NSECSD mode, only Non-Secure Flash, RAM and Peripherals can be accessed. In this mode, a Secure .elf must be pre-programmed (provisioned) into the device, and only a Non-Secure .elf file will be downloaded.

When in NSECSD mode access to Secure elements will be blocked and data displayed as ????????.

In NSECSD mode, it is not possible to set breakpoints on Secure code or data.

It is not possible to step into Secure code; the debugger will perform a step-over of any Secure function calls. Should the user press the Suspend button during execution, the debugger will stop at the next Non-Secure code access.

Assuming Secure memory region finishes at 32K (0x8000) in NSECSD debug mode (colour coding added for indication only), memory will be displayed as shown in the following figure.

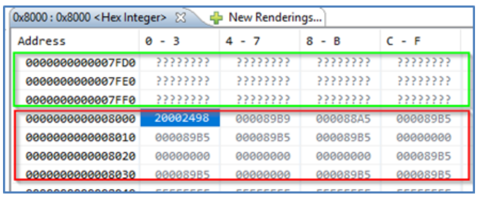


Figure 119: Memory Display in NSECSD Debug Mode

Disassembly will be displayed as shown in the following figure.

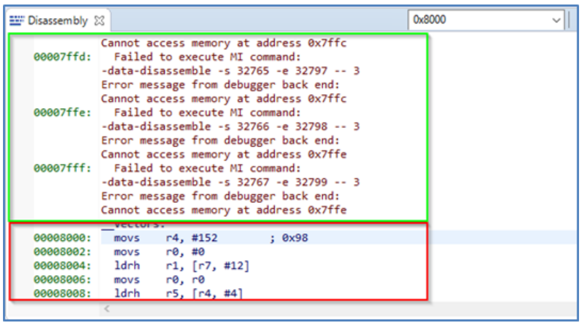


Figure 120: Disassembly Display in NSECSD Debug Mode

2.5.8 Debugger support

Renesas E2, E2 Lite, and SEGGER J-Link are supported in e^2 studio for TZ projects.

Debugger Support for TZ Projects

Feature	E2 Lite	E2	J-Link	J-Link OB	ULINK	IAR i-Jet
JTAG	Yes	Yes	Yes	No	Yes	Yes
SWD	Yes	Yes	Yes	Yes	Yes	Yes
ETB trace	Yes	Yes	Yes	Yes	Yes	Yes
ETM trace	No	Yes	Yes	No	Yes	Yes
TZ partition programming	Yes	Yes	Yes	Yes	No	No
Non secure debug	Yes	Yes	Yes	Yes	Yes	Yes
e^2 studio	Yes	Yes	Yes	Yes	No	TBC
IAR EW Arm	Under consideration	Under consideration	Yes	Yes	No	Yes
Keil MDK	Under consideration	Under consideration	Yes	Yes	Yes	No

2.5.9 Third-Party IDEs

Third-party IDEs such as IAR Systems EWARM and Keil MDK (uVision) are supported by the RA Smart Configurator (RA SC).

In general, RA SC offers the same configurator functionality as e^2 studio documented above. Project generators are available to initialise workspaces in the target IDEs as well as setting up debug configurations and so forth. However, there are some limitations that need to be noted especially with regards to IDAU TZ partition register programming. See the specific RA SC documentation for usage details.

2.5.10 Renesas Flash Programmer (RFP)

Updated versions of Renesas Flash Programmer (RFP) are available to support setting of partitions, DLM state and Authentication keys.

RFP can be downloaded free of charge on the Renesas web site.

A new mode has been added to Program Flash Options as shown in the following graphics.

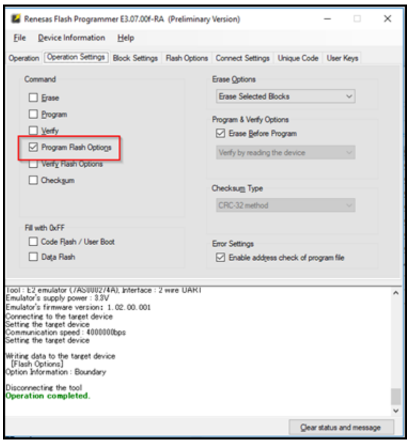


Figure 121: RFP Program Flash Options

Options to set partition boundaries are shown in the following figure.

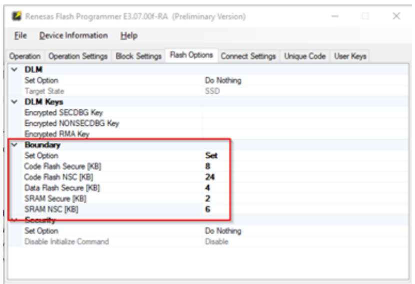


Figure 122: RFP Partition Boundaries

Options to set DLM state, Authentication keys, and Security settings are shown in the following figure.

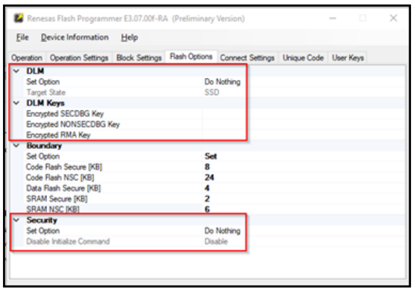


Figure 123: RFP DLM State, Authentication Keys, and Security Settings

Great care is needed here as some DLM states can **permanently** turn off debug and boot mode on the devices. Equally programming a security access authentication key can lead to permanently locked devices if the key is lost.

2.5.11 Glossary

IDAU

Implementation Defined Attribute Unit. Used to program TZ partitions in SCI book mode.

NSECSD

Non-Secure Software Development mode

SSD

Secure Software Development mode

NSC

Non-Secure Callable. Special Secure memory region used for Veneer to allow access to Secure APIs from Non-Secure code.

Provisioned

Device with Secure code pre-programmed and DLM state set to **NSECSD**

Flat project

All code, data and peripherals are configured as secure with the exception of the EDMAC RAM buffer which are placed in Non-Secure RAM due to the configuration of the internal bus masters.

Veneer

Code that resides in Non-Secure Callable region

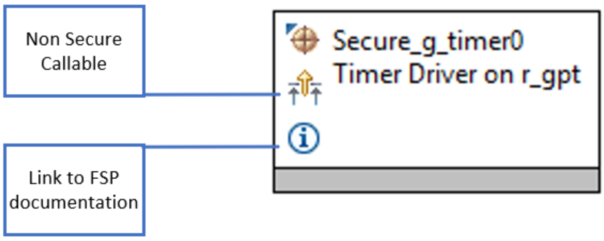
Combined model

Development engineer has access to both Secure and Non-Secure project and source code

Split model

Development Engineer has access to only the Non-Secure partition. No visibility of Secure source code. Secure code will be provisioned into device.

2.5.11.1 Configurator Icon Glossary



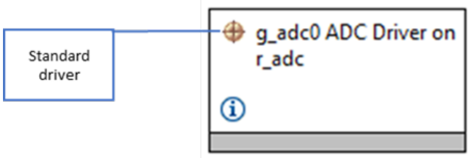


Figure 124: Configurator Icons

2.6 RA SC User Guide for MDK and IAR

2.6.1 What is RA SC?

The Renesas RA Smart Configurator (RA SC) is a desktop application designed to configure device hardware such as clock set up and pin assignment as well as initialization of FSP software components for a Renesas RA microcontroller project when using a 3rd-party IDE and toolchain.

The RA Smart Configurator can currently be used with

1. Keil MDK and the Arm compiler toolchain.
2. IAR EWARM with IAR toolchain for Arm

Projects can be configured and the project content generated in the same way as in e2 studio. Please refer to Configuring a Project section for more details.

2.6.2 Using RA Smart Configurator with Keil MDK

2.6.2.1 Prerequisites

- Keil MDK and Arm compiler are installed and licensed. Please refer to the RASC Release notes for the version to be installed.
- Import the RA device pack. Download the RA device pack archive file (ex: MDK_Device_Packs_2.x.x.zip) from the FSP GitHub release page. Extract the archive file to locate the RA device pack. To import the RA device pack, launch the PackInstaller.exe from <keil_mdk_install_dir>\UV4. Select the menu item **File > Import...** and browse to the extracted .pack file.
- Verify that the latest updates for RA devices are included in Keil MDK. To verify, select the menu "Packs" in Pack Installer and verify that the menu item **Check for Updates on Launch** is selected. If not, select **Check for Updates on Launch** and relaunch Pack

Installer.
- For flashing and debugging, the latest Segger J-Link DLL is installed into Keil MDK.
- Install RA SC and FSP using the Platform Installer from the GitHub release page.

2.6.2.2 Create new RA project

The following steps are required to create an RA project using Keil MDK, RA SC and FSP:

1. Start the RA Smart Configurator.

2. Enter a project folder and project name.

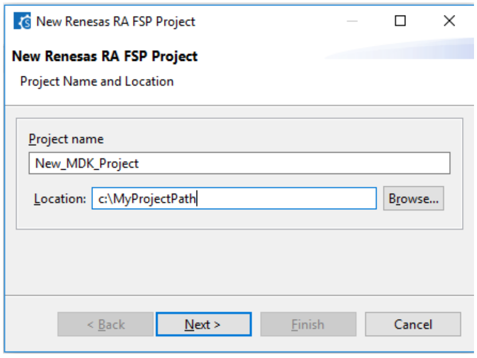


Figure 125: RA SC project settings

3. Select the target device and IDE.

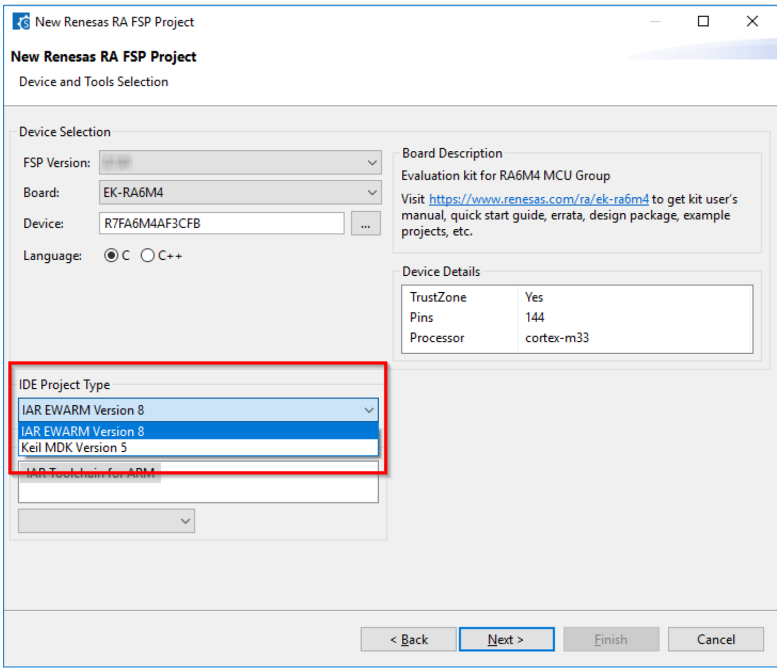


Figure 126: Target device and IDE selection

4. The rest of the project generator and FSP configuration is the same as e2 studio. Please

refer to the previous sections for details.

5. On completion of the FSP configuration, press "Generate Project Content"

6. A new Keil MDK project file will be generated in the project path. Double click this file to open MDK and continue development as usual.

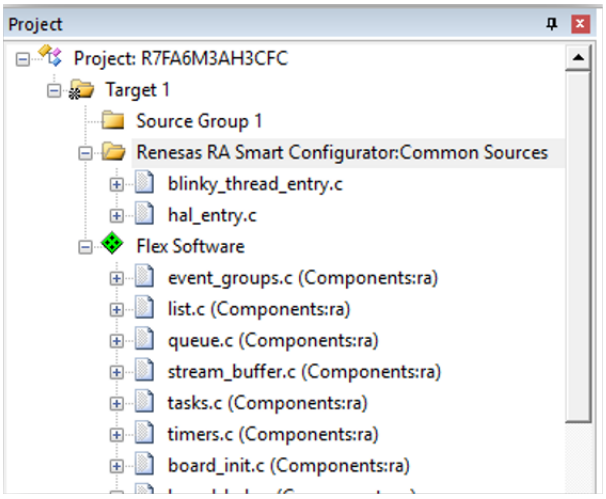


Figure 127: uVision project workspace with imported project data

2.6.2.3 Modify existing RA project

Once an initial project has been generated and configured, it is also possible to make changes using RA SC as follows.

Note

This setup only needs to be done once per project.

Set up the following links to RA SC:

1. In Keil MDK uVision, select **Tools > Customize Tools Menu...**.
2. Select the **new** icon and fill in the fields as follows for each tool:
   a. RA Smart Configurator:
      - Menu item name: Enter: RA Smart Configurator
      - Command: Select "..." and navigate to rasc.exe
      - Initial Folder: Enter: $P
      - Arguments: Enter: --device $D ––compiler ARMv6 configuration.xml
   b. Device Partition Manager:
      - Menu item name: Enter: Device Partition Manager
      - Command: Select "..." and navigate to rasc.exe
      - Initial Folder: Enter: $P
      - Arguments: Enter: -application com.renesas.cdt.ddsc.dpm.ui.dpmapplication configuration.xml "SL%L"

To reconfigure an existing project select **Tools > RA Smart Configurator**

To reconfigure the TrustZone partitions select **Tools > Device Partition Manager**

2.6.2.4 Build and Debug RA project

The project can be built by selecting the menu item **Project > Build Target** or tool bar item **Rebuild** or the keyboard shortcut F7.

Assembler, Compiler, Linker and Debugger settings can be changed in **Options for Target** dialog, which can be launched using the menu item **Project > Options for Target**, the tool bar item **Options for Target** or the keyboard shortcut Alt+F7.

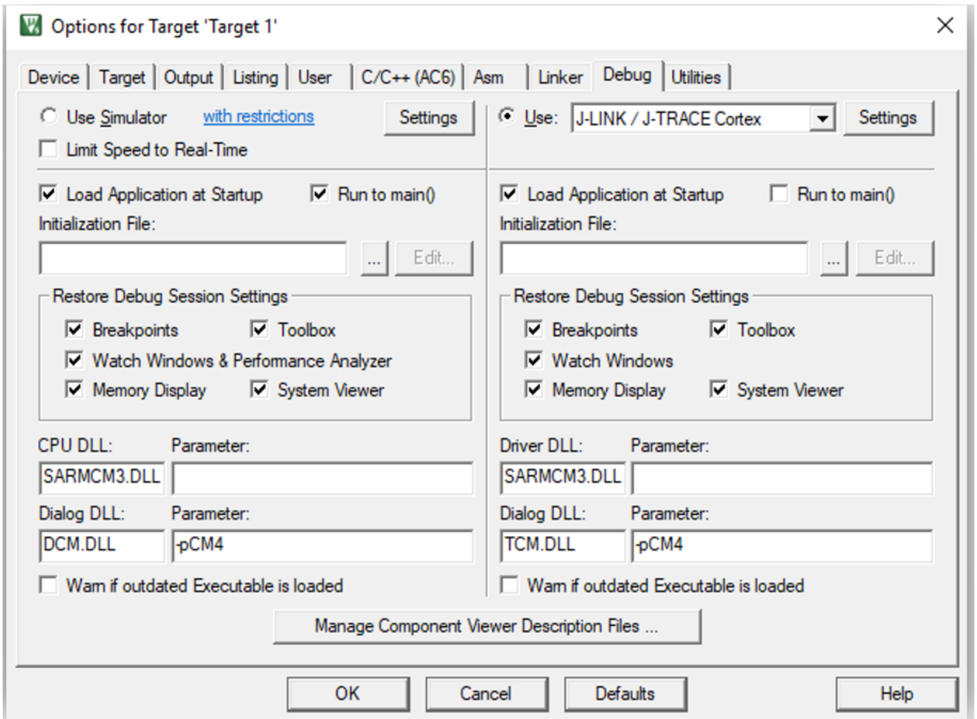


Figure 128: Options for Target

RA SC will set up the uVision project to debug the selected device using J-Link or J-Link OB debugger by default.

A Debug session can be started or stopped by selecting the menu item **Debug > Start/Stop Debug Session** or keyboard shortcut CTRL+F5. When debugging for the first time, J-Link firmware update may be needed if requested by the tool.

Refer to the documentation from Keil to get more information on the debug features in uVision. Note that not all features supported by uVision debugger are implemented in the J-Link interface. Consult SEGGER J-Link documentation for more information.

2.6.2.5 Notes and Restrictions

1. **When debugging a TrustZone based project, the Secure project image MUST be downloaded before the Non Secure project.**

2. For TrustZone enabled devices, the user will need to manually set up the memory partitions using the "Renesas Device Partition Manager" from inside RA SC before downloading.

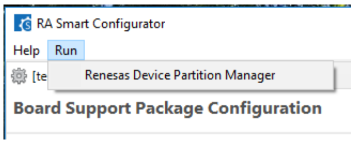


Figure 129: Renesas Device partition Manager

3. RA FSP contains a full set of drivers and middleware and may not be compatible with other CMSIS packs from Keil, Arm or third parties.
4. Flash programming is currently only supported through the debugger connection.

2.6.3 Using RA Smart Configurator with IAR EWARM

IAR Systems Embedded Workbench for Arm (EWARM) includes support for Renesas RA devices. These can be set up as bare metal designs within EWARM. However, most RA developers will want to integrate RA FSP drivers and middleware into their designs. RA SC will facilitate this.

RA SC generates a "Project Connection" file that can be loaded directly into EWARM to update project files.

2.6.3.1 Prerequisites

- IAR EWARM installed and licensed. Pleae refer to the Release notes for the version to be installed.
- RA SC and FSP Installed

2.6.3.2 Create new RA project

The following steps are required to create an RA project using IAR EWARM, RA SC and FSP:

1. Start the RA Smart Configurator.

2. Enter a project folder and project name.

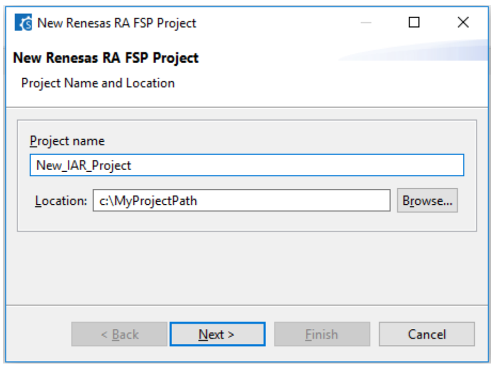


Figure 130: RA SC project settings

3. Select the target device and IDE.

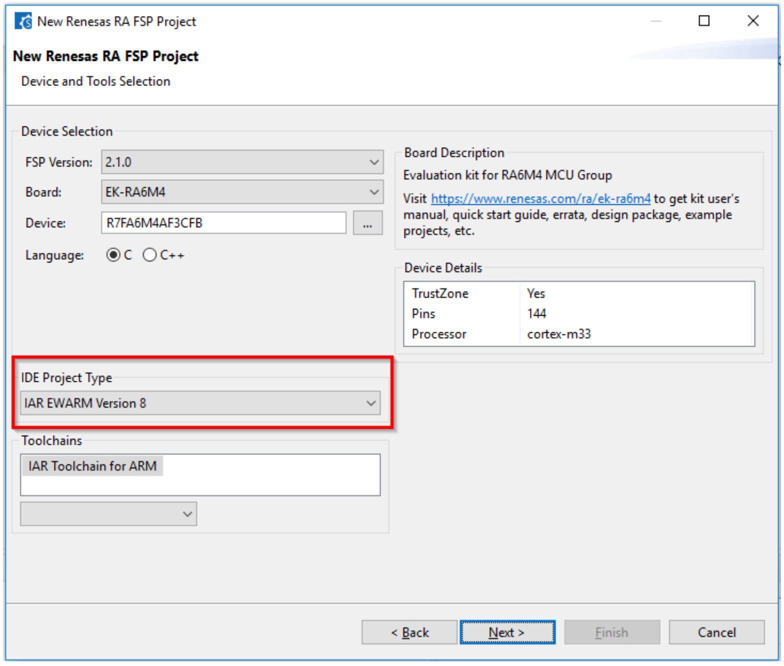


Figure 131: Target device and IDE selection

4. The rest of the project generator and FSP configuration operates the same as e2 studio. Refer to the previous sections for details.
5. On completion of the FSP configuration, press **Generate Project Content**.
6. A new IAR EWARM project file will be generated in the project path. Double click this file to open IAR EWARM and continue development as usual.
7. To Use RA SC with EWARM, RA SC needs to configured as a tool in EWARM by selecting the menu item **Tools > Configure Tools…**. Select **New** to create a new tool in the dialog shown and add the following information:
    - Menu Text: **RA Smart Configurator**
        a. Command: Select **Browse…** and navigate to rasc.exe in the installed RA SC
        b. Argument: –compiler IAR configuration.xml
        c. Initial Directory: $PROJ_DIR$
        d. Tool Available: Always
    - Menu Text: **Device Partition Manager**
        a. Command: Select **Browse…** and navigate to rasc.exe in the installed RA SC
        b. Argument: -application com.renesas.cdt.ddsc.dpm.ui.dpmapplication configuration.xml "$TARGET_PATH$"
        c. Initial Directory: $PROJ_DIR$
        d. Tool Available: Always
8. RA SC can now be re-launched from EWARM using the menu item **Tools > RA Smart Configurator**.
9. A Project connection needs to be set up in EWARM to build the project. Select **Project > Add Project Connection** in EWARM and select **IAR Project Connection**. Navigate to the project folder and select buildinfo.ipcf and click **Open**. The project can now build in EWARM.

2.6.3.3 Notes and Restrictions

When starting a TrustZone enabled debug session Partition sizes are checked automatically.

- If partition sizes are set correctly, the debug session will launch as normal.

- If partition sizes need to be changed, IAR EWARM will prompt to run the Renesas Device Partition Manager. Select **Yes**. The Device Partition Manager will start with the required partition sizes prefilled.
- Select **Set TrustZone secure / non-secure boundaries** as the only action.
- Enter debugger details, if required.
- Select **Run** to program the partitions.
- Return to the IDE and relaunch the debug session

Chapter 3 FSP Architecture

3.1 FSP Architecture Overview

This guide describes the Renesas Flexible Software Package (FSP) architecture and how to use the FSP Application Programming Interface (API).

3.1.1 C99 Use

The FSP uses the ISO/IEC 9899:1999 (C99) C programming language standard. Specific features introduced in C99 that are used include standard integer types (stdint.h), booleans (stdbool.h), designated initializers, and the ability to intermingle declarations and code.

3.1.2 Doxygen

Doxygen is the default documentation tool used by FSP. You can find Doxygen comments throughout the FSP source.

3.1.3 Weak Symbols

Weak symbols are used occasionally in the FSP. They are used to ensure that a project builds even when the user has not defined an optional function.

3.1.4 Memory Allocation

Dynamic memory allocation through use of the malloc() and free() functions are not used in FSP modules; all memory required by FSP modules is allocated in the application and passed to the module in a pointer. Exceptions are considered only for ports of 3rd party code that require dynamic memory.

3.1.5 FSP Terms

Term	Description	Reference
BSP	Short for Board Support Package. In the FSP the BSP provides just enough foundation to allow other FSP modules to work together without issue.	MCU Board Support Package

Module	Modules can be peripheral drivers, purely software, or anything in between. Each module consists of a folder with source code, documentation, and anything else that the customer needs to use the code effectively. Modules are independent units, but they may depend on other modules. Applications can be built by combining multiple modules to provide the user with the features they need.	FSP Modules
Driver	A driver is a specific kind of module that directly modifies registers on the MCU.	-
Interface	An interface contains API definitions that can be shared by modules with similar features. Interfaces are definitions only and do not add to code size.	FSP Interfaces
Stacks	The FSP architecture is designed such that modules work together to form a stack. A stack consists of a top level module and all its dependencies.	FSP Stacks
Module Instance	Single and independent instantiation of a module. An application may require two GPT timers. Each of these timers is a module instance of the r_gpt module.	-
Application	Code that is owned and maintained by the user. Application code may be based on sample application code provided by Renesas, but it is the responsibility of the user to maintain as necessary.	-

Callback Function	This term refers to a function that is called when an event occurs. As an example, suppose the user would like to be notified every second based on the RTC. As part of the RTC configuration, a callback function can be supplied that will be jumped to during each RTC interrupt. When a single callback services multiple events, the arguments contain the triggering event. Callback functions for interrupts should be kept short and handled carefully because when they are called the MCU is still inside of an interrupt, delaying any pending interrupts.	-

3.2 FSP Modules

Modules are the core building block of FSP. Modules can do many different things, but all modules share the basic concept of providing functionality upwards and requiring functionality from below.

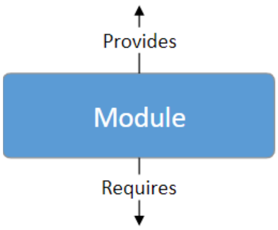


Figure 132: Modules

The amount of functionality provided by a module is determined based on functional use cases. Common functionality required by multiple modules is often placed into a self-contained submodule so it can be reused. Code size, speed and complexity are also considered when defining a module.

The simplest FSP application consists of one module with the Board Support Package (BSP) and the user application on top.

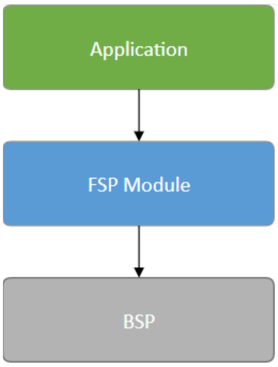


Figure 133: Module with application

The Board Support Package (BSP) is the foundation for FSP modules, providing functionality to determine the MCU used as well as configuring clocks, interrupts and pins. For the sake of clarity, the BSP will be omitted from further diagrams.

3.3 FSP Stacks

When modules are layered atop one another, an FSP stack is formed. The stacking process is performed by matching what one module provides with what another module requires. For example, the SPI module (Serial Peripheral Interface (r_spi)) requires a module that provides the transfer interface (Transfer Interface) to send or receive data without a CPU interrupt. The transfer interface requirement can be fulfilled by the DTC driver module (Data Transfer Controller (r_dtc)).

Through this methodology the same code can be shared by several modules simultaneously. The example below illustrates how the same DTC module can be used with SPI (Serial Peripheral Interface (r_spi)), UART (Serial Communications Interface (SCI) UART (r_sci_uart)) and SDHI (SD/MMC Host Interface (r_sdhi)).

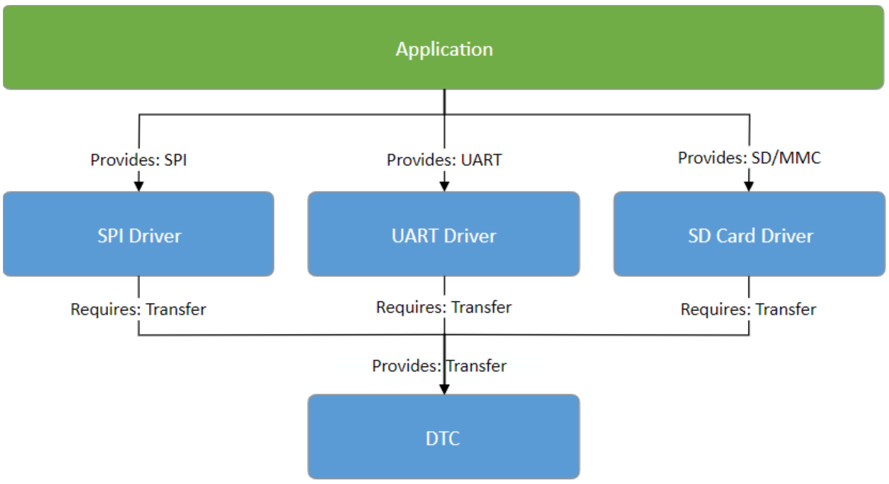


Figure 134: Stacks -- Shared DTC Module

The ability to stack modules ensures the flexibility of the architecture as a whole. If multiple modules include the same functionality issues arise when application features must work across different user designs. To ensure that modules are reusable, any dependent modules must be capable of being swapped out for other modules that provide the same features. The FSP architecture provides this flexibility to swap modules in and out through the use of FSP interfaces.

3.4 FSP Interfaces

At the architecture level, interfaces are the way that modules provide common features. This commonality allows modules that adhere to the same interface to be used interchangeably. Interfaces can be thought of as a contract between two modules - the modules agree to work together using the information that was established in the contract.

On RA hardware there is occasionally an overlap of features between different peripherals. For example, I2C communications can be achieved through use of the IIC peripheral or the SCI peripheral. However, there is a difference in the level of features provided by both peripherals; in I2C mode the SCI peripheral will only support a subset of the capabilities of the fully-featured IIC.

Interfaces aim to provide support for the common features that most users would expect. This means that some of the advanced features of a peripheral (such as IIC) might not be available in the interface. In most cases these features are still available through interface extensions.

In FSP design, interfaces are defined in header files. All interface header files are located in the folder ra/fsp/inc/api and end with *_api.h. Interface extensions are defined in header files in the folder ra/fsp/inc/instances. The following sections detail what makes up an interface.

3.4.1 FSP Interface Enumerations

Whenever possible, interfaces use typed enumerations for function parameters and structure members.

```
typedef enum e_i2c_master_addr_mode
{
    I2C_MASTER_ADDR_MODE_7BIT = 1,    ///< Use 7-bit addressing mode
    I2C_MASTER_ADDR_MODE_10BIT = 2,    ///< Use 10-bit addressing mode
} i2c_master_addr_mode_t;
```

Enumerations remove uncertainty when deciding what values are available for a parameter. FSP enumeration options follow a strict naming convention where the name of the type is prefixed on the available options. Combining the naming convention with the autocomplete feature available in e2 studio (Ctrl + Space) provides the benefits of rapid coding while maintaining high readability.

3.4.2 FSP Interface Callback Functions

Callback functions allow modules to asynchronously alert the user application when an event has occurred, such as when a byte has been received over a UART channel or an IRQ pin is toggled. FSP driver modules define and handle the interrupt service routines for RA MCU peripherals to ensure any required hardware procedures are implemented. The interrupt service routines in FSP modules then call the user-defined callbacks to allow the application to respond.

Callback functions must be defined in the user application. They always return void and take a structure for their one parameter. The structure is defined in the interface for the module and is named <interface>_callback_args_t. The contents of the structure may vary depending on the interface, but two members are common: event and p_context.

The event member is an enumeration defined in the interface used by the application to determine why the callback was called. Using the UART example, the callback could be triggered for many different reasons, including when a byte is received, all bytes have been transmitted, or a framing error has occurred. The event member allows the application to determine which of these three events has occurred and handle it appropriately.

The p_context member is used for providing user-specified data to the callback function. In many cases a callback function is shared between multiple channels or module instances; when the callback occurs, the code handling the callback needs context information so that it can determine which module instance the callback is for. For example, if the callback wanted to make an FSP API call in the callback, then at a minimum the callback will need a reference to the relevant control structure. To make this easy, the user can provide a pointer to the control structure as the p_context. When the callback occurs, the control structure is passed in the p_context element of the callback structure.

Callback functions are called from within an interrupt service routine. For this reason callback functions should be kept as short as possible so they do not affect the real time performance of the user's system. An example skeleton function for the flash interface callback is shown below.

```
void flash_callback (flash_callback_args_t * p_args)
{
 /* See what event caused this callback. */
 switch (p_args->event)
    {
 case FLASH_EVENT_ERASE_COMPLETE:
      {
 /* Handle event. */
 break;
       }
 case FLASH_EVENT_WRITE_COMPLETE:
      {
 /* Handle event. */
 break;
       }
 case FLASH_EVENT_BLANK:
      {
 /* Handle event. */
 break;
       }
 case FLASH_EVENT_NOT_BLANK:
```

```
     {
/* Handle event. */
break;
      }
case FLASH_EVENT_ERR_DF_ACCESS:
     {
/* Handle error. */
break;
      }
case FLASH_EVENT_ERR_CF_ACCESS:
     {
/* Handle error. */
break;
      }
case FLASH_EVENT_ERR_CMD_LOCKED:
     {
/* Handle error. */
break;
      }
case FLASH_EVENT_ERR_FAILURE:
     {
/* Handle error. */
break;
      }
case FLASH_EVENT_ERR_ONE_BIT:
     {
/* Handle error. */
break;
      }
   }
}
```

When a module is not directly used in the user application (that is, it is not the top layer of the stack), its callback function will be handled by the module above. For example, if a module requires a UART interface module the upper layer module will control and use the UART's callback function. In this case the user would not need to create a callback function for the UART module in their application code.

3.4.3 FSP Interface Data Structures

At a minimum, all FSP interfaces include three data structures: a configuration structure, an API structure, and an instance structure.

3.4.3.1 FSP Interface Configuration Structure

The configuration structure is used for the initial configuration of a module during the <MODULE>_Open() call. The structure consists of members such as channel number, bitrate, and operating mode.

The configuration structure is used purely as an input into the module. It may be stored and referenced by the module, so the configuration structure and anything it references must persist as long as the module is open.

The configuration structure is allocated for each module instance in files generated by the RA Configuration editor.

When FSP stacks are used, it is also important to understand that configuration structures only have members that apply to the current interface. If multiple layers in the same stack define the same configuration parameters then it becomes difficult to know where to modify the option. For example, the baud rate for a UART is only defined in the UART module instance. Any modules that use the UART interface rely on the baud rate being provided in the UART module instance and do not offer it in their own configuration structures.

3.4.3.2 FSP Interface API Structure

All interfaces include an API structure which contains function pointers for all the supported interface functions. An example structure for the Digital to Analog Converter (r_dac) is shown below.

```
typedef struct st_dac_api
{
    /** Initial configuration.
    * @par Implemented as
    * - @ref R_DAC_Open()
    * - @ref R_DAC8_Open()
    *
    * @param[in] p_ctrl Pointer to control block. Must be declared by user. Elements
set here.
    * @param[in] p_cfg Pointer to configuration structure. All elements of this
structure must be set by user.
    */
    fsp_err_t (* open)(dac_ctrl_t * const p_ctrl, dac_cfg_t const * const p_cfg);
```

```
    /** Close the D/A Converter.
    * @par Implemented as
    * - @ref R_DAC_Close()
    * - @ref R_DAC8_Close()
    *
    * @param[in] p_ctrl Control block set in @ref dac_api_t::open call for this
timer.
    */
    fsp_err_t (* close)(dac_ctrl_t * const p_ctrl);
    /** Write sample value to the D/A Converter.
    * @par Implemented as
    * - @ref R_DAC_Write()
    * - @ref R_DAC8_Write()
    *
    * @param[in] p_ctrl Control block set in @ref dac_api_t::open call for this
timer.
    * @param[in] value Sample value to be written to the D/A Converter.
    */
    fsp_err_t (* write)(dac_ctrl_t * const p_ctrl, uint16_t value);
    /** Start the D/A Converter if it has not been started yet.
    * @par Implemented as
    * - @ref R_DAC_Start()
    * - @ref R_DAC8_Start()
    *
    * @param[in] p_ctrl Control block set in @ref dac_api_t::open call for this
timer.
    */
    fsp_err_t (* start)(dac_ctrl_t * const p_ctrl);
    /** Stop the D/A Converter if the converter is running.
    * @par Implemented as
    * - @ref R_DAC_Stop()
    * - @ref R_DAC8_Stop()
    *
    * @param[in] p_ctrl Control block set in @ref dac_api_t::open call for this
```

```
timer.
    */
    fsp_err_t (* stop)(dac_ctrl_t * const p_ctrl);
} dac_api_t;
```

The API structure is what allows for modules to easily be swapped in and out for other modules that are instances of the same interface. Let's look at an example application using the DAC interface above.

RA MCUs have an internal DAC peripheral. If the DAC API structure in the DAC interface is not used the application can make calls directly into the module. In the example below the application is making calls to the R_DAC_Write() function which is provided in the r_dac module.

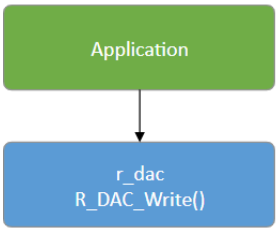


Figure 135: DAC Write example

Now let's assume that the user needs more DAC channels than are available on the MCU and decides to add an external DAC module named dac_external using I2C for communications. The application must now distinguish between the two modules, adding complexity and further dependencies to the application.

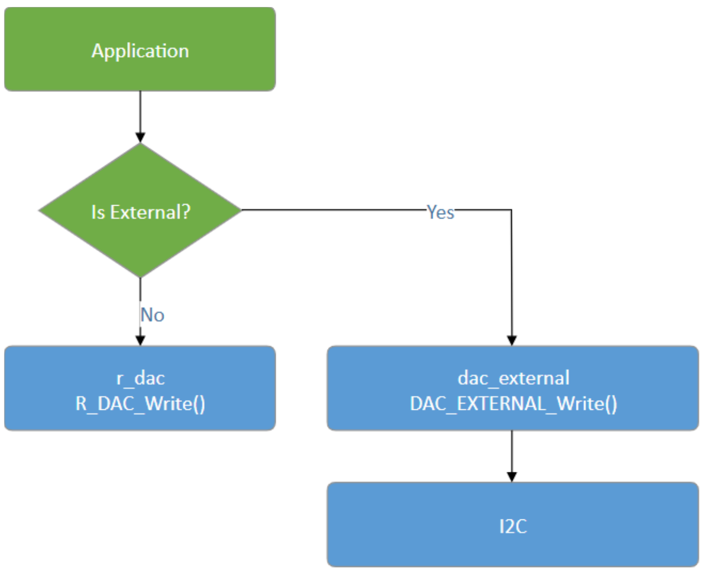


Figure 136: DAC Write with two write modules

The use of interfaces and the API structure allows for the use of an abstracted DAC. This means that no extra logic is needed if the user's dac_external module implements the FSP DAC interface, so the application no longer depends upon hard-coded module function names. Instead the application now depends on the DAC interface API which can be implemented by any number of modules.

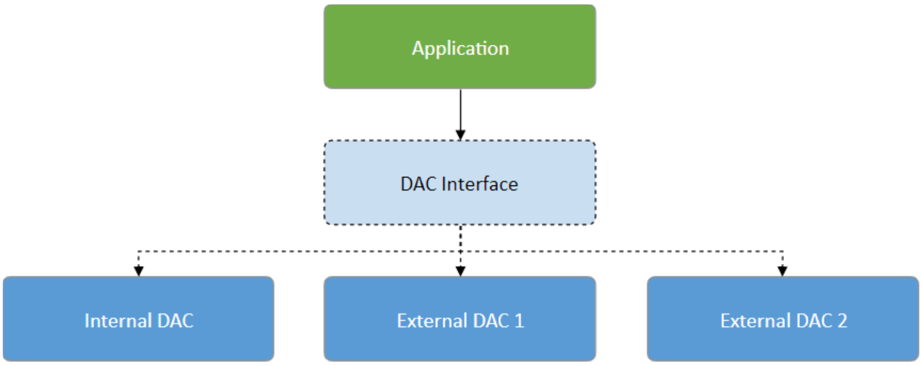


Figure 137: DAC Interface

3.4.3.3 FSP Interface Instance Structure

Every FSP interface also has an instance structure. The instance structure encapsulates everything required to use the module:

- A pointer to the instance API structure (FSP Instance API)
- A pointer to the configuration structure
- A pointer to the control structure

The instance structure is not required at the application layer. It is used to connect modules to their dependencies (other than the BSP).

Instance structures have a standardized name of <interface>_instance_t. An example from the Transfer Interface is shown below.

```
typedef struct st_transfer_instance
{
  transfer_ctrl_t     * p_ctrl; ///< Pointer to the control structure for this
instance
    transfer_cfg_t const * p_cfg;      ///< Pointer to the configuration structure
for this instance
    transfer_api_t const * p_api;      ///< Pointer to the API structure for this
instance
} transfer_instance_t;
```

Note that when an instance structure variable is declared, the API is the only thing that is instance specific, not *module instance* specific. This is because all module instances of the same module share the same underlying module source code. If SPI is being used on SCI channels 0 and 2 then both module instances use the same API while the configuration and control structures are typically different.

3.5 FSP Instances

While interfaces dictate the features that are provided, instances actually implement those features. Each instance is tied to a specific interface. Instances use the enumerations, data structures, and API prototypes from the interface. This allows an application that uses an interface to swap out the instance when needed.

On RA MCUs some peripherals are used to implement multiple interfaces. In the example below the IIC and SPI peripherals map to only one interface each while the SCI peripheral implements three interfaces.

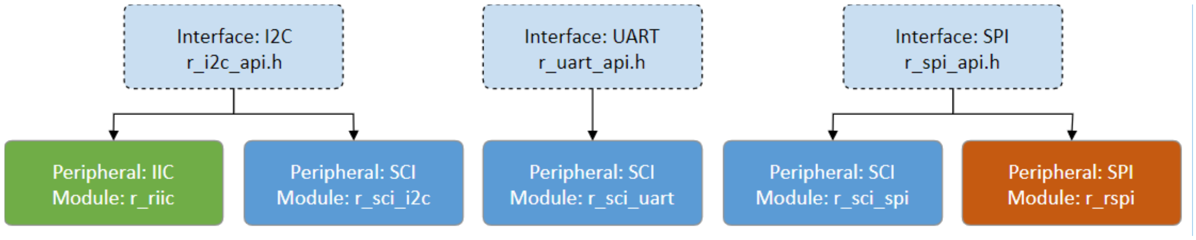


Figure 138: Instances

In FSP design, instances consist of the interface extension and API defined in the instance header file located in the folder ra/fsp/inc/instances and the module source ra/fsp/src/<module>.

3.5.1 FSP Instance Control Structure

The control structure is used as a unique identifier for the module instance and contains memory required by the module. Elements in the control structure are owned by the module and *must not be modified* by the application. The user allocates storage for a control structure, often as a global variable, then sends a pointer to it into the <MODULE>_Open() call for a module. At this point, the module initializes the structure as needed. The user must then send in a pointer to the control structure for all subsequent module calls.

3.5.2 FSP Interface Extensions

In some cases, instances require more information than is provided in the interface. This situation can occur in the following cases:

- An instance offers extra features that are not common to most instances of the interface. An example of this is the start source selection of the GPT (General PWM Timer (r_gpt)). The GPT can be configured to start based on hardware events such as a falling edge on a trigger pin. This feature is not common to all timers, so it is included in the GPT instance.
- An interface must be very generic out of necessity. As an interface becomes more generic, the number of possible instances increases. An example of an interface that must be generic is a block media interface that abstracts functions required by a file system. Possible instances include SD card, SPI Flash, SDRAM, USB, and many more.

The p_extend member provides this extension function.

Use of interface extensions is not always necessary. Some instances do not offer an extension since all functionality is provided in the interface. In these cases the p_extend member can be set to NULL. The documentation for each instance indicates whether an interface extension is available and whether it is mandatory or optional.

3.5.2.1 FSP Extended Configuration Structure

When extended configuration is required it can be supplied through the p_extend parameter of the interface configuration structure.

The extended configuration structure is part of the instance, but it is also still considered to be part of the configuration structure. All usage notes about the configuration structure described in FSP Interface Configuration Structure apply to the extended configuration structure as well.

The extended configuration structure and all typed structures and enumerations required to define it make up the interface extension.

3.5.3 FSP Instance API

Each instance includes a constant global variable tying the interface API functions to the functions provided by the module. The name of this structure is standardized as g_<interface>_on_<instance>. Examples include g_spi_on_spi, g_transfer_on_dtc, and g_adc_on_adc. This structure is available to be used through an extern in the instance header file (r_spi.h, r_dtc.h, and r_adc.h respectively).

3.6 FSP API Standards

3.6.1 FSP Function Names

FSP functions start with the uppercase module name (<MODULE>). All modules have <MODULE>_Open() and <MODULE>_Close() functions. The <MODULE>_Open() function must be called before any of the other functions.

Other functions that will commonly be found are <MODULE>_Read(), <MODULE>_Write(), <MODULE>_InfoGet(), and <MODULE>_StatusGet(). The <MODULE>_StatusGet() function provides a status that could change asynchronously, while <MODULE>_InfoGet() provides information that cannot change after open or can only be updated by API calls. Example function names include:

- R_SPI_Read(), R_SPI_Write(), R_SPI_WriteRead()
- R_SDHI_StatusGet()
- R_RTC_CalendarAlarmSet(), R_RTC_CalendarAlarmGet()
- R_FLASH_HP_AccessWindowSet(), R_FLASH_HP_AccessWindowClear()

3.6.2 Use of const in API parameters

The const qualifier is used with API parameters whenever possible. An example case is shown below.

```
fsp_err_t R_FLASH_HP_Open(flash_ctrl_t * const p_api_ctrl, flash_cfg_t const * const
p_cfg);
```

In this example, flash_cfg_t is a structure of configuration parameters for the r_flash_hp module. The parameter p_cfg is a pointer to this structure. The first const qualifier on p_cfg ensures the flash_cfg_t structure cannot be modified by R_FLASH_HP_Open(). This allows the structure to be allocated as a const variable and stored in ROM instead of RAM.

The const qualifier after the pointer star for both p_ctrl and p_cfg ensures the FSP function does not modify the input pointer addresses. While not fool-proof by any means this does provide some extra checking inside the FSP code to ensure that arguments that should not be altered are treated as such.

3.6.3 FSP Version Information

The BSP provides a function R_FSP_VersionGet() which fills in a structure of type fsp_pack_version_t. This can be used to determine the FSP version at runtime.

There are also FSP_VERSION_* macros in fsp_version.h that can be used to determine the FSP version at build time.

3.7 FSP Build Time Configurations

All modules have a build-time configuration header file. Most configuration options are supplied at run time, though options that are rarely used or apply to all instances of a module may be moved to build time. The advantage of using a build-time configuration option is to potentially reduce code size reduction by removing an unused feature.

All modules have a build time option to enable or disable parameter checking for the module. FSP modules check function arguments for validity when possible, though this feature is disabled by default to reduce code size. Enabling it can help catch parameter errors during development and debugging. By default, each module's parameter checking configuration inherits the BSP parameter checking setting (set on the BSP tab of the RA Configuration editor). Leaving each module's parameter checking configuration set to Default (BSP) allows parameter checking to be enabled or disabled globally in all FSP code through the parameter checking setting on the BSP tab.

If an error condition can reasonably be avoided it is only checked in a section of code that can be disabled by disabling parameter checking. Most FSP APIs can only return FSP_SUCCESS if parameter checking is disabled. An example of an error that cannot be reasonably avoided is the "bus busy" error that occurs when another master is using an I2C bus. This type of error can be returned even if parameter checking is disabled.

3.8 FSP File Structure

The high-level file structure of an FSP project is shown below.

```
ra_gen
ra
+---fsp
    +---inc
    |   +---api
    |   \---instances
    \---src
       +---bsp
```

```
       \---r_module
ra_cfg
+---fsp_cfg
    +---bsp
    +---driver
```

Directly underneath the base ra folder the folders are split into the source and include folders. Include folders are kept separate from the source for easy browsing and easy setup of include paths.

The ra_gen folder contains code generated by the RA Configuration editor. This includes global variables for the control structure and configuration structure for each module.

The ra_cfg folder is where configuration header files are stored for each module. See FSP Build Time Configurations for information on what is provided in these header files.

3.9 FSP TrustZone Support

TrustZone support for FSP is primarily handled in the RA Configuration Tool.

3.9.1 FSP TrustZone Projects

During development of a TrustZone project, users create an RA TrustZone Secure Project first, followed by an RA TrustZone Non-secure Project that is linked to the RA TrustZone Secure Project. Allocation of secure memory is handled automatically within the tooling. The non-secure project starts at the required alignment boundary beyond the memory taken by the secure project.

3.9.2 Non-Secure Callable Guard Functions

The tooling generates guard functions for any module marked as Non-secure Callable. These guard functions are owned by the application once generated, so they can be modified as necessary by the secure application developer.

The default non-secure callable guard functions limit the configuration and control structure to the structures generated in the secure project. They also check any input pointers to ensure the caller does not overwrite secure memory.

3.9.3 Callbacks in Non-Secure from Non-Secure Callable Modules

If the non-secure project needs a callback function from a non-secure callable module, the callback can be registered after the module is opened using the callback_set() guard function.

3.9.4 Additional TrustZone Information

The following resources provide technical background, application notes and example projects that demonstrate key TrustZone concepts and implementation procedures.

- The Benefits of Using Arm® TrustZone® in Your Design (Brochure)
- RA Arm® TrustZone® Tooling Primer (Application Note)
- Renesas RA Family Security Design with Arm® TrustZone® - IP Protection (Application

Note)

- Renesas RA Family Securing Data at Rest Using the Arm® TrustZone® (Application Note)

3.10 FSP Architecture in Practice

3.10.1 FSP Connecting Layers

FSP modules are meant to be both reusable and stackable. It is important to remember that modules are not dependent upon other modules, but upon other interfaces. The user is then free to fulfill the interface using the instance that best fits their needs.

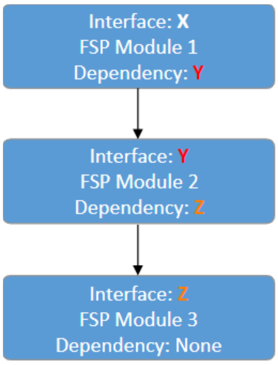


Figure 139: Connecting layers

In the image above interface Y is a dependency of interface X and has its own dependency on interface Z. Interface X only has a dependency on interface Y. Interface X has no knowledge of interface Z. This is a requirement for ensuring that layers can easily be swapped out.

3.10.2 Using FSP Modules in an Application

The typical use of an FSP module involves generating required module data then using the API in the application.

3.10.2.1 Create a Module Instance in the RA Configuration Editor

The RA Configuration editor (available both in the Renesas e2 studio IDE as well as through the standalone RA Smart Configurator) provides a graphical user interface for setting the parameters of the interface and instance configuration structures. It also automatically includes those structures (once they are configured in the GUI) in application-specific header files that can be included in application code.

The RA Configuration editor allocates storage for the control structures, all required configuration structures, and the instance structure in generated files in the ra_gen folder. Use the **Properties** window to set the values for the members of the configuration structures as needed. Refer to the Configuration section of the module usage notes for documentation about the configuration options.

If the interface has a callback function option then the application must declare and define the function. The return value is always of type void and the parameter to the function is a typed structure of name <interface>_callback_args_t. Once the function has been defined, assign its name to the p_callback member of the configuration structure. Callback function names can be assigned through the **Properties** window for the selected module.

3.10.2.2 Use the Instance API in the Application

Call the module's <MODULE>_Open() function. Pass pointers to the generated control structure and configuration structure. The names of these structures are based on the 'Name' field provided in the configuration editor. The control structure is <Name>_ctrl and the configuration structure is <Name>_cfg. An example <MODULE>_Open() call for an r_rtc module instance named g_clock is:

```
R_RTC_Open(&g_clock_ctrl, &g_clock_cfg);
```

Note

Each layer in the FSP Stack is responsible for calling the API functions of its dependencies. This means that users are only responsible for calling the API functions at the layer at which they are interfacing. Using the example above of a SPI module with a DTC dependency, the application uses only SPI APIs. The application starts by calling R_SPI_Open(). Internally, the SPI module opens the DTC. It locates R_DTC_Open() by accessing the dependent transfer interface function pointers from the pointers DTC instances (spi_cfg_t::p_transfer_tx and spi_cfg_t::p_transfer_rx) to open the DTC.

Refer to the module usage notes for example code to help get started with any particular module.

Chapter 4 API Reference

This section includes the FSP API Reference for the Module and Interface level functions.

►BSP	Common code shared by FSP drivers
►Modules	Modules are the smallest unit of software available in the FSP. Each module implements one interface
►Interfaces	The FSP interfaces provide APIs for common functionality. They can be implemented by one or more modules. Modules can use other modules as dependencies using this interface layer

4.1 BSP

Detailed Description

Common code shared by FSP drivers.

Modules

	Common Error Codes
	MCU Board Support Package The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor.
	BSP I/O access This module provides basic read/write access to port pins.

Data Structures

union	fsp_pack_version_t
struct	fsp_pack_version_t.__unnamed__

Macros

#define	FSP_VERSION_MAJOR
#define	FSP_VERSION_MINOR
#define	FSP_VERSION_PATCH
#define	FSP_VERSION_BUILD
#define	FSP_VERSION_STRING
#define	FSP_VERSION_BUILD_STRING

Data Structure Documentation

◆ fsp_pack_version_t

union fsp_pack_version_t

FSP Pack version structure

Data Fields		
uint32_t	version_id	Version id
struct fsp_pack_version_t	__unnamed__	Code version parameters, little endian order.

◆ fsp_pack_version_t.__unnamed__

struct fsp_pack_version_t.__unnamed__

Code version parameters, little endian order.

Data Fields		
uint8_t	build	Build version of FSP Pack.
uint8_t	patch	Patch version of FSP Pack.
uint8_t	minor	Minor version of FSP Pack.
uint8_t	major	Major version of FSP Pack.

Macro Definition Documentation

◆ FSP_VERSION_MAJOR

#define FSP_VERSION_MAJOR

FSP pack major version.

◆ FSP_VERSION_MINOR

#define FSP_VERSION_MINOR
FSP pack minor version.

◆ FSP_VERSION_PATCH

#define FSP_VERSION_PATCH
FSP pack patch version.

◆ FSP_VERSION_BUILD

#define FSP_VERSION_BUILD
FSP pack version build number (currently unused).

◆ FSP_VERSION_STRING

#define FSP_VERSION_STRING
Public FSP version name.

◆ FSP_VERSION_BUILD_STRING

#define FSP_VERSION_BUILD_STRING
Unique FSP version ID.

4.1.1 Common Error Codes

BSP

Detailed Description

All FSP modules share these common error codes.

Macros

#define	FSP_PARAMETER_NOT_USED(p)
#define	FSP_CPP_HEADER

#define	FSP_HEADER
#define	FSP_SECURE_ARGUMENT

Enumerations

enum	fsp_err_t

Macro Definition Documentation

◆ FSP_PARAMETER_NOT_USED

#define FSP_PARAMETER_NOT_USED (p)

This macro is used to suppress compiler messages about a parameter not being used in a function. The nice thing about using this implementation is that it does not take any extra RAM or ROM.

◆ FSP_CPP_HEADER

#define FSP_CPP_HEADER

Determine if a C++ compiler is being used. If so, ensure that standard C is used to process the API information.

◆ FSP_HEADER

#define FSP_HEADER

FSP Header and Footer definitions

◆ FSP_SECURE_ARGUMENT

#define FSP_SECURE_ARGUMENT

Macro to be used when argument to function is ignored since function call is NSC and the parameter is statically defined on the Secure side.

Enumeration Type Documentation

◆ fsp_err_t

enum fsp_err_t

Common error codes

Enumerator	
FSP_ERR_ASSERTION	A critical assertion has failed.
FSP_ERR_INVALID_POINTER	Pointer points to invalid memory location.
FSP_ERR_INVALID_ARGUMENT	Invalid input parameter.
FSP_ERR_INVALID_CHANNEL	Selected channel does not exist.
FSP_ERR_INVALID_MODE	Unsupported or incorrect mode.
FSP_ERR_UNSUPPORTED	Selected mode not supported by this API.
FSP_ERR_NOT_OPEN	Requested channel is not configured or API not open.
FSP_ERR_IN_USE	Channel/peripheral is running/busy.
FSP_ERR_OUT_OF_MEMORY	Allocate more memory in the driver's cfg.h.
FSP_ERR_HW_LOCKED	Hardware is locked.
FSP_ERR_IRQ_BSP_DISABLED	IRQ not enabled in BSP.
FSP_ERR_OVERFLOW	Hardware overflow.
FSP_ERR_UNDERFLOW	Hardware underflow.
FSP_ERR_ALREADY_OPEN	Requested channel is already open in a different configuration.
FSP_ERR_APPROXIMATION	Could not set value to exact result.
FSP_ERR_CLAMPED	Value had to be limited for some reason.
FSP_ERR_INVALID_RATE	Selected rate could not be met.
FSP_ERR_ABORTED	An operation was aborted.
FSP_ERR_NOT_ENABLED	Requested operation is not enabled.
FSP_ERR_TIMEOUT	Timeout error.

FSP_ERR_INVALID_BLOCKS	Invalid number of blocks supplied.
FSP_ERR_INVALID_ADDRESS	Invalid address supplied.
FSP_ERR_INVALID_SIZE	Invalid size/length supplied for operation.
FSP_ERR_WRITE_FAILED	Write operation failed.
FSP_ERR_ERASE_FAILED	Erase operation failed.
FSP_ERR_INVALID_CALL	Invalid function call is made.
FSP_ERR_INVALID_HW_CONDITION	Detected hardware is in invalid condition.
FSP_ERR_INVALID_FACTORY_FLASH	Factory flash is not available on this MCU.
FSP_ERR_INVALID_STATE	API or command not valid in the current state.
FSP_ERR_NOT_ERASED	Erase verification failed.
FSP_ERR_SECTOR_RELEASE_FAILED	Sector release failed.
FSP_ERR_NOT_INITIALIZED	Required initialization not complete.
FSP_ERR_NOT_FOUND	The requested item could not be found.
FSP_ERR_NO_CALLBACK_MEMORY	Non-secure callback memory not provided for non-secure callback.
FSP_ERR_BUFFER_EMPTY	No data available in buffer.
FSP_ERR_INVALID_DATA	Accuracy of data is not guaranteed.
FSP_ERR_INTERNAL	Internal error.
FSP_ERR_WAIT_ABORTED	Wait aborted.
FSP_ERR_FRAMING	Framing error occurs.
FSP_ERR_BREAK_DETECT	Break signal detects.
FSP_ERR_PARITY	Parity error occurs.
FSP_ERR_RXBUF_OVERFLOW	Receive queue overflow.
FSP_ERR_QUEUE_UNAVAILABLE	Can't open s/w queue.
FSP_ERR_INSUFFICIENT_SPACE	Not enough space in transmission circular

	buffer.
FSP_ERR_INSUFFICIENT_DATA	Not enough data in receive circular buffer.
FSP_ERR_TRANSFER_ABORTED	The data transfer was aborted.
FSP_ERR_MODE_FAULT	Mode fault error.
FSP_ERR_READ_OVERFLOW	Read overflow.
FSP_ERR_SPI_PARITY	Parity error.
FSP_ERR_OVERRUN	Overrun error.
FSP_ERR_CLOCK_INACTIVE	Inactive clock specified as system clock.
FSP_ERR_CLOCK_ACTIVE	Active clock source cannot be modified without stopping first.
FSP_ERR_NOT_STABILIZED	Clock has not stabilized after its been turned on/off.
FSP_ERR_PLL_SRC_INACTIVE	PLL initialization attempted when PLL source is turned off.
FSP_ERR_OSC_STOP_DET_ENABLED	Illegal attempt to stop LOCO when Oscillation stop is enabled.
FSP_ERR_OSC_STOP_DETECTED	The Oscillation stop detection status flag is set.
FSP_ERR_OSC_STOP_CLOCK_ACTIVE	Attempt to clear Oscillation Stop Detect Status with PLL/MAIN_OSC active.
FSP_ERR_CLKOUT_EXCEEDED	Output on target output clock pin exceeds maximum supported limit.
FSP_ERR_USB_MODULE_ENABLED	USB clock configure request with USB Module enabled.
FSP_ERR_HARDWARE_TIMEOUT	A register read or write timed out.
FSP_ERR_LOW_VOLTAGE_MODE	Invalid clock setting attempted in low voltage mode.
FSP_ERR_PE_FAILURE	Unable to enter Programming mode.
FSP_ERR_CMD_LOCKED	Peripheral in command locked state.
FSP_ERR_FCLK	

	FCLK must be >= 4 MHz.
FSP_ERR_INVALID_LINKED_ADDRESS	Function or data are linked at an invalid region of memory.
FSP_ERR_BLANK_CHECK_FAILED	Blank check operation failed.
FSP_ERR_INVALID_CAC_REF_CLOCK	Measured clock rate < reference clock rate.
FSP_ERR_CLOCK_GENERATION	Clock cannot be specified as system clock.
FSP_ERR_INVALID_TIMING_SETTING	Invalid timing parameter.
FSP_ERR_INVALID_LAYER_SETTING	Invalid layer parameter.
FSP_ERR_INVALID_ALIGNMENT	Invalid memory alignment found.
FSP_ERR_INVALID_GAMMA_SETTING	Invalid gamma correction parameter.
FSP_ERR_INVALID_LAYER_FORMAT	Invalid color format in layer.
FSP_ERR_INVALID_UPDATE_TIMING	Invalid timing for register update.
FSP_ERR_INVALID_CLUT_ACCESS	Invalid access to CLUT entry.
FSP_ERR_INVALID_FADE_SETTING	Invalid fade-in/fade-out setting.
FSP_ERR_INVALID_BRIGHTNESS_SETTING	Invalid gamma correction parameter.
FSP_ERR_JPEG_ERR	JPEG error.
FSP_ERR_JPEG_SOI_NOT_DETECTED	SOI not detected until EOI detected.
FSP_ERR_JPEG_SOF1_TO_SOFF_DETECTED	SOF1 to SOFF detected.
FSP_ERR_JPEG_UNSUPPORTED_PIXEL_FORMAT	Unprovided pixel format detected.
FSP_ERR_JPEG_SOF_ACCURACY_ERROR	SOF accuracy error: other than 8 detected.
FSP_ERR_JPEG_DQT_ACCURACY_ERROR	DQT accuracy error: other than 0 detected.
FSP_ERR_JPEG_COMPONENT_ERROR1	Component error 1: the number of SOF0 header components detected is other than 1, 3, or 4.
FSP_ERR_JPEG_COMPONENT_ERROR2	Component error 2: the number of components differs between SOF0 header and SOS.
FSP_ERR_JPEG_SOF0_DQT_DHT_NOT_DETECTED	SOF0, DQT, and DHT not detected when SOS

	detected.
FSP_ERR_JPEG_SOS_NOT_DETECTED	SOS not detected: SOS not detected until EOI detected.
FSP_ERR_JPEG_EOI_NOT_DETECTED	EOI not detected (default)
FSP_ERR_JPEG_RESTART_INTERVAL_DATA_NUMBER_ERROR	Restart interval data number error detected.
FSP_ERR_JPEG_IMAGE_SIZE_ERROR	Image size error detected.
FSP_ERR_JPEG_LAST_MCU_DATA_NUMBER_ERROR	Last MCU data number error detected.
FSP_ERR_JPEG_BLOCK_DATA_NUMBER_ERROR	Block data number error detected.
FSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH	User provided buffer size not enough.
FSP_ERR_JPEG_UNSUPPORTED_IMAGE_SIZE	JPEG Image size is not aligned with MCU.
FSP_ERR_CALIBRATE_FAILED	Calibration failed.
FSP_ERR_IP_HARDWARE_NOT_PRESENT	Requested IP does not exist on this device.
FSP_ERR_IP_UNIT_NOT_PRESENT	Requested unit does not exist on this device.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel does not exist on this device.
FSP_ERR_NO_MORE_BUFFER	No more buffer found in the memory block pool.
FSP_ERR_ILLEGAL_BUFFER_ADDRESS	Buffer address is out of block memory pool.
FSP_ERR_INVALID_WORKBUFFER_SIZE	Work buffer size is invalid.
FSP_ERR_INVALID_MSG_BUFFER_SIZE	Message buffer size is invalid.
FSP_ERR_TOO_MANY_BUFFERS	Number of buffer is too many.
FSP_ERR_NO_SUBSCRIBER_FOUND	No message subscriber found.
FSP_ERR_MESSAGE_QUEUE_EMPTY	No message found in the message queue.
FSP_ERR_MESSAGE_QUEUE_FULL	No room for new message in the message queue.
FSP_ERR_ILLEGAL_SUBSCRIBER_LISTS	Message subscriber lists is illegal.

FSP_ERR_BUFFER_RELEASED	Buffer has been released.
FSP_ERR_D2D_ERROR_INIT	D/AVE 2D has an error in the initialization.
FSP_ERR_D2D_ERROR_DEINIT	D/AVE 2D has an error in the initialization.
FSP_ERR_D2D_ERROR_RENDERING	D/AVE 2D has an error in the rendering.
FSP_ERR_D2D_ERROR_SIZE	D/AVE 2D has an error in the rendering.
FSP_ERR_ETHER_ERROR_NO_DATA	No Data in Receive buffer.
FSP_ERR_ETHER_ERROR_LINK	ETHERC/EDMAC has an error in the Auto-negotiation.
FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE	As a Magic Packet is being detected, and transmission/reception is not enabled.
FSP_ERR_ETHER_ERROR_TRANSMIT_BUFFER_FULL	Transmit buffer is not empty.
FSP_ERR_ETHER_ERROR_FILTERING	Detect multicast frame when multicast frame filtering enable.
FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION	ETHERC/EDMAC has an error in the phy communication.
FSP_ERR_ETHER_RECEIVE_BUFFER_ACTIVE	Receive buffer is active.
FSP_ERR_ETHER_PHY_ERROR_LINK	PHY is not link up.
FSP_ERR_ETHER_PHY_NOT_READY	PHY has an error in the Auto-negotiation.
FSP_ERR_QUEUE_FULL	Queue is full, cannot queue another data.
FSP_ERR_QUEUE_EMPTY	Queue is empty, no data to dequeue.
FSP_ERR_CTSU_SCANNING	Scanning.
FSP_ERR_CTSU_NOT_GET_DATA	Not processed previous scan data.
FSP_ERR_CTSU_INCOMPLETE_TUNING	Incomplete initial offset tuning.
FSP_ERR_CTSU_DIAG_NOT_YET	Diagnosis of data collected no yet.
FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE	Diagnosis of LDO over voltage failed.
FSP_ERR_CTSU_DIAG_CCO_HIGH	Diagnosis of CCO into 19.2uA failed.
FSP_ERR_CTSU_DIAG_CCO_LOW	

	Diagnosis of CCO into 2.4uA failed.
FSP_ERR_CTSU_DIAG_SSCG	Diagnosis of SSCG frequency failed.
FSP_ERR_CTSU_DIAG_DAC	Diagnosis of non-touch count value failed.
FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE	Diagnosis of LDO output voltage failed.
FSP_ERR_CTSU_DIAG_OVER_VOLTAGE	Diagnosis of over voltage detection circuit failed.
FSP_ERR_CTSU_DIAG_OVER_CURRENT	Diagnosis of over current detection circuit failed.
FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE	Diagnosis of LDO internal resistance value failed.
FSP_ERR_CTSU_DIAG_CURRENT_SOURCE	Diagnosis of Current source value failed.
FSP_ERR_CTSU_DIAG_SENSCLK_GAIN	Diagnosis of SENSCLK frequency gain failed.
FSP_ERR_CTSU_DIAG_SUCLK_GAIN	Diagnosis of SUCLK frequency gain failed.
FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY	Diagnosis of SUCLK clock recovery function failed.
FSP_ERR_CTSU_DIAG_CFC_GAIN	Diagnosis of CFC oscillator gain failed.
FSP_ERR_CARD_INIT_FAILED	SD card or eMMC device failed to initialize.
FSP_ERR_CARD_NOT_INSERTED	SD card not installed.
FSP_ERR_DEVICE_BUSY	Device is holding DAT0 low or another operation is ongoing.
FSP_ERR_CARD_NOT_INITIALIZED	SD card was removed.
FSP_ERR_CARD_WRITE_PROTECTED	Media is write protected.
FSP_ERR_TRANSFER_BUSY	Transfer in progress.
FSP_ERR_RESPONSE	Card did not respond or responded with an error.
FSP_ERR_MEDIA_FORMAT_FAILED	Media format failed.
FSP_ERR_MEDIA_OPEN_FAILED	Media open failed.
FSP_ERR_CAN_DATA_UNAVAILABLE	No data available.

FSP_ERR_CAN_MODE_SWITCH_FAILED	Switching operation modes failed.
FSP_ERR_CAN_INIT_FAILED	Hardware initialization failed.
FSP_ERR_CAN_TRANSMIT_NOT_READY	Transmit in progress.
FSP_ERR_CAN_RECEIVE_MAILBOX	Mailbox is setup as a receive mailbox.
FSP_ERR_CAN_TRANSMIT_MAILBOX	Mailbox is setup as a transmit mailbox.
FSP_ERR_CAN_MESSAGE_LOST	Receive message has been overwritten or overrun.
FSP_ERR_CAN_TRANSMIT_FIFO_FULL	Transmit FIFO is full.
FSP_ERR_WIFI_CONFIG_FAILED	WiFi module Configuration failed.
FSP_ERR_WIFI_INIT_FAILED	WiFi module initialization failed.
FSP_ERR_WIFI_TRANSMIT_FAILED	Transmission failed.
FSP_ERR_WIFI_INVALID_MODE	API called when provisioned in client mode.
FSP_ERR_WIFI_FAILED	WiFi Failed.
FSP_ERR_WIFI_SCAN_COMPLETE	Wifi scan has completed.
FSP_ERR_WIFI_AP_NOT_CONNECTED	WiFi module is not connected to access point.
FSP_ERR_CELLULAR_CONFIG_FAILED	Cellular module Configuration failed.
FSP_ERR_CELLULAR_INIT_FAILED	Cellular module initialization failed.
FSP_ERR_CELLULAR_TRANSMIT_FAILED	Transmission failed.
FSP_ERR_CELLULAR_FW_UPTODATE	Firmware is uptodate.
FSP_ERR_CELLULAR_FW_UPGRADE_FAILED	Firmware upgrade failed.
FSP_ERR_CELLULAR_FAILED	Cellular Failed.
FSP_ERR_CELLULAR_INVALID_STATE	API Called in invalid state.
FSP_ERR_CELLULAR_REGISTRATION_FAILED	Cellular Network registration failed.
FSP_ERR_BLE_FAILED	BLE operation failed.
FSP_ERR_BLE_INIT_FAILED	BLE device initialization failed.

FSP_ERR_BLE_CONFIG_FAILED	BLE device configuration failed.
FSP_ERR_BLE_PRF_ALREADY_ENABLED	BLE device Profile already enabled.
FSP_ERR_BLE_PRF_NOT_ENABLED	BLE device not enabled.
FSP_ERR_BLE_ABS_INVALID_OPERATION	Invalid operation is executed.
FSP_ERR_BLE_ABS_NOT_FOUND	Valid data or free space is not found.
FSP_ERR_CRYPTO_CONTINUE	Continue executing function.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	Hardware resource busy.
FSP_ERR_CRYPTO_SCE_FAIL	Internal I/O buffer is not empty.
FSP_ERR_CRYPTO_SCE_HRK_INVALID_INDEX	Invalid index.
FSP_ERR_CRYPTO_SCE_RETRY	Retry.
FSP_ERR_CRYPTO_SCE_VERIFY_FAIL	Verify is failed.
FSP_ERR_CRYPTO_SCE_ALREADY_OPEN	HW SCE module is already opened.
FSP_ERR_CRYPTO_NOT_OPEN	Hardware module is not initialized.
FSP_ERR_CRYPTO_UNKNOWN	Some unknown error occurred.
FSP_ERR_CRYPTO_NULL_POINTER	Null pointer input as a parameter.
FSP_ERR_CRYPTO_NOT_IMPLEMENTED	Algorithm/size not implemented.
FSP_ERR_CRYPTO_RNG_INVALID_PARAM	An invalid parameter is specified.
FSP_ERR_CRYPTO_RNG_FATAL_ERROR	A fatal error occurred.
FSP_ERR_CRYPTO_INVALID_SIZE	Size specified is invalid.
FSP_ERR_CRYPTO_INVALID_STATE	Function used in an valid state.
FSP_ERR_CRYPTO_ALREADY_OPEN	control block is already opened
FSP_ERR_CRYPTO_INSTALL_KEY_FAILED	Specified input key is invalid.
FSP_ERR_CRYPTO_AUTHENTICATION_FAILED	Authentication failed.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Failure to Init Cipher.

FSP_ERR_CRYPTO_SCE_AUTHENTICATION	Authentication failed.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input date is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function call occurred.
FSP_ERR_CRYPTO_COMMON_NOT_OPENED	Crypto Framework Common is not opened.
FSP_ERR_CRYPTO_HAL_ERROR	Cryoto HAL module returned an error.
FSP_ERR_CRYPTO_KEY_BUF_NOT_ENOUGH	Key buffer size is not enough to generate a key.
FSP_ERR_CRYPTO_BUF_OVERFLOW	Attempt to write data larger than what the buffer can hold.
FSP_ERR_CRYPTO_INVALID_OPERATION_MODE	Invalid operation mode.
FSP_ERR_MESSAGE_TOO_LONG	Message for RSA encryption is too long.
FSP_ERR_RSA_DECRYPTION_ERROR	RSA Decryption error.
FSP_ERR_SENSOR_INVALID_DATA	Data is invalid. *Note* *SF_CRYPTO APIs may return an error code starting from 0x10000 which is of Crypto module. Refer to sf_cryoto_err.h for Crypto error codes.*
FSP_ERR_SENSOR_IN_STABILIZATION	Sensor is stabilizing.
FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED	Measurement is not finished.
FSP_ERR_COMMS_BUS_NOT_OPEN	Bus is not open.

4.1.2 MCU Board Support Package

BSP

Functions

fsp_err_t	R_FSP_VersionGet (fsp_pack_version_t *const p_version)
void	Reset_Handler (void)
void	Default_Handler (void)

void	NMI_Handler (void)
void	SystemInit (void)
void	R_BSP_WarmStart (bsp_warm_start_event_t event)
uint32_t	R_BSP_SourceClockHzGet (fsp_priv_source_clock_t clock)
__STATIC_INLINE IRQn_Type	R_FSP_CurrentIrqGet (void)
__STATIC_INLINE uint32_t	R_FSP_SystemClockHzGet (fsp_priv_clock_t clock)
__STATIC_INLINE bsp_unique_id_t const *	R_BSP_UniqueIdGet ()
__STATIC_INLINE void	R_BSP_FlashCacheDisable ()
__STATIC_INLINE void	R_BSP_FlashCacheEnable ()
void	R_BSP_SoftwareDelay (uint32_t delay, bsp_delay_units_t units)
fsp_err_t	R_BSP_GroupIrqWrite (bsp_grp_irq_t irq, void(*p_callback)(bsp_grp_irq_t irq))
void	R_BSP_RegisterProtectEnable (bsp_reg_protect_t regs_to_protect)
void	R_BSP_RegisterProtectDisable (bsp_reg_protect_t regs_to_unprotect)
BSP_TFU_INLINE float	__sinf (float angle)
BSP_TFU_INLINE float	__cosf (float angle)
BSP_TFU_INLINE void	__sincosf (float angle, float *sin, float *cos)
BSP_TFU_INLINE float	__atan2f (float y_cord, float x_cord)
BSP_TFU_INLINE float	__hypotf (float x_cord, float y_cord)
BSP_TFU_INLINE void	__atan2hypotf (float y_cord, float x_cord, float *atan2, float *hypot)

Detailed Description

The BSP is responsible for getting the MCU from reset to the user's application. Before reaching the user's application, the BSP sets up the stacks, heap, clocks, interrupts, C runtime environment, and stack monitor.

- BSP Features

Overview

BSP Features

BSP Clock Configuration

All system clocks are set up during BSP initialization based on the settings in bsp_clock_cfg.h. These settings are derived from clock configuration information provided from the RA Configuration editor **Clocks** tab.

- Clock configuration is performed prior to initializing the C runtime environment to speed up the startup process, as it is possible to start up on a relatively slow (that is, 32 kHz) clock.
- The BSP implements the required delays to allow the selected clock to stabilize.
- The BSP will configure the CMSIS SystemCoreClock variable after clock initialization with the current system clock frequency.

System Interrupts

As RA MCUs are based on the Cortex-M ARM architecture, the NVIC Nested Vectored Interrupt Controller (NVIC) handles exceptions and interrupt configuration, prioritization and interrupt masking. In the ARM architecture, the NVIC handles exceptions. Some exceptions are known as System Exceptions. System exceptions are statically located at the "top" of the vector table and occupy vector numbers 1 to 15. Vector zero is reserved for the MSP Main Stack Pointer (MSP). The remaining 15 system exceptions are shown below:

- Reset
- NMI
- Cortex-M4 Hard Fault Handler
- Cortex-M4 MPU Fault Handler
- Cortex-M4 Bus Fault Handler
- Cortex-M4 Usage Fault Handler
- Reserved
- Reserved
- Reserved
- Reserved
- Cortex-M4 SVCall Handler
- Cortex-M4 Debug Monitor Handler
- Reserved

- Cortex-M4 PendSV Handler
- Cortex-M4 SysTick Handler

NMI and Hard Fault exceptions are enabled out of reset and have fixed priorities. Other exceptions have configurable priorities and some can be disabled.

Group Interrupts

Group interrupt is the term used to describe the 12 sources that can trigger the Non-Maskable Interrupt (NMI). When an NMI occurs the NMI Handler examines the NMISR (status register) to determine the source of the interrupt. NMI interrupts take precedence over all interrupts, are usable only as CPU interrupts, and cannot activate the RA peripherals Data Transfer Controller (DTC) or Direct Memory Access Controller (DMAC).

Possible group interrupt sources include:

- IWDT Underflow/Refresh Error
- WDT Underflow/Refresh Error
- Voltage-Monitoring 1 Interrupt
- Voltage-Monitoring 2 Interrupt
- VBATT monitor Interrupt
- Oscillation Stop is detected
- NMI pin
- RAM Parity Error
- RAM ECC Error
- MPU Bus Slave Error
- MPU Bus Master Error
- MPU Stack Error
- TrustZone Filter Error A user may enable notification for one or more group interrupts by registering a callback using the BSP API function R_BSP_GroupIrqWrite(). When an NMI interrupt occurs, the NMI handler checks to see if there is a callback registered for the cause of the interrupt and if so calls the registered callback function.

External and Peripheral Interrupts

User configurable interrupts begin with slot 16. These may be external, or peripheral generated interrupts.

Although the number of available slots for the NVIC interrupt vector table may seem small, the BSP defines up to 512 events that are capable of generating an interrupt. By using Event Mapping, the BSP maps user-enabled events to NVIC interrupts. For an RA6M3 MCU, only 96 of these events may be active at any one time, but the user has flexibility by choosing which events generate the active event.

By allowing the user to select only the events they are interested in as interrupt sources, we are able to provide an interrupt service routine that is fast and event specific.

For example, on other microcontrollers a standard NVIC interrupt vector table might contain a single vector entry for the SCI0 (Serial Communications Interface) peripheral. The interrupt service routine for this would have to check a status register for the 'real' source of the interrupt. In the RA implementation there is a vector entry for each of the SCI0 events that we are interested in.

BSP Weak Symbols

You might wonder how the BSP is able to place ISR addresses in the NVIC table without the user having explicitly defined one. All that is required by the BSP is that the interrupt event be given a priority.

This is accomplished through the use of the 'weak' attribute. The weak attribute causes the declaration to be emitted as a weak symbol rather than a global. A weak symbol is one that can be overridden by an accompanying strong reference with the same name. When the BSP declares a function as weak, user code can define the same function and it will be used in place of the BSP function. By defining all possible interrupt sources as weak, the vector table can be built at compile time and any user declarations (strong references) will be used at runtime.

Weak symbols are supported for ELF targets and also for a.out targets when using the GNU assembler and linker.

Note that in CMSIS system.c, there is also a weak definition (and a function body) for the Warm Start callback function R_BSP_WarmStart(). Because this function is defined in the same file as the weak declaration, it will be called as the 'default' implementation. The function may be overridden by the user by copying the body into their user application and modifying it as necessary. The linker identifies this as the 'strong' reference and uses it.

Warm Start Callbacks

As the BSP is in the process of bringing up the board out of reset, there are three points where the user can request a callback. These are defined as the 'Pre Clock Init', 'Post Clock Init' and 'Post C' warm start callbacks.

As described above, this function is already weakly defined as R_BSP_WarmStart(), so it is a simple matter of redefining the function or copying the existing body from CMSIS system.c into the application code to get a callback. R_BSP_WarmStart() takes an event parameter of type bsp_warm_start_event_t which describes the type of warm start callback being made.

This function is not enabled/disabled and is always called for both events as part of the BSP startup. Therefore it needs a function body, which will not be called if the user is overriding it. The function body is located in system.c. To use this function just copy this function into your own code and modify it to meet your needs.

C Runtime Initialization

This BSP configuration allows the user to skip the FSP C runtime initialization code by setting the "C Runtime Initialization" to "Disabled" on the BSP tab of the RA Configuration editor. Disabling this option is useful in cases where a non-standard linker script is being used or other modifications to the runtime initialization are desired. If this macro is disabled, the user must use the 'Post Clock Init' event from the warm start (described above) to run their own runtime initialization code.

Heap Allocation

The relatively low amount of on-chip SRAM available and lack of memory protection in an MCU means that heap use must be very carefully controlled to avoid memory leaks, overruns and attempted overallocation. Further, many RTOSes provide their own dynamic memory allocation system. For these reasons the default heap size is set at 0 bytes, effectively disabling dynamic memory. If it is required for an application setting a positive value to the "Heap size (bytes)" option in the RA Common configurations on the **BSP** tab will allocate a heap.

Note

> *When using printf/sprintf (and other variants) to output floating point numbers a heap is required. A minimum size of 0x1000 (4096) bytes is recommended when starting development in this case.*

Error Logging

When error logging is enabled, the error logging function can be redefined on the command line by defining FSP_ERROR_LOG(err) to the desired function call. The default function implementation is FSP_ERROR_LOG(err)=fsp_error_log(err, **FILE**, **LINE**). This implementation uses the predefined macros **FILE** and **LINE** to help identify the location where the error occurred. Removing the line from the function call can reduce code size when error logging is enabled. Some compilers may support other predefined macros like **FUNCTION**, which could be helpful for customizing the error logger.

Register Protection

The BSP register protection functions utilize reference counters to ensure that an application which has specified a certain register and subsequently calls another function doesn't have its register protection settings inadvertently modified.

Each time R_BSP_RegisterProtectDisable() is called, the respective reference counter is incremented.

Each time R_BSP_RegisterProtectEnable() is called, the respective reference counter is decremented.

Both functions will only modify the protection state if their reference counter is zero.

```
/* Enable writing to protected CGC registers */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_CGC);
/* Insert code to modify protected CGC registers. */
/* Disable writing to protected CGC registers */
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_CGC);
```

Option-setting memory

Option-setting memory includes OFS registers OFS0 and OFS1, OSIS debugger ID code, and block protections settings BPS and PBPS. Option-setting memory is MCU specific, and not all MCUs implement all option-setting registers. Option-setting configurations available on the selected device are configurable in the BSP properties. These configurations are placed in sections to be loaded at the required flash address by the linker.

The ID code is a 16-byte value that can be used to protect the MCU from being connected to a debugger or from connecting in Serial Boot Mode. There are different settings that can be set for the ID code; please refer to the hardware manual for your device for available options.

On MCUs that support TrustZone, option-setting registers are placed in a different locations for Non-Secure projects than for Secure or Flat projects. This is handled automatically by the BSP and linker scripts.

All *_SEL registers default to allowing both Secure and Non-Secure access unless otherwise noted here. If block protection is configured in a Secure project, the BSP sets the corresponding configuration to Secure access only by updating the corresponding *_SEL register. Similarly, the LVD related settings in the OFSn_SEL registers are automatically set to Secure if the corresponding LVD monitor is used in the Secure project.

TrustZone Security Attribution Registers

On MCUs that support TrustZone, Security Attribution Registers for modules used in the Secure project are configured to allow Secure access only as part of the startup code of the Secure project. This logic is skipped for Flat projects.

Software Delay

Implements a blocking software delay. A delay can be specified in microseconds, milliseconds or seconds. The delay is implemented based on the system clock rate.

```
 /* Delay at least 1 second. Depending on the number of wait states required for the
region of memory
  * that the software_delay_loop has been linked in this could take longer. The
default is 4 cycles per loop.
  * This can be modified by redefining DELAY_LOOP_CYCLES. BSP_DELAY_UNITS_SECONDS,
BSP_DELAY_UNITS_MILLISECONDS,
  * and BSP_DELAY_UNITS_MICROSECONDS can all be used with R_BSP_SoftwareDelay. */
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
```

Trignometric Function

Implements Trignometric math inline functions utilizing TFU hardware. These functions can calculate sine, cosine, arctangent and hypotenuse. The trigonometric library functions sinf(), cosf(), atan2f(), and hypotf() can be mapped to respective TFU functions by enabling TFU Mathlib property in FSP Configuration tool. Extended functions sincosf() and atan2hypotf() are also available when the TFU Mathlib property is enabled in the RA Configuration editor.

TFU functions are not reentrant. Disable the TFU Mathlib property in RA Configuration editor if reentrant access to trigonometric library functions is required.

Note

Refer to the MCU hardware user's manual or datasheet to determine if it has TFU support.

Critical Section Macros

Implements a critical section. Some MCUs (MCUs with the BASEPRI register) support allowing high priority interrupts to execute during critical sections. On these MCUs, interrupts with priority less than or equal to BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION are not serviced in critical sections. Interrupts with higher priority than BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION still execute in critical sections.

```
    FSP_CRITICAL_SECTION_DEFINE;
 /* Store the current interrupt posture. */
 FSP_CRITICAL_SECTION_ENTER;
 /* Interrupts cannot run in this section unless their priority is less than
BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION. */
```

```
/* Restore saved interrupt posture. */
FSP_CRITICAL_SECTION_EXIT;
```

OctaClock Update

Supports changing the Octal-SPI Clock (OCTACLK) during runtime if supported by the MCU. The OCTACLK source and clock divisor can be updated. It is user's responsibility to ensure the selected clock source is running before attempting to update OCTACLK.

Sealing the Main Stack (TrustZone Secure Projects)

In TrustZone secure projects, the BSP seals the main stack by placing the value 0xFEF5EDA5 above the stack top. For more information, refer to section 3.5 "Sealing a Stack" in "Secure software guidelines for ARMv8-M": https://developer.arm.com/documentation/100720/0300.

Board Specific Features

The BSP will call the board's initialization function (bsp_init) which can initialize board specific features. Possible board features are listed below.

Board Feature	Description
SDRAM Support	The BSP will initialize SDRAM if the board supports it
QSPI Support	The BSP will initialize QSPI if the board supports it and put it into ROM mode. Use the R_QSPI module to write and erase the QSPI chip.

Configuration

The BSP is heavily data driven with most features and functionality being configured based on the content from configuration files. Configuration files represent the settings specified by the user and are generated when the project is built and/or when the Generate Project Content button is clicked in the RA Configuration editor.

Build Time Configurations for fsp_common

The following build time configurations are defined in fsp_cfg/bsp/bsp_cfg.h:

Configuration	Options	Default	Description
Main stack size (bytes)	Value must be an integer multiple of 8 and between 8 and 0xFFFFFFFF	0x400	Set the size of the main program stack. NOTE: This entry is for the main stack. When using an RTOS, thread stacks can be

			configured in the properties for each thread.
Heap size (bytes)	Value must be 0 or an integer multiple of 8 between 8 and 0xFFFFFFFF.	0	The main heap is disabled by default. Set the heap size to a positive integer divisible by 8 to enable it. A minimum of 4K (0x1000) is recommended if standard library functions are to be used.
MCU Vcc (mV)	Value must between 0 and 5500 (5.5V)	3300	Some peripherals require different settings based on the supplied voltage. Entering Vcc here (in mV) allows the relevant driver modules to configure the associated peripherals accordingly.
Parameter checking	• Enabled • Disabled	Disabled	When enabled, parameter checking for the BSP is turned on. In addition, any modules whose parameter checking configuration is set to 'Default (BSP)' will perform parameter checking as well.
Assert Failures	• Return FSP_ERR_ASSERTION • Call fsp_error_log then Return FSP_ERR_ASSERTION • Use assert() to Halt Execution • Disable checks that would return FSP_ERR_ASSERTION	Return FSP_ERR_ASSERTION	Define the behavior of the FSP_ASSERT() macro.
Error Log	• No Error Log • Errors Logged via fsp_error_log	No Error Log	Specify error logging behavior.

Clock Registers not Reset Values during Startup	• Disabled • Enabled	Disabled	If enabled, registers are assumed to be set to their reset value during startup. Enable this if another application such as a bootloader or Secure project has already configured the clocks before the startup code runs.
Main Oscillator Populated	• Populated • Not Populated	Populated	Select whether or not there is a main oscillator (XTAL) on the board. This setting can be overridden in board_cfg.h.
PFS Protect	• Disabled • Enabled	Enabled	Keep the PFS registers locked when they are not being modified. If disabled they will be unlocked during startup.
C Runtime Initialization	• Enabled • Disabled	Enabled	Select if the C runtime initialization in the BSP is to be used. If disabled, use the BSP_WARM_START_POST_CLOCK event to run user defined equivalent.
Early BSP Initialization	• Enabled • Disabled	Disabled	Enable this option to use BSP functions before C runtime initialization (BSP_WARM_START_RESET or BSP_WARM_START_POST_CLOCK).
Main Oscillator Clock Source	• External Oscillator • Crystal or Resonator	Crystal or Resonator	Select the main oscillator clock source. This setting can be overridden in board_cfg.h
Subclock Populated	• Populated • Not Populated	Populated	Select whether or not there is a subclock crystal on the board. This setting can be overridden in board_cfg.h.
Subclock Drive (Drive capacitance availability varies by MCU)	• Standard/Normal mode • Low/Low power	Standard/Normal mode	Select the subclock oscillator drive capacitance. This

	mode 1 • Low power mode 2 • Low power mode 3		setting can be overridden in board_cfg.h
Subclock Stabilization Time (ms)	Value must between 0 and 10000	1000	Select the subclock oscillator stabilization time. This is only used in the startup code if the subclock is selected as the system clock on the Clocks tab or if the HOCO FLL function is enabled. This setting can be overridden in board_cfg.h

Modules

	RA2A1
	RA2E1
	RA2E2
	RA2L1
	RA4E1
	RA4M1
	RA4M2
	RA4M3
	RA4W1
	RA6E1
	RA6M1
	RA6M2
	RA6M3
	RA6M4
	RA6M5
	RA6T1

	RA6T2

Macros

#define	BSP_IRQ_DISABLED
#define	FSP_LOG_PRINT(X)
#define	FSP_RETURN(err)
#define	FSP_ERROR_LOG(err)
#define	FSP_ASSERT(a)
#define	FSP_ERROR_RETURN(a, err)
#define	FSP_CRITICAL_SECTION_ENTER
#define	FSP_CRITICAL_SECTION_EXIT
#define	FSP_INVALID_VECTOR
#define	BSP_CFG_HANDLE_UNRECOVERABLE_ERROR(x)
#define	BSP_STACK_ALIGNMENT
#define	R_BSP_MODULE_START(ip, channel)
#define	R_BSP_MODULE_STOP(ip, channel)

Enumerations

enum	fsp_ip_t
enum	fsp_signal_t
enum	bsp_warm_start_event_t
enum	fsp_priv_source_clock_t
enum	bsp_delay_units_t
enum	bsp_grp_irq_t
enum	bsp_reg_protect_t

Variables

uint32_t SystemCoreClock	BSP_SECTION_EARLY_INIT

Macro Definition Documentation

◆ BSP_IRQ_DISABLED

#define BSP_IRQ_DISABLED
Used to signify that an ELC event is not able to be used as an interrupt.

◆ FSP_LOG_PRINT

#define FSP_LOG_PRINT (X)
Macro that can be defined in order to enable logging in FSP modules.

◆ FSP_RETURN

#define FSP_RETURN (err)
Macro to log and return error without an assertion.

◆ FSP_ERROR_LOG

#define FSP_ERROR_LOG (err)
This function is called before returning an error code. To stop on a runtime error, define fsp_error_log in user code and do required debugging (breakpoints, stack dump, etc) in this function.

◆ FSP_ASSERT

#define FSP_ASSERT (a)
Default assertion calls FSP_ERROR_RETURN if condition "a" is false. Used to identify incorrect use of API's in FSP functions.

◆ FSP_ERROR_RETURN

#define FSP_ERROR_RETURN (a, err)
All FSP error codes are returned using this macro. Calls FSP_ERROR_LOG function if condition "a" is false. Used to identify runtime errors in FSP functions.

◆ FSP_CRITICAL_SECTION_ENTER

#define FSP_CRITICAL_SECTION_ENTER
This macro temporarily saves the current interrupt state and disables interrupts.

◆ FSP_CRITICAL_SECTION_EXIT

#define FSP_CRITICAL_SECTION_EXIT
This macro restores the previously saved interrupt state, reenabling interrupts.

◆ FSP_INVALID_VECTOR

#define FSP_INVALID_VECTOR
Used to signify that the requested IRQ vector is not defined in this system.

◆ BSP_CFG_HANDLE_UNRECOVERABLE_ERROR

#define BSP_CFG_HANDLE_UNRECOVERABLE_ERROR (x)
In the event of an unrecoverable error the BSP will by default call the __BKPT() intrinsic function which will alert the user of the error. The user can override this default behavior by defining their own BSP_CFG_HANDLE_UNRECOVERABLE_ERROR macro.

◆ BSP_STACK_ALIGNMENT

#define BSP_STACK_ALIGNMENT
Stacks (and heap) must be sized and aligned to an integer multiple of this number.

◆ R_BSP_MODULE_START

#define R_BSP_MODULE_START (ip, channel)

Cancels the module stop state.

Parameters

ip	fsp_ip_t enum value for the module to be stopped
channel	The channel. Use channel 0 for modules without channels.

◆ R_BSP_MODULE_STOP

```
#define R_BSP_MODULE_STOP ( ip, channel )
```

Enables the module stop state.

Parameters

ip	fsp_ip_t enum value for the module to be stopped
channel	The channel. Use channel 0 for modules without channels.

Enumeration Type Documentation

◆ fsp_ip_t

enum fsp_ip_t

Available modules.

Enumerator	
FSP_IP_CFLASH	Code Flash.
FSP_IP_DFLASH	Data Flash.
FSP_IP_RAM	RAM.
FSP_IP_LVD	Low Voltage Detection.
FSP_IP_CGC	Clock Generation Circuit.
FSP_IP_LPM	Low Power Modes.
FSP_IP_FCU	Flash Control Unit.
FSP_IP_ICU	Interrupt Control Unit.
FSP_IP_DMAC	DMA Controller.
FSP_IP_DTC	Data Transfer Controller.
FSP_IP_IOPORT	I/O Ports.
FSP_IP_PFS	Pin Function Select.
FSP_IP_ELC	Event Link Controller.
FSP_IP_MPU	Memory Protection Unit.
FSP_IP_MSTP	Module Stop.
FSP_IP_MMF	Memory Mirror Function.
FSP_IP_KEY	Key Interrupt Function.
FSP_IP_CAC	Clock Frequency Accuracy Measurement Circuit.
FSP_IP_DOC	Data Operation Circuit.
FSP_IP_CRC	Cyclic Redundancy Check Calculator.
FSP_IP_SCI	Serial Communications Interface.

FSP_IP_IIC	I2C Bus Interface.
FSP_IP_SPI	Serial Peripheral Interface.
FSP_IP_CTSU	Capacitive Touch Sensing Unit.
FSP_IP_SCE	Secure Cryptographic Engine.
FSP_IP_SLCDC	Segment LCD Controller.
FSP_IP_AES	Advanced Encryption Standard.
FSP_IP_TRNG	True Random Number Generator.
FSP_IP_FCACHE	Flash Cache.
FSP_IP_SRAM	SRAM.
FSP_IP_ADC	A/D Converter.
FSP_IP_DAC	12-Bit D/A Converter
FSP_IP_TSN	Temperature Sensor.
FSP_IP_DAAD	D/A A/D Synchronous Unit.
FSP_IP_ACMPHS	High Speed Analog Comparator.
FSP_IP_ACMPLP	Low Power Analog Comparator.
FSP_IP_OPAMP	Operational Amplifier.
FSP_IP_SDADC	Sigma Delta A/D Converter.
FSP_IP_RTC	Real Time Clock.
FSP_IP_WDT	Watch Dog Timer.
FSP_IP_IWDT	Independent Watch Dog Timer.
FSP_IP_GPT	General PWM Timer.
FSP_IP_POEG	Port Output Enable for GPT.
FSP_IP_OPS	Output Phase Switch.
FSP_IP_AGT	Asynchronous General-Purpose Timer.

FSP_IP_CAN	Controller Area Network.
FSP_IP_IRDA	Infrared Data Association.
FSP_IP_QSPI	Quad Serial Peripheral Interface.
FSP_IP_USBFS	USB Full Speed.
FSP_IP_SDHI	SD/MMC Host Interface.
FSP_IP_SRC	Sampling Rate Converter.
FSP_IP_SSI	Serial Sound Interface.
FSP_IP_DALI	Digital Addressable Lighting Interface.
FSP_IP_ETHER	Ethernet MAC Controller.
FSP_IP_EDMAC	Ethernet DMA Controller.
FSP_IP_EPTPC	Ethernet PTP Controller.
FSP_IP_PDC	Parallel Data Capture Unit.
FSP_IP_GLCDC	Graphics LCD Controller.
FSP_IP_DRW	2D Drawing Engine
FSP_IP_JPEG	JPEG.
FSP_IP_DAC8	8-Bit D/A Converter
FSP_IP_USBHS	USB High Speed.
FSP_IP_OSPI	Octa Serial Peripheral Interface.
FSP_IP_CEC	HDMI CEC.
FSP_IP_TFU	Trigonometric Function Unit.
FSP_IP_IIRFA	IIR Filter Accelerator.
FSP_IP_CANFD	CAN-FD.

◆ fsp_signal_t

enum fsp_signal_t	
Signals that can be mapped to an interrupt.	
Enumerator	
FSP_SIGNAL_ADC_COMPARE_MATCH	ADC COMPARE MATCH.
FSP_SIGNAL_ADC_COMPARE_MISMATCH	ADC COMPARE MISMATCH.
FSP_SIGNAL_ADC_SCAN_END	ADC SCAN END.
FSP_SIGNAL_ADC_SCAN_END_B	ADC SCAN END B.
FSP_SIGNAL_ADC_WINDOW_A	ADC WINDOW A.
FSP_SIGNAL_ADC_WINDOW_B	ADC WINDOW B.
FSP_SIGNAL_AES_RDREQ	AES RDREQ.
FSP_SIGNAL_AES_WRREQ	AES WRREQ.
FSP_SIGNAL_AGT_COMPARE_A	AGT COMPARE A.
FSP_SIGNAL_AGT_COMPARE_B	AGT COMPARE B.
FSP_SIGNAL_AGT_INT	AGT INT.
FSP_SIGNAL_CAC_FREQUENCY_ERROR	CAC FREQUENCY ERROR.
FSP_SIGNAL_CAC_MEASUREMENT_END	CAC MEASUREMENT END.
FSP_SIGNAL_CAC_OVERFLOW	CAC OVERFLOW.
FSP_SIGNAL_CAN_ERROR	CAN ERROR.
FSP_SIGNAL_CAN_FIFO_RX	CAN FIFO RX.
FSP_SIGNAL_CAN_FIFO_TX	CAN FIFO TX.
FSP_SIGNAL_CAN_MAILBOX_RX	CAN MAILBOX RX.
FSP_SIGNAL_CAN_MAILBOX_TX	CAN MAILBOX TX.
FSP_SIGNAL_CGC_MOSC_STOP	CGC MOSC STOP.
FSP_SIGNAL_LPM_SNOOZE_REQUEST	LPM SNOOZE REQUEST.

FSP_SIGNAL_LVD_LVD1	LVD LVD1.
FSP_SIGNAL_LVD_LVD2	LVD LVD2.
FSP_SIGNAL_VBATT_LVD	VBATT LVD.
FSP_SIGNAL_LVD_VBATT	LVD VBATT.
FSP_SIGNAL_ACMPHS_INT	ACMPHS INT.
FSP_SIGNAL_ACMPLP_INT	ACMPLP INT.
FSP_SIGNAL_CTSU_END	CTSU END.
FSP_SIGNAL_CTSU_READ	CTSU READ.
FSP_SIGNAL_CTSU_WRITE	CTSU WRITE.
FSP_SIGNAL_DALI_DEI	DALI DEI.
FSP_SIGNAL_DALI_CLI	DALI CLI.
FSP_SIGNAL_DALI_SDI	DALI SDI.
FSP_SIGNAL_DALI_BPI	DALI BPI.
FSP_SIGNAL_DALI_FEI	DALI FEI.
FSP_SIGNAL_DALI_SDI_OR_BPI	DALI SDI OR BPI.
FSP_SIGNAL_DMAC_INT	DMAC INT.
FSP_SIGNAL_DOC_INT	DOC INT.
FSP_SIGNAL_DRW_INT	DRW INT.
FSP_SIGNAL_DTC_COMPLETE	DTC COMPLETE.
FSP_SIGNAL_DTC_END	DTC END.
FSP_SIGNAL_EDMAC_EINT	EDMAC EINT.
FSP_SIGNAL_ELC_SOFTWARE_EVENT_0	ELC SOFTWARE EVENT 0.
FSP_SIGNAL_ELC_SOFTWARE_EVENT_1	ELC SOFTWARE EVENT 1.
FSP_SIGNAL_EPTPC_IPLS	EPTPC IPLS.

FSP_SIGNAL_EPTPC_MINT	EPTPC MINT.
FSP_SIGNAL_EPTPC_PINT	EPTPC PINT.
FSP_SIGNAL_EPTPC_TIMER0_FALL	EPTPC TIMER0 FALL.
FSP_SIGNAL_EPTPC_TIMER0_RISE	EPTPC TIMER0 RISE.
FSP_SIGNAL_EPTPC_TIMER1_FALL	EPTPC TIMER1 FALL.
FSP_SIGNAL_EPTPC_TIMER1_RISE	EPTPC TIMER1 RISE.
FSP_SIGNAL_EPTPC_TIMER2_FALL	EPTPC TIMER2 FALL.
FSP_SIGNAL_EPTPC_TIMER2_RISE	EPTPC TIMER2 RISE.
FSP_SIGNAL_EPTPC_TIMER3_FALL	EPTPC TIMER3 FALL.
FSP_SIGNAL_EPTPC_TIMER3_RISE	EPTPC TIMER3 RISE.
FSP_SIGNAL_EPTPC_TIMER4_FALL	EPTPC TIMER4 FALL.
FSP_SIGNAL_EPTPC_TIMER4_RISE	EPTPC TIMER4 RISE.
FSP_SIGNAL_EPTPC_TIMER5_FALL	EPTPC TIMER5 FALL.
FSP_SIGNAL_EPTPC_TIMER5_RISE	EPTPC TIMER5 RISE.
FSP_SIGNAL_FCU_FIFERR	FCU FIFERR.
FSP_SIGNAL_FCU_FRDYI	FCU FRDYI.
FSP_SIGNAL_GLCDC_LINE_DETECT	GLCDC LINE DETECT.
FSP_SIGNAL_GLCDC_UNDERFLOW_1	GLCDC UNDERFLOW 1.
FSP_SIGNAL_GLCDC_UNDERFLOW_2	GLCDC UNDERFLOW 2.
FSP_SIGNAL_GPT_CAPTURE_COMPARE_A	GPT CAPTURE COMPARE A.
FSP_SIGNAL_GPT_CAPTURE_COMPARE_B	GPT CAPTURE COMPARE B.
FSP_SIGNAL_GPT_COMPARE_C	GPT COMPARE C.
FSP_SIGNAL_GPT_COMPARE_D	GPT COMPARE D.
FSP_SIGNAL_GPT_COMPARE_E	GPT COMPARE E.

FSP_SIGNAL_GPT_COMPARE_F	GPT COMPARE F.
FSP_SIGNAL_GPT_COUNTER_OVERFLOW	GPT COUNTER OVERFLOW.
FSP_SIGNAL_GPT_COUNTER_UNDERFLOW	GPT COUNTER UNDERFLOW.
FSP_SIGNAL_GPT_AD_TRIG_A	GPT AD TRIG A.
FSP_SIGNAL_GPT_AD_TRIG_B	GPT AD TRIG B.
FSP_SIGNAL_OPS_UVW_EDGE	OPS UVW EDGE.
FSP_SIGNAL_ICU_IRQ0	ICU IRQ0.
FSP_SIGNAL_ICU_IRQ1	ICU IRQ1.
FSP_SIGNAL_ICU_IRQ2	ICU IRQ2.
FSP_SIGNAL_ICU_IRQ3	ICU IRQ3.
FSP_SIGNAL_ICU_IRQ4	ICU IRQ4.
FSP_SIGNAL_ICU_IRQ5	ICU IRQ5.
FSP_SIGNAL_ICU_IRQ6	ICU IRQ6.
FSP_SIGNAL_ICU_IRQ7	ICU IRQ7.
FSP_SIGNAL_ICU_IRQ8	ICU IRQ8.
FSP_SIGNAL_ICU_IRQ9	ICU IRQ9.
FSP_SIGNAL_ICU_IRQ10	ICU IRQ10.
FSP_SIGNAL_ICU_IRQ11	ICU IRQ11.
FSP_SIGNAL_ICU_IRQ12	ICU IRQ12.
FSP_SIGNAL_ICU_IRQ13	ICU IRQ13.
FSP_SIGNAL_ICU_IRQ14	ICU IRQ14.
FSP_SIGNAL_ICU_IRQ15	ICU IRQ15.
FSP_SIGNAL_ICU_SNOOZE_CANCEL	ICU SNOOZE CANCEL.
FSP_SIGNAL_IIC_ERI	IIC ERI.

FSP_SIGNAL_IIC_RXI	IIC RXI.
FSP_SIGNAL_IIC_TEI	IIC TEI.
FSP_SIGNAL_IIC_TXI	IIC TXI.
FSP_SIGNAL_IIC_WUI	IIC WUI.
FSP_SIGNAL_IOPORT_EVENT_1	IOPORT EVENT 1.
FSP_SIGNAL_IOPORT_EVENT_2	IOPORT EVENT 2.
FSP_SIGNAL_IOPORT_EVENT_3	IOPORT EVENT 3.
FSP_SIGNAL_IOPORT_EVENT_4	IOPORT EVENT 4.
FSP_SIGNAL_IOPORT_EVENT_B	IOPORT EVENT B.
FSP_SIGNAL_IOPORT_EVENT_C	IOPORT EVENT C.
FSP_SIGNAL_IOPORT_EVENT_D	IOPORT EVENT D.
FSP_SIGNAL_IOPORT_EVENT_E	IOPORT EVENT E.
FSP_SIGNAL_IWDT_UNDERFLOW	IWDT UNDERFLOW.
FSP_SIGNAL_JPEG_JDTI	JPEG JDTI.
FSP_SIGNAL_JPEG_JEDI	JPEG JEDI.
FSP_SIGNAL_KEY_INT	KEY INT.
FSP_SIGNAL_PDC_FRAME_END	PDC FRAME END.
FSP_SIGNAL_PDC_INT	PDC INT.
FSP_SIGNAL_PDC_RECEIVE_DATA_READY	PDC RECEIVE DATA READY.
FSP_SIGNAL_POEG_EVENT	POEG EVENT.
FSP_SIGNAL_QSPI_INT	QSPI INT.
FSP_SIGNAL_RTC_ALARM	RTC ALARM.
FSP_SIGNAL_RTC_PERIOD	RTC PERIOD.
FSP_SIGNAL_RTC_CARRY	RTC CARRY.

FSP_SIGNAL_SCE_INTEGRATE_RDRDY	SCE INTEGRATE RDRDY.
FSP_SIGNAL_SCE_INTEGRATE_WRRDY	SCE INTEGRATE WRRDY.
FSP_SIGNAL_SCE_LONG_PLG	SCE LONG PLG.
FSP_SIGNAL_SCE_PROC_BUSY	SCE PROC BUSY.
FSP_SIGNAL_SCE_RDRDY_0	SCE RDRDY 0.
FSP_SIGNAL_SCE_RDRDY_1	SCE RDRDY 1.
FSP_SIGNAL_SCE_ROMOK	SCE ROMOK.
FSP_SIGNAL_SCE_TEST_BUSY	SCE TEST BUSY.
FSP_SIGNAL_SCE_WRRDY_0	SCE WRRDY 0.
FSP_SIGNAL_SCE_WRRDY_1	SCE WRRDY 1.
FSP_SIGNAL_SCE_WRRDY_4	SCE WRRDY 4.
FSP_SIGNAL_SCI_AM	SCI AM.
FSP_SIGNAL_SCI_ERI	SCI ERI.
FSP_SIGNAL_SCI_RXI	SCI RXI.
FSP_SIGNAL_SCI_RXI_OR_ERI	SCI RXI OR ERI.
FSP_SIGNAL_SCI_TEI	SCI TEI.
FSP_SIGNAL_SCI_TXI	SCI TXI.
FSP_SIGNAL_SDADC_ADI	SDADC ADI.
FSP_SIGNAL_SDADC_SCANEND	SDADC SCANEND.
FSP_SIGNAL_SDADC_CALIEND	SDADC CALIEND.
FSP_SIGNAL_SDHIMMC_ACCS	SDHIMMC ACCS.
FSP_SIGNAL_SDHIMMC_CARD	SDHIMMC CARD.
FSP_SIGNAL_SDHIMMC_DMA_REQ	SDHIMMC DMA REQ.
FSP_SIGNAL_SDHIMMC_SDIO	SDHIMMC SDIO.

FSP_SIGNAL_SPI_ERI	SPI ERI.
FSP_SIGNAL_SPI_IDLE	SPI IDLE.
FSP_SIGNAL_SPI_RXI	SPI RXI.
FSP_SIGNAL_SPI_TEI	SPI TEI.
FSP_SIGNAL_SPI_TXI	SPI TXI.
FSP_SIGNAL_SRC_CONVERSION_END	SRC CONVERSION END.
FSP_SIGNAL_SRC_INPUT_FIFO_EMPTY	SRC INPUT FIFO EMPTY.
FSP_SIGNAL_SRC_OUTPUT_FIFO_FULL	SRC OUTPUT FIFO FULL.
FSP_SIGNAL_SRC_OUTPUT_FIFO_OVERFLOW	SRC OUTPUT FIFO OVERFLOW.
FSP_SIGNAL_SRC_OUTPUT_FIFO_UNDERFLOW	SRC OUTPUT FIFO UNDERFLOW.
FSP_SIGNAL_SSI_INT	SSI INT.
FSP_SIGNAL_SSI_RXI	SSI RXI.
FSP_SIGNAL_SSI_TXI	SSI TXI.
FSP_SIGNAL_SSI_TXI_RXI	SSI TXI RXI.
FSP_SIGNAL_TRNG_RDREQ	TRNG RDREQ.
FSP_SIGNAL_USB_FIFO_0	USB FIFO 0.
FSP_SIGNAL_USB_FIFO_1	USB FIFO 1.
FSP_SIGNAL_USB_INT	USB INT.
FSP_SIGNAL_USB_RESUME	USB RESUME.
FSP_SIGNAL_USB_USB_INT_RESUME	USB USB INT RESUME.
FSP_SIGNAL_WDT_UNDERFLOW	WDT UNDERFLOW.

◆ bsp_warm_start_event_t

enum bsp_warm_start_event_t	
Different warm start entry locations in the BSP.	
Enumerator	
BSP_WARM_START_RESET	Called almost immediately after reset. No C runtime environment, clocks, or IRQs.
BSP_WARM_START_POST_CLOCK	Called after clock initialization. No C runtime environment or IRQs.
BSP_WARM_START_POST_C	Called after clocks and C runtime environment have been set up.

◆ fsp_priv_source_clock_t

enum fsp_priv_source_clock_t	
Enumerator	
FSP_PRIV_CLOCK_HOCO	The high speed on chip oscillator.
FSP_PRIV_CLOCK_MOCO	The middle speed on chip oscillator.
FSP_PRIV_CLOCK_LOCO	The low speed on chip oscillator.
FSP_PRIV_CLOCK_MAIN_OSC	The main oscillator.
FSP_PRIV_CLOCK_SUBCLOCK	The subclock oscillator.
FSP_PRIV_CLOCK_PLL	The PLL oscillator.
FSP_PRIV_CLOCK_PLL2	The PLL2 oscillator.

◆ bsp_delay_units_t

enum bsp_delay_units_t	
Available delay units for R_BSP_SoftwareDelay(). These are ultimately used to calculate a total # of microseconds	
Enumerator	
BSP_DELAY_UNITS_SECONDS	Requested delay amount is in seconds.
BSP_DELAY_UNITS_MILLISECONDS	Requested delay amount is in milliseconds.
BSP_DELAY_UNITS_MICROSECONDS	Requested delay amount is in microseconds.

◆ bsp_grp_irq_t

enum bsp_grp_irq_t	
Which interrupts can have callbacks registered.	
Enumerator	
BSP_GRP_IRQ_IWDT_ERROR	IWDT underflow/refresh error has occurred.
BSP_GRP_IRQ_WDT_ERROR	WDT underflow/refresh error has occurred.
BSP_GRP_IRQ_LVD1	Voltage monitoring 1 interrupt.
BSP_GRP_IRQ_LVD2	Voltage monitoring 2 interrupt.
BSP_GRP_IRQ_VBATT	VBATT monitor interrupt.
BSP_GRP_IRQ_OSC_STOP_DETECT	Oscillation stop is detected.
BSP_GRP_IRQ_NMI_PIN	NMI Pin interrupt.
BSP_GRP_IRQ_RAM_PARITY	RAM Parity Error.
BSP_GRP_IRQ_RAM_ECC	RAM ECC Error.
BSP_GRP_IRQ_MPU_BUS_SLAVE	MPU Bus Slave Error.
BSP_GRP_IRQ_MPU_BUS_MASTER	MPU Bus Master Error.
BSP_GRP_IRQ_MPU_STACK	MPU Stack Error.
BSP_GRP_IRQ_TRUSTZONE	MPU Stack Error.
BSP_GRP_IRQ_CACHE_PARITY	MPU Stack Error.

◆ bsp_reg_protect_t

enum bsp_reg_protect_t

The different types of registers that can be protected.

Enumerator	
BSP_REG_PROTECT_CGC	Enables writing to the registers related to the clock generation circuit.
BSP_REG_PROTECT_OM_LPC_BATT	Enables writing to the registers related to operating modes, low power consumption, and battery backup function.
BSP_REG_PROTECT_LVD	Enables writing to the registers related to the LVD: LVCMPCR, LVDLVLR, LVD1CR0, LVD1CR1, LVD1SR, LVD2CR0, LVD2CR1, LVD2SR.
BSP_REG_PROTECT_SAR	Enables writing to the registers related to the security function.

Function Documentation

◆ R_FSP_VersionGet()

fsp_err_t R_FSP_VersionGet (fsp_pack_version_t *const *p_version*)

Get the FSP version based on compile time macros.

Parameters

[out]	p_version	Memory address to return version information to.

Return values

FSP_SUCCESS	Version information stored.
FSP_ERR_ASSERTION	The parameter p_version is NULL.

◆ Reset_Handler()

void Reset_Handler (void)

MCU starts executing here out of reset. Main stack pointer is set up already.

◆ Default_Handler()

void Default_Handler (void)
Default exception handler.

◆ NMI_Handler()

void NMI_Handler (void)
Non-maskable interrupt handler. This exception is defined by the BSP, unlike other system exceptions, because there are many sources that map to the NMI exception.

◆ SystemInit()

void SystemInit (void)
Initialize the MCU and the runtime environment.

◆ R_BSP_WarmStart()

void R_BSP_WarmStart (bsp_warm_start_event_t *event*)

This function is called at various points during the startup process. This function is declared as a weak symbol higher up in this file because it is meant to be overridden by a user implemented version. One of the main uses for this function is to call functional safety code during the startup process. To use this function just copy this function into your own code and modify it to meet your needs.

Parameters

[in]	event	Where the code currently is in the start up process

◆ R_BSP_SourceClockHzGet()

uint32_t R_BSP_SourceClockHzGet (fsp_priv_source_clock_t *clock*)

Gets the frequency of a source clock.

Parameters

[in]	clock	Pointer to Octaclk setting structure which provides information regarding Octaclk source and divider settings to be applied.

Returns
Frequency of requested clock in Hertz.

◆ R_FSP_CurrentIrqGet()

__STATIC_INLINE IRQn_Type R_FSP_CurrentIrqGet (void)

Return active interrupt vector number value

Returns
Active interrupt vector number value

◆ R_FSP_SystemClockHzGet()

__STATIC_INLINE uint32_t R_FSP_SystemClockHzGet (fsp_priv_clock_t *clock*)

Gets the frequency of a system clock.

Returns
Frequency of requested clock in Hertz.

◆ R_BSP_UniqueIdGet()

__STATIC_INLINE bsp_unique_id_t const* R_BSP_UniqueIdGet ( )

Get unique ID for this device.

Returns
A pointer to the unique identifier structure

◆ R_BSP_FlashCacheDisable()

__STATIC_INLINE void R_BSP_FlashCacheDisable ( )

Disables the flash cache.

◆ R_BSP_FlashCacheEnable()

__STATIC_INLINE void R_BSP_FlashCacheEnable ( )
Enables the flash cache.

◆ R_BSP_SoftwareDelay()

void R_BSP_SoftwareDelay (uint32_t *delay*, bsp_delay_units_t *units*)

Delay for at least the specified duration in units and return.

Parameters

[in]	delay	The number of 'units' to delay.
[in]	units	The 'base' (bsp_delay_units_t) for the units specified. Valid values are: BSP_DELAY_UNITS_SECONDS, BSP_DELAY_UNITS_MILLISECONDS, BSP_DELAY_UNITS_MICROSECONDS. For example: At 1 MHz one cycle takes 1 microsecond (.000001 seconds). At 12 MHz one cycle takes 1/12 microsecond or 83 nanoseconds. Therefore one run through bsp_prv_software_delay_loop() takes: ~ (83 * BSP_DELAY_LOOP_CYCLES) or 332 ns. A delay of 2 us therefore requires 2000ns/332ns or 6 loops.

The 'theoretical' maximum delay that may be obtained is determined by a full 32 bit loop count and the system clock rate. @120MHz: ((0xFFFFFFFF loops * 4 cycles /loop) / 120000000) = 143 seconds. @32MHz: ((0xFFFFFFFF loops * 4 cycles /loop) / 32000000) = 536 seconds

Note that requests for very large delays will be affected by rounding in the calculations and the actual delay achieved may be slightly longer. @32 MHz, for example, a request for 532 seconds will be closer to 536 seconds.

Note also that if the calculations result in a loop_cnt of zero, the bsp_prv_software_delay_loop() function is not called at all. In this case the requested delay is too small (nanoseconds) to be carried out by the loop itself, and the overhead associated with executing the code to just get to this point has certainly satisfied the requested delay.

Note

This function calls bsp_cpu_clock_get() which ultimately calls R_CGC_SystemClockFreqGet() and therefore requires that the BSP has already initialized the CGC (which it does as part of the Sysinit). Care should be taken to ensure this remains the case if in the future this function were to be called as part of the BSP initialization.
*This function will delay for **at least** the specified duration. Due to overhead in calculating the correct number of loops to delay, very small delay values (generally 1-5 microseconds) may be significantly longer than specified. Approximate overhead for this function is as follows:*

- *CM4: 20-50 cycles*
- *CM33: 10-60 cycles*
- *CM23: 75-200 cycles*

If more accurate microsecond timing must be performed in software it is recommended to use bsp_prv_software_delay_loop() directly. In this case, use BSP_DELAY_LOOP_CYCLES or BSP_DELAY_LOOPS_CALCULATE() to convert a calculated delay cycle count to a number of software delay loops.
Delays may be longer than expected when compiler optimization is turned off.

Warning

The delay will be longer than specified on CM23 devices when the core clock is greater than 32 MHz. Setting BSP_DELAY_LOOP_CYCLES to 6 will improve accuracy at 48 MHz but will result in shorter than expected delays at lower speeds.

◆ R_BSP_GroupIrqWrite()

fsp_err_t R_BSP_GroupIrqWrite (bsp_grp_irq_t *irq*, void(*)(bsp_grp_irq_t irq) *p_callback*)

Register a callback function for supported interrupts. If NULL is passed for the callback argument then any previously registered callbacks are unregistered.

Parameters

[in]	irq	Interrupt for which to register a callback.
[in]	p_callback	Pointer to function to call when interrupt occurs.

Return values

FSP_SUCCESS	Callback registered
FSP_ERR_ASSERTION	Callback pointer is NULL

◆ R_BSP_RegisterProtectEnable()

void R_BSP_RegisterProtectEnable (bsp_reg_protect_t *regs_to_protect*)

Enable register protection. Registers that are protected cannot be written to. Register protection is enabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR).

Parameters

[in]	regs_to_protect	Registers which have write protection enabled.

◆ R_BSP_RegisterProtectDisable()

void R_BSP_RegisterProtectDisable (bsp_reg_protect_t *regs_to_unprotect*)

Disable register protection. Registers that are protected cannot be written to. Register protection is disabled by using the Protect Register (PRCR) and the MPC's Write-Protect Register (PWPR).

Parameters

[in]	regs_to_unprotect	Registers which have write protection disabled.

◆ __sinf()

BSP_TFU_INLINE float __sinf (float *angle*)

Calculates sine of the given angle.

Parameters

[in]	angle	The value of an angle in radian.

Return values

Sine	value of an angle.

◆ __cosf()

BSP_TFU_INLINE float __cosf (float *angle*)

Calculates cosine of the given angle.

Parameters

[in]	angle	The value of an angle in radian.

Return values

Cosine	value of an angle.

◆ __sincosf()

BSP_TFU_INLINE void __sincosf (float *angle*, float * *sin*, float * *cos*)

Calculates sine and cosine of the given angle.

Parameters

[in]	angle	The value of an angle in radian.
[out]	sin	Sine value of an angle.
[out]	cos	Cosine value of an angle.

◆ __atan2f()

BSP_TFU_INLINE float __atan2f (float *y_cord*, float *x_cord*)

Calculates the arc tangent based on given X-cordinate and Y-cordinate values.

Parameters

[in]	y_cord	Y-Axis cordinate value.
[in]	x_cord	X-Axis cordinate value.

Return values

Arc	tangent for given values.

◆ __hypotf()

BSP_TFU_INLINE float __hypotf (float *x_cord*, float *y_cord*)

Calculates the hypotenuse based on given X-cordinate and Y-cordinate values.

Parameters

[in]	y_cord	Y-cordinate value.
[in]	x_cord	X-cordinate value.

Return values

Hypotenuse	for given values.

◆ __atan2hypotf()

BSP_TFU_INLINE void __atan2hypotf (float *y_cord*, float *x_cord*, float * *atan2*, float * *hypot*)

Calculates the arc tangent and hypotenuse based on given X-cordinate and Y-cordinate values.

Parameters

[in]	y_cord	Y-cordinate value.
[in]	x_cord	X-cordinate value.
[out]	atan2	Arc tangent for given values.
[out]	hypot	Hypotenuse for given values.

Variable Documentation

◆ BSP_SECTION_EARLY_INIT

uint32_t SystemCoreClock BSP_SECTION_EARLY_INIT

System Clock Frequency (Core Clock)

4.1.2.1 RA2A1

BSP » MCU Board Support Package

Detailed Description

Build Time Configurations for ra2a1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	

OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled
OFS0 register settings > Independent WDT > Stop Control	• Counting continues • Stop counting when in Sleep, Snooze mode, or Software Standby	Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End	• 75% • 50%	0% (no window end position)

Position	• 25% • 0% (no window end position)		
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)	
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset	
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode	
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset	
OFS1 register settings > Voltage Detection 0 Level	• 3.84 V • 2.82 V • 2.51 V • 1.90 V • 1.70 V	1.90 V	
OFS1 register settings > HOCO Oscillation Enable	HOCO oscillation is enabled after reset	HOCO oscillation is enabled after reset	HOCO must be enabled out of reset because the MCU starts up in low voltage mode and the HOCO must be operating in low voltage mode.
MPU > Enable or disable PC Region 0	• Enabled • Disabled	Disabled	
MPU > PC0 Start	Value must be an integer between 0 and 0x000FFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	0x000FFFFC	
MPU > PC0 End	Value must be an integer between 0x00000003 and 0x000FFFFF (ROM) or between 0x1FF00003	0x000FFFFF	

	and 0x200FFFFF (RAM)	
MPU > Enable or disable PC Region 1	• Enabled • Disabled	Disabled
MPU > PC1 Start	Value must be an integer between 0 and 0x000FFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	0x000FFFFC
MPU > PC1 End	Value must be an integer between 0x00000003 and 0x000FFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	0x000FFFFF
MPU > Enable or disable Memory Region 0	• Enabled • Disabled	Disabled
MPU > Memory Region 0 Start	Value must be an integer between 0 and 0x000FFFFC	0x000FFFFC
MPU > Memory Region 0 End	Value must be an integer between 0x00000003 and 0x000FFFFF	0x000FFFFF
MPU > Enable or disable Memory Region 1	• Enabled • Disabled	Disabled
MPU > Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFFC	0x200FFFFC
MPU > Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFF	0x200FFFFF
MPU > Enable or disable Memory Region 2	• Enabled • Disabled	Disabled
MPU > Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x407FFFFC
MPU > Memory Region 2 End	Value must be an integer between 0x400C0003 and	0x407FFFFF

	0x400DFFFF or between 0x40100003 and 0x407FFFFF		
MPU > Enable or disable Memory Region 3	• Enabled • Disabled	Disabled	
MPU > Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x400DFFFC	
MPU > Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x400DFFFF	
Use Low Voltage Mode	• Enabled • Disabled	Disabled	Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used.
Main Oscillator Wait Time	• 2 cycles • 1024 cycles • 2048 cycles • 4096 cycles • 8192 cycles • 16384 cycles • 32768 cycles • 65536 cycles • 131072 cycles • 262144 cycles	262144 cycles	Number of cycles to wait for the main oscillator clock to stabilize.
ID Code Mode	• Unlocked (Ignore ID) • Locked with All Erase support • Locked	Unlocked (Ignore ID)	When set to 'Locked with All Erase support', the ID Code must be set in the debugger to read or write data to the MCU, but the All Erase command is still accepted regardless. When set to 'Locked', all erase/download/debug access is disabled unless the ID Code is provided.
ID Code (32 Hex	Value must be a 32	FFFFFFFFFFFFFFFFFFFF	Set the ID Code for

Characters)	character long hex string	FFFFFFFFFFFF	locking debug access. This setting is only used when the ID Code Mode is not set to Unlocked.

Enumerations

enum	elc_event_t

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

Note

This list may change based on based on the device.

4.1.2.2 RA2E1

BSP » MCU Board Support Package

Detailed Description

Build Time Configurations for ra2e1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128	

OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled
OFS0 register settings > Independent WDT > Stop Control	• Counting continues • Stop counting when in Sleep, Snooze mode, or Software Standby	Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75%	100% (no window start position)

	• 100% (no window start position)		
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset	
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode	
OFS1 register settings > Internal Clock Supply Architecture Type	• Type B • Type A	Type A	
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset	
OFS1 register settings > Voltage Detection 0 Level	• 3.84 V • 2.82 V • 2.51 V • 1.90 V • 1.70 V	1.90 V	
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset	HOCO oscillation is enabled after reset	HOCO must be enabled out of reset because the MCU starts up in low voltage mode and the HOCO must be operating in low voltage mode.
MPU > Enable or disable PC Region 0	• Enabled • Disabled	Disabled	
MPU > PC0 Start	Value must be an integer between 0 and 0x000FFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	0x000FFFFC	
MPU > PC0 End	Value must be an integer between 0x00000003 and 0x000FFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	0x000FFFFF	
MPU > Enable or	• Enabled	Disabled	

disable PC Region 1	• Disabled	
MPU > PC1 Start	Value must be an integer between 0 and 0x000FFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	0x000FFFFC
MPU > PC1 End	Value must be an integer between 0x00000003 and 0x000FFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	0x000FFFFF
MPU > Enable or disable Memory Region 0	• Enabled • Disabled	Disabled
MPU > Memory Region 0 Start	Value must be an integer between 0 and 0x000FFFFC	0x000FFFFC
MPU > Memory Region 0 End	Value must be an integer between 0x00000003 and 0x000FFFFF	0x000FFFFF
MPU > Enable or disable Memory Region 1	• Enabled • Disabled	Disabled
MPU > Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFFC	0x200FFFFC
MPU > Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFF	0x200FFFFF
MPU > Enable or disable Memory Region 2	• Enabled • Disabled	Disabled
MPU > Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x407FFFFC
MPU > Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x407FFFFF

MPU > Enable or disable Memory Region 3	• Enabled • Disabled	Disabled	
MPU > Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x400DFFFC	
MPU > Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x400DFFFF	
Use Low Voltage Mode	Not Supported	config.bsp.low_voltage_mode.disabled	Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used.
Main Oscillator Wait Time	• 2 cycles • 1024 cycles • 2048 cycles • 4096 cycles • 8192 cycles • 16384 cycles • 32768 cycles • 65536 cycles • 131072 cycles • 262144 cycles	262144 cycles	Number of cycles to wait for the main oscillator clock to stabilize.
ID Code Mode	• Unlocked (Ignore ID) • Locked with All Erase support • Locked	Unlocked (Ignore ID)	When set to 'Locked with All Erase support', the ID Code must be set in the debugger to read or write data to the MCU, but the All Erase command is still accepted regardless. When set to 'Locked', all erase/download/debug access is disabled unless the ID Code is provided.
ID Code (32 Hex Characters)	Value must be a 32 character long hex string	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	Set the ID Code for locking debug access. This setting is only used when the ID Code

Mode is not set to Unlocked.

Enumerations

enum	elc_event_t
enum	icu_event_t

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

Note
This list may change based on based on the device.

◆ icu_event_t

enum icu_event_t

Events to be used with the IELSR register to link interrupt events to the NVIC

Note
This list is device specific.

4.1.2.3 RA2E2

BSP » MCU Board Support Package

Detailed Description

Build Time Configurations for ra2e2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings	• 128 cycles	2048 cycles	

> Independent WDT > Timeout Period	• 512 cycles • 1024 cycles • 2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled
OFS0 register settings > Independent WDT > Stop Control	• Counting continues • Stop counting when in Sleep, Snooze mode, or Software Standby	Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048	128

	• 8192		
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)	
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)	
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset	
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode	
OFS1 register settings > Internal Clock Supply Architecture Type	• Type B • Type A	Type A	
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset	
OFS1 register settings > Voltage Detection 0 Level	• 3.84 V • 2.82 V • 2.51 V • 1.90 V • 1.70 V	1.90 V	
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset	HOCO oscillation is enabled after reset	HOCO must be enabled out of reset because the MCU starts up in low voltage mode and the HOCO must be operating in low voltage mode.
MPU > Enable or disable PC Region 0	• Enabled • Disabled	Disabled	
MPU > PC0 Start	Value must be an integer between 0 and 0x000FFFFC (ROM) or	0x000FFFFC	

	between 0x1FF00000 and 0x200FFFFC (RAM)	
MPU > PC0 End	Value must be an integer between 0x00000003 and 0x000FFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	0x000FFFFF
MPU > Enable or disable PC Region 1	• Enabled • Disabled	Disabled
MPU > PC1 Start	Value must be an integer between 0 and 0x000FFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	0x000FFFFC
MPU > PC1 End	Value must be an integer between 0x00000003 and 0x000FFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	0x000FFFFF
MPU > Enable or disable Memory Region 0	• Enabled • Disabled	Disabled
MPU > Memory Region 0 Start	Value must be an integer between 0 and 0x000FFFFC	0x000FFFFC
MPU > Memory Region 0 End	Value must be an integer between 0x00000003 and 0x000FFFFF	0x000FFFFF
MPU > Enable or disable Memory Region 1	• Enabled • Disabled	Disabled
MPU > Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFFC	0x200FFFFC
MPU > Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFF	0x200FFFFF
MPU > Enable or disable Memory Region 2	• Enabled • Disabled	Disabled
MPU > Memory Region 2 Start	Value must be an integer between	0x407FFFFC

	0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC		
MPU > Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x407FFFFF	
MPU > Enable or disable Memory Region 3	• Enabled • Disabled	Disabled	
MPU > Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x400DFFFC	
MPU > Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x400DFFFF	
Use Low Voltage Mode	Not Supported	config.bsp.low_voltage_mode.disabled	Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used.
ID Code Mode	• Unlocked (Ignore ID) • Locked with All Erase support • Locked	Unlocked (Ignore ID)	When set to 'Locked with All Erase support', the ID Code must be set in the debugger to read or write data to the MCU, but the All Erase command is still accepted regardless. When set to 'Locked', all erase/download/debug access is disabled unless the ID Code is provided.
ID Code (32 Hex Characters)	Value must be a 32 character long hex string	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	Set the ID Code for locking debug access. This setting is only used when the ID Code

Mode is not set to Unlocked.

Enumerations

enum	elc_event_t
enum	icu_event_t

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

Note
This list may change based on based on the device.

◆ icu_event_t

enum icu_event_t

Events to be used with the IELSR register to link interrupt events to the NVIC

Note
This list is device specific.

4.1.2.4 RA2L1

BSP » MCU Board Support Package

Functions

bsp_power_mode_t	R_BSP_PowerModeSet (bsp_power_mode_t mode)

Detailed Description

Build Time Configurations for ra2l1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after	IWDT is Disabled	

	a reset (Autostart mode)	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled
OFS0 register settings > Independent WDT > Stop Control	• Counting continues • Stop counting when in Sleep, Snooze mode, or Software Standby	Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles

OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128	
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)	
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)	
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset	
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode	
OFS1 register settings > Internal Clock Supply Architecture Type	• Type B • Type A	Type A	
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset	
OFS1 register settings > Voltage Detection 0 Level	• 3.84 V • 2.82 V • 2.51 V • 1.90 V • 1.70 V	1.90 V	
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset	HOCO oscillation is enabled after reset	HOCO must be enabled out of reset because the MCU starts up in low voltage mode and the HOCO must be operating in low voltage mode.

MPU > Enable or disable PC Region 0	• Enabled • Disabled	Disabled
MPU > PC0 Start	Value must be an integer between 0 and 0x000FFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	0x000FFFFC
MPU > PC0 End	Value must be an integer between 0x00000003 and 0x000FFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	0x000FFFFF
MPU > Enable or disable PC Region 1	• Enabled • Disabled	Disabled
MPU > PC1 Start	Value must be an integer between 0 and 0x000FFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	0x000FFFFC
MPU > PC1 End	Value must be an integer between 0x00000003 and 0x000FFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	0x000FFFFF
MPU > Enable or disable Memory Region 0	• Enabled • Disabled	Disabled
MPU > Memory Region 0 Start	Value must be an integer between 0 and 0x000FFFFC	0x000FFFFC
MPU > Memory Region 0 End	Value must be an integer between 0x00000003 and 0x000FFFFF	0x000FFFFF
MPU > Enable or disable Memory Region 1	• Enabled • Disabled	Disabled
MPU > Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFFC	0x200FFFFC
MPU > Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFF	0x200FFFFF

MPU > Enable or disable Memory Region 2	• Enabled • Disabled	Disabled	
MPU > Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x407FFFFC	
MPU > Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x407FFFFF	
MPU > Enable or disable Memory Region 3	• Enabled • Disabled	Disabled	
MPU > Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x400DFFFC	
MPU > Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x400DFFFF	
Power > DC-DC Regulator	• Disabled • Enabled • Enabled at startup	Disabled	To use the DCDC regulator an external inductor and capacitor must be connected as specified in chapter 40 of the RA2L1 manual. In addition the supply voltage must be above 2.4V and ICLK must be 2 MHz or higher. When set to 'Enabled at startup' the BSP will switch to the DCDC regulator during startup using the voltage range specified below.
Power > DC-DC Supply Range	• 2.4V to 2.7V • 2.7V to 3.6V • 3.6V to 4.5V	2.7V to 3.6V	Set this to the expected MCU supply voltage (Vcc) at startup

	• 4.5V to 5.5V		when using the DCDC regulator.
Use Low Voltage Mode	Not Supported	config.bsp.low_voltage_mode.disabled	Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4 when oscillation stop detection is used.
Main Oscillator Wait Time	• 2 cycles • 1024 cycles • 2048 cycles • 4096 cycles • 8192 cycles • 16384 cycles • 32768 cycles • 65536 cycles • 131072 cycles • 262144 cycles	262144 cycles	Number of cycles to wait for the main oscillator clock to stabilize.
ID Code Mode	• Unlocked (Ignore ID) • Locked with All Erase support • Locked	Unlocked (Ignore ID)	When set to 'Locked with All Erase support', the ID Code must be set in the debugger to read or write data to the MCU, but the All Erase command is still accepted regardless. When set to 'Locked', all erase/download/debug access is disabled unless the ID Code is provided.
ID Code (32 Hex Characters)	Value must be a 32 character long hex string	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	Set the ID Code for locking debug access. This setting is only used when the ID Code Mode is not set to Unlocked.

Common macro for FSP header files. There is also a corresponding FSP_FOOTER macro at the end of this file.

Enumerations

enum	elc_event_t
enum	icu_event_t
enum	bsp_power_mode_t

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

Note

This list may change based on based on the device.

◆ icu_event_t

enum icu_event_t

Events to be used with the IELSR register to link interrupt events to the NVIC

Note

This list is device specific.

◆ bsp_power_mode_t

enum bsp_power_mode_t

Voltage regulator mode

Enumerator	
BSP_POWER_MODE_DCDC_2V4_TO_2V7	DCDC mode; 2.4V to 2.7V supply.
BSP_POWER_MODE_DCDC_2V7_TO_3V6	DCDC mode; 2.7V to 3.6V supply.
BSP_POWER_MODE_DCDC_3V6_TO_4V5	DCDC mode; 3.6V to 4.5V supply.
BSP_POWER_MODE_DCDC_4V5_TO_5V5	DCDC mode; 4.5V to 5.5V supply.
BSP_POWER_MODE_LDO	LDO mode.

Function Documentation

◆ R_BSP_PowerModeSet()

bsp_power_mode_t R_BSP_PowerModeSet (bsp_power_mode_t *mode*)
Select either the LDO or DCDC regulator and/or update the MCU supply voltage range. Returns the previously selected mode. ***Note*** *DCDC mode has the following limitations:* ◦ *Supply voltage must be 2.4V or greater* ◦ *Low- and Subosc-speed modes are not available* ◦ *Software Standby is not available Ensure these limitations are respected before entering DCDC mode. If supply voltage may drop below 2.4V during operation, configure a LVD channel to interrupt or reset the MCU near this threshold to switch back to the LDO.* *Switching to DCDC mode temporarily disables all interrupts and blocks for 22 microseconds; switching to LDO from DCDC temporarily disables all peripherals and interrupts and blocks for 60 microseconds.* *If the supply voltage falls outside the range originally specified when starting the DCDC regulator, call this function again with the updated supply voltage.* **Returns** The previously selected power mode.

4.1.2.5 RA4E1

BSP » MCU Board Support Package

Detailed Description

Build Time Configurations for ra4e1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
Security > Exceptions > Exception Response	• Non-Maskable Interrupt • Reset	Non-Maskable Interrupt	Configure the result of a TrustZone Filter exception. This exception is generated when a the TrustZone Filter detects access to a protected region. This setting is only valid when building projects with TrustZone.
Security > Exceptions > BusFault, HardFault, and NMI Target	• Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether

			exceptions target the Non-secure HardFault exception. This setting is only valid when building projects with TrustZone.
Security > Exceptions > Prioritize Secure Exceptions	• Enabled • Disabled	Disabled	Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > SRAM Protection	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAMPRCR is write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > SRAM ECC	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAM ECC registers are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > Standby RAM	• Regions 7-0 are all Secure. • Region 7 is Non-secure. Regions 6-0 are Secure. • Regions 7-6 are	config.bsp.fsp.tz.stbramsar.both	Defines whether Standby RAM registers are accessible for the Non-secure application. This setting is only

	Non-secure. Regions 5-0 are Secure. • Regions 7-5 are Non-secure. Regions 4-0 are Secure. • Regions 7-4 are Non-secure. Regions 3-0 are Secure. • Regions 7-3 are Non-secure. Regions 2-0 are Secure. • Regions 7-2 are Non-secure. Regions 1-0 are Secure. • Regions 7-1 are Non-secure. Region 0 is Secure. • Regions 7-0 are all Non-secure.		valid when building projects with TrustZone.
Security > BUS Accessibility > Bus Security Attribution Register A	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > BUS Accessibility > Bus Security Attribution Register B	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > System Reset Request Accessibility	• Both Secure and Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-

			secure use. This setting is only valid when building projects with TrustZone.
Security > System Reset Status Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > Battery Backup Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected. This setting is only valid when building projects with TrustZone.
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128	

OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled
OFS0 register settings > Independent WDT > Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) • Stop counting when in Sleep, Snooze mode, or Software Standby	Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings	• 75%	0% (no window end

> WDT > Window End Position	• 50% • 25% • 0% (no window end position)	position)	
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)	
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset	
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode	
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset	
OFS1 register settings > Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V	2.80 V	
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset	HOCO oscillation is enabled after reset	
Block Protection Settings (BPS) > BPS	Refer to the RA Configuration tool for available options.	0U	Configure Block Protection Register
Permanent Block Protection Settings (PBPS) > PBPS	Refer to the RA Configuration tool for available options.	0U	Configure Permanent Block Protection Register
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain

			restrictions. The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup. When FLL is enabled Software Standby and Deep Software Standby modes are not available.
Main Oscillator Wait Time	• 3 cycles • 35 cycles • 67 cycles • 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to stabilize.

Enumerations

enum	elc_event_t

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

Note

This list may change based on based on the device.

4.1.2.6 RA4M1

BSP » MCU Board Support Package

Detailed Description

Build Time Configurations for ra4m1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128	
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)	
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)	
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled	
OFS0 register settings > Independent WDT > Stop Control	• Counting continues • Stop counting when in Sleep, Snooze mode, or Software Standby	Stop counting when in Sleep, Snooze mode, or Software Standby	
OFS0 register settings	• Automatically	Stop WDT after a reset	

> WDT > Start Mode Select	activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	(register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset
OFS1 register settings > Voltage Detection 0 Level	• 3.84 V • 2.82 V • 2.51 V • 1.90 V • 1.70 V	1.90 V

OFS1 register settings > HOCO Oscillation Enable	HOCO oscillation is enabled after reset	HOCO oscillation is enabled after reset	HOCO must be enabled out of reset because the MCU starts up in low voltage mode and the HOCO must be operating in low voltage mode.
MPU > Enable or disable PC Region 0	• Enabled • Disabled	Disabled	
MPU > PC0 Start	Value must be an integer between 0 and 0x00FFFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	0x00FFFFFC	
MPU > PC0 End	Value must be an integer between 0x00000003 and 0x00FFFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	0x00FFFFFF	
MPU > Enable or disable PC Region 1	• Enabled • Disabled	Disabled	
MPU > PC1 Start	Value must be an integer between 0 and 0x00FFFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	0x00FFFFFC	
MPU > PC1 End	Value must be an integer between 0x00000003 and 0x00FFFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	0x00FFFFFF	
MPU > Enable or disable Memory Region 0	• Enabled • Disabled	Disabled	
MPU > Memory Region 0 Start	Value must be an integer between 0 and 0x00FFFFFC	0x00FFFFFC	
MPU > Memory Region 0 End	Value must be an integer between 0x00000003 and 0x00FFFFFF	0x00FFFFFF	
MPU > Enable or disable Memory Region 1	• Enabled • Disabled	Disabled	
MPU > Memory Region 1 Start	Value must be an integer between	0x200FFFFC	

	0x1FF00000 and 0x200FFFFC		
MPU > Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFF	0x200FFFFF	
MPU > Enable or disable Memory Region 2	• Enabled • Disabled	Disabled	
MPU > Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x407FFFFC	
MPU > Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x407FFFFF	
MPU > Enable or disable Memory Region 3	• Enabled • Disabled	Disabled	
MPU > Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x400DFFFC	
MPU > Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x400DFFFF	
Use Low Voltage Mode	• Enabled • Disabled	Disabled	Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4.
Main Oscillator Wait Time	• 2 cycles • 1024 cycles • 2048 cycles • 4096 cycles • 8192 cycles • 16384 cycles	262144 cycles	Number of cycles to wait for the main oscillator clock to stabilize.

	• 32768 cycles • 65536 cycles • 131072 cycles • 262144 cycles		
ID Code Mode	• Unlocked (Ignore ID) • Locked with All Erase support • Locked	Unlocked (Ignore ID)	When set to 'Locked with All Erase support', the ID Code must be set in the debugger to read or write data to the MCU, but the All Erase command is still accepted regardless. When set to 'Locked', all erase/download/debug access is disabled unless the ID Code is provided.
ID Code (32 Hex Characters)	Value must be a 32 character long hex string	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	Set the ID Code for locking debug access. This setting is only used when the ID Code Mode is not set to Unlocked.

Enumerations

enum	elc_event_t

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

Note

This list may change based on based on the device.

4.1.2.7 RA4M2

BSP » MCU Board Support Package

Build Time Configurations for ra4m2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description

Security > Exceptions > Exception Response	• Non-Maskable Interrupt • Reset	Non-Maskable Interrupt	Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region. This setting is only valid when building projects with TrustZone.
Security > Exceptions > BusFault, HardFault, and NMI Target	• Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception. This setting is only valid when building projects with TrustZone.
Security > Exceptions > Prioritize Secure Exceptions	• Enabled • Disabled	Disabled	Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > SRAM Protection	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAMPRCR is write accessible for the Non-secure application. This setting is only valid when building

			projects with TrustZone.
Security > SRAM Accessibility > SRAM ECC	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAM ECC registers are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > Standby RAM	• Regions 7-0 are all Secure. • Region 7 is Non-secure. Regions 6-0 are Secure. • Regions 7-6 are Non-secure. Regions 5-0 are Secure. • Regions 7-5 are Non-secure. Regions 4-0 are Secure. • Regions 7-4 are Non-secure. Regions 3-0 are Secure. • Regions 7-3 are Non-secure. Regions 2-0 are Secure. • Regions 7-2 are Non-secure. Regions 1-0 are Secure. • Regions 7-1 are Non-secure. Region 0 is Secure. • Regions 7-0 are all Non-secure.	config.bsp.fsp.tz.stbramsar.both	Defines whether Standby RAM registers are accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > BUS Accessibility > Bus Security Attribution Register A	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.

Security > BUS Accessibility > Bus Security Attribution Register B	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > System Reset Request Accessibility	• Both Secure and Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use. This setting is only valid when building projects with TrustZone.
Security > Cache Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Cache registers are write accessible for the Non-secure application. This setting is only valid when building protjects with TrustZone.
Security > System Reset Status Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > Battery Backup Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are

			both read and write protected. This setting is only valid when building projects with TrustZone.
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128	
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)	
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)	
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled	
OFS0 register settings > Independent WDT > Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software	Stop counting when in Sleep, Snooze mode, or Software Standby	

	Standby Mode) • Stop counting when in Sleep, Snooze mode, or Software Standby	
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after	Voltage monitor 0 reset is disabled after reset

	reset		
OFS1 register settings > Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V	2.80 V	
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset	HOCO oscillation is disabled after reset	
Block Protection Settings (BPS) > BPS0	Refer to the RA Configuration tool for available options.	0U	Configure Block Protection Register 0
Permanent Block Protection Settings (PBPS) > PBPS0	Refer to the RA Configuration tool for available options.	0U	Configure Permanent Block Protection Register 0
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup. When FLL is enabled Software Standby and Deep Software Standby modes are not available.
Startup C-Cache Line Size	• 32 Bytes • 64 Bytes	32 Bytes	Set the C-Cache line size configured during startup.
Main Oscillator Wait Time	• 3 cycles • 35 cycles • 67 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to

	• 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles		stabilize.

4.1.2.8 RA4M3

BSP » MCU Board Support Package

Build Time Configurations for ra4m3_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
Security > Exceptions > Exception Response	• Non-Maskable Interrupt • Reset	Non-Maskable Interrupt	Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region. This setting is only valid when building projects with TrustZone.
Security > Exceptions > BusFault, HardFault, and NMI Target	• Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception. This setting is only valid when building projects with TrustZone.
Security > Exceptions > Prioritize Secure Exceptions	• Enabled • Disabled	Disabled	Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their

			priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > SRAM Protection	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAMPRCR is write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > SRAM ECC	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAM ECC registers are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > Standby RAM	• Regions 7-0 are all Secure. • Region 7 is Non-secure. Regions 6-0 are Secure. • Regions 7-6 are Non-secure. Regions 5-0 are Secure. • Regions 7-5 are Non-secure. Regions 4-0 are Secure. • Regions 7-4 are Non-secure. Regions 3-0 are Secure. • Regions 7-3 are Non-secure. Regions 2-0 are Secure.	config.bsp.fsp.tz.stbramsar.both	Defines whether Standby RAM registers are accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.

	• Regions 7-2 are Non-secure. Regions 1-0 are Secure. • Regions 7-1 are Non-secure. Region 0 is Secure. • Regions 7-0 are all Non-secure.		
Security > BUS Accessibility > Bus Security Attribution Register A	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > BUS Accessibility > Bus Security Attribution Register B	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > System Reset Request Accessibility	• Both Secure and Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use. This setting is only valid when building projects with TrustZone.
Security > Cache Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Cache registers are write accessible for the Non-secure application. This setting is only valid when building protjects with

			TrustZone.
Security > System Reset Status Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > Battery Backup Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected. This setting is only valid when building projects with TrustZone.
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128	
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)	

OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled
OFS0 register settings > Independent WDT > Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) • Stop counting when in Sleep, Snooze mode, or Software Standby	Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings	• 25%	100% (no window start

> WDT > Window Start Position	• 50% • 75% • 100% (no window start position)	position)	
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset	
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode	
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset	
OFS1 register settings > Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V	2.80 V	
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset	HOCO oscillation is disabled after reset	
Block Protection Settings (BPS) > BPS0	Refer to the RA Configuration tool for available options.	0U	Configure Block Protection Register 0
Block Protection Settings (BPS) > BPS1	• Flash Block 32 • Flash Block 33 • Flash Block 34 • Flash Block 35 • Flash Block 36 • Flash Block 37	0U	Configure Block Protection Register 1
Permanent Block Protection Settings (PBPS) > PBPS0	Refer to the RA Configuration tool for available options.	0U	Configure Permanent Block Protection Register 0
Permanent Block Protection Settings (PBPS) > PBPS1	• Flash Block 32 • Flash Block 33 • Flash Block 34 • Flash Block 35 • Flash Block 36	0U	Configure Permanent Block Protection Register 1

	• Flash Block 37		
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup. When FLL is enabled Software Standby and Deep Software Standby modes are not available.
Startup C-Cache Line Size	• 32 Bytes • 64 Bytes	32 Bytes	Set the C-Cache line size configured during startup.
Main Oscillator Wait Time	• 3 cycles • 35 cycles • 67 cycles • 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to stabilize.

4.1.2.9 RA4W1

BSP » MCU Board Support Package

Detailed Description

Build Time Configurations for ra4w1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128	
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)	
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)	
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled	
OFS0 register settings > Independent WDT > Stop Control	• Counting continues • Stop counting when in Sleep, Snooze mode, or Software Standby	Stop counting when in Sleep, Snooze mode, or Software Standby	
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode)	Stop WDT after a reset (register-start mode)	

	• Stop WDT after a reset (register-start mode)	
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset
OFS1 register settings > Voltage Detection 0 Level	• 2.82 V • 2.51 V • 1.90 V	1.90 V
OFS1 register settings > HOCO Oscillation Enable	HOCO oscillation is enabled after reset	config.bsp.fsp.OFS1.hoco_osc.disabled
MPU > Enable or disable PC Region 0	• Enabled • Disabled	Disabled

MPU > PC0 Start	Value must be an integer between 0 and 0x00FFFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	0x00FFFFFC
MPU > PC0 End	Value must be an integer between 0x00000003 and 0x00FFFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	0x00FFFFFF
MPU > Enable or disable PC Region 1	• Enabled • Disabled	Disabled
MPU > PC1 Start	Value must be an integer between 0 and 0x00FFFFFC (ROM) or between 0x1FF00000 and 0x200FFFFC (RAM)	0x00FFFFFC
MPU > PC1 End	Value must be an integer between 0x00000003 and 0x00FFFFFF (ROM) or between 0x1FF00003 and 0x200FFFFF (RAM)	0x00FFFFFF
MPU > Enable or disable Memory Region 0	• Enabled • Disabled	Disabled
MPU > Memory Region 0 Start	Value must be an integer between 0 and 0x00FFFFFC	0x00FFFFFC
MPU > Memory Region 0 End	Value must be an integer between 0x00000003 and 0x00FFFFFF	0x00FFFFFF
MPU > Enable or disable Memory Region 1	• Enabled • Disabled	Disabled
MPU > Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFFC	0x200FFFFC
MPU > Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFF	0x200FFFFF
MPU > Enable or disable Memory Region 2	• Enabled • Disabled	Disabled

MPU > Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x407FFFFC	
MPU > Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x407FFFFF	
MPU > Enable or disable Memory Region 3	• Enabled • Disabled	Disabled	
MPU > Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x400DFFFC	
MPU > Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x400DFFFF	
Use Low Voltage Mode	• Enabled • Disabled	Disabled	Use the low voltage mode. This limits the ICLK operating frequency to 4 MHz and requires all clock dividers to be at least 4.
Main Oscillator Wait Time	• 2 cycles • 1024 cycles • 2048 cycles • 4096 cycles • 8192 cycles • 16384 cycles • 32768 cycles • 65536 cycles • 131072 cycles • 262144 cycles	262144 cycles	Number of cycles to wait for the main oscillator clock to stabilize.
ID Code Mode	• Unlocked (Ignore ID) • Locked with All Erase support • Locked	Unlocked (Ignore ID)	When set to 'Locked with All Erase support', the ID Code must be set in the debugger to read or write data to the MCU, but the All Erase command is still

			accepted regardless. When set to 'Locked', all erase/download/debug access is disabled unless the ID Code is provided.
ID Code (32 Hex Characters)	Value must be a 32 character long hex string	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	Set the ID Code for locking debug access. This setting is only used when the ID Code Mode is not set to Unlocked.

Enumerations

enum	elc_event_t

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU1

Note

This list may change based on device. This list is for RA4W1.

4.1.2.10 RA6E1

BSP » MCU Board Support Package

Detailed Description

Build Time Configurations for ra6e1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
Security > Exceptions > Exception Response	• Non-Maskable Interrupt • Reset	Non-Maskable Interrupt	Configure the result of a TrustZone Filter exception. This exception is generated when a the TrustZone Filter detects access to a protected region. This setting is only valid when building

			projects with TrustZone.
Security > Exceptions > BusFault, HardFault, and NMI Target	• Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception. This setting is only valid when building projects with TrustZone.
Security > Exceptions > Prioritize Secure Exceptions	• Enabled • Disabled	Disabled	Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > SRAM Protection	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAMPRCR is write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > SRAM ECC	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAM ECC registers are write accessible for the Non-secure application. This setting is only valid when building

			projects with TrustZone.
Security > SRAM Accessibility > Standby RAM	• Regions 7-0 are all Secure. • Region 7 is Non-secure. Regions 6-0 are Secure. • Regions 7-6 are Non-secure. Regions 5-0 are Secure. • Regions 7-5 are Non-secure. Regions 4-0 are Secure. • Regions 7-4 are Non-secure. Regions 3-0 are Secure. • Regions 7-3 are Non-secure. Regions 2-0 are Secure. • Regions 7-2 are Non-secure. Regions 1-0 are Secure. • Regions 7-1 are Non-secure. Region 0 is Secure. • Regions 7-0 are all Non-secure.	config.bsp.fsp.tz.stbramsar.both	Defines whether Standby RAM registers are accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > BUS Accessibility > Bus Security Attribution Register A	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > BUS Accessibility > Bus Security Attribution Register B	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application. This setting is only valid when building

			projects with TrustZone.
Security > System Reset Request Accessibility	• Both Secure and Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use. This setting is only valid when building projects with TrustZone.
Security > Cache Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Cache registers are write accessible for the Non-secure application. This setting is only valid when building protjects with TrustZone.
Security > System Reset Status Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > Battery Backup Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected. This setting is only valid when building projects with TrustZone.
OFS0 register settings > Independent WDT >	• IWDT is Disabled	IWDT is Disabled	

Start Mode	• IWDT is automatically activated after a reset (Autostart mode)	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled
OFS0 register settings > Independent WDT > Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) • Stop counting when in Sleep, Snooze mode, or Software Standby	Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset	Stop WDT after a reset (register-start mode)

	(auto-start mode) • Stop WDT after a reset (register-start mode)	
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset
OFS1 register settings > Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V	2.80 V
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset	HOCO oscillation is enabled after reset

	• HOCO oscillation is disabled after reset		
Block Protection Settings (BPS) > BPS0	Refer to the RA Configuration tool for available options.	0U	Configure Block Protection Register 0
Block Protection Settings (BPS) > BPS1	• Flash Block 32 • Flash Block 33 • Flash Block 34 • Flash Block 35 • Flash Block 36 • Flash Block 37	0U	Configure Block Protection Register 1
Block Protection Settings (BPS) > BPS2	Refer to the RA Configuration tool for available options.	0U	Configure Block Protection Register 2
Permanent Block Protection Settings (PBPS) > PBPS0	Refer to the RA Configuration tool for available options.	0U	Configure Permanent Block Protection Register 0
Permanent Block Protection Settings (PBPS) > PBPS1	• Flash Block 32 • Flash Block 33 • Flash Block 34 • Flash Block 35 • Flash Block 36 • Flash Block 37	0U	Configure Permanent Block Protection Register 1
Permanent Block Protection Settings (PBPS) > PBPS2	Refer to the RA Configuration tool for available options.	0U	Configure Permanent Block Protection Register 2
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup. When FLL is enabled

			Software Standby and Deep Software Standby modes are not available.
Startup C-Cache Line Size	• 32 Bytes • 64 Bytes	32 Bytes	Set the C-Cache line size configured during startup.
Dual Bank Mode	• Enabled • Disabled	Disabled	Enabling dual bank mode splits the flash into two banks that can be swapped by programming the BANKSEL non-volatile register. When enabled, one bank will start at address 0x0 and the other will start at 0x200000. Each bank contains exactly half the capacity of the entire code flash. When Dual Bank mode is enabled, Startup Program Protection and Block Swap functions cannot be used.
Main Oscillator Wait Time	• 3 cycles • 35 cycles • 67 cycles • 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to stabilize.

Enumerations

enum	elc_event_t

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

Note
This list may change based on based on the device.

4.1.2.11 RA6M1

BSP » MCU Board Support Package

Detailed Description

Build Time Configurations for ra6m1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128	
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)	
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)	
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled	
OFS0 register settings > Independent WDT >	• Counting continues	Stop counting when in Sleep, Snooze mode, or	

Stop Control	(Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) • Stop counting when in Sleep, Snooze mode, or Software Standby	Software Standby
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode

OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset
OFS1 register settings > Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V	2.80 V
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset	HOCO oscillation is disabled after reset
MPU > Enable or disable PC Region 0	• Enabled • Disabled	Disabled
MPU > PC0 Start	Value must be an integer between 0 and 0xFFFFFFFC	0xFFFFFFFC
MPU > PC0 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF	0xFFFFFFFF
MPU > Enable or disable PC Region 1	• Enabled • Disabled	Disabled
MPU > PC1 Start	Value must be an integer between 0 and 0xFFFFFFFC	0xFFFFFFFC
MPU > PC1 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF	0xFFFFFFFF
MPU > Enable or disable Memory Region 0	• Enabled • Disabled	Disabled
MPU > Memory Region 0 Start	Value must be an integer between 0 and 0x00FFFFFC	0x00FFFFFC
MPU > Memory Region 0 End	Value must be an integer between 0x00000003 and 0x00FFFFFF	0x00FFFFFF

MPU > Enable or disable Memory Region 1	• Enabled • Disabled	Disabled	
MPU > Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFFC	0x200FFFFC	
MPU > Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFF	0x200FFFFF	
MPU > Enable or disable Memory Region 2	• Enabled • Disabled	Disabled	
MPU > Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x407FFFFC	
MPU > Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x407FFFFF	
MPU > Enable or disable Memory Region 3	• Enabled • Disabled	Disabled	
MPU > Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x400DFFFC	
MPU > Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x400DFFFF	
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions.

			The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup. When FLL is enabled Software Standby and Deep Software Standby modes are not available.
Main Oscillator Wait Time	• 35 cycles • 67 cycles • 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to stabilize. Drive capability automatic switching function is by default disabled.
ID Code Mode	• Unlocked (Ignore ID) • Locked with All Erase support • Locked	Unlocked (Ignore ID)	When set to 'Locked with All Erase support', the ID Code must be set in the debugger to read or write data to the MCU, but the All Erase command is still accepted regardless. When set to 'Locked', all erase/download/debug access is disabled unless the ID Code is provided.
ID Code (32 Hex Characters)	Value must be a 32 character long hex string	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	Set the ID Code for locking debug access. This setting is only used when the ID Code Mode is not set to Unlocked.

Enumerations

enum	elc_event_t

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

Note
This list may change based on based on the device.

4.1.2.12 RA6M2

BSP » MCU Board Support Package

Detailed Description

Build Time Configurations for ra6m2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128	
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)	
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75%	100% (no window start position)	

	• 100% (no window start position)	
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled
OFS0 register settings > Independent WDT > Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) • Stop counting when in Sleep, Snooze mode, or Software Standby	Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no	100% (no window start position)

	window start position)	
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset
OFS1 register settings > Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V	2.80 V
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset	HOCO oscillation is disabled after reset
MPU > Enable or disable PC Region 0	• Enabled • Disabled	Disabled
MPU > PC0 Start	Value must be an integer between 0 and 0xFFFFFFFC	0xFFFFFFFC
MPU > PC0 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF	0xFFFFFFFF
MPU > Enable or disable PC Region 1	• Enabled • Disabled	Disabled
MPU > PC1 Start	Value must be an integer between 0 and 0xFFFFFFFC	0xFFFFFFFC
MPU > PC1 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF	0xFFFFFFFF

MPU > Enable or disable Memory Region 0	• Enabled • Disabled	Disabled
MPU > Memory Region 0 Start	Value must be an integer between 0 and 0x00FFFFFC	0x00FFFFFC
MPU > Memory Region 0 End	Value must be an integer between 0x00000003 and 0x00FFFFFF	0x00FFFFFF
MPU > Enable or disable Memory Region 1	• Enabled • Disabled	Disabled
MPU > Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFFC	0x200FFFFC
MPU > Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFF	0x200FFFFF
MPU > Enable or disable Memory Region 2	• Enabled • Disabled	Disabled
MPU > Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x407FFFFC
MPU > Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x407FFFFF
MPU > Enable or disable Memory Region 3	• Enabled • Disabled	Disabled
MPU > Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x400DFFFC
MPU > Memory Region 3 End	Value must be an integer between 0x400C0003 and	0x400DFFFF

	0x400DFFFF or between 0x40100003 and 0x407FFFFF		
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup. When FLL is enabled Software Standby and Deep Software Standby modes are not available.
Main Oscillator Wait Time	• 35 cycles • 67 cycles • 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to stabilize. Drive capability automatic switching function is by default disabled.
ID Code Mode	• Unlocked (Ignore ID) • Locked with All Erase support • Locked	Unlocked (Ignore ID)	When set to 'Locked with All Erase support', the ID Code must be set in the debugger to read or write data to the MCU, but the All Erase command is still accepted regardless. When set to 'Locked', all erase/download/debug access is disabled unless the ID Code is provided.

ID Code (32 Hex Characters)	Value must be a 32 character long hex string	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	Set the ID Code for locking debug access. This setting is only used when the ID Code Mode is not set to Unlocked.

Enumerations

enum	elc_event_t

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

Note

This list may change based on based on the device.

4.1.2.13 RA6M3

BSP » MCU Board Support Package

Detailed Description

Build Time Configurations for ra6m3_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128	128	

	• 256	
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled
OFS0 register settings > Independent WDT > Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) • Stop counting when in Sleep, Snooze mode, or Software Standby	Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128

OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset
OFS1 register settings > Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V	2.80 V
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset	HOCO oscillation is disabled after reset
MPU > Enable or disable PC Region 0	• Enabled • Disabled	Disabled
MPU > PC0 Start	Value must be an integer between 0 and 0xFFFFFFFC	0xFFFFFFFC
MPU > PC0 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF	0xFFFFFFFF
MPU > Enable or	• Enabled	Disabled

disable PC Region 1	• Disabled	
MPU > PC1 Start	Value must be an integer between 0 and 0xFFFFFFFC	0xFFFFFFFC
MPU > PC1 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF	0xFFFFFFFF
MPU > Enable or disable Memory Region 0	• Enabled • Disabled	Disabled
MPU > Memory Region 0 Start	Value must be an integer between 0 and 0x00FFFFFC	0x00FFFFFC
MPU > Memory Region 0 End	Value must be an integer between 0x00000003 and 0x00FFFFFF	0x00FFFFFF
MPU > Enable or disable Memory Region 1	• Enabled • Disabled	Disabled
MPU > Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFFC	0x200FFFFC
MPU > Memory Region 1 End	Value must be an integer between 0x1FF00003 and 0x200FFFFF	0x200FFFFF
MPU > Enable or disable Memory Region 2	• Enabled • Disabled	Disabled
MPU > Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x407FFFFC
MPU > Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x407FFFFF
MPU > Enable or disable Memory Region 3	• Enabled • Disabled	Disabled

MPU > Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x400DFFFC	
MPU > Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x400DFFFF	
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup. When FLL is enabled Software Standby and Deep Software Standby modes are not available.
Main Oscillator Wait Time	• 35 cycles • 67 cycles • 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to stabilize. Drive capability automatic switching function is by default disabled.
ID Code Mode	• Unlocked (Ignore ID) • Locked with All Erase support	Unlocked (Ignore ID)	When set to 'Locked with All Erase support', the ID Code must be set in the debugger to

	• Locked		read or write data to the MCU, but the All Erase command is still accepted regardless. When set to 'Locked', all erase/download/debug access is disabled unless the ID Code is provided.
ID Code (32 Hex Characters)	Value must be a 32 character long hex string	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	Set the ID Code for locking debug access. This setting is only used when the ID Code Mode is not set to Unlocked.

Enumerations

enum	elc_event_t

Enumeration Type Documentation

◆ **elc_event_t**

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

Note

This list may change based on based on the device.

4.1.2.14 RA6M4

BSP » MCU Board Support Package

Build Time Configurations for ra6m4_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
Security > Exceptions > Exception Response	• Non-Maskable Interrupt • Reset	Non-Maskable Interrupt	Configure the result of a TrustZone Filter exception. This exception is generated when the TrustZone Filter detects access to a protected region. This setting is only

			valid when building projects with TrustZone.
Security > Exceptions > BusFault, HardFault, and NMI Target	• Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception. This setting is only valid when building projects with TrustZone.
Security > Exceptions > Prioritize Secure Exceptions	• Enabled • Disabled	Disabled	Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > SRAM Protection	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAMPRCR is write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > SRAM ECC	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAM ECC registers are write accessible for the Non-secure application. This setting is only

			valid when building projects with TrustZone.
Security > SRAM Accessibility > Standby RAM	• Regions 7-0 are all Secure. • Region 7 is Non-secure. Regions 6-0 are Secure. • Regions 7-6 are Non-secure. Regions 5-0 are Secure. • Regions 7-5 are Non-secure. Regions 4-0 are Secure. • Regions 7-4 are Non-secure. Regions 3-0 are Secure. • Regions 7-3 are Non-secure. Regions 2-0 are Secure. • Regions 7-2 are Non-secure. Regions 1-0 are Secure. • Regions 7-1 are Non-secure. Region 0 is Secure. • Regions 7-0 are all Non-secure.	config.bsp.fsp.tz.stbramsar.both	Defines whether Standby RAM registers are accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > BUS Accessibility > Bus Security Attribution Register A	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > BUS Accessibility > Bus Security Attribution Register B	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application. This setting is only

			valid when building projects with TrustZone.
Security > System Reset Request Accessibility	• Both Secure and Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use. This setting is only valid when building projects with TrustZone.
Security > Cache Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Cache registers are write accessible for the Non-secure application. This setting is only valid when building protjects with TrustZone.
Security > System Reset Status Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > Battery Backup Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected. This setting is only valid when building projects with TrustZone.
OFS0 register settings	• IWDT is	IWDT is Disabled	

> Independent WDT > Start Mode	Disabled • IWDT is automatically activated after a reset (Autostart mode)	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled
OFS0 register settings > Independent WDT > Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) • Stop counting when in Sleep, Snooze mode, or Software Standby	Stop counting when in Sleep, Snooze mode, or Software Standby
OFS0 register settings > WDT > Start Mode	• Automatically activate WDT	Stop WDT after a reset (register-start mode)

Select	after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset
OFS1 register settings > Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V	2.80 V
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after	HOCO oscillation is disabled after reset

	reset • HOCO oscillation is disabled after reset		
Block Protection Settings (BPS) > BPS0	Refer to the RA Configuration tool for available options.	0U	Configure Block Protection Register 0
Block Protection Settings (BPS) > BPS1	• Flash Block 32 • Flash Block 33 • Flash Block 34 • Flash Block 35 • Flash Block 36 • Flash Block 37	0U	Configure Block Protection Register 1
Block Protection Settings (BPS) > BPS2	Refer to the RA Configuration tool for available options.	0U	Configure Block Protection Register 2
Permanent Block Protection Settings (PBPS) > PBPS0	Refer to the RA Configuration tool for available options.	0U	Configure Permanent Block Protection Register 0
Permanent Block Protection Settings (PBPS) > PBPS1	• Flash Block 32 • Flash Block 33 • Flash Block 34 • Flash Block 35 • Flash Block 36 • Flash Block 37	0U	Configure Permanent Block Protection Register 1
Permanent Block Protection Settings (PBPS) > PBPS2	Refer to the RA Configuration tool for available options.	0U	Configure Permanent Block Protection Register 2
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup.

			When FLL is enabled Software Standby and Deep Software Standby modes are not available.
Startup C-Cache Line Size	• 32 Bytes • 64 Bytes	32 Bytes	Set the C-Cache line size configured during startup.
Dual Bank Mode	• Enabled • Disabled	Disabled	Enabling dual bank mode splits the flash into two banks that can be swapped by programming the BANKSEL non-volatile register. When enabled, one bank will start at address 0x0 and the other will start at 0x200000. Each bank contains exactly half the capacity of the entire code flash. When Dual Bank mode is enabled, Startup Program Protection and Block Swap functions cannot be used.
Main Oscillator Wait Time	• 3 cycles • 35 cycles • 67 cycles • 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to stabilize.

4.1.2.15 RA6M5

BSP » MCU Board Support Package

Build Time Configurations for ra6m5_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
Security > Exceptions > Exception Response	• Non-Maskable Interrupt	Non-Maskable Interrupt	Configure the result of a TrustZone Filter

	• Reset		exception. This exception is generated when the TrustZone Filter detects access to a protected region. This setting is only valid when building projects with TrustZone.
Security > Exceptions > BusFault, HardFault, and NMI Target	• Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception. This setting is only valid when building projects with TrustZone.
Security > Exceptions > Prioritize Secure Exceptions	• Enabled • Disabled	Disabled	Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the prioritization of non-secure interrupts is correct. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > SRAM Protection	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAMPRCR is write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.

Security > SRAM Accessibility > SRAM ECC	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAM ECC registers are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > Standby RAM	• Regions 7-0 are all Secure. • Region 7 is Non-secure. Regions 6-0 are Secure. • Regions 7-6 are Non-secure. Regions 5-0 are Secure. • Regions 7-5 are Non-secure. Regions 4-0 are Secure. • Regions 7-4 are Non-secure. Regions 3-0 are Secure. • Regions 7-3 are Non-secure. Regions 2-0 are Secure. • Regions 7-2 are Non-secure. Regions 1-0 are Secure. • Regions 7-1 are Non-secure. Region 0 is Secure. • Regions 7-0 are all Non-secure.	config.bsp.fsp.tz.stbramsar.both	Defines whether Standby RAM registers are accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > BUS Accessibility > Bus Security Attribution Register A	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > BUS Accessibility > Bus	• Both Secure and Non-Secure	Both Secure and Non-Secure State	Defines whether the Bus and DMAC/DTC

Security Attribution Register B	State • Secure State		Error Clear Registers are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > System Reset Request Accessibility	• Both Secure and Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use. This setting is only valid when building projects with TrustZone.
Security > Cache Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Cache registers are write accessible for the Non-secure application. This setting is only valid when building protjects with TrustZone.
Security > System Reset Status Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > Battery Backup Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the battery backup registers are accessible for the Non-secure application. If Secure State is selected, all battery backup registers are read only except for VBTBKRn registers which are both read and write protected.

			This setting is only valid when building projects with TrustZone.
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128	
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)	
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)	
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled	
OFS0 register settings > Independent WDT > Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) • Stop counting	Stop counting when in Sleep, Snooze mode, or Software Standby	

	when in Sleep, Snooze mode, or Software Standby	
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset

OFS1 register settings > Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V	2.80 V	
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset	HOCO oscillation is disabled after reset	
Block Protection Settings (BPS) > BPS0	Refer to the RA Configuration tool for available options.	0U	Configure Block Protection Register 0
Block Protection Settings (BPS) > BPS1	• Flash Block 32 • Flash Block 33 • Flash Block 34 • Flash Block 35 • Flash Block 36 • Flash Block 37	0U	Configure Block Protection Register 1
Block Protection Settings (BPS) > BPS2	Refer to the RA Configuration tool for available options.	0U	Configure Block Protection Register 2
Permanent Block Protection Settings (PBPS) > PBPS0	Refer to the RA Configuration tool for available options.	0U	Configure Permanent Block Protection Register 0
Permanent Block Protection Settings (PBPS) > PBPS1	• Flash Block 32 • Flash Block 33 • Flash Block 34 • Flash Block 35 • Flash Block 36 • Flash Block 37	0U	Configure Permanent Block Protection Register 1
Permanent Block Protection Settings (PBPS) > PBPS2	Refer to the RA Configuration tool for available options.	0U	Configure Permanent Block Protection Register 2
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP

			will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup. When FLL is enabled Software Standby and Deep Software Standby modes are not available.
Startup C-Cache Line Size	• 32 Bytes • 64 Bytes	32 Bytes	Set the C-Cache line size configured during startup.
Dual Bank Mode	• Enabled • Disabled	Disabled	Enabling dual bank mode splits the flash into two banks that can be swapped by programming the BANKSEL non-volatile register. When enabled, one bank will start at address 0x0 and the other will start at 0x200000. Each bank contains exactly half the capacity of the entire code flash. When Dual Bank mode is enabled, Startup Program Protection and Block Swap functions cannot be used.
Main Oscillator Wait Time	• 3 cycles • 35 cycles • 67 cycles • 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to stabilize.

4.1.2.16 RA6T1

BSP » MCU Board Support Package

Detailed Description

Build Time Configurations for ra6t1_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128	
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)	
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)	
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled	
OFS0 register settings > Independent WDT > Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) •	Stop counting when in Sleep, Snooze mode, or Software Standby	

	• Stop counting when in Sleep, Snooze mode, or Software Standby	
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset

OFS1 register settings > Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V	2.80 V
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset	HOCO oscillation is disabled after reset
MPU > Enable or disable PC Region 0	• Enabled • Disabled	Disabled
MPU > PC0 Start	Value must be an integer between 0 and 0xFFFFFFFC	0xFFFFFFFC
MPU > PC0 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF	0xFFFFFFFF
MPU > Enable or disable PC Region 1	• Enabled • Disabled	Disabled
MPU > PC1 Start	Value must be an integer between 0 and 0xFFFFFFFC	0xFFFFFFFC
MPU > PC1 End	Value must be an integer between 0x00000003 and 0xFFFFFFFF	0xFFFFFFFF
MPU > Enable or disable Memory Region 0	• Enabled • Disabled	Disabled
MPU > Memory Region 0 Start	Value must be an integer between 0 and 0x00FFFFFC	0x00FFFFFC
MPU > Memory Region 0 End	Value must be an integer between 0x00000003 and 0x00FFFFFF	0x00FFFFFF
MPU > Enable or disable Memory Region 1	• Enabled • Disabled	Disabled
MPU > Memory Region 1 Start	Value must be an integer between 0x1FF00000 and 0x200FFFFC	0x200FFFFC
MPU > Memory Region	Value must be an	0x200FFFFF

1 End	integer between 0x1FF00003 and 0x200FFFFF		
MPU > Enable or disable Memory Region 2	• Enabled • Disabled	Disabled	
MPU > Memory Region 2 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x407FFFFC	
MPU > Memory Region 2 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x407FFFFF	
MPU > Enable or disable Memory Region 3	• Enabled • Disabled	Disabled	
MPU > Memory Region 3 Start	Value must be an integer between 0x400C0000 and 0x400DFFFC or between 0x40100000 and 0x407FFFFC	0x400DFFFC	
MPU > Memory Region 3 End	Value must be an integer between 0x400C0003 and 0x400DFFFF or between 0x40100003 and 0x407FFFFF	0x400DFFFF	
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL

			Stabilization Time when setting up clocks at startup. When FLL is enabled Software Standby and Deep Software Standby modes are not available.
Main Oscillator Wait Time	• 35 cycles • 67 cycles • 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to stabilize. Drive capability automatic switching function is by default disabled.
ID Code Mode	• Unlocked (Ignore ID) • Locked with All Erase support • Locked	Unlocked (Ignore ID)	When set to 'Locked with All Erase support', the ID Code must be set in the debugger to read or write data to the MCU, but the All Erase command is still accepted regardless. When set to 'Locked', all erase/download/debug access is disabled unless the ID Code is provided.
ID Code (32 Hex Characters)	Value must be a 32 character long hex string	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	Set the ID Code for locking debug access. This setting is only used when the ID Code Mode is not set to Unlocked.

Enumerations

enum	elc_event_t

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

Note

This list may change based on based on the device.

4.1.2.17 RA6T2

BSP » MCU Board Support Package

Detailed Description

Build Time Configurations for ra6t2_fsp

The following build time configurations are defined in fsp_cfg/bsp/bsp_mcu_family_cfg.h:

Configuration	Options	Default	Description
Security > Exceptions > Exception Response	• Non-Maskable Interrupt • Reset	Non-Maskable Interrupt	Configure the result of a TrustZone Filter exception. This exception is generated when a the TrustZone Filter detects access to a protected region. This setting is only valid when building projects with TrustZone.
Security > Exceptions > BusFault, HardFault, and NMI Target	• Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit BFHFNMINS. Defines whether BusFault and NMI exceptions are Non-secure, and whether exceptions target the Non-secure HardFault exception. This setting is only valid when building projects with TrustZone.
Security > Exceptions > Prioritize Secure Exceptions	• Enabled • Disabled	Disabled	Value for SCB->AIRCR register bit PRIS. When enabled, all Non-secure interrupt priorities are automatically demoted by right shifting their priority by one then setting the most significant bit. As there is effectively one less bit care must be taken to ensure the

			prioritization of non-secure interrupts is correct. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > SRAM Protection	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAMPRCR is write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > SRAM ECC	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether SRAM ECC registers are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > SRAM Accessibility > Standby RAM	• Regions 7-0 are all Secure. • Region 7 is Non-secure. Regions 6-0 are Secure. • Regions 7-6 are Non-secure. Regions 5-0 are Secure. • Regions 7-5 are Non-secure. Regions 4-0 are Secure. • Regions 7-4 are Non-secure. Regions 3-0 are Secure. • Regions 7-3 are Non-secure. Regions 2-0 are Secure. • Regions 7-2 are Non-secure. Regions 1-0 are Secure. • Regions 7-1 are Non-secure.	config.bsp.fsp.tz.stbramsar.both	Defines whether Standby RAM registers are accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.

	Region 0 is Secure. • Regions 7-0 are all Non-secure.		
Security > BUS Accessibility > Bus Security Attribution Register A	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Slave Bus Control Registers (BUSSCNT<slave>) are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > BUS Accessibility > Bus Security Attribution Register B	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Bus and DMAC/DTC Error Clear Registers are write accessible for the Non-secure application. This setting is only valid when building projects with TrustZone.
Security > System Reset Request Accessibility	• Both Secure and Non-Secure State • Secure State	Secure State	Value for SCB->AIRCR register bit SYSRESETREQS. Defines whether the SYSRESETREQ bit is functional for Non-secure use. This setting is only valid when building projects with TrustZone.
Security > Cache Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the Cache registers are write accessible for the Non-secure application. This setting is only valid when building protjects with TrustZone.
Security > System Reset Status Accessibility	• Both Secure and Non-Secure State • Secure State	Both Secure and Non-Secure State	Defines whether the reset status registers (RSTSRn) can be cleared from the Non-secure application.

			This setting is only valid when building projects with TrustZone.
OFS0 register settings > Independent WDT > Start Mode	• IWDT is Disabled • IWDT is automatically activated after a reset (Autostart mode)	IWDT is Disabled	
OFS0 register settings > Independent WDT > Timeout Period	• 128 cycles • 512 cycles • 1024 cycles • 2048 cycles	2048 cycles	
OFS0 register settings > Independent WDT > Dedicated Clock Frequency Divisor	• 1 • 16 • 32 • 64 • 128 • 256	128	
OFS0 register settings > Independent WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)	
OFS0 register settings > Independent WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)	
OFS0 register settings > Independent WDT > Reset Interrupt Request Select	• NMI request or interrupt request is enabled • Reset is enabled	Reset is enabled	
OFS0 register settings > Independent WDT > Stop Control	• Counting continues (Note: Device will not enter Deep Standby Mode when selected. Device will enter Software Standby Mode) • Stop counting	Stop counting when in Sleep, Snooze mode, or Software Standby	

	when in Sleep, Snooze mode, or Software Standby	
OFS0 register settings > WDT > Start Mode Select	• Automatically activate WDT after a reset (auto-start mode) • Stop WDT after a reset (register-start mode)	Stop WDT after a reset (register-start mode)
OFS0 register settings > WDT > Timeout Period	• 1024 cycles • 4096 cycles • 8192 cycles • 16384 cycles	16384 cycles
OFS0 register settings > WDT > Clock Frequency Division Ratio	• 4 • 64 • 128 • 512 • 2048 • 8192	128
OFS0 register settings > WDT > Window End Position	• 75% • 50% • 25% • 0% (no window end position)	0% (no window end position)
OFS0 register settings > WDT > Window Start Position	• 25% • 50% • 75% • 100% (no window start position)	100% (no window start position)
OFS0 register settings > WDT > Reset Interrupt Request	• NMI • Reset	Reset
OFS0 register settings > WDT > Stop Control	• Counting continues • Stop counting when entering Sleep mode	Stop counting when entering Sleep mode
OFS1 register settings > Voltage Detection 0 Circuit Start	• Voltage monitor 0 reset is enabled after reset • Voltage monitor 0 reset is disabled after reset	Voltage monitor 0 reset is disabled after reset

OFS1 register settings > Voltage Detection 0 Level	• 2.94 V • 2.87 V • 2.80 V	2.80 V	
OFS1 register settings > HOCO Oscillation Enable	• HOCO oscillation is enabled after reset • HOCO oscillation is disabled after reset	HOCO oscillation is disabled after reset	
OFS1 register settings > PGA0 Pseudo-Differential Input Enable	• Disabled (single-ended input) after reset • Enabled after reset	Disabled (single-ended input) after reset	
OFS1 register settings > PGA1 Pseudo-Differential Input Enable	• Disabled (single-ended input) after reset • Enabled after reset	Disabled (single-ended input) after reset	
OFS1 register settings > PGA2 Pseudo-Differential Input Enable	• Disabled (single-ended input) after reset • Enabled after reset	Disabled (single-ended input) after reset	
OFS1 register settings > PGA3 Pseudo-Differential Input Enable	• Disabled (single-ended input) after reset • Enabled after reset	Disabled (single-ended input) after reset	
Block Protection Settings (BPS) > BPS0	Refer to the RA Configuration tool for available options.	0U	Configure Block Protection Register 0
Permanent Block Protection Settings (PBPS) > PBPS0	Refer to the RA Configuration tool for available options.	0U	Configure Permanent Block Protection Register 0
Clocks > HOCO FLL Function	• Enabled • Disabled	Disabled	Setting this option to Enabled improves HOCO accuracy significantly by using the subclock, but incurs certain restrictions. The FLL function requires the subclock oscillator to be running and stabilized. When enabled and running

			the PLL or system clock from HOCO, the BSP will wait for both the Subclock Stabilization Time as well as the FLL Stabilization Time when setting up clocks at startup. When FLL is enabled Software Standby and Deep Software Standby modes are not available.
Startup C-Cache Line Size	• 32 Bytes • 64 Bytes	32 Bytes	Set the C-Cache line size configured during startup.
TFU Mathlib	• Disabled • Enabled	Enabled	If enabled, trigonometric library functions sinf, cosf, atan2f, and hypotf are replaced with hardware accelerated TFU functions. Disable this if reentrant access to these functions is required.
Main Oscillator Wait Time	• 3 cycles • 35 cycles • 67 cycles • 131 cycles • 259 cycles • 547 cycles • 1059 cycles • 2147 cycles • 4291 cycles • 8163 cycles	8163 cycles	Number of cycles to wait for the main oscillator clock to stabilize.

Enumerations

enum	elc_event_t

Enumeration Type Documentation

◆ elc_event_t

enum elc_event_t

Sources of event signals to be linked to other peripherals or the CPU

Note

This list may change based on based on the device.

4.1.3 BSP I/O access

BSP

Functions

__STATIC_INLINE uint32_t	R_BSP_PinRead (bsp_io_port_pin_t pin)
__STATIC_INLINE void	R_BSP_PinWrite (bsp_io_port_pin_t pin, bsp_io_level_t level)
__STATIC_INLINE void	R_BSP_PinCfg (bsp_io_port_pin_t pin, uint32_t cfg)
__STATIC_INLINE void	R_BSP_PinAccessEnable (void)
__STATIC_INLINE void	R_BSP_PinAccessDisable (void)

Detailed Description

This module provides basic read/write access to port pins.

Enumerations

enum	bsp_io_level_t
enum	bsp_io_direction_t
enum	bsp_io_port_t
enum	bsp_io_port_pin_t

Enumeration Type Documentation

◆ bsp_io_level_t

enum bsp_io_level_t

Levels that can be set and read for individual pins

Enumerator	
BSP_IO_LEVEL_LOW	Low.
BSP_IO_LEVEL_HIGH	High.

◆ bsp_io_direction_t

enum bsp_io_direction_t	
Direction of individual pins	
Enumerator	
BSP_IO_DIRECTION_INPUT	Input.
BSP_IO_DIRECTION_OUTPUT	Output.

◆ bsp_io_port_t

enum bsp_io_port_t	
Superset list of all possible IO ports.	
Enumerator	
BSP_IO_PORT_00	IO port 0.
BSP_IO_PORT_01	IO port 1.
BSP_IO_PORT_02	IO port 2.
BSP_IO_PORT_03	IO port 3.
BSP_IO_PORT_04	IO port 4.
BSP_IO_PORT_05	IO port 5.
BSP_IO_PORT_06	IO port 6.
BSP_IO_PORT_07	IO port 7.
BSP_IO_PORT_08	IO port 8.
BSP_IO_PORT_09	IO port 9.
BSP_IO_PORT_10	IO port 10.
BSP_IO_PORT_11	IO port 11.
BSP_IO_PORT_12	IO port 12.
BSP_IO_PORT_13	IO port 13.
BSP_IO_PORT_14	IO port 14.

◆ bsp_io_port_pin_t

enum bsp_io_port_pin_t	
Superset list of all possible IO port pins.	
Enumerator	
BSP_IO_PORT_00_PIN_00	IO port 0 pin 0.
BSP_IO_PORT_00_PIN_01	IO port 0 pin 1.

BSP_IO_PORT_00_PIN_02	IO port 0 pin 2.
BSP_IO_PORT_00_PIN_03	IO port 0 pin 3.
BSP_IO_PORT_00_PIN_04	IO port 0 pin 4.
BSP_IO_PORT_00_PIN_05	IO port 0 pin 5.
BSP_IO_PORT_00_PIN_06	IO port 0 pin 6.
BSP_IO_PORT_00_PIN_07	IO port 0 pin 7.
BSP_IO_PORT_00_PIN_08	IO port 0 pin 8.
BSP_IO_PORT_00_PIN_09	IO port 0 pin 9.
BSP_IO_PORT_00_PIN_10	IO port 0 pin 10.
BSP_IO_PORT_00_PIN_11	IO port 0 pin 11.
BSP_IO_PORT_00_PIN_12	IO port 0 pin 12.
BSP_IO_PORT_00_PIN_13	IO port 0 pin 13.
BSP_IO_PORT_00_PIN_14	IO port 0 pin 14.
BSP_IO_PORT_00_PIN_15	IO port 0 pin 15.
BSP_IO_PORT_01_PIN_00	IO port 1 pin 0.
BSP_IO_PORT_01_PIN_01	IO port 1 pin 1.
BSP_IO_PORT_01_PIN_02	IO port 1 pin 2.
BSP_IO_PORT_01_PIN_03	IO port 1 pin 3.
BSP_IO_PORT_01_PIN_04	IO port 1 pin 4.
BSP_IO_PORT_01_PIN_05	IO port 1 pin 5.
BSP_IO_PORT_01_PIN_06	IO port 1 pin 6.
BSP_IO_PORT_01_PIN_07	IO port 1 pin 7.
BSP_IO_PORT_01_PIN_08	IO port 1 pin 8.
BSP_IO_PORT_01_PIN_09	IO port 1 pin 9.

BSP_IO_PORT_01_PIN_10	IO port 1 pin 10.
BSP_IO_PORT_01_PIN_11	IO port 1 pin 11.
BSP_IO_PORT_01_PIN_12	IO port 1 pin 12.
BSP_IO_PORT_01_PIN_13	IO port 1 pin 13.
BSP_IO_PORT_01_PIN_14	IO port 1 pin 14.
BSP_IO_PORT_01_PIN_15	IO port 1 pin 15.
BSP_IO_PORT_02_PIN_00	IO port 2 pin 0.
BSP_IO_PORT_02_PIN_01	IO port 2 pin 1.
BSP_IO_PORT_02_PIN_02	IO port 2 pin 2.
BSP_IO_PORT_02_PIN_03	IO port 2 pin 3.
BSP_IO_PORT_02_PIN_04	IO port 2 pin 4.
BSP_IO_PORT_02_PIN_05	IO port 2 pin 5.
BSP_IO_PORT_02_PIN_06	IO port 2 pin 6.
BSP_IO_PORT_02_PIN_07	IO port 2 pin 7.
BSP_IO_PORT_02_PIN_08	IO port 2 pin 8.
BSP_IO_PORT_02_PIN_09	IO port 2 pin 9.
BSP_IO_PORT_02_PIN_10	IO port 2 pin 10.
BSP_IO_PORT_02_PIN_11	IO port 2 pin 11.
BSP_IO_PORT_02_PIN_12	IO port 2 pin 12.
BSP_IO_PORT_02_PIN_13	IO port 2 pin 13.
BSP_IO_PORT_02_PIN_14	IO port 2 pin 14.
BSP_IO_PORT_02_PIN_15	IO port 2 pin 15.
BSP_IO_PORT_03_PIN_00	IO port 3 pin 0.
BSP_IO_PORT_03_PIN_01	IO port 3 pin 1.

BSP_IO_PORT_03_PIN_02	IO port 3 pin 2.
BSP_IO_PORT_03_PIN_03	IO port 3 pin 3.
BSP_IO_PORT_03_PIN_04	IO port 3 pin 4.
BSP_IO_PORT_03_PIN_05	IO port 3 pin 5.
BSP_IO_PORT_03_PIN_06	IO port 3 pin 6.
BSP_IO_PORT_03_PIN_07	IO port 3 pin 7.
BSP_IO_PORT_03_PIN_08	IO port 3 pin 8.
BSP_IO_PORT_03_PIN_09	IO port 3 pin 9.
BSP_IO_PORT_03_PIN_10	IO port 3 pin 10.
BSP_IO_PORT_03_PIN_11	IO port 3 pin 11.
BSP_IO_PORT_03_PIN_12	IO port 3 pin 12.
BSP_IO_PORT_03_PIN_13	IO port 3 pin 13.
BSP_IO_PORT_03_PIN_14	IO port 3 pin 14.
BSP_IO_PORT_03_PIN_15	IO port 3 pin 15.
BSP_IO_PORT_04_PIN_00	IO port 4 pin 0.
BSP_IO_PORT_04_PIN_01	IO port 4 pin 1.
BSP_IO_PORT_04_PIN_02	IO port 4 pin 2.
BSP_IO_PORT_04_PIN_03	IO port 4 pin 3.
BSP_IO_PORT_04_PIN_04	IO port 4 pin 4.
BSP_IO_PORT_04_PIN_05	IO port 4 pin 5.
BSP_IO_PORT_04_PIN_06	IO port 4 pin 6.
BSP_IO_PORT_04_PIN_07	IO port 4 pin 7.
BSP_IO_PORT_04_PIN_08	IO port 4 pin 8.
BSP_IO_PORT_04_PIN_09	IO port 4 pin 9.

BSP_IO_PORT_04_PIN_10	IO port 4 pin 10.
BSP_IO_PORT_04_PIN_11	IO port 4 pin 11.
BSP_IO_PORT_04_PIN_12	IO port 4 pin 12.
BSP_IO_PORT_04_PIN_13	IO port 4 pin 13.
BSP_IO_PORT_04_PIN_14	IO port 4 pin 14.
BSP_IO_PORT_04_PIN_15	IO port 4 pin 15.
BSP_IO_PORT_05_PIN_00	IO port 5 pin 0.
BSP_IO_PORT_05_PIN_01	IO port 5 pin 1.
BSP_IO_PORT_05_PIN_02	IO port 5 pin 2.
BSP_IO_PORT_05_PIN_03	IO port 5 pin 3.
BSP_IO_PORT_05_PIN_04	IO port 5 pin 4.
BSP_IO_PORT_05_PIN_05	IO port 5 pin 5.
BSP_IO_PORT_05_PIN_06	IO port 5 pin 6.
BSP_IO_PORT_05_PIN_07	IO port 5 pin 7.
BSP_IO_PORT_05_PIN_08	IO port 5 pin 8.
BSP_IO_PORT_05_PIN_09	IO port 5 pin 9.
BSP_IO_PORT_05_PIN_10	IO port 5 pin 10.
BSP_IO_PORT_05_PIN_11	IO port 5 pin 11.
BSP_IO_PORT_05_PIN_12	IO port 5 pin 12.
BSP_IO_PORT_05_PIN_13	IO port 5 pin 13.
BSP_IO_PORT_05_PIN_14	IO port 5 pin 14.
BSP_IO_PORT_05_PIN_15	IO port 5 pin 15.
BSP_IO_PORT_06_PIN_00	IO port 6 pin 0.
BSP_IO_PORT_06_PIN_01	IO port 6 pin 1.

BSP_IO_PORT_06_PIN_02	IO port 6 pin 2.
BSP_IO_PORT_06_PIN_03	IO port 6 pin 3.
BSP_IO_PORT_06_PIN_04	IO port 6 pin 4.
BSP_IO_PORT_06_PIN_05	IO port 6 pin 5.
BSP_IO_PORT_06_PIN_06	IO port 6 pin 6.
BSP_IO_PORT_06_PIN_07	IO port 6 pin 7.
BSP_IO_PORT_06_PIN_08	IO port 6 pin 8.
BSP_IO_PORT_06_PIN_09	IO port 6 pin 9.
BSP_IO_PORT_06_PIN_10	IO port 6 pin 10.
BSP_IO_PORT_06_PIN_11	IO port 6 pin 11.
BSP_IO_PORT_06_PIN_12	IO port 6 pin 12.
BSP_IO_PORT_06_PIN_13	IO port 6 pin 13.
BSP_IO_PORT_06_PIN_14	IO port 6 pin 14.
BSP_IO_PORT_06_PIN_15	IO port 6 pin 15.
BSP_IO_PORT_07_PIN_00	IO port 7 pin 0.
BSP_IO_PORT_07_PIN_01	IO port 7 pin 1.
BSP_IO_PORT_07_PIN_02	IO port 7 pin 2.
BSP_IO_PORT_07_PIN_03	IO port 7 pin 3.
BSP_IO_PORT_07_PIN_04	IO port 7 pin 4.
BSP_IO_PORT_07_PIN_05	IO port 7 pin 5.
BSP_IO_PORT_07_PIN_06	IO port 7 pin 6.
BSP_IO_PORT_07_PIN_07	IO port 7 pin 7.
BSP_IO_PORT_07_PIN_08	IO port 7 pin 8.
BSP_IO_PORT_07_PIN_09	IO port 7 pin 9.

BSP_IO_PORT_07_PIN_10	IO port 7 pin 10.
BSP_IO_PORT_07_PIN_11	IO port 7 pin 11.
BSP_IO_PORT_07_PIN_12	IO port 7 pin 12.
BSP_IO_PORT_07_PIN_13	IO port 7 pin 13.
BSP_IO_PORT_07_PIN_14	IO port 7 pin 14.
BSP_IO_PORT_07_PIN_15	IO port 7 pin 15.
BSP_IO_PORT_08_PIN_00	IO port 8 pin 0.
BSP_IO_PORT_08_PIN_01	IO port 8 pin 1.
BSP_IO_PORT_08_PIN_02	IO port 8 pin 2.
BSP_IO_PORT_08_PIN_03	IO port 8 pin 3.
BSP_IO_PORT_08_PIN_04	IO port 8 pin 4.
BSP_IO_PORT_08_PIN_05	IO port 8 pin 5.
BSP_IO_PORT_08_PIN_06	IO port 8 pin 6.
BSP_IO_PORT_08_PIN_07	IO port 8 pin 7.
BSP_IO_PORT_08_PIN_08	IO port 8 pin 8.
BSP_IO_PORT_08_PIN_09	IO port 8 pin 9.
BSP_IO_PORT_08_PIN_10	IO port 8 pin 10.
BSP_IO_PORT_08_PIN_11	IO port 8 pin 11.
BSP_IO_PORT_08_PIN_12	IO port 8 pin 12.
BSP_IO_PORT_08_PIN_13	IO port 8 pin 13.
BSP_IO_PORT_08_PIN_14	IO port 8 pin 14.
BSP_IO_PORT_08_PIN_15	IO port 8 pin 15.
BSP_IO_PORT_09_PIN_00	IO port 9 pin 0.
BSP_IO_PORT_09_PIN_01	IO port 9 pin 1.

BSP_IO_PORT_09_PIN_02	IO port 9 pin 2.
BSP_IO_PORT_09_PIN_03	IO port 9 pin 3.
BSP_IO_PORT_09_PIN_04	IO port 9 pin 4.
BSP_IO_PORT_09_PIN_05	IO port 9 pin 5.
BSP_IO_PORT_09_PIN_06	IO port 9 pin 6.
BSP_IO_PORT_09_PIN_07	IO port 9 pin 7.
BSP_IO_PORT_09_PIN_08	IO port 9 pin 8.
BSP_IO_PORT_09_PIN_09	IO port 9 pin 9.
BSP_IO_PORT_09_PIN_10	IO port 9 pin 10.
BSP_IO_PORT_09_PIN_11	IO port 9 pin 11.
BSP_IO_PORT_09_PIN_12	IO port 9 pin 12.
BSP_IO_PORT_09_PIN_13	IO port 9 pin 13.
BSP_IO_PORT_09_PIN_14	IO port 9 pin 14.
BSP_IO_PORT_09_PIN_15	IO port 9 pin 15.
BSP_IO_PORT_10_PIN_00	IO port 10 pin 0.
BSP_IO_PORT_10_PIN_01	IO port 10 pin 1.
BSP_IO_PORT_10_PIN_02	IO port 10 pin 2.
BSP_IO_PORT_10_PIN_03	IO port 10 pin 3.
BSP_IO_PORT_10_PIN_04	IO port 10 pin 4.
BSP_IO_PORT_10_PIN_05	IO port 10 pin 5.
BSP_IO_PORT_10_PIN_06	IO port 10 pin 6.
BSP_IO_PORT_10_PIN_07	IO port 10 pin 7.
BSP_IO_PORT_10_PIN_08	IO port 10 pin 8.
BSP_IO_PORT_10_PIN_09	IO port 10 pin 9.

BSP_IO_PORT_10_PIN_10	IO port 10 pin 10.
BSP_IO_PORT_10_PIN_11	IO port 10 pin 11.
BSP_IO_PORT_10_PIN_12	IO port 10 pin 12.
BSP_IO_PORT_10_PIN_13	IO port 10 pin 13.
BSP_IO_PORT_10_PIN_14	IO port 10 pin 14.
BSP_IO_PORT_10_PIN_15	IO port 10 pin 15.
BSP_IO_PORT_11_PIN_00	IO port 11 pin 0.
BSP_IO_PORT_11_PIN_01	IO port 11 pin 1.
BSP_IO_PORT_11_PIN_02	IO port 11 pin 2.
BSP_IO_PORT_11_PIN_03	IO port 11 pin 3.
BSP_IO_PORT_11_PIN_04	IO port 11 pin 4.
BSP_IO_PORT_11_PIN_05	IO port 11 pin 5.
BSP_IO_PORT_11_PIN_06	IO port 11 pin 6.
BSP_IO_PORT_11_PIN_07	IO port 11 pin 7.
BSP_IO_PORT_11_PIN_08	IO port 11 pin 8.
BSP_IO_PORT_11_PIN_09	IO port 11 pin 9.
BSP_IO_PORT_11_PIN_10	IO port 11 pin 10.
BSP_IO_PORT_11_PIN_11	IO port 11 pin 11.
BSP_IO_PORT_11_PIN_12	IO port 11 pin 12.
BSP_IO_PORT_11_PIN_13	IO port 11 pin 13.
BSP_IO_PORT_11_PIN_14	IO port 11 pin 14.
BSP_IO_PORT_11_PIN_15	IO port 11 pin 15.
BSP_IO_PORT_12_PIN_00	IO port 12 pin 0.
BSP_IO_PORT_12_PIN_01	IO port 12 pin 1.

BSP_IO_PORT_12_PIN_02	IO port 12 pin 2.
BSP_IO_PORT_12_PIN_03	IO port 12 pin 3.
BSP_IO_PORT_12_PIN_04	IO port 12 pin 4.
BSP_IO_PORT_12_PIN_05	IO port 12 pin 5.
BSP_IO_PORT_12_PIN_06	IO port 12 pin 6.
BSP_IO_PORT_12_PIN_07	IO port 12 pin 7.
BSP_IO_PORT_12_PIN_08	IO port 12 pin 8.
BSP_IO_PORT_12_PIN_09	IO port 12 pin 9.
BSP_IO_PORT_12_PIN_10	IO port 12 pin 10.
BSP_IO_PORT_12_PIN_11	IO port 12 pin 11.
BSP_IO_PORT_12_PIN_12	IO port 12 pin 12.
BSP_IO_PORT_12_PIN_13	IO port 12 pin 13.
BSP_IO_PORT_12_PIN_14	IO port 12 pin 14.
BSP_IO_PORT_12_PIN_15	IO port 12 pin 15.
BSP_IO_PORT_13_PIN_00	IO port 13 pin 0.
BSP_IO_PORT_13_PIN_01	IO port 13 pin 1.
BSP_IO_PORT_13_PIN_02	IO port 13 pin 2.
BSP_IO_PORT_13_PIN_03	IO port 13 pin 3.
BSP_IO_PORT_13_PIN_04	IO port 13 pin 4.
BSP_IO_PORT_13_PIN_05	IO port 13 pin 5.
BSP_IO_PORT_13_PIN_06	IO port 13 pin 6.
BSP_IO_PORT_13_PIN_07	IO port 13 pin 7.
BSP_IO_PORT_13_PIN_08	IO port 13 pin 8.
BSP_IO_PORT_13_PIN_09	IO port 13 pin 9.

BSP_IO_PORT_13_PIN_10	IO port 13 pin 10.
BSP_IO_PORT_13_PIN_11	IO port 13 pin 11.
BSP_IO_PORT_13_PIN_12	IO port 13 pin 12.
BSP_IO_PORT_13_PIN_13	IO port 13 pin 13.
BSP_IO_PORT_13_PIN_14	IO port 13 pin 14.
BSP_IO_PORT_13_PIN_15	IO port 13 pin 15.
BSP_IO_PORT_14_PIN_00	IO port 14 pin 0.
BSP_IO_PORT_14_PIN_01	IO port 14 pin 1.
BSP_IO_PORT_14_PIN_02	IO port 14 pin 2.
BSP_IO_PORT_14_PIN_03	IO port 14 pin 3.
BSP_IO_PORT_14_PIN_04	IO port 14 pin 4.
BSP_IO_PORT_14_PIN_05	IO port 14 pin 5.
BSP_IO_PORT_14_PIN_06	IO port 14 pin 6.
BSP_IO_PORT_14_PIN_07	IO port 14 pin 7.
BSP_IO_PORT_14_PIN_08	IO port 14 pin 8.
BSP_IO_PORT_14_PIN_09	IO port 14 pin 9.
BSP_IO_PORT_14_PIN_10	IO port 14 pin 10.
BSP_IO_PORT_14_PIN_11	IO port 14 pin 11.
BSP_IO_PORT_14_PIN_12	IO port 14 pin 12.
BSP_IO_PORT_14_PIN_13	IO port 14 pin 13.
BSP_IO_PORT_14_PIN_14	IO port 14 pin 14.
BSP_IO_PORT_14_PIN_15	IO port 14 pin 15.

Function Documentation

◆ R_BSP_PinRead()

__STATIC_INLINE uint32_t R_BSP_PinRead (bsp_io_port_pin_t *pin*)

Read the current input level of the pin.

Parameters

[in]	pin	The pin

Return values

Current	input level

◆ R_BSP_PinWrite()

__STATIC_INLINE void R_BSP_PinWrite (bsp_io_port_pin_t *pin*, bsp_io_level_t *level*)

Set a pin to output and set the output level to the level provided. If PFS protection is enabled, disable PFS protection using R_BSP_PinAccessEnable() before calling this function.

Parameters

[in]	pin	The pin
[in]	level	The level

◆ R_BSP_PinCfg()

__STATIC_INLINE void R_BSP_PinCfg (bsp_io_port_pin_t *pin*, uint32_t *cfg*)

Configure a pin. If PFS protection is enabled, disable PFS protection using R_BSP_PinAccessEnable() before calling this function.

Parameters

[in]	pin	The pin
[in]	cfg	Configuration for the pin (PmnPFS register setting)

◆ R_BSP_PinAccessEnable()

__STATIC_INLINE void R_BSP_PinAccessEnable (void)

Enable access to the PFS registers. Uses a reference counter to protect against interrupts that could occur via multiple threads or an ISR re-entering this code.

◆ R_BSP_PinAccessDisable()

__STATIC_INLINE void R_BSP_PinAccessDisable (void)
Disable access to the PFS registers. Uses a reference counter to protect against interrupts that could occur via multiple threads or an ISR re-entering this code.

4.2 Modules

Detailed Description

Modules are the smallest unit of software available in the FSP. Each module implements one interface.

For more information on FSP Modules and Interfaces review FSP Modules, FSP Stacks and FSP Interfaces in the FSP Architecutre section of this manual.

Note

Not all modules are available for all MCUs. For more information, see the User's Manual for the specific MCU.

Organization of Module Sections

Each module within FSP has a detailed Users' Guide listed below. Each guide typically includes the following content:

- Functions: A list of all the API functions associated with the module
- Detailed Description: A short description of the module and the peripherals used
- Overview: An operational summary and a list of high level features provided by the module
- Configuration: A description of module specific settings available in the configuration tool including clock and pin configurations
- Usage Notes: Module specific documentation and limitations
- Examples: Illustrative code snippets that help the user better understand API use and operation
- Data Structure and Enumeration: Definitions for data structures, enumerations and similar elements used by the module API
- Function Documentation: Details on each API function, including the function prototype, a function summary, a simple use example, list of return values and links to documentation for any needed parameter definitions

Modules

	High-Speed Analog Comparator (r_acmphs) Driver for the ACMPHS peripheral on RA MCUs. This module implements the Comparator Interface.
	Low-Power Analog Comparator (r_acmplp) Driver for the ACMPLP peripheral on RA MCUs. This module

	implements the Comparator Interface.
	Analog to Digital Converter (r_adc) Driver for the ADC12, ADC14, and ADC16 peripherals on RA MCUs. This module implements the ADC Interface.
	Analog to Digital Converter (r_adc_b) Driver for the ADC_B peripheral on RA MCUs. This module implements the ADC Interface.
	Asynchronous General Purpose Timer (r_agt) Driver for the AGT peripheral on RA MCUs. This module implements the Timer Interface.
	Bluetooth Low Energy Library of Balance Configuration (r_ble) Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface.
	Bluetooth Low Energy Library of Compact Configuration (r_ble) Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface.
	Bluetooth Low Energy Library of Extended Configuration (r_ble) Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface.
	Clock Frequency Accuracy Measurement Circuit (r_cac) Driver for the CAC peripheral on RA MCUs. This module implements the CAC Interface.
	Controller Area Network (r_can) Driver for the CAN peripheral on RA MCUs. This module implements the CAN Interface.
	Controller Area Network - Flexible Data (r_canfd) Driver for the CANFD peripheral on RA MCUs. This module implements the CAN Interface.

	Consumer Electronics Control (r_cec) Driver for the CEC peripheral on RA MCUs. This module implements the CEC Interface.
	Clock Generation Circuit (r_cgc) Driver for the CGC peripheral on RA MCUs. This module implements the CGC Interface.
	Cyclic Redundancy Check (CRC) Calculator (r_crc) Driver for the CRC peripheral on RA MCUs. This module implements the CRC Interface.
	Capacitive Touch Sensing Unit (r_ctsu) This HAL driver supports the Capacitive Touch Sensing Unit (CTSU). It implements the CTSU Interface.
	Digital to Analog Converter (r_dac) Driver for the DAC12 peripheral on RA MCUs. This module implements the DAC Interface.
	Digital to Analog Converter (r_dac8) Driver for the DAC8 peripheral on RA MCUs. This module implements the DAC Interface.
	Direct Memory Access Controller (r_dmac) Driver for the DMAC peripheral on RA MCUs. This module implements the Transfer Interface.
	Data Operation Circuit (r_doc) Driver for the DOC peripheral on RA MCUs. This module implements the DOC Interface.
	D/AVE 2D Port Interface (r_drw) Driver for the DRW peripheral on RA MCUs. This module is a port of D/AVE 2D.
	Data Transfer Controller (r_dtc)

	Driver for the DTC peripheral on RA MCUs. This module implements the Transfer Interface.
	Event Link Controller (r_elc) Driver for the ELC peripheral on RA MCUs. This module implements the ELC Interface.
	Ethernet (r_ether) Driver for the Ethernet peripheral on RA MCUs. This module implements the Ethernet Interface.
	Ethernet PHY (r_ether_phy) The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications that use the ETHERC peripheral. It implements the Ethernet PHY Interface.
	High-Performance Flash Driver (r_flash_hp) Driver for the flash memory on RA high-performance MCUs. This module implements the Flash Interface.
	Low-Power Flash Driver (r_flash_lp) Driver for the flash memory on RA low-power MCUs. This module implements the Flash Interface.
	Graphics LCD Controller (r_glcdc) Driver for the GLCDC peripheral on RA MCUs. This module implements the Display Interface.
	General PWM Timer (r_gpt) Driver for the GPT32 and GPT16 peripherals on RA MCUs. This module implements the Timer Interface.
	General PWM Timer Three-Phase Motor Control Driver (r_gpt_three_phase) Driver for 3-phase motor control using the GPT peripheral on RA MCUs. This module implements the Three-Phase Interface.
	I3C Bus Interface (r_i3c)

	Driver for the I3C peripheral on RA MCUs. This module implements the I3C Interface.
	Interrupt Controller Unit (r_icu) Driver for the ICU peripheral on RA MCUs. This module implements the External IRQ Interface.
	I2C Master on IIC/I3C (r_iic_b_master) I2C Driver for the IIC/I3C peripheral on RA MCUs. This module implements the I2C Master Interface.
	I2C Slave on IIC/I3C (r_iic_b_slave) Driver for the IIC/I3C peripheral on RA MCUs. This module implements the I2C Slave Interface.
	I2C Master on IIC (r_iic_master) Driver for the IIC peripheral on RA MCUs. This module implements the I2C Master Interface.
	I2C Slave on IIC (r_iic_slave) Driver for the IIC peripheral on RA MCUs. This module implements the I2C Slave Interface.
	I/O Ports (r_ioport) Driver for the I/O Ports peripheral on RA MCUs. This module implements the I/O Port Interface.
	Independent Watchdog Timer (r_iwdt) Driver for the IWDT peripheral on RA MCUs. This module implements the WDT Interface.
	JPEG Codec (r_jpeg) Driver for the JPEG peripheral on RA MCUs. This module implements the JPEG Codec Interface.
	Key Interrupt (r_kint) Driver for the KINT peripheral on RA MCUs. This module implements

	the Key Matrix Interface.
	Low Power Modes (r_lpm) Driver for the LPM peripheral on RA MCUs. This module implements the Low Power Modes Interface.
	Low Voltage Detection (r_lvd) Driver for the LVD peripheral on RA MCUs. This module implements the Low Voltage Detection Interface.
	Operational Amplifier (r_opamp) Driver for the OPAMP peripheral on RA MCUs. This module implements the OPAMP Interface.
	Octa Serial Peripheral Interface for Flash and RAM devices (r_ospi) Driver for the OSPI peripheral on RA MCUs. This module implements the SPI Flash Interface.
	Parallel Data Capture (r_pdc) Driver for the PDC peripheral on RA MCUs. This module implements the PDC Interface.
	Port Output Enable for GPT (r_poeg) Driver for the POEG peripheral on RA MCUs. This module implements the POEG Interface.
	Precision Time Protocol (r_ptp) Driver for the PTP peripheral on RA MCUs. This module implements the PTP Interface.
	Quad Serial Peripheral Interface Flash (r_qspi) Driver for the QSPI peripheral on RA MCUs. This module implements the SPI Flash Interface.
	Realtime Clock (r_rtc) Driver for the RTC peripheral on RA MCUs. This module implements the RTC Interface.

	Secure Crypto Engine (r_sce_protected) Driver for the Secure Crypto Engine (SCE9) on RA MCUs.
	Serial Communications Interface (SCI) I2C (r_sci_b_i2c) Driver for the SCI_B peripheral on RA MCUs. This module implements the I2C Master Interface.
	Serial Communications Interface (SCI) SPI (r_sci_b_spi) Driver for the SCI peripheral on RA MCUs. This module implements the SPI Interface.
	Serial Communications Interface (SCI) UART (r_sci_b_uart) Driver for the SCI peripheral on RA MCUs. This module implements the UART Interface.
	Serial Communications Interface (SCI) I2C (r_sci_i2c) Driver for the SCI peripheral on RA MCUs. This module implements the I2C Master Interface.
	Serial Communications Interface (SCI) SPI (r_sci_spi) Driver for the SCI peripheral on RA MCUs. This module implements the SPI Interface.
	Serial Communications Interface (SCI) UART (r_sci_uart) Driver for the SCI peripheral on RA MCUs. This module implements the UART Interface.
	Sigma Delta Analog to Digital Converter (r_sdadc) Driver for the SDADC24 peripheral on RA MCUs. This module implements the ADC Interface.
	SD/MMC Host Interface (r_sdhi) Driver for the SD/MMC Host Interface (SDHI) peripheral on RA MCUs. This module implements the SD/MMC Interface.
	Segment LCD Controller (r_slcdc)

	Driver for the SLCDC peripheral on RA MCUs. This module implements the SLCDC Interface.
	Serial Peripheral Interface (r_spi) Driver for the SPI peripheral on RA MCUs. This module implements the SPI Interface.
	Serial Peripheral Interface (r_spi_b) Driver for the SPI peripheral on RA MCUs. This module implements the SPI Interface.
	Serial Sound Interface (r_ssi) Driver for the SSIE peripheral on RA MCUs. This module implements the I2S Interface.
	USB (r_usb_basic) Driver for the USB peripheral on RA MCUs. This module implements the USB Interface.
	USB Composite Class (r_usb_composite)
	USB Host Communications Device Class Driver (r_usb_hcdc) This module provides a USB Host Communications Device Class (HCDC) driver. It implements the USB HCDC Interface.
	USB Host Human Interface Device Class Driver (r_usb_hhid) This module provides a USB Host Human Interface Device Class Driver (HHID). It implements the USB HHID Interface.
	USB Host Mass Storage Class Driver (r_usb_hmsc) This module provides a USB Host Mass Storage Class (HMSC) driver. It implements the USB HMSC Interface.
	USB Host Vendor Class (r_usb_hvnd)
	USB Peripheral Communications Device Class (r_usb_pcdc) This module provides a USB Peripheral Communications Device Class Driver (PCDC). It implements the USB PCDC Interface.

	USB Peripheral Human Interface Device Class (r_usb_phid) This module is USB Peripheral Human Interface Device Class Driver (PHID). It implements the USB PHID Interface.
	USB Peripheral Mass Storage Class (r_usb_pmsc) This module provides a USB Peripheral Mass Storage Class (PMSC) driver. It implements the USB PMSC Interface.
	USB Peripheral Vendor Class (r_usb_pvnd)
	Watchdog Timer (r_wdt) Driver for the WDT peripheral on RA MCUs. This module implements the WDT Interface.
	ADPCM Decoder (rm_adpcm_decoder) Middleware to implement the ADPCM Audio Decoder. This module implements the ADPCM Decoder Interface.
	Audio Playback with PWM (rm_audio_playback_pwm) Driver for the Audio Playback middleware on RA MCUs. This module implements the AUDIO PLAYBACK Interface.
	AWS PKCS11 PAL (rm_aws_pkcs11_pal) PKCS#11 PAL layer implementation for use by FreeRTOS TLS.
	AWS PKCS11 PAL LITTLEFS (rm_aws_pkcs11_pal_littlefs) PKCS#11 PAL LittleFS layer implementation for use by FreeRTOS TLS.
	Bluetooth Low Energy Abstraction (rm_ble_abs) Middleware for the Bluetooth peripheral on RA MCUs. This module implements the BLE ABS Interface.
	Bluetooth Low Energy Abstraction with RYZ012 (rm_ble_abs_spp) Middleware for the Bluetooth peripheral on RA MCUs. This module implements the BLE ABS Interface.

	SD/MMC Block Media Implementation (rm_block_media_sdmmc) Middleware to implement the block media interface on SD cards. This module implements the Block Media Interface.
	SPI Block Media Implementation (rm_block_media_spi) Middleware to implement the block media interface on SPI flash memory. This module implements the Block Media Interface.
	USB HMSC Block Media Implementation (rm_block_media_usb) Middleware to implement the block media interface on USB mass storage devices. This module implements the Block Media Interface.
	User Block Media Implementation (rm_block_media_user) Middleware that implements a block media interface on the media of your choice. This module implements the Block Media Interface.
	I2C Communicatons Middleware (rm_comms_i2c) Middleware to implement the I2C communications interface. This module implements the Communicatons Middleware Interface.
	SEGGER emWin Port (rm_emwin_port) SEGGER emWin port for RA MCUs.
	Azure RTOS FileX Block Media I/O Driver (rm_filex_block_media) Middleware for the Azure RTOS FileX File System control using Block Media on RA MCUs.
	Azure RTOS FileX LevelX I/O Driver (rm_filex_levelx_nor) Middleware for the Azure RTOS FileX File System control using LevelX NOR on RA MCUs.
	FreeRTOS+FAT Port (rm_freertos_plus_fat) Middleware for the FAT File System control on RA MCUs.
	FreeRTOS Plus TCP (rm_freertos_plus_tcp) Middleware for using TCP on RA MCUs.

	FreeRTOS Port (rm_freertos_port) FreeRTOS port for RA MCUs.
	RTOS Context Management (rm_tz_context) RTOS Context Management for RA MCUs.
	FS2012 Sensor Middleware (rm_fs2012) Middleware to implement the FS2012 sensor interface. This module implements the FSXXXX Middleware Interface.
	Azure RTOS GUIX Port (rm_guix_port)
	HS300X Sensor Middleware (rm_hs300x) Middleware to implement the HS300X sensor interface. This module implements the HS300X Middleware Interface.
	Azure RTOS LevelX NOR SPI Driver (rm_levelx_nor_spi) Middleware for using Azure RTOS LevelX on NOR SPI memory.
	LittleFS Flash Port (rm_littlefs_flash) Middleware for the LittleFS File System control on RA MCUs.
	MCUboot Port (rm_mcuboot_port) MCUboot Port for RA MCUs.
	Motor 120 Control Hall (motor_120_control_hall) Calculation process for the motor control on RA MCUs. This module implements the Interface.
	Motor 120 Control Sensorless (motor_120_control_sensorless) Calculation process for the motor control on RA MCUs. This module implements the Interface.
	Motor 120 Degree (rm_motor_120_degree) Usual control of a SPM (Surface Permanent Magnet) motor on RA MCUs. This module implements the Motor 120 Degree

	(rm_motor_120_degree).
	Motor 120 degree driver (rm_motor_120_driver) Calculation process for the motor control on RA MCUs. This module implements the Interface.
	Motor Current (rm_motor_current) Calculation process for the motor control on RA MCUs. This module implements the Motor current Interface.
	Motor Driver (rm_motor_driver) Calculation process for the motor control on RA MCUs. This module implements the Motor driver Interface.
	Motor encoder vector control (rm_motor_encoder) Control a SPM motor on RA MCUs. This module implements the Motor encoder vector control (rm_motor_encoder).
	Motor Angle and Speed Estimation (rm_motor_estimate) Calculation proccess for the motor control on RA MCUs. This module implements the Motor angle Interface.
	Motor Position (rm_motor_position) Calculation process for the motor control on RA MCUs. This module implements the Motor position Interface.
	Motor Angle and Speed Calculation with an Encoder (rm_motor_sense_encoder) Calculation proccess for the motor control on RA MCUs. This module implements the Motor angle Interface.
	Motor Sensorless Vector Control (rm_motor_sensorless) Usual control of a SPM motor on RA MCUs. This module implements the Motor Sensorless Vector Control (rm_motor_sensorless).
	Motor Speed (rm_motor_speed) Calculation process for the motor control on RA MCUs. This module implements the Motor speed Interface.

	Azure RTOS NetX Secure Crypto Hardware Acceleration (rm_netx_secure_crypto) Hardware acceleration for the Netx Crypto implementation of the Microsoft Azure RTOS NetX Crypto API.
	Azure RTOS NetX Duo Ether Driver (rm_netxduo_ether)
	Azure RTOS NetX Duo WiFi Driver (rm_netxduo_wifi)
	Crypto Middleware (rm_psa_crypto) Hardware acceleration for the mbedCrypto implementation of the ARM PSA Crypto API.
	Azure RTOS ThreadX Port (rm_threadx_port) ThreadX port for RA MCUs.
	Intel TinyCrypt (rm_tinycrypt_port) AES128 Hardware acceleration for TinyCrypt on the RA2 family.
	Capacitive Touch Middleware (rm_touch) This module supports the Capacitive Touch Sensing Unit (CTSU). It implements the Touch Middleware Interface.
	USBX Porting Layer (rm_usbx_port)
	Virtual EEPROM (rm_vee_flash) Virtual EEPROM on RA MCUs. This module implements the Virtual EEPROM Interface.
	AWS Device Provisioning AWS Device Provisioning example software.
	AWS HTTPS This module provides the AWS HTTPS integration documentation.
	AWS MQTT This module provides the AWS MQTT integration documentation.

	Wifi Middleware (rm_wifi_onchip_silex) Wifi and Socket implementation using the Silex SX-ULPGN WiFi module on RA MCUs.
	AWS Secure Sockets This module provides the AWS Secure Sockets implementation.
	ZMOD4XXX Sensor Middleware (rm_zmod4xxx) Middleware to implement the ZMOD4XXX sensor interface. This module implements the ZMOD4XXX Middleware Interface.

4.2.1 High-Speed Analog Comparator (r_acmphs)

Modules

Functions

fsp_err_t	R_ACMPHS_Open (comparator_ctrl_t *p_ctrl, comparator_cfg_t const *const p_cfg)
fsp_err_t	R_ACMPHS_OutputEnable (comparator_ctrl_t *const p_ctrl)
fsp_err_t	R_ACMPHS_InfoGet (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info)
fsp_err_t	R_ACMPHS_StatusGet (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status)
fsp_err_t	R_ACMPHS_Close (comparator_ctrl_t *const p_ctrl)

Detailed Description

Driver for the ACMPHS peripheral on RA MCUs. This module implements the Comparator Interface.

Overview

Features

The ACMPHS HAL module supports the following features:

- Callback on rising edge, falling edge or both
- Configurable debounce filter
- Option for comparator output on VCOUT, CMPOUTn[1], or CMPOUT012[1] pin

- ELC event output

Note

1. This output pin is not available on all MCUs.

Configuration

Build Time Configurations for r_acmphs

The following build time configurations are defined in fsp_cfg/r_acmphs_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Analog > Comparator, High-Speed (r_acmphs)

This module can be added to the Stacks tab via New Stack > Analog > Comparator, High-Speed (r_acmphs).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_comparator0	Module name.
Channel	Value must be a non-negative integer	0	Select the hardware channel.
Trigger Edge Selector	• Rising • Falling • Both Edge	Both Edge	The trigger specifies when a comparator callback event should occur. Unused if the interrupt priority is disabled or the callback is NULL.
Noise Filter	• No Filter • 8 • 16 • 32	No Filter	Select the PCLK divisor for the hardware digital debounce filter. Larger divisors provide a longer debounce and take longer for the output to update.
Maximum status retries (CMPMON)	Must be a valid non-negative integer between 2 and 32-bit maximum value	1024	Maximum number of status retries.
Output Polarity	• Not Inverted • Inverted	Not Inverted	When enabled comparator output is inverted. This affects

			the output read from R_ACMPHS_StatusGet(), the pin output level, and the edge trigger.
Pin Output	• Disabled • Enabled	Disabled	Turn this on to enable the CMPOUTn signal for this channel. The CMPOUTn signal for each channel is OR'd together and the result is output to VCOUT. More pin output options are available on select MCUs.
Callback	Name must be a valid C symbol	NULL	Define this function in the application. It is called when the Trigger event occurs.
Comparator Interrupt Priority	MCU Specific Options		Select the interrupt priority for the comparator interrupt.
Analog Input Voltage Source (IVCMP)	MCU Specific Options		Select the Analog input voltage source. Channel mentioned in the options represents channel in ACMPHS
Reference Voltage Input Source (IVREF)	MCU Specific Options		Select the Analog reference voltage source. Channel mentioned in the options represents channel in ACMPHS

Clock Configuration

The ACMPHS peripheral is clocked from PCLKB. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

Pin Configuration

Comparator output can be enabled or disabled on each channel individually. The VCOUT pin is a logical OR of all comparator outputs.

The IVCMPn pins are used as comparator inputs. The IVREFn pins are used as comparator reference values.

Usage Notes

Noise Filter

When the noise filter is enabled, the ACMPHP0/ACMPHP1 signal is sampled three times based on the sampling clock selected. The filter clock frequency is determined by PCLKB and the comparator_filter_t setting.

Output Polarity

If output polarity is configured as "Inverted" then the VCOUT signal will be inverted and the R_ACMPHS_StatusGet() will return an inverted status.

Limitations

- Once the analog comparator is configured, the program must wait for the stabilization time to elapse before using the comparator.
- When the noise filter is not enabled the hardware requires software debouncing of the output (two consecutive equal values). This is automatically managed in R_ACMPHS_StatusGet but may result in delay or an API error in rare edge cases.
- Constraints apply on the simultaneous use of ACMPHS analog input and ADC analog input. Refer to the "Usage Notes" section in your MCU's User's Manual for the ADC unit(s) for more details.
- To allow ACMPHS0 to cancel Software Standby mode or enter Snooze, set the CSTEN bit to 1 and the CDFS bits to 00 in the CMPCTL0 register.

Examples

Basic Example

The following is a basic example of using the ACMPHS to detect when the analog voltage input to IVCMP rises above the analog voltage input to IVREF. A GPIO output acts as the comparator input and is externally connected to the IVCMP input of the ACMPHS. An analog voltage input should also be supplied to the IVREF input pin.

```
#define ADC_PGA_BYPASS_VALUE (0x9999)
/* Connect this control pin to the IVCMP input of the comparator. This can be any
GPIO pin
 * that is not input only. */
#define ACMPHS_EXAMPLE_CONTROL_PIN (BSP_IO_PORT_05_PIN_03)
volatile uint32_t g_comparator_events = 0U;
/* This callback is called when a comparator event occurs. */
void acmphs_example_callback (comparator_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_comparator_events++;
}
void acmphs_example ()
{
 fsp_err_t err = FSP_SUCCESS;
```

```
 /* Disable pin register write protection, if enabled */
 R_BSP_PinAccessEnable();
 /*
  * Start with the IVCMP pin low. This example assumes the comparator is configured
to trigger
  * when the voltage of the analog input to IVCMP rises above voltage of the analog
input to
  * IVREF.
  */
    (void) R_BSP_PinWrite(ACMPHS_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_LOW);
 /* Initialize the ACMPHS module */
    err = R_ACMPHS_Open(&g_comparator_ctrl, &g_comparator_cfg);
    assert(FSP_SUCCESS == err);
 /*
  * If an ADC PGA exists for the analog input pin, then the PGA must be manually
configured in order for the pin to be used as
  * an IVCMP input. This procedure is slightly different depending on the MCU (See
below).
  */
#if BSP_MCU_GROUP_RA6M3
 /* The following applies for MCUs with the ADC peripheral:
  *
  * Bypass the PGA on ADC unit 0.
  * (See Table 50.2 "Input source configuration of the ACMPHS" in the RA6M3 User's
Manual (R01UH0886EJ0100)) */
 R_BSP_MODULE_START(FSP_IP_ADC, 0);
    R_ADC0->ADPGACR   = ADC_PGA_BYPASS_VALUE;
    R_ADC0->ADPGADCR0 = 0;
#elif BSP_MCU_GROUP_RA6T2
 /* The following applies for MCUs with the ADC_B peripheral:
  *
  * Configure PGA on ADC unit 0.
  * (See Table 36.11 "PGA Settings and Available Related Functions" in the RA6T2
User's Manual (R01UH0951EJ0100)) */
```

```
 R_BSP_MODULE_START(FSP_IP_ADC, 0);
    R_ADC_B->ADPGACR[0] = R_ADC_B0_ADPGACR_PGAGEN_Msk;
#endif
 /* Wait for the minimum stabilization wait time before enabling output. */
 comparator_info_t info;
 R_ACMPHS_InfoGet(&g_comparator_ctrl, &info);
 R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
 /* Enable the comparator output */
    (void) R_ACMPHS_OutputEnable(&g_comparator_ctrl);
 /* Set the IVCMP pin high. */
    (void) R_BSP_PinWrite(ACMPHS_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_HIGH);
 while (0 == g_comparator_events)
    {
 /* Wait for interrupt. */
    }
 comparator_status_t status;
 /* Check status of comparator, Status will be COMPARATOR_STATE_OUTPUT_HIGH */
    (void) R_ACMPHS_StatusGet(&g_comparator_ctrl, &status);
}
```

Function Documentation

◆ R_ACMPHS_Open()

fsp_err_t R_ACMPHS_Open (comparator_ctrl_t *const *p_ctrl*, comparator_cfg_t const *const *p_cfg*)

Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. comparator_api_t::outputEnable() should be called after the output has stabilized. Implements comparator_api_t::open().

Comparator inputs must be configured in the application code prior to calling this function.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An input pointer is NULL
FSP_ERR_INVALID_ARGUMENT	An argument is invalid. Window mode (COMPARATOR_MODE_WINDOW) and filter of 1 (COMPARATOR_FILTER_1) are not supported in this implementation.
FSP_ERR_ALREADY_OPEN	The control block is already open or the hardware lock is taken.

◆ R_ACMPHS_OutputEnable()

fsp_err_t R_ACMPHS_OutputEnable (comparator_ctrl_t *const *p_ctrl*)

Enables the comparator output, which can be polled using comparator_api_t::statusGet(). Also enables pin output and interrupts as configured during comparator_api_t::open(). Implements comparator_api_t::outputEnable().

Return values

FSP_SUCCESS	Comparator output is enabled.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_ACMPHS_InfoGet()

fsp_err_t R_ACMPHS_InfoGet (comparator_ctrl_t *const *p_ctrl*, comparator_info_t *const *p_info*)

Provides the minimum stabilization wait time in microseconds. Implements comparator_api_t::infoGet().

Return values

FSP_SUCCESS	Information stored in p_info.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_ACMPHS_StatusGet()

fsp_err_t R_ACMPHS_StatusGet (comparator_ctrl_t *const *p_ctrl*, comparator_status_t *const *p_status*)

Provides the operating status of the comparator. Implements comparator_api_t::statusGet().

Return values

FSP_SUCCESS	Operating status of the comparator is provided in p_status.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.
FSP_ERR_TIMEOUT	The debounce filter is off and 2 consecutive matching values were not read within 1024 attempts.

◆ R_ACMPHS_Close()

fsp_err_t R_ACMPHS_Close (comparator_ctrl_t * *p_ctrl*)

Stops the comparator. Implements comparator_api_t::close().

Return values

FSP_SUCCESS	Instance control block closed successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

4.2.2 Low-Power Analog Comparator (r_acmplp)

Modules

Functions

fsp_err_t	R_ACMPLP_Open (comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg)
fsp_err_t	R_ACMPLP_OutputEnable (comparator_ctrl_t *const p_ctrl)
fsp_err_t	R_ACMPLP_InfoGet (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info)
fsp_err_t	R_ACMPLP_StatusGet (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status)
fsp_err_t	R_ACMPLP_Close (comparator_ctrl_t *const p_ctrl)

Detailed Description

Driver for the ACMPLP peripheral on RA MCUs. This module implements the Comparator Interface.

Overview

Features

The ACMPLP HAL module supports the following features:

- Normal mode or window mode
- Callback on rising edge, falling edge or both
- Configurable debounce filter
- Option for comparator output on VCOUT pin
- ELC event output

Configuration

Build Time Configurations for r_acmplp

The following build time configurations are defined in fsp_cfg/r_acmplp_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Reference Voltage Selection for ACMPLP1	• IVREF0 • IVREF1	IVREF1	ACMPLP1 may optionally be

(Standard mode only)			configured to use IVREF0 as a reference input instead of IVREF1. Note that if IVREF0 is selected, ACMPLP0 and ACMPLP1 must use the same setting for IVREF.

Configurations for Analog > Comparator, Low-Power (r_acmplp)

This module can be added to the Stacks tab via New Stack > Analog > Comparator, Low-Power (r_acmplp).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_comparator0	Module name.
Channel	Value must be a non-negative integer	0	Select the hardware channel.
Mode	• Standard • Window	Standard	In standard mode, comparator output is high if VCMP > VREF. In window mode, comparator output is high if VCMP is outside the range of VREF0 to VREF1.
Trigger	• Rising • Falling • Both Edge	Both Edge	The trigger specifies when a comparator callback event should occur. Unused if the interrupt priority is disabled or the callback is NULL.
Filter	• No sampling (bypass) • Sampling at PCLKB • Sampling at PCLKB/8 • Sampling at PCLKB/32	No sampling (bypass)	Select the PCLK divisor for the hardware digital debounce filter. Larger divisors provide a longer debounce and take longer for the output to update.
Output Polarity	• Not Inverted • Inverted	Not Inverted	When enabled comparator output is inverted. This affects the output read from R_ACMPLP_StatusGet(), the pin output level, and the edge trigger.
Pin Output (VCOUT)	• Disabled	Disabled	Turn this on to include

	• Enabled		the output from this comparator on VCOUT. The comparator output on VCOUT is OR'd with output from all other ACMPHS and ACMPLP comparators.
Vref (Standard mode only)	• Enabled • Disabled	Disabled	If reference voltage selection is enabled then internal reference voltage is used as comparator input
Callback	Name must be a valid C symbol	NULL	Define this function in the application. It is called when the Trigger event occurs.
Comparator Interrupt Priority	MCU Specific Options		Select the interrupt priority for the comparator interrupt.
Analog Input Voltage Source (IVCMP)	MCU Specific Options		Select the comparator input source. Only options for the configured channel are valid.
Reference Voltage Input Source (IVREF)	MCU Specific Options		Select the comparator reference voltage source. If channel 1 is seleected and the 'Reference Voltage Selection (ACMPLP1)' config option is set to IVREF0, select one of the Channel 0 options. In all other cases, only options for the configured channel are valid.

Clock Configuration

The ACMPLP peripheral is clocked from PCLKB. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

Pin Configuration

Comparator output can be enabled or disabled on each channel individually. The VCOUT pin is a logical OR of all comparator outputs.

The CMPINn pins are used as comparator inputs. The CMPREFn pins are used as comparator reference values.

Usage Notes

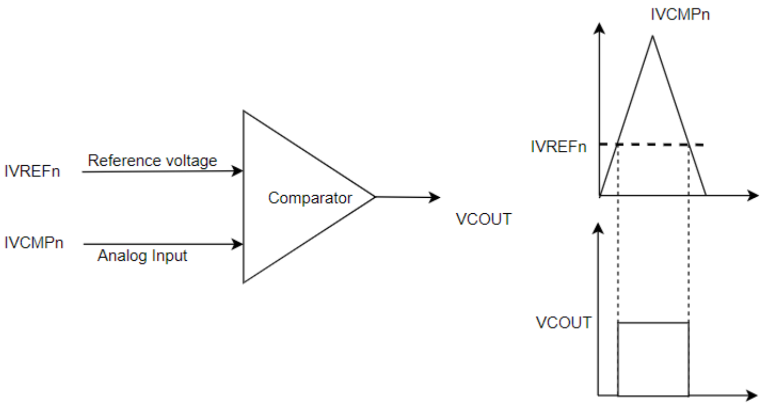


Figure 140: ACMPLP Standard Mode Operation

Noise Filter

When the noise filter is enabled, the ACMPLP0/ACMPLP1 signal is sampled three times based on the sampling clock selected. The filter clock frequency is determined by PCLKB and the comparator_filter_t setting.

Output Polarity

If output polarity is configured as "Inverted" then the VCOUT signal will be inverted and the R_ACMPLP_StatusGet() will return an inverted status.

Window Mode

In window mode, the comparator indicates if the analog input voltage falls within the window (low and high reference voltage) or is outside the window.

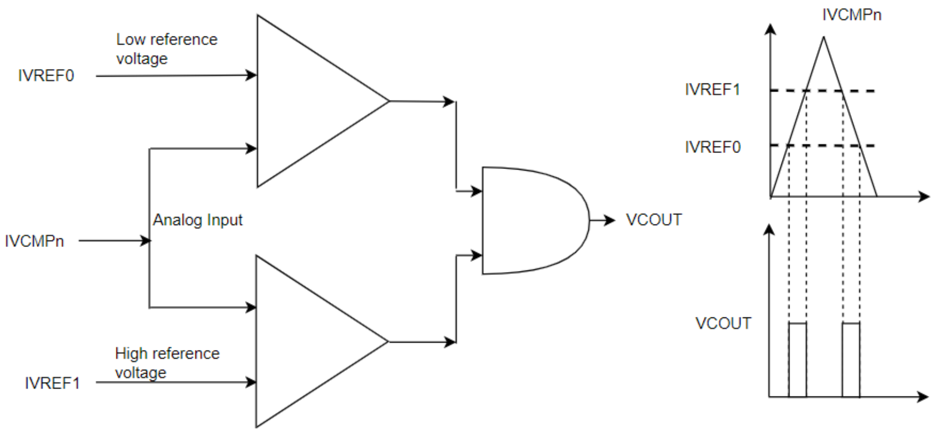


Figure 141: ACMPLP Window Mode Operation

Limitations

- Once the analog comparator is configured, the program must wait for the stabilization time to elapse before using the comparator.

- Low speed is not supported by the ACMPLP driver.

Examples

Basic Example

The following is a basic example of minimal use of the ACMPLP. The comparator is configured to trigger a callback when the input rises above the internal reference voltage (VREF). A GPIO output acts as the comparator input and is externally connected to the CMPIN input of the ACMPLP.

```
/* Connect this control pin to the VCMP input of the comparator. This can be any GPIO
pin
 * that is not input only. */
#define ACMPLP_EXAMPLE_CONTROL_PIN (BSP_IO_PORT_04_PIN_08)
volatile uint32_t g_comparator_events = 0U;
/* This callback is called when a comparator event occurs. */
void acmplp_example_callback (comparator_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_comparator_events++;
}
void acmplp_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Disable pin register write protection, if enabled */
 R_BSP_PinAccessEnable();
 /* Start with the VCMP pin low. This example assumes the comparator is configured to
trigger
  * when VCMP rises above VREF. */
    (void) R_BSP_PinWrite(ACMPLP_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_LOW);
 /* Initialize the ACMPLP module */
    err = R_ACMPLP_Open(&g_comparator_ctrl, &g_comparator_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait for the minimum stabilization wait time before enabling output. */
 comparator_info_t info;
 R_ACMPLP_InfoGet(&g_comparator_ctrl, &info);
 R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
```

```
/* Enable the comparator output */
   (void) R_ACMPLP_OutputEnable(&g_comparator_ctrl);
/* Set VCMP low. */
   (void) R_BSP_PinWrite(ACMPLP_EXAMPLE_CONTROL_PIN, BSP_IO_LEVEL_HIGH);
while (0 == g_comparator_events)
   {
/* Wait for interrupt. */
   }
comparator_status_t status;
/* Check status of comparator, Status will be COMPARATOR_STATE_OUTPUT_HIGH */
   (void) R_ACMPLP_StatusGet(&g_comparator_ctrl, &status);
}
```

Enumerations

enum	acmplp_input_t
enum	acmplp_reference_t

Enumeration Type Documentation

◆ acmplp_input_t

enum acmplp_input_t	
Enumerator	
ACMPLP_INPUT_AMPO	Not available on all MCUs.
ACMPLP_INPUT_CMPIN_1	Not available on all MCUs.

◆ acmplp_reference_t

enum acmplp_reference_t	
Enumerator	
ACMPLP_REFERENCE_CMPREF_1	Not available on all MCUs.

Function Documentation

◆ R_ACMPLP_Open()

fsp_err_t R_ACMPLP_Open (comparator_ctrl_t *const *p_ctrl*, comparator_cfg_t const *const *p_cfg*)

Configures the comparator and starts operation. Callbacks and pin output are not active until outputEnable() is called. comparator_api_t::outputEnable() should be called after the output has stabilized. Implements comparator_api_t::open().

Comparator inputs must be configured in the application code prior to calling this function.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An input pointer is NULL
FSP_ERR_INVALID_ARGUMENT	An argument is invalid. Window mode (COMPARATOR_MODE_WINDOW) and filter of 1 (COMPARATOR_FILTER_1) are not supported in this implementation. p_cfg->p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback function.
FSP_ERR_ALREADY_OPEN	The control block is already open or the hardware lock is taken.
FSP_ERR_IN_USE	The channel is already in use.

◆ R_ACMPLP_OutputEnable()

fsp_err_t R_ACMPLP_OutputEnable (comparator_ctrl_t *const *p_ctrl*)

Enables the comparator output, which can be polled using comparator_api_t::statusGet(). Also enables pin output and interrupts as configured during comparator_api_t::open(). Implements comparator_api_t::outputEnable().

Return values

FSP_SUCCESS	Comparator output is enabled.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_ACMPLP_InfoGet()

fsp_err_t R_ACMPLP_InfoGet (comparator_ctrl_t *const *p_ctrl*, comparator_info_t *const *p_info*)

Provides the minimum stabilization wait time in microseconds. Implements comparator_api_t::infoGet().

Return values

FSP_SUCCESS	Information stored in p_info.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_ACMPLP_StatusGet()

fsp_err_t R_ACMPLP_StatusGet (comparator_ctrl_t *const *p_ctrl*, comparator_status_t *const *p_status*)

Provides the operating status of the comparator. Implements comparator_api_t::statusGet().

Return values

FSP_SUCCESS	Operating status of the comparator is provided in p_status.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_ACMPLP_Close()

fsp_err_t R_ACMPLP_Close (comparator_ctrl_t * *p_ctrl*)

Stops the comparator. Implements comparator_api_t::close().

Return values

FSP_SUCCESS	Instance control block closed successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

4.2.3 Analog to Digital Converter (r_adc)

Modules

Functions

fsp_err_t	R_ADC_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg)
fsp_err_t	R_ADC_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_channel_cfg)
fsp_err_t	R_ADC_InfoGet (adc_ctrl_t *p_ctrl, adc_info_t *p_adc_info)
fsp_err_t	R_ADC_ScanStart (adc_ctrl_t *p_ctrl)
fsp_err_t	R_ADC_ScanGroupStart (adc_ctrl_t *p_ctrl, adc_group_mask_t group_id)
fsp_err_t	R_ADC_ScanStop (adc_ctrl_t *p_ctrl)
fsp_err_t	R_ADC_StatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status)
fsp_err_t	R_ADC_Read (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data)
fsp_err_t	R_ADC_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data)
fsp_err_t	R_ADC_SampleStateCountSet (adc_ctrl_t *p_ctrl, adc_sample_state_t *p_sample)
fsp_err_t	R_ADC_Close (adc_ctrl_t *p_ctrl)
fsp_err_t	R_ADC_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t offset)
fsp_err_t	R_ADC_Calibrate (adc_ctrl_t *const p_ctrl, void const *p_extend)
fsp_err_t	R_ADC_CallbackSet (adc_ctrl_t *const p_api_ctrl, void(*p_callback)(adc_callback_args_t *), void const *const p_context, adc_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the ADC12, ADC14, and ADC16 peripherals on RA MCUs. This module implements the ADC Interface.

Overview

Features

The ADC module supports the following features:

- 12, 14, or 16 bit maximum resolution depending on the MCU
- Configure scans to include:
    - Multiple analog channels
    - Temperature sensor channel
    - Voltage sensor channel
- Configurable scan start trigger:
    - Software scan triggers
    - Hardware scan triggers (timer expiration, for example)
    - External scan triggers from the ADTRGn port pins
- Configurable scan mode:
    - Single scan mode, where each trigger starts a single scan
    - Continuous scan mode, where all channels are scanned continuously
    - Group scan mode, where channels are grouped into group A and group B. The groups can be assigned different start triggers, and group A can be given priority over group B. When group A has priority over group B, a group A trigger suspends an ongoing group B scan.
- Supports adding and averaging converted samples
- Optional callback when scan completes
- Sample and hold support
- Double-trigger support
- Hardware comparator with interrupt and event output

Configuration

Build Time Configurations for r_adc

The following build time configurations are defined in fsp_cfg/r_adc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Analog > ADC (r_adc)

This module can be added to the Stacks tab via New Stack > Analog > ADC (r_adc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_adc0	Module name
General > Unit	Unit must be a non-negative integer	0	Specifies the ADC Unit to be used.
General > Resolution	MCU Specific Options		Specifies the conversion resolution for this unit.
General > Alignment	MCU Specific Options		Specifies the conversion result

			alignment.
General > Clear after read	• Off • On	On	Specifies if the result register will be automatically cleared after the conversion result is read.
General > Mode	• Single Scan • Continuous Scan • Group Scan	Single Scan	Specifies the mode that this ADC unit is used in.
General > Double-trigger	• Disabled • Enabled • Enabled (extended mode)	Disabled	When enabled, the scan-end interrupt for Group A is only thrown on every second scan. Extended double-trigger mode (single-scan only) triggers on both ELC events, allowing (for example) a scan on two different timer compare match values. In group mode Group B is unaffected.
Input > Sample and Hold > Sample and Hold Channels (Available only on selected MCUs)	• Channel 0 • Channel 1 • Channel 2		Specifies if this channel is included in the Sample and Hold Mask.
Input > Sample and Hold > Sample Hold States (Applies only to channels 0, 1, 2)	Must be a valid non-negative integer with configurable value 4 to 255	24	Specifies the updated sample-and-hold count for the channel dedicated sample-and-hold circuit
Input > Window Compare > Window A > Enable	• Disabled • Enabled	Disabled	Enable or disable comparison with Window A.
Input > Window Compare > Window A > Channels to compare (channel availability varies by MCU and unit)	Refer to the RA Configuration tool for available options.		Select channels to be compared to Window A.
Input > Window Compare > Window A > Channel comparison mode (channel	Refer to the RA Configuration tool for available options.		Checking a box sets the comparison mode for that channel to Greater Than or Inside

availability varies by MCU and unit)			Window depending on whether Window Mode is disabled or enabled (respectively). If left unchecked the comparison mode will likewise be Less Than or Outside Window (respectively).
Input > Window Compare > Window A > Lower Reference	Must be a positive 16-bit integer.	0	Set the lower comparison value.
Input > Window Compare > Window A > Upper Reference	Must be a positive 16-bit integer.	0	Set the upper comparison value.
Input > Window Compare > Window B > Enable	• Disabled • Enabled	Disabled	Enable or disable comparison with Window B.
Input > Window Compare > Window B > Channel to compare (channel availability varies by MCU and unit)	Refer to the RA Configuration tool for available options.	Channel 0	Select a channel to be compared to Window B.
Input > Window Compare > Window B > Comparison mode	• Less Than or Outside Window • Greater Than or Inside Window	module.driver.adc.compare.window_b.mode	Select the comparison mode for Window B. For each option, the first condition applies when Window Mode is disabled and the second option applies when Window Mode is enabled.
Input > Window Compare > Window B > Lower Reference	Must be a positive 16-bit integer.	0	Set the lower comparison value.
Input > Window Compare > Window B > Upper Reference	Must be a positive 16-bit integer.	0	Set the upper comparison value.
Input > Window Compare > Window Mode	• Disabled • Enabled	Disabled	When disabled, ADC values will be compared only with the lower reference on each comparator. When enabled, both the lower and upper reference values will be used to create a comparison window.

Input > Window Compare > Event Output	• OR • XOR • AND	OR	Select how comparison results should be composited for event output.
Input > Channel Scan Mask (channel availability varies by MCU)	Refer to the RA Configuration tool for available options.		In Normal mode of operation, this bitmask field specifies the channels that are enabled in that ADC unit. In group mode, this field specifies which channels belong to group A.
Input > Group B Scan Mask (channel availability varies by MCU)	Refer to the RA Configuration tool for available options.		In group mode, this field specifies which channels belong to group B.
Input > Add/Average Count	MCU Specific Options		Specifies if addition or averaging needs to be done for any of the channels in this unit.
Input > Reference Voltage control	MCU Specific Options		Specify VREFH/VREFADC output voltage control.
Input > Addition/Averaging Mask (channel availability varies by MCU and unit)	Refer to the RA Configuration tool for available options.		Select channels to include in the Addition/Averaging Mask
Interrupts > Normal/Group A Trigger	MCU Specific Options		Specifies the trigger type to be used for this unit.
Interrupts > Group B Trigger	MCU Specific Options		Specifies the trigger for Group B scanning in group scanning mode. This event is also used to trigger Group A in extended double-trigger mode.
Interrupts > Group Priority (Valid only in Group Scan Mode)	• Group A cannot interrupt Group B • Group A can interrupt Group B; Group B scan restarts at next trigger • Group A can interrupt Group B; Group B scan	Group A cannot interrupt Group B	Determines whether an ongoing group B scan can be interrupted by a group A trigger, whether it should abort on a group A trigger, or if it should pause to allow group A scan and restart immediately after group A scan is complete.

	restarts immediately • Group A can interrupt Group B; Group B scan restarts immediately and scans continuously		
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the ADC scan completes.
Interrupts > Scan End Interrupt Priority	MCU Specific Options		Select scan end interrupt priority.
Interrupts > Scan End Group B Interrupt Priority	MCU Specific Options		Select group B scan end interrupt priority.
Interrupts > Window Compare A Interrupt Priority	MCU Specific Options		Select Window Compare A interrupt priority.
Interrupts > Window Compare B Interrupt Priority	MCU Specific Options		Select Window Compare B interrupt priority.
Extra > ADC Ring Buffer	MCU Specific Options		ADC Ring Buffer to be used only with DMAC transfers, keep this property disabled for normal ADC operations. When enabled, ADC converted data is stored in ADBUF registers in place of ADDR registers. The read API will not read from this location for normal ADC operations.

Clock Configuration

The ADC clock is PCLKC if the MCU has PCLKC, or PCLKD otherwise.

The clock for this module is derived from the following peripheral clock for each MCU group:

MCU Group	Peripheral Clock
RA2A1	PCLKD
RA2E1	PCLKD

RA2E2	PCLKD
RA2L1	PCLKD
RA4E1	PCLKC
RA4M1	PCLKC
RA4M2	PCLKC
RA4M3	PCLKC
RA4W1	PCLKC
RA6E1	PCLKC
RA6M1	PCLKC
RA6M2	PCLKC
RA6M3	PCLKC
RA6M4	PCLKC
RA6M5	PCLKC
RA6T1	PCLKC
RA6T2	PCLKC

The ADC clock must be at least 1 MHz when the ADC is used. Many MCUs also have PCLK ratio restrictions when the ADC is used. For details on PCLK ratio restrictions, reference the footnotes in the second table of the Clock Generation Circuit chapter of the MCU User's Manual (for example, Table 9.2 "Specifications of the clock generation circuit for the internal clocks" in the RA6M3 manual R01UH0886EJ0100).

Pin Configuration

The ANxxx pins are analog input channels that can be used with the ADC.

ADTRG0 and ADTRG1 can be used to start scans with an external trigger for unit 0 and 1 respectively. When external triggers are used, ADC scans begin on the falling edge of the ADTRG pin.

Usage Notes

Sample Hold

Enabling the sample and hold functionality reduces the maximum scan frequency because the sample and hold time is added to each scan. Refer to the hardware manual for details on the sample and hold time.

ADC Operational Modes

The driver supports three operation modes: single-scan, continuous-scan, and group-scan modes. In each mode, analog channels are converted in ascending order of channel number, followed by scans of the temperature sensor and voltage sensor if they are included in the mask of channels to scan.

Single-scan Mode

In single scan mode, one or more specified channels are scanned once per trigger.

Continuous-scan Mode

In continuous scan mode, a single trigger is required to start the scan. Scans continue until R_ADC_ScanStop() is called.

Group-scan Mode

Group-scan mode allows the application to allocate channels to one of two groups (A and B). Conversion begins when the specified ELC start trigger for that group is received.

With the priority configuration parameter, you can optionally give group A priority over group B. If group A has priority over group B, a group B scan is interrupted when a group A scan trigger occurs. The following options exist for group B when group A has priority:

- To restart the interrupted group B scan after the group A scan completes.
- To wait for another group B trigger and forget the interrupted scan.
- To continuously scan group B and suspend scanning group B only when a group A trigger is received.
  Note
  > *If this option is selected, group B scanning begins immediately after R_ADC_ScanCfg(). Group A scan triggers must be enabled by R_ADC_ScanStart() and can be disabled by R_ADC_ScanStop(). Group B scans can only be disabled by reconfiguring the group A priority to a different mode.*

Double-triggering

When double-triggering is enabled a single channel is selected to be scanned twice before an interrupt is thrown. The first scan result when using double-triggering is always saved to the selected channel's data register. The second result is saved to the data duplexing register (ADC_CHANNEL_DUPLEX).

Double-triggering uses Group A; only one channel can be selected when enabled. No other scanning is possible on Group A while double-trigger mode is selected. In addition, any special ADC channels (such as temperature sensors or voltage references) are not valid double-trigger channels.

When extended double-triggering is enabled both ADC input events are routed to Group A. The interrupt is still thrown after every two scans regardless of the triggering event(s). While the first and second scan are saved to the selected ADC data register and the ADC duplexing register as before, scans associated with event A and B are additionally copied into duplexing register A and B, respectively (ADC_CHANNEL_DUPLEX_A and ADC_CHANNEL_DUPLEX_B).

When Interrupts Are Not Enabled

If interrupts are not enabled, the R_ADC_StatusGet API can be used to poll the ADC to determine when the scan has completed. The read API function is used to access the converted ADC result. This applies to both normal scans and calibration scans for MCUs that support calibration.

Window Compare Function

The ADC contains comparators that allow scan data to be compared to user-provided reference values. When a value meets the configured condition an interrupt and/or an ELC event can be produced.

Each unit has two configurable comparison units, Window A and Window B. Window A allows for configuring multiple simultaneous channels to compare while Window B only allows one channel at a time.

The window compare function can be configured both through the RA Configuration tool and at runtime by providing a pointer to an adc_window_cfg_t struct to adc_channel_cfg_t::p_window_cfg when calling R_ADC_ScanCfg. The available comparison modes are shown below:

Window setting	Channel mode 0	Channel mode 1
Disabled	Scan < Low Ref	Scan > Low Ref
Enabled	(Scan < Low Ref) OR (Scan > High Ref)	Low Ref < Scan < High Ref

Note

The window setting applies to all channels configured on a unit.

Sample-State Count Setting

The application program can modify the setting of the sample-state count for analog channels by calling the R_ADC_SampleStateCountSet() API function. The application program only needs to modify the sample-state count settings from their default values to increase the sampling time. This can be either because the impedance of the input signal is too high to secure sufficient sampling time under the default setting or if the ADCLK is too slow. To modify the sample-state count for a given channel, set the channel number and the number of states when calling the R_ADC_SampleStateCountSet() API function. Valid sample state counts are 7-255.

Note

Although the hardware supports a minimum number of sample states of 5, some MCUs require 7 states, so the minimum is set to 7. At the lowest supported ADC conversion clock rate (1 MHz), these extra states will lead to, at worst case, a 2 microsecond increase in conversion time. At 60 MHz the extra states will add 33.4 ns to the conversion time.

If the sample state count needs to be changed for multiple channels, the application program must call the R_ADC_SampleStateCountSet() API function repeatedly, with appropriately modified arguments for each channel.

If the ADCLK frequency changes, the sample states may need to be updated.

Sample States for Temperature Sensor and Internal Voltage Reference

Sample states for the temperature sensor and the internal reference voltage are calculated during R_ADC_ScanCfg() based on the ADCLK frequency at the time. The sample states for the temperature sensor and internal voltage reference cannot be updated with R_ADC_SampleStateCountSet(). If the ADCLK frequency changes, call R_ADC_ScanCfg() before using the temperature sensor or internal reference voltage again to ensure the sampling time for the temperature sensor and internal voltage reference is optimal.

Selecting Reference Voltage

The ADC16 can select VREFH0 or VREFADC as the high-potential reference voltage on selected MCU's. When using VREFADC stabilization time of 1500us is required after call for R_ADC_Open().

Using the Temperature Sensor with the ADC

The ADC HAL module supports reading the data from the on-chip temperature sensor. The value returned from the sensor can be converted into degrees Celsius or Fahrenheit in the application program using the following formula, T = (Vs - V1)/slope + T1, where:

- T: Measured temperature (degrees C)
- Vs: Voltage output by the temperature sensor at the time of temperature measurement (Volts)
- T1: Temperature experimentally measured at one point (degrees C)
- V1: Voltage output by the temperature sensor at the time of measurement of T1 (Volts)
- T2: Temperature at the experimental measurement of another point (degrees C)
- V2: Voltage output by the temperature sensor at the time of measurement of T2 (Volts)
- Slope: Temperature gradient of the temperature sensor (V/degrees C); slope = (V2 - V1)/(T2 - T1)

Note

The slope value can be obtained from the hardware manual for each device in the Electrical Characteristics Chapter - TSN Characteristics Table, Temperature slope entry.

Reading CTSU TSCAP with ADC

Some MCUs support reading CTSU TSCAP with ADC. CTSU TSCAP is connected to ADC0 channel 16. Use existing enums for channel 16 to set sample states for the sensor connected to CTSU TSCAP, enable scanning of CTSU TSCAP, and read results for CTSU TSCAP.

Usage Notes for ADC16

Calibration

Calibration is required to use the ADC16 peripheral. When using this driver on an MCU that has ADC16, call R_ADC_Calibrate() after open, and prior to any other function.

Range of ADC16 Results

The range of the ADC16 is from 0 (lowest) to 0x7FFF (highest) when used in single-ended mode. This driver only supports single ended mode.

Limitations

Developers should be aware of the following limitations when using the ADC:

- When using the Window Compare function:
  - Only Single Scan mode may be configured when match or mismatch ELC events are used.
  - When one compare window is configured to check the temperature sensor or voltage reference the other window cannot be used.
  - Both windows cannot reference the same channel.

Examples

Basic Example

This is a basic example of minimal use of the ADC in an application.

```
/* A channel configuration is generated by the RA Configuration editor based on the
```

```
options selected. If additional
 * configurations are desired additional adc_channel_cfg_t elements can be defined
and passed to R_ADC_ScanCfg. */
const adc_channel_cfg_t g_adc0_channel_cfg =
{
    .scan_mask          = ADC_MASK_CHANNEL_0 | ADC_MASK_CHANNEL_1,
    .scan_mask_group_b = 0,
    .priority_group_a   = (adc_group_a_t) 0,
    .add_mask          = 0,
    .sample_hold_mask   = 0,
    .sample_hold_states = 0,
};
void adc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable channels. */
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* In software trigger mode, start a scan by calling R_ADC_ScanStart(). In other
modes, enable external
  * triggers by calling R_ADC_ScanStart(). */
    (void) R_ADC_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. */
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
       (void) R_ADC_StatusGet(&g_adc0_ctrl, &status);
    }
 /* Read converted data. */
```

```
    uint16_t channel1_conversion_result;
    err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_1, &channel1_conversion_result);
    assert(FSP_SUCCESS == err);
}
```

Temperature Sensor Example

This example shows how to calculate the MCU temperature using the ADC and the temperature sensor.

```
#define ADC_EXAMPLE_CALIBRATION_DATA_RA6M1 (0x7D5)
#define ADC_EXAMPLE_VCC_MICROVOLT (3300000)
#define ADC_EXAMPLE_TEMPERATURE_RESOLUTION (12U)
#define ADC_EXAMPLE_REFERENCE_CALIBRATION_TEMPERATURE (127)
void adc_temperature_example (void)
{
 /* The following example calculates the temperature on an RA6M1 device using the
data provided in the section
  * 44.3.1 "Preparation for Using the Temperature Sensor" of the RA6M1 manual
R01UH0884EJ0100. */
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable temperature sensor. */
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* In software trigger mode, start a scan by calling R_ADC_ScanStart(). In other
modes, enable external
  * triggers by calling R_ADC_ScanStart(). */
    (void) R_ADC_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. */
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
```

```
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
       (void) R_ADC_StatusGet(&g_adc0_ctrl, &status);
    }
 /* Read converted data. */
    uint16_t temperature_conversion_result;
    err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_TEMPERATURE,
&temperature_conversion_result);
    assert(FSP_SUCCESS == err);
 /* If the MCU does not provide calibration data, use the value in the hardware
manual or determine it
  * experimentally. */
 /* Get Calibration data from the MCU if available. */
    int32_t    reference_calibration_data;
 adc_info_t adc_info;
    (void) R_ADC_InfoGet(&g_adc0_ctrl, &adc_info);
    reference_calibration_data = (int32_t) adc_info.calibration_data;
 /* NOTE: The slope of the temperature sensor varies from sensor to sensor. Renesas
recommends calculating
  * the slope of the temperature sensor experimentally.
  *
  * This example uses the typical slope provided in Table 52.38 "TSN characteristics"
in the RA6M1 manual
  * R01UM0011EU0050. */
    int32_t slope_uv_per_c = BSP_FEATURE_ADC_TSN_SLOPE;
 /* Formula for calculating temperature copied from section 44.3.1 "Preparation for
Using the Temperature Sensor"
  * of the RA6M1 manual R01UH0884EJ0100:
  *
  * In this MCU, the TSCDR register stores the temperature value (CAL127) of the
temperature sensor measured
  * under the condition Ta = Tj = 127 C and AVCC0 = 3.3 V. By using this value as the
sample measurement result
  * at the first point, preparation before using the temperature sensor can be
```

```
omitted.
  *
  * If V1 is calculated from CAL127,
  * V1 = 3.3 * CAL127 / 4096 [V]
  *
  * Using this, the measured temperature can be calculated according to the following
formula.
  *
  * T = (Vs - V1) / Slope + 127 [C]
  * T: Measured temperature (C)
  * Vs: Voltage output by the temperature sensor when the temperature is measured (V)
  * V1: Voltage output by the temperature sensor when Ta = Tj = 127 C and AVCC0 = 3.3
V (V)
  * Slope: Temperature slope given in Table 52.38 / 1000 (V/C)
  */
    int32_t v1_uv = (ADC_EXAMPLE_VCC_MICROVOLT >> ADC_EXAMPLE_TEMPERATURE_RESOLUTION)
*
                   reference_calibration_data;
    int32_t vs_uv = (ADC_EXAMPLE_VCC_MICROVOLT >> ADC_EXAMPLE_TEMPERATURE_RESOLUTION)
*
                   temperature_conversion_result;
    int32_t temperature_c = (vs_uv - v1_uv) / slope_uv_per_c +
ADC_EXAMPLE_REFERENCE_CALIBRATION_TEMPERATURE;
 /* Expect room temperature, break if temperature is outside the range of 20 C to 25
C. */
 if ((temperature_c < 20) || (temperature_c > 25))
    {
       __BKPT(0);
    }
}
```

Double-Trigger Example

This example demonstrates reading data from a double-trigger scan. A flag is used to wait for a callback event. Two scans must occur before the callback is called. These results are read via

R_ADC_Read using the selected channel enum value as well as ADC_CHANNEL_DUPLEX.

```
volatile bool scan_complete_flag = false;
void adc_callback (adc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    scan_complete_flag = true;
}
void adc_double_trigger_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the module. */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable double-trigger channel. */
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* Enable scan triggering from ELC events. */
    (void) R_ADC_ScanStart(&g_adc0_ctrl);
 /* Wait for conversion to complete. Two scans must be triggered before a callback
occurs. */
    scan_complete_flag = false;
 while (!scan_complete_flag)
    {
 /* Wait for callback to set flag. */
    }
 /* Read converted data from both scans. */
    uint16_t channel1_conversion_result_0;
    uint16_t channel1_conversion_result_1;
    err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_1, &channel1_conversion_result_0);
    assert(FSP_SUCCESS == err);
    err = R_ADC_Read(&g_adc0_ctrl, ADC_CHANNEL_DUPLEX,
&channel1_conversion_result_1);
    assert(FSP_SUCCESS == err);
```

```
}
```

ADC-DMAC Repeat-Block Transfer Example

This example demonstrates writing multiple data from DAC peripheral to ADC channels and storing the data in memory through DMAC using Repeat-Block Transfer mode. It creates single block to multiple ring buffer type of transfer topology. Ping-Pong mechanism is used to read the data from memory in between the transfers. This example is valid only for MCUs that have ADBUF.

```
 #define ADC_DMAC_EXAMPLE_DATA_LOW (0U)
 #define ADC_DMAC_EXAMPLE_DATA_HIGH (0x000FU)
 #define ADC_DMAC_EXAMPLE_DELAY_1000_MS (1000U)
 #define ADC_DMAC_EXAMPLE_NUM_PING_PONG_BUFFERS (2)
static uint16_t g_adc_dmac_example_buffer[ADC_DMAC_EXAMPLE_NUM_PING_PONG_BUFFERS][
    ADC_DMAC_EXAMPLE_ADC_CHANNELS_PER_BLOCK][ADC_DMAC_EXAMPLE_SAMPLES_PER_CHANNEL];
// Destination buffer for DMAC transfers
static volatile uint16_t g_adc_dmac_example_ping_pong_index = 0U;
static volatile void * gp_read_data;
/* DMAC callback */
void adc_dmac_callback (dmac_callback_args_t * p_args)
{
    (void) p_args;
 /* Store the pointer to the last buffer that was written
  * An array of data for the first enabled channel is at
&g_adc_dmac_example_buffer[g_adc_dmac_example_ping_pong_index][0][0],
  * an array of data for the next channel is at
&g_adc_dmac_example_buffer[g_adc_dmac_example_ping_pong_index][1][0], etc.
  */
    gp_read_data =
&g_adc_dmac_example_buffer[g_adc_dmac_example_ping_pong_index][0][0];
 /* Select the other ping-pong buffer which is free for writing */
    g_adc_dmac_example_ping_pong_index = !g_adc_dmac_example_ping_pong_index;
 /* Reset the destination pointer and DMAC peripheral */
 R_DMAC_Reset(&g_transfer0_ctrl,
                NULL,
                (void *)
```

```
&g_adc_dmac_example_buffer[g_adc_dmac_example_ping_pong_index][0][0],
                ADC_DMAC_EXAMPLE_SAMPLES_PER_CHANNEL);
 FSP_PARAMETER_NOT_USED(gp_read_data);
}
void adc_dmac_repeat_block_transfer_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open ADC Module and configure the channels */
 /* Enable the ADBUF property from configurations */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* Open DMAC channel for repeat-block transfer with following configurations
  * (1) Destination address as
&g_adc_dmac_example_buffer[g_adc_dmac_example_ping_pong_index][0][0]
  * (2) Enable end of transfer interrupt
  * (3) Configure source address mode as incremented and destination address mode as
offset addition,
  * fixed to address of ADBUF register by configurator with ADC-DMAC module
  * (4) Configure source buffer size as total size of source buffer - Refer RA6M4
Hardware Manual R01UH0890EJ0110,
  * section 16.2.15 for source buffer size limitations
  * (5) Configure transfer mode as Repeat-Block mode
  * (6) Refer RA6M4 Hardware Manual R01UH0890EJ0110,
  * section 16.2.16 for total number of blocks which decides destination buffer size
  * (7) Number of blocks is determined by the samples per channel property for ADC-
DMAC module
  * (8) Size of block is determined using the enabled ADC channels in the
configurator when using ADC-DMAC-module
  * (9) Configure DMAC activation source as A/D scan end interrupt
  */
    err = R_DMAC_Open(&g_transfer0_ctrl, &g_transfer0_cfg);
    assert(FSP_SUCCESS == err);
```

```
    err = R_DMAC_Enable(&g_transfer0_ctrl);
    assert(FSP_SUCCESS == err);
    uint16_t    count = ADC_DMAC_EXAMPLE_DATA_LOW;
 adc_status_t adc_status;
 /* Trigger the ADC scan for "count" times, this can be replaced by triggering the
ADC using a timer */
 while (count <= (uint16_t) ADC_DMAC_EXAMPLE_DATA_HIGH)
    {
 /* Scan the data with ADC channels*/
       err = R_ADC_ScanStart(&g_adc0_ctrl);
      assert(FSP_SUCCESS == err);
 /* Wait for conversion to complete */
       uint16_t timeout = UINT16_MAX;
       adc_status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while ((ADC_STATE_SCAN_IN_PROGRESS == adc_status.state) && (timeout > 0U))
      {
            timeout--;
 R_ADC_StatusGet(&g_adc0_ctrl, &adc_status);
      }
 R_BSP_SoftwareDelay(ADC_DMAC_EXAMPLE_DELAY_1000_MS, BSP_DELAY_UNITS_MICROSECONDS);
       count++;
    }
}
```

Window Compare Example

This example shows how to configure the window compare function at runtime as well as how to handle events and obtain comparison results through a callback.

```
adc_window_cfg_t g_adc0_window_cfg =
{
 /* Enable Window A and Window B; enable Window mode */
    .compare_cfg       =
       (adc_compare_cfg_t) (ADC_COMPARE_CFG_A_ENABLE | ADC_COMPARE_CFG_B_ENABLE |
ADC_COMPARE_CFG_WINDOW_ENABLE),
```

```
 /* Compare scan values from Channels 0 and 1 */
    .compare_mask       = ADC_MASK_CHANNEL_0 | ADC_MASK_CHANNEL_1,
 /* Set Channel 1 condition to be inside the window instead of outside */
    .compare_mode_mask = ADC_MASK_CHANNEL_1,
 /* Set reference voltage levels for Window A */
    .compare_ref_low = ADC_SCAN_MAX / 3,
    .compare_ref_high = ADC_SCAN_MAX * 2 / 3,
 /* Configure Window B to compare Channel 2 (inside window) */
    .compare_b_channel = ADC_WINDOW_B_CHANNEL_2,
    .compare_b_mode    = ADC_WINDOW_B_MODE_GREATER_THAN_OR_INSIDE,
 /* Set reference voltage levels for Window B */
    .compare_b_ref_low = ADC_SCAN_MAX / 4,
    .compare_b_ref_high = ADC_SCAN_MAX * 3 / 4,
};
void adc0_callback (adc_callback_args_t * p_args)
{
 if (ADC_EVENT_WINDOW_COMPARE_A == p_args->event)
    {
 /* Get channel that met the comparison criteria */
 adc_channel_t channel = p_args->channel;
 /* Process event here */
 FSP_PARAMETER_NOT_USED(channel);
    }
 else if (ADC_EVENT_WINDOW_COMPARE_B == p_args->event)
    {
 /* Process Window B events here */
    }
 else
    {
 /* ... */
    }
}
void adc_window_compare_example (void)
{
```

```
fsp_err_t err = FSP_SUCCESS;
/* Open the ADC module */
   err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
   assert(FSP_SUCCESS == err);
/* Set the window compare configuration in the channel config */
   g_adc0_channel_runtime_cfg.p_window_cfg = &g_adc0_window_cfg;
/* The window compare function is configured as part of the scan configuration */
   err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_runtime_cfg);
   assert(FSP_SUCCESS == err);
/* Main program loop - scan the ADC every second */
while (1)
   {
/* Start a scan */
      err = R_ADC_ScanStart(&g_adc0_ctrl);
     assert(FSP_SUCCESS == err);
/* Delay; any compare events will be handled by the callback */
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
   }
}
```

Data Structures

struct	adc_sample_state_t
struct	adc_window_cfg_t
struct	adc_extended_cfg_t
struct	adc_channel_cfg_t
struct	adc_instance_ctrl_t

Enumerations

enum	adc_mask_t
enum	adc_add_t
enum	adc_clear_t
enum	adc_vref_control_t

enum	adc_sample_state_reg_t
enum	adc_compare_cfg_t
enum	adc_window_b_channel_t
enum	adc_window_b_mode_t
enum	adc_group_a_t
enum	adc_double_trigger_t

Data Structure Documentation

◆ adc_sample_state_t

struct adc_sample_state_t

ADC sample state configuration

Data Fields		
adc_sample_state_reg_t	reg_id	Sample state register ID.
uint8_t	num_states	Number of sampling states for conversion. Ch16-20/21 use the same value.

◆ adc_window_cfg_t

struct adc_window_cfg_t

ADC Window Compare configuration

Data Fields		
uint32_t	compare_mask	Channel mask to compare with Window A.
uint32_t	compare_mode_mask	Per-channel condition mask for Window A.
adc_compare_cfg_t	compare_cfg	Window Compare configuration.
uint16_t	compare_ref_low	Window A lower reference value.
uint16_t	compare_ref_high	Window A upper reference value.
uint16_t	compare_b_ref_low	Window B lower reference value.
uint16_t	compare_b_ref_high	Window A upper reference value.
adc_window_b_channel_t	compare_b_channel	Window B channel.
adc_window_b_mode_t	compare_b_mode	Window B condition setting.

◆ adc_extended_cfg_t

struct adc_extended_cfg_t		
Extended configuration structure for ADC.		
Data Fields		
adc_add_t	add_average_count	Add or average samples.
adc_clear_t	clearing	Clear after read.
adc_trigger_t	trigger_group_b	Group B trigger source; valid only for group mode.
adc_double_trigger_t	double_trigger_mode	Double-trigger mode setting.
adc_vref_control_t	adc_vref_control	VREFADC output voltage control.
uint8_t	enable_adbuf	Enable ADC Ring Buffer, Valid only to use along with DMAC transfer.
IRQn_Type	window_a_irq	IRQ number for Window Compare A interrupts.
IRQn_Type	window_b_irq	IRQ number for Window Compare B interrupts.
uint8_t	window_a_ipl	Priority for Window Compare A interrupts.
uint8_t	window_b_ipl	Priority for Window Compare B interrupts.

◆ adc_channel_cfg_t

struct adc_channel_cfg_t		
ADC channel(s) configuration		
Data Fields		
uint32_t	scan_mask	Channels/bits: bit 0 is ch0; bit 15 is ch15.
uint32_t	scan_mask_group_b	Valid for group modes.
uint32_t	add_mask	Valid if add enabled in Open().
adc_window_cfg_t *	p_window_cfg	Pointer to Window Compare configuration.
adc_group_a_t	priority_group_a	Valid for group modes.
uint8_t	sample_hold_mask	Channels/bits 0-2.
uint8_t	sample_hold_states	Number of states to be used for sample and hold. Affects channels 0-2.

◆ adc_instance_ctrl_t

struct adc_instance_ctrl_t
ADC instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open().

Enumeration Type Documentation

◆ adc_mask_t

enum adc_mask_t	
For ADC Scan configuration adc_channel_cfg_t::scan_mask, adc_channel_cfg_t::scan_mask_group_b , adc_channel_cfg_t::add_mask and adc_channel_cfg_t::sample_hold_mask. Use bitwise OR to combine these masks for desired channels and sensors.	
Enumerator	
ADC_MASK_OFF	No channels selected.
ADC_MASK_CHANNEL_0	Channel 0 mask.
ADC_MASK_CHANNEL_1	Channel 1 mask.
ADC_MASK_CHANNEL_2	Channel 2 mask.
ADC_MASK_CHANNEL_3	Channel 3 mask.
ADC_MASK_CHANNEL_4	Channel 4 mask.
ADC_MASK_CHANNEL_5	Channel 5 mask.
ADC_MASK_CHANNEL_6	Channel 6 mask.
ADC_MASK_CHANNEL_7	Channel 7 mask.
ADC_MASK_CHANNEL_8	Channel 8 mask.
ADC_MASK_CHANNEL_9	Channel 9 mask.
ADC_MASK_CHANNEL_10	Channel 10 mask.
ADC_MASK_CHANNEL_11	Channel 11 mask.
ADC_MASK_CHANNEL_12	Channel 12 mask.
ADC_MASK_CHANNEL_13	Channel 13 mask.
ADC_MASK_CHANNEL_14	Channel 14 mask.
ADC_MASK_CHANNEL_15	Channel 15 mask.

ADC_MASK_CHANNEL_16	Channel 16 mask.
ADC_MASK_CHANNEL_17	Channel 17 mask.
ADC_MASK_CHANNEL_18	Channel 18 mask.
ADC_MASK_CHANNEL_19	Channel 19 mask.
ADC_MASK_CHANNEL_20	Channel 20 mask.
ADC_MASK_CHANNEL_21	Channel 21 mask.
ADC_MASK_CHANNEL_22	Channel 22 mask.
ADC_MASK_CHANNEL_23	Channel 23 mask.
ADC_MASK_CHANNEL_24	Channel 24 mask.
ADC_MASK_CHANNEL_25	Channel 25 mask.
ADC_MASK_CHANNEL_26	Channel 26 mask.
ADC_MASK_CHANNEL_27	Channel 27 mask.
ADC_MASK_TEMPERATURE	Temperature sensor channel mask.
ADC_MASK_VOLT	Voltage reference channel mask.
ADC_MASK_SENSORS	All sensor channel mask.

◆ adc_add_t

enum adc_add_t	
ADC data sample addition and averaging options	
Enumerator	
ADC_ADD_OFF	Addition turned off for channels/sensors.
ADC_ADD_TWO	Add two samples.
ADC_ADD_THREE	Add three samples.
ADC_ADD_FOUR	Add four samples.
ADC_ADD_SIXTEEN	Add sixteen samples.
ADC_ADD_AVERAGE_TWO	Average two samples.
ADC_ADD_AVERAGE_FOUR	Average four samples.
ADC_ADD_AVERAGE_EIGHT	Average eight samples.
ADC_ADD_AVERAGE_SIXTEEN	Add sixteen samples.

◆ adc_clear_t

enum adc_clear_t	
ADC clear after read definitions	
Enumerator	
ADC_CLEAR_AFTER_READ_OFF	Clear after read off.
ADC_CLEAR_AFTER_READ_ON	Clear after read on.

◆ adc_vref_control_t

enum adc_vref_control_t

Enumerator	
ADC_VREF_CONTROL_VREFH	VREFAMPCNT reset value. VREFADC Output voltage is Hi-Z.
ADC_VREF_CONTROL_1_5V_OUTPUT	BGR turn ON. VREFADC Output voltage is 1.5 V.
ADC_VREF_CONTROL_2_0V_OUTPUT	BGR turn ON. VREFADC Output voltage is 2.0 V.
ADC_VREF_CONTROL_2_5V_OUTPUT	BGR turn ON. VREFADC Output voltage is 2.5 V.
ADC_VREF_CONTROL_AVCC0_AVSS0	High potential is AVCC0, low potential is AVSS0.
ADC_VREF_CONTROL_VREFH0_AVSS0	High potential is VREFH0, low potential is AVSS0.
ADC_VREF_CONTROL_IVREF_AVSS0	High potential is internal reference voltage, low potential is AVSS0. When the high potential is set to the internal reference voltage, wait 5 us after R_ADC_Open() to start an ADC measurement.
ADC_VREF_CONTROL_AVCC0_VREFL0	High potential is AVCC0, low potential is VREFL0.
ADC_VREF_CONTROL_VREFH0_VREFL0	High potential is VREFH0, low potential is VREFL0.
ADC_VREF_CONTROL_IVREF_VREFL0	High potential is internal reference voltage, low potential is VREFL0. When the high potential is set to the internal reference voltage, wait 5 us after R_ADC_Open() to start an ADC measurement.

◆ adc_sample_state_reg_t

enum adc_sample_state_reg_t
ADC sample state registers

Enumerator	
ADC_SAMPLE_STATE_CHANNEL_0	Sample state register channel 0.
ADC_SAMPLE_STATE_CHANNEL_1	Sample state register channel 1.
ADC_SAMPLE_STATE_CHANNEL_2	Sample state register channel 2.
ADC_SAMPLE_STATE_CHANNEL_3	Sample state register channel 3.
ADC_SAMPLE_STATE_CHANNEL_4	Sample state register channel 4.
ADC_SAMPLE_STATE_CHANNEL_5	Sample state register channel 5.
ADC_SAMPLE_STATE_CHANNEL_6	Sample state register channel 6.
ADC_SAMPLE_STATE_CHANNEL_7	Sample state register channel 7.
ADC_SAMPLE_STATE_CHANNEL_8	Sample state register channel 8.
ADC_SAMPLE_STATE_CHANNEL_9	Sample state register channel 9.
ADC_SAMPLE_STATE_CHANNEL_10	Sample state register channel 10.
ADC_SAMPLE_STATE_CHANNEL_11	Sample state register channel 11.
ADC_SAMPLE_STATE_CHANNEL_12	Sample state register channel 12.
ADC_SAMPLE_STATE_CHANNEL_13	Sample state register channel 13.
ADC_SAMPLE_STATE_CHANNEL_14	Sample state register channel 14.
ADC_SAMPLE_STATE_CHANNEL_15	Sample state register channel 15.
ADC_SAMPLE_STATE_CHANNEL_16_TO_31	Sample state register channel 16 to 31.

◆ adc_compare_cfg_t

enum adc_compare_cfg_t
ADC comparison settings

◆ adc_window_b_channel_t

enum adc_window_b_channel_t
ADC Window B channel

◆ adc_window_b_mode_t

enum adc_window_b_mode_t
ADC Window B comparison mode

◆ adc_group_a_t

enum adc_group_a_t
ADC action for group A interrupts group B scan. This enumeration is used to specify the priority between Group A and B in group mode.

Enumerator	
ADC_GROUP_A_PRIORITY_OFF	Group A ignored and does not interrupt ongoing group B scan.
ADC_GROUP_A_GROUP_B_WAIT_FOR_TRIGGER	Group A interrupts Group B(single scan) which restarts at next Group B trigger.
ADC_GROUP_A_GROUP_B_RESTART_SCAN	Group A interrupts Group B(single scan) which restarts immediately after Group A scan is complete.
ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN	Group A interrupts Group B(continuous scan) which continues scanning without a new Group B trigger.

◆ adc_double_trigger_t

enum adc_double_trigger_t	
ADC double-trigger mode definitions	
Enumerator	
ADC_DOUBLE_TRIGGER_DISABLED	Double-triggering disabled.
ADC_DOUBLE_TRIGGER_ENABLED	Double-triggering enabled.
ADC_DOUBLE_TRIGGER_ENABLED_EXTENDED	Double-triggering enabled on both ADC ELC events.

Function Documentation

◆ R_ADC_Open()

fsp_err_t R_ADC_Open (adc_ctrl_t * *p_ctrl*, adc_cfg_t const *const *p_cfg*)

Sets the operational mode, trigger sources, interrupt priority, and configurations for the peripheral as a whole. If interrupt is enabled, the function registers a callback function pointer for notifying the user whenever a scan has completed.

Return values

FSP_SUCCESS	Module is ready for use.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_ALREADY_OPEN	The instance control structure has already been opened.
FSP_ERR_IRQ_BSP_DISABLED	A callback is provided, but the interrupt is not enabled.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The requested unit does not exist on this MCU.
FSP_ERR_INVALID_HW_CONDITION	The ADC clock must be at least 1 MHz

◆ R_ADC_ScanCfg()

fsp_err_t R_ADC_ScanCfg (adc_ctrl_t * *p_ctrl*, void const *const *p_channel_cfg*)

Configures the ADC scan parameters. Channel specific settings are set in this function. Pass a pointer to adc_channel_cfg_t to p_channel_cfg.

Note

This starts group B scans if adc_channel_cfg_t::priority_group_a is set to ADC_GROUP_A_GROUP_B_CONTINUOUS_SCAN.

Return values

FSP_SUCCESS	Channel specific settings applied.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ R_ADC_InfoGet()

fsp_err_t R_ADC_InfoGet (adc_ctrl_t * *p_ctrl*, adc_info_t * *p_adc_info*)

Returns the address of the lowest number configured channel and the total number of bytes to be read in order to read the results of the configured channels and return the ELC Event name. If no channels are configured, then a length of 0 is returned.

Also provides the temperature sensor slope and the calibration data for the sensor if available on this MCU. Otherwise, invalid calibration data of 0xFFFFFFFF will be returned.

Note

In group mode, information is returned for group A only. Calculating information for group B is not currently supported.

Return values

FSP_SUCCESS	Information stored in p_adc_info.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ R_ADC_ScanStart()

fsp_err_t R_ADC_ScanStart (adc_ctrl_t * *p_ctrl*)

Starts a software scan or enables the hardware trigger for a scan depending on how the triggers were configured in the R_ADC_Open call. If the unit was configured for ELC or external hardware triggering, then this function allows the trigger signal to get to the ADC unit. The function is not able to control the generation of the trigger itself. If the unit was configured for software triggering, then this function starts the software triggered scan.

Precondition

Call R_ADC_ScanCfg after R_ADC_Open before starting a scan.
On MCUs that support calibration, call R_ADC_Calibrate and wait for calibration to complete before starting a scan.

Return values

FSP_SUCCESS	Scan started (software trigger) or hardware triggers enabled.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit is not initialized.
FSP_ERR_IN_USE	Another scan is still in progress (software trigger).

◆ R_ADC_ScanGroupStart()

fsp_err_t R_ADC_ScanGroupStart (adc_ctrl_t * *p_ctrl*, adc_group_mask_t *group_id*)

adc_api_t::scanStart is not supported on the ADCH. Use scanStart instead.

Return values

FSP_ERR_UNSUPPORTED	Function not supported in this implementation.

◆ R_ADC_ScanStop()

fsp_err_t R_ADC_ScanStop (adc_ctrl_t * *p_ctrl*)

Stops the software scan or disables the unit from being triggered by the hardware trigger (ELC or external) based on what type of trigger the unit was configured for in the R_ADC_Open function. Stopping a hardware triggered scan via this function does not abort an ongoing scan, but prevents the next scan from occurring. Stopping a software triggered scan aborts an ongoing scan.

Return values

FSP_SUCCESS	Scan stopped (software trigger) or hardware triggers disabled.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit is not initialized.

◆ R_ADC_StatusGet()

fsp_err_t R_ADC_StatusGet (adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status*)

Provides the status of any scan process that was started, including scans started by ELC or external triggers and calibration scans on MCUs that support calibration.

Return values

FSP_SUCCESS	Module status stored in the provided pointer p_status
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ R_ADC_Read()

fsp_err_t R_ADC_Read (adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint16_t *const *p_data*)

Reads conversion results from a single channel or sensor.

Return values

FSP_SUCCESS	Data read into provided p_data.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit is not initialized.

◆ R_ADC_Read32()

fsp_err_t R_ADC_Read32 (adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint32_t *const *p_data*)

Reads conversion results from a single channel or sensor register into a 32-bit result.

Return values

FSP_SUCCESS	Data read into provided p_data.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit is not initialized.

◆ R_ADC_SampleStateCountSet()

fsp_err_t R_ADC_SampleStateCountSet (adc_ctrl_t * *p_ctrl*, adc_sample_state_t * *p_sample*)

Sets the sample state count for individual channels. This only needs to be set for special use cases. Normally, use the default values out of reset.

Note

The sample states for the temperature and voltage sensor are set in R_ADC_ScanCfg.

Return values

FSP_SUCCESS	Sample state count updated.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_INITIALIZED	Unit is not initialized.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ R_ADC_Close()

fsp_err_t R_ADC_Close (adc_ctrl_t * *p_ctrl*)

This function ends any scan in progress, disables interrupts, and removes power to the A/D peripheral.

Return values

FSP_SUCCESS	Module closed.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ R_ADC_OffsetSet()

fsp_err_t R_ADC_OffsetSet (adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t *offset*)

adc_api_t::offsetSet is not supported on the ADC.

Return values

FSP_ERR_UNSUPPORTED	Function not supported in this implementation.

◆ R_ADC_Calibrate()

fsp_err_t R_ADC_Calibrate (adc_ctrl_t *const *p_ctrl*, void const * *p_extend*)

Initiates calibration of the ADC on MCUs that require calibration. This function must be called before starting a scan on MCUs that require calibration.

Calibration is complete when the callback is called with ADC_EVENT_CALIBRATION_COMPLETE or when R_ADC_StatusGet returns ADC_STATUS_IDLE. Reference Figure 32.35 "Software flow and operation example of calibration operation." in the RA2A1 manual R01UH0888EJ0100.

ADC calibration time: 12 PCLKB + 774,930 ADCLK. (Reference Table 32.16 "Required calibration time (shown as the number of ADCLK and PCLKB cycles)" in the RA2A1 manual R01UH0888EJ0100. The lowest supported ADCLK is 1MHz.

Calibration will take a minimum of 24 milliseconds at 32 MHz PCLKB and ADCLK. This wait could take up to 780 milliseconds for a 1 MHz PCLKD (ADCLK).

Parameters

[in]	p_ctrl	Pointer to the instance control structure
[in]	p_extend	Unused argument. Pass NULL.

Return values

FSP_SUCCESS	Calibration successfully initiated.
FSP_ERR_INVALID_HW_CONDITION	A scan is in progress or hardware triggers are enabled.
FSP_ERR_UNSUPPORTED	Calibration not supported on this MCU.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ R_ADC_CallbackSet()

fsp_err_t R_ADC_CallbackSet (adc_ctrl_t *const *p_api_ctrl*, void(*)(adc_callback_args_t *) *p_callback*, void const *const *p_context*, adc_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements adc_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.4 Analog to Digital Converter (r_adc_b)

Modules

Functions

fsp_err_t	R_ADC_B_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg)
fsp_err_t	R_ADC_B_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_scan_cfg)
fsp_err_t	R_ADC_B_InfoGet (adc_ctrl_t *p_ctrl, adc_info_t *p_adc_info)
fsp_err_t	R_ADC_B_ScanStart (adc_ctrl_t *p_ctrl)
fsp_err_t	R_ADC_B_ScanGroupStart (adc_ctrl_t *p_ctrl, adc_group_mask_t group_mask)
fsp_err_t	R_ADC_B_ScanStop (adc_ctrl_t *p_ctrl)
fsp_err_t	R_ADC_B_StatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status)
fsp_err_t	R_ADC_B_Read (adc_ctrl_t *p_ctrl, adc_channel_t const channel_id, uint16_t *const p_data)
fsp_err_t	R_ADC_B_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const channel_id, uint32_t *const p_data)

fsp_err_t	R_ADC_B_FifoRead (adc_ctrl_t *p_ctrl, adc_group_mask_t const group_mask, adc_b_fifo_read_t *const p_data)
fsp_err_t	R_ADC_B_Close (adc_ctrl_t *p_ctrl)
fsp_err_t	R_ADC_B_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t offset)
fsp_err_t	R_ADC_B_Calibrate (adc_ctrl_t *const p_ctrl, void const *p_extend)
fsp_err_t	R_ADC_B_CallbackSet (adc_ctrl_t *const p_api_ctrl, void(*p_callback)(adc_callback_args_t *), void const *const p_context, adc_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the ADC_B peripheral on RA MCUs. This module implements the ADC Interface.

Overview

Features

The ADC_B module supports the following features:

- 12 bit resolution
- Configure scans to include:
  - Multiple analog channels
  - Temperature sensor channel
  - Reference Voltage sensor channel
  - Self-Diagnostic channel
- Configurable scan start trigger:
  - Software scan triggers
  - Hardware scan triggers (timer expiration, for example)
  - External scan triggers from the ADTRGn port pins
- Configurable scan modes:
  - Single scan mode, where each trigger starts a single scan
  - Continuous scan mode, where all channels are scanned continuously
  - Synchronous scan mode, where A/D converters operate synchronously
- Variable sampling time
- Self-calibration
- Channel-dedicated sample-and-hold circuits
- Supports adding and averaging converted samples
- Selectable data format (16, 14, 12, and 10-bit)
- Limiter clip function
- User offset adjustment function
- User gain adjustment function
- Built-in FIFO
- Channel-dedicated programable gain amplifier (PGA):
  - Support single-ended or pseudo-differential input
  - 2.5x to 13.33x gain (1.5x to 5.56x for pseudo-differential inputs)
- Optional callback when scan completes, FIFO data is ready, an error occurrs, or other

conditions are triggered.

Configuration

Virtual Channels and Scan Groups

A virtual channel is a group of registers that stores the A/D conversion configuration for a single analog pin. Each virtual channel has a number of options including the channel for conversion, settings for conversion, data processing method and so on.

To perform A/D conversion of an analog pin, the channel associated with the pin must first be assigned to a virtual channel. That virtual channel is then assigned to a scan group, which brings together one or more virtual channels to be converted in sequence with a specified conversion unit.

Note

Analog channels may be assigned to more than one virtual channel. However, a virtual channel can be assigned to only one scan group. When performing A/D conversion on one analog channel in different scan groups or when converting a channel several times within the same scan group, assign several virtual channels to one analog channel.
To avoid data being overwritten, when converting a channel multiple times within the same scan group use R_ADC_B_FifoRead() instead of R_ADC_B_Read().

Configuring a Scan

To perform A/D conversion of a scan group the following should be configured:

- Assign the analog channel for conversion to a virtual channel.
- Assign the virtual channel to a scan group.
- Assign the scan group to an A/D Converter.

Note

Up to 8 virtual channels can be assigned to a scan group. If more than 8 are assigned only the lowest 8 will be targeted for A/D conversion.

Build Time Configurations for r_adc_b

The following build time configurations are defined in fsp_cfg/r_adc_b_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	bsp	If selected, code for parameter checking is included in the build.

Note

The instance configurations available in this driver are too numerous to list here. Please refer to the RA Configuration editor in e2 studio for further details.

Clock Configuration

The ADC_B conversion clock source may be configured to use PCLKC, PCLKA, or GPT with a selectable division ratio. The ADC_B clock may operate between 25 MHz at a minimum and 60 MHz at a maximum.

Pin Configuration

The ANxxx pins are analog input channels that can be used with the ADC_B.

ADTRG0 and ADTRG1 can be used to start scans with an external trigger. When external triggers are used, ADC_B scans begin on the falling edge of the ADTRG pin.

Usage Notes

Self-Calibration

Calibration is required to use this peripheral; call R_ADC_B_Calibrate() after open, and prior to any other function. Self-Calibration should be performed any time ADC operating characteristics are modified, including after reset, releasing module-stop, when returning from software standby or deep software standby mode, or each time the ADC open function is called.

Note

Self-calibration is a non-blocking operation. The application should wait for an ADC_EVENT_CALIBRATION_COMPLETE callback before using other ADC_B functionality.

ADC_B Operational Modes

The driver supports two primary operational modes, single-scan and continuous-scan. In single scan mode, one or more groups are scanned once per trigger. In continuous scan mode, one or more groups are started with a single trigger. Scans continue until R_ADC_B_ScanStop() is called. In each mode, analog groups and virtual channels are converted in ascending order.

Synchronous-Scan Mode

When synchronous operation is enabled all A/D conversions are guaranteed to begin and end based on a user-configured period. When both conversion units are selected for synchronous scan they run from the same period, allowing for consistent timing of simultaneous conversions.

Sample and Hold

Enabling sample and hold on one or more channels instructs the ADC to perform sampling on all channels as soon as a group scan is started. Internal circuitry holds the sampled voltages until the conversion unit is ready.

Note

Each sample and hold unit is connected to two analog channels (0/1, 2/3 etc). When this function is enabled on both members of a pair only one of the two may be scanned at a time.
Enabling sample and hold functionality reduces maximum scan frequency because the sample hold time is added to each scan. Refer to the hardware manual for details on the sample and hold time.

Self-Diagnosis

ADC_B has a built-in self-diagnosis function that can be used to confirm the unit is working correctly. One of three self-diagnosis voltages can be converted and compared to reference values as shown below:

Self-diagnosis mode	Expected reference data	Error condition
Self-diagnosis mode 1	0x0000	Greater or less than reference data (16-bit signed)

Self-diagnosis mode 1	0x8000	Greater than reference data
Self-diagnosis mode 1	0x7FFF	Less than reference data

Add/Average Funtion

The ADC can be configured to automatically add or average a number of conversions into a single result. When enabled only the result of the operation is returned.

Note

When the A/D-converted value addition/average function is used, overflow of conversion data may occur. However, in certain conditions overflow may not be detected. See the A/D Conversion Overflow section below for details.

Data Format

The A/D converter in this peripheral has a resolution of 12 bits. When 16-bit or 14-bit data format is selected, the lower 2 or 4 bits (respectively) of the A/D conversion result are extended bits for data processing, error calibration (Self-Calibration), gain/offset adjustment and the averaging function. For 10-bit data length, the upper 6 bits (bit 15 to bit 10) are always 0.

Limiter Clip

The limit clipping function allows for setting upper and lower bounds on converted data. When the A/D conversion data exceeds the specified upper limit value, it is clipped to the upper limit value. If the A/D conversion data falls below the specified lower limit, it is clipped to the lower limit value.

The upper and lower limits are set in one of eight table entries. To perform limit clip functionality, each virtual channel may (optionally) have one of these entries assigned. Interrupts may be enabled for when limiter clip conditions are triggered.

Note

When 14-/12-/10-bit is selected as the data length of the A/D conversion data, the lower bits are cut based on the data-format selection. When 16-bit format is selected, the data length is not rounded.

Using the Temperature Sensor with the ADC_B

The ADC_B HAL module supports reading the data from the on-chip temperature sensor. The value returned from the sensor can be converted into degrees Celsius or Fahrenheit in the application program using the following formula:

T = (Vs - V1)/slope + T1

- T: Measured temperature (degrees C)
- Vs: Voltage output by the temperature sensor at the time of measurement (Volts)
- T1: Temperature experimentally measured at one point (degrees C)
- V1: Voltage output by the temperature sensor at the time of measurement of T1 (Volts)
- slope: Temperature gradient of the temperature sensor (V/degrees C), given as (V2 - V1) / (T2 - T1)
  - T2: Temperature at the experimental measurement of another point (degrees C)
  - V2: Voltage output by the temperature sensor at the time of measurement of T2 (Volts)

Note

The slope value can be obtained from the hardware manual for each device in the Electrical Characteristics Chapter - TSN Characteristics Table, Temperature slope entry.

User Offset and Gain

The user offset adjustment function adds or subtracts a constant value to or from the A/D conversion data. Virtual channels select an offset from a table of values specified by the user.

The user gain adjustment function multiplies the A/D conversion data by an arbitrary coefficient value. As with offset adjustment, virtual channels may select a gain value from a table specified by the user.

Note

When the offset or gain adjusting functions are used overflow of A/D conversion data may occur. See the A/D Conversion Overflow section below for details.
When 14-/12-/10-bit is selected as the data length of converted data the lower bits of offset values are cut based on the data-format selection.

FIFO

The FIFOs consist of 8 stages and can hold up to 8 A/D conversion data. One FIFO is implemented for each scan group. Each FIFO acts as a ring buffer and data will be lost if the FIFO is not read as needed. Interrupts may be enabled for a specific data storage threshold and on overrun.

Programmable Gain Amplifier

ADC has built-in Programmable Gain Amplifier (PGA). The PGA amplifies an external analog input signal and outputs it to A/D converter, Channel-dedicated sample-and-hold circuit, and High-Speed Analog Comparator (ACMPHS). PGA units are channel specific and utilize two analog input pins per unit. Please refer to the RA Configuration editor in e2 studio or the hardware manual for further details.

Note

When PGA is used, the analog input pin assigned to PGAVSS pin cannot be input to A/D conversion or Channel-dedicated sample-and-hold circuit.

Single-ended input

In single-ended input mode, PGA amplifies the input from PGAIN pin with the specified gain, between ×2.000 to ×13.333. When operating the PGA in single-ended input mode, PGAIN should be connected to the signal source and PGAVSS should be connected to the analog ground (AVSS0). The input voltage to PGAIN must not exceed the range specified in the Electrical Characteristics.

Pseudo-differential input

In pseudo differential input mode, PGA amplifies the difference between PGAIN pin and PGAVSS pin with the specified gain and output the voltage obtained by adding the offset of 0.5 * AVcc. Settable gains are ×1.500, ×2.333, ×4.000, and ×5.667.

When operating the PGA in Pseudo Differential Input Mode, PGAIN should be connected to the signal source, and PGAVSS should be connected to the reference ground of the signal source. The inputs to PGAIN and PGAVSS pins must not exceed the range specified in the Electrical Characteristics.

When Interrupts Are Not Enabled

Interrupts are enabled by default. If scan-complete interrupts are disabled, R_ADC_B_StatusGet() can be used to poll the ADC_B driver to determine when the scan has completed. R_ADC_B_Read() is used to access the converted ADC_B result.

A/D Conversion Overflow

A/D conversion overflow is detected when converted data exceeds the range that can be handled in the specified data format. When overflow occurs, data is restricted to the upper or lower limit value of the specified data format. Overflow is detected in the following cases:

- When the input to the A/D converter exceeds VREFH0 or falls below VREFL0
- When overflow occurs by the internal processing (calculation) for the A/D conversion data due to the following: – Gain Error and Offset Error Calibration – User Gain/Offset adjustment function – When using A/D-Converted Value Addition/Averaging Function – Data Formatting Process

Examples

Basic Example

This is a basic example of minimal use of the ADC_B in an application.

```
/* A channel configuration is generated by the RA Configuration editor based on the
options selected. If additional
 * configurations are desired additional adc_channel_cfg_t elements can be defined
and passed to R_ADC_B_ScanCfg. */
extern const adc_b_scan_cfg_t g_adc_b0_scan_cfg;
void adc_b_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_ADC_B_Open(&g_adc_b0_ctrl, &g_adc_b0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_ADC_B_Calibrate(&g_adc_b0_ctrl, NULL);
    assert(FSP_SUCCESS == err);
 /* Wait for calibration to complete */
 adc_status_t status = {.state = ADC_STATE_SCAN_IN_PROGRESS};
 while ((ADC_STATE_SCAN_IN_PROGRESS == status.state) &&
          (FSP_SUCCESS == err))
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
       err = R_ADC_B_StatusGet(&g_adc_b0_ctrl, &status);
    }
    assert(FSP_SUCCESS == err);
 /* Enable channels. */
```

```
    err = R_ADC_B_ScanCfg(&g_adc_b0_ctrl, &g_adc_b0_scan_cfg);
    assert(FSP_SUCCESS == err);
 /* Start one or more scan groups by calling R_ADC_B_ScanGroupStart(). Alternatively,
all scan groups may be started
  * by calling R_ADC_B_ScanCfg(). */
    (void) R_ADC_B_ScanGroupStart(&g_adc_b0_ctrl, ADC_GROUP_MASK_0);
 /* Wait for conversion to complete. */
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
       (void) R_ADC_B_StatusGet(&g_adc_b0_ctrl, &status);
    }
 /* Read converted data. */
    uint16_t channel_0_conversion_result;
    err = R_ADC_B_Read(&g_adc_b0_ctrl, ADC_CHANNEL_0, &channel_0_conversion_result);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	adc_b_fifo_data_t
struct	adc_b_fifo_read_t
struct	adc_b_group_cfg_t
struct	adc_b_scan_cfg_t
struct	adc_b_isr_cfg_t
struct	adc_b_extended_cfg_t
struct	adc_b_instance_ctrl_t

Enumerations

enum	adc_b_clock_source_t
enum	adc_b_clock_divider_t
enum	adc_b_resolution_t
enum	adc_b_virtual_channel_t

enum	adc_b_channel_mask_t
enum	adc_b_limit_clip_table_id_t
enum	adc_b_unit_id_t
enum	adc_b_unit_mask_t
enum	adc_b_add_avg_mode_t
enum	adc_b_add_avg_count_t
enum	adc_b_gpt_trigger_t
enum	adc_b_external_trigger_t
enum	adc_b_self_diagnosis_mode_t
enum	adc_b_sample_and_hold_mask_t
enum	adc_b_pga_gain_t
enum	adc_b_sampling_state_table_id_t
enum	adc_b_user_gain_table_id_t
enum	adc_b_user_offset_table_selection_id_t

Data Structure Documentation

◆ adc_b_fifo_data_t

struct adc_b_fifo_data_t

ADC FIFO data type

Data Fields

uint32_t	data: 16	Conversion Data.
uint32_t	__pad0__: 8	
adc_channel_t	physical_channel: 7	Channel number for data.
uint32_t	err: 1	Error bit.

◆ adc_b_fifo_read_t

struct adc_b_fifo_read_t

ADC FIFO Read data structure

Data Fields

uint8_t	count	Number of valid FIFO data read.
adc_b_fifo_data_t	fifo_data[8]	FIFO data.

◆ adc_b_group_cfg_t

struct adc_b_group_cfg_t		
ADC Group configuration data		
Data Fields		
adc_group_id_t	scan_group_id	Scan Group ID.
adc_b_unit_id_t	converter_selection	Converter selection.
bool	scan_group_enable	Scan Group enable state.
uint8_t	virtual_channel_count	Virtual Channel count.
bool	scan_end_interrupt_enable	Scan End Interrupt enable.
adc_b_external_trigger_t	external_trigger_enable_mask	External Trigger mask.
elc_peripheral_t	elc_trigger_enable_mask	ELC Trigger mask.
adc_b_gpt_trigger_t	gpt_trigger_enable_mask	GPT Trigger mask.
uint8_t	conversion_start_delay	Conversion start delay.
uint32_t	self_diagnosis_mask	Self-Diagnosis register data.
bool	limit_clip_interrupt_enable	Limiter Clip interrupt enable.
adc_b_virtual_channel_cfg_t **	p_virtual_channels	Pointer to virtual channel configuration array of size virtual_channel_count.

◆ adc_b_scan_cfg_t

struct adc_b_scan_cfg_t		
ADC Scan Group configuraiton		
Data Fields		
uint8_t	group_count	Group Count.
adc_b_group_cfg_t **	p_adc_groups	Pointer to ADC group configuration data.

◆ adc_b_isr_cfg_t

struct adc_b_isr_cfg_t		
ADC ISR configuration structure		
Data Fields		
uint8_t	calibration_end_ipl_adc_0	Calibration end IPL for A/D converter unit 0.
uint8_t	calibration_end_ipl_adc_1	Calibration end IPL for A/D converter unit 1.

uint8_t	conversion_error_ipl_adc_0	Conversion error IPL for A/D converter unit 0.
uint8_t	conversion_error_ipl_adc_1	Conversion error IPL for A/D converter unit 1.
uint8_t	fifo_overflow_ipl	FIFO Overflow IPL.
uint8_t	fifo_read_ipl_group_0	FIFO Read threshold request IPL for Group 0.
uint8_t	fifo_read_ipl_group_1	FIFO Read threshold request IPL for Group 1.
uint8_t	fifo_read_ipl_group_2	FIFO Read threshold request IPL for Group 2.
uint8_t	fifo_read_ipl_group_3	FIFO Read threshold request IPL for Group 3.
uint8_t	fifo_read_ipl_group_4	FIFO Read threshold request IPL for Group 4.
uint8_t	fifo_read_ipl_group_5678	FIFO Read threshold request IPL for Groups 5, 6, 7, and 8.
uint8_t	limit_clip_ipl	Limiter Clip IPL.
uint8_t	overflow_error_ipl_adc_0	Overflow error IPL for A/D converter unit 0.
uint8_t	overflow_error_ipl_adc_1	Overflow error IPL for A/D converter unit 1.
uint8_t	scan_end_ipl_group_0	Scan End IPL for A/D Group 0.
uint8_t	scan_end_ipl_group_1	Scan End IPL for A/D Group 1.
uint8_t	scan_end_ipl_group_2	Scan End IPL for A/D Group 2.
uint8_t	scan_end_ipl_group_3	Scan End IPL for A/D Group 3.
uint8_t	scan_end_ipl_group_4	Scan End IPL for A/D Group 4.
uint8_t	scan_end_ipl_group_5678	Scan End IRQ for A/D Groups 5, 6, 7, and 8.
IRQn_Type	calibration_end_irq_adc_0	Calibration end IRQ for A/D converter unit 0.
IRQn_Type	calibration_end_irq_adc_1	Calibration end IRQ for A/D converter unit 1.
IRQn_Type	conversion_error_irq_adc_0	Conversion error IRQ for A/D converter unit 0.
IRQn_Type	conversion_error_irq_adc_1	Conversion error IRQ for A/D converter unit 1.
IRQn_Type	fifo_overflow_irq	FIFO Overflow IRQ.
IRQn_Type	fifo_read_irq_group_0	FIFO Read threshold request IRQ for Group 0.

IRQn_Type	fifo_read_irq_group_1	FIFO Read threshold request IRQ for Group 1.
IRQn_Type	fifo_read_irq_group_2	FIFO Read threshold request IRQ for Group 2.
IRQn_Type	fifo_read_irq_group_3	FIFO Read threshold request IRQ for Group 3.
IRQn_Type	fifo_read_irq_group_4	FIFO Read threshold request IRQ for Group 4.
IRQn_Type	fifo_read_irq_group_5678	FIFO Read threshold request IRQ for Groups 5, 6, 7, and 8.
IRQn_Type	limit_clip_irq	Limiter Clip IRQ.
IRQn_Type	overflow_error_irq_adc_0	Overflow error IRQ for A/D converter unit 0.
IRQn_Type	overflow_error_irq_adc_1	Overflow error IRQ for A/D converter unit 1.
IRQn_Type	scan_end_irq_group_0	Scan End IRQ for A/D Group 0.
IRQn_Type	scan_end_irq_group_1	Scan End IRQ for A/D Group 1.
IRQn_Type	scan_end_irq_group_2	Scan End IRQ for A/D Group 2.
IRQn_Type	scan_end_irq_group_3	Scan End IRQ for A/D Group 3.
IRQn_Type	scan_end_irq_group_4	Scan End IRQ for A/D Group 4.
IRQn_Type	scan_end_irq_group_5678	Scan End IRQ for A/D Groups 5, 6, 7, and 8.

◆ adc_b_extended_cfg_t

struct adc_b_extended_cfg_t		
ADC extended configuration data		
Data Fields		
adc_b_pga_gain_t	pga_gain[4]	PGA Gain selection.
union adc_b_extended_cfg_t	__unnamed__	
uint32_t	clock_control_data	Clock control register data.
uint32_t	sync_operation_control	Synchronous Operation Control register data.
uint32_t	adc_b_mode	ADC_B mode register data.
uint32_t	scan_group_enable	Scan Group enable register data.
uint32_t	converter_selection_0	Converter Selection register data for groups 0,1,2,3.
uint32_t	converter_selection_1	Converter Selection register data for groups 4,5,6,7.

uint32_t	converter_selection_2	Converter Selection register data for group 8.
uint32_t	fifo_enable_mask	FIFO enable register data.
uint32_t	fifo_interrupt_enable_mask	FIFO interrupt enable register data.
uint32_t	fifo_interrupt_level0	FIFO data threshold interrupt level register data for Group 0 and 1.
uint32_t	fifo_interrupt_level1	FIFO data threshold interrupt level register data for Group 2 and 3.
uint32_t	fifo_interrupt_level2	FIFO data threshold interrupt level register data for Group 4 and 5.
uint32_t	fifo_interrupt_level3	FIFO data threshold interrupt level register data for Group 6 and 7.
uint32_t	fifo_interrupt_level4	FIFO data threshold interrupt level register data for Group 8.
uint32_t	start_trigger_delay_0	Start trigger delay register data for group 0 and 1.
uint32_t	start_trigger_delay_1	Start trigger delay register data for group 2 and 3.
uint32_t	start_trigger_delay_2	Start trigger delay register data for group 4 and 5.
uint32_t	start_trigger_delay_3	Start trigger delay register data for group 6 and 7.
uint32_t	start_trigger_delay_4	Start trigger delay register data for group 8.
uint32_t	calibration_adc_state	Calibration State register data.
uint32_t	calibration_sample_and_hold	Calibration Sample and Hold register data.
const adc_b_isr_cfg_t *	p_isr_cfg	Pointer to ISR configuration.
uint32_t	sampling_state_tables[8]	Sampling State Table register data.
uint8_t	sample_and_hold_enable_mask	Sample and Hold enable register data.
uint32_t	sample_and_hold_config_012	Sample and Hold configuration register data.
uint32_t	sample_and_hold_config_456	Sample and Hold configuration register data.
uint32_t	conversion_state	ADC 0/1 Successive

		Approximation Time Configuration.
int32_t	user_offset_tables[8]	User Offset Table register data.
uint32_t	user_gain_tables[8]	User Gain Table register data.
uint32_t	limiter_clip_interrupt_enable_mask	Limiter clip interrupt enable register data.
uint32_t	limiter_clip_tables[8]	Limiter clip Table register data.

◆ adc_b_instance_ctrl_t

struct adc_b_instance_ctrl_t
ADC instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open().

Enumeration Type Documentation

◆ adc_b_clock_source_t

enum adc_b_clock_source_t	
ADC Clock source selection	
Enumerator	
ADC_B_CLOCK_SOURCE_PCLKC	ADC Clock Source PCLKC.
ADC_B_CLOCK_SOURCE_GPT	ADC Clock Source GPT.
ADC_B_CLOCK_SOURCE_PCLKA	ADC Clock Source PCLKA.

◆ adc_b_clock_divider_t

enum adc_b_clock_divider_t	
ADC clock divider selection	
Enumerator	
ADC_B_CLOCK_DIV_1	ADC Clock Division 1/1.
ADC_B_CLOCK_DIV_2	ADC Clock Division 1/2.
ADC_B_CLOCK_DIV_3	ADC Clock Division 1/3.
ADC_B_CLOCK_DIV_4	ADC Clock Division 1/4.
ADC_B_CLOCK_DIV_5	ADC Clock Division 1/5.
ADC_B_CLOCK_DIV_6	ADC Clock Division 1/6.
ADC_B_CLOCK_DIV_7	ADC Clock Division 1/7.
ADC_B_CLOCK_DIV_8	ADC Clock Division 1/8.

◆ adc_b_resolution_t

enum adc_b_resolution_t	
ADC_B data resolution definitions	
Enumerator	
ADC_B_RESOLUTION_16_BIT	16 bit adc_b resolution
ADC_B_RESOLUTION_14_BIT	14 bit adc_b resolution
ADC_B_RESOLUTION_12_BIT	12 bit adc_b resolution
ADC_B_RESOLUTION_10_BIT	10 bit adc_b resolution

◆ adc_b_virtual_channel_t

enum adc_b_virtual_channel_t	
ADC channels	
Enumerator	
ADC_B_VIRTUAL_CHANNEL_0	ADC B virtual channel 0.
ADC_B_VIRTUAL_CHANNEL_1	ADC B virtual channel 1.
ADC_B_VIRTUAL_CHANNEL_2	ADC B virtual channel 2.
ADC_B_VIRTUAL_CHANNEL_3	ADC B virtual channel 3.
ADC_B_VIRTUAL_CHANNEL_4	ADC B virtual channel 4.
ADC_B_VIRTUAL_CHANNEL_5	ADC B virtual channel 5.
ADC_B_VIRTUAL_CHANNEL_6	ADC B virtual channel 6.
ADC_B_VIRTUAL_CHANNEL_7	ADC B virtual channel 7.
ADC_B_VIRTUAL_CHANNEL_8	ADC B virtual channel 8.
ADC_B_VIRTUAL_CHANNEL_9	ADC B virtual channel 9.
ADC_B_VIRTUAL_CHANNEL_10	ADC B virtual channel 10.
ADC_B_VIRTUAL_CHANNEL_11	ADC B virtual channel 11.
ADC_B_VIRTUAL_CHANNEL_12	ADC B virtual channel 12.
ADC_B_VIRTUAL_CHANNEL_13	ADC B virtual channel 13.
ADC_B_VIRTUAL_CHANNEL_14	ADC B virtual channel 14.
ADC_B_VIRTUAL_CHANNEL_15	ADC B virtual channel 15.
ADC_B_VIRTUAL_CHANNEL_16	ADC B virtual channel 16.
ADC_B_VIRTUAL_CHANNEL_17	ADC B virtual channel 17.
ADC_B_VIRTUAL_CHANNEL_18	ADC B virtual channel 18.
ADC_B_VIRTUAL_CHANNEL_19	ADC B virtual channel 19.
ADC_B_VIRTUAL_CHANNEL_20	ADC B virtual channel 20.

ADC_B_VIRTUAL_CHANNEL_21	ADC B virtual channel 21.
ADC_B_VIRTUAL_CHANNEL_22	ADC B virtual channel 22.
ADC_B_VIRTUAL_CHANNEL_23	ADC B virtual channel 23.
ADC_B_VIRTUAL_CHANNEL_24	ADC B virtual channel 24.
ADC_B_VIRTUAL_CHANNEL_25	ADC B virtual channel 25.
ADC_B_VIRTUAL_CHANNEL_26	ADC B virtual channel 26.
ADC_B_VIRTUAL_CHANNEL_27	ADC B virtual channel 27.
ADC_B_VIRTUAL_CHANNEL_28	ADC B virtual channel 28.
ADC_B_VIRTUAL_CHANNEL_29	ADC B virtual channel 29.
ADC_B_VIRTUAL_CHANNEL_30	ADC B virtual channel 30.
ADC_B_VIRTUAL_CHANNEL_31	ADC B virtual channel 31.
ADC_B_VIRTUAL_CHANNEL_32	ADC B virtual channel 32.
ADC_B_VIRTUAL_CHANNEL_33	ADC B virtual channel 33.
ADC_B_VIRTUAL_CHANNEL_34	ADC B virtual channel 34.
ADC_B_VIRTUAL_CHANNEL_35	ADC B virtual channel 35.
ADC_B_VIRTUAL_CHANNEL_36	ADC B virtual channel 36.

◆ adc_b_channel_mask_t

enum adc_b_channel_mask_t	
ADC channel mask	
Enumerator	
ADC_B_CHANNEL_MASK_0	Channel 0.
ADC_B_CHANNEL_MASK_1	Channel 1.
ADC_B_CHANNEL_MASK_2	Channel 2.
ADC_B_CHANNEL_MASK_3	Channel 3.
ADC_B_CHANNEL_MASK_4	Channel 4.
ADC_B_CHANNEL_MASK_5	Channel 5.
ADC_B_CHANNEL_MASK_6	Channel 6.
ADC_B_CHANNEL_MASK_7	Channel 7.
ADC_B_CHANNEL_MASK_8	Channel 8.
ADC_B_CHANNEL_MASK_9	Channel 9.
ADC_B_CHANNEL_MASK_10	Channel 10.
ADC_B_CHANNEL_MASK_11	Channel 11.
ADC_B_CHANNEL_MASK_12	Channel 12.
ADC_B_CHANNEL_MASK_13	Channel 13.
ADC_B_CHANNEL_MASK_14	Channel 14.
ADC_B_CHANNEL_MASK_15	Channel 15.
ADC_B_CHANNEL_MASK_16	Channel 16.
ADC_B_CHANNEL_MASK_17	Channel 17.
ADC_B_CHANNEL_MASK_18	Channel 18.
ADC_B_CHANNEL_MASK_19	Channel 19.
ADC_B_CHANNEL_MASK_20	Channel 20.

ADC_B_CHANNEL_MASK_21	Channel 21.
ADC_B_CHANNEL_MASK_22	Channel 22.
ADC_B_CHANNEL_MASK_23	Channel 23.
ADC_B_CHANNEL_MASK_24	Channel 24.
ADC_B_CHANNEL_MASK_25	Channel 25.
ADC_B_CHANNEL_MASK_26	Channel 26.
ADC_B_CHANNEL_MASK_27	Channel 27.
ADC_B_CHANNEL_MASK_28	Channel 28.
ADC_B_CHANNEL_MASK_DIAGNOSIS	Self-Diagnosis Channel.
ADC_B_CHANNEL_MASK_TEMPERATURE	Temperature sensor channel.
ADC_B_CHANNEL_MASK_VOLT	Voltage Reference channel.
ADC_B_CHANNEL_MASK_DAC0	DAC 0 Channel.
ADC_B_CHANNEL_MASK_DAC1	DAC 1 Channel.
ADC_B_CHANNEL_MASK_DAC2	DAC 2 Channel.
ADC_B_CHANNEL_MASK_DAC3	DAC 3 Channel.

◆ adc_b_limit_clip_table_id_t

enum adc_b_limit_clip_table_id_t	
ADC limiter clipping table id selection options	
Enumerator	
ADC_B_LIMIT_CLIP_TABLE_SELECTION_NONE	Limiter Clip Disabled.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_0	Limiter Clip Table 0.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_1	Limiter Clip Table 1.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_2	Limiter Clip Table 2.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_3	Limiter Clip Table 3.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_4	Limiter Clip Table 4.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_5	Limiter Clip Table 5.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_6	Limiter Clip Table 6.
ADC_B_LIMIT_CLIP_TABLE_SELECTION_7	Limiter Clip Table 7.

◆ adc_b_unit_id_t

enum adc_b_unit_id_t	
ADC unit selection options	
Enumerator	
ADC_B_UNIT_ID_0	ADC Unit ID 0.
ADC_B_UNIT_ID_1	ADC Unit ID 1.

◆ adc_b_unit_mask_t

enum adc_b_unit_mask_t	
ADC unit selection options	
Enumerator	
ADC_B_UNIT_MASK_0	ADC Unit Mask 0.
ADC_B_UNIT_MASK_1	ADC Unit Mask 1.
ADC_B_UNIT_MASK_UNDEFINED	ADC Unit Mask Unknown.

◆ adc_b_add_avg_mode_t

enum adc_b_add_avg_mode_t	
ADC data sample addition and averaging options	
Enumerator	
ADC_B_ADD_AVERAGE_OFF	Add/Average turned off for channels/sensors.
ADC_B_ADD_AVERAGE_ADDITION_ENABLE	Addition Mode Enabled.
ADC_B_ADD_AVERAGE_AVERAGE_ENABLE	Average Mode Enabled.

◆ adc_b_add_avg_count_t

enum adc_b_add_avg_count_t	
ADC data sample addition and averaging options	
Enumerator	
ADC_B_ADD_AVERAGE_1	Addition turned off for channels/sensors.
ADC_B_ADD_AVERAGE_2	Add/Average 2 samples.
ADC_B_ADD_AVERAGE_4	Add/Average 4 samples.
ADC_B_ADD_AVERAGE_8	Add/Average 8 samples.
ADC_B_ADD_AVERAGE_16	Add/Average 16 samples.
ADC_B_ADD_AVERAGE_32	Add/Average 32 samples.
ADC_B_ADD_AVERAGE_64	Add/Average 64 samples.
ADC_B_ADD_AVERAGE_128	Add/Average 128 samples.
ADC_B_ADD_AVERAGE_256	Add/Average 256 samples.
ADC_B_ADD_AVERAGE_512	Add/Average 512 samples.
ADC_B_ADD_AVERAGE_1024	Add/Average 1024 samples.

◆ adc_b_gpt_trigger_t

enum adc_b_gpt_trigger_t	
ADC GPT Trigger options	
Enumerator	
ADC_B_GPT_TRIGGER_NONE	GPT Trigger Disabled.
ADC_B_GPT_TRIGGER_A0	GPT Trigger A0.
ADC_B_GPT_TRIGGER_A1	GPT Trigger A1.
ADC_B_GPT_TRIGGER_A2	GPT Trigger A2.
ADC_B_GPT_TRIGGER_A3	GPT Trigger A3.
ADC_B_GPT_TRIGGER_A4	GPT Trigger A4.

ADC_B_GPT_TRIGGER_A5	GPT Trigger A5.
ADC_B_GPT_TRIGGER_A6	GPT Trigger A6.
ADC_B_GPT_TRIGGER_A7	GPT Trigger A7.
ADC_B_GPT_TRIGGER_A8	GPT Trigger A8.
ADC_B_GPT_TRIGGER_A9	GPT Trigger A9.
ADC_B_GPT_TRIGGER_B0	GPT Trigger B0.
ADC_B_GPT_TRIGGER_B1	GPT Trigger B1.
ADC_B_GPT_TRIGGER_B2	GPT Trigger B2.
ADC_B_GPT_TRIGGER_B3	GPT Trigger B3.
ADC_B_GPT_TRIGGER_B4	GPT Trigger B4.
ADC_B_GPT_TRIGGER_B5	GPT Trigger B5.
ADC_B_GPT_TRIGGER_B6	GPT Trigger B6.
ADC_B_GPT_TRIGGER_B7	GPT Trigger B7.
ADC_B_GPT_TRIGGER_B8	GPT Trigger B8.
ADC_B_GPT_TRIGGER_B9	GPT Trigger B9.

◆ adc_b_external_trigger_t

enum adc_b_external_trigger_t

ADC External Trigger options

Enumerator	
ADC_B_EXTERNAL_TRIGGER_NONE	External Trigger Disabled.
ADC_B_EXTERNAL_TRIGGER_ADTRG0	External Trigger ADTRG0 Selection.
ADC_B_EXTERNAL_TRIGGER_ADTRG1	External Trigger ADTRG1 Selection.

◆ adc_b_self_diagnosis_mode_t

enum adc_b_self_diagnosis_mode_t	
ADC Self-Diagnosis mode options	
Enumerator	
ADC_B_SELF_DIAGNOSIS_DISABLED	Self-Diagnosis Disabled.
ADC_B_SELF_DIAGNOSIS_MODE_1	Self-Diagnosis Mode 1.
ADC_B_SELF_DIAGNOSIS_MODE_2	Self-Diagnosis Mode 2.
ADC_B_SELF_DIAGNOSIS_MODE_3	Self-Diagnosis Mode 3.

◆ adc_b_sample_and_hold_mask_t

enum adc_b_sample_and_hold_mask_t	
ADC Sample-and-Hold unit enable mask	
Enumerator	
ADC_B_SAMPLE_AND_HOLD_MASK_NONE	Sample-and-Hold Disabled.
ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_0	Sample-and-Hold Unit 0.
ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_1	Sample-and-Hold Unit 1.
ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_2	Sample-and-Hold Unit 2.
ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_4	Sample-and-Hold Unit 3.
ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_5	Sample-and-Hold Unit 4.
ADC_B_SAMPLE_AND_HOLD_MASK_UNIT_6	Sample-and-Hold Unit 5.

◆ adc_b_pga_gain_t

enum adc_b_pga_gain_t	
ADC PGA Gain	
Enumerator	
ADC_B_PGA_GAIN_DISABLED	PGA Gain Disabled.
ADC_B_PGA_GAIN_DIFFERENTIAL_1_500	PGA Gain Setting 1.500.
ADC_B_PGA_GAIN_DIFFERENTIAL_2_333	PGA Gain Setting 2.333.
ADC_B_PGA_GAIN_DIFFERENTIAL_4_000	PGA Gain Setting 4.000.
ADC_B_PGA_GAIN_DIFFERENTIAL_5_667	PGA Gain Setting 5.667.
ADC_B_PGA_GAIN_SINGLE_ENDED_2_500	PGA Gain Setting 2.500.
ADC_B_PGA_GAIN_SINGLE_ENDED_2_667	PGA Gain Setting 2.667.
ADC_B_PGA_GAIN_SINGLE_ENDED_2_857	PGA Gain Setting 2.857.
ADC_B_PGA_GAIN_SINGLE_ENDED_3_077	PGA Gain Setting 3.077.
ADC_B_PGA_GAIN_SINGLE_ENDED_3_333	PGA Gain Setting 3.333.
ADC_B_PGA_GAIN_SINGLE_ENDED_3_636	PGA Gain Setting 3.636.
ADC_B_PGA_GAIN_SINGLE_ENDED_4_000	PGA Gain Setting 4.000.
ADC_B_PGA_GAIN_SINGLE_ENDED_4_444	PGA Gain Setting 4.444.
ADC_B_PGA_GAIN_SINGLE_ENDED_5_000	PGA Gain Setting 5.000.
ADC_B_PGA_GAIN_SINGLE_ENDED_5_714	PGA Gain Setting 5.714.
ADC_B_PGA_GAIN_SINGLE_ENDED_6_667	PGA Gain Setting 6.667.
ADC_B_PGA_GAIN_SINGLE_ENDED_8_000	PGA Gain Setting 8.000.
ADC_B_PGA_GAIN_SINGLE_ENDED_10_000	PGA Gain Setting 10.000.
ADC_B_PGA_GAIN_SINGLE_ENDED_13_333	PGA Gain Setting 13.333.

◆ adc_b_sampling_state_table_id_t

enum adc_b_sampling_state_table_id_t	
ADC Sampling State table selection options	
Enumerator	
ADC_B_SAMPLING_STATE_TABLE_0	Sampling State Table 0.
ADC_B_SAMPLING_STATE_TABLE_1	Sampling State Table 1.
ADC_B_SAMPLING_STATE_TABLE_2	Sampling State Table 2.
ADC_B_SAMPLING_STATE_TABLE_3	Sampling State Table 3.
ADC_B_SAMPLING_STATE_TABLE_4	Sampling State Table 4.
ADC_B_SAMPLING_STATE_TABLE_5	Sampling State Table 5.
ADC_B_SAMPLING_STATE_TABLE_6	Sampling State Table 6.
ADC_B_SAMPLING_STATE_TABLE_7	Sampling State Table 7.
ADC_B_SAMPLING_STATE_TABLE_8	Sampling State Table 8.
ADC_B_SAMPLING_STATE_TABLE_9	Sampling State Table 9.
ADC_B_SAMPLING_STATE_TABLE_10	Sampling State Table 10.
ADC_B_SAMPLING_STATE_TABLE_11	Sampling State Table 12.
ADC_B_SAMPLING_STATE_TABLE_12	Sampling State Table 13.
ADC_B_SAMPLING_STATE_TABLE_13	Sampling State Table 14.
ADC_B_SAMPLING_STATE_TABLE_14	Sampling State Table 15.
ADC_B_SAMPLING_STATE_TABLE_15	Sampling State Table 16.

◆ adc_b_user_gain_table_id_t

enum adc_b_user_gain_table_id_t

ADC User Gain table options

Enumerator	
ADC_B_USER_GAIN_TABLE_SELECTION_DISABLED	User Gain disabled.
ADC_B_USER_GAIN_TABLE_SELECTION_0	User Gain table 0.
ADC_B_USER_GAIN_TABLE_SELECTION_1	User Gain table 1.
ADC_B_USER_GAIN_TABLE_SELECTION_2	User Gain table 2.
ADC_B_USER_GAIN_TABLE_SELECTION_3	User Gain table 3.
ADC_B_USER_GAIN_TABLE_SELECTION_4	User Gain table 4.
ADC_B_USER_GAIN_TABLE_SELECTION_5	User Gain table 5.
ADC_B_USER_GAIN_TABLE_SELECTION_6	User Gain table 6.
ADC_B_USER_GAIN_TABLE_SELECTION_7	User Gain table 7.

◆ adc_b_user_offset_table_selection_id_t

enum adc_b_user_offset_table_selection_id_t	
ADC User Offset table options	
Enumerator	
ADC_B_USER_OFFSET_TABLE_SELECTION_DISABLED	User Offset disabled.
ADC_B_USER_OFFSET_TABLE_SELECTION_0	User Offset table 0.
ADC_B_USER_OFFSET_TABLE_SELECTION_1	User Offset table 1.
ADC_B_USER_OFFSET_TABLE_SELECTION_2	User Offset table 2.
ADC_B_USER_OFFSET_TABLE_SELECTION_3	User Offset table 3.
ADC_B_USER_OFFSET_TABLE_SELECTION_4	User Offset table 4.
ADC_B_USER_OFFSET_TABLE_SELECTION_5	User Offset table 5.
ADC_B_USER_OFFSET_TABLE_SELECTION_6	User Offset table 6.
ADC_B_USER_OFFSET_TABLE_SELECTION_7	User Offset table 7.

Function Documentation

◆ R_ADC_B_Open()

```
fsp_err_t R_ADC_B_Open ( adc_ctrl_t * p_ctrl, adc_cfg_t const *const p_cfg )
```

Sets the operational mode, trigger sources, interrupt priority, and configurations for the peripheral as a whole. If provided, the function registers a callback function pointer for notifying the user whenever a scan has completed, error has occurred, FIFO read request is generated, or other ADC interrupt event occurrs. Implements adc_api_t::open.

Return values

FSP_SUCCESS	Module is ready for use.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_ALREADY_OPEN	The instance control structure has already been opened.

◆ R_ADC_B_ScanCfg()

fsp_err_t R_ADC_B_ScanCfg (adc_ctrl_t * *p_ctrl*, void const *const *p_scan_cfg*)

Configures the ADC_B scan parameters. Channel specific settings are set in this function. Pass a pointer to adc_b_scan_cfg_t to p_channel_cfg. Implements adc_api_t::scanCfg.

Note

This starts group B scans if adc_b_scan_cfg_t::priority_group_a is set to ADC_B_GROUP_A_GROUP_B_CONTINUOUS_SCAN.

Return values

FSP_SUCCESS	Channel specific settings applied.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_INVALID_STATE	Invalid Scan Configuration.

◆ R_ADC_B_InfoGet()

fsp_err_t R_ADC_B_InfoGet (adc_ctrl_t * *p_ctrl*, adc_info_t * *p_adc_info*)

Provides the temperature sensor slope and the calibration data for the sensor if available on this MCU. Otherwise, invalid calibration data of 0xFFFFFFFF will be returned.

Return values

FSP_SUCCESS	Info is read into p_adc_info.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.

◆ R_ADC_B_ScanStart()

fsp_err_t R_ADC_B_ScanStart (adc_ctrl_t * *p_ctrl*)

Enables the hardware trigger for a scan depending on how the triggers were configured in the R_ADC_B_ScanCfg call. If the unit was configured for ELC, GPT, or external hardware triggering, then this function allows the trigger signal to get to the ADC unit. The function is not able to control the generation of the trigger itself. If the unit was configured for software triggering, This function was added to this ADC version for compatability with r_adc driver. For additional flexibility, it is recommended R_ADC_B_GroupScanStart.

Precondition

Call R_ADC_B_ScanCfg after R_ADC_B_Open before starting a scan.
Call R_ADC_B_Calibrate and wait for calibration to complete before starting a scan.

Return values

FSP_SUCCESS	Scan started (software trigger) or hardware triggers enabled.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_INVALID_ARGUMENT	No hardware triggers configured for groups.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.

◆ R_ADC_B_ScanGroupStart()

fsp_err_t R_ADC_B_ScanGroupStart (adc_ctrl_t * *p_ctrl*, adc_group_mask_t *group_mask*)

Starts a software scan or enables the hardware trigger for a scan depending on how triggers were configured. If the group was configured for ELC, GPT, or external hardware triggering then this function allows the trigger signal to get to the ADC unit. The function itself is not able to control the generation of peripheral triggers. If the unit was configured for software triggering, then this function starts the software triggered scan.

Precondition

Call R_ADC_B_ScanCfg after R_ADC_B_Open before starting a scan.
Call R_ADC_B_Calibrate and wait for calibration to complete before starting a scan.

Return values

FSP_SUCCESS	Scan started (software trigger) or hardware triggers enabled.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_INVALID_ARGUMENT	An invalid group has been provided.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.
FSP_ERR_IN_USE	Another scan is still in progress (software trigger).

◆ R_ADC_B_ScanStop()

fsp_err_t R_ADC_B_ScanStop (adc_ctrl_t * *p_ctrl*)

Disables the hardware trigger for a scan and immediately stops all active converters. This function will abortall active conversions.

Return values

FSP_SUCCESS	All scans stopped.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_INVALID_ARGUMENT	No hardware triggers configured for groups.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.

◆ R_ADC_B_StatusGet()

fsp_err_t R_ADC_B_StatusGet (adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status*)

Provides the status of any scan process that was started, including scans started by ELC or external triggers and calibration scans on MCUs that support calibration.

Return values

FSP_SUCCESS	Module status stored in the provided pointer p_status
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ R_ADC_B_Read()

fsp_err_t R_ADC_B_Read (adc_ctrl_t * *p_ctrl*, adc_channel_t const *channel_id*, uint16_t *const *p_data*)

Reads conversion results from a single channel or sensor.

Return values

FSP_SUCCESS	Data read into provided p_data.
FSP_ERR_INVALID_DATA	Accuracy of data cannot be guaranteed.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.
FSP_ERR_INVALID_CHANNEL	Invalid channel provided.

◆ R_ADC_B_Read32()

fsp_err_t R_ADC_B_Read32 (adc_ctrl_t * *p_ctrl*, adc_channel_t const *channel_id*, uint32_t *const *p_data*)

Reads conversion results from a single channel or sensor register into a 32-bit result.

Return values

FSP_SUCCESS	Data read into provided p_data.
FSP_ERR_INVALID_DATA	Accuracy of data cannot be guaranteed.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.
FSP_ERR_INVALID_CHANNEL	Invalid channel provided.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- adc_api_t::read

◆ R_ADC_B_FifoRead()

fsp_err_t R_ADC_B_FifoRead (adc_ctrl_t * *p_ctrl*, adc_group_mask_t const *group_mask*, adc_b_fifo_read_t *const *p_data*)

Reads conversion results from FIFO for the given group mask.

Return values

FSP_SUCCESS	Data read into provided p_data.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.
FSP_ERR_NOT_INITIALIZED	Unit not initialized.
FSP_ERR_INVALID_ARGUMENT	Invalid group provided.
FSP_ERR_UNDERFLOW	FIFO empty.

◆ R_ADC_B_Close()

fsp_err_t R_ADC_B_Close (adc_ctrl_t * *p_ctrl*)

This function ends any scan in progress, disables interrupts, and removes power to the A/D peripheral.

Return values

FSP_SUCCESS	Module closed.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ R_ADC_B_OffsetSet()

fsp_err_t R_ADC_B_OffsetSet (adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t *offset*)

adc_api_t::offsetSet is not supported on the ADC_B.

Return values

FSP_ERR_UNSUPPORTED	Function not supported in this implementation.

◆ R_ADC_B_Calibrate()

fsp_err_t R_ADC_B_Calibrate (adc_ctrl_t *const *p_ctrl*, void const * *p_extend*)

Initiates calibration of the ADC_B. This function must be called before starting a scan and again whenever ADC_B configuration or state is changed.

Parameters

[in]	p_ctrl	Pointer to the instance control structure
[in]	p_extend	Unused argument.

Return values

FSP_SUCCESS	Calibration successfully initiated.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_INVALID_HW_CONDITION	Error occurred during calibration.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ R_ADC_B_CallbackSet()

fsp_err_t R_ADC_B_CallbackSet (adc_ctrl_t *const *p_api_ctrl*, void(*)(adc_callback_args_t *) *p_callback*, void const *const *p_context*, adc_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements adc_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.5 Asynchronous General Purpose Timer (r_agt)

Modules

Functions

fsp_err_t	R_AGT_Close (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_AGT_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts)
fsp_err_t	R_AGT_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)
fsp_err_t	R_AGT_Reset (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_AGT_Start (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_AGT_Enable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_AGT_Disable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_AGT_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)
fsp_err_t	R_AGT_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)

fsp_err_t	R_AGT_Stop (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_AGT_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)
fsp_err_t	R_AGT_CallbackSet (timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void const *const p_context, timer_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the AGT peripheral on RA MCUs. This module implements the Timer Interface.

Overview

Features

The AGT module has the following features:

- Supports periodic mode, one-shot mode, and PWM mode.
- Signal can be output to a pin.
- Configurable period (counts per timer cycle).
- Configurable duty cycle in PWM mode.
- Configurable clock source, including PCLKB, LOCO, SUBCLK, and external sources input to AGTIO.
- Supports runtime reconfiguration of period.
- Supports runtime reconfiguration of duty cycle in PWM mode.
- Supports counting based on an external clock input to AGTIO.
- Supports debounce filter on AGTIO pins.
- Supports measuring pulse width or pulse period.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value.

Selecting a Timer

RA MCUs have two timer peripherals: the General PWM Timer (GPT) and the Asynchronous General Purpose Timer (AGT). When selecting between them, consider these factors:

	GPT	AGT
Low Power Modes	The GPT can operate in sleep mode.	The AGT can operate in all low power modes (when count source is LOCO or subclock).
Available Channels	The number of GPT channels is device specific. All currently supported MCUs have at least 7 GPT channels.	All MCUs have 2 AGT channels.
Timer Resolution	All MCUs have at least one 32-bit GPT timer.	The AGT timers are 16-bit timers.
Clock Source	The GPT runs off PCLKD with a	The AGT runs off PCLKB, LOCO,

	configurable divider up to 1024. It can also be configured to count ELC events or external pulses.	or subclock with a configurable divider up to 8 for PCLKB or up to 128 for LOCO or subclock.

Configuration

Build Time Configurations for r_agt

The following build time configurations are defined in fsp_cfg/r_agt_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Pin Output Support	• Disabled • Enabled	Disabled	If selected code for outputting a waveform to a pin is included in the build.
Pin Input Support	• Disabled • Enabled	Disabled	Enable input support to use pulse width measurement mode, pulse period measurement mode, or input from P402, P402, or AGTIO.

Configurations for Timers > Timer, Low-Power (r_agt)

This module can be added to the Stacks tab via New Stack > Timers > Timer, Low-Power (r_agt). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_timer0	Module name.
General > Channel	Channel number does not exist	0	Physical hardware channel.
General > Mode	• Periodic • One-Shot • PWM	Periodic	Mode selection. Note: One-shot mode is implemented in software. ISR's must be enabled for one shot even if callback is unused.
General > Period	Value must be non-negative	0x10000	Specify the timer period based on the selected unit.

			When the unit is set to 'Raw Counts', setting the period to 0x10000 results in the maximum period at the lowest divisor (fastest timer tick). Set the period to 0x10000 for a free running timer, pulse width measurement or pulse period measurement. Setting the period higher will automatically select a higher divider; the period can be set up to 0x80000 when counting from PCLKB or 0x800000 when counting from LOCO/subclock, which will use a divider of 8 or 128 respectively with the maximum period. If the requested period cannot be achieved, the settings with the largest possible period that is less than or equal to the requested period are used. The theoretical calculated period is printed in a comment in the timer_cfg_t structure.
General > Period Unit	• Raw Counts • Nanoseconds • Microseconds • Milliseconds • Seconds • Hertz • Kilohertz	Raw Counts	Unit of the period specified above
General > Count Source	• PCLKB • LOCO • SUBCLOCK • AGT Underflow • P402 Input • P403 Input • AGTIO Input	PCLKB	AGT counter clock source. NOTE: The divisor is calculated automatically based on the selected period. See agt_count_source_t documentation for details.

Output > Duty Cycle Percent (only applicable in PWM mode)	Value must be between 0 and 100	50	Specify the timer duty cycle percent. Only used in PWM mode.
Output > AGTOA Output	• Disabled • Start Level Low • Start Level High	Disabled	Configure AGTOA output.
Output > AGTOB Output	• Disabled • Start Level Low • Start Level High	Disabled	Configure AGTOB output.
Output > AGTO Output	• Disabled • Start Level Low • Start Level High	Disabled	Configure AGTO output.
Input > Measurement Mode	• Measure Disabled • Measure Low Level Pulse Width • Measure High Level Pulse Width • Measure Pulse Period	Measure Disabled	Select if the AGT should be used to measure pulse width or pulse period. In high level pulse width measurement mode, the AGT counts when AGTIO is high and starts counting immediately in the middle of a pulse if AGTIO is high when R_AGT_Start() is called. In low level pulse width measurement mode, the AGT counts when AGTIO is low and could start counting in the middle of a pulse if AGTIO is low when R_AGT_Start() is called.
Input > Input Filter	• No Filter • Filter sampled at PCLKB • Filter sampled at PCLKB / 8 • Filter sampled at PCLKB / 32	No Filter	Input filter, applies AGTIO in pulse period measurement, pulse width measurement, or event counter mode. The filter requires the signal to be at the same level for 3 successive reads at the specified filter frequency.
Input > Enable Pin	• Enable Pin Not Used • Enable Pin Active Low • Enable Pin Active High	Enable Pin Not Used	Select active edge for the AGTEE pin if used. Only applies if the count source is P402, P403 or AGTIO.

Input > Trigger Edge	• Trigger Edge Rising • Trigger Edge Falling • Trigger Edge Both	Trigger Edge Rising	Select the trigger edge. Applies if measurement mode is pulse period, or if the count source is P402, P403, or AGTIO. Do not select Trigger Edge Both with pulse period measurement.
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the timer period elapses.
Interrupts > Underflow Interrupt Priority	MCU Specific Options		Timer interrupt priority.

Clock Configuration

The AGT clock is based on the PCLKB, LOCO, or Subclock frequency. You can set the clock frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

Pin Configuration

This module can use the AGTOA and AGTOB pins as output pins for periodic, one-shot, or PWM signals.

For input capture, the input signal must be applied to the AGTIOn pin.

For event counting, the AGTEEn enable pin is optional.

Timer Period

The RA Configuration editor will automatically calculate the period count value and source clock divider based on the selected period time, units, and clock speed.

When the selected unit is "Raw counts", the maximum allowed period setting varies depending on the selected clock source:

Clock source	Maximum period (counts)
LOCO/Subclock	0x800000
PCLKB	0x80000
All other sources	0x10000

Note

Though the AGT is a 16-bit timer, because the period interrupt occurs when the counter underflows, setting the period register to 0 results in an effective period of 1 count. For this reason all user-provided raw count values reflect the actual number of period counts (not the raw register values).

Usage Notes

Starting and Stopping the AGT

After starting or stopping the timer, AGT registers cannot be accessed until the AGT state is updated after 3 AGTCLK cycles. If another AGT function is called before the 3 AGTCLK period elapses, the function spins waiting for the AGT state to update. The required wait time after starting or stopping the timer can be determined using the frequency of AGTCLK, which is derived from timer_cfg_t::source_div and agt_extended_cfg_t::count_source.

The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning

> The subclock can take seconds to stabilize. The RA startup code does not wait for subclock stabilization unless the subclock is the main clock source. When running AGT or RTC off the subclock, the application must ensure the subclock is stable before starting operation.

Low Power Modes

The AGT1 (channel 1 only) can be used to enter snooze mode or to wake the MCU from snooze, software standby, or deep software standby modes when a counter underflow occurs. The compare match A and B events can also be used to wake from software standby or snooze modes.

One-Shot Mode

The AGT timer does not support one-shot mode natively. One-shot mode is achieved by stopping the timer in the interrupt service routine before the callback is called. If the interrupt is not serviced before the timer period expires again, the timer generates more than one event. The callback is only called once in this case, but multiple events may be generated if the timer is linked to the Data Transfer Controller (r_dtc).

One-Shot Mode Output

The output waveform in one-shot mode is one AGT clock cycle less than the configured period. The configured period must be at least 2 counts to generate an output pulse.

Examples of one-shot signals that can be generated by this module are shown below:

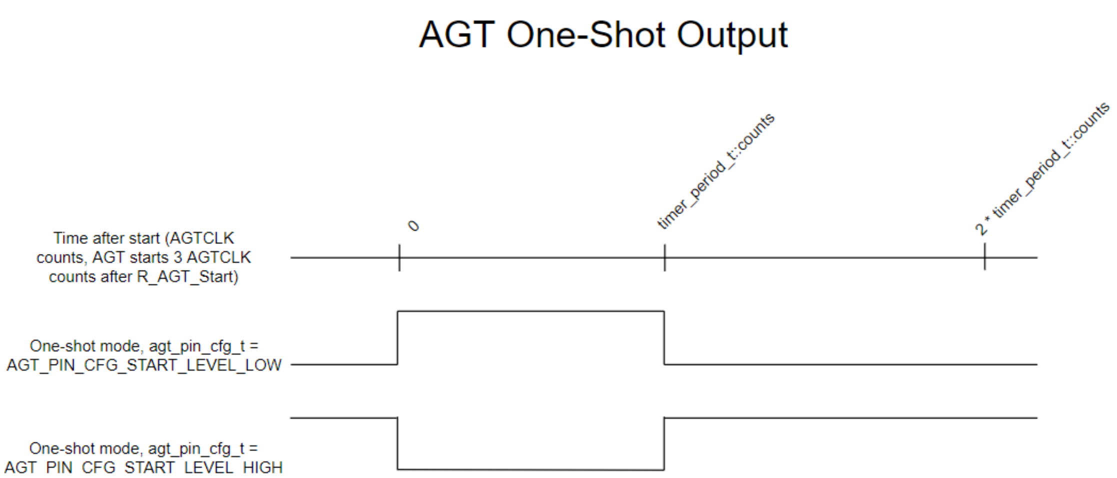


Figure 142: AGT One-Shot Output

Periodic Output

The AGTOA or AGTOB pin toggles twice each time the timer expires in periodic mode. This is achieved by defining a PWM wave at a 50 percent duty cycle so that the period of the resulting square (from rising edge to rising edge) matches the period of the AGT timer. Since the periodic output is actually a PWM output, the time at the stop level is one cycle shorter than the time opposite the stop level for odd period values.

Examples of periodic signals that can be generated by this module are shown below:

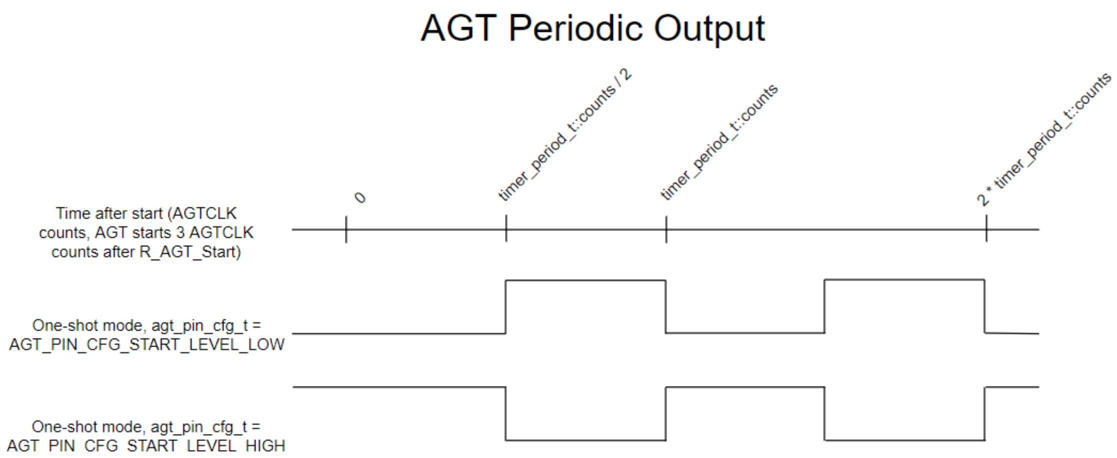


Figure 143: AGT Periodic Output

PWM Output

This module does not support in phase PWM output. The PWM output signal is low at the beginning of the cycle and high at the end of the cycle.

Examples of PWM signals that can be generated by this module are shown below:

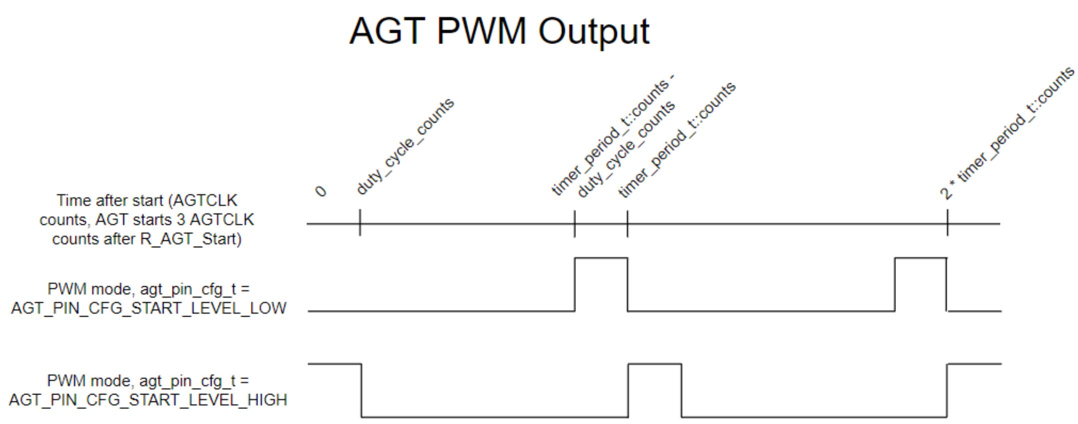


Figure 144: AGT PWM Output

Triggering ELC Events with AGT

The AGT timer can trigger the start of other peripherals. The Event Link Controller (r_elc) guide provides a list of all available peripherals.

Examples

AGT Basic Example

This is a basic example of minimal use of the AGT in an application.

```
void agt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_AGT_Start(&g_timer0_ctrl);
}
```

AGT Callback Example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

AGT Free Running Counter Example

To use the AGT as a free running counter, select periodic mode and set the the Period to 0xFFFF.

```
void agt_counter_example (void)
```

```
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_AGT_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_AGT_Stop(&g_timer0_ctrl);
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_AGT_StatusGet(&g_timer0_ctrl, &status);
}
```

AGT Input Capture Example

This is an example of using the AGT to capture pulse width or pulse period measurements.

```
/* Example callback called when a capture occurs. */
uint64_t g_captured_time    = 0U;
uint32_t g_capture_overflows = 0U;
void timer_capture_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CAPTURE_A == p_args->event)
    {
 /* (Optional) Get the current period if not known. */
 timer_info_t info;
       (void) R_AGT_InfoGet(&g_timer0_ctrl, &info);
       uint32_t period = info.period_counts;
 /* Process capture from AGTIO. */
       g_captured_time    = ((uint64_t) period * g_capture_overflows) +
p_args->capture;
       g_capture_overflows = 0U;
    }
```

```
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* An overflow occurred during capture. This must be accounted for at the
application layer. */
       g_capture_overflows++;
    }
}
void agt_capture_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable captures. Captured values arrive in the interrupt. */
    (void) R_AGT_Enable(&g_timer0_ctrl);
 /* (Optional) Disable captures. */
    (void) R_AGT_Disable(&g_timer0_ctrl);
}
```

AGT Period Update Example

This an example of updating the period.

```
#define AGT_EXAMPLE_MSEC_PER_SEC (1000)
#define AGT_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void agt_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
```

```
    (void) R_AGT_Start(&g_timer0_ctrl);
 /* Get the source clock frequency (in Hz). There are several ways to do this in FSP:
  * - If LOCO or subclock is chosen in agt_extended_cfg_t::clock_source
  * - The source clock frequency is BSP_LOCO_HZ >> timer_cfg_t::source_div
  * - If PCLKB is chosen in agt_extended_cfg_t::clock_source and the PCLKB frequency
has not changed since reset,
  * - The source clock frequency is BSP_STARTUP_PCLKB_HZ >> timer_cfg_t::source_div
  * - Use the R_AGT_InfoGet function (it accounts for the clock source and divider).
  * - Calculate the current PCLKB frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) and right shift
  * by timer_cfg_t::source_div.
  *
  * This example uses the last option (R_FSP_SystemClockHzGet).
  */
    uint32_t timer_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkb_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) timer_freq_hz * AGT_EXAMPLE_DESIRED_PERIOD_MSEC) /
AGT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
  * period is larger than UINT16_MAX. */
    err = R_AGT_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
}
```

AGT Duty Cycle Update Example

This an example of updating the duty cycle.

```
#define AGT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT (25)
```

```
#define AGT_EXAMPLE_MAX_PERCENT (100)
/* This example shows how to calculate a new duty cycle value at runtime. */
void agt_duty_cycle_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_AGT_Start(&g_timer0_ctrl);
 /* Get the current period setting. */
 timer_info_t info;
    (void) R_AGT_InfoGet(&g_timer0_ctrl, &info);
    uint32_t current_period_counts = info.period_counts;
 /* Calculate the desired duty cycle based on the current period. */
    uint32_t duty_cycle_counts = (current_period_counts *
AGT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                                 AGT_EXAMPLE_MAX_PERCENT;
 /* Set the calculated duty cycle. */
    err = R_AGT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, AGT_OUTPUT_PIN_AGTOA
);
    assert(FSP_SUCCESS == err);
}
```

AGT Cascaded Timers Example

This an example of using underflow from an even AGT channel as the count source for the next channel (in this case, AGT0 and AGT1).

```
/* This example shows how use cascaded timers. The count source for AGT channel 1 is
set to AGT0 underflow. */
void agt_cascaded_timers_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
```

```
/* Initialize the timers in any order. */
    err = R_AGT_Open(&g_timer_channel0_ctrl, &g_timer_channel0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_AGT_Open(&g_timer_channel1_ctrl, &g_timer_channel1_cfg);
    assert(FSP_SUCCESS == err);
/* Start AGT channel 1 first. */
    (void) R_AGT_Start(&g_timer_channel1_ctrl);
    (void) R_AGT_Start(&g_timer_channel0_ctrl);
/* (Optional) Stop AGT channel 0 first. */
    (void) R_AGT_Stop(&g_timer_channel0_ctrl);
    (void) R_AGT_Stop(&g_timer_channel1_ctrl);
/* Read the current counter value. Counter value is in status.counter. */
timer_status_t status;
    (void) R_AGT_StatusGet(&g_timer_channel1_ctrl, &status);
}
```

Data Structures

struct	agt_instance_ctrl_t
struct	agt_extended_cfg_t

Enumerations

enum	agt_clock_t
enum	agt_measure_t
enum	agt_agtio_filter_t
enum	agt_enable_pin_t
enum	agt_trigger_edge_t
enum	agt_output_pin_t
enum	agt_pin_cfg_t

Data Structure Documentation

◆ agt_instance_ctrl_t

struct agt_instance_ctrl_t

Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called.

◆ agt_extended_cfg_t

struct agt_extended_cfg_t

Optional AGT extension data structure.

Data Fields		
agt_clock_t	count_source	AGT channel clock source. Valid values are: AGT_CLOCK_PCLKB, AGT_CLOCK_LOCO, AGT_CLOCK_FSUB.
union agt_extended_cfg_t	__unnamed__	
agt_pin_cfg_t	agto: 3	Configure AGTO pin. *Note* *AGTIO polarity is opposite AGTO*
agt_measure_t	measurement_mode	Measurement mode.
agt_agtio_filter_t	agtio_filter	Input filter for AGTIO.
agt_enable_pin_t	enable_pin	Enable pin (event counting only)
agt_trigger_edge_t	trigger_edge	Trigger edge to start pulse period measurement or count external event.

Enumeration Type Documentation

◆ agt_clock_t

enum agt_clock_t

Count source

Enumerator	
AGT_CLOCK_PCLKB	PCLKB count source, division by 1, 2, or 8 allowed.
AGT_CLOCK_LOCO	LOCO count source, division by 1, 2, 4, 8, 16, 32, 64, or 128 allowed.
AGT_CLOCK_AGT_UNDERFLOW	Underflow event signal from next lowest AGT channel, division must be 1.
AGT_CLOCK_SUBCLOCK	Subclock count source, division by 1, 2, 4, 8, 16, 32, 64, or 128 allowed.
AGT_CLOCK_P402	Counts events on P402, events are counted in deep software standby mode.
AGT_CLOCK_P403	Counts events on P403, events are counted in deep software standby mode.
AGT_CLOCK_AGTIO	Counts events on AGTIOn, events are not counted in software standby modes.

◆ agt_measure_t

enum agt_measure_t

Enable pin for event counting mode.

Enumerator	
AGT_MEASURE_DISABLED	AGT used as a counter.
AGT_MEASURE_PULSE_WIDTH_LOW_LEVEL	AGT used to measure low level pulse width.
AGT_MEASURE_PULSE_WIDTH_HIGH_LEVEL	AGT used to measure high level pulse width.
AGT_MEASURE_PULSE_PERIOD	AGT used to measure pulse period.

◆ agt_agtio_filter_t

enum agt_agtio_filter_t

Input filter, applies AGTIO in pulse period measurement, pulse width measurement, or event counter mode. The filter requires the signal to be at the same level for 3 successive reads at the specified filter frequency.

Enumerator	
AGT_AGTIO_FILTER_NONE	No filter.
AGT_AGTIO_FILTER_PCLKB	Filter at PCLKB.
AGT_AGTIO_FILTER_PCLKB_DIV_8	Filter at PCLKB / 8.
AGT_AGTIO_FILTER_PCLKB_DIV_32	Filter at PCLKB / 32.

◆ agt_enable_pin_t

enum agt_enable_pin_t

Enable pin for event counting mode.

Enumerator	
AGT_ENABLE_PIN_NOT_USED	AGTEE/AGTWEE is not used.
AGT_ENABLE_PIN_ACTIVE_LOW	Events are only counted when AGTEE/AGTWEE is low.
AGT_ENABLE_PIN_ACTIVE_HIGH	Events are only counted when AGTEE/AGTWEE is high.

◆ agt_trigger_edge_t

enum agt_trigger_edge_t	
Trigger edge for pulse period measurement mode and event counting mode.	
Enumerator	
AGT_TRIGGER_EDGE_RISING	Measurement starts or events are counted on rising edge.
AGT_TRIGGER_EDGE_FALLING	Measurement starts or events are counted on falling edge.
AGT_TRIGGER_EDGE_BOTH	Events are counted on both edges (n/a for pulse period mode)

◆ agt_output_pin_t

enum agt_output_pin_t	
Output pins, used to select which duty cycle to update in R_AGT_DutyCycleSet().	
Enumerator	
AGT_OUTPUT_PIN_AGTOA	AGTOA.
AGT_OUTPUT_PIN_AGTOB	AGTOB.

◆ agt_pin_cfg_t

enum agt_pin_cfg_t	
Level of AGT pin	
Enumerator	
AGT_PIN_CFG_DISABLED	Not used as output pin.
AGT_PIN_CFG_START_LEVEL_LOW	Pin level low.
AGT_PIN_CFG_START_LEVEL_HIGH	Pin level high.

Function Documentation

◆ R_AGT_Close()

fsp_err_t R_AGT_Close (timer_ctrl_t *const *p_ctrl*)

Stops counter, disables interrupts, disables output pins, and clears internal driver data. Implements timer_api_t::close.

Return values

FSP_SUCCESS	Timer closed.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_AGT_PeriodSet()

fsp_err_t R_AGT_PeriodSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts*)

Updates period. The new period is updated immediately and the counter is reset to the maximum value. Implements timer_api_t::periodSet.

Warning

If periodic output is used, the duty cycle buffer registers are updated after the period buffer register. If this function is called while the timer is running and an AGT underflow occurs during processing, the duty cycle will not be the desired 50% duty cycle until the counter underflow after processing completes.
Stop the timer before calling this function if one-shot output is used.

Example:

```
 /* Get the source clock frequency (in Hz). There are several ways to do this in FSP:
  * - If LOCO or subclock is chosen in agt_extended_cfg_t::clock_source
  * - The source clock frequency is BSP_LOCO_HZ >> timer_cfg_t::source_div
  * - If PCLKB is chosen in agt_extended_cfg_t::clock_source and the PCLKB frequency
has not changed since reset,
  * - The source clock frequency is BSP_STARTUP_PCLKB_HZ >> timer_cfg_t::source_div
  * - Use the R_AGT_InfoGet function (it accounts for the clock source and divider).
  * - Calculate the current PCLKB frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) and right shift
  * by timer_cfg_t::source_div.
  *
  * This example uses the last option (R_FSP_SystemClockHzGet).
  */
    uint32_t timer_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKB) >>
```

```
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkb_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) timer_freq_hz * AGT_EXAMPLE_DESIRED_PERIOD_MSEC) /
AGT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. This will return an error if parameter checking is
enabled and the calculated
  * period is larger than UINT16_MAX. */
    err = R_AGT_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Period value updated.
FSP_ERR_ASSERTION	A required pointer was NULL, or the period was not in the valid range of 1 to 0xFFFF.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_AGT_DutyCycleSet()

fsp_err_t R_AGT_DutyCycleSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin*)

Updates duty cycle. If the timer is counting, the new duty cycle is reflected after the next counter underflow. Implements timer_api_t::dutyCycleSet.

Example:

```
/* Get the current period setting. */
timer_info_t info;
   (void) R_AGT_InfoGet(&g_timer0_ctrl, &info);
   uint32_t current_period_counts = info.period_counts;
/* Calculate the desired duty cycle based on the current period. */
   uint32_t duty_cycle_counts = (current_period_counts *
AGT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                                 AGT_EXAMPLE_MAX_PERCENT;
/* Set the calculated duty cycle. */
   err = R_AGT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, AGT_OUTPUT_PIN_AGTOA
);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Duty cycle updated.
FSP_ERR_ASSERTION	A required pointer was NULL, or the pin was not AGT_AGTO_AGTOA or AGT_AGTO_AGTOB.
FSP_ERR_INVALID_ARGUMENT	Duty cycle was not in the valid range of 0 to period (counts) - 1
FSP_ERR_NOT_OPEN	The instance control structure is not opened.
FSP_ERR_UNSUPPORTED	AGT_CFG_OUTPUT_SUPPORT_ENABLE is 0.

◆ R_AGT_Reset()

fsp_err_t R_AGT_Reset (timer_ctrl_t *const *p_ctrl*)

Resets the counter value to the period minus one. Implements timer_api_t::reset.

Return values

FSP_SUCCESS	Counter reset.
FSP_ERR_ASSERTION	p_ctrl is NULL
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_AGT_Start()

fsp_err_t R_AGT_Start (timer_ctrl_t *const *p_ctrl*)

Starts timer. Implements timer_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_AGT_Start(&g_timer0_ctrl);
```

Return values

FSP_SUCCESS	Timer started.
FSP_ERR_ASSERTION	p_ctrl is null.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_AGT_Enable()

fsp_err_t R_AGT_Enable (timer_ctrl_t *const *p_ctrl*)

Enables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::enable.

Example:

```
/* Enable captures. Captured values arrive in the interrupt. */
   (void) R_AGT_Enable(&g_timer0_ctrl);
```

Return values

FSP_SUCCESS	External events successfully enabled.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_AGT_Disable()

fsp_err_t R_AGT_Disable (timer_ctrl_t *const *p_ctrl*)

Disables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::disable.

Example:

```
/* (Optional) Disable captures. */
   (void) R_AGT_Disable(&g_timer0_ctrl);
```

Return values

FSP_SUCCESS	External events successfully disabled.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_AGT_InfoGet()

fsp_err_t R_AGT_InfoGet (timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info*)

Gets timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Example:

```
/* (Optional) Get the current period if not known. */
timer_info_t info;
      (void) R_AGT_InfoGet(&g_timer0_ctrl, &info);
      uint32_t period = info.period_counts;
```

Return values

FSP_SUCCESS	Period, count direction, and frequency stored in p_info.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_AGT_StatusGet()

fsp_err_t R_AGT_StatusGet (timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status*)

Retrieves the current state and counter value stores them in p_status. Implements timer_api_t::statusGet.

Example:

```
/* Read the current counter value. Counter value is in status.counter. */
timer_status_t status;
   (void) R_AGT_StatusGet(&g_timer0_ctrl, &status);
```

Return values

FSP_SUCCESS	Current status and counter value provided in p_status.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_AGT_Stop()

fsp_err_t R_AGT_Stop (timer_ctrl_t *const *p_ctrl*)

Stops the timer. Implements timer_api_t::stop.

Example:

```
/* (Optional) Stop the timer. */
   (void) R_AGT_Stop(&g_timer0_ctrl);
```

Return values

FSP_SUCCESS	Timer stopped.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance control structure is not opened.

◆ R_AGT_Open()

fsp_err_t R_AGT_Open (timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg*)

Initializes the AGT module instance. Implements timer_api_t::open.

The AGT hardware does not support one-shot functionality natively. The one-shot feature is therefore implemented in the AGT HAL layer. For a timer configured as a one-shot timer, the timer is stopped upon the first timer expiration.

The AGT implementation of the general timer can accept an optional agt_extended_cfg_t extension parameter. For AGT, the extension specifies the clock to be used as timer source and the output pin configurations. If the extension parameter is not specified (NULL), the default clock PCLKB is used and the output pins are disabled.

Example:

```
/* Initializes the module. */
    err = R_AGT_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful and timer has started.
FSP_ERR_ASSERTION	A required input pointer is NULL or the period is not in the valid range of 1 to 0xFFFF.
FSP_ERR_ALREADY_OPEN	R_AGT_Open has already been called for this p_ctrl.
FSP_ERR_IRQ_BSP_DISABLED	A required interrupt has not been enabled in the vector table.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel number is not available on AGT.

◆ R_AGT_CallbackSet()

fsp_err_t R_AGT_CallbackSet (timer_ctrl_t *const *p_api_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void const *const *p_context*, timer_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements timer_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.6 Bluetooth Low Energy Library of Balance Configuration (r_ble)

Modules

Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface.

Overview

The bluetooth low energy library of balance configuration (r_ble) provides an API to control the Radio peripheral. This module is configured via the QE for BLE. QE for BLE provides standard services defined by standardization organization and custom services defined by user. Bluetooth LE Profile API Document User's Manual describes the APIs for standard services.

Features

- Common
  - Open/Close the BLE protocol stack.
  - Execute the BLE job.
  - Add an event in the BLE protocol stack internal queue.
- GAP
  - Initialization of the Host stack.
  - Start/Stop Advertising.
  - Start/Stop Scan.
  - Connect/Disconnect a link.
  - Initiate/Respond a pairing request.
- GATT Common
  - Get MTU size.

- GATT Server
  - Initialization of GATT Server.
  - Notification/Indication.
- GATT Client
  - Discovery services, characteristics.
  - Read/Write characteristic.
- Vendor Specific
  - DTM.
  - Set/Get transmit power.
  - Set/Get BD_ADDR.

Supported functions

The supported functions are listed in the table below. Choose the configuration that best suits the functions that target system requires.

BLE library feature	Extended	Balance	Compact
Common API	Supported	**Supported**	Supported
GAP API	Supported	***1 Limited**	*1 *2 Limited
GATT Common API	Supported	**Supported**	Supported
GATT Client API	Supported	**Supported**	Supported
GATT Client API	Supported	**Supported**	Supported
L2CAP API	Supported	**Not Supported**	Not supported
Vendor Specific API	Supported	**Supported**	Supported

Note

1. *This configuration dose not support LE Advertising Extensions functionality APIs.*
2. *This configuration dose not support Central and Observer functionality APIs.*

Target Devices

The Renesas Bluetooth Low Energy Library supports the following devices.

- RA4W1

Configuration

Clock Configuration

Note

System clock (ICLK): 8 MHz or more
Peripheral module clock A (PCLKA): 8MHz or more
The BLE Protocol Stack is optimized for ICLK and PCLKA frequencies of 32 MHz.
It is recommended that the clock be set so that the ICLK and PCLKA frequencies are 32MHz in order to get the best performance from the BLE.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Figure shows the software structure of the BLE FSP module.

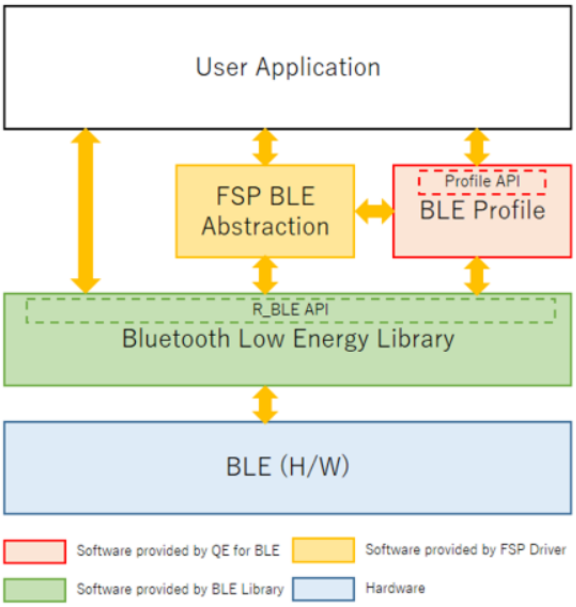


Figure 145: BLE software structure

The BLE FSP module consists of the BLE library.
The BLE Application uses the BLE functions via the R_BLE API provided by the BLE Library.
The QE for BLE generates the source codes (BLE base skeleton program) as a base for the BLE Application and the BLE Profile codes including the Profile API.

Limitations

Developers should be aware of the following limitations when using the ble:

- This configuration not supports LE Advertising Extensions functionality APIs. If those functionality needs your system, select Bluetooth Low Energy Library of Extended Configuration (r_ble) configration.
- This configuration dose not supports L2CAP functionality APIs. If those functionality needs your system, select Bluetooth Low Energy Library of Extended Configuration (r_ble) configration.

4.2.7 Bluetooth Low Energy Library of Compact Configuration (r_ble)

Modules

Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface.

Overview

The bluetooth low energy library of compact configuration (r_ble) provides an API to control the Radio peripheral. This module is configured via the QE for BLE. QE for BLE provides standard services defined by standardization organization and custom services defined by user. Bluetooth LE Profile API Document User's Manual describes the APIs for standard services.

Features

- Common
  - Open/Close the BLE protocol stack.
  - Execute the BLE job.
  - Add an event in the BLE protocol stack internal queue.
- GAP
  - Initialization of the Host stack.
  - Start/Stop Advertising.
  - Initiate/Respond a pairing request.
- GATT Common
  - Get MTU size.
- GATT Server
  - Initialization of GATT Server.
  - Notification/Indication.
- GATT Client
  - Discovery services, characteristics.
  - Read/Write characteristic.
- Vendor Specific
  - DTM.
  - Set/Get transmit power.
  - Set/Get BD_ADDR.

Supported functions

The supported functions are listed in the table below. Choose the configuration that best suits the functions that target system requires.

BLE library feature	Extended	Balance	Compact
Common API	Supported	Supported	**Supported**
GATT Common API	Supported	*1 Limited	***1 *2 Limited**
GATT Common API	Supported	Supported	**Supported**
GATT Server API	Supported	Supported	**Supported**
GATT Client API	Supported	Supported	**Supported**
L2CAP API	Supported	Not supported	**Not supported**
Vendor Specific API	Supported	Supported	**Supported**

Note

1. *This configuration dose not support LE Advertising Extensions functionality APIs.*
2. *This configuration dose not support Central and Observer functionality APIs.*

Target Devices

The Renesas Bluetooth Low Energy Library supports the following devices.

- RA4W1

Configuration

Clock Configuration

Note

System clock (ICLK): 8 MHz or more
Peripheral module clock A (PCLKA): 8MHz or more
The BLE Protocol Stack is optimized for ICLK and PCLKA frequencies of 32 MHz.
It is recommended that the clock be set so that the ICLK and PCLKA frequencies are 32MHz in order to get the best performance from the BLE.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Figure shows the software structure of the BLE FSP module.

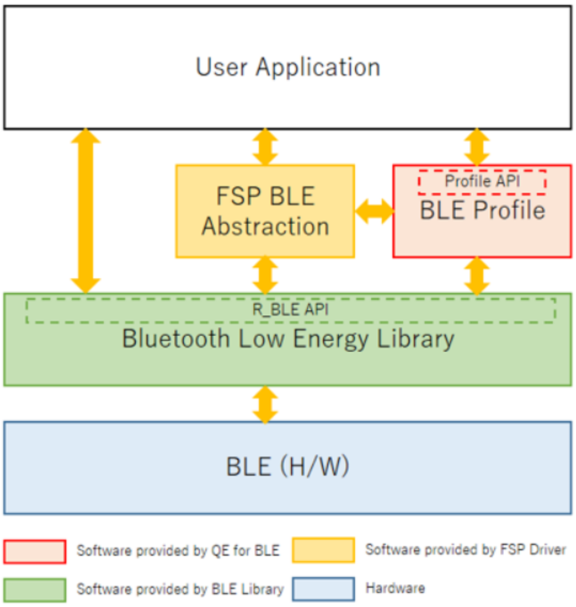


Figure 146: BLE software structure

The BLE FSP module consists of the BLE library.
The BLE Application uses the BLE functions via the R_BLE API provided by the BLE Library.
The QE for BLE generates the source codes (BLE base skeleton program) as a base for the BLE Application and the BLE Profile codes including the Profile API.

Limitations

Developers should be aware of the following limitations when using the ble:

- This configuration dose not supports LE Advertising Extensions functionality APIs. If those functionality needs your system, select Bluetooth Low Energy Library of Extended Configuration (r_ble) configration.
- This configuration dose not supports Central and Observer functionality APIs. If those functionality needs your system, select Bluetooth Low Energy Library of Extended Configuration (r_ble) or Bluetooth Low Energy Library of Balance Configuration (r_ble) configration.
- This configuration dose not supports L2CAP functionality APIs. If those functionality needs your system, select Bluetooth Low Energy Library of Extended Configuration (r_ble) configration.

4.2.8 Bluetooth Low Energy Library of Extended Configuration (r_ble)

Modules

Driver for the Radio peripheral on RA MCUs. This module implements the BLE Interface.

Overview

The bluetooth low energy library of extended configuration (r_ble) provides an API to control the Radio peripheral. This module is configured via the QE for BLE. QE for BLE provides standard services defined by standardization organization and custom services defined by user. Bluetooth LE Profile API Document User's Manual describes the APIs for standard services.

Features

- Common
  - Open/Close the BLE protocol stack.
  - Execute the BLE job.
  - Add an event in the BLE protocol stack internal queue.
- GAP
  - Initialization of the Host stack.
  - Start/Stop Advertising (Support LE Advertising Extensions).
  - Start/Stop Scan.
  - Connect/Disconnect a link.
  - Initiate/Respond a pairing request.
- GATT Common
  - Get MTU size.
- GATT Server
  - Initialization of GATT Server.
  - Notification/Indication.
- GATT Client
  - Discovery services, characteristics.
  - Read/Write characteristic.
- L2CAP
  - Credit-based flow control transaction.
- Vendor Specific
  - DTM.
  - Set/Get transmit power.
  - Set/Get BD_ADDR.

Supported functions

The supported functions are listed in the table below. Choose the configuration that best suits the functions that target system requires.

BLE library feature	Extended	Balance	Compact
Common API	**Supported**	Supported	Supported
GAP API	**Supported**	*1 Limited	*1 *2 Limited
GATT Common API	**Supported**	Supported	Supported
GATT Server API	**Supported**	Supported	Supported
GATT Client API	**Supported**	Supported	Supported
L2CAP API	**Supported**	Not supported	Not supported
Vendor Specific API	**Supported**	Supported	Supported

Note

1. This configuration dose not support LE Advertising Extensions functionality APIs.
2. This configuration dose not support Central and Observer functionality APIs.

Target Devices

The Renesas Bluetooth Low Energy Library supports the following devices.

- RA4W1

Configuration

Clock Configuration

Note

System clock (ICLK): 8 MHz or more
Peripheral module clock A (PCLKA): 8MHz or more
The BLE Protocol Stack is optimized for ICLK and PCLKA frequencies of 32 MHz.
It is recommended that the clock be set so that the ICLK and PCLKA frequencies are 32MHz in order to get the best performance from the BLE.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Figure shows the software structure of the BLE FSP module.

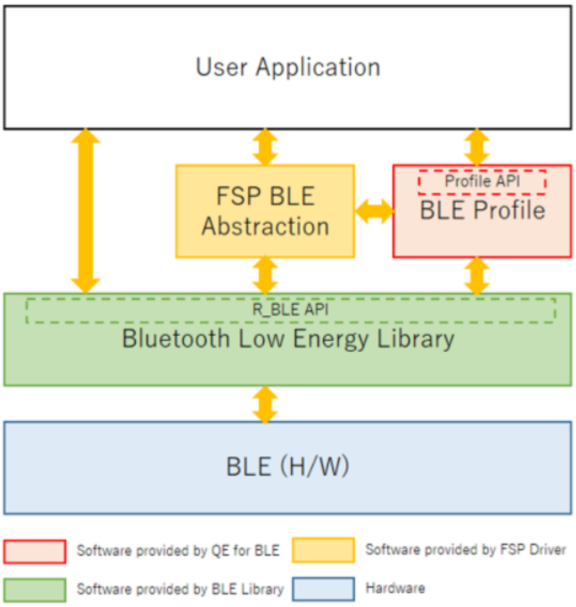


Figure 147: BLE software structure

The BLE FSP module consists of the BLE library.
The BLE Application uses the BLE functions via the R_BLE API provided by the BLE Library.
The QE for BLE generates the source codes (BLE base skeleton program) as a base for the BLE Application and the BLE Profile codes including the Profile API.

Limitations

Developers should be aware of the following limitations when using the ble:

4.2.9 Clock Frequency Accuracy Measurement Circuit (r_cac)

Modules

Functions

fsp_err_t	R_CAC_Open (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)
fsp_err_t	R_CAC_StartMeasurement (cac_ctrl_t *const p_ctrl)
fsp_err_t	R_CAC_StopMeasurement (cac_ctrl_t *const p_ctrl)
fsp_err_t	R_CAC_Read (cac_ctrl_t *const p_ctrl, uint16_t *const p_counter)
fsp_err_t	R_CAC_Close (cac_ctrl_t *const p_ctrl)
fsp_err_t	R_CAC_CallbackSet (cac_ctrl_t *const p_ctrl, void(*p_callback)(cac_callback_args_t *), void const *const p_context, cac_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the CAC peripheral on RA MCUs. This module implements the CAC Interface.

Overview

The interface for the clock frequency accuracy measurement circuit (CAC) peripheral is used to check a system clock frequency with a reference clock signal by counting the number of measurement clock edges that occur between two edges of the reference clock.

Features

- Supports clock frequency-measurement and monitoring based on a reference signal input
- Reference can be either an externally supplied clock source or an internal clock source
- An interrupt request may optionally be generated by a completed measurement, a detected frequency error, or a counter overflow.
- A digital filter is available for an externally supplied reference clock, and dividers are available for both internally supplied measurement and reference clocks.
- Edge-detection options for the reference clock are configurable as rising, falling, or both.

Configuration

Build Time Configurations for r_cac

The following build time configurations are defined in fsp_cfg/r_cac_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Monitoring > Clock Accuracy Circuit (r_cac)

This module can be added to the Stacks tab via New Stack > Monitoring > Clock Accuracy Circuit (r_cac). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_cac0	Module name.
Reference clock divider	• 32 • 128 • 1024 • 8192	32	Reference clock divider.
Reference clock source	• Main Oscillator • Sub-clock • HOCO	Main Oscillator	Reference clock source.

	• MOCO • LOCO • PCLKB • IWDT • External		
Reference clock digital filter	• Disabled • Sampling clock =Measuring freq • Sampling clock =Measuring freq/4 • Sampling clock =Measuring freq/16	Disabled	Reference clock digital filter.
Reference clock edge detect	• Rising • Falling • Both	Rising	Reference clock edge detection.
Measurement clock divider	• 1 • 4 • 8 • 32	1	Measurement clock divider.
Measurement clock source	• Main Oscillator • Sub-clock • HOCO • MOCO • LOCO • PCLKB • IWDT	HOCO	Measurement clock source.
Upper Limit Threshold	Value must be a non-negative integer, between 0 to 65535	0	Top end of allowable range for measurement completion.
Lower Limit Threshold	Value must be a non-negative integer, between 0 to 65535	0	Bottom end of allowable range for measurement completion.
Frequency Error Interrupt Priority	MCU Specific Options		CAC frequency error interrupt priority.
Measurement End Interrupt Priority	MCU Specific Options		CAC measurement end interrupt priority.
Overflow Interrupt Priority	MCU Specific Options		CAC overflow interrupt priority.
Callback	Name must be a valid C symbol	NULL	Function name for callback

Clock Configuration

The CAC measurement clock source can be configured as the following:

1. MAIN_OSC
2. SUBCLOCK
3. HOCO
4. MOCO
5. LOCO
6. PCLKB
7. IWDT

The CAC reference clock source can be configured as the following:

1. MAIN_OSC
2. SUBCLOCK
3. HOCO
4. MOCO
5. LOCO
6. PCLKB
7. IWDT
8. External Clock Source (CACREF)

Pin Configuration

The CACREF pin can be configured to provide the reference clock for CAC measurements.

Usage Notes

Measurement Accuracy

The clock measurement result may be off by up to one pulse depending on the phase difference between the edge detection circuit, digital filter, and CACREF pin signal, if applicable.

Frequency Error Interrupt

The frequency error interrupt is only triggered at the end of a CAC measurement. This means that there will be a measurement complete interrupt in addition to the frequency error interrupt.

Examples

Basic Example

This is a basic example of minimal use of the CAC in an application.

```
volatile uint32_t g_callback_complete;
void cac_basic_example ()
{
    g_callback_complete = 0;
 fsp_err_t err = R_CAC_Open(&g_cac_ctrl, &g_cac_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    (void) R_CAC_StartMeasurement(&g_cac_ctrl);
```

```
 /* Wait for measurement to complete. */
 while (0 == g_callback_complete)
    {
    }
    uint16_t value;
 /* Read the CAC measurement. */
    (void) R_CAC_Read(&g_cac_ctrl, &value);
}
/* Called when measurement is completed. */
static void r_cac_callback (cac_callback_args_t * p_args)
{
 if (CAC_EVENT_MEASUREMENT_COMPLETE == p_args->event)
    {
       g_callback_complete = 1U;
    }
}
```

Data Structures

struct	cac_instance_ctrl_t

Data Structure Documentation

◆ cac_instance_ctrl_t

struct cac_instance_ctrl_t
CAC instance control block. DO NOT INITIALIZE.

Function Documentation

◆ R_CAC_Open()

fsp_err_t R_CAC_Open (cac_ctrl_t *const *p_ctrl*, cac_cfg_t const *const *p_cfg*)

The Open function configures the CAC based on the provided user configuration settings.

Return values

FSP_SUCCESS	CAC is available and available for measurement(s).
FSP_ERR_ASSERTION	An argument is invalid.
FSP_ERR_ALREADY_OPEN	The CAC has already been opened.

Note

There is only a single CAC peripheral.

◆ R_CAC_StartMeasurement()

fsp_err_t R_CAC_StartMeasurement (cac_ctrl_t *const *p_ctrl*)

Start the CAC measurement process.

Return values

FSP_SUCCESS	CAC measurement started.
FSP_ERR_ASSERTION	NULL provided for p_instance_ctrl or p_cfg.
FSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.

◆ R_CAC_StopMeasurement()

fsp_err_t R_CAC_StopMeasurement (cac_ctrl_t *const *p_ctrl*)

Stop the CAC measurement process.

Return values

FSP_SUCCESS	CAC measuring has been stopped.
FSP_ERR_ASSERTION	NULL provided for p_instance_ctrl or p_cfg.
FSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.

◆ R_CAC_Read()

fsp_err_t R_CAC_Read (cac_ctrl_t *const *p_ctrl*, uint16_t *const *p_counter*)

Read and return the CAC status and counter registers.

Return values

FSP_SUCCESS	CAC read successful.
FSP_ERR_ASSERTION	An argument is NULL.
FSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.

◆ R_CAC_Close()

fsp_err_t R_CAC_Close (cac_ctrl_t *const *p_ctrl*)

Release any resources that were allocated by the Open() or any subsequent CAC operations.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	NULL provided for p_instance_ctrl or p_cfg.
FSP_ERR_NOT_OPEN	R_CAC_Open() has not been successfully called.

◆ R_CAC_CallbackSet()

fsp_err_t R_CAC_CallbackSet (cac_ctrl_t *const *p_ctrl*, void(*)(cac_callback_args_t *) *p_callback*, void const *const *p_context*, cac_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements cac_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.10 Controller Area Network (r_can)

Modules

Functions

fsp_err_t	R_CAN_Open (can_ctrl_t *const p_api_ctrl, can_cfg_t const *const p_cfg)
fsp_err_t	R_CAN_Close (can_ctrl_t *const p_api_ctrl)
fsp_err_t	R_CAN_Write (can_ctrl_t *const p_api_ctrl, uint32_t const mailbox, can_frame_t *const p_frame)
fsp_err_t	R_CAN_Read (can_ctrl_t *const p_api_ctrl, uint32_t mailbox, can_frame_t *const p_frame)
fsp_err_t	R_CAN_ModeTransition (can_ctrl_t *const p_api_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode)
fsp_err_t	R_CAN_InfoGet (can_ctrl_t *const p_api_ctrl, can_info_t *const p_info)
fsp_err_t	R_CAN_CallbackSet (can_ctrl_t *const p_api_ctrl, void(*p_callback)(can_callback_args_t *), void const *const p_context, can_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the CAN peripheral on RA MCUs. This module implements the CAN Interface.

Overview

The Controller Area network (CAN) HAL module provides a high-level API for CAN applications and supports the CAN peripherals available on RA microcontroller hardware. A user-callback function must be defined that the driver will invoke when transmit, receive or error interrupts are received. The callback is passed a parameter which indicates the channel, mailbox and event as well as the received data (if available).

Features

- Supports both standard (11-bit) and extended (29-bit) messaging formats
- Supports speeds upto 1 Mbps
- Support for bit timing configuration as defined in the CAN specification
- Supports up to 32 transmit or receive mailboxes with standard or extended ID frames
- Optional support for a 4-stage transmit and receive FIFO
- Receive mailboxes can be configured to capture either data or remote CAN Frames
- Receive mailboxes can be configured to receive a range of IDs using mailbox masks
- Mailboxes can be configured with Overwrite or Overrun mode
- Supports a user-callback function when transmit, receive, or error interrupts are received

Configuration

Build Time Configurations for r_can

The following build time configurations are defined in fsp_cfg/r_can_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
FIFO Support	• Disabled • Enabled	Disabled	When FIFOs are enabled, a transmit FIFO replaces mailboxes 24-27 and a receive FIFO replaces mailboxes 28-31.

Configurations for Connectivity > CAN (r_can)

This module can be added to the Stacks tab via New Stack > Connectivity > CAN (r_can). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_can0	Module name.
General > Channel	Channel should be 0 or 1	0	Specify the CAN channel to use.
General > Clock Source	MCU Specific Options		Select the CAN clock source.
General > Overwrite/Overrrun Mode	• Overwrite Mode • Overrrun Mode	Overwrite Mode	Select whether receive mailbox will be overwritten or overrun if data is not read in time.
General > Global ID Mode	• Standard ID Mode • Extended ID Mode • Mixed ID Mode	Standard ID Mode	Select whether the driver will use CAN Standard IDs, Extended IDs or a mix of both.
General > Number of Mailboxes	• 4 Mailboxes • 8 Mailboxes • 16 Mailboxes • 24 Mailboxes • 32 Mailboxes	32 Mailboxes	Select 4, 8, 16, 24 or 32 mailboxes. In FIFO mailbox mode up to 24 mailboxes are available.
Baud Rate Settings > Auto-generated Settings > Sample-	Must be a valid integer between 0 and 100. Ignore when Override	75	Sample-Point = (TSEG1 + 1) / (TSEG1 + TSEG2 + 1).

Point (%)	Baud Settings is Enabled.		
Baud Rate Settings > Auto-generated Settings > CAN Baud Rate (Hz)	Must be a valid integer configurable upto maximum 1MHz. Ignore when Override Baud Settings is Enabled.	500000	Specify baud rate in Hz. If the requested baud rate cannot be achieved, the settings with the largest possible baud rate that is less than or equal to the requested baud rate is used. If multiple combinations would result in the best baud rate, the combination with the least absolute error for the ratio is chosen. The theoretical calculated baud rate and ratio are printed in a comment in the generated can_bit_timing_cfg_t structure.
Baud Rate Settings > Override Auto-generated Settings > Override Baud Settings	• Enabled • Disabled	Disabled	Override calculated baudrate parameters and instead use the ones specified below. This option ignores the parameters specified under Sample-Point (%) and CAN Baud Rate (Hz)
Baud Rate Settings > Override Auto-generated Settings > Baud Rate Prescaler	Value must be a non-negative integer between 1 and 1024.	1	Specify division value of baud rate prescaler (baud rate prescalar + 1).
Baud Rate Settings > Override Auto-generated Settings > Time Segment 1	Refer to the RA Configuration tool for available options.	4 Time Quanta	Select the time segment 1 value. (4-16). Check module usage notes for how to calculate this value.
Baud Rate Settings > Override Auto-generated Settings > Time Segment 2	• 2 Time Quanta • 3 Time Quanta • 4 Time Quanta • 5 Time Quanta • 6 Time Quanta • 7 Time Quanta • 8 Time Quanta	2 Time Quanta	Select the time segment 2 value (2-8). Check module usage notes for how to calculate this value.
Baud Rate Settings >	• 1 Time Quanta	1 Time Quanta	Select the

Override Auto-generated Settings > Synchronization Jump Width	• 2 Time Quanta • 3 Time Quanta • 4 Time Quanta		Synchronization Jump Width value (1-4). Check module usage notes for how to calculate this value.
Interrupts > Callback	Name must be a valid C symbol	can_callback	A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time any interrupt occurs.
Interrupts > Interrupt Priority Level	MCU Specific Options		Transmit/Receive/Error interrupt priority.
Interrupts > Transmit FIFO Interrupt Mode	• Every Message • Empty	Every Message	Select whether the receive FIFO should throw an interrupt on every received message or when it becomes empty.
Input > Receive FIFO > Receive ID 1 > ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0	Select the first ID for the receive FIFO, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs.
Input > Receive FIFO > Receive ID 1 > ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the receive FIFO is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Receive FIFO > Receive ID 1 > Frame Type	• Data Mailbox • Remote Mailbox	Remote Mailbox	Select whether the receive FIFO is used to capture data frames or remote frames.
Input > Receive FIFO > Receive ID 1 > Mask	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0x1FFFFFFF	Select the Mask for the receive FIFO. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value.
Input > Receive FIFO > Receive ID 2 > ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0	Select the second ID for the receive FIFO, between 0 and 0x7ff when using standard IDs, between 0 and

			0x1FFFFFFF when using extended IDs.
Input > Receive FIFO > Receive ID 2 > ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the receive FIFO is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Receive FIFO > Receive ID 2 > Frame Type	• Data Mailbox • Remote Mailbox	Remote Mailbox	Select whether the receive FIFO is used to capture data frames or remote frames.
Input > Receive FIFO > Receive ID 2 > Mask	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0x1FFFFFFF	Select the Mask for the receive FIFO. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value.
Input > Mailbox 0-3 Group > Mailbox ID > Mailbox 0 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0	Select the receive ID for mailbox 0, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 0-3 Group > Mailbox ID > Mailbox 1 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	1	Select the receive ID for mailbox 1, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 0-3 Group > Mailbox ID > Mailbox 2 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	2	Select the receive ID for mailbox 2, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 0-3 Group > Mailbox ID > Mailbox 3 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	3	Select the receive ID for mailbox 3, between 0 and 0x7ff when using standard IDs, between

			0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 0-3 Group > Mailbox ID Mode > Mailbox 0 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 0-3 Group > Mailbox ID Mode > Mailbox 1 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 0-3 Group > Mailbox ID Mode > Mailbox 2 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 0-3 Group > Mailbox ID Mode > Mailbox 3 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 0-3 Group > Mailbox Type > Mailbox 0 Type	• Receive Mailbox • Transmit Mailbox	Transmit Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 0-3 Group > Mailbox Type > Mailbox 1 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 0-3 Group > Mailbox Type > Mailbox 2 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 0-3	• Receive	Receive Mailbox	Select whether the

Group > Mailbox Type > Mailbox 3 Type	Mailbox • Transmit Mailbox		mailbox is used for receive or transmit.
Input > Mailbox 0-3 Group > Mailbox Frame Type > Mailbox 0 Frame Type	• Data Mailbox • Remote Mailbox	Remote Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 0-3 Group > Mailbox Frame Type > Mailbox 1 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 0-3 Group > Mailbox Frame Type > Mailbox 2 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 0-3 Group > Mailbox Frame Type > Mailbox 3 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 0-3 Group > Mailbox 0-3 Group Mask	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0x1FFFFFFF	Select the Mask for mailboxes 0-3. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value.
Input > Mailbox 4-7 Group > Mailbox ID > Mailbox 4 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	4	Select the receive ID for mailbox 4, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 4-7 Group > Mailbox ID > Mailbox 5 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	5	Select the receive ID for mailbox 5, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.

Input > Mailbox 4-7 Group > Mailbox ID > Mailbox 6 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	6	Select the receive ID for mailbox 6, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 4-7 Group > Mailbox ID > Mailbox 7 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	7	Select the receive ID for mailbox 7, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 4-7 Group > Mailbox ID Mode > Mailbox 4 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 4-7 Group > Mailbox ID Mode > Mailbox 5 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 4-7 Group > Mailbox ID Mode > Mailbox 6 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 4-7 Group > Mailbox ID Mode > Mailbox 7 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 4-7	• Receive	Receive Mailbox	Select whether the

Group > Mailbox Type > Mailbox 4 Type	Mailbox • Transmit Mailbox		mailbox is used for receive or transmit.
Input > Mailbox 4-7 Group > Mailbox Type > Mailbox 5 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 4-7 Group > Mailbox Type > Mailbox 6 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 4-7 Group > Mailbox Type > Mailbox 7 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 4-7 Group > Mailbox Frame Type > Mailbox 4 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 4-7 Group > Mailbox Frame Type > Mailbox 5 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 4-7 Group > Mailbox Frame Type > Mailbox 6 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 4-7 Group > Mailbox Frame Type > Mailbox 7 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 4-7 Group > Mailbox 4-7 Group Mask	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0x1FFFFFFF	>Select the Mask for mailboxes 4-7. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value.
Input > Mailbox 8-11 Group > Mailbox ID > Mailbox 8 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	8	Select the receive ID for mailbox 8, between 0 and 0x7ff when using standard IDs, between

			0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 8-11 Group > Mailbox ID > Mailbox 9 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	9	Select the receive ID for mailbox 9, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 8-11 Group > Mailbox ID > Mailbox 10 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	10	Select the receive ID for mailbox 10, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 8-11 Group > Mailbox ID > Mailbox 11 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	11	Select the receive ID for mailbox 11, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 8-11 Group > Mailbox ID Mode > Mailbox 8 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 8-11 Group > Mailbox ID Mode > Mailbox 9 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 8-11	• Standard ID	Standard ID	Select whether the

Group > Mailbox ID Mode > Mailbox 10 ID Mode	• Extended ID		mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 8-11 Group > Mailbox ID Mode > Mailbox 11 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 8-11 Group > Mailbox Type > Mailbox 8 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 8-11 Group > Mailbox Type > Mailbox 9 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 8-11 Group > Mailbox Type > Mailbox 10 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 8-11 Group > Mailbox Type > Mailbox 11 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 8-11 Group > Mailbox Frame Type > Mailbox 8 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 8-11 Group > Mailbox Frame Type > Mailbox 9 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 8-11 Group > Mailbox Frame Type > Mailbox 10 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).

Input > Mailbox 8-11 Group > Mailbox Frame Type > Mailbox 11 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 8-11 Group > Mailbox 8-11 Group Mask	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0x1FFFFFFF	Select the Mask for mailboxes 8-11. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value.
Input > Mailbox 12-15 Group > Mailbox ID > Mailbox 12 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	12	Select the receive ID for mailbox 12, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 12-15 Group > Mailbox ID > Mailbox 13 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	13	Select the receive ID for mailbox 13, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 12-15 Group > Mailbox ID > Mailbox 14 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	14	Select the receive ID for mailbox 14, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 12-15 Group > Mailbox ID > Mailbox 15 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	15	Select the receive ID for mailbox 15, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.

Input > Mailbox 12-15 Group > Mailbox ID Mode > Mailbox 12 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 12-15 Group > Mailbox ID Mode > Mailbox 13 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 12-15 Group > Mailbox ID Mode > Mailbox 14 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 12-15 Group > Mailbox ID Mode > Mailbox 15 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 12-15 Group > Mailbox Type > Mailbox 12 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 12-15 Group > Mailbox Type > Mailbox 13 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 12-15 Group > Mailbox Type > Mailbox 14 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 12-15 Group > Mailbox Type > Mailbox 15 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 12-15 Group > Mailbox Frame	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to

Type > Mailbox 12 Frame Type			capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 12-15 Group > Mailbox Frame Type > Mailbox 13 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 12-15 Group > Mailbox Frame Type > Mailbox 14 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 12-15 Group > Mailbox Frame Type > Mailbox 15 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 12-15 Group > Mailbox 12-15 Group Mask	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0x1FFFFFFF	Select the Mask for mailboxes 12-15. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value.
Input > Mailbox 16-19 Group > Mailbox ID > Mailbox 16 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	16	Select the receive ID for mailbox 16, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 16-19 Group > Mailbox ID > Mailbox 17 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	17	Select the receive ID for mailbox 17, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 16-19 Group > Mailbox ID > Mailbox 18 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	18	Select the receive ID for mailbox 18, between 0 and 0x7ff

			when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 16-19 Group > Mailbox ID > Mailbox 19 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	19	Select the receive ID for mailbox 19, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 16-19 Group > Mailbox ID Mode > Mailbox 16 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 16-19 Group > Mailbox ID Mode > Mailbox 17 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 16-19 Group > Mailbox ID Mode > Mailbox 18 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 16-19 Group > Mailbox ID Mode > Mailbox 19 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 16-19 Group > Mailbox Type	• Receive Mailbox	Receive Mailbox	Select whether the mailbox is used for

> Mailbox 16 Type	• Transmit Mailbox		receive or transmit.
Input > Mailbox 16-19 Group > Mailbox Type > Mailbox 17 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 16-19 Group > Mailbox Type > Mailbox 18 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 16-19 Group > Mailbox Type > Mailbox 19 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 16-19 Group > Mailbox Frame Type > Mailbox 16 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 16-19 Group > Mailbox Frame Type > Mailbox 17 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 16-19 Group > Mailbox Frame Type > Mailbox 18 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 16-19 Group > Mailbox Frame Type > Mailbox 19 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 16-19 Group > Mailbox 16-19 Group Mask	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0x1FFFFFFF	Select the Mask for mailboxes 16-19. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value.
Input > Mailbox 20-23 Group > Mailbox ID > Mailbox 20 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	20	Select the receive ID for mailbox 20, between 0 and 0x7ff when using standard IDs, between 0 and

			0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 20-23 Group > Mailbox ID > Mailbox 21 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	21	Select the receive ID for mailbox 21, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 20-23 Group > Mailbox ID > Mailbox 22 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	22	Select the receive ID for mailbox 22, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 20-23 Group > Mailbox ID > Mailbox 23 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	23	Select the receive ID for mailbox 23, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 20-23 Group > Mailbox ID Mode > Mailbox 20 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 20-23 Group > Mailbox ID Mode > Mailbox 21 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.

Input > Mailbox 20-23 Group > Mailbox ID Mode > Mailbox 22 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 20-23 Group > Mailbox ID Mode > Mailbox 23 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 20-23 Group > Mailbox Type > Mailbox 20 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 20-23 Group > Mailbox Type > Mailbox 21 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 20-23 Group > Mailbox Type > Mailbox 22 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 20-23 Group > Mailbox Type > Mailbox 23 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 20-23 Group > Mailbox Frame Type > Mailbox 20 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 20-23 Group > Mailbox Frame Type > Mailbox 21 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 20-23 Group > Mailbox Frame Type > Mailbox 22 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).

Input > Mailbox 20-23 Group > Mailbox Frame Type > Mailbox 23 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 20-23 Group > Mailbox 20-23 Group Mask	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0x1FFFFFFF	Select the Mask for mailboxes 20-23. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value.
Input > Mailbox 24-27 Group > Mailbox ID > Mailbox 24 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	24	Select the receive ID for mailbox 24, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 24-27 Group > Mailbox ID > Mailbox 25 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	25	Select the receive ID for mailbox 25, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 24-27 Group > Mailbox ID > Mailbox 26 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	26	Select the receive ID for mailbox 26, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 24-27 Group > Mailbox ID > Mailbox 27 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	27	Select the receive ID for mailbox 27, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.

Input > Mailbox 24-27 Group > Mailbox ID Mode > Mailbox 24 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 24-27 Group > Mailbox ID Mode > Mailbox 25 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 24-27 Group > Mailbox ID Mode > Mailbox 26 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 24-27 Group > Mailbox ID Mode > Mailbox 27 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 24-27 Group > Mailbox Type > Mailbox 24 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 24-27 Group > Mailbox Type > Mailbox 25 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 24-27 Group > Mailbox Type > Mailbox 26 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 24-27 Group > Mailbox Type > Mailbox 27 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 24-27 Group > Mailbox Frame	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to

Type > Mailbox 24 Frame Type			capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 24-27 Group > Mailbox Frame Type > Mailbox 25 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 24-27 Group > Mailbox Frame Type > Mailbox 26 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 24-27 Group > Mailbox Frame Type > Mailbox 27 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 24-27 Group > Mailbox 24-27 Group Mask	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0x1FFFFFFF	Select the Mask for mailboxes 24-27. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value.
Input > Mailbox 28-31 Group > Mailbox ID > Mailbox 28 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	28	Select the receive ID for mailbox 28, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 28-31 Group > Mailbox ID > Mailbox 29 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	29	Select the receive ID for mailbox 29, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 28-31 Group > Mailbox ID > Mailbox 30 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	30	Select the receive ID for mailbox 30, between 0 and 0x7ff

			when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 28-31 Group > Mailbox ID > Mailbox 31 ID	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	31	Select the receive ID for mailbox 31, between 0 and 0x7ff when using standard IDs, between 0 and 0x1FFFFFFF when using extended IDs. Value is not used when the mailbox is set as transmit type.
Input > Mailbox 28-31 Group > Mailbox ID Mode > Mailbox 28 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 28-31 Group > Mailbox ID Mode > Mailbox 29 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 28-31 Group > Mailbox ID Mode > Mailbox 30 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 28-31 Group > Mailbox ID Mode > Mailbox 31 ID Mode	• Standard ID • Extended ID	Standard ID	Select whether the mailbox is used to receive Standard or Extended ID messages. This setting is only valid when Global ID Mode is set to 'Mixed ID Mode'.
Input > Mailbox 28-31 Group > Mailbox Type	• Receive Mailbox	Receive Mailbox	Select whether the mailbox is used for

> Mailbox 28 Type	• Transmit Mailbox		receive or transmit.
Input > Mailbox 28-31 Group > Mailbox Type > Mailbox 29 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 28-31 Group > Mailbox Type > Mailbox 30 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 28-31 Group > Mailbox Type > Mailbox 31 Type	• Receive Mailbox • Transmit Mailbox	Receive Mailbox	Select whether the mailbox is used for receive or transmit.
Input > Mailbox 28-31 Group > Mailbox Frame Type > Mailbox 28 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 28-31 Group > Mailbox Frame Type > Mailbox 29 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 28-31 Group > Mailbox Frame Type > Mailbox 30 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 28-31 Group > Mailbox Frame Type > Mailbox 31 Frame Type	• Data Mailbox • Remote Mailbox	Data Mailbox	Select whether the mailbox is used to capture data frames or remote frames (ignored for transmit mailboxes).
Input > Mailbox 28-31 Group > Mailbox 28-31 Group Mask	Value must be decimal or HEX integer of 0x1FFFFFFF or less.	0x1FFFFFFF	Select the Mask for mailboxes 28-31. In Mixed ID Mode the Standard ID mask is the upper 11 bits of the full 29-bit mask value.

Clock Configuration

The CAN peripheral uses the CANMCLK (main-clock oscillator) or PCLKB as its clock source (fCAN, CAN System Clock.) The default CAN configuration will provide a CAN bit rate of 500 Kbit using CANMCLK as the clock source. To set the PCLKB frequency, use the **Clocks** tab of the RA

Configuration editor. To change the clock frequency at run-time, use the CGC Interface. Refer to the CGC module guide for more information on configuring clocks.

Clock Limitations

The following clock limitations apply when using the CAN peripheral:

- When using the main oscillator (CANMCLK) as the clock source:
  - fPCLKB >= fCANCLK (fCANCLK = XTAL / Baud Rate Prescaler)
  - The user application must start the main-clock oscillator (XTAL) at run-time using the CGC Interface if it has not already started (for example, if it is not used as the MCU clock source.)
- When using PCLKB as the clock source:
  - For RA6 and RA4 MCUs, the source of the peripheral module clocks must be PLL for the CAN HAL module.
- For RA4M1 and RA4W1 MCUs, the clock frequency ratio of PCLKA and PCLKB must be 2:1 when using the CAN HAL module. Operation is not guaranteed for other settings.
- For RA2 MCUs only CANMCLK (XTAL) may be used as a clock source. The clock frequency ratio of ICLK and PCLKB must be 2:1 when using the CAN HAL module. Operation is not guaranteed for other settings.

  Note

  > *When using CANMCLK (XTAL) as the CAN clock source while running at a reduced main clock speed (under 2x XTAL) be sure to confirm that the XTAL frequency divided by the baud rate prescaler is equal to or less than PCLKB.*

Pin Configuration

The CAN peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. A CAN channel would consist of two pins - CRX and CTX for data transmission/reception.

Usage Notes

Bit Rate Calculation

For convenience, the baudrate of the CAN peripheral is automatically set through the RA Configuration editor using a best effort approach. If the auto-generated baud settings cause deviation that is not tolerable by the application, the user can override the auto-generated settings and put in manually calculated values through RA Configuration editor. For more details on how the baudrate is set refer to section 37.4 "Data Transfer Rate Configuration" of the RA6M3 User's Manual (R01UH0886EJ0100).

FIFO Support

When FIFO Support is enabled, mailboxes 24-27 form a 4-stage transmit FIFO and mailboxes 28-31 form a 4-stage receive FIFO. The receive FIFO supports two independent ID/mask settings for acceptance filtering.

Note

> *Only the base mailbox of each FIFO may be accessed. When writing to the TX FIFO it is recommended to use CAN_MAILBOX_ID_TX_FIFO.*

Limitations

Developers should be aware of the following limitations when using CAN:

- The can_frame_t::id_mode field is only used when Global ID Mode is set to Mixed ID. It is ignored in all other modes.

Examples

Basic Example

This is a basic example of minimal use of the CAN in an application.

```
can_frame_t g_can_tx_frame;
can_frame_t g_can_rx_frame;
volatile bool     g_rx_flag = false;
volatile bool     g_tx_flag = false;
volatile bool     g_err_flag = false;
volatile uint32_t g_rx_id;
void can_callback (can_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case CAN_EVENT_RX_COMPLETE:    /* Receive complete event. */
      {
            g_rx_flag = true;
            g_rx_id   = p_args->frame.id;
 /* Read received frame */
            g_can_rx_frame = p_args->frame;
 break;
       }
 case CAN_EVENT_TX_COMPLETE:    /* Transmit complete event. */
      {
            g_tx_flag = true;
 break;
       }
 case CAN_EVENT_ERR_BUS_OFF:          /* Bus error event. (bus off) */
 case CAN_EVENT_ERR_PASSIVE:          /* Bus error event. (error passive) */
 case CAN_EVENT_ERR_WARNING:          /* Bus error event. (error warning) */
 case CAN_EVENT_BUS_RECOVERY:         /* Bus error event. (bus recovery) */
```

```
 case CAN_EVENT_MAILBOX_MESSAGE_LOST: /* Overwrite/overrun error */
      {
 /* Set error flag */
            g_err_flag = true;
 break;
       }
 default:
      {
 break;
       }
    }
}
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = CAN_BUSY_DELAY;
 /* Initialize the CAN module */
    err = R_CAN_Open(&g_can0_ctrl, &g_can0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    g_can_tx_frame.id               = CAN_DESTINATION_DEVICE_MAILBOX_NUMBER; /* CAN
Destination Device ID */
    g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = CAN_FRAME_TRANSMIT_DATA_BYTES;
 /* Write some data to the transmit frame */
 for (i = 0; i < sizeof(g_can_tx_frame.data); i++)
    {
       g_can_tx_frame.data[i] = (uint8_t) i;
    }
 /* Send data on the bus */
    g_tx_flag = false;
    g_err_flag = false;
    err = R_CAN_Write(&g_can0_ctrl, CAN_MAILBOX_NUMBER_31, &g_can_tx_frame);
```

```
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((true != g_tx_flag) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (true == g_err_flag)
    {
       __BKPT(0);
    }
}
```

External Loop-back Test

This example requires the CTX and CRX pins to be connected. If a CAN tranciever is onboard a 120 Ohm resistor should be connected across CANH and CANL instead. The mailbox numbers are arbitrarily chosen.

```
void can_external_loopback_example (void)
{
 fsp_err_t            err;
    uint32_t             timeout_ms    = CAN_BUSY_DELAY;
 can_operation_mode_t operation_mode = CAN_OPERATION_MODE_NORMAL;
 can_test_mode_t      test_mode      = CAN_TEST_MODE_LOOPBACK_EXTERNAL;
 int      diff = 0;
    uint32_t i    = 0;
    err = R_CAN_Open(&g_can0_ctrl, &g_can0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    err = R_CAN_ModeTransition(&g_can0_ctrl, operation_mode, test_mode);
    assert(FSP_SUCCESS == err);
 /* Clear the data part of receive frame */
    memset(g_can_rx_frame.data, 0, CAN_FRAME_TRANSMIT_DATA_BYTES);
 /* CAN Destination Device ID, in this case it is the same device with another
mailbox */
```

```
    g_can_tx_frame.id                  = CAN_MAILBOX_NUMBER_4;
    g_can_tx_frame.type                = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = CAN_FRAME_TRANSMIT_DATA_BYTES;
 /* Write some data to the transmit frame */
 for (i = 0; i < sizeof(g_can_tx_frame.data); i++)
    {
       g_can_tx_frame.data[i] = (uint8_t) i;
    }
 /* Send data on the bus */
    g_rx_flag = false;
    g_err_flag = false;
    err = R_CAN_Write(&g_can0_ctrl, CAN_MAILBOX_NUMBER_31, &g_can_tx_frame);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((true != g_rx_flag) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if (true == g_err_flag)
    {
       __BKPT(0);
    }
 /* Verify received data */
    diff = memcmp(&g_can_rx_frame.data[0], &g_can_tx_frame.data[0],
CAN_FRAME_TRANSMIT_DATA_BYTES);
 if (0 != diff)
    {
       __BKPT(0);
    }
}
```

Data Structures

struct	can_mailbox_t

struct	can_fifo_interrupt_cfg_t
struct	can_rx_fifo_cfg_t
struct	can_extended_cfg_t

Enumerations

enum	can_status_t
enum	can_error_t
enum	can_mailbox_number_t
enum	can_mailbox_send_receive_t
enum	can_global_id_mode_t
enum	can_message_mode_t
enum	can_clock_source_t
enum	can_fifo_interrupt_mode_t

Data Structure Documentation

◆ can_mailbox_t

struct can_mailbox_t

CAN Mailbox

Data Fields		
uint32_t	mailbox_id	Mailbox ID.
can_id_mode_t	id_mode	Standard or Extended ID. Only used in Mixed ID mode.
can_frame_type_t	frame_type	Frame type for receive mailbox.
can_mailbox_send_receive_t	mailbox_type	Receive or Transmit mailbox type.

◆ can_fifo_interrupt_cfg_t

struct can_fifo_interrupt_cfg_t

CAN FIFO interrupt configuration

Data Fields		
can_fifo_interrupt_mode_t	fifo_int_mode	FIFO interrupts mode (RX and TX combined).
IRQn_Type	tx_fifo_irq	TX FIFO IRQ.

IRQn_Type	rx_fifo_irq	RX FIFO IRQ.

◆ can_rx_fifo_cfg_t

struct can_rx_fifo_cfg_t		
CAN RX FIFO configuration		
Data Fields		
uint32_t	rx_fifo_mask1	RX FIFO acceptance filter mask 1.
uint32_t	rx_fifo_mask2	RX FIFO acceptance filter mask 1.
can_mailbox_t	rx_fifo_id1	RX FIFO acceptance filter ID 1.
can_mailbox_t	rx_fifo_id2	RX FIFO acceptance filter ID 2.

◆ can_extended_cfg_t

struct can_extended_cfg_t		
CAN extended configuration		
Data Fields		
can_clock_source_t	clock_source	Source of the CAN clock.
uint32_t *	p_mailbox_mask	Mailbox mask, one for every 4 mailboxes.
can_mailbox_t *	p_mailbox	Pointer to mailboxes.
can_global_id_mode_t	global_id_mode	Standard or Extended ID mode.
uint32_t	mailbox_count	Number of mailboxes.
can_message_mode_t	message_mode	Overwrite message or overrun.
can_fifo_interrupt_cfg_t const *	p_fifo_int_cfg	Pointer to FIFO interrupt configuration.
can_rx_fifo_cfg_t *	p_rx_fifo_cfg	Pointer to RX FIFO configuration.

Enumeration Type Documentation

◆ can_status_t

enum can_status_t	
CAN Status	
Enumerator	
CAN_STATUS_NEW_DATA	New Data status flag.
CAN_STATUS_SENT_DATA	Sent Data status flag.
CAN_STATUS_RECEIVE_FIFO	Receive FIFO status flag.
CAN_STATUS_TRANSMIT_FIFO	Transmit FIFO status flag.
CAN_STATUS_NORMAL_MBOX_MESSAGE_LOST	Normal mailbox message lost status flag.
CAN_STATUS_FIFO_MBOX_MESSAGE_LOST	FIFO mailbox message lost status flag.
CAN_STATUS_TRANSMISSION_ABORT	Transmission abort status flag.
CAN_STATUS_ERROR	Error status flag.
CAN_STATUS_RESET_MODE	Reset mode status flag.
CAN_STATUS_HALT_MODE	Halt mode status flag.
CAN_STATUS_SLEEP_MODE	Sleep mode status flag.
CAN_STATUS_ERROR_PASSIVE	Error-passive status flag.
CAN_STATUS_BUS_OFF	Bus-off status flag.

◆ can_error_t

enum can_error_t	
CAN Error Code	
Enumerator	
CAN_ERROR_STUFF	Stuff Error.
CAN_ERROR_FORM	Form Error.
CAN_ERROR_ACK	ACK Error.
CAN_ERROR_CRC	CRC Error.
CAN_ERROR_BIT_RECESSIVE	Bit Error (recessive) Error.
CAN_ERROR_BIT_DOMINANT	Bit Error (dominant) Error.
CAN_ERROR_ACK_DELIMITER	ACK Delimiter Error.

◆ can_mailbox_number_t

enum can_mailbox_number_t
CAN Mailbox IDs (MB + FIFO)

◆ can_mailbox_send_receive_t

enum can_mailbox_send_receive_t	
CAN Mailbox type	
Enumerator	
CAN_MAILBOX_RECEIVE	Mailbox is for receiving.
CAN_MAILBOX_TRANSMIT	Mailbox is for sending.

◆ can_global_id_mode_t

enum can_global_id_mode_t	
Global CAN ID mode settings	
Enumerator	
CAN_GLOBAL_ID_MODE_STANDARD	Standard IDs of 11 bits used.
CAN_GLOBAL_ID_MODE_EXTENDED	Extended IDs of 29 bits used.
CAN_GLOBAL_ID_MODE_MIXED	Both Standard and Extended IDs used.

◆ can_message_mode_t

enum can_message_mode_t	
CAN Message Modes	
Enumerator	
CAN_MESSAGE_MODE_OVERWRITE	Receive data will be overwritten if not read before the next frame.
CAN_MESSAGE_MODE_OVERRUN	Receive data will be retained until it is read.

◆ can_clock_source_t

enum can_clock_source_t	
CAN Source Clock	
Enumerator	
CAN_CLOCK_SOURCE_PCLKB	PCLKB is the source of the CAN Clock.
CAN_CLOCK_SOURCE_CANMCLK	CANMCLK is the source of the CAN Clock.

◆ can_fifo_interrupt_mode_t

enum can_fifo_interrupt_mode_t
CAN FIFO Interrupt Modes

Function Documentation

◆ R_CAN_Open()

fsp_err_t R_CAN_Open (can_ctrl_t *const *p_api_ctrl*, can_cfg_t const *const *p_cfg*)

Open and configure the CAN channel for operation.

Example:

```
/* Initialize the CAN module */
   err = R_CAN_Open(&g_can0_ctrl, &g_can0_cfg);
```

Return values

FSP_SUCCESS	Channel opened successfully
FSP_ERR_ALREADY_OPEN	Driver already open.
FSP_ERR_CAN_INIT_FAILED	Channel failed to initialize.
FSP_ERR_ASSERTION	Null pointer presented.

◆ R_CAN_Close()

fsp_err_t R_CAN_Close (can_ctrl_t *const *p_api_ctrl*)

Close the CAN channel.

Return values

FSP_SUCCESS	Channel closed successfully.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	Null pointer presented.

◆ R_CAN_Write()

fsp_err_t R_CAN_Write (can_ctrl_t *const *p_api_ctrl*, uint32_t *mailbox*, can_frame_t *const *p_frame*)

Write data to the CAN channel. Write up to eight bytes to the channel mailbox.

Example:

```
err = R_CAN_Write(&g_can0_ctrl, CAN_MAILBOX_NUMBER_31, &g_can_tx_frame);
assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_CAN_TRANSMIT_NOT_READY	Transmit in progress, cannot write data at this time.
FSP_ERR_CAN_TRANSMIT_FIFO_FULL	Transmit FIFO is full.
FSP_ERR_CAN_RECEIVE_MAILBOX	Mailbox is setup for receive and cannot send.
FSP_ERR_INVALID_ARGUMENT	Data length or frame type invalid.
FSP_ERR_ASSERTION	Null pointer presented

◆ R_CAN_Read()

fsp_err_t R_CAN_Read (can_ctrl_t *const *p_api_ctrl*, uint32_t *mailbox*, can_frame_t *const *p_frame*)

Read data from a mailbox or FIFO.

Note

This function is not supported.

Return values

FSP_ERR_UNSUPPORTED	Function not supported.

◆ R_CAN_ModeTransition()

fsp_err_t R_CAN_ModeTransition (can_ctrl_t *const *p_api_ctrl*, can_operation_mode_t *operation_mode*, can_test_mode_t *test_mode*)

CAN Mode Transition is used to change CAN driver state.

Example:

```
err = R_CAN_ModeTransition(&g_can0_ctrl, operation_mode, test_mode);
assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	Null pointer presented

◆ R_CAN_InfoGet()

fsp_err_t R_CAN_InfoGet (can_ctrl_t *const *p_api_ctrl*, can_info_t *const *p_info*)

Get CAN state and status information for the channel.

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	Null pointer presented

◆ R_CAN_CallbackSet()

fsp_err_t R_CAN_CallbackSet (can_ctrl_t *const *p_api_ctrl*, void(*)(can_callback_args_t *) *p_callback*, void const *const *p_context*, can_callback_args_t *const *p_callback_memory*)
Updates the user callback with the option to provide memory for the callback argument structure. Implements can_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.11 Controller Area Network - Flexible Data (r_canfd)

Modules

Functions

fsp_err_t	R_CANFD_Open (can_ctrl_t *const p_api_ctrl, can_cfg_t const *const p_cfg)
fsp_err_t	R_CANFD_Close (can_ctrl_t *const p_api_ctrl)
fsp_err_t	R_CANFD_Write (can_ctrl_t *const p_api_ctrl, uint32_t const buffer, can_frame_t *const p_frame)
fsp_err_t	R_CANFD_Read (can_ctrl_t *const p_api_ctrl, uint32_t const buffer, can_frame_t *const p_frame)
fsp_err_t	R_CANFD_ModeTransition (can_ctrl_t *const p_api_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode)
fsp_err_t	R_CANFD_InfoGet (can_ctrl_t *const p_api_ctrl, can_info_t *const p_info)
fsp_err_t	R_CANFD_CallbackSet (can_ctrl_t *const p_api_ctrl, void(*p_callback)(can_callback_args_t *), void const *const p_context, can_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the CANFD peripheral on RA MCUs. This module implements the CAN Interface.

Overview

The CANFD module can be used to communicate over CAN networks, optionally using Flexible Data (CAN-FD) to accelerate the data phase. A variety of message filtering and buffer options are available.

Features

Common Features

- Compatibility
  - Send and receive CAN 2.0 and CAN-FD frames on the same channel
  - Bitrate up to 1 Mbps with FD data phase speeds up to 5-8 Mbps
  - Supports both ISO 11898-1
- Buffers
  - 32 global receive Message Buffers (RX MBs)
  - 2-8 global receive FIFOs (RX FIFOs)
  - 4-16 transmit Message Buffers (TX MBs) per channel
- Filtering
  - Each filter rule can be individually configured to accept messages based on:
    - ID
    - Standard or Extended ID (IDE bit)
    - Data or Remote Frame (RTR bit)
    - ID/IDE/RTR mask
    - Minimum DLC (data length) value
- Interrupts
  - Configurable Global RX FIFO Interrupt
    - Configurable per FIFO
    - Interrupt at a certain depth or on every received message
  - Channel TX Interrupt
  - Global Error
    - DLC Check
    - Message Lost
    - FD Payload Overflow
  - Channel Error
    - Bus Error
    - Error Warning
    - Error Passive
    - Bus-Off Entry
    - Bus-Off Recovery
    - Overload
    - Bus Lock
    - Arbitration Loss
    - Transmission Aborted

Per-MCU Specifications

	RA6M5	RA6T2
Channels	2	1
Max nominal bitrate	1 Mbps	1 Mbps
Max data bitrate	8 Mbps	5 Mbps

Filter rules	128	32
TX message buffers	16/ch	4
RX message buffers	32	32
RX FIFOs	8	2
RX Buffer RAM	4864 bytes	1216 bytes
Standards	ISO or Bosch	ISO

Note

Each message buffer comprises 12 header bytes plus data length (8-64 bytes). The above buffer RAM values therefore correspond to the following capacities:

	RA6M5	*RA6T2*
Max 64-byte storage	*64 messages*	*16 messages*
Max 8-byte storage	*243 messages*	*60 messages*

Configuration

Build Time Configurations for r_canfd

The following build time configurations are defined in fsp_cfg/r_canfd_cfg.h:

Configuration	Options	Default	Description
Global Error Interrupt > Priority	MCU Specific Options		This interrupt is fired for each of the error sources selected below.
Global Error Interrupt > Sources	• DLC Check • Message Lost • FD Payload Overflow	0U	Select which errors should trigger an interrupt.
Global Error Interrupt > Callback Channel	• Channel 0 • Channel 1	Channel 0	Specify which channel callback should be called to handle global errors. When starting the driver this channel must be opened first.
Flexible Data (FD) > FD Frame Format	MCU Specific Options		Select whether to use the FD frame standard provided by ISO or Bosch.
Flexible Data (FD) > Protocol Exceptions	• Enabled (ISO 11898-1) • Disabled	Enabled (ISO 11898-1)	Select whether to enter the protocol exception handling state when a RES bit is sampled recessive as defined in

			ISO 11898-1.
Flexible Data (FD) > Payload Overflow	• Reject • Truncate	Reject	Configure whether received messages larger than the destination buffer should be truncated or rejected.
Reception > Message Buffers > Number of Buffers	RX Message Buffer number must be an integer between 0 and 32.	0	Number of message buffers available for reception. As there is no interrupt for message buffer reception it is recommended to use RX FIFOs instead. Set this value to 0 to disable RX Message Buffers.
Reception > Message Buffers > Payload Size	• 8 bytes • 12 bytes • 16 bytes • 20 bytes • 24 bytes • 32 bytes • 48 bytes • 64 bytes	8 bytes	Payload size for all RX Message Buffers.
Reception > FIFOs > FIFO 0 > Enable	• Enabled • Disabled	Enabled	Enable or disable RX FIFO 0.
Reception > FIFOs > FIFO 0 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 0. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 0 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 0. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 0 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 0.
Reception > FIFOs > FIFO 0 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 0.

Reception > FIFOs > FIFO 1 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 1.
Reception > FIFOs > FIFO 1 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 1. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 1 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 1. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 1 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 1.
Reception > FIFOs > FIFO 1 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 1.
Reception > FIFOs > FIFO 2 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 2.
Reception > FIFOs > FIFO 2 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 2. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 2 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 2. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 2 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 2.
Reception > FIFOs > FIFO 2 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 2.
Reception > FIFOs > FIFO 3 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 3.
Reception > FIFOs > FIFO 3 > Interrupt	MCU Specific Options		Set the interrupt mode for RX FIFO 3.

Mode			Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 3 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 3. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 3 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 3.
Reception > FIFOs > FIFO 3 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 3.
Reception > FIFOs > FIFO 4 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 4.
Reception > FIFOs > FIFO 4 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 4. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 4 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 4. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 4 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 4.
Reception > FIFOs > FIFO 4 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 4.
Reception > FIFOs > FIFO 5 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 5.
Reception > FIFOs > FIFO 5 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 5. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set

			below.
Reception > FIFOs > FIFO 5 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 5. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 5 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 5.
Reception > FIFOs > FIFO 5 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 5.
Reception > FIFOs > FIFO 6 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 6.
Reception > FIFOs > FIFO 6 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 6. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 6 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 6. This setting is only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 6 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 6.
Reception > FIFOs > FIFO 6 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 6.
Reception > FIFOs > FIFO 7 > Enable	• Enabled • Disabled	Disabled	Enable or disable RX FIFO 7.
Reception > FIFOs > FIFO 7 > Interrupt Mode	MCU Specific Options		Set the interrupt mode for RX FIFO 7. Threshold mode will only fire an interrupt each time an incoming message crosses the threshold value set below.
Reception > FIFOs > FIFO 7 > Interrupt Threshold	MCU Specific Options		Set the interrupt threshold value for RX FIFO 7. This setting is

			only applicable when the Interrupt Mode is set to 'At Threshold Value'.
Reception > FIFOs > FIFO 7 > Payload Size	MCU Specific Options		Select the message payload size for RX FIFO 7.
Reception > FIFOs > FIFO 7 > Depth	MCU Specific Options		Select the number of stages for RX FIFO 7.
Reception > FIFOs > Interrupt Priority	MCU Specific Options		This priority level will apply to all FIFO interrupts globally.
Reception > Acceptance Filtering > Channel 0 Rule Count	The number of AFL rules must be an integer between 0 and 128.	32	Number of acceptance filter list rules dedicated to Channel 0.
Reception > Acceptance Filtering > Channel 1 Rule Count	The number of AFL rules must be an integer between 0 and 128.	0	Number of acceptance filter list rules dedicated to Channel 1.
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Transmission Priority	• Message ID • Buffer Number	Buffer Number	Select how messages should be prioritized for transmission. In either case, lower numbers indicate higher priority.
DLC Check	• Disabled • Enabled • Enabled w/truncate	config.driver.canfd.dlc_check.disabled	When enabled received messages will be rejected if their DLC field is less than the value configured in the associated AFL rule. If 'Enabled w/truncate' is selected and a message passes the DLC check the DLC field is set to the value in the associated AFL rule and any excess data is discarded.

Configurations for Connectivity > CAN FD (r_canfd)

This module can be added to the Stacks tab via New Stack > Connectivity > CAN FD (r_canfd). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_canfd0	Module name.
General > Channel	Channel should be 0 or 1	0	Specify the CAN channel to use.
Bitrate > Automatic > Nominal Rate (bps)	Must be a valid integer with a maximum of 1MHz.	500000	Specify nominal bitrate in bits per second.
Bitrate > Automatic > Data Rate (bps)	Must be a valid integer greater than 0.	2000000	Specify data bitrate in bits per second.
Bitrate > Automatic > Sample Point (%)	Must be a valid integer between 60 and 99.	75	Specify desired sample point.
Bitrate > Manual > Nominal > Prescaler (divisor)	Value must be a non-negative integer between 1 and 1024.	1	Specify clock divisor for nominal bitrate.
Bitrate > Manual > Nominal > Time Segment 1 (Tq)	Value must be a non-negative integer between 2 and 256.	29	Select the Time Segment 1 value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Nominal > Time Segment 2 (Tq)	Value must be a non-negative integer between 2 and 128.	10	Select the Time Segment 2 value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Nominal > Sync Jump Width (Tq)	Value must be a non-negative integer between 1 and 128.	4	Select the Synchronization Jump Width value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Data > Prescaler (divisor)	Value must be a non-negative integer between 1 and 1024.	1	Specify clock divisor for data bitrate.
Bitrate > Manual > Data > Time Segment 1 (Tq)	Value must be a non-negative integer between 2 and 256.	5	Select the Time Segment 1 value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Data > Time Segment 2 (Tq)	Value must be a non-negative integer between 2 and 128.	2	Select the Time Segment 2 value. Check module usage notes for how to calculate this value.
Bitrate > Manual > Data > Sync Jump Width (Tq)	Value must be a non-negative integer between 1 and 128.	1	Select the Synchronization Jump Width value. Check

			module usage notes for how to calculate this value.
Bitrate > Manual > Use manual settings	• Yes • No	No	Select whether or not to override automatic baudrate generation and instead use the values specified here.
Bitrate > Delay Compensation	• Enable • Disable	Enable	When enabled the CANFD module will automatically compensate for any transceiver or bus delay between transmitted and received bits. When manually supplying bit timing values with delay compensation enabled be sure the data prescaler is 2 or smaller for correct operation.
Interrupts > Callback	Name must be a valid C symbol	canfd0_callback	A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time any interrupt occurs.
Interrupts > Channel Interrupt Priority Level	MCU Specific Options		Channel Error/Transmit interrupt priority.
Transmit Interrupts	MCU Specific Options		Select which TX Message Buffers should trigger an interrupt when transmission is complete.
Channel Error Interrupts	• Error Warning • Error Passive • Bus-Off Entry • Bus-Off Recovery • Overload	0U	Select which channel error interrupt sources to enable.
Filter List Array	Name must be a valid C symbol	p_canfd0_afl	Acceptance Filter List (AFL) rule array symbol name.

Clock Configuration

The CANFD peripheral uses either PLL, PLL2 or the main oscillator as its clock source. The RA Configuration editor will attempt to get as close as possible to the supplied bitrate with the configured clock source. To achieve an exact bitrate the CANFD source clock or divisor may need to be adjusted to meet the criteria in the formula below:

```
bitrate = canfd_clock_hz / ((time_segment_1 + time_segment_2 + 1) * prescalar)
```

For CANFD, the possible values for each element are as follows:

Element	Min	Max (Nominal)	Max (Data)
Bitrate	-	1 Mbps	5-8 Mbps*
Time Segment 1	2 Tq	256 Tq	32 Tq
Time Segment 2	2 Tq	128 Tq	16 Tq
Sync Jump Width	1 Tq	Time Segment 2	Time Segment 2
Prescalar	1	1024	256

- RA6M5 supports up to 8 Mbps; RA6T2 supports up to 5 Mbps.

Use the **Clocks** tab of the RA Configuration editor to configure the CANFD clock source/divisor as well as to set the frequency of PLL or PLL2. To change the clock frequency at run-time, use the CGC Interface. Refer to the CGC module guide for more information on configuring clocks.

Pin Configuration

CANFD channels each control two pins: CRX (receive) and CTX (transmit).

Usage Notes

Buffers

The CANFD driver provides three types of buffers: Transmit Message Buffers (TX MBs), Receive Message Buffers (RX MBs) and Receive FIFOs (RX FIFOs).

TX Message Buffers

TX MBs are used for transmission only. Refer to the hardware manual for your device for information on which TX MBs are available.

Note

The CANFD peripheral continually scans TX MBs for new data. Depending on the provided clock it may be possible to write to multiple TX MBs before transmission begins. In this case, messages will be sent in the priority specified by the Transmission Priority option in the RA Configuration editor.

RX Message Buffers

RX MBs are for reception only and may only hold one message at a time. 32 total RX MBs are available and are shared across all channels.

No interrupts are provided for RX MBs. Use R_CANFD_InfoGet and R_CANFD_Read to poll and read them, respectively.

RX FIFOs

RX FIFOs provide interrupt-driven queue functionality for receiving messages and are shared across all channels. All FIFOs have the following capabilities:

- Up to 64 byte payloads
- Up to 48 (RA6T2) or 128 (RA6M5) message capacity
- Interrupt events:
  - On every received frame OR when filled to a specified fraction of its capacity
  - When a message is overwritten (message received on full FIFO)

Once an interrupt is fired it will continue to fire until the FIFO is emptied and all messages have been passed to user code via the callback. When using the threshold interrupt mode a FIFO can be checked for data and read between interrupts by calling R_CANFD_InfoGet and R_CANFD_Read, respectively.

RX Buffer Pool

The CANFD peripheral has a limited amount of buffer pool RAM available for allocating RX MBs and FIFO stages. The RA Configuration editor will provide a warning when the limit is exceeded.

The number of bytes used by RX MBs and individual FIFOs can be calculated as follows:

```
Total RX MB bytes used = (number of RX MBs enabled) * (RX MB payload size + 12 header
bytes)
RX FIFO bytes used = (number of FIFO stages) * (FIFO payload size + 12 header bytes)
```

Message Filtering (Acceptance Filter List)

To filter messages to the desired message buffer or FIFO the CANFD peripheral uses an Acceptance Filter List (AFL). Each entry in the AFL provides a rule to check a message against along with destination and other filtering information. When a message is received the CANFD peripheral internally checks against every configured AFL rule for the channel. If a match is found the message is transferred to the destination(s) specified in the rule. The default template with one entry is shown below:

```
static const canfd_afl_entry_t p_canfd0_afl[CANFD_CFG_AFL_CH0_RULE_NUM] =
{
    {
       .id =
      {
 /* Specify the ID, ID type and frame type to accept. */
            .id         = 0x00000000,
            .frame_type = CAN_FRAME_TYPE_DATA,
            .id_mode    = CAN_ID_MODE_EXTENDED,
       },
```

```
       .mask =
      {
 /* These values mask which ID/mode bits to compare when filtering messages. */
            .mask_id         = 0x1FFFFFFF,
            .mask_frame_type = 1,
            .mask_id_mode    = 1,
       },
       .destination =
      {
 /* If DLC checking is enabled any messages shorter than the below setting will be
rejected. */
            .minimum_dlc = CANFD_MINIMUM_DLC_0,
 /* Optionally specify a Receive Message Buffer (RX MB) to store accepted frames. RX
MBs do not have an
  * interrupt or overwrite protection and must be checked with R_CANFD_InfoGet and
R_CANFD_Read. */
            .rx_buffer   = CANFD_RX_MB_NONE,
 /* Specify which FIFO(s) to send filtered messages to. Multiple FIFOs can be OR'd
together. */
            .fifo_select_flags = CANFD_RX_FIFO_0,
       }
    }
};
```

AFL templates can be easily added to a project using the Developer Assistance feature in e2 studio. Once the CANFD module is added to a project, drag and drop the elements circled below to build a filter list:

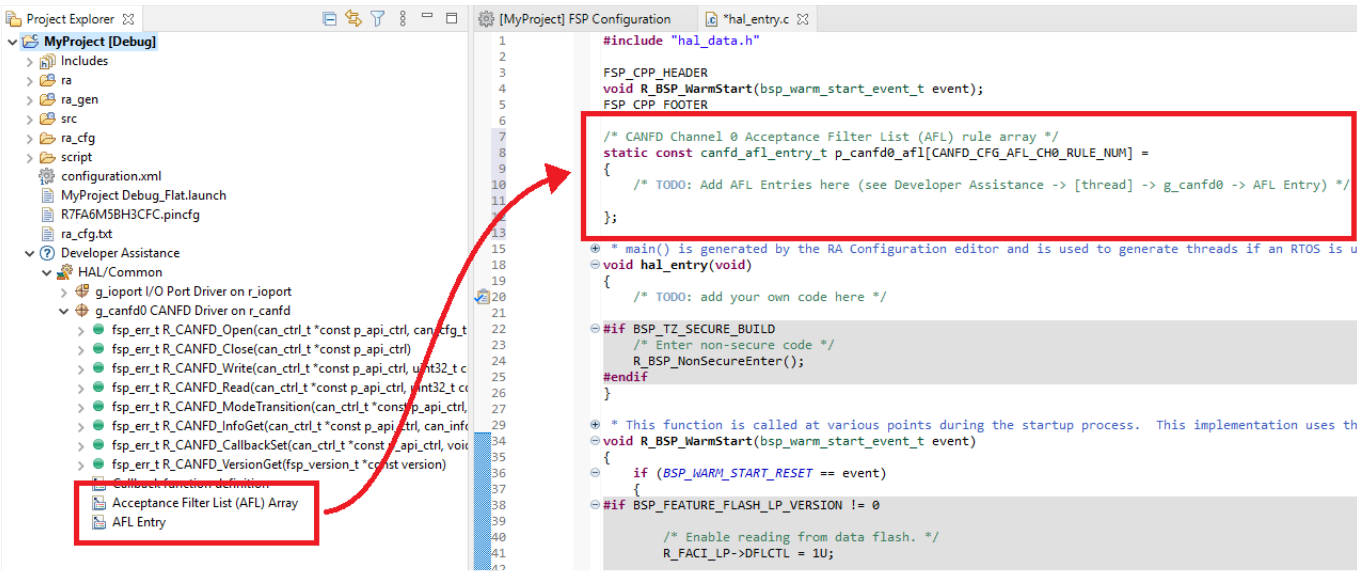


Figure 148: CANFD Developer Assistance AFL Templates

For an example configuration refer to the AFL Example below.

Flexible Data (FD)

Flexible Data is an extension of the CAN protocol allowing for messages up to 64 bytes and higher data bitrates, among other features. The CANFD driver supports the following:

- Sending and receiving FD messages
- Bitrate switching for data phase
- Manual and automatic setting of the error state (ESI) bit

To specify one or more of these options when transmitting set can_frame_t::options with combined values from canfd_frame_options_t. Received messages will automatically have this field filled, if applicable.

```
/* Configure a frame to write 64 bytes with bitrate switching (BRS) enabled */
    g_can_tx_frame.id               = CAN_EXAMPLE_ID;
    g_can_tx_frame.id_mode          = CAN_ID_MODE_STANDARD;
    g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = CAN_EXAMPLE_64_BYTES;
    g_can_tx_frame.options          = CANFD_FRAME_OPTION_FD | CANFD_FRAME_OPTION_BRS;
```

Note

When using bitrate switching be sure to configure the Data Bitrate as desired in the RA Configuration editor.

Bit Rate Calculation

For convenience, the baudrate of the CAN peripheral is automatically set through the RA

Configuration editor using a best effort approach. To increase compliance and reliability two extra options are provided: Link Prescalers and Delay Compensation.

When Link Prescalers is enabled the prescalers for Nominal and Data bitrates will be kept the same, improving clock tolerance in networks using FD frames. This option somewhat limits the available options for bitrate settings, so be sure to check that the automatically generated values are acceptable.

Enabling Delay Compensation instructs the CANFD peripheral to measure TX to RX tranceiver delay and automatically adjust for it, improving the reliability of high-speed FD messages. This option may severely limit available bitrate settings depending on the source clock; it is highly recommended to check the generated values when enabled.

If the auto-generated baud settings cause deviation that is not tolerable by the application the user can override the auto-generated settings and put in manually calculated values through the RA Configuration editor. For more details on how the bitrate is calculated refer to the Clock Configuration section above.

Sync Jump Width

The Sync Jump Width option specifies the maximum number of time quanta that the sample point may be delayed by to account for differences in oscillators on the bus. It should be set to a value between 1 and the configured Time Segment 2 value depending on the maximum permissible clock error.

Error Handling

The CANFD peripheral provides two types of error interrupts: Channel and Global. As the names imply, each channel has its own Channel Error interrupt but there is only one Global Error interrupt. Only the configured channel will receive callbacks for Global Errors.

Error interrupt callbacks will pass either CAN_EVENT_ERR_CHANNEL or CAN_EVENT_ERR_GLOBAL in the can_callback_args_t::event field. A second field, can_callback_args_t::error, provides the actual error code as canfd_error_t. Cast to this enum to retrieve the error condition. See the callback in the Basic Example below for a demonstration.

DLC Checking

When DLC Checking is enabled messages are checked against the destination.minimum_dlc value of each AFL rule. If the data length of a message is less than this value the message will be rejected. When DLC checking is set to "Enabled w/truncate" in the RA Configuration editor any data in excess of the minimum DLC setting will be truncated and the DLC value for the frame will be set to match.

FD Payload Overflow

When an FD message is received with a DLC larger than the destination buffer an FD Payload Overflow interrupt is thrown (if configured). When Payload Overflow is set to "Truncate" the message will still be accepted but only data up to the buffer capacity will be preserved. The DLC value is unchanged in this case; any data beyond this value in the can_frame_t::data array should not be used.

Test Modes

The CANFD peripheral provides three basic test modes: Listen Only, Internal Loopback and External Loopback. Use R_CANFD_ModeTransition to switch to a test mode.

On some MCUs an additional "Internal Bus" test mode is available that allows connecting both CANFD channels together on an internal bus, effectively creating an internal CAN network. See the Internal Bus example below for details.

Limitations

Developers should be aware of the following limitations when using CANFD:

- On RA6M5, RX Message Buffers do not have an associated interrupt. To use them in an application one of the following is recommended:
  - Use R_CANFD_InfoGet to determine if any RX MBs have received data, then use R_CANFD_Read to obtain it
  - Select an RX FIFO as an additional destination for the relevant filter rules and configure the FIFO interrupt/callback as desired
- The CANFD peripheral has a limited amount of buffer pool RAM available for allocating RX MBs and FIFO stages. See the RX Buffer Pool section above for more information.
- When switching modes with R_CANFD_ModeTransition a delay of up to several CAN frames may be incurred. Consult Section 32.3.4.2 "Timing of Channel Mode Change" in the RA6M5 User's Manual (R01UH0891EJ0100) for details.
- Only one channel will receive callbacks for Global Errors. If a different channel is opened first these error interrupts will be suppressed until the specified handler channel is opened.

Examples

AFL Example

The below is an example Acceptance Filter List (AFL) declaration with two rules.

```
const canfd_afl_entry_t p_canfd0_afl[CANFD_CFG_AFL_CH1_RULE_NUM] =
{
 /* Store all data frames with at least 4 bytes from Standard IDs 0x40-0x4F in RX
FIFO 0 and RX FIFO 1 */
    {
       .id =
      {
            .id         = 0x40,
            .frame_type = CAN_FRAME_TYPE_DATA,
            .id_mode    = CAN_ID_MODE_STANDARD
       },
       .mask =
      {
            .mask_id         = 0x7F0,
            .mask_frame_type = 1,
            .mask_id_mode    = 1
```

```
        },
        .destination =
       {
              .minimum_dlc       = CANFD_MINIMUM_DLC_4,
              .rx_buffer         = CANFD_RX_MB_NONE,
              .fifo_select_flags = (canfd_rx_fifo_t) (CANFD_RX_FIFO_0 |
CANFD_RX_FIFO_1)
        }
     },
 /* Store all frames from Extended ID 0x1100 in RX FIFO 2 and RX MB 0 */
     {
        .id =
       {
              .id         = 0x1100,
              .frame_type = CAN_FRAME_TYPE_DATA, // This setting is ignored by the mask
              .id_mode    = CAN_ID_MODE_EXTENDED
        },
        .mask =
       {
              .mask_id         = 0x1FFFFFFF,
              .mask_frame_type = 0,
              .mask_id_mode    = 1
        },
        .destination =
       {
              .minimum_dlc       = CANFD_MINIMUM_DLC_0,
              .rx_buffer         = CANFD_RX_MB_0,
              .fifo_select_flags = CANFD_RX_FIFO_2
        }
     }
};
```

Basic Example

This is a basic example of minimal use of the CANFD module in an application.

Note

On RA6M5 it is recommended to use RX FIFOs for reception as there are no interrupts for RX message buffers.

```
#define CAN_EXAMPLE_ID (0x20)
can_frame_t g_can_tx_frame;
can_frame_t g_can_rx_frame;
volatile canfd_error_t g_err_status = (canfd_error_t) 0;
void canfd_callback (can_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case CAN_EVENT_RX_COMPLETE:    /* Receive complete event. */
      {
 /* Read received frame */
            memcpy(&g_can_rx_frame, p_args->p_frame, sizeof(can_frame_t));
 /* Handle event */
 break;
       }
 case CAN_EVENT_TX_COMPLETE:    /* Transmit complete event. */
      {
 /* Handle event */
 break;
       }
 case CAN_EVENT_ERR_GLOBAL:                       /* Global error. */
 case CAN_EVENT_ERR_CHANNEL:                      /* Channel error. */
      {
 /* Get error status */
            g_err_status = (canfd_error_t) p_args->error; /* Check error code with
canfd_error_t. */
 /* Handle event */
 break;
       }
 default:
      {
 break;
```

```
      }
    }
}
void canfd_basic_example (void)
{
 fsp_err_t err;
 /* Initialize the CAN module */
    err = R_CANFD_Open(&g_canfd0_ctrl, &g_canfd0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Setup frame to write to CAN ID 0x20 */
    g_can_tx_frame.id               = CAN_EXAMPLE_ID;
    g_can_tx_frame.id_mode          = CAN_ID_MODE_STANDARD;
    g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = 8;
    g_can_tx_frame.options          = 0;
 /* Write some data to the transmit frame */
 for (uint32_t i = 0; i < 8; i++)
    {
       g_can_tx_frame.data[i] = (uint8_t) i;
    }
 /* Send data on the bus */
    err = R_CANFD_Write(&g_canfd0_ctrl, CANFD_TX_MB_0, &g_can_tx_frame);
    assert(FSP_SUCCESS == err);
 /* Wait for a transmit callback event */
}
```

Flexible Data

This example demonstrates sending an FD message with bitrate switching over external loopback. The CTX and CRX pins must be connected when using external loopback, though if a CAN tranciever is onboard a 120 Ohm resistor should be connected across CANH and CANL instead.

```
#define CAN_EXAMPLE_64_BYTES 64
void canfd_fd_loopback_example (void)
{
```

```
fsp_err_t err;
    err = R_CANFD_Open(&g_canfd0_ctrl, &g_canfd0_cfg);
/* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
/* Switch to external loopback mode */
    err = R_CANFD_ModeTransition(&g_canfd0_ctrl, CAN_OPERATION_MODE_NORMAL,
CAN_TEST_MODE_LOOPBACK_EXTERNAL);
    assert(FSP_SUCCESS == err);
/* Configure a frame to write 64 bytes with bitrate switching (BRS) enabled */
    g_can_tx_frame.id               = CAN_EXAMPLE_ID;
    g_can_tx_frame.id_mode          = CAN_ID_MODE_STANDARD;
    g_can_tx_frame.type             = CAN_FRAME_TYPE_DATA;
    g_can_tx_frame.data_length_code = CAN_EXAMPLE_64_BYTES;
    g_can_tx_frame.options          = CANFD_FRAME_OPTION_FD | CANFD_FRAME_OPTION_BRS;
/* Write some data to the transmit frame */
for (uint32_t i = 0; i < CAN_DATA_BUFFER_LENGTH; i++)
    {
        g_can_tx_frame.data[i] = (uint8_t) i;
    }
/* Send data on the bus */
    err = R_CANFD_Write(&g_canfd0_ctrl, CANFD_TX_MB_0, &g_can_tx_frame);
    assert(FSP_SUCCESS == err);
/* Wait for a transmit and/or receive callback event */
}
```

Internal Bus

In this example two CANFD channels are connected to the Internal Bus test mode. API error checking has been omitted for clarity.

Note

> *Internal Bus mode is only available on MCUs with more than one CANFD channel. In addition, use of Global Modes for any other purpose is not recommended without consulting the device User's Manual.*

Data Structures

struct	canfd_afl_entry_t
struct	canfd_global_cfg_t

struct	canfd_extended_cfg_t

Enumerations

enum	canfd_status_t
enum	canfd_error_t
enum	canfd_tx_mb_t
enum	canfd_rx_buffer_t
enum	canfd_rx_mb_t
enum	canfd_rx_fifo_t
enum	canfd_minimum_dlc_t
enum	canfd_frame_options_t

Data Structure Documentation

◆ canfd_afl_entry_t

struct canfd_afl_entry_t
AFL Entry (based on R_CANFD_CFDGAFL_Type in renesas.h)

◆ canfd_global_cfg_t

struct canfd_global_cfg_t

CANFD Global Configuration

Data Fields		
uint32_t	global_interrupts	Global control options (CFDGCTR register setting)
uint32_t	global_config	Global configuration options (CFDGCFG register setting)
uint32_t	rx_fifo_config[8]	RX FIFO configuration (CFDRFCCn register settings)
uint32_t	rx_mb_config	Number and size of RX Message Buffers (CFDRMNB register setting)
uint8_t	global_err_ipl	Global Error interrupt priority.
uint8_t	rx_fifo_ipl	RX FIFO interrupt priority.

◆ canfd_extended_cfg_t

struct canfd_extended_cfg_t		
CANFD Extended Configuration		
Data Fields		
canfd_afl_entry_t const *	p_afl	AFL rules list.
uint64_t	txmb_txi_enable	Array of TX Message Buffer enable bits.
uint32_t	error_interrupts	Error interrupt enable bits.
can_bit_timing_cfg_t *	p_data_timing	FD Data Rate (when bitrate switching is used)
uint8_t	delay_compensation	FD Transceiver Delay Compensation (enable or disable)
canfd_global_cfg_t *	p_global_cfg	Global configuration (global error callback channel only)

Enumeration Type Documentation

◆ canfd_status_t

enum canfd_status_t	
CANFD Status	
Enumerator	
CANFD_STATUS_RESET_MODE	Channel in Reset mode.
CANFD_STATUS_HALT_MODE	Channel in Halt mode.
CANFD_STATUS_SLEEP_MODE	Channel in Sleep mode.
CANFD_STATUS_ERROR_PASSIVE	Channel in error-passive state.
CANFD_STATUS_BUS_OFF	Channel in bus-off state.
CANFD_STATUS_TRANSMITTING	Channel is transmitting.
CANFD_STATUS_RECEIVING	Channel is receiving.
CANFD_STATUS_READY	Channel is ready for communication.
CANFD_STATUS_ESI	At least one CAN-FD message was received with the ESI flag set.

◆ canfd_error_t

enum canfd_error_t	
CANFD Error Code	
Enumerator	
CANFD_ERROR_CHANNEL_BUS	Bus Error.
CANFD_ERROR_CHANNEL_WARNING	Error Warning (TX/RX error count over 0x5F)
CANFD_ERROR_CHANNEL_PASSIVE	Error Passive (TX/RX error count over 0x7F)
CANFD_ERROR_CHANNEL_BUS_OFF_ENTRY	Bus-Off State Entry.
CANFD_ERROR_CHANNEL_BUS_OFF_RECOVERY	Recovery from Bus-Off State.
CANFD_ERROR_CHANNEL_OVERLOAD	Overload.
CANFD_ERROR_CHANNEL_BUS_LOCK	Bus Locked.
CANFD_ERROR_CHANNEL_ARBITRATION_LOSS	Arbitration Lost.
CANFD_ERROR_CHANNEL_STUFF	Stuff Error.
CANFD_ERROR_CHANNEL_FORM	Form Error.
CANFD_ERROR_CHANNEL_ACK	ACK Error.
CANFD_ERROR_CHANNEL_CRC	CRC Error.
CANFD_ERROR_CHANNEL_BIT_RECESSIVE	Bit Error (recessive) Error.
CANFD_ERROR_CHANNEL_BIT_DOMINANT	Bit Error (dominant) Error.
CANFD_ERROR_CHANNEL_ACK_DELIMITER	ACK Delimiter Error.
CANFD_ERROR_GLOBAL_DLC	DLC Error.
CANFD_ERROR_GLOBAL_MESSAGE_LOST	Message Lost.
CANFD_ERROR_GLOBAL_PAYLOAD_OVERFLOW	FD Payload Overflow.
CANFD_ERROR_GLOBAL_TXQ_OVERWRITE	TX Queue Message Overwrite.
CANFD_ERROR_GLOBAL_TXQ_MESSAGE_LOST	TX Queue Message Lost.
CANFD_ERROR_GLOBAL_CH0_SCAN_FAIL	Channel 0 RX Scan Failure.

CANFD_ERROR_GLOBAL_CH1_SCAN_FAIL	Channel 1 RX Scan Failure.
CANFD_ERROR_GLOBAL_CH0_ECC	Channel 0 ECC Error.
CANFD_ERROR_GLOBAL_CH1_ECC	Channel 1 ECC Error.

◆ canfd_tx_mb_t

enum canfd_tx_mb_t
CANFD Transmit Message Buffer (TX MB)

◆ canfd_rx_buffer_t

enum canfd_rx_buffer_t
CANFD Receive Buffer (MB + FIFO)

◆ canfd_rx_mb_t

enum canfd_rx_mb_t
CANFD Receive Message Buffer (RX MB)

◆ canfd_rx_fifo_t

enum canfd_rx_fifo_t
CANFD Receive FIFO (RX FIFO)

◆ canfd_minimum_dlc_t

enum canfd_minimum_dlc_t
CANFD AFL Minimum DLC settings

◆ canfd_frame_options_t

enum canfd_frame_options_t

CANFD Frame Options

Enumerator	
CANFD_FRAME_OPTION_ERROR	Error state set (ESI).
CANFD_FRAME_OPTION_BRS	Bit Rate Switching (BRS) enabled.
CANFD_FRAME_OPTION_FD	Flexible Data frame (FDF).

Function Documentation

◆ R_CANFD_Open()

fsp_err_t R_CANFD_Open (can_ctrl_t *const *p_api_ctrl*, can_cfg_t const *const *p_cfg*)

Open and configure the CANFD channel for operation.

Example:

```
/* Initialize the CAN module */
    err = R_CANFD_Open(&g_canfd0_ctrl, &g_canfd0_cfg);
```

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ALREADY_OPEN	Driver already open.
FSP_ERR_IN_USE	Channel is already in use.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel does not exist on this MCU.
FSP_ERR_ASSERTION	A required pointer was NULL.
FSP_ERR_CAN_INIT_FAILED	The provided nominal or data bitrate is invalid.
FSP_ERR_CLOCK_INACTIVE	CANFD source clock is disabled (PLL or PLL2).

◆ R_CANFD_Close()

fsp_err_t R_CANFD_Close (can_ctrl_t *const *p_api_ctrl*)

Close the CANFD channel.

Return values

FSP_SUCCESS	Channel closed successfully.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	Null pointer presented.

◆ R_CANFD_Write()

fsp_err_t R_CANFD_Write (can_ctrl_t *const *p_api_ctrl*, uint32_t *buffer*, can_frame_t *const *p_frame*)

Write data to the CANFD channel.

Example:

```
/* Send data on the bus */
    err = R_CANFD_Write(&g_canfd0_ctrl, CANFD_TX_MB_0, &g_can_tx_frame);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_CAN_TRANSMIT_NOT_READY	Transmit in progress, cannot write data at this time.
FSP_ERR_INVALID_ARGUMENT	Data length or buffer number invalid.
FSP_ERR_INVALID_MODE	An FD option was set on a non-FD frame.
FSP_ERR_ASSERTION	Null pointer presented

◆ R_CANFD_Read()

fsp_err_t R_CANFD_Read (can_ctrl_t *const *p_api_ctrl*, uint32_t *buffer*, can_frame_t *const *p_frame*)

Read data from a CANFD Message Buffer or FIFO.

Example: snippet r_canfd_example.c R_CANFD_Read

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_ARGUMENT	Buffer number invalid.
FSP_ERR_ASSERTION	p_api_ctrl or p_frame is NULL.
FSP_ERR_BUFFER_EMPTY	Buffer or FIFO is empty.

◆ R_CANFD_ModeTransition()

fsp_err_t R_CANFD_ModeTransition (can_ctrl_t *const *p_api_ctrl*, can_operation_mode_t *operation_mode*, can_test_mode_t *test_mode*)

Switch to a different channel, global or test mode.

Example:

```
/* Switch to external loopback mode */
   err = R_CANFD_ModeTransition(&g_canfd0_ctrl, CAN_OPERATION_MODE_NORMAL,
CAN_TEST_MODE_LOOPBACK_EXTERNAL);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	Null pointer presented
FSP_ERR_INVALID_MODE	Cannot change to the requested mode from the current global mode.

◆ R_CANFD_InfoGet()

fsp_err_t R_CANFD_InfoGet (can_ctrl_t *const *p_api_ctrl*, can_info_t *const *p_info*)

Get CANFD state and status information for the channel.

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	Null pointer presented

◆ R_CANFD_CallbackSet()

fsp_err_t R_CANFD_CallbackSet (can_ctrl_t *const *p_api_ctrl*, void(*)(can_callback_args_t *) *p_callback*, void const *const *p_context*, can_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements can_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.12 Consumer Electronics Control (r_cec)

Modules

Functions

fsp_err_t	R_CEC_Open (cec_ctrl_t *const p_ctrl, cec_cfg_t const *const p_cfg)
fsp_err_t	R_CEC_MediaInit (cec_ctrl_t *const p_ctrl, cec_addr_t local_address)
fsp_err_t	R_CEC_Close (cec_ctrl_t *const p_ctrl)
fsp_err_t	R_CEC_Write (cec_ctrl_t *const p_ctrl, cec_message_t const *const p_message, uint32_t message_size)

fsp_err_t	R_CEC_StatusGet (cec_ctrl_t *const p_ctrl, cec_status_t *const p_status)
fsp_err_t	R_CEC_CallbackSet (cec_ctrl_t *const p_ctrl, void(*p_callback)(cec_callback_args_t *), void const *const p_context, cec_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the CEC peripheral on RA MCUs. This module implements the CEC Interface.

Overview

The HDMI CEC HAL module provides a high-level API for CEC applications and supports the CEC peripheral available on RA microcontroller hardware. A user-callback function must be defined that the driver will invoke when data received, transmission complete, or error interrupts are received. The callback is passed a parameter which indicates the event as well as received data (if available).

Features

- Conforms to High Definition Multimedia Interface (HDMI) Consumer Electronics Control (CEC) standard Ver. 1.4b.
- Full range of local address settings (TV, Recording Device, Playback Device, etc.)
- Data filtering based on matching destination address and local address.
- Supports a user-callback function (required), invoked when transmit, receive, or error interrupts are received.

Configuration

Build Time Configurations for r_cec

The following build time configurations are defined in fsp_cfg/r_cec_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
CEC Message Max Data Size	Manual Entry	14	Maximum Data Size for CEC Message Transmission/Reception.

Configurations for Connectivity > CEC (r_cec)

This module can be added to the Stacks tab via New Stack > Connectivity > CEC (r_cec). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
General > Name	Name must be a valid	g_cec0	Module name

	C symbol		
General > Callback Function	Manual Entry	g_rm_cec0_callback	Callback function
Control Configuration > Clock Select	• PCLKB / 32 • PCLKB / 64 • PCLKB / 128 • PCLKB / 256 • PCLKB / 512 • PCLKB / 1024	PCLKB / 1024	CEC Clock Select Configuration
Control Configuration > Ack Bit Timing Error Enable	• Disabled • Enabled	Enabled	CEC Ack Bit Timing Error Enable
Control Configuration > Signal-Free Time Bit Width	• 3-data bit width • 5-data bit width • 7-data bit width • Does not detect signal-free time.	7-data bit width	Signal-Free Time Data Bit Width Select
Control Configuration > Start Bit Error Detection Enable	• Disabled • Enabled	Enabled	Enable to detect timing errors during start bit reception.
Control Configuration > Bus Lock Detection Enable	• Disabled • Enabled	Enabled	Enable to detect sticking of receive data to high or low.
Control Configuration > Digital Filter Enable	• Disabled • Enabled	Enabled	Enable to use a digital filter.
Control Configuration > Long Bit Width Error Pulse Output Enable	• Disabled • Enabled	Disabled	Enable to output an error handling pulse when a long bit width error is detected.
Control Configuration > Start Detection Reception Restart Enable	• Disabled • Enabled	Enabled	Enable to restart reception after a start bit error is detected.
Bit Width Timing > Transmit > Start Bit Low Time	CEC transmission start bit low width setting must be a positive integer.	180	CEC transmission start bit low width setting (CEC Clock Cycles).
Bit Width Timing > Transmit > Start Bit High Time	CEC transmission start bit high width setting must be a positive integer.	220	CEC transmission start bit high width setting (CEC Clock Cycles).
Bit Width Timing > Transmit > Logical Zero Low Time	CEC transmission logical zero low width setting must be a positive integer.	73	CEC transmission logical zero low width setting (CEC Clock Cycles).
Bit Width Timing >	CEC transmission	29	CEC transmission

Transmit > Logical One Low Time	logical one low width setting must be a positive integer.		logical one low width setting (CEC Clock Cycles).
Bit Width Timing > Transmit > Overall Bit Width Time	CEC transmission overall data bit width time setting must be a positive integer.	117	CEC transmission overall data bit width time setting (CEC Clock Cycles).
Bit Width Timing > Receive > Data Sample Time	CEC reception data sampling time must be a positive integer.	49	CEC reception data sampling time setting (CEC Clock Cycles).
Bit Width Timing > Receive > Data Bit Reference Width	CEC reception data sampling time must be a positive integer.	117	CEC data bit reference width setting (CEC Clock Cycles).
Bit Width Timing > Receive > Start Bit Low Min Time	CEC reception start bit minimum low width setting must be a positive integer.	171	CEC reception start bit minimum low width setting (CEC Clock Cycles). Not used when Start Bit Error Detection and restart Rx on Error are not enabled.
Bit Width Timing > Receive > Start Bit Low Max Time	CEC reception start bit maximum low width setting must be a positive integer.	190	CEC reception start bit maximum low width setting (CEC Clock Cycles). Not used when Start Bit Error Detection and restart Rx on Error are not enabled.
Bit Width Timing > Receive > Start Bit Min Time	CEC start bit minimum time setting must be a positive integer.	210	CEC start bit minimum time setting (CEC Clock Cycles). Not used when Start Bit Error Detection and restart Rx on Error are not enabled.
Bit Width Timing > Receive > Start Bit Max Time	CEC reception start bit maximum time setting must be a positive integer.	229	CEC start bit maximum time setting (CEC Clock Cycles). Not used when Start Bit Error Detection and restart Rx on Error are not enabled.
Bit Width Timing > Receive > Logical Zero Low Min Time	CEC reception logical zero minimum low width setting must be a positive integer.	64	CEC reception logical zero minimum low width setting (CEC Clock Cycles).
Bit Width Timing > Receive > Logical Zero	CEC reception locical zero maximum low	83	CEC reception logical zero maximum low

Low Max Time	width setting must be a positive integer.		width setting (CEC Clock Cycles).
Bit Width Timing > Receive > Logical One Low Min Time	CEC reception logical one minimum low width setting must be a positive integer.	20	CEC reception logical one minimum low width setting (CEC Clock Cycles).
Bit Width Timing > Receive > Logical One Low Max Time	CEC reception logical one maximum low width setting must be a positive integer.	39	CEC reception logical one maximum low width (CEC Clock Cycles).
Bit Width Timing > Receive > Overall Bit Width Min Time	CEC reception overall minimum bit width setting must be a positive integer.	100	CEC reception overall minimum bit width setting (CEC Clock Cycles).
Bit Width Timing > Receive > Overall Bit Width Max Time	CEC reception overall maximum bit width setting must be a positive integer.	134	CEC reception overall maximum bit width setting (CEC Clock Cycles).
Interrupts > Interrupt Priority Level	MCU Specific Options		Error/Data/Message interrupt priority level.
Interrupts > Communication Complete Interrupt Timing	• After Last Frame and Signal Free Time • After Last Frame • After Signal Free Time	After Last Frame and Signal Free Time	Communication Complete Interrupt (INTCE) Generation Timing Select
Interrupts > Address Mismatch Interrupt Enable	• Disabled • Enabled	Disabled	Enable to generate an interrupt when the addresses do not match.
Interrupts > Data Interrupt Timing Selection	• EOM timing (9th bit of data) • ACK Timing (10th bit of data)	EOM timing (9th bit of data)	INTDA reception interrupt timing selection (EOM or ACK).

Clock Configuration

The CEC peripheral uses the CECCLK or PCLKB as its clock source. To set the PCLKB frequency, use the Clocks tab of the RA Configuration editor.

Note

The selected clock and configured divider must be configured in the range of 23.4375 to 78.125 kHz.

Pin Configuration

A CEC channel uses one data pin - CECIO for data transmission and reception.

The output type for each pin should be set to **n-ch open drain** for most hardware designs. This can be configured in **Pins** tab of the RA Configuration editor by selecting the pin under Pin Selection->Ports.

Usage Notes

CEC Device Addresses

The CEC standard provides 13 device addresses that may be requested based on a device's primary function. Use R_CEC_MediaInit to request a specific address before starting communication with other devices.

Note

Address 0 is always the primary display (TV). Do not attempt to allocate this address unless your device is intended to function as a display.

Limitations

Developers should be aware of the following limitations when using the CEC module:

- R_CEC_MediaInit may return FSP_ERR_IN_USE for up to 45 milliseconds after R_CEC_Open while the hardware initializes.
- The CECIO pin must be set to n-ch open drain mode.

Examples

Basic Example

This is a basic example of minimal use of the CEC in an application.

```
/*************************************************************************************
***********************************
 * Application defined callback
 * - May be assigned at compile-time via the e2 Studio configuration tool or set at
run-time via R_CEC_CallbackSet()
 ************************************************************************************
**********************************/
void cec_callback (cec_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case CEC_EVENT_READY:
      {
 /* Application processing for address allocation success. */
 break;
```

```
      }
 case CEC_EVENT_TX_COMPLETE:
     {
 /* Any required processing after transmission has completed. */
 break;
      }
 case CEC_EVENT_ERR:
     {
 /* Error processing. See cec_error_t for possible errors. */
 break;
      }
 case CEC_EVENT_RX_DATA:
     {
 /* Application to store and process received data bytes. */
 break;
      }
 case CEC_EVENT_RX_COMPLETE:
     {
 /* Application processing for message reception complete. */
      }
    }
}
/*************************************************************************************
***********************************
 * Basic example
 ************************************************************************************
**********************************/
#define CEC_TIMEOUT_MS (50)
#define CEC_MSG_STANDBY (0X36)  /* See CEC Specification for message definitions */
void basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the CEC module */
    err = R_CEC_Open(&g_cec0_ctrl, &g_cec0_cfg);
```

```
    assert(FSP_SUCCESS == err);
 /* Initialize the CEC module and allocate an address */
    uint32_t timeout_ms = CEC_TIMEOUT_MS;
 do
    {
 /* R_CEC_MediaInit may return FSP_ERR_IN_USE for up to 45 milliseconds after calling
R_CEC_Open */
       err = R_CEC_MediaInit(&g_cec0_ctrl, CEC_ADDR_TV);
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
    } while ((FSP_ERR_IN_USE == err) && --timeout_ms);
    assert(timeout_ms);
    assert(FSP_SUCCESS == err);
 /* Wait for local address allocation and CEC bus to be free */
    cec_status_t status;
    err = R_CEC_StatusGet(&g_cec0_ctrl, &status);
 while ((FSP_SUCCESS == err) && (CEC_STATE_READY != status.state))
    {
      err = R_CEC_StatusGet(&g_cec0_ctrl, &status);
      assert(FSP_SUCCESS == err);
    }
 cec_message_t cec_msg;
    uint8_t       total_transmit_size;
    cec_msg.destination = CEC_ADDR_BROADCAST;       /* For this example, send message
to all devices on the bus */
    cec_msg.opcode      = CEC_MSG_STANDBY;          /* Send Standby Request */
    memset(cec_msg.data, 0U, sizeof(cec_msg.data)); /* See CEC Specification for
other message data structures */
    total_transmit_size = 2U;                      /* Total message size, including
header, opcode, and data */
 /* Send asynchronous message.
  * - Application will then be free for other processing while message is being sent.
  * - Do not modify the message buffer until transmission has completed. */
    err = R_CEC_Write(&g_cec0_ctrl, &cec_msg, total_transmit_size);
    assert(FSP_SUCCESS == err);
```

```
}
```

Function Documentation

◆ R_CEC_Open()

fsp_err_t R_CEC_Open (cec_ctrl_t *const *p_ctrl*, cec_cfg_t const *const *p_cfg*)

Open and configure the CEC module for operation.

Example:

```
/* Open the CEC module */
    err = R_CEC_Open(&g_cec0_ctrl, &g_cec0_cfg);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	CEC Module opened successfully.
FSP_ERR_ALREADY_OPEN	Driver already open.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_IRQ_BSP_DISABLED	Interrupts are not enabled.

◆ R_CEC_MediaInit()

fsp_err_t R_CEC_MediaInit (cec_ctrl_t *const *p_ctrl*, cec_addr_t *local_address*)

Allocate provided CEC Local Address and Initialize the CEC module for operation.

Note

After calling R_CEC_Open this function may return FSP_ERR_IN_USE for up to 45 milliseconds.

Example:

```
/* Initialize the CEC module and allocate an address */
   uint32_t timeout_ms = CEC_TIMEOUT_MS;
do
   {
/* R_CEC_MediaInit may return FSP_ERR_IN_USE for up to 45 milliseconds after calling
R_CEC_Open */
      err = R_CEC_MediaInit(&g_cec0_ctrl, CEC_ADDR_TV);
R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_MILLISECONDS);
   } while ((FSP_ERR_IN_USE == err) && --timeout_ms);
   assert(timeout_ms);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	CEC Module Initialized successfully.
FSP_ERR_ASSERTION	An input argument is invalid or callback has not been set.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_IN_USE	HDMI CEC Bus is currently in use. Try again later.

◆ R_CEC_Close()

fsp_err_t R_CEC_Close (cec_ctrl_t *const *p_ctrl*)

Close the CEC module.

Return values

FSP_SUCCESS	CEC Module closed successfully.
FSP_ERR_ASSERTION	An input argument is invalid.

◆ R_CEC_Write()

fsp_err_t R_CEC_Write (cec_ctrl_t *const *p_ctrl*, cec_message_t const *const *p_message*, uint32_t *message_size*)

Write data to the CEC bus. Data transmission is asynchronous. Provided message buffer should not be modified until transmission is complete.

Data Transmission follows the pattern defined be the HDMI CEC Specification:

Data	Description	Size
Start Bit	Managed by Hardware, per config	N/A
Header Block	Source/Destination Identifier	1 Byte
Data Block 1	Opcode Value (Optional)	1 Byte
Data Block 2	Operands (Optional)	Variable (0-14 Bytes Typical)

Example:

```
 cec_message_t cec_msg;
    uint8_t       total_transmit_size;
    cec_msg.destination = CEC_ADDR_BROADCAST;       /* For this example, send message
to all devices on the bus */
    cec_msg.opcode      = CEC_MSG_STANDBY;          /* Send Standby Request */
    memset(cec_msg.data, 0U, sizeof(cec_msg.data)); /* See CEC Specification for
other message data structures */
    total_transmit_size = 2U;                      /* Total message size, including
header, opcode, and data */
 /* Send asynchronous message.
  * - Application will then be free for other processing while message is being sent.
  * - Do not modify the message buffer until transmission has completed. */
    err = R_CEC_Write(&g_cec0_ctrl, &cec_msg, total_transmit_size);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_NOT_INITIALIZED	Module has not been successfully initialized.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_INVALID_SIZE	Invalid message size.
FSP_ERR_IN_USE	HDMI CEC Bus is currently in use. Try again

	later.

◆ **R_CEC_StatusGet()**

fsp_err_t R_CEC_StatusGet (cec_ctrl_t *const *p_ctrl*, cec_status_t *const *p_status*)

Provides the state and status information according to the provided CEC control instance.

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_ASSERTION	An input argument is invalid.

◆ **R_CEC_CallbackSet()**

fsp_err_t R_CEC_CallbackSet (cec_ctrl_t *const *p_ctrl*, void(*)(cec_callback_args_t *) *p_callback*, void const *const *p_context*, cec_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements cec_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.13 Clock Generation Circuit (r_cgc)

Modules

Functions

fsp_err_t	R_CGC_Open (cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg)
fsp_err_t	R_CGC_ClocksCfg (cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg)
fsp_err_t	R_CGC_ClockStart (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source,

	cgc_pll_cfg_t const *const p_pll_cfg)
fsp_err_t	R_CGC_ClockStop (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)
fsp_err_t	R_CGC_ClockCheck (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)
fsp_err_t	R_CGC_SystemClockSet (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg)
fsp_err_t	R_CGC_SystemClockGet (cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg)
fsp_err_t	R_CGC_OscStopDetectEnable (cgc_ctrl_t *const p_ctrl)
fsp_err_t	R_CGC_OscStopDetectDisable (cgc_ctrl_t *const p_ctrl)
fsp_err_t	R_CGC_OscStopStatusClear (cgc_ctrl_t *const p_ctrl)
fsp_err_t	R_CGC_CallbackSet (cgc_ctrl_t *const p_api_ctrl, void(*p_callback)(cgc_callback_args_t *), void const *const p_context, cgc_callback_args_t *const p_callback_memory)
fsp_err_t	R_CGC_Close (cgc_ctrl_t *const p_ctrl)

Detailed Description

Driver for the CGC peripheral on RA MCUs. This module implements the CGC Interface.

Note

> ***This module is not required for the initial clock configuration. Initial clock settings are configurable on the **Clocks*** *tab of the RA Configuration editor. The initial clock settings are applied by the BSP during the startup process before main.***

Overview

Features

The CGC module supports runtime modifications of clock settings. Key features include the following:

- Supports changing the system clock source to any of the following options (provided they are supported on the MCU):
  - High-speed on-chip oscillator (HOCO)
  - Middle-speed on-chip oscillator (MOCO)
  - Low-speed on-chip oscillator (LOCO)
  - Main oscillator (external resonator or external clock input frequency)
  - Sub-clock oscillator (external resonator)
  - PLL/PLL2 (not available on all MCUs)
- When the system core clock frequency changes, the following things are updated:
  - The CMSIS standard global variable SystemCoreClock is updated to reflect the new clock frequency.

  - Wait states for ROM and RAM are adjusted to the minimum supported value for the new clock frequency.
  - The operating power control mode is updated to the minimum supported value for the new clock settings.
- Supports starting or stopping any of the system clock sources
- Supports changing dividers for the internal clocks
- Supports the oscillation stop detection feature

Internal Clocks

The RA microcontrollers have up to seven internal clocks. Not all internal clocks exist on all MCUs. Each clock domain has its own divider that can be updated in R_CGC_SystemClockSet(). The dividers are subject to constraints described in the footnote of the table "Specifications of the Clock Generation Circuit for the internal clocks" in the hardware manual.

The internal clocks include:

- System clock (ICLK): core clock used for CPU, flash, internal SRAM, DTC, and DMAC
- PCLKA/PCLKB/PCLKC/PCLKD: Peripheral clocks, refer to the table "Specifications of the Clock Generation Circuit for the internal clocks" in the hardware manual to see which peripherals are controlled by which clocks.
- FCLK: Clock source for reading data flash and for programming/erasure of both code and data flash.
- BCLK: External bus clock

Configuration

Note

The initial clock settings are configurable on the ***Clocks*** *tab of the RA Configuration editor.*
There is a configuration to enable the HOCO on reset in the OFS1 settings on the BSP tab.
The following clock related settings are configurable in the RA Common section on the BSP tab:

  - *Main Oscillator Wait Time*
  - *Main Oscillator Clock Source (external oscillator or crystal/resonator)*
  - *Subclock Populated*
  - *Subclock Drive*
  - *Subclock Stabilization Time (ms)*

The default stabilization times are determined based on development boards provided by Renesas, but are generally valid for most designs. Depending on the target board hardware configuration and requirements these values may need to be adjusted for reliability or startup speed.

Build Time Configurations for r_cgc

The following build time configurations are defined in fsp_cfg/r_cgc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for System > Clock Generation Circuit (r_cgc)

This module can be added to the Stacks tab via New Stack > System > Clock Generation Circuit (r_cgc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_cgc0	Module name.
NMI Callback	Name must be a valid C symbol	NULL	A user callback function must be provided if oscillation stop detection is used. If this callback function is provided, it is called from the NMI handler if the main oscillator stops.

Clock Configuration

This module is used to configure the system clocks. There are no module specific clock configurations required to use it.

Pin Configuration

The CGC module controls the output of the CLOCKOUT signal.

If an external oscillator is used the XTAL and EXTAL pins must be configured accordingly. When running from an on chip oscillator there is no requirement for the main clock external oscillator. In this case, the XTAL and EXTAL pins can be set to a different function in the RA Configuration editor.

The functionality of the subclock external oscillator pins XCIN and XCOUT is fixed.

Usage Notes

NMI Interrupt

The CGC timer uses the NMI for oscillation stop detection of the main oscillator after R_CGC_OscStopDetectEnable is called. The NMI is enabled by default. No special configuration is required. When the NMI is triggered, the callback function registered during R_CGC_Open() is called.

Starting or Stopping the Subclock

If the Subclock Populated property is set to Populated on the BSP configuration tab, then the subclock is started in the BSP startup routine. Otherwise, it is stopped in the BSP startup routine. Starting and stopping the subclock at runtime is not recommended since the stabilization requirements typically negate the negligible power savings.

The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning

> The subclock can take up to several seconds to stabilize. RA startup code does not wait for subclock stabilization unless the subclock is the main clock source. In this case the default wait time is 1000ms (1 second). When running AGT or RTC off the subclock, the application must ensure the subclock is stable before starting operation. Because there is no hardware stabilization status bit for the subclock R_CGC_ClockCheck cannot be used to optimize this wait.

Changing the subclock state during R_CGC_ClocksCfg() is not supported.

Low Power Operation

If "Use Low Voltage Mode" is enabled in the BSP MCU specific properties (not available on all MCUs), the MCU is always in low voltage mode and no other power modes are considered. The following conditions must be met for the MCU to run in low voltage mode:

- Requires HOCO to be running, so HOCO cannot be stopped in low voltage mode
- Requires PLL to be stopped, so PLL APIs are not available in low voltage mode
- Requires ICLK <= 4 MHz
- If oscillation stop detection is used, dividers of 1 or 2 cannot be used for any clock

If "Use Low Voltage Mode" is not enabled, the MCU applies the lowest power mode by searching through the following list in order and applying the first power mode that is supported under the current conditions:

- Subosc-speed mode (lowest power)
    - Requires system clock to be LOCO or subclock
    - Requires MOCO, HOCO, main oscillator, and PLL (if present) to be stopped
    - Requires ICLK and FCLK dividers to be 1
- Low-speed mode
    - Requires PLL to be stopped
    - Requires ICLK <= 1 MHz
    - If oscillation stop detection is used, dividers of 1, 2, 4, or 8 cannot be used for any clock
- Middle-speed mode (not supported on all MCUs)
    - Requires ICLK <= 8 MHz
- High-speed mode
    - Default mode if no other operating mode is supported

Refer to the section "Function for Lower Operating Power Consumption" in the "Low Power Modes" chapter of the hardware manual for MCU specific information about operating power control modes.

Note

The DCDC regulator (if present) is only available in Middle- and High-speed modes. The BSP will automatically switch between DCDC and LDO when switching between compatible and incompatible modes if the DCDC regulator is in use. Switching to the LDO incurs a 60 microsecond critical section wherein all interrupts AND peripherals are stopped. Switching back to DCDC from the LDO incurs an additional 22 microsecond critical section (peripherals running).

When low voltage mode is not used, the following functions adjust the operating power control mode to ensure it remains within the hardware specification and to ensure the MCU is running at the optimal operating power control mode:

- R_CGC_ClockStart()
- R_CGC_ClockStop()
- R_CGC_SystemClockSet()
- R_CGC_OscStopDetectEnable()
- R_CGC_OscStopDetectDisable()

Note

FSP APIs, including these APIs, are not thread safe. These APIs and any other user code that modifies the operating power control mode must not be allowed to interrupt each other. Proper care must be taken during application design if these APIs are used in threads or interrupts to ensure this constraint is met.

No action is required by the user of these APIs. This section is provided for informational purposes only.

Examples

Basic Example

This is a basic example of minimal use of the CGC in an application.

```
void cgc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the CGC module. */
    err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Change the system clock to LOCO for power saving. */
 /* Start the LOCO. */
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_LOCO, NULL);
    assert(FSP_SUCCESS == err);
 /* Wait for the LOCO stabilization wait time.
  *
  * NOTE: The MOCO, LOCO and subclock do not have stabilization status bits, so any
stabilization time must be
  * performed via a software wait when starting these oscillators. For all other
oscillators, R_CGC_ClockCheck can
  * be used to verify stabilization status.
  */
 R_BSP_SoftwareDelay(BSP_FEATURE_CGC_LOCO_STABILIZATION_MAX_US,
BSP_DELAY_UNITS_MICROSECONDS);
 /* Set divisors. Divisors for clocks that don't exist on the MCU are ignored. */
 cgc_divider_cfg_t dividers =
    {
 /* PCLKB is not used in this application, so select the maximum divisor for lowest
power. */
       .pclkb_div = CGC_SYS_CLOCK_DIV_64,
 /* PCLKD is not used in this application, so select the maximum divisor for lowest
```

```
power. */
       .pclkd_div = CGC_SYS_CLOCK_DIV_64,
 /* ICLK is the MCU clock, allow it to run as fast as the LOCO is capable. */
       .iclk_div = CGC_SYS_CLOCK_DIV_1,
 /* These clocks do not exist on some devices. If any clocks don't exist, set the
divider to 1. */
       .pclka_div = CGC_SYS_CLOCK_DIV_1,
       .pclkc_div = CGC_SYS_CLOCK_DIV_1,
       .fclk_div = CGC_SYS_CLOCK_DIV_1,
       .bclk_div = CGC_SYS_CLOCK_DIV_1,
    };
 /* Switch the system clock to LOCO. */
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_LOCO, &dividers);
    assert(FSP_SUCCESS == err);
}
```

Configuring Multiple Clocks

This example demonstrates switching to a new source clock and stopping the previous source clock in a single function call using R_CGC_ClocksCfg().

```
void cgc_clocks_cfg_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the CGC module. */
    err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Change the system clock to PLL running from the main oscillator. */
 /* Assuming the system clock is MOCO, switch to HOCO. */
 cgc_clocks_cfg_t clocks_cfg;
    clocks_cfg.system_clock          = CGC_CLOCK_PLL;
    clocks_cfg.pll_state             = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.pll_cfg.source_clock = CGC_CLOCK_MAIN_OSC; // unused
    clocks_cfg.pll_cfg.multiplier    = CGC_PLL_MUL_10_0;   // unused
```

```
    clocks_cfg.pll_cfg.divider        = CGC_PLL_DIV_2;      // unused
    clocks_cfg.divider_cfg.iclk_div = CGC_SYS_CLOCK_DIV_1;
    clocks_cfg.divider_cfg.pclka_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.pclkb_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.pclkc_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.pclkd_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.bclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.fclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.mainosc_state          = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.hoco_state             = CGC_CLOCK_CHANGE_START;
    clocks_cfg.moco_state             = CGC_CLOCK_CHANGE_STOP;
    clocks_cfg.loco_state             = CGC_CLOCK_CHANGE_NONE;
    err = R_CGC_ClocksCfg(&g_cgc0_ctrl, &clocks_cfg);
    assert(FSP_SUCCESS == err);
#if BSP_FEATURE_CGC_HAS_PLL
 /* Assuming the system clock is HOCO, switch to PLL running from main oscillator and
stop MOCO. */
    clocks_cfg.system_clock           = CGC_CLOCK_PLL;
    clocks_cfg.pll_state              = CGC_CLOCK_CHANGE_START;
    clocks_cfg.pll_cfg.source_clock = CGC_CLOCK_MAIN_OSC;
    clocks_cfg.pll_cfg.multiplier     = (cgc_pll_mul_t) BSP_CFG_PLL_MUL;
    clocks_cfg.pll_cfg.divider        = (cgc_pll_div_t) BSP_CFG_PLL_DIV;
    clocks_cfg.divider_cfg.iclk_div = CGC_SYS_CLOCK_DIV_1;
    clocks_cfg.divider_cfg.pclka_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.pclkb_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.pclkc_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.pclkd_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.bclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.divider_cfg.fclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.mainosc_state          = CGC_CLOCK_CHANGE_START;
    clocks_cfg.hoco_state             = CGC_CLOCK_CHANGE_STOP;
    clocks_cfg.moco_state             = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.loco_state             = CGC_CLOCK_CHANGE_NONE;
    err = R_CGC_ClocksCfg(&g_cgc0_ctrl, &clocks_cfg);
```

```
    assert(FSP_SUCCESS == err);
#endif
}
```

Oscillation Stop Detection

This example demonstrates registering a callback for oscillation stop detection of the main oscillator.

```
/* Example callback called when oscillation stop is detected. */
void oscillation_stop_callback (cgc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) If the MCU was running on the main oscillator, the MCU is now running
on MOCO. Switch clocks if
  * desired. This example shows switching to HOCO. */
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_HOCO, NULL);
    assert(FSP_SUCCESS == err);
 do
    {
 /* Wait for HOCO to stabilize. */
       err = R_CGC_ClockCheck(&g_cgc0_ctrl, CGC_CLOCK_HOCO);
    } while (FSP_SUCCESS != err);
 cgc_divider_cfg_t dividers =
    {
       .pclkb_div = CGC_SYS_CLOCK_DIV_4,
       .pclkd_div = CGC_SYS_CLOCK_DIV_4,
       .iclk_div = CGC_SYS_CLOCK_DIV_1,
       .pclka_div = CGC_SYS_CLOCK_DIV_4,
       .pclkc_div = CGC_SYS_CLOCK_DIV_4,
       .fclk_div = CGC_SYS_CLOCK_DIV_4,
       .bclk_div = CGC_SYS_CLOCK_DIV_4,
    };
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_HOCO, &dividers);
    assert(FSP_SUCCESS == err);
```

```
#if BSP_FEATURE_CGC_HAS_PLL
 /* (Optional) If the MCU was running on the PLL, the PLL is now in free-running
mode. Switch clocks if
  * desired. This example shows switching to the PLL running on HOCO. */
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_HOCO, NULL);
    assert(FSP_SUCCESS == err);
 do
    {
 /* Wait for HOCO to stabilize. */
       err = R_CGC_ClockCheck(&g_cgc0_ctrl, CGC_CLOCK_HOCO);
    } while (FSP_SUCCESS != err);
 cgc_pll_cfg_t pll_cfg =
    {
       .source_clock = CGC_CLOCK_HOCO,
       .multiplier   = (cgc_pll_mul_t) BSP_CFG_PLL_MUL,
       .divider      = (cgc_pll_div_t) BSP_CFG_PLL_DIV,
    };
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_PLL, &pll_cfg);
    assert(FSP_SUCCESS == err);
 do
    {
 /* Wait for PLL to stabilize. */
       err = R_CGC_ClockCheck(&g_cgc0_ctrl, CGC_CLOCK_PLL);
    } while (FSP_SUCCESS != err);
 cgc_divider_cfg_t pll_dividers =
    {
       .pclkb_div = CGC_SYS_CLOCK_DIV_4,
       .pclkd_div = CGC_SYS_CLOCK_DIV_4,
       .iclk_div = CGC_SYS_CLOCK_DIV_1,
       .pclka_div = CGC_SYS_CLOCK_DIV_4,
       .pclkc_div = CGC_SYS_CLOCK_DIV_4,
       .fclk_div = CGC_SYS_CLOCK_DIV_4,
       .bclk_div = CGC_SYS_CLOCK_DIV_4,
    };
```

```
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_PLL, &pll_dividers);
    assert(FSP_SUCCESS == err);
#endif
 /* (Optional) Clear the error flag. Only clear this flag after switching the MCU
clock source away from the main
  * oscillator and if the main oscillator is stable again. */
    err = R_CGC_OscStopStatusClear(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
}
void cgc_osc_stop_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the module. */
    err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable oscillation stop detection. The main oscillator must be running at this
point. */
    err = R_CGC_OscStopDetectEnable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
 /* (Optional) Oscillation stop detection must be disabled before entering any low
power mode. */
    err = R_CGC_OscStopDetectDisable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
    __WFI();
 /* (Optional) Reenable oscillation stop detection after waking from low power mode.
*/
    err = R_CGC_OscStopDetectEnable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	cgc_instance_ctrl_t

Data Structure Documentation

◆ cgc_instance_ctrl_t

struct cgc_instance_ctrl_t

CGC private control block. DO NOT MODIFY. Initialization occurs when R_CGC_Open() is called.

Data Fields

void const *	p_context

Field Documentation

◆ p_context

void const* cgc_instance_ctrl_t::p_context

Placeholder for user data. Passed to the user callback in cgc_callback_args_t.

Function Documentation

◆ R_CGC_Open()

fsp_err_t R_CGC_Open (cgc_ctrl_t *const *p_ctrl*, cgc_cfg_t const *const *p_cfg*)

Initialize the CGC API. Implements cgc_api_t::open.

Example:

```
/* Initializes the CGC module. */
   err = R_CGC_Open(&g_cgc0_ctrl, &g_cgc0_cfg);
```

Return values

FSP_SUCCESS	CGC successfully initialized.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_ALREADY_OPEN	Module is already open.

◆ R_CGC_ClocksCfg()

fsp_err_t R_CGC_ClocksCfg (cgc_ctrl_t *const *p_ctrl*, cgc_clocks_cfg_t const *const *p_clock_cfg*)

Reconfigures all main system clocks. This API can be used for any of the following purposes:

- start or stop clocks
- change the system clock source
- configure the PLL/PLL2 multiplication and division ratios when starting the PLL
- change the system dividers

If the requested system clock source has a stabilization flag, this function blocks waiting for the stabilization flag of the requested system clock source to be set. If the requested system clock source was just started and it has no stabilization flag, this function blocks for the stabilization time required by the requested system clock source according to the Electrical Characteristics section of the hardware manual. If the requested system clock source has no stabilization flag and it is already running, it is assumed to be stable and this function will not block. If the requested system clock is the subclock, the subclock must be stable prior to calling this function.

The internal dividers (cgc_clocks_cfg_t::divider_cfg) are subject to constraints described in footnotes of the hardware manual table detailing specifications for the clock generation circuit for the internal clocks for the MCU. For example:

- RA6M3: see footnotes of Table 9.2 "Specifications of the clock generation circuit for the internal clocks" in the RA6M3 manual R01UH0886EJ0100
- RA2A1: see footnotes of Table 9.2 "Clock generation circuit specifications for the internal clocks" in the RA2A1 manual R01UH0888EJ0100

Do not attempt to stop the requested clock source or the source of a PLL if the PLL will be running after this operation completes.

Implements cgc_api_t::clocksCfg.

Example:

```
/* Assuming the system clock is MOCO, switch to HOCO. */
cgc_clocks_cfg_t clocks_cfg;
   clocks_cfg.system_clock          = CGC_CLOCK_PLL;
   clocks_cfg.pll_state             = CGC_CLOCK_CHANGE_NONE;
   clocks_cfg.pll_cfg.source_clock = CGC_CLOCK_MAIN_OSC; // unused
   clocks_cfg.pll_cfg.multiplier    = CGC_PLL_MUL_10_0;   // unused
   clocks_cfg.pll_cfg.divider       = CGC_PLL_DIV_2;      // unused
   clocks_cfg.divider_cfg.iclk_div = CGC_SYS_CLOCK_DIV_1;
   clocks_cfg.divider_cfg.pclka_div = CGC_SYS_CLOCK_DIV_4;
   clocks_cfg.divider_cfg.pclkb_div = CGC_SYS_CLOCK_DIV_4;
   clocks_cfg.divider_cfg.pclkc_div = CGC_SYS_CLOCK_DIV_4;
   clocks_cfg.divider_cfg.pclkd_div = CGC_SYS_CLOCK_DIV_4;
   clocks_cfg.divider_cfg.bclk_div = CGC_SYS_CLOCK_DIV_4;
```

```
    clocks_cfg.divider_cfg.fclk_div = CGC_SYS_CLOCK_DIV_4;
    clocks_cfg.mainosc_state         = CGC_CLOCK_CHANGE_NONE;
    clocks_cfg.hoco_state            = CGC_CLOCK_CHANGE_START;
    clocks_cfg.moco_state            = CGC_CLOCK_CHANGE_STOP;
    clocks_cfg.loco_state            = CGC_CLOCK_CHANGE_NONE;
    err = R_CGC_ClocksCfg(&g_cgc0_ctrl, &clocks_cfg);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Clock configuration applied successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_IN_USE	Attempt to stop the current system clock or the PLL source clock.
FSP_ERR_CLOCK_ACTIVE	PLL configuration cannot be changed while PLL is running.
FSP_ERR_OSC_STOP_DET_ENABLED	PLL multiplier must be less than 20 if oscillation stop detect is enabled and the input frequency is less than 12.5 MHz.
FSP_ERR_NOT_STABILIZED	PLL clock source is not stable.
FSP_ERR_PLL_SRC_INACTIVE	PLL clock source is not running.
FSP_ERR_INVALID_STATE	The subclock must be running before activating HOCO with FLL.

◆ R_CGC_ClockStart()

fsp_err_t R_CGC_ClockStart (cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source*, cgc_pll_cfg_t const *const *p_pll_cfg*)

Start the specified clock if it is not currently active. The PLL configuration cannot be changed while the PLL is running. Implements cgc_api_t::clockStart.

The PLL source clock must be operating and stable prior to starting the PLL.

Example:

```
/* Start the LOCO. */
    err = R_CGC_ClockStart(&g_cgc0_ctrl, CGC_CLOCK_LOCO, NULL);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Clock initialized successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_STABILIZED	The clock source is not stabilized after being turned off or PLL clock source is not stable.
FSP_ERR_PLL_SRC_INACTIVE	PLL clock source is not running.
FSP_ERR_CLOCK_ACTIVE	PLL configuration cannot be changed while PLL is running.
FSP_ERR_OSC_STOP_DET_ENABLED	PLL multiplier must be less than 20 if oscillation stop detect is enabled and the input frequency is less than 12.5 MHz.
FSP_ERR_INVALID_STATE	The subclock must be running before activating HOCO with FLL.

◆ R_CGC_ClockStop()

fsp_err_t R_CGC_ClockStop (cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source*)

Stop the specified clock if it is active. Implements cgc_api_t::clockStop.

Do not attempt to stop the current system clock source. Do not attempt to stop the source clock of a PLL if the PLL is running.

Return values

FSP_SUCCESS	Clock stopped successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_IN_USE	Attempt to stop the current system clock or the PLL source clock.
FSP_ERR_OSC_STOP_DET_ENABLED	Attempt to stop MOCO when Oscillation stop is enabled.
FSP_ERR_NOT_STABILIZED	Clock not stabilized after starting.

◆ R_CGC_ClockCheck()

fsp_err_t R_CGC_ClockCheck (cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source*)

Check the specified clock for stability. Implements cgc_api_t::clockCheck.

Return values

FSP_SUCCESS	Clock is running and stable.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_STABILIZED	Clock not stabilized.
FSP_ERR_CLOCK_INACTIVE	Clock not turned on.

◆ R_CGC_SystemClockSet()

fsp_err_t R_CGC_SystemClockSet (cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *clock_source*, cgc_divider_cfg_t const *const *p_divider_cfg*)

Set the specified clock as the system clock and configure the internal dividers for ICLK, PCLKA, PCLKB, PCLKC, PCLKD, BCLK, and FCLK. Implements cgc_api_t::systemClockSet.

The requested clock source must be running and stable prior to calling this function. The internal dividers are subject to constraints described in the hardware manual table "Specifications of the Clock Generation Circuit for the internal clocks".

The internal dividers (p_divider_cfg) are subject to constraints described in footnotes of the hardware manual table detailing specifications for the clock generation circuit for the internal clocks for the MCU. For example:

- RA6M3: see footnotes of Table 9.2 "Specifications of the clock generation circuit for the internal clocks" in the RA6M3 manual R01UH0886EJ0100
- RA2A1: see footnotes of Table 9.2 "Clock generation circuit specifications for the internal clocks" in the RA2A1 manual R01UH0888EJ0100

This function also updates the RAM and ROM wait states, the operating power control mode, and the SystemCoreClock CMSIS global variable.

Example:

```
 /* Set divisors. Divisors for clocks that don't exist on the MCU are ignored. */
 cgc_divider_cfg_t dividers =
    {
 /* PCLKB is not used in this application, so select the maximum divisor for lowest
power. */
       .pclkb_div = CGC_SYS_CLOCK_DIV_64,
 /* PCLKD is not used in this application, so select the maximum divisor for lowest
power. */
       .pclkd_div = CGC_SYS_CLOCK_DIV_64,
 /* ICLK is the MCU clock, allow it to run as fast as the LOCO is capable. */
       .iclk_div = CGC_SYS_CLOCK_DIV_1,
 /* These clocks do not exist on some devices. If any clocks don't exist, set the
divider to 1. */
       .pclka_div = CGC_SYS_CLOCK_DIV_1,
       .pclkc_div = CGC_SYS_CLOCK_DIV_1,
       .fclk_div = CGC_SYS_CLOCK_DIV_1,
       .bclk_div = CGC_SYS_CLOCK_DIV_1,
    };
 /* Switch the system clock to LOCO. */
```

```
    err = R_CGC_SystemClockSet(&g_cgc0_ctrl, CGC_CLOCK_LOCO, &dividers);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation performed successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CLOCK_INACTIVE	The specified clock source is inactive.
FSP_ERR_NOT_STABILIZED	The clock source has not stabilized

◆ R_CGC_SystemClockGet()

fsp_err_t R_CGC_SystemClockGet (cgc_ctrl_t *const *p_ctrl*, cgc_clock_t *const *p_clock_source*, cgc_divider_cfg_t *const *p_divider_cfg*)

Return the current system clock source and configuration. Implements cgc_api_t::systemClockGet.

Return values

FSP_SUCCESS	Parameters returned successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.

◆ R_CGC_OscStopDetectEnable()

fsp_err_t R_CGC_OscStopDetectEnable (cgc_ctrl_t *const *p_ctrl*)

Enable the oscillation stop detection for the main clock. Implements cgc_api_t::oscStopDetectEnable.

The MCU will automatically switch the system clock to MOCO when a stop is detected if Main Clock is the system clock. If the system clock is the PLL, then the clock source will not be changed and the PLL free running frequency will be the system clock frequency.

Example:

```
 /* Enable oscillation stop detection. The main oscillator must be running at this
point. */
    err = R_CGC_OscStopDetectEnable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation performed successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_LOW_VOLTAGE_MODE	Settings not allowed in low voltage mode.

◆ R_CGC_OscStopDetectDisable()

fsp_err_t R_CGC_OscStopDetectDisable (cgc_ctrl_t *const *p_ctrl*)

Disable the oscillation stop detection for the main clock. Implements cgc_api_t::oscStopDetectDisable.

Example:

```
 /* (Optional) Oscillation stop detection must be disabled before entering any low
power mode. */
    err = R_CGC_OscStopDetectDisable(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
    __WFI();
```

Return values

FSP_SUCCESS	Operation performed successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_OSC_STOP_DETECTED	The Oscillation stop detect status flag is set. Under this condition it is not possible to disable the Oscillation stop detection function.

◆ R_CGC_OscStopStatusClear()

fsp_err_t R_CGC_OscStopStatusClear (cgc_ctrl_t *const *p_ctrl*)

Clear the Oscillation Stop Detection Status register. This register is not cleared automatically if the stopped clock is restarted. Implements cgc_api_t::oscStopStatusClear.

After clearing the status, oscillation stop detection is no longer enabled.

This register cannot be cleared while the main oscillator is the system clock or the PLL source clock.

Example:

```
 /* (Optional) Clear the error flag. Only clear this flag after switching the MCU
clock source away from the main
  * oscillator and if the main oscillator is stable again. */
    err = R_CGC_OscStopStatusClear(&g_cgc0_ctrl);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Operation performed successfully.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CLOCK_INACTIVE	Main oscillator must be running to clear the oscillation stop detection flag.
FSP_ERR_OSC_STOP_CLOCK_ACTIVE	The Oscillation Detect Status flag cannot be cleared if the Main Osc or PLL is set as the system clock. Change the system clock before attempting to clear this bit.
FSP_ERR_INVALID_HW_CONDITION	Oscillation stop status was not cleared. Check preconditions and try again.

◆ R_CGC_CallbackSet()

fsp_err_t R_CGC_CallbackSet (cgc_ctrl_t *const *p_api_ctrl*, void(*)(cgc_callback_args_t *) *p_callback*, void const *const *p_context*, cgc_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements cgc_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_CGC_Close()

fsp_err_t R_CGC_Close (cgc_ctrl_t *const *p_ctrl*)

Closes the CGC module. Implements cgc_api_t::close.

Return values

FSP_SUCCESS	The module is successfully closed.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Module is not open.

4.2.14 Cyclic Redundancy Check (CRC) Calculator (r_crc)

Modules

Functions

fsp_err_t	R_CRC_Open (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)
fsp_err_t	R_CRC_Close (crc_ctrl_t *const p_ctrl)
fsp_err_t	R_CRC_Calculate (crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *calculatedValue)
fsp_err_t	R_CRC_CalculatedValueGet (crc_ctrl_t *const p_ctrl, uint32_t *calculatedValue)

fsp_err_t	R_CRC_SnoopEnable (crc_ctrl_t *const p_ctrl, uint32_t crc_seed)
fsp_err_t	R_CRC_SnoopDisable (crc_ctrl_t *const p_ctrl)

Detailed Description

Driver for the CRC peripheral on RA MCUs. This module implements the CRC Interface.

Overview

The CRC module provides a API to calculate 8, 16 and 32-bit CRC values on a block of data in memory or a stream of data over a Serial Communication Interface (SCI) channel using industry-standard polynomials.

Features

- CRC module supports the following 8 and 16 bit CRC polynomials which operates on 8-bit data in parallel
  - X^8+X^2+X+1 (CRC-8)
  - X^16+X^15+X^2+1 (CRC-16)
  - X^16+X^12+X^5+1 (CRC-CCITT)
- CRC module supports the following 32 bit CRC polynomials which operates on 32-bit data in parallel
  - X^32+X^26+X^23+X^22+X^16+X^12+X^11+X^10+X^8+X^7+X^5+X^4+X^2+X+ 1 (CRC-32)
  - X^32+ X^28+ X^27+ X^26+ X^25+ X^23+ X^22+ X^20+ X^19+ X^18+X^14+X^13+X^11+X^10+X^9+X^8+X^6+1 (CRC-32C)
- CRC module can calculate CRC with LSB first or MSB first bit order.

Configuration

Build Time Configurations for r_crc

The following build time configurations are defined in fsp_cfg/r_crc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Monitoring > CRC (r_crc)

This module can be added to the Stacks tab via New Stack > Monitoring > CRC (r_crc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description

Name	Name must be a valid C symbol	g_crc0	Module name.
CRC Polynomial	• CRC-8 • CRC-16 • CRC-CCITT • CRC-32 • CRC-32C	CRC-32C	Select the CRC polynomial.
Bit Order	• LSB • MSB	MSB	Select the CRC bit order.
Snoop Address	Refer to the RA Configuration tool for available options.	NONE	Select the SCI register address CRC snoop

Clock Configuration

There is no clock configuration for the CRC module.

Pin Configuration

This module does not use I/O pins.

Usage Notes

CRC Snoop

The CRC snoop function monitors reads from and writes to a specified I/O register address and performs CRC calculation on the data read from and written to the register address automatically. Instead of calling R_CRC_Calculate on a block of data, R_CRC_SnoopEnable is called to start monitoring reads/writes and R_CRC_CalculatedValueGet is used to obtain the current CRC.

Note

Snoop mode is available for transmit/receive operations on SCI only.

Limitations

When using CRC32 polynomial functions the CRC module produces the same results as popular online CRC32 calculators, but it is important to remember a few important points.

- Online CRC32 calculators allow the input to be any number of bytes. The FSP CRC32 API function uses 32-bit words. This means the online calculations must be 'padded' to end on a 32-bit boundary.
- Online CRC32 calculators usually invert the output prior to presenting it as a result. It is up to the application program to include this step if needed.
- The seed value of 0xFFFFFFFF needs to be used by both the online calculator and the R_CRC module API (CRC32 polynomials)
- Make sure the bit orientation of the R_CRC CRC32 is set for LSB and that you have CRC32 selected and not CRC32C.
- Some online CRC tools XOR the final result with 0xFFFFFFFF.

Examples

Basic Example

This is a basic example of minimal use of the CRC module in an application.

```
void crc_example ()
{
    uint32_t length;
    uint32_t uint8_calculated_value;
    length = sizeof(g_data_8bit) / sizeof(g_data_8bit[0]);
 crc_input_t example_input =
    {
       .p_input_buffer = g_data_8bit,
       .num_bytes      = length,
       .crc_seed       = 0,
    };
 /* Open CRC module with 8 bit polynomial */
 R_CRC_Open(&crc_ctrl, &g_crc_test_cfg);
 /* 8-bit CRC calculation */
 R_CRC_Calculate(&crc_ctrl, &example_input, &uint8_calculated_value);
}
```

Snoop Example

This example demonstrates CRC snoop operation.

```
void crc_snoop_example ()
{
 /* Open CRC module with 8 bit polynomial */
 R_CRC_Open(&crc_ctrl, &g_crc_test_cfg);
 /* Open SCI Driver */
 /* Configure Snoop address and enable snoop mode */
 R_CRC_SnoopEnable(&crc_ctrl, 0);
 /* Perfrom SCI read/Write operation depending on the SCI snoop address configure */
 /* Read CRC value */
 R_CRC_CalculatedValueGet(&crc_ctrl, &g_crc_buff);
}
```

Data Structures

struct	crc_instance_ctrl_t

Data Structure Documentation

◆ crc_instance_ctrl_t

struct crc_instance_ctrl_t
Driver instance control structure.

Function Documentation

◆ R_CRC_Open()

fsp_err_t R_CRC_Open (crc_ctrl_t *const *p_ctrl*, crc_cfg_t const *const *p_cfg*)

Open the CRC driver module

Implements crc_api_t::open

Open the CRC driver module and initialize the driver control block according to the passed-in configuration structure.

Return values

FSP_SUCCESS	Configuration was successful.
FSP_ERR_ASSERTION	p_ctrl or p_cfg is NULL.
FSP_ERR_ALREADY_OPEN	Module already open

◆ R_CRC_Close()

fsp_err_t R_CRC_Close (crc_ctrl_t *const *p_ctrl*)

Close the CRC module driver.

Implements crc_api_t::close

Return values

FSP_SUCCESS	Configuration was successful.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	The driver is not opened.

◆ R_CRC_Calculate()

fsp_err_t R_CRC_Calculate (crc_ctrl_t *const *p_ctrl*, crc_input_t *const *p_crc_input*, uint32_t * *calculatedValue*)

Perform a CRC calculation on a block of 8-bit/32-bit (for 32-bit polynomial) data.

Implements crc_api_t::calculate

This function performs a CRC calculation on an array of 8-bit/32-bit (for 32-bit polynomial) values and returns an 8-bit/32-bit (for 32-bit polynomial) calculated value

Return values

FSP_SUCCESS	Calculation successful.
FSP_ERR_ASSERTION	Either p_ctrl, inputBuffer, or calculatedValue is NULL.
FSP_ERR_INVALID_ARGUMENT	length value is NULL.
FSP_ERR_NOT_OPEN	The driver is not opened.

◆ R_CRC_CalculatedValueGet()

fsp_err_t R_CRC_CalculatedValueGet (crc_ctrl_t *const *p_ctrl*, uint32_t * *calculatedValue*)

Return the current calculated value.

Implements crc_api_t::crcResultGet

CRC calculation operates on a running value. This function returns the current calculated value.

Return values

FSP_SUCCESS	Return of calculated value successful.
FSP_ERR_ASSERTION	Either p_ctrl or calculatedValue is NULL.
FSP_ERR_NOT_OPEN	The driver is not opened.

◆ R_CRC_SnoopEnable()

fsp_err_t R_CRC_SnoopEnable (crc_ctrl_t *const *p_ctrl*, uint32_t *crc_seed*)

Configure the snoop channel and set the CRC seed.

Implements crc_api_t::snoopEnable

The CRC calculator can operate on reads and writes over any of the first ten SCI channels. For example, if set to channel 0, transmit, every byte written out SCI channel 0 is also sent to the CRC calculator as if the value was explicitly written directly to the CRC calculator.

Return values

FSP_SUCCESS	Snoop configured successfully.
FSP_ERR_ASSERTION	Pointer to control stucture is NULL
FSP_ERR_NOT_OPEN	The driver is not opened.

◆ R_CRC_SnoopDisable()

fsp_err_t R_CRC_SnoopDisable (crc_ctrl_t *const *p_ctrl*)

Disable snooping.

Implements crc_api_t::snoopDisable

Return values

FSP_SUCCESS	Snoop disabled.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	The driver is not opened.

4.2.15 Capacitive Touch Sensing Unit (r_ctsu)

Modules

Functions

fsp_err_t	R_CTSU_Open (ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg) Opens and configures the CTSU driver module. Implements ctsu_api_t::open. More...
fsp_err_t	R_CTSU_ScanStart (ctsu_ctrl_t *const p_ctrl)

	This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with R_CTSU_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements ctsu_api_t::scanStart. More...
fsp_err_t	R_CTSU_DataGet (ctsu_ctrl_t *const p_ctrl, uint16_t *p_data) This function gets the sensor values as scanned by the CTSU. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements ctsu_api_t::dataGet. More...
fsp_err_t	R_CTSU_ScanStop (ctsu_ctrl_t *const p_ctrl) This function scan stops the sensor as scanning by the CTSU. Implements ctsu_api_t::scanStop. More...
fsp_err_t	R_CTSU_CallbackSet (ctsu_ctrl_t *const p_api_ctrl, void(*p_callback)(ctsu_callback_args_t *), void const *const p_context, ctsu_callback_args_t *const p_callback_memory)
fsp_err_t	R_CTSU_Diagnosis (ctsu_ctrl_t *const p_ctrl) Diagnosis the CTSU peripheral. Implements ctsu_api_t::diagnosis. More...
fsp_err_t	R_CTSU_Close (ctsu_ctrl_t *const p_ctrl) Disables specified CTSU control block. Implements ctsu_api_t::close. More...
fsp_err_t	R_CTSU_SpecificDataGet (ctsu_ctrl_t *const p_ctrl, uint16_t *p_specific_data, ctsu_specific_data_type_t specific_data_type) This function gets the sensor specific data values as scanned by the CTSU. Call this function after calling the R_CTSU_DataGet() function. More...
fsp_err_t	R_CTSU_DataInsert (ctsu_ctrl_t *const p_ctrl, uint16_t *p_insert_data) This function inserts the value of the second argument as the measurement result value. Call this function after calling the R_CTSU_DataInsert() function. Implements ctsu_api_t::dataInsert. More...

Detailed Description

This HAL driver supports the Capacitive Touch Sensing Unit (CTSU). It implements the CTSU Interface
.

Overview

The capacitive touch sensing unit HAL driver (r_ctsu) provides an API to control the CTSU peripheral. This module performs capacitance measurement based on various settings defined by the configuration. This module is configured via the QE for Capacitive Touch.

Features

- Supports multiple scan modes
  - Self-capacitance multi scan mode (CTSU2 support active shield)
  - Mutual-capacitance full scan mode
  - Mutual-capacitance parallel scan mode (CTSU2)
  - Current Measurement mode (CTSU2)
  - Diagnosis scan mode
- Scans may be started by software or an external trigger
- Returns measured capacitance data on scan completion
- Support DTC transfer of scanned data
- Supports TrustZone
- Corrects accuracy for temperature drift (CTSU2)

Configuration

Note

> ***This module is configured via the QE for Capacitive Touch. For information on how to use the QE tool, once the tool is installed click Help -> Help Contents in e2 studio and search for "QE".***

Build Time Configurations for r_ctsu

The following build time configurations are defined in fsp_cfg/r_ctsu_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Support for using DTC	• Enabled • Disabled	Disabled	Enable DTC support for the CTSU module.
Interrupt priority level	MCU Specific Options		Priority level of all CTSU interrupt (CSTU_ WR,CTSU_RD,CTSU_FN)

Configurations for CapTouch > CTSU (r_ctsu)

This module can be added to the Stacks tab via New Stack > CapTouch > CTSU (r_ctsu). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description

Scan Start Trigger	MCU Specific Options	CTSU Scan Start Trigger Select

Interrupt Configuration

The first R_CTSU_Open function call sets CTSU peripheral interrupts. The user should provide a callback function to be invoked at the end of the CTSU scan sequence. The callback argument will contain information about the scan status.

Clock Configuration

The CTSU peripheral module uses PCLKB as its clock source. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

Note

The CTSU Drive pulse will be calculated and set by the tooling depending on the selected transfer rate.

Pin Configuration

The TSn pins are sensor pins for the CTSU.

The TSCAP pin is used for an internal low-pass filter and must be connected to an external decoupling capacitor.

Usage Notes

The CTSU module is a CTSU driver for the Touch module. The CTSU module assumes the access from the Touch middleware layer, and it is also accessible from an user application.
CTSU and CTSU2 are functionally different, so CTSU and CTSU2 are described in this application note as below.
Common description for CTSU and CTSU2 -> CTSU
Description only for CTSU -> CTSU1
Description only for CTSU2 -> CTSU2
Without mention, it means the common description for CTSU and CTSU2.

Functions

The CTSU module supports the following functions.

Measurements and Obtaining Data

Measurements can be started by a software trigger or by an external event triggered by the Event Link Controller (ELC).
As the measurement process is carried out by the CTSU2 peripheral, it does not use up main processor processing time.
The CTSU module processes INTCTSUWR and INTCTSURD if generated during a measurement. The data transfer controller (DTC) can also be used for these processes.
When the measurement complete interrupt (INTCTSUFN) process is complete, the application is notified in a callback function. Make sure you obtain the measurement results before the next measurement is started as internal processes are also executed when a measurement is completed.
Start the measurement with API function R_CTSU_ScanStart().
Obtain the measurement results with API function R_CTSU_DataGet().

Sensor ICO Correction function

The CTSU2 peripheral has a built-in correction circuit to handle the potential microvariations related to the manufacturing process of the sensor ICO MCU.
The module temporarily transitions to the correction process during initialization after power is turned on. In the correction process, the correction circuit is used to generate a correction coefficient (factor) to ensure accurate sensor measurement values.
When temperature correction is enabled, an external resistor connected to a TS terminal is used to periodically update the correction coefficient. By using an external resistor that is not dependent on temperature, you can even correct the temperature drift of the sensor ICO.

Initial Offset Adjustment

The CTSU2 peripheral was designed with a built-in offset current circuit in consideration of the amount of change in current due to touch. The offset current circuit cancels enough of the parasitic capacitance for it to fit within the sensor ICO dynamic range.
This module automatically adjusts the offset current setting. As the adjustment uses the normal measurement process, R_CTSU_ScanStart() and R_CTSU_DataGet() must be repeated several times after startup. Because the ctsu_element_cfg_t member “so” is the starting point for adjustments, you can set the appropriate value for “so” in order to reduce the number of times the two functions must be run to complete the adjustment. Normally, the value used for “so” is a value adjusted by QE for Capacitive Touch.

Mode	Default target value
Self-capacitance	15360 (37.5%)
Self-capacitance using active shield	6144 (15%)
Mutual-capacitance	10240 (20%)

The percentage is for the CCO's input limit. 100% is the measured value 40960.
The default target value is based on 526us(CTSU1) or 256us(CTSU2).
When the measurement time is changed, the target value is adjusted by the ratio with the base time.

Example of target value in combination of CTSUSNUM and CTSUSDPA.
CTSU1 (CTSU clock = 32MHz, Self-capacitance mode)

Target value	CTSUSNUM	CTSUSDPA	Measurement time
15360	0x3	0x7	526usec
30720	0x7	0x7	1052usec
30720	0x3	0xF	1052usec
7680	0x1	0x7	263usec
7680	0x3	0x3	263usec

The measurement time changes depending on the combination of CTSUSNUM and CTSUSDPA.
In the above table, CTSUPRRTIO is the recommended value of 3, and CSTUPRMODE is the recommended value of 2.
When changing CTSUPRRATIO and CTSUPRMODE from the recommended values, follow the Hardware Manual for the measurement time.

CTSU2 (Self-capacitance mode)

Target value	Target value (multi frequency)	CTSUSNUM	Measurement time
7680	15360 (128us + 128us)	0x7	128usec
15360	30720 (256us + 256us)	0xF	256usec
3840	7680 (64us + 64us)	0x3	64usec

The measurement time changes depending on CTSUSNUM.
If STCLK cannot be set to 0.5MHz, it will not support the table above.
When setting STCLK to other than 0.5MHz because the CTSU clock is not an integer, follow the hardware manual for the measurement time.

Random Pulse Frequency Measurement (CTSU1)

The CTSU1 peripheral measures at one drive frequency.
The drive frequency determines the amperage to the electrode and generally uses the value tuned with QE for Capacitive Touch.
The drive frequency is calculated as below.
It is determined by PCLK frequency input to CTSU, CTSU Count Source Select bit(CTSUCLK), and CTSU Sensor Drive pulse Division Control bit(CTSUSDPA). For example, If it is set PCLK =32MHz, CTSUCLK = PLCK/2, and CTSUSDPA = 1/16, then drive frequency is 0.5MHz. CTSUSDPA can change for each TS port.

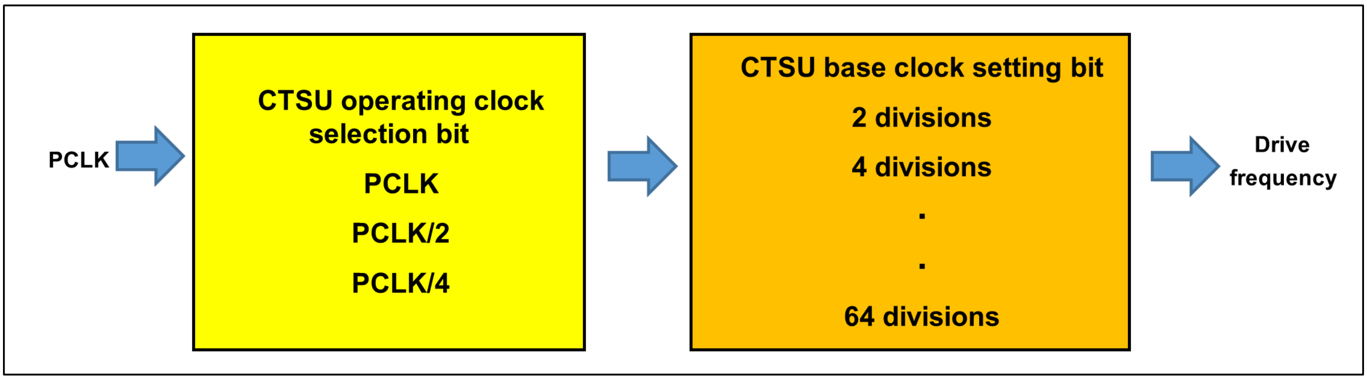


Figure 149: Drive Frequency Settings

The actual drive pulse is phase-shifted and frequency-spread with respect to the clock based on the drive frequency as a measure against external environmental noise. This module is fixed at initialization and sets the following.
CTSUSOFF = 0,CTSUSSMOD = 0,CTSUSSCNT = 3

Multi-frequency Measurements (CTSU2)

The CTSU2 peripheral can measure in one of four drive frequencies to avoid synchronous noise.
With the default settings, the module takes measurements at three different frequencies. After standardizing the results obtained at the three frequencies in accordance with the first frequency reference value, the measured value is determined based on majority in a process referred to as "normalization."
The three values standardized to the first frequency reference value are called correction data.

You can get the three correction data with R_CTSU_SpecificDataGet().

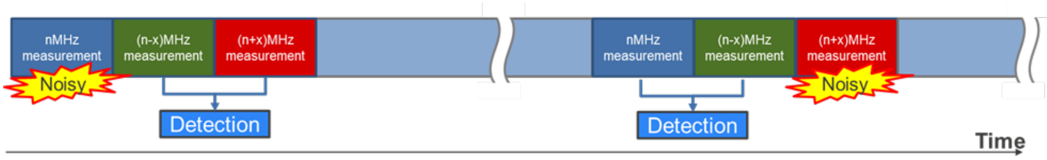


Figure 150: Multi-frequency Measurements

Drive frequency is determined based on the config settings. The module sets registers according to the config settings, and sets the three drive frequencies.
Drive frequency is calculated in the following equation:
(PCLKB frequency / CLK / STCLK) x SUMULTIn / 2 / SDPA : n = 0, 1, 2
The figure below shows the settings for generating a 2MHz drive frequency when the PCLKB frequency is 32 MHz. SDPA can be set for each touch interface configuration.

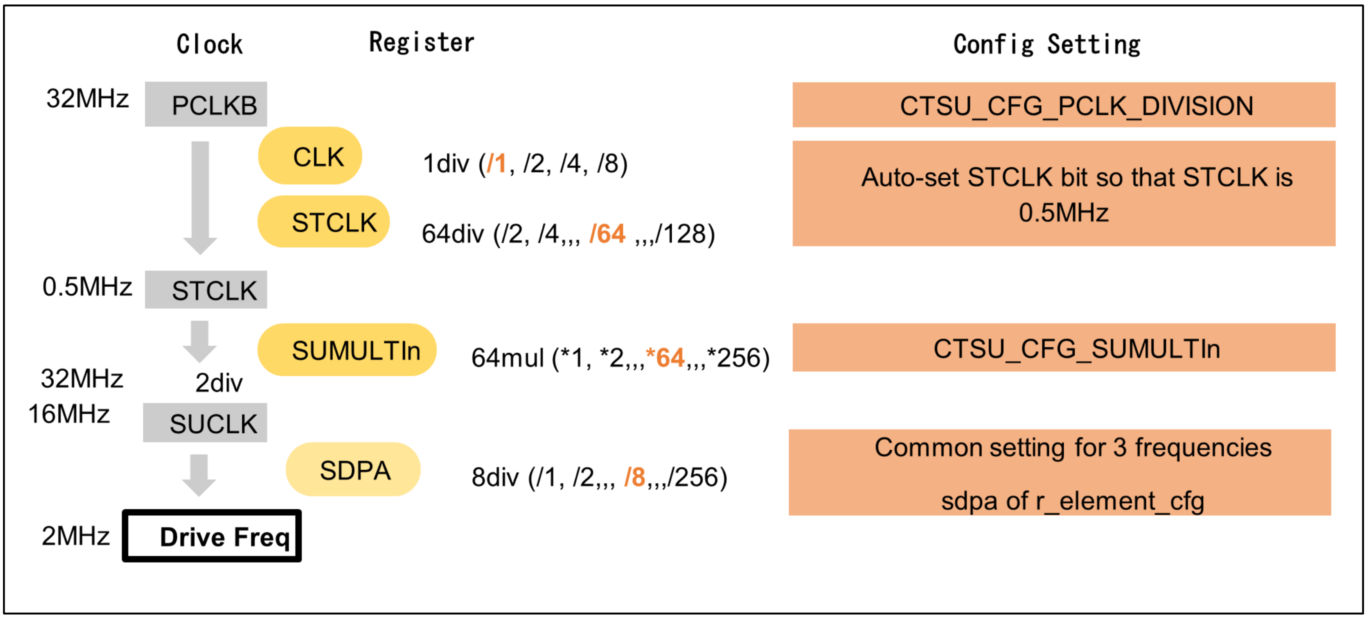


Figure 151: Drive Frequency Settings

Shield Function(CTSU2)

The CTSU2 peripheral has a built-in function that outputs a shield signal in phase with the drive pulse from the shield terminal and the non-measurement terminal in order to shield against external influences while suppressing any increase in parasitic capacitance. This function can only be used during self-capacitance measurements.
This module allows the user to set a shield for each touch interface configuration.
For example, for the electrode configuration shown in , the members of ctsu_cfg_t should be set as follows. Other members have been omitted for the example.
.txvsel = CTSU_TXVSEL_INTERNAL_POWER,
.txvsel2 = CTSU_TXVSEL_MODE,
.md = CTSU_MODE_SELF_MULTI_SCAN,
.posel = CTSU_POSEL_SAME_PULSE,
.ctsuchac0 = 0x0F,
.ctsuchtrc0 = 0x08,

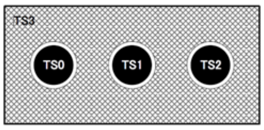


Figure 152: Example of Shield Electrode Structure

Measurement Error Message

When the CTSU2 peripheral detects an abnormal measurement, it sets the status register bit to 1. In the measurement complete interrupt process, the module reads ICOMP1, ICOMP0, and SENSOVF of the status register and notifies the results in the callback function. The status register is reset after the contents are read. For more details on abnormal measurements, refer to "member event" in the ctsu_callback_args_t callback function argument.

Moving Average

This function calculates the moving average of the measured results.
Set the number of times the moving average should be calculated in the config settings.

Diagnosis Function

The CTSU peripheral has a built-in function that diagnoses its own inner circuit. This diagnosis function provides the API for diagnosing the inner circuit.
The diagnostic requirements are different for CTSU1 and CTSU2 providing 5 types of diagnosis for CTSU1 and 9 types for CTSU2.
The diagnosis function is executed by calling the API function. This is executed independently from the other measurements and does not affect them.
To enable the diagnosis function, set CTSU_CFG_DIAG_SUPPORT_ENABLE to 1.
For CTSU1, a 27pF condenser should be connected externally.
For CTSU2, use Analog to Digital Converter (r_adc).

Measurement Mode

This module supports all three modes offered by the CTSU2 peripheral: self-capacitance, mutual-capacitance, and current measurement modes. The temperature correction mode is also offered as a mode for updating the correction coefficient.

Self-capacitance Mode

The self-capacitance mode is used to measure the capacitance of each terminal (TS).
The CTSU2 peripheral measures the terminals in ascending order according to the TS numbers, then stores the data. For example, even if you want to use TS5, TS8, TS2, TS3 and TS6 in your application in that order, they will still be measured and stored in the order of TS2, TS3, TS5, TS6, and TS8. Therefore, you will need to reference buffer indexes [2], [4], [0], [1], and [3].

[CTSU1]
In default settings, the measurement period for each TS is wait-time plus approximately 526us.

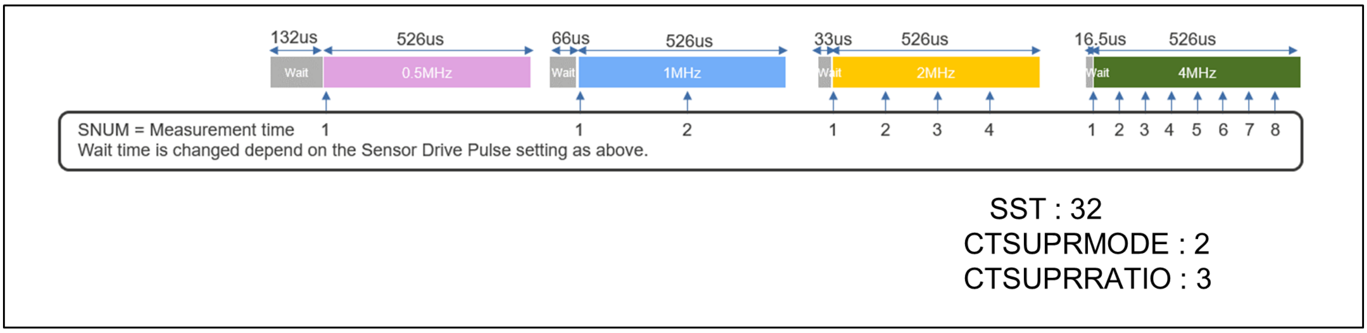


Figure 153: Self-capacitance Measurement Period (CTSU1)

[CTSU2]
In default settings, the measurement period for each TS is approximately 576us.

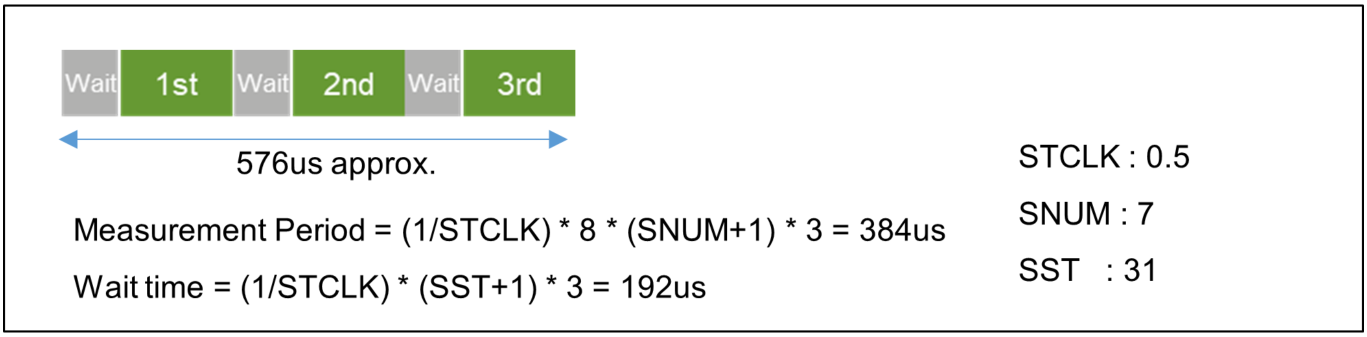


Figure 154: Self-capacitance Measurement Period (CTSU2)

Mutual-Capacitance Mode

The mutual-capacitance mode is used to measure the capacitance generated between the receive TS (Rx) and transmit TS (Tx), and therefore requires at least two terminals.
The CTSU2 peripheral measures all specified combinations of Rx and Tx. For example, when Rx is TS1 and TS3, and Tx is TS2, TS7 and TS4, the combinations are measured in the following order and the data is stored.
TS3-TS2, TS3-TS4, TS3-TS7, TS10-TS2, TS10-TS4, TS10-TS7
To measure the mutual-capacitance generated between electrodes, the CTSU2 peripheral performs the measurement process on the same electrode twice.
The mutual-capacitance is obtained by inverting the phase relationship of the pulse output and switched capacitor in the primary and secondary measurements, and calculating the difference between the two measurements. This module does not calculate the difference, but outputs the secondary measured result.
[CTSU1]
In default settings, the measurement period for each TS is twice of wait-time plus approximately 526us.
[CTSU2]
In default settings, the measurement period for each TS is approximately 1152us.

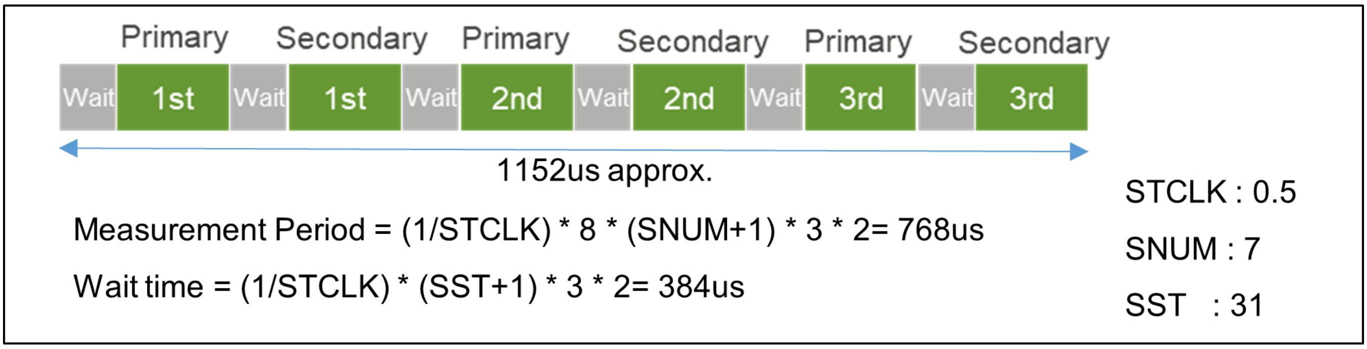


Figure 155: Mutual-capacitance Measurement Period

Mutual-capacitance parallel scan mode(CTSU2)

This mode provides fast measurement time by parallel scanning the RX lines with a CFC circuit. Operation is otherwise identical to normal CTSU mutual scanning.

- Scan Order
  - The hardware scans all RX pins simultaneously for each TX pin.
  - For example, if sensors TS10, TS11, and TS03 are specified as RX sensors, and sensors TS02, TS07, and TS04 are specified as TX sensors, the hardware will scan them in the following sensor-pair order:
    TS02-(TS03, TS10, TS11), TS04-(TS03, TS10, TS11), TS07-(TS03, TS10, TS11)
- Element
  - An element refers to the index of a sensor-pair within the scan order. Using the previous example, TS07-TS10 is element 7.
- Scan Time
  - Because the RX lines are scanned in parallel, CFC mutual-capacitance scan is the same amount of times faster than a basic mutual matrix scan as the number of RX lines. In other words, on a matrix with N receive lines, CFC mutual scanning is N times faster than basic mutual scanning. Set CTSU_MODE_MUTUAL_CFC_SCAN to "md" of ctsu_cfg_t.
    Also, add the number of matrix used for this measurement to CTSU_CFG_NUM_MUTUAL_ELEMENTS. In addition, set the number of CTSU_CFG_NUM_CFC and CTSU_CFG_NUM_CFC_TX.
    For details, refer to the configuration and sample application output by QE for Capacitive Touch.

Current Measurement Mode(CTSU2)

The current measurement mode is used to measure the minute current input to the TS terminal. The order of measurement and data storage is the same as that of the self-capacitance mode. As this does not involve the switched capacitor operation, the measurement is only performed once. The measurement period for one TS under default settings is approximately 256us. The current measurement mode requires a longer stable wait time than the other modes, so the SST is set to 63.

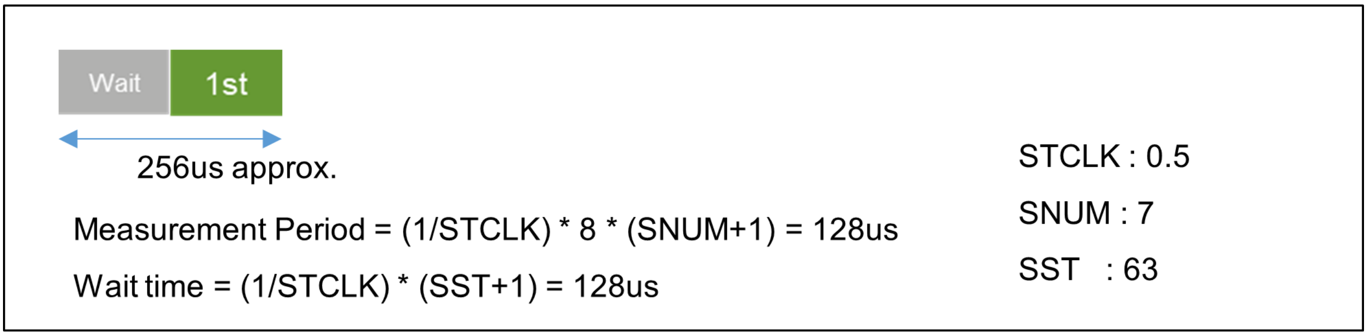


Figure 156: Current Measurement Period

Temperature Correction Mode(CTSU2)

The temperature correction mode is used to periodically update the correction coefficient using an external resistor connected to a TS terminal. This involves three processes as described below. Also refer to the timing chart in Figure of Temperature Correction Measurement Timing Chart.

1. Measure the correction circuit. One set comprises twelve measurements.

2. Measure the current when TSCAP voltage is applied to the external resistor to create a correction coefficient based on an external resistor that does not depend on temperature. Execute the next measurement after the previous measurement set is completed (as described in step 1).
3. Flow offset current to the external resistor and measure the voltage with the ADC. This will adjust the RTRIM register and handle the temperature drift of the internal reference resistor. In the config settings, set the number of times step 2 should be executed before carrying out this measurement.

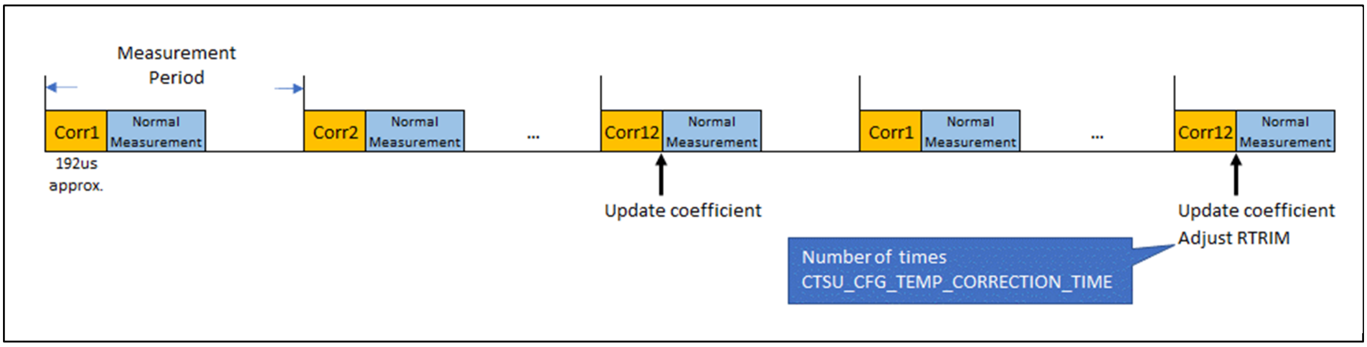


Figure 157: Temperature Correction Measurement Timing Chart

Diagnosis Mode

The diagnosis mode is a mode in which various internal measurement values are scanned by using this diagnosis function R_CTSU_Diagnosis().

Measurement Timing

Measurements are initiated by a software trigger or an external event which is triggered by the Event Link Controller (ELC).

The most common method is using a timer to carry out periodic measurements. Make sure to set the timer interval to allow the measurement and internal value update processes to complete before the next measurement period. The measurement period differs according to touch interface configuration and measurement mode.
The execution timing of software triggers and external triggers differ slightly.
Since a software trigger sets the start flag after setting the touch interface configuration with R_CTSU_ScanStart(), there is a slight delay after the timer event occurrence. However, as the delay is much smaller than the measurement period, a software trigger is recommended for most instances as it is easy to set.
An external trigger is recommended for applications in which this slight delay is not acceptable or that require low-power consumption operations. When using an external trigger with multiple touch interface configurations, use R_CTSU_ScanStart() to set another touch interface configuration after one measurement is completed.

TrustZone Support

In r_ctsu and rm_touch module, Non-Secure Callable Guard Functions are only generated from QE for Capacitive Touch. QE can be used for tuning in secure or flat project, but not in non-secure project. If you want to use in non-secure project, copy the output file from secure or flat project. Refer to QE Help for more information.

Data flow

The flow of storing data in RAM is as follows.

(CTSU1)

1. Read registers and stored in RAM as raw data.
2. ICO correction calculation of raw data and stored in RAM as correction data.
3. The correction data is calculated by moving average and stored in RAM as measurement results.

(CTSU2)

1. Reads a register and stores raw data measured at three different frequencies in RAM.
2. Three raw data is ICO-corrected, standardized to the fitst frequency reference value, and stored in RAM as three correction data.
3. Three correction data are calculated by majority decision and moving average, and stored in RAM as measurement results.

Add user's filter

There are two ways to add the user's filter.

1. Instead of filter calculation of R_CTSU_DataGet(), perform user filter calculation and use R_CTSU_DataInsert() to input user filter calculation result.
2. Using the correction data obtained by R_CTSU_SpecificDataGet(), instead of majority decision calculation and filter calculation of R_CTSU_DataGet(), perform user majority decision calculation & filter calculation and use R_CTSU_DataInsert() to input user majority decision calculation & filter calculation result.

Please check example.

User's filter additional Example

Examples

Basic Example

This is a basic example of minimal use of the CTSU in an application.

```
volatile bool g_scan_flag = false;
void ctsu_callback (ctsu_callback_args_t * p_args)
{
 if (CTSU_EVENT_SCAN_COMPLETE == p_args->event)
    {
       g_scan_flag = true;
    }
}
void ctsu_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS];
    err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
       err = R_CTSU_ScanStart(&g_ctsu_ctrl);
      assert(FSP_SUCCESS == err);
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
       err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
 if (FSP_SUCCESS == err)
      {
 /* Application specific data processing. */
      }
```

```
    }
}
```

Multi-configuration Example

This is a optional example of using both Self-capacitance and Mutual-capacitance configurations in the same project.

```
void ctsu_optional_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS + (CTSU_CFG_NUM_MUTUAL_ELEMENTS * 2)];
    err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
    assert(FSP_SUCCESS == err);
    err = R_CTSU_Open(&g_ctsu_ctrl_mutual, &g_ctsu_cfg_mutual);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 R_CTSU_ScanStart(&g_ctsu_ctrl);
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
 R_CTSU_ScanStart(&g_ctsu_ctrl_mutual);
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
       err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
      assert(FSP_SUCCESS == err);
 if (FSP_SUCCESS == err)
      {
 /* Application specific data processing. */
```

```
      }
       err = R_CTSU_DataGet(&g_ctsu_ctrl_mutual, data);
      assert(FSP_SUCCESS == err);
 if (FSP_SUCCESS == err)
      {
 /* Application specific data processing. */
      }
    }
}
```

Diagnosis function Example

This is a Diagnosis function example of using the configuration in the basic example.

```
void ctsu_diag_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS];
    uint16_t dummy;
 R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
    assert(FSP_SUCCESS == err);
 R_CTSU_Open(&g_ctsu_ctrl_diagnosis, &g_ctsu_cfg_diagnosis);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 R_CTSU_ScanStart(&g_ctsu_ctrl);
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
       err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
      assert(FSP_SUCCESS == err);
 R_CTSU_ScanStart(&g_ctsu_ctrl_diagnosis);
 while (!g_scan_flag)
```

```
     {
/* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_DataGet(&g_ctsu_ctrl_diagnosis, &dummy);
     assert(FSP_SUCCESS == err);
if (FSP_SUCCESS == err)
     {
          err = R_CTSU_Diagnosis(&g_ctsu_ctrl_diagnosis);
     assert(FSP_SUCCESS == err);
if (FSP_SUCCESS == err)
     {
break;
     }
     }
   }
}
```

User's filter additional Example

This is a user's filter additiional example of using the configuration in the basic example.
To perform user's filter calculation, change the num_moving_average of the element in the target ctsu_cfg_t to 1.

Perform user filter calculation and use R_CTSU_DataInsert() to input user filter calculation result.

```
void ctsu_user_filter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS];
    uint16_t filter_data[CTSU_CFG_NUM_SELF_ELEMENTS];
 /* If you want to make a touch judgment, call RM_TOUCH_Open()instead of the
following. */
    err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
```

```
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* If you want to make a touch judgment, call RM_TOUCH_ScanStart()instead of the
following. */
       err = R_CTSU_ScanStart(&g_ctsu_ctrl);
      assert(FSP_SUCCESS == err);
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
       err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
 if (FSP_SUCCESS == err)
      {
 /* User original function. */
           ctsu_user_filter(data, filter_data);
           err = R_CTSU_DataInsert(&g_ctsu_ctrl, filter_data);
      assert(FSP_SUCCESS == err);
 /* Call RM_TOUCH_DataGet() to make a touch decision. */
      }
    }
}
```

Using the correction data obtained by R_CTSU_SpecificDataGet(). Perform user majority decision calculation & filter calculation and use R_CTSU_DataInsert() to input user majority decision calculation & filter calculation result.

```
void ctsu_user_majority_decition_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    uint16_t data[CTSU_CFG_NUM_SELF_ELEMENTS];
    uint16_t corr_data[CTSU_CFG_NUM_SELF_ELEMENTS * CTSU_CFG_NUM_SUMULTI];
    uint16_t filter_data[CTSU_CFG_NUM_SELF_ELEMENTS];
```

```
 /* If you want to make a touch judgment, call RM_TOUCH_Open()instead of the
following. */
    err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* If you want to make a touch judgment, call RM_TOUCH_ScanStart()instead of the
following. */
       err = R_CTSU_ScanStart(&g_ctsu_ctrl);
      assert(FSP_SUCCESS == err);
 while (!g_scan_flag)
      {
 /* Wait for scan end callback */
      }
       g_scan_flag = false;
       err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
 if (FSP_SUCCESS == err)
      {
            err = R_CTSU_SpecificDataGet(&g_ctsu_ctrl, corr_data,
CTSU_SPECIFIC_CORRECTION_DATA);
      assert(FSP_SUCCESS == err);
 /* User original function */
            ctsu_user_filter(corr_data, filter_data);
            err = R_CTSU_DataInsert(&g_ctsu_ctrl, filter_data);
      assert(FSP_SUCCESS == err);
 /* Call RM_TOUCH_DataGet() to make a touch decision. */
      }
    }
}
```

Data Structures

struct	ctsu_ctsuwr_t
struct	ctsu_self_buf_t

struct	ctsu_mutual_buf_t
struct	ctsu_correction_info_t
struct	ctsu_instance_ctrl_t

Enumerations

enum	ctsu_state_t
enum	ctsu_tuning_t
enum	ctsu_correction_status_t
enum	ctsu_range_t

Data Structure Documentation

◆ ctsu_ctsuwr_t

struct ctsu_ctsuwr_t

CTSUWR write register value

Data Fields

uint16_t	ctsussc	Copy from (ssdiv << 8) by Open API.
uint16_t	ctsuso0	Copy from ((snum << 10) | so) by Open API.
uint16_t	ctsuso1	Copy from (sdpa << 8) by Open API. ICOG and RICOA is set recommend value.

◆ ctsu_self_buf_t

struct ctsu_self_buf_t

Scan buffer data formats (Self)

Data Fields

uint16_t	sen	Sensor counter data.
uint16_t	ref	Reference counter data (Not used)

◆ ctsu_mutual_buf_t

struct ctsu_mutual_buf_t

Scan buffer data formats (Mutual)

Data Fields

uint16_t	pri_sen	Primary sensor data.

uint16_t	pri_ref	Primary reference data (Not used)
uint16_t	snd_sen	Secondary sensor data.
uint16_t	snd_ref	Secondary reference data (Not used)

◆ ctsu_correction_info_t

struct ctsu_correction_info_t		
Correction information		
Data Fields		
ctsu_correction_status_t	status	Correction status.
ctsu_ctsuwr_t	ctsuwr	Correction scan parameter.
volatile ctsu_self_buf_t	scanbuf	Correction scan buffer.
uint16_t	first_val	1st correction value
uint16_t	second_val	2nd correction value
uint32_t	first_coefficient	1st correction coefficient
uint32_t	second_coefficient	2nd correction coefficient
uint32_t	ctsu_clock	CTSU clock [MHz].

◆ ctsu_instance_ctrl_t

struct ctsu_instance_ctrl_t	
CTSU private control block. DO NOT MODIFY. Initialization occurs when R_CTSU_Open() is called.	
Data Fields	
uint32_t	open
	Whether or not driver is open.
volatile ctsu_state_t	state
	CTSU run state.
ctsu_cap_t	cap
	CTSU Scan Start Trigger Select.
ctsu_md_t	md
	CTSU Measurement Mode Select(copy to cfg)

ctsu_tuning_t	tuning
	CTSU Initial offset tuning status.
uint16_t	num_elements
	Number of elements to scan.
uint16_t	wr_index
	Word index into ctsuwr register array.
uint16_t	rd_index
	Word index into scan data buffer.
uint8_t *	p_tuning_count
	Pointer to tuning count of each element. g_ctsu_tuning_count[] is set by Open API.
int32_t *	p_tuning_diff
	Pointer to difference from base value of each element. g_ctsu_tuning_diff[] is set by Open API.
uint16_t	average
	CTSU Moving average counter.
uint16_t	num_moving_average
	Copy from config by Open API.
uint8_t	ctsucr1
	Copy from (atune1 << 3, md << 6) by Open API. CLK, ATUNE0, CSW, and PON is set by HAL driver.

ctsu_ctsuwr_t *	p_ctsuwr
	CTSUWR write register value. g_ctsu_ctsuwr[] is set by Open API.
ctsu_self_buf_t *	p_self_raw
	Pointer to Self raw data. g_ctsu_self_raw[] is set by Open API.
uint16_t *	p_self_corr
	Pointer to Self correction data. g_ctsu_self_corr[] is set by Open API.
uint16_t *	p_self_data
	Pointer to Self moving average data. g_ctsu_self_data[] is set by Open API.
ctsu_mutual_buf_t *	p_mutual_raw
	Pointer to Mutual raw data. g_ctsu_mutual_raw[] is set by Open API.
uint16_t *	p_mutual_pri_corr
	Pointer to Mutual primary correction data. g_ctsu_self_corr[] is set by Open API.
uint16_t *	p_mutual_snd_corr
	Pointer to Mutual secondary correction data. g_ctsu_self_corr[] is set by Open API.
uint16_t *	p_mutual_pri_data
	Pointer to Mutual primary moving average data. g_ctsu_mutual_pri_data[] is set by Open API.
uint16_t *	p_mutual_snd_data

	Pointer to Mutual secondary moving average data. g_ctsu_mutual_snd_data[] is set by Open API.
ctsu_correction_info_t *	p_correction_info
	Pointer to correction info.
ctsu_txvsel_t	txvsel
	CTSU Transmission Power Supply Select.
ctsu_txvsel2_t	txvsel2
	CTSU Transmission Power Supply Select 2 (CTSU2 Only)
uint8_t	ctsuchac0
	TS00-TS07 enable mask.
uint8_t	ctsuchac1
	TS08-TS15 enable mask.
uint8_t	ctsuchac2
	TS16-TS23 enable mask.
uint8_t	ctsuchac3
	TS24-TS31 enable mask.
uint8_t	ctsuchac4
	TS32-TS39 enable mask.
uint8_t	ctsuchtrc0

	TS00-TS07 mutual-tx mask.
uint8_t	ctsuchtrc1
	TS08-TS15 mutual-tx mask.
uint8_t	ctsuchtrc2
	TS16-TS23 mutual-tx mask.
uint8_t	ctsuchtrc3
	TS24-TS31 mutual-tx mask.
uint8_t	ctsuchtrc4
	TS32-TS39 mutual-tx mask.
uint16_t	self_elem_index
	self element index number for Current instance.
uint16_t	mutual_elem_index
	mutual element index number for Current instance.
uint16_t	ctsu_elem_index
	CTSU element index number for Current instance.
ctsu_cfg_t const *	p_ctsu_cfg
	Pointer to initial configurations.
IRQn_Type	write_irq
	Copy from config by Open API. CTSU_CTSUWR interrupt vector.

IRQn_Type	read_irq
	Copy from config by Open API. CTSU_CTSURD interrupt vector.
IRQn_Type	end_irq
	Copy from config by Open API. CTSU_CTSUFN interrupt vector.
void(*	p_callback)(ctsu_callback_args_t *)
	Callback provided when a CTSUFN occurs.
uint8_t	interrupt_reverse_flag
	Flag in which read interrupt and end interrupt are reversed.
ctsu_event_t	error_status
	error status variable to send to QE for serial tuning.
ctsu_callback_args_t *	p_callback_memory
	Pointer to non-secure memory that can be used to pass arguments to a callback in non-secure memory.
void const *	p_context
	Placeholder for user data.
bool	serial_tuning_enable
	Flag of serial tuning status.
uint16_t	serial_tuning_mutual_cnt
	Word index into ctsuwr register array.

uint16_t	tuning_self_target_value
	Target self value for initial offset tuning.
uint16_t	tuning_mutual_target_value
	Target mutual value for initial offset tuning.

Enumeration Type Documentation

◆ ctsu_state_t

enum ctsu_state_t

CTSU run state

Enumerator	
CTSU_STATE_INIT	Not open.
CTSU_STATE_IDLE	Opened.
CTSU_STATE_SCANNING	Scanning now.
CTSU_STATE_SCANNED	Scan end.

◆ ctsu_tuning_t

enum ctsu_tuning_t

CTSU Initial offset tuning status

Enumerator	
CTSU_TUNING_INCOMPLETE	Initial offset tuning incomplete.
CTSU_TUNING_COMPLETE	Initial offset tuning complete.

◆ ctsu_correction_status_t

enum ctsu_correction_status_t	
CTSU Correction status	
Enumerator	
CTSU_CORRECTION_INIT	Correction initial status.
CTSU_CORRECTION_RUN	Correction scan running.
CTSU_CORRECTION_COMPLETE	Correction complete.
CTSU_CORRECTION_ERROR	Correction error.

◆ ctsu_range_t

enum ctsu_range_t	
CTSU range definition	
Enumerator	
CTSU_RANGE_20UA	20uA mode
CTSU_RANGE_40UA	40uA mode
CTSU_RANGE_80UA	80uA mode
CTSU_RANGE_160UA	160uA mode
CTSU_RANGE_NUM	number of range

Function Documentation

◆ R_CTSU_Open()

```
fsp_err_t R_CTSU_Open ( ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg )
```

Opens and configures the CTSU driver module. Implements ctsu_api_t::open.

Example:

```
err = R_CTSU_Open(&g_ctsu_ctrl, &g_ctsu_cfg);
```

Return values

FSP_SUCCESS	CTSU successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

Note

In the first Open, measurement for correction works, and it takes several tens of milliseconds.

◆ R_CTSU_ScanStart()

fsp_err_t R_CTSU_ScanStart (ctsu_ctrl_t *const *p_ctrl*)

This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with R_CTSU_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements ctsu_api_t::scanStart.

Example:

```
while (true)
   {
      err = R_CTSU_ScanStart(&g_ctsu_ctrl);
     assert(FSP_SUCCESS == err);
while (!g_scan_flag)
     {
/* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
     }
   }
```

Return values

FSP_SUCCESS	CTSU successfully configured.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CTSU_SCANNING	Scanning this instance or other.
FSP_ERR_CTSU_NOT_GET_DATA	The previous data has not been retrieved by DataGet.

◆ R_CTSU_DataGet()

fsp_err_t R_CTSU_DataGet (ctsu_ctrl_t *const *p_ctrl*, uint16_t * *p_data*)

This function gets the sensor values as scanned by the CTSU. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements ctsu_api_t::dataGet.

Example:

```
while (true)
   {
      err = R_CTSU_ScanStart(&g_ctsu_ctrl);
     assert(FSP_SUCCESS == err);
while (!g_scan_flag)
     {
/* Wait for scan end callback */
     }
      g_scan_flag = false;
      err = R_CTSU_DataGet(&g_ctsu_ctrl, data);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
     }
   }
```

Return values

FSP_SUCCESS	CTSU successfully configured.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CTSU_SCANNING	Scanning this instance.
FSP_ERR_CTSU_INCOMPLETE_TUNING	Incomplete initial offset tuning.
FSP_ERR_CTSU_DIAG_NOT_YET	Diagnosis of data collected no yet.

◆ R_CTSU_ScanStop()

fsp_err_t R_CTSU_ScanStop (ctsu_ctrl_t *const *p_ctrl*)

This function scan stops the sensor as scanning by the CTSU. Implements ctsu_api_t::scanStop.

Return values

FSP_SUCCESS	CTSU successfully scan stop.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

◆ R_CTSU_CallbackSet()

fsp_err_t R_CTSU_CallbackSet (ctsu_ctrl_t *const *p_api_ctrl*, void(*)(ctsu_callback_args_t *) *p_callback*, void const *const *p_context*, ctsu_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements ctsu_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_CTSU_Diagnosis()

fsp_err_t R_CTSU_Diagnosis (ctsu_ctrl_t *const *p_ctrl*)

Diagnosis the CTSU peripheral. Implements ctsu_api_t::diagnosis.

Example:

```
        err = R_CTSU_Diagnosis(&g_ctsu_ctrl_diagnosis);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	CTSU successfully configured.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CTSU_NOT_GET_DATA	The previous data has not been retrieved by DataGet.
FSP_ERR_CTSU_DIAG_LDO_OVER_VOLTAGE	Diagnosis of LDO over voltage failed.
FSP_ERR_CTSU_DIAG_CCO_HIGH	Diagnosis of CCO into 19.2uA failed.
FSP_ERR_CTSU_DIAG_CCO_LOW	Diagnosis of CCO into 2.4uA failed.
FSP_ERR_CTSU_DIAG_SSCG	Diagnosis of SSCG frequency failed.
FSP_ERR_CTSU_DIAG_DAC	Diagnosis of non-touch count value failed.
FSP_ERR_CTSU_DIAG_OUTPUT_VOLTAGE	Diagnosis of LDO output voltage failed.
FSP_ERR_CTSU_DIAG_OVER_VOLTAGE	Diagnosis of over voltage detection circuit failed.
FSP_ERR_CTSU_DIAG_OVER_CURRENT	Diagnosis of over current detection circuit failed.
FSP_ERR_CTSU_DIAG_LOAD_RESISTANCE	Diagnosis of LDO internal resistance value failed.
FSP_ERR_CTSU_DIAG_CURRENT_SOURCE	Diagnosis of LDO internal resistance value failed.
FSP_ERR_CTSU_DIAG_SENSCLK_GAIN	Diagnosis of SENSCLK frequency gain failed.
FSP_ERR_CTSU_DIAG_SUCLK_GAIN	Diagnosis of SUCLK frequency gain failed.
FSP_ERR_CTSU_DIAG_CLOCK_RECOVERY	Diagnosis of SUCLK clock recovery function failed.
FSP_ERR_CTSU_DIAG_CFC_GAIN	Diagnosis of CFC oscillator gain failed.

◆ R_CTSU_Close()

fsp_err_t R_CTSU_Close (ctsu_ctrl_t *const *p_ctrl*)

Disables specified CTSU control block. Implements ctsu_api_t::close.

Return values

FSP_SUCCESS	CTSU successfully configured.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

◆ R_CTSU_SpecificDataGet()

fsp_err_t R_CTSU_SpecificDataGet (ctsu_ctrl_t *const *p_ctrl*, uint16_t * *p_specific_data*, ctsu_specific_data_type_t *specific_data_type*)

This function gets the sensor specific data values as scanned by the CTSU. Call this function after calling the R_CTSU_DataGet() function.

By setting the third argument to CTSU_SPECIFIC_RAW_DATA, RAW data can be output from the second argument.

By setting the third argument to CTSU_SPECIFIC_CORRECTION_DATA, the corrected data can be output from the second argument.

By setting the third argument to CTSU_SPECIFIC_SELECTED_DATA, Get bitmap of the frequency values used in majority decision from the second argument. The bitch map is shown as follows. ||2bit | 1bit | 0bit || ||3rd frequency value | 2nd frequency value | 1st frequency value ||

Implements ctsu_api_t::specificDataGet.

Example:

```
            err = R_CTSU_SpecificDataGet(&g_ctsu_ctrl, corr_data,
CTSU_SPECIFIC_CORRECTION_DATA);
```

Return values

FSP_SUCCESS	CTSU successfully configured.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CTSU_SCANNING	Scanning this instance.
FSP_ERR_CTSU_INCOMPLETE_TUNING	Incomplete initial offset tuning.
FSP_ERR_NOT_ENABLED	CTSU_SPECIFIC_SELECTED_FREQ is not enabled in CTSU1.(CTSU2 Only)

◆ R_CTSU_DataInsert()

fsp_err_t R_CTSU_DataInsert (ctsu_ctrl_t *const *p_ctrl*, uint16_t * *p_insert_data*)

This function inserts the value of the second argument as the measurement result value. Call this function after calling the R_CTSU_DataInsert() function. Implements ctsu_api_t::dataInsert.

Example:

```
err = R_CTSU_DataInsert(&g_ctsu_ctrl, filter_data);
```

Return values

FSP_SUCCESS	CTSU successfully configured.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CTSU_SCANNING	Scanning this instance.
FSP_ERR_CTSU_INCOMPLETE_TUNING	Incomplete initial offset tuning.

4.2.16 Digital to Analog Converter (r_dac)

Modules

Functions

fsp_err_t	R_DAC_Open (dac_ctrl_t *p_api_ctrl, dac_cfg_t const *const p_cfg)
fsp_err_t	R_DAC_Write (dac_ctrl_t *p_api_ctrl, uint16_t value)
fsp_err_t	R_DAC_Start (dac_ctrl_t *p_api_ctrl)
fsp_err_t	R_DAC_Stop (dac_ctrl_t *p_api_ctrl)
fsp_err_t	R_DAC_Close (dac_ctrl_t *p_api_ctrl)

Detailed Description

Driver for the DAC12 peripheral on RA MCUs. This module implements the DAC Interface.

Overview

Features

The DAC module outputs one of 4096 voltage levels between the positive and negative reference voltages.

- Supports setting left-justified or right-justified 12-bit value format for the 16-bit input data registers
- Supports output amplifiers on selected MCUs
- Supports charge pump on selected MCUs
- Supports synchronization with the Analog-to-Digital Converter (ADC) module

Configuration

Note

For MCUs supporting more than one channel, the following configuration options are shared by all the DAC channels:
- *Synchronize with ADC*
- *Data Format*
- *Charge Pump*

Build Time Configurations for r_dac

The following build time configurations are defined in fsp_cfg/r_dac_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Analog > DAC (r_dac)

This module can be added to the Stacks tab via New Stack > Analog > DAC (r_dac). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_dac0	Module name.
Channel	Value must be an integer greater than or equal to 0	0	Specify the hardware channel.
Synchronize with ADC	MCU Specific Options		Enable DA/AD synchronization.
Data Format	• Right Justified • Left Justified	Right Justified	Specify the DAC data format.
Output Amplifier	MCU Specific Options		Enable the DAC output amplifier.
Charge Pump (Requires MOCO active)	MCU Specific Options		Enable the DAC charge pump.
Internal Output	MCU Specific Options		Enable DAC output to internal modules.
ELC Trigger Source	MCU Specific Options		ELC event source that will trigger the DAC to

start a conversion.

Clock Configuration

The DAC peripheral module uses PCLKB as its clock source.

Pin Configuration

The DAn pins are used as analog outputs. Each DAC channel has one output pin.

The AVCC0 and AVSS0 pins are power and ground supply pins for the DAC and ADC.

The VREFH and VREFL pins are top and ground voltage reference pins for the DAC and ADC.

Usage Notes

Charge Pump

The charge pump must be enabled when using DAC pin output while operating at AV_{CC} < 2.7V.

Note

The MOCO must be running to use the charge pump.
If the DAC output is to be routed to an internal signal, do not enable the charge pump.

Output to Internal Modules

The DAC output can be used as an analog input to other peripherals on the MCU (eg. ACMPHS, ADC) without outputting the voltage to an external pin.

On some MCUs this functionality must be enabled during configuration using dac_extended_cfg_t::internal_output_enabled. When internal output is enabled, the DAC output will be routed to internal modules. If the DAC output amplifier is enabled or when internal output is disabled, the output will be routed to the DAC output pin (DAn).

Synchronization with ADC

When ADC synchronization is enabled and an ADC conversion is in progress, if a DAC conversion is started it will automatically be delayed until after the ADC conversion is complete.

Limitations

- For MCUs supporting ADC unit 1:
  - Once synchronization between DAC and ADC unit 1 is turned on during R_DAC_Open synchronization cannot be turned off by the driver. In order to desynchronize DAC with ADC unit 1, manually clear DAADSCR.DAADST to 0 when the ADCSR.ADST bit is 0 and ADC unit 1 is halted.
  - The DAC module can only be synchronized with ADC unit 1.
  - For MCUs having more than 1 DAC channel, both channels are synchronized with ADC unit 1 if synchronization is enabled.
- Pin configuration does not support setting ASEL and PSEL bit fields for the same pin. In order to set the PSEL when DAC pin is enabled, maually update the setting of the DAC pin to .pin_cfg = ((uint32_t)IOPORT_CFG_ANALOG_ENABLE | IOPORT_CFG_PERIPHERAL_PIN | IOPORT_PERIPHERAL_CAC_AD)}.

Examples

Basic Example

This is a basic example of minimal use of the R_DAC in an application. This example shows how this driver can be used for basic Digital to Analog Conversion operations.

```
void basic_example (void)
{
 fsp_err_t err;
    uint16_t value;
 /* Pin configuration: Output enable DA0 as Analog. */
 /* Initialize the DAC channel */
    err = R_DAC_Open(&g_dac_ctrl, &g_dac_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    value = (uint16_t) DAC_EXAMPLE_VALUE_ABC;
    err = R_DAC_Write(&g_dac_ctrl, value);
    assert(FSP_SUCCESS == err);
    err = R_DAC_Start(&g_dac_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	dac_instance_ctrl_t
struct	dac_extended_cfg_t

Data Structure Documentation

◆ dac_instance_ctrl_t

struct dac_instance_ctrl_t
DAC instance control block.

◆ dac_extended_cfg_t

struct dac_extended_cfg_t		
DAC extended configuration		
Data Fields		
bool	enable_charge_pump	Enable DAC charge pump

		available on selected MCUs.
bool	output_amplifier_enabled	Output amplifier enable available on selected MCUs.
bool	internal_output_enabled	Internal output enable available on selected MCUs.
dac_data_format_t	data_format	Data format.

Function Documentation

◆ R_DAC_Open()

fsp_err_t R_DAC_Open (dac_ctrl_t * *p_api_ctrl*, dac_cfg_t const *const *p_cfg*)

Perform required initialization described in hardware manual. Implements dac_api_t::open. Configures a single DAC channel, starts the channel, and provides a handle for use with the DAC API Write and Close functions. Must be called once prior to calling any other DAC API functions. After a channel is opened, Open should not be called again for the same channel without calling Close first.

Return values

FSP_SUCCESS	The channel was successfully opened.
FSP_ERR_ASSERTION	Parameter check failure due to one or more reasons below: 1. One or both of the following parameters may be NULL: p_api_ctrl or p_cfg 2. data_format value in p_cfg is out of range. 3. Extended configuration structure is set to NULL for MCU supporting charge pump.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel ID requested in p_cfg may not available on the devices.
FSP_ERR_ALREADY_OPEN	The control structure is already opened.

◆ R_DAC_Write()

fsp_err_t R_DAC_Write (dac_ctrl_t * *p_api_ctrl*, uint16_t *value*)

Write data to the D/A converter and enable the output if it has not been enabled.

Return values

FSP_SUCCESS	Data is successfully written to the D/A Converter.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

◆ R_DAC_Start()

fsp_err_t R_DAC_Start (dac_ctrl_t * *p_api_ctrl*)

Start the D/A conversion output if it has not been started.

Return values

FSP_SUCCESS	The channel is started successfully.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_IN_USE	Attempt to re-start a channel.
FSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

◆ R_DAC_Stop()

fsp_err_t R_DAC_Stop (dac_ctrl_t * *p_api_ctrl*)

Stop the D/A conversion and disable the output signal.

Return values

FSP_SUCCESS	The control is successfully stopped.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

◆ R_DAC_Close()

fsp_err_t R_DAC_Close (dac_ctrl_t * *p_api_ctrl*)

Stop the D/A conversion, stop output, and close the DAC channel.

Return values

FSP_SUCCESS	The channel is successfully closed.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Channel associated with p_ctrl has not been opened.

4.2.17 Digital to Analog Converter (r_dac8)

Modules

Functions

fsp_err_t	R_DAC8_Open (dac_ctrl_t *const p_ctrl, dac_cfg_t const *const p_cfg)
fsp_err_t	R_DAC8_Close (dac_ctrl_t *const p_ctrl)
fsp_err_t	R_DAC8_Write (dac_ctrl_t *const p_ctrl, uint16_t value)
fsp_err_t	R_DAC8_Start (dac_ctrl_t *const p_ctrl)
fsp_err_t	R_DAC8_Stop (dac_ctrl_t *const p_ctrl)

Detailed Description

Driver for the DAC8 peripheral on RA MCUs. This module implements the DAC Interface.

Overview

Features

The DAC8 module outputs one of 256 voltage levels between the positive and negative reference voltages. DAC8 on selected MCUs have below features

- Charge pump control
- Synchronization with the Analog-to-Digital Converter (ADC) module
- Multiple Operation Modes
    - Normal
    - Real-Time (Event Link)

Configuration

Note

For MCUs supporting more than one channel, the following configuration options are shared by all the DAC8 channels:

- *Synchronize with ADC*
- *Charge Pump*

Build Time Configurations for r_dac8

The following build time configurations are defined in fsp_cfg/r_dac8_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Analog > DAC8 (r_dac8)

This module can be added to the Stacks tab via New Stack > Analog > DAC8 (r_dac8).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_dac8_0	Module name.
Channel	Value must be an integer greater than or equal to 0	0	Specify the hardware channel.
D/A A/D Synchronous Conversion	MCU Specific Options		Synchronize the DAC8 update with the ADC to reduce interference with A/D conversions.
DAC Mode	MCU Specific Options		Select the DAC operating mode
Real-time Trigger Event	MCU Specific Options		Specify the event used to trigger conversion in Real-time mode. This setting is only valid when Real-time mode is enabled.
Charge Pump (Requires MOCO active)	MCU Specific Options		Enable the DAC charge pump.

Clock Configuration

The DAC8 peripheral module uses the PCLKB as its clock source.

Pin Configuration

The DA8_n pins are used as analog outputs. Each DAC8 channel has one output pin.

The AVCC0 and AVSS0 pins are power and ground supply and reference pins for the DAC8.

Usage Notes

Charge Pump

The charge pump must be enabled when using DAC8 pin output while operating at AV_{CC} < 2.7V.

Note

The MOCO must be running to use the charge pump.
If DAC8 output is to be routed to an internal signal, do not enable the charge pump.

Synchronization with ADC

When ADC synchronization is enabled and an ADC conversion is in progress, if a DAC8 conversion is started it will automatically be delayed until after the ADC conversion is complete.

Real-time Mode

When Real-time mode is selected, the DAC8 will perform a conversion each time the selected ELC event is received.

Limitations

- Synchronization between DAC8 and ADC is activated when calling R_DAC8_Open. At this point synchronization cannot be deactivated by the driver. In order to desynchronize DAC8 with ADC, manually clear DACADSCR.DACADST to 0 while the ADCSR.ADST bit is 0 and the ADC is halted.
- For MCUs having more than 1 DAC8 channel, both channels are synchronized with ADC if synchronization is enabled.
- Pin configuration does not support setting ASEL and PSEL bit fields for the same pin. In order to set the PSEL when DAC pin is enabled, maually update the setting of the DAC pin to .pin_cfg = ((uint32_t)IOPORT_CFG_ANALOG_ENABLE | IOPORT_CFG_PERIPHERAL_PIN | IOPORT_PERIPHERAL_CAC_AD)).

Examples

Basic Example

This is a basic example of minimal use of the R_DAC8 in an application. This example shows how this driver can be used for basic 8 bit Digital to Analog Conversion operations.

```
dac8_instance_ctrl_t g_dac8_ctrl;
dac_cfg_t g_dac8_cfg =
{
    .channel             = 0U,
    .ad_da_synchronized = false,
    .p_extend          = &g_dac8_cfg_extend
```

```
};
void basic_example (void)
{
 fsp_err_t err;
    uint16_t value;
 /* Pin configuration: Output enable DA8_0(RA2A1) as Analog. */
 /* Initialize the DAC8 channel */
    err = R_DAC8_Open(&g_dac8_ctrl, &g_dac8_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    value = (uint8_t) DAC8_EXAMPLE_VALUE_ABC;
 /* Write value to DAC module */
    err = R_DAC8_Write(&g_dac8_ctrl, value);
    assert(FSP_SUCCESS == err);
 /* Start DAC8 conversion */
    err = R_DAC8_Start(&g_dac8_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	dac8_instance_ctrl_t
struct	dac8_extended_cfg_t

Enumerations

enum	dac8_mode_t

Data Structure Documentation

◆ dac8_instance_ctrl_t

struct dac8_instance_ctrl_t
DAC8 instance control block. DO NOT INITIALIZE.

◆ dac8_extended_cfg_t

struct dac8_extended_cfg_t
DAC8 extended configuration
Data Fields

bool	enable_charge_pump	Enable DAC charge pump.
dac8_mode_t	dac_mode	DAC mode.

Enumeration Type Documentation

◆ dac8_mode_t

enum dac8_mode_t

Enumerator	
DAC8_MODE_NORMAL	DAC Normal mode.
DAC8_MODE_REAL_TIME	DAC Real-time (event link) mode.

Function Documentation

◆ R_DAC8_Open()

fsp_err_t R_DAC8_Open (dac_ctrl_t *const *p_ctrl*, dac_cfg_t const *const *p_cfg*)

Perform required initialization described in hardware manual.

Implements dac_api_t::open.

Configures a single DAC channel. Must be called once prior to calling any other DAC API functions. After a channel is opened, Open should not be called again for the same channel without calling Close first.

Return values

FSP_SUCCESS	The channel was successfully opened.
FSP_ERR_ASSERTION	One or both of the following parameters may be NULL: p_ctrl or p_cfg
FSP_ERR_ALREADY_OPEN	The instance control structure has already been opened.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	An invalid channel was requested.
FSP_ERR_NOT_ENABLED	Setting DACADSCR is not enabled when ADCSR.ADST = 0.

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

◆ R_DAC8_Close()

fsp_err_t R_DAC8_Close (dac_ctrl_t *const *p_ctrl*)

Stop the D/A conversion, stop output, and close the DAC channel.

Return values

FSP_SUCCESS	The channel is successfully closed.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Channel associated with p_instance_ctrl has not been opened.

◆ R_DAC8_Write()

fsp_err_t R_DAC8_Write (dac_ctrl_t *const *p_ctrl*, uint16_t *value*)

Write data to the D/A converter.

Return values

FSP_SUCCESS	Data is successfully written to the D/A Converter.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Channel associated with p_instance_ctrl has not been opened.
FSP_ERR_OVERFLOW	Data overflow when data value exceeds 8-bit limit.

◆ R_DAC8_Start()

fsp_err_t R_DAC8_Start (dac_ctrl_t *const *p_ctrl*)

Start the D/A conversion output.

Return values

FSP_SUCCESS	The channel is started successfully.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Channel associated with p_instance_ctrl has not been opened.
FSP_ERR_IN_USE	Attempt to re-start a channel.

◆ R_DAC8_Stop()

fsp_err_t R_DAC8_Stop (dac_ctrl_t *const *p_ctrl*)

Stop the D/A conversion and disable the output signal.

Return values

FSP_SUCCESS	The control is successfully stopped.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Channel associated with p_instance_ctrl has not been opened.

4.2.18 Direct Memory Access Controller (r_dmac)

Modules

Functions

fsp_err_t	R_DMAC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg)
fsp_err_t	R_DMAC_Reconfigure (transfer_ctrl_t *const p_api_ctrl, transfer_info_t *p_info)
fsp_err_t	R_DMAC_Reset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const num_transfers)
fsp_err_t	R_DMAC_SoftwareStart (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode)
fsp_err_t	R_DMAC_SoftwareStop (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DMAC_Enable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DMAC_Disable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DMAC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_info)
fsp_err_t	R_DMAC_Close (transfer_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the DMAC peripheral on RA MCUs. This module implements the Transfer Interface.

Overview

The Direct Memory Access Controller (DMAC) transfers data from one memory location to another without using the CPU.

Features

- Supports multiple transfer modes
  - Normal transfer
  - Repeat transfer
  - Block transfer
  - Repeat-Block transfer (Not available on all MCUs)
- Address increment, decrement, fixed, or offset modes
- Triggered by ELC events
  - Some exceptions apply, see the Event table in the Event Numbers section of the Interrupt Controller Unit chapter of the hardware manual
- Supports 1, 2, and 4 byte data units

Configuration

Build Time Configurations for r_dmac

The following build time configurations are defined in fsp_cfg/r_dmac_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Transfer > Transfer (r_dmac)

This module can be added to the Stacks tab via New Stack > Transfer > Transfer (r_dmac).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_transfer0	Module name.
Channel	Value must be a non-negative integer	0	Specify the hardware channel.
Mode	• Normal • Repeat • Block • Repeat-Block	Normal	Select the transfer mode. Normal: One transfer per activation, transfer ends after Number of Transfers; Repeat: One transfer per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of

			Blocks; Block: Number of Blocks per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of Blocks.
Transfer Size	• 1 Byte • 2 Bytes • 4 Bytes	2 Bytes	Select the transfer size.
Destination Address Mode	• Fixed • Offset addition • Incremented • Decremented	Fixed	Select the address mode for the destination.
Source Address Mode	• Fixed • Offset addition • Incremented • Decremented	Fixed	Select the address mode for the source.
Repeat Area (Unused in Normal Mode)	• Destination • Source	Source	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	Manual Entry	NULL	Specify the transfer destination pointer.
Source Pointer	Manual Entry	NULL	Specify the transfer source pointer.
Number of Transfers	Value must be a non-negative integer	1	Specify the number of transfers for repeat and normal mode or block size for repeat-block transfer mode.
Number of Blocks (Valid only in Repeat,Block or Repeat-Block Mode)	Value must be a non-negative integer	0	Specify the number of blocks to transfer in Repeat,Block or Repeat-Block mode.
Activation Source	MCU Specific Options		Select the DMAC transfer start event. If no ELC event is chosen then software start can be used.
Callback	Name must be a valid C symbol	NULL	A user callback that is called at the end of the transfer.
Context	Manual Entry	NULL	Pointer to the context structure passed

			through the callback argument.
Transfer End Interrupt Priority	MCU Specific Options		Select the transfer end interrupt priority.
Interrupt Frequency	• Interrupt after all transfers have completed • Interrupt after each block, or repeat size is transfered	Interrupt after all transfers have completed	Select to have interrupt after each transfer or after last transfer.
Offset value (Valid only when address mode is \'Offset\')	Value must be a 24 bit signed integer.	1	Offset value is added to the address after each transfer.
Source Buffer Size	Value must be a non-negative integer	1	Specify the size of whole source buffer (valid only for Repeat-Block transfer mode with source address update mode other than offset addition).

Clock Configuration

The DMAC peripheral module uses ICLK as the clock source. The ICLK frequency is set by using the **Clocks** tab of the RA Configuration editor prior to a build, or by using the CGC module at run-time.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Transfer Modes

The DMAC Module supports three modes of operation.

- **Normal Mode** - In normal mode, a single data unit is transfered every time the configured ELC event is received by the DMAC channel. A data unit can be 1-byte, 2-bytes, or 4-bytes. The source and destination addresses can be fixed, increment, decrement, or add an offset to the next data unit after each transfer. A 16-bit counter decrements after each transfer. When the counter reaches 0, transfers will no longer be triggered by the ELC event and the CPU can be interrupted to signal that all transfers have finished.
- **Repeat Mode** - Repeat mode works the same way as normal mode, however the length is limited to an integer in the range[1,1024]. When the transfer counter reaches 0, the counter is reset to its configured value, the repeat area (source or destination address) resets to its starting address and the block count remaining will decrement by 1. When the block count reaches 0, transfers will no longer be triggered by the ELC event and the CPU may be interrupted to signal that all transfers have finished.
- **Block Mode** - In block mode, the amount of data units transfered by each interrupt can be set to an integer in the range [1,1024]. The number of blocks to transfer can also be configured to a 16-bit number. After each block transfer the repeat area (source or

destination address) will reset to the original address and the other address will be incremented or decremented to the next block.
- **Repeat-Block Mode** - In repeat-block mode, the amount of data units transfered by each interrupt can be set to an integer in the range [1,1024]. The number of blocks to transfer can be configured to a 16 bit number. If the destination address mode is offset mode, maximum configurable number of blocks is 0xFFFF for block size(length) of one with data transfer size as byte,0x7FFF for block size of one with data transfer size as half word and 0x3FFF for block size of one with data size as word. After each block transfer the source address and the destination address will be incremented or decremented to the next block address. In case of offset address mode for source address, the source address size is the total size of source buffer after which the source area is rolled over, block size can be smaller than the source buffer size.For source address mode as offset mode, the maximum configurable source buffer size is 0xFFFF for transfer data size of a byte,0x7FFF for transfer data size of half word and 0x3FFF for transfer data size of word. Repeat-block mode can be used to implement single ring buffer to multiple ring buffer transfer type design.

Selecting the DTC or DMAC

The Transfer API is implemented by both DTC and the DMAC so that applications can switch between the DTC and the DMAC. When selecting between them, consider these factors:

	DTC	DMAC
Repeat Mode	• Repeats forever • Max repeat size is 256 x 4 bytes	• Configurable number of repeats • Max repeat size is 1024 x 4 bytes
Block Mode	• Max block size is 256 x 4 bytes	• Max block size is 1024 x 4 bytes
Channels	• One instance per interrupt	• MCU specific (8 channels or less)
Chained Transfers	• Supported	• Not Supported
Software Trigger	• Must use the software ELC event	• Has support for software trigger without using software ELC event • Supports TRANSFER_START_MODE_SINGLE and TRANSFER_START_MODE_REPEAT
Offset Address Mode	• Not supported	• Supported

Interrupts

The DTC and DMAC interrupts behave differently. The DTC uses the configured IELSR event IRQ as the interrupt source whereas each DMAC channel has its own IRQ.

The transfer_info_t::irq setting also behaves a little differently depending on which mode is selected.

Normal Mode

	DTC	DMAC

TRANSFER_IRQ_EACH	Interrupt after each transfer	N/A
TRANSFER_IRQ_END	Interrupt after last transfer	Interrupt after last transfer

Repeat Mode

	DTC	DMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	Interrupt after each repeat
TRANSFER_IRQ_END	Interrupt after each repeat	Interrupt after last transfer

Block Mode

	DTC	DMAC
TRANSFER_IRQ_EACH	Interrupt after each block	Interrupt after each block
TRANSFER_IRQ_END	Interrupt after last block	Interrupt after last block

Additional Considerations

- The DTC requires a moderate amount of RAM (one transfer_info_t struct per open instance + DTC_VECTOR_TABLE_SIZE).
- The DTC stores transfer information in RAM and writes back to RAM after each transfer whereas the DMAC stores all transfer information in registers.
- When transfers are configured for more than one activation source, the DTC must fetch the transfer info from RAM on each interrupt. This can cause a higher latency between transfers.

Offset Address Mode

When the source or destination mode is configured to offset mode, a configurable offset is added to the source or destination pointer after each transfer. The offset is a signed 24 bit number.

Examples

Basic Example

This is a basic example of minimal use of the DMAC in an application. In this case, one or more events have been routed to the DMAC for handling so it only needs to be enabled to start accepting transfers.

```
void dmac_minimal_example (void)
{
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_DMAC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable the DMAC so that it responds to transfer requests. */
    err = R_DMAC_Enable(&g_transfer_ctrl);
```

```
    assert(FSP_SUCCESS == err);
}
```

CRC32 Example

In this example the DMAC is used to feed the CRC peripheral to perform a CRC32 operation.

```
volatile bool g_transfer_complete = false;
void dmac_callback (dmac_callback_args_t * cb_data)
{
 FSP_PARAMETER_NOT_USED(cb_data);
    g_transfer_complete = true;
}
void dmac_crc_example (void)
{
    uint8_t p_src[TRANSFER_LENGTH];
 /* Initialize p_src to [ABC..OP] */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       p_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Set transfer source address to p_src */
    g_transfer_cfg.p_info->p_src = (void *) p_src;
 /* Set transfer destination address to the CRC data input register */
    g_transfer_cfg.p_info->p_dest = (void *) &R_CRC->CRCDIR;
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_DMAC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable DMAC transfers. */
    (void) R_DMAC_Enable(&g_transfer_ctrl);
 /* Open the CRC module. */
    err = R_CRC_Open(&g_crc_ctrl, &g_crc_cfg);
    assert(FSP_SUCCESS == err);
 /* Clear the transfer complete flag. */
```

```
    g_transfer_complete = false;
/* Trigger the transfer using software. */
    err = R_DMAC_SoftwareStart(&g_transfer_ctrl, TRANSFER_START_MODE_SINGLE);
    assert(FSP_SUCCESS == err);
while (!g_transfer_complete)
    {
/* Wait for transfer complete interrupt */
    }
/* Get CRC result and perform final XOR. */
    uint32_t crc32;
    (void) R_CRC_CalculatedValueGet(&g_crc_ctrl, &crc32);
    crc32 ^= CRC32_FINAL_XOR_VALUE;
/* Verify that the CRC32 is calculated correctly. */
/* CRC32("ABCD...NOP") = 0xE0E8FF4D. */
const uint32_t expected_crc32 = 0xE0E8FF4D;
if (expected_crc32 != crc32)
    {
/* Handle any CRC errors. This function should be defined by the user. */
        handle_crc_error();
    }
}
```

Data Structures

struct	dmac_instance_ctrl_t
struct	dmac_callback_args_t
struct	dmac_extended_cfg_t

Macros

#define	DMAC_MAX_NORMAL_TRANSFER_LENGTH
#define	DMAC_MAX_REPEAT_TRANSFER_LENGTH
#define	DMAC_MAX_BLOCK_TRANSFER_LENGTH
#define	DMAC_MAX_REPEAT_COUNT
#define	DMAC_MAX_BLOCK_COUNT

Data Structure Documentation

◆ dmac_instance_ctrl_t

struct dmac_instance_ctrl_t
Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in transfer_api_t::open.

◆ dmac_callback_args_t

struct dmac_callback_args_t		
Callback function parameter data.		
Data Fields		
void const *	p_context	Placeholder for user data. Set in r_transfer_t::open function in transfer_cfg_t.

◆ dmac_extended_cfg_t

struct dmac_extended_cfg_t	
DMAC transfer configuration extension. This extension is required.	
Data Fields	
uint8_t	channel
	Channel number, does not apply to all HAL drivers.
IRQn_Type	irq
	DMAC interrupt number.
uint8_t	ipl
	DMAC interrupt priority.
int32_t	offset
	Offset value used with transfer_addr_mode_t::TRANSFER_ADDR_MODE_OFFSET.
uint16_t	src_buffer_size

elc_event_t	activation_source
void(*	p_callback)(dmac_callback_args_t *cb_data)
void const *	p_context

Field Documentation

◆ src_buffer_size

uint16_t dmac_extended_cfg_t::src_buffer_size

Source ring buffer size for TRANSFER_MODE_REPEAT_BLOCK.

◆ activation_source

elc_event_t dmac_extended_cfg_t::activation_source

Select which event will trigger the transfer.

Note

Select ELC_EVENT_NONE for software activation in order to use softwareStart and softwareStart to trigger transfers.

◆ p_callback

void(* dmac_extended_cfg_t::p_callback) (dmac_callback_args_t *cb_data)

Callback for transfer end interrupt.

◆ p_context

void const* dmac_extended_cfg_t::p_context

Placeholder for user data. Passed to the user p_callback in dmac_callback_args_t.

Macro Definition Documentation

◆ DMAC_MAX_NORMAL_TRANSFER_LENGTH

#define DMAC_MAX_NORMAL_TRANSFER_LENGTH

Max configurable number of transfers in TRANSFER_MODE_NORMAL.

◆ DMAC_MAX_REPEAT_TRANSFER_LENGTH

#define DMAC_MAX_REPEAT_TRANSFER_LENGTH

Max number of transfers per repeat for TRANSFER_MODE_REPEAT.

◆ DMAC_MAX_BLOCK_TRANSFER_LENGTH

```
#define DMAC_MAX_BLOCK_TRANSFER_LENGTH
```

Max number of transfers per block in TRANSFER_MODE_BLOCK

◆ DMAC_MAX_REPEAT_COUNT

```
#define DMAC_MAX_REPEAT_COUNT
```

Max configurable number of repeats to trasnfer in TRANSFER_MODE_REPEAT

◆ DMAC_MAX_BLOCK_COUNT

```
#define DMAC_MAX_BLOCK_COUNT
```

Max configurable number of blocks to transfer in TRANSFER_MODE_BLOCK

Function Documentation

◆ R_DMAC_Open()

```
fsp_err_t R_DMAC_Open ( transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg )
```

Configure a DMAC channel.

Return values

FSP_SUCCESS	Successful open.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The configured channel is invalid.
FSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the activation source is not enabled in the BSP.
FSP_ERR_ALREADY_OPEN	The control structure is already opened.

◆ R_DMAC_Reconfigure()

fsp_err_t R_DMAC_Reconfigure (transfer_ctrl_t *const *p_api_ctrl*, transfer_info_t * *p_info*)

Reconfigure the transfer with new transfer info.

Return values

FSP_SUCCESS	Transfer is configured and will start when trigger occurs.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_ENABLED	DMAC is not enabled. The current configuration must not be valid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

◆ R_DMAC_Reset()

fsp_err_t R_DMAC_Reset (transfer_ctrl_t *const *p_api_ctrl*, void const *volatile *p_src*, void *volatile *p_dest*, uint16_t const *num_transfers*)

Reset transfer source, destination, and number of transfers.

Return values

FSP_SUCCESS	Transfer reset successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_ENABLED	DMAC is not enabled. The current configuration must not be valid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

◆ R_DMAC_SoftwareStart()

fsp_err_t R_DMAC_SoftwareStart (transfer_ctrl_t *const *p_api_ctrl*, transfer_start_mode_t *mode*)

If the mode is TRANSFER_START_MODE_SINGLE initiate a single transfer with software. If the mode is TRANSFER_START_MODE_REPEAT continue triggering transfers until all of the transfers are completed.

Return values

FSP_SUCCESS	Transfer started written successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.
FSP_ERR_UNSUPPORTED	Handle was not configured for software activation.

◆ R_DMAC_SoftwareStop()

fsp_err_t R_DMAC_SoftwareStop (transfer_ctrl_t *const *p_api_ctrl*)

Stop software transfers if they were started with TRANSFER_START_MODE_REPEAT.

Return values

FSP_SUCCESS	Transfer stopped written successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

◆ R_DMAC_Enable()

fsp_err_t R_DMAC_Enable (transfer_ctrl_t *const *p_api_ctrl*)

Enable transfers for the configured activation source.

Return values

FSP_SUCCESS	Counter value written successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

◆ R_DMAC_Disable()

fsp_err_t R_DMAC_Disable (transfer_ctrl_t *const *p_api_ctrl*)

Disable transfers so that they are no longer triggered by the activation source.

Return values

FSP_SUCCESS	Counter value written successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

◆ R_DMAC_InfoGet()

fsp_err_t R_DMAC_InfoGet (transfer_ctrl_t *const *p_api_ctrl*, transfer_properties_t *const *p_info*)

Set driver specific information in provided pointer.

Return values

FSP_SUCCESS	Information has been written to p_info.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.
FSP_ERR_ASSERTION	An input parameter is invalid.

◆ R_DMAC_Close()

fsp_err_t R_DMAC_Close (transfer_ctrl_t *const *p_api_ctrl*)

Disable transfer and clean up internal data. Implements transfer_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DMAC_Open to initialize the control block.

4.2.19 Data Operation Circuit (r_doc)

Modules

Functions

fsp_err_t	R_DOC_Open (doc_ctrl_t *const p_api_ctrl, doc_cfg_t const *const p_cfg)
fsp_err_t	R_DOC_Close (doc_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DOC_StatusGet (doc_ctrl_t *const p_api_ctrl, doc_status_t *const p_status)
fsp_err_t	R_DOC_Read (doc_ctrl_t *const p_api_ctrl, uint32_t *p_result)
fsp_err_t	R_DOC_Write (doc_ctrl_t *const p_api_ctrl, uint32_t data)
fsp_err_t	R_DOC_CallbackSet (doc_ctrl_t *const p_api_ctrl, void(*p_callback)(doc_callback_args_t *), void const *const p_context, doc_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the DOC peripheral on RA MCUs. This module implements the DOC Interface.

Overview

Features

The DOC HAL module peripheral is used to compare, add or subtract 16-bit or 32-bit[1] data and can detect the following events:

- Comparison Mode
  - Data is equal to the configured reference data setting.
  - Data is not equal to the configured reference data setting.
  - Data is less than the configured reference data setting[2].
  - Data is greater than the configured reference data setting[2].
  - Data is inside of a configurable pair of reference data settings[2].
  - Data is outside of a configurable pair of reference data settings[2].
- Addition Mode - Overflow of an addition operation
- Subtraction Mode - Underflow of a subtraction operation

A user-defined callback can be created to inform the CPU when any of above events occur.

Note

1. *Operating on 32-bit data is not supported on all MCUs.*
2. *This comparison mode is not supported on all MCUs.*

Configuration

Build Time Configurations for r_doc

The following build time configurations are defined in fsp_cfg/r_doc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Monitoring > Data Operation Circuit (r_doc)

This module can be added to the Stacks tab via New Stack > Monitoring > Data Operation Circuit (r_doc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_doc0	Module name.
Event	MCU Specific Options		Select the event that will trigger the DOC interrupt.
Bit Width	MCU Specific Options		The bit width for DOC operations.
Reference/Initial Data	Value must be an integer greater than or equal to 0.	0	Enter Initial Value for Addition/Subtraction or enter reference value for comparison.
Additional Reference Data	Value must be an integer greater than or equal to 0.	0	Additional reference data used for Window Compare modes.
Callback	Name must be a valid C symbol	NULL	A user callback function must be provided. This will be called from the interrupt service routine (ISR) when the configured DOC event occurs.
DOC Interrupt Priority	MCU Specific Options		Select the DOC interrupt priority.

Clock Configuration

The DOC HAL module does not require a specific clock configuration.

Pin Configuration

The DOC HAL module does not require and specific pin configurations.

Usage Notes

DMAC/DTC Integration

DOC can be used with Direct Memory Access Controller (r_dmac) or Data Transfer Controller (r_dtc) to write to the input register without CPU intervention. DMAC is more useful for most DOC applications because it can be started directly from software. To write DOC input data with DTC/DMAC, set transfer_info_t::p_dest to R_DOC->DODIR.

Examples

Basic Example

This is a basic example of minimal use of the R_DOC in an application. This example shows how this driver can be used for continuous 16 bit addition operation while reading the result at every overflow event.

```
#define DOC_EXAMPLE_VALUE 0xF000
uint32_t g_callback_event_counter = 0;
/* This callback is called when DOC overflow event occurs. It is registered in
doc_cfg_t when R_DOC_Open is
 * called. */
void doc_callback (doc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_callback_event_counter++;
}
void basic_example (void)
{
 fsp_err_t err;
 /* Initialize the DOC module for addition with initial value specified in
doc_cfg_t::doc_data. */
    err = R_DOC_Open(&g_doc_ctrl, &g_doc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write data to the DOC Data Input Register and read the result of addition from
status register when an
  * interrupt occurs. */
 for (int i = 0; i < 5; i++)
    {
```

```
      err = R_DOC_Write(&g_doc_ctrl, DOC_EXAMPLE_VALUE);
     assert(FSP_SUCCESS == err);
 if (g_callback_event_counter >= 1)
     {
           uint32_t result;
 /* Read the result of the operation */
           err = R_DOC_Read(&g_doc_ctrl, &result);
     assert(FSP_SUCCESS == err);
      }
   }
}
```

Function Documentation

◆ R_DOC_Open()

fsp_err_t R_DOC_Open (doc_ctrl_t *const *p_api_ctrl*, doc_cfg_t const *const *p_cfg*)

Opens and configures the Data Operation Circuit (DOC) in comparison, addition or subtraction mode and sets initial data for addition or subtraction, or reference data for comparison.

Example:

```
 /* Initialize the DOC module for addition with initial value specified in
doc_cfg_t::doc_data. */
   err = R_DOC_Open(&g_doc_ctrl, &g_doc_cfg);
```

Return values

FSP_SUCCESS	DOC successfully configured.
FSP_ERR_ALREADY_OPEN	Module already open.
FSP_ERR_ASSERTION	One or more pointers point to NULL or callback is NULL or the interrupt vector is invalid.

◆ R_DOC_Close()

fsp_err_t R_DOC_Close (doc_ctrl_t *const *p_api_ctrl*)

Closes the module driver. Enables module stop mode.

Return values

FSP_SUCCESS	Module successfully closed.
FSP_ERR_NOT_OPEN	Driver not open.
FSP_ERR_ASSERTION	Pointer pointing to NULL.

Note

This function will disable the DOC interrupt in the NVIC.

◆ R_DOC_StatusGet()

fsp_err_t R_DOC_StatusGet (doc_ctrl_t *const *p_api_ctrl*, doc_status_t *const *p_status*)

DEPRECATED - Returns the result of addition/subtraction.

Return values

FSP_SUCCESS	Status successfully read.
FSP_ERR_NOT_OPEN	Driver not open.
FSP_ERR_ASSERTION	One or more pointers point to NULL.

◆ R_DOC_Read()

fsp_err_t R_DOC_Read (doc_ctrl_t *const *p_api_ctrl*, uint32_t * *p_result*)

Returns the result of addition/subtraction.

Example:

```
            uint32_t result;
/* Read the result of the operation */
            err = R_DOC_Read(&g_doc_ctrl, &result);
      assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Status successfully read.
FSP_ERR_NOT_OPEN	Driver not open.
FSP_ERR_ASSERTION	One or more pointers point to NULL.

◆ R_DOC_Write()

fsp_err_t R_DOC_Write (doc_ctrl_t *const *p_api_ctrl*, uint32_t *data*)

Writes to the DODIR - DOC Input Register.

Example:

```
    err = R_DOC_Write(&g_doc_ctrl, DOC_EXAMPLE_VALUE);
  assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Values successfully written to the registers.
FSP_ERR_NOT_OPEN	Driver not open.
FSP_ERR_ASSERTION	One or more pointers point to NULL.

◆ R_DOC_CallbackSet()

fsp_err_t R_DOC_CallbackSet (doc_ctrl_t *const *p_api_ctrl*, void(*)(doc_callback_args_t *) *p_callback*, void const *const *p_context*, doc_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements doc_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.20 D/AVE 2D Port Interface (r_drw)

Modules

Driver for the DRW peripheral on RA MCUs. This module is a port of D/AVE 2D.

Overview

Note

The D/AVE 2D Port Interface (D1 layer) is a HAL layer for the D/AVE D2 layer API and does not provide any interfaces to the user. Consult the TES Dave2D Driver Documentation for further information on using the D2 API.
For cross-platform compatibility purposes the D1 and D2 APIs are not bound by the FSP coding guidelines for function names and general module functionality.

Configuration

Build Time Configurations for r_drw

The following build time configurations are defined in fsp_cfg/r_drw_cfg.h:

Configuration	Options	Default	Description
Allow Indirect Mode	• Enabled • Disabled	Enabled	Enable indirect mode to allow no-copy mode for d2_adddlist (see the TES Dave2D Driver Documentation for details).
Memory Allocation	• Default • Custom	Default	Set Memory Allocation to Default to use built-in dynamic memory allocation for the D2 heap. This will use an RTOS heap if configured; otherwise, standard C malloc and free will be used. Set to Custom to define your own allocation scheme for the D2 heap. In this case, the developer will need to define the following functions: void * d1_malloc(size_t size) void d1_free(void * ptr)

Configurations for Graphics > D/AVE 2D Port Interface (r_drw)

This module can be added to the Stacks tab via New Stack > Graphics > D/AVE 2D Port Interface (r_drw).

Configuration	Options	Default	Description
D2 Device Handle Name	Name must be a valid C symbol	d2_handle0	Set the name for the d2_device handle used when calling D2 layer

		functions.
DRW Interrupt Priority	MCU Specific Options	Select the DRW_INT (display list completion) interrupt priority.

Heap Size

The D1 port layer allows the D2 driver to allocate memory as needed. There are three ways the driver can accomplish this:

1. Allocate memory using the main heap
2. Allocate memory using a heap provided by an RTOS
3. Allocate memory via user-provided functions

When the "Memory Allocation" configuration option is set to "Default" the driver will use an RTOS implementation if available and the main heap otherwise. Setting the option to "Custom" allows the user to define their own scheme using the following prototypes:

```
void * d1_malloc(size_t size);

void d1_free(void * ptr);
```

Warning

If there is no RTOS-based allocation scheme the main heap will be used. Be sure that it is enabled by setting the "Heap size (bytes)" property under RA Common on the **BSP** tab of the RA Configuration editor.

Note

It is recommended to add 32KB of additional heap space for the D2 driver until the actual usage can be determined in your application.

Interrupt

The D1 port includes one interrupt to handle various events like display list completion or bus error. This interrupt is managed internally by the D2 driver and no callback function is available.

Usage Notes

Limitations

Developers should be aware of the following limitations when using the DRW engine:

- The DRW module supports two additional interrupt types - bus error and render complete. These interrupts are not needed for D2 layer operation and thus are not supported.
- If the DRW module is stopped during rendering the render will continue once the module is started again. If this behavior is undesirable in your application it is recommended to call d2_flushframe before stopping the peripheral.

4.2.21 Data Transfer Controller (r_dtc)

Modules

Functions

fsp_err_t	R_DTC_Open (transfer_ctrl_t *const p_api_ctrl, transfer_cfg_t const *const p_cfg)
fsp_err_t	R_DTC_Reconfigure (transfer_ctrl_t *const p_api_ctrl, transfer_info_t *p_info)
fsp_err_t	R_DTC_Reset (transfer_ctrl_t *const p_api_ctrl, void const *volatile p_src, void *volatile p_dest, uint16_t const num_transfers)
fsp_err_t	R_DTC_SoftwareStart (transfer_ctrl_t *const p_api_ctrl, transfer_start_mode_t mode)
fsp_err_t	R_DTC_SoftwareStop (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DTC_Enable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DTC_Disable (transfer_ctrl_t *const p_api_ctrl)
fsp_err_t	R_DTC_InfoGet (transfer_ctrl_t *const p_api_ctrl, transfer_properties_t *const p_properties)
fsp_err_t	R_DTC_Close (transfer_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the DTC peripheral on RA MCUs. This module implements the Transfer Interface.

Overview

The Data Transfer Controller (DTC) transfers data from one memory location to another without using the CPU.

The DTC uses a RAM based vector table. Each entry in the vector table corresponds to an entry in the ISR vector table. When the DTC is triggered by an interrupt, it reads the DTC vector table, fetches the transfer information, and then executes the transfer. After the transfer is executed, the DTC writes the updated transfer info back to the location pointed to by the DTC vector table.

Features

- Supports multiple transfer modes
  - Normal transfer
  - Repeat transfer
  - Block transfer
- Chain transfers

- Address increment, decrement or fixed modes
- Can be triggered by any event that has reserved a slot in the interrupt vector table.
    - Some exceptions apply, see the Event table in the Event Numbers section of the Interrupt Controller Unit chapter of the hardware manual
- Supports 1, 2, and 4 byte data units

Configuration

Build Time Configurations for r_dtc

The following build time configurations are defined in fsp_cfg/r_dtc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Linker section to keep DTC vector table	Manual Entry	.fsp_dtc_vector_table	Section to place the DTC vector table.

Configurations for Transfer > Transfer (r_dtc)

This module can be added to the Stacks tab via New Stack > Transfer > Transfer (r_dtc).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_transfer0	Module name.
Mode	• Normal • Repeat • Block	Normal	Select the transfer mode. Select the transfer mode. Normal: One transfer per activation, transfer ends after Number of Transfers; Repeat: One transfer per activation, Repeat Area address reset after Number of Transfers, transfer repeats until stopped; Block: Number of Blocks per activation, Repeat Area address reset after Number of Transfers, transfer ends after Number of Blocks.
Transfer Size	• 1 Byte • 2 Bytes • 4 Bytes	2 Bytes	Select the transfer size.

Destination Address Mode	• Fixed • Incremented • Decremented	Fixed	Select the address mode for the destination.
Source Address Mode	• Fixed • Incremented • Decremented	Fixed	Select the address mode for the source.
Repeat Area (Unused in Normal Mode)	• Destination • Source	Source	Select the repeat area. Either the source or destination address resets to its initial value after completing Number of Transfers in Repeat or Block mode.
Destination Pointer	Manual Entry	NULL	Specify the transfer destination pointer.
Source Pointer	Manual Entry	NULL	Specify the transfer source pointer.
Interrupt Frequency	• After all transfers have completed • After each transfer	After all transfers have completed	Select to have interrupt after each transfer or after last transfer.
Number of Transfers	Value must be a non-negative integer	0	Specify the number of transfers.
Number of Blocks (Valid only in Block Mode)	Must be a valid non-negative integer with a maximum configurable value of 65536. Applicable only in Block Mode.	0	Specify the number of blocks to transfer in Block mode.
Activation Source	MCU Specific Options		Select the DTC transfer start event.

Clock Configuration

The DTC peripheral module uses ICLK as the clock source. The ICLK frequency is set by using the **Clocks** tab of the RA Configuration editor prior to a build or by using the CGC module at runtime.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Transfer Modes

The DTC Module supports three modes of operation.

- **Normal Mode** - In normal mode, a single data unit is transfered every time an interrupt is

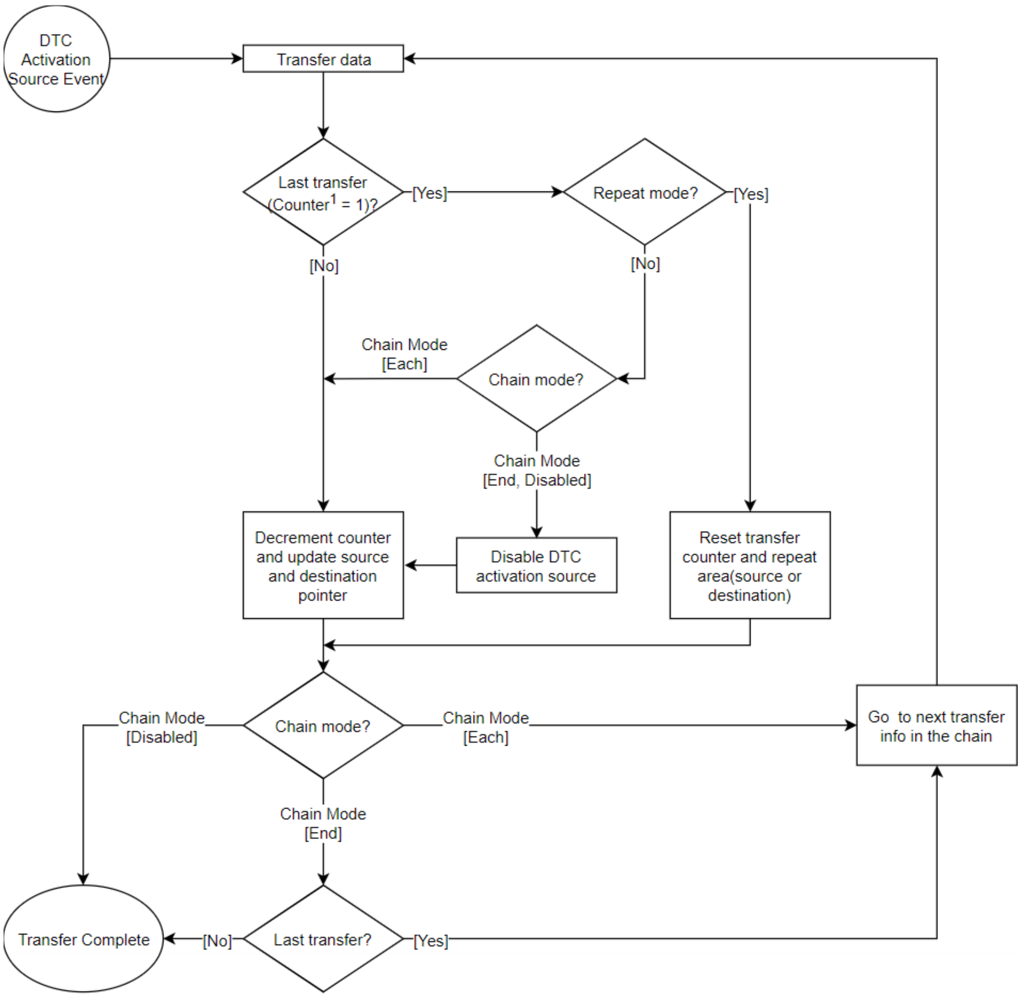
received by the DTC. A data unit can be 1-byte, 2-bytes, or 4-bytes. The source and destination addresses can be fixed, increment or decrement to the next data unit after each transfer. A 16-bit counter (length) decrements after each transfer. When the counter reaches 0, transfers will no longer be triggered by the interrupt source and the CPU can be interrupted to signal that all transfers have finished.
- **Repeat Mode** - Repeat mode works the same way as normal mode, however the length is limited to an integer in the range[1,256]. When the tranfer counter reaches 0, the counter is reset to its configured value and the repeat area (source or destination address) resets to its starting address and transfers will still be triggered by the interrupt.
- **Block Mode** - In block mode, the amount of data units transfered by each interrupt can be set to an integer in the range [1,256]. The number of blocks to transfer can also be configured to a 16-bit number. After each block transfer the repeat area (source or destination address) will reset to the original address and the other address will be incremented or decremented to the next block.

Note

1. The source and destination address of the transfer must be aligned to the configured data unit.
2. In normal mode the length can be set to [0,65535]. When the length is set to 0, than the transaction will execute 65536 transfers not 0.
3. In block mode, num_blocks can be set to [0,65535]. When the length is set to 0, than the transaction will execute 65536 transfers not 0.

Chaining Transfers

Multiple transfers can be configured for the same interrupt source by specifying an array of transfer_info_t structs instead of just passing a pointer to one. In this configuration, every transfer_info_t struct must be configured for a chain mode except for the last one. There are two types of chain mode; CHAIN_MODE_EACH and CHAIN_MODE_END. If a transfer is configured in CHAIN_MODE_EACH then it triggers the next transfer in the chain after it completes each transfer. If a transfer is configured in CHAIN_MODE_END then it triggers the next transfer in the chain after it completes its last transfer.



1. **Counter** refers to transfer_info_t::**length** in normal and repeat mode and transfer_info_t::**num_blocks** in block mode.

Figure 158: DTC Transfer Flowchart

Selecting the DTC or DMAC

The Transfer API is implemented by both DTC and the DMAC so that applications can switch between the DTC and the DMAC. When selecting between them, consider these factors:

	DTC	DMAC
Repeat Mode	• Repeats forever • Max repeat size is 256 x 4 bytes	• Configurable number of repeats • Max repeat size is 1024 x 4 bytes
Block Mode	• Max block size is 256 x 4 bytes	• Max block size is 1024 x 4 bytes
Channels	• One instance per interrupt	• MCU specific (8 channels or less)
Chained Transfers	• Supported	• Not Supported
Software Trigger	• Must use the software ELC event	• Has support for software trigger without using

		software ELC event • Supports TRANSFER_START_MODE_SINGLE and TRANSFER_START_MODE_REPEAT
Offset Address Mode	• Not supported	• Supported

Additional Considerations

- The DTC requires a moderate amount of RAM (one transfer_info_t struct per open instance + DTC_VECTOR_TABLE_SIZE).
- The DTC stores transfer information in RAM and writes back to RAM after each transfer whereas the DMAC stores all transfer information in registers.
- When transfers are configured for more than one activation source, the DTC must fetch the transfer info from RAM on each interrupt. This can cause a higher latency between transfers.
- The DTC interrupts the CPU using the activation source's IRQ. Each DMAC channel has its own IRQ.

Interrupts

The DTC and DMAC interrupts behave differently. The DTC uses the configured IELSR event IRQ as the interrupt source whereas each DMAC channel has its own IRQ.

The transfer_info_t::irq setting also behaves a little differently depending on which mode is selected.

Normal Mode

	DTC	DMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	N/A
TRANSFER_IRQ_END	Interrupt after last transfer	Interrupt after last transfer

Repeat Mode

	DTC	DMAC
TRANSFER_IRQ_EACH	Interrupt after each transfer	Interrupt after each repeat
TRANSFER_IRQ_END	Interrupt after each repeat	Interrupt after last transfer

Block Mode

	DTC	DMAC
TRANSFER_IRQ_EACH	Interrupt after each block	Interrupt after each block
TRANSFER_IRQ_END	Interrupt after last block	Interrupt after last block

Note

DTC_VECTOR_TABLE_SIZE = (ICU_NVIC_IRQ_SOURCES x 4) Bytes

Peripheral Interrupts and DTC

When an interrupt is configured to trigger DTC transfers, the peripheral ISR will trigger on the following conditions:

- Each transfer completed (transfer_info_t::irq = TRANSFER_IRQ_EACH)
- Last transfer completed (transfer_info_t::irq = TRANSFER_IRQ_END)

For example, if SCI1_RXI is configured to trigger DTC transfers and a SCI1_RXI event occurs, the interrupt will not fire until the DTC transfer is completed. If the DTC transfer_info_t::irq is configured to only interrupt on the last transfer, than no RXI interrupts will occur until the last transfer is completed.

Note

1. The DTC activation source must be enabled in the NVIC in order to trigger DTC transfers (Modules that are designed to integrate the R_DTC module will automatically handle this).
2. The DTC prioritizes activation sources by granting the smaller interrupt vector numbers higher priority. The priority of interrupts to the CPU is determined by the NVIC priority.

Low Power Modes

DTCST must be set to 0 before transitioning to any of the following:

- Module-stop state
- Software Standby mode without Snooze mode transition
- Deep Software Standby mode

Note

1. R_LPM Module stops the DTC before entering deep software standby mode and software standby without snooze mode transition.
2. For more information see 18.9 and 18.10 in the RA6M3 manual R01UH0886EJ0100.

Limitations

Developers should be aware of the following limitations when using the DTC:

- If the DTC is configured to service many different activation sources, the system could run in to performance issues due to memory contention. To address this issue, it is reccomended that the DTC vector table and transfer information be moved to their own dedicated memory area (Ex: SRAM0, SRAM1, SRAMHS). This allows memory accesses from different BUS Masters (CPU, DTC, DMAC, EDMAC and Graphics IPs) to occur in parallel.

Examples

Basic Example

This is a basic example of minimal use of the DTC in an application.

```
void dtc_minimal_example (void)
{
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_DTC_Open(&g_transfer_ctrl, &g_transfer_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
```

```
/* Enable the DTC to handle incoming transfer requests. */
    err = R_DTC_Enable(&g_transfer_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	dtc_extended_cfg_t
struct	dtc_instance_ctrl_t

Macros

#define	DTC_MAX_NORMAL_TRANSFER_LENGTH
#define	DTC_MAX_REPEAT_TRANSFER_LENGTH
#define	DTC_MAX_BLOCK_TRANSFER_LENGTH
#define	DTC_MAX_BLOCK_COUNT

Data Structure Documentation

◆ dtc_extended_cfg_t

struct dtc_extended_cfg_t

DTC transfer configuration extension. This extension is required.

Data Fields		
IRQn_Type	activation_source	Select which IRQ will trigger the transfer.

◆ dtc_instance_ctrl_t

struct dtc_instance_ctrl_t

Control block used by driver. DO NOT INITIALIZE - this structure will be initialized in transfer_api_t::open.

Macro Definition Documentation

◆ DTC_MAX_NORMAL_TRANSFER_LENGTH

#define DTC_MAX_NORMAL_TRANSFER_LENGTH

Max configurable number of transfers in NORMAL MODE

◆ DTC_MAX_REPEAT_TRANSFER_LENGTH

#define DTC_MAX_REPEAT_TRANSFER_LENGTH
Max number of transfers per repeat for REPEAT MODE

◆ DTC_MAX_BLOCK_TRANSFER_LENGTH

#define DTC_MAX_BLOCK_TRANSFER_LENGTH
Max number of transfers per block in BLOCK MODE

◆ DTC_MAX_BLOCK_COUNT

#define DTC_MAX_BLOCK_COUNT
Max configurable number of blocks to transfer in BLOCK MODE

Function Documentation

◆ R_DTC_Open()

fsp_err_t R_DTC_Open (transfer_ctrl_t *const *p_api_ctrl*, transfer_cfg_t const *const *p_cfg*)

Configure the vector table if it hasn't been configured, enable the Module and copy the pointer to the transfer info into the DTC vector table. Implements transfer_api_t::open.

Example:

```
/* Open the transfer instance with initial configuration. */
fsp_err_t err = R_DTC_Open(&g_transfer_ctrl, &g_transfer_cfg);
```

Return values

FSP_SUCCESS	Successful open. Transfer transfer info pointer copied to DTC Vector table. Module started. DTC vector table configured.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_UNSUPPORTED	Address Mode Offset is selected.
FSP_ERR_ALREADY_OPEN	The control structure is already opened.
FSP_ERR_IN_USE	The index for this IRQ in the DTC vector table is already configured.
FSP_ERR_IRQ_BSP_DISABLED	The IRQ associated with the activation source is not enabled in the BSP.

◆ R_DTC_Reconfigure()

fsp_err_t R_DTC_Reconfigure (transfer_ctrl_t *const *p_api_ctrl*, transfer_info_t * *p_info*)

Copy pointer to transfer info into the DTC vector table and enable transfer in ICU. Implements transfer_api_t::reconfigure.

Return values

FSP_SUCCESS	Transfer is configured and will start when trigger occurs.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
FSP_ERR_NOT_ENABLED	Transfer source address is NULL or is not aligned corrrectly. Transfer destination address is NULL or is not aligned corrrectly.

Note

p_info must persist until all transfers are completed.

◆ R_DTC_Reset()

fsp_err_t R_DTC_Reset (transfer_ctrl_t *const *p_api_ctrl*, void const *volatile *p_src*, void *volatile *p_dest*, uint16_t const *num_transfers*)

Reset transfer source, destination, and number of transfers. Implements transfer_api_t::reset.

Return values

FSP_SUCCESS	Transfer reset successfully (transfers are enabled).
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
FSP_ERR_NOT_ENABLED	Transfer source address is NULL or is not aligned corrrectly. Transfer destination address is NULL or is not aligned corrrectly.

◆ R_DTC_SoftwareStart()

fsp_err_t R_DTC_SoftwareStart (transfer_ctrl_t *const *p_api_ctrl*, transfer_start_mode_t *mode*)

Placeholder for unsupported softwareStart function. Implements transfer_api_t::softwareStart.

Return values

FSP_ERR_UNSUPPORTED	DTC software start is not supported.

◆ R_DTC_SoftwareStop()

fsp_err_t R_DTC_SoftwareStop (transfer_ctrl_t *const *p_api_ctrl*)

Placeholder for unsupported softwareStop function. Implements transfer_api_t::softwareStop.

Return values

FSP_ERR_UNSUPPORTED	DTC software stop is not supported.

◆ R_DTC_Enable()

fsp_err_t R_DTC_Enable (transfer_ctrl_t *const *p_api_ctrl*)

Enable transfers on this activation source. Implements transfer_api_t::enable.

Example:

```
/* Enable the DTC to handle incoming transfer requests. */
   err = R_DTC_Enable(&g_transfer_ctrl);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Transfers will be triggered by the activation source
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_UNSUPPORTED	Address Mode Offset is selected.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.

◆ R_DTC_Disable()

fsp_err_t R_DTC_Disable (transfer_ctrl_t *const *p_api_ctrl*)

Disable transfer on this activation source. Implements transfer_api_t::disable.

Return values

FSP_SUCCESS	Transfers will not occur on activation events.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
FSP_ERR_ASSERTION	An input parameter is invalid.

◆ R_DTC_InfoGet()

fsp_err_t R_DTC_InfoGet (transfer_ctrl_t *const *p_api_ctrl*, transfer_properties_t *const *p_properties*)

Provides information about this transfer. Implements transfer_api_t::infoGet.

Return values

FSP_SUCCESS	p_info updated with current instance information.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.
FSP_ERR_ASSERTION	An input parameter is invalid.

◆ R_DTC_Close()

fsp_err_t R_DTC_Close (transfer_ctrl_t *const *p_api_ctrl*)

Disables DTC activation in the ICU, then clears transfer data from the DTC vector table. Implements transfer_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_DTC_Open to initialize the control block.

4.2.22 Event Link Controller (r_elc)

Modules

Functions

fsp_err_t	R_ELC_Open (elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg)
fsp_err_t	R_ELC_Close (elc_ctrl_t *const p_ctrl)
fsp_err_t	R_ELC_SoftwareEventGenerate (elc_ctrl_t *const p_ctrl, elc_software_event_t event_number)
fsp_err_t	R_ELC_LinkSet (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal)
fsp_err_t	R_ELC_LinkBreak (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral)
fsp_err_t	R_ELC_Enable (elc_ctrl_t *const p_ctrl)
fsp_err_t	R_ELC_Disable (elc_ctrl_t *const p_ctrl)

Detailed Description

Driver for the ELC peripheral on RA MCUs. This module implements the ELC Interface.

Overview

The event link controller (ELC) uses the event requests generated by various peripheral modules as source signals to connect (link) them to different modules, allowing direct cooperation between the modules without central processing unit (CPU) intervention. The conceptual diagram below illustrates a potential setup where a pin interrupt triggers a timer which later triggers an ADC conversion and CTSU scan, while at the same time a serial communication interrupt automatically starts a data transfer. These tasks would be automatically handled without the need for polling or interrupt management.

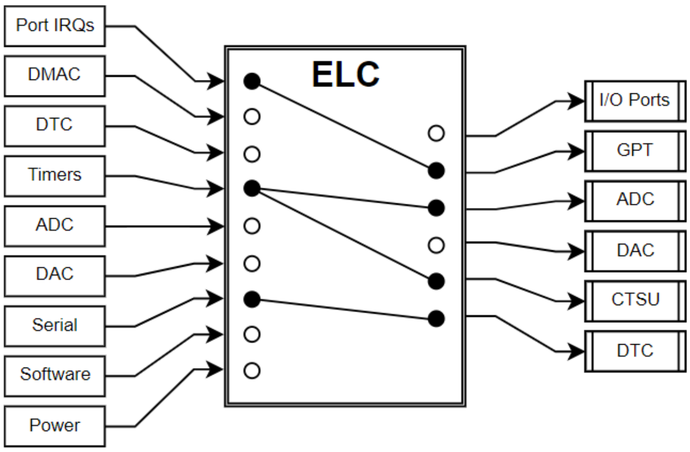


Figure 159: Event Link Controller Conceptual Diagram

In essence, the ELC is an array of multiplexers to route a wide variety of interrupt signals to a subset of peripheral functions. Events are linked by setting the multiplexer for the desired function to the desired signal (through R_ELC_LinkSet). The diagram below illustrates one peripheral output of the ELC. In this example, a conversion start is triggered for ADC0 Group A when the GPT0 counter overflows:

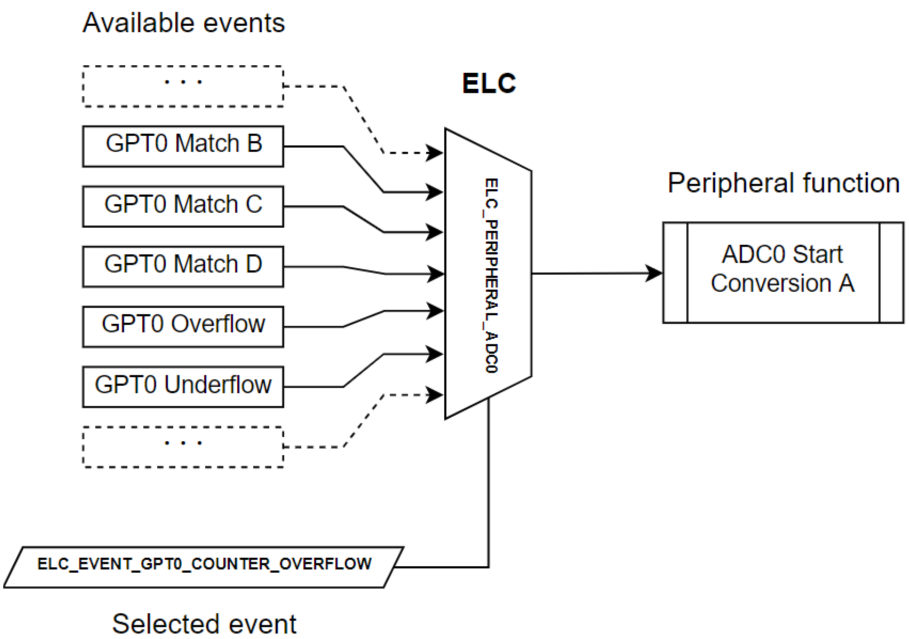


Figure 160: ELC Example

Features

The ELC HAL module can perform the following functions:

- Initialize the ELC to a pre-defined set of links
- Create an event link between two blocks
- Break an event link between two blocks
- Generate one of two software events that interrupt the CPU

- Globally enable or disable event links

A variety of functions can be activated via events, including:

- General-purpose timer (GPT) control
- ADC and DAC conversion start
- Synchronized I/O port output (ports 1-4 only)
- Capacitive touch unit (CTSU) measurement activation

Note

The available sources and peripherals may differ between devices. A full list of selectable peripherals and events is available in the User's Manual for your device.
Some peripherals have specific settings related to ELC event generation and/or reception. Details on how to enable event functionality for each peripheral are located in the usage notes for the related module(s) as well as in the User's Manual for your device.

Configuration

Note

Event links will be automatically generated based on the selections made in module properties. To view the currently linked events check the Event Links tab in the RA Configuration editor.
Calling R_ELC_Open followed by R_ELC_Enable will automatically link all events shown in the Event Links tab.

To manually link an event to a peripheral at runtime perform the following steps:

1. Configure the operation of the destination peripheral (including any configuration necessary to receive events)
2. Use R_ELC_LinkSet to set the desired event link to the peripheral
3. Use R_ELC_Enable to enable transmission of event signals
4. Configure the signaling module to output the desired event (typically an interrupt)

To disable the event, either use R_ELC_LinkBreak to clear the link for a specific event or R_ELC_Disable to globally disable event linking.

Note

The ELC module needs no pin, clocking or interrupt configuration; it is merely a mechanism to connect signals between peripherals. However, when linking I/O Ports via the ELC the relevant I/O pins need to be configured as inputs or outputs.

Build Time Configurations for r_elc

The following build time configurations are defined in fsp_cfg/r_elc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for System > Event Link Controller (r_elc)

This module can be added to the Stacks tab via New Stack > System > Event Link Controller (r_elc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	ELC instance name must be g_elc to match elc_cfg_t data structure created in elc_data.c	g_elc	Module name. Fixed to g_elc.

Usage Notes

Limitations

Developers should be aware of the following limitations when using the ELC:

- To link events it is necessary for the ELC and the related modules to be enabled. The ELC cannot operate if the related modules are in the module stop state or the MCU is in a low power consumption mode for which the module is stopped.
- If two modules are linked across clock domains there may be a 1 to 2 cycle delay between event signaling and reception. The delay timing is based on the frequency of the slowest clock.

Examples

Basic Example

Below is a basic example of minimal use of event linking in an application.

```
/* This struct is automatically generated based on the events configured by
peripherals in the RA Configuration editor. */
static const elc_cfg_t g_elc_cfg =
{
    .link[ELC_PERIPHERAL_GPT_A]    = ELC_EVENT_ICU_IRQ0,
    .link[ELC_PERIPHERAL_IOPORT1] = ELC_EVENT_GPT4_COUNTER_OVERFLOW
};
void elc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the software and sets the links defined in the control structure. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Create or modify a link between a peripheral function and an event source. */
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0,
ELC_EVENT_GPT4_COUNTER_OVERFLOW);
```

```
    assert(FSP_SUCCESS == err);
 /* Globally enable event linking in the ELC. */
    err = R_ELC_Enable(&g_elc_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Software-Generated Events

This example demonstrates how to use a software-generated event to signal a peripheral. This can be useful when the desired event source is not supported by the ELC hardware.

```
/* Interrupt handler for peripheral event not supported by the ELC */
void peripheral_isr (void)
{
 fsp_err_t err;
 /* Generate an event signal through software to the linked peripheral. */
    err = R_ELC_SoftwareEventGenerate(&g_elc_ctrl, ELC_SOFTWARE_EVENT_0);
    assert(FSP_SUCCESS == err);
}
void elc_software_event (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the module. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Link ADC0 conversion start to software event 0. */
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0,
ELC_EVENT_ELC_SOFTWARE_EVENT_0);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Application code here. */
    }
}
```

Data Structures

struct	elc_instance_ctrl_t

Data Structure Documentation

◆ elc_instance_ctrl_t

struct elc_instance_ctrl_t
ELC private control block. DO NOT MODIFY. Initialization occurs when R_ELC_Open() is called.

Function Documentation

◆ R_ELC_Open()

fsp_err_t R_ELC_Open (elc_ctrl_t *const *p_ctrl*, elc_cfg_t const *const *p_cfg*)

Initialize all the links in the Event Link Controller. Implements elc_api_t::open

The configuration structure passed in to this function includes links for every event source included in the ELC and sets them all at once. To set or clear an individual link use R_ELC_LinkSet and R_ELC_LinkBreak respectively.

Example:

```
/* Initializes the software and sets the links defined in the control structure. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful
FSP_ERR_ASSERTION	p_ctrl or p_cfg was NULL
FSP_ERR_ALREADY_OPEN	The module is currently open

◆ R_ELC_Close()

fsp_err_t R_ELC_Close (elc_ctrl_t *const *p_ctrl*)

Globally disable ELC linking. Implements elc_api_t::close

Return values

FSP_SUCCESS	The ELC was successfully disabled
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

◆ R_ELC_SoftwareEventGenerate()

fsp_err_t R_ELC_SoftwareEventGenerate (elc_ctrl_t *const *p_ctrl*, elc_software_event_t *event_number*)

Generate a software event in the Event Link Controller. Implements elc_api_t::softwareEventGenerate

Example:

```
/* Generate an event signal through software to the linked peripheral. */
    err = R_ELC_SoftwareEventGenerate(&g_elc_ctrl, ELC_SOFTWARE_EVENT_0);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Initialization was successful
FSP_ERR_ASSERTION	Invalid event number or p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

◆ R_ELC_LinkSet()

fsp_err_t R_ELC_LinkSet (elc_ctrl_t *const *p_ctrl*, elc_peripheral_t *peripheral*, elc_event_t *signal*)

Create a single event link. Implements elc_api_t::linkSet

Example:

```
/* Create or modify a link between a peripheral function and an event source. */
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0,
ELC_EVENT_GPT4_COUNTER_OVERFLOW);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Initialization was successful
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

◆ R_ELC_LinkBreak()

fsp_err_t R_ELC_LinkBreak (elc_ctrl_t *const *p_ctrl*, elc_peripheral_t *peripheral*)

Break an event link. Implements elc_api_t::linkBreak

Return values

FSP_SUCCESS	Event link broken
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

◆ R_ELC_Enable()

fsp_err_t R_ELC_Enable (elc_ctrl_t *const *p_ctrl*)

Enable the operation of the Event Link Controller. Implements elc_api_t::enable

Return values

FSP_SUCCESS	ELC enabled.
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

◆ R_ELC_Disable()

fsp_err_t R_ELC_Disable (elc_ctrl_t *const *p_ctrl*)

Disable the operation of the Event Link Controller. Implements elc_api_t::disable

Return values

FSP_SUCCESS	ELC disabled.
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

4.2.23 Ethernet (r_ether)

Modules

Functions

fsp_err_t	R_ETHER_Open (ether_ctrl_t *const p_ctrl, ether_cfg_t const *const

	p_cfg) After ETHERC, EDMAC and PHY-LSI are reset in software, an auto negotiation of PHY-LSI is begun. Afterwards, the link signal change interrupt is permitted. Implements ether_api_t::open. More...
fsp_err_t	R_ETHER_Close (ether_ctrl_t *const p_ctrl) Disables interrupts. Removes power and releases hardware lock. Implements ether_api_t::close. More...
fsp_err_t	R_ETHER_Read (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t *const length_bytes) Receive Ethernet frame. Receives data to the location specified by the pointer to the receive buffer. In zero copy mode, the address of the receive buffer is returned. In non zero copy mode, the received data in the internal buffer is copied to the pointer passed by the argument. Implements ether_api_t::read. More...
fsp_err_t	R_ETHER_BufferRelease (ether_ctrl_t *const p_ctrl) Move to the next buffer in the circular receive buffer list. Implements ether_api_t::bufferRelease. More...
fsp_err_t	R_ETHER_RxBufferUpdate (ether_ctrl_t *const p_ctrl, void *const p_buffer) Change the buffer pointer of the current rx buffer descriptor. Implements ether_api_t::rxBufferUpdate. More...
fsp_err_t	R_ETHER_Write (ether_ctrl_t *const p_ctrl, void *const p_buffer, uint32_t const frame_length) Transmit Ethernet frame. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length. In the non zero copy mode, transmits data after being copied to the internal buffer. Implements ether_api_t::write. More...
fsp_err_t	R_ETHER_LinkProcess (ether_ctrl_t *const p_ctrl) The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements ether_api_t::linkProcess. More...
fsp_err_t	R_ETHER_WakeOnLANEnable (ether_ctrl_t *const p_ctrl) The setting of ETHERC is changed from normal sending and

	receiving mode to magic packet detection mode. Implements ether_api_t::wakeOnLANEnable. More...
fsp_err_t	R_ETHER_TxStatusGet (ether_ctrl_t *const p_ctrl, void *const p_buffer_address)

Detailed Description

Driver for the Ethernet peripheral on RA MCUs. This module implements the Ethernet Interface.

Overview

This module performs Ethernet frame transmission and reception using an Ethernet controller and an Ethernet DMA controller.

Features

The Ethernet module supports the following features:

- Transmit/receive processing
- Optional zero-copy buffering
- Callback function with returned event code
- Magic packet detection mode support
- Auto negotiation support
- Flow control support
- Multicast filtering support
- Broadcast filtering support
- Promiscuous mode support

Configuration

Build Time Configurations for r_ether

The following build time configurations are defined in fsp_cfg/r_ether_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
ET0_LINKSTA Pin Status Flag	• Fall -> Rise • Rise -> Fall	Fall -> Rise	Specify the polarity of the link signal output by the PHY-LSI. When 0 is specified, link-up and link-down correspond respectively to the fall and rise of the LINKSTA signal. When 1 is specified, link-up and link-down correspond respectively to the rise

			and fall of the LINKSTA signal.
Link Signal Change Flag	• Unused • Used	Unused	Use LINKSTA signal for detect link status changes 0 = unused (use PHY-LSI status register) 1 = use (use LINKSTA signal)

Configurations for Networking > Ethernet (r_ether)

This module can be added to the Stacks tab via New Stack > Networking > Ethernet (r_ether).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_ether0	Module name.
General > Channel	0	0	Select the ether channel number.
General > MAC address	Must be a valid MAC address	00:11:22:33:44:55	MAC address of this channel.
General > Zero-copy Mode	• Disable • Enable	Disable	Enable or disable zero-copy mode.
General > Flow control functionality	• Disable • Enable	Disable	Enable or disable flow control.
Filters > Multicast Mode	• Disable • Enable	Enable	Enable or disable multicast frame reception.
Filters > Promiscuous Mode	• Disable • Enable	Disable	Enable this option to receive packets addressed to other NICs.
Filters > Broadcast filter	Must be a valid non-negative integer with maximum configurable value of 65535.	0	Limit of the number of broadcast frames received continuously
Buffers > Number of TX buffer	Must be an integer from 1 to 8	1	Number of transmit buffers
Buffers > Number of RX buffer	Must be an integer from 1 to 8	1	Number of receive buffers
Buffers > Allocate RX buffer	• Disable • Enable	Enable	Allocates the RX buffer when generating the configuration structure
Buffers > Buffer size	Must be at least 1514 which is the maximum Ethernet frame size.	1514	Size of Ethernet buffer
Buffers > Padding size	• Disable	Disable	The padding size that

	• 1 Byte • 2 Bytes • 3 Bytes		is automatically inserted into the received packets
Buffers > Padding offset	Must be less than 64 bytes.	0	The offset into a receive buffer to insert padding bytes.
Interrupts > Interrupt priority	MCU Specific Options		Select the EDMAC interrupt priority.
Interrupts > Callback	Name must be a valid C symbol	NULL	Callback provided when an ISR occurs

Interrupt Configuration

The first R_ETHER_Open function call sets EINT interrupts. The user could provide callback function which would be invoked when EINT interrupt handler has been completed. The callback arguments will contain information about a channel number, the ETHERC and EDMAC status, the event code, and a pointer to the user defined context.

Callback Configuration

The user could provide callback function which would be invoked when either a magic packet or a link signal change is detected. When the callback function is called, a variable in which the channel number for which the detection occurred and a constant shown in Table 2.4 are stored is passed as an argument. If the value of this argument is to be used outside the callback function, its value should be copied into, for example, a global variable.

Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

MCU Group	Peripheral Clock
RA6E1	PCLKA
RA6M2	PCLKA
RA6M3	PCLKA
RA6M4	PCLKA
RA6M5	PCLKA

Note

1. *When using ETHERC, the PCLKA frequency is in the range 12.5 MHz <= PCLKA <= 120 MHz.*
2. *When using ETHERC, PCLKA = ICLK.*

Pin Configuration

To use the Ethernet module, input/output signals of the peripheral function have to be allocated to pins with the multi-function pin controller (MPC). Please perform the pin setting before calling the R_ETHER_Open function.

Usage Notes

Ethernet Frame Format

The Ethernet module supports the Ethernet II/IEEE 802.3 frame format.

Frame Format for Data Transmission and Reception

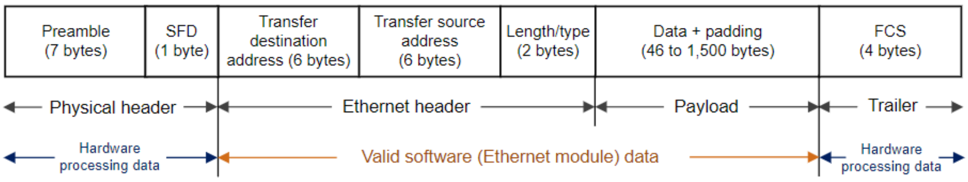


Figure 161: Frame Format Image

The preamble and SFD signal the start of an Ethernet frame. The FCS contains the CRC of the Ethernet frame and is calculated on the transmitting side. When data is received the CRC value of the frame is calculated in hardware, and the Ethernet frame is discarded if the values do not match. When the hardware determines that the data is normal, the valid range of receive data is: (transmission destination address) + (transmission source address) + (length/type) + (data).

PAUSE Frame Format

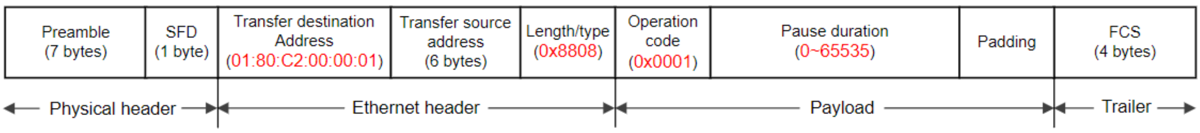


Figure 162: Pause Frame Format Image

The transmission destination address is specified as 01:80:C2:00:00:01 (a multicast address reserved for PAUSE frames). At the start of the payload the length/type is specified as 0x8808 and the operation code as 0x0001. The pause duration in the payload is specified by the value of the automatic PAUSE (AP) bits in the automatic PAUSE frame setting register (APR), or the manual PAUSE time setting (MP) bits in the manual PAUSE frame setting register (MPR).

Magic Packet Frame Format

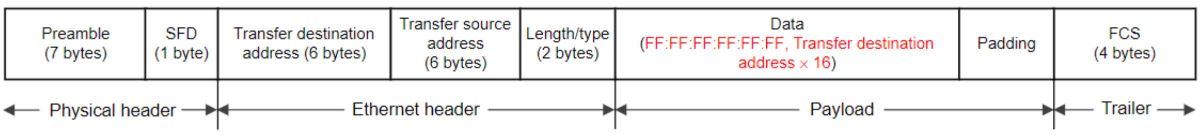


Figure 163: Magic Packet Frame Format Image

In a Magic Packet, the value FF:FF:FF:FF:FF:FF followed by the transmission destination address repeated 16 times is inserted somewhere in the Ethernet frame data.

Limitations

Memory alignment limitation for Ethernet buffer

The Ethernet Driver has several alignment constraints:

- 16-byte alignment for the descriptor
- 32-byte aligned read buffer for R_ETHER_RxBufferUpdate when zero copy mode is enabled

Functional limitations in TrustZone Security Extensions

The Ethernet Driver has several security constraints:

MCU	Has Security Extension	Support Flat project	Support TZ project	
			Secure	Non-Secure
RA6M2	-	x	-	-
RA6M3	-	x	-	-
RA6M4	- *1	x	-	x
RA6M5	- *1	x	-	x

Note

1. ETHERC/EDMAC is always Non-secure peripheral in this MCU.

Examples

ETHER Basic Example

This is a basic example of minimal use of the ETHER in an application.

Note

In this example zero-copy mode is disabled and there are no restrictions on buffer alignment.

```
#define ETHER_EXAMPLE_MAXIMUM_ETHERNET_FRAME_SIZE (1514)
#define ETHER_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE (60)
#define ETHER_EXAMPLE_SOURCE_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79, 0x01
#define ETHER_EXAMPLE_DESTINATION_MAC_ADDRESS 0x74, 0x90, 0x50, 0x00, 0x79, 0x02
#define ETHER_EXAMPLE_FRAME_TYPE 0x00, 0x2E
#define ETHER_EXAMPLE_PAYLOAD 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \

/* Receive data buffer */
static uint8_t gp_read_buffer[ETHER_EXAMPLE_MAXIMUM_ETHERNET_FRAME_SIZE] = {0};
/* Transmit data buffer */
```

```
static uint8_t gp_send_data[ETHER_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE] =
{
    ETHER_EXAMPLE_DESTINATION_MAC_ADDRESS, /* Destination MAC address */
    ETHER_EXAMPLE_SOURCE_MAC_ADDRESS,      /* Source MAC address */
    ETHER_EXAMPLE_FRAME_TYPE,             /* Type field */
    ETHER_EXAMPLE_PAYLOAD                  /* Payload value (46byte) */
};
void ether_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Source MAC Address */
 static uint8_t mac_address_source[6] = {ETHER_EXAMPLE_SOURCE_MAC_ADDRESS};
    uint32_t read_data_size = 0;
    g_ether0_cfg.p_mac_address = mac_address_source;
 /* Open the ether instance with initial configuration. */
    err = R_ETHER_Open(&g_ether0_ctrl, &g_ether0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 do
    {
 /* When the Ethernet link status read from the PHY-LSI Basic Status register is link-
up,
  * Initializes the module and make auto negotiation. */
       err = R_ETHER_LinkProcess(&g_ether0_ctrl);
    } while (FSP_SUCCESS != err);
 /* Transmission is non-blocking. */
 /* User data copy to internal buffer and is transferred by DMA in the background. */
    err = R_ETHER_Write(&g_ether0_ctrl, (void *) gp_send_data, sizeof(gp_send_data));
    assert(FSP_SUCCESS == err);
 /* received data copy to user buffer from internal buffer. */
    err = R_ETHER_Read(&g_ether0_ctrl, (void *) gp_read_buffer, &read_data_size);
    assert(FSP_SUCCESS == err);
 /* Disable transmission and receive function and close the ether instance. */
 R_ETHER_Close(&g_ether0_ctrl);
```

```
}
```

ETHER Advanced Example

The example demonstrates using send and receive function in zero copy mode. Transmit buffers must be 32-byte aligned and the receive buffer must be released once its contents have been used.

```
#define ETHER_EXAMPLE_FLAG_ON (1U)
#define ETHER_EXAMPLE_FLAG_OFF (0U)
#define ETHER_EXAMPLE_ETHER_ISR_EE_FR_MASK (1UL << 18)
#define ETHER_EXAMPLE_ETHER_ISR_EE_TC_MASK (1UL << 21)
#define ETHER_EXAMPLE_ETHER_ISR_EC_MPD_MASK (1UL << 1)
#define ETHER_EXAMPLE_ALIGNMENT_32_BYTE (32)
static volatile uint32_t g_example_receive_complete = 0;
static volatile uint32_t g_example_transfer_complete = 0;
static volatile uint32_t g_example_magic_packet_done = 0;
static uint8_t gp_send_data_internal[ETHER_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE] =
{
    ETHER_EXAMPLE_DESTINATION_MAC_ADDRESS, /* Destination MAC address */
    ETHER_EXAMPLE_SOURCE_MAC_ADDRESS,      /* Source MAC address */
    ETHER_EXAMPLE_FRAME_TYPE,             /* Type field */
    ETHER_EXAMPLE_PAYLOAD                  /* Payload value (46byte) */
};
void ether_example_callback (ether_callback_args_t * p_args) {
 switch (p_args->event)
    {
 case ETHER_EVENT_INTERRUPT:
      {
 if (ETHER_EXAMPLE_ETHER_ISR_EC_MPD_MASK == (p_args->status_ecsr &
ETHER_EXAMPLE_ETHER_ISR_EC_MPD_MASK))
      {
                g_example_magic_packet_done = ETHER_EXAMPLE_FLAG_ON;
            }
 if (ETHER_EXAMPLE_ETHER_ISR_EE_TC_MASK == (p_args->status_eesr &
ETHER_EXAMPLE_ETHER_ISR_EE_TC_MASK))
```

```
      {
                g_example_transfer_complete = ETHER_EXAMPLE_FLAG_ON;
            }
 if (ETHER_EXAMPLE_ETHER_ISR_EE_FR_MASK == (p_args->status_eesr &
ETHER_EXAMPLE_ETHER_ISR_EE_FR_MASK))
      {
                g_example_receive_complete = ETHER_EXAMPLE_FLAG_ON;
            }
 break;
       }
 default:
      {
       }
    }
}
void ether_advanced_use_internal_buffer_example (void) {
 fsp_err_t err = FSP_SUCCESS;
 /* Source MAC Address */
 static uint8_t mac_address_source[6] = {ETHER_EXAMPLE_SOURCE_MAC_ADDRESS};
 static uint8_t * p_read_buffer_nocopy;
    uint32_t         read_data_size = 0;
    g_ether0_cfg.p_mac_address = mac_address_source;
    g_ether0_cfg.zerocopy   = ETHER_ZEROCOPY_ENABLE;
    g_ether0_cfg.p_callback = (void (*)(ether_callback_args_t
*))ether_example_callback;
 /* Open the ether instance with initial configuration. */
    err = R_ETHER_Open(&g_ether0_ctrl, &g_ether0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 do
    {
 /* When the Ethernet link status read from the PHY-LSI Basic Status register is link-
up,
  * Initializes the module and make auto negotiation. */
```

```
      err = R_ETHER_LinkProcess(&g_ether0_ctrl);
    } while (FSP_SUCCESS != err);
 /* Set user buffer to TX descriptor and enable transmission. */
    err = R_ETHER_Write(&g_ether0_ctrl, (void *) gp_send_data_internal, sizeof
(gp_send_data_internal));
 if (FSP_SUCCESS == err)
    {
 /* Wait for the transmission to complete. */
 /* Data array should not change in zero copy mode until transfer complete. */
 while (ETHER_EXAMPLE_FLAG_ON != g_example_transfer_complete)
      {
            ;
       }
    }
 /* Get receive buffer from RX descriptor. */
    err = R_ETHER_Read(&g_ether0_ctrl, (void *) &p_read_buffer_nocopy,
&read_data_size);
    assert(FSP_SUCCESS == err);
 /* Process received data here */
 /* Release receive buffer to RX descriptor. */
    err = R_ETHER_BufferRelease(&g_ether0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Disable transmission and receive function and close the ether instance. */
 R_ETHER_Close(&g_ether0_ctrl);
}
```

```
#define ETHER_EXAMPLE_ALIGNMENT_32_BYTE (32)
#define ETHER_EXAMPLE_ETHERNET_FRAME_PAYLOAD_OFFSET (14)
/* The data buffer must be 32-byte aligned when using zero copy mode. */
static uint8_t gp_send_data_external[ETHER_EXAMPLE_TRANSMIT_ETHERNET_FRAME_SIZE] =
{
    ETHER_EXAMPLE_DESTINATION_MAC_ADDRESS, /* Destination MAC address */
    ETHER_EXAMPLE_SOURCE_MAC_ADDRESS,      /* Source MAC address */
    ETHER_EXAMPLE_FRAME_TYPE,             /* Type field */
```

```
    ETHER_EXAMPLE_PAYLOAD                  /* Payload value (46byte) */
};
typedef struct st_buffer_node
{
    uint8_t               * p_buffer;
 struct st_buffer_node * p_next;
} buffer_node_t;
void ether_advanced_use_external_buffer_example (void) {
 fsp_err_t err = FSP_SUCCESS;
 /* Source MAC Address */
    uint8_t mac_address_source[6] = {ETHER_EXAMPLE_SOURCE_MAC_ADDRESS};
    uint8_t       * p_tx_buffer          = NULL;
    uint8_t       * p_rx_buffer          = NULL;
    uint8_t       * p_rx_allocate_buffer = NULL;
    uint8_t       * p_tx_last_sent_buffer = NULL;
    buffer_node_t * p_tx_buffer_head;
    buffer_node_t * p_tx_buffer_tail;
    uint32_t read_data_size = 0;
    uint8_t i;
    g_ether0_cfg.p_mac_address = mac_address_source;
    g_ether0_cfg.zerocopy         = ETHER_ZEROCOPY_ENABLE;
    g_ether0_cfg.pp_ether_buffers = NULL;
 /* Create ring buffer structure to manage transmit buffer.*/
    p_tx_buffer_head = (buffer_node_t *) malloc(sizeof(buffer_node_t));
    p_tx_buffer_tail = p_tx_buffer_head;
 for (i = 0; i < g_ether0_cfg.num_tx_descriptors - 1; i++)
    {
       p_tx_buffer_tail->p_buffer = NULL;
       p_tx_buffer_tail->p_next   = (buffer_node_t *) malloc(sizeof(buffer_node_t));
       p_tx_buffer_tail          = p_tx_buffer_tail->p_next;
    }
    p_tx_buffer_tail->p_buffer = NULL;
    p_tx_buffer_tail->p_next   = p_tx_buffer_head;
 /* Open the ether instance with initial configuration. */
```

```
    err = R_ETHER_Open(&g_ether0_ctrl, &g_ether0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    i = 0;
 /* Initialize receive buffer in Ethernet driver. */
 while (i < g_ether0_cfg.num_rx_descriptors)
    {
 if (posix_memalign((void **) &p_rx_allocate_buffer,
                           ETHER_EXAMPLE_ALIGNMENT_32_BYTE,
                           g_ether0_cfg.ether_buffer_size * sizeof(char)))
      {
 /* Set receive buffer to Ethernet driver. */
            err = R_ETHER_RxBufferUpdate(&g_ether0_ctrl, (void *)
p_rx_allocate_buffer);
 if (FSP_SUCCESS == err)
      {
                i++;
            }
       }
 else
      {
            assert(0);
       }
    }
 do
    {
 /* When the Ethernet link status read from the PHY-LSI Basic Status register is link-
up,
  * Initializes the module and make auto negotiation. */
       err = R_ETHER_LinkProcess(&g_ether0_ctrl);
    } while (FSP_SUCCESS != err);
 while (1)
    {
 if (NULL == p_tx_buffer_tail->p_buffer)
```

```
     {
 /* Allocate memory to transmit buffer */
            p_tx_buffer = (uint8_t *) malloc(sizeof(gp_send_data_external));
 /* Process transmit data here. */
            memcpy(p_tx_buffer, gp_send_data_external, sizeof
(gp_send_data_external));
            gp_send_data_external[ETHER_EXAMPLE_ETHERNET_FRAME_PAYLOAD_OFFSET]++;
 /* Set user buffer to TX descriptor and enable transmission. */
            err = R_ETHER_Write(&g_ether0_ctrl, (void *) gp_send_data_external,
sizeof(gp_send_data_external));
 /* Register transmit buffer to ring buffer. */
 if (FSP_SUCCESS == err)
     {
                p_tx_buffer_tail->p_buffer = p_tx_buffer;
         p_tx_buffer_tail          = p_tx_buffer_tail->p_next;
            }
 else
     {
 /* Release transmit buffer. */
                free(p_tx_buffer);
            }
      }
 /* Get receive buffer from RX descriptor. */
      err = R_ETHER_Read(&g_ether0_ctrl, (void *) &p_rx_buffer, &read_data_size);
 if (FSP_SUCCESS == err)
     {
 /* Allocate new receive buffer and update receive buffer to RX descriptor. */
 if (0 == posix_memalign((void **) &p_rx_allocate_buffer,
                             ETHER_EXAMPLE_ALIGNMENT_32_BYTE,
                             g_ether0_cfg.ether_buffer_size * sizeof(char)))
     {
 R_ETHER_RxBufferUpdate(&g_ether0_ctrl, p_rx_allocate_buffer);
            }
 else
```

```
      {
              assert(0);
            }
 /* Process received data here. */
 /* Release receive buffer. */
            free(p_rx_buffer);
       }
 /* Release all transmitted buffer from the ring buffer. */
 if (FSP_SUCCESS == R_ETHER_TxStatusGet(&g_ether0_ctrl, (void *)
&p_tx_last_sent_buffer))
      {
            buffer_node_t * p_tx_buffer_current = p_tx_buffer_head;
 for (i = 0; i < g_ether0_cfg.num_tx_descriptors; i++)
      {
 if (p_tx_last_sent_buffer == p_tx_buffer_current->p_buffer)
      {
 do
      {
                        free(p_tx_buffer_head->p_buffer);
                        p_tx_buffer_head->p_buffer = NULL;
                        p_tx_buffer_head          = p_tx_buffer_head->p_next;
              } while (p_tx_buffer_head != p_tx_buffer_current);
                   free(p_tx_buffer_head->p_buffer);
                   p_tx_buffer_head->p_buffer = NULL;
 break;
              }
                p_tx_buffer_current = p_tx_buffer_current->p_next;
            }
       }
    }
}
```

Data Structures

struct	ether_instance_ctrl_t

Enumerations

enum	ether_previous_link_status_t
enum	ether_link_change_t
enum	ether_magic_packet_t
enum	ether_link_establish_status_t

Data Structure Documentation

◆ ether_instance_ctrl_t

struct ether_instance_ctrl_t

ETHER control block. DO NOT INITIALIZE. Initialization occurs when ether_api_t::open is called.

Data Fields

uint32_t	open	Used to determine if the channel is configured.
ether_cfg_t const *	p_ether_cfg	Pointer to initial configurations.
ether_instance_descriptor_t *	p_rx_descriptor	Pointer to the currently referenced transmit descriptor.
ether_instance_descriptor_t *	p_tx_descriptor	Pointer to the currently referenced receive descriptor.
void *	p_reg_etherc	Base register of ethernet controller for this channel.
void *	p_reg_edmac	Base register of EDMA controller for this channel.
ether_previous_link_status_t	previous_link_status	Previous link status.
ether_link_change_t	link_change	status of link change
ether_magic_packet_t	magic_packet	status of magic packet detection
ether_link_establish_status_t	link_establish_status	Current Link status.

Enumeration Type Documentation

◆ ether_previous_link_status_t

enum ether_previous_link_status_t

Enumerator

ETHER_PREVIOUS_LINK_STATUS_DOWN	Previous link status is down.
ETHER_PREVIOUS_LINK_STATUS_UP	Previous link status is up.

◆ ether_link_change_t

enum ether_link_change_t

Enumerator	
ETHER_LINK_CHANGE_NO_CHANGE	Link status is no change.
ETHER_LINK_CHANGE_LINK_DOWN	Link status changes to down.
ETHER_LINK_CHANGE_LINK_UP	Link status changes to up.

◆ ether_magic_packet_t

enum ether_magic_packet_t

Enumerator	
ETHER_MAGIC_PACKET_NOT_DETECTED	Magic packet is not detected.
ETHER_MAGIC_PACKET_DETECTED	Magic packet is detected.

◆ ether_link_establish_status_t

enum ether_link_establish_status_t

Enumerator	
ETHER_LINK_ESTABLISH_STATUS_DOWN	Link establish status is down.
ETHER_LINK_ESTABLISH_STATUS_UP	Link establish status is up.

Function Documentation

◆ R_ETHER_Open()

fsp_err_t R_ETHER_Open (ether_ctrl_t *const *p_ctrl*, ether_cfg_t const *const *p_cfg*)

After ETHERC, EDMAC and PHY-LSI are reset in software, an auto negotiation of PHY-LSI is begun. Afterwards, the link signal change interrupt is permitted. Implements ether_api_t::open.

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ASSERTION	Pointer to ETHER control block or configuration structure is NULL.
FSP_ERR_ALREADY_OPEN	Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure.
FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION	Initialization of PHY-LSI failed.
FSP_ERR_INVALID_CHANNEL	Invalid channel number is given.
FSP_ERR_INVALID_POINTER	Pointer to MAC address is NULL.
FSP_ERR_INVALID_ARGUMENT	Interrupt is not enabled.
FSP_ERR_ETHER_PHY_ERROR_LINK	Initialization of PHY-LSI failed.

◆ R_ETHER_Close()

fsp_err_t R_ETHER_Close (ether_ctrl_t *const *p_ctrl*)

Disables interrupts. Removes power and releases hardware lock. Implements ether_api_t::close.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_ETHER_Read()

fsp_err_t R_ETHER_Read (ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*, uint32_t *const *length_bytes*)

Receive Ethernet frame. Receives data to the location specified by the pointer to the receive buffer. In zero copy mode, the address of the receive buffer is returned. In non zero copy mode, the received data in the internal buffer is copied to the pointer passed by the argument. Implements ether_api_t::read.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_ETHER_ERROR_NO_DATA	There is no data in receive buffer.
FSP_ERR_ETHER_ERROR_LINK	Auto-negotiation is not completed, and reception is not enabled.
FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE	As a Magic Packet is being detected, transmission and reception is not enabled.
FSP_ERR_ETHER_ERROR_FILTERING	Multicast Frame filter is enable, and Multicast Address Frame is received.
FSP_ERR_INVALID_POINTER	Value of the pointer is NULL.

◆ R_ETHER_BufferRelease()

fsp_err_t R_ETHER_BufferRelease (ether_ctrl_t *const *p_ctrl*)

Move to the next buffer in the circular receive buffer list. Implements ether_api_t::bufferRelease.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_ETHER_ERROR_LINK	Auto-negotiation is not completed, and reception is not enabled.
FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE	As a Magic Packet is being detected, transmission and reception is not enabled.

◆ R_ETHER_RxBufferUpdate()

fsp_err_t R_ETHER_RxBufferUpdate (ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*)

Change the buffer pointer of the current rx buffer descriptor. Implements ether_api_t::rxBufferUpdate.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	A pointer argument is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	The pointer of buffer is NULL or not aligned on a 32-bit boundary.
FSP_ERR_INVALID_MODE	Driver is configured to non zero copy mode.
FSP_ERR_ETHER_RECEIVE_BUFFER_ACTIVE	All descriptor is active.

◆ R_ETHER_Write()

fsp_err_t R_ETHER_Write (ether_ctrl_t *const *p_ctrl*, void *const *p_buffer*, uint32_t const *frame_length*)

Transmit Ethernet frame. Transmits data from the location specified by the pointer to the transmit buffer, with the data size equal to the specified frame length. In the non zero copy mode, transmits data after being copied to the internal buffer. Implements ether_api_t::write.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_ETHER_ERROR_LINK	Auto-negotiation is not completed, and reception is not enabled.
FSP_ERR_ETHER_ERROR_MAGIC_PACKET_MODE	As a Magic Packet is being detected, transmission and reception is not enabled.
FSP_ERR_ETHER_ERROR_TRANSMIT_BUFFER_FULL	Transmit buffer is not empty.
FSP_ERR_INVALID_POINTER	Value of the pointer is NULL.
FSP_ERR_INVALID_ARGUMENT	Value of the send frame size is out of range.

◆ R_ETHER_LinkProcess()

fsp_err_t R_ETHER_LinkProcess (ether_ctrl_t *const *p_ctrl*)

The Link up processing, the Link down processing, and the magic packet detection processing are executed. Implements ether_api_t::linkProcess.

Return values

FSP_SUCCESS	Link is up.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_ETHER_ERROR_LINK	Link is down.
FSP_ERR_ETHER_ERROR_PHY_COMMUNICATION	When reopening the PHY interface initialization of the PHY-LSI failed.
FSP_ERR_ALREADY_OPEN	When reopening the PHY interface it was already opened.
FSP_ERR_INVALID_CHANNEL	When reopening the PHY interface an invalid channel was passed.
FSP_ERR_INVALID_POINTER	When reopening the PHY interface the MAC address pointer was NULL.
FSP_ERR_INVALID_ARGUMENT	When reopening the PHY interface the interrupt was not enabled.
FSP_ERR_ETHER_PHY_ERROR_LINK	Initialization of the PHY-LSI failed.

◆ R_ETHER_WakeOnLANEnable()

fsp_err_t R_ETHER_WakeOnLANEnable (ether_ctrl_t *const *p_ctrl*)

The setting of ETHERC is changed from normal sending and receiving mode to magic packet detection mode. Implements ether_api_t::wakeOnLANEnable.

Return values

FSP_SUCCESS	Processing completed successfully.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_ETHER_ERROR_LINK	Auto-negotiation is not completed, and reception is not enabled.
FSP_ERR_ETHER_PHY_ERROR_LINK	Initialization of PHY-LSI failed.

◆ R_ETHER_TxStatusGet()

fsp_err_t R_ETHER_TxStatusGet (ether_ctrl_t *const *p_ctrl*, void *const *p_buffer_address*)

Provides status of Ethernet driver in the user provided pointer. Implements ether_api_t::txStatusGet.

Return values

FSP_SUCCESS	Transmit buffer address is stored in provided p_buffer_address.
FSP_ERR_ASSERTION	Pointer to ETHER control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_INVALID_POINTER	p_status is NULL.
FSP_ERR_NOT_FOUND	Transmit buffer address has been overwritten in transmit descriptor.

4.2.24 Ethernet PHY (r_ether_phy)

Modules

Functions

fsp_err_t	R_ETHER_PHY_Open (ether_phy_ctrl_t *const p_ctrl, ether_phy_cfg_t const *const p_cfg) Resets Ethernet PHY device. Implements ether_phy_api_t::open. More...
fsp_err_t	R_ETHER_PHY_Close (ether_phy_ctrl_t *const p_ctrl) Close Ethernet PHY device. Implements ether_phy_api_t::close. More...
fsp_err_t	R_ETHER_PHY_StartAutoNegotiate (ether_phy_ctrl_t *const p_ctrl) Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate. More...
fsp_err_t	R_ETHER_PHY_LinkPartnerAbilityGet (ether_phy_ctrl_t *const p_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause) Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet. More...

fsp_err_t	R_ETHER_PHY_LinkStatusGet (ether_phy_ctrl_t *const p_ctrl) Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet. More...

Detailed Description

The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications that use the ETHERC peripheral. It implements the Ethernet PHY Interface.

Overview

The Ethernet PHY module is used to setup and manage an external Ethernet PHY device for use with the on-chip Ethernet Controller (ETHERC) peripheral. It performs auto-negotiation to determine the optimal connection parameters between link partners. Once initialized the connection between the external PHY and the onboard controller is automatically managed in hardware.

Features

The Ethernet PHY module supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support

Configuration

Build Time Configurations for r_ether_phy

The following build time configurations are defined in fsp_cfg/r_ether_phy_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Select PHY	• Default • Other • KSZ8091RNB • KSZ8041 • DP83620	Default	Select PHY chip to use. Selecting 'Default' will automatically choose the correct option when using a Renesas development board.
Reference Clock	• Default • Enabled • Disabled	Default	Select whether to use the RMII reference clock. Selecting 'Default' will automatically choose the correct option when using a Renesas development board.

Configurations for Networking > Ethernet (r_ether_phy)

This module can be added to the Stacks tab via New Stack > Networking > Ethernet (r_ether_phy).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_ether_phy0	Module name.
Channel	• 0 • 1	0	Select the Ethernet controller channel number.
PHY-LSI Address	Specify a value between 0 and 31.	0	Specify the address of the PHY-LSI used.
PHY-LSI Reset Completion Timeout	Specify a value between 0x1 and 0xFFFFFFFF.	0x00020000	Specify the number of times to read the PHY-LSI control register while waiting for reset completion. This value should be adjusted experimentally based on the PHY-LSI used.
Select MII type	• MII • RMII	RMII	Specify whether to use MII or RMII.
MII/RMII Register Access Wait-time	Specify a value between 0x1 and 0x7FFFFFFF.	8	Specify the bit timing for MII/RMII register accesses during PHY initialization. This value should be adjusted experimentally based on the PHY-LSI used.
Flow Control	• Disable • Enable	Disable	Select whether to enable or disable flow control.

Usage Notes

Note

See the example below for details on how to initialize the Ethernet PHY module.

Accessing the MII and RMII Registers

Use the PIR register to access the MII and RMII registers in the PHY-LSI. Serial data in the MII and RMII management frame format is transmitted and received through the ET0_MDC and ET0_MDIO pins controlled by software.

MII and RMII management frame format

The below table lists the MII and RMII management frame formats.

Access type	MII and RMII management frame								
	Item	PRE	ST	OP	PHYAD	REGAD	TA	DATA	IDLE
	Number of bits	32	2	2	5	5	2	16	1
Read		1...1	01	10	00001	RRRRR	Z0	DDDDDDDDDDDDDDDD	Z
Write		1...1	01	01	00001	RRRRR	10	DDDDDDDDDDDDDDDD	Z

Note

- PRE (preamble): Send 32 consecutive 1s.
- ST (start of frame): Send 01b.
- OP (operation code): Send 10b for read or 01b for write.
- PHYAD (PHY address): Up to 32 PHY-LSIs can be connected to one MAC. PHY-LSIs are selected with these 5 bits. When the PHY-LSI address is 1, send 00001b.
- REGAD (register address): One register is selected from up to 32 registers in the PHY-LSI. When the register address is 1, send 00001b.
- TA (turnaround): Use 2-bit turnaround time to avoid contention between the register address and data during a read operation.
Send 10b during a write operation. Release the bus for 1 bit during a read operation (Z is output).
(This is indicated as Z0 because 0 is output from the PHY-LSI on the next clock cycle.)
- DATA (data): 16-bit data. Sequentially send or receive starting from the MSB.
- IDLE (IDLE condition): Wait time before inputting the next MII or RMII management format. Release the bus during a write
operation (Z is output). No control is required, because a bus was already released during a read operation.

Limitations

- The r_ether_phy module may need to be customized for PHY devices other than the ones currently supported (KSZ8091RNB, KSZ8041 and DP83620). Use the existing code as a starting point for creating a custom implementation.

Examples

ETHER PHY Basic Example

This is a basic example of minimal use of the ETHER PHY in an application.

```
void ether_phy_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_ether_phy0_ctrl.open    = 0U;
    g_ether_phy0_cfg.channel = 0;
 /* Initializes the module. */
```

```
    err = R_ETHER_PHY_Open(&g_ether_phy0_ctrl, &g_ether_phy0_cfg);
/* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
/* Start auto negotiation. */
    err = R_ETHER_PHY_StartAutoNegotiate(&g_ether_phy0_ctrl);
    assert(FSP_SUCCESS == err);
/* Polling until link is established. */
while (FSP_SUCCESS != R_ETHER_PHY_LinkStatusGet(&g_ether_phy0_ctrl))
    {
/* Do nothing */
    }
/* Get link partner ability from phy interface. */
    err = R_ETHER_PHY_LinkPartnerAbilityGet(&g_ether_phy0_ctrl,
                                             &g_ether_phy0_line_speed_duplex,
                                             &g_ether_phy0_local_pause,
                                             &g_ether_phy0_partner_pause);
    assert(FSP_SUCCESS == err);
/* Check current link status. */
    err = R_ETHER_PHY_LinkStatusGet(&g_ether_phy0_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	ether_phy_instance_ctrl_t

Data Structure Documentation

◆ ether_phy_instance_ctrl_t

struct ether_phy_instance_ctrl_t

ETHER PHY control block. DO NOT INITIALIZE. Initialization occurs when ether_phy_api_t::open is called.

Data Fields

uint32_t	open	Used to determine if the channel is configured.
ether_phy_cfg_t const *	p_ether_phy_cfg	Pointer to initial configurations.
volatile uint32_t *	p_reg_pir	Pointer to ETHERC peripheral registers.

uint32_t	local_advertise	Capabilities bitmap for local advertising.

Function Documentation

◆ R_ETHER_PHY_Open()

fsp_err_t R_ETHER_PHY_Open (ether_phy_ctrl_t *const *p_ctrl*, ether_phy_cfg_t const *const *p_cfg*)

Resets Ethernet PHY device. Implements ether_phy_api_t::open.

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ASSERTION	Pointer to ETHER_PHY control block or configuration structure is NULL.
FSP_ERR_ALREADY_OPEN	Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure.
FSP_ERR_INVALID_CHANNEL	Invalid channel number is given.
FSP_ERR_INVALID_POINTER	Pointer to p_cfg is NULL.
FSP_ERR_TIMEOUT	PHY-LSI Reset wait timeout.

◆ R_ETHER_PHY_Close()

fsp_err_t R_ETHER_PHY_Close (ether_phy_ctrl_t *const *p_ctrl*)

Close Ethernet PHY device. Implements ether_phy_api_t::close.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	Pointer to ETHER_PHY control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_ETHER_PHY_StartAutoNegotiate()

fsp_err_t R_ETHER_PHY_StartAutoNegotiate (ether_phy_ctrl_t *const *p_ctrl*)

Starts auto-negotiate. Implements ether_phy_api_t::startAutoNegotiate.

Return values

FSP_SUCCESS	ETHER_PHY successfully starts auto-negotiate.
FSP_ERR_ASSERTION	Pointer to ETHER_PHY control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_ETHER_PHY_LinkPartnerAbilityGet()

fsp_err_t R_ETHER_PHY_LinkPartnerAbilityGet (ether_phy_ctrl_t *const *p_ctrl*, uint32_t *const *p_line_speed_duplex*, uint32_t *const *p_local_pause*, uint32_t *const *p_partner_pause*)

Reports the other side's physical capability. Implements ether_phy_api_t::linkPartnerAbilityGet.

Return values

FSP_SUCCESS	ETHER_PHY successfully get link partner ability.
FSP_ERR_ASSERTION	Pointer to ETHER_PHY control block is NULL.
FSP_ERR_INVALID_POINTER	Pointer to arguments are NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_ETHER_PHY_ERROR_LINK	PHY-LSI is not link up.
FSP_ERR_ETHER_PHY_NOT_READY	The auto-negotiation isn't completed

◆ R_ETHER_PHY_LinkStatusGet()

fsp_err_t R_ETHER_PHY_LinkStatusGet (ether_phy_ctrl_t *const *p_ctrl*)

Returns the status of the physical link. Implements ether_phy_api_t::linkStatusGet.

Return values

FSP_SUCCESS	ETHER_PHY successfully get link partner ability.
FSP_ERR_ASSERTION	Pointer to ETHER_PHY control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_ETHER_PHY_ERROR_LINK	PHY-LSI is not link up.

4.2.25 High-Performance Flash Driver (r_flash_hp)

Modules

Functions

fsp_err_t	R_FLASH_HP_Open (flash_ctrl_t *const p_api_ctrl, flash_cfg_t const *const p_cfg)
fsp_err_t	R_FLASH_HP_Write (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t flash_address, uint32_t const num_bytes)
fsp_err_t	R_FLASH_HP_Erase (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t const num_blocks)
fsp_err_t	R_FLASH_HP_BlankCheck (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t num_bytes, flash_result_t *blank_check_result)
fsp_err_t	R_FLASH_HP_Close (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_FLASH_HP_StatusGet (flash_ctrl_t *const p_api_ctrl, flash_status_t *const p_status)
fsp_err_t	R_FLASH_HP_AccessWindowSet (flash_ctrl_t *const p_api_ctrl, uint32_t const start_addr, uint32_t const end_addr)
fsp_err_t	R_FLASH_HP_AccessWindowClear (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_FLASH_HP_IdCodeSet (flash_ctrl_t *const p_api_ctrl, uint8_t const *const p_id_code, flash_id_code_mode_t mode)
fsp_err_t	R_FLASH_HP_Reset (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_FLASH_HP_UpdateFlashClockFreq (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_FLASH_HP_StartUpAreaSelect (flash_ctrl_t *const p_api_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)
fsp_err_t	R_FLASH_HP_CallbackSet (flash_ctrl_t *const p_api_ctrl, void(*p_callback)(flash_callback_args_t *), void const *const p_context, flash_callback_args_t *const p_callback_memory)
fsp_err_t	R_FLASH_HP_BankSwap (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_FLASH_HP_InfoGet (flash_ctrl_t *const p_api_ctrl, flash_info_t *const p_info)

Detailed Description

Driver for the flash memory on RA high-performance MCUs. This module implements the Flash Interface.

Overview

The Flash HAL module APIs allow an application to write, erase and blank check both the data and ROM flash areas that reside within the MCU. The amount of flash memory available varies across MCU parts.

Features

The R_FLASH_HP module has the following key features:

- Blocking and non-blocking erasing, writing and blank-checking of data flash.
- Blocking erasing, writing and blank-checking of code flash.
- Callback functions for completion of non-blocking data flash operations.
- Access window (write protection) for ROM Flash, allowing only specified areas of code flash to be erased or written.
- Boot block-swapping.
- ID code programming support.

Configuration

Build Time Configurations for r_flash_hp

The following build time configurations are defined in fsp_cfg/r_flash_hp_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Code Flash Programming Enable	• Enabled • Disabled	Disabled	Controls whether or not code-flash programming is enabled. Disabling reduces the amount of ROM and RAM used by the API.
Data Flash Programming Enable	• Enabled • Disabled	Enabled	Controls whether or not data-flash programming is enabled. Disabling reduces the amount of ROM used by the API.

Configurations for Storage > Flash (r_flash_hp)

This module can be added to the Stacks tab via New Stack > Storage > Flash (r_flash_hp). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_flash0	Module name.
Data Flash Background Operation	• Enabled • Disabled	Enabled	Enabling allows Flash API calls that reference data-flash to return immediately, with the operation continuing in the background.
Callback	Name must be a valid C symbol	NULL	A user callback function can be specified. Callback function called when a Data Flash Background Operation completes or errors.
Flash Ready Interrupt Priority	MCU Specific Options		Select the flash ready interrupt priority.
Flash Error Interrupt Priority	MCU Specific Options		Select the flash error interrupt priority.

Clock Configuration

Flash uses FCLK as the clock source depending on the MCU. When writing and erasing the clock source must be at least 4 MHz.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Warning

> It is highly recommended that the developer reviews sections 5 and 6 of the Flash Memory section of the target MCUs Hardware User's Manual prior to using the r_flash_hp module. In particular, understanding ID Code and Access Window functionality can help avoid unrecoverable flash scenarios.

Data Flash Background Operation (BGO) Precautions

When using the data flash BGO (Background Operation) mode, you can still access the user ROM, RAM and external memory. You must ensure that the data flash is not accessed during a data flash operation. This includes interrupts that may access the data flash.

Code Flash Precautions

Code flash cannot be accessed while writing, erasing or blank checking code flash. Code flash cannot be accessed while modifying the access window, selecting the startup area or setting the ID code. In order to support modifying code flash all supporting code must reside in RAM. This is only done when code flash programming is enabled. BGO mode is not supported for code flash, so a code flash operation will not return before the operation has completed. By default, the vector table resides in the code flash. If an interrupt occurs during the code flash operation, then code flash will be accessed to fetch the interrupt's starting address and an error will occur. The simplest work-around is to disable interrupts during code flash operations. Another option is to copy the vector table to RAM, update the VTOR (Vector Table Offset Register) accordingly and ensure that any interrupt service routines execute out of RAM. Similarly, you must insure that if in a multi-threaded environment, threads running from code flash cannot become active while a code flash operation is in progress.

Flash Clock (FCLK)

The flash clock source is the clock used by the Flash peripheral in performing all Flash operations. As part of the flash_api_t::open function the Flash clock source is checked will return FSP_ERR_FCLK if it is invalid. Once the Flash API has been opened, if the flash clock source frequency is changed, the flash_api_t::updateFlashClockFreq API function must be called to inform the API of the change. Failure to do so could result in flash operation failures and possibly damage the part.

Interrupts

Enable the flash ready interrupt only if you plan to use the data flash BGO. In this mode, the application can initiate a data flash operation and then be asynchronously notified of its completion, or an error, using a user supplied-callback function. The callback function is passed a structure containing event information that indicates the source of the callback event (for example, flash_api_t::FLASH_EVENT_ERASE_COMPLETE) When the FLASH FRDYI interrupt is enabled, the corresponding ISR will be defined in the flash driver. The ISR will call a user-callback function if one was registered with the flash_api_t::open API.

Note

The Flash HP supports an additional flash-error interrupt and if the BGO mode is enabled for the FLASH HP then both the Flash Ready Interrupt and Flash Error Interrupts must be enabled (assigned a priority).

Limitations

- Write operations must be aligned on page boundaries and must be a multiple of the page boundary size.
- Erase operations will erase the entire block the provided address resides in.
- Data flash is better suited for storing data as it can be erased and written to while code is still executing from code flash. Data flash is also guaranteed for a larger number of reprogramming/erasure cycles than code flash.
- Read values of erased data flash blocks are not guaranteed to be 0xFF. Blank check should be used to determine if memory has been erased but not yet programmed.

Examples

High-Performance Flash Basic Example

This is a basic example of erasing and writing to data flash and code flash.

```
#define FLASH_DF_BLOCK_0 0x40100000U /* 64 B: 0x40100000 - 0x4010003F */
#define FLASH_CF_BLOCK_8 0x00010000  /* 32 KB: 0x00010000 - 0x00017FFF */
#define FLASH_DATA_BLOCK_SIZE (1024)
#define FLASH_HP_EXAMPLE_WRITE_SIZE 32
```

```
uint8_t       g_dest[TRANSFER_LENGTH];
uint8_t       g_src[TRANSFER_LENGTH];
flash_result_t blank_check_result;
void r_flash_hp_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the flash hp instance. */
 fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_cfg);
    assert(FSP_SUCCESS == err);
 /* Erase 1 block of data flash starting at block 0. */
    err = R_FLASH_HP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
    assert(FSP_SUCCESS == err);
 /* Check if block 0 is erased. */
    err = R_FLASH_HP_BlankCheck(&g_flash_ctrl, FLASH_DF_BLOCK_0,
FLASH_DATA_BLOCK_SIZE, &blank_check_result);
    assert(FSP_SUCCESS == err);
 /* Verify the previously erased area is blank */
    assert(FLASH_RESULT_BLANK == blank_check_result);
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_HP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE);
    assert(FSP_SUCCESS == err);
    assert(0 == memcmp(g_src, (uint8_t *) FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE));
 /* Disable interrupts to prevent vector table access while code flash is in P/E
mode. */
    __disable_irq();
 /* Erase 1 block of code flash starting at block 10. */
    err = R_FLASH_HP_Erase(&g_flash_ctrl, FLASH_CF_BLOCK_8, 1);
    assert(FSP_SUCCESS == err);
```

```
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_HP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_CF_BLOCK_8,
FLASH_HP_EXAMPLE_WRITE_SIZE);
    assert(FSP_SUCCESS == err);
 /* Enable interrupts after code flash operations are complete. */
    __enable_irq();
    assert(0 == memcmp(g_src, (uint8_t *) FLASH_CF_BLOCK_8,
FLASH_HP_EXAMPLE_WRITE_SIZE));
}
```

High-Performance Flash Advanced Example

This example demonstrates using BGO to do non-blocking operations on the data flash.

```
bool interrupt_called;
flash_event_t flash_event;
static flash_cfg_t g_flash_bgo_example_cfg =
{
    .p_callback    = flash_callback,
    .p_context      = 0,
    .p_extend       = NULL,
    .data_flash_bgo = true,
    .ipl            = 5,
    .irq            = BSP_VECTOR_FLASH_HP_FRDYI_ISR,
};
void r_flash_hp_bgo_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the flash hp instance. */
 fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_bgo_example_cfg);
 /* Handle any errors. */
```

```
    assert(FSP_SUCCESS == err);
    interrupt_called = false;
 /* Erase 1 block of data flash starting at block 0. */
    err = R_FLASH_HP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
    assert(FSP_SUCCESS == err);
 while (!interrupt_called)
    {
       ;
    }
    assert(FLASH_EVENT_ERASE_COMPLETE == flash_event);
    interrupt_called = false;
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_HP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE);
    assert(FSP_SUCCESS == err);
 flash_status_t status;
 /* Wait until the current flash operation completes. */
 do
    {
       err = R_FLASH_HP_StatusGet(&g_flash_ctrl, &status);
    } while ((FSP_SUCCESS == err) && (FLASH_STATUS_BUSY == status));
 /* If the interrupt wasn't called process the error. */
    assert(interrupt_called);
 /* If the event wasn't a write complete process the error. */
    assert(FLASH_EVENT_WRITE_COMPLETE == flash_event);
 /* Verify the data was written correctly. */
    assert(0 == memcmp(g_src, (uint8_t *) FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE));
}
void flash_callback (flash_callback_args_t * p_args)
{
    interrupt_called = true;
    flash_event      = p_args->event;
}
```

High-Performance Flash Bank Swap Example

This example demonstrates swapping which flash bank is located at address 0. This feature is only on select MCUs.

```
void r_flash_hp_bankswap_example (void)
{
 /* Open the flash hp instance. */
 fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_cfg);
 /* Handle any errors. */
    assert(FSP_SUCCESS == err);
 /* Write the new application starting at 0x00200000. */
 /* Swap the block at address 0 with the one at 0x00200000 after the next restart.
  * The application at 0x00200000 must be written there by application code. */
    err = R_FLASH_HP_BankSwap(&g_flash_ctrl);
 /* Handle any errors. */
    assert(FSP_SUCCESS == err);
 /* Handle any pre-reset operations here */
 /* Reset the MCU to swap to the other bank */
    __NVIC_SystemReset();
}
```

Data Structures

struct	flash_hp_instance_ctrl_t

Enumerations

enum	flash_bgo_operation_t

Data Structure Documentation

◆ flash_hp_instance_ctrl_t

struct flash_hp_instance_ctrl_t	
Flash HP instance control block. DO NOT INITIALIZE.	
Data Fields	
uint32_t	opened
	To check whether api has been opened or not.

flash_bgo_operation_t	current_operation
	Operation in progress, for example, FLASH_OPERATION_CF_ERASE.

Enumeration Type Documentation

◆ flash_bgo_operation_t

enum flash_bgo_operation_t

Possible Flash operation states

Function Documentation

◆ R_FLASH_HP_Open()

fsp_err_t R_FLASH_HP_Open (flash_ctrl_t *const *p_api_ctrl*, flash_cfg_t const *const *p_cfg*)

Initializes the high performance flash peripheral. Implements flash_api_t::open.

The Open function initializes the Flash.

Example:

```
/* Open the flash hp instance. */
fsp_err_t err = R_FLASH_HP_Open(&g_flash_ctrl, &g_flash_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful and timer has started.
FSP_ERR_ALREADY_OPEN	The flash control block is already open.
FSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
FSP_ERR_IRQ_BSP_DISABLED	Caller is requesting BGO but the Flash interrupts are not enabled.
FSP_ERR_FCLK	FCLK must be a minimum of 4 MHz for Flash operations.

◆ R_FLASH_HP_Write()

fsp_err_t R_FLASH_HP_Write (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *src_address*, uint32_t *flash_address*, uint32_t const *num_bytes*)

Writes to the specified Code or Data Flash memory area. Implements flash_api_t::write.

Example:

```
/* Write 32 bytes to the first block of data flash. */
   err = R_FLASH_HP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_HP_EXAMPLE_WRITE_SIZE);
```

Return values

FSP_SUCCESS	Operation successful. If BGO is enabled this means the operation was started successfully.
FSP_ERR_IN_USE	The Flash peripheral is busy with a prior on-going transaction.
FSP_ERR_NOT_OPEN	The Flash API is not Open.
FSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Write an area that is protected by an Access Window.
FSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation. This may be returned if the requested Flash area is not blank.
FSP_ERR_TIMEOUT	Timed out waiting for FCU operation to complete.
FSP_ERR_INVALID_SIZE	Number of bytes provided was not a multiple of the programming size or exceeded the maximum range.
FSP_ERR_INVALID_ADDRESS	Invalid address was input or address not on programming boundary.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_PE_FAILURE	Failed to enter or exit P/E mode.

◆ R_FLASH_HP_Erase()

fsp_err_t R_FLASH_HP_Erase (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t const *num_blocks*)

Erases the specified Code or Data Flash blocks. Implements flash_api_t::erase by the block_erase_address.

Note

Code flash may contain blocks of different sizes. When erasing code flash it is important to take this into consideration to prevent erasing a larger address space than desired.

Example:

```
/* Erase 1 block of data flash starting at block 0. */
    err = R_FLASH_HP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
```

Return values

FSP_SUCCESS	Successful open.
FSP_ERR_INVALID_BLOCKS	Invalid number of blocks specified
FSP_ERR_INVALID_ADDRESS	Invalid address specified. If the address is in code flash then code flash programming must be enabled.
FSP_ERR_IN_USE	Other flash operation in progress, or API not initialized
FSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Erase an area that is protected by an Access Window.
FSP_ERR_ASSERTION	NULL provided for p_ctrl
FSP_ERR_NOT_OPEN	The Flash API is not Open.
FSP_ERR_ERASE_FAILED	Status is indicating a Erase error.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.
FSP_ERR_PE_FAILURE	Failed to enter or exit P/E mode.

◆ R_FLASH_HP_BlankCheck()

fsp_err_t R_FLASH_HP_BlankCheck (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t *num_bytes*, flash_result_t * *p_blank_check_result*)

Performs a blank check on the specified address area. Implements flash_api_t::blankCheck.

Example:

```
/* Check if block 0 is erased. */
    err = R_FLASH_HP_BlankCheck(&g_flash_ctrl, FLASH_DF_BLOCK_0,
FLASH_DATA_BLOCK_SIZE, &blank_check_result);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Blank check operation completed with result in p_blank_check_result, or blank check started and in-progess (BGO mode).
FSP_ERR_INVALID_ADDRESS	Invalid data flash address was input.
FSP_ERR_INVALID_SIZE	'num_bytes' was either too large or not aligned for the CF/DF boundary size.
FSP_ERR_IN_USE	Other flash operation in progress or API not initialized.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of attempting to Erase an area that is protected by an Access Window.
FSP_ERR_NOT_OPEN	The Flash API is not Open.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.
FSP_ERR_PE_FAILURE	Failed to enter or exit P/E mode.
FSP_ERR_BLANK_CHECK_FAILED	Blank check operation failed.

◆ R_FLASH_HP_Close()

fsp_err_t R_FLASH_HP_Close (flash_ctrl_t *const *p_api_ctrl*)

Releases any resources that were allocated by the Open() or any subsequent Flash operations. Implements flash_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_NOT_OPEN	The control block is not open.
FSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.

◆ R_FLASH_HP_StatusGet()

fsp_err_t R_FLASH_HP_StatusGet (flash_ctrl_t *const *p_api_ctrl*, flash_status_t *const *p_status*)

Query the FLASH peripheral for its status. Implements flash_api_t::statusGet.

Example:

```
flash_status_t status;
/* Wait until the current flash operation completes. */
do
   {
      err = R_FLASH_HP_StatusGet(&g_flash_ctrl, &status);
   } while ((FSP_SUCCESS == err) && (FLASH_STATUS_BUSY == status));
```

Return values

FSP_SUCCESS	FLASH peripheral is ready to use.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_NOT_OPEN	The Flash API is not Open.

◆ R_FLASH_HP_AccessWindowSet()

fsp_err_t R_FLASH_HP_AccessWindowSet (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *start_addr*, uint32_t const *end_addr*)

Configure an access window for the Code Flash memory using the provided start and end address. An access window defines a contiguous area in Code Flash for which programming/erase is enabled. This area is on block boundaries. The block containing start_addr is the first block. The block containing end_addr is the last block. The access window then becomes first block –> last block inclusive. Anything outside this range of Code Flash is then write protected.

Note

If the start address and end address are set to the same value, then the access window is effectively removed. This accomplishes the same functionality as R_FLASH_HP_AccessWindowClear().

Implements flash_api_t::accessWindowSet.

Return values

FSP_SUCCESS	Access window successfully configured.
FSP_ERR_INVALID_ADDRESS	Invalid settings for start_addr and/or end_addr.
FSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_UNSUPPORTED	Code Flash Programming is not enabled.
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.
FSP_ERR_PE_FAILURE	Failed to enter or exit Code Flash P/E mode.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.
FSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation.
FSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of having received an illegal command.

◆ R_FLASH_HP_AccessWindowClear()

fsp_err_t R_FLASH_HP_AccessWindowClear (flash_ctrl_t *const *p_api_ctrl*)

Remove any access window that is currently configured in the Code Flash. Subsequent to this call all Code Flash is writable. Implements flash_api_t::accessWindowClear.

Return values

FSP_SUCCESS	Access window successfully removed.
FSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_UNSUPPORTED	Code Flash Programming is not enabled.
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.
FSP_ERR_PE_FAILURE	Failed to enter or exit Code Flash P/E mode.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.
FSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation.
FSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of having received an illegal command.

◆ R_FLASH_HP_IdCodeSet()

fsp_err_t R_FLASH_HP_IdCodeSet (flash_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_id_code*, flash_id_code_mode_t *mode*)

Implements flash_api_t::idCodeSet.

Return values

FSP_SUCCESS	ID Code successfully configured.
FSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_UNSUPPORTED	Code Flash Programming is not enabled.
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.
FSP_ERR_PE_FAILURE	Failed to enter or exit Code Flash P/E mode.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.
FSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation.
FSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of having received an illegal command.

◆ R_FLASH_HP_Reset()

fsp_err_t R_FLASH_HP_Reset (flash_ctrl_t *const *p_api_ctrl*)

Reset the FLASH peripheral. Implements flash_api_t::reset.

No attempt is made to check if the flash is busy before executing the reset since the assumption is that a reset will terminate any existing operation.

Return values

FSP_SUCCESS	Flash circuit successfully reset.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_NOT_OPEN	The control block is not open.
FSP_ERR_PE_FAILURE	Failed to enter or exit P/E mode.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.
FSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of having received an illegal command.

◆ R_FLASH_HP_UpdateFlashClockFreq()

fsp_err_t R_FLASH_HP_UpdateFlashClockFreq (flash_ctrl_t *const *p_api_ctrl*)

Indicate to the already open Flash API that the FCLK has changed. Implements flash_api_t::updateFlashClockFreq.

This could be the case if the application has changed the system clock, and therefore the FCLK. Failure to call this function subsequent to changing the FCLK could result in damage to the flash macro.

Return values

FSP_SUCCESS	Start-up area successfully toggled.
FSP_ERR_IN_USE	Flash is busy with an on-going operation.
FSP_ERR_ASSERTION	NULL provided for p_ctrl
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.
FSP_ERR_FCLK	FCLK is not within the acceptable range.

◆ R_FLASH_HP_StartUpAreaSelect()

fsp_err_t R_FLASH_HP_StartUpAreaSelect (flash_ctrl_t *const *p_api_ctrl*, flash_startup_area_swap_t *swap_type*, bool *is_temporary*)

Selects which block, Default (Block 0) or Alternate (Block 1), is used as the startup area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary), or permanent subsequent to the next reset. Doing a temporary switch might appear to have limited usefulness. If there is an access window in place such that Block 0 is write protected, then one could do a temporary switch, update the block and switch them back without having to touch the access window. Implements flash_api_t::startupAreaSelect.

Return values

FSP_SUCCESS	Start-up area successfully toggled.
FSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_NOT_OPEN	The control block is not open.
FSP_ERR_UNSUPPORTED	Code Flash Programming is not enabled.
FSP_ERR_PE_FAILURE	Failed to enter or exit Code Flash P/E mode.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.
FSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation.
FSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of having received an illegal command.

◆ R_FLASH_HP_CallbackSet()

fsp_err_t R_FLASH_HP_CallbackSet (flash_ctrl_t *const *p_api_ctrl*, void(*)(flash_callback_args_t *) *p_callback*, void const *const *p_context*, flash_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements flash_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_FLASH_HP_BankSwap()

fsp_err_t R_FLASH_HP_BankSwap (flash_ctrl_t *const *p_api_ctrl*)

Swaps the flash bank located at address 0x00000000 and address 0x00200000. This can only be done when in dual bank mode. Dual bank mode can be enabled in the FSP Configuration Tool under BSP Properties. After a bank swap is done the MCU will need to be reset for the changes to take place. flash_api_t::bankSwap.

Return values

FSP_SUCCESS	Start-up area successfully toggled.
FSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_NOT_OPEN	The control block is not open.
FSP_ERR_UNSUPPORTED	Code Flash Programming is not enabled.
FSP_ERR_PE_FAILURE	Failed to enter or exit Code Flash P/E mode.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.
FSP_ERR_INVALID_MODE	Cannot switch banks while flash is in Linear mode.
FSP_ERR_WRITE_FAILED	Flash write operation failed.
FSP_ERR_CMD_LOCKED	FCU is in locked state, typically as a result of having received an illegal command.

◆ R_FLASH_HP_InfoGet()

fsp_err_t R_FLASH_HP_InfoGet (flash_ctrl_t *const *p_api_ctrl*, flash_info_t *const *p_info*)

Returns the information about the flash regions. Implements flash_api_t::infoGet.

Return values

FSP_SUCCESS	Successful retrieved the request information.
FSP_ERR_NOT_OPEN	The control block is not open.
FSP_ERR_ASSERTION	NULL provided for p_ctrl or p_info.

4.2.26 Low-Power Flash Driver (r_flash_lp)

Modules

Functions

fsp_err_t	R_FLASH_LP_Open (flash_ctrl_t *const p_api_ctrl, flash_cfg_t const *const p_cfg)
fsp_err_t	R_FLASH_LP_Write (flash_ctrl_t *const p_api_ctrl, uint32_t const src_address, uint32_t flash_address, uint32_t const num_bytes)
fsp_err_t	R_FLASH_LP_Erase (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t const num_blocks)
fsp_err_t	R_FLASH_LP_BlankCheck (flash_ctrl_t *const p_api_ctrl, uint32_t const address, uint32_t num_bytes, flash_result_t *blank_check_result)
fsp_err_t	R_FLASH_LP_Close (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_FLASH_LP_StatusGet (flash_ctrl_t *const p_api_ctrl, flash_status_t *const p_status)
fsp_err_t	R_FLASH_LP_AccessWindowSet (flash_ctrl_t *const p_api_ctrl, uint32_t const start_addr, uint32_t const end_addr)
fsp_err_t	R_FLASH_LP_AccessWindowClear (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_FLASH_LP_IdCodeSet (flash_ctrl_t *const p_api_ctrl, uint8_t const *const p_id_code, flash_id_code_mode_t mode)
fsp_err_t	R_FLASH_LP_Reset (flash_ctrl_t *const p_api_ctrl)

fsp_err_t	R_FLASH_LP_StartUpAreaSelect (flash_ctrl_t *const p_api_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)
fsp_err_t	R_FLASH_LP_CallbackSet (flash_ctrl_t *const p_api_ctrl, void(*p_callback)(flash_callback_args_t *), void const *const p_context, flash_callback_args_t *const p_callback_memory)
fsp_err_t	R_FLASH_LP_BankSwap (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_FLASH_LP_UpdateFlashClockFreq (flash_ctrl_t *const p_api_ctrl)
fsp_err_t	R_FLASH_LP_InfoGet (flash_ctrl_t *const p_api_ctrl, flash_info_t *const p_info)

Detailed Description

Driver for the flash memory on RA low-power MCUs. This module implements the Flash Interface.

Overview

The Flash HAL module APIs allow an application to write, erase and blank check both the data and code flash areas that reside within the MCU. The amount of flash memory available varies across MCU parts.

Features

The Low-Power Flash HAL module has the following key features:

- Blocking and non-blocking erasing, writing and blank-checking of data flash.
- Blocking erasing, writing and blank checking of code flash.
- Callback functions for completion of non-blocking data flash operations.
- Access window (write protection) for code flash, allowing only specified areas of code flash to be erased or written.
- Boot block-swapping.
- ID code programming support.

Configuration

Build Time Configurations for r_flash_lp

The following build time configurations are defined in fsp_cfg/r_flash_lp_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Code Flash Programming	• Enabled • Disabled	Disabled	Controls whether or not code-flash programming is

			enabled. Disabling reduces the amount of ROM and RAM used by the API.
Data Flash Programming	• Enabled • Disabled	Enabled	Controls whether or not data-flash programming is enabled. Disabling reduces the amount of ROM used by the API.

Configurations for Storage > Flash (r_flash_lp)

This module can be added to the Stacks tab via New Stack > Storage > Flash (r_flash_lp).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_flash0	Module name.
Data Flash Background Operation	• Enabled • Disabled	Enabled	Enabling allows Flash API calls that reference data-flash to return immediately, with the operation continuing in the background.
Callback	Name must be a valid C symbol	NULL	A user callback function can be specified. Callback function called when a Data Flash Background Operation completes or errors.
Flash Ready Interrupt Priority	MCU Specific Options		Select the flash ready interrupt priority.

Clock Configuration

Flash either uses FCLK or ICLK as the clock source depending on the MCU. When writing and erasing the clock source must be at least 4 MHz.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Warning

It is highly recommended that the developer reviews sections 5 and 6 of the Flash Memory section of the target MCUs Hardware User's Manual prior to using the r_flash_lp module. In particular, understanding ID Code and Access Window functionality can help avoid unrecoverable flash scenarios.

Data Flash Background Operation (BGO) Precautions

When using the data flash BGO, the code flash, RAM and external memory can still be accessed. You must ensure that the data flash is not accessed during a data flash operation. This includes interrupts that may access the data flash.

Code Flash Precautions

Code flash cannot be accessed while writing, erasing or blank checking code flash. Code flash cannot be accessed while modifying the access window, selecting the startup area or setting the ID code. In order to support modifying code flash all supporting code must reside in RAM. This is only done when code flash programming is enabled. BGO mode is not supported for code flash, so a code flash operation will not return before the operation has completed. By default, the vector table resides in the code flash. If an interrupt occurs during the code flash operation, then code flash will be accessed to fetch the interrupt's starting address and an error will occur. The simplest work-around is to disable interrupts during code flash operations. Another option is to copy the vector table to RAM, update the VTOR (Vector Table Offset Register) accordingly and ensure that any interrupt service routines execute out of RAM. Similarly, you must insure that if in a multi-threaded environment, threads running from code flash cannot become active while a code flash operation is in progress.

Flash Clock Source

The flash clock source is the clock used by the Flash peripheral in performing all Flash operations. As part of the flash_api_t::open function the Flash clock source is checked will return FSP_ERR_FCLK if it is invalid. Once the Flash API has been opened, if the flash clock source frequency is changed, the flash_api_t::updateFlashClockFreq API function must be called to inform the API of the change. Failure to do so could result in flash operation failures and possibly damage the part.

Interrupts

Enable the flash ready interrupt only if you plan to use the data flash BGO. In this mode, the application can initiate a data flash operation and then be asynchronously notified of its completion, or an error, using a user supplied-callback function. The callback function is passed a structure containing event information that indicates the source of the callback event (for example, flash_api_t::FLASH_EVENT_ERASE_COMPLETE) When the FLASH FRDYI interrupt is enabled, the corresponding ISR will be defined in the flash driver. The ISR will call a user-callback function if one was registered with the flash_api_t::open API.

Note

> *The Flash HP supports an additional flash-error interrupt and if the BGO mode is enabled for the FLASH HP then both the Flash Ready Interrupt and Flash Error Interrupts must be enabled (assigned a priority).*

Limitations

- Write operations must be aligned on page boundaries and must be a multiple of the page boundary size.
- Erase operations will erase the entire block the provided address resides in.
- Data flash is better suited for storing data as it can be erased and written to while code is still executing from code flash. Data flash is also guaranteed for a larger number of reprogramming/erasure cycles than code flash.
- Read values of erased blocks are not guaranteed to be 0xFF. Blank check should be used to determine if memory has been erased but not yet programmed.

Examples

Low-Power Flash Basic Example

This is a basic example of erasing and writing to data flash and code flash.

```
#define FLASH_DF_BLOCK_0 0x40100000U /* 1 KB: 0x40100000 - 0x401003FF */
#define FLASH_CF_BLOCK_10 0x00005000  /* 2 KB: 0x00005000 - 0x000057FF */
#define FLASH_DATA_BLOCK_SIZE (1024)
#define FLASH_LP_EXAMPLE_WRITE_SIZE 32
uint8_t       g_dest[TRANSFER_LENGTH];
uint8_t       g_src[TRANSFER_LENGTH];
flash_result_t blank_check_result;
void R_FLASH_LP_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the flash lp instance. */
 fsp_err_t err = R_FLASH_LP_Open(&g_flash_ctrl, &g_flash_cfg);
    assert(FSP_SUCCESS == err);
 /* Erase 1 block of data flash starting at block 0. */
    err = R_FLASH_LP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
    assert(FSP_SUCCESS == err);
 /* Check if block 0 is erased. */
    err = R_FLASH_LP_BlankCheck(&g_flash_ctrl, FLASH_DF_BLOCK_0,
FLASH_DATA_BLOCK_SIZE, &blank_check_result);
    assert(FSP_SUCCESS == err);
 /* Verify the previously erased area is blank */
    assert(FLASH_RESULT_BLANK == blank_check_result);
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_LP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE);
    assert(FSP_SUCCESS == err);
```

```
    assert(0 == memcmp(g_src, (uint8_t *) FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE));
 /* Disable interrupts to prevent vector table access while code flash is in P/E
mode. */
    __disable_irq();
 /* Erase 1 block of code flash starting at block 10. */
    err = R_FLASH_LP_Erase(&g_flash_ctrl, FLASH_CF_BLOCK_10, 1);
    assert(FSP_SUCCESS == err);
 /* Write 32 bytes to the first block of data flash. */
    err = R_FLASH_LP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_CF_BLOCK_10,
FLASH_LP_EXAMPLE_WRITE_SIZE);
    assert(FSP_SUCCESS == err);
 /* Enable interrupts after code flash operations are complete. */
    __enable_irq();
    assert(0 == memcmp(g_src, (uint8_t *) FLASH_CF_BLOCK_10,
FLASH_LP_EXAMPLE_WRITE_SIZE));
}
```

Low-Power Flash Advanced Example

This example demonstrates using BGO to do non-blocking operations on the data flash.

```
bool interrupt_called;
flash_event_t flash_event;
static flash_cfg_t g_flash_bgo_example_cfg =
{
    .p_callback    = flash_callback,
    .p_context      = 0,
    .p_extend       = NULL,
    .data_flash_bgo = true,
    .ipl            = 5,
    .irq            = BSP_VECTOR_FLASH_LP_FRDYI_ISR,
};
void R_FLASH_LP_bgo_example (void)
{
```

```
/* Initialize p_src to known data */
for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
   {
      g_src[i] = (uint8_t) ('A' + (i % 26));
   }
/* Open the flash lp instance. */
fsp_err_t err = R_FLASH_LP_Open(&g_flash_ctrl, &g_flash_bgo_example_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
   interrupt_called = false;
/* Erase 1 block of data flash starting at block 0. */
   err = R_FLASH_LP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
   assert(FSP_SUCCESS == err);
while (!interrupt_called)
   {
      ;
   }
   assert(FLASH_EVENT_ERASE_COMPLETE == flash_event);
   interrupt_called = false;
/* Write 32 bytes to the first block of data flash. */
   err = R_FLASH_LP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE);
   assert(FSP_SUCCESS == err);
flash_status_t status;
/* Wait until the current flash operation completes. */
do
   {
      err = R_FLASH_LP_StatusGet(&g_flash_ctrl, &status);
   } while ((FSP_SUCCESS == err) && (FLASH_STATUS_BUSY == status));
/* If the interrupt wasn't called process the error. */
   assert(interrupt_called);
/* If the event wasn't a write complete process the error. */
   assert(FLASH_EVENT_WRITE_COMPLETE == flash_event);
/* Verify the data was written correctly. */
```

```
    assert(0 == memcmp(g_src, (uint8_t *) FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE));
}
void flash_callback (flash_callback_args_t * p_args)
{
    interrupt_called = true;
    flash_event      = p_args->event;
}
```

Data Structures

struct	flash_lp_instance_ctrl_t

Data Structure Documentation

◆ flash_lp_instance_ctrl_t

struct flash_lp_instance_ctrl_t
Flash instance control block. DO NOT INITIALIZE. Initialization occurs when R_FLASH_LP_Open() is called.

Function Documentation

◆ R_FLASH_LP_Open()

fsp_err_t R_FLASH_LP_Open (flash_ctrl_t *const *p_api_ctrl*, flash_cfg_t const *const *p_cfg*)

Initialize the Low Power flash peripheral. Implements flash_api_t::open.

The Open function initializes the Flash.

This function must be called once prior to calling any other FLASH API functions. If a user supplied callback function is supplied, then the Flash Ready interrupt will be configured to call the users callback routine with an Event type describing the source of the interrupt for Data Flash operations.

Example:

```
/* Open the flash lp instance. */
fsp_err_t err = R_FLASH_LP_Open(&g_flash_ctrl, &g_flash_cfg);
```

Note

Providing a callback function in the supplied p_cfg->callback field automatically configures the Flash for Data Flash to operate in non-blocking background operation (BGO) mode.

Return values

FSP_SUCCESS	Initialization was successful and timer has started.
FSP_ERR_ASSERTION	NULL provided for p_ctrl, p_cfg or p_callback if BGO is enabled.
FSP_ERR_IRQ_BSP_DISABLED	Caller is requesting BGO but the Flash interrupts are not enabled.
FSP_ERR_FCLK	FCLK must be a minimum of 4 MHz for Flash operations.
FSP_ERR_ALREADY_OPEN	Flash Open() has already been called.
FSP_ERR_TIMEOUT	Failed to exit P/E mode after configuring flash.
FSP_ERR_INVALID_STATE	The system is not running from the required clock.

◆ R_FLASH_LP_Write()

fsp_err_t R_FLASH_LP_Write (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *src_address*, uint32_t *flash_address*, uint32_t const *num_bytes*)

Write to the specified Code or Data Flash memory area. Implements flash_api_t::write.

Example:

```
/* Write 32 bytes to the first block of data flash. */
   err = R_FLASH_LP_Write(&g_flash_ctrl, (uint32_t) g_src, FLASH_DF_BLOCK_0,
FLASH_LP_EXAMPLE_WRITE_SIZE);
```

Return values

FSP_SUCCESS	Operation successful. If BGO is enabled this means the operation was started successfully.
FSP_ERR_IN_USE	The Flash peripheral is busy with a prior on-going transaction.
FSP_ERR_NOT_OPEN	The Flash API is not Open.
FSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation. This may be returned if the requested Flash area is not blank.
FSP_ERR_TIMEOUT	Timed out waiting for FCU operation to complete.
FSP_ERR_INVALID_SIZE	Number of bytes provided was not a multiple of the programming size or exceeded the maximum range.
FSP_ERR_INVALID_ADDRESS	Invalid address was input or address not on programming boundary.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.

◆ R_FLASH_LP_Erase()

fsp_err_t R_FLASH_LP_Erase (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t const *num_blocks*)

Erase the specified Code or Data Flash blocks. Implements flash_api_t::erase.

Example:

```
/* Erase 1 block of data flash starting at block 0. */
   err = R_FLASH_LP_Erase(&g_flash_ctrl, FLASH_DF_BLOCK_0, 1);
```

Return values

FSP_SUCCESS	Successful open.
FSP_ERR_INVALID_BLOCKS	Invalid number of blocks specified
FSP_ERR_INVALID_ADDRESS	Invalid address specified
FSP_ERR_IN_USE	Other flash operation in progress, or API not initialized
FSP_ERR_ASSERTION	NULL provided for p_ctrl
FSP_ERR_NOT_OPEN	The Flash API is not Open.
FSP_ERR_TIMEOUT	Timed out waiting for FCU to be ready.
FSP_ERR_ERASE_FAILED	Status is indicating a Erase error.

◆ R_FLASH_LP_BlankCheck()

fsp_err_t R_FLASH_LP_BlankCheck (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *address*, uint32_t *num_bytes*, flash_result_t * *p_blank_check_result*)

Perform a blank check on the specified address area. Implements flash_api_t::blankCheck.

Example:

```
/* Check if block 0 is erased. */
    err = R_FLASH_LP_BlankCheck(&g_flash_ctrl, FLASH_DF_BLOCK_0,
FLASH_DATA_BLOCK_SIZE, &blank_check_result);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Blankcheck operation completed with result in p_blank_check_result, or blankcheck started and in-progess (BGO mode).
FSP_ERR_INVALID_ADDRESS	Invalid data flash address was input
FSP_ERR_INVALID_SIZE	'num_bytes' was either too large or not aligned for the CF/DF boundary size.
FSP_ERR_IN_USE	Flash is busy with an on-going operation.
FSP_ERR_ASSERTION	NULL provided for p_ctrl
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.
FSP_ERR_BLANK_CHECK_FAILED	An error occurred during blank checking.

◆ R_FLASH_LP_Close()

fsp_err_t R_FLASH_LP_Close (flash_ctrl_t *const *p_api_ctrl*)

Release any resources that were allocated by the Flash API. Implements flash_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	NULL provided for p_ctrl or p_cfg.
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.
FSP_ERR_IN_USE	The flash is currently in P/E mode.

◆ R_FLASH_LP_StatusGet()

fsp_err_t R_FLASH_LP_StatusGet (flash_ctrl_t *const *p_api_ctrl*, flash_status_t *const *p_status*)

Query the FLASH for its status. Implements flash_api_t::statusGet.

Example:

```
flash_status_t status;
/* Wait until the current flash operation completes. */
do
   {
      err = R_FLASH_LP_StatusGet(&g_flash_ctrl, &status);
   } while ((FSP_SUCCESS == err) && (FLASH_STATUS_BUSY == status));
```

Return values

FSP_SUCCESS	Flash is ready and available to accept commands.
FSP_ERR_ASSERTION	NULL provided for p_ctrl
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.

◆ R_FLASH_LP_AccessWindowSet()

fsp_err_t R_FLASH_LP_AccessWindowSet (flash_ctrl_t *const *p_api_ctrl*, uint32_t const *start_addr*, uint32_t const *end_addr*)

Configure an access window for the Code Flash memory. Implements flash_api_t::accessWindowSet .

An access window defines a contiguous area in Code Flash for which programming/erase is enabled. This area is on block boundaries. The block containing start_addr is the first block. The block containing end_addr is the last block. The access window then becomes first block (inclusive) –> last block (exclusive). Anything outside this range of Code Flash is then write protected. As an example, if you wanted to place an accesswindow on Code Flash Blocks 0 and 1, such that only those two blocks were writable, you would need to specify (address in block 0, address in block 2) as the respective start and end address.

Note

If the start address and end address are set to the same value, then the access window is effectively removed. This accomplishes the same functionality as R_FLASH_LP_AccessWindowClear().

The invalid address and programming boundaries supported and enforced by this function are dependent on the MCU in use as well as the part package size. Please see the User manual and/or requirements document for additional information.

Parameters

	p_api_ctrl	The p api control
[in]	start_addr	The start address
[in]	end_addr	The end address

Return values

FSP_SUCCESS	Access window successfully configured.
FSP_ERR_INVALID_ADDRESS	Invalid settings for start_addr and/or end_addr.
FSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_UNSUPPORTED	Code Flash Programming is not enabled.
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.
FSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation.

◆ R_FLASH_LP_AccessWindowClear()

fsp_err_t R_FLASH_LP_AccessWindowClear (flash_ctrl_t *const *p_api_ctrl*)

Remove any access window that is configured in the Code Flash. Implements flash_api_t::accessWindowClear. On successful return from this call all Code Flash is writable.

Return values

FSP_SUCCESS	Access window successfully removed.
FSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_UNSUPPORTED	Code Flash Programming is not enabled.
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.
FSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation.

◆ R_FLASH_LP_IdCodeSet()

fsp_err_t R_FLASH_LP_IdCodeSet (flash_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_id_code*, flash_id_code_mode_t *mode*)

Write the ID code provided to the id code registers. Implements flash_api_t::idCodeSet.

Return values

FSP_SUCCESS	ID code successfully configured.
FSP_ERR_IN_USE	FLASH peripheral is busy with a prior operation.
FSP_ERR_ASSERTION	NULL provided for p_ctrl.
FSP_ERR_UNSUPPORTED	Code Flash Programming is not enabled.
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.
FSP_ERR_TIMEOUT	Timed out waiting for completion of extra command.
FSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation.

◆ R_FLASH_LP_Reset()

fsp_err_t R_FLASH_LP_Reset (flash_ctrl_t *const *p_api_ctrl*)

Reset the FLASH peripheral. Implements flash_api_t::reset.

No attempt is made to check if the flash is busy before executing the reset since the assumption is that a reset will terminate any existing operation.

Return values

FSP_SUCCESS	Flash circuit successfully reset.
FSP_ERR_ASSERTION	NULL provided for p_ctrl
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.

◆ R_FLASH_LP_StartUpAreaSelect()

fsp_err_t R_FLASH_LP_StartUpAreaSelect (flash_ctrl_t *const *p_api_ctrl*, flash_startup_area_swap_t *swap_type*, bool *is_temporary*)

Select which block is used as the startup area block. Implements flash_api_t::startupAreaSelect.

Selects which block - Default (Block 0) or Alternate (Block 1) is used as the startup area block. The provided parameters determine which block will become the active startup block and whether that action will be immediate (but temporary), or permanent subsequent to the next reset. Doing a temporary switch might appear to have limited usefulness. If there is an access window in place such that Block 0 is write protected, then one could do a temporary switch, update the block and switch them back without having to touch the access window.

Return values

FSP_SUCCESS	Start-up area successfully toggled.
FSP_ERR_IN_USE	Flash is busy with an on-going operation.
FSP_ERR_ASSERTION	NULL provided for p_ctrl
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.
FSP_ERR_WRITE_FAILED	Status is indicating a Programming error for the requested operation.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.
FSP_ERR_UNSUPPORTED	Code Flash Programming is not enabled. Cannot set FLASH_STARTUP_AREA_BTFLG when the temporary flag is false.

◆ R_FLASH_LP_CallbackSet()

fsp_err_t R_FLASH_LP_CallbackSet (flash_ctrl_t *const *p_api_ctrl*, void(*)(flash_callback_args_t *) *p_callback*, void const *const *p_context*, flash_callback_args_t *const *p_callback_memory*)

Stub function Implements flash_api_t::callbackSet.

Return values

FSP_ERR_UNSUPPORTED	Function has not been implemented.

◆ R_FLASH_LP_BankSwap()

fsp_err_t R_FLASH_LP_BankSwap (flash_ctrl_t *const *p_api_ctrl*)

Unsupported flash_api_t::bankSwap.

Return values

FSP_ERR_UNSUPPORTED	Module does not support Bank Swap.

◆ R_FLASH_LP_UpdateFlashClockFreq()

fsp_err_t R_FLASH_LP_UpdateFlashClockFreq (flash_ctrl_t *const *p_api_ctrl*)

Indicate to the already open Flash API that the FCLK has changed. Implements flash_api_t::updateFlashClockFreq.

This could be the case if the application has changed the system clock, and therefore the FCLK. Failure to call this function subsequent to changing the FCLK could result in damage to the flash macro.

Return values

FSP_SUCCESS	Start-up area successfully toggled.
FSP_ERR_IN_USE	Flash is busy with an on-going operation.
FSP_ERR_FCLK	Invalid flash clock source frequency.
FSP_ERR_ASSERTION	NULL provided for p_ctrl
FSP_ERR_NOT_OPEN	Flash API has not yet been opened.
FSP_ERR_TIMEOUT	Timed out waiting for the FCU to become ready.

◆ R_FLASH_LP_InfoGet()

fsp_err_t R_FLASH_LP_InfoGet (flash_ctrl_t *const *p_api_ctrl*, flash_info_t *const *p_info*)

Returns the information about the flash regions. Implements flash_api_t::infoGet.

Return values

FSP_SUCCESS	Successful retrieved the request information.
FSP_ERR_ASSERTION	NULL provided for p_ctrl or p_info.
FSP_ERR_NOT_OPEN	The flash is not open.

4.2.27 Graphics LCD Controller (r_glcdc)

Modules

Functions

fsp_err_t	R_GLCDC_Open (display_ctrl_t *const p_api_ctrl, display_cfg_t const *const p_cfg)
fsp_err_t	R_GLCDC_Close (display_ctrl_t *const p_api_ctrl)
fsp_err_t	R_GLCDC_Start (display_ctrl_t *const p_api_ctrl)
fsp_err_t	R_GLCDC_Stop (display_ctrl_t *const p_api_ctrl)
fsp_err_t	R_GLCDC_LayerChange (display_ctrl_t const *const p_api_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t layer)
fsp_err_t	R_GLCDC_BufferChange (display_ctrl_t const *const p_api_ctrl, uint8_t *const framebuffer, display_frame_layer_t layer)
fsp_err_t	R_GLCDC_ColorCorrection (display_ctrl_t const *const p_api_ctrl, display_correction_t const *const p_correction)
fsp_err_t	R_GLCDC_ClutUpdate (display_ctrl_t const *const p_api_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t layer)
fsp_err_t	R_GLCDC_ClutEdit (display_ctrl_t const *const p_api_ctrl, display_frame_layer_t layer, uint8_t index, uint32_t color)
fsp_err_t	R_GLCDC_StatusGet (display_ctrl_t const *const p_api_ctrl, display_status_t *const status)

Detailed Description

Driver for the GLCDC peripheral on RA MCUs. This module implements the Display Interface.

Overview

The GLCDC is a multi-stage graphics output peripheral designed to automatically generate timing and data signals for LCD panels. As part of its internal pipeline the two internal graphics layers can be repositioned, alpha blended, color corrected, dithered and converted to and from a wide variety of pixel formats.

Features

The following features are available:

Feature	Options
Input color formats	ARGB8888, ARGB4444, ARGB1555, RGB888 (32-bit), RGB565, CLUT 8bpp, CLUT 4bpp, CLUT 1bpp
Output color formats	RGB888, RGB666, RGB565, Serial RGB888 (8-bit parallel)
Correction processes	Alpha blending, positioning, brightness and contrast, gamma correction, dithering
Timing signals	Dot clock, Vsync, Hsync, Vertical and horizontal data enable (DE)
Maximum resolution	Up to 1020 x 1008 pixels (dependent on sync signal width)
Maximum dot clock	60MHz for serial RGB mode, 54MHz otherwise
Internal clock divisors	1-9, 12, 16, 24, 32
Interrupts	Vsync (line detect), Layer 1 underflow, Layer 2 underflow
Other functions	Byte-order and endianness control, line repeat function

Configuration

Build Time Configurations for r_glcdc

The following build time configurations are defined in fsp_cfg/r_glcdc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP)	Default (BSP)	If selected, code for

	• Enabled • Disabled		parameter checking is included in the build.
Color Correction	• On • Off	Off	If selected, code to adjust brightness, contrast and gamma settings is included in the build. When disabled all color correction configuration options are ignored.

Configurations for Graphics > Graphics LCD (r_glcdc)

This module can be added to the Stacks tab via New Stack > Graphics > Graphics LCD (r_glcdc).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_display0	Module name.
Interrupts > Callback Function	Name must be a valid C symbol	NULL	A user callback function can be defined here.
Interrupts > Line Detect Interrupt Priority	MCU Specific Options		Select the line detect (Vsync) interrupt priority.
Interrupts > Underflow 1 Interrupt Priority	MCU Specific Options		Select the underflow interrupt priority for layer 1.
Interrupts > Underflow 2 Interrupt Priority	MCU Specific Options		Select the underflow interrupt priority for layer 2.
Input > Graphics Layer 1 > General > Enabled	• Yes • No	Yes	Specify Used if the graphics layer 1 is used. If so a framebuffer will be automatically generated based on the specified height and horizontal stride.
Input > Graphics Layer 1 > General > Horizontal size	Value must be between 16 and 1016	480	Specify the number of horizontal pixels.
Input > Graphics Layer 1 > General > Vertical size	Value must be between 16 and 1020	272	Specify the number of vertical pixels.
Input > Graphics Layer 1 > General > Horizontal position	Must be a valid non-negative integer with a maximum configurable	0	Specify the horizontal offset in pixels of the graphics layer from the

	value of 4091		background layer.
Input > Graphics Layer 1 > General > Vertical position	Must be a valid non-negative integer with a maximum configurable value of 4094	0	Specify the vertical offset in pixels of the graphics layer from the background layer.
Input > Graphics Layer 1 > General > Color format	• ARGB8888 (32-bit) • RGB888 (32-bit) • RGB565 (16-bit) • ARGB1555 (16-bit) • ARGB4444 (16-bit) • CLUT8 (8-bit) • CLUT4 (4-bit) • CLUT1 (1-bit)	RGB565 (16-bit)	Specify the graphics layer Input format. If selecting CLUT formats, you must write the CLUT table data before starting output.
Input > Graphics Layer 1 > General > Line descending mode	• Enabled • Disabled	Disabled	Select Used if the framebuffer starts from the bottom of the line.
Input > Graphics Layer 1 > Background Color > Alpha	Value must be between 0 and 255	255	Based on the alpha value, either the graphics Layer 2 (foreground graphics layer) is blended into the graphics Layer 1 (background graphics layer) or the graphics Layer 1 is blended into the monochrome background layer.
Input > Graphics Layer 1 > Background Color > Red	Value must be between 0 and 255	255	Red component of the background color for layer 1.
Input > Graphics Layer 1 > Background Color > Green	Value must be between 0 and 255	255	Green component of the background color for layer 1.
Input > Graphics Layer 1 > Background Color > Blue	Value must be between 0 and 255	255	Blue component of the background color for layer 1.
Input > Graphics Layer 1 > Framebuffer > Framebuffer name	This property must be a valid C symbol	fb_background	Specify the name for the framebuffer for Layer 1.
Input > Graphics Layer 1 > Framebuffer > Number of framebuffers	Must be a valid non-negative integer with a maximum configurable value of 65535	2	Number of framebuffers allocated for Graphics Layer 1.
Input > Graphics Layer 1 > Framebuffer > Section for framebuffer	Manual Entry	.bss	Specify the section in which to allocate the framebuffer. When Arm

allocation			Compiler 6 is used to place this memory in on-chip SRAM, the section name must be .bss or start with .bss. to avoid consuming unnecessary ROM space.
Input > Graphics Layer 1 > Line Repeat > Enable	• On • Off	Off	Select On if the display will be repeated from a smaller section of the framebuffer.
Input > Graphics Layer 1 > Line Repeat > Repeat count	Must be a valid non-negative integer with a maximum configurable value of 65535 i.e (vertical size) x (lines repeat times) must be equal to the panel vertical size	0	Specify the number of times the image is repeated.
Input > Graphics Layer 1 > Fading > Mode	• None • Fade-in • Fade-out	None	Select the fade method.
Input > Graphics Layer 1 > Fading > Speed	Value must be between 0 and 255	0	Specify the number of frames for the fading transition to complete.
Input > Graphics Layer 2 > General > Enabled	• Yes • No	No	Specify Used if the graphics layer 2 is used. If so a framebuffer will be automatically generated based on the specified height and horizontal stride.
Input > Graphics Layer 2 > General > Horizontal size	Value must be between 16 and 1016	480	Specify the number of horizontal pixels.
Input > Graphics Layer 2 > General > Vertical size	Value must be between 16 and 1020	272	Specify the number of vertical pixels.
Input > Graphics Layer 2 > General > Horizontal position	Must be a valid non-negative integer with a maximum configurable value of 4091	0	Specify the horizontal offset in pixels of the graphics layer from the background layer.
Input > Graphics Layer 2 > General > Vertical position	Must be a valid non-negative integer with a maximum configurable value of 4094	0	Specify the vertical offset in pixels of the graphics layer from the background layer.
Input > Graphics Layer	• ARGB8888	RGB565 (16-bit)	Specify the graphics

2 > General > Color format	(32-bit) • RGB888 (32-bit) • RGB565 (16-bit) • ARGB1555 (16-bit) • ARGB4444 (16-bit) • CLUT8 (8-bit) • CLUT4 (4-bit) • CLUT1 (1-bit)		layer Input format. If selecting CLUT formats, you must write the CLUT table data before starting output.
Input > Graphics Layer 2 > General > Line descending mode	• Enabled • Disabled	Disabled	Select Used if the framebuffer starts from the bottom of the line.
Input > Graphics Layer 2 > Background Color > Alpha	Value must be between 0 and 255	255	Based on the alpha value, either the graphics Layer 2 (foreground graphics layer) is blended into the graphics Layer 1 (background graphics layer) or the graphics Layer 1 is blended into the monochrome background layer.
Input > Graphics Layer 2 > Background Color > Red	Value must be between 0 and 255	255	Red component of the background color for layer 2.
Input > Graphics Layer 2 > Background Color > Green	Value must be between 0 and 255	255	Green component of the background color for layer 2.
Input > Graphics Layer 2 > Background Color > Blue	Value must be between 0 and 255	255	Blue component of the background color for layer 2.
Input > Graphics Layer 2 > Framebuffer > Framebuffer name	This property must be a valid C symbol	fb_foreground	Specify the name for the framebuffer for Layer 2.
Input > Graphics Layer 2 > Framebuffer > Number of framebuffers	Must be a valid non-negative integer with a maximum configurable value of 65535	2	Number of framebuffers allocated for Graphics Layer 2.
Input > Graphics Layer 2 > Framebuffer > Section for framebuffer allocation	Manual Entry	.bss	Specify the section in which to allocate the framebuffer. When Arm Compiler 6 is used to place this memory in on-chip SRAM, the section name must be .bss or start with .bss. to avoid consuming unnecessary ROM

			space.
Input > Graphics Layer 2 > Line Repeat > Enable	• On • Off	Off	Select On if the display will be repeated from a smaller section of the framebuffer.
Input > Graphics Layer 2 > Line Repeat > Repeat count	Must be a valid non-negative integer with a maximum configurable value of 65535 i.e (vertical size) x (lines repeat times) must be equal to the panel vertical size	0	Specify the number of times the image is repeated.
Input > Graphics Layer 2 > Fading > Mode	• None • Fade-in • Fade-out	None	Select the fade method.
Input > Graphics Layer 2 > Fading > Speed	Value must be between 0 and 255	0	Specify the number of frames for the fading transition to complete.
Output > Timing > Horizontal total cycles	Value must be between 24 and 1024	525	Specify the total cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system
Output > Timing > Horizontal active video cycles	Value must be between 16 and 1016	480	Specify the number of active video cycles in a horizontal line (including front and back porch). Set to the number of cycles defined in the data sheet of LCD panel sheet in your system.
Output > Timing > Horizontal back porch cycles	Value must be between 6 and 1006	40	Specify the number of back porch cycles in a horizontal line. Back porch starts from the beginning of Hsync cycles, which means back porch cycles contain Hsync cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system.
Output > Timing > Horizontal sync signal cycles	Value must be between 0 and 1023	1	Specify the number of Hsync signal assertion cycles. Set to the

			number of cycles defined in the data sheet of LCD panel sheet in your system.
Output > Timing > Horizontal sync signal polarity	• Low active • High active	Low active	Select the polarity of Hsync signal to match your system.
Output > Timing > Vertical total lines	Value must be between 20 and 1024	316	Specify number of total lines in a frame (including front and back porch).
Output > Timing > Vertical active video lines	Value must be between 16 and 1020	272	Specify the number of active video lines in a frame.
Output > Timing > Vertical back porch lines	Value must be between 3 and 1007	8	Specify the number of back porch lines in a frame. Back porch starts from the beginning of Vsync lines, which means back porch lines contain Vsync lines.
Output > Timing > Vertical sync signal lines	Value must be between 0 and 1023	1	Specify the Vsync signal assertion lines in a frame.
Output > Timing > Vertical sync signal polarity	• Low active • High active	Low active	Select the polarity of Vsync signal to match to your system.
Output > Timing > Data Enable Signal Polarity	• Low active • High active	High active	Select the polarity of Data Enable signal to match to your system.
Output > Timing > Sync edge	• Rising edge • Falling edge	Rising edge	Select the polarity of Sync signals to match to your system.
Output > Format > Color format	• 24bits RGB888 • 18bits RGB666 • 16bits RGB565 • 8bits serial	16bits RGB565	Specify the graphics layer output format to match to your LCD panel.
Output > Format > Color order	• RGB • BGR	RGB	Select data order for output signal to LCD panel.
Output > Format > Endian	• Little endian • Big endian	Little endian	Select data endianness for output signal to LCD panel.
Output > Background > Alpha	Value must be between 0 and 255	255	Alpha component of the background color.
Output > Background	Value must be between	0	Red component of the

> Red	0 and 255		background color.
Output > Background > Green	Value must be between 0 and 255	0	Green component of the background color.
Output > Background > Blue	Value must be between 0 and 255	0	Blue component of the background color.
CLUT > Enabled	• Yes • No	No	Specify Used if selecting CLUT formats for a graphics layer input format. If used, a buffer (CLUT_buffer) will be automatically generated based on the selected pixel width.
CLUT > Size	Must be a valid non-negative integer with a maximum configurable value of 256	256	Specify the number of entries for the CLUT source data buffer. Each entry consumes 4 bytes (1 word).
TCON > Hsync pin select	• Not used • LCD_TCON0 • LCD_TCON1 • LCD_TCON2 • LCD_TCON3	LCD_TCON0	Select the TCON pin used for the Hsync signal to match to your system.
TCON > Vsync pin select	• Not used • LCD_TCON0 • LCD_TCON1 • LCD_TCON2 • LCD_TCON3	LCD_TCON1	Select TCON pin used for Vsync signal to match to your system.
TCON > Data enable (DE) pin select	• Not used • LCD_TCON0 • LCD_TCON1 • LCD_TCON2 • LCD_TCON3	LCD_TCON2	Select TCON pin used for DataEnable signal to match to your system.
TCON > Panel clock source	• Internal clock (GLCDCLK) • External clock (LCD_EXTCLK)	Internal clock (GLCDCLK)	Choose between an internal GLCDCLK generated from PCLKA or an external clock provided to the LCD_EXTCLK pin.
TCON > Panel clock division ratio	Refer to the RA Configuration tool for available options.	1/24	Select the clock source divider value.
Color Correction > Brightness > Enabled	• Yes • No	No	Enable brightness color correction.
Color Correction > Brightness > Red channel	Value must be between 0 and 1023	512	Red component of the brightness calibration. This value is divided by

			512 to determine gain.
Color Correction > Brightness > Green channel	Value must be between 0 and 1023	512	Green component of the brightness calibration. This value is divided by 512 to determine gain.
Color Correction > Brightness > Blue channel	Value must be between 0 and 1023	512	Blue component of the brightness calibration. This value is divided by 512 to determine gain.
Color Correction > Contrast > Enabled	• Yes • No	No	Enable contrast color correction.
Color Correction > Contrast > Red channel gain	Value must be between 0 and 255	128	Red component of the contrast calibration. This value is divided by 128 to determine gain.
Color Correction > Contrast > Green channel gain	Value must be between 0 and 255	128	Green component of the contrast calibration. This value is divided by 128 to determine gain.
Color Correction > Contrast > Blue channel gain	Value must be between 0 and 255	128	Blue component of the contrast calibration. This value is divided by 128 to determine gain.
Color Correction > Gamma > Tables > Red > Gain > 0	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 1	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 2	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 3	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 4	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain

			of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 5	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 6	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 7	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 8	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 9	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 10	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 11	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 12	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 13	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables >	Manual Entry	1024	Enter a gain value between 0 and 2047

Red > Gain > 14			(corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Gain > 15	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Red > Threshold > 1	Manual Entry	64	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 2	Manual Entry	128	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 3	Manual Entry	192	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 4	Manual Entry	256	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 5	Manual Entry	320	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 6	Manual Entry	384	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 7	Manual Entry	448	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 8	Manual Entry	512	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 9	Manual Entry	576	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 10	Manual Entry	640	Enter a threshold value between the surrounding values less than 1023.

Color Correction > Gamma > Tables > Red > Threshold > 11	Manual Entry	704	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 12	Manual Entry	768	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 13	Manual Entry	832	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 14	Manual Entry	896	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Red > Threshold > 15	Manual Entry	960	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Gain > 0	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 1	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 2	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 3	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 4	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 5	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999,

			respectively).
Color Correction > Gamma > Tables > Green > Gain > 6	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 7	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 8	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 9	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 10	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 11	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 12	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 13	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 14	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Gain > 15	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain

			of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Green > Threshold > 1	Manual Entry	64	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 2	Manual Entry	128	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 3	Manual Entry	192	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 4	Manual Entry	256	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 5	Manual Entry	320	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 6	Manual Entry	384	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 7	Manual Entry	448	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 8	Manual Entry	512	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 9	Manual Entry	576	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 10	Manual Entry	640	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 11	Manual Entry	704	Enter a threshold value between the surrounding values less than 1023.
Color Correction >	Manual Entry	768	Enter a threshold value

Gamma > Tables > Green > Threshold > 12			between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 13	Manual Entry	832	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 14	Manual Entry	896	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Green > Threshold > 15	Manual Entry	960	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Gain > 0	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 1	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 2	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 3	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 4	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 5	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 6	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999,

			respectively).
Color Correction > Gamma > Tables > Blue > Gain > 7	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 8	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 9	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 10	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 11	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 12	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 13	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 14	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Gain > 15	Manual Entry	1024	Enter a gain value between 0 and 2047 (corresponding to gain of 0 and 1.999, respectively).
Color Correction > Gamma > Tables > Blue > Threshold > 1	Manual Entry	64	Enter a threshold value between the surrounding values less

			than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 2	Manual Entry	128	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 3	Manual Entry	192	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 4	Manual Entry	256	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 5	Manual Entry	320	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 6	Manual Entry	384	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 7	Manual Entry	448	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 8	Manual Entry	512	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 9	Manual Entry	576	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 10	Manual Entry	640	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 11	Manual Entry	704	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 12	Manual Entry	768	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables >	Manual Entry	832	Enter a threshold value between the

Blue > Threshold > 13			surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 14	Manual Entry	896	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Tables > Blue > Threshold > 15	Manual Entry	960	Enter a threshold value between the surrounding values less than 1023.
Color Correction > Gamma > Red	• On • Off	Off	Enable gamma color correction for the red channel.
Color Correction > Gamma > Green	• On • Off	Off	Enable gamma color correction for the green channel.
Color Correction > Gamma > Blue	• On • Off	Off	Enable gamma color correction for the blue channel.
Color Correction > Gamma > Table Mode	• Constant • Variable	Variable	Set to Constant to override the automatically-generated RAM gamma tables with a constant declaration using the provided values.
Color Correction > Process order	• Brightness/contrast first • Gamma first	Brightness/contrast first	Select the color correction processing order.
Dithering > Enabled	• Yes • No	No	Enable dithering to reduce the effect of color banding.
Dithering > Mode	• Truncate • Round off • 2x2 Pattern	Truncate	Select the dithering mode.
Dithering > Pattern A	• Pattern 00 • Pattern 01 • Pattern 10 • Pattern 11	Pattern 11	Select the dithering pattern.
Dithering > Pattern B	• Pattern 00 • Pattern 01 • Pattern 10 • Pattern 11	Pattern 11	Select the dithering pattern.
Dithering > Pattern C	• Pattern 00 • Pattern 01 • Pattern 10 • Pattern 11	Pattern 11	Select the dithering pattern.

Dithering > Pattern D	• Pattern 00 • Pattern 01 • Pattern 10 • Pattern 11	Pattern 11	Select the dithering pattern.

Clock Configuration

The peripheral clock for this module is PCLKA.

The dot clock is typically generated from the PLL with a maximum output frequency of 54 MHz in most pixel formats (60 MHz for serial RGB). Optionally, a clock signal can be provided to the LCD_EXTCLK pin for finer framerate control (60 MHz maximum input). With either clock source dividers of 1-9, 12, 16, 24 and 32 may be used. Clocks must be initialized and settled prior to starting this module.

Pin Configuration

This module controls a variety of pins necessary for LCD data and timing signal output:

Pin Name	Function	Notes
LCD_EXTCLK	External clock signal input	The maximum input clock frequency is 60MHz.
LCD_CLK	Dot clock output	The maximum output frequency is 54MHz (60MHz in serial RGB mode).
LCD_DATAn	Pixel data output lines	Pin assignment and color order is based on the output block configuration. See the RA6M3 User's Manual (R01UH0886EJ0100) section 58.1.4 "Output Control for Data Format" for details.
LCD_TCONn	Panel timing signal output	These pins can be configured to output vertical and horizontal synchronization and data valid signals.

Note

There are two banks of pins listed for the GLCDC in the RA6M3 User's Manual (_A and _B). In most cases the _B bank will be used as _A conflicts with SDRAM pins. In either case, it is generally recommended to only use pins from only one bank at a time as this allows for superior signal routing both inside and outside the package. If _A and _B pins must be mixed be sure to note the timing precision penalty detailed in Table 60.33 in in the RA6M3 User's Manual.

Usage Notes

Overview

The GLCDC peripheral is a combination of several sub-peripherals that form a pixel data processing pipeline. Each block passes pixel data to the next but otherwise they are disconnected from one another - in other words, changing timing block parameters does not affect the output generation block configuration and vice versa.

Initial Configuration

During R_GLCDC_Open all configured parameters are set in the GLCDC peripheral fully preparing it for operation. Once opened, calling R_GLCDC_Start is typically all that is needed for basic operation. Background generation, timing and output parameters are not configurable at runtime, though layer control and color correction options can be altered.

Framebuffer Allocation

The framebuffer should be allocated in the highest-speed region available (excluding SRAMHS) without displacing the stack, heap and other program-critical structures. While the RA6M3 does contain a relatively large 640K of on-chip SRAM, for many screen sizes and color depths SDRAM will be required. Regardless of the placement two rules must be followed to ensure correct operation of the GLCDC:

- The framebuffer must be aligned on a 64-byte boundary
- The horizontal stride of the buffer must be a multiple of 64 bytes

Note

Framebuffers allocated through the RA Configuraton tool automatically follow the alignment and size requirements.

If your framebuffer will be placed into internal SRAM please note the following best practices:

- The framebuffer should ideally not be placed in the SRAMHS block of SRAM as there is no speed advantage for doing so. In particular, it is important to ensure the framebuffer does not push the stack or any heaps outside of SRAMHS to preserve CPU performance.
- It is recommended to not cross the boundary between SRAM0 and SRAM1 with a single framebuffer for performance reasons.
- If double-buffering is desired (and possible within SRAM), place one framebuffer in SRAM0 and the other in SRAM1.

If you are using SRAM for the framebuffer, to ensure correct placement you will need to edit the linker script to add new sections. Below is an example of the required edits in the GCC and IAR formats:

GCC Linker

```
/*
  Linker File for RA6M3 MCU
*/
/* Linker script to configure memory regions. */
MEMORY
{
  FLASH (rx)         : ORIGIN = 0x00000000, LENGTH = 0x0200000 /*   2M */
  RAM (rwx)          : ORIGIN = 0x1FFE0000, LENGTH = 0x00A0000 /* 640K */
  FB0 (rwx)          : ORIGIN = 0x20000000, LENGTH = 0x0080000 /* 512K */ // Section
```

```
for framebuffer 0 (or only framebuffer)
  FB1 (rwx)          : ORIGIN = 0x20040000, LENGTH = 0x0040000 /* 256K */ // Section
for framebuffer 1
  DATA_FLASH (rx)    : ORIGIN = 0x40100000, LENGTH = 0x0010000 /* 64K */
  QSPI_FLASH (rx)    : ORIGIN = 0x60000000, LENGTH = 0x4000000 /* 64M */
  SDRAM (rwx)       : ORIGIN = 0x90000000, LENGTH = 0x2000000 /* 32M */
  ID_CODE (rx)       : ORIGIN = 0x0100A150, LENGTH = 0x10    /* 16 bytes */
}
// ...
    .noinit (NOLOAD):
    {
       . = ALIGN(4);
       __noinit_start = .;
       KEEP(*(.noinit*))
       __noinit_end = .;
    } > RAM
 /* Place framebuffer sections first, then the rest of RAM */
    .fb0 :
    {
       . = ALIGN(64);
       __fb0_start = .;
       *(.fb0*);
       __fb0_end = .;
    } > FB0
    .fb1 :
    {
       . = ALIGN(64);
       __fb1_start = .;
       *(.fb1*);
       __fb1_end = .;
    } > FB1
    .bss :
    {
       . = ALIGN(4);
```

```
        __bss_start__ = .;
        *(.bss*)
        *(COMMON)
        . = ALIGN(4);
        __bss_end__ = .;
    } > RAM
// ...
```

IAR Linker

Note

> *The IAR linker does not place items correctly when sections overlap. As a result, it is advised to place your framebuffer(s) as high as possible in the SRAM region in the linker script to maximize the RAM available for everything else. The below is a general case that should be used unedited only if RAM usage (excluding framebuffers) is less than 128K.*

```
/* ... */
/*-Memory Regions-*/
define symbol region_VECT_start    = 0x00000000;
define symbol region_VECT_end      = 0x000003FF;
define symbol region_ROMREG_start = 0x00000400;
define symbol region_ROMREG_end    = 0x000004FF;
define symbol region_FLASH_start   = 0x00000500;
define symbol region_FLASH_end    = 0x001FFFFF;
define symbol region_RAM_start    = 0x1FFE0000;
define symbol region_RAM_end       = 0x1FFFFFFF; /* RAM limited to SRAMHS */
define symbol region_FB0_start    = 0x20000000;
define symbol region_FB0_end       = 0x2003FFFF; /* SRAM0 dedicated to framebuffer 0
*/
define symbol region_FB1_start    = 0x20040000;
define symbol region_FB1_end       = 0x2007FFFF; /* SRAM1 dedicated to framebuffer 1
*/
define symbol region_DF_start      = 0x40100000;
define symbol region_DF_end       = 0x4010FFFF;
define symbol region_SDRAM_start   = 0x90000000;
define symbol region_SDRAM_end    = 0x91FFFFFF;
define symbol region_QSPI_start    = 0x60000000;
```

```
define symbol region_QSPI_end      = 0x63FFFFFF;
/* ... */
define memory mem with size    = 4G;
define region VECT_region       = mem:[from region_VECT_start    to region_VECT_end];
define region ROMREG_region    = mem:[from region_ROMREG_start to region_ROMREG_end];
define region FLASH_region      = mem:[from region_FLASH_start   to
region_FLASH_end];
define region RAM_region       = mem:[from region_RAM_start    to region_RAM_end];
define region FB0_region       = mem:[from region_FB0_start    to region_FB0_end]; /*
Define framebuffer 0 region */
define region FB1_region       = mem:[from region_FB1_start    to region_FB1_end]; /*
Define framebuffer 1 region */
define region DF_region         = mem:[from region_DF_start      to region_DF_end];
define region SDRAM_region      = mem:[from region_SDRAM_start   to
region_SDRAM_end];
define region QSPI_region       = mem:[from region_QSPI_start    to region_QSPI_end];
/* ... */
define block START_OF_RAM with fixed order { rw section .fsp_dtc_vector_table,
                                             block RAM_CODE };
place at start of RAM_region { block START_OF_RAM };
/* Place framebuffer sections first, then the rest of RAM */
place in FB0_region { rw section .fb0 };
place in FB1_region { rw section .fb1 };
place in RAM_region     { rw,
                         rw section .noinit,
                         rw section .bss,
                         rw section .data,
                         rw section HEAP,
                         rw section .stack };
```

Graphics Layers and Timing Parameters

The GLCDC synthesizes graphics data through two configurable graphics layers onto a background layer. The background is used as a solid-color canvas upon which to composite data from the graphics layers. The two graphics layers are blended on top of each other (Layer 2 above Layer 1) and overlaid on the background layer based on their individual configuration. The placement of the layers (as well as LCD timing parameters) are detailed in Figure 1. The colors of the dimensions indicate which element of the display_cfg_t struct is being referenced - for example, the width of the background layer would be [display_cfg].output.htiming.display_cyc as shown in the figure below.

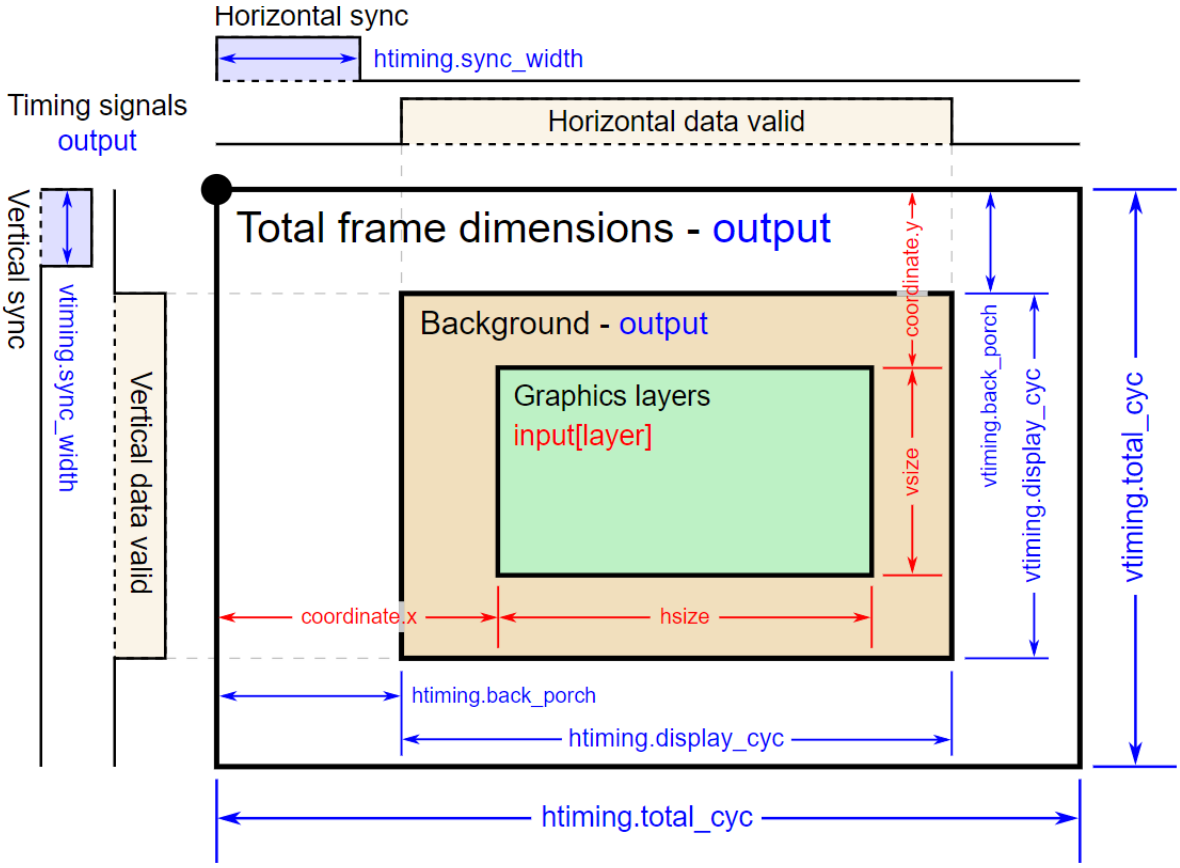


Figure 164: GLCDC layers and timing

Note

The data enable signal (if configured) is the logical AND of the horizontal and vertical data valid signals. In the GLCDC layers and timing figure, only one graphics layer is shown for simplicity. Additionally, in most applications the graphics layer(s) will be the same dimensions as the background layer.

Runtime Configuration Options

Note

All runtime configurations detailed below are also automatically configured during R_GLCDC_Open based on the options selected in the RA Configuration editor.

Blend processing

Control of layer positioning, alpha blending and fading is possible at runtime via R_GLCDC_LayerChange. This function takes a display_runtime_cfg_t parameter which contains the same input and layer elements as the display_cfg_t control block. Refer to the documentation for display_runtime_cfg_t as well as the Examples below to see what options are configurable.

Brightness and contrast

Brightness and contrast correction can be controlled through R_GLCDC_ColorCorrection. The display_correction_t parameter is used to control enabling, disabling and gain values for both corrections as shown below:

```
display_correction_t correction;
/* Brightness values are 0-1023 with +512 offset being neutral */
   correction.brightness.r = 512;
   correction.brightness.g = 512;
   correction.brightness.b = 512;
/* Contrast values are 0-255 representing gain of 0-2 (128 is gain of 1) */
   correction.contrast.r = 128;
   correction.contrast.g = 128;
   correction.contrast.b = 128;
/* Brightness and contrast correction can be enabled or disabled independent of one
another */
   correction.brightness.enable = true;
   correction.contrast.enable   = true;
/* Enable correction */
R_GLCDC_ColorCorrection(&g_disp_ctrl, &correction);
```

Color Look-Up Table (CLUT) Modes

The GLCDC supports 1-, 4- and 8-bit color look-up table (CLUT) formats for input pixel data. By using these modes the framebuffer size in memory can be reduced significantly, allowing even high-resolution displays to be buffered in on-chip SRAM. To enable CLUT modes for a layer the color format must be set to a CLUT mode (either at startup or through R_GLCDC_LayerChange) in addition to filling the CLUT as appropriate via R_GLCDC_ClutUpdate as shown below:

```
/* Basic 4-bit (16-color) CLUT definition */
   uint32_t clut_4[16] =
   {
      0xFF000000,                  // Black
      0xFFFFFFFF,                  // White
      0xFF0000FF,                  // Blue
      0xFF0080FF,                  // Turquoise
      0xFF00FFFF,                  // Cyan
      0xFF00FF80,                  // Mint Green
      0xFF00FF00,                  // Green
```

```
        0xFF80FF00,                   // Lime Green
        0xFFFFFF00,                   // Yellow
        0xFFFF8000,                   // Orange
        0xFFFF0000,                   // Red
        0xFFFF0080,                   // Pink
        0xFFFF00FF,                   // Magenta
        0xFF8000FF,                   // Purple
        0xFF808080,                   // Gray
        0x00000000                     // Transparent
    };
/* Define the CLUT configuration */
display_clut_cfg_t clut_cfg =
    {
        .start = 0,
        .size   = 16,
        .p_base = clut_4
    };
/* Update the CLUT in the GLCDC */
R_GLCDC_ClutUpdate(&g_disp_ctrl, &clut_cfg, DISPLAY_FRAME_LAYER_1);
```

Note

If individual elements of the CLUT must be changed or if elements must be changed one at a time (for instance, when using emWin) it is recommended to use R_GLCDC_ClutEdit to avoid repeated memcpy operations.

Other Configuration Options

Gamma correction

Gamma correction is performed based on a gain curved defined in the RA Configuration editor. Each point on the curve is defined by a threshold and a gain value - each gain value represents a multiplier from 0x-2x (set as 0-2047) that sets the Y-value of the slope of the gain curve, while each threshold interval sets the X-value respectively. For a more detailed explanation refer to the RA6M3 User's Manual (R01UH0886EJ0100) Figure 58.12 "Calculation of gamma correction value" and the related description above it.

When setting threshold values three rules must be followed:

- Each threshold value must be greater than the previous value
- Threshold values must be greater than zero and less than 1024
- Threshold values can equal the previous value only if they are 1023 (maximum)

Note

Gamma correction can only be applied via R_GLCDC_Open.

Dithering

Dithering is a method of pixel blending that allows for smoother transitions between colors when using a limited palette. A full description of dithering is outside the scope of this document. For more information on the pattern settings and how to configure them refer to the RA6M3 User's Manual (R01UH0886EJ0100) Figure 58.13 "Configuration of dither correction block" and Figure 58.14 "Addition value selection method for 2x2 pattern dither".

Bus Utilization

Note

The data provided in this section consists of estimates only. Experimentation is necessary to obtain real-world performance data on any platform.

While the GLCDC is very flexible in size and color depth of displays there are considerations to be made in the tradeoff between color depth, framerate and bus utilization. Below is a table showing estimates of the load at various resolutions, framerates and color depths based on a PLL frequency of 120MHz (default) and an effective SDRAM throughput of 60 MB/sec. Bus utilization percentages are provided for the following use cases:

- Static image display (**GLCDC only**): One read
- Redrawing one framebuffer every display frame (**minimal redraw**): One write, one read
- Blitting one buffer to another then redrawing the entire buffer every display frame (**worst case**): Two writes, three reads

Name	Width	Height	Input color depth (bits)	Framerate (FPS)	Buffer size (bytes)	SRAM use	SRAM bus (GLCDC only)	SDRAM bus (GLCDC only)	SRAM bus (minimal redraw)	SDRAM bus (minimal redraw)	SRAM bus (worst case)	SDRAM bus (worst case)
HQVGA	240	160	8	60	38400	6%	1%	4%	2%	8%	5%	19%
HQVGA	240	160	16	60	76800	12%	2%	8%	4%	15%	10%	38%
QVGA	320	240	16	60	153600	23%	4%	15%	8%	31%	19%	77%
WQVGA	400	240	8	60	96000	15%	2%	10%	5%	19%	12%	48%
WQVGA	400	240	16	60	192000	29%	5%	19%	10%	38%	24%	96%
HVGA	480	320	16	60	307200	47%	8%	31%	15%	61%	38%	154%
VGA	640	480	16	30	614400	—	—	31%	—	61%	—	154%
WVGA	800	480	8	60	384000	59%	10%	38%	19%	77%	48%	192%

WVGA	800	480	16	30	768000	—	—	38%	—	77%	—	192%
WVGA	800	480	32	15	1536000	—	—	38%	—	77%	—	192%
FWVGA	960	480	8	30	460800	70%	6%	23%	12%	46%	29%	115%
FWVGA	960	480	16	30	921600	—	—	46%	—	92%	—	230%
qHD	960	540	8	30	518400	79%	6%	26%	13%	52%	32%	130%

Note

Bus utilization values over 100% indicate that the bandwidth for that bus is exceeded in that scenario and GLCDC underflow and/or dropped frames may result depending on the bus priority setting. ***It is recommended to avoid these scenarios if at all possible by reducing the buffer drawing rate, number of draw/copy operations or the input color depth.*** *Relaxing vertical timing (increasing total line count) or increasing the clock divider are the easiest ways to increase the time per frame.*

Limitations

Developers should be aware of the following limitations when using the GLCDC API:

- Due to a limitation of the GLCDC hardware, if the horizontal back porch is less than the number of pixels in a graphics burst read (64 bytes) for a layer and the layer is positioned at a negative X-value then the layer X-position will be locked to the nearest 64-byte boundary, rounded toward zero.
- The GLCDC peripheral offers a chroma-key function that can be used to perform a green-screen-like color replacement. This functionality is not exposed through the GLCDC API. See the descriptions for GRn.AB7 through .AB9 in the RA6M3 User's Manual for further details.
- Use of R_GLCDC_ClutUpdate and R_GLCDC_ClutEdit may not be mixed on the same frame.

Examples

Basic Example

This is a basic example showing the minimum code required to initialize and start the GLCDC module. If the entire display can be drawn within the vertical blanking period no further code may be necessary.

```
void glcdc_init (void)
{
 fsp_err_t err;
 // Open the GLCDC driver
    err = R_GLCDC_Open(&g_disp_ctrl, &g_disp_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 // Start display output
```

```
    err = R_GLCDC_Start(&g_disp_ctrl);
    assert(FSP_SUCCESS == err);
}
```

Layer Transitions

This example demonstrates how to set up and execute both a sliding and fading layer transition. This is most useful in static image transition scenarios as switching between two actively-drawing graphics layers may require up to four framebuffers to eliminate tearing.

```
volatile uint32_t g_vsync_count = 0;
/* Callback function for GLCDC interrupts */
static void glcdc_callback (display_callback_args_t * p_args)
{
 if (p_args->event == DISPLAY_EVENT_LINE_DETECTION)
    {
       g_vsync_count++;
    }
}
/* Simple wait that returns 1 if no Vsync happened within the timeout period */
uint8_t vsync_wait (void)
{
    uint32_t timeout_timer = GLCDC_VSYNC_TIMEOUT;
    g_vsync_count = 0;
 while (!g_vsync_count && --timeout_timer)
    {
 /* Spin here until DISPLAY_EVENT_LINE_DETECTION callback or timeout */
    }
 return timeout_timer ? 0 : 1;
}
/* Initiate a fade on Layer 2
 *
 * Parameters:
 * direction True for fade in, false for fade out
 * speed number of frames over which to fade
 */
```

```
void glcdc_layer_transition_fade (display_runtime_cfg_t * disp_rt_cfg, bool
direction, uint16_t speed)
{
 fsp_err_t err;
 if (direction)
    {
 /* Set the runtime struct to the desired buffer */
       disp_rt_cfg->input.p_base       = (uint32_t *) g_framebuffer_1;
       disp_rt_cfg->layer.fade_control = DISPLAY_FADE_CONTROL_FADEIN;
    }
 else
    {
       disp_rt_cfg->layer.fade_control = DISPLAY_FADE_CONTROL_FADEOUT;
    }
 /* Ensure speed is at least 1 frame */
 if (!speed)
    {
       speed = 1;
    }
 /* Set the fade speed to the desired change in alpha per frame */
    disp_rt_cfg->layer.fade_speed = UINT8_MAX / speed;
 /* Initiate the fade (will start on the next Vsync) */
    err = R_GLCDC_LayerChange(&g_disp_ctrl, disp_rt_cfg, DISPLAY_FRAME_LAYER_2);
    assert(FSP_SUCCESS == err);
}
/* Slide Layer 1 out to the left while sliding Layer 2 in from the right */
void glcdc_layer_transition_sliding (display_runtime_cfg_t * disp_rt_cfg_in,
display_runtime_cfg_t * disp_rt_cfg_out)
{
 fsp_err_t err;
 /* Set the config for the incoming layer to be just out of bounds on the right side
*/
    disp_rt_cfg_in->input.p_base       = (uint32_t *) g_framebuffer_1;
    disp_rt_cfg_in->layer.coordinate.x = DISPLAY_WIDTH;
```

```
 /* Move layer 1 out and layer 2 in at a fixed rate of 4 pixels per frame */
 for (int32_t x = disp_rt_cfg_in->layer.coordinate.x; x >= 0; x -= 4)
    {
 /* Wait for a Vsync before starting */
       vsync_wait();
 /* Set the X-coordinate of both layers then update them */
       disp_rt_cfg_out->layer.coordinate.x = (int16_t) (x - DISPLAY_WIDTH);
       disp_rt_cfg_in->layer.coordinate.x = (int16_t) x;
       err = R_GLCDC_LayerChange(&g_disp_ctrl, disp_rt_cfg_out, DISPLAY_FRAME_LAYER_1
);
      assert(FSP_SUCCESS == err);
       err = R_GLCDC_LayerChange(&g_disp_ctrl, disp_rt_cfg_in, DISPLAY_FRAME_LAYER_2
);
      assert(FSP_SUCCESS == err);
    }
}
```

Double-Buffering

Using a double-buffer allows one to be output to the LCD while the other is being drawn to memory, eliminating tearing and in some cases reducing bus load. The following is a basic example showing integration of the line detect (Vsync) interrupt to set the timing for buffer swapping and drawing.

```
/* User-defined function to draw the current display to a framebuffer */
void display_draw (uint8_t * framebuffer)
{
 FSP_PARAMETER_NOT_USED(framebuffer);
 /* Draw buffer here */
}
/* This function is an example of a basic double-buffered display thread */
void display_thread (void)
{
    uint8_t * p_framebuffer = NULL;
 fsp_err_t err;
 /* Initialize and start the R_GLCDC module */
    glcdc_init();
```

```
 while (1)
    {
 /* Swap the active framebuffer */
       p_framebuffer = (p_framebuffer == g_framebuffer_0) ? g_framebuffer_1 :
g_framebuffer_0;
 /* Draw the new framebuffer now */
       display_draw(p_framebuffer);
 /* Now that the framebuffer is ready, update the GLCDC buffer pointer on the next
Vsync */
       err = R_GLCDC_BufferChange(&g_disp_ctrl, p_framebuffer, DISPLAY_FRAME_LAYER_1
);
      assert(FSP_SUCCESS == err);
 /* Wait for a Vsync event */
       vsync_wait();
    }
}
```

Data Structures

struct	glcdc_instance_ctrl_t
struct	glcdc_extended_cfg_t

Enumerations

enum	glcdc_clk_src_t
enum	glcdc_panel_clk_div_t
enum	glcdc_tcon_pin_t
enum	glcdc_bus_arbitration_t
enum	glcdc_correction_proc_order_t
enum	glcdc_tcon_signal_select_t
enum	glcdc_clut_plane_t
enum	glcdc_dithering_mode_t
enum	glcdc_dithering_pattern_t

enum	glcdc_input_interface_format_t
enum	glcdc_output_interface_format_t
enum	glcdc_dithering_output_format_t

Data Structure Documentation

◆ glcdc_instance_ctrl_t

struct glcdc_instance_ctrl_t
Display control block. DO NOT INITIALIZE.

◆ glcdc_extended_cfg_t

struct glcdc_extended_cfg_t
GLCDC hardware specific configuration

Data Fields		
glcdc_tcon_pin_t	tcon_hsync	GLCDC TCON output pin select.
glcdc_tcon_pin_t	tcon_vsync	GLCDC TCON output pin select.
glcdc_tcon_pin_t	tcon_de	GLCDC TCON output pin select.
glcdc_correction_proc_order_t	correction_proc_order	Correction control route select.
glcdc_clk_src_t	clksrc	Clock Source selection.
glcdc_panel_clk_div_t	clock_div_ratio	Clock divide ratio for dot clock.
glcdc_dithering_mode_t	dithering_mode	Dithering mode.
glcdc_dithering_pattern_t	dithering_pattern_A	Dithering pattern A.
glcdc_dithering_pattern_t	dithering_pattern_B	Dithering pattern B.
glcdc_dithering_pattern_t	dithering_pattern_C	Dithering pattern C.
glcdc_dithering_pattern_t	dithering_pattern_D	Dithering pattern D.

Enumeration Type Documentation

◆ glcdc_clk_src_t

enum glcdc_clk_src_t
Clock source select

Enumerator	
GLCDC_CLK_SRC_INTERNAL	Internal.
GLCDC_CLK_SRC_EXTERNAL	External.

◆ glcdc_panel_clk_div_t

enum glcdc_panel_clk_div_t	
Clock frequency division ratio	
Enumerator	
GLCDC_PANEL_CLK_DIVISOR_1	Division Ratio 1/1.
GLCDC_PANEL_CLK_DIVISOR_2	Division Ratio 1/2.
GLCDC_PANEL_CLK_DIVISOR_3	Division Ratio 1/3.
GLCDC_PANEL_CLK_DIVISOR_4	Division Ratio 1/4.
GLCDC_PANEL_CLK_DIVISOR_5	Division Ratio 1/5.
GLCDC_PANEL_CLK_DIVISOR_6	Division Ratio 1/6.
GLCDC_PANEL_CLK_DIVISOR_7	Division Ratio 1/7.
GLCDC_PANEL_CLK_DIVISOR_8	Division Ratio 1/8.
GLCDC_PANEL_CLK_DIVISOR_9	Division Ratio 1/9.
GLCDC_PANEL_CLK_DIVISOR_12	Division Ratio 1/12.
GLCDC_PANEL_CLK_DIVISOR_16	Division Ratio 1/16.
GLCDC_PANEL_CLK_DIVISOR_24	Division Ratio 1/24.
GLCDC_PANEL_CLK_DIVISOR_32	Division Ratio 1/32.

◆ glcdc_tcon_pin_t

enum glcdc_tcon_pin_t	
LCD TCON output pin select	
Enumerator	
GLCDC_TCON_PIN_NONE	No output.
GLCDC_TCON_PIN_0	LCD_TCON0.
GLCDC_TCON_PIN_1	LCD_TCON1.
GLCDC_TCON_PIN_2	LCD_TCON2.
GLCDC_TCON_PIN_3	LCD_TCON3.

◆ glcdc_bus_arbitration_t

enum glcdc_bus_arbitration_t	
Bus Arbitration setting	
Enumerator	
GLCDC_BUS_ARBITRATION_ROUNDROBIN	Round robin.
GLCDC_BUS_ARBITRATION_FIX_PRIORITY	Fixed.

◆ glcdc_correction_proc_order_t

enum glcdc_correction_proc_order_t	
Correction circuit sequence control	
Enumerator	
GLCDC_CORRECTION_PROC_ORDER_BRIGHTNESS_CONTRAST2GAMMA	Brightness -> contrast -> gamma correction.
GLCDC_CORRECTION_PROC_ORDER_GAMMA2BRIGHTNESS_CONTRAST	Gamma correction -> brightness -> contrast.

◆ glcdc_tcon_signal_select_t

enum glcdc_tcon_signal_select_t	
Timing signals for driving the LCD panel	
Enumerator	
GLCDC_TCON_SIGNAL_SELECT_STVA_VS	STVA/VS.
GLCDC_TCON_SIGNAL_SELECT_STVB_VE	STVB/VE.
GLCDC_TCON_SIGNAL_SELECT_STHA_HS	STH/SP/HS.
GLCDC_TCON_SIGNAL_SELECT_STHB_HE	STB/LP/HE.
GLCDC_TCON_SIGNAL_SELECT_DE	DE.

◆ glcdc_clut_plane_t

enum glcdc_clut_plane_t	
Clock phase adjustment for serial RGB output	
Enumerator	
GLCDC_CLUT_PLANE_0	GLCDC CLUT plane 0.
GLCDC_CLUT_PLANE_1	GLCDC CLUT plane 1.

◆ glcdc_dithering_mode_t

enum glcdc_dithering_mode_t	
Dithering mode	
Enumerator	
GLCDC_DITHERING_MODE_TRUNCATE	No dithering (truncate)
GLCDC_DITHERING_MODE_ROUND_OFF	Dithering with round off.
GLCDC_DITHERING_MODE_2X2PATTERN	Dithering with 2x2 pattern.

◆ glcdc_dithering_pattern_t

enum glcdc_dithering_pattern_t	
Dithering mode	
Enumerator	
GLCDC_DITHERING_PATTERN_00	2x2 pattern '00'
GLCDC_DITHERING_PATTERN_01	2x2 pattern '01'
GLCDC_DITHERING_PATTERN_10	2x2 pattern '10'
GLCDC_DITHERING_PATTERN_11	2x2 pattern '11'

◆ glcdc_input_interface_format_t

enum glcdc_input_interface_format_t	
Output interface format	
Enumerator	
GLCDC_INPUT_INTERFACE_FORMAT_RGB565	Input interface format RGB565.
GLCDC_INPUT_INTERFACE_FORMAT_RGB888	Input interface format RGB888.
GLCDC_INPUT_INTERFACE_FORMAT_ARGB1555	Input interface format ARGB1555.
GLCDC_INPUT_INTERFACE_FORMAT_ARGB4444	Input interface format ARGB4444.
GLCDC_INPUT_INTERFACE_FORMAT_ARGB8888	Input interface format ARGB8888.
GLCDC_INPUT_INTERFACE_FORMAT_CLUT8	Input interface format CLUT8.
GLCDC_INPUT_INTERFACE_FORMAT_CLUT4	Input interface format CLUT4.
GLCDC_INPUT_INTERFACE_FORMAT_CLUT1	Input interface format CLUT1.

◆ glcdc_output_interface_format_t

enum glcdc_output_interface_format_t	
Output interface format	
Enumerator	
GLCDC_OUTPUT_INTERFACE_FORMAT_RGB888	Output interface format RGB888.
GLCDC_OUTPUT_INTERFACE_FORMAT_RGB666	Output interface format RGB666.
GLCDC_OUTPUT_INTERFACE_FORMAT_RGB565	Output interface format RGB565.
GLCDC_OUTPUT_INTERFACE_FORMAT_SERIAL_RGB	Output interface format Serial RGB.

◆ glcdc_dithering_output_format_t

enum glcdc_dithering_output_format_t	
Dithering output format	
Enumerator	
GLCDC_DITHERING_OUTPUT_FORMAT_RGB888	Dithering output format RGB888.
GLCDC_DITHERING_OUTPUT_FORMAT_RGB666	Dithering output format RGB666.
GLCDC_DITHERING_OUTPUT_FORMAT_RGB565	Dithering output format RGB565.

Function Documentation

◆ R_GLCDC_Open()

fsp_err_t R_GLCDC_Open (display_ctrl_t *const *p_api_ctrl*, display_cfg_t const *const *p_cfg*)

Open GLCDC module. Implements display_api_t::open.

Return values

FSP_SUCCESS	Device was opened successfully.
FSP_ERR_ALREADY_OPEN	Device was already open.
FSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
FSP_ERR_CLOCK_GENERATION	Dot clock cannot be generated from clock source.
FSP_ERR_INVALID_TIMING_SETTING	Invalid panel timing parameter.
FSP_ERR_INVALID_LAYER_SETTING	Invalid layer setting found.
FSP_ERR_INVALID_ALIGNMENT	Input buffer alignment invalid.
FSP_ERR_INVALID_GAMMA_SETTING	Invalid gamma correction setting found
FSP_ERR_INVALID_BRIGHTNESS_SETTING	Invalid brightness correction setting found

Note

PCLKA must be supplied to Graphics LCD Controller (GLCDC) and GLCDC pins must be set in IOPORT before calling this API.

◆ R_GLCDC_Close()

fsp_err_t R_GLCDC_Close (display_ctrl_t *const *p_api_ctrl*)

Close GLCDC module. Implements display_api_t::close.

Return values

FSP_SUCCESS	Device was closed successfully.
FSP_ERR_ASSERTION	Pointer to the control block is NULL.
FSP_ERR_NOT_OPEN	The function call is performed when the driver state is not equal to DISPLAY_STATE_CLOSED.
FSP_ERR_INVALID_UPDATE_TIMING	A function call is performed when the GLCDC is updating register values internally.

Note

This API can be called when the driver is not in DISPLAY_STATE_CLOSED state. It returns an error if the register update operation for the background screen generation block is being held.

◆ R_GLCDC_Start()

fsp_err_t R_GLCDC_Start (display_ctrl_t *const *p_api_ctrl*)

Start GLCDC module. Implements display_api_t::start.

Return values

FSP_SUCCESS	Device was started successfully.
FSP_ERR_NOT_OPEN	GLCDC module has not been opened.
FSP_ERR_ASSERTION	Pointer to the control block is NULL.

Note

This API can be called when the driver is not in DISPLAY_STATE_OPENED status.

◆ R_GLCDC_Stop()

fsp_err_t R_GLCDC_Stop (display_ctrl_t *const *p_api_ctrl*)

Stop GLCDC module. Implements display_api_t::stop.

Return values

FSP_SUCCESS	Device was stopped successfully
FSP_ERR_ASSERTION	Pointer to the control block is NULL
FSP_ERR_INVALID_MODE	Function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING.
FSP_ERR_INVALID_UPDATE_TIMING	The function call is performed while the GLCDC is updating register values internally.

Note

This API can be called when the driver is in the DISPLAY_STATE_DISPLAYING state. It returns an error if the register update operation for the background screen generation blocks, the graphics data I/F blocks, or the output control block is being held.

◆ R_GLCDC_LayerChange()

fsp_err_t R_GLCDC_LayerChange (display_ctrl_t const *const *p_api_ctrl*, display_runtime_cfg_t const *const *p_cfg*, display_frame_layer_t *layer*)

Change layer parameters of GLCDC module at runtime. Implements display_api_t::layerChange.

Return values

FSP_SUCCESS	Changed layer parameters of GLCDC module successfully.
FSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
FSP_ERR_INVALID_MODE	A function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING.
FSP_ERR_INVALID_UPDATE_TIMING	A function call is performed while the GLCDC is updating register values internally.

Note

This API can be called when the driver is in DISPLAY_STATE_DISPLAYING state. It returns an error if the register update operation for the background screen generation blocks or the graphics data I/F block is being held.

◆ R_GLCDC_BufferChange()

fsp_err_t R_GLCDC_BufferChange (display_ctrl_t const *const *p_api_ctrl*, uint8_t *const *framebuffer*, display_frame_layer_t *layer*)

Change the framebuffer pointer for a layer. Implements display_api_t::bufferChange.

Return values

FSP_SUCCESS	Changed layer parameters of GLCDC module successfully.
FSP_ERR_ASSERTION	Pointer to the control block is NULL.
FSP_ERR_INVALID_MODE	A function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING.
FSP_ERR_INVALID_ALIGNMENT	The framebuffer pointer is not 64-byte aligned.
FSP_ERR_INVALID_UPDATE_TIMING	A function call is performed while the GLCDC is updating register values internally.

Note

This API can be called when the driver is in DISPLAY_STATE_OPENED state or higher. It returns an error if the register update operation for the background screen generation blocks or the graphics data I/F block is being held.

◆ R_GLCDC_ColorCorrection()

fsp_err_t R_GLCDC_ColorCorrection (display_ctrl_t const *const *p_api_ctrl*, display_correction_t const *const *p_correction*)

Perform color correction through the GLCDC module. Implements display_api_t::correction.

Return values

FSP_SUCCESS	Color correction by GLCDC module was performed successfully.
FSP_ERR_ASSERTION	Pointer to the control block or the display correction structure is NULL.
FSP_ERR_INVALID_MODE	Function call is performed when the driver state is not DISPLAY_STATE_DISPLAYING.
FSP_ERR_INVALID_UPDATE_TIMING	A function call is performed while the GLCDC is updating registers internally.
FSP_ERR_INVALID_BRIGHTNESS_SETTING	Invalid brightness correction setting found

Note

This API can be called when the driver is in the DISPLAY_STATE_DISPLAYING state. It returns an error if the register update operation for the background screen generation blocks or the output control block is being held.

◆ R_GLCDC_ClutUpdate()

fsp_err_t R_GLCDC_ClutUpdate (display_ctrl_t const *const *p_api_ctrl*, display_clut_cfg_t const *const *p_clut_cfg*, display_frame_layer_t *layer*)

Write an entire color look-up table (CLUT) in the GLCDC module. Implements display_api_t::clut.

Return values

FSP_SUCCESS	CLUT written successfully.
FSP_ERR_ASSERTION	Pointer to the control block or CLUT source data is NULL.
FSP_ERR_INVALID_UPDATE_TIMING	R_GLCDC_ClutEdit was already used to edit the specified CLUT this frame.
FSP_ERR_INVALID_CLUT_ACCESS	Illegal CLUT entry or size is specified.

Note

This API can be called any time. The written data will be used after the next vertical sync event.

◆ R_GLCDC_ClutEdit()

fsp_err_t R_GLCDC_ClutEdit (display_ctrl_t const *const *p_api_ctrl*, display_frame_layer_t *layer*, uint8_t *index*, uint32_t *color*)

Update an element of a color look-up table (CLUT) in the GLCDC module. Implements display_api_t::clutEdit.

Return values

FSP_SUCCESS	CLUT element updated successfully.
FSP_ERR_ASSERTION	Pointer to the control block is NULL.

Note

This API can be called any time. The written data will be used after the next vertical sync event.

◆ R_GLCDC_StatusGet()

fsp_err_t R_GLCDC_StatusGet (display_ctrl_t const *const *p_api_ctrl*, display_status_t *const *p_status*)

Get status of GLCDC module. Implements display_api_t::statusGet.

Return values

FSP_SUCCESS	Got status successfully.
FSP_ERR_ASSERTION	Pointer to the control block or the status structure is NULL.

Note

The GLCDC hardware starts the fading processing at the first Vsync after the previous LayerChange() call is held. Due to this behavior of the hardware, this API may not return DISPLAY_FADE_STATUS_FADING_UNDERWAY as the fading status, if it is called before the first Vsync after LayerChange() is called. In this case, the API returns DISPLAY_FADE_STATUS_PENDING, instead of DISPLAY_FADE_STATUS_NOT_UNDERWAY.

4.2.28 General PWM Timer (r_gpt)

Modules

Functions

fsp_err_t	R_GPT_Open (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)
fsp_err_t	R_GPT_Stop (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_Start (timer_ctrl_t *const p_ctrl)

fsp_err_t	R_GPT_Reset (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_Enable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_Disable (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_PeriodSet (timer_ctrl_t *const p_ctrl, uint32_t const period_counts)
fsp_err_t	R_GPT_DutyCycleSet (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)
fsp_err_t	R_GPT_InfoGet (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)
fsp_err_t	R_GPT_StatusGet (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)
fsp_err_t	R_GPT_CounterSet (timer_ctrl_t *const p_ctrl, uint32_t counter)
fsp_err_t	R_GPT_OutputEnable (timer_ctrl_t *const p_ctrl, gpt_io_pin_t pin)
fsp_err_t	R_GPT_OutputDisable (timer_ctrl_t *const p_ctrl, gpt_io_pin_t pin)
fsp_err_t	R_GPT_AdcTriggerSet (timer_ctrl_t *const p_ctrl, gpt_adc_compare_match_t which_compare_match, uint32_t compare_match_value)
fsp_err_t	R_GPT_PwmOutputDelaySet (timer_ctrl_t *const p_ctrl, gpt_pwm_output_delay_edge_t edge, gpt_pwm_output_delay_setting_t delay_setting, uint32_t const pin)
fsp_err_t	R_GPT_CallbackSet (timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void const *const p_context, timer_callback_args_t *const p_callback_memory)
fsp_err_t	R_GPT_Close (timer_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_PwmOutputDelayInitialize ()

Detailed Description

Driver for the GPT32 and GPT16 peripherals on RA MCUs. This module implements the Timer Interface.

Overview

The GPT module can be used to count events, measure external input signals, generate a periodic interrupt, or output a periodic or PWM signal to a GTIOC pin.

This module supports the GPT peripherals GPT32EH, GPT32E, GPT32, and GPT16. GPT16 is a 16-bit timer. The other peripherals (GPT32EH, GPT32E, and GPT32) are 32-bit timers. The 32-bit timers are all treated the same in this module from the API perspective.

Features

The GPT module has the following features:

- Supports periodic mode, one-shot mode, and PWM mode.
- Supports count source of PCLK, GTETRG pins, GTIOC pins, or ELC events.
- Supports debounce filter on GTIOC pins.
- Signal can be output to a pin.
- Configurable period (counts per timer cycle).
- Configurable duty cycle in PWM mode.
- Supports runtime reconfiguration of period.
- Supports runtime reconfiguration of duty cycle in PWM mode.
- APIs are provided to start, stop, and reset the counter.
- APIs are provided to get the current period, source clock frequency, and count direction.
- APIs are provided to get the current timer status and counter value.
- Supports start, stop, clear, count up, count down, and capture by external sources from GTETRG pins, GTIOC pins, or ELC events.
- Supports symmetric and asymmetric PWM waveform generation.
- Supports automatic addition of dead time.
- Supports generating ELC events to start an ADC scan at a compare match value (see Event Link Controller (r_elc)) and updating the compare match value.
- Supports linking with a POEG channel to automatically disable GPT output when an error condition is detected.
- Supports setting the counter value while the timer is stopped.
- Supports enabling and disabling output pins.
- Supports skipping up to seven overflow/underflow (crest/trough) interrupts at a time
- Supports generating custom PWM waveforms by configuring the pin's output level at each compare match and cycle end.

Selecting a Timer

RA MCUs have two timer peripherals: the General PWM Timer (GPT) and the Asynchronous General Purpose Timer (AGT). When selecting between them, consider these factors:

	GPT	AGT
Low Power Modes	The GPT can operate in sleep mode.	The AGT can operate in all low power modes.
Available Channels	The number of GPT channels is device specific. All currently supported MCUs have at least 7 GPT channels.	All MCUs have 2 AGT channels.
Timer Resolution	All MCUs have at least one 32-bit GPT timer.	The AGT timers are 16-bit timers.
Clock Source	The GPT runs off PCLKD with a configurable divider up to 1024. It can also be configured to count ELC events or external pulses.	The AGT runs off PCLKB, LOCO, or subclock.

Configuration

Build Time Configurations for r_gpt

The following build time configurations are defined in fsp_cfg/r_gpt_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Pin Output Support	• Disabled • Enabled • Enabled with Extra Features	Disabled	Enables or disables support for outputting PWM waveforms on GTIOCx pins. The "Enabled with Extra Features" option enables support for Triangle wave modes and also enables the features located in the "Extra Features" section of each module instance.
Write Protect Enable	• Enabled • Disabled	Disabled	If selected write protection is applied to all GPT channels.
Clock Source	MCU Specific Options		Select either the synchronous clock (PCLKD) or asynchronous clock (GPTCLK) as the source for the GPT core clock and PWM delay circuit.

Configurations for Timers > Timer, General PWM (r_gpt)

This module can be added to the Stacks tab via New Stack > Timers > Timer, General PWM (r_gpt). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_timer0	Module name.
General > Channel	Channel number must exist on this MCU	0	Specify the hardware channel.
General > Mode	• Periodic • One-Shot	Periodic	Mode selection. Periodic: Generates

	• PWM • Triangle-Wave Symmetric PWM • Triangle-Wave Asymmetric PWM • Triangle-Wave Asymmetric PWM (Mode 3)		periodic interrupts or square waves. One-shot: Generate a single interrupt or a pulse wave. Note: One-shot mode is implemented in software. ISRs must be enabled for one-shot even if callback is unused. PWM: Generates basic PWM waveforms. Triangle-Wave Symmetric PWM: Generates symmetric PWM waveforms with duty cycle determined by compare match set during a crest interrupt and updated at the next trough. Triangle-Wave Asymmetric PWM: Generates asymmetric PWM waveforms with duty cycle determined by compare match set during a crest/trough interrupt and updated at the next trough/crest.
General > Period	Value must be a non-negative integer less than or equal to 0x40000000000	0x100000000	Specify the timer period in units selected below. Setting the period to 0x100000000 raw counts results in the maximum period. Set the period to 0x100000000 raw counts for a free running timer or an input capture configuration. The period can be set up to 0x40000000000, which will use a divider of 1024 with the maximum period. If the requested period cannot be achieved, the settings with the largest possible period

			that is less than or equal to the requested period are used. The theoretical calculated period is printed in a comment in the generated timer_cfg_t structure.
General > Period Unit	• Raw Counts • Nanoseconds • Microseconds • Milliseconds • Seconds • Hertz • Kilohertz	Raw Counts	Unit of the period specified above
Output > Custom Waveform > GTIOA > Initial Output Level	• Pin Level Low • Pin Level High	Pin Level Low	Set the initial output level of GTIOCxA.
Output > Custom Waveform > GTIOA > Cycle End Output Level	• Pin Level Retain • Pin Level Low • Pin Level High • Pin Level Toggle	Pin Level Retain	Set the output level of GTIOCxA at cycle end.
Output > Custom Waveform > GTIOA > Compare Match Output Level	• Pin Level Retain • Pin Level Low • Pin Level High • Pin Level Toggle	Pin Level Retain	Set the output level of GTIOCxA at compare match.
Output > Custom Waveform > GTIOA > Retain Output Level at Count Stop	• Disabled • Enabled	Disabled	Retain the current GTIOxA output level when counting is stopped.
Output > Custom Waveform > GTIOB > Initial Output Level	• Pin Level Low • Pin Level High	Pin Level Low	Set the initial output level of GTIOCxB.
Output > Custom Waveform > GTIOB > Cycle End Output Level	• Pin Level Retain • Pin Level Low • Pin Level High • Pin Level Toggle	Pin Level Retain	Set the output level of GTIOCxB at cycle end.
Output > Custom Waveform > GTIOB > Compare Match Output Level	• Pin Level Retain • Pin Level Low • Pin Level High • Pin Level Toggle	Pin Level Retain	Set the output level of GTIOCxB at compare match.
Output > Custom Waveform > GTIOB > Retain Output Level at Count Stop	• Disabled • Enabled	Disabled	Retain the current GTIOxB output level when counting is stopped.

Output > Custom Waveform > Custom Waveform Enable	• Disabled • Enabled	Disabled	Enable custom waveform configuration.
Output > Duty Cycle Percent (only applicable in PWM mode)	Value must be between 0 and 100	50	Specify the timer duty cycle percent. Only used in PWM mode.
Output > GTIOCA Output Enabled	• True • False	False	Enable the output of GTIOCA on a pin.
Output > GTIOCA Stop Level	• Pin Level Low • Pin Level High	Pin Level Low	Select the behavior of the output pin when the timer is stopped.
Output > GTIOCB Output Enabled	• True • False	False	Enable the output of GTIOCB on a pin.
Output > GTIOCB Stop Level	• Pin Level Low • Pin Level High	Pin Level Low	Select the behavior of the output pin when the timer is stopped.
Input > Count Up Source	MCU Specific Options		Select external source that will increment the counter. If any count up source is selected, the timer will count the external sources only. It will not count PCLKD cycles.
Input > Count Down Source	MCU Specific Options		Select external source that will decrement the counter. If any count down source is selected, the timer will count the external sources only. It will not count PCLKD cycles.
Input > Start Source	MCU Specific Options		Select external source that will start the timer. For pulse width measurement, set the Start Source and the Clear Source to the trigger edge (the edge to start the measurement), and set the Stop Source and Capture Source (either A or B) to the opposite edge (the edge to stop the measurement). For pulse period

			measurement, set the Start Source, the Clear Source, and the Capture Source (either A or B) to the trigger edge (the edge to start the measurement).
Input > Stop Source	MCU Specific Options		Select external source that will stop the timer.
Input > Clear Source	MCU Specific Options		Select external source that will clear the timer.
Input > Capture A Source	MCU Specific Options		Select external source that will trigger a capture A event.
Input > Capture B Source	MCU Specific Options		Select external source that will trigger a capture B event.
Input > Noise Filter A Sampling Clock Select	• No Filter • Filter PCLKD / 1 • Filter PCLKD / 4 • Filter PCLKD / 16 • Filter PCLKD / 64	No Filter	Select the input filter for GTIOCA.
Input > Noise Filter B Sampling Clock Select	• No Filter • Filter PCLKD / 1 • Filter PCLKD / 4 • Filter PCLKD / 16 • Filter PCLKD / 64	No Filter	Select the input filter for GTIOCB.
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function can be specified here. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the timer period elapses
Interrupts > Overflow/Crest Interrupt Priority	MCU Specific Options		Select the overflow interrupt priority. This is the crest interrupt for triangle-wave PWM.
Interrupts > Capture A Interrupt Priority	MCU Specific Options		Select the interrupt priority for capture A.
Interrupts > Capture B	MCU Specific Options		Select the interrupt

Interrupt Priority			priority for capture B.
Interrupts > Underflow/Trough Interrupt Priority	MCU Specific Options		Select the interrupt priority for the trough interrupt (triangle-wave PWM only).
Extra Features > Output Disable > POEG Link	• POEG Channel 0 • POEG Channel 1 • POEG Channel 2 • POEG Channel 3	POEG Channel 0	Select which POEG to link this GPT channel to.
Extra Features > Output Disable > Output Disable POEG Trigger	• Dead Time Error • GTIOCA and GTIOCB High Level • GTIOCA and GTIOCB Low Level		Select which errors send an output disable trigger to POEG. Dead time error is only available on GPT32E and GPT32EH variants.
Extra Features > Output Disable > GTIOCA Disable Setting	• Disable Prohibited • Set Hi Z • Level Low • Level High	Disable Prohibited	Select the disable setting for GTIOCA.
Extra Features > Output Disable > GTIOCB Disable Setting	• Disable Prohibited • Set Hi Z • Level Low • Level High	Disable Prohibited	Select the disable setting for GTIOCB.
Extra Features > ADC Trigger > Start Event Trigger (GPTE/GPTEH only)	• Trigger Event A/D Converter Start Request A During Up Counting • Trigger Event A/D Converter Start Request A During Down Counting • Trigger Event A/D Converter Start Request B During Up Counting • Trigger Event A/D Converter Start Request B During Down Counting		Select which A/D converter start request interrupts to generate and at which point in the cycle to generate them. This value only applies to the GPT32E and GPT32EH variants.

Extra Features > Dead Time > Dead Time Count Up (Raw Counts)	Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff).	0	Select the dead time to apply during up counting. This value also applies during down counting for the GPT32 and GPT16 variants.
Extra Features > Dead Time > Dead Time Count Down (Raw Counts) (GPTE/GPTEH only)	Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff).	0	Select the dead time to apply during down counting. This value only applies to the GPT32E and GPT32EH variants.
Extra Features > ADC Trigger (GPTE/GPTEH only) > ADC A Compare Match (Raw Counts)	Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff).	0	Select the compare match value that generates a GPTn AD TRIG A event. This value only applies to the GPT32E and GPT32EH variants.
Extra Features > ADC Trigger (GPTE/GPTEH only) > ADC B Compare Match (Raw Counts)	Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff).	0	Select the compare match value that generates a GPTn AD TRIG B event. This value only applies to the GPT32E and GPT32EH variants.
Extra Features > Interrupt Skipping (GPTE/GPTEH only) > Interrupt to Count	• None • Overflow and Underflow (sawtooth) • Crest (triangle) • Trough (triangle)	None	Select the count source for interrupt skipping. The interrupt skip counter increments after each source event. All crest/overflow and trough/underflow interrupts are skipped when the interrupt skip counter is non-zero. This value only applies to the GPT32E and GPT32EH variants.
Extra Features > Interrupt Skipping (GPTE/GPTEH only) > Interrupt Skip Count	• 0 • 1 • 2 • 3 • 4 • 5 • 6 • 7	0	Select the number of interrupts to skip. This value only applies to the GPT32E and GPT32EH variants.
Extra Features > Interrupt Skipping	• None • ADC A Compare	module.driver.timer.interrupt_skip.adc.none	Select ADC events to suppress when the

(GPTE/GPTEH only) > Skip ADC Events	Match • ADC B Compare Match • ADC A and B Compare Match		interrupt skip count is not zero. This value only applies to the GPT32E and GPT32EH variants.
Extra Features > Extra Features	• Enabled • Disabled	Disabled	Select whether to enable extra features on this channel.

Clock Configuration

The GPT clock is based on the PCLKD frequency. You can set the PCLKD frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

Pin Configuration

This module can use GTETRGA, GTETRGB, GTETRGC, GTETRGD, GTIOCA and GTIOCB pins as count sources.

This module can use GTIOCA and GTIOCB pins as output pins for periodic or PWM signals.

This module can use GTIOCA and GTIOCB as input pins to measure input signals.

Usage Notes

Maximum Period for GPT32

The RA Configuration editor will automatically calculate the period count value and source clock divider based on the selected period time, units and clock speed.

When the selected period unit is "Raw counts", the maximum period setting is 0x40000000000 on a 32-bit timer or 0x0x4000000 on a 16-bit timer. This will configure the timer with the maximum period and a count clock divisor of 128.

Note

When manually changing the timer period counts the maximum value for a 32-bit GPT is 0x100000000. This number overflows the 32-bit value for timer_cfg_t::period_counts. To configure the timer for the maximum period, set timer_cfg_t::period_counts to 0.

Updating Period and Duty Cycle

The period and duty cycle are updated after the next counter overflow after calling R_GPT_PeriodSet() or R_GPT_DutyCycleSet(). To force them to update before the next counter overflow, call R_GPT_Reset() while the counter is running.

One-Shot Mode

The GPT timer does not support one-shot mode natively. One-shot mode is achieved by stopping the timer in the interrupt service routine before the callback is called. If the interrupt is not serviced before the timer period expires again, the timer generates more than one event. The callback is only called once in this case, but multiple events may be generated if the timer is linked to the Data Transfer Controller (r_dtc).

One-Shot Mode Output

The output waveform in one-shot mode is one PCLKD cycle less than the configured period. The configured period must be at least 2 counts to generate an output pulse.

Examples of one-shot signals that can be generated by this module are shown below:

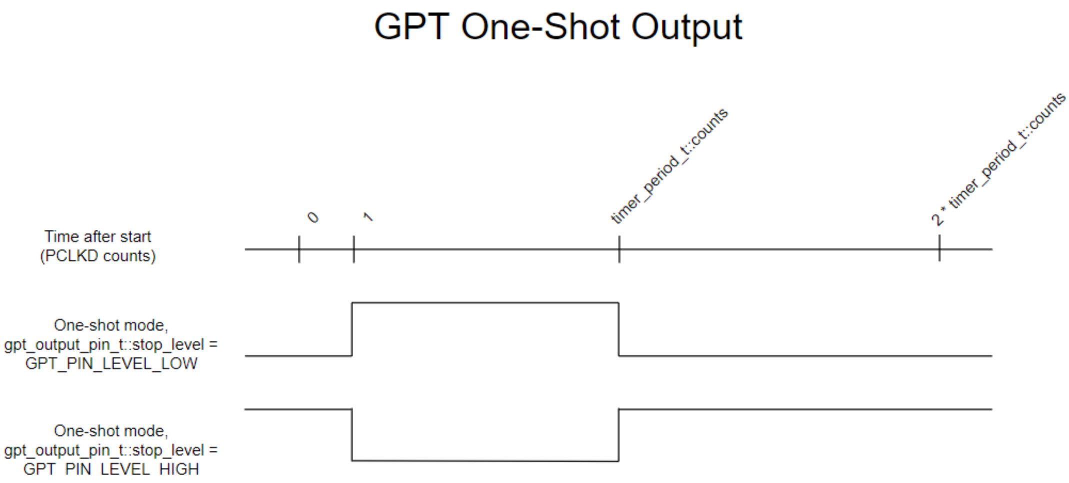


Figure 165: GPT One-Shot Output

Periodic Output

The GTIOC pin toggles twice each time the timer expires in periodic mode. This is achieved by defining a PWM wave at a 50 percent duty cycle so that the period of the resulting square wave (from rising edge to rising edge) matches the period of the GPT timer. Since the periodic output is actually a PWM output, the time at the stop level is one cycle shorter than the time opposite the stop level for odd period values.

Examples of periodic signals that can be generated by this module are shown below:

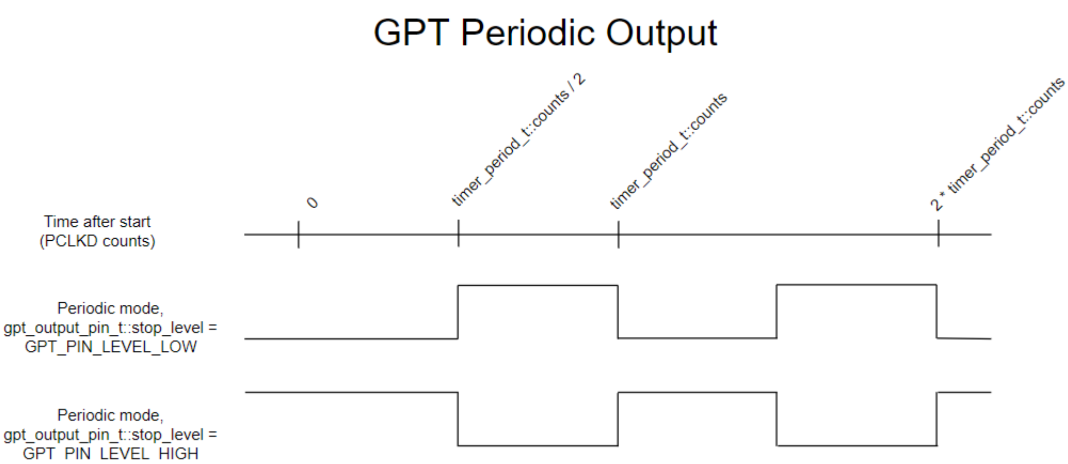


Figure 166: GPT Periodic Output

PWM Output

The PWM output signal is high at the beginning of the cycle and low at the end of the cycle.

Examples of PWM signals that can be generated by this module are shown below:

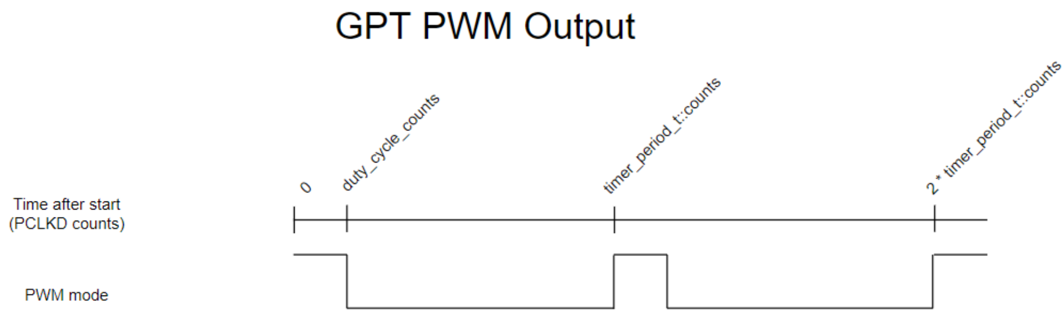


Figure 167: GPT PWM Output

Triangle-Wave PWM Output

Examples of PWM signals that can be generated by this module are shown below. The duty_cycle_counts can be modified using R_GPT_DutyCycleSet() in the crest interrupt and updated at the following trough for symmetric PWM or modified in both the crest/trough interrupts and updated at the following trough/crest for asymmetric PWM.

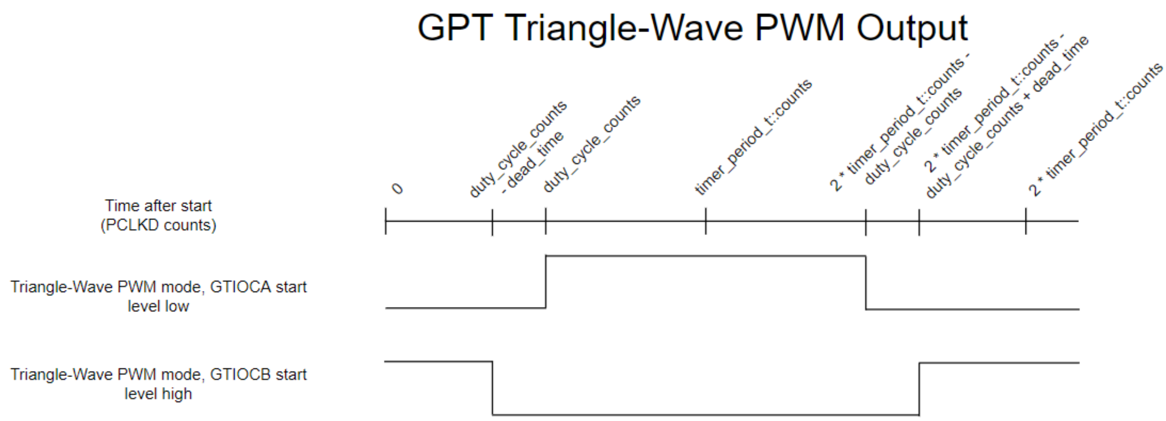


Figure 168: GPT Triangle-Wave PWM Output

PWM Output Delay Circuit

On select MCUs, an additional PWM output delay circuit can be configured in order to fine tune the rising and falling edge delays in increments of 1/32 times the period of the GPT core clock. The PWM output delay function must be configured prior to initializing the GPT channels using R_GPT_PwmOutputDelayInitialize.

Note

1. In Saw-wave PWM mode, the output delay setting cannot be changed while the capture compare setting (GTCCRn) is greater than or equal the period setting (GTPR) - 2.
2. In Triangle PWM modes, the output delay setting cannot be changed while the counter is counting down, and the capture compare setting (GTCCRn) is less than or equal to 2.
3. When the PWM Output Delay Circuit is enabled, the PWM signal is delayed by 3 GPT core clock cycles.
4. When the GPTCLK is used as the GPT core clock, the following delay is required between writes to the rising or falling edge output delay setting for a given pin: Write_Interval[ns] = Period_of_PCLKA [ns] × 6 + Period_of_GPTCLK [ns] × 4).

Event Counting

Event counting can be done by selecting up or down counting sources from GTETRG pins, ELC events, or GTIOC pins. In event counting mode, the GPT counter is not affected by PCLKD.

Note

In event counting mode, the application must call R_GPT_Start() to enable event counting. The counter will not change after calling R_GPT_Start() until an event occurs.

Pulse Measurement

If the capture edge occurs before the start edge in pulse measurement, the first capture is invalid (0).

Controlling GPT with GTETRG Edges

The GPT timer can be configured to stop, start, clear, count up, or count down when a GTETRG rising or falling edge occurs.

Note

The GTETRG pins are shared by all GPT channels.
GTETRG pins require POEG to be on (example code for this is provided in GPT Free Running Counter Example).
If input filtering is required on the GTETRG pins, that must also be handled outside this module.

Controlling GPT with ELC Events

The GPT timer can be configured to stop, start, clear, count up, or count down when an ELC event occurs.

Note

The configurable ELC GPT sources are shared by all GPT channels.
The event links for the ELC must be configured outside this module.

Triggering ELC Events with GPT

The GPT timer can trigger the start of other peripherals. The Event Link Controller (r_elc) guide provides a list of all available peripherals.

Enabling External Sources for Start, Stop, Clear, or Capture

R_GPT_Enable() must be called when external sources are used for start, stop, clear, or capture.

Interrupt Skipping

When an interrupt skipping source is selected a hardware counter will increment each time the selected event occurs. Each interrupt past the first (up to the specified skip count) will be suppressed. If ADC events are selected for skipping they will also be suppressed except during the timer period leading to the selected interrupt skipping event (see below diagram).

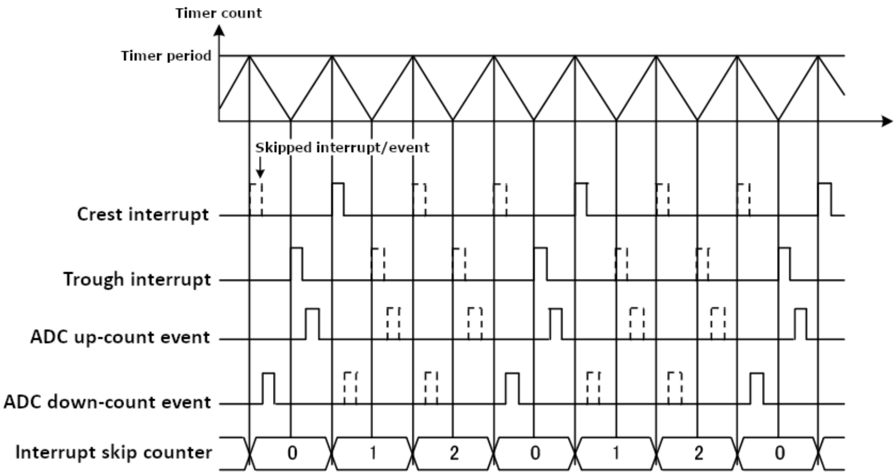


Figure 169: Crest interrupt skipping in triangle-wave PWM modes (skip count 2)

Complementary Output

By using the Custom Waveform option the output pins can be made to output complementary waveforms. To ensure these waveforms stay in sync, the duty cycle for both pins can be set simultaneously by calling R_GPT_DutyCycleSet once with a pin parameter of GPT_IO_PIN_GTIOCA_AND_GTIOCB.

Note

> *The pin level for 0% and 100% duty cycle is determined by the Cycle End Output Level in normal PWM mode and the Initial Output Level in triangle PWM modes. 100% duty will output the configured level and 0% will output the opposite. Do not use Pin Level Toggle or Pin Level Retain for the Cycle End Output Level if normal PWM waveforms are desired.*

Examples

GPT Basic Example

This is a basic example of minimal use of the GPT in an application.

```
void gpt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
}
```

GPT Callback Example

This is an example of a timer callback.

```
/* Example callback called when timer expires. */
void timer_callback (timer_callback_args_t * p_args)
{
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* Add application code to be called periodically here. */
    }
}
```

GPT Free Running Counter Example

To use the GPT as a free running counter, select periodic mode and set the the Period to 0xFFFFFFFF for a 32-bit timer or 0xFFFF for a 16-bit timer.

```
void gpt_counter_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) If event count mode is used to count edges on a GTETRG pin, POEG must
be started to use GTETRG.
  * Reference Note 1 of Table 23.2 "GPT functions" in the RA6M3 manual
R01UH0886EJ0100. */
 R_BSP_MODULE_START(FSP_IP_POEG, 0U);
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* (Optional) Stop the timer. */
    (void) R_GPT_Stop(&g_timer0_ctrl);
```

```
 /* Read the current counter value. Counter value is in status.counter. */
 timer_status_t status;
    (void) R_GPT_StatusGet(&g_timer0_ctrl, &status);
}
```

GPT Input Capture Example

This is an example of using the GPT to capture pulse width or pulse period measurements.

```
/* Example callback called when a capture occurs. */
uint64_t g_captured_time    = 0U;
uint32_t g_capture_overflows = 0U;
void timer_capture_callback (timer_callback_args_t * p_args)
{
 if ((TIMER_EVENT_CAPTURE_A == p_args->event) || (TIMER_EVENT_CAPTURE_B ==
p_args->event))
    {
 /* (Optional) Get the current period if not known. */
 timer_info_t info;
       (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
       uint64_t period = info.period_counts;
 /* The maximum period is one more than the maximum 32-bit number, but will be
reflected as 0 in
  * timer_info_t::period_counts. */
 if (0U == period)
      {
            period = UINT32_MAX + 1U;
      }
       g_captured_time    = (period * g_capture_overflows) + p_args->capture;
       g_capture_overflows = 0U;
    }
 if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
 /* An overflow occurred during capture. This must be accounted for at the
application layer. */
```

```
        g_capture_overflows++;
    }
}
void gpt_capture_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable captures. Captured values arrive in the interrupt. */
    (void) R_GPT_Enable(&g_timer0_ctrl);
 /* (Optional) Disable captures. */
    (void) R_GPT_Disable(&g_timer0_ctrl);
}
```

GPT Period Update Example

This an example of updating the period.

```
#define GPT_EXAMPLE_MSEC_PER_SEC (1000)
#define GPT_EXAMPLE_DESIRED_PERIOD_MSEC (20)
/* This example shows how to calculate a new period value at runtime. */
void gpt_period_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* Get the source clock frequency (in Hz). There are 3 ways to do this in FSP:
  * - If the PCLKD frequency has not changed since reset, the source clock frequency
is
```

```
  * BSP_STARTUP_PCLKD_HZ >> timer_cfg_t::source_div
  * - Use the R_GPT_InfoGet function (it accounts for the divider).
  * - Calculate the current PCLKD frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) and right shift
  * by timer_cfg_t::source_div.
  *
  * This example uses the 3rd option (R_FSP_SystemClockHzGet).
  */
    uint32_t pclkd_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkd_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) pclkd_freq_hz * GPT_EXAMPLE_DESIRED_PERIOD_MSEC) /
GPT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. */
    err = R_GPT_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
}
```

GPT Duty Cycle Update Example

This an example of updating the duty cycle.

```
#define GPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT (25)
#define GPT_EXAMPLE_MAX_PERCENT (100)
/* This example shows how to calculate a new duty cycle value at runtime. */
void gpt_duty_cycle_calculation_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
 /* Handle any errors. This function should be defined by the user. */
```

```
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_Start(&g_timer0_ctrl);
 /* Get the current period setting. */
 timer_info_t info;
    (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
    uint32_t current_period_counts = info.period_counts;
 /* Calculate the desired duty cycle based on the current period. Note that if the
period could be larger than
  * UINT32_MAX / 100, this calculation could overflow. A cast to uint64_t is used to
prevent this. The cast is
  * not required for 16-bit timers. */
    uint32_t duty_cycle_counts =
       (uint32_t) (((uint64_t) current_period_counts *
GPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                   GPT_EXAMPLE_MAX_PERCENT);
 /* Set the calculated duty cycle. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, GPT_IO_PIN_GTIOCB);
    assert(FSP_SUCCESS == err);
}
```

GPT A/D Converter Start Request Example

This is an example of using the GPT to start the ADC at a configurable A/D converter compare match value.

```
#if ((1U << GPT_EXAMPLE_CHANNEL) & (BSP_FEATURE_GPTEH_CHANNEL_MASK |
BSP_FEATURE_GPTE_CHANNEL_MASK))
/* This example shows how to configure the GPT to generate an A/D start request at an
A/D start request compare
 * match value. This example can only be used with GPTE or GPTEH variants. */
void gpt_adc_start_request_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize and configure the ELC. */
```

```
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Configure the ELC to start a scan on ADC unit 0 when GPT channel 0. Note: This is
typically configured in
  * g_elc_cfg and already set during R_ELC_Open. */
    err = R_ELC_LinkSet(&g_elc_ctrl, ELC_PERIPHERAL_ADC0, ELC_EVENT_GPT0_AD_TRIG_A);
    assert(FSP_SUCCESS == err);
 /* Globally enable ELC events. */
    err = R_ELC_Enable(&g_elc_ctrl);
    assert(FSP_SUCCESS == err);
 /* Initialize the ADC to start a scan based on an ELC event trigger. Set
adc_cfg_t::trigger to
  * ADC_TRIGGER_SYNC_ELC. */
    err = R_ADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_ADC_ScanCfg(&g_adc0_ctrl, &g_adc0_channel_cfg);
    assert(FSP_SUCCESS == err);
 /* Enable ELC triggers by calling R_ADC_ScanStart(). */
    (void) R_ADC_ScanStart(&g_adc0_ctrl);
 /* Initializes the GPT module. Configure gpt_extended_pwm_cfg_t::adc_trigger to set
when the A/D start request
  * is generated. Set gpt_extended_pwm_cfg_t::adc_a_compare_match to set the desired
compare match value. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
    assert(FSP_SUCCESS == err);
 /* Start the timer. A/D converter start request events are generated each time the
counter is equal to the
  * A/D start request compare match value. */
    (void) R_GPT_Start(&g_timer0_ctrl);
}
#endif
```

Data Structures

struct	gpt_output_pin_t

struct	gpt_gtior_setting_t
struct	gpt_instance_ctrl_t
struct	gpt_extended_pwm_cfg_t
struct	gpt_extended_cfg_t

Enumerations

enum	gpt_io_pin_t
enum	gpt_pin_level_t
enum	gpt_source_t
enum	gpt_capture_filter_t
enum	gpt_adc_trigger_t
enum	gpt_poeg_link_t
enum	gpt_output_disable_t
enum	gpt_gtioc_disable_t
enum	gpt_adc_compare_match_t
enum	gpt_interrupt_skip_source_t
enum	gpt_interrupt_skip_count_t
enum	gpt_interrupt_skip_adc_t
enum	gpt_pwm_output_delay_setting_t
enum	gpt_pwm_output_delay_edge_t

Data Structure Documentation

◆ gpt_output_pin_t

struct gpt_output_pin_t

Configurations for output pins.

Data Fields		
bool	output_enabled	Set to true to enable output, false to disable output.

gpt_pin_level_t	stop_level	Select a stop level from gpt_pin_level_t.

◆ gpt_gtior_setting_t

struct gpt_gtior_setting_t
Custom GTIOR settings used for configuring GTIOCxA and GTIOCxB pins.

◆ gpt_instance_ctrl_t

struct gpt_instance_ctrl_t
Channel control block. DO NOT INITIALIZE. Initialization occurs when timer_api_t::open is called.

◆ gpt_extended_pwm_cfg_t

struct gpt_extended_pwm_cfg_t		
GPT extension for advanced PWM features.		
Data Fields		
uint8_t	trough_ipl	Trough interrupt priority.
IRQn_Type	trough_irq	Trough interrupt.
gpt_poeg_link_t	poeg_link	Select which POEG channel controls output disable for this GPT channel.
gpt_output_disable_t	output_disable	Select which trigger sources request output disable from POEG.
gpt_adc_trigger_t	adc_trigger	Select trigger sources to start A/D conversion.
uint32_t	dead_time_count_up	Set a dead time value for counting up.
uint32_t	dead_time_count_down	Set a dead time value for counting down (available on GPT32E and GPT32EH only)
uint32_t	adc_a_compare_match	Select the compare match value used to trigger an A/D conversion start request using ELC_EVENT_GPT<channel>_AD_TRIG_A.
uint32_t	adc_b_compare_match	Select the compare match value used to trigger an A/D conversion start request using ELC_EVENT_GPT<channel>_AD_TRIG_B.
gpt_interrupt_skip_source_t	interrupt_skip_source	Interrupt source to count for interrupt skipping.

gpt_interrupt_skip_count_t	interrupt_skip_count	Number of interrupts to skip between events.
gpt_interrupt_skip_adc_t	interrupt_skip_adc	ADC events to skip when interrupt skipping is enabled.
gpt_gtioc_disable_t	gtioca_disable_setting	DEPRECATED - Select how to configure GTIOCA when output is disabled.
gpt_gtioc_disable_t	gtiocb_disable_setting	DEPRECATED - Select how to configure GTIOCB when output is disabled.

◆ gpt_extended_cfg_t

struct gpt_extended_cfg_t		
GPT extension configures the output pins for GPT.		
Data Fields		
gpt_output_pin_t	gtioca	DEPRECATED - Configuration for GPT I/O pin A.
gpt_output_pin_t	gtiocb	DEPRECATED - Configuration for GPT I/O pin B.
gpt_source_t	start_source	Event sources that trigger the timer to start.
gpt_source_t	stop_source	Event sources that trigger the timer to stop.
gpt_source_t	clear_source	Event sources that trigger the timer to clear.
gpt_source_t	capture_a_source	Event sources that trigger capture of GTIOCA.
gpt_source_t	capture_b_source	Event sources that trigger capture of GTIOCB.
gpt_source_t	count_up_source	Event sources that trigger a single up count. If GPT_SOURCE_NONE is selected for both count_up_source and count_down_source, then the timer count source is PCLK.
gpt_source_t	count_down_source	Event sources that trigger a single down count. If GPT_SOURCE_NONE is selected for both count_up_source and count_down_source, then the timer count source is PCLK.
gpt_capture_filter_t	capture_filter_gtioca	
gpt_capture_filter_t	capture_filter_gtiocb	

uint8_t	capture_a_ipl	Capture A interrupt priority.
uint8_t	capture_b_ipl	Capture B interrupt priority.
IRQn_Type	capture_a_irq	Capture A interrupt.
IRQn_Type	capture_b_irq	Capture B interrupt.
gpt_extended_pwm_cfg_t const *	p_pwm_cfg	Advanced PWM features, optional.
gpt_gtior_setting_t	gtior_setting	Custom GTIOR settings used for configuring GTIOCxA and GTIOCxB pins.

Enumeration Type Documentation

◆ gpt_io_pin_t

enum gpt_io_pin_t	
Input/Output pins, used to select which duty cycle to update in R_GPT_DutyCycleSet().	
Enumerator	
GPT_IO_PIN_GTIOCA	GTIOCA.
GPT_IO_PIN_GTIOCB	GTIOCB.
GPT_IO_PIN_GTIOCA_AND_GTIOCB	GTIOCA and GTIOCB.
GPT_IO_PIN_TROUGH	Used in R_GPT_DutyCycleSet when Triangle-wave PWM Mode 3 is selected.
GPT_IO_PIN_CREST	Used in R_GPT_DutyCycleSet when Triangle-wave PWM Mode 3 is selected.

◆ gpt_pin_level_t

enum gpt_pin_level_t	
Level of GPT pin	
Enumerator	
GPT_PIN_LEVEL_LOW	Pin level low.
GPT_PIN_LEVEL_HIGH	Pin level high.

◆ gpt_source_t

enum gpt_source_t

Sources can be used to start the timer, stop the timer, count up, or count down. These enumerations represent a bitmask. Multiple sources can be ORed together.

Enumerator	
GPT_SOURCE_NONE	No active event sources.
GPT_SOURCE_GTETRGA_RISING	Action performed on GTETRGA rising edge.
GPT_SOURCE_GTETRGA_FALLING	Action performed on GTETRGA falling edge.
GPT_SOURCE_GTETRGB_RISING	Action performed on GTETRGB rising edge.
GPT_SOURCE_GTETRGB_FALLING	Action performed on GTETRGB falling edge.
GPT_SOURCE_GTETRGC_RISING	Action performed on GTETRGC rising edge.
GPT_SOURCE_GTETRGC_FALLING	Action performed on GTETRGC falling edge.
GPT_SOURCE_GTETRGD_RISING	Action performed on GTETRGB rising edge.
GPT_SOURCE_GTETRGD_FALLING	Action performed on GTETRGB falling edge.
GPT_SOURCE_GTIOCA_RISING_WHILE_GTIOCB_LOW	Action performed when GTIOCA input rises while GTIOCB is low.
GPT_SOURCE_GTIOCA_RISING_WHILE_GTIOCB_HIGH	Action performed when GTIOCA input rises while GTIOCB is high.
GPT_SOURCE_GTIOCA_FALLING_WHILE_GTIOCB_LOW	Action performed when GTIOCA input falls while GTIOCB is low.
GPT_SOURCE_GTIOCA_FALLING_WHILE_GTIOCB_HIGH	Action performed when GTIOCA input falls while GTIOCB is high.
GPT_SOURCE_GTIOCB_RISING_WHILE_GTIOCA_LOW	Action performed when GTIOCB input rises while GTIOCA is low.
GPT_SOURCE_GTIOCB_RISING_WHILE_GTIOCA_HIGH	Action performed when GTIOCB input rises while GTIOCA is high.
GPT_SOURCE_GTIOCB_FALLING_WHILE_GTIOCA_LOW	Action performed when GTIOCB input falls while GTIOCA is low.
GPT_SOURCE_GTIOCB_FALLING_WHILE_GTIOCA_HIGH	Action performed when GTIOCB input falls while GTIOCA is high.

GPT_SOURCE_GPT_A	Action performed on ELC GPTA event.
GPT_SOURCE_GPT_B	Action performed on ELC GPTB event.
GPT_SOURCE_GPT_C	Action performed on ELC GPTC event.
GPT_SOURCE_GPT_D	Action performed on ELC GPTD event.
GPT_SOURCE_GPT_E	Action performed on ELC GPTE event.
GPT_SOURCE_GPT_F	Action performed on ELC GPTF event.
GPT_SOURCE_GPT_G	Action performed on ELC GPTG event.
GPT_SOURCE_GPT_H	Action performed on ELC GPTH event.

◆ gpt_capture_filter_t

enum gpt_capture_filter_t

Input capture signal noise filter (debounce) setting. Only available for input signals GTIOCxA and GTIOCxB. The noise filter samples the external signal at intervals of the PCLK divided by one of the values. When 3 consecutive samples are at the same level (high or low), then that level is passed on as the observed state of the signal. See "Noise Filter Function" in the hardware manual, GPT section.

Enumerator	
GPT_CAPTURE_FILTER_NONE	None - no filtering.
GPT_CAPTURE_FILTER_PCLKD_DIV_1	PCLK/1 - fast sampling.
GPT_CAPTURE_FILTER_PCLKD_DIV_4	PCLK/4.
GPT_CAPTURE_FILTER_PCLKD_DIV_16	PCLK/16.
GPT_CAPTURE_FILTER_PCLKD_DIV_64	PCLK/64 - slow sampling.

◆ gpt_adc_trigger_t

enum gpt_adc_trigger_t	
Trigger options to start A/D conversion.	
Enumerator	
GPT_ADC_TRIGGER_NONE	None - no output disable request.
GPT_ADC_TRIGGER_UP_COUNT_START_ADC_A	Request A/D conversion from ADC unit 0 at up counting compare match of gpt_extended_pwm_cfg_t::adc_a_compare_match.
GPT_ADC_TRIGGER_DOWN_COUNT_START_ADC_A	Request A/D conversion from ADC unit 0 at down counting compare match of gpt_extended_pwm_cfg_t::adc_a_compare_match.
GPT_ADC_TRIGGER_UP_COUNT_START_ADC_B	Request A/D conversion from ADC unit 1 at up counting compare match of gpt_extended_pwm_cfg_t::adc_b_compare_match.
GPT_ADC_TRIGGER_DOWN_COUNT_START_ADC_B	Request A/D conversion from ADC unit 1 at down counting compare match of gpt_extended_pwm_cfg_t::adc_b_compare_match.

◆ gpt_poeg_link_t

enum gpt_poeg_link_t	
POEG channel to link to this channel.	
Enumerator	
GPT_POEG_LINK_POEG0	Link this GPT channel to POEG channel 0 (GTETRGA)
GPT_POEG_LINK_POEG1	Link this GPT channel to POEG channel 1 (GTETRGB)
GPT_POEG_LINK_POEG2	Link this GPT channel to POEG channel 2 (GTETRGC)
GPT_POEG_LINK_POEG3	Link this GPT channel to POEG channel 3 (GTETRGD)

◆ gpt_output_disable_t

enum gpt_output_disable_t

Select trigger to send output disable request to POEG.

Enumerator	
GPT_OUTPUT_DISABLE_NONE	None - no output disable request.
GPT_OUTPUT_DISABLE_DEAD_TIME_ERROR	Request output disable if a dead time error occurs.
GPT_OUTPUT_DISABLE_GTIOCA_GTIOCB_HIGH	Request output disable if GTIOCA and GTIOCB are high at the same time.
GPT_OUTPUT_DISABLE_GTIOCA_GTIOCB_LOW	Request output disable if GTIOCA and GTIOCB are low at the same time.

◆ gpt_gtioc_disable_t

enum gpt_gtioc_disable_t

Disable level options for GTIOC pins.

Enumerator	
GPT_GTIOC_DISABLE_PROHIBITED	Do not allow output disable.
GPT_GTIOC_DISABLE_SET_HI_Z	Set GTIOC to high impedance when output is disabled.
GPT_GTIOC_DISABLE_LEVEL_LOW	Set GTIOC level low when output is disabled.
GPT_GTIOC_DISABLE_LEVEL_HIGH	Set GTIOC level high when output is disabled.

◆ gpt_adc_compare_match_t

enum gpt_adc_compare_match_t

Trigger options to start A/D conversion.

Enumerator	
GPT_ADC_COMPARE_MATCH_ADC_A	Set A/D conversion start request value for GPT A/D converter start request A.
GPT_ADC_COMPARE_MATCH_ADC_B	Set A/D conversion start request value for GPT A/D converter start request B.

◆ gpt_interrupt_skip_source_t

enum gpt_interrupt_skip_source_t	
Interrupt skipping modes	
Enumerator	
GPT_INTERRUPT_SKIP_SOURCE_NONE	Do not skip interrupts.
GPT_INTERRUPT_SKIP_SOURCE_OVERFLOW_UNDERFLOW	Count and skip overflow and underflow interrupts.
GPT_INTERRUPT_SKIP_SOURCE_CREST	Count crest interrupts for interrupt skipping. Skip the number of crest and trough interrupts configured in gpt_interrupt_skip_count_t. When the interrupt does fire, the trough interrupt fires before the crest interrupt.
GPT_INTERRUPT_SKIP_SOURCE_TROUGH	Count trough interrupts for interrupt skipping. Skip the number of crest and trough interrupts configured in gpt_interrupt_skip_count_t. When the interrupt does fire, the crest interrupt fires before the trough interrupt.

◆ gpt_interrupt_skip_count_t

enum gpt_interrupt_skip_count_t	
Number of interrupts to skip between events	
Enumerator	
GPT_INTERRUPT_SKIP_COUNT_0	Do not skip interrupts.
GPT_INTERRUPT_SKIP_COUNT_1	Skip one interrupt.
GPT_INTERRUPT_SKIP_COUNT_2	Skip two interrupts.
GPT_INTERRUPT_SKIP_COUNT_3	Skip three interrupts.
GPT_INTERRUPT_SKIP_COUNT_4	Skip four interrupts.
GPT_INTERRUPT_SKIP_COUNT_5	Skip five interrupts.
GPT_INTERRUPT_SKIP_COUNT_6	Skip six interrupts.
GPT_INTERRUPT_SKIP_COUNT_7	Skip seven interrupts.

◆ gpt_interrupt_skip_adc_t

enum gpt_interrupt_skip_adc_t

ADC events to skip during interrupt skipping

Enumerator	
GPT_INTERRUPT_SKIP_ADC_NONE	Do not skip ADC events.
GPT_INTERRUPT_SKIP_ADC_A	Skip ADC A events.
GPT_INTERRUPT_SKIP_ADC_B	Skip ADC B events.
GPT_INTERRUPT_SKIP_ADC_A_AND_B	Skip ADC A and B events.

◆ gpt_pwm_output_delay_setting_t

enum gpt_pwm_output_delay_setting_t

Delay setting for the PWM Delay Generation Circuit (PDG).

Enumerator	
GPT_PWM_OUTPUT_DELAY_SETTING_0_32	Delay is not applied.
GPT_PWM_OUTPUT_DELAY_SETTING_1_32	Delay of 1 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_2_32	Delay of 2 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_3_32	Delay of 3 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_4_32	Delay of 4 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_5_32	Delay of 5 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_6_32	Delay of 6 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_7_32	Delay of 7 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_8_32	Delay of 8 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_9_32	Delay of 9 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_10_32	Delay of 10 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_11_32	Delay of 11 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_12_32	Delay of 12 / 32 GTCLK period applied.

GPT_PWM_OUTPUT_DELAY_SETTING_13_32	Delay of 13 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_14_32	Delay of 14 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_15_32	Delay of 15 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_16_32	Delay of 16 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_17_32	Delay of 17 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_18_32	Delay of 18 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_19_32	Delay of 19 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_20_32	Delay of 20 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_21_32	Delay of 21 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_22_32	Delay of 22 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_23_32	Delay of 23 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_24_32	Delay of 24 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_25_32	Delay of 25 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_26_32	Delay of 26 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_27_32	Delay of 27 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_28_32	Delay of 28 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_29_32	Delay of 29 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_30_32	Delay of 30 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_31_32	Delay of 31 / 32 GTCLK period applied.
GPT_PWM_OUTPUT_DELAY_SETTING_BYPASS	Bypass the PWM Output Delay Circuit.

◆ gpt_pwm_output_delay_edge_t

enum gpt_pwm_output_delay_edge_t	
Select which PWM Output Delay setting to apply.	
Enumerator	
GPT_PWM_OUTPUT_DELAY_EDGE_RISING	Configure the PWM Output Delay setting for rising edge.
GPT_PWM_OUTPUT_DELAY_EDGE_FALLING	Configure the PWM Output Delay setting for falling edge.

Function Documentation

◆ R_GPT_Open()

fsp_err_t R_GPT_Open (timer_ctrl_t *const *p_ctrl*, timer_cfg_t const *const *p_cfg*)

Initializes the timer module and applies configurations. Implements timer_api_t::open.

GPT hardware does not support one-shot functionality natively. When using one-shot mode, the timer will be stopped in an ISR after the requested period has elapsed.

The GPT implementation of the general timer can accept a gpt_extended_cfg_t extension parameter.

Example:

```
/* Initializes the module. */
    err = R_GPT_Open(&g_timer0_ctrl, &g_timer0_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful and timer has started.
FSP_ERR_ASSERTION	A required input pointer is NULL or the source divider is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IRQ_BSP_DISABLED	timer_cfg_t::mode is TIMER_MODE_ONE_SHOT or timer_cfg_t::p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use one-shot mode or callback.
FSP_ERR_INVALID_MODE	Triangle wave PWM is only supported if GPT_CFG_OUTPUT_SUPPORT_ENABLE is 2. Selected channel does not support external count sources.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel requested in the p_cfg parameter is not available on this device.

◆ R_GPT_Stop()

fsp_err_t R_GPT_Stop (timer_ctrl_t *const *p_ctrl*)

Stops timer. Implements timer_api_t::stop.

Example:

```
/* (Optional) Stop the timer. */
   (void) R_GPT_Stop(&g_timer0_ctrl);
```

Return values

FSP_SUCCESS	Timer successfully stopped.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_Start()

fsp_err_t R_GPT_Start (timer_ctrl_t *const *p_ctrl*)

Starts timer. Implements timer_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_GPT_Start(&g_timer0_ctrl);
```

Return values

FSP_SUCCESS	Timer successfully started.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_Reset()

fsp_err_t R_GPT_Reset (timer_ctrl_t *const *p_ctrl*)

Resets the counter value to 0. Implements timer_api_t::reset.

Note

This function also updates to the new period if no counter overflow has occurred since the last call to R_GPT_PeriodSet().

Return values

FSP_SUCCESS	Counter value written successfully.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_Enable()

fsp_err_t R_GPT_Enable (timer_ctrl_t *const *p_ctrl*)

Enables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::enable.

Example:

```
/* Enable captures. Captured values arrive in the interrupt. */
   (void) R_GPT_Enable(&g_timer0_ctrl);
```

Return values

FSP_SUCCESS	External events successfully enabled.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_Disable()

fsp_err_t R_GPT_Disable (timer_ctrl_t *const *p_ctrl*)

Disables external event triggers that start, stop, clear, or capture the counter. Implements timer_api_t::disable.

Note

The timer could be running after R_GPT_Disable(). To ensure it is stopped, call R_GPT_Stop().

Example:

```
/* (Optional) Disable captures. */
   (void) R_GPT_Disable(&g_timer0_ctrl);
```

Return values

FSP_SUCCESS	External events successfully disabled.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_PeriodSet()

fsp_err_t R_GPT_PeriodSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *period_counts*)

Sets period value provided. If the timer is running, the period will be updated after the next counter overflow. If the timer is stopped, this function resets the counter and updates the period. Implements timer_api_t::periodSet.

Warning

If periodic output is used, the duty cycle buffer registers are updated after the period buffer register. If this function is called while the timer is running and a GPT overflow occurs during processing, the duty cycle will not be the desired 50% duty cycle until the counter overflow after processing completes.

Example:

```
/* Get the source clock frequency (in Hz). There are 3 ways to do this in FSP:
  * - If the PCLKD frequency has not changed since reset, the source clock frequency
is
  * BSP_STARTUP_PCLKD_HZ >> timer_cfg_t::source_div
  * - Use the R_GPT_InfoGet function (it accounts for the divider).
  * - Calculate the current PCLKD frequency using
R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) and right shift
  * by timer_cfg_t::source_div.
  *
```

```
  * This example uses the 3rd option (R_FSP_SystemClockHzGet).
  */
    uint32_t pclkd_freq_hz = R_FSP_SystemClockHzGet(FSP_PRIV_CLOCK_PCLKD) >>
g_timer0_cfg.source_div;
 /* Calculate the desired period based on the current clock. Note that this
calculation could overflow if the
  * desired period is larger than UINT32_MAX / pclkd_freq_hz. A cast to uint64_t is
used to prevent this. */
    uint32_t period_counts =
       (uint32_t) (((uint64_t) pclkd_freq_hz * GPT_EXAMPLE_DESIRED_PERIOD_MSEC) /
GPT_EXAMPLE_MSEC_PER_SEC);
 /* Set the calculated period. */
    err = R_GPT_PeriodSet(&g_timer0_ctrl, period_counts);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Period value written successfully.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_DutyCycleSet()

fsp_err_t R_GPT_DutyCycleSet (timer_ctrl_t *const *p_ctrl*, uint32_t const *duty_cycle_counts*, uint32_t const *pin*)

Sets duty cycle on requested pin. Implements timer_api_t::dutyCycleSet.

Duty cycle is updated in the buffer register. The updated duty cycle is reflected after the next cycle end (counter overflow).

Example:

```
 /* Get the current period setting. */
 timer_info_t info;
    (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
    uint32_t current_period_counts = info.period_counts;
 /* Calculate the desired duty cycle based on the current period. Note that if the
period could be larger than
  * UINT32_MAX / 100, this calculation could overflow. A cast to uint64_t is used to
```

```
prevent this. The cast is
  * not required for 16-bit timers. */
    uint32_t duty_cycle_counts =
       (uint32_t) (((uint64_t) current_period_counts *
GPT_EXAMPLE_DESIRED_DUTY_CYCLE_PERCENT) /
                   GPT_EXAMPLE_MAX_PERCENT);
 /* Set the calculated duty cycle. */
    err = R_GPT_DutyCycleSet(&g_timer0_ctrl, duty_cycle_counts, GPT_IO_PIN_GTIOCB);
    assert(FSP_SUCCESS == err);
```

Parameters

[in]	p_ctrl	Pointer to instance control block.
[in]	duty_cycle_counts	Duty cycle to set in counts.
[in]	pin	Use gpt_io_pin_t to select GPT_IO_PIN_GTIOCA or GPT_IO_PIN_GTIOCB

Return values

FSP_SUCCESS	Duty cycle updated successfully.
FSP_ERR_ASSERTION	p_ctrl was NULL or the pin is not one of gpt_io_pin_t
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_INVALID_ARGUMENT	Duty cycle is larger than period.
FSP_ERR_INVALID_MODE	GPT_IO_PIN_TROUGH, and GPT_IO_PIN_CREST settings are invalid in the this mode.
FSP_ERR_UNSUPPORTED	GPT_CFG_OUTPUT_SUPPORT_ENABLE is 0.

◆ R_GPT_InfoGet()

fsp_err_t R_GPT_InfoGet (timer_ctrl_t *const *p_ctrl*, timer_info_t *const *p_info*)

Get timer information and store it in provided pointer p_info. Implements timer_api_t::infoGet.

Example:

```
/* (Optional) Get the current period if not known. */
timer_info_t info;
      (void) R_GPT_InfoGet(&g_timer0_ctrl, &info);
      uint64_t period = info.period_counts;
/* The maximum period is one more than the maximum 32-bit number, but will be
reflected as 0 in
 * timer_info_t::period_counts. */
if (0U == period)
     {
           period = UINT32_MAX + 1U;
     }
```

Return values

FSP_SUCCESS	Period, count direction, frequency, and ELC event written to caller's structure successfully.
FSP_ERR_ASSERTION	p_ctrl or p_info was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_StatusGet()

fsp_err_t R_GPT_StatusGet (timer_ctrl_t *const *p_ctrl*, timer_status_t *const *p_status*)

Get current timer status and store it in provided pointer p_status. Implements timer_api_t::statusGet.

Example:

```
/* Read the current counter value. Counter value is in status.counter. */
timer_status_t status;
    (void) R_GPT_StatusGet(&g_timer0_ctrl, &status);
```

Return values

FSP_SUCCESS	Current timer state and counter value set successfully.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_CounterSet()

fsp_err_t R_GPT_CounterSet (timer_ctrl_t *const *p_ctrl*, uint32_t *counter*)

Set counter value.

Note

Do not call this API while the counter is counting. The counter value can only be updated while the counter is stopped.

Return values

FSP_SUCCESS	Counter value updated.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_IN_USE	The timer is running. Stop the timer before calling this function.

◆ R_GPT_OutputEnable()

fsp_err_t R_GPT_OutputEnable (timer_ctrl_t *const *p_ctrl*, gpt_io_pin_t *pin*)

Enable output for GTIOCA and/or GTIOCB.

Return values

FSP_SUCCESS	Output is enabled.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_OutputDisable()

fsp_err_t R_GPT_OutputDisable (timer_ctrl_t *const *p_ctrl*, gpt_io_pin_t *pin*)

Disable output for GTIOCA and/or GTIOCB.

Return values

FSP_SUCCESS	Output is disabled.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_AdcTriggerSet()

fsp_err_t R_GPT_AdcTriggerSet (timer_ctrl_t *const *p_ctrl*, gpt_adc_compare_match_t *which_compare_match*, uint32_t *compare_match_value*)

Set A/D converter start request compare match value.

Return values

FSP_SUCCESS	Counter value updated.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_PwmOutputDelaySet()

fsp_err_t R_GPT_PwmOutputDelaySet (timer_ctrl_t *const *p_ctrl*, gpt_pwm_output_delay_edge_t *edge*, gpt_pwm_output_delay_setting_t *delay_setting*, uint32_t const *pin*)

Set the Output Delay setting for the PWM output pin.

Return values

FSP_SUCCESS	The output delay was set.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_INVALID_CHANNEL	The channel does not support this feature.
FSP_ERR_NOT_INITIALIZED	The PWM Output Delay Circuit has not been initialized.
FSP_ERR_INVALID_STATE	The PWM Output Delay setting cannot be updated in the current state.
FSP_ERR_UNSUPPORTED	This feature is not supported on this MCU.

◆ R_GPT_CallbackSet()

fsp_err_t R_GPT_CallbackSet (timer_ctrl_t *const *p_api_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void const *const *p_context*, timer_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements timer_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_GPT_Close()

fsp_err_t R_GPT_Close (timer_ctrl_t *const *p_ctrl*)

Stops counter, disables output pins, and clears internal driver data. Implements timer_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_PwmOutputDelayInitialize()

fsp_err_t R_GPT_PwmOutputDelayInitialize ( )

Initialize the PWM Delay Generation Circuit (PDG). This function must be called before calling R_GPT_PwmOutputDelaySet.

Note

This function will delay for 20 microseconds.

Return values

FSP_SUCCESS	Initialization sequence completed successfully.
FSP_ERR_INVALID_STATE	The source clock frequncy is out of the required range for the PDG.
FSP_ERR_UNSUPPORTED	This feature is not supported.

4.2.29 General PWM Timer Three-Phase Motor Control Driver (r_gpt_three_phase)

Modules

Functions

fsp_err_t	R_GPT_THREE_PHASE_Open (three_phase_ctrl_t *const p_ctrl, three_phase_cfg_t const *const p_cfg)
fsp_err_t	R_GPT_THREE_PHASE_Stop (three_phase_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_THREE_PHASE_Start (three_phase_ctrl_t *const p_ctrl)

fsp_err_t	R_GPT_THREE_PHASE_Reset (three_phase_ctrl_t *const p_ctrl)
fsp_err_t	R_GPT_THREE_PHASE_DutyCycleSet (three_phase_ctrl_t *const p_ctrl, three_phase_duty_cycle_t *const p_duty_cycle)
fsp_err_t	R_GPT_THREE_PHASE_CallbackSet (three_phase_ctrl_t *const p_ctrl, void(*p_callback)(timer_callback_args_t *), void const *const p_context, timer_callback_args_t *const p_callback_memory)
fsp_err_t	R_GPT_THREE_PHASE_Close (three_phase_ctrl_t *const p_ctrl)

Detailed Description

Driver for 3-phase motor control using the GPT peripheral on RA MCUs. This module implements the Three-Phase Interface.

Overview

The General PWM Timer (GPT) Three-Phase driver provides basic functionality for synchronously starting and stopping three PWM channels for use in 3-phase motor control applications. A function is additionally provided to allow setting duty cycle values for all three channels, optionally with double-buffering.

Features

The GPT Three-Phase driver provides the following functions:

- Synchronize configuration of three GPT channels
- Synchronously start, stop and reset all three GPT channels
- Set duty cycle on all three channels with one function

Configuration

Build Time Configurations for r_gpt_three_phase

The following build time configurations are defined in fsp_cfg/r_gpt_three_phase_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Timers > Three-Phase PWM (r_gpt_three_phase)

This module can be added to the Stacks tab via New Stack > Timers > Three-Phase PWM (r_gpt_three_phase). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_three_phase0	Module name.
General > Mode	• Triangle-Wave Symmetric PWM • Triangle-Wave Asymmetric PWM • Triangle-Wave Asymmetric PWM (Mode 3)	Triangle-Wave Symmetric PWM	Mode selection. Triangle-Wave Symmetric PWM: Generates symmetric PWM waveforms with duty cycle determined by compare match set during a crest interrupt and updated at the next trough. Triangle-Wave Asymmetric PWM: Generates asymmetric PWM waveforms with duty cycle determined by compare match set during a crest/trough interrupt and updated at the next trough/crest.
General > Period	Value must be a non-negative integer less than or equal to 0x40000000000	15	Specify the timer period in units selected below. Setting the period to 0x100000000 raw counts results in the maximum period. Set the period to 0x100000000 raw counts for a free running timer or an input capture configuration. The period can be set up to 0x40000000000, which will use a divider of 1024 with the maximum period. If the requested period cannot be achieved, the settings with the largest possible period that is less than or equal to the requested period are used. The theoretical calculated period is printed in a comment in the generated timer_cfg_t structures for each

			timer.
General > Period Unit	• Raw Counts • Nanoseconds • Microseconds • Milliseconds • Seconds • Hertz • Kilohertz	Kilohertz	Unit of the period specified above
General > GPT U-Channel	Value must be an integer greater than or equal to 0	0	Specify the GPT channel for U signal output.
General > GPT V-Channel	Value must be an integer greater than or equal to 0	1	Specify the GPT channel for V signal output.
General > GPT W-Channel	Value must be an integer greater than or equal to 0	2	Specify the GPT channel for W signal output.
General > Callback Channel	• U-Channel • V-Channel • W-Channel	U-Channel	Specify the GPT channel to set a callback for when using R_GPT_THREE_PHASE_CallbackSet.
General > Buffer Mode	• Single Buffer • Double Buffer	Single Buffer	When Double Buffer is selected the 'duty_buffer' array in three_phase_duty_cycle_t is used as a buffer for the 'duty' array. This allows setting the duty cycle for the next two crest/trough events in asymmetric mode with only one call to R_GPT_THREE_PHASE_DutyCycleSet.
General > GTIOCA Stop Level	• Pin Level Low • Pin Level High	Pin Level Low	Select the behavior of the output pin when the timer is stopped.
General > GTIOCB Stop Level	• Pin Level Low • Pin Level High	Pin Level Low	Select the behavior of the output pin when the timer is stopped.
Extra Features > Dead Time > Dead Time Count Up (Raw Counts)	Must be a valid non-negative integer with a maximum configurable value of 4294967295 (0xffffffff).	0	Select the dead time to apply during up counting. This value also applies during down counting for the GPT32 and GPT16 variants.
Extra Features > Dead	Must be a valid non-	0	Select the dead time to

Time > Dead Time Count Down (Raw Counts) (GPTE/GPTEH only)	negative integer with a maximum configurable value of 4294967295 (0xffffffff).		apply during down counting. This value only applies to the GPT32E and GPT32EH variants.

Clock Configuration

Please refer to the General PWM Timer (r_gpt) section for more information.

Pin Configuration

Please refer to the General PWM Timer (r_gpt) section for more information.

Usage Notes

Warning

Be sure the GTIOCA/B stop level and dead time values are set appropriately for your application before attempting to drive a motor. Failure to do so may result in damage to the motor drive circuitry and/or the motor itself if the timer is stopped by software.

Initial Setup

The following should be configured once the GPT Three-Phase module has been added to a project:

1. Set "Pin Output Support" in one of the GPT submodules to "Enabled with Extra Features"
2. Configure common settings in the GPT Three-Phase module properties
3. Set the crest and trough interrupt priority and callback function in **one** of the three GPT submodules (if desired)
4. Set the "Extra Features -> Output Disable" settings in each GPT submodule as needed for your application

Note

Because all three modules are operated synchronously with the same period interrupts only need to be enabled in one of the three GPT modules.

Buffer Modes

There are two buffering modes available for duty cycle values - single- and double-buffered. In single buffer mode only the values specified in the duty array element of three_phase_duty_cycle_t are used by R_GPT_THREE_PHASE_DutyCycleSet. At the next trough or crest event the output duty cycle will be internally updated to the set values.

In double buffer mode the duty_buffer array values are used as buffer values for the duty elements. Once passed to R_GPT_THREE_PHASE_DutyCycleSet, the next trough or crest event will update the output duty cycle to the values specified in duty as before. However, at the following crest or trough event the output duty cycle will be updated to the values in duty_buffer. This allows the duty cycle for both sides of an asymmetric PWM waveform to be set at only one trough or crest event per period instead of at every event.

Examples

GPT Three-Phase Basic Example

This is a basic example of minimal use of the GPT Three-Phase module in an application. The duty cycle is updated at every timer trough with the previously loaded buffer value, then the duty cycle buffer is reloaded in the trough interrupt callback.

```
void gpt_callback (timer_callback_args_t * p_args)
{
 fsp_err_t                err;
 three_phase_duty_cycle_t duty_cycle;
 if (TIMER_EVENT_TROUGH == p_args->event)
    {
 /* Update duty cycle values (example) */
       duty_cycle.duty[THREE_PHASE_CHANNEL_U] =
get_duty_counts(THREE_PHASE_CHANNEL_U);
       duty_cycle.duty[THREE_PHASE_CHANNEL_V] =
get_duty_counts(THREE_PHASE_CHANNEL_V);
       duty_cycle.duty[THREE_PHASE_CHANNEL_W] =
get_duty_counts(THREE_PHASE_CHANNEL_W);
 /* Update duty cycle values */
       err = R_GPT_THREE_PHASE_DutyCycleSet(&g_gpt_three_phase_ctrl, &duty_cycle);
       assert(FSP_SUCCESS == err);
    }
 else
    {
 /* Handle crest event. */
    }
}
void gpt_three_phase_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_GPT_THREE_PHASE_Open(&g_gpt_three_phase_ctrl, &g_gpt_three_phase_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the timer. */
    (void) R_GPT_THREE_PHASE_Start(&g_gpt_three_phase_ctrl);
}
```

Data Structures

struct	gpt_three_phase_instance_ctrl_t

Data Structure Documentation

◆ gpt_three_phase_instance_ctrl_t

struct gpt_three_phase_instance_ctrl_t

Channel control block. DO NOT INITIALIZE. Initialization occurs when three_phase_api_t::open is called.

Function Documentation

◆ R_GPT_THREE_PHASE_Open()

fsp_err_t R_GPT_THREE_PHASE_Open (three_phase_ctrl_t *const *p_ctrl*, three_phase_cfg_t const *const *p_cfg*)

Initializes the 3-phase timer module (and associated timers) and applies configurations. Implements three_phase_api_t::open.

Example:

```
/* Initializes the module. */
    err = R_GPT_THREE_PHASE_Open(&g_gpt_three_phase_ctrl, &g_gpt_three_phase_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	A required input pointer is NULL.
FSP_ERR_ALREADY_OPEN	Module is already open.

◆ R_GPT_THREE_PHASE_Stop()

fsp_err_t R_GPT_THREE_PHASE_Stop (three_phase_ctrl_t *const *p_ctrl*)

Stops all timers synchronously. Implements three_phase_api_t::stop.

Return values

FSP_SUCCESS	Timers successfully stopped.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_THREE_PHASE_Start()

fsp_err_t R_GPT_THREE_PHASE_Start (three_phase_ctrl_t *const *p_ctrl*)

Starts all timers synchronously. Implements three_phase_api_t::start.

Example:

```
/* Start the timer. */
   (void) R_GPT_THREE_PHASE_Start(&g_gpt_three_phase_ctrl);
```

Return values

FSP_SUCCESS	Timers successfully started.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_THREE_PHASE_Reset()

fsp_err_t R_GPT_THREE_PHASE_Reset (three_phase_ctrl_t *const *p_ctrl*)

Resets the counter values to 0. Implements three_phase_api_t::reset.

Return values

FSP_SUCCESS	Counters were reset successfully.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_GPT_THREE_PHASE_DutyCycleSet()

fsp_err_t R_GPT_THREE_PHASE_DutyCycleSet (three_phase_ctrl_t *const *p_ctrl*, three_phase_duty_cycle_t *const *p_duty_cycle*)

Sets duty cycle for all three timers. Implements three_phase_api_t::dutyCycleSet.

In symmetric PWM mode duty cycle values are reflected after the next trough. In asymmetric PWM mode values are reflected at the next trough OR crest, whichever comes first.

When double-buffering is enabled the values in three_phase_duty_cycle_t::duty_buffer are set to the double-buffer registers. When values are reflected the first time the single buffer values (three_phase_duty_cycle_t::duty) are used. On the second reflection the duty_buffer values are used. In asymmetric PWM mode this enables both count-up and count-down PWM values to be set at trough (or crest) exclusively.

Note

It is recommended to call this function in a high-priority callback to ensure that it is not interrupted and that no GPT events occur during setting that would result in a duty cycle buffer load operation.

Example:

```
/* Update duty cycle values */
    err = R_GPT_THREE_PHASE_DutyCycleSet(&g_gpt_three_phase_ctrl, &duty_cycle);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Duty cycle updated successfully.
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The instance is not opened.
FSP_ERR_INVALID_ARGUMENT	One or more duty cycle count values was outside the range 0..(period - 1).

◆ R_GPT_THREE_PHASE_CallbackSet()

fsp_err_t R_GPT_THREE_PHASE_CallbackSet (three_phase_ctrl_t *const *p_ctrl*, void(*)(timer_callback_args_t *) *p_callback*, void const *const *p_context*, timer_callback_args_t *const *p_callback_memory*)

Updates the user callback for the GPT U-channel with the option to provide memory for the callback argument structure. Implements three_phase_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ R_GPT_THREE_PHASE_Close()

fsp_err_t R_GPT_THREE_PHASE_Close (three_phase_ctrl_t *const *p_ctrl*)

Stops counters, disables output pins, and clears internal driver data. Implements three_phase_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

4.2.30 I3C Bus Interface (r_i3c)

Modules

Functions

fsp_err_t	R_I3C_Open (i3c_ctrl_t *const p_api_ctrl, i3c_cfg_t const *const p_cfg)
fsp_err_t	R_I3C_Enable (i3c_ctrl_t *const p_api_ctrl)
fsp_err_t	R_I3C_DeviceCfgSet (i3c_ctrl_t *const p_api_ctrl, i3c_device_cfg_t const *const p_device_cfg)
fsp_err_t	R_I3C_MasterDeviceTableSet (i3c_ctrl_t *const p_api_ctrl, uint32_t device_index, i3c_device_table_cfg_t const *const p_device_table_cfg)
fsp_err_t	R_I3C_SlaveStatusSet (i3c_ctrl_t *const p_api_ctrl, i3c_device_status_t status)
fsp_err_t	R_I3C_DeviceSelect (i3c_ctrl_t *const p_api_ctrl, uint32_t device_index, uint32_t bitrate_mode)
fsp_err_t	R_I3C_DynamicAddressAssignmentStart (i3c_ctrl_t *const p_api_ctrl, i3c_address_assignment_mode_t address_assignment_mode, uint32_t starting_device_index, uint32_t device_count)
fsp_err_t	R_I3C_CommandSend (i3c_ctrl_t *const p_api_ctrl, i3c_command_descriptor_t *p_command_descriptor)
fsp_err_t	R_I3C_Write (i3c_ctrl_t *const p_api_ctrl, uint8_t const *const p_data, uint32_t length, bool restart)
fsp_err_t	R_I3C_Read (i3c_ctrl_t *const p_api_ctrl, uint8_t *const p_data,

	uint32_t length, bool restart)
fsp_err_t	R_I3C_IbiWrite (i3c_ctrl_t *const p_api_ctrl, i3c_ibi_type_t ibi_type, uint8_t const *const p_data, uint32_t length)
fsp_err_t	R_I3C_IbiRead (i3c_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t length)
fsp_err_t	R_I3C_Close (i3c_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the I3C peripheral on RA MCUs. This module implements the I3C Interface.

Overview

I3C is a communication protocol defined by MIPI that aims to improve on I2C by increasing the maximum transfer rate, as well as providing other features like "In-band Interrupts", "Dynamic Address Assignment", and a set of standard "Common Command Codes".

Features

- I3C Master Mode
- I3C Slave Mode
- Dynamic Address Assignment (ENTDAA/SETDASA)
- SDR Read/Write transfers
- I2C Legacy Read/Write transfers
- In-Band Interrupts (Interrupt Requests, Hot-Join Requests)
- Common Command Codes
- Clock Stalling
- Timeout Detection

Master Mode

On an I3C bus, only one device may operate in master mode at a time. The current master is responsible for initiating I2C Legacy transfers, SDR transfers, Common Command Codes, and handling IBIs (Interrupt Requests, Hot-Join Requests). In order to perform these operations, the driver has an internal device table that is used for storing configuration information for each device on the bus (See i3c_device_table_cfg_t). Each entry in the device table contains the static or dynamic address of the device, and IBI permissions for accepting or rejecting IBI requests from the device. The device table has four entries as well as one extended device entry that only contains the static or dynamic address of a device (See below).

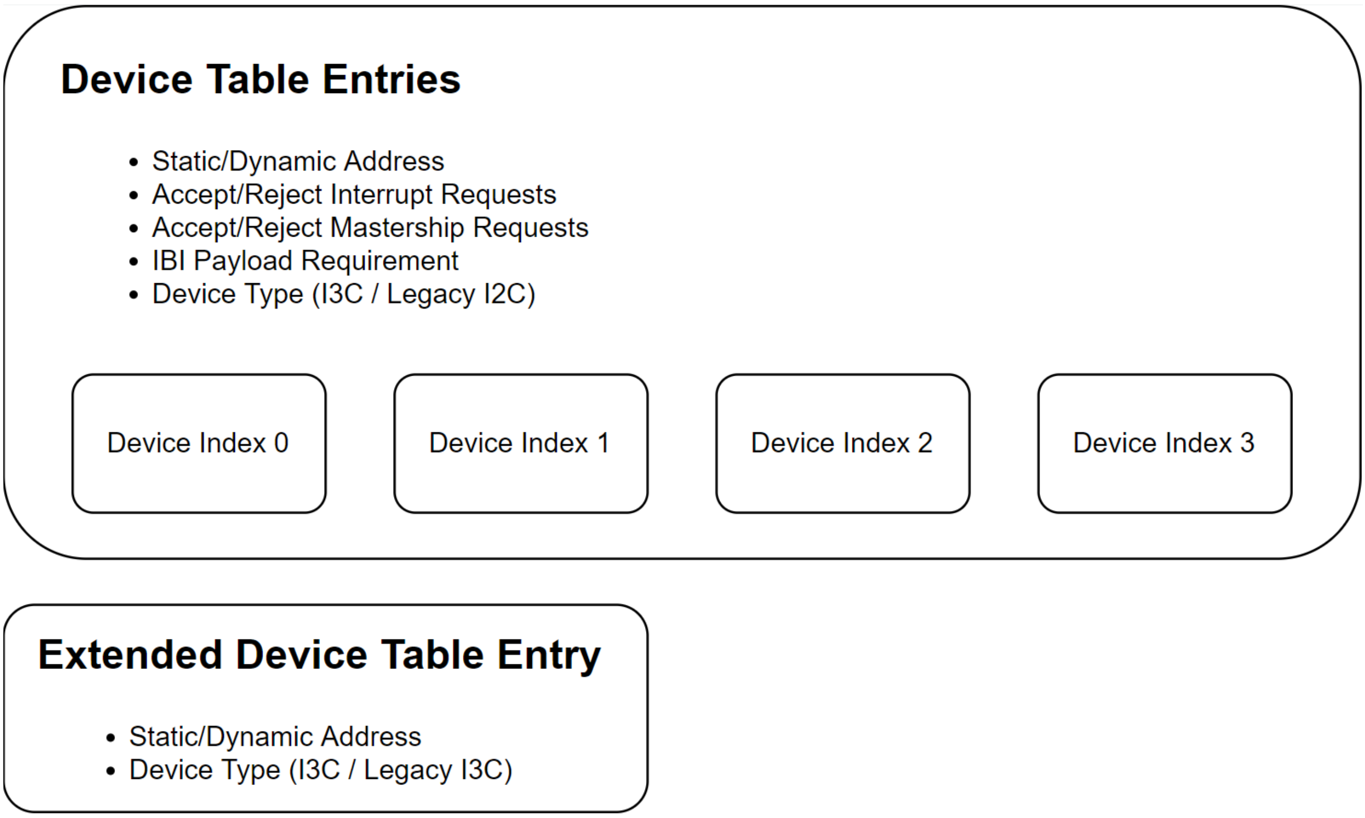


Figure 170: Master Device Table

In order to initiate I2C Legacy transfers, SDR transfers, or Common Command Codes, the master must select a device entry from the device table using i3c_api_t::deviceSelect. Once a device has been selected, all subsequent operations will be directed to the selected device until a new device is selected using i3c_api_t::deviceSelect.

The master may also receive IBI requests that are initiated by slave devices on the bus. If there is a payload, then the driver will write the data into a buffer that is provided by the application by calling i3c_api_t::ibiRead. If the application has not provided an IBI buffer prior to receiving an IBI, then the it will get a callback requesting an IBI buffer. Once the IBI is completed, the application will be notified by a callback.

Note

1. Even though there are only four device table entries and one extended device table entry, the application can operate on more devices by maintaining its own list of devices and updating the extended device entry as needed. Note however that devices defined in the extended device table entry will not be able to initiate IBI requests.

Main Master

The main master is responsible for configuring the dynamic address of all devices on the bus. The driver initiates this procedure by calling i3c_api_t::dynamicAddressAssignmentStart. Before starting address assignment, the application must configure the device table using i3c_api_t::masterDeviceTableSet.

Enter Dynamic Address Assignment (ENTDAA):

The application initiates the ENTDAA operation by calling i3c_api_t::dynamicAddressAssignmentStart with a starting index into the master device table and a count specifying the number of devices to configure. The master starts by sending the ENTDAA command. Every I3C device on the bus that has not already been initialized will acknowledge the command and attempt to write its Provisional ID, DCR, and BCR registers. The device with the smallest value in these registers will win arbitration and be assigned with the first dynamic address defined in the master device table. The master will then increment the index and repeat the process by assigning the dynamic address to the next device. The process continues until the specified number of devices have been initialized or until there are no more devices to configure.

Note

1. The IBI payload setting will automatically be updated in the master device table based on the BCR setting that was read during ENTDAA.
2. After each device successfully writes its Provisional ID, DCR, and BCR registers, the application will get a callback that will provide the value of the registers.
3. If the starting index is set to the extended device entry, then the device count must be set to 1.
4. The main master assigns its own dynamic address with i3c_api_t::deviceCfgSet.

Set Dynamic Address from Static Address (SETDASA):

The application initiates the SETDASA operation by calling i3c_api_t::dynamicAddressAssignmentStart with an index into the master device table. The master sends the SETDASA command to the static address defined in the given device table entry, and then assigns the associated dynamic address.

Note

1. Set the count to 0 when using SETDASA.

Slave Mode

In slave mode, the device configures its static address, Provisional ID, BCR, and DCR registers using i3c_api_t::deviceCfgSet, and then waits for the master to initiate communication. Prior to being assigned a dynamic address, the slave will operate as an I2C device using its static address. The application will receive a callback when the master assigns it a dynamic address, after which point, the slave will operate as an I3C device until it receives the RSTDAA command.

Depending on the capabilities defined in its BCR register, the slave may also initiate IBI Interrupt Requests, and Hot-Join Requests using i3c_api_t::ibiWrite

Configuration

Build Time Configurations for r_i3c

The following build time configurations are defined in fsp_cfg/r_i3c_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Unaligned Buffer Support	• Enabled • Disabled	Enabled	Unaligned buffer support may be optionally disabled for

			improved performance.
Master Support	• Enabled • Disabled	Enabled	If only slave mode is required, disable master support to decrease code size.
Slave Support	• Enabled • Disabled	Enabled	If only master mode is required, disable slave support to decrease code size.

Configurations for Connectivity > I3C (r_i3c)

This module can be added to the Stacks tab via New Stack > Connectivity > I3C (r_i3c).

Configuration	Options	Default	Description
Bitrate Settings > Standard Mode > Open-Drain > Logic High Period (ns)	Must be an integer greater than 0.	167	The Logic High period of SCL during Standard Mode Open Drain transfers.
Bitrate Settings > Standard Mode > Open-Drain > Frequency	Must be an integer greater than 0.	1000000	The Frequency of SCL during Standard Mode Open Drain transfers.
Bitrate Settings > Standard Mode > Push-Pull > Logic High Period (ns)	Must be an integer greater than 0.	167	The Logic High period of SCL during Standard Mode Push Pull transfers.
Bitrate Settings > Standard Mode > Push-Pull > Frequency	Push-Pull frequency must be greater than or equal to 10000 Hz.	3400000	The Frequency of SCL during Standard Mode Push-Pull transfers.
Bitrate Settings > Extended Mode > Open-Drain > Logic High Period (ns)	Must be an integer greater than 0.	167	The Logic High period of SCL during Extended Mode Open Drain transfers.
Bitrate Settings > Extended Mode > Open-Drain > Frequency	Must be an integer greater than 0.	1000000	The Frequency of SCL during Extended Mode Open Drain transfers.
Bitrate Settings > Extended Mode > Push-Pull > Logic High Period (ns)	Must be an integer greater than 0.	167	The Logic High period of SCL during Extended Mode Push Pull transfers.
Bitrate Settings > Extended Mode > Push-Pull > Frequency	Push-Pull frequency must be greater than or equal to 10000 Hz.	3400000	The Frequency of SCL during Extended Mode Push-Pull transfers.
Bitrate Settings > Bus Timing > Open Drain Rising Time (ns)	Rising time must be greater than or equal to 0 nanoseconds.	0	The Open Drain rising time in nanoseconds.
Bitrate Settings > Bus	Falling time must be	0	The Open Drain falling

Timing > Open Drain Falling Time (ns)	greater than or equal to 0 nanoseconds.		time in nanoseconds.
Bitrate Settings > Bus Timing > Push-Pull Rising Time (ns)	Rising time must be greater than or equal to 0 nanoseconds.	0	The Push-Pull rising time in nanoseconds.
Bitrate Settings > Bus Timing > Push-Pull Falling Time (ns)	Falling time must be greater than or equal to 0 nanoseconds.	0	The Push-Pull rising time in nanoseconds.
Bitrate Settings > Clock Stalling > Address Assignment Phase	• Enabled • Disabled	Disabled	Enable clock stalling during the Address Assignment Phase of ENTDAA.
Bitrate Settings > Clock Stalling > Transition Phase	• Enabled • Disabled	Disabled	Enable clock stalling during the Transition Bit of a read transfer.
Bitrate Settings > Clock Stalling > Parity Phase	• Enabled • Disabled	Disabled	Enable clock stalling during the Parity Bit of a write transfer.
Bitrate Settings > Clock Stalling > Ack Phase	• Enabled • Disabled	Disabled	Enable clock stalling during the ACK phase of a transfer.
Bitrate Settings > Clock Stalling > Time (us)	Must be greater than or equal to 0.	0	The amount of time to stall the clock during the Address Assignment Phase, Transition Phase, Parity Phase, and ACK Phase.
Master Mode > ACK Hot-Join Requests	• Enabled • Disabled	Disabled	If enabled, the I3C instance will ACK Hot-Join Requests and notify the application.
Master Mode > Notify Rejected Hot-Join Requests.	• Enabled • Disabled	Disabled	If enabled, the application will get a callback when an IBI Hot-Join Request is rejected.
Master Mode > Notify Rejected Mastership Requests.	• Enabled • Disabled	Disabled	If enabled, the application will get a callback when an IBI Mastership Request is rejected.
Master Mode > Notify Rejected Interrupt Requests.	• Enabled • Disabled	Disabled	If enabled, the application will get a callback when an IBI Interrupt Request is rejected.
Slave Mode >	• Enabled	Disabled	Configure whether the

Command Response Info > ENEC/DISEC > In-Band Interrupts	• Disabled		slave can issue IBI requests.
Slave Mode > Command Response Info > ENEC/DISEC > Hot-Join Requests	• Enabled • Disabled	Disabled	Configure whether the slave can issue Hot-Join requests.
Slave Mode > Command Response Info > ENTASn > Activity State	• Activity State 0 • Activity State 1 • Activity State 2 • Activity State 3	Activity State 0	Configure the starting activity state of the slave.
Slave Mode > Command Response Info > SETMWL/GETMWL > Max Write Length	Write length must be in the range of [8, 65535].	65535	Set the max write length.
Slave Mode > Command Response Info > SETMRL/GETMRL > Max Read Length	Read length must be in the range of [16, 65535].	65535	Set the max read length.
Slave Mode > Command Response Info > SETMRL/GETMRL > Max IBI Payload Length	Read length must be in the range of [0, 255].	0	Set the max IBI payload length, or set it to 0 for unlimited.
Slave Mode > Command Response Info > GETMXDS > Write Data Rate	• FSCL_MAX • 8Mhz • 6Mhz • 4Mhz • 2Mhz	2Mhz	Set the max write data rate.
Slave Mode > Command Response Info > GETMXDS > Read Data Rate	• FSCL_MAX • 8Mhz • 6Mhz • 4Mhz • 2Mhz	2Mhz	Set the max read data rate.
Slave Mode > Command Response Info > GETMXDS > Clock to Data Turnaround Time	• 8 Nanoseconds • 9 Nanoseconds • 10 Nanoseconds • 11 Nanoseconds • 12 Nanoseconds • Greater than 12 Nanoseconds	8 Nanoseconds	Set the clock to data turnaround time.
Slave Mode > Command Response Info > GETMXDS > Include Max Read	• Enabled • Disabled	Disabled	Configure whether the Max Read Turnaround time will be transmitted.

Turnaround Time			
Slave Mode > Command Response Info > GETMXDS > Max Read Turnaround Time	Value must be in the range [0, 255].	0	Set max read turnaround time.
Slave Mode > Command Response Info > GETXTIME > Frequency Byte	Value must be in the range [0, 255].	0	Set the internal oscillator frequency in increments of 0.5 Mhz.
Slave Mode > Command Response Info > GETXTIME > Inaccuracy Byte	Value must be in the range [0, 255].	0	Set the oscillator inaccuracy byte in increments of 0.5%
Interrupts > Interrupt Priority	MCU Specific Options		The interrupt priority of the RX, TX, RESPONSE, RCV_STATUS, and IBI ISRs.
Interrupts > Error and Event Interrupt Priority	MCU Specific Options		The interrupt priority of the EEI ISR which is used to notify the application when an Internal Error, HDR Exit Pattern, or Timeout is detected.
Interrupts > Receive Status ICU Group 3	MCU Specific Options		The Receive Status IRQ can use ICU Group 3 or Group 5.
Interrupts > Receive Status ICU Group 5	MCU Specific Options		The Receive Status IRQ can use ICU Group 3 or Group 5.
Name	Name must be a valid C symbol	g_i3c0	Module name.
Callback	Name must be a valid C symbol	g_i3c0_callback	A user callback function must be provided. This will be called in order to notify the application of I3C events and provide status information.
Callback Context	Name must be a valid C symbol	NULL	A pointer to additional application specific information that is provided to the callback.
Device Type	• Main Master • Slave	Slave	The role that the I3C instance will take on the I3C bus.

Bus Free Condition Detection Time (ns)	Must be greater than or equal to 38.4 nanoseconds.	38.4	The minimum period occurring after a STOP and before a START.
Bus Available Condition Detection Time (us)	Must be greater than or equal to 1 microsecond.	1	The minimum period occurring after the Bus Free Condition when Slaves can initiate IBI requests.
Bus Idle Condition Detection Time (us)	Must be greater than or equal to 1000 microseconds.	1000	The minimum period occurring after the Bus Available Condition when Slaves can initiate Hot-Join requests.
Timeout Detection	• Enabled • Disabled	Disabled	If enabled, the application will get a callback if SCL is stuck at a logic high or logic low level for more than 65535 cycles of the I3C source clock.

Clock Configuration

The I3C peripheral is clocked off of PCLKB and PCLKD. The SCL clock is generated by the I3C internal clock which is derived from PCLKD.

The following settings are used to configure SCL.

- Frequency of PCLKD
- Standard Mode
  - Open Drain High Period (T_{HIGH})
  - Open Drain Frequency
  - Push-Pull High Period (T_{HIGH})
  - Push-Pull Frequency
- Extended Mode
  - Open Drain High Period (T_{HIGH})
  - Open Drain Frequency
  - Push-Pull High Period (T_{HIGH})
  - Push-Pull Frquency
- i3c_bitrate_mode_t (Set during i3c_api_t::deviceSelect)

The Standard and Extended Mode settings define two seperate SCL configurations that can be selected at run-time using i3c_api_t::deviceSelect.

In addition to selecting between the Standard and Extended Mode settings, the base SCL period can also be multiplied using the following options:

- I3C_BITRATE_MODE_I3C_SDR2_STDBR_X2: Multiple the base Standard Open Drain and Push-Pull period by 2.
- I3C_BITRATE_MODE_I3C_SDR3_EXTBR_X2: Multiple the base Extended Open Drain and Push-Pull period by 2.
- I3C_BITRATE_MODE_I3C_SDR4_EXTBR_X4: Multiple the base Extended Open Drain and Push-

Pull period by 4.

In order to get accurate frequency calculations, the Rising and Falling edges must be input into the calculation. These values will depend on the topology the I3C bus that will be different for every application.

Note

1. The Standard and Extended Open Drain period settings define the period to use during legacy I2C transfers (Only use the following i3c_bitrate_mode_t settings with I2C transfers: I3C_BITRATE_MODE_I2C_STDBR, I3C_BITRATE_MODE_I2C_EXTBR).
2. The clock frequency ratio of PCLKD and PCLKB must be 2:1 or 1:1 when using the I3C module. Operation is not guaranteed for other settings.
3. T_{HIGH} is defined in Figure 31 in the MIPI I3C Specification v1.0 and describe the Logic High period.

Pin Configuration

The I3C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I3C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

Usage Notes

Read and Write in Slave Mode

In slave mode, calling read or write does not start a transfer. Instead, calling read or write will configure the driver to perform the next read or write transfer using the user provided buffer.

Provided that a transfer is not already in progress, i3c_api_t::read and i3c_api_t::write can be called to update the internal buffers even if the transfer has not been completed yet. Both the read and write buffers can be configured at the same time in order to prepare the driver for when the master initiates a read or a write transfer.

If there is no space remaining in a user configured read buffer, the application will get a I3C_EVENT_READ_BUFFER_FULL callback requesting for a new read buffer to be provided.

Event Status

When a write, read, ibiWrite, ibiRead or commandSend, operation is completed, the i3c_callback_args_t::event_status should be checked. The event_status will provide information about the success or failure of the operation.

The following are possible statuses:

- I3C_EVENT_STATUS_SUCCESS
- I3C_EVENT_STATUS_PARITY
- I3C_EVENT_STATUS_FRAME
- I3C_EVENT_STATUS_ADDRESS_HEADER
- I3C_EVENT_STATUS_NACK
- I3C_EVENT_STATUS_OVERFLOW
- I3C_EVENT_STATUS_ABORTED_TO_MASTER
- I3C_EVENT_STATUS_ABORTED

Note

In master mode, if the master issues a stop condition before the slave ends the transfer via the 'T' bit, the status will

be I3C_EVENT_STATUS_ABORTED.
In slave mode, if the master issues a stop condition before the slave ends the transfer via the 'T' bit, the status will be I3C_EVENT_STATUS_ABORTED_TO_MASTER.

Direct Get Common Command Codes in Slave Mode

When an I3C slave receives a Command Code of type Direct Get, the response is automatically sent from the device Special Function Registers (SFR). The SFR contains information for each command code and can be configured during open (See i3c_extended_cfg_t::slave_command_response_info). This allows the slave to respond to Direct Get Command Codes much faster, and removes the burden of responding to these commands from the application.

The response to the GETSTATUS command can be configured at run-time using i3c_api_t::slaveStatusSet.

Disabling Unaligned Buffer Support

Support for performing read and write operations on unaligned buffers can be disabled in order to improve performance. When unaligned buffer support is disabled, all buffers passed to read, ibiRead, write, ibiWrite, and commandSend must be aligned to 4 bytes and the size of the buffers must be a multiple of 4 bytes.

In master mode, the value of the length passed to i3c_api_t::read and i3c_api_t::write sets the total length of the operation in bytes. During the read or write operation, the driver may read or write to the last word of memory during the operation. This means that the allocated memory for the buffer passed to read and write needs to be a multiple of 4 bytes even though the transfer length is not a multiple of 4 bytes.

In slave mode, the length passed to i3c_api_t::read must be a multiple of 4 bytes. The length passed to i3c_api_t::write sets the number of bytes that the slave will write. The size of the buffer passed to write still needs to be a multiple of 4 bytes.

Max Data Speed Limitation

In Slave Mode, it is highly reccomended that BCR[0] be set to '1' in order to indicate to the master that the device doesn't support the max data speed. The master is then required to use the GETMXDS command to get the device specific data speed limitations.

This will alow the slave to specify its Maximum Supported Data Rate for read/write transfers and the Maximum Read Turnaround Time (See GETMXDS in the MIPI I3C Specification v1.0).

Mixed Fast Bus

The MIPI I3C Specification v1.0 defines a Mixed Fast Bus as a bus that has legacy I2C devices that all have a 50ns Spike Filter.

On Mixed Fast Buses, SCL has the following constraints during I3C SDR transfers:

- SCL High Period: $t_{DIG\_H\_MIXED(MIN)}$ to $t_{DIG\_H\_MIXED(MAX)}$
- SCL Low Period: Up to $t_{DIG\_L(MAX)}$

In this case, configure the Extended Bitrate settings for I3C SDR transfers, and use the Standard Bitrate settings for I2C transfers.

Note

1. *See section 5.1.2.4 in the MIPI I3C Specification v1.0.*

Mixed Slow Bus

The MIPI I3C Specification v1.0 defines a Mixed Slow Bus as a bus that has legacy I2C devices that do not have a 50ns Spike Filter.

In this case, the SCL frequency is limited to I2C Fast Mode or I2C Fast Mode Plus.

Note

1. *See section 5.1.2.4 in the MIPI I3C Specification v1.0.*

Limitations

Developers should be aware of the following limitations when using the I3C:

- The I3C peripheral on the RA2E2 has a hardware defect related to using ENTDAA in slave mode. When the ENTDAA command is received, the peripheral will continue driving its PID, DCR, and BCR registers even after its dynamic address has been configured. This will prevent any other devices from being configured on the bus. One workaround for this limitation is to configure the device Provisional ID to all '1's in order to ensure that the device is initialized last. If more than one device with this limitation is present on the same bus, consider initializing them using another method (Eg. SETDASA).
- The MIPI Reserved area and Vendor Extension area of Command Codes are not supported.
- Mixed Fast Bus topology has the following limitation on RA2E2 MCUs. The minimum SCL high period (T_{DIG_H}) is 156 nanoseconds when PCLKD is 48 Mhz, and 120 nanoseconds when PCLKD is 64 Mhz. On a Mixed Fast Bus, the high period $T_{DIG\_HIGH\_MIXED}$ must be less than 45 nanoseconds in order to ensure that Legacy I2C devices do not interpret I3C signaling as valid I2C signaling (See Table 111 Push-Pull-Timing Parameters in the MIPI I3C Specification v1.1). This required high period cannot be achievable with RA2E2 MCUs.
- Secondary Master device role is not currently supported.

Examples

I3C Master Basic Example

This is a basic example of minimal use of the I3C Master in an application.

```
void i3c_master_basic_example (void)
{
 /* Initializes the module. */
 fsp_err_t status = R_I3C_Open(&g_i3c_ctrl, &g_i3c_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_cfg_t master_device_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address = EXAMPLE_MASTER_STATIC_ADDRESS,
```

```
 /* If the device is a main master, it must configure its own dynamic address. */
       .dynamic_address = EXAMPLE_MASTER_DYNAMIC_ADDRESS,
    };
    status = R_I3C_DeviceCfgSet(&g_i3c_ctrl, &master_device_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_table_cfg_t device_table_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address       = EXAMPLE_STATIC_ADDRESS,
 /* Dynamic address is not used in I2C. */
       .dynamic_address       = EXAMPLE_DYNAMIC_ADDRESS,
 /* This is the type of device. It may be either an I2C device or an I3C device. */
       .device_protocol       = I3C_DEVICE_PROTOCOL_I3C,
       .ibi_accept            = false,
 /* Depending on the device the IBI requests may have a data payload.
  * Note that this field will be automatically updated if the device is configured
using ENTDAA.
  */
       .ibi_payload          = false,
 /* Master requests cannot be accepted because Secondary Master is not supported. */
       .master_request_accept = false,
    };
 /* Set the device configuration in the master device table. */
    status = R_I3C_MasterDeviceTableSet(&g_i3c_ctrl, 0, &device_table_cfg);
    assert(FSP_SUCCESS == status);
 /* Enable the I3C device. */
    status = R_I3C_Enable(&g_i3c_ctrl);
    assert(FSP_SUCCESS == status);
 /* Start assigning dynamic addresses to devices on the bus using the ENTDAA command.
*/
    status = R_I3C_DynamicAddressAssignmentStart(&g_i3c_ctrl,
I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA, 0, 1);
    assert(FSP_SUCCESS == status);
```

```
 /* Wait for dynamic address assignment to complete. */
    i3c_app_event_wait(I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
 /* Select the configured device and bitrate mode for the following operations. */
    status = R_I3C_DeviceSelect(&g_i3c_ctrl, 0, I3C_BITRATE_MODE_I3C_SDR0_STDBR);
    assert(FSP_SUCCESS == status);
 /* Start a write transfer. */
 static uint8_t p_write_buffer[] = {1, 2, 3, 4, 5};
    status = R_I3C_Write(&g_i3c_ctrl, p_write_buffer, sizeof(p_write_buffer), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the write transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_WRITE_COMPLETE);
 /* Start a read transfer. */
 static uint8_t p_read_buffer[16];
    status = R_I3C_Read(&g_i3c_ctrl, p_read_buffer, sizeof(p_read_buffer), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the read transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_READ_COMPLETE);
}
/* This function is called by the I3C driver from ISRs in order to notify the
application of I3C events. */
void i3c_master_basic_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_ENTDAA_ADDRESS_PHASE:
      {
 /* The device PID, DCR, and BCR registers will be available in
i3c_callback_args_t::p_slave_info. */
 break;
       }
 case I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE:
      {
            i3c_app_event_notify(I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
 break;
```

```
      }
 case I3C_EVENT_WRITE_COMPLETE:
     {
            i3c_app_event_notify(I3C_EVENT_WRITE_COMPLETE);
 break;
      }
 case I3C_EVENT_READ_COMPLETE:
     {
 /* The number of bytes read from the slave will be available in
i3c_callback_args_t::transfer_size. */
            i3c_app_event_notify(I3C_EVENT_READ_COMPLETE);
 break;
      }
 default:
     {
 break;
      }
    }
}
```

I3C Slave Basic Example

This is a basic example of minimal use of the I3C Slave in an application.

```
void i3c_slave_basic_example (void)
{
 /* Initializes the module. */
 fsp_err_t status = R_I3C_Open(&g_i3c_ctrl, &g_i3c_cfg);
   assert(FSP_SUCCESS == status);
 static i3c_device_cfg_t slave_device_cfg =
   {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address = EXAMPLE_STATIC_ADDRESS,
 /* The dynamic address will be automatically updated when the master configures this
```

```
device using ENTDAA. */
       .dynamic_address = 0,
 /* Device Registers that are read by the master. */
       .slave_info      =
      {
            .bcr = EXAMPLE_BCR_SETTING,
            .dcr = EXAMPLE_DCR_SETTING,
            .pid =
      {
                0, 1, 2, 3, 4, 5
         }
       }
    };
 /* Set the device configuration for this device. */
    status = R_I3C_DeviceCfgSet(&g_i3c_ctrl, &slave_device_cfg);
    assert(FSP_SUCCESS == status);
 /* Enable Slave Mode. */
    status = R_I3C_Enable(&g_i3c_ctrl);
    assert(FSP_SUCCESS == status);
 static uint8_t p_read_buffer[EXAMPLE_READ_BUFFER_SIZE];
 static uint8_t p_write_buffer[EXAMPLE_WRITE_BUFFER_SIZE];
 /* Set the buffer for storing data received during a read transfer. */
    status = R_I3C_Read(&g_i3c_ctrl, p_read_buffer, sizeof(p_read_buffer), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the master to complete a read transfer. */
    i3c_app_event_wait(I3C_EVENT_READ_COMPLETE);
 /* Set the write buffer that will be transmitted during a write transfer. */
    status = R_I3C_Write(&g_i3c_ctrl, p_write_buffer, sizeof(p_write_buffer), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the master to complete a write transfer. */
    i3c_app_event_wait(I3C_EVENT_WRITE_COMPLETE);
}
void i3c_slave_basic_example_callback (i3c_callback_args_t const * const p_args)
{
```

```
 switch (p_args->event)
    {
 case I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE:
      {
            i3c_app_event_notify(I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE);
 break;
       }
 case I3C_EVENT_READ_BUFFER_FULL:
      {
 /* If there is no user provided read buffer, or if the user provided read buffer has
been filled,
  * the driver will notify the application that the buffer is full. The application
may provide
  * a new read buffer by calling i3c_api_t::read. If no read buffer is provided, then
any remaining bytes
  * in the transfer will be dropped. */
            uint8_t * p_read_buffer = i3c_app_next_read_buffer_get();
 R_I3C_Read(&g_i3c_ctrl, p_read_buffer, EXAMPLE_READ_BUFFER_SIZE, false);
 break;
       }
 case I3C_EVENT_READ_COMPLETE:
      {
 /* The number of bytes read by the slave will be available in
i3c_callback_args_t::transfer_size. */
            i3c_app_event_notify(I3C_EVENT_READ_COMPLETE);
 /* Note that the application may also call i3c_api_t::read or i3c_api_t::write from
this event
  * In order to set the transfer buffers for the next transfer. */
 break;
       }
 case I3C_EVENT_WRITE_COMPLETE:
      {
 /* The number of bytes written by the slave will be available in
i3c_callback_args_t::transfer_size. */
```

```
            i3c_app_event_notify(I3C_EVENT_WRITE_COMPLETE);
 /* Note that the application may also call i3c_api_t::read or i3c_api_t::write from
this event
  * In order to set the transfer buffers for the next transfer. */
 break;
       }
 default:
      {
 break;
       }
    }
}
```

I2C Legacy Basic Example

This is a basic example of minimal use of I2C Legacy transfers in an application.

```
void i2c_legacy_basic_example (void)
{
 /* Initializes the module. */
 fsp_err_t status = R_I3C_Open(&g_i3c_ctrl, &g_i3c_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_cfg_t master_device_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
*/
       .static_address = EXAMPLE_MASTER_STATIC_ADDRESS,
 /* If the device is a main master, it must configure its own dynamic address. */
       .dynamic_address = EXAMPLE_MASTER_DYNAMIC_ADDRESS,
    };
    status = R_I3C_DeviceCfgSet(&g_i3c_ctrl, &master_device_cfg);
    assert(FSP_SUCCESS == status);
 static i3c_device_table_cfg_t device_table_cfg =
    {
 /* This is the Static I3C / I2C Legacy address defined by the device manufacturer.
```

```
*/
       .static_address       = EXAMPLE_STATIC_ADDRESS,
 /* Dynamic address is not used in I2C. */
       .dynamic_address       = 0,
 /* This is the type of device. It may be either an I2C device or an I3C device. */
       .device_protocol       = I3C_DEVICE_PROTOCOL_I2C,
 /* These options are not used in I2C. */
       .ibi_accept            = false,
 /* Depending on the device the IBI requests may have a data payload.
  * Note that this field will be automatically updated if the device is configured
using ENTDAA.
  */
       .ibi_payload          = false,
 /* Master requests cannot be accepted because Secondary Master is not supported. */
       .master_request_accept = false,
    };
 /* Set the device configuration in the master device table. */
    status = R_I3C_MasterDeviceTableSet(&g_i3c_ctrl, 0, &device_table_cfg);
    assert(FSP_SUCCESS == status);
 /* Enable the I3C device. */
    status = R_I3C_Enable(&g_i3c_ctrl);
    assert(FSP_SUCCESS == status);
 /* Select the configured device for the following operations. */
    status = R_I3C_DeviceSelect(&g_i3c_ctrl, 0, I3C_BITRATE_MODE_I2C_STDBR);
    assert(FSP_SUCCESS == status);
 /* Start a write transfer. */
 static uint8_t p_write_data[] = {1, 2, 3, 4, 5};
    status = R_I3C_Write(&g_i3c_ctrl, p_write_data, sizeof(p_write_data), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the write transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_WRITE_COMPLETE);
 /* Start a read transfer. */
 static uint8_t p_read_data[16];
    status = R_I3C_Read(&g_i3c_ctrl, p_read_data, sizeof(p_read_data), false);
```

```
    assert(FSP_SUCCESS == status);
 /* Wait for the read transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_READ_COMPLETE);
}
void i2c_legacy_basic_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_WRITE_COMPLETE:
      {
            i3c_app_event_notify(I3C_EVENT_WRITE_COMPLETE);
 break;
       }
 case I3C_EVENT_READ_COMPLETE:
      {
 /* The number of bytes read from the slave will be available in
i3c_callback_args_t::transfer_size. */
            i3c_app_event_notify(I3C_EVENT_READ_COMPLETE);
 break;
       }
 default:
      {
 break;
       }
    }
}
```

I3C Master In-band Interrupts Example

This is a basic example of reading In-band Interrupts in I3C Master mode.

```
void i3c_master_ibi_basic_example (void)
{
 static uint8_t p_ibi_read_buffer[EXAMPLE_READ_BUFFER_SIZE];
 /* Set the buffer for storing IBI data that is read from the slave. */
```

```
 fsp_err_t status = R_I3C_IbiRead(&g_i3c_ctrl, p_ibi_read_buffer, sizeof
(p_ibi_read_buffer));
    assert(FSP_SUCCESS == status);
 /* Wait for the ibiRead transfer to complete.
  * Note that the master does not need to wait for the IBI, and can start other
operations. */
    i3c_app_event_wait(I3C_EVENT_IBI_READ_COMPLETE);
}
void i3c_master_ibi_basic_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_IBI_READ_BUFFER_FULL:
      {
 /* If there is no user provided ibiRead buffer, or if the user provided ibiRead
buffer has been filled,
  * the driver will notify the application that the buffer is full. The application
may provide
  * a new read buffer by calling i3c_api_t::ibiRead. If no read buffer is provided,
then any remaining bytes
  * in the transfer will be dropped. */
            uint8_t * p_read_buffer = i3c_app_next_read_buffer_get();
 R_I3C_IbiRead(&g_i3c_ctrl, p_read_buffer, EXAMPLE_READ_BUFFER_SIZE);
 break;
       }
 case I3C_EVENT_IBI_READ_COMPLETE:
      {
 /* When an IBI is completed, the transfer_size, ibi_type, and ibi_address will be
available in p_args. */
 switch (p_args->ibi_type)
      {
 case I3C_IBI_TYPE_INTERRUPT:
      {
 /* Notify the application that an IBI was read. */
```

```
                   i3c_app_event_notify(I3C_EVENT_IBI_READ_COMPLETE);
 break;
         }
 case I3C_IBI_TYPE_HOT_JOIN:
      {
 /* If a Hot-Join event is received, then the master can initiate the dynamic address
assignment procedure. */
 R_I3C_DynamicAddressAssignmentStart(&g_i3c_ctrl, I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA,
0, 1);
 break;
         }
 default:
      {
 break;
         }
         }
       }
 default:
      {
 break;
       }
    }
}
```

I3C Slave In-band Interrupts Example

This is a basic example of writing In-band Interrupts in I3C Slave mode.

```
void i3c_slave_ibi_write_basic_example (void)
{
    uint8_t ibi_write_buffer[EXAMPLE_WRITE_BUFFER_SIZE];
 /* Initiate an In-band interrupt in slave mode.
  * Note: If the slave does not have an IBI payload or if it is a Hot-Join request,
the write buffer should be set
  * to NULL and the write length should be set to 0. */
```

```
 fsp_err_t status = R_I3C_IbiWrite(&g_i3c_ctrl, I3C_IBI_TYPE_INTERRUPT,
ibi_write_buffer, sizeof(ibi_write_buffer));
    assert(FSP_SUCCESS == status);
 /* Wait for the ibiWrite transfer to complete. */
    i3c_app_event_wait(I3C_EVENT_IBI_WRITE_COMPLETE);
}
void i3c_slave_ibi_write_basic_example_callback (i3c_callback_args_t const * const
p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_IBI_WRITE_COMPLETE:
      {
 /* Notify the application that the IBI write is complete. */
            i3c_app_event_notify(I3C_EVENT_IBI_WRITE_COMPLETE);
 break;
       }
 default:
      {
 break;
       }
    }
}
```

I3C Master Common Command Codes Example

This is a basic example of sending Common Command Codes in I3C Master mode.

```
void i3c_master_ccc_example (void)
{
 static uint8_t command_buffer[EXAMPLE_READ_BUFFER_SIZE];
 /* Setup the command descriptor. */
 static i3c_command_descriptor_t command_descriptor =
    {
       .command_code = I3C_CCC_DIRECT_GETSTATUS,
```

```
 /* Set a buffer for storing the data read by the command. */
       .p_buffer    = command_buffer,
 /* The length for a GETSTATUS command is 2 bytes. */
       .length       = 2,
 /* Terminate the transfer with a STOP condition. */
       .restart      = false,
 /* The GETSTATUS command is a Direct Get Command so rnw should be true. */
       .rnw          = true,
    };
 /* Send the command. */
 fsp_err_t status = R_I3C_CommandSend(&g_i3c_ctrl, &command_descriptor);
    assert(FSP_SUCCESS == status);
 /* Wait for the command to complete. */
    i3c_app_event_wait(I3C_EVENT_COMMAND_COMPLETE);
 /* The command_buffer will have the status info that was read from the slave device.
*/
}
void i3c_master_ccc_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_COMMAND_COMPLETE:
      {
 /* Notify the application that the command is complete. */
            i3c_app_event_notify(I3C_EVENT_COMMAND_COMPLETE);
 break;
       }
 default:
      {
 break;
       }
    }
}
```

I3C Slave Common Command Codes Example

This is a basic example of receiving Common Command Codes in I3C Slave mode.

```
void i3c_slave_ccc_example (void)
{
 static uint8_t read_buffer[EXAMPLE_READ_BUFFER_SIZE];
 /* Broadcast and Direct Set commands will be read into the read_buffer the same way
that
  * a normal SDR Master Write / Slave Read transfer is read. */
 fsp_err_t status = R_I3C_Read(&g_i3c_ctrl, read_buffer, sizeof(read_buffer), false);
    assert(FSP_SUCCESS == status);
 /* Wait for the command to complete. */
    i3c_app_event_wait(I3C_EVENT_COMMAND_COMPLETE);
}
void i3c_slave_ccc_example_callback (i3c_callback_args_t const * const p_args)
{
 switch (p_args->event)
    {
 case I3C_EVENT_COMMAND_COMPLETE:
      {
 /* The command code and transfer size will be available in p_args.
  * If the command code is a Broadcast or Direct Set, then data will
  * be stored in the read buffer provided by i3c_api_t::read.
  * If the command code is a Direct Get, then the data will be automatically
  * sent from device SFR. */
            i3c_app_event_notify(I3C_EVENT_COMMAND_COMPLETE);
 break;
       }
 default:
      {
 break;
       }
    }
}
```

Data Structures

struct	i3c_clock_stalling_t
struct	i3c_bitrate_settings_t
struct	i3c_ibi_control_t
struct	i3c_slave_command_response_info_t
struct	i3c_instance_ctrl_t
struct	i3c_extended_cfg_t

Macros

#define	I3C_DEVICE_INDEX_EXTENDED_DEVICE
#define	I3C_EVENT_STATUS_SUCCESS The transfer was completed as expected. More...
#define	I3C_EVENT_STATUS_PARITY A parity error was detected.
#define	I3C_EVENT_STATUS_FRAME A frame error was detected.
#define	I3C_EVENT_STATUS_ADDRESS_HEADER An Address Header error wasdetected.
#define	I3C_EVENT_STATUS_NACK The transfer was NACK'd.
#define	I3C_EVENT_STATUS_OVERFLOW A Receive FIFO overflow or Transmit FIFO underflow occurred.
#define	I3C_EVENT_STATUS_ABORTED_TO_MASTER In slave mode, the write transfer was ended via the 'T' bit.
#define	I3C_EVENT_STATUS_ABORTED

	In master mode, the transfer was aborted.

#define	I3C_EVENT_STATUS_IBI_NACK_DISABLED An IBI was NACK'd and the a DISEC command was sent.

Enumerations

enum	i3c_bitrate_mode_t
enum	i3c_activity_state_t
enum	i3c_data_rate_setting_t
enum	i3c_clock_data_turnaround_t

Data Structure Documentation

◆ i3c_clock_stalling_t

struct i3c_clock_stalling_t

Clock stalling settings.

Data Fields		
uint32_t	assigned_address_phase_enable: 1	Enable Clock Stalling during the address phase of the ENTDAA command.
uint32_t	transition_phase_enable: 1	Enable Clock Stalling during the transition bit in read transfers.
uint32_t	parity_phase_enable: 1	Enable Clock Stalling during the parity bit period in write transfers.
uint32_t	ack_phase_enable: 1	Enable Clock Stalling during the ACK/NACK phase.
uint16_t	clock_stalling_time	The amount of time to stall the clock in I3C source clock ticks.

◆ i3c_bitrate_settings_t

struct i3c_bitrate_settings_t

Bitrate settings for configuring the SCL clock frequency.

Data Fields		
uint32_t	stdbr	The standard bitrate settings.
uint32_t	extbr	The extended bitrate settings.
i3c_clock_stalling_t	clock_stalling	Clock Stalling settings (See

		Master Clock Stalling in the MIPI I3C Specification v1.0).

◆ i3c_ibi_control_t

struct i3c_ibi_control_t		
Settings for controlling the drivers behavior in response to IBIs.		
Data Fields		
uint32_t	hot_join_acknowledge: 1	If false, NACK all Hot Join requests.
uint32_t	notify_rejected_hot_join_requests: 1	Notify the application when an IBI Hot-Join request has been NACK'd.
uint32_t	notify_rejected_mastership_requests: 1	Notify the application when an IBI Mastership request has been NACK'd.
uint32_t	notify_rejected_interrupt_requests: 1	Notify the application when an IBI Interrupt request has been NACK'd.

◆ i3c_slave_command_response_info_t

struct i3c_slave_command_response_info_t		
Default configuration settings for the slave response to Direct Get Common Command Codes.		
Data Fields		
bool	inband_interrupt_enable	Enable IBI interrupts. Slave Event Settings (See ENEC and DISEC in the MIPI I3C Specification v1.0).
bool	mastership_request_enable	Enable Mastership requests.
bool	hotjoin_request_enable	Enable Hot-Join requests.
i3c_activity_state_t	activity_state	Starting Activity State (See ENTASn in the MIPI I3C Specification v1.0).
uint16_t	write_length	Max Write Length (See SETMWL and GETMWL in the MIPI I3C Specification v1.0).
uint16_t	read_length	Max Read Length (See SETMRL and GETMRL in the MIPI I3C Specification v1.0).
uint8_t	ibi_payload_length	Number of bytes that will be written by an IBI (See SETMRL and GETMRL in the MIPI I3C Specification v1.0).

i3c_data_rate_setting_t	write_data_rate	Max Write Data Rate. Max Data Rate Settings (See GETMXDS in the MIPI I3C Specification v1.0).
i3c_data_rate_setting_t	read_data_rate	Max Read Data Rate.
i3c_clock_data_turnaround_t	clock_data_turnaround	Max Data Speed Turnaround.
bool	read_turnaround_time_enable	Enable transmission of the of the Max Read Max Read Turnaround Time.
uint32_t	read_turnaround_time	Max Read Turnaround Time.
uint8_t	oscillator_frequency	This byte represents the Slave's internal oscillator frequency in increments of 0.5 MHz (500kHz), up to 127.5 MHz. (See GETXTIME in the MIPI I3C Specification v1.1).
uint8_t	oscillator_inaccuracy	Oscillator inaccuracy in 0.5% increments of 0% up to 25.5% (See GETXTIME in the MIPI I3C Specification v1.1).

◆ i3c_instance_ctrl_t

struct i3c_instance_ctrl_t	
Channel control block. DO NOT INITIALIZE. Initialization occurs when i3c_api_t::open is called.	
Public Member Functions	
i3c_slave_info_t current_slave_info	BSP_ALIGN_VARIABLE (4)
	The last i3c_slave_info_t read during ENTDAA.
Data Fields	
uint32_t	open
	Indicates whether the open() API has been successfully called.
R_I3C0_Type *	p_reg
	Base register for this channel.
volatile uint32_t	internal_state
	Used to track the current state of the driver.

uint8_t	current_command_code
	The current Common Command Code that is being transferred.
uint32_t	device_index
	The device index selected using i3c_api_t::deviceSelect.
i3c_bitrate_mode_t	device_bitrate_mode
	Runtime bitrate settings to use for the next transfer.
uint32_t	next_word
	The next word that will be written to the FIFO.
uint32_t	ibi_next_word
	The next word that will be written to the IBI FIFO.
i3c_write_buffer_descriptor_t	write_buffer_descriptor
	Buffer descriptor for keeping track of a write transfer.
i3c_read_buffer_descriptor_t	read_buffer_descriptor
	Buffer descriptor for keeping track of a read transfer.
i3c_read_buffer_descriptor_t	ibi_buffer_descriptor
	Buffer descriptor for keeping track of an IBI read/write transfer.
volatile uint32_t	read_transfer_count_final
	The total number of bytes read during a read transfer.

volatile uint32_t	ibi_transfer_count_final
	The total number of bytes read during an IBI transfer.
i3c_cfg_t const *	p_cfg
	A pointer to the configuration structure provided during open.

◆ **i3c_extended_cfg_t**

struct i3c_extended_cfg_t

Extended configuration for r_i3c.

Data Fields

i3c_bitrate_settings_t	bitrate_settings	Bitrate settings configuring the frequency and duty cycle for SCL.
i3c_ibi_control_t	ibi_control	Configure the driver's behavior in response to IBIs.
uint32_t	bus_free_detection_time	The time in I3C reference clock ticks needed in order to detect the bus free condition (See "Bus Free Condition" in the MIPI I3C Specification v1.0).
uint32_t	bus_available_detection_time	The time in I3C reference clock ticks needed in order to detect the bus available condition (See "Bus Available Condition" in the MIPI I3C Specification v1.0).
uint32_t	bus_idle_detection_time	The time in I3C reference clock ticks needed in order to detect the bus idle condition (See "Bus Idle Condition" in the MIPI I3C Specification v1.0).
bool	timeout_detection_enable	Notify the application if SCL is stuck high or low.
i3c_slave_command_response_info_t	slave_command_response_info	Initial settings for configuring the slave's responses to received commands.
IRQn_Type	resp_irq	Response Queue Full IRQ number.
IRQn_Type	rx_irq	Receive FIFO Full IRQ number.
IRQn_Type	tx_irq	Transmit FIFO Empty IRQ

		number.
IRQn_Type	rcv_irq	Receive Status Queue Full IRQ number.
IRQn_Type	ibi_irq	IBI IRQ number.
IRQn_Type	eei_irq	EEI IRQ number.
uint8_t	ipl	Interrupt Priority for Resp, Rx, Tx, and RCV IRQs.
uint8_t	eei_ipl	Error and Event Interrupt Priority.

Macro Definition Documentation

◆ I3C_DEVICE_INDEX_EXTENDED_DEVICE

```
#define I3C_DEVICE_INDEX_EXTENDED_DEVICE
```

Index for selecting the device defined in the extended address table.

◆ I3C_EVENT_STATUS_SUCCESS

```
#define I3C_EVENT_STATUS_SUCCESS
```

The transfer was completed as expected.

Event Status Provided by the callback.

Enumeration Type Documentation

◆ i3c_bitrate_mode_t

enum i3c_bitrate_mode_t	
Bitrate settings that can be selected at run-time using i3c_api_t::deviceSelect.	
Enumerator	
I3C_BITRATE_MODE_I2C_STDBR	Use the period settings defined in STDBRH/L.
I3C_BITRATE_MODE_I2C_EXTBR	Use the period settings defined in EXTBRH/L.
I3C_BITRATE_MODE_I3C_SDR0_STDBR	Use the period settings defined in STDBRH/L (I3C Devices only).
I3C_BITRATE_MODE_I3C_SDR1_EXTBR	Use the period settings defined in EXTBRH/L (I3C Devices only).
I3C_BITRATE_MODE_I3C_SDR2_STDBR_X2	Use the period settings defined in STDBRH/L x 2 (I3C Devices only).
I3C_BITRATE_MODE_I3C_SDR3_EXTBR_X2	Use the period settings defined in EXTBRH/L x 2 (I3C Devices only).
I3C_BITRATE_MODE_I3C_SDR4_EXTBR_X4	Use the period settings defined in EXTBRH/L x 4 (I3C Devices only).

◆ i3c_activity_state_t

enum i3c_activity_state_t	
Supported activity states for ENTASn Command (See ENTASn in the MIPI I3C Specification v1.0).	
Enumerator	
I3C_ACTIVITY_STATE_ENTAS0	Activity Interval (1 microsecond).
I3C_ACTIVITY_STATE_ENTAS1	Activity Interval (100 microseconds).
I3C_ACTIVITY_STATE_ENTAS2	Activity Interval (2 milliseconds).
I3C_ACTIVITY_STATE_ENTAS3	Activity Interval (50 milliseconds).

◆ i3c_data_rate_setting_t

enum i3c_data_rate_setting_t	
Maximum Sustained Data Rate for non-CCC messages sent by Master Device to Slave Device (See GETMXDS in the MIPI I3C Specification v1.0).	
Enumerator	
I3C_DATA_RATE_SETTING_FSCL_MAX	There is no data rate limit.
I3C_DATA_RATE_SETTING_8MHZ	The max sustained data rate is 8 Mhz.
I3C_DATA_RATE_SETTING_6MHZ	The max sustained data rate is 6 Mhz.
I3C_DATA_RATE_SETTING_4MHZ	The max sustained data rate is 4 Mhz.
I3C_DATA_RATE_SETTING_2MHZ	The max sustained data rate is 2 Mhz.

◆ i3c_clock_data_turnaround_t

enum i3c_clock_data_turnaround_t	
Clock to Data Turnaround Time (See GETMXDS in the MIPI I3C Specification v1.0).	
Enumerator	
I3C_CLOCK_DATA_TURNAROUND_8NS	Clock to turnaround time is 8 nanoseconds or less.
I3C_CLOCK_DATA_TURNAROUND_9NS	Clock to turnaround time is 9 nanoseconds or less.
I3C_CLOCK_DATA_TURNAROUND_10NS	Clock to turnaround time is 10 nanoseconds or less.
I3C_CLOCK_DATA_TURNAROUND_11NS	Clock to turnaround time is 11 nanoseconds or less.
I3C_CLOCK_DATA_TURNAROUND_12NS	Clock to turnaround time is 12 nanoseconds or less.
I3C_CLOCK_DATA_TURNAROUND_EXTENDED	Clock to turnaround time is greater than 12 nanoseconds.

Function Documentation

◆ R_I3C_Open()

fsp_err_t R_I3C_Open (i3c_ctrl_t *const *p_api_ctrl*, i3c_cfg_t const *const *p_cfg*)

Configure an I3C instance. Implements i3c_api_t::open.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was invalid.
FSP_ERR_ALREADY_OPEN	Open has already been called for this instance.
FSP_ERR_UNSUPPORTED	A selected feature is not supported with the current configuration.

◆ R_I3C_Enable()

fsp_err_t R_I3C_Enable (i3c_ctrl_t *const *p_api_ctrl*)

Enable the I3C device.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_INVALID_MODE	This instance is already enabled.

◆ R_I3C_DeviceCfgSet()

fsp_err_t R_I3C_DeviceCfgSet (i3c_ctrl_t *const *p_api_ctrl*, i3c_device_cfg_t const *const *p_device_cfg*)

Set the configuration for this device.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_UNSUPPORTED	The device cannot be a secondary master if master support is disabled.

◆ R_I3C_MasterDeviceTableSet()

fsp_err_t R_I3C_MasterDeviceTableSet (i3c_ctrl_t *const *p_api_ctrl*, uint32_t *device_index*, i3c_device_table_cfg_t const *const *p_device_table_cfg*)

Configure an entry in the master device table. This function is called in master mode in order to configure the devices on the I3C bus. It may also be called in slave mode when the slave receives the DEFSVLS command.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_UNSUPPORTED	Mastership requests must be rejected is slave support is disabled.

◆ R_I3C_SlaveStatusSet()

fsp_err_t R_I3C_SlaveStatusSet (i3c_ctrl_t *const *p_api_ctrl*, i3c_device_status_t *status*)

Set the status returned to the master in response to a GETSTATUS command.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_INVALID_MODE	The instance is not in slave mode.
FSP_ERR_UNSUPPORTED	Slave support is disabled.

◆ R_I3C_DeviceSelect()

fsp_err_t R_I3C_DeviceSelect (i3c_ctrl_t *const *p_api_ctrl*, uint32_t *device_index*, uint32_t *bitrate_mode*)

In master mode, select the device for the next transfer. This function is not used in slave mode.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_INVALID_MODE	This operation is prohibited in slave mode.
FSP_ERR_UNSUPPORTED	Master support is disabled.

◆ R_I3C_DynamicAddressAssignmentStart()

fsp_err_t R_I3C_DynamicAddressAssignmentStart (i3c_ctrl_t *const *p_api_ctrl*, i3c_address_assignment_mode_t *address_assignment_mode*, uint32_t *starting_device_index*, uint32_t *device_count*)

Start the Dynamic Address Assignment Process. Implements i3c_api_t::dynamicAddressAssignmentStart.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL or invalid.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_INVALID_MODE	This operation is prohibited in slave mode.
FSP_ERR_IN_USE	The operation could not be completed because the driver is busy.
FSP_ERR_UNSUPPORTED	Master support is disabled.

◆ R_I3C_CommandSend()

fsp_err_t R_I3C_CommandSend (i3c_ctrl_t *const *p_api_ctrl*, i3c_command_descriptor_t * *p_command_descriptor*)

Send a broadcast or direct command to slave devices on the bus. Implements i3c_api_t::commandSend.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_IN_USE	The operation could not be completed because the driver is busy.
FSP_ERR_INVALID_MODE	This driver is not in master mode.
FSP_ERR_INVALID_ALIGNMENT	The buffer must be aligned to 4 bytes. If it is a read operation, the length also be a multiple of 4 bytes.
FSP_ERR_UNSUPPORTED	Master support must be enabled to call this function. Slave support must be enabled when sending the GETACCMST command.

◆ R_I3C_Write()

fsp_err_t R_I3C_Write (i3c_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_data*, uint32_t *length*, bool *restart*)

Set the write buffer for the transfer. In master mode, start the transfer. When the transfer is completed send a stop condition or a repeated-start. Implements i3c_api_t::write.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_IN_USE	The operation could not be completed because the driver is busy.
FSP_ERR_INVALID_MODE	This driver is disabled.
FSP_ERR_INVALID_ALIGNMENT	The buffer must be aligned to 4 bytes.

◆ R_I3C_Read()

fsp_err_t R_I3C_Read (i3c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t *length*, bool *restart*)

Set the read buffer for the transfer. In master mode, start the transfer. When the transfer is completed send a stop condition or a repeated-start. Implements i3c_api_t::read.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_IN_USE	The operation could not be completed because the driver is busy.
FSP_ERR_INVALID_MODE	This driver is disabled.
FSP_ERR_INVALID_ALIGNMENT	The buffer must be aligned to 4 bytes and the length must be a multiple of 4 bytes.

◆ R_I3C_IbiWrite()

fsp_err_t R_I3C_IbiWrite (i3c_ctrl_t *const *p_api_ctrl*, i3c_ibi_type_t *ibi_type*, uint8_t const *const *p_data*, uint32_t *length*)

Initiate an IBI write operation (This function is only used in slave mode).

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_IN_USE	The operation could not be completed because the driver is busy.
FSP_ERR_INVALID_MODE	This function is only called in slave mode.
FSP_ERR_INVALID_ALIGNMENT	The buffer must be aligned to 4 bytes.
FSP_ERR_UNSUPPORTED	Slave supoprt is disabled.

◆ R_I3C_IbiRead()

fsp_err_t R_I3C_IbiRead (i3c_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t *length*)

Set the read buffer for storing received IBI data (This function is only used in master mode).

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.
FSP_ERR_INVALID_MODE	This function is only called in master mode.
FSP_ERR_INVALID_ALIGNMENT	The buffer must be aligned to 4 bytes and the length must be a multiple of 4 bytes.
FSP_ERR_UNSUPPORTED	Master support is disabled.

◆ R_I3C_Close()

fsp_err_t R_I3C_Close (i3c_ctrl_t *const *p_api_ctrl*)

Close the I3C instance. Implements i3c_api_t::close.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_NOT_OPEN	This instance has not been opened yet.

4.2.31 Interrupt Controller Unit (r_icu)

Modules

Functions

fsp_err_t	R_ICU_ExternalIrqOpen (external_irq_ctrl_t *const p_api_ctrl, external_irq_cfg_t const *const p_cfg)
fsp_err_t	R_ICU_ExternalIrqEnable (external_irq_ctrl_t *const p_api_ctrl)
fsp_err_t	R_ICU_ExternalIrqDisable (external_irq_ctrl_t *const p_api_ctrl)

fsp_err_t	R_ICU_ExternalIrqCallbackSet (external_irq_ctrl_t *const p_api_ctrl, void(*p_callback)(external_irq_callback_args_t *), void const *const p_context, external_irq_callback_args_t *const p_callback_memory)
fsp_err_t	R_ICU_ExternalIrqClose (external_irq_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the ICU peripheral on RA MCUs. This module implements the External IRQ Interface.

Overview

The Interrupt Controller Unit (ICU) controls which event signals are linked to the NVIC, DTC, and DMAC modules. The R_ICU software module only implements the External IRQ Interface. The external_irq interface is for configuring interrupts to fire when a trigger condition is detected on an external IRQ pin.

Note

Multiple instances are used when more than one external interrupt is needed. Configure each instance with different channels and properties as needed for the specific interrupt.

Features

- Supports configuring interrupts for IRQ pins on the target MCUs
  - Enabling and disabling interrupt generation.
  - Configuring interrupt trigger on rising edge, falling edge, both edges, or low level signal.
  - Enabling and disabling the IRQ noise filter.
- Supports configuring a user callback function, which will be invoked by the HAL module when an external pin interrupt is generated.

Configuration

Build Time Configurations for r_icu

The following build time configurations are defined in fsp_cfg/r_icu_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Input > External IRQ (r_icu)

This module can be added to the Stacks tab via New Stack > Input > External IRQ (r_icu). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_external_irq0	Module name.
Channel	Value must be an integer between 0 and 15	0	Specify the hardware channel.
Trigger	• Falling • Rising • Both Edges • Low Level	Rising	Select the signal edge or state that triggers an interrupt.
Digital Filtering	• Enabled • Disabled	Disabled	Select if the digital noise filter should be enabled.
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	• PCLK / 1 • PCLK / 8 • PCLK / 32 • PCLK / 64	PCLK / 64	Select the clock divider for the digital noise filter.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided here. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers
Pin Interrupt Priority	MCU Specific Options		Select the PIN interrupt priority.

Clock Configuration

The ICU peripheral module doesn't require any specific clock settings.

Note

The digital filter uses PCLKB as the clock source for sampling the IRQ pin.

Pin Configuration

The pin for the external interrupt channel must be configured as an input with IRQ Input Enabled.

Usage Notes

Digital Filter

The digital filter is used to reject trigger conditions that are too short. The trigger condition must be longer than three periods of the filter clock. The filter clock frequency is determined by PCLKB and the external_irq_pclk_div_t setting.

MIN_PULSE_WIDTH = EXTERNAL_IRQ_PCLKB_DIV / PCLKB_FREQUENCY * 3

DMAC/DTC

When using an External IRQ pin to trigger a DMAC/DTC transfer, the External IRQ pin must be opened before the transfer instance is opened.

Examples

Basic Example

This is a basic example of minimal use of the ICU in an application.

```
#define ICU_IRQN_PIN BSP_IO_PORT_02_PIN_06
#define ICU_IRQN 6
/* Called from icu_irq_isr */
void external_irq_callback (external_irq_callback_args_t * p_args)
{
    (void) p_args;
    g_external_irq_complete = 1;
}
void simple_example ()
{
 /* Example Configuration */
 external_irq_cfg_t icu_cfg =
    {
       .channel       = ICU_IRQN,
       .trigger       = EXTERNAL_IRQ_TRIG_RISING,
       .filter_enable = false,
       .pclk_div      = EXTERNAL_IRQ_PCLK_DIV_BY_1,
       .p_callback    = external_irq_callback,
       .p_context    = 0,
       .ipl          = 0,
       .irq          = (IRQn_Type) 0,
    };
 /* Configure the external interrupt. */
 fsp_err_t err = R_ICU_ExternalIrqOpen(&g_icu_ctrl, &icu_cfg);
    assert(FSP_SUCCESS == err);
 /* Enable the external interrupt. */
 /* Enable not required when used with ELC or DMAC. */
```

```
    err = R_ICU_ExternalIrqEnable(&g_icu_ctrl);
    assert(FSP_SUCCESS == err);
 while (0 == g_external_irq_complete)
    {
 /* Wait for interrupt. */
    }
}
```

Data Structures

struct	icu_instance_ctrl_t

Data Structure Documentation

◆ icu_instance_ctrl_t

struct icu_instance_ctrl_t

ICU private control block. DO NOT MODIFY. Initialization occurs when R_ICU_ExternalIrqOpen is called.

Data Fields

uint32_t	open
	Used to determine if channel control block is in use.
IRQn_Type	irq
	NVIC interrupt number.
uint8_t	channel
	Channel.
void const *	p_context

Field Documentation

◆ p_context

void const* icu_instance_ctrl_t::p_context

Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.

Function Documentation

◆ R_ICU_ExternalIrqOpen()

fsp_err_t R_ICU_ExternalIrqOpen (external_irq_ctrl_t *const *p_api_ctrl*, external_irq_cfg_t const *const *p_cfg*)

Configure an IRQ input pin for use with the external interrupt interface. Implements external_irq_api_t::open.

The Open function is responsible for preparing an external IRQ pin for operation.

Return values

FSP_SUCCESS	Open successful.
FSP_ERR_ASSERTION	One of the following is invalid: • p_ctrl or p_cfg is NULL
FSP_ERR_ALREADY_OPEN	The channel specified has already been opened. No configurations were changed. Call the associated Close function to reconfigure the channel.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel requested in p_cfg is not available on the device selected in r_bsp_cfg.h.
FSP_ERR_INVALID_ARGUMENT	p_cfg->p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback function.

Note

This function is reentrant for different channels. It is not reentrant for the same channel.

◆ R_ICU_ExternalIrqEnable()

fsp_err_t R_ICU_ExternalIrqEnable (external_irq_ctrl_t *const *p_api_ctrl*)

Enable external interrupt for specified channel at NVIC. Implements external_irq_api_t::enable.

Return values

FSP_SUCCESS	Interrupt Enabled successfully.
FSP_ERR_ASSERTION	The p_ctrl parameter was null.
FSP_ERR_NOT_OPEN	The channel is not opened.
FSP_ERR_IRQ_BSP_DISABLED	Requested IRQ is not defined in this system

◆ R_ICU_ExternalIrqDisable()

fsp_err_t R_ICU_ExternalIrqDisable (external_irq_ctrl_t *const *p_api_ctrl*)

Disable external interrupt for specified channel at NVIC. Implements external_irq_api_t::disable.

Return values

FSP_SUCCESS	Interrupt disabled successfully.
FSP_ERR_ASSERTION	The p_ctrl parameter was null.
FSP_ERR_NOT_OPEN	The channel is not opened.
FSP_ERR_IRQ_BSP_DISABLED	Requested IRQ is not defined in this system

◆ R_ICU_ExternalIrqCallbackSet()

fsp_err_t R_ICU_ExternalIrqCallbackSet (external_irq_ctrl_t *const *p_api_ctrl*, void(*)(external_irq_callback_args_t *) *p_callback*, void const *const *p_context*, external_irq_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements external_irq_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_ICU_ExternalIrqClose()

fsp_err_t R_ICU_ExternalIrqClose (external_irq_ctrl_t *const *p_api_ctrl*)

Close the external interrupt channel. Implements external_irq_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
FSP_ERR_NOT_OPEN	The channel is not opened.

4.2.32 I2C Master on IIC/I3C (r_iic_b_master)

Modules

Functions

fsp_err_t	R_IIC_B_MASTER_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg)
fsp_err_t	R_IIC_B_MASTER_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)
fsp_err_t	R_IIC_B_MASTER_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)
fsp_err_t	R_IIC_B_MASTER_Abort (i2c_master_ctrl_t *const p_api_ctrl)
fsp_err_t	R_IIC_B_MASTER_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode)
fsp_err_t	R_IIC_B_MASTER_Close (i2c_master_ctrl_t *const p_api_ctrl)
fsp_err_t	R_IIC_B_MASTER_CallbackSet (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void const *const p_context, i2c_master_callback_args_t *const p_callback_memory)
fsp_err_t	R_IIC_B_MASTER_StatusGet (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status)

Detailed Description

I2C Driver for the IIC/I3C peripheral on RA MCUs. This module implements the I2C Master Interface.

Overview

The I2C master on IIC/I3C HAL module supports transactions with an I2C Slave device. Callbacks must be provided which are invoked when a transmit or receive operation has completed. The callback argument will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- I2C Master Read from a slave device.

- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
    - Optional (build time) DTC support for read and write respectively.
    - Optional (build time) support for 10-bit slave addressing.

Configuration

Build Time Configurations for r_iic_b_master

The following build time configurations are defined in fsp_cfg/r_iic_b_master_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DTC on Transmission and Reception	• Enabled • Disabled	Disabled	If enabled, DTC instances will be included in the build for both transmission and reception.
10-bit slave addressing	• Enabled • Disabled	Disabled	If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode.

Configurations for Connectivity > I2C Master (r_iic_b_master)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Master (r_iic_b_master). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_i2c_master0	Module name.
Channel	Value must be a non-negative integer	0	Specify the IIC channel.
Rate	• Standard • Fast-mode • Fast-mode plus	Standard	Select the transfer rate. If the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or

			equal to the requested transfer rate are used. The theoretical calculated transfer rate and duty cycle are printed in a comment in the generated iic_b_master_extended_cfg_t structure.
Rise Time (ns)	Value must be a non-negative integer	120	Set the rise time (tr) in nanoseconds.
Fall Time (ns)	Value must be a non-negative integer	120	Set the fall time (tf) in nanoseconds.
Duty Cycle (%)	Value must be an integer between 0 and 100	50	Set the SCL duty cycle.
Slave Address	Value must be non-negative	0x00	Specify the slave address.
Address Mode	• 7-Bit • 10-Bit	7-Bit	Select the slave address mode. Ensure 10-bit slave addressing is enabled in the configuration to use 10-Bit setting here.
Timeout Mode	• Short Mode • Long Mode	Short Mode	Select the timeout mode to detect bus hang.
Timeout during SCL Low	• Enabled • Disabled	Enabled	Select if the timeout can occur when SCL is held low for a duration longer than what is set in the timeout mode.
Callback	Name must be a valid C symbol	g_iic_b_master0_callback	A user callback function must be provided. This will be called from the interrupt service routine (ISR) upon IIC transaction completion reporting the transaction status.
Interrupt Priority Level	MCU Specific Options		Select the interrupt priority level. This is set for TXI, RXI, TEI and ERI interrupts.

Clock Configuration

The I3C peripheral module uses the IICCLK or PCLKD (based on the MCU) as its clock source for the bus clock. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the clocks are configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

Pin Configuration

The I3C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

Usage Notes

Interrupt Configuration

- The IIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel used must be enabled in the properties of the selected device.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

IIC Master Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current IICCLK/PCLKD (based on the MCU) settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

Enabling DTC with the IIC

- DTC transfer support is configurable and is disabled from the build by default. IIC driver provides two DTC instances for transmission and reception respectively. The DTC instances can be enabled individually during configuration.
- DTC is helpful for minimizing interrupts during large transactions. Many I2C applications have shorter transactions. These applications will likely not see any improvement with DTC. I2C often runs at a much slower speed than the CPU core clock. Some applications with longer transactions may prefer servicing the interrupts at the I2C bitrate to the overhead of bringing in the DTC driver.
- For further details on DTC please refer Data Transfer Controller (r_dtc)

Multiple Devices on the Bus

- A single IIC instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet API.

Multi-Master Support

- If multiple masters are connected on the same bus, the I2C Master is capable of detecting bus busy state before initiating the communication.

Restart

- IIC master can hold the the bus after an I2C transaction by issuing a repeated start condition.

Examples

Basic Example

This is a basic example of minimal use of the r_iic_master in an application. This example shows how this driver can be used for basic read and write operations.

```
iic_b_master_instance_ctrl_t g_i2c_device_ctrl_1;
i2c_master_cfg_t g_i2c_device_cfg_1 =
{
    .channel       = I2C_CHANNEL,
    .rate          = I2C_MASTER_RATE_FAST,
    .slave         = I2C_SLAVE_EEPROM,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_callback,    // Callback
    .p_context    = &g_i2c_device_ctrl_1,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_b_master_cfg_extend
};
void i2c_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_callback_event = p_args->event;
}
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the IIC module */
    err = R_IIC_B_MASTER_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
```

```
      g_i2c_tx_buffer[i] = (uint8_t) i;
   }
 /* Send data to I2C slave */
   g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
   err = R_IIC_B_MASTER_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
   assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
   {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
      timeout_ms--;;
   }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
   {
      __BKPT(0);
   }
 /* Read data back from the I2C slave */
   g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
   timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
   err = R_IIC_B_MASTER_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
   assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
   {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
      timeout_ms--;;
   }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
   {
      __BKPT(0);
   }
 /* Verify the read data */
```

```
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

Multiple Slave devices on the same channel (bus)

This example demonstrates how a single IIC driver can be used to communicate with different slave devices which are on the same channel.

Note

The callback function from the first example applies to this example as well.

```
iic_b_master_instance_ctrl_t g_i2c_device_ctrl_2;
i2c_master_cfg_t g_i2c_device_cfg_2 =
{
    .channel       = I2C_CHANNEL,
    .rate          = I2C_MASTER_RATE_STANDARD,
    .slave         = I2C_SLAVE_TEMP_SENSOR,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_callback,    // Callback
    .p_context    = &g_i2c_device_ctrl_2,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_b_master_cfg_extend
};
void single_channel_multi_slave (void)
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_B_MASTER_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Clear the recieve buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
```

```
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_IIC_B_MASTER_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
    err = R_IIC_B_MASTER_SlaveAddressSet(&g_i2c_device_ctrl_2,
I2C_SLAVE_DISPLAY_ADAPTER, I2C_MASTER_ADDR_MODE_7BIT);
    assert(FSP_SUCCESS == err);
    g_i2c_tx_buffer[0]   = 0xAA;       // NOLINT
    g_i2c_tx_buffer[1]   = 0xBB;       // NOLINT
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_B_MASTER_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
}
```

Data Structures

struct	iic_b_master_clock_settings_t
struct	iic_b_master_instance_ctrl_t
struct	iic_b_master_extended_cfg_t

Enumerations

enum	iic_b_master_timeout_mode_t
enum	iic_b_master_timeout_scl_low_t

Data Structure Documentation

◆ iic_b_master_clock_settings_t

struct iic_b_master_clock_settings_t

I2C clock settings

Data Fields		
uint8_t	cks_value	Internal Reference Clock Select.
uint8_t	brh_value	High-level period of SCL clock.
uint8_t	brl_value	Low-level period of SCL clock.

◆ iic_b_master_instance_ctrl_t

struct iic_b_master_instance_ctrl_t

I2C control structure. DO NOT INITIALIZE.

◆ iic_b_master_extended_cfg_t

struct iic_b_master_extended_cfg_t

R_IIC_B extended configuration

Data Fields		
iic_b_master_timeout_mode_t	timeout_mode	Timeout Detection Time Select: Long Mode = 0 and Short Mode = 1.
iic_b_master_timeout_scl_low_t	timeout_scl_low	Allows timeouts to occur when SCL is held low.
iic_b_master_clock_settings_t	clock_settings	I2C Clock settings.
uint32_t	iic_clock_freq	I2C Clock frequency in Hz.

Enumeration Type Documentation

◆ iic_b_master_timeout_mode_t

enum iic_b_master_timeout_mode_t

I2C Timeout mode parameter definition

Enumerator	
IIC_B_MASTER_TIMEOUT_MODE_LONG	Timeout Detection Time Select: Long Mode -> TMOS = 0.
IIC_B_MASTER_TIMEOUT_MODE_SHORT	Timeout Detection Time Select: Short Mode -> TMOS = 1.

◆ iic_b_master_timeout_scl_low_t

enum iic_b_master_timeout_scl_low_t

Enumerator	
IIC_B_MASTER_TIMEOUT_SCL_LOW_DISABLED	Timeout detection during SCL low disabled.
IIC_B_MASTER_TIMEOUT_SCL_LOW_ENABLED	Timeout detection during SCL low enabled.

Function Documentation

◆ R_IIC_B_MASTER_Open()

fsp_err_t R_IIC_B_MASTER_Open (i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg*)

Opens the I2C device.

Return values

FSP_SUCCESS	Requested clock rate was set exactly.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel is not available on this MCU.
FSP_ERR_ASSERTION	Parameter check failure due to one or more reasons below: 1. p_api_ctrl or p_cfg is NULL. 2. extended parameter is NULL. 3. Callback parameter is NULL. 4. Set the rate to fast mode plus on a channel which does not support it. 5. Invalid IRQ number assigned

◆ R_IIC_B_MASTER_Read()

fsp_err_t R_IIC_B_MASTER_Read (i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart*)

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	p_api_ctrl, p_dest or bytes is NULL.
FSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer.
FSP_ERR_IN_USE	Bus busy condition. Another transfer was in progress.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block.

◆ R_IIC_B_MASTER_Write()

fsp_err_t R_IIC_B_MASTER_Write (i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart*)

Performs a write to the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_TX_COMPLETE in the callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	p_api_ctrl or p_src is NULL.
FSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer.
FSP_ERR_IN_USE	Bus busy condition. Another transfer was in progress.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block.

◆ R_IIC_B_MASTER_Abort()

fsp_err_t R_IIC_B_MASTER_Abort (i2c_master_ctrl_t *const *p_api_ctrl*)

Safely aborts any in-progress transfer and forces the IIC peripheral into ready state.

Return values

FSP_SUCCESS	Channel was reset successfully.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block.

Note

A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.

◆ R_IIC_B_MASTER_SlaveAddressSet()

fsp_err_t R_IIC_B_MASTER_SlaveAddressSet (i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode*)

Sets address and addressing mode of the slave device. This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

Return values

FSP_SUCCESS	Address of the slave is set correctly.
FSP_ERR_ASSERTION	Pointer to control structure is NULL.
FSP_ERR_IN_USE	Another transfer was in-progress.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block.

◆ R_IIC_B_MASTER_Close()

fsp_err_t R_IIC_B_MASTER_Close (i2c_master_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. May power down IIC peripheral. This function will safely terminate any in-progress I2C transfers.

Return values

FSP_SUCCESS	Device closed without issue.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_B_MASTER_Open to initialize the control block.

Note

A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.

◆ R_IIC_B_MASTER_CallbackSet()

fsp_err_t R_IIC_B_MASTER_CallbackSet (i2c_master_ctrl_t *const *p_api_ctrl*, void(*)(i2c_master_callback_args_t *) *p_callback*, void const *const *p_context*, i2c_master_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements i2c_master_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_IIC_B_MASTER_StatusGet()

fsp_err_t R_IIC_B_MASTER_StatusGet (i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_status_t * *p_status*)

Provides driver status.

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.

4.2.33 I2C Slave on IIC/I3C (r_iic_b_slave)

Modules

Functions

fsp_err_t	R_IIC_B_SLAVE_Open (i2c_slave_ctrl_t *const p_api_ctrl, i2c_slave_cfg_t const *const p_cfg)
fsp_err_t	R_IIC_B_SLAVE_Read (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t	R_IIC_B_SLAVE_Write (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes)
fsp_err_t	R_IIC_B_SLAVE_Close (i2c_slave_ctrl_t *const p_api_ctrl)
fsp_err_t	R_IIC_B_SLAVE_CallbackSet (i2c_slave_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_slave_callback_args_t *), void const *const p_context, i2c_slave_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the IIC/I3C peripheral on RA MCUs. This module implements the I2C Slave Interface.

Overview

Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- Reads data written by master device.
- Write data which is read by master device.

- Can accept 0x00 as slave address.
- Can be assigned a 10-bit address.
- Clock stretching is supported and can be implemented via callbacks.
- Provides Transmission/Reception transaction size in the callback.
- I2C Slave can notify the following events via callbacks: Transmission/Reception Request, Transmission/Reception Request for more data, Transmission/Reception Completion, Error Condition.

Configuration

Build Time Configurations for r_iic_b_slave

The following build time configurations are defined in fsp_cfg/r_iic_b_slave_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Connectivity > I2C Slave (r_iic_b_slave)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Slave (r_iic_b_slave). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Interrupt Priority Level > Transmit, Receive, and Transmit End	MCU Specific Options		Select the interrupt priority level. This is set for TXI, RXI, and TEI interrupts.
Interrupt Priority Level > Error	MCU Specific Options		Select the interrupt priority level. This is set for ERI interrupt.
Name	Name must be a valid C symbol	g_i2c_slave0	Module name.
Channel	Value must be a non-negative integer	0	Specify the IIC channel.
Rate	• Standard • Fast-mode • Fast-mode plus	Standard	Select the transfer rate. If the delay for the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated

			delay is printed in a comment in the generated iic_b_slave_extended_cfg_t structure.
Internal Reference Clock	• I2C Clock / 1 • I2C Clock / 2 • I2C Clock / 4 • I2C Clock / 8 • I2C Clock / 16 • I2C Clock / 32 • I2C Clock / 64 • I2C Clock / 128	I2C Clock / 1	Select the internal reference clock for IIC slave. The internal reference clock is used only to determine the clock frequency of the noise filter samples. I2C Clock can be either IICCLK or PCLKD based on the MCU.
Digital Noise Filter Stage Select	• Disabled • Single-stage filter • 2-stage filter • 3-stage filter • 4-stage filter	3-stage filter	Select the number of digital filter stages for IIC Slave.
Slave Address	Value must be non-negative	0x00	Specify the slave address.
General Call	• Enabled • Disabled	Disabled	Allows the slave to respond to general call address: 0x00.
Address Mode	• 7-Bit • 10-Bit	7-Bit	Select the slave address mode.
Clock Stretching	• Enabled • Disabled	Disabled	Configure Clock Stretching.
Callback	Name must be a valid C symbol	g_iic_b_slave0_callback	A user callback function must be provided. This will be called from the interrupt service routine (ISR) to report I2C Slave transaction events and status.

Clock Configuration

The IIC/I3C peripheral module uses the IICCLK or PCLKD (based on the MCU) as its clock source for the bus clock. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the clocks are configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

Pin Configuration

The IIC/I3C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

Usage Notes

Interrupt Configuration

- The IIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel must be enabled in the properties of the selected device.
- The interrupt priority of ERI can be set higher than or equal to the interrupt priorities of RXI, TXI and TEI.
  Note
  : During master-write slave-read type of operations if the slave device requires to perform clock stretching after the last data byte is received, a higher priority ERI will ensure that the ongoing transaction is completed (by accepting the Stop/Restart condition from the master) before the next transaction is initiated.
  : To support clock stretching (Holding SCL low after the falling edge of the 9th clock cycle), 'Clock Stretching' configuration must be enabled.

Callback

- A callback function must be provided which will be invoked for the cases below:
  - An I2C Master initiates a transmission or reception: I2C_SLAVE_EVENT_TX_REQUEST; I2C_SLAVE_EVENT_RX_REQUEST
  - A Transmission or reception has been completed: I2C_SLAVE_EVENT_TX_COMPLETE; I2C_SLAVE_EVENT_RX_COMPLETE
  - An I2C Master is requesting to read or write more data: I2C_SLAVE_EVENT_TX_MORE_REQUEST; I2C_SLAVE_EVENT_RX_MORE_REQUEST
  - Error conditions: I2C_SLAVE_EVENT_ABORTED
  - An I2C Master initiates a general call by passing 0x00 as slave address: I2C_SLAVE_EVENT_GENERAL_CALL
- The callback arguments will contain information about the transaction status/events, bytes transferred and a pointer to the user defined context.
- Clock stretching is enabled by the use of callbacks. This means that the IIC slave can hold the clock line SCL LOW to force the I2C Master into a wait state.
- The table below shows I2C Slave event handling expected in user code:

IIC Slave Callback Event	IIC Slave API expected to be called
I2C_SLAVE_EVENT_ABORTED	Handle event based on application
I2C_SLAVE_EVENT_RX_COMPLETE	Handle event based on application
I2C_SLAVE_EVENT_TX_COMPLETE	Handle event based on application
I2C_SLAVE_EVENT_RX_REQUEST	R_IIC_B_SLAVE_Read API. If the slave is a Write Only device call this API with 0 bytes to send a NACK to the master.
I2C_SLAVE_EVENT_TX_REQUEST	R_IIC_B_SLAVE_Write API
I2C_SLAVE_EVENT_RX_MORE_REQUEST	R_IIC_B_SLAVE_Read API. If the slave cannot read any more data call this API with 0 bytes to send a NACK to the master.
I2C_SLAVE_EVENT_TX_MORE_REQUEST	R_IIC_B_SLAVE_Write API

I2C_SLAVE_EVENT_GENERAL_CALL	R_IIC_B_SLAVE_Read

- If parameter checking is enabled and R_IIC_B_SLAVE_Read API is not called for I2C_SLAVE_EVENT_RX_REQUEST and/or I2C_SLAVE_EVENT_RX_MORE_REQUEST, the slave will send a NACK to the master and would eventually timeout.
- R_IIC_B_SLAVE_Write API is not called for I2C_SLAVE_EVENT_TX_REQUEST and/or I2C_SLAVE_EVENT_TX_MORE_REQUEST:
    - Slave timeout is less than Master timeout: The slave will timeout and release the bus causing the master to read 0xFF for every remaining byte.
    - Slave timeout is more than Master timeout: The master will timeout first followed by the slave.

IIC Slave Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current I2C Clock (IICCLK or PCLKD based on the MCU) settings is calculated and used.

Limitations

- When 'Clock Stretching' configuration is enabled, the receive operation will not utilize the double buffer arrangement in hardware for a continuous read. This means that the read operation would happen in single byte units such that the active master would send the next byte only when the slave has read the current byte of data.

Examples

Basic Example

This is a basic example of minimal use of the R_IIC_B_SLAVE in an application. This example shows how this driver can be used for basic read and write operations.

```
iic_b_master_instance_ctrl_t g_i2c_master_ctrl;

i2c_master_cfg_t g_i2c_master_cfg =

{

    .channel       = I2C_MASTER_CHANNEL_2,

    .rate          = I2C_MASTER_RATE_STANDARD,

    .slave         = I2C_7BIT_ADDR_IIC_SLAVE,

    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,

    .p_callback    = i2c_master_callback, // Callback

    .p_context    = &g_i2c_master_ctrl,

    .p_transfer_tx = NULL,

    .p_transfer_rx = NULL,

    .p_extend      = &g_iic_master_cfg_extend_standard_mode

};
```

```
iic_b_slave_instance_ctrl_t g_i2c_slave_ctrl;
i2c_slave_cfg_t g_i2c_slave_cfg =
{
    .channel    = I2C_SLAVE_CHANNEL_0,
    .rate       = I2C_SLAVE_RATE_STANDARD,
    .slave      = I2C_7BIT_ADDR_IIC_SLAVE,
    .addr_mode = I2C_SLAVE_ADDR_MODE_7BIT,
    .p_callback = i2c_slave_callback, // Callback
    .p_context = &g_i2c_slave_ctrl,
    .p_extend   = &g_iic_slave_cfg_extend_standard_mode
};
void i2c_master_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_master_callback_event = p_args->event;
}
void i2c_slave_callback (i2c_slave_callback_args_t * p_args)
{
    g_i2c_slave_callback_event = p_args->event;
 if ((p_args->event == I2C_SLAVE_EVENT_RX_COMPLETE) || (p_args->event ==
I2C_SLAVE_EVENT_TX_COMPLETE))
    {
 /* Transaction Successful */
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_RX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_RX_MORE_REQUEST))
    {
 /* Read from Master */
       err = R_IIC_B_SLAVE_Read(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       assert(FSP_SUCCESS == err);
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_TX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_TX_MORE_REQUEST))
    {
```

```
 /* Write to master */
       err = R_IIC_B_SLAVE_Write(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       assert(FSP_SUCCESS == err);
    }
 else
    {
 /* Error Event - reported through g_i2c_slave_callback_event */
    }
}
void basic_example (void)
{
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    g_slave_transfer_length = I2C_BUFFER_SIZE_BYTES;
 /* Pin connections:
  * Channel 0 SDA <--> Channel 2 SDA
  * Channel 0 SCL <--> Channel 2 SCL
  */
 /* Initialize the IIC Slave module */
    err = R_IIC_B_SLAVE_Open(&g_i2c_slave_ctrl, &g_i2c_slave_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the IIC Master module */
    err = R_IIC_B_MASTER_Open(&g_i2c_master_ctrl, &g_i2c_master_cfg);
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_master_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
    g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
```

```
    err = R_IIC_B_MASTER_Write(&g_i2c_master_ctrl, &g_i2c_master_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers
  * The Slave Callback will call the R_IIC_B_SLAVE_Read API to service the Master
Write Request.
  */
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_master_callback_event ||
 I2C_SLAVE_EVENT_RX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
       (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
    {
       __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
    g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
    timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_B_MASTER_Read(&g_i2c_master_ctrl, &g_i2c_master_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers
  * The Slave Callback will call the R_IIC_SLAVE_Write API to service the Master Read
Request.
  */
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_master_callback_event ||
 I2C_SLAVE_EVENT_TX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
```

```
    }
 if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
       (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
    {
       __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_master_tx_buffer, g_i2c_master_rx_buffer,
I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

Data Structures

struct	iic_b_slave_clock_settings_t
struct	iic_b_slave_extended_cfg_t

Data Structure Documentation

◆ iic_b_slave_clock_settings_t

struct iic_b_slave_clock_settings_t		
I2C clock settings		
Data Fields		
uint8_t	cks_value	Internal Reference Clock Select.
uint8_t	brl_value	Low-level period of SCL clock.
uint8_t	digital_filter_stages	Number of digital filter stages based on brl_value.

◆ iic_b_slave_extended_cfg_t

struct iic_b_slave_extended_cfg_t		
R_IIC_SLAVE extended configuration		
Data Fields		
iic_b_slave_clock_settings_t	clock_settings	I2C Clock settings.

Function Documentation

◆ R_IIC_B_SLAVE_Open()

fsp_err_t R_IIC_B_SLAVE_Open (i2c_slave_ctrl_t *const *p_api_ctrl*, i2c_slave_cfg_t const *const *p_cfg*)

Opens the I2C slave device.

Return values

FSP_SUCCESS	I2C slave device opened successfully.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel is not available on this MCU.
FSP_ERR_INVALID_ARGUMENT	Error interrupt priority is lower than Transmit, Receive and Transmit End interrupt priority
FSP_ERR_ASSERTION	Parameter check failure due to one or more reasons below: 1. p_api_ctrl or p_cfg is NULL. 2. extended parameter is NULL. 3. Callback parameter is NULL. 4. Set the rate to fast mode plus on a channel which does not support it. 5. Invalid IRQ number assigned

◆ R_IIC_B_SLAVE_Read()

fsp_err_t R_IIC_B_SLAVE_Read (i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Performs a read from the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C slave read operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_RX_COMPLETE in the callback. In case the master continues to write more data, an I2C_SLAVE_EVENT_RX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

Return values

FSP_SUCCESS	Function executed without issue
FSP_ERR_ASSERTION	p_api_ctrl, bytes or p_dest is NULL.
FSP_ERR_IN_USE	Another transfer was in progress.
FSP_ERR_NOT_OPEN	Device is not open.

◆ R_IIC_B_SLAVE_Write()

fsp_err_t R_IIC_B_SLAVE_Write (i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*)

Performs a write to the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C slave write operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_TX_COMPLETE in the callback. In case the master continues to read more data, an I2C_SLAVE_EVENT_TX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	p_api_ctrl or p_src is NULL.
FSP_ERR_IN_USE	Another transfer was in progress.
FSP_ERR_NOT_OPEN	Device is not open.

◆ R_IIC_B_SLAVE_Close()

fsp_err_t R_IIC_B_SLAVE_Close (i2c_slave_ctrl_t *const *p_api_ctrl*)

Closes the I2C device.

Return values

FSP_SUCCESS	Device closed successfully.
FSP_ERR_NOT_OPEN	Device not opened.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.

◆ R_IIC_B_SLAVE_CallbackSet()

fsp_err_t R_IIC_B_SLAVE_CallbackSet (i2c_slave_ctrl_t *const *p_api_ctrl*, void(*)(i2c_slave_callback_args_t *) *p_callback*, void const *const *p_context*, i2c_slave_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements i2c_slave_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.34 I2C Master on IIC (r_iic_master)

Modules

Functions

fsp_err_t	R_IIC_MASTER_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg)
fsp_err_t	R_IIC_MASTER_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)
fsp_err_t	R_IIC_MASTER_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)
fsp_err_t	R_IIC_MASTER_Abort (i2c_master_ctrl_t *const p_api_ctrl)
fsp_err_t	R_IIC_MASTER_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode)
fsp_err_t	R_IIC_MASTER_Close (i2c_master_ctrl_t *const p_api_ctrl)
fsp_err_t	R_IIC_MASTER_CallbackSet (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void const *const p_context, i2c_master_callback_args_t *const p_callback_memory)
fsp_err_t	R_IIC_MASTER_StatusGet (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status)

Detailed Description

Driver for the IIC peripheral on RA MCUs. This module implements the I2C Master Interface.

Overview

The I2C master on IIC HAL module supports transactions with an I2C Slave device. Callbacks must be provided which are invoked when a transmit or receive operation has completed. The callback argument will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
  - Optional (build time) DTC support for read and write respectively.
  - Optional (build time) support for 10-bit slave addressing.

Configuration

Build Time Configurations for r_iic_master

The following build time configurations are defined in fsp_cfg/r_iic_master_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DTC on Transmission and Reception	• Enabled • Disabled	Disabled	If enabled, DTC instances will be included in the build for both transmission and reception.
10-bit slave addressing	• Enabled • Disabled	Disabled	If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode.

Configurations for Connectivity > I2C Master (r_iic_master)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Master (r_iic_master). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_i2c_master0	Module name.
Channel	Value must be a non-negative integer	0	Specify the IIC channel.
Rate	• Standard • Fast-mode • Fast-mode plus	Standard	Select the transfer rate. If the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated transfer rate and duty cycle are printed in a comment in the generated iic_master_extended_cfg_t structure.
Rise Time (ns)	Value must be a non-negative integer	120	Set the rise time (tr) in nanoseconds.
Fall Time (ns)	Value must be a non-negative integer	120	Set the fall time (tf) in nanoseconds.
Duty Cycle (%)	Value must be an integer between 0 and 100	50	Set the SCL duty cycle.
Slave Address	Value must be non-negative	0x00	Specify the slave address.
Address Mode	• 7-Bit • 10-Bit	7-Bit	Select the slave address mode. Ensure 10-bit slave addressing is enabled in the configuration to use 10-Bit setting here.
Timeout Mode	• Short Mode • Long Mode	Short Mode	Select the timeout mode to detect bus hang.
Timeout during SCL Low	• Enabled • Disabled	Enabled	Select if the timeout can occur when SCL is held low for a duration longer than what is set in the timeout mode.

Callback	Name must be a valid C symbol	NULL	A user callback function must be provided. This will be called from the interrupt service routine (ISR) upon IIC transaction completion reporting the transaction status.
Interrupt Priority Level	MCU Specific Options		Select the interrupt priority level. This is set for TXI, RXI, TEI and ERI interrupts.

Clock Configuration

The IIC peripheral module uses the PCLKB as its clock source. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the PCLKB is configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

Pin Configuration

The IIC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

Usage Notes

Interrupt Configuration

- The IIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel used must be enabled in the properties of the selected device.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

IIC Master Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

Enabling DTC with the IIC

- DTC transfer support is configurable and is disabled from the build by default. IIC driver provides two DTC instances for transmission and reception respectively. The DTC instances can be enabled individually during configuration.
- DTC is helpful for minimizing interrupts during large transactions. Many I2C applications have shorter transactions. These applications will likely not see any improvement with DTC. I2C often runs at a much slower speed than the CPU core clock. Some applications with longer transactions may prefer servicing the interrupts at the I2C bitrate to the overhead of bringing in the DTC driver.
- For further details on DTC please refer Data Transfer Controller (r_dtc)

Multiple Devices on the Bus

- A single IIC instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet API.

Multi-Master Support

- If multiple masters are connected on the same bus, the I2C Master is capable of detecting bus busy state before initiating the communication.

Restart

- IIC master can hold the the bus after an I2C transaction by issuing a repeated start condition.

Examples

Basic Example

This is a basic example of minimal use of the r_iic_master in an application. This example shows how this driver can be used for basic read and write operations.

```
iic_master_instance_ctrl_t g_i2c_device_ctrl_1;
i2c_master_cfg_t g_i2c_device_cfg_1 =
{
    .channel        = I2C_CHANNEL,
    .rate           = I2C_MASTER_RATE_FAST,
    .slave          = I2C_SLAVE_EEPROM,
    .addr_mode     = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback     = i2c_callback,    // Callback
    .p_context     = &g_i2c_device_ctrl_1,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend       = &g_iic_master_cfg_extend
};
void i2c_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_callback_event = p_args->event;
}
void basic_example (void)
{
 fsp_err_t err;
```

```
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the IIC module */
    err = R_IIC_MASTER_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_IIC_MASTER_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_MASTER_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
```

```
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

Multiple Slave devices on the same channel (bus)

This example demonstrates how a single IIC driver can be used to communicate with different slave devices which are on the same channel.

Note

The callback function from the first example applies to this example as well.

```
iic_master_instance_ctrl_t g_i2c_device_ctrl_2;
i2c_master_cfg_t g_i2c_device_cfg_2 =
{
    .channel       = I2C_CHANNEL,
    .rate          = I2C_MASTER_RATE_STANDARD,
    .slave         = I2C_SLAVE_TEMP_SENSOR,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_callback,    // Callback
    .p_context    = &g_i2c_device_ctrl_2,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend
};
void single_channel_multi_slave (void)
```

```
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_MASTER_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Clear the recieve buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_IIC_MASTER_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
    err = R_IIC_MASTER_SlaveAddressSet(&g_i2c_device_ctrl_2,
I2C_SLAVE_DISPLAY_ADAPTER, I2C_MASTER_ADDR_MODE_7BIT);
    assert(FSP_SUCCESS == err);
    g_i2c_tx_buffer[0]   = 0xAA;       // NOLINT
    g_i2c_tx_buffer[1]   = 0xBB;       // NOLINT
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_IIC_MASTER_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
```

```
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
        timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
        __BKPT(0);
    }
}
```

Data Structures

struct	iic_master_clock_settings_t
struct	iic_master_instance_ctrl_t
struct	iic_master_extended_cfg_t

Enumerations

enum	iic_master_timeout_mode_t
enum	iic_master_timeout_scl_low_t

Data Structure Documentation

◆ iic_master_clock_settings_t

struct iic_master_clock_settings_t

I2C clock settings

Data Fields		
uint8_t	cks_value	Internal Reference Clock Select.
uint8_t	brh_value	High-level period of SCL clock.
uint8_t	brl_value	Low-level period of SCL clock.

◆ iic_master_instance_ctrl_t

struct iic_master_instance_ctrl_t

I2C control structure. DO NOT INITIALIZE.

◆ iic_master_extended_cfg_t

struct iic_master_extended_cfg_t

R_IIC extended configuration

Data Fields

iic_master_timeout_mode_t	timeout_mode	Timeout Detection Time Select: Long Mode = 0 and Short Mode = 1.
iic_master_timeout_scl_low_t	timeout_scl_low	Allows timeouts to occur when SCL is held low.
iic_master_clock_settings_t	clock_settings	I2C Clock settings.

Enumeration Type Documentation

◆ iic_master_timeout_mode_t

enum iic_master_timeout_mode_t	
I2C Timeout mode parameter definition	
Enumerator	
IIC_MASTER_TIMEOUT_MODE_LONG	Timeout Detection Time Select: Long Mode -> TMOS = 0.
IIC_MASTER_TIMEOUT_MODE_SHORT	Timeout Detection Time Select: Short Mode -> TMOS = 1.

◆ iic_master_timeout_scl_low_t

enum iic_master_timeout_scl_low_t	
Enumerator	
IIC_MASTER_TIMEOUT_SCL_LOW_DISABLED	Timeout detection during SCL low disabled.
IIC_MASTER_TIMEOUT_SCL_LOW_ENABLED	Timeout detection during SCL low enabled.

Function Documentation

◆ R_IIC_MASTER_Open()

fsp_err_t R_IIC_MASTER_Open (i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg*)

Opens the I2C device.

Return values

FSP_SUCCESS	Requested clock rate was set exactly.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel is not available on this MCU.
FSP_ERR_ASSERTION	Parameter check failure due to one or more reasons below: 1. p_api_ctrl or p_cfg is NULL. 2. extended parameter is NULL. 3. Callback parameter is NULL. 4. Set the rate to fast mode plus on a channel which does not support it. 5. Invalid IRQ number assigned

◆ R_IIC_MASTER_Read()

fsp_err_t R_IIC_MASTER_Read (i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart*)

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	p_api_ctrl, p_dest or bytes is NULL.
FSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer.
FSP_ERR_IN_USE	Bus busy condition. Another transfer was in progress.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block.

◆ R_IIC_MASTER_Write()

fsp_err_t R_IIC_MASTER_Write (i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart*)

Performs a write to the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_TX_COMPLETE in the callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	p_api_ctrl or p_src is NULL.
FSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer.
FSP_ERR_IN_USE	Bus busy condition. Another transfer was in progress.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block.

◆ R_IIC_MASTER_Abort()

fsp_err_t R_IIC_MASTER_Abort (i2c_master_ctrl_t *const *p_api_ctrl*)

Safely aborts any in-progress transfer and forces the IIC peripheral into ready state.

Return values

FSP_SUCCESS	Channel was reset successfully.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block.

Note

A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.

◆ R_IIC_MASTER_SlaveAddressSet()

fsp_err_t R_IIC_MASTER_SlaveAddressSet (i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode*)

Sets address and addressing mode of the slave device. This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

Return values

FSP_SUCCESS	Address of the slave is set correctly.
FSP_ERR_ASSERTION	Pointer to control structure is NULL.
FSP_ERR_IN_USE	Another transfer was in-progress.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block.

◆ R_IIC_MASTER_Close()

fsp_err_t R_IIC_MASTER_Close (i2c_master_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. May power down IIC peripheral. This function will safely terminate any in-progress I2C transfers.

Return values

FSP_SUCCESS	Device closed without issue.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Handle is not initialized. Call R_IIC_MASTER_Open to initialize the control block.

Note

A callback will not be invoked in case an in-progress transfer gets aborted by calling this API.

◆ R_IIC_MASTER_CallbackSet()

fsp_err_t R_IIC_MASTER_CallbackSet (i2c_master_ctrl_t *const *p_api_ctrl*, void(*)(i2c_master_callback_args_t *) *p_callback*, void const *const *p_context*, i2c_master_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements i2c_master_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_IIC_MASTER_StatusGet()

fsp_err_t R_IIC_MASTER_StatusGet (i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_status_t * *p_status*)

Provides driver status.

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.

4.2.35 I2C Slave on IIC (r_iic_slave)

Modules

Functions

fsp_err_t	R_IIC_SLAVE_Open (i2c_slave_ctrl_t *const p_api_ctrl, i2c_slave_cfg_t const *const p_cfg)
fsp_err_t	R_IIC_SLAVE_Read (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t	R_IIC_SLAVE_Write (i2c_slave_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes)

fsp_err_t	R_IIC_SLAVE_Close (i2c_slave_ctrl_t *const p_api_ctrl)
fsp_err_t	R_IIC_SLAVE_CallbackSet (i2c_slave_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_slave_callback_args_t *), void const *const p_context, i2c_slave_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the IIC peripheral on RA MCUs. This module implements the I2C Slave Interface.

Overview

Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100-kHz transaction rate.
  - Fast Mode Support with up to 400-kHz transaction rate.
  - Fast Mode Plus Support with up to 1-MHz transaction rate.
- Reads data written by master device.
- Write data which is read by master device.
- Can accept 0x00 as slave address.
- Can be assigned a 10-bit address.
- Clock stretching is supported and can be implemented via callbacks.
- Provides Transmission/Reception transaction size in the callback.
- I2C Slave can notify the following events via callbacks: Transmission/Reception Request, Transmission/Reception Request for more data, Transmission/Reception Completion, Error Condition.

Configuration

Build Time Configurations for r_iic_slave

The following build time configurations are defined in fsp_cfg/r_iic_slave_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Connectivity > I2C Slave (r_iic_slave)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Slave (r_iic_slave). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Interrupt Priority Level > Transmit, Receive, and Transmit End	MCU Specific Options		Select the interrupt priority level. This is set for TXI, RXI, and TEI interrupts.

Interrupt Priority Level > Error	MCU Specific Options		Select the interrupt priority level. This is set for ERI interrupt.
Name	Name must be a valid C symbol	g_i2c_slave0	Module name.
Channel	Value must be a non-negative integer	0	Specify the IIC channel.
Rate	• Standard • Fast-mode • Fast-mode plus	Standard	Select the transfer rate. If the delay for the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated delay is printed in a comment in the generated iic_slave_extended_cfg_t structure.
Internal Reference Clock	• PCLKB / 1 • PCLKB / 2 • PCLKB / 4 • PCLKB / 8 • PCLKB / 16 • PCLKB / 32 • PCLKB / 64 • PCLKB / 128	PCLKB / 1	Select the internal reference clock for IIC slave. The internal reference clock is used only to determine the clock frequency of the noise filter samples.
Digital Noise Filter Stage Select	• Disabled • Single-stage filter • 2-stage filter • 3-stage filter • 4-stage filter	3-stage filter	Select the number of digital filter stages for IIC Slave.
Slave Address	Value must be non-negative	0x00	Specify the slave address.
General Call	• Enabled • Disabled	Disabled	Allows the slave to respond to general call address: 0x00.
Address Mode	• 7-Bit • 10-Bit	7-Bit	Select the slave address mode.
Clock Stretching	• Enabled • Disabled	Disabled	Configure Clock Stretching.
Callback	Name must be a valid	NULL	A user callback

	C symbol	function must be provided. This will be called from the interrupt service routine (ISR) to report I2C Slave transaction events and status.

Clock Configuration

The IIC peripheral module uses the PCLKB as its clock source. The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate. If the PCLKB is configured in such a manner that the selected transfer rate cannot be achieved, an error will be returned.

Pin Configuration

The IIC peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

Usage Notes

Interrupt Configuration

- The IIC error (EEI), receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel must be enabled in the properties of the selected device.
- The interrupt priority of ERI can be set higher than or equal to the interrupt priorities of RXI, TXI and TEI.
  Note
  : During master-write slave-read type of operations if the slave device requires to perform clock stretching after the last data byte is received, a higher priority ERI will ensure that the ongoing transaction is completed (by accepting the Stop/Restart condition from the master) before the next transaction is initiated.
  : To support clock stretching (Holding SCL low after the falling edge of the 9th clock cycle), 'Clock Stretching' configuration must be enabled.

Callback

- A callback function must be provided which will be invoked for the cases below:
  - An I2C Master initiates a transmission or reception: I2C_SLAVE_EVENT_TX_REQUEST; I2C_SLAVE_EVENT_RX_REQUEST
  - A Transmission or reception has been completed: I2C_SLAVE_EVENT_TX_COMPLETE; I2C_SLAVE_EVENT_RX_COMPLETE
  - An I2C Master is requesting to read or write more data: I2C_SLAVE_EVENT_TX_MORE_REQUEST; I2C_SLAVE_EVENT_RX_MORE_REQUEST
  - Error conditions: I2C_SLAVE_EVENT_ABORTED
  - An I2C Master initiates a general call by passing 0x00 as slave address: I2C_SLAVE_EVENT_GENERAL_CALL
- The callback arguments will contain information about the transaction status/events, bytes transferred and a pointer to the user defined context.
- Clock stretching is enabled by the use of callbacks. This means that the IIC slave can hold the clock line SCL LOW to force the I2C Master into a wait state.
- The table below shows I2C Slave event handling expected in user code:

IIC Slave Callback Event	IIC Slave API expected to be called
I2C_SLAVE_EVENT_ABORTED	Handle event based on application
I2C_SLAVE_EVENT_RX_COMPLETE	Handle event based on application
I2C_SLAVE_EVENT_TX_COMPLETE	Handle event based on application
I2C_SLAVE_EVENT_RX_REQUEST	R_IIC_SLAVE_Read API. If the slave is a Write Only device call this API with 0 bytes to send a NACK to the master.
I2C_SLAVE_EVENT_TX_REQUEST	R_IIC_SLAVE_Write API
I2C_SLAVE_EVENT_RX_MORE_REQUEST	R_IIC_SLAVE_Read API. If the slave cannot read any more data call this API with 0 bytes to send a NACK to the master.
I2C_SLAVE_EVENT_TX_MORE_REQUEST	R_IIC_SLAVE_Write API
I2C_SLAVE_EVENT_GENERAL_CALL	R_IIC_SLAVE_Read

- If parameter checking is enabled and R_IIC_SLAVE_Read API is not called for I2C_SLAVE_EVENT_RX_REQUEST and/or I2C_SLAVE_EVENT_RX_MORE_REQUEST, the slave will send a NACK to the master and would eventually timeout.
- R_IIC_SLAVE_Write API is not called for I2C_SLAVE_EVENT_TX_REQUEST and/or I2C_SLAVE_EVENT_TX_MORE_REQUEST:
  - Slave timeout is less than Master timeout: The slave will timeout and release the bus causing the master to read 0xFF for every remaining byte.
  - Slave timeout is more than Master timeout: The master will timeout first followed by the slave.

IIC Slave Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLKB settings is calculated and used.

Limitations

- When 'Clock Stretching' configuration is enabled, the receive operation will not utilize the double buffer arrangement in hardware for a continuous read. This means that the read operation would happen in single byte units such that the active master would send the next byte only when the slave has read the current byte of data.

Examples

Basic Example

This is a basic example of minimal use of the R_IIC_SLAVE in an application. This example shows how this driver can be used for basic read and write operations.

```
iic_master_instance_ctrl_t g_i2c_master_ctrl;
i2c_master_cfg_t g_i2c_master_cfg =
```

```
{
    .channel       = I2C_MASTER_CHANNEL_2,
    .rate          = I2C_MASTER_RATE_STANDARD,
    .slave         = I2C_7BIT_ADDR_IIC_SLAVE,
    .addr_mode    = I2C_MASTER_ADDR_MODE_7BIT,
    .p_callback    = i2c_master_callback, // Callback
    .p_context    = &g_i2c_master_ctrl,
    .p_transfer_tx = NULL,
    .p_transfer_rx = NULL,
    .p_extend      = &g_iic_master_cfg_extend_standard_mode
};
iic_slave_instance_ctrl_t g_i2c_slave_ctrl;
i2c_slave_cfg_t g_i2c_slave_cfg =
{
    .channel    = I2C_SLAVE_CHANNEL_0,
    .rate       = I2C_SLAVE_RATE_STANDARD,
    .slave      = I2C_7BIT_ADDR_IIC_SLAVE,
    .addr_mode = I2C_SLAVE_ADDR_MODE_7BIT,
    .p_callback = i2c_slave_callback, // Callback
    .p_context = &g_i2c_slave_ctrl,
    .p_extend   = &g_iic_slave_cfg_extend_standard_mode
};
void i2c_master_callback (i2c_master_callback_args_t * p_args)
{
    g_i2c_master_callback_event = p_args->event;
}
void i2c_slave_callback (i2c_slave_callback_args_t * p_args)
{
    g_i2c_slave_callback_event = p_args->event;
 if ((p_args->event == I2C_SLAVE_EVENT_RX_COMPLETE) || (p_args->event ==
I2C_SLAVE_EVENT_TX_COMPLETE))
    {
 /* Transaction Successful */
    }
```

```
 else if ((p_args->event == I2C_SLAVE_EVENT_RX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_RX_MORE_REQUEST))
    {
 /* Read from Master */
       err = R_IIC_SLAVE_Read(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       assert(FSP_SUCCESS == err);
    }
 else if ((p_args->event == I2C_SLAVE_EVENT_TX_REQUEST) || (p_args->event ==
I2C_SLAVE_EVENT_TX_MORE_REQUEST))
    {
 /* Write to master */
       err = R_IIC_SLAVE_Write(&g_i2c_slave_ctrl, g_i2c_slave_buffer,
g_slave_transfer_length);
       assert(FSP_SUCCESS == err);
    }
 else
    {
 /* Error Event - reported through g_i2c_slave_callback_event */
    }
}
void basic_example (void)
{
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    g_slave_transfer_length = I2C_BUFFER_SIZE_BYTES;
 /* Pin connections:
  * Channel 0 SDA <--> Channel 2 SDA
  * Channel 0 SCL <--> Channel 2 SCL
  */
 /* Initialize the IIC Slave module */
    err = R_IIC_SLAVE_Open(&g_i2c_slave_ctrl, &g_i2c_slave_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
```

```
/* Initialize the IIC Master module */
   err = R_IIC_MASTER_Open(&g_i2c_master_ctrl, &g_i2c_master_cfg);
   assert(FSP_SUCCESS == err);
/* Write some data to the transmit buffer */
for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
   {
      g_i2c_master_tx_buffer[i] = (uint8_t) i;
   }
/* Send data to I2C slave */
   g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
   g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
   err = R_IIC_MASTER_Write(&g_i2c_master_ctrl, &g_i2c_master_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
   assert(FSP_SUCCESS == err);
/* Since there is nothing else to do, block until Callback triggers
 * The Slave Callback will call the R_IIC_SLAVE_Read API to service the Master Write
Request.
 */
while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_master_callback_event ||
I2C_SLAVE_EVENT_RX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
   {
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
      timeout_ms--;
   }
if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
      (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
   {
      __BKPT(0);
   }
/* Read data back from the I2C slave */
   g_i2c_master_callback_event = I2C_MASTER_EVENT_ABORTED;
   g_i2c_slave_callback_event = I2C_SLAVE_EVENT_ABORTED;
   timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
   err = R_IIC_MASTER_Read(&g_i2c_master_ctrl, &g_i2c_master_rx_buffer[0],
```

```
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers
  * The Slave Callback will call the R_IIC_SLAVE_Write API to service the Master Read
Request.
  */
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_master_callback_event ||
 I2C_SLAVE_EVENT_TX_COMPLETE != g_i2c_slave_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;
    }
 if ((I2C_MASTER_EVENT_ABORTED == g_i2c_master_callback_event) ||
       (I2C_SLAVE_EVENT_ABORTED == g_i2c_slave_callback_event))
    {
       __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_master_tx_buffer, g_i2c_master_rx_buffer,
I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

Data Structures

struct	iic_slave_clock_settings_t
struct	iic_slave_extended_cfg_t

Data Structure Documentation

◆ iic_slave_clock_settings_t

struct iic_slave_clock_settings_t
I2C clock settings
Data Fields

uint8_t	cks_value	Internal Reference Clock Select.
uint8_t	brl_value	Low-level period of SCL clock.
uint8_t	digital_filter_stages	Number of digital filter stages based on brl_value.

◆ iic_slave_extended_cfg_t

struct iic_slave_extended_cfg_t		
R_IIC_SLAVE extended configuration		
Data Fields		
iic_slave_clock_settings_t	clock_settings	I2C Clock settings.

Function Documentation

◆ R_IIC_SLAVE_Open()

fsp_err_t R_IIC_SLAVE_Open (i2c_slave_ctrl_t *const *p_api_ctrl*, i2c_slave_cfg_t const *const *p_cfg*)

Opens the I2C slave device.

Return values

FSP_SUCCESS	I2C slave device opened successfully.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel is not available on this MCU.
FSP_ERR_INVALID_ARGUMENT	Error interrupt priority is lower than Transmit, Receive and Transmit End interrupt priority
FSP_ERR_ASSERTION	Parameter check failure due to one or more reasons below: 1. p_api_ctrl or p_cfg is NULL. 2. extended parameter is NULL. 3. Callback parameter is NULL. 4. Set the rate to fast mode plus on a channel which does not support it. 5. Invalid IRQ number assigned

◆ R_IIC_SLAVE_Read()

fsp_err_t R_IIC_SLAVE_Read (i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Performs a read from the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C slave read operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_RX_COMPLETE in the callback. In case the master continues to write more data, an I2C_SLAVE_EVENT_RX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

Return values

FSP_SUCCESS	Function executed without issue
FSP_ERR_ASSERTION	p_api_ctrl, bytes or p_dest is NULL.
FSP_ERR_IN_USE	Another transfer was in progress.
FSP_ERR_NOT_OPEN	Device is not open.

◆ R_IIC_SLAVE_Write()

fsp_err_t R_IIC_SLAVE_Write (i2c_slave_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*)

Performs a write to the I2C Master device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C slave write operation will begin. The caller will be notified when the operation has finished by an I2C_SLAVE_EVENT_TX_COMPLETE in the callback. In case the master continues to read more data, an I2C_SLAVE_EVENT_TX_MORE_REQUEST will be issued via callback. In case of errors, an I2C_SLAVE_EVENT_ABORTED will be issued via callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	p_api_ctrl or p_src is NULL.
FSP_ERR_IN_USE	Another transfer was in progress.
FSP_ERR_NOT_OPEN	Device is not open.

◆ R_IIC_SLAVE_Close()

fsp_err_t R_IIC_SLAVE_Close (i2c_slave_ctrl_t *const *p_api_ctrl*)

Closes the I2C device.

Return values

FSP_SUCCESS	Device closed successfully.
FSP_ERR_NOT_OPEN	Device not opened.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.

◆ R_IIC_SLAVE_CallbackSet()

fsp_err_t R_IIC_SLAVE_CallbackSet (i2c_slave_ctrl_t *const *p_api_ctrl*, void(*)(i2c_slave_callback_args_t *) *p_callback*, void const *const *p_context*, i2c_slave_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements i2c_slave_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.36 I/O Ports (r_ioport)

Modules

Functions

fsp_err_t	R_IOPORT_Open (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)
fsp_err_t	R_IOPORT_Close (ioport_ctrl_t *const p_ctrl)
fsp_err_t	R_IOPORT_PinsCfg (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)
fsp_err_t	R_IOPORT_PinCfg (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin,

	uint32_t cfg)
fsp_err_t	R_IOPORT_PinEventInputRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event)
fsp_err_t	R_IOPORT_PinEventOutputWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value)
fsp_err_t	R_IOPORT_PinRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value)
fsp_err_t	R_IOPORT_PinWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level)
fsp_err_t	R_IOPORT_PortDirectionSet (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask)
fsp_err_t	R_IOPORT_PortEventInputRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *event_data)
fsp_err_t	R_IOPORT_PortEventOutputWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value)
fsp_err_t	R_IOPORT_PortRead (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value)
fsp_err_t	R_IOPORT_PortWrite (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask)

Detailed Description

Driver for the I/O Ports peripheral on RA MCUs. This module implements the I/O Port Interface.

Overview

The I/O port pins operate as general I/O port pins, I/O pins for peripheral modules, interrupt input pins, analog I/O, port group function for the ELC, or bus control pins.

Features

The IOPORT HAL module can configure the following pin settings:

- Pin direction
- Default output state
- Pull-up
- NMOS/PMOS
- Drive strength
- Event edge trigger (falling, rising or both)
- Whether the pin is to be used as an IRQ pin

- Whether the pin is to be used as an analog pin
- Peripheral connection

The module also provides the following functionality:

- Read/write GPIO pins/ports
- Sets event output data
- Reads event input data

Configuration

The I/O PORT HAL module must be configured by the user for the desired operation. The operating state of an I/O pin can be set via the RA Configuraton tool. When the project is built a pin configuration file is created. The BSP will automatically configure the MCU IO ports accordingly at startup using the same API functions mentioned in this document.

Build Time Configurations for r_ioport

The following build time configurations are defined in fsp_cfg/r_ioport_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for System > I/O Port (r_ioport)

This module can be added to the Stacks tab via New Stack > System > I/O Port (r_ioport).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_ioport	Module name.
Port 1 ELC Trigger Source	MCU Specific Options		ELC source that will trigger PORT1
Port 2 ELC Trigger Source	MCU Specific Options		ELC source that will trigger PORT2
Port 3 ELC Trigger Source	MCU Specific Options		ELC source that will trigger PORT3
Port 4 ELC Trigger Source	MCU Specific Options		ELC source that will trigger PORT4
Port B ELC Trigger Source	MCU Specific Options		ELC source that will trigger PORTB
Port C ELC Trigger Source	MCU Specific Options		ELC source that will trigger PORTC
Port D ELC Trigger Source	MCU Specific Options		ELC source that will trigger PORTD

Port E ELC Trigger Source	MCU Specific Options		ELC source that will trigger PORTE
Pin Configuration Name	Name must be a valid C symbol	g_bsp_pin_cfg	Name for pin configuration structure

Clock Configuration

The I/O PORT HAL module does not require a specific clock configuration.

Pin Configuration

The IOPORT module is used for configuring pins.

Usage Notes

Port Group Function for ELC

Depending on pin configuration, the IOPORT module can perform automatic reads and writes on ports 1-4 on receipt of an ELC event.

When an event is received by a port, the state of the input pins on the port is saved in a hardware register. Simultaneously, the state of output pins on the port is set or cleared based on settings configured by the user. The functions R_IOPORT_PinEventInputRead and R_IOPORT_PortEventInputRead allow reading the last event input state of a pin or port, and event-triggered pin output can be configured through R_IOPORT_PinEventOutputWrite and R_IOPORT_PortEventOutputWrite.

In addition, each pin on ports 1-4 can be configured to trigger an ELC event on rising, falling or both edges. This event can be used to activate other modules when the pin changes state.

Note

The number of ELC-aware ports vary across MCUs. Refer to the Hardware User's Manual for your device for more details.

Examples

Basic Example

This is a basic example of minimal use of the IOPORT in an application.

```
void basic_example ()
{
 bsp_io_level_t readLevel;
 fsp_err_t      err;
 /* Initialize the IOPORT module and configure the pins
  * Note: The default pin configuration name in the RA Configuraton tool is
g_bsp_pin_cfg */
    err = R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
```

```
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Call R_IOPORT_PinsCfg if the configuration was not part of initial configurations
made in open */
    err = R_IOPORT_PinsCfg(&g_ioport_ctrl, &g_runtime_pin_cfg);
    assert(FSP_SUCCESS == err);
 /* Set Pin 00 of Port 06 to High */
    err = R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_06_PIN_00, BSP_IO_LEVEL_HIGH
);
    assert(FSP_SUCCESS == err);
 /* Read Pin 00 of Port 06*/
    err = R_IOPORT_PinRead(&g_ioport_ctrl, BSP_IO_PORT_06_PIN_00, &readLevel);
    assert(FSP_SUCCESS == err);
}
```

Blinky Example

This example uses IOPORT to configure and toggle a pin to blink an LED.

```
void blinky_example ()
{
 fsp_err_t err;
 /* Initialize the IOPORT module and configure the pins */
    err = R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Configure Pin as output
  * Call the R_IOPORT_PinCfg if the configuration was not part of initial
configurations made in open */
    err = R_IOPORT_PinCfg(&g_ioport_ctrl, BSP_IO_PORT_06_PIN_00,
BSP_IO_DIRECTION_OUTPUT);
    assert(FSP_SUCCESS == err);
 bsp_io_level_t level = BSP_IO_LEVEL_LOW;
 while (1)
    {
```

```
 /* Determine the next state of the LEDs */
 if (BSP_IO_LEVEL_LOW == level)
      {
            level = BSP_IO_LEVEL_HIGH;
       }
 else
      {
            level = BSP_IO_LEVEL_LOW;
       }
 /* Update LED on RA6M3-PK */
       err = R_IOPORT_PinWrite(&g_ioport_ctrl, BSP_IO_PORT_06_PIN_00, level);
      assert(FSP_SUCCESS == err);
 /* Delay */
 R_BSP_SoftwareDelay(100, BSP_DELAY_UNITS_MILLISECONDS); // NOLINT
    }
}
```

ELC Example

This is an example of using IOPORT with ELC events. The ELC event system allows the captured data to be stored when it occurs and then read back at a later time.

```
static elc_instance_ctrl_t g_elc_ctrl;
static elc_cfg_t g_elc_cfg;
void ioport_elc_example ()
{
 bsp_io_level_t eventValue;
 fsp_err_t      err;
 /* Initializes the software and sets the links defined in the control structure. */
    err = R_ELC_Open(&g_elc_ctrl, &g_elc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Create or modify a link between a peripheral function and an event source. */
    err = R_ELC_LinkSet(&g_elc_ctrl, (elc_peripheral_t) ELC_PERIPHERAL_IOPORT2,
ELC_EVENT_ELC_SOFTWARE_EVENT_0);
```

```
    assert(FSP_SUCCESS == err);
 /* Globally enable event linking in the ELC. */
    err = R_ELC_Enable(&g_elc_ctrl);
    assert(FSP_SUCCESS == err);
 /* Initialize the IOPORT module and configure the pins */
    err = R_IOPORT_Open(&g_ioport_ctrl, &g_bsp_pin_cfg);
    assert(FSP_SUCCESS == err);
 /* Call the R_IOPORT_PinCfg if the configuration was not part of initial
configurations made in open */
    err = R_IOPORT_PinCfg(&g_ioport_ctrl, BSP_IO_PORT_02_PIN_00,
BSP_IO_DIRECTION_INPUT);
    assert(FSP_SUCCESS == err);
 /* Generate an event signal through software to the linked peripheral. */
    err = R_ELC_SoftwareEventGenerate(&g_elc_ctrl, ELC_SOFTWARE_EVENT_0);
    assert(FSP_SUCCESS == err);
 /* Read Pin Event Input. The data(BSP_IO_LEVEL_HIGH/ BSP_IO_LEVEL_LOW) from
BSP_IO_PORT_02_PIN_00 is read into the
  * EIDR bit */
    err = R_IOPORT_PinEventInputRead(&g_ioport_ctrl, BSP_IO_PORT_02_PIN_00,
&eventValue);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	ioport_instance_ctrl_t

Enumerations

enum	ioport_port_pin_t

Data Structure Documentation

◆ ioport_instance_ctrl_t

struct ioport_instance_ctrl_t
IOPORT private control block. DO NOT MODIFY. Initialization occurs when R_IOPORT_Open() is called.

Enumeration Type Documentation

◆ ioport_port_pin_t

enum ioport_port_pin_t	
Superset list of all possible IO port pins.	
Enumerator	
IOPORT_PORT_00_PIN_00	IO port 0 pin 0.
IOPORT_PORT_00_PIN_01	IO port 0 pin 1.
IOPORT_PORT_00_PIN_02	IO port 0 pin 2.
IOPORT_PORT_00_PIN_03	IO port 0 pin 3.
IOPORT_PORT_00_PIN_04	IO port 0 pin 4.
IOPORT_PORT_00_PIN_05	IO port 0 pin 5.
IOPORT_PORT_00_PIN_06	IO port 0 pin 6.
IOPORT_PORT_00_PIN_07	IO port 0 pin 7.
IOPORT_PORT_00_PIN_08	IO port 0 pin 8.
IOPORT_PORT_00_PIN_09	IO port 0 pin 9.
IOPORT_PORT_00_PIN_10	IO port 0 pin 10.
IOPORT_PORT_00_PIN_11	IO port 0 pin 11.
IOPORT_PORT_00_PIN_12	IO port 0 pin 12.
IOPORT_PORT_00_PIN_13	IO port 0 pin 13.
IOPORT_PORT_00_PIN_14	IO port 0 pin 14.
IOPORT_PORT_00_PIN_15	IO port 0 pin 15.
IOPORT_PORT_01_PIN_00	IO port 1 pin 0.
IOPORT_PORT_01_PIN_01	IO port 1 pin 1.
IOPORT_PORT_01_PIN_02	IO port 1 pin 2.
IOPORT_PORT_01_PIN_03	IO port 1 pin 3.
IOPORT_PORT_01_PIN_04	

	IO port 1 pin 4.
IOPORT_PORT_01_PIN_05	IO port 1 pin 5.
IOPORT_PORT_01_PIN_06	IO port 1 pin 6.
IOPORT_PORT_01_PIN_07	IO port 1 pin 7.
IOPORT_PORT_01_PIN_08	IO port 1 pin 8.
IOPORT_PORT_01_PIN_09	IO port 1 pin 9.
IOPORT_PORT_01_PIN_10	IO port 1 pin 10.
IOPORT_PORT_01_PIN_11	IO port 1 pin 11.
IOPORT_PORT_01_PIN_12	IO port 1 pin 12.
IOPORT_PORT_01_PIN_13	IO port 1 pin 13.
IOPORT_PORT_01_PIN_14	IO port 1 pin 14.
IOPORT_PORT_01_PIN_15	IO port 1 pin 15.
IOPORT_PORT_02_PIN_00	IO port 2 pin 0.
IOPORT_PORT_02_PIN_01	IO port 2 pin 1.
IOPORT_PORT_02_PIN_02	IO port 2 pin 2.
IOPORT_PORT_02_PIN_03	IO port 2 pin 3.
IOPORT_PORT_02_PIN_04	IO port 2 pin 4.
IOPORT_PORT_02_PIN_05	IO port 2 pin 5.
IOPORT_PORT_02_PIN_06	IO port 2 pin 6.
IOPORT_PORT_02_PIN_07	IO port 2 pin 7.
IOPORT_PORT_02_PIN_08	IO port 2 pin 8.
IOPORT_PORT_02_PIN_09	IO port 2 pin 9.
IOPORT_PORT_02_PIN_10	IO port 2 pin 10.
IOPORT_PORT_02_PIN_11	IO port 2 pin 11.
IOPORT_PORT_02_PIN_12	

	IO port 2 pin 12.
IOPORT_PORT_02_PIN_13	IO port 2 pin 13.
IOPORT_PORT_02_PIN_14	IO port 2 pin 14.
IOPORT_PORT_02_PIN_15	IO port 2 pin 15.
IOPORT_PORT_03_PIN_00	IO port 3 pin 0.
IOPORT_PORT_03_PIN_01	IO port 3 pin 1.
IOPORT_PORT_03_PIN_02	IO port 3 pin 2.
IOPORT_PORT_03_PIN_03	IO port 3 pin 3.
IOPORT_PORT_03_PIN_04	IO port 3 pin 4.
IOPORT_PORT_03_PIN_05	IO port 3 pin 5.
IOPORT_PORT_03_PIN_06	IO port 3 pin 6.
IOPORT_PORT_03_PIN_07	IO port 3 pin 7.
IOPORT_PORT_03_PIN_08	IO port 3 pin 8.
IOPORT_PORT_03_PIN_09	IO port 3 pin 9.
IOPORT_PORT_03_PIN_10	IO port 3 pin 10.
IOPORT_PORT_03_PIN_11	IO port 3 pin 11.
IOPORT_PORT_03_PIN_12	IO port 3 pin 12.
IOPORT_PORT_03_PIN_13	IO port 3 pin 13.
IOPORT_PORT_03_PIN_14	IO port 3 pin 14.
IOPORT_PORT_03_PIN_15	IO port 3 pin 15.
IOPORT_PORT_04_PIN_00	IO port 4 pin 0.
IOPORT_PORT_04_PIN_01	IO port 4 pin 1.
IOPORT_PORT_04_PIN_02	IO port 4 pin 2.
IOPORT_PORT_04_PIN_03	IO port 4 pin 3.
IOPORT_PORT_04_PIN_04	

	IO port 4 pin 4.
IOPORT_PORT_04_PIN_05	IO port 4 pin 5.
IOPORT_PORT_04_PIN_06	IO port 4 pin 6.
IOPORT_PORT_04_PIN_07	IO port 4 pin 7.
IOPORT_PORT_04_PIN_08	IO port 4 pin 8.
IOPORT_PORT_04_PIN_09	IO port 4 pin 9.
IOPORT_PORT_04_PIN_10	IO port 4 pin 10.
IOPORT_PORT_04_PIN_11	IO port 4 pin 11.
IOPORT_PORT_04_PIN_12	IO port 4 pin 12.
IOPORT_PORT_04_PIN_13	IO port 4 pin 13.
IOPORT_PORT_04_PIN_14	IO port 4 pin 14.
IOPORT_PORT_04_PIN_15	IO port 4 pin 15.
IOPORT_PORT_05_PIN_00	IO port 5 pin 0.
IOPORT_PORT_05_PIN_01	IO port 5 pin 1.
IOPORT_PORT_05_PIN_02	IO port 5 pin 2.
IOPORT_PORT_05_PIN_03	IO port 5 pin 3.
IOPORT_PORT_05_PIN_04	IO port 5 pin 4.
IOPORT_PORT_05_PIN_05	IO port 5 pin 5.
IOPORT_PORT_05_PIN_06	IO port 5 pin 6.
IOPORT_PORT_05_PIN_07	IO port 5 pin 7.
IOPORT_PORT_05_PIN_08	IO port 5 pin 8.
IOPORT_PORT_05_PIN_09	IO port 5 pin 9.
IOPORT_PORT_05_PIN_10	IO port 5 pin 10.
IOPORT_PORT_05_PIN_11	IO port 5 pin 11.
IOPORT_PORT_05_PIN_12	

	IO port 5 pin 12.
IOPORT_PORT_05_PIN_13	IO port 5 pin 13.
IOPORT_PORT_05_PIN_14	IO port 5 pin 14.
IOPORT_PORT_05_PIN_15	IO port 5 pin 15.
IOPORT_PORT_06_PIN_00	IO port 6 pin 0.
IOPORT_PORT_06_PIN_01	IO port 6 pin 1.
IOPORT_PORT_06_PIN_02	IO port 6 pin 2.
IOPORT_PORT_06_PIN_03	IO port 6 pin 3.
IOPORT_PORT_06_PIN_04	IO port 6 pin 4.
IOPORT_PORT_06_PIN_05	IO port 6 pin 5.
IOPORT_PORT_06_PIN_06	IO port 6 pin 6.
IOPORT_PORT_06_PIN_07	IO port 6 pin 7.
IOPORT_PORT_06_PIN_08	IO port 6 pin 8.
IOPORT_PORT_06_PIN_09	IO port 6 pin 9.
IOPORT_PORT_06_PIN_10	IO port 6 pin 10.
IOPORT_PORT_06_PIN_11	IO port 6 pin 11.
IOPORT_PORT_06_PIN_12	IO port 6 pin 12.
IOPORT_PORT_06_PIN_13	IO port 6 pin 13.
IOPORT_PORT_06_PIN_14	IO port 6 pin 14.
IOPORT_PORT_06_PIN_15	IO port 6 pin 15.
IOPORT_PORT_07_PIN_00	IO port 7 pin 0.
IOPORT_PORT_07_PIN_01	IO port 7 pin 1.
IOPORT_PORT_07_PIN_02	IO port 7 pin 2.
IOPORT_PORT_07_PIN_03	IO port 7 pin 3.
IOPORT_PORT_07_PIN_04	

	IO port 7 pin 4.
IOPORT_PORT_07_PIN_05	IO port 7 pin 5.
IOPORT_PORT_07_PIN_06	IO port 7 pin 6.
IOPORT_PORT_07_PIN_07	IO port 7 pin 7.
IOPORT_PORT_07_PIN_08	IO port 7 pin 8.
IOPORT_PORT_07_PIN_09	IO port 7 pin 9.
IOPORT_PORT_07_PIN_10	IO port 7 pin 10.
IOPORT_PORT_07_PIN_11	IO port 7 pin 11.
IOPORT_PORT_07_PIN_12	IO port 7 pin 12.
IOPORT_PORT_07_PIN_13	IO port 7 pin 13.
IOPORT_PORT_07_PIN_14	IO port 7 pin 14.
IOPORT_PORT_07_PIN_15	IO port 7 pin 15.
IOPORT_PORT_08_PIN_00	IO port 8 pin 0.
IOPORT_PORT_08_PIN_01	IO port 8 pin 1.
IOPORT_PORT_08_PIN_02	IO port 8 pin 2.
IOPORT_PORT_08_PIN_03	IO port 8 pin 3.
IOPORT_PORT_08_PIN_04	IO port 8 pin 4.
IOPORT_PORT_08_PIN_05	IO port 8 pin 5.
IOPORT_PORT_08_PIN_06	IO port 8 pin 6.
IOPORT_PORT_08_PIN_07	IO port 8 pin 7.
IOPORT_PORT_08_PIN_08	IO port 8 pin 8.
IOPORT_PORT_08_PIN_09	IO port 8 pin 9.
IOPORT_PORT_08_PIN_10	IO port 8 pin 10.
IOPORT_PORT_08_PIN_11	IO port 8 pin 11.
IOPORT_PORT_08_PIN_12	

	IO port 8 pin 12.
IOPORT_PORT_08_PIN_13	IO port 8 pin 13.
IOPORT_PORT_08_PIN_14	IO port 8 pin 14.
IOPORT_PORT_08_PIN_15	IO port 8 pin 15.
IOPORT_PORT_09_PIN_00	IO port 9 pin 0.
IOPORT_PORT_09_PIN_01	IO port 9 pin 1.
IOPORT_PORT_09_PIN_02	IO port 9 pin 2.
IOPORT_PORT_09_PIN_03	IO port 9 pin 3.
IOPORT_PORT_09_PIN_04	IO port 9 pin 4.
IOPORT_PORT_09_PIN_05	IO port 9 pin 5.
IOPORT_PORT_09_PIN_06	IO port 9 pin 6.
IOPORT_PORT_09_PIN_07	IO port 9 pin 7.
IOPORT_PORT_09_PIN_08	IO port 9 pin 8.
IOPORT_PORT_09_PIN_09	IO port 9 pin 9.
IOPORT_PORT_09_PIN_10	IO port 9 pin 10.
IOPORT_PORT_09_PIN_11	IO port 9 pin 11.
IOPORT_PORT_09_PIN_12	IO port 9 pin 12.
IOPORT_PORT_09_PIN_13	IO port 9 pin 13.
IOPORT_PORT_09_PIN_14	IO port 9 pin 14.
IOPORT_PORT_09_PIN_15	IO port 9 pin 15.
IOPORT_PORT_10_PIN_00	IO port 10 pin 0.
IOPORT_PORT_10_PIN_01	IO port 10 pin 1.
IOPORT_PORT_10_PIN_02	IO port 10 pin 2.
IOPORT_PORT_10_PIN_03	IO port 10 pin 3.
IOPORT_PORT_10_PIN_04	

	IO port 10 pin 4.
IOPORT_PORT_10_PIN_05	IO port 10 pin 5.
IOPORT_PORT_10_PIN_06	IO port 10 pin 6.
IOPORT_PORT_10_PIN_07	IO port 10 pin 7.
IOPORT_PORT_10_PIN_08	IO port 10 pin 8.
IOPORT_PORT_10_PIN_09	IO port 10 pin 9.
IOPORT_PORT_10_PIN_10	IO port 10 pin 10.
IOPORT_PORT_10_PIN_11	IO port 10 pin 11.
IOPORT_PORT_10_PIN_12	IO port 10 pin 12.
IOPORT_PORT_10_PIN_13	IO port 10 pin 13.
IOPORT_PORT_10_PIN_14	IO port 10 pin 14.
IOPORT_PORT_10_PIN_15	IO port 10 pin 15.
IOPORT_PORT_11_PIN_00	IO port 11 pin 0.
IOPORT_PORT_11_PIN_01	IO port 11 pin 1.
IOPORT_PORT_11_PIN_02	IO port 11 pin 2.
IOPORT_PORT_11_PIN_03	IO port 11 pin 3.
IOPORT_PORT_11_PIN_04	IO port 11 pin 4.
IOPORT_PORT_11_PIN_05	IO port 11 pin 5.
IOPORT_PORT_11_PIN_06	IO port 11 pin 6.
IOPORT_PORT_11_PIN_07	IO port 11 pin 7.
IOPORT_PORT_11_PIN_08	IO port 11 pin 8.
IOPORT_PORT_11_PIN_09	IO port 11 pin 9.
IOPORT_PORT_11_PIN_10	IO port 11 pin 10.
IOPORT_PORT_11_PIN_11	IO port 11 pin 11.
IOPORT_PORT_11_PIN_12	

	IO port 11 pin 12.
IOPORT_PORT_11_PIN_13	IO port 11 pin 13.
IOPORT_PORT_11_PIN_14	IO port 11 pin 14.
IOPORT_PORT_11_PIN_15	IO port 11 pin 15.
IOPORT_PORT_12_PIN_00	IO port 12 pin 0.
IOPORT_PORT_12_PIN_01	IO port 12 pin 1.
IOPORT_PORT_12_PIN_02	IO port 12 pin 2.
IOPORT_PORT_12_PIN_03	IO port 12 pin 3.
IOPORT_PORT_12_PIN_04	IO port 12 pin 4.
IOPORT_PORT_12_PIN_05	IO port 12 pin 5.
IOPORT_PORT_12_PIN_06	IO port 12 pin 6.
IOPORT_PORT_12_PIN_07	IO port 12 pin 7.
IOPORT_PORT_12_PIN_08	IO port 12 pin 8.
IOPORT_PORT_12_PIN_09	IO port 12 pin 9.
IOPORT_PORT_12_PIN_10	IO port 12 pin 10.
IOPORT_PORT_12_PIN_11	IO port 12 pin 11.
IOPORT_PORT_12_PIN_12	IO port 12 pin 12.
IOPORT_PORT_12_PIN_13	IO port 12 pin 13.
IOPORT_PORT_12_PIN_14	IO port 12 pin 14.
IOPORT_PORT_12_PIN_15	IO port 12 pin 15.
IOPORT_PORT_13_PIN_00	IO port 13 pin 0.
IOPORT_PORT_13_PIN_01	IO port 13 pin 1.
IOPORT_PORT_13_PIN_02	IO port 13 pin 2.
IOPORT_PORT_13_PIN_03	IO port 13 pin 3.
IOPORT_PORT_13_PIN_04	

	IO port 13 pin 4.
IOPORT_PORT_13_PIN_05	IO port 13 pin 5.
IOPORT_PORT_13_PIN_06	IO port 13 pin 6.
IOPORT_PORT_13_PIN_07	IO port 13 pin 7.
IOPORT_PORT_13_PIN_08	IO port 13 pin 8.
IOPORT_PORT_13_PIN_09	IO port 13 pin 9.
IOPORT_PORT_13_PIN_10	IO port 13 pin 10.
IOPORT_PORT_13_PIN_11	IO port 13 pin 11.
IOPORT_PORT_13_PIN_12	IO port 13 pin 12.
IOPORT_PORT_13_PIN_13	IO port 13 pin 13.
IOPORT_PORT_13_PIN_14	IO port 13 pin 14.
IOPORT_PORT_13_PIN_15	IO port 13 pin 15.
IOPORT_PORT_14_PIN_00	IO port 14 pin 0.
IOPORT_PORT_14_PIN_01	IO port 14 pin 1.
IOPORT_PORT_14_PIN_02	IO port 14 pin 2.
IOPORT_PORT_14_PIN_03	IO port 14 pin 3.
IOPORT_PORT_14_PIN_04	IO port 14 pin 4.
IOPORT_PORT_14_PIN_05	IO port 14 pin 5.
IOPORT_PORT_14_PIN_06	IO port 14 pin 6.
IOPORT_PORT_14_PIN_07	IO port 14 pin 7.
IOPORT_PORT_14_PIN_08	IO port 14 pin 8.
IOPORT_PORT_14_PIN_09	IO port 14 pin 9.
IOPORT_PORT_14_PIN_10	IO port 14 pin 10.
IOPORT_PORT_14_PIN_11	IO port 14 pin 11.
IOPORT_PORT_14_PIN_12	

	IO port 14 pin 12.
IOPORT_PORT_14_PIN_13	IO port 14 pin 13.
IOPORT_PORT_14_PIN_14	IO port 14 pin 14.
IOPORT_PORT_14_PIN_15	IO port 14 pin 15.

Function Documentation

◆ R_IOPORT_Open()

fsp_err_t R_IOPORT_Open (ioport_ctrl_t *const *p_ctrl*, const ioport_cfg_t * *p_cfg*)

Initializes internal driver data, then calls pin configuration function to configure pins.

Return values

FSP_SUCCESS	Pin configuration data written to PFS register(s)
FSP_ERR_ASSERTION	NULL pointer
FSP_ERR_ALREADY_OPEN	Module is already open.

◆ R_IOPORT_Close()

fsp_err_t R_IOPORT_Close (ioport_ctrl_t *const *p_ctrl*)

Resets IOPORT registers. Implements ioport_api_t::close

Return values

FSP_SUCCESS	The IOPORT was successfully uninitialized
FSP_ERR_ASSERTION	p_ctrl was NULL
FSP_ERR_NOT_OPEN	The module has not been opened

◆ R_IOPORT_PinsCfg()

fsp_err_t R_IOPORT_PinsCfg (ioport_ctrl_t *const *p_ctrl*, const ioport_cfg_t * *p_cfg*)

Configures the functions of multiple pins by loading configuration data into pin PFS registers. Implements ioport_api_t::pinsCfg.

This function initializes the supplied list of PmnPFS registers with the supplied values. This data can be generated by the Pins tab of the RA Configuration editor or manually by the developer. Different pin configurations can be loaded for different situations such as low power modes and testing.

Return values

FSP_SUCCESS	Pin configuration data written to PFS register(s)
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_ASSERTION	NULL pointer

◆ R_IOPORT_PinCfg()

fsp_err_t R_IOPORT_PinCfg (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, uint32_t *cfg*)

Configures the settings of a pin. Implements ioport_api_t::pinCfg.

Return values

FSP_SUCCESS	Pin configured
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_ASSERTION	NULL pointer

Note

This function is re-entrant for different pins. This function will change the configuration of the pin with the new configuration. For example it is not possible with this function to change the drive strength of a pin while leaving all the other pin settings unchanged. To achieve this the original settings with the required change will need to be written using this function.

◆ R_IOPORT_PinEventInputRead()

fsp_err_t R_IOPORT_PinEventInputRead (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t * *p_pin_event*)

Reads the value of the event input data of a specific pin. Implements ioport_api_t::pinEventInputRead.

The pin event data is captured in response to a trigger from the ELC. This function enables this data to be read. Using the event system allows the captured data to be stored when it occurs and then read back at a later time.

Return values

FSP_SUCCESS	Pin read
FSP_ERR_ASSERTION	NULL pointer
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_INVALID_ARGUMENT	Port is not valid ELC PORT.

Note

This function is re-entrant.

◆ R_IOPORT_PinEventOutputWrite()

fsp_err_t R_IOPORT_PinEventOutputWrite (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t *pin_value*)

This function writes the event output data value to a pin. Implements ioport_api_t::pinEventOutputWrite.

Using the event system enables a pin state to be stored by this function in advance of being output on the pin. The output to the pin will occur when the ELC event occurs.

Return values

FSP_SUCCESS	Pin event data written
FSP_ERR_INVALID_ARGUMENT	Port or Pin or value not valid
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_ASSERTION	NULL pointer

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PinRead()

fsp_err_t R_IOPORT_PinRead (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t * *p_pin_value*)

Reads the level on a pin. Implements ioport_api_t::pinRead.

Return values

FSP_SUCCESS	Pin read
FSP_ERR_ASSERTION	NULL pointer
FSP_ERR_NOT_OPEN	The module has not been opened

Note

This function is re-entrant for different pins.

◆ R_IOPORT_PinWrite()

fsp_err_t R_IOPORT_PinWrite (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_pin_t *pin*, bsp_io_level_t *level*)

Sets a pin's output either high or low. Implements ioport_api_t::pinWrite.

Return values

FSP_SUCCESS	Pin written to
FSP_ERR_INVALID_ARGUMENT	The pin and/or level not valid
FSP_ERR_NOT_OPEN	The module has not been opene
FSP_ERR_ASSERTION	NULL pointerd

Note

This function is re-entrant for different pins. This function makes use of the PCNTR3 register to atomically modify the level on the specified pin on a port.

◆ R_IOPORT_PortDirectionSet()

fsp_err_t R_IOPORT_PortDirectionSet (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t *direction_values*, ioport_size_t *mask*)

Sets the direction of individual pins on a port. Implements ioport_api_t::portDirectionSet().

Multiple pins on a port can be set to inputs or outputs at once. Each bit in the mask parameter corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. If a bit is set to 1 then the corresponding pin will be changed to an input or an output as specified by the direction values. If a mask bit is set to 0 then the direction of the pin will not be changed.

Return values

FSP_SUCCESS	Port direction updated
FSP_ERR_INVALID_ARGUMENT	The port and/or mask not valid
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_ASSERTION	NULL pointer

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PortEventInputRead()

fsp_err_t R_IOPORT_PortEventInputRead (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t * *p_event_data*)

Reads the value of the event input data. Implements ioport_api_t::portEventInputRead().

The event input data for the port will be read. Each bit in the returned value corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on.

The port event data is captured in response to a trigger from the ELC. This function enables this data to be read. Using the event system allows the captured data to be stored when it occurs and then read back at a later time.

Return values

FSP_SUCCESS	Port read
FSP_ERR_INVALID_ARGUMENT	Port not a valid ELC port
FSP_ERR_ASSERTION	NULL pointer
FSP_ERR_NOT_OPEN	The module has not been opened

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PortEventOutputWrite()

fsp_err_t R_IOPORT_PortEventOutputWrite (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t *event_data*, ioport_size_t *mask_value*)

This function writes the set and reset event output data for a port. Implements ioport_api_t::portEventOutputWrite.

Using the event system enables a port state to be stored by this function in advance of being output on the port. The output to the port will occur when the ELC event occurs.

The input value will be written to the specified port when an ELC event configured for that port occurs. Each bit in the value parameter corresponds to a bit on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. Each bit in the mask parameter corresponds to a pin on the port.

Return values

FSP_SUCCESS	Port event data written
FSP_ERR_INVALID_ARGUMENT	Port or Mask not valid
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_ASSERTION	NULL pointer

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PortRead()

fsp_err_t R_IOPORT_PortRead (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t * *p_port_value*)

Reads the value on an IO port. Implements ioport_api_t::portRead.

The specified port will be read, and the levels for all the pins will be returned. Each bit in the returned value corresponds to a pin on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on.

Return values

FSP_SUCCESS	Port read
FSP_ERR_ASSERTION	NULL pointer
FSP_ERR_NOT_OPEN	The module has not been opened

Note

This function is re-entrant for different ports.

◆ R_IOPORT_PortWrite()

fsp_err_t R_IOPORT_PortWrite (ioport_ctrl_t *const *p_ctrl*, bsp_io_port_t *port*, ioport_size_t *value*, ioport_size_t *mask*)

Writes to multiple pins on a port. Implements ioport_api_t::portWrite.

The input value will be written to the specified port. Each bit in the value parameter corresponds to a bit on the port. For example, bit 7 corresponds to pin 7, bit 6 to pin 6, and so on. Each bit in the mask parameter corresponds to a pin on the port.

Only the bits with the corresponding bit in the mask value set will be updated. For example, value = 0xFFFF, mask = 0x0003 results in only bits 0 and 1 being updated.

Return values

FSP_SUCCESS	Port written to
FSP_ERR_INVALID_ARGUMENT	The port and/or mask not valid
FSP_ERR_NOT_OPEN	The module has not been opened
FSP_ERR_ASSERTION	NULL pointerd

Note

This function is re-entrant for different ports. This function makes use of the PCNTR3 register to atomically modify the levels on the specified pins on a port.

4.2.37 Independent Watchdog Timer (r_iwdt)

Modules

Functions

fsp_err_t	R_IWDT_Refresh (wdt_ctrl_t *const p_api_ctrl)
fsp_err_t	R_IWDT_Open (wdt_ctrl_t *const p_api_ctrl, wdt_cfg_t const *const p_cfg)
fsp_err_t	R_IWDT_StatusClear (wdt_ctrl_t *const p_api_ctrl, const wdt_status_t status)
fsp_err_t	R_IWDT_StatusGet (wdt_ctrl_t *const p_api_ctrl, wdt_status_t *const p_status)
fsp_err_t	R_IWDT_CounterGet (wdt_ctrl_t *const p_api_ctrl, uint32_t *const p_count)
fsp_err_t	R_IWDT_TimeoutGet (wdt_ctrl_t *const p_api_ctrl, wdt_timeout_values_t *const p_timeout)

fsp_err_t	R_IWDT_CallbackSet (wdt_ctrl_t *const p_ctrl, void(*p_callback)(wdt_callback_args_t *), void const *const p_context, wdt_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the IWDT peripheral on RA MCUs. This module implements the WDT Interface.

Overview

The independent watchdog timer is used to recover from unexpected errors in an application. The timer must be refreshed periodically in the permitted count window by the application. If the count is allowed to underflow or refresh occurs outside of the valid refresh period, the IWDT resets the device or generates an NMI.

Features

The IWDT HAL module has the following key features:

- When the IWDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:
  - Resetting of the device
  - Generation of an NMI
- The IWDT begins counting at reset.

Selecting a Watchdog

RA MCUs have two watchdog peripherals: the watchdog timer (WDT) and the independent watchdog timer (IWDT). When selecting between them, consider these factors:

	WDT	IWDT
Start Mode	The WDT can be started from the application (register start mode) or configured by hardware to start automatically (auto start mode).	The IWDT can only be configured by hardware to start automatically.
Clock Source	The WDT runs off a peripheral clock.	The IWDT has its own clock source which improves safety.

Configuration

Build Time Configurations for r_iwdt

The following build time configurations are defined in fsp_cfg/r_iwdt_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled	Default (BSP)	If selected code for parameter checking is

	• Disabled		included in the build.

Configurations for Monitoring > Independent Watchdog (r_iwdt)

This module can be added to the Stacks tab via New Stack > Monitoring > Independent Watchdog (r_iwdt). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_wdt0	Module name.
NMI callback	Name must be a valid C symbol	NULL	A user callback function can be provided here. If this callback function is provided, it is called from the interrupt service routine (ISR) when the watchdog triggers.

Note

*The IWDT has additional configurable settings in the OFS0 register in the **BSP** tab properties window. These settings include the following:*

- *Start Mode*
- *Timeout Period*
- *Dedicated Clock Frequency Divisor*
- *Window End Position*
- *Window Start Position*
- *Reset Interrupt Request Select*
- *Stop Control*

Review the OFS0 properties window to see additional details.

Clock Configuration

The IWDT clock is based on the IWDTCLK frequency. You can set the IWDTCLK frequency divider using the **BSP** tab of the RA Configuration editor.

Pin Configuration

This module does not use I/O pins.

Usage Notes

NMI Interrupt

The independent watchdog timer uses the NMI, which is enabled by default. No special configuration is required. When the NMI is triggered, the callback function registered during open is called.

Period Calculation

The IWDT operates from IWDTCLK. With a IWDTCLK of 15000 Hz, the maximum time from the last refresh to device reset or NMI generation will be just below 35 seconds as detailed below.

IWDTCLK = 15000 Hz
Clock division ratio = IWDTCLK / 256
Timeout period = 2048 cycles
WDT clock frequency = 15000 Hz / 256 = 58.59 Hz
Cycle time = 1 / 58.59 Hz = 17.067 ms
Timeout = 17.067 ms x 2048 cycles = 34.95 seconds

Limitations

Developers should be aware of the following limitations when using the IWDT:

- When using a J-Link debugger the IWDT counter does not count and therefore will not reset the device or generate an NMI. To enable the watchdog to count and generate a reset or NMI while debugging, add this line of code in the application:

```
/* (Optional) Enable the IWDT to count and generate NMI or reset when the
 * debugger is connected. */
   R_DEBUG->DBGSTOPCR_b.DBGSTOP_IWDT = 0;
```

- If the IWDT is configured to stop the counter in low power mode, then your application must restart the watchdog by calling R_IWDT_Refresh() after the MCU wakes from low power mode.

Examples

IWDT Basic Example

This is a basic example of minimal use of the IWDT in an application.

```
void iwdt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* In auto start mode, the IWDT starts counting immediately when the MCU is powered
on. */
 /* Initializes the module. */
    err = R_IWDT_Open(&g_iwdt0_ctrl, &g_iwdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Application work here. */
 /* Refresh before the counter underflows to prevent reset or NMI based on the
setting. */
       (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
```

```
    }
}
```

IWDT Advanced Example

This example demonstrates using a start window and gives an example callback to handle an NMI generated by an underflow or refresh error.

```
#define IWDT_TIMEOUT_COUNTS (2048U)
#define IWDT_MAX_COUNTER (IWDT_TIMEOUT_COUNTS - 1U)
#define IWDT_START_WINDOW_75 ((IWDT_MAX_COUNTER * 3) / 4)
/* Example callback called when a watchdog NMI occurs. */
void iwdt_callback (wdt_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) Determine the source of the NMI. */
 wdt_status_t status = WDT_STATUS_NO_ERROR;
    err = R_IWDT_StatusGet(&g_iwdt0_ctrl, &status);
    assert(FSP_SUCCESS == err);
 /* (Optional) Log source of NMI and any other debug information. */
 /* (Optional) Clear the error flags. */
    err = R_IWDT_StatusClear(&g_iwdt0_ctrl, status);
    assert(FSP_SUCCESS == err);
 /* (Optional) Issue a software reset to reset the MCU. */
    __NVIC_SystemReset();
}
void iwdt_advanced_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) Enable the IWDT to count and generate NMI or reset when the
  * debugger is connected. */
    R_DEBUG->DBGSTOPCR_b.DBGSTOP_IWDT = 0;
 /* (Optional) Check if the IWDTRF flag is set to know if the system is
  * recovering from a IWDT reset. */
```

```
if (R_SYSTEM->RSTSR1_b.IWDTRF)
   {
/* Clear the flag. */
      R_SYSTEM->RSTSR1 = 0U;
   }
/* Open the module. */
   err = R_IWDT_Open(&g_iwdt0_ctrl, &g_iwdt0_cfg);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Initialize other application code. */
/* Do not call R_IWDT_Refresh() in auto start mode unless the
 * counter is in the acceptable refresh window. */
   (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
while (true)
   {
/* Application work here. */
/* (Optional) If there is a chance the application takes less time than
 * the start window, verify the IWDT counter is past the start window
 * before refreshing the IWDT. */
      uint32_t iwdt_counter = 0U;
do
     {
/* Read the current IWDT counter value. */
           err = R_IWDT_CounterGet(&g_iwdt0_ctrl, &iwdt_counter);
     assert(FSP_SUCCESS == err);
     } while (iwdt_counter >= IWDT_START_WINDOW_75);
/* Refresh before the counter underflows to prevent reset or NMI. */
      (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
   }
}
```

Data Structures

struct	iwdt_instance_ctrl_t

Data Structure Documentation

◆ iwdt_instance_ctrl_t

struct iwdt_instance_ctrl_t	
IWDT control block. DO NOT INITIALIZE. Initialization occurs when wdt_api_t::open is called.	
Data Fields	
uint32_t	wdt_open
	Indicates whether the open() API has been successfully called.
void const *	p_context
	Placeholder for user data. Passed to the user callback in wdt_callback_args_t.
R_IWDT_Type *	p_reg
	Pointer to register base address.
void(*	p_callback)(wdt_callback_args_t *p_args)
	Callback provided when a WDT NMI ISR occurs.

Function Documentation

◆ R_IWDT_Refresh()

fsp_err_t R_IWDT_Refresh (wdt_ctrl_t *const *p_api_ctrl*)

Refresh the Independent Watchdog Timer. If the refresh fails due to being performed outside of the permitted refresh period the device will either reset or trigger an NMI ISR to run.

Example:

```
/* Refresh before the counter underflows to prevent reset or NMI based on the
setting. */
        (void) R_IWDT_Refresh(&g_iwdt0_ctrl);
```

Return values

FSP_SUCCESS	IWDT successfully refreshed.
FSP_ERR_ASSERTION	One or more parameters are NULL pointers.
FSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.

◆ R_IWDT_Open()

fsp_err_t R_IWDT_Open (wdt_ctrl_t *const *p_api_ctrl*, wdt_cfg_t const *const *p_cfg*)

Register the IWDT NMI callback.

Example:

```
/* Initializes the module. */
    err = R_IWDT_Open(&g_iwdt0_ctrl, &g_iwdt0_cfg);
```

Return values

FSP_SUCCESS	IWDT successfully configured.
FSP_ERR_ASSERTION	Null Pointer.
FSP_ERR_NOT_ENABLED	An attempt to open the IWDT when the OFS0 register is not configured for auto-start mode.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_STATE	The security state of the NMI and the module do not match.

◆ R_IWDT_StatusClear()

fsp_err_t R_IWDT_StatusClear (wdt_ctrl_t *const *p_api_ctrl*, const wdt_status_t *status*)

Clear the IWDT status and error flags. Implements wdt_api_t::statusClear.

Example:

```
/* (Optional) Clear the error flags. */
   err = R_IWDT_StatusClear(&g_iwdt0_ctrl, status);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	IWDT flag(s) successfully cleared.
FSP_ERR_ASSERTION	Null pointer as a parameter.
FSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.

◆ R_IWDT_StatusGet()

fsp_err_t R_IWDT_StatusGet (wdt_ctrl_t *const *p_api_ctrl*, wdt_status_t *const *p_status*)

Read the IWDT status flags. When the IWDT is configured to output a reset on underflow or refresh error reading the status and error flags can be read after reset to establish if the IWDT caused the reset. Reading the status and error flags in NMI output mode indicates whether the IWDT generated the NMI interrupt.

Indicates both status and error conditions.

Example:

```
/* (Optional) Determine the source of the NMI. */
wdt_status_t status = WDT_STATUS_NO_ERROR;
   err = R_IWDT_StatusGet(&g_iwdt0_ctrl, &status);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	IWDT status successfully read.
FSP_ERR_ASSERTION	Null pointer as a parameter.
FSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.

◆ R_IWDT_CounterGet()

fsp_err_t R_IWDT_CounterGet (wdt_ctrl_t *const *p_api_ctrl*, uint32_t *const *p_count*)

Read the current count value of the IWDT. Implements wdt_api_t::counterGet.

Example:

```
/* Read the current IWDT counter value. */
          err = R_IWDT_CounterGet(&g_iwdt0_ctrl, &iwdt_counter);
     assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	IWDT current count successfully read.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.

◆ R_IWDT_TimeoutGet()

fsp_err_t R_IWDT_TimeoutGet (wdt_ctrl_t *const *p_api_ctrl*, wdt_timeout_values_t *const *p_timeout*)

Read timeout information for the watchdog timer. Implements wdt_api_t::timeoutGet.

Return values

FSP_SUCCESS	IWDT timeout information retrieved successfully.
FSP_ERR_ASSERTION	One or more parameters are NULL pointers.
FSP_ERR_NOT_OPEN	The driver has not been opened. Perform R_IWDT_Open() first.

◆ R_IWDT_CallbackSet()

fsp_err_t R_IWDT_CallbackSet (wdt_ctrl_t *const *p_ctrl*, void(*)(wdt_callback_args_t *) *p_callback*, void const *const *p_context*, wdt_callback_args_t *const *p_callback_memory*)
Updates the user callback and has option of providing memory for callback structure. Implements wdt_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.38 JPEG Codec (r_jpeg)

Modules

Functions

fsp_err_t	R_JPEG_Open (jpeg_ctrl_t *const p_api_ctrl, jpeg_cfg_t const *const p_cfg)
fsp_err_t	R_JPEG_OutputBufferSet (jpeg_ctrl_t *p_api_ctrl, void *output_buffer, uint32_t output_buffer_size)
fsp_err_t	R_JPEG_InputBufferSet (jpeg_ctrl_t *constp_api_ctrl, void *p_data_buffer, uint32_t data_buffer_size)
fsp_err_t	R_JPEG_StatusGet (jpeg_ctrl_t *p_api_ctrl, jpeg_status_t *p_status)
fsp_err_t	R_JPEG_Close (jpeg_ctrl_t *p_api_ctrl)
fsp_err_t	R_JPEG_EncodeImageSizeSet (jpeg_ctrl_t *const p_api_ctrl, jpeg_encode_image_size_t *p_image_size)
fsp_err_t	R_JPEG_DecodeLinesDecodedGet (jpeg_ctrl_t *const p_api_ctrl, uint32_t *const p_lines)
fsp_err_t	R_JPEG_DecodeHorizontalStrideSet (jpeg_ctrl_t *p_api_ctrl, uint32_t horizontal_stride)

fsp_err_t	R_JPEG_DecodeImageSizeGet (jpeg_ctrl_t *p_api_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)
fsp_err_t	R_JPEG_DecodeImageSubsampleSet (jpeg_ctrl_t *const p_api_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)
fsp_err_t	R_JPEG_DecodePixelFormatGet (jpeg_ctrl_t *p_api_ctrl, jpeg_color_space_t *p_color_space)
fsp_err_t	R_JPEG_ModeSet (jpeg_ctrl_t *const p_api_ctrl, jpeg_mode_t mode)

Detailed Description

Driver for the JPEG peripheral on RA MCUs. This module implements the JPEG Codec Interface.

Overview

The JPEG Codec is a hardware block providing accelerated JPEG image encode and decode functionality independent of the CPU. Images can optionally be partially processed facilitating streaming applications.

Features

The JPEG Codec provides a number of options useful in a variety of applications:

- Basic encoding and decoding
- Streaming input and/or output
- Decoding JPEGs of unknown size
- Shrink (sub-sample) an image during the decoding process
- Rearrange input and output byte order (byte, word and/or longword swap)
- JPEG error detection

The specifications for the codec are as follows:

Feature	Options
Decompression input formats	Baseline JPEG Y'CbCr 4:4:4, 4:2:2, 4:2:0 and 4:1:1
Decompression output formats	ARGB8888, RGB565
Compression input formats	Raw Y'CbCr 4:2:2 only
Compression output formats	Baseline JPEG Y'CbCr 4:2:2 only
Byte reordering	Byte, halfword and/or word swapping on input and output
Interrupt sources	Image size acquired, input/output data pause, decode complete, error

Compatible image sizes	See Minimum Coded Unit (MCU) below

Configuration

Build Time Configurations for r_jpeg

The following build time configurations are defined in fsp_cfg/r_jpeg_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected, code for parameter checking is included in the build.
Decode Support	• Enabled • Disabled	Enabled	If selected, code for decoding JPEG images is included in the build.
Encode Support	• Enabled • Disabled	Disabled	If selected, code for encoding JPEG images is included in the build.

Configurations for Graphics > JPEG Codec (r_jpeg)

This module can be added to the Stacks tab via New Stack > Graphics > JPEG Codec (r_jpeg).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_jpeg0	Module name.
General > Default mode	• Decode • Encode	Decode	Set the mode to use when calling R_JPEG_Open. This parameter is only used when both Encode and Decode support are enabled.
Decode > Input byte order	MCU Specific Options		Select the byte order of the input data for decoding.
Decode > Output byte order	MCU Specific Options		Select the byte order of the output data for decoding.
Decode > Output color format	• ARGB8888 (32-bit) • RGB565 (16-bit)	RGB565 (16-bit)	Select the output pixel format for decode operations.
Decode > Output alpha (ARGB8888 only)	Value must be an 8-bit integer (0-255)	255	Specify the alpha value to apply to each output pixel when ARGB8888 format is chosen.
Decode > Callback	Name must be a valid	NULL	If a callback function is

	C symbol		provided it will be called from the interrupt service routine (ISR) each time a related IRQ triggers.
Encode > Horizontal resolution	Value cannot be greater than 65535 and must be a non-negative integer divisible by 16	480	Horizontal resolution of the raw image (in pixels). This value can be configured at runtime via R_JPEG_ImageSizeSet.
Encode > Vertical resolution	Value cannot be greater than 65535 and must be a non-negative integer divisible by 8	272	Vertical resolution of the raw image. This value can be configured at runtime via R_JPEG_ImageSizeSet.
Encode > Horizontal stride	Value cannot be greater than 65535 and must be a non-negative integer	480	Horizontal stride of the raw image buffer (in pixels). This value can be configured at runtime via R_JPEG_ImageSizeSet.
Encode > Input byte order	MCU Specific Options		Select the byte order of the input data for encoding.
Encode > Output byte order	MCU Specific Options		Select the byte order of the output data for encoding.
Encode > Reset interval	Value cannot be greater than 65535 and must be a non-negative integer	512	Set the number of MCUs between RST markers. A value of 0 will disable DRI and RST marker output.
Encode > Quality factor	Value must be between 1 and 100 and must be an integer	50	Set the quality factor for encoding (1-100). Lower values produce smaller images at the cost of image quality.
Encode > Callback	Name must be a valid C symbol	NULL	If a callback function is provided it will be called from the interrupt service routine (ISR) each time a related IRQ triggers.
Interrupts > Decode Process Interrupt Priority	MCU Specific Options		Select the decompression interrupt priority.
Interrupts > Data	MCU Specific Options		Select the data transfer

Transfer Interrupt Priority	interrupt priority.

Clock Configuration

The peripheral clock for this module is PCLKA. No clocks are provided by this module.

Pin Configuration

This module does not have any input or output pin connections.

Usage Notes

Overview

The JPEG Codec contains both decode and encode hardware. While these two functions are largely independent in configuration only one can be used at a time.

To switch from decode to encode mode (or vice versa) use R_JPEG_ModeSet while the JPEG Codec is idle.

Status

The status value (jpeg_status_t) provided by the callback and by R_JPEG_StatusGet is a bitfield that encompasses all potential status indication conditions. One or more statuses can be set simultaneously.

Decoding Process

JPEG decoding can be performed in several ways depending on the application:

- To perform the simplest decode operation where all dimensions are known:
  - Set the input buffer, stride and output buffer then wait for a callback with status JPEG_STATUS_OPERATION_COMPLETE.
- To pause after decoding the JPEG header (in order to acquire image dimensions and secure an output buffer):
  - Call R_JPEG_InputBufferSet before setting the output buffer and wait for a callback with status JPEG_STATUS_IMAGE_SIZE_READY.
- To decode a partial JPEG image then pause until the next chunk is available:
  - Specify a size smaller than the full JPEG data when calling R_JPEG_InputBufferSet.
- To pause decoding once an output buffer is filled:
  - Specify a size smaller than the full decoded image when calling R_JPEG_OutputBufferSet.

The flowchart below illustrates the steps necessary to handle any decode operation. The statuses given in blue are part of jpeg_status_t with the JPEG_DECODE_STATUS prefix omitted.

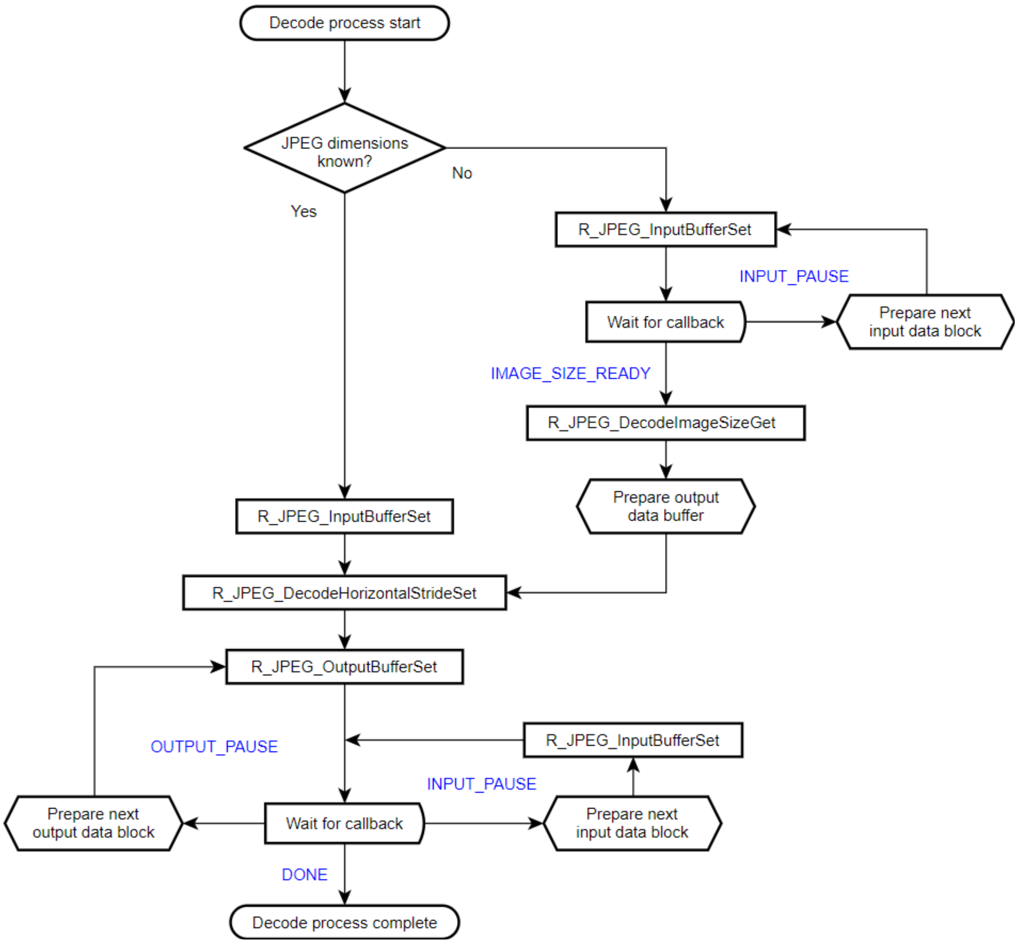


Figure 171: JPEG Decode Operational Flow

Encoding Process

As compared to decoding, encoding is fairly straightforward. The only option available is to stream input data if desired. The flowchart below details the steps needed to compress an image.

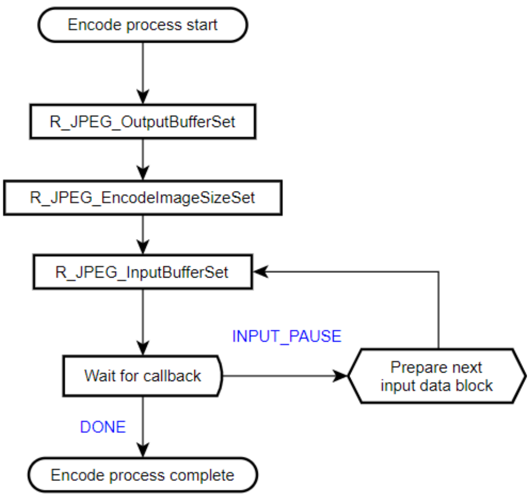


Figure 172: JPEG Encode Operational Flow

Handling Failed Operations

If an encode or decode operation fails or times out while the codec is running, the peripheral must be reset before it is used again. To reset the JPEG Codec simply close and re-open the module by calling R_JPEG_Close followed by R_JPEG_Open.

Limitations

Developers should be aware of the following limitations when using the JPEG API.

Minimum Coded Unit (MCU)

The JPEG Codec can only correctly process images that are an even increment of minimum coded units (MCUs). In other words, depending on the format the width and height of an image to be encoded or decoded must be divisible by the following:

Format	Horizontal	Vertical
Y'CbCr 4:4:4	8 pixels	8 lines
Y'CbCr 4:2:2	16 pixels	8 lines
Y'CbCr 4:1:1	32 pixels	8 lines
Y'CbCr 4:2:0	16 pixels	16 lines

Note

Because encoding is limited to Y'CbCr 4:2:2, raw pixel input data must always be in whole increments of 16x8 pixels.

Encoding Input Format

The encoding unit only supports Y'CbCr 4:2:2 input. Raw RGB888 data can be converted to this format as follows:

```
y =       (0.299000f * r) + (0.587000f * g) + (0.114000f * b);
cb = 128 - (0.168736f * r) - (0.331264f * g) + (0.500000f * b);
cr = 128 + (0.500000f * r) - (0.418688f * g) - (0.081312f * b);
```

While these equations are mathematically simple they do use the floating-point unit. To speed things up we can multiply the coefficients by 256 and divide the sum by 256...

```
y =       ((76.5440f * r) + (150.272f * g) + (29.1840f * b)) / 256;
cb = 128 - ((43.1964f * r) - (84.8036f * g) + (128.000f * b)) / 256;
cr = 128 + ((128.000f * r) - (107.184f * g) - (20.8159f * b)) / 256;
```

...which allows the formulas to be calculated entirely with shifts and addition (coefficients rounded to the nearest integer):

```
    y =       (   (r << 6) + (r << 3) + (r << 2) + r
                 + (g << 7) + (g << 4) + (g << 2) + (g << 1)
                 + (b << 4) + (b << 3) + (b << 2) + b
                ) >> 8;
    cb = 128 - (   (r << 5) + (r << 3) + (r << 1) + r
                 + (g << 6) + (g << 4) + (g << 2) + g
                 - (b << 7)
                ) >> 8;
    cr = 128 + (   (r << 7)
                 - (g << 6) - (g << 5) - (g << 3) - (g << 1) - g
                 - (b << 4) - (b << 2) - b)
                ) >> 8;
```

To compose the final Y'CbCr 4:2:2 data the chroma of every two pixels must be averaged. **In addition, the JPEG Codec expects chrominance values to be in the range -127..127 instead of the standard 1..255.**

```
    cb = (uint8_t) ((int8_t) ((cb0 + cb1 + 1) >> 1) - 128);
    cr = (uint8_t) ((int8_t) ((cr0 + cr1 + 1) >> 1) - 128);
```

Finally, the below equation composes two 4:2:2 output pixels at a time with standard byte order (JPEG_DATA_ORDER_NORMAL):

```
    out = y0 + (cb << 8) + (y1 << 16) + (cr << 24);
```

Note

RGB565 pixels must be upscaled to RGB888 before using the above formulas. Refer to the below example on Y'CbCr Conversion for implementation details.

Examples

Basic Decode Example

This is a basic example showing the minimum code required to initialize the JPEG Codec and decode an image.

```
void jpeg_decode_basic (void)
{
```

```
 fsp_err_t err;
 /* Open JPEG Codec */
    err = R_JPEG_Open(&g_jpeg_ctrl, &g_jpeg_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set input buffer */
    err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, JPEG_PTR, JPEG_SIZE_BYTES);
    assert(FSP_SUCCESS == err);
 /* Set horizontal stride of output buffer */
    err = R_JPEG_DecodeHorizontalStrideSet(&g_jpeg_ctrl, JPEG_HSIZE);
    assert(FSP_SUCCESS == err);
 /* Set output buffer */
    err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, decode_buffer, sizeof(decode_buffer));
    assert(FSP_SUCCESS == err);
 /* Wait for decode completion */
 jpeg_status_t status = (jpeg_status_t) 0;
 while (!(status & (JPEG_STATUS_OPERATION_COMPLETE | JPEG_STATUS_ERROR)))
    {
       err = R_JPEG_StatusGet(&g_jpeg_ctrl, &status);
      assert(FSP_SUCCESS == err);
    }
}
```

Streaming Input/Output Example

In this example JPEG data is read in 512-byte chunks. Decoding is paused when a chunk is read and once the JPEG header is decoded. The image is decoded 16 lines at a time.

Note

> *Streaming is always bypassed when a given buffer's size encompasses the entire input or output image, respectively. Though this example decodes via smaller chunks the input and output data are still contiguous for ease of demonstration. Refer to the comments for further insight as to how to implement streaming with different JPEG/output buffer size combinations.*

```
#define JPEG_INPUT_SIZE_BYTES 512U
/* JPEG Codec status */
static volatile jpeg_status_t g_jpeg_status = JPEG_STATUS_NONE;
/* JPEG event flag */
```

```
static volatile uint8_t jpeg_event = 0;
/* Callback function for JPEG decode interrupts */
void jpeg_decode_callback (jpeg_callback_args_t * p_args)
{
 /* Get JPEG Codec status */
    g_jpeg_status = p_args->status;
 /* Set JPEG flag */
    jpeg_event = 1;
}
/* Simple wait that returns 1 if no event happened within the timeout period */
static uint8_t jpeg_event_wait (void)
{
    uint32_t timeout_timer = JPEG_EVENT_TIMEOUT;
 while (!jpeg_event && --timeout_timer)
    {
 /* Spin here until an event callback or timeout */
    }
    jpeg_event = 0;
 return timeout_timer ? 0 : 1;
}
/* Decode a JPEG image to a buffer using streaming input and output */
void jpeg_decode_streaming (void)
{
    uint8_t    * p_jpeg = (uint8_t *) JPEG_PTR;
 jpeg_status_t status = (jpeg_status_t) 0;
    uint8_t       timeout = 0;
 fsp_err_t    err;
 /* Number of input bytes to read at a time */
    uint32_t input_bytes = JPEG_INPUT_SIZE_BYTES;
 /* Open JPEG unit and start decode */
    err = R_JPEG_Open(&g_jpeg_ctrl, &g_jpeg_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (!(status & JPEG_STATUS_ERROR) && !timeout)
```

```
    {
 /* Set the input buffer to read `input_bytes` bytes at a time */
       err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, p_jpeg, input_bytes);
      assert(FSP_SUCCESS == err);
 /* This delay is required for streaming input mode to function correctly.
  * (Without this delay the JPEG Codec will not correctly locate markers in the file
header.) */
 R_BSP_SoftwareDelay(10, BSP_DELAY_UNITS_MICROSECONDS);
 /* Wait for a callback */
       timeout = jpeg_event_wait();
 /* Get the status from the callback */
       status = g_jpeg_status;
 /* Break if the header has finished decoding */
 if (status & JPEG_STATUS_IMAGE_SIZE_READY)
      {
 break;
      }
 /* Move pointer to next block of input data (if needed) */
       p_jpeg = (uint8_t *) ((uint32_t) p_jpeg + input_bytes);
    }
 /* Get image size */
    uint16_t horizontal;
    uint16_t vertical;
    err = R_JPEG_DecodeImageSizeGet(&g_jpeg_ctrl, &horizontal, &vertical);
    assert(FSP_SUCCESS == err);
 /* Prepare output data buffer here if needed (already allocated in this example) */
    uint8_t * p_output = decode_buffer;
 /* Set horizontal stride */
    err = R_JPEG_DecodeHorizontalStrideSet(&g_jpeg_ctrl, horizontal);
    assert(FSP_SUCCESS == err);
 /* Calculate the number of bytes that will fit in the buffer (16 lines in this
example) */
    uint32_t output_size = horizontal * 16U * 4U;
 /* Start decoding by setting the output buffer */
```

```
    err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, p_output, output_size);
    assert(FSP_SUCCESS == err);
while (!(status & JPEG_STATUS_ERROR) && !timeout)
    {
/* Wait for a callback */
        timeout = jpeg_event_wait();
/* Get the status from the callback */
        status = g_jpeg_status;
/* Break if decoding is complete */
if (status & JPEG_STATUS_OPERATION_COMPLETE)
     {
break;
     }
if (status & JPEG_STATUS_OUTPUT_PAUSE)
     {
/* Draw the JPEG work buffer to the framebuffer here (if needed) */
/* Move pointer to next block of output data (if needed) */
           p_output += output_size;
/* Set the output buffer to the next 16-line block */
     err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, p_output, output_size);
     assert(FSP_SUCCESS == err);
     }
if (status & JPEG_STATUS_INPUT_PAUSE)
     {
/* Get next block of input data */
        p_jpeg = (uint8_t *) ((uint32_t) p_jpeg + input_bytes);
/* Set the new input buffer pointer */
        err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, p_jpeg, input_bytes);
     assert(FSP_SUCCESS == err);
     }
    }
/* Close driver to allow encode operations if needed */
    err = R_JPEG_Close(&g_jpeg_ctrl);
    assert(FSP_SUCCESS == err);
```

```
}
```

Encode Example

This is a basic example showing the minimum code required to initialize the JPEG Codec and encode an image.

Note

This example assumes image dimensions are provided in the configuration. If this is not the case, R_JPEG_EncodeImageSizeSet must be used to set the size before calling R_JPEG_InputBufferSet.

```
void jpeg_encode_basic (void)
{
 fsp_err_t err;
 /* Open JPEG Codec */
    err = R_JPEG_Open(&g_jpeg_ctrl, &g_jpeg_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set output buffer */
    err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, jpeg_buffer, sizeof(jpeg_buffer));
    assert(FSP_SUCCESS == err);
 /* Set input buffer */
    err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, RAW_YCBCR_IMAGE_PTR, IMAGE_SIZE_BYTES);
    assert(FSP_SUCCESS == err);
 /* Wait for decode completion */
 jpeg_status_t status = (jpeg_status_t) 0;
 while (!(status & JPEG_STATUS_OPERATION_COMPLETE))
    {
       err = R_JPEG_StatusGet(&g_jpeg_ctrl, &status);
      assert(FSP_SUCCESS == err);
    }
}
```

Streaming Encode Example

In this example the raw input data is provided in smaller chunks. This can help significantly reduce buffer size and improve throughput when streaming in raw data from an outside source.

```
/* Callback function for JPEG encode interrupts */
void jpeg_encode_callback (jpeg_callback_args_t * p_args)
{
 /* Get JPEG Codec status */
    g_jpeg_status = p_args->status;
 /* Set JPEG flag */
    jpeg_event = 1;
}
void jpeg_encode_streaming (void)
{
    uint8_t   timeout = 0;
    uint8_t * p_chunk = (uint8_t *) RAW_YCBCR_IMAGE_PTR;
 fsp_err_t err;
 /* Open JPEG Codec */
    err = R_JPEG_Open(&g_jpeg_ctrl, &g_jpeg_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set output buffer */
    err = R_JPEG_OutputBufferSet(&g_jpeg_ctrl, jpeg_buffer, sizeof(jpeg_buffer));
    assert(FSP_SUCCESS == err);
 /* Set the image size */
 jpeg_encode_image_size_t image_size;
    image_size.horizontal_resolution    = X_RESOLUTION;
    image_size.vertical_resolution      = Y_RESOLUTION;
    image_size.horizontal_stride_pixels = H_STRIDE;
    err = R_JPEG_EncodeImageSizeSet(&g_jpeg_ctrl, &image_size);
    assert(FSP_SUCCESS == err);
 /* Calculate the size of the input data chunk (16 lines in this example) */
    uint32_t chunk_size = H_STRIDE * 16U * YCBCR_BYTES_PER_PIXEL;
 while (!timeout)
    {
 /* Set the input buffer */
       err = R_JPEG_InputBufferSet(&g_jpeg_ctrl, p_chunk, chunk_size);
      assert(FSP_SUCCESS == err);
```

```
 /* Wait for a callback */
       timeout = jpeg_event_wait();
 if (g_jpeg_status & JPEG_STATUS_OPERATION_COMPLETE)
      {
 /* Encode complete */
 break;
      }
 if (g_jpeg_status & JPEG_STATUS_INPUT_PAUSE)
      {
 /* Load next block of input data here (if needed) */
            p_chunk += chunk_size;
      }
    }
}
```

Y'CbCr Conversion

The below function is provided as a reference for how to convert RGB values to Y'CbCr for use with the JPEG Codec.

Note

This function is only partially optimized for clarity. Further appllication-specific size- or speed-based optimizations should be considered when implementing in an actual project.

```
#define RGB565_G_MASK 0x07E0
#define RGB565_B_MASK 0x001F
#define C_0 128
typedef enum e_pixel_format
{
    PIXEL_FORMAT_ARGB8888,
    PIXEL_FORMAT_RGB565
} pixel_format_t;
/* 5-bit to 8-bit LUT */
const uint8_t lut_32[] =
{
    0,   8,   16, 25, 33, 41, 49, 58,
    66, 74, 82, 90, 99, 107, 115, 123,
```

```
    132, 140, 148, 156, 165, 173, 181, 189,
    197, 206, 214, 222, 230, 239, 247, 255
};
/* 6-bit to 8-bit LUT */
const uint8_t lut_64[] =
{
    0,   4,   8,   12, 16, 20, 24, 28,
    32, 36, 40, 45, 49, 53, 57, 61,
    65, 69, 73, 77, 81, 85, 89, 93,
    97, 101, 105, 109, 113, 117, 121, 125,
    130, 134, 138, 142, 146, 150, 154, 158,
    162, 166, 170, 174, 178, 182, 186, 190,
    194, 198, 202, 206, 210, 215, 219, 223,
    227, 231, 235, 239, 243, 247, 251, 255
};
void bitmap_rgb2ycbcr(uint32_t * out, uint8_t * in, uint32_t len, pixel_format_t
format);
/*************************************************************************************
***********************************
 * Convert an RGB buffer to Y'CbCr 4:2:2.
 *
 * NOTE: The width (in pixels) of the image to be converted must be divisible by 2.
 *
 * Parameters:
 * out Pointer to output buffer
 * in Pointer to input buffer
 * len Length of input buffer (in pixels)
 * format Input buffer format (ARGB8888 or RGB565)
 ************************************************************************************
**********************************/
void bitmap_rgb2ycbcr (uint32_t * out, uint8_t * in, uint32_t len, pixel_format_t
format)
{
    uint16_t in0;
```

```
   uint16_t in1;
   uint32_t r0;
   uint32_t g0;
   uint32_t b0;
   uint32_t r1;
   uint32_t g1;
   uint32_t b1;
   uint8_t y0;
   uint8_t y1;
   uint8_t cb0;
   uint8_t cr0;
   uint8_t cb1;
   uint8_t cr1;
/* Divide length by 2 as we're working with two pixels at a time */
   len >>= 1;
/* Perform the conversion */
while (len)
   {
/* Get R, G and B channel values */
if (format == PIXEL_FORMAT_RGB565)
     {
/* Get next two 16-bit values */
           in0 = *((uint16_t *) in);
           in += 2;
           in1 = *((uint16_t *) in);
           in += 2;
/* Decompose into individual channels */
           r0 = in0 >> 11;
           g0 = (in0 & RGB565_G_MASK) >> 5;
           b0 = in0 & RGB565_B_MASK;
           r1 = in1 >> 11;
           g1 = (in1 & RGB565_G_MASK) >> 5;
           b1 = in1 & RGB565_B_MASK;
     }
```

```
else
     {
/* Get each ARGB8888 channel in sequence, skipping alpha */
           b0 = *in++;
           g0 = *in++;
           r0 = *in++;
           in++;
           b1 = *in++;
           g1 = *in++;
           r1 = *in++;
           in++;
     }
/* Convert RGB565 data to RGB888 */
if (PIXEL_FORMAT_RGB565 == format)
     {
           r0 = lut_32[r0];
           g0 = lut_64[g0];
           b0 = lut_32[b0];
           r1 = lut_32[r1];
           g1 = lut_64[g1];
           b1 = lut_32[b1];
     }
/* Calculate Y'CbCr 4:4:4 values for the two pixels */
/* Algorithm based on method shown here: https://sistenix.com/rgb2ycbcr.html */
/* Original coefficients from https://en.wikipedia.org/wiki/YCbCr#JPEG_conversion */
      y0 = (uint8_t) (((r0 << 6) + (r0 << 3) + (r0 << 2) + r0 +
                       (g0 << 7) + (g0 << 4) + (g0 << 2) + (g0 << 1) +
                       (b0 << 4) + (b0 << 3) + (b0 << 2) + b0
                       ) >> 8);
      cb0 = (uint8_t) (C_0 - (((r0 << 5) + (r0 << 3) + (r0 << 1) + r0 +
                               (g0 << 6) + (g0 << 4) + (g0 << 2) + g0 -
                               (b0 << 7)
                               ) >> 8));
      cr0 = (uint8_t) (C_0 + (((r0 << 7) -
```

```
                                (g0 << 6) - (g0 << 5) - (g0 << 3) - (g0 << 1) - g0 -
                                (b0 << 4) - (b0 << 2) - b0
                                ) >> 8));
       y1 = (uint8_t) (((r1 << 6) + (r1 << 3) + (r1 << 2) + r1 +
                        (g1 << 7) + (g1 << 4) + (g1 << 2) + (g1 << 1) +
                        (b1 << 4) + (b1 << 3) + (b1 << 2) + b1
                        ) >> 8);
       cb1 = (uint8_t) (C_0 - (((r1 << 5) + (r1 << 3) + (r1 << 1) + r1 +
                                (g1 << 6) + (g1 << 4) + (g1 << 2) + g1 -
                                (b1 << 7)
                                ) >> 8));
       cr1 = (uint8_t) (C_0 + (((r1 << 7) -
                                (g1 << 6) - (g1 << 5) - (g1 << 3) - (g1 << 1) - g1 -
                                (b1 << 4) - (b1 << 2) - b1
                                ) >> 8));
 /* The above code is based on the floating point method shown here: */
 // y0 = (uint8_t) ((0.299F * (float) r0) + (0.587F * (float) g0) + (0.114F * (float)
b0));
 // y1 = (uint8_t) ((0.299F * (float) r1) + (0.587F * (float) g1) + (0.114F * (float)
b1));
 // cb0 = (uint8_t) (128.0F - (0.168736F * (float) r0) - (0.331264F * (float) g0) +
(0.5F * (float) b0));
 // cb1 = (uint8_t) (128.0F - (0.168736F * (float) r1) - (0.331264F * (float) g1) +
(0.5F * (float) b1));
 // cr0 = (uint8_t) (128.0F + (0.5F * (float) r0) - (0.418688F * (float) g0) -
(0.081312F * (float) b0));
 // cr1 = (uint8_t) (128.0F + (0.5F * (float) r1) - (0.418688F * (float) g1) -
(0.081312F * (float) b1));
 /* NOTE: The JPEG Codec expects signed instead of unsigned chrominance values. */
 /* Convert chrominance to -127..127 instead of 1..255 */
       cb0 = (uint8_t) ((int8_t) ((cb0 + cb1 + 1) >> 1) - C_0);
       cr0 = (uint8_t) ((int8_t) ((cr0 + cr1 + 1) >> 1) - C_0);
 /* Convert the two 4:4:4 values into 4:2:2 by averaging the chroma, then write to
output */
```

```
        *out++ = (uint32_t) (y0 + (cb0 << 8) + (y1 << 16) + (cr0 << 24));
        len--;
    }
}
```

Data Structures

struct	jpeg_instance_ctrl_t

Data Structure Documentation

◆ jpeg_instance_ctrl_t

struct jpeg_instance_ctrl_t

JPEG Codec module control block. DO NOT INITIALIZE. Initialization occurs when jpep_api_t::open is called.

Data Fields

uint32_t	open	JPEG Codec driver status.
jpeg_status_t	status	JPEG Codec operational status.
fsp_err_t	error_code	JPEG Codec error code (if any).
jpeg_mode_t	mode	Current mode (decode or encode).
uint32_t	horizontal_stride_bytes	Horizontal Stride settings.
uint32_t	output_buffer_size	Output buffer size.
jpeg_cfg_t const *	p_cfg	JPEG Decode configuration struct.
void const *	p_extend	JPEG Codec hardware dependent configuration */.
jpeg_decode_pixel_format_t	pixel_format	Pixel format.
uint16_t	total_lines_decoded	Track the number of lines decoded so far.
jpeg_decode_subsample_t	horizontal_subsample	Horizontal sub-sample setting.
uint16_t	lines_to_encode	Number of lines to encode.
uint16_t	vertical_resolution	vertical size
uint16_t	total_lines_encoded	Number of lines encoded.

Function Documentation

◆ R_JPEG_Open()

fsp_err_t R_JPEG_Open (jpeg_ctrl_t *const *p_api_ctrl*, jpeg_cfg_t const *const *p_cfg*)

Initialize the JPEG Codec module.

Note

This function configures the JPEG Codec for operation and sets up the registers for data format and pixel format based on user-supplied configuration parameters. Interrupts are enabled to support callbacks.

Return values

FSP_SUCCESS	JPEG Codec module is properly configured and is ready to take input data.
FSP_ERR_ALREADY_OPEN	JPEG Codec is already open.
FSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
FSP_ERR_IRQ_BSP_DISABLED	JEDI interrupt does not have an IRQ number.
FSP_ERR_INVALID_ARGUMENT	(Encode only) Quality factor, horizontal resolution and/or vertical resolution are invalid.
FSP_ERR_INVALID_ALIGNMENT	(Encode only) The horizontal resolution (at 16bpp) is not divisible by 8 bytes.

◆ R_JPEG_OutputBufferSet()

fsp_err_t R_JPEG_OutputBufferSet (jpeg_ctrl_t * *p_api_ctrl*, void * *p_output_buffer*, uint32_t *output_buffer_size*)

Assign a buffer to the JPEG Codec for storing output data.

Note

In Decode mode, the number of image lines to be decoded depends on the size of the buffer and the horizontal stride settings. Once the output buffer size is known, the horizontal stride value is known, and the input pixel format is known (the input pixel format is obtained by the JPEG decoder from the JPEG headers), the driver automatically computes the number of lines that can be decoded into the output buffer. After these lines are decoded, the JPEG engine pauses and a callback function is triggered, so the application is able to provide the next buffer for the JPEG module to resume the operation.

The JPEG decoding operation automatically starts after both the input buffer and the output buffer are set and the output buffer is big enough to hold at least eight lines of decoded image data.

Return values

FSP_SUCCESS	The output buffer is properly assigned to JPEG codec device.
FSP_ERR_ASSERTION	Pointer to the control block or output_buffer is NULL or output_buffer_size is 0.
FSP_ERR_INVALID_ALIGNMENT	Buffer starting address is not 8-byte aligned.
FSP_ERR_NOT_OPEN	JPEG not opened.
FSP_ERR_JPEG_UNSUPPORTED_IMAGE_SIZE	The number of horizontal pixels exceeds horizontal memory stride.
FSP_ERR_JPEG_BUFFERSIZE_NOT_ENOUGH	Invalid buffer size.
FSP_ERR_IN_USE	The output buffer cannot be changed during codec operation.

◆ R_JPEG_InputBufferSet()

fsp_err_t R_JPEG_InputBufferSet (jpeg_ctrl_t *const *p_api_ctrl*, void * *p_data_buffer*, uint32_t *data_buffer_size*)

Assign an input data buffer to the JPEG codec for processing.

Note

After the amount of data is processed, the JPEG driver triggers a callback function with the flag JPEG_PRV_OPERATION_INPUT_PAUSE set. The application supplies the next chunk of data to the driver so processing can resume.
The JPEG decoding operation automatically starts after both the input buffer and the output buffer are set, and the output buffer is big enough to hold at least one line of decoded image data.

If zero is provided for the decode data buffer size the JPEG Codec will never pause for more input data and will continue to read until either an image has been fully decoded or an error condition occurs.

Note

When encoding images the minimum data buffer size is 8 lines by 16 Y'CbCr 4:2:2 pixels (256 bytes). This corresponds to one minimum coded unit (MCU) of the resulting JPEG output.

Return values

FSP_SUCCESS	The input data buffer is properly assigned to JPEG Codec device.
FSP_ERR_ASSERTION	Pointer to the control block is NULL, or the pointer to the input_buffer is NULL, or the input_buffer_size is 0.
FSP_ERR_INVALID_ALIGNMENT	Buffer starting address is not 8-byte aligned.
FSP_ERR_NOT_OPEN	JPEG not opened.
FSP_ERR_IN_USE	The input buffer cannot be changed while the codec is running.
FSP_ERR_INVALID_CALL	In encode mode the output buffer must be set first.
FSP_ERR_JPEG_IMAGE_SIZE_ERROR	The buffer size is smaller than the minimum coded unit (MCU).

◆ R_JPEG_StatusGet()

fsp_err_t R_JPEG_StatusGet (jpeg_ctrl_t * *p_api_ctrl*, jpeg_status_t * *p_status*)

Get the status of the JPEG codec. This function can also be used to poll the device.

Return values

FSP_SUCCESS	The status information is successfully retrieved.
FSP_ERR_ASSERTION	Pointer to the control block or p_status is NULL.
FSP_ERR_NOT_OPEN	JPEG is not opened.

◆ R_JPEG_Close()

fsp_err_t R_JPEG_Close (jpeg_ctrl_t * *p_api_ctrl*)

Cancel an outstanding JPEG codec operation and close the device.

Return values

FSP_SUCCESS	The JPEG unit is stopped and the driver is closed.
FSP_ERR_ASSERTION	Pointer to the control block is NULL.
FSP_ERR_NOT_OPEN	JPEG not opened.

◆ R_JPEG_EncodeImageSizeSet()

fsp_err_t R_JPEG_EncodeImageSizeSet (jpeg_ctrl_t *const *p_api_ctrl*, jpeg_encode_image_size_t * *p_image_size*)

Set the image dimensions for an encode operation.

Note

Image dimensions must be set before setting the input buffer.

Return values

FSP_SUCCESS	Image size was successfully written to the JPEG Codec.
FSP_ERR_ASSERTION	Pointer to the control block or p_image_size is NULL.
FSP_ERR_INVALID_ALIGNMENT	Horizontal stride is not 8-byte aligned.
FSP_ERR_INVALID_ARGUMENT	Horizontal or vertical resolution is invalid or zero.
FSP_ERR_NOT_OPEN	JPEG not opened.
FSP_ERR_IN_USE	Image parameters cannot be changed while the codec is running.

◆ R_JPEG_DecodeLinesDecodedGet()

fsp_err_t R_JPEG_DecodeLinesDecodedGet (jpeg_ctrl_t * *p_api_ctrl*, uint32_t * *p_lines*)

Returns the number of lines decoded into the output buffer.

Note

Use this function to retrieve the number of image lines written to the output buffer after a partial decode operation. Combined with the horizontal stride settings and the output pixel format the application can compute the amount of data to read from the output buffer.

Return values

FSP_SUCCESS	Line count successfully returned.
FSP_ERR_ASSERTION	Pointer to the control block or p_lines is NULL.
FSP_ERR_NOT_OPEN	JPEG not opened.

◆ R_JPEG_DecodeHorizontalStrideSet()

fsp_err_t R_JPEG_DecodeHorizontalStrideSet (jpeg_ctrl_t * *p_api_ctrl*, uint32_t *horizontal_stride*)

Configure horizontal stride setting for decode operations.

Note

If the image size is known prior to the open call and/or the output buffer stride is constant, pass the horizontal stride value in the jpeg_cfg_t structure. Otherwise, after the image size becomes available use this function to set the output buffer horizontal stride value.

Return values

FSP_SUCCESS	Horizontal stride value is properly configured.
FSP_ERR_ASSERTION	Pointer to the control block is NULL.
FSP_ERR_INVALID_ALIGNMENT	Horizontal stride is zero or is not 8-byte aligned.
FSP_ERR_NOT_OPEN	JPEG not opened.

◆ R_JPEG_DecodeImageSizeGet()

fsp_err_t R_JPEG_DecodeImageSizeGet (jpeg_ctrl_t * *p_api_ctrl*, uint16_t * *p_horizontal_size*, uint16_t * *p_vertical_size*)

Obtain the size of an image being decoded.

Return values

FSP_SUCCESS	The image size is available and the horizontal and vertical values are stored in the memory pointed to by p_horizontal_size and p_vertical_size.
FSP_ERR_ASSERTION	Pointer to the control block is NULL and/or size is not ready.
FSP_ERR_NOT_OPEN	JPEG is not opened.

◆ R_JPEG_DecodeImageSubsampleSet()

fsp_err_t R_JPEG_DecodeImageSubsampleSet (jpeg_ctrl_t *const *p_api_ctrl*, jpeg_decode_subsample_t *horizontal_subsample*, jpeg_decode_subsample_t *vertical_subsample*)

Configure horizontal and vertical subsampling.

Note

This function can be used to scale the output of decoded image data.

Return values

FSP_SUCCESS	Horizontal subsample value is properly configured.
FSP_ERR_ASSERTION	Pointer to the control block is NULL.
FSP_ERR_NOT_OPEN	JPEG not opened.

◆ R_JPEG_DecodePixelFormatGet()

fsp_err_t R_JPEG_DecodePixelFormatGet (jpeg_ctrl_t * *p_api_ctrl*, jpeg_color_space_t * *p_color_space*)

Get the color format of the JPEG being decoded.

Return values

FSP_SUCCESS	The color format was successfully retrieved.
FSP_ERR_ASSERTION	Pointer to the control block is NULL.
FSP_ERR_NOT_OPEN	JPEG is not opened.

◆ R_JPEG_ModeSet()

fsp_err_t R_JPEG_ModeSet (jpeg_ctrl_t *const *p_api_ctrl*, jpeg_mode_t *mode*)

Switch between encode and decode mode (or vice-versa).

Note

The codec must not be idle in order to switch modes.

Return values

FSP_SUCCESS	Mode changed successfully.
FSP_ERR_ASSERTION	p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_IN_USE	JPEG Codec is currently in use.
FSP_ERR_INVALID_ARGUMENT	(Encode only) Quality factor, horizontal resolution and/or vertical resolution are invalid.
FSP_ERR_INVALID_ALIGNMENT	(Encode only) The horizontal resolution (at 16bpp) is not divisible by 8 bytes.

4.2.39 Key Interrupt (r_kint)

Modules

Functions

fsp_err_t	R_KINT_Open (keymatrix_ctrl_t *const p_api_ctrl, keymatrix_cfg_t const *const p_cfg)
fsp_err_t	R_KINT_Enable (keymatrix_ctrl_t *const p_api_ctrl)
fsp_err_t	R_KINT_Disable (keymatrix_ctrl_t *const p_api_ctrl)
fsp_err_t	R_KINT_Close (keymatrix_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the KINT peripheral on RA MCUs. This module implements the Key Matrix Interface.

Overview

The KINT module configures the Key Interrupt (KINT) peripheral to detect rising or falling edges on any of the KINT channels. When such an event is detected on any of the configured pins, the module generates an interrupt.

Features

- Detect rising or falling edges on KINT channels
- Callback for notifying the application when edges are detected on the configured channels

Configuration

Build Time Configurations for r_kint

The following build time configurations are defined in fsp_cfg/r_kint_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Input > Key Matrix (r_kint)

This module can be added to the Stacks tab via New Stack > Input > Key Matrix (r_kint).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_kint0	Module name.
Input > Key Interrupt Flag Mask	• Channel 0 • Channel 1 • Channel 2 • Channel 3 • Channel 4 • Channel 5 • Channel 6 • Channel 7		Select channels to enable.
Interrupts > Trigger Type	• Falling Edge • Rising Edge	Rising Edge	Specifies if the enabled channels detect a rising edge or a falling edge. NOTE: either all channels detecting a rising edge or all channels detecting a falling edge.
Interrupts > Callback	Name must be a valid C symbol	kint_callback	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQ triggers.

Interrupts > Key Interrupt Priority	MCU Specific Options	Select the key interrupt priority.

Clock Configuration

The KINT peripheral runs on PCLKB.

Pin Configuration

The KRn pins are key switch matrix row input pins.

Usage Notes

Connecting a Switch Matrix

The KINT module is designed to scan the rows of a switch matrix where each row is connected to a number of columns through switches. A periodic timer (or other mechanism) sets one column pin high at a time. Any switches that are pressed on the driven column cause a rising (or falling) edge on the row pin (KRn) causing an interrupt.

Note

In applications where multiple keys may be pressed at the same time it is recommended to put a diode inline with each switch to prevent ghosting.

Handling Multiple Pins

When an edge is detected on multiple pins at the same time, a single IRQ will be generated. A mask of all the pins that detected an edge will be passed to the callback.

Examples

Basic Example

This is a basic example of minimal use of the KINT in an application.

```
static volatile uint32_t g_channel_mask;
static volatile uint32_t g_kint_edge_detected = 0U;
/* Called from key_int_isr */
void r_kint_callback (keymatrix_callback_args_t * p_args)
{
    g_channel_mask       = p_args->channel_mask;
    g_kint_edge_detected = 1U;
}
void r_kint_example ()
{
 /* Configure the KINT. */
 fsp_err_t err = R_KINT_Open(&g_kint_ctrl, &g_kint_cfg);
```

```
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Enable the KINT. */
    err = R_KINT_Enable(&g_kint_ctrl);
    assert(FSP_SUCCESS == err);
 while (0 == g_kint_edge_detected)
    {
 /* Wait for interrupt. */
    }
}
```

Data Structures

struct	kint_instance_ctrl_t

Data Structure Documentation

◆ kint_instance_ctrl_t

struct kint_instance_ctrl_t
Channel instance control block. DO NOT INITIALIZE. Initialization occurs when keymatrix_api_t::open is called.

Function Documentation

◆ R_KINT_Open()

fsp_err_t R_KINT_Open (keymatrix_ctrl_t *const *p_api_ctrl*, keymatrix_cfg_t const *const *p_cfg*)

Configure all the Key Input (KINT) channels and provides a handle for use with the rest of the KINT API functions. Implements keymatrix_api_t::open.

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	One of the following parameters may be NULL: p_cfg, or p_ctrl or the callback.
FSP_ERR_ALREADY_OPEN	The module has already been opened.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel mask is invalid.

◆ R_KINT_Enable()

fsp_err_t R_KINT_Enable (keymatrix_ctrl_t *const *p_api_ctrl*)

This function enables interrupts for the KINT peripheral after clearing any pending requests. Implements keymatrix_api_t::enable.

Return values

FSP_SUCCESS	Interrupt enabled successfully.
FSP_ERR_ASSERTION	The p_ctrl parameter was null.
FSP_ERR_NOT_OPEN	The peripheral is not opened.

◆ R_KINT_Disable()

fsp_err_t R_KINT_Disable (keymatrix_ctrl_t *const *p_api_ctrl*)

This function disables interrupts for the KINT peripheral. Implements keymatrix_api_t::disable.

Return values

FSP_SUCCESS	Interrupt disabled successfully.
FSP_ERR_ASSERTION	The p_ctrl parameter was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

◆ R_KINT_Close()

fsp_err_t R_KINT_Close (keymatrix_ctrl_t *const *p_api_ctrl*)

Clear the KINT configuration and disable the KINT IRQ. Implements keymatrix_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
FSP_ERR_NOT_OPEN	The module is not opened.

4.2.40 Low Power Modes (r_lpm)

Modules

Functions

fsp_err_t	R_LPM_Open (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)
fsp_err_t	R_LPM_Close (lpm_ctrl_t *const p_api_ctrl)
fsp_err_t	R_LPM_LowPowerReconfigure (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)
fsp_err_t	R_LPM_LowPowerModeEnter (lpm_ctrl_t *const p_api_ctrl)
fsp_err_t	R_LPM_IoKeepClear (lpm_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the LPM peripheral on RA MCUs. This module implements the Low Power Modes Interface.

Overview

The low power modes driver is used to configure and place the device into the desired low power mode. Various sources can be configured to wake from standby, request snooze mode, end snooze mode or end deep standby mode.

Features

The LPM HAL module has the following key features:

- Supports the follwowing low power modes:
  - Deep Software Standby mode (On supported MCUs)
  - Software Standby mode
  - Sleep mode
  - Snooze mode
- Supports reducing power consumption when in deep software standby mode through internal power supply control and by resetting the states of I/O ports.
- Supports disabling and enabling the MCU's other hardware peripherals

Configuration

Build Time Configurations for r_lpm

The following build time configurations are defined in fsp_cfg/r_lpm_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Standby Limit	• Enabled • Disabled	Disabled	If enabled, standby configuration only

applies in R_LPM_LowPowerModeEnter. Otherwise, standby configuration applies to any WFI call.

Configurations for Power > Low Power Modes (r_lpm)

This module can be added to the Stacks tab via New Stack > Power > Low Power Modes (r_lpm). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_lpm0	Module name.
General > Low Power Mode	MCU Specific Options		Power mode to be entered.
General > Output port state in standby and deep standby	MCU Specific Options		Select the state of output pins during standby. Applies to address output, data output, and other bus control output pins.
Standby Options > Wake Sources	MCU Specific Options		Enable wake from standby from these Sources.
Standby Options > Snooze Request Source	MCU Specific Options		Select the event that will enter snooze.
Standby Options > Snooze End Sources	MCU Specific Options		Enable wake from snooze from these sources.
Standby Options > DTC state in Snooze Mode	• Disabled • Enabled	Disabled	Enable wake from snooze from this source.
Standby Options > Snooze Cancel Source	MCU Specific Options		Select an interrupt source to cancel snooze.
Deep Standby Options > I/O Port Retention	MCU Specific Options		Select the state of the IO Pins after exiting deep standby mode.
Deep Standby Options > Power-Supply Control	MCU Specific Options		Select the state of the internal power supply in deep standby mode.
Deep Standby Options > Cancel Sources	MCU Specific Options		Enable wake from deep standby using these sources.
Deep Standby Options	MCU Specific Options		Falling edge trigger is

> Cancel Edges	default. Select sources to enable wake from deep standby with rising edge.

Clock Configuration

This module does not have any selectable clock sources.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Sleep Mode

At power on, by default sleep is set as the low-power mode. Sleep mode is the most convenient low-power mode available, as it does not require any special configuration (other than configuring and enabling a suitable interrupt or event to wake the MCU from sleep) to return to normal program-execution mode. The states of the SRAM, the processor registers, and the hardware peripherals are all maintained in sleep mode, and the time needed to enter and wake from sleep is minimal. Any interrupt causes the MCU device to wake from sleep mode, including the Systick interrupt used by the RTOS scheduler.

Software Standby Mode

In software-standby mode, the CPU, as well as most of the on-chip peripheral functions and all of the internal oscillators, are stopped. The contents of the CPU internal registers and SRAM data, the states of on-chip peripheral functions, and I/O Ports are all retained. Software-standby mode allows significant reduction in power consumption, because most of the oscillators are stopped in this mode. Like sleep mode, standby mode requires an interrupt or event be configured and enabled to wake up.

Snooze Mode

Snooze mode can be used with some MCU peripherals to execute basic tasks while keeping the MCU in a low-power state. Many core peripherals and all clocks can be selected to run during Snooze, allowing for more flexible low-power configuration than Software Standby mode. To enable Snooze, select "Software Standby mode with Snooze mode enabled" for the "Low Power Mode" configuration option. Snooze mode settings (including entry/exit sources) are available under "Standby Options".

Deep Software Standby Mode

Deep Software Standby Mode is only available on some MCU devices. The MCU always wakes from Deep Software Standby Mode by going through reset, either by the negation of the reset pin or by one of the wakeup sources configurable in the "Deep Standby Options" configuration group.

The Reset Status Registers can be used to determine if the reset occured after coming out of deep sofware standby. For example, R_SYSTEM->RSTSR0_b.DPSRSTF is set to 1 after a deep software standby reset.

I/O Port Retention can be enabled to maintain I/O port configuration across a deep software standby reset. Retention can be cancelled through the R_LPM_IoKeepClear API.

Limitations

Developers should be aware of the following limitations when using the LPM:

- Flash stop (code flash disable) is not supported. See the section "Flash Operation Control Register (FLSTOP)" of the RA2/RA4 Family Hardware User's Manual.
- Reduced SRAM retention area in software standby mode is not supported. See the section "Power Save Memory Control Register (PSMCR)" of the RA4 Hardware User's Manual.
- Only one Snooze Request Source can be used at a time.
- When using Snooze mode with SCI0 RXD as the snooze source the system clock must be HOCO and the MOCO, Main Oscillator and PLL clocks must be turned off.
- If the main oscillator or PLL with main oscillator source is used for the system clock, the wake time from standby mode can be affected by the Main Oscillator Wait Time Setting in the MOSCWTCR register. This register setting is available to be changed through the Main Oscillator Wait Time setting in the CGC module properties. See the "Wakeup Timing and Duration" table in Electrical Characteristics for more information.
- When using the DC-DC regulator (where available), the MCU will temporarily switch to the LDO if Software Standby or Snooze is requested and back again when it is cancelled. Switching to the LDO incurs a 60 microsecond critical section wherein all interrupts AND peripherals are stopped. Switching back to DCDC from the LDO incurs an additional 22 microsecond critical section (peripherals running).

Examples

LPM Sleep Example

This is a basic example of minimal use of the LPM in an application. The LPM instance is opened and the configured low-power mode is entered.

```
void r_lpm_sleep (void)
{
 fsp_err_t err = R_LPM_Open(&g_lpm_ctrl, &g_lpm_cfg_sleep);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    err = R_LPM_LowPowerModeEnter(&g_lpm_ctrl);
    assert(FSP_SUCCESS == err);
}
```

LPM Deep Software Standby Example

```
void r_lpm_deep_software_standby (void)
{
 fsp_err_t err;
```

```
    err = R_LPM_Open(&g_lpm_ctrl, &g_lpm_cfg_deep_software_standby);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Check the Deep Software Standby Reset Flag. */
 if (1U == R_SYSTEM->RSTSR0_b.DPSRSTF)
    {
 /* Clear the IOKEEP bit to allow I/O Port use. */
       err = R_LPM_IoKeepClear(&g_lpm_ctrl);
      assert(FSP_SUCCESS == err);
    }
 /* Add user code here. */
 /* Reconfigure the module to set the IOKEEP bit before entering deep software
standby. */
    err = R_LPM_LowPowerReconfigure(&g_lpm_ctrl, &g_lpm_cfg_deep_software_standby);
    assert(FSP_SUCCESS == err);
    err = R_LPM_LowPowerModeEnter(&g_lpm_ctrl);
 /* Code after R_LPM_LowPowerModeEnter when using Deep Software Standby never be
executed.
  * Deep software standby exits by resetting the MCU. */
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	lpm_instance_ctrl_t

Data Structure Documentation

◆ lpm_instance_ctrl_t

struct lpm_instance_ctrl_t
LPM private control block. DO NOT MODIFY. Initialization occurs when R_LPM_Open() is called.

Function Documentation

◆ R_LPM_Open()

fsp_err_t R_LPM_Open (lpm_ctrl_t *const *p_api_ctrl*, lpm_cfg_t const *const *p_cfg*)

Perform any necessary initialization

Return values

FSP_SUCCESS	LPM instance opened
FSP_ERR_ASSERTION	Null Pointer
FSP_ERR_ALREADY_OPEN	LPM instance is already open
FSP_ERR_UNSUPPORTED	This MCU does not support Deep Software Standby
FSP_ERR_INVALID_ARGUMENT	One of the following: • Invalid snooze entry source • Invalid snooze end sources
FSP_ERR_INVALID_MODE	One of the following: • Invalid low power mode • Invalid DTC option for snooze mode • Invalid deep standby end sources • Invalid deep standby end sources edges • Invalid power supply option for deep standby • Invalid IO port option for deep standby • Invalid output port state setting for standby or deep standby • Invalid sources for wake from standby mode • Invalid power supply option for standby • Invalid IO port option for standby • Invalid standby end sources • Invalid standby end sources edges

◆ R_LPM_Close()

fsp_err_t R_LPM_Close (lpm_ctrl_t *const *p_api_ctrl*)

Close the LPM Instance

Return values

FSP_SUCCESS	LPM driver closed
FSP_ERR_NOT_OPEN	LPM instance is not open
FSP_ERR_ASSERTION	Null Pointer

◆ R_LPM_LowPowerReconfigure()

fsp_err_t R_LPM_LowPowerReconfigure (lpm_ctrl_t *const *p_api_ctrl*, lpm_cfg_t const *const *p_cfg*)

Configure a low power mode

NOTE: This function does not enter the low power mode, it only configures parameters of the mode. Execution of the WFI instruction is what causes the low power mode to be entered.

Return values

FSP_SUCCESS	Low power mode successfuly applied
FSP_ERR_ASSERTION	Null Pointer
FSP_ERR_NOT_OPEN	LPM instance is not open
FSP_ERR_UNSUPPORTED	This MCU does not support Deep Software Standby
FSP_ERR_INVALID_ARGUMENT	One of the following: • Invalid snooze entry source • Invalid snooze end sources
FSP_ERR_INVALID_MODE	One of the following: • Invalid low power mode • Invalid DTC option for snooze mode • Invalid deep standby end sources • Invalid deep standby end sources edges • Invalid power supply option for deep standby • Invalid IO port option for deep standby • Invalid output port state setting for standby or deep standby • Invalid sources for wake from standby mode • Invalid power supply option for standby • Invalid IO port option for standby • Invalid standby end sources • Invalid standby end sources edges

◆ R_LPM_LowPowerModeEnter()

fsp_err_t R_LPM_LowPowerModeEnter (lpm_ctrl_t *const *p_api_ctrl*)

Enter low power mode (sleep/standby/deep standby) using WFI macro.

Function will return after waking from low power mode.

Return values

FSP_SUCCESS	Successful.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	LPM instance is not open
FSP_ERR_INVALID_MODE	One of the following: • HOCO was not system clock when using snooze mode with SCI0/RXD0. • HOCO was not stable when using snooze mode with SCI0/RXD0. • MOCO was running when using snooze mode with SCI0/RXD0. • MAIN OSCILLATOR was running when using snooze mode with SCI0/RXD0. • PLL was running when using snooze mode with SCI0/RXD0. • Unable to disable ocillator stop detect when using standby or deep standby.

◆ R_LPM_IoKeepClear()

fsp_err_t R_LPM_IoKeepClear (lpm_ctrl_t *const *p_api_ctrl*)

Clear the IOKEEP bit after deep software standby

Return values

FSP_SUCCESS	DPSBYCR_b.IOKEEP bit cleared Successfully.
FSP_ERR_UNSUPPORTED	Deep standby mode not supported on this MCU.

4.2.41 Low Voltage Detection (r_lvd)

Modules

Functions

fsp_err_t	R_LVD_Open (lvd_ctrl_t *const p_api_ctrl, lvd_cfg_t const *const p_cfg)
fsp_err_t	R_LVD_Close (lvd_ctrl_t *const p_api_ctrl)
fsp_err_t	R_LVD_StatusGet (lvd_ctrl_t *const p_api_ctrl, lvd_status_t *p_lvd_status)
fsp_err_t	R_LVD_StatusClear (lvd_ctrl_t *const p_api_ctrl)
fsp_err_t	R_LVD_CallbackSet (lvd_ctrl_t *const p_api_ctrl, void(*p_callback)(lvd_callback_args_t *), void const *const p_context, lvd_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the LVD peripheral on RA MCUs. This module implements the Low Voltage Detection Interface.

Overview

The Low Voltage Detection module configures the voltage monitors to detect when V_{CC} crosses a specified threshold.

Features

The LVD HAL module supports the following functions:

- Two run-time configurable voltage monitors (Voltage Monitor 1, Voltage Monitor 2)
  - Configurable voltage threshold
  - Digital filter (Available on specific MCUs)
  - Support for both interrupt or polling
    - NMI or maskable interrupt can be configured
  - Rising, falling, or both edge event detection
  - Support for resetting the MCU when V_{CC} falls below configured threshold.

Configuration

Build Time Configurations for r_lvd

The following build time configurations are defined in fsp_cfg/r_lvd_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Monitoring > Low Voltage Detection (r_lvd)

This module can be added to the Stacks tab via New Stack > Monitoring > Low Voltage Detection (r_lvd). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_lvd	Module name.
Monitor Number	MCU Specific Options		Select the LVD monitor.
Digital Filter	MCU Specific Options		Enable the digital filter and select the digital filter clock divider.
Voltage Threshold	MCU Specific Options		Select the low voltage detection threshold.
Detection Response	• Maskable interrupt • Non-maskable interrupt • Reset MCU (Only available for falling edge) • No response (Voltage monitor status will be polled)	No response (Voltage monitor status will be polled)	Select what happens when the voltage crosses the threshold voltage.
Voltage Slope	• Falling voltage • Rising voltage • Rising or falling voltage	Falling voltage	Select detection on rising voltage, falling voltage or both.
Negation Delay	• Delay from reset • Delay from voltage returning to normal range	Delay from reset	Negation of the monitor signal can either be delayed from the reset event or from voltage returning to normal range.
Monitor Interrupt Callback	Name must be a valid C symbol.	NULL	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQ triggers.
LVD Monitor Interrupt Priority	MCU Specific Options		Select the LVD Monitor interrupt priority.

Clock Configuration

The LOCO clock must be enabled in order to use the digital filter.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Startup Edge Detection

If V_{CC} is below the threshold prior to configuring the voltage monitor for falling edge detection, the monitor will immediately detect the a falling edge condition. If V_{CC} is above the threshold prior to configuring the monitor for rising edge detection, the monitor will not detect a rising edge condition until V_{CC} falls below the threshold and then rises above it again.

Voltage Monitor 0

The LVD HAL module only supports configuring voltage monitor 1 and voltage monitor 2. Voltage monitor 0 can be configured by setting the appropriate bits in the OFS1 register. This means that voltage monitor 0 settings cannot be changed at runtime.

Voltage monitor 0 supports the following features

- Configurable Voltage Threshold (V_{DET0})
- Reset the device when V_{CC} falls below V_{DET0}

Limitations

- The digital filter must be disabled when using voltage monitors in Software Standby or Deep Software Standby.
- Deep Software Standby mode is not possible if the voltage monitor is configured to reset the MCU.
- When the detection response is set to reset, only voltage falling edge detection is possible.

Examples

Basic Example

This is a basic example of minimal use of the LVD in an application.

```
void basic_example (void)
{
 fsp_err_t err = R_LVD_Open(&g_lvd_ctrl, &g_lvd_cfg);
    assert(FSP_SUCCESS == err);
 while (1)
    {
 lvd_status_t status;
       err = R_LVD_StatusGet(&g_lvd_ctrl, &status);
```

```
      assert(FSP_SUCCESS == err);
 if (LVD_THRESHOLD_CROSSING_DETECTED == status.crossing_detected)
      {
            err = R_LVD_StatusClear(&g_lvd_ctrl);
      assert(FSP_SUCCESS == err);
 /* Do something */
       }
    }
}
```

Interrupt Example

This is a basic example of using a LVD instance that is configured to generate an interrupt.

```
void interrupt_example (void)
{
 fsp_err_t err = R_LVD_Open(&g_lvd_ctrl, &g_lvd_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (1)
    {
 /* Application Process */
 /* Application will be interrupted when Vcc crosses the configured threshold. */
    }
}
/* Called when Vcc crosses configured threshold. */
void lvd_callback (lvd_callback_args_t * p_args)
{
 if (LVD_CURRENT_STATE_BELOW_THRESHOLD == p_args->current_state)
    {
 /* Do Something */
    }
}
```

Reset Example

This is a basic example of using a LVD instance that is configured to reset the MCU.

```
void reset_example (void)
{
 if (1U == R_SYSTEM->RSTSR0_b.LVD1RF)
    {
 /* The system is coming out of reset because Vcc crossed configured voltage
threshold. */
 /* Clear Voltage Monitor 1 Reset Detect Flag. */
       R_SYSTEM->RSTSR0_b.LVD1RF = 0;
    }
 fsp_err_t err = R_LVD_Open(&g_lvd_ctrl, &g_lvd_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (1)
    {
 /* Application Process */
 /* Application will reset when Vcc crosses the configured threshold. */
    }
}
```

Data Structures

struct	lvd_instance_ctrl_t

Data Structure Documentation

◆ lvd_instance_ctrl_t

struct lvd_instance_ctrl_t
LVD instance control structure

Function Documentation

◆ R_LVD_Open()

fsp_err_t R_LVD_Open (lvd_ctrl_t *const *p_api_ctrl*, lvd_cfg_t const *const *p_cfg*)

Initializes a voltage monitor and detector according to the passed-in configuration structure.

Parameters

[in]	p_api_ctrl	Pointer to the control structure for the driver instance
[in]	p_cfg	Pointer to the configuration structure for the driver instance

Note

Digital filter is not to be used with standby modes.
Startup time can take on the order of milliseconds for some configurations.

Example:

```
fsp_err_t err = R_LVD_Open(&g_lvd_ctrl, &g_lvd_cfg);
```

Return values

FSP_SUCCESS	Successful
FSP_ERR_ASSERTION	Requested configuration was invalid
FSP_ERR_ALREADY_OPEN	The instance was already opened
FSP_ERR_IN_USE	Another instance is already using the desired monitor
FSP_ERR_UNSUPPORTED	Digital filter was enabled on a device that does not support it

◆ R_LVD_Close()

fsp_err_t R_LVD_Close (lvd_ctrl_t *const *p_api_ctrl*)

Disables the LVD peripheral. Closes the driver instance.

Parameters

[in]	p_api_ctrl	Pointer to the control block structure for the driver instance

Return values

FSP_SUCCESS	Successful
FSP_ERR_ASSERTION	An argument was NULL
FSP_ERR_NOT_OPEN	Driver is not open

◆ R_LVD_StatusGet()

fsp_err_t R_LVD_StatusGet (lvd_ctrl_t *const *p_api_ctrl*, lvd_status_t * *p_lvd_status*)

Get the current state of the monitor (threshold crossing detected, voltage currently above or below threshold).

Parameters

[in]	p_api_ctrl	Pointer to the control structure for the driver instance
[out]	p_lvd_status	Pointer to status structure

Example:

```
    err = R_LVD_StatusGet(&g_lvd_ctrl, &status);
```

Return values

FSP_SUCCESS	Successful
FSP_ERR_ASSERTION	An argument was NULL
FSP_ERR_NOT_OPEN	Driver is not open

◆ R_LVD_StatusClear()

fsp_err_t R_LVD_StatusClear (lvd_ctrl_t *const *p_api_ctrl*)

Clears the latched status of the monitor.

Parameters

[in]	p_api_ctrl	Pointer to the control structure for the driver instance

Return values

FSP_SUCCESS	Successful
FSP_ERR_ASSERTION	An argument was NULL
FSP_ERR_NOT_OPEN	Driver is not open

◆ R_LVD_CallbackSet()

fsp_err_t R_LVD_CallbackSet (lvd_ctrl_t *const *p_api_ctrl*, void(*)(lvd_callback_args_t *) *p_callback*, void const *const *p_context*, lvd_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements lvd_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.42 Operational Amplifier (r_opamp)

Modules

Functions

fsp_err_t	R_OPAMP_Open (opamp_ctrl_t *const p_api_ctrl, opamp_cfg_t const *const p_cfg)

fsp_err_t	R_OPAMP_InfoGet (opamp_ctrl_t *const p_api_ctrl, opamp_info_t *const p_info)
fsp_err_t	R_OPAMP_Start (opamp_ctrl_t *const p_api_ctrl, uint32_t const channel_mask)
fsp_err_t	R_OPAMP_Stop (opamp_ctrl_t *const p_api_ctrl, uint32_t const channel_mask)
fsp_err_t	R_OPAMP_StatusGet (opamp_ctrl_t *const p_api_ctrl, opamp_status_t *const p_status)
fsp_err_t	R_OPAMP_Trim (opamp_ctrl_t *const p_api_ctrl, opamp_trim_cmd_t const cmd, opamp_trim_args_t const *const p_args)
fsp_err_t	R_OPAMP_Close (opamp_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the OPAMP peripheral on RA MCUs. This module implements the OPAMP Interface.

Overview

The OPAMP HAL module provides a high level API for signal amplification applications and supports the OPAMP peripheral available on RA MCUs.

Features

- Low power or high-speed mode
- Start by software or AGT compare match
- Stop by software or ADC conversion end (stop by ADC conversion end only supported on op-amp channels configured to start by AGT compare match)
- Trimming available on some MCUs (see hardware manual)

Configuration

Build Time Configurations for r_opamp

The following build time configurations are defined in fsp_cfg/r_opamp_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Analog > Operational Amplifier (r_opamp)

This module can be added to the Stacks tab via New Stack > Analog > Operational Amplifier (r_opamp).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_opamp0	Module name.
AGT Start Trigger Configuration (N/A unless AGT Start Trigger is Selected for the Channel)	• AGT1 Compare Match Starts OPAMPs 0 and 2 if configured for AGT Start, AGT0 Compare Match Starts OPAMPs 1 and 3 if configured for AGT Start • AGT1 Compare Match Starts OPAMPs 0 and 1 if configured for AGT Start, AGT0 Compare Match Starts OPAMPs 2 and 3 if configured for AGT Start • AGT1 Compare Match Starts all OPAMPs configured for AGT Start	AGT1 Compare Match Starts all OPAMPs configured for AGT Start	Configure which AGT channel event triggers which op-amp channel. The AGT compare match event only starts the op-amp channel if the AGT Start trigger is selected in the Trigger configuration for the channel.
Power Mode	MCU Specific Options		Configure the op-amp based on power or speed requirements. This setting affects the minimum required stabilization time. Middle speed is not available for all MCUs.
Trigger Channel 0	MCU Specific Options		Select the event triggers to start or stop op-amp channel 0. If the event trigger is selected for start, the start() API enables the event trigger for this channel. If the event trigger is selected for stop, the stop() API disables the event trigger for this channel.
Trigger Channel 1	MCU Specific Options		Select the event triggers to start or stop

			op-amp channel 1. If the event trigger is selected for start, the start() API enables the event trigger for this channel. If the event trigger is selected for stop, the stop() API disables the event trigger for this channel.
Trigger Channel 2	• Software Start Software Stop • AGT Start Software Stop • AGT Start ADC Stop	Software Start Software Stop	Select the event triggers to start or stop op-amp channel 2. If the event trigger is selected for start, the start() API enables the event trigger for this channel. If the event trigger is selected for stop, the stop() API disables the event trigger for this channel.
Trigger Channel 3	MCU Specific Options		Select the event triggers to start or stop op-amp channel 3. If the event trigger is selected for start, the start() API enables the event trigger for this channel. If the event trigger is selected for stop, the stop() API disables the event trigger for this channel.
OPAMP AMP0OS	MCU Specific Options		Select output to connect to AMP0O pin
OPAMP AMP0PS	MCU Specific Options		Select input to connect to AMP0+ pin
OPAMP AMP0MS	MCU Specific Options		Select input to connect to AMP0- pin
OPAMP AMP1PS	MCU Specific Options		Select input to connect to AMP1+ pin
OPAMP AMP1MS	MCU Specific Options		Select input to connect to AMP1- pin
OPAMP AMP2PS	MCU Specific Options		Select input to connect to AMP2+ pin
OPAMP AMP2MS	MCU Specific Options		Select input to connect to AMP2- pin

Clock Configuration

The OPAMP runs on PCLKB.

Pin Configuration

To use the OPAMP HAL module, the port pins for the channels receiving the analog input must be set as inputs on the **Pins** tab of the RA Configuration editor.

Refer to the most recent FSP Release Notes for any additional operational limitations for this module.

Usage Notes

Trimming the OPAMP

- On MCUs that support trimming, the op-amp trim register is set to the factory default after the Open API is called.
- This function allows the application to trim the operational amplifier to a user setting, which overwrites the factory default trim values.
- Supported on selected MCUs. See hardware manual for details.
- Not supported if configured for low power mode (OPAMP_MODE_LOW_POWER).
- This function is not reentrant. Only one side of one op-amp can be trimmed at a time. Complete the procedure for one side of one channel before calling the trim API with the command OPAMP_TRIM_CMD_START again.
    - The trim procedure works as follows:
    - Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
    - Connect a fixed voltage to the Pch (+) input.
    - Connect the Nch (-) input to the op-amp output to create a voltage follower.
    - Ensure the op-amp is operating and stabilized.
    - Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
    - Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
    - Iterate over the following loop 5 times:
        - Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
        - Measure the op-amp output using the SAR ADC (referred to as B in the next step).
        - If A <= B, call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.
    - Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_START.
    - Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
    - Iterate over the following loop 5 times:
        - Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
        - Measure the op-amp output using the SAR ADC (referred to as B in the next step).
        - If A <= B, call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.

Examples

Basic Example

This is a basic example of minimal use of the R_OPAMP in an application. The example demonstrates configuring OPAMP channel 0 for high speed mode, starting the OPAMP and reading the status of the OPAMP channel running. It also verifies that the stabilization wait time is the expected time for selected power mode

```
#define OPAMP_EXAMPLE_CHANNEL (0U)
void basic_example (void)
{
 fsp_err_t err;
 /* Initialize the OPAMP module. */
    err = R_OPAMP_Open(&g_opamp_ctrl, &g_opamp_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the OPAMP module. */
    err = R_OPAMP_Start(&g_opamp_ctrl, 1 << OPAMP_EXAMPLE_CHANNEL);
    assert(FSP_SUCCESS == err);
 /* Look up the required stabilization wait time. */
 opamp_info_t info;
    err = R_OPAMP_InfoGet(&g_opamp_ctrl, &info);
    assert(FSP_SUCCESS == err);
 /* Wait for the OPAMP to stabilize. */
 R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
}
```

Trim Example

This example demonstrates the typical trimming procedure for opamp channel 0 using R_OPAMP_Trim() API.

```
#ifndef OPAMP_EXAMPLE_CHANNEL
 #define OPAMP_EXAMPLE_CHANNEL (0U)
#endif
#ifndef OPAMP_EXAMPLE_ADC_CHANNEL
 #define OPAMP_EXAMPLE_ADC_CHANNEL (ADC_CHANNEL_2)
#endif
#define ADC_SCAN_END_DELAY (100U)
#define OPAMP_TRIM_LOOP_COUNT (5)
#define ADC_SCAN_END_MAX_TIMEOUT (0xFFFF)
```

```
uint32_t          g_callback_event_counter = 0;
opamp_trim_args_t trim_args_ch =
{
    .channel = OPAMP_EXAMPLE_CHANNEL,
    .input   = OPAMP_TRIM_INPUT_PCH
};
/* This callback is called when ADC Scan Complete event is generated. */
void adc_callback (adc_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
    g_callback_event_counter++;
}
void trimming_example (void)
{
 fsp_err_t err;
 /* On RA2A1, configure negative feedback and put DAC12 signal on AMP0+ Pin. */
    g_opamp_cfg_extend.plus_input_select_opamp0 = OPAMP_PLUS_INPUT_AMPPS7;
    g_opamp_cfg_extend.minus_input_select_opamp0 = OPAMP_MINUS_INPUT_AMPMS7;
 /* Initialize the OPAMP module. */
    err = R_OPAMP_Open(&g_opamp_ctrl, &g_opamp_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start the OPAMP module. */
    err = R_OPAMP_Start(&g_opamp_ctrl, 1 << OPAMP_EXAMPLE_CHANNEL);
    assert(FSP_SUCCESS == err);
 /* Look up the required stabilization wait time. */
 opamp_info_t info;
    err = R_OPAMP_InfoGet(&g_opamp_ctrl, &info);
    assert(FSP_SUCCESS == err);
 /* Wait for the OPAMP to stabilize. */
 R_BSP_SoftwareDelay(info.min_stabilization_wait_us, BSP_DELAY_UNITS_MICROSECONDS);
 /* Call trim() for the Pch (+) side input */
    trim_procedure(&trim_args_ch);
    assert(FSP_SUCCESS == err);
```

```
    trim_args_ch.input = OPAMP_TRIM_INPUT_NCH;
 /* Call trim() for the Nch (-) side input */
    trim_procedure(&trim_args_ch);
}
void trim_procedure (opamp_trim_args_t * trim_args)
{
 fsp_err_t err;
 /* Call trim() for the selected channel and input with command OPAMP_TRIM_CMD_START.
*/
    err = R_OPAMP_Trim(&g_opamp_ctrl, OPAMP_TRIM_CMD_START, trim_args);
    assert(FSP_SUCCESS == err);
 /* Measure the fixed voltage connected to the channel input using the SAR ADC and
save the value
  * (referred to as result_a later in this procedure). */
 /* Reset the ADC callback counter */
    g_callback_event_counter = 0;
    err = R_ADC_ScanStart(&g_adc_ctrl);
    assert(FSP_SUCCESS == err);
 /* Wait for ADC scan complete flag */
    uint32_t timeout = ADC_SCAN_END_MAX_TIMEOUT;
 while (g_callback_event_counter == 0 && timeout != 0)
    {
       timeout--;
    }
 if (0 == timeout)
    {
       err = FSP_ERR_TIMEOUT;
      assert(FSP_SUCCESS == err);
    }
    uint16_t result_a;
    err = R_ADC_Read(&g_adc_ctrl, OPAMP_EXAMPLE_ADC_CHANNEL, &result_a);
    assert(FSP_SUCCESS == err);
 /* Iterate over the following loop 5 times: */
 /* Call trim() with command OPAMP_TRIM_CMD_NEXT_STEP for the selected channel and
```

```
given input. */
    uint8_t count = OPAMP_TRIM_LOOP_COUNT;
 while (count > 0)
    {
       count--;
       err = R_OPAMP_Trim(&g_opamp_ctrl, OPAMP_TRIM_CMD_NEXT_STEP, trim_args);
      assert(FSP_SUCCESS == err);
 /* Reset the ADC callback counter */
      g_callback_event_counter = 0;
 /* Read converted value after trim completes. */
      err = R_ADC_ScanStart(&g_adc_ctrl);
      assert(FSP_SUCCESS == err);
 /* Wait for ADC scan complete flag */
       timeout = ADC_SCAN_END_MAX_TIMEOUT;
 while (g_callback_event_counter == 0 && timeout != 0)
      {
         timeout--;
      }
 if (0 == timeout)
      {
         err = FSP_ERR_TIMEOUT;
      assert(FSP_SUCCESS == err);
      }
       uint16_t result_b;
       err = R_ADC_Read(&g_adc_ctrl, OPAMP_EXAMPLE_ADC_CHANNEL, &result_b);
      assert(FSP_SUCCESS == err);
 /* Measure the op-amp output using the SAR ADC (referred to as result_b in the next
step). */
 /* If result_a <= result_b, call trim() for the selected channel and input with
command OPAMP_TRIM_CMD_CLEAR_BIT. */
 if (result_a <= result_b)
      {
           err = R_OPAMP_Trim(&g_opamp_ctrl, OPAMP_TRIM_CMD_CLEAR_BIT, trim_args);
      assert(FSP_SUCCESS == err);
```

```
        }
    }
}
```

Data Structures

struct	opamp_extended_cfg_t
struct	opamp_instance_ctrl_t

Macros

#define	OPAMP_MASK_CHANNEL_0

Enumerations

enum	opamp_trigger_t
enum	opamp_agt_link_t
enum	opamp_mode_t
enum	opamp_plus_input_t
enum	opamp_minus_input_t
enum	opamp_output_t

Variables

const opamp_api_t	g_opamp_on_opamp

Data Structure Documentation

◆ opamp_extended_cfg_t

struct opamp_extended_cfg_t

OPAMP configuration extension. This extension is required and must be provided in opamp_cfg_t::p_extend.

Data Fields		
opamp_agt_link_t	agt_link	Configure which AGT links are paired to which channel. Only applies to channels if OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP or OPAMP_TRIGGER_AGT_START_ADC_STOP is selected for the channel.
opamp_mode_t	mode	Low power, middle speed, or high speed mode.

opamp_trigger_t	trigger_channel_0	Start and stop triggers for channel 0.
opamp_trigger_t	trigger_channel_1	Start and stop triggers for channel 1.
opamp_trigger_t	trigger_channel_2	Start and stop triggers for channel 2.
opamp_trigger_t	trigger_channel_3	Start and stop triggers for channel 3.
opamp_plus_input_t	plus_input_select_opamp0	OPAMP0+ connection.
opamp_minus_input_t	minus_input_select_opamp0	OPAMP0- connection.
opamp_output_t	output_select_opamp0	OPAMP0O connection.
opamp_plus_input_t	plus_input_select_opamp1	OPAMP1+ connection.
opamp_minus_input_t	minus_input_select_opamp1	OPAMP1- connection.
opamp_plus_input_t	plus_input_select_opamp2	OPAMP2+ connection.
opamp_minus_input_t	minus_input_select_opamp2	OPAMP2- connection.

◆ opamp_instance_ctrl_t

struct opamp_instance_ctrl_t
OPAMP instance control block. DO NOT INITIALIZE. Initialized in opamp_api_t::open().

Macro Definition Documentation

◆ OPAMP_MASK_CHANNEL_0

#define OPAMP_MASK_CHANNEL_0
Version of code that implements the API defined in this file

Enumeration Type Documentation

◆ opamp_trigger_t

enum opamp_trigger_t	
Start and stop trigger for the op-amp.	
Enumerator	
OPAMP_TRIGGER_SOFTWARE_START_SOFTWARE_STOP	Start and stop with APIs.
OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP	Start by AGT compare match and stop with API.
OPAMP_TRIGGER_AGT_START_ADC_STOP	Start by AGT compare match and stop after ADC conversion.

◆ opamp_agt_link_t

enum opamp_agt_link_t	
Which AGT timer starts the op-amp. Only applies to channels if OPAMP_TRIGGER_AGT_START_SOFTWARE_STOP or OPAMP_TRIGGER_AGT_START_ADC_STOP is selected for the channel. If OPAMP_TRIGGER_SOFTWARE_START_SOFTWARE_STOP is selected for a channel, then no AGT compare match event will start that op-amp channel.	
Enumerator	
OPAMP_AGT_LINK_AGT1_OPAMP_0_2_AGT0_OPAMP_1_3	OPAMP channel 0 and 2 are started by AGT1 compare match. OPAMP channel 1 and 3 are started by AGT0 compare match.
OPAMP_AGT_LINK_AGT1_OPAMP_0_1_AGT0_OPAMP_2_3	OPAMP channel 0 and 1 are started by AGT1 compare match. OPAMP channel 2 and 3 are started by AGT0 compare match.
OPAMP_AGT_LINK_AGT1_OPAMP_0_1_2_3	All OPAMP channels are started by AGT1 compare match.

◆ opamp_mode_t

enum opamp_mode_t	
Op-amp mode.	
Enumerator	
OPAMP_MODE_LOW_POWER	Low power mode.
OPAMP_MODE_MIDDLE_SPEED	Middle speed mode (not supported on all MCUs)
OPAMP_MODE_HIGH_SPEED	High speed mode.

◆ opamp_plus_input_t

enum opamp_plus_input_t	
Options to connect AMPnPS pins.	
Enumerator	
OPAMP_PLUS_INPUT_NONE	No Connection.
OPAMP_PLUS_INPUT_AMPPS0	Set AMPPS0. See hardware manual for channel specific options.
OPAMP_PLUS_INPUT_AMPPS1	Set AMPPS1. See hardware manual for channel specific options.
OPAMP_PLUS_INPUT_AMPPS2	Set AMPPS2. See hardware manual for channel specific options.
OPAMP_PLUS_INPUT_AMPPS3	Set AMPPS3. See hardware manual for channel specific options.
OPAMP_PLUS_INPUT_AMPPS7	Set AMPPS7. See hardware manual for channel specific options.

◆ opamp_minus_input_t

enum opamp_minus_input_t	
Options to connect AMPnMS pins.	
Enumerator	
OPAMP_MINUS_INPUT_NONE	No Connection.
OPAMP_MINUS_INPUT_AMPMS0	Set AMPMS0. See hardware manual for channel specific options.
OPAMP_MINUS_INPUT_AMPMS1	Set AMPMS1. See hardware manual for channel specific options.
OPAMP_MINUS_INPUT_AMPMS2	Set AMPMS2. See hardware manual for channel specific options.
OPAMP_MINUS_INPUT_AMPMS3	Set AMPMS3. See hardware manual for channel specific options.
OPAMP_MINUS_INPUT_AMPMS4	Set AMPMS4. See hardware manual for channel specific options.
OPAMP_MINUS_INPUT_AMPMS7	Set AMPMS7. See hardware manual for channel specific options.

◆ opamp_output_t

enum opamp_output_t	
Options to connect AMP0OS pin.	
Enumerator	
OPAMP_OUTPUT_NONE	No Connection.
OPAMP_OUTPUT_AMPOS0	Set AMPOS0. See hardware manual for channel specific options.
OPAMP_OUTPUT_AMPOS1	Set AMPOS1. See hardware manual for channel specific options.
OPAMP_OUTPUT_AMPOS2	Set AMPOS2. See hardware manual for channel specific options.
OPAMP_OUTPUT_AMPOS3	Set AMPOS3. See hardware manual for channel specific options.

Function Documentation

◆ R_OPAMP_Open()

fsp_err_t R_OPAMP_Open (opamp_ctrl_t *const *p_api_ctrl*, opamp_cfg_t const *const *p_cfg*)

Applies power to the OPAMP and initializes the hardware based on the user configuration. Implements opamp_api_t::open.

The op-amp is not operational until the opamp_api_t::start is called. If the op-amp is configured to start after AGT compare match, the op-amp is not operational until opamp_api_t::start and the associated AGT compare match event occurs.

Some MCUs have switches that must be set before starting the op-amp. These switches must be set in the application code after opamp_api_t::open and before opamp_api_t::start.

Example:

```
/* Initialize the OPAMP module. */
   err = R_OPAMP_Open(&g_opamp_ctrl, &g_opamp_cfg);
```

Return values

FSP_SUCCESS	Configuration successful.
FSP_ERR_ASSERTION	An input pointer is NULL.
FSP_ERR_ALREADY_OPEN	Control block is already opened.
FSP_ERR_INVALID_ARGUMENT	An attempt to configure OPAMP in middle speed mode on MCU that does not support middle speed mode.

◆ R_OPAMP_InfoGet()

fsp_err_t R_OPAMP_InfoGet (opamp_ctrl_t *const *p_api_ctrl*, opamp_info_t *const *p_info*)

Provides the minimum stabilization wait time in microseconds. Implements opamp_api_t::infoGet.

- Example:

```
/* Look up the required stabilization wait time. */
opamp_info_t info;
   err = R_OPAMP_InfoGet(&g_opamp_ctrl, &info);
   assert(FSP_SUCCESS == err);
/* Wait for the OPAMP to stabilize. */
R_BSP_SoftwareDelay(info.min_stabilization_wait_us,
BSP_DELAY_UNITS_MICROSECONDS);
```

Return values

FSP_SUCCESS	information on opamp_power_mode stored in p_info.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_OPAMP_Start()

fsp_err_t R_OPAMP_Start (opamp_ctrl_t *const *p_api_ctrl*, uint32_t const *channel_mask*)

If the OPAMP is configured for hardware triggers, enables hardware triggers. Otherwise, starts the op-amp. Implements opamp_api_t::start.

Some MCUs have switches that must be set before starting the op-amp. These switches must be set in the application code after opamp_api_t::open and before opamp_api_t::start.

Example:

```
/* Start the OPAMP module. */
    err = R_OPAMP_Start(&g_opamp_ctrl, 1 << OPAMP_EXAMPLE_CHANNEL);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Op-amp started or hardware triggers enabled successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.
FSP_ERR_INVALID_ARGUMENT	channel_mask includes a channel that does not exist on this MCU.

◆ R_OPAMP_Stop()

fsp_err_t R_OPAMP_Stop (opamp_ctrl_t *const *p_api_ctrl*, uint32_t const *channel_mask*)

Stops the op-amp. If the OPAMP is configured for hardware triggers, disables hardware triggers. Implements opamp_api_t::stop.

Return values

FSP_SUCCESS	Op-amp stopped or hardware triggers disabled successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.
FSP_ERR_INVALID_ARGUMENT	channel_mask includes a channel that does not exist on this MCU.

◆ R_OPAMP_StatusGet()

fsp_err_t R_OPAMP_StatusGet (opamp_ctrl_t *const *p_api_ctrl*, opamp_status_t *const *p_status*)

Provides the operating status for each op-amp in a bitmask. This bit is set when operation begins, before the stabilization wait time has elapsed. Implements opamp_api_t::statusGet.

Return values

FSP_SUCCESS	Operating status of each op-amp provided in p_status.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_OPAMP_Trim()

fsp_err_t R_OPAMP_Trim (opamp_ctrl_t *const *p_api_ctrl*, opamp_trim_cmd_t const *cmd*, opamp_trim_args_t const *const *p_args*)

On MCUs that support trimming, the op-amp trim register is set to the factory default after open(). This function allows the application to trim the operational amplifier to a user setting, which overwrites the factory default factory trim values.

Not supported on all MCUs. See hardware manual for details. Not supported if configured for low power mode (OPAMP_MODE_LOW_POWER).

This function is not reentrant. Only one side of one op-amp can be trimmed at a time. Complete the procedure for one side of one channel before calling trim() with command OPAMP_TRIM_CMD_START again.

Implements opamp_api_t::trim.

Reference: Section 37.9 "User Offset Trimming" RA2A1 hardware manual R01UM0008EU0130. The trim procedure works as follows:

- Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
- Connect a fixed voltage to the Pch (+) input.
- Connect the Nch (-) input to the op-amp output to create a voltage follower.
- Ensure the op-amp is operating and stabilized.
- Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_START.
- Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
- Iterate over the following loop 5 times:
  - Call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_NEXT_STEP.
  - Measure the op-amp output using the SAR ADC (referred to as B in the next step).
  - If A <= B, call trim() for the Pch (+) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.
- Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_START.
- Measure the fixed voltage connected to the Pch (+) input using the SAR ADC and save the value (referred to as A later in this procedure).
- Iterate over the following loop 5 times:
  - Call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_NEXT_STEP.

- Measure the op-amp output using the SAR ADC (referred to as B in the next step).
- If A <= B, call trim() for the Nch (-) side input with command OPAMP_TRIM_CMD_CLEAR_BIT.

Return values

FSP_SUCCESS	Conversion result in p_data.
FSP_ERR_UNSUPPORTED	Trimming is not supported on this MCU.
FSP_ERR_INVALID_STATE	The command is not valid in the current state of the trim state machine.
FSP_ERR_INVALID_ARGUMENT	The requested channel is not operating or the trim procedure is not in progress for this channel/input combination.
FSP_ERR_INVALID_MODE	Trim is not allowed in low power mode.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_OPAMP_Close()

fsp_err_t R_OPAMP_Close (opamp_ctrl_t *const *p_api_ctrl*)

Stops the op-amps. Implements opamp_api_t::close.

Return values

FSP_SUCCESS	Instance control block closed successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

Variable Documentation

◆ g_opamp_on_opamp

const opamp_api_t g_opamp_on_opamp

OPAMP Implementation of OPAMP interface.

4.2.43 Octa Serial Peripheral Interface for Flash and RAM devices (r_ospi)

Modules

Functions

fsp_err_t	R_OSPI_Open (spi_flash_ctrl_t *p_ctrl, spi_flash_cfg_t const *const p_cfg)
fsp_err_t	R_OSPI_Close (spi_flash_ctrl_t *p_ctrl)
fsp_err_t	R_OSPI_DirectWrite (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write)
fsp_err_t	R_OSPI_DirectRead (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t	R_OSPI_DirectTransfer (spi_flash_ctrl_t *p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction)
fsp_err_t	R_OSPI_SpiProtocolSet (spi_flash_ctrl_t *p_ctrl, spi_flash_protocol_t spi_protocol)
fsp_err_t	R_OSPI_XipEnter (spi_flash_ctrl_t *p_ctrl)
fsp_err_t	R_OSPI_XipExit (spi_flash_ctrl_t *p_ctrl)
fsp_err_t	R_OSPI_Write (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count)
fsp_err_t	R_OSPI_Erase (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_device_address, uint32_t byte_count)
fsp_err_t	R_OSPI_StatusGet (spi_flash_ctrl_t *p_ctrl, spi_flash_status_t *const p_status)
fsp_err_t	R_OSPI_BankSet (spi_flash_ctrl_t *p_ctrl, uint32_t bank)
fsp_err_t	R_OSPI_AutoCalibrate (spi_flash_ctrl_t *p_ctrl)

Detailed Description

Driver for the OSPI peripheral on RA MCUs. This module implements the SPI Flash Interface.

Overview

The OSPI peripheral interfaces with an external OctaFlash and/or OctaRAM chip(s) to perform data I/O Operations. When both OctaFlash and OctaRAM devices are interfaced, they must be connected to their own chip-select lines. The devices cannot share a single chip-select line.

Features

The OSPI driver has the following key features to support the **OctaFlash** device:

- Perform data I/O Operation in both SPI and OPI modes
- Can be configured with OctaFlash device on either of the 2 channels
- Memory mapped read access to the OctaFlash
- Programming the OctaFlash device using single continuous write
- Erasing the OctaFlash device
- Sending device specific commands and reading back responses
- Entering and exiting XIP (Single Continuous Read) mode
- 3 byte addressing for SPI
- 4 byte addressing for SPI and OPI
- Auto-calibration for OPI mode (SOPI and DOPI)

The OSPI driver has the following key features to support the **OctaRAM** device:

- Perform data I/O Operation in DOPI mode
- Can be configured with OctaRAM device on either of the 2 channels
- Memory mapped read amd write access to the OctaRAM using single continuous mode
- Sending device specific commands and reading back responses
- Auto-calibration for DOPI mode
- Uninitialized global variables or buffers can be allocated in the OSPI RAM address space.

Additional build-time features:

- Optional (build-time) DMAC support for data transmission when used with OctaFlash.

Note

For OctaFlash, use of DMAC for data transmission is strongly recommended. Without the use of DMAC, due to the high-speed hardware design of the OSPI peripheral, data transmission can be sensitive to timing variance, which could cause software-based memory-mapped operations to fail unexpectedly.

Configuration

OSPI Flash:

Build Time Configurations for r_ospi

The following build time configurations are defined in driver/r_ospi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DMAC Support	• Enable • Disable	Disable	Enable DMAC support for the OSPI module.

Configurations for Storage > OSPI Flash (r_ospi)

This module can be added to the Stacks tab via New Stack > Storage > OSPI Flash (r_ospi).

Configuration	Options	Default	Description
General > Single Continuous Mode > Read Idle Time	Must be an integer greater than 0 with maximum configurable	100	Specify the read idle time.

	value of 127		
General > Single Continuous Mode > Write Idle Time	Must be an integer greater than 0 with maximum configurable value of 127	100	Specify the write idle time.
General > Name	Name must be a valid C symbol	g_ospi0	Module name.
General > Channel	Channel should be 0 or 1	0	Specify the OSPI chip select line to use.
General > Flash Size	Must be an integer greater than 0 with maximum configurable value of 0x3FFFFFFF	0x04000000	Specify the OctaFlash size in bytes.
General > SPI Protocol	• SPI • Single data rate OPI • Dual data rate OPI	SPI	Select the initial SPI protocol. SPI protocol can be changed on the OctaFlash using R_OSPI_DirectTransfer( ).
General > Address Bytes	• 3 • 4	4	Select the number of address bytes.
OPI Mode > Auto-Calibration > Data latching delay	Must be a valid non-negative integer with maximum configurable value of 0xFF	0x80	Set this to 0 to enable auto-calibration. 0x80 is the default value calculated at 3.3V and 25°C
OPI Mode > Auto-Calibration > Auto-Calibration Address	Must be a valid non-negative integer with maximum configurable value of 0xFFFFFFFF	0x00	Set the address of the read/write destination to be performed for auto-calibration.
OPI Mode > Command Definitions > Page Program Command	Must be a 16-bit OSPI Page Program Command under OPI Mode|Command Definitions	0x12ED	The command to program a page in OPI mode.
OPI Mode > Command Definitions > Read Command	Must be a 16-bit OSPI Read Command under OPI Mode|Command Definitions	0xEC13	The command to read in SOPI mode (8READ).
OPI Mode > Command Definitions > Dual Read Command	Must be a 16-bit OSPI Dual Read Command under OPI Mode|Command Definitions	0xEE11	The command to read in DOPI mode (8DTRD).
OPI Mode > Command Definitions > Write Enable Command	Must be a 16-bit OSPI Write Enable Command under OPI	0x06F9	The command to enable write in OPI mode.

	Mode|Command Definitions		
OPI Mode > Command Definitions > Status Command	Must be a 16-bit OSPI Status Command under OPI Mode|Command Definitions	0x05FA	The command to query the status of a write or erase command in OPI mode.
OPI Mode > OM_DQS Enable Counter > SOPI	Must be an integer between 0 and 255	9	OM_DQS enable counter for memory access. Setting for SOPI mode.
OPI Mode > OM_DQS Enable Counter > DOPI	Must be an integer between 0 and 255	6	OM_DQS enable counter for memory access. Setting for DOPI mode.
OPI Mode > Command Length Bytes	Must be an integer between 1 and 2	2	Command length in bytes
OPI Mode > Memory Read Dummy Cycles	Must be an integer between 6 and 10	10	Memory read dummy cycles
OPI Mode > DOPI Byte Order	• Byte0, Byte1, Byte2, Byte3 • Byte1, Byte0, Byte3, Byte2	Byte0, Byte1, Byte2, Byte3	Byte order on the external bus
SPI Mode > Command Definitions > Page Program Command	Must be a 8-bit OSPI Page Program Command under SPI Mode|Command Definitions	0x12	The command to program a page in SPI mode.
SPI Mode > Command Definitions > Read Command	Must be a 8-bit OSPI Read Command under SPI Mode|Command Definitions	0x13	The command to read in SPI mode.
SPI Mode > Command Definitions > Write Enable Command	Must be a 16-bit OSPI Write Enable Command under SPI Mode|Command Definitions	0x06	The command to enable write in SPI mode.
SPI Mode > Command Definitions > Status Command	Must be a 16-bit OSPI Status Command under SPI Mode|Command Definitions	0x05	The command to query the status of a write or erase command in SPI mode.
Common Command Definitions > Sector Erase Command	Must be a value greater than or equal to 0	0x21DE	The command to erase a sector. Set Sector Erase Size to 0 if unused.
Common Command Definitions > Block Erase Command	Must be a value greater than or equal to 0	0xDC23	The command to erase a block. Set Block Erase Size to 0 if

			unused.
Common Command Definitions > Chip Erase Command	Must be a value greater than or equal to 0	0xC738	The command to erase the entire chip. Set Chip Erase Command to 0 if unused.
Common Command Definitions > Write Status Bit	Must be an integer between 0 and 7	0	Which bit contains the write in progress status returned from the Write Status Command.
Common Command Definitions > Write Enable Bit	Must be an integer between 0 and 7	1	Which bit contains the write enable status returned from the Write Enable Command.
Common Command Definitions > Sector Erase Size	Must be an integer greater than or equal to 0	4096	The sector erase size. Set Sector Erase Size to 0 if Sector Erase is not supported.
Common Command Definitions > Block Erase Size	Must be an integer greater than or equal to 0	65536	The block erase size. Set Block Erase Size to 0 if Block Erase is not supported.
Chip Select Timing Setting > Memory Mapped Read Command Interval	• 2 • 5 • 7 • 9 • 11 • 13 • 15 • 17	2	Memory mapped read command execution interval setting in OCTACLK units
Chip Select Timing Setting > Memory Mapped Write Command Interval	• 2 • 5 • 7 • 9 • 11 • 13 • 15 • 17	2	Memory mapped write command execution interval setting in OCTACLK units
Chip Select Timing Setting > Command Interval	• 2 • 5 • 7 • 9 • 11 • 13 • 15 • 17	2	Command execution interval setting in OCTACLK units
Chip Select Timing Setting > Memory	• 5 SPI/SOPI • 6 SPI/SOPI	5 SPI/SOPI	Memory mapped read signal pull-up timing

Mapped Read Pull-up Timing	• 7 SPI/SOPI, 6.5 DOPI • 8 SPI/SOPI, 7.5 DOPI • 9 SPI/SOPI, 8.5 DOPI		setting in OCTACLK units
Chip Select Timing Setting > Memory Mapped Write Pull-up Timing	• 2 SPI/SOPI, 1.5 DOPI • 3 SPI/SOPI, 2.5 DOPI • 4 SPI/SOPI, 3.5 DOPI • 5 SPI/SOPI, 4.5 DOPI • 6 SPI/SOPI, 5.5 DOPI • 7 SPI/SOPI, 6.5 DOPI • 8 SPI/SOPI, 7.5 DOPI • 9 SPI/SOPI, 8.5 DOPI	2 SPI/SOPI, 1.5 DOPI	Memory mapped write signal pull-up timing setting in OCTACLK units
Chip Select Timing Setting > Pull-up Timing	• 5 SPI/SOPI • 6 SPI/SOPI • 7 SPI/SOPI, 6.5 DOPI • 8 SPI/SOPI, 7.5 DOPI • 9 SPI/SOPI, 8.5 DOPI	5 SPI/SOPI	Signal pull-up timing setting in OCTACLK units
Chip Select Timing Setting > Memory Mapped Read Pull-down Timing	• 3 SPI/SOPI, 2.5 DOPI • 4 SPI/SOPI, 3.5 DOPI • 5 SPI/SOPI, 4.5 DOPI	3 SPI/SOPI, 2.5 DOPI	Memory mapped read signal pull-down timing setting in OCTACLK units
Chip Select Timing Setting > Memory Mapped Write Pull-down Timing	• 3 SPI/SOPI, 2.5 DOPI • 4 SPI/SOPI, 3.5 DOPI • 5 SPI/SOPI, 4.5 DOPI	3 SPI/SOPI, 2.5 DOPI	Memory mapped write signal pull-down timing setting in OCTACLK units
Chip Select Timing Setting > Pull-down Timing	• 3 SPI/SOPI, 2.5 DOPI • 4 SPI/SOPI, 3.5 DOPI • 5 SPI/SOPI, 4.5 DOPI	3 SPI/SOPI, 2.5 DOPI	Signal pull-down timing setting in OCTACLK units

Note

The user is expected to modify the command definitions based on the OctaFlash chip and SPI communication mode. The default mode is SPI mode and default erase commands are set for OPI mode based on Macronix

OctaFlash MX25LM51245G.

OSPI RAM:

Build Time Configurations for r_ospi

The following build time configurations are defined in driver/r_ospi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Storage > OSPI RAM (r_ospi)

This module can be added to the Stacks tab via New Stack > Storage > OSPI RAM (r_ospi).

Configuration	Options	Default	Description
General > Single Continuous Mode > Read Idle Time	Must be an integer greater than 0 with maximum configurable value of 127	127	Specify the read idle time.
General > Single Continuous Mode > Write Idle Time	Must be an integer greater than 0 with maximum configurable value of 127	127	Specify the write idle time.
General > Name	Name must be a valid C symbol	g_ospi_ram0	Module name.
General > Channel	Channel should be 0 or 1 [Channel 0 recommended]	0	Specify the OSPI chip select line to use.
General > RAM Size	Must be an integer greater than 0 with maximum configurable value of 0x00800000	0x00800000	Specify the OctaRam size in bytes.
General > SPI Protocol	Dual data rate OPI	Dual data rate OPI	Select the initial SPI protocol. OctaRAM only supports DOPI mode.
General > Address Bytes	4	4	Select the number of address bytes. OctaRAM only supports 4 byte addresses in DOPI mode.
Auto-Calibration > Data latching delay	Must be a valid non-negative integer with maximum configurable value of 0xFF	0x80	Set this to 0 to enable auto-calibration. 0x80 is the default value calculated at 3.3V and 25°C

Auto-Calibration > Auto-Calibration Address	Must be a valid non-negative integer with maximum configurable value of 0xFFFFFFFF	0x00	Set the address of the read/write destination to be performed for auto-calibration.
Command Definitions > Memory Read Command	Must be a 16-bit OSPI Dual Read Command under Command Definitions	0xA000	The command to read in DOPI mode.
Command Definitions > Memory Write Command	Must be a 16-bit OSPI Write Command under Command Definitions	0x2000	The command to write in DOPI mode.
OM_DQS Enable Counter > DOPI	Must be an integer between 0 and 255	3	OM_DQS enable counter for memory access. Setting for DOPI mode.
Chip Select Timing Setting > Memory Mapped Read Command Interval	• 2 • 5 • 7 • 9 • 11 • 13 • 15 • 17	2	Memory mapped read command execution interval setting in OCTACLK units
Chip Select Timing Setting > Memory Mapped Write Command Interval	• 2 • 5 • 7 • 9 • 11 • 13 • 15 • 17	2	Memory mapped write command execution interval setting in OCTACLK units
Chip Select Timing Setting > Command Interval	• 2 • 5 • 7 • 9 • 11 • 13 • 15 • 17	2	Command execution interval setting in OCTACLK units
Chip Select Timing Setting > Memory Mapped Read Pull-up Timing	• 6.5 DOPI • 7.5 DOPI • 8.5 DOPI	6.5 DOPI	Memory mapped read signal pull-up timing setting in OCTACLK units
Chip Select Timing Setting > Memory Mapped Write Pull-up Timing	• 1.5 DOPI • 2.5 DOPI • 3.5 DOPI • 4.5 DOPI • 5.5 DOPI • 6.5 DOPI • 7.5 DOPI	1.5 DOPI	Memory mapped write signal pull-up timing setting in OCTACLK units

	• 8.5 DOPI		
Chip Select Timing Setting > Pull-up Timing	• 6.5 DOPI • 7.5 DOPI • 8.5 DOPI	6.5 DOPI	Signal pull-up timing setting in OCTACLK units
Chip Select Timing Setting > Memory Mapped Read Pull-down Timing	• 2.5 DOPI • 3.5 DOPI • 4.5 DOPI	2.5 DOPI	Memory mapped read signal pull-down timing setting in OCTACLK units
Chip Select Timing Setting > Memory Mapped Write Pull-down Timing	• 2.5 DOPI • 3.5 DOPI • 4.5 DOPI	2.5 DOPI	Memory mapped write signal pull-down timing setting in OCTACLK units
Chip Select Timing Setting > Pull-down Timing	• 2.5 DOPI • 3.5 DOPI • 4.5 DOPI	2.5 DOPI	Signal pull-down timing setting in OCTACLK units
Command Length Bytes	Must be an integer between 1 and 2	2	Command length in bytes
Memory Read Dummy Cycles	Must be an integer between 3 and 8	4	Memory read dummy cycles
Memory Write Dummy Cycles	Must be an integer between 3 and 8	4	Memory write dummy cycles
DOPI Byte Order	• Byte0, Byte1, Byte2, Byte3 • Byte1, Byte0, Byte3, Byte2	Byte1, Byte0, Byte3, Byte2	Byte order on the external bus
Chip Select Maximum Low Time (us)	Must be an integer greater than or equal to 0	4	Chip Select Maximum Low Time (tCSM).

Clock Configuration

PCLKB is the Octal-SPI bus interface, and PCLKA is used to set OSPI registers.

The signals to the OSPI device are derived from OCTASPICLK. The OMSCLK signal is OCTASPICLK / 2. Data can be output at the OCTASPICLK rate if SPI Protocol is set to Dual Data Rate OPI.

The PCLKB, PCLKA, and OCTASPICLK frequencies can be set on the **Clocks** tab of the RA Configuration editor.

Pin Configuration

The following pins are available to connect to an external OSPI device:

- OMSCLK: OSPI clock output (OCTASPICLK / 2)
- OMDQS: OSPI data strobe signal
- OMCS0: OSPI device 0 select
- OMCS1: OSPI device 1 select
- OMSIO0: Data 0 I/O
- OMSIO1: Data 1 I/O

- OMSIO2: Data 2 I/O
- OMSIO3: Data 3 I/O
- OMSIO4: Data 4 I/O
- OMSIO5: Data 5 I/O
- OMSIO6: Data 6 I/O
- OMSIO7: Data 7 I/O

Note

Data pins must be configured with IOPORT_CFG_DRIVE_HS_HIGH.
Chip Select pins should be configured with at least IOPORT_CFG_DRIVE_MEDIUM.

Usage Notes

Usage Notes for OctaFlash support

Enabling DMAC

DMAC data transmission support is configurable for OSPI Flash and is disabled from the build by default. Use of a high-priority (low channel number) DMAC for data transmission is strongly recommended.

For further details on DMAC please refer Direct Memory Access Controller (r_dmac).

OSPI Memory Mapped Access

After R_OSPI_Open() completes successfully, the OctaFlash device contents are mapped to address 0x68000000 (channel 0) or 0x70000000 (channel 1) based on the channel configured and can be read like on-chip flash. Channel 0 supports 128 MB while Channel 1 supports 256 MB of address space.

Auto-calibration

Auto-calibration procedure is triggered automatically when the 'Data latching delay' field in the configurator properties is set to 0. The user application is responsible for setting the appropriate preamble pattern before calling R_OSPI_Open() with SOPI/DOPI mode or changing the SPI protocol to SOPI/DOPI using R_OSPI_SpiProtocolSet() API. The appropriate preamble pattern can be written to the desired address using the R_OSPI_Write() API while in the SPI mode (recommended). Ensure that the same address is passed through the configurator. If the OctaFlash chip is already in SOPI/DOPI mode, the preamble pattern must be programmed using the debugger before calling R_OSPI_Open().

Chip Select Latencies

Chip select latencies can be set through the configurator. The default settings support SOPI and SPI at minimum latency. In case the driver is opened in SPI mode and will be switched to DOPI mode later using R_OSPI_SpiProtocolSet(), please select latencies required for DOPI before calling R_OSPI_Open().

OctaFlash Commands

- Set the erase commands based on intended mode of operation (SPI or OPI). These commands cannot be changed during run-time.
- Read, Write and Status commands for both SPI and OPI are configured allowing switching between the modes at run-time.

Usage Notes for OctaRAM support

OSPI Memory Mapped Access

After R_OSPI_Open() completes successfully, the OctaRAM device contents are mapped to address 0x68000000 (channel 0) or 0x70000000 (channel 1) based on the channel configured and can be written to or read from like on-chip RAM. Channel 0 and 1 support 8 MB of address space.

Auto-calibration

Since the OctaRAM only supports DOPI mode, the driver allows the user to call R_OSPI_Open() without performing the auto-calibration procedure automatically when 'Data latching delay' field is set to 0 in the configurator properties. This is done so that the user can write the appropriate preamble pattern to the desired address using memory mapped writes while in DOPI mode. Ensure that the same address is passed through the configurator. R_OSPI_AutoCalibrate() should be then called to perform auto-calibration.

Chip Select Latencies

Chip select latencies can be set through the configurator. The default settings support DOPI at minimum latency.

Limitations

Developers should be aware of the following limitations when using the OSPI driver:

OctaFlash

- Single continuous read in SPI mode is not supported by the peripheral. The maximum amount of data that can be read using a single read command is 4-bytes (When doing a 32-bit access).
- Fast Reads would be slower than regular reads as the SPI mode cannot be operated with an OMSCLK greater than 50MHz.

Examples

OSPI Flash:

Basic Example

This is a basic example of minimal use of the OSPI in an application with OctaFlash.

```
#define OSPI_EXAMPLE_DATA_LENGTH (1024)
uint8_t g_dest[OSPI_EXAMPLE_DATA_LENGTH];
/* Place data in the .ospi_flash section to flash it during programming. */
const uint8_t g_src[OSPI_EXAMPLE_DATA_LENGTH] BSP_PLACE_IN_SECTION(".ospi_flash") =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
/* Place code in the .code_in_ospi section to flash it during programming. */
void r_ospi_example_function(void) BSP_PLACE_IN_SECTION(".code_in_ospi")
__attribute__((noinline));
void r_ospi_example_function (void)
```

```
{
 /* Add code here. */
}
void r_ospi_basic_example (void)
{
 /* Open the OSPI instancee */
 fsp_err_t err = R_OSPI_Open(&g_ospi0_ctrl, &g_ospi0_cfg);
    assert(FSP_SUCCESS == err);
 /* (Optional) Change SPI to DOPI mode */
    r_ospi_example_spi_to_dopi();
 /* After R_OSPI_Open() and any required device specific intiialization, data can be
read directly from the OSPI flash. */
    memcpy(&g_dest[0], &g_src[0], OSPI_EXAMPLE_DATA_LENGTH);
 /* After R_OSPI_Open() and any required device specific intiialization, functions in
the OSPI flash can be called. */
    r_ospi_example_function();
}
```

Reading Status Register Example (R_OSPI_DirectTransfer)

This is an example of using R_OSPI_DirectWrite followed by R_OSPI_DirectRead to send the read status register command and read back the status register from the device.

```
#define OSPI_COMMAND_READ_STATUS_REGISTER (0x05U)
void r_ospi_direct_example (void)
{
 spi_flash_direct_transfer_t ospi_test_direct_transfer =
    {
       .command        = OSPI_TEST_READ_STATUS_COMMAND_SPI_MODE,
       .address        = 0U,
       .data           = 0U,
       .command_length = 1U,
       .address_length = 0U,
       .data_length    = 0U,
       .dummy_cycles   = 0U
```

```
    };
 /* Open the OSPI instance. */
 fsp_err_t err = R_OSPI_Open(&g_ospi0_ctrl, &g_ospi0_cfg);
    assert(FSP_SUCCESS == err);
 /* Write Enable */
    err = R_OSPI_DirectTransfer(&g_ospi0_ctrl, &ospi_test_direct_transfer,
SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
 /* Read Status Register */
    ospi_test_direct_transfer.command    = OSPI_TEST_READ_STATUS_COMMAND_SPI_MODE;
    ospi_test_direct_transfer.data_length = 1U;
    err = R_OSPI_DirectTransfer(&g_ospi0_ctrl, &ospi_test_direct_transfer,
SPI_FLASH_DIRECT_TRANSFER_DIR_READ);
    assert(FSP_SUCCESS == err);
 /* Check if Write Enable is set */
 if (OSPI_WEN_BIT_MASK != (ospi_test_direct_transfer.data & OSPI_WEN_BIT_MASK))
    {
       __BKPT(0);
    }
}
```

Auto-calibration Example (R_OSPI_DirectTransfer, R_OSPI_Write, R_OSPI_SpiProtocolSet)

This is an example of using R_OSPI_SpiProtocolSet to change the operating mode from SPI to SOPI and allow the driver to initiate auto-calibration.

```
#define OSPI_DOPI_PREAMBLE_PATTERN_LENGTH_BYTES (16U)
#define OSPI_EXAMPLE_PREAMBLE_ADDRESS (0x68000000U) /* Device connected to CS0 */
const uint8_t g_preamble_bytes[OSPI_DOPI_PREAMBLE_PATTERN_LENGTH_BYTES] =
{
    0x00, 0x00, 0xFF, 0xFF, 0xFF, 0x00, 0x08, 0x00, 0x00, 0xF7, 0xFF, 0x00, 0x08,
0xF7, 0x00, 0xF7
};
void ospi_example_wait_until_wip (void)
{
```

```
 fsp_err_t          err = FSP_SUCCESS;
 spi_flash_status_t status;
    status.write_in_progress = true;
    uint32_t timeout = UINT32_MAX;
 while ((status.write_in_progress) && (--timeout))
    {
       err = R_OSPI_StatusGet(&g_ospi0_ctrl, &status);
      assert(FSP_SUCCESS == err);
    }
 if (0 == timeout)
    {
      assert(FSP_SUCCESS == err);
    }
}
void r_ospi_auto_calibrate_example (void)
{
 /* Open the OSPI instance. */
 /* Set data_latch_delay_clocks to 0x0 to enable auto-calibration */
 fsp_err_t err = R_OSPI_Open(&g_ospi0_ctrl, &g_ospi0_cfg);
    assert(FSP_SUCCESS == err);
    uint8_t * preamble_pattern_addr = (uint8_t *) OSPI_EXAMPLE_PREAMBLE_ADDRESS;
    err = R_OSPI_Write(&g_ospi0_ctrl, g_preamble_bytes, preamble_pattern_addr,
OSPI_EXAMPLE_PREAMBLE_ADDRESS);
    assert(FSP_SUCCESS == err);
 /* Wait until write has been completed */
    ospi_example_wait_until_wip();
 /* Change from SPI to DOPI mode */
    r_ospi_example_spi_to_dopi();
}
```

Octaclk Update Example (R_OSPI_SpiProtocolSet)

This is an example of using R_BSP_OctaclkUpdate to change the Octal-SPI clock frequency during run time. The OCTACLK frequency must be updated before calling the R_OSPI_SpiProtocolSet with appropriate clock source and divider settings required to be set for the new SPI protocol mode. Ensure that the clock source selected is started.

```
static void ospi_example_change_omclk (void)
{
 /* Ensure clock source (PLL2 in this example) is running before changing the OCTACLK
frequency */
    bsp_octaclk_settings_t octaclk_settings;
    octaclk_settings.source_clock = BSP_CLOCKS_CLOCK_PLL2;
    octaclk_settings.divider      = BSP_CLOCKS_OCTACLK_DIV_2;
    R_BSP_OctaclkUpdate(&octaclk_settings);
}
```

OSPI Data and IAR

When using the IAR compiler, OSPI data must be const qualified to be downloaded by the debugger.

OSPI RAM:

Basic Example

This is a basic example of minimal use of the OSPI in an application with OctaRAM.

```
#define OSPI_RAM_EXAMPLE_DATA_LENGTH (1024)
uint8_t g_dest[OSPI_RAM_EXAMPLE_DATA_LENGTH];
/* Place uninitialized data buffers in the ospi_device_0_no_load section.
 * Use ospi_device_1_no_load section if the OctaRAM is configured on channel 1.
 */
uint8_t g_src_1[OSPI_RAM_EXAMPLE_DATA_LENGTH]
BSP_PLACE_IN_SECTION(".ospi_device_0_no_load");
uint8_t g_src_2[OSPI_RAM_EXAMPLE_DATA_LENGTH]
BSP_PLACE_IN_SECTION(".ospi_device_0_no_load");
void r_ospi_ram_basic_example (void)
{
 /* Open the OSPI instancee.
  * Ensure valid setting of the 'Data latching delay' field in the configurator.
  * To successfully perform OSPI RAM reads this value must not be 0.
  */
 fsp_err_t err = R_OSPI_Open(&g_ospi_ram0_ctrl, &g_ospi_ram0_cfg);
    assert(FSP_SUCCESS == err);
 /* After R_OSPI_Open() and any required device specific intiialization, data can be
```

```
read from or written to directly from the OSPI RAM. */
    memcpy(&g_dest[0], &g_src_1[0], OSPI_RAM_EXAMPLE_DATA_LENGTH);
    memcpy(&g_src_2[0], &g_src_1[0], OSPI_RAM_EXAMPLE_DATA_LENGTH);
}
```

Auto-calibration Example (R_OSPI_DirectTransfer, R_OSPI_AutoCalibrate)

This is an example of using R_OSPI_AutoCalibrate to calibrate OSPI peripheral to read data from the OctaRAM device.

```
#define OSPI_RAM_EXAMPLE_PREAMBLE_ADDRESS (0x68000000U) /* Device connected to CS0 */
#define OSPI_RAM_EXAMPLE_OCTARAM_CR_LATENCY_COUNTER_MASK (0x00F0U)
#define OSPI_RAM_EXAMPLE_OCTARAM_CR_LATENCY_COUNTER_POS (4U)
#define OSPI_RAM_EXAMPLE_OCTARAM_100MHZ_4CLOCKS_CR_SETTING (1U)
void r_ospi_ram_auto_calibrate_example (void)
{
 /* Open the OSPI instancee */
 fsp_err_t err = R_OSPI_Open(&g_ospi_ram0_ctrl, &g_ospi_ram0_cfg);
    assert(FSP_SUCCESS == err);
 /* OctaRAM Configuration Register (cr) read and write command definition */
 spi_flash_direct_transfer_t read_cr =
    {
       .command        = 0xC000U,    // NOLINT(readability-magic-numbers)
       .address        = 0x00040000U, // NOLINT(readability-magic-numbers)
       .data           = 0U,
       .command_length = 2U,
       .address_length = 4U,
       .data_length    = 2U,
 /* Dummy Cycles set to the default value specified in the OctaRAM device
Configuration Register */
       .dummy_cycles   = 5U
    };
 spi_flash_direct_transfer_t write_cr =
    {
       .command        = 0x4000U,    // NOLINT(readability-magic-numbers)
```

```
       .address       = 0x00040000U, // NOLINT(readability-magic-numbers)
       .data          = 0U,
       .command_length = 2U,
       .address_length = 4U,
       .data_length    = 2U,
       .dummy_cycles   = 0U
    };
 /* Read OctaRAM device Configuration Register */
    err = R_OSPI_DirectTransfer(&g_ospi_ram0_ctrl, &read_cr,
SPI_FLASH_DIRECT_TRANSFER_DIR_READ);
    assert(FSP_SUCCESS == err);
    uint16_t config_reg = (uint16_t) (((uint16_t) (read_cr.data) &
~OSPI_RAM_EXAMPLE_OCTARAM_CR_LATENCY_COUNTER_MASK) |
                                     ((uint16_t)
(OSPI_RAM_EXAMPLE_OCTARAM_100MHZ_4CLOCKS_CR_SETTING <<
                                                    OSPI_RAM_EXAMPLE_OCTARAM_CR_LATENC
Y_COUNTER_POS) &
                                       OSPI_RAM_EXAMPLE_OCTARAM_CR_LATENCY_COUNTER_MA
SK));
 /* Write Configuration Register */
    write_cr.data = (uint32_t) config_reg;
    err          = R_OSPI_DirectTransfer(&g_ospi_ram0_ctrl, &write_cr,
SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
    assert(FSP_SUCCESS == err);
    read_cr.data = 0;
 /* Set Dummy Clocks to value configured above (4 Clocks) */
    read_cr.dummy_cycles = 4U;
 /* Read Configuration Register */
    err = R_OSPI_DirectTransfer(&g_ospi_ram0_ctrl, &read_cr,
SPI_FLASH_DIRECT_TRANSFER_DIR_READ);
    assert(FSP_SUCCESS == err);
 /* Confirm the intended Configuration Register value */
    assert(config_reg == (uint16_t) (read_cr.data & UINT16_MAX));
 volatile uint32_t * ram_addr = (uint32_t *) OSPI_RAM_EXAMPLE_PREAMBLE_ADDRESS;
```

```
 /* Write the auto-calibration preamble pattern for DOPI mode as specified by the
Hardware Manual */
    ram_addr[0] = 0xFFFF0000;          // NOLINT(readability-magic-numbers)
    ram_addr[1] = 0x0800FF00;          // NOLINT(readability-magic-numbers)
    ram_addr[2] = 0xFF0000F7;          // NOLINT(readability-magic-numbers)
    ram_addr[3] = 0x00F708F7;          // NOLINT(readability-magic-numbers)
    err = R_OSPI_AutoCalibrate(&g_ospi_ram0_ctrl);
    assert(FSP_SUCCESS == err);
 /* After Auto-calibration data can be read from or written to directly from the OSPI
RAM. */
    memcpy(&g_dest[0], &g_src_1[0], OSPI_RAM_EXAMPLE_DATA_LENGTH);
    memcpy(&g_src_2[0], &g_src_1[0], OSPI_RAM_EXAMPLE_DATA_LENGTH);
}
```

Data Structures

struct	ospi_instance_ctrl_t

Enumerations

enum	ospi_device_number_t
enum	ospi_device_type_t
enum	ospi_command_cs_pullup_clocks_t
enum	ospi_command_cs_pulldown_clocks_t
enum	ospi_dopi_byte_order_t

Data Structure Documentation

◆ ospi_instance_ctrl_t

struct ospi_instance_ctrl_t
Instance control block. DO NOT INITIALIZE. Initialization occurs when spi_flash_api_t::open is called

Enumeration Type Documentation

◆ ospi_device_number_t

enum ospi_device_number_t	
Enumerator	
OSPI_DEVICE_NUMBER_0	Device connected to Chip-Select 0.
OSPI_DEVICE_NUMBER_1	Device connected to Chip-Select 1.

◆ ospi_device_type_t

enum ospi_device_type_t	
Enumerator	
OSPI_DEVICE_FLASH	Device Memory type OctaFlash.
OSPI_DEVICE_RAM	Device Memory type OctaRAM.

◆ ospi_command_cs_pullup_clocks_t

enum ospi_command_cs_pullup_clocks_t	
Enumerator	
OSPI_COMMAND_CS_PULLUP_CLOCKS_2	1.5 clocks DOPI mode; 2 Clocks all other modes; Unsupported for DOPI Read
OSPI_COMMAND_CS_PULLUP_CLOCKS_3	2.5 clocks DOPI mode; 3 Clocks all other modes; Unsupported for DOPI Read
OSPI_COMMAND_CS_PULLUP_CLOCKS_4	3.5 clocks DOPI mode; 4 Clocks all other modes; Unsupported for DOPI Read
OSPI_COMMAND_CS_PULLUP_CLOCKS_5	4.5 clocks DOPI mode; 5 Clocks all other modes; Unsupported for DOPI Read
OSPI_COMMAND_CS_PULLUP_CLOCKS_6	5.5 clocks DOPI mode; 6 Clocks all other modes; Unsupported for DOPI Read
OSPI_COMMAND_CS_PULLUP_CLOCKS_7	6.5 clocks DOPI mode; 7 Clocks all other modes
OSPI_COMMAND_CS_PULLUP_CLOCKS_8	7.5 clocks DOPI mode; 8 Clocks all other modes
OSPI_COMMAND_CS_PULLUP_CLOCKS_9	8.5 clocks DOPI mode; 9 Clocks all other modes

◆ ospi_command_cs_pulldown_clocks_t

enum ospi_command_cs_pulldown_clocks_t	
Enumerator	
OSPI_COMMAND_CS_PULLDOWN_CLOCKS_3	2.5 clocks DOPI mode; 3 Clocks all other modes
OSPI_COMMAND_CS_PULLDOWN_CLOCKS_4	3.5 clocks DOPI mode; 4 Clocks all other modes
OSPI_COMMAND_CS_PULLDOWN_CLOCKS_5	4.5 clocks DOPI mode; 5 Clocks all other modes

◆ ospi_dopi_byte_order_t

enum ospi_dopi_byte_order_t	
Enumerator	
OSPI_DOPI_BYTE_ORDER_0123	DOPI byte order byte 0, byte 1, byte 2, byte 3.
OSPI_DOPI_BYTE_ORDER_1032	DOPI byte order byte 1, byte 0, byte 3, byte 2.

Function Documentation

◆ R_OSPI_Open()

fsp_err_t R_OSPI_Open (spi_flash_ctrl_t * *p_ctrl*, spi_flash_cfg_t const *const *p_cfg*)

Open the OSPI driver module. After the driver is open, the OSPI can be accessed like internal flash memory.

Implements spi_flash_api_t::open.

Example:

```
/* Open the OSPI instancee */
fsp_err_t err = R_OSPI_Open(&g_ospi0_ctrl, &g_ospi0_cfg);
```

Return values

FSP_SUCCESS	Configuration was successful.
FSP_ERR_ASSERTION	The parameter p_ctrl or p_cfg is NULL.
FSP_ERR_ALREADY_OPEN	Driver has already been opened with the same p_ctrl.
FSP_ERR_CALIBRATE_FAILED	Failed to perform auto-calibrate.
FSP_ERR_INVALID_ARGUMENT	Attempting to open the driver with an invalid SPI protocol for OctaRAM.

◆ R_OSPI_Close()

fsp_err_t R_OSPI_Close (spi_flash_ctrl_t * *p_ctrl*)

Close the OSPI driver module.

Implements spi_flash_api_t::close.

Return values

FSP_SUCCESS	Configuration was successful.
FSP_ERR_ASSERTION	p_instance_ctrl is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.

◆ R_OSPI_DirectWrite()

fsp_err_t R_OSPI_DirectWrite (spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*, bool const *read_after_write*)

Writes raw data directly to the OctaFlash. API not supported. Use R_OSPI_DirectTransfer

Implements spi_flash_api_t::directWrite.

Return values

FSP_ERR_UNSUPPORTED	API not supported by OSPI.

◆ R_OSPI_DirectRead()

fsp_err_t R_OSPI_DirectRead (spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Reads raw data directly from the OctaFlash. API not supported. Use R_OSPI_DirectTransfer.

Implements spi_flash_api_t::directRead.

Return values

FSP_ERR_UNSUPPORTED	API not supported by OSPI.

◆ R_OSPI_DirectTransfer()

fsp_err_t R_OSPI_DirectTransfer (spi_flash_ctrl_t * *p_ctrl*, spi_flash_direct_transfer_t *const *p_transfer*, spi_flash_direct_transfer_dir_t *direction*)

Read/Write raw data directly with the OctaFlash/OctaRAM device.

Implements spi_flash_api_t::directTransfer.

Example:

```
/* Write Enable */
   err = R_OSPI_DirectTransfer(&g_ospi0_ctrl, &ospi_test_direct_transfer,
SPI_FLASH_DIRECT_TRANSFER_DIR_WRITE);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	The flash was programmed successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.

◆ R_OSPI_SpiProtocolSet()

fsp_err_t R_OSPI_SpiProtocolSet (spi_flash_ctrl_t * *p_ctrl*, spi_flash_protocol_t *spi_protocol*)

Sets the SPI protocol.

Implements spi_flash_api_t::spiProtocolSet.

Return values

FSP_SUCCESS	SPI protocol updated on MCU peripheral.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_CALIBRATE_FAILED	Failed to perform auto-calibrate.
FSP_ERR_INVALID_ARGUMENT	Attempting to set an invalid SPI protocol for OctaRAM.

◆ R_OSPI_XipEnter()

fsp_err_t R_OSPI_XipEnter (spi_flash_ctrl_t * *p_ctrl*)

Enters Single Continuous Read/Write mode.

Implements spi_flash_api_t::xipEnter.

Return values

FSP_SUCCESS	The flash was programmed successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_UNSUPPORTED	API not supported by OSPI - OctaRAM.

◆ R_OSPI_XipExit()

fsp_err_t R_OSPI_XipExit (spi_flash_ctrl_t * *p_ctrl*)

Exits XIP (execute in place) mode.

Implements spi_flash_api_t::xipExit.

Return values

FSP_SUCCESS	The flash was programmed successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_UNSUPPORTED	API not supported by OSPI - OctaRAM.

◆ R_OSPI_Write()

fsp_err_t R_OSPI_Write (spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint8_t *const *p_dest*, uint32_t *byte_count*)

Program a page of data to the flash.

Implements spi_flash_api_t::write.

Example:

```
    err = R_OSPI_Write(&g_ospi0_ctrl, g_preamble_bytes, preamble_pattern_addr,
OSPI_EXAMPLE_PREAMBLE_ADDRESS);
    assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	The flash was programmed successfully.
FSP_ERR_ASSERTION	p_instance_ctrl, p_dest or p_src is NULL, or byte_count crosses a page boundary.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_DEVICE_BUSY	Another Write/Erase transaction is in progress.
FSP_ERR_INVALID_SIZE	Write operation crosses page-boundary.
FSP_ERR_UNSUPPORTED	API not supported by OSPI - OctaRAM.
FSP_ERR_WRITE_FAILED	The write enable bit was not set.

◆ R_OSPI_Erase()

fsp_err_t R_OSPI_Erase (spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_device_address*, uint32_t *byte_count*)

Erase a block or sector of flash. The byte_count must exactly match one of the erase sizes defined in spi_flash_cfg_t. For chip erase, byte_count must be SPI_FLASH_ERASE_SIZE_CHIP_ERASE.

Implements spi_flash_api_t::erase.

Return values

FSP_SUCCESS	The command to erase the flash was executed successfully.
FSP_ERR_ASSERTION	p_instance_ctrl or p_device_address is NULL, byte_count doesn't match an erase size defined in spi_flash_cfg_t, or byte_count is set to 0.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_DEVICE_BUSY	The device is busy.
FSP_ERR_UNSUPPORTED	API not supported by OSPI - OctaRAM.
FSP_ERR_WRITE_FAILED	The write enable bit was not set.

◆ R_OSPI_StatusGet()

fsp_err_t R_OSPI_StatusGet (spi_flash_ctrl_t * *p_ctrl*, spi_flash_status_t *const *p_status*)

Gets the write or erase status of the flash.

Implements spi_flash_api_t::statusGet.

Example:

```
err = R_OSPI_StatusGet(&g_ospi0_ctrl, &status);
assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	The write status is in p_status.
FSP_ERR_ASSERTION	p_instance_ctrl or p_status is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_UNSUPPORTED	API not supported by OSPI - OctaRAM.

◆ R_OSPI_BankSet()

fsp_err_t R_OSPI_BankSet (spi_flash_ctrl_t * *p_ctrl*, uint32_t *bank*)

Selects the bank to access.

Implements spi_flash_api_t::bankSet.

Return values

FSP_ERR_UNSUPPORTED	API not supported by OSPI.

◆ R_OSPI_AutoCalibrate()

fsp_err_t R_OSPI_AutoCalibrate (spi_flash_ctrl_t * *p_ctrl*)

Auto-calibrate the OctaRAM device using the preamble pattern.

Note

The preamble pattern must be written to the configured address before calling this API. Implements spi_flash_api_t::autoCalibrate.

Return values

FSP_SUCCESS	SPI protocol updated on MCU peripheral.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_CALIBRATE_FAILED	Failed to perform auto-calibrate.
FSP_ERR_UNSUPPORTED	API not supported by OSPI - OctaFlash.

4.2.44 Parallel Data Capture (r_pdc)

Modules

Functions

fsp_err_t	R_PDC_Open (pdc_ctrl_t *const p_api_ctrl, pdc_cfg_t const *const p_cfg) Powers on PDC, handles required initialization described in the hardware manual. More...
fsp_err_t	R_PDC_Close (pdc_ctrl_t *const p_api_ctrl) Stops and closes the transfer interface, disables and powers off the PDC, clears internal driver data and disables interrupts. More...

fsp_err_t	R_PDC_CaptureStart (pdc_ctrl_t *const p_api_ctrl, uint8_t *const p_buffer) Starts a capture. Enables interrupts. More...

Detailed Description

Driver for the PDC peripheral on RA MCUs. This module implements the PDC Interface.

Overview

The PDC peripheral supports interfacing with external cameras by accepting timing and data signals in order to capture incoming data. A callback is invoked every time a frame of data is accepted.

Features

- Capture incoming data into a user defined buffer
- Data bytes per pixel can be configured
- Endianess of the incoming data can be specified
- Supports configuring capture width and height
- Supports configuring vertical and horizontal sync polarity
- Horizontal and Vertical position for image/data capture can be specified
- External clock to the camera module can be adjusted
- Choice between DMA and DTC to transfer out the captured data
- The specified user callback is invoked when a data frame is captured

Configuration

Build Time Configurations for r_pdc

The following build time configurations are defined in fsp_cfg/r_pdc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Graphics > Parallel Data Capture (r_pdc)

This module can be added to the Stacks tab via New Stack > Graphics > Parallel Data Capture (r_pdc).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_pdc0	Module name.
Input > Signal polarity > HSYNC	• High • Low	High	Specify the active polarity of the HSYNC signal.

Input > Signal polarity > VSYNC	• High • Low	High	Specify the active polarity of the VSYNC signal.
Input > Capture Specifications > Number of pixels to capture horizontally	Value must be an integer greater than 0	640	Specify the number of horizontal pixels to capture.
Input > Capture Specifications > Number of lines to capture vertically	Value must be an integer greater than 0	480	Specify the number of vertical pixels to capture.
Input > Capture Specifications > Horizontal pixel to start capture from	Value must be an integer	0	Specify the horizontal pixel to start capturing image data from. Allows an image smaller than the native resolution of a camera to be captured.
Input > Capture Specifications > Line to start capture from	Value must be an integer	0	Specify the vertical line to start capturing image data from. Allows an image smaller than the native resolution of a camera to be captured.
Input > Bytes per pixel	Value must be an integer greater than 0	2	Specify the number of bytes per pixel of the captured image data.
Input > Clock divider	• CLK/2 • CLK/4 • CLK/6 • CLK/8 • CLK/10 • CLK/12 • CLK/14 • CLK/16	CLK/2	Specify the clock divider of the clock input to the PDC peripheral.
Input > Endianess	• Little • Big	Little	Specify the endianness of the captured image data.
Output > Buffer > Image buffer name	Name must be a valid C symbol	g_user_buffer	Specify the name of the data buffer to create or set to NULL, if it is to be created by the user external to the PDC driver.
Output > Buffer > Image buffer section	This property must be a valid section name	.bss	Specify the RAM section for the image data buffer. Typically .bss (internal RAM) or

			.sdram. When Arm Compiler 6 is used to place this memory in on-chip SRAM, the section name must be .bss or start with .bss. to avoid consuming unnecessary ROM space.
Output > Buffer > Number of image buffers	Value must be an integer greater than 0	1	Specify the number of buffers to create.
Interrupts > Callback	Name must be a valid C symbol	g_pdc_user_callback	A user callback function must be provided. This callback is invoked for every successful frame capture and any error conditions
Interrupts > PDC Interrupt Priority	MCU Specific Options		Select the PDC interrupt priority.
Interrupts > DTC Interrupt Priority	MCU Specific Options		Select the DTC interrupt priority.

Clock Configuration

The PDC peripheral module uses the PCLKB as its clock source. The maximum clock to the camera module is PCLKB / 2.

Pin Configuration

The PCKO pin is a clock output and should be connected to the clock input of the camera. The PIXCLK pin is a clock input and should be connected to the output pixel clock of the camera. Likewise, the HSYNC and VSYNC pins must be connected to the horizontal and vertical sync signals of the camera, respectively. The PIXD0-PIXD7 pins are the 8-bit data bus input and should be connected to the relevant output pins of the camera.

Note

Camera control and serial communication pins must be configured separately and are not controlled by this module.

Usage Notes

Interrupt Configuration

- PDC error interrupts are used by this module for reporting errors such as overrun, underrun, vertical line number setting and horizontal byte number setting errors.
- In addition to the PDC error interrupts, DMA or DTC interrupts are also used internally to perform data transfer from this peripheral to the specified image buffer.
- Receive data ready interrupt is used as activation source for DMA and DTC trigger.

Enabling Transfer Modules

- An option to select between DMAC or DTC is provided with DMA as the default transfer choice.
- For further details on DMA please refer Direct Memory Access Controller (r_dmac)
- For further details on DTC please refer Data Transfer Controller (r_dtc)

PDC setup with external camera

- Before configuring the external camera device the PDC Open API must be called in order to start clock output.
- Ensure that the memory pointed to by p_buffer is both valid and large enough to store a complete image.
- The amount of space required (in bytes) can be calculated as: size (bytes) = image width (pixels) * image height (lines) * number of bytes per pixel
- Ensure that the size above is divisible by and aligned to 32 bytes.

Examples

Basic Example

This is a basic example of minimal use of the PDC in an application. This example shows how this driver can be used for capturing data from an external I/O device such as an image sensor.

```
void g_pdc_user_callback (pdc_callback_args_t * p_args)
{
 if (PDC_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_capture_ready = true;
    }
}
void basic_example (void)
{
 fsp_err_t err;
 /* Initialize the PDC module */
    err = R_PDC_Open(&g_pdc0_ctrl, &g_pdc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the camera module at this point. This implementation is camera vendor
specific. */
    camera_module_initialization();
 /* Initialize capture ready flag to false. This gets set to true in PDC callback
upon successful frame capture. */
    g_capture_ready = false;
```

```
    err = R_PDC_CaptureStart(&g_pdc0_ctrl, g_user_buffer);
    assert(FSP_SUCCESS == err);
    uint32_t timeout_ms = PDC_DELAY_MS;
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((true != g_capture_ready) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (0U == timeout_ms)
    {
       __BKPT(0);
    }
}
static void camera_module_initialization (void)
{
 /* Camera vendor specific initialization to be done here */
}
```

Data Structures

struct	pdc_instance_ctrl_t

Data Structure Documentation

◆ pdc_instance_ctrl_t

struct pdc_instance_ctrl_t
PDC instance control block. DO NOT INITIALIZE.

Function Documentation

◆ R_PDC_Open()

fsp_err_t R_PDC_Open (pdc_ctrl_t *const *p_api_ctrl*, pdc_cfg_t const *const *p_cfg*)

Powers on PDC, handles required initialization described in the hardware manual.

Implements pdc_api_t::open.

The Open function provides initial configuration for the PDC module. It powers on the module and enables the PCLKO output and the PIXCLK input. Further initialization requires the PIXCLK input to be running in order to be able to reset the PDC as part of its initialization. This clock is input from a camera module and so the reset and further initialization is performed in pdc_api_t::captureStart. This function should be called once prior to calling any other PDC API functions. After the PDC is opened the Open function should not be called again without first calling the Close function.

Example:

```
/* Initialize the PDC module */
    err = R_PDC_Open(&g_pdc0_ctrl, &g_pdc0_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	One or more of the following parameters is NULL 1. p_cfg is NULL 2. p_api_ctrl is NULL 3. The pointer to the transfer interface in the p_cfg parameter is NULL 4. Callback parameter is NULL. 5. Invalid IRQ number assigned
FSP_ERR_INVALID_ARGUMENT	One or more of the following parameters is incorrect 1. bytes_per_pixel is zero 2. x_capture_pixels is zero 3. y_capture_pixels is zero 4. x_capture_start_pixel + x_capture_pixels is greater than 4095, OR 5. y_capture_start_pixel + y_capture_pixels is greater than 4095
FSP_ERR_ALREADY_OPEN	Module is already open.

◆ R_PDC_Close()

fsp_err_t R_PDC_Close (pdc_ctrl_t *const *p_api_ctrl*)

Stops and closes the transfer interface, disables and powers off the PDC, clears internal driver data and disables interrupts.

Implements pdc_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	p_api_ctrl is NULL
FSP_ERR_NOT_OPEN	Open has not been successfully called.

◆ R_PDC_CaptureStart()

fsp_err_t R_PDC_CaptureStart (pdc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_buffer*)

Starts a capture. Enables interrupts.

Implements pdc_api_t::captureStart.

Sets up the transfer interface to transfer data from the PDC into the specified buffer. Configures the PDC settings as previously set by the pdc_api_t::open API. These settings are configured here as the PIXCLK input must be active for the PDC reset operation. When a capture is complete the callback registered during pdc_api_t::open API call will be called.

Example:

```
err = R_PDC_CaptureStart(&g_pdc0_ctrl, g_user_buffer);
assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Capture start successful.
FSP_ERR_ASSERTION	One or more of the following parameters is NULL 1. p_api_ctrl is NULL 2. p_buffer is NULL while p_buffer field of the control structure is NULL
FSP_ERR_NOT_OPEN	Open has not been successfully called.
FSP_ERR_IN_USE	PDC transfer is already in progress.
FSP_ERR_TIMEOUT	Reset operation timed out.

4.2.45 Port Output Enable for GPT (r_poeg)

Modules

Functions

fsp_err_t	R_POEG_Open (poeg_ctrl_t *const p_ctrl, poeg_cfg_t const *const p_cfg)
fsp_err_t	R_POEG_StatusGet (poeg_ctrl_t *const p_ctrl, poeg_status_t *const p_status)
fsp_err_t	R_POEG_CallbackSet (poeg_ctrl_t *const p_ctrl, void(*p_callback)(poeg_callback_args_t *), void const *const p_context, poeg_callback_args_t *const p_callback_memory)
fsp_err_t	R_POEG_OutputDisable (poeg_ctrl_t *const p_ctrl)
fsp_err_t	R_POEG_Reset (poeg_ctrl_t *const p_ctrl)
fsp_err_t	R_POEG_Close (poeg_ctrl_t *const p_ctrl)

Detailed Description

Driver for the POEG peripheral on RA MCUs. This module implements the POEG Interface.

Overview

The POEG module can be used to configure events to disable GPT GTIOC output pins.

Features

The POEG module has the following features:

- Supports disabling GPT output pins based on GTETRG input pin level.
- Supports disabling GPT output pins based on comparator crossing events (configurable in the High-Speed Analog Comparator (r_acmphs) driver).
- Supports disabling GPT output pins when GTIOC pins are the same level (configurable in the General PWM Timer (r_gpt) driver).
- Supports disabling GPT output pins when main oscillator stop is detected.
- Supports disabling GPT output pins by software API.
- Supports notifying the application when GPT output pins are disabled by POEG.
- Supports resetting POEG status.

Configuration

Build Time Configurations for r_poeg

The following build time configurations are defined in fsp_cfg/r_poeg_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Timers > Port Output Enable for GPT (r_poeg)

This module can be added to the Stacks tab via New Stack > Timers > Port Output Enable for GPT (r_poeg). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_poeg0	Module name.
General > Channel	Must be a valid POEG channel	0	Specify the hardware channel.
General > Trigger	MCU Specific Options		Select the trigger sources that will enable POEG. Software disable is always supported. This configuration can only be set once after reset. It cannot be modified after the initial setting.
Input > GTETRG Polarity	• Active High • Active Low	Active High	Select the polarity of the GTETRG pin. Only applicable if GTETRG pin is selected under Trigger.
Input > GTETRG Noise Filter	• Disabled • PCLKB/1 • PCLKB/8 • PCLKB/32 • PCLKB/128	Disabled	Configure the noise filter for the GTETRG pin. Only applicable if GTETRG pin is selected under Trigger.
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function can be specified here. If this callback function is provided, it will be called from the interrupt service routine (ISR) when GPT output pins are disabled by POEG.
Interrupts > Interrupt Priority	MCU Specific Options		Select the POEG interrupt priority.

Clock Configuration

The POEG clock is based on the PCLKB frequency.

Pin Configuration

This module can use GTETRGA, GTETRGB, GTETRGC, or GTETRGD as an input signal to disable GPT output pins.

Usage Notes

POEG GTETRG Pin and Channel

The POEG channel number corresponds to the GTETRG input pin that can be used with the channel. GTETRGA must be used with POEG channel 0, GTETRGB must be used with POEG channel 1, etc.

Limitations

The user should be aware of the following limitations when using POEG:

- The POEG trigger source can only be set once per channel. Modifying the POEG trigger source after it is set is not allowed by the hardware.
- The POEG cannot be disabled using this API. The interrupt is disabled in R_POEG_Close(), but the POEG will still disable the GPT output pins if a trigger is detected even if the module is closed.

Examples

POEG Basic Example

This is a basic example of minimal use of the POEG in an application.

```
void poeg_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the POEG. */
    err = R_POEG_Open(&g_poeg0_ctrl, &g_poeg0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

POEG Callback Example

This is an example of a using the POEG callback to restore GPT output operation.

```
/* Example callback called when POEG disables GPT output pins. */
void poeg_callback (poeg_callback_args_t * p_args)
{
```

```
 FSP_PARAMETER_NOT_USED(p_args);
 /* (Optional) Determine the cause of the POEG event. */
 poeg_status_t status;
    (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
 /* Correct the cause of the POEG event before resetting POEG. */
 /* Reset the POEG before exiting the callback. */
    (void) R_POEG_Reset(&g_poeg0_ctrl);
 /* Wait for the status to clear after reset before exiting the callback to ensure
the interrupt does not fire
  * again. */
 do
    {
      (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
    } while (POEG_STATE_NO_DISABLE_REQUEST != status.state);
 /* Alternatively, if the POEG cannot be reset, disable the POEG interrupt to prevent
it from firing continuously.
  * Update the 0 in the macro below to match the POEG channel number. */
    NVIC_DisableIRQ(VECTOR_NUMBER_POEG0_EVENT);
}
```

Data Structures

struct	poeg_instance_ctrl_t

Data Structure Documentation

◆ poeg_instance_ctrl_t

struct poeg_instance_ctrl_t
Channel control block. DO NOT INITIALIZE. Initialization occurs when poeg_api_t::open is called.

Function Documentation

◆ R_POEG_Open()

fsp_err_t R_POEG_Open (poeg_ctrl_t *const *p_ctrl*, poeg_cfg_t const *const *p_cfg*)

Initializes the POEG module and applies configurations. Implements poeg_api_t::open.

Note

The poeg_cfg_t::trigger setting can only be configured once after reset. Reopening with a different trigger configuration is not possible.

Example:

```
/* Initializes the POEG. */
    err = R_POEG_Open(&g_poeg0_ctrl, &g_poeg0_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful.
FSP_ERR_ASSERTION	A required input pointer is NULL or the source divider is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_IRQ_BSP_DISABLED	poeg_cfg_t::p_callback is not NULL, but ISR is not enabled. ISR must be enabled to use callback.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel requested in the p_cfg parameter is not available on this device.

◆ R_POEG_StatusGet()

fsp_err_t R_POEG_StatusGet (poeg_ctrl_t *const *p_ctrl*, poeg_status_t *const *p_status*)

Get current POEG status and store it in provided pointer p_status. Implements poeg_api_t::statusGet.

Example:

```
/* (Optional) Determine the cause of the POEG event. */
poeg_status_t status;
    (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
```

Return values

FSP_SUCCESS	Current POEG state stored successfully.
FSP_ERR_ASSERTION	p_ctrl or p_status was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_POEG_CallbackSet()

fsp_err_t R_POEG_CallbackSet (poeg_ctrl_t *const *p_ctrl*, void(*)(poeg_callback_args_t *) *p_callback*, void const *const *p_context*, poeg_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements poeg_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_POEG_OutputDisable()

fsp_err_t R_POEG_OutputDisable (poeg_ctrl_t *const *p_ctrl*)

Disables GPT output pins. Implements poeg_api_t::outputDisable.

Return values

FSP_SUCCESS	GPT output pins successfully disabled.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_POEG_Reset()

fsp_err_t R_POEG_Reset (poeg_ctrl_t *const *p_ctrl*)

Resets status flags. Implements poeg_api_t::reset.

Note

Status flags are only reset if the original POEG trigger is resolved. Check the status using R_POEG_StatusGet after calling this function to verify the status is cleared.

Example:

```
/* Correct the cause of the POEG event before resetting POEG. */
/* Reset the POEG before exiting the callback. */
   (void) R_POEG_Reset(&g_poeg0_ctrl);
/* Wait for the status to clear after reset before exiting the callback to ensure
the interrupt does not fire
  * again. */
do
   {
     (void) R_POEG_StatusGet(&g_poeg0_ctrl, &status);
   } while (POEG_STATE_NO_DISABLE_REQUEST != status.state);
```

Return values

FSP_SUCCESS	Function attempted to clear status flags.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

◆ R_POEG_Close()

fsp_err_t R_POEG_Close (poeg_ctrl_t *const *p_ctrl*)

Disables POEG interrupt. Implements poeg_api_t::close.

Note

This function does not disable the POEG.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	p_ctrl was NULL.
FSP_ERR_NOT_OPEN	The instance is not opened.

4.2.46 Precision Time Protocol (r_ptp)

Modules

Functions

fsp_err_t	R_PTP_Open (ptp_ctrl_t *const p_ctrl, ptp_cfg_t const *const p_cfg)
fsp_err_t	R_PTP_MacAddrSet (ptp_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr)
fsp_err_t	R_PTP_IpAddrSet (ptp_ctrl_t *const p_ctrl, uint32_t ip_addr)
fsp_err_t	R_PTP_LocalClockIdSet (ptp_ctrl_t *const p_ctrl, uint8_t const *const p_clock_id)
fsp_err_t	R_PTP_MasterClockIdSet (ptp_ctrl_t *const p_ctrl, uint8_t const *const p_clock_id, uint16_t port_id)
fsp_err_t	R_PTP_MessageFlagsSet (ptp_ctrl_t *const p_ctrl, ptp_message_type_t message_type, ptp_message_flags_t flags)
fsp_err_t	R_PTP_CurrentUtcOffsetSet (ptp_ctrl_t *const p_ctrl, uint16_t offset)
fsp_err_t	R_PTP_PortStateSet (ptp_ctrl_t *const p_ctrl, uint32_t state)
fsp_err_t	R_PTP_MessageSend (ptp_ctrl_t *const p_ctrl, ptp_message_t const *const p_message, uint8_t const *const p_tlv_data, uint16_t tlv_data_size)
fsp_err_t	R_PTP_LocalClockValueSet (ptp_ctrl_t *const p_ctrl, ptp_time_t const *const p_time)
fsp_err_t	R_PTP_LocalClockValueGet (ptp_ctrl_t *const p_ctrl, ptp_time_t *const p_time)
fsp_err_t	R_PTP_PulseTimerCommonConfig (ptp_ctrl_t *const p_ctrl, ptp_pulse_timer_common_cfg_t *const p_timer_cfg)
fsp_err_t	R_PTP_PulseTimerEnable (ptp_ctrl_t *const p_ctrl, uint32_t channel, ptp_pulse_timer_cfg_t *const p_timer_cfg)
fsp_err_t	R_PTP_PulseTimerDisable (ptp_ctrl_t *const p_ctrl, uint32_t channel)
fsp_err_t	R_PTP_Close (ptp_ctrl_t *const p_ctrl)
fsp_err_t	R_PTP_BestMasterClock (ptp_message_t const *const p_announce1, ptp_message_t const *const p_announce2, int8_t *const

p_comparison)

Detailed Description

Driver for the PTP peripheral on RA MCUs. This module implements the PTP Interface.

Overview

PTP allows for multiple devices on a network to synchronize their clocks with very high precision. The PTP peripheral generates and processes PTP messages automatically. In slave mode, it also corrects the local time in order to adjust for any offset from the master clock time.

Features

- Ordinary clock
  - Master mode
  - Slave mode
- Peer-to-peer
- End-to-end
- Frame fromats
  - Ethernet II frames
  - IEEE802.3 + LLC + SNAP frames
  - IPv4 + UDP
- Clock correction modes
  - Mode 1: Add the offsetFromMaster value to the local time whenever it is updated.
  - Mode 2: Calculate a clock gradient and continuously adjust the local time in order to minimize the offsetFromMaster value.

Configuration

Build Time Configurations for r_ptp

The following build time configurations are defined in fsp_cfg/r_ptp_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Networking > PTP (r_ptp)

This module can be added to the Stacks tab via New Stack > Networking > PTP (r_ptp).

Configuration	Options	Default	Description
Clock Properties > Priority 1	Value must in the range [0,255].	128	Priority1 field advertised in generated announce packets.
Clock Properties > Class	Value must in the range [0,255].	248	Class field advertised in generated announce

			packets.
Clock Properties > Accuracy	Value must in the range [0,255].	0xFE	Accuracy field advertised in generated announce packets.
Clock Properties > Variance	Value must in the range [0,65535].	0xFFFF	Variance field advertised in generated announce packets.
Clock Properties > Priority 2	Value must in the range [0,255].	128	Priority2 field advertised in generated announce packets.
Clock Properties > Time Source	Value must in the range [0,255].	160	Time Source field advertised in generated announce packets.
Ethernet > Multicast Filter MAC address	Must be a valid MAC address	01:1B:19:00:00:00	In Multicast Filtered mode, only multicast addresses that match this address are received by the ETHERC EDMAC.
Ethernet > Primary Destination MAC address	Must be a valid MAC address	01:1B:19:00:00:00	The destination MAC address for primary PTP messages.
Ethernet > PDelay Destination MAC address	Must be a valid MAC address	01:80:C2:00:00:0E	The destination MAC address for PDelay messages.
IP > Primary Destination IP address	Must be a valid IP address	224.0.1.129	The destination IPv4 address for primary messages.
IP > PDelay Destination IP address	Must be a valid IP address	224.0.0.107	The destination IPv4 address for PDelay messages.
IP > Event Message TOS	Value must in the range [0,255].	0	The IP packet TOS for event messages.
IP > General Message TOS	Value must in the range [0,255].	0	The IP packet TOS for general messages.
IP > Primary Message TTL	Value must in the range [0,255].	1	The IP packet TTL for primary messages.
IP > PDelay Message TTL	Value must in the range [0,255].	1	The IP packet TTL for p_delay messages.
IP > Event Port	Value must in the range [0,65535].	319	The UDP port for event messages.

IP > General Port	Value must in the range [0,65535].	320	The UDP port for general messages.
Synchronization Detection > Threshold (Nanoseconds)	Value must be greater than 0.	1000000	The minimum offsetFromMaster value required in order to synchronize with the master clock.
Synchronization Detection > Count	Value must in the range [0,255].	5	The number of times the calculated offsetFromMaster value must be less than the threshold in order to synchronize with the master clock.
Synchronization Lost Detection > Threshold (Nanoseconds)	Value must be greater than 0.	10000000	The minimum offsetFromMaster value required in order to lose synchronization with the master clock.
Synchronization Lost Detection > Count	Value must in the range [0,255].	5	The number of times the calculated offsetFromMaster value must be greater than the threshold in order to lose synchronization with the master.
Interrupts > Callback	Name must be a valid C symbol	${module.driver.ptp.name}_callback	Called when a STCA/SYNFP event occurs, a PTP message is received, or if a Pulse Timer event occurs.
Interrupts > MINT Interrupt priority	MCU Specific Options		Select the EPTPC MINT interrupt priority.
Interrupts > Pulse Timer Interrupt priority	MCU Specific Options		Select the EPTPC IPLS priority.
Name	Name must be a valid C symbol	g_ptp0	Module name.
Ethernet PHY Interface Type	• MII • RMII	RMII	The interface type used to communicate with the Ethernet PHY.
Frame Filter	• Extended Promiscuous Mode • Unicast and Multicast • Unicast and Multicast Filtered • Unicast	Unicast	Selects how packets are filtered based on their destination MAC address. Packets that pass the filter are transferred to the ETHERC EDMAC.

Frame Format	• Ethernet II • Ethernet II | IPv4 | UDP • IEEE802.3 | LLC | SNAP • IEEE802.3 | LLC | SNAP | IPv4 | UDP	Ethernet II	The format of the frames that encapsulate the PTP messages.
Clock Domain	Value must in the range [0,255].	0	The PTP clock will only respond to clocks in its domain.
Clock Domain Filter	• Enable • Disable	Enable	Filter out PTP messages from other clock domains.
Buffer Size	Value must in the range [64,1536].	1536	The maximum Ethernet packet size that can be transmitted or received by the application from the EDMAC.
Number of transmit buffers	Value must in the range [1,16].	4	The number of transmit buffers in the packet queue.
Number of receive buffers	Value must in the range [1,16].	4	The number of receive buffers in the packet queue.
Announce message interval.	MCU Specific Options		The period of time between generated announce messages.
Sync message interval.	MCU Specific Options		The period of time between generated sync messages.
Delay_req message interval.	MCU Specific Options		The period of time between generated delay_req messages.
Message timeout	Value must be greater than 0.	4000	The time in milliseconds needed to generate timeout events after not receiving a sync or delay_resp message.
Clock Source	• PCLKA / 1 • PCLKA / 2 • PCLKA / 3 • PCLKA / 4 • PCLKA / 5 • PCLKA / 6 • REF50CK0	PCLKA / 6	The STCA clock source must be 20Mhz, 25Mhz, 50Mhz, or 100Mhz. When REF50CK0 is selected, the STCA frequency is 25Mhz.
Clock Correction Mode	• Clock	Clock Correction Mode	Clock correction mode

	Correction Mode 1 • Clock Correction Mode 2	1	1 corrects the local clock using the current offsetFromMaster value. Clock correction mode 2 calculates a clock gradient in order to continuously correct the local clock.
Gradient Worst10 Interval	Value must in the range [0,255].	32	The number of sync messages to use when calculating the worst10 gradient values (Only applies to clock correction mode 2).

Clock Configuration

The STCA input clock can be the following clock sources:

- PCLKA / 1
- PCLKA / 2
- PCLKA / 3
- PCLKA / 4
- PCLKA / 5
- PCLKA / 6
- REF50CK0

The STCA input clock is restricted to the following frequencies:

- 20 Mhz
- 25 Mhz
- 50 Mhz
- 100 Mhz

When REF50CK0 is selected, the input clock frequency is 25 Mhz.

Pin Configuration

The PTP module requires the Ethernet (r_ether) instance in order to initialize the Ethernet PHY. This means that the ETHERC pins must be configured.

Usage Notes

PTP Port State

The current PTP port state determines which messages need to be generated and processed by the PTP peripheral. It is the application's responsibility to determine what the current state of the PTP port should be.

The following messages can be generated by the PTP peripheral:

- Announce
- Sync
- Delay_req

- Delay_resp
- PDelay_req
- PDelay_resp

The following messages can be processed by the PTP peripheral:

- Sync
- Follow_up
- Delay_req
- Delay_resp
- PDelay_req
- PDelay_resp
- PDelay_resp_follow_up

The application must receive the following messages in order to determine the current state of its PTP port:

- Announce
- Management
- Signaling

The following messages can only be sent manually:

- Management
- Signaling

The PTP API defines the following states:

State	Generated Messages	Processed Messages	Received Messages
Disabled	N/A	N/A	N/A
Passive	N/A	N/A	Announce, Signaling, Management
E2E/P2P Slave	Delay_req/(PDelay_req, PDelay_resp)	Sync, Follow_up, Delay_resp/(PDelay_req , PDelay_resp)	Announce, Signaling, Management
E2E/P2P Master	Announce, Sync, Delay_resp/(PDelay_req , PDelay_resp)	delay_req/(PDelay_req, PDelay_resp)	Announce, Signaling, Management

Pulse Timers

Pulse Timers are configurable timers used to generate interrupts and ELC events. Each pulse timer has a configurable start time, pulse, and period. At the start of each timer period, a rising edge occurs. After the pulse time has elapsed, a falling edge occurs. ELC events and IRQs can be generated on rising and/or falling edges for each Pulse Timer. There are two types of interrupts generated by each Pulse Timer; MINT and IPLS.

MINT Interrupts

MINT IRQs are only generated on the rising edge of a Pulse Timer channel. The callback will provide the channel number of the pulse timer that caused the interrupt.

IPLS Interrupts

Each Pulse Timer channel can be configured as a source for generating IPLS IRQs. All of the pulse timers that are selected as IPLS sources are OR'd together and rising and falling edge IRQs can be generated from the resulting signal. Below is an example of a resulting signal from two IPLS sources. Unlike MINT interrupts, IPLS interrupts do not provide any information about which Pulse Timer caused the IRQ because the IRQs from all the Pulse Timers are OR'd together.

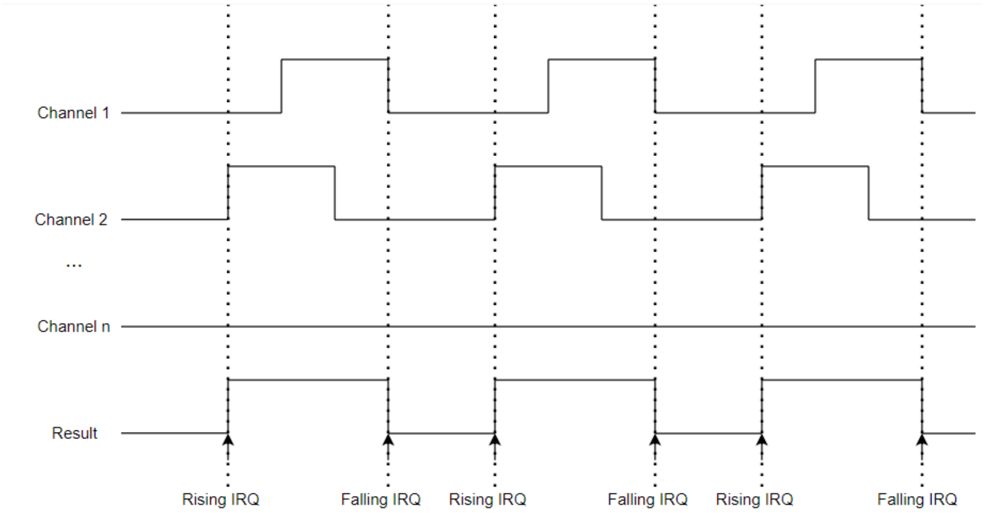


Figure 173: IPLS IRQ Generation

Ethernet Frame Filter

The PTP driver can filter Ethernet frames that are received by Ethernet (r_ether). There are four different filtering modes:

- Extended Promiscuous - All Ethernet frames are received by Ethernet (r_ether).
- Unicast and Multicast - All Unicast frames destined for the PTP and Multicast frames are received by Ethernet (r_ether).
- Unicast and Multicast Filtered - All Unicast frames destined for the PTP are received by Ethernet (r_ether). All multicast frames that match ptp_synfp_cfg_t::p_multicast_addr_filter are received by Ethernet (r_ether).
- Unicast - Only Unicast frames destined for the PTP are received by Ethernet (r_ether).

Limitations

Developers should be aware of the following limitations when using the PTP:

- PTP will not automatically initialize Ethernet (r_ether). This provides flexibility by allowing PTP to be used alongside 3rd party IP stacks (Eg. FreeRTOS Plus TCP (rm_freertos_plus_tcp)), however this means the application must execute the Ethernet (r_ether) link process in order to use PTP.
- The driver will not detect announce message timeouts. This functionality must be handled by the application.
- When IP + UDP frame format is selected, the driver will not automatically join the multicast group. This must be done by the application.
- In order to call PTP API functions from ISRs, the MINT and IPLS interrupt priorities must be configured to be lower than BSP_CFG_IRQ_MASK_LEVEL_FOR_CRITICAL_SECTION. This is to guarantee that PTP register accesses are atomic.

Examples

Slave Mode

This is a basic example of minimal use of PTP in slave mode.

```
volatile bool g_first_announce_message_received = false;
volatile bool g_sync_acquired = false;
void slave_mode_basic_example (void)
{
 /* The PTP Instance must be opened before R_ETHER is opened. */
 fsp_err_t err = R_PTP_Open(&g_ptp_ctrl, &g_ptp_cfg);
    assert(FSP_SUCCESS == err);
 /* Configure the PTP MAC address. */
    err = R_PTP_MacAddrSet(&g_ptp_ctrl, g_ptp_mac_address);
    assert(FSP_SUCCESS == err);
 /* Configure the PTP Local Clock ID (Usually generated from MAC address). */
    err = R_PTP_LocalClockIdSet(&g_ptp_ctrl, g_ptp_clock_id);
    assert(FSP_SUCCESS == err);
 /* Open the r_ether_api instance. */
    err = R_ETHER_Open(&g_ether_ctrl, &g_ether_cfg);
    assert(FSP_SUCCESS == err);
 /* Wait for the link to be established. */
 do
    {
       err = R_ETHER_LinkProcess(&g_ether_ctrl);
    } while (FSP_SUCCESS != err);
 /* Set the PTP instance to passive state and listen for announce message. */
    err = R_PTP_PortStateSet(&g_ptp_ctrl, PTP_PORT_STATE_PASSIVE);
    assert(FSP_SUCCESS == err);
 /* Wait for the first announce message (This will provide the master clock ID). */
    uint32_t timeout = EXAMPLE_TIMEOUT;
 while (!g_first_announce_message_received && --timeout)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
    }
```

```
    assert(0U != timeout);
 /* When a master clock is found, change to the slave state to start synchronizing
  * the local clock to the master clock. */
    err = R_PTP_PortStateSet(&g_ptp_ctrl, PTP_PORT_STATE_E2E_SLAVE);
    assert(FSP_SUCCESS == err);
 /* Wait for local clock to be synchronized with the master clock. */
    timeout = EXAMPLE_TIMEOUT;
 while (!g_sync_acquired && --timeout)
    {
 R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);
    }
    assert(0U != timeout);
 /* The local clock is now synchronized with the grand master clock. */
}
/* Callback called whenever a PTP event occurs. */
void g_ptp_slave_callback_example (ptp_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case PTP_EVENT_SYNC_ACQUIRED:
      {
 /* The offsetFromMaster value is now within the configured threshold to be
  * synchronized with the master clock.
  */
            g_sync_acquired = true;
 break;
      }
 case PTP_EVENT_MESSAGE_RECEIVED:
      {
 static ptp_message_t g_current_master_announce_message;
 switch (p_args->p_message->header.message_type)
      {
 case PTP_MESSAGE_TYPE_ANNOUNCE:
      {
```

```
                  int8_t comparison = 0;
 if (!g_first_announce_message_received)
      {
 /* If this is the first announce packet, immediately switch to this master clock. */
                       comparison = 1;
                       g_first_announce_message_received = true;
      }
 else
      {
 /*
  * Run the "Best Master Clock Algorithm" to determine if the clock defined in this
announce
  * packet is better than the current master clock.
  */
 fsp_err_t err = R_PTP_BestMasterClock(&g_current_master_announce_message,
                                                             p_args->p_message,
                                                             &comparison);
      assert(FSP_SUCCESS == err);
      }
 if (1 == comparison)
      {
 /* Save the message as the new master announce message. */
                       g_current_master_announce_message = *p_args->p_message;
 /* Set the master clock ID and sourcePortID in the PTP instance so that it
  * synchronizes with the new best master clock.
  */
 fsp_err_t err = R_PTP_MasterClockIdSet(&g_ptp_ctrl,
                                                               g_current_master_annou
nce_message.header.clock_id,
                                                               g_current_master_annou
nce_message.header.source_port_id);
      assert(FSP_SUCCESS == err);
      }
 break;
```

```
      }
 default:
      {
 break;
      }
      }
 break;
      }
 default:
      {
 break;
      }
    }
}
```

Master Mode

This is a basic example of minimal use of PTP in master mode.

```
#define PTP_EXAMPLE_CURRENT_UTC_OFFSET (37)
void master_mode_basic_example (void)
{
 /* The PTP Instance must be opened before R_ETHER is opened. */
 fsp_err_t err = R_PTP_Open(&g_ptp_ctrl, &g_ptp_cfg);
    assert(FSP_SUCCESS == err);
 /* Configure the PTP MAC address. */
    err = R_PTP_MacAddrSet(&g_ptp_ctrl, g_ptp_mac_address);
    assert(FSP_SUCCESS == err);
 /* Configure the PTP Local Clock ID (Usually generated from MAC address). */
    err = R_PTP_LocalClockIdSet(&g_ptp_ctrl, g_ptp_clock_id);
    assert(FSP_SUCCESS == err);
 /* Get the current time from an external time source (Eg. RTC). */
 ptp_time_t current_time;
    get_current_time_example(&current_time);
 /* Set the PTP local time to the current time. */
```

```
    err = R_PTP_LocalClockValueSet(&g_ptp_ctrl, &current_time);
    assert(FSP_SUCCESS == err);
 /* Set the currentUtcOffset field in announce messages. */
    err = R_PTP_CurrentUtcOffsetSet(&g_ptp_ctrl, PTP_EXAMPLE_CURRENT_UTC_OFFSET);
    assert(FSP_SUCCESS == err);
 /* Set message flags in announce messages to indicate that the current UTC offset is
valid and that the PTP timescale is used. */
 ptp_message_flags_t flags;
    flags.value = 0;
    flags.value_b.currentUtcOffsetValid = 1;
    flags.value_b.ptpTimescale          = 1;
    err = R_PTP_MessageFlagsSet(&g_ptp_ctrl, PTP_MESSAGE_TYPE_ANNOUNCE, flags);
    assert(FSP_SUCCESS == err);
 /* Open the r_ether_api instance. */
    err = R_ETHER_Open(&g_ether_ctrl, &g_ether_cfg);
    assert(FSP_SUCCESS == err);
 /* Wait for the link to be established. */
 do
    {
       err = R_ETHER_LinkProcess(&g_ether_ctrl);
    } while (FSP_SUCCESS != err);
 /* Set the PTP instance to passive state and listen for announce message. */
    err = R_PTP_PortStateSet(&g_ptp_ctrl, PTP_PORT_STATE_E2E_MASTER);
    assert(FSP_SUCCESS == err);
 /*
  * The master clock is now operational and will automatically generate announce and
sync messages
  * as well as respond to delay_req messages.
  */
}
```

Send PTP Messages

This is a basic example of how to send PTP messages.

```
#define PTP_MANAGEMENT_ACTION_GET (0U)
#define PTP_TLV_TYPE_MANAGEMENT (1U)
#define PTP_TLV_MANAGEMENT_ID_CLOCK_DESCRIPTION (1U)
static uint32_t g_transmit_complete = 0U;
void send_message_example (void)
{
 static ptp_message_t message;
 static uint8_t       p_tlv_data[6];
    memset(&message, 0, sizeof(ptp_message_t));
 /* Fill in the required fields for the message header (Note that appropriate fields
will be endian swapped). */
    message.header.message_type = PTP_MESSAGE_TYPE_MANAGEMENT;
    message.header.version      = 2;
 /* The message length is the total number of bytes in the PTP message (Including the
message header). */
    message.header.message_length = (uint16_t) (sizeof(ptp_message_header_t) +
 sizeof(ptp_message_management_t) +
 sizeof(p_tlv_data));
    memcpy(message.header.clock_id, g_ptp_clock_id, sizeof(g_ptp_clock_id));
    message.header.control_field = PTP_CTRL_FIELD_MANAGEMENT;
 /* Fill in the required fields for the management message. */
    memcpy(message.management.target_clock_id, g_target_clock_id, sizeof
(g_target_clock_id));
    message.management.target_port_id         = 1;
    message.management.starting_boundary_hops = 1;
    message.management.boundary_hops          = 1;
    message.management.action                = PTP_MANAGEMENT_ACTION_GET;
 /*
  * Fill in TLV data (Note that TLV data is big endian).
  *
  * Type (Management)
  */
    p_tlv_data[0] = 0;
    p_tlv_data[1] = PTP_TLV_TYPE_MANAGEMENT;
```

```
 /* Length */
    p_tlv_data[2] = 0;
    p_tlv_data[3] = 2;
 /* Management ID (Clock Description) */
    p_tlv_data[4] = 0;
    p_tlv_data[5] = PTP_TLV_MANAGEMENT_ID_CLOCK_DESCRIPTION;
 /* Send the message. */
 fsp_err_t err = R_PTP_MessageSend(&g_ptp_ctrl, &message, p_tlv_data, sizeof
(p_tlv_data));
    assert(FSP_SUCCESS == err);
    uint32_t timeout = EXAMPLE_TIMEOUT;
 while (0U == g_transmit_complete && --timeout)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MICROSECONDS);
    }
}
/* Callback called whenever a PTP event occurs. */
void g_ptp_send_message_callback_example (ptp_callback_args_t * p_args)
{
 switch (p_args->event)
    {
 case PTP_EVENT_MESSAGE_TRANSMIT_COMPLETE:
      {
            g_transmit_complete = 1U;
 break;
      }
 case PTP_EVENT_MESSAGE_RECEIVED:
      {
 switch (p_args->p_message->header.message_type)
      {
 case PTP_MESSAGE_TYPE_MANAGEMENT:
      {
 /* Handle the response message. */
 break;
```

```
        }
default:
        {
break;
        }
        }
        }
default:
        {
break;
        }
    }
}
```

Data Structures

struct	ptp_instance_ctrl_t

Data Structure Documentation

◆ ptp_instance_ctrl_t

struct ptp_instance_ctrl_t

PTP instance control block.

Data Fields		
uint32_t	open	Marks if the instance has been opened.
uint32_t	tx_buffer_write_index	Index into the descriptor list to write the next packet.
uint32_t	tx_buffer_complete_index	Index into the descriptor list of the last transmitted packet.
uint32_t	rx_buffer_index	Index into the descriptor of the last received packet.
uint32_t	tslatr	Keep track of whether tslatr was set.
ptp_cfg_t const *	p_cfg	Pointer to the configuration structure.

Function Documentation

◆ R_PTP_Open()

fsp_err_t R_PTP_Open (ptp_ctrl_t *const *p_ctrl*, ptp_cfg_t const *const *p_cfg*)

This function initializes PTP. Implements ptp_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Configures the peripheral registers acording to the configuration.
- Initialize the control structure for use in other PTP Interface functions.

Return values

FSP_SUCCESS	The instance has been successfully configured.
FSP_ERR_ALREADY_OPEN	Instance was already initialized.
FSP_ERR_NOT_OPEN	The EDMAC instance was not opened correctly.
FSP_ERR_ASSERTION	An invalid argument was given in the configuration structure.

◆ R_PTP_MacAddrSet()

fsp_err_t R_PTP_MacAddrSet (ptp_ctrl_t *const *p_ctrl*, uint8_t const *const *p_mac_addr*)

This function sets the MAC address for the PTP instance. Implements ptp_api_t::macAddrSet.

Note

This function may only be called while the PTP instance is in ptp_port_state_t::PTP_PORT_STATE_DISABLE.

Return values

FSP_SUCCESS	The MAC address has been set.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL
FSP_ERR_INVALID_MODE	The instance is not in the correct state.

◆ R_PTP_IpAddrSet()

fsp_err_t R_PTP_IpAddrSet (ptp_ctrl_t *const *p_ctrl*, uint32_t *ip_addr*)

This function sets the IP address for the PTP instance. Implements ptp_api_t::ipAddrSet.

Note

This function may only be called while the PTP instance is in ptp_port_state_t::PTP_PORT_STATE_DISABLE.

Return values

FSP_SUCCESS	The IP address has been set.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL.
FSP_ERR_INVALID_MODE	The configured ptp_synfp_cfg_t::frame_format is not configured to use IP packets, or the instance is not in the correct state.

◆ R_PTP_LocalClockIdSet()

fsp_err_t R_PTP_LocalClockIdSet (ptp_ctrl_t *const *p_ctrl*, uint8_t const *const *p_clock_id*)

This function sets the local clock ID for the PTP instance. Implements ptp_api_t::localClockIdSet.

Note

This function may only be called while the PTP instance is in ptp_port_state_t::PTP_PORT_STATE_DISABLE.
Typically the clock ID is derived from the MAC address (E.g. {b1,b2,b3,0xFF,0xFE,b4,b5,b6}).

Return values

FSP_SUCCESS	The local clock ID has been set.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL
FSP_ERR_INVALID_MODE	The instance is not in the correct state.

◆ R_PTP_MasterClockIdSet()

fsp_err_t R_PTP_MasterClockIdSet (ptp_ctrl_t *const *p_ctrl*, uint8_t const *const *p_clock_id*, uint16_t *port_id*)

This function sets the master clock ID and port ID that the local clock will synchronize with. Implements ptp_api_t::masterClockIdSet.

Return values

FSP_SUCCESS	The master clock ID and port ID have been set.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL

◆ R_PTP_MessageFlagsSet()

fsp_err_t R_PTP_MessageFlagsSet (ptp_ctrl_t *const *p_ctrl*, ptp_message_type_t *message_type*, ptp_message_flags_t *flags*)

This function sets the flags field for the given message type. Implements ptp_api_t::messageFlagsSet.

Return values

FSP_SUCCESS	The master clock ID and port ID have been set.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL or invalid.

◆ R_PTP_CurrentUtcOffsetSet()

fsp_err_t R_PTP_CurrentUtcOffsetSet (ptp_ctrl_t *const *p_ctrl*, uint16_t *offset*)

This function sets the currentUtcOffset value in announce messages. ptp_api_t::currentUtcOffsetSet .

Return values

FSP_SUCCESS	The currentUtcOffset has been updated.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL or invalid.

◆ R_PTP_PortStateSet()

fsp_err_t R_PTP_PortStateSet (ptp_ctrl_t *const *p_ctrl*, uint32_t *state*)

This function changes the current state of the PTP instance. Implements ptp_api_t::portStateSet.

Return values

FSP_SUCCESS	The instance will transition to the new state.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL

◆ R_PTP_MessageSend()

fsp_err_t R_PTP_MessageSend (ptp_ctrl_t *const *p_ctrl*, ptp_message_t const *const *p_message*, uint8_t const *const *p_tlv_data*, uint16_t *tlv_data_size*)

This function sends a PTP message. ptp_api_t::messageSend.

Return values

FSP_SUCCESS	The packet has been written to the transmit descriptor.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL or invalid.
FSP_ERR_ETHER_ERROR_TRANSMIT_BUFFER_FULL	There is no space for the packet in the transmit queue.

◆ R_PTP_LocalClockValueSet()

fsp_err_t R_PTP_LocalClockValueSet (ptp_ctrl_t *const *p_ctrl*, ptp_time_t const *const *p_time*)

This function sets the local clock value. Implements ptp_api_t::localClockValueSet.

Return values

FSP_SUCCESS	The local clock value has been set.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL or invalid.

◆ R_PTP_LocalClockValueGet()

fsp_err_t R_PTP_LocalClockValueGet (ptp_ctrl_t *const *p_ctrl*, ptp_time_t *const *p_time*)

This function gets the local clock value. Implements ptp_api_t::localClockValueGet.

Return values

FSP_SUCCESS	The local clock value has been written in p_time.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL

◆ R_PTP_PulseTimerCommonConfig()

fsp_err_t R_PTP_PulseTimerCommonConfig (ptp_ctrl_t *const *p_ctrl*, ptp_pulse_timer_common_cfg_t *const *p_timer_cfg*)

This function configures IPLS IRQ settings that are common to all pulse timer channels. Implements ptp_api_t::pulseTimerCommonConfig.

Return values

FSP_SUCCESS	The pulse timer has been enabled.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL or invalid.

◆ R_PTP_PulseTimerEnable()

fsp_err_t R_PTP_PulseTimerEnable (ptp_ctrl_t *const *p_ctrl*, uint32_t *channel*, ptp_pulse_timer_cfg_t *const *p_timer_cfg*)

This function enables a pulse timer channel. Implements ptp_api_t::pulseTimerEnable.

Return values

FSP_SUCCESS	The pulse timer has been enabled.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL or invalid.

◆ R_PTP_PulseTimerDisable()

fsp_err_t R_PTP_PulseTimerDisable (ptp_ctrl_t *const *p_ctrl*, uint32_t *channel*)

This function disables a pulse timer channel. Implements ptp_api_t::pulseTimerDisable.

Return values

FSP_SUCCESS	The pulse timer has been disabled.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL or invalid.

◆ R_PTP_Close()

fsp_err_t R_PTP_Close (ptp_ctrl_t *const *p_ctrl*)

Disable the PTP instance. Implements ptp_api_t::close.

Return values

FSP_SUCCESS	The pulse timer has been disabled.
FSP_ERR_NOT_OPEN	The instance has not been opened.
FSP_ERR_ASSERTION	An argument was NULL or invalid.

◆ R_PTP_BestMasterClock()

fsp_err_t R_PTP_BestMasterClock (ptp_message_t const *const *p_announce1*, ptp_message_t const *const *p_announce2*, int8_t *const *p_comparison*)

This function compares two clocks to determine which one is the better master clock.

p_comparison:

- Set to -1 if p_announce1 defines the best master clock.
- Set to 1 if p_announce2 defines the best master clock.
- Set to 0 if p_announce1 and p_announce2 define the same clock.

Return values

FSP_SUCCESS	The valid result has been written to p_use_announce_clock.
FSP_ERR_ASSERTION	An argument was NULL.

4.2.47 Quad Serial Peripheral Interface Flash (r_qspi)

Modules

Functions

fsp_err_t	R_QSPI_Open (spi_flash_ctrl_t *p_ctrl, spi_flash_cfg_t const *const p_cfg)
fsp_err_t	R_QSPI_Close (spi_flash_ctrl_t *p_ctrl)
fsp_err_t	R_QSPI_DirectWrite (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write)
fsp_err_t	R_QSPI_DirectRead (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t	R_QSPI_SpiProtocolSet (spi_flash_ctrl_t *p_ctrl, spi_flash_protocol_t spi_protocol)
fsp_err_t	R_QSPI_XipEnter (spi_flash_ctrl_t *p_ctrl)
fsp_err_t	R_QSPI_XipExit (spi_flash_ctrl_t *p_ctrl)
fsp_err_t	R_QSPI_Write (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count)
fsp_err_t	R_QSPI_Erase (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_device_address, uint32_t byte_count)
fsp_err_t	R_QSPI_StatusGet (spi_flash_ctrl_t *p_ctrl, spi_flash_status_t *const p_status)
fsp_err_t	R_QSPI_BankSet (spi_flash_ctrl_t *p_ctrl, uint32_t bank)
fsp_err_t	R_QSPI_DirectTransfer (spi_flash_ctrl_t *p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction)
fsp_err_t	R_QSPI_AutoCalibrate (spi_flash_ctrl_t *p_ctrl)

Detailed Description

Driver for the QSPI peripheral on RA MCUs. This module implements the SPI Flash Interface.

Overview

Features

The QSPI driver has the following key features:

- Memory mapped read access to the QSPI flash

- Programming the QSPI flash device
- Erasing the QSPI flash device
- Sending device specific commands and reading back responses
- Entering and exiting QPI mode
- Entering and exiting XIP mode
- 3 or 4 byte addressing

Configuration

Build Time Configurations for r_qspi

The following build time configurations are defined in driver/r_qspi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Support Multiple Line Program in Extended SPI Mode	• Enabled • Disabled	Disabled	If selected code for programming on multiple lines in extended SPI mode is included in the build.

Configurations for Storage > QSPI (r_qspi)

This module can be added to the Stacks tab via New Stack > Storage > QSPI (r_qspi).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_qspi0	Module name.
General > SPI Protocol	• Extended SPI • QPI	Extended SPI	Select the initial SPI protocol. SPI protocol can be changed in R_QSPI_Direct().
General > Address Bytes	• 3 • 4 • 4 with 4-byte read code	3	Select the number of address bytes. Selecting '4 with 4-byte read code' converts the default read code determined in Read Mode to the 4-byte version. If 4-byte mode is selected without using 4-byte commands, the application must issue the EN4B command using R_QSPI_Direct().
General > Read Mode	• Standard Read • Fast Read	Fast Read Quad I/O	Select the read mode for memory mapped

	• Fast Read Dual Output • Fast Read Dual I/O • Fast Read Quad Output • Fast Read Quad I/O		access.
General > Dummy Clocks for Fast Read	Refer to the RA Configuration tool for available options.	Default	Select the number of dummy clocks for fast read operations. Default is 6 clocks for Fast Read Quad I/O, 4 clocks for Fast Read Dual I/O, and 8 clocks for other fast read instructions including Fast Read Quad Output, Fast Read Dual Output, and Fast Read
General > Page Size Bytes	Must be an integer greater than 0	256	The maximum number of bytes allowed for a single write.
Command Definitions > Page Program Command	Must be an 8-bit QSPI Page Program Command under Command Definitions	0x02	The command to program a page. If 'Support Multiple Line Program in Extended SPI Mode' is Enabled, this command must use the same number of data lines as the selected read mode.
Command Definitions > Page Program Address Lines	• 1 • 2 • 4	1	Select the number of lines to use for the address bytes during write operations. This can be determined by referencing the datasheet for the external QSPI. It should either be 1 or match the number of data lines used for memory mapped fast read operations.
Command Definitions > Write Enable Command	Must be an 8-bit QSPI Write Enable Command under Command Definitions	0x06	The command to enable write.
Command Definitions > Status Command	Must be an 8-bit QSPI Status Command under	0x05	The command to query the status of a write or

	Command Definitions		erase command.
Command Definitions > Write Status Bit	Must be an integer between 0 and 7	0	Which bit contains the write in progress status returned from the Write Status Command.
Command Definitions > Sector Erase Command	Must be an 8-bit QSPI Sector Erase Command under Command Definitions	0x20	The command to erase a sector. Set Sector Erase Size to 0 if unused.
Command Definitions > Sector Erase Size	Must be an integer greater than or equal to 0	4096	The sector erase size. Set Sector Erase Size to 0 if Sector Erase is not supported.
Command Definitions > Block Erase Command	Must be an 8-bit QSPI Block Erase Command under Command Definitions	0xD8	The command to erase a block. Set Block Erase Size to 0 if unused.
Command Definitions > Block Erase Size	Must be an integer greater than or equal to 0	65536	The block erase size. Set Block Erase Size to 0 if Block Erase is not supported.
Command Definitions > Block Erase 32KB Command	Must be an 8-bit QSPI Block Erase 32KB Command under Command Definitions	0x52	The command to erase a 32KB block. Set Block Erase Size to 0 if unused.
Command Definitions > Block Erase 32KB Size	Must be an integer greater than or equal to 0	32768	The block erase 32KB size. Set Block Erase 32KB Size to 0 if Block Erase 32KB is not supported.
Command Definitions > Chip Erase Command	Must be an 8-bit QSPI Chip Erase Command under Command Definitions	0xC7	The command to erase the entire chip. Set Chip Erase Command to 0 if unused.
Command Definitions > XIP Enter M7-M0	Must be an 8-bit QSPI XIP Enter M7-M0 command under Command Definitions	0x20	How to set M7-M0 to enter XIP mode.
Command Definitions > XIP Exit M7-M0	Must be an 8-bit QSPI XIP Exit M7-M0 command under Command Definitions	0xFF	How to set M7-M0 exit XIP mode.
Bus Timing > QSPKCLK Divisor	Refer to the RA Configuration tool for available options.	2	Select the divisor to apply to PCLK to get QSPCLK.
Bus Timing > Minimum	Refer to the RA	4 QSPCLK	Define the minimum

QSSL Deselect Cycles	Configuration tool for available options.		number of QSPCLK cycles for QSSL to remain high beween operations.

Clock Configuration

The QSPI clock is derived from PCLKA.

Pin Configuration

The following pins are available to connect to an external QSPI device:

- QSPCLK: QSPI clock output
- QSSL: QSPI slave select
- QIO0: Data 0 I/O
- QIO1: Data 1 I/O
- QIO2: Data 2 I/O
- QIO3: Data 3 I/O

Note

It is recommended to configure the pins with IOPORT_CFG_DRIVE_HIGH.

Usage Notes

QSPI Memory Mapped Access

After R_QSPI_Open() completes successfully, the QSPI flash device contents are mapped to address 0x60000000 and can be read like on-chip flash.

Limitations

Developers should be aware of the following limitations when using the QSPI driver:

- Only P305-P310 are currently supported by the J-Link driver to flash the QSPI.
- The default J-Link downloader requires the device to be in extended SPI mode (not QPI mode).

Examples

Basic Example

This is a basic example of minimal use of the QSPI in an application.

```
#define QSPI_EXAMPLE_DATA_LENGTH (1024)
uint8_t g_dest[QSPI_EXAMPLE_DATA_LENGTH];
/* Place data in the .qspi_flash section to flash it during programming. */
const uint8_t g_src[QSPI_EXAMPLE_DATA_LENGTH] BSP_PLACE_IN_SECTION(".qspi_flash") =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
/* Place code in the .code_in_qspi section to flash it during programming. */
```

```
void r_qspi_example_function(void) BSP_PLACE_IN_SECTION(".code_in_qspi")
__attribute__((noinline));
void r_qspi_example_function (void)
{
 /* Add code here. */
}
void r_qspi_basic_example (void)
{
 /* Open the QSPI instance. */
 fsp_err_t err = R_QSPI_Open(&g_qspi0_ctrl, &g_qspi0_cfg);
    assert(FSP_SUCCESS == err);
 /* (Optional) Send device specific initialization commands. */
    r_qspi_example_init();
 /* After R_QSPI_Open() and any required device specific intiialization, data can be
read directly from the QSPI flash. */
    memcpy(&g_dest[0], &g_src[0], QSPI_EXAMPLE_DATA_LENGTH);
 /* After R_QSPI_Open() and any required device specific intiialization, functions in
the QSPI flash can be called. */
    r_qspi_example_function();
}
```

Initialization Command Structure Example

This is an example of the types of commands that can be used to initialize the QSPI.

```
#define QSPI_COMMAND_WRITE_ENABLE (0x06U)
#define QSPI_COMMAND_WRITE_STATUS_REGISTER (0x01U)
#define QSPI_COMMAND_ENTER_QPI_MODE (0x38U)
#define QSPI_EXAMPLE_STATUS_REGISTER_1 (0x40)
#define QSPI_EXAMPLE_STATUS_REGISTER_2 (0x00)
static void r_qspi_example_init (void)
{
 /* Write status registers */
 /* Write one byte to enable writing to the status register, then deassert QSSL. */
    uint8_t   data[4];
```

```
 fsp_err_t err;
    data[0] = QSPI_COMMAND_WRITE_ENABLE;
    err     = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data[0], 1, false);
    assert(FSP_SUCCESS == err);
 /* Write 3 bytes, including the write status register command followed by values for
both status registers. In the
  * status registers, set QE to 1 and other bits to their default setting. After all
data is written, deassert the
  * QSSL line. */
    data[0] = QSPI_COMMAND_WRITE_STATUS_REGISTER;
    data[1] = QSPI_EXAMPLE_STATUS_REGISTER_1;
    data[2] = QSPI_EXAMPLE_STATUS_REGISTER_2;
    err     = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data[0], 3, false);
    assert(FSP_SUCCESS == err);
 /* Wait for status register to update. */
 spi_flash_status_t status;
 do
    {
       (void) R_QSPI_StatusGet(&g_qspi0_ctrl, &status);
    } while (true == status.write_in_progress);
 /* Write one byte to enter QSPI mode, then deassert QSSL. After entering QPI mode on
the device, change the SPI
  * protocol to QPI mode on the MCU peripheral. */
    data[0] = QSPI_COMMAND_ENTER_QPI_MODE;
    err     = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data[0], 1, false);
    assert(FSP_SUCCESS == err);
    (void) R_QSPI_SpiProtocolSet(&g_qspi0_ctrl, SPI_FLASH_PROTOCOL_QPI);
}
```

Reading Status Register Example (R_QSPI_DirectWrite, R_QSPI_DirectRead)

This is an example of using R_QSPI_DirectWrite followed by R_QSPI_DirectRead to send the read status register command and read back the status register from the device.

```
#define QSPI_COMMAND_READ_STATUS_REGISTER (0x05U)
```

```
void r_qspi_direct_example (void)
{
 /* Read a status register. */
 /* Write one byte to read the status register. Do not deassert QSSL. */
    uint8_t   data;
 fsp_err_t err;
    data = QSPI_COMMAND_READ_STATUS_REGISTER;
    err = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data, 1, true);
    assert(FSP_SUCCESS == err);
 /* Read one byte. After all data is read, deassert the QSSL line. */
    err = R_QSPI_DirectRead(&g_qspi0_ctrl, &data, 1);
    assert(FSP_SUCCESS == err);
 /* Status register contents are available in variable 'data'. */
}
```

Querying Device Size Example (R_QSPI_DirectWrite, R_QSPI_DirectRead)

This is an example of using R_QSPI_DirectWrite followed by R_QSPI_DirectRead to query the device size.

```
#define QSPI_EXAMPLE_COMMAND_READ_ID (0x9F)
#define QSPI_EXAMPLE_COMMAND_READ_SFDP (0x5A)
void r_qspi_size_example (void)
{
 /* Many QSPI devices support more than one way to query the device size. Consult the
datasheet for your
  * QSPI device to determine which of these methods are supported (if any). */
    uint32_t device_size_bytes;
 fsp_err_t err;
#ifdef QSPI_EXAMPLE_COMMAND_READ_ID
 /* This example shows how to get the device size by reading the manufacturer ID. */
    uint8_t data[4];
    data[0] = QSPI_EXAMPLE_COMMAND_READ_ID;
    err    = R_QSPI_DirectWrite(&g_qspi0_ctrl, &data[0], 1, true);
    assert(FSP_SUCCESS == err);
```

```
 /* Read 3 bytes. The third byte often represents the size of the QSPI, where the
size of the QSPI = 2 ^ N. */
    err = R_QSPI_DirectRead(&g_qspi0_ctrl, &data[0], 3);
    assert(FSP_SUCCESS == err);
    device_size_bytes = 1U << data[2];
 FSP_PARAMETER_NOT_USED(device_size_bytes);
#endif
#ifdef QSPI_EXAMPLE_COMMAND_READ_SFDP
 /* Read the JEDEC SFDP header to locate the JEDEC flash parameters table. Reference
JESD216 "Serial Flash
  * Discoverable Parameters (SFDP)". */
 /* Send the standard 0x5A command followed by 3 address bytes (SFDP header is at
address 0). */
    uint8_t buffer[16];
    memset(&buffer[0], 0, sizeof(buffer));
    buffer[0] = QSPI_EXAMPLE_COMMAND_READ_SFDP;
    err       = R_QSPI_DirectWrite(&g_qspi0_ctrl, &buffer[0], 4, true);
    assert(FSP_SUCCESS == err);
 /* Read out 16 bytes (1 dummy byte followed by 15 data bytes). */
    err = R_QSPI_DirectRead(&g_qspi0_ctrl, &buffer[0], 16);
    assert(FSP_SUCCESS == err);
 /* Read the JEDEC flash parameters to locate the memory size. */
 /* Send the standard 0x5A command followed by 3 address bytes (located in big endian
order at offset 0xC-0xE).
  * These bytes are accessed at 0xD-0xF because the first byte read is a dummy byte.
*/
    buffer[0] = QSPI_EXAMPLE_COMMAND_READ_SFDP;
    buffer[1] = buffer[0xF];
    buffer[2] = buffer[0xE];
    buffer[3] = buffer[0xD];
    err       = R_QSPI_DirectWrite(&g_qspi0_ctrl, &buffer[0], 4, true);
    assert(FSP_SUCCESS == err);
 /* Read out 9 bytes (1 dummy byte followed by 8 data bytes). */
    err = R_QSPI_DirectRead(&g_qspi0_ctrl, &buffer[0], 9);
```

```
    assert(FSP_SUCCESS == err);
 /* Read the memory density (located in big endian order at offset 0x4-0x7). These
bytes are accessed at 0x5-0x8
  * because the first byte read is a dummy byte. */
    uint32_t memory_density = (uint32_t) ((buffer[8] << 24) | (buffer[7] << 16) |
(buffer[6] << 8) | buffer[5]);
 if ((1U << 31) & memory_density)
    {
 /* For densities 4 gigabits and above, bit-31 is set to 1b. The field 30:0 defines
'N' where the density is
  * computed as 2^N bits (N must be >= 32). This code subtracts 3 from N to divide by
8 to get the size in
  * bytes instead of bits. */
       device_size_bytes = 1U << ((memory_density & ~(1U << 31)) - 3U);
    }
 else
    {
 /* For densities 2 gigabits or less, bit-31 is set to 0b. The field 30:0 defines the
size in bits. This
  * code divides the memory density by 8 to get the size in bytes instead of bits. */
       device_size_bytes = (memory_density / 8) + 1;
    }
 FSP_PARAMETER_NOT_USED(device_size_bytes);
#endif
}
```

Data Structures

struct	qspi_instance_ctrl_t

Enumerations

enum	qspi_qssl_min_high_level_t
enum	qspi_qspclk_div_t

Data Structure Documentation

◆ qspi_instance_ctrl_t

struct qspi_instance_ctrl_t
Instance control block. DO NOT INITIALIZE. Initialization occurs when spi_flash_api_t::open is called

Enumeration Type Documentation

◆ qspi_qssl_min_high_level_t

enum qspi_qssl_min_high_level_t

Enumerator	
QSPI_QSSL_MIN_HIGH_LEVEL_1_QSPCLK	QSSL deselected for at least 1 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_2_QSPCLK	QSSL deselected for at least 2 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_3_QSPCLK	QSSL deselected for at least 3 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_4_QSPCLK	QSSL deselected for at least 4 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_5_QSPCLK	QSSL deselected for at least 5 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_6_QSPCLK	QSSL deselected for at least 6 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_7_QSPCLK	QSSL deselected for at least 7 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_8_QSPCLK	QSSL deselected for at least 8 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_9_QSPCLK	QSSL deselected for at least 9 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_10_QSPCLK	QSSL deselected for at least 10 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_11_QSPCLK	QSSL deselected for at least 11 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_12_QSPCLK	QSSL deselected for at least 12 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_13_QSPCLK	QSSL deselected for at least 13 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_14_QSPCLK	QSSL deselected for at least 14 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_15_QSPCLK	QSSL deselected for at least 15 QSPCLK.
QSPI_QSSL_MIN_HIGH_LEVEL_16_QSPCLK	QSSL deselected for at least 16 QSPCLK.

◆ qspi_qspclk_div_t

enum qspi_qspclk_div_t	
Enumerator	
QSPI_QSPCLK_DIV_2	QSPCLK = PCLK / 2.
QSPI_QSPCLK_DIV_3	QSPCLK = PCLK / 3.
QSPI_QSPCLK_DIV_4	QSPCLK = PCLK / 4.
QSPI_QSPCLK_DIV_5	QSPCLK = PCLK / 5.
QSPI_QSPCLK_DIV_6	QSPCLK = PCLK / 6.
QSPI_QSPCLK_DIV_7	QSPCLK = PCLK / 7.
QSPI_QSPCLK_DIV_8	QSPCLK = PCLK / 8.
QSPI_QSPCLK_DIV_9	QSPCLK = PCLK / 9.
QSPI_QSPCLK_DIV_10	QSPCLK = PCLK / 10.
QSPI_QSPCLK_DIV_11	QSPCLK = PCLK / 11.
QSPI_QSPCLK_DIV_12	QSPCLK = PCLK / 12.
QSPI_QSPCLK_DIV_13	QSPCLK = PCLK / 13.
QSPI_QSPCLK_DIV_14	QSPCLK = PCLK / 14.
QSPI_QSPCLK_DIV_15	QSPCLK = PCLK / 15.
QSPI_QSPCLK_DIV_16	QSPCLK = PCLK / 16.
QSPI_QSPCLK_DIV_17	QSPCLK = PCLK / 17.
QSPI_QSPCLK_DIV_18	QSPCLK = PCLK / 18.
QSPI_QSPCLK_DIV_20	QSPCLK = PCLK / 20.
QSPI_QSPCLK_DIV_22	QSPCLK = PCLK / 22.
QSPI_QSPCLK_DIV_24	QSPCLK = PCLK / 24.
QSPI_QSPCLK_DIV_26	QSPCLK = PCLK / 26.
QSPI_QSPCLK_DIV_28	

	QSPCLK = PCLK / 28.
QSPI_QSPCLK_DIV_30	QSPCLK = PCLK / 30.
QSPI_QSPCLK_DIV_32	QSPCLK = PCLK / 32.
QSPI_QSPCLK_DIV_34	QSPCLK = PCLK / 34.
QSPI_QSPCLK_DIV_36	QSPCLK = PCLK / 36.
QSPI_QSPCLK_DIV_38	QSPCLK = PCLK / 38.
QSPI_QSPCLK_DIV_40	QSPCLK = PCLK / 40.
QSPI_QSPCLK_DIV_42	QSPCLK = PCLK / 42.
QSPI_QSPCLK_DIV_44	QSPCLK = PCLK / 44.
QSPI_QSPCLK_DIV_46	QSPCLK = PCLK / 46.
QSPI_QSPCLK_DIV_48	QSPCLK = PCLK / 48.

Function Documentation

◆ R_QSPI_Open()

fsp_err_t R_QSPI_Open (spi_flash_ctrl_t * *p_ctrl*, spi_flash_cfg_t const *const *p_cfg*)

Open the QSPI driver module. After the driver is open, the QSPI can be accessed like internal flash memory starting at address 0x60000000.

Implements spi_flash_api_t::open.

Return values

FSP_SUCCESS	Configuration was successful.
FSP_ERR_ASSERTION	The parameter p_instance_ctrl or p_cfg is NULL.
FSP_ERR_ALREADY_OPEN	Driver has already been opened with the same p_instance_ctrl.

◆ R_QSPI_Close()

fsp_err_t R_QSPI_Close (spi_flash_ctrl_t * *p_ctrl*)

Close the QSPI driver module.

Implements spi_flash_api_t::close.

Return values

FSP_SUCCESS	Configuration was successful.
FSP_ERR_ASSERTION	p_instance_ctrl is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.

◆ R_QSPI_DirectWrite()

fsp_err_t R_QSPI_DirectWrite (spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*, bool const *read_after_write*)

Writes raw data directly to the QSPI.

Implements spi_flash_api_t::directWrite.

Return values

FSP_SUCCESS	The flash was programmed successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_INVALID_MODE	This function can't be called when XIP mode is enabled.
FSP_ERR_DEVICE_BUSY	The device is busy.

◆ R_QSPI_DirectRead()

fsp_err_t R_QSPI_DirectRead (spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Reads raw data directly from the QSPI. This API can only be called after R_QSPI_DirectWrite with read_after_write set to true.

Implements spi_flash_api_t::directRead.

Return values

FSP_SUCCESS	The flash was programmed successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_INVALID_MODE	This function must be called after R_QSPI_DirectWrite with read_after_write set to true.

◆ R_QSPI_SpiProtocolSet()

fsp_err_t R_QSPI_SpiProtocolSet (spi_flash_ctrl_t * *p_ctrl*, spi_flash_protocol_t *spi_protocol*)

Sets the SPI protocol.

Implements spi_flash_api_t::spiProtocolSet.

Return values

FSP_SUCCESS	SPI protocol updated on MCU peripheral.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_INVALID_ARGUMENT	Invalid SPI protocol requested.

◆ R_QSPI_XipEnter()

fsp_err_t R_QSPI_XipEnter (spi_flash_ctrl_t * *p_ctrl*)

Enters XIP (execute in place) mode.

Implements spi_flash_api_t::xipEnter.

Return values

FSP_SUCCESS	The flash was programmed successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.

◆ R_QSPI_XipExit()

fsp_err_t R_QSPI_XipExit (spi_flash_ctrl_t * *p_ctrl*)

Exits XIP (execute in place) mode.

Implements spi_flash_api_t::xipExit.

Return values

FSP_SUCCESS	The flash was programmed successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.

◆ R_QSPI_Write()

fsp_err_t R_QSPI_Write (spi_flash_ctrl_t * *p_ctrl*, uint8_t const *const *p_src*, uint8_t *const *p_dest*, uint32_t *byte_count*)

Program a page of data to the flash.

Implements spi_flash_api_t::write.

Return values

FSP_SUCCESS	The flash was programmed successfully.
FSP_ERR_ASSERTION	p_instance_ctrl, p_dest or p_src is NULL, or byte_count crosses a page boundary.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_INVALID_MODE	This function can't be called when XIP mode is enabled.
FSP_ERR_DEVICE_BUSY	The device is busy.

◆ R_QSPI_Erase()

fsp_err_t R_QSPI_Erase (spi_flash_ctrl_t * *p_ctrl*, uint8_t *const *p_device_address*, uint32_t *byte_count*)

Erase a block or sector of flash. The byte_count must exactly match one of the erase sizes defined in spi_flash_cfg_t. For chip erase, byte_count must be SPI_FLASH_ERASE_SIZE_CHIP_ERASE.

Implements spi_flash_api_t::erase.

Return values

FSP_SUCCESS	The command to erase the flash was executed successfully.
FSP_ERR_ASSERTION	p_instance_ctrl or p_device_address is NULL, or byte_count doesn't match an erase size defined in spi_flash_cfg_t, or device is in XIP mode.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_INVALID_MODE	This function can't be called when XIP mode is enabled.
FSP_ERR_DEVICE_BUSY	The device is busy.

◆ R_QSPI_StatusGet()

fsp_err_t R_QSPI_StatusGet (spi_flash_ctrl_t * *p_ctrl*, spi_flash_status_t *const *p_status*)

Gets the write or erase status of the flash.

Implements spi_flash_api_t::statusGet.

Return values

FSP_SUCCESS	The write status is in p_status.
FSP_ERR_ASSERTION	p_instance_ctrl or p_status is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.
FSP_ERR_INVALID_MODE	This function can't be called when XIP mode is enabled.

◆ R_QSPI_BankSet()

fsp_err_t R_QSPI_BankSet (spi_flash_ctrl_t * *p_ctrl*, uint32_t *bank*)

Selects the bank to access. A bank is a 64MB sliding access window into the QSPI device flash memory space. To access chip address 0x4000000, select bank 1, then read from internal flash address 0x60000000. To access chip address 0x8001000, select bank 2, then read from internal flash address 0x60001000.

This function is not required for memory devices less than or equal to 512 Mb (64MB).

Implements spi_flash_api_t::bankSet.

Return values

FSP_SUCCESS	Bank successfully selected.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	Driver is not opened.

◆ R_QSPI_DirectTransfer()

fsp_err_t R_QSPI_DirectTransfer (spi_flash_ctrl_t * *p_ctrl*, spi_flash_direct_transfer_t *const *p_transfer*, spi_flash_direct_transfer_dir_t *direction*)

Read/Write raw data directly with the OctaFlash/OctaRAM device. Unsupported by QSPI.

Implements spi_flash_api_t::directTransfer.

Return values

FSP_ERR_UNSUPPORTED	API not supported by QSPI.

◆ R_QSPI_AutoCalibrate()

fsp_err_t R_QSPI_AutoCalibrate (spi_flash_ctrl_t * *p_ctrl*)
Auto-calibrate the OctaRAM device using the preamble pattern. Unsupported by QSPI. Implements spi_flash_api_t::autoCalibrate.

Return values

FSP_ERR_UNSUPPORTED	API not supported by QSPI

4.2.48 Realtime Clock (r_rtc)

Modules

Functions

fsp_err_t	R_RTC_Open (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)
fsp_err_t	R_RTC_Close (rtc_ctrl_t *const p_ctrl)
fsp_err_t	R_RTC_CalendarTimeSet (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)
fsp_err_t	R_RTC_CalendarTimeGet (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)
fsp_err_t	R_RTC_CalendarAlarmSet (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)
fsp_err_t	R_RTC_CalendarAlarmGet (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)
fsp_err_t	R_RTC_PeriodicIrqRateSet (rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate)
fsp_err_t	R_RTC_ErrorAdjustmentSet (rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg)
fsp_err_t	R_RTC_InfoGet (rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info)
fsp_err_t	R_RTC_CallbackSet (rtc_ctrl_t *const p_ctrl, void(*p_callback)(rtc_callback_args_t *), void const *const p_context, rtc_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the RTC peripheral on RA MCUs. This module implements the RTC Interface.

Overview

The RTC HAL module configures the RTC module and controls clock, calendar and alarm functions. A callback can be used to respond to the alarm and periodic interrupt.

Features

- RTC time and date get and set.
- RTC time and date alarm get and set.
- RTC alarm and periodic event notification.

The RTC HAL module supports three different interrupt types:

- An alarm interrupt generated on a match of any combination of year, month, day, day of the week, hour, minute or second
- A periodic interrupt generated every 2, 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, or 1/256 second(s)
- A carry interrupt is used internally when reading time from the RTC calender to get accurant time readings.

Note

See section "23.3.5 Reading 64-Hz Counter and Time" of the RA6M3 manual R01UH0886EJ0100 for more details.

A user-defined callback function can be registered (in the rtc_api_t::open API call) and will be called from the interrupt service routine (ISR) for alarm and periodic interrupt. When called, it is passed a pointer to a structure (rtc_callback_args_t) that holds a user-defined context pointer and an indication of which type of interrupt was fired.

Date and Time validation

"Parameter Checking" needs to be enabled if date and time validation is required for calendarTimeSet and calendarAlarmSet APIs. If "Parameter Checking" is enabled, the 'day of the week' field is automatically calculated and updated by the driver for the provided date. When using the calendarAlarmSet API, only the fields which have their corresponding match flag set are written to the registers. Other register fields are reset to default value.

Sub-Clock error adjustment (Time Error Adjustment Function)

The time error adjustment function is used to correct errors, running fast or slow, in the time caused by variation in the precision of oscillation by the sub-clock oscillator. Because 32,768 cycles of the sub-clock oscillator constitute 1 second of operation when the sub-clock oscillator is selected, the clock runs fast if the sub-clock frequency is high and slow if the sub-clock frequency is low. The time error adjustment functions include:

- Automatic adjustment
- Adjustment by software

The error adjustment is reset every time RTC is reconfigured or time is set.

Note

RTC driver configurations do not do error adjustment internally while initiliazing the driver. Application must *make calls to the error adjustment api's for desired adjustment. See section 26.3.8 "Time Error Adjustment Function" of the RA6M3 manual R01UH0886EJ0100) for more details on this feature*

Configuration

Build Time Configurations for r_rtc

The following build time configurations are defined in fsp_cfg/r_rtc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Timers > Realtime Clock (r_rtc)

This module can be added to the Stacks tab via New Stack > Timers > Realtime Clock (r_rtc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rtc0	Module name.
Clock Source	• Sub-Clock • LOCO	LOCO	Select the RTC clock source.
Frequency Comparision Value (LOCO)	Value must be a positive integer between 7 and 511	255	Frequency comparison value when using LOCO
Automatic Adjustment Mode	• Enabled • Disabled	Enabled	Enable/ Disable the Error Adjustment mode
Automatic Adjustment Period	• 10 Seconds • 1 Minute • NONE	10 Seconds	Select the Error Adjustment Period for Automatic Adjustment
Adjustment Type (Plus-Minus)	• NONE • Addition • Subtraction	NONE	Select the Error Adjustment type
Error Adjustment Value	Value must be a positive integer less than equal to 63	0	Specify the Adjustment Value (the number of sub-clock cycles) from the prescaler
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).

Alarm Interrupt Priority	MCU Specific Options	Select the alarm interrupt priority.
Period Interrupt Priority	MCU Specific Options	Select the period interrupt priority.
Carry Interrupt Priority	MCU Specific Options	Select the carry interrupt priority.

Note

See 23.2.20 Frequency Register (RFRH/RFRL) of the RA6M3 manual R01UH0886EJ0100) for more details

Interrupt Configuration

To activate interrupts for the RTC module, the desired interrupts must be enabled, The underlying implementation will be expected to handle any interrupts it can support and notify higher layers via callback.

Clock Configuration

The RTC HAL module can use the following clock sources:

- LOCO (Low Speed On-Chip Oscillator) with less accuracy
- Sub-clock oscillator with increased accuracy

The LOCO is the default selection during configuration.

Pin Configuration

This module does not use I/O pins.

Usage Notes

System Initialization

- RTC driver does not start the sub-clock. The application is responsible for ensuring required clocks are started and stable before accessing MCU peripheral registers.

Warning

The subclock can take seconds to stabilize. The RA startup code does not wait for subclock stabilization unless the subclock is the main clock source. When running AGT or RTC off the subclock, the application must ensure the subclock is stable before starting operation.

- Carry interrupt priority must be set to avoid incorrect time returned from calendarTimeGet API during roll-over.
- Even when only running in Periodic Interrupt mode R_RTC_CalendarTimeSet must be called successfully to start the RTC.

Limitations

Developers should be aware of the following limitations when using the RTC:

- R_RTC operates in 24-hour mode.
- Binary-count mode is not supported.
- The R_RTC_CalendarTimeGet() cannot be used from an interrupt that has higher priority

than the carry interrupt. Also, it must not be called with interrupts disabled globally, as this API internally uses carry interrupt for its processing. API may return incorrect time if this is done.

Examples

RTC Basic Example

This is a basic example of minimal use of the RTC in an application.

```
/* rtc_time_t is an alias for the C Standard time.h struct 'tm' */
rtc_time_t set_time =
{
    .tm_sec = 10,
    .tm_min = 11,
    .tm_hour = 12,
    .tm_mday = 6,
    .tm_wday = 3,
    .tm_mon = 11,
    .tm_year = YEARS_SINCE_1900,
};
rtc_time_t get_time;
void rtc_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the RTC module */
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the calendar time */
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Get the calendar time */
 R_RTC_CalendarTimeGet(&g_rtc0_ctrl, &get_time);
}
```

RTC Periodic interrupt example

This is an example of periodic interrupt in RTC.

```
void rtc_periodic_irq_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the RTC module*/
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* R_RTC_CalendarTimeSet must be called at least once to start the RTC */
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time);
 /* Set the periodic interrupt rate to 1 second */
 R_RTC_PeriodicIrqRateSet(&g_rtc0_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
 /* Wait for the periodic interrupt */
 while (1)
    {
 /* Wait for interrupt */
    }
}
```

RTC Alarm interrupt example

This is an example of alarm interrupt in RTC.

```
void rtc_alarm_irq_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /*Initialize the RTC module*/
    err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time1.time);
 R_RTC_CalendarAlarmSet(&g_rtc0_ctrl, &set_time1);
 /* Wait for the Alarm interrupt */
 while (1)
    {
 /* Wait for interrupt */
```

```
    }
}
```

RTC Error Adjustment example

This is an example of modifying error adjustment in RTC.

```
void rtc_erroradj_example ()
{
 fsp_err_t err = FSP_SUCCESS;
 /*Initialize the RTC module*/
 R_RTC_Open(&g_rtc0_ctrl, &g_rtc1_cfg);
 R_RTC_CalendarTimeSet(&g_rtc0_ctrl, &set_time1.time);
 /* Modify Error Adjustment after RTC is running */
    err = R_RTC_ErrorAdjustmentSet(&g_rtc0_ctrl, &err_cfg2);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	rtc_instance_ctrl_t

Data Structure Documentation

◆ rtc_instance_ctrl_t

struct rtc_instance_ctrl_t

Channel control block. DO NOT INITIALIZE. Initialization occurs when rtc_api_t::open is called

Data Fields

uint32_t	open
	Whether or not driver is open.
const rtc_cfg_t *	p_cfg
	Pointer to initial configurations.
volatile bool	carry_isr_triggered
	Was the carry isr triggered.

Function Documentation

◆ R_RTC_Open()

fsp_err_t R_RTC_Open (rtc_ctrl_t *const *p_ctrl*, rtc_cfg_t const *const *p_cfg*)

Opens and configures the RTC driver module. Implements rtc_api_t::open. Configuration includes clock source, and interrupt callback function.

Example:

```
/* Initialize the RTC module */
   err = R_RTC_Open(&g_rtc0_ctrl, &g_rtc0_cfg);
```

Return values

FSP_SUCCESS	Initialization was successful and RTC has started.
FSP_ERR_ASSERTION	Invalid p_ctrl or p_cfg pointer.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_INVALID_ARGUMENT	Invalid time parameter field.

◆ R_RTC_Close()

fsp_err_t R_RTC_Close (rtc_ctrl_t *const *p_ctrl*)

Close the RTC driver. Implements rtc_api_t::close

Return values

FSP_SUCCESS	De-Initialization was successful and RTC driver closed.
FSP_ERR_ASSERTION	Invalid p_ctrl.
FSP_ERR_NOT_OPEN	Driver not open already for close.

◆ R_RTC_CalendarTimeSet()

fsp_err_t R_RTC_CalendarTimeSet (rtc_ctrl_t *const *p_ctrl*, rtc_time_t *const *p_time*)

Set the calendar time.

Implements rtc_api_t::calendarTimeSet.

Return values

FSP_SUCCESS	Calendar time set operation was successful.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.
FSP_ERR_INVALID_ARGUMENT	Invalid time parameter field.

◆ R_RTC_CalendarTimeGet()

fsp_err_t R_RTC_CalendarTimeGet (rtc_ctrl_t *const *p_ctrl*, rtc_time_t *const *p_time*)

Get the calendar time.

Warning

Do not call this function from a critical section or from an interrupt with higher priority than the carry interrupt, or the time returned may be inaccurate.

Implements rtc_api_t::calendarTimeGet

Return values

FSP_SUCCESS	Calendar time get operation was successful.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.
FSP_ERR_IRQ_BSP_DISABLED	User IRQ parameter not valid

◆ R_RTC_CalendarAlarmSet()

fsp_err_t R_RTC_CalendarAlarmSet (rtc_ctrl_t *const *p_ctrl*, rtc_alarm_time_t *const *p_alarm*)

Set the calendar alarm time.

Implements rtc_api_t::calendarAlarmSet.

Precondition
The calendar counter must be running before the alarm can be set.

Return values

FSP_SUCCESS	Calendar alarm time set operation was successful.
FSP_ERR_INVALID_ARGUMENT	Invalid time parameter field.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.
FSP_ERR_IRQ_BSP_DISABLED	User IRQ parameter not valid

◆ R_RTC_CalendarAlarmGet()

fsp_err_t R_RTC_CalendarAlarmGet (rtc_ctrl_t *const *p_ctrl*, rtc_alarm_time_t *const *p_alarm*)

Get the calendar alarm time.

Implements rtc_api_t::calendarAlarmGet

Return values

FSP_SUCCESS	Calendar alarm time get operation was successful.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.

◆ R_RTC_PeriodicIrqRateSet()

fsp_err_t R_RTC_PeriodicIrqRateSet (rtc_ctrl_t *const *p_ctrl*, rtc_periodic_irq_select_t const *rate*)

Set the periodic interrupt rate and enable periodic interrupt.

Implements rtc_api_t::periodicIrqRateSet

Note

To start the RTC R_RTC_CalendarTimeSet must be called at least once.

Example:

```
/* Set the periodic interrupt rate to 1 second */
R_RTC_PeriodicIrqRateSet(&g_rtc0_ctrl, RTC_PERIODIC_IRQ_SELECT_1_SECOND);
```

Return values

FSP_SUCCESS	The periodic interrupt rate was successfully set.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.
FSP_ERR_IRQ_BSP_DISABLED	User IRQ parameter not valid

◆ R_RTC_ErrorAdjustmentSet()

fsp_err_t R_RTC_ErrorAdjustmentSet (rtc_ctrl_t *const *p_ctrl*, rtc_error_adjustment_cfg_t const *const *err_adj_cfg*)

This function sets time error adjustment

Implements rtc_api_t::errorAdjustmentSet

Return values

FSP_SUCCESS	Time error adjustment successful.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open for operation.
FSP_ERR_UNSUPPORTED	The clock source is not sub-clock.
FSP_ERR_INVALID_ARGUMENT	Invalid error adjustment value.

◆ R_RTC_InfoGet()

fsp_err_t R_RTC_InfoGet (rtc_ctrl_t *const *p_ctrl*, rtc_info_t *const *p_rtc_info*)

Set RTC clock source and running status information ad store it in provided pointer p_rtc_info

Implements rtc_api_t::infoGet

Return values

FSP_SUCCESS	Get information Successful.
FSP_ERR_ASSERTION	Invalid input argument.
FSP_ERR_NOT_OPEN	Driver not open already for operation.

◆ R_RTC_CallbackSet()

fsp_err_t R_RTC_CallbackSet (rtc_ctrl_t *const *p_ctrl*, void(*)(rtc_callback_args_t *) *p_callback*, void const *const *p_context*, rtc_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements rtc_api_t::callbackSet

Return values

FSP_SUCCESS	Baud rate was successfully changed.
FSP_ERR_ASSERTION	Pointer to RTC control block is NULL or the RTC is not configured to use the internal clock.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.49 Secure Crypto Engine (r_sce_protected)

Modules

Functions

fsp_err_t	R_SCE_Open (sce_ctrl_t *const p_ctrl, sce_cfg_t const *const p_cfg)
fsp_err_t	R_SCE_Close (sce_ctrl_t *const p_ctrl)
fsp_err_t	R_SCE_SoftwareReset (void)

fsp_err_t	R_SCE_AES128_WrappedKeyGenerate (sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_AES256_WrappedKeyGenerate (sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_RSA1024_WrappedKeyPairGenerate (sce_rsa1024_wrapped_pair_key_t *wrapped_pair_key)
fsp_err_t	R_SCE_RSA2048_WrappedKeyPairGenerate (sce_rsa2048_wrapped_pair_key_t *wrapped_pair_key)
fsp_err_t	R_SCE_ECC_secp192r1_WrappedKeyPairGenerate (sce_ecc_wrapped_pair_key_t *wrapped_pair_key)
fsp_err_t	R_SCE_ECC_secp224r1_WrappedKeyPairGenerate (sce_ecc_wrapped_pair_key_t *wrapped_pair_key)
fsp_err_t	R_SCE_ECC_secp256r1_WrappedKeyPairGenerate (sce_ecc_wrapped_pair_key_t *wrapped_pair_key)
fsp_err_t	R_SCE_ECC_secp384r1_WrappedKeyPairGenerate (sce_ecc_wrapped_pair_key_t *wrapped_pair_key)
fsp_err_t	R_SCE_RandomNumberGenerate (uint32_t *random)
fsp_err_t	R_SCE_AES128_EncryptedKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_AES256_EncryptedKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_SHA256HMAC_EncryptedKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_hmac_sha_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_RSA1024_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa1024_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_RSA1024_EncryptedPrivateKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa1024_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_RSA2048_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa2048_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_RSA2048_EncryptedPrivateKeyWrap (uint8_t *initial_vector,

	uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa2048_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_RSA3072_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa3072_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_RSA4096_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa4096_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECC_secp192r1_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECC_secp224r1_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECC_secp256r1_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECC_secp384r1_EncryptedPublicKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECC_secp192r1_EncryptedPrivateKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECC_secp224r1_EncryptedPrivateKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECC_secp256r1_EncryptedPrivateKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECC_secp384r1_EncryptedPrivateKeyWrap (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_AES128ECB_EncryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_AES128ECB_EncryptUpdate (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)
fsp_err_t	R_SCE_AES128ECB_EncryptFinal (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)

fsp_err_t	R_SCE_AES128ECB_DecryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_AES128ECB_DecryptUpdate (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)
fsp_err_t	R_SCE_AES128ECB_DecryptFinal (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)
fsp_err_t	R_SCE_AES256ECB_EncryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_AES256ECB_EncryptUpdate (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)
fsp_err_t	R_SCE_AES256ECB_EncryptFinal (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)
fsp_err_t	R_SCE_AES256ECB_DecryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_AES256ECB_DecryptUpdate (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)
fsp_err_t	R_SCE_AES256ECB_DecryptFinal (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)
fsp_err_t	R_SCE_AES128CBC_EncryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)
fsp_err_t	R_SCE_AES128CBC_EncryptUpdate (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)
fsp_err_t	R_SCE_AES128CBC_EncryptFinal (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)
fsp_err_t	R_SCE_AES128CBC_DecryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)
fsp_err_t	R_SCE_AES128CBC_DecryptUpdate (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)
fsp_err_t	R_SCE_AES128CBC_DecryptFinal (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)
fsp_err_t	R_SCE_AES256CBC_EncryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)
fsp_err_t	R_SCE_AES256CBC_EncryptUpdate (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

fsp_err_t	R_SCE_AES256CBC_EncryptFinal (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)
fsp_err_t	R_SCE_AES256CBC_DecryptInit (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)
fsp_err_t	R_SCE_AES256CBC_DecryptUpdate (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)
fsp_err_t	R_SCE_AES256CBC_DecryptFinal (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)
fsp_err_t	R_SCE_AES128GCM_EncryptInit (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)
fsp_err_t	R_SCE_AES128GCM_EncryptUpdate (sce_gcm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_data_length, uint8_t *aad, uint32_t aad_length)
fsp_err_t	R_SCE_AES128GCM_EncryptFinal (sce_gcm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_data_length, uint8_t *atag)
fsp_err_t	R_SCE_AES128GCM_DecryptInit (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)
fsp_err_t	R_SCE_AES128GCM_DecryptUpdate (sce_gcm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_data_length, uint8_t *aad, uint32_t aad_length)
fsp_err_t	R_SCE_AES128GCM_DecryptFinal (sce_gcm_handle_t *handle, uint8_t *plain, uint32_t *plain_data_length, uint8_t *atag, uint32_t atag_length)
fsp_err_t	R_SCE_AES256GCM_EncryptInit (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)
fsp_err_t	R_SCE_AES256GCM_EncryptUpdate (sce_gcm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_data_length, uint8_t *aad, uint32_t aad_length)
fsp_err_t	R_SCE_AES256GCM_EncryptFinal (sce_gcm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_data_length, uint8_t *atag)
fsp_err_t	R_SCE_AES256GCM_DecryptInit (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)

fsp_err_t	R_SCE_AES256GCM_DecryptUpdate (sce_gcm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_data_length, uint8_t *aad, uint32_t aad_length)
fsp_err_t	R_SCE_AES256GCM_DecryptFinal (sce_gcm_handle_t *handle, uint8_t *plain, uint32_t *plain_data_length, uint8_t *atag, uint32_t atag_length)
fsp_err_t	R_SCE_AES128CCM_EncryptInit (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t adata_length, uint32_t payload_length, uint32_t mac_length)
fsp_err_t	R_SCE_AES128CCM_EncryptUpdate (sce_ccm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)
fsp_err_t	R_SCE_AES128CCM_EncryptFinal (sce_ccm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length, uint8_t *mac, uint32_t mac_length)
fsp_err_t	R_SCE_AES128CCM_DecryptInit (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t adata_length, uint32_t payload_length, uint32_t mac_length)
fsp_err_t	R_SCE_AES128CCM_DecryptUpdate (sce_ccm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)
fsp_err_t	R_SCE_AES128CCM_DecryptFinal (sce_ccm_handle_t *handle, uint8_t *plain, uint32_t *plain_length, uint8_t *mac, uint32_t mac_length)
fsp_err_t	R_SCE_AES256CCM_EncryptInit (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t adata_length, uint32_t payload_length, uint32_t mac_length)
fsp_err_t	R_SCE_AES256CCM_EncryptUpdate (sce_ccm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)
fsp_err_t	R_SCE_AES256CCM_EncryptFinal (sce_ccm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length, uint8_t *mac, uint32_t mac_length)
fsp_err_t	R_SCE_AES256CCM_DecryptInit (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t adata_length, uint32_t payload_length, uint32_t mac_length)
fsp_err_t	R_SCE_AES256CCM_DecryptUpdate (sce_ccm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)
fsp_err_t	R_SCE_AES256CCM_DecryptFinal (sce_ccm_handle_t *handle, uint8_t *plain, uint32_t *plain_length, uint8_t *mac, uint32_t mac_length)

fsp_err_t	R_SCE_AES128CMAC_GenerateInit (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_AES128CMAC_GenerateUpdate (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)
fsp_err_t	R_SCE_AES128CMAC_GenerateFinal (sce_cmac_handle_t *handle, uint8_t *mac)
fsp_err_t	R_SCE_AES128CMAC_VerifyInit (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_AES128CMAC_VerifyUpdate (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)
fsp_err_t	R_SCE_AES128CMAC_VerifyFinal (sce_cmac_handle_t *handle, uint8_t *mac, uint32_t mac_length)
fsp_err_t	R_SCE_AES256CMAC_GenerateInit (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_AES256CMAC_GenerateUpdate (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)
fsp_err_t	R_SCE_AES256CMAC_GenerateFinal (sce_cmac_handle_t *handle, uint8_t *mac)
fsp_err_t	R_SCE_AES256CMAC_VerifyInit (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_AES256CMAC_VerifyUpdate (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)
fsp_err_t	R_SCE_AES256CMAC_VerifyFinal (sce_cmac_handle_t *handle, uint8_t *mac, uint32_t mac_length)
fsp_err_t	R_SCE_SHA256_Init (sce_sha_md5_handle_t *handle)
fsp_err_t	R_SCE_SHA256_Update (sce_sha_md5_handle_t *handle, uint8_t *message, uint32_t message_length)
fsp_err_t	R_SCE_SHA256_Final (sce_sha_md5_handle_t *handle, uint8_t *digest, uint32_t *digest_length)
fsp_err_t	R_SCE_SHA256HMAC_GenerateInit (sce_hmac_sha_handle_t *handle, sce_hmac_sha_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_SHA256HMAC_GenerateUpdate (sce_hmac_sha_handle_t *handle, uint8_t *message, uint32_t message_length)

fsp_err_t	R_SCE_SHA256HMAC_GenerateFinal (sce_hmac_sha_handle_t *handle, uint8_t *mac)
fsp_err_t	R_SCE_SHA256HMAC_VerifyInit (sce_hmac_sha_handle_t *handle, sce_hmac_sha_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_SHA256HMAC_VerifyUpdate (sce_hmac_sha_handle_t *handle, uint8_t *message, uint32_t message_length)
fsp_err_t	R_SCE_SHA256HMAC_VerifyFinal (sce_hmac_sha_handle_t *handle, uint8_t *mac, uint32_t mac_length)
fsp_err_t	R_SCE_RSASSA_PKCS1024_SignatureGenerate (sce_rsa_byte_data_t *message_hash, sce_rsa_byte_data_t *signature, sce_rsa1024_private_wrapped_key_t *wrapped_key, uint8_t hash_type)
fsp_err_t	R_SCE_RSASSA_PKCS1024_SignatureVerify (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa1024_public_wrapped_key_t *wrapped_key, uint8_t hash_type)
fsp_err_t	R_SCE_RSASSA_PKCS2048_SignatureGenerate (sce_rsa_byte_data_t *message_hash, sce_rsa_byte_data_t *signature, sce_rsa2048_private_wrapped_key_t *wrapped_key, uint8_t hash_type)
fsp_err_t	R_SCE_RSASSA_PKCS2048_SignatureVerify (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa2048_public_wrapped_key_t *wrapped_key, uint8_t hash_type)
fsp_err_t	R_SCE_RSASSA_PKCS3072_SignatureVerify (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa3072_public_wrapped_key_t *wrapped_key, uint8_t hash_type)
fsp_err_t	R_SCE_RSASSA_PKCS4096_SignatureVerify (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa4096_public_wrapped_key_t *wrapped_key, uint8_t hash_type)
fsp_err_t	R_SCE_RSAES_PKCS1024_Encrypt (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa1024_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_RSAES_PKCS1024_Decrypt (sce_rsa_byte_data_t *cipher, sce_rsa_byte_data_t *plain, sce_rsa1024_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_RSAES_PKCS2048_Encrypt (sce_rsa_byte_data_t *plain,

	sce_rsa_byte_data_t *cipher, sce_rsa2048_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_RSAES_PKCS2048_Decrypt (sce_rsa_byte_data_t *cipher, sce_rsa_byte_data_t *plain, sce_rsa2048_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_RSAES_PKCS3072_Encrypt (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa3072_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_RSAES_PKCS4096_Encrypt (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa4096_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECDSA_secp192r1_SignatureGenerate (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECDSA_secp224r1_SignatureGenerate (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECDSA_secp256r1_SignatureGenerate (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECDSA_secp384r1_SignatureGenerate (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECDSA_secp192r1_SignatureVerify (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECDSA_secp224r1_SignatureVerify (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECDSA_secp256r1_SignatureVerify (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECDSA_secp384r1_SignatureVerify (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECDH_secp256r1_Init (sce_ecdh_handle_t *handle, uint32_t key_type, uint32_t use_key_id)
fsp_err_t	R_SCE_ECDH_secp256r1_PublicKeySign (sce_ecdh_handle_t *handle,

	sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, sce_ecc_private_wrapped_key_t *ecc_private_wrapped_key, uint8_t *public_key, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECDH_secp256r1_PublicKeyVerify (sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, uint8_t *public_key_data, sce_ecdsa_byte_data_t *signature, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECDH_secp256r1_PublicKeyReadWithoutSignature (sce_ecdh_handle_t *handle, uint8_t *public_key_data, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_ECDH_secp256r1_SharedSecretCalculate (sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, sce_ecc_private_wrapped_key_t *ecc_private_wrapped_key, sce_ecdh_wrapped_key_t *shared_secret_wrapped_key)
fsp_err_t	R_SCE_ECDH_secp256r1_KeyDerivation (sce_ecdh_handle_t *handle, sce_ecdh_wrapped_key_t *shared_secret_wrapped_key, uint32_t key_type, uint32_t kdf_type, uint8_t *other_info, uint32_t other_info_length, sce_hmac_sha_wrapped_key_t *salt_wrapped_key, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_TLS_RootCertificateRSA2048PublicKeyInstall (uint8_t *encrypted_provisioning_key, uint8_t *initial_vector, uint8_t *encrypted_key, sce_tls_ca_certification_public_wrapped_key_t *wrapped_key)
fsp_err_t	R_SCE_TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate (sce_tls_p256_ecc_wrapped_key_t *tls_p256_ecc_wrapped_key, uint8_t *ephemeral_ecdh_public_key)
fsp_err_t	R_SCE_TLS_RootCertificateVerify (uint32_t public_key_type, uint8_t *certificate, uint32_t certificate_length, uint32_t public_key_n_start_position, uint32_t public_key_n_end_position, uint32_t public_key_e_start_position, uint32_t public_key_e_end_position, uint8_t *signature, uint32_t *encrypted_root_public_key)
fsp_err_t	R_SCE_TLS_CertificateVerify (uint32_t public_key_type, uint32_t *encrypted_input_public_key, uint8_t *certificate, uint32_t certificate_length, uint8_t *signature, uint32_t public_key_n_start_position, uint32_t public_key_n_end_position, uint32_t public_key_e_start_position, uint32_t public_key_e_end_position, uint32_t *encrypted_output_public_key)
fsp_err_t	R_SCE_TLS_PreMasterSecretEncryptWithRSA2048 (uint32_t *encrypted_public_key, uint32_t *sce_pre_master_secret, uint8_t *encrypted_pre_master_secret)

fsp_err_t	R_SCE_TLS_PreMasterSecretGenerateForRSA2048 (uint32_t *sce_pre_master_secret)
fsp_err_t	R_SCE_TLS_MasterSecretGenerate (uint32_t select_cipher_suite, uint32_t *sce_pre_master_secret, uint8_t *client_random, uint8_t *server_random, uint32_t *sce_master_secret)
fsp_err_t	R_SCE_TLS_SessionKeyGenerate (uint32_t select_cipher_suite, uint32_t *sce_master_secret, uint8_t *client_random, uint8_t *server_random, uint8_t *nonce_explicit, sce_hmac_sha_wrapped_key_t *client_mac_wrapped_key, sce_hmac_sha_wrapped_key_t *server_mac_wrapped_key, sce_aes_wrapped_key_t *client_crypto_wrapped_key, sce_aes_wrapped_key_t *server_crypto_wrapped_key, uint8_t *client_initial_vector, uint8_t *server_initial_vector)
fsp_err_t	R_SCE_TLS_VerifyDataGenerate (uint32_t select_verify_data, uint32_t *sce_master_secret, uint8_t *hand_shake_hash, uint8_t *verify_data)
fsp_err_t	R_SCE_TLS_ServerKeyExchangeVerify (uint32_t public_key_type, uint8_t *client_random, uint8_t *server_random, uint8_t *server_ephemeral_ecdh_public_key, uint8_t *server_key_exchange_signature, uint32_t *encrypted_public_key, uint32_t *encrypted_ephemeral_ecdh_public_key)
fsp_err_t	R_SCE_TLS_PreMasterSecretGenerateForECC_secp256r1 (uint32_t *encrypted_public_key, sce_tls_p256_ecc_wrapped_key_t *tls_p256_ecc_wrapped_key, uint32_t *sce_pre_master_secret)

Detailed Description

Driver for the Secure Crypto Engine (SCE9) on RA MCUs.

Overview

This module provides SCE functions in protected mode.

Note

For a detailed description of the different SCE9 operating modes, refer to Application Note R11AN0498.

HW Overview

Crypto Peripheral version	Devices
SCE9 (Protected mode)	RA4M2, RA4M3, RA6M4, RA6M5

Features

The SCE module supports for the following features.

- Cryptography
  - Symmetric Encryption/Decryption

        - AES
            - ECB 128/256bit
            - CBC 128/256bit
            - GCM 128/256bit
            - CCM 128/256bit
    - Asymmetric Encryption/Decryption
        - RSA
            - RSAES-PKCS1-V1_5 1024/2048bit
            - RSAES-PKCS1-V1_5 3072/4096bit (Encryption only)
            - RSASSA-PKCS1-V1_5 1024/2048bit
            - RSASSA-PKCS1-V1_5 3072/4096bit (Verification only)
        - ECC
            - ECDSA secp192r1/secp224r1/secp256r1/secp384r1
            - ECDH secp192r1/secp224r1/secp256r1/secp384r1
    - Hash Functions
        - SHA-2
            - SHA-256
- Message Authentication Code
    - HMAC-SHA256bit
    - AES-CMAC 128/256bit
- Key Support
    - AES 128/256bit
    - RSA 1024/2048bit
    - RSA 3072/4096bit (public key only)
    - ECC secp192r1/secp224r1/secp256r1/secp384r1
    - HMAC-SHA256bit
- TRNG

Configuration

Clock Configuration

This module does not require a specific clock configuration.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Getting Started: Creating a SCE Protected Mode Project

Start by creating a new project in e2 studio or RA SC. On the Stacks tab, add New > Driver > Crypto > SCE Protected Mode. For information on how to install and update secure keys, refer to the Application Note R11AN0496.

Limitations

Usage of R_SCE_ECDSA_secp384r1_SignatureGenerate/Verify

The SCE does not support SHA-384 in hardware, so the APIs listed below require the user to create a SHA-384 function for signature generation and verification. To use the APIs listed below, enable SCE_USER_SHA_384_ENABLED on RA Smart Configurator and prepare a function called SCE_USER_SHA_384_FUNCTION. The interface of SCE_USER_SHA_384_FUNCTION, which is called by the following APIs, is described below.

- R_SCE_ECDSA_secp384r1_SignatureGenerate()
- R_SCE_ECDSA_secp384r1_SignatureVerify()

SCE_USER_SHA_384_FUNCTION()

```
uint32_t SCE_USER_SHA_384_FUNCTION(uint8_t * message, uint8_t * digest, uint32_t
message_length)
```

SHA-384 hash calculation is performed for an area extending the number of bytes specified by the argument message_length from the address specified by the argument message. The calculation result should be stored at the address specified by the argument digest.

Parameters

message	[in] Start address of message
digest	[in,out] address for storing hash calculation result (48 bytes)
message_length	[in] Effective byte count of message

Return values

0	Hash value stored successfully.
others	Storing of hash value failed.

Examples

AES Example

This is an example of AES-256 encryption and decryption.

```
#include <string.h>
#include "r_sce.h"
#define BLOCK 16
void r_sce_example_aes();
sce_instance_ctrl_t sce_ctrl;
sce_cfg_t sce_cfg =
{
    .lifecycle = SCE_SSD
};
static uint8_t plain[BLOCK * 2] =
```

```
{
    0x52, 0x65, 0x6e, 0x65, 0x73, 0x61, 0x73, 0x20, 0x45, 0x6c, 0x65, 0x63, 0x74,
0x72, 0x6f, 0x6e,
    0x69, 0x63, 0x73, 0x20, 0x43, 0x6f, 0x72, 0x70, 0x6f, 0x72, 0x61, 0x74, 0x69,
0x6f, 0x6e, 0x00
};
void r_sce_example_aes ()
{
 sce_aes_handle_t      handle;
 sce_aes_wrapped_key_t wrapped_key;
    uint8_t               cipher_calculated[32] = {0};
    uint8_t               plain_calculated[32] = {0};
    uint32_t             dummy;
 /* SCE power on */
 R_SCE_Open(&sce_ctrl, &sce_cfg);
 /* Generate a random key */
 R_SCE_AES256_WrappedKeyGenerate(&wrapped_key);
 /* Encrypt a plain text */
 R_SCE_AES256ECB_EncryptInit(&handle, &wrapped_key);
 R_SCE_AES256ECB_EncryptUpdate(&handle, plain, cipher_calculated, BLOCK * 2);
 R_SCE_AES256ECB_EncryptFinal(&handle, cipher_calculated, &dummy);
 /* Decrypt a cipher text using same key as Encryption */
 R_SCE_AES256ECB_DecryptInit(&handle, &wrapped_key);
 R_SCE_AES256ECB_DecryptUpdate(&handle, cipher_calculated, plain_calculated, BLOCK *
2);
 R_SCE_AES256ECB_DecryptFinal(&handle, plain_calculated, &dummy);
 /* SCE power off */
 R_SCE_Close(&sce_ctrl);
 /* Compare plain and plain_calculated */
 if (memcmp(plain, plain_calculated, BLOCK * 2))
    {
 while (1)
      {
 /* plain and plain_calculated are different (incorrect) */
```

```
      }
   }
 else
   {
 while (1)
     {
 /* plain and plain_calculated are the same (correct) */
      }
   }
}
```

Data Structures

struct	sce_instance_ctrl_t

Data Structure Documentation

◆ sce_instance_ctrl_t

struct sce_instance_ctrl_t
SCE private control block. DO NOT MODIFY. Initialization occurs when R_SCE_Open() is called.

Function Documentation

◆ R_SCE_Open()

fsp_err_t R_SCE_Open (sce_ctrl_t *const *p_ctrl*, sce_cfg_t const *const *p_cfg*)

Enables use of SCE functionality.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	The error-detection self-test failed to terminate normally.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_RETRY	Indicates that an entropy evaluation failure occurred. Run the function again.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

Note

The valid pre-run state is SCE disabled. The pre-run state is SCE Disabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_Close()

fsp_err_t R_SCE_Close (sce_ctrl_t *const *p_ctrl*)

Stops supply of power to the SCE.

Parameters

[in]	p_ctrl	Pointer to control structure.

Return values

FSP_SUCCESS	Normal termination

Note

The pre-run state is any state. After the function runs the state transitions to SCE Disabled State.

◆ R_SCE_SoftwareReset()

fsp_err_t R_SCE_SoftwareReset (void)

Software reset to SCE.

Reverts the state to the SCE initial state.

Return values

FSP_SUCCESS	Normal termination

Note

The pre-run state is any state. After the function runs the state transitions to SCE Disabled State.

◆ R_SCE_AES128_WrappedKeyGenerate()

fsp_err_t R_SCE_AES128_WrappedKeyGenerate (sce_aes_wrapped_key_t * *wrapped_key*)

This API outputs 128-bit AES wrapped key from a random number.

This API generates a wrapped key from a random number in the SCE. Accordingly, user key input is unnecessary. By encrypting data using the wrapped key is output by this API, dead copying of data can be prevented.

Parameters

[in,out]	wrapped_key	128-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Disabled State.

◆ R_SCE_AES256_WrappedKeyGenerate()

fsp_err_t R_SCE_AES256_WrappedKeyGenerate (sce_aes_wrapped_key_t * *wrapped_key*)

This API outputs 256-bit AES wrapped key from a random number.

This API generates a wrapped key from a random number in the SCE. Accordingly, user key input is unnecessary. By encrypting data using the wrapped key is output by this API, dead copying of data can be prevented.

Parameters

[in,out]	wrapped_key	256-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Disabled State.

◆ R_SCE_RSA1024_WrappedKeyPairGenerate()

fsp_err_t R_SCE_RSA1024_WrappedKeyPairGenerate (sce_rsa1024_wrapped_pair_key_t * *wrapped_pair_key*)

This API outputs a wrapped key pair for a 1024-bit RSA public key and private key pair. These keys are generated from a random value produced internally by the SCE. Consequently, there is no need to input a user key. Dead copying of data can be prevented by encrypting the data using the wrapped key output by this API. A public wrapped key is generated by wrapped_pair_key->pub_key, and a private wrapped key is generated by wrapped_pair_key->priv_key. As the public key exponent, only 0x00010001 is generated.

Parameters

[in,out]	wrapped_pair_key	User key index for RSA 1024-bit public key and private key pair

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred. Key generation failed.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSA2048_WrappedKeyPairGenerate()

fsp_err_t R_SCE_RSA2048_WrappedKeyPairGenerate (sce_rsa2048_wrapped_pair_key_t * *wrapped_pair_key*)

This API outputs a wrapped key pair for a 2048-bit RSA public key and private key pair. These keys are generated from a random value produced internally by the SCE. Consequently, there is no need to input a user key. Dead copying of data can be prevented by encrypting the data using the wrapped key output by this API. A public wrapped key is generated by wrapped_pair_key->pub_key, and a private wrapped key is generated by wrapped_pair_key->priv_key. As the public key exponent, only 0x00010001 is generated.

Parameters

[in,out]	wrapped_pair_key	User key index for RSA 2048-bit public key and private key pair

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred. Key generation failed.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECC_secp192r1_WrappedKeyPairGenerate()

fsp_err_t R_SCE_ECC_secp192r1_WrappedKeyPairGenerate (sce_ecc_wrapped_pair_key_t * *wrapped_pair_key*)

This is an API for outputting a wrapped key pair for secp192r1 public key and private key pair. These keys are generated from a random number value internally within the SCE. There is therefore no need to input user keys. It is possible to prevent dead copying of data by using the wrapped key output by this API to encrypt the data. The public key index is generated in wrapped_pair_key->pub_key, and the private key index is generated in wrapped_pair_key->priv_key.

Parameters

[in,out]	wrapped_pair_key	Wrapped pair key for secp192r1 public key and private key pair

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECC_secp224r1_WrappedKeyPairGenerate()

fsp_err_t R_SCE_ECC_secp224r1_WrappedKeyPairGenerate (sce_ecc_wrapped_pair_key_t * *wrapped_pair_key*)

This is an API for outputting a wrapped key pair for secp224r1 public key and private key pair. These keys are generated from a random number value internally within the SCE. There is therefore no need to input user keys. It is possible to prevent dead copying of data by using the wrapped key output by this API to encrypt the data. The public key index is generated in wrapped_pair_key->pub_key, and the private key index is generated in wrapped_pair_key->priv_key.

Parameters

[in,out]	wrapped_pair_key	Wrapped pair key for secp224r1 public key and private key pair

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECC_secp256r1_WrappedKeyPairGenerate()

fsp_err_t R_SCE_ECC_secp256r1_WrappedKeyPairGenerate (sce_ecc_wrapped_pair_key_t * *wrapped_pair_key*)

This is an API for outputting a wrapped key pair for secp256r1 public key and private key pair. These keys are generated from a random number value internally within the SCE. There is therefore no need to input user keys. It is possible to prevent dead copying of data by using the wrapped key output by this API to encrypt the data. The public key index is generated in wrapped_pair_key->pub_key, and the private key index is generated in wrapped_pair_key->priv_key.

Parameters

[in,out]	wrapped_pair_key	Wrapped pair key for secp256r1 public key and private key pair

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECC_secp384r1_WrappedKeyPairGenerate()

fsp_err_t R_SCE_ECC_secp384r1_WrappedKeyPairGenerate (sce_ecc_wrapped_pair_key_t * *wrapped_pair_key*)

This is an API for outputting a wrapped key pair for secp384r1 public key and private key pair. These keys are generated from a random number value internally within the SCE. There is therefore no need to input user keys. It is possible to prevent dead copying of data by using the wrapped key output by this API to encrypt the data. The public key index is generated in wrapped_pair_key->pub_key, and the private key index is generated in wrapped_pair_key->priv_key.

Parameters

[in,out]	wrapped_pair_key	Wrapped pair key for secp384r1 public key and private key pair

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RandomNumberGenerate()

fsp_err_t R_SCE_RandomNumberGenerate (uint32_t * *random*)

This API can generate 4 words random number.

Parameters

[in,out]	random	Stores 4words (16 bytes) random data.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128_EncryptedKeyWrap()

fsp_err_t R_SCE_AES128_EncryptedKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_aes_wrapped_key_t * *wrapped_key*)

This API wraps 128-bit AES key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	128-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256_EncryptedKeyWrap()

fsp_err_t R_SCE_AES256_EncryptedKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_aes_wrapped_key_t * *wrapped_key*)

This API wraps 256-bit AES key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	256-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_SHA256HMAC_EncryptedKeyWrap()

fsp_err_t R_SCE_SHA256HMAC_EncryptedKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_hmac_sha_wrapped_key_t * *wrapped_key*)

This API wraps HMAC-SHA256 key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	HMAC-SHA256 wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSA1024_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_RSA1024_EncryptedPublicKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_rsa1024_public_wrapped_key_t * *wrapped_key*)

This API wraps 1024-bit RSA public key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	1024-bit RSA public wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSA1024_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_RSA1024_EncryptedPrivateKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_rsa1024_private_wrapped_key_t * *wrapped_key*)

This API wraps 1024-bit RSA private key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	1024-bit RSA private wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSA2048_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_RSA2048_EncryptedPublicKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_rsa2048_public_wrapped_key_t * *wrapped_key*)

This API wraps 2048-bit RSA public key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	1024-bit RSA public wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSA2048_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_RSA2048_EncryptedPrivateKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_rsa2048_private_wrapped_key_t * *wrapped_key*)

This API wraps 2048-bit RSA private key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	2048-bit RSA private wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSA3072_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_RSA3072_EncryptedPublicKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_rsa3072_public_wrapped_key_t * *wrapped_key*)

This API wraps 3072-bit RSA public key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	3072-bit RSA public wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSA4096_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_RSA4096_EncryptedPublicKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_rsa4096_public_wrapped_key_t * *wrapped_key*)

This API wraps 4096-bit RSA public key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	1024-bit RSA public wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECC_secp192r1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp192r1_EncryptedPublicKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_public_wrapped_key_t * *wrapped_key*)

This API wraps secp192r1 public key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp192r1 public wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECC_secp224r1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp224r1_EncryptedPublicKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_public_wrapped_key_t * *wrapped_key*)

This API wraps secp224r1 public key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp224r1 public wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECC_secp256r1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp256r1_EncryptedPublicKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_public_wrapped_key_t * *wrapped_key*)

This API wraps secp256r1 public key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp256r1 public wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECC_secp384r1_EncryptedPublicKeyWrap()

fsp_err_t R_SCE_ECC_secp384r1_EncryptedPublicKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_public_wrapped_key_t * *wrapped_key*)

This API wraps secp384r1 public key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp384r1 public wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECC_secp192r1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp192r1_EncryptedPrivateKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_private_wrapped_key_t * *wrapped_key*)

This API wraps secp192r1 private key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp192r1 private wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECC_secp224r1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp224r1_EncryptedPrivateKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_private_wrapped_key_t * *wrapped_key*)

This API wraps secp224r1 private key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp224r1 private wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECC_secp256r1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp256r1_EncryptedPrivateKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_private_wrapped_key_t * *wrapped_key*)

This API wraps secp256r1 private key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp256r1 private wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECC_secp384r1_EncryptedPrivateKeyWrap()

fsp_err_t R_SCE_ECC_secp384r1_EncryptedPrivateKeyWrap (uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_key_update_key_t * *key_update_key*, sce_ecc_private_wrapped_key_t * *wrapped_key*)

This API wraps secp384r1 private key within the user routine.

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp384r1 private wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128ECB_EncryptInit()

fsp_err_t R_SCE_AES128ECB_EncryptInit (sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*)

The R_SCE_AES128ECB_EncryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128ECB_EncryptUpdate() function and R_SCE_AES128ECB_EncryptFinal() function.

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	128-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Input illegal wrapped key.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128ECB_EncryptUpdate()

fsp_err_t R_SCE_AES128ECB_EncryptUpdate (sce_aes_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_length*)

The R_SCE_AES128ECB_EncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_SCE_AES128ECB_EncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in,out]	plain_length	plaintext data length (must be a multiple of 16)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128ECB_EncryptFinal()

fsp_err_t R_SCE_AES128ECB_EncryptFinal (sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_length*)

Using the handle specified in the first argument, handle, the R_SCE_AES128ECB_EncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for a portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	cipher	ciphertext data area (nothing ever written here)
[in,out]	cipher_length	ciphertext data length (0 always written here)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128ECB_DecryptInit()

fsp_err_t R_SCE_AES128ECB_DecryptInit (sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*)

The R_SCE_AES128ECB_DecryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128ECB_DecryptUpdate() function and R_SCE_AES128ECB_DecryptFinal() function.

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	128-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Input illegal wrapped key.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128ECB_DecryptUpdate()

fsp_err_t R_SCE_AES128ECB_DecryptUpdate (sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_length*)

The R_SCE_AES128ECB_DecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After plaintext input is completed, call R_SCE_AES128ECB_DecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES handler (work area)
[in]	cipher	ciphertext data area
[in,out]	plain	plaintext data area
[in,out]	cipher_length	ciphertext data length (must be a multiple of 16)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128ECB_DecryptFinal()

fsp_err_t R_SCE_AES128ECB_DecryptFinal (sce_aes_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_length*)

Using the handle specified in the first argument, handle, the R_SCE_AES128ECB_DecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for a portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	plain	plaintext data area (nothing ever written here)
[in,out]	plain_length	plaintext data length (0 always written here)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256ECB_EncryptInit()

fsp_err_t R_SCE_AES256ECB_EncryptInit (sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*)

The R_SCE_AES256ECB_EncryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES256ECB_EncryptUpdate() function and R_SCE_AES256ECB_EncryptFinal() function.

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	128-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Input illegal wrapped key.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256ECB_EncryptUpdate()

fsp_err_t R_SCE_AES256ECB_EncryptUpdate (sce_aes_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_length*)

The R_SCE_AES256ECB_EncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_SCE_AES256ECB_EncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in,out]	plain_length	plaintext data length (must be a multiple of 16)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256ECB_EncryptFinal()

fsp_err_t R_SCE_AES256ECB_EncryptFinal (sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_length*)

Using the handle specified in the first argument, handle, the R_SCE_AES256ECB_EncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for a portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	cipher	ciphertext data area (nothing ever written here)
[in,out]	cipher_length	ciphertext data length (0 always written here)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256ECB_DecryptInit()

fsp_err_t R_SCE_AES256ECB_DecryptInit (sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*)

The R_SCE_AES128ECB_DecryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128ECB_DecryptUpdate() function and R_SCE_AES128ECB_DecryptFinal() function.

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	128-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Input illegal wrapped key.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256ECB_DecryptUpdate()

fsp_err_t R_SCE_AES256ECB_DecryptUpdate (sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_length*)

The R_SCE_AES256ECB_DecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After plaintext input is completed, call R_SCE_AES256ECB_DecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES handler (work area)
[in]	cipher	ciphertext data area
[in,out]	plain	plaintext data area
[in,out]	cipher_length	ciphertext data length (must be a multiple of 16)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256ECB_DecryptFinal()

fsp_err_t R_SCE_AES256ECB_DecryptFinal (sce_aes_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_length*)

Using the handle specified in the first argument, handle, the R_SCE_AES256ECB_DecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for a portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	plain	plaintext data area (nothing ever written here)
[in,out]	plain_length	plaintext data length (0 always written here)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CBC_EncryptInit()

fsp_err_t R_SCE_AES128CBC_EncryptInit (sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector*)

The R_SCE_AES128CBC_EncryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128CBC_EncryptUpdate() function and R_SCE_AES128CBC_EncryptFinal() function.

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	128-bit AES wrapped key
[in]	initial_vector	initialization vector area (16byte)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Input illegal wrapped key.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CBC_EncryptUpdate()

fsp_err_t R_SCE_AES128CBC_EncryptUpdate (sce_aes_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_length*)

The R_SCE_AES128CBC_EncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_SCE_AES128CBC_EncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in,out]	plain_length	plaintext data length (must be a multiple of 16)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CBC_EncryptFinal()

fsp_err_t R_SCE_AES128CBC_EncryptFinal (sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_length*)

Using the handle specified in the first argument, handle, the R_SCE_AES128CBC_EncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for a portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	cipher	ciphertext data area (nothing ever written here)
[in,out]	cipher_length	ciphertext data length (0 always written here)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CBC_DecryptInit()

fsp_err_t R_SCE_AES128CBC_DecryptInit (sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector*)

The R_SCE_AES128CBC_DecryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128CBC_DecryptUpdate() function and R_SCE_AES128CBC_DecryptFinal() function.

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	128-bit AES wrapped key
[in]	initial_vector	initialization vector area (16byte)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Input illegal wrapped key.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CBC_DecryptUpdate()

fsp_err_t R_SCE_AES128CBC_DecryptUpdate (sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_length*)

The R_SCE_AES128CBC_DecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After plaintext input is completed, call R_SCE_AES128CBC_DecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES handler (work area)
[in]	cipher	ciphertext data area
[in,out]	plain	plaintext data area
[in,out]	cipher_length	ciphertext data length (must be a multiple of 16)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CBC_DecryptFinal()

fsp_err_t R_SCE_AES128CBC_DecryptFinal (sce_aes_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_length*)

Using the handle specified in the first argument, handle, the R_SCE_AES128CBC_DecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for a portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	plain	plaintext data area (nothing ever written here)
[in,out]	plain_length	plaintext data length (0 always written here)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CBC_EncryptInit()

fsp_err_t R_SCE_AES256CBC_EncryptInit (sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector*)

The R_SCE_AES256CBC_EncryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES256CBC_EncryptUpdate() function and R_SCE_AES256CBC_EncryptFinal() function.

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	256-bit AES wrapped key
[in]	initial_vector	initial vector area (16byte)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Input illegal wrapped key.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CBC_EncryptUpdate()

fsp_err_t R_SCE_AES256CBC_EncryptUpdate (sce_aes_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_length*)

The R_SCE_AES256CBC_EncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_SCE_AES256CBC_EncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in,out]	plain_length	plaintext data length (must be a multiple of 16)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CBC_EncryptFinal()

fsp_err_t R_SCE_AES256CBC_EncryptFinal (sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_length*)

Using the handle specified in the first argument, handle, the R_SCE_AES256CBC_EncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for a portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	cipher	ciphertext data area (nothing ever written here)
[in,out]	cipher_length	ciphertext data length (0 always written here)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CBC_DecryptInit()

fsp_err_t R_SCE_AES256CBC_DecryptInit (sce_aes_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector*)

The R_SCE_AES256CBC_DecryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES256CBC_DecryptUpdate() function and R_SCE_AES256CBC_DecryptFinal() function.

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	256-bit AES wrapped key
[in]	initial_vector	initialization vector area (16byte)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Input illegal wrapped key.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CBC_DecryptUpdate()

fsp_err_t R_SCE_AES256CBC_DecryptUpdate (sce_aes_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_length*)

The R_SCE_AES256CBC_DecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After plaintext input is completed, call R_SCE_AES256CBC_DecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES handler (work area)
[in]	cipher	ciphertext data area
[in,out]	plain	plaintext data area
[in,out]	cipher_length	ciphertext data length (must be a multiple of 16)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CBC_DecryptFinal()

fsp_err_t R_SCE_AES256CBC_DecryptFinal (sce_aes_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_length*)

Using the handle specified in the first argument, handle, the R_SCE_AES256CBC_DecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for a portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	plain	plaintext data area (nothing ever written here)
[in,out]	plain_length	plaintext data length (0 always written here)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128GCM_EncryptInit()

fsp_err_t R_SCE_AES128GCM_EncryptInit (sce_gcm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector*, uint32_t *initial_vector_length*)

The R_SCE_AES128GCM_EncryptInit() function performs preparations for the execution of an GCM calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128GCM_EncryptUpdate() function and R_SCE_AES128GCM_EncryptFinal() function.

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in]	wrapped_key	128-bit AES wrapped key
[in]	initial_vector	initialization vector area (initial_vector_length byte)
[in]	initial_vector_length	initialization vector length (1 ore more bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128GCM_EncryptUpdate()

```
fsp_err_t R_SCE_AES128GCM_EncryptUpdate ( sce_gcm_handle_t * handle, uint8_t * plain, uint8_t * cipher, uint32_t plain_data_length, uint8_t * aad, uint32_t aad_length )
```

The R_SCE_AES128GCM_EncryptUpdate() function encrypts the plaintext specified in the second argument, plain, in GCM mode using the values specified for wrapped_key and initial_vector in R_SCE_AES128GCM_EncryptInit(), along with the additional authentication data specified in the fifth argument, aad. Inside this function, the data that is input by the user is buffered until the input values of aad and plain exceed 16 bytes. After the input data from plain reaches 16 bytes or more, the encryption result is output to the ciphertext data area specified in the third argument, cipher. The lengths of the plain and aad data to input are respectively specified in the fourth argument, plain_data_length, and the sixth argument, aad_length. For these, specify not the total byte count for the aad and plain input data, but rather the data length to input when the user calls this function. If the input values plain and aad are not divisible by 16 bytes, they will be padded inside the function. First process the data that is input from aad, and then process the data that is input from plain. If aad data is input after starting to input plain data, an error will occur. If aad data and plain data are input to this function at the same time, the aad data will be processed, and then the function will transition to the plain data input state.

Specify areas for plain and cipher that do not overlap. For plain, cipher, initial_vector, and aad, specify RAM addresses that are multiples of 4

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in]	plain_data_length	plaintext data length (0 or more bytes)
[in]	aad	additional authentication data (aad_length byte)
[in]	aad_length	additional authentication data length (0 or more bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	After the data from plain was input, an invalid handle was input from aad.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128GCM_EncryptFinal()

fsp_err_t R_SCE_AES128GCM_EncryptFinal (sce_gcm_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_data_length*, uint8_t * *atag*)

If there is 16-byte fractional data indicated by the total data length of the value of plain that was input by R_SCE_AES128GCM_EncryptUpdate (), the R_SCE_AES128GCM_EncryptFinal() function will output the result of encrypting that fractional data to the ciphertext data area specified in the second argument, cipher. Here, the portion that does not reach 16 bytes will be padded with zeros. The authentication tag is output to the fourth argument, atag. For cipher and atag, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in,out]	cipher	ciphertext data area (cipher_data_length byte)
[in,out]	cipher_data_length	ciphertext data length (0 always written here)
[in,out]	atag	authentication tag area

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128GCM_DecryptInit()

fsp_err_t R_SCE_AES128GCM_DecryptInit (sce_gcm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector*, uint32_t *initial_vector_length*)

The R_SCE_AES128GCM_DecryptInit() function performs preparations for the execution of an GCM calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128GCM_DecryptUpdate() function and R_SCE_AES128GCM_DecryptFinal() function.

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in]	wrapped_key	128-bit AES wrapped key
[in]	initial_vector	initialization vector area (initial_vector_length byte)
[in]	initial_vector_length	initialization vector length (1 ore more bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128GCM_DecryptUpdate()

fsp_err_t R_SCE_AES128GCM_DecryptUpdate (sce_gcm_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_data_length*, uint8_t * *aad*, uint32_t *aad_length*)

The R_SCE_AES128GCM_DecryptUpdate() function decrypts the ciphertext specified in the second argument, cipher, in GCM mode using the values specified for wrapped_key and initial_vector in R_SCE_AES128GCM_DecryptInit(), along with the additional authentication data specified in the fifth argument, aad. Inside this function, the data that is input by the user is buffered until the input values of aad and plain exceed 16 bytes. After the input data from cipher reaches 16 bytes or more, the decryption result is output to the plaintext data area specified in the third argument, plain. The lengths of the cipher and aad data to input are respectively specified in the fourth argument, cipher_data_length, and the sixth argument, aad_length. For these, specify not the total byte count for the aad and cipher input data, but rather the data length to input when the user calls this function. If the input values cipher and aad are not divisible by 16 bytes, they will be padded inside the function. First process the data that is input from aad, and then process the data that is input from cipher. If aad data is input after starting to input cipher data, an error will occur. If aad data and cipher data are input to this function at the same time, the aad data will be processed, and then the function will transition to the cipher data input state. Specify areas for plain and cipher that do not overlap. For plain, cipher, stage, initial_vector, and aad, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in,out]	cipher	ciphertext data area
[in]	plain	plaintext data area
[in]	cipher_data_length	ciphertext data length (0 or more bytes)
[in]	aad	additional authentication data (aad_length byte)
[in]	aad_length	additional authentication data length (0 or more bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	After the data from plain was input, an invalid handle was input from aad.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128GCM_DecryptFinal()

fsp_err_t R_SCE_AES128GCM_DecryptFinal (sce_gcm_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_data_length*, uint8_t * *atag*, uint32_t *atag_length*)

The R_SCE_AES128GCM_DecryptFinal() function decrypts, in GCM mode, the fractional ciphertext specified by R_SCE_AES128GCM_DecryptUpdate() that does not reach 16 bytes, and ends GCM decryption. The encryption data and authentication tag are respectively output to the plaintext data area specified in the second argument, plain, and the authentication tag area specified in the fourth argument, atag. The decoded data length is output to the third argument, plain_data_length. If authentication fails, the return value will be TSIP_ERR_AUTHENTICATION. For the fourth argument, atag, input 16 bytes or less. If it is less than 16 bytes, it will be padded with zeros inside the function. For plain and atag, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in,out]	plain	plaintext data area (cipher_data_length byte)
[in,out]	plain_data_length	plaintext data length (0 always written here)
[in,out]	atag	authentication tag area (atag_length byte)
[in]	atag_length	authentication tag length (4,8,12,13,14,15,16 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_AUTHENTICATION	Authentication failed
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256GCM_EncryptInit()

fsp_err_t R_SCE_AES256GCM_EncryptInit (sce_gcm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector*, uint32_t *initial_vector_length*)

The R_SCE_AES256GCM_EncryptInit() function performs preparations for the execution of an GCM calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES256GCM_EncryptUpdate() function and R_SCE_AES256GCM_EncryptFinal() function.

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in]	wrapped_key	256-bit AES wrapped key
[in]	initial_vector	initialization vector area (initial_vector_length byte)
[in]	initial_vector_length	initialization vector length (1 ore more bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256GCM_EncryptUpdate()

fsp_err_t R_SCE_AES256GCM_EncryptUpdate (sce_gcm_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_data_length*, uint8_t * *aad*, uint32_t *aad_length*)

The R_SCE_AES256GCM_EncryptUpdate() function encrypts the plaintext specified in the second argument, plain, in GCM mode using the values specified for wrapped_key and initial_vector in R_SCE_AES256GCM_EncryptInit(), along with the additional authentication data specified in the fifth argument, aad. Inside this function, the data that is input by the user is buffered until the input values of aad and plain exceed 16 bytes. After the input data from plain reaches 16 bytes or more, the encryption result is output to the ciphertext data area specified in the third argument, cipher. The lengths of the plain and aad data to input are respectively specified in the fourth argument, plain_data_length, and the sixth argument, aad_length. For these, specify not the total byte count for the aad and plain input data, but rather the data length to input when the user calls this function. If the input values plain and aad are not divisible by 16 bytes, they will be padded inside the function. First process the data that is input from aad, and then process the data that is input from plain. If aad data is input after starting to input plain data, an error will occur. If aad data and plain data are input to this function at the same time, the aad data will be processed, and then the function will transition to the plain data input state.

Specify areas for plain and cipher that do not overlap. For plain, cipher, initial_vector, and aad, specify RAM addresses that are multiples of 4

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in]	plain_data_length	plaintext data length (0 or more bytes)
[in]	aad	additional authentication data (aad_length byte)
[in]	aad_length	additional authentication data length (0 or more bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	After the data from plain was input, an invalid handle was input from aad.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256GCM_EncryptFinal()

fsp_err_t R_SCE_AES256GCM_EncryptFinal (sce_gcm_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_data_length*, uint8_t * *atag*)

If there is 16-byte fractional data indicated by the total data length of the value of plain that was input by R_SCE_AES256GCM_EncryptUpdate (), the R_SCE_AES256GCM_EncryptFinal() function will output the result of encrypting that fractional data to the ciphertext data area specified in the second argument, cipher. Here, the portion that does not reach 16 bytes will be padded with zeros. The authentication tag is output to the fourth argument, atag. For cipher and atag, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in,out]	cipher	ciphertext data area (cipher_data_length byte)
[in,out]	cipher_data_length	ciphertext data length (0 always written here)
[in,out]	atag	authentication tag area

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256GCM_DecryptInit()

fsp_err_t R_SCE_AES256GCM_DecryptInit (sce_gcm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *initial_vector*, uint32_t *initial_vector_length*)

The R_SCE_AES256GCM_DecryptInit() function performs preparations for the execution of an GCM calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES256GCM_DecryptUpdate() function and R_SCE_AES256GCM_DecryptFinal() function.

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in]	wrapped_key	256-bit AES wrapped key
[in]	initial_vector	initialization vector area (initial_vector_length byte)
[in]	initial_vector_length	initialization vector length (1 ore more bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256GCM_DecryptUpdate()

fsp_err_t R_SCE_AES256GCM_DecryptUpdate (sce_gcm_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_data_length*, uint8_t * *aad*, uint32_t *aad_length*)

The R_SCE_AES256GCM_DecryptUpdate() function decrypts the ciphertext specified in the second argument, cipher, in GCM mode using the values specified for wrapped_key and initial_vector in R_SCE_AES256GCM_DecryptInit(), along with the additional authentication data specified in the fifth argument, aad. Inside this function, the data that is input by the user is buffered until the input values of aad and plain exceed 16 bytes. After the input data from cipher reaches 16 bytes or more, the decryption result is output to the plaintext data area specified in the third argument, plain. The lengths of the cipher and aad data to input are respectively specified in the fourth argument, cipher_data_length, and the sixth argument, aad_length. For these, specify not the total byte count for the aad and cipher input data, but rather the data length to input when the user calls this function. If the input values cipher and aad are not divisible by 16 bytes, they will be padded inside the function. First process the data that is input from aad, and then process the data that is input from cipher. If aad data is input after starting to input cipher data, an error will occur. If aad data and cipher data are input to this function at the same time, the aad data will be processed, and then the function will transition to the cipher data input state. Specify areas for plain and cipher that do not overlap. For plain, cipher, stage, initial_vector, and aad, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in,out]	cipher	ciphertext data area
[in]	plain	plaintext data area
[in]	cipher_data_length	ciphertext data length (0 or more bytes)
[in]	aad	additional authentication data (aad_length byte)
[in]	aad_length	additional authentication data length (0 or more bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	After the data from plain was input, an invalid handle was input from aad.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256GCM_DecryptFinal()

fsp_err_t R_SCE_AES256GCM_DecryptFinal (sce_gcm_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_data_length*, uint8_t * *atag*, uint32_t *atag_length*)

The R_SCE_AES256GCM_DecryptFinal() function decrypts, in GCM mode, the fractional ciphertext specified by R_SCE_AES256GCM_DecryptUpdate() that does not reach 16 bytes, and ends GCM decryption. The encryption data and authentication tag are respectively output to the plaintext data area specified in the second argument, plain, and the authentication tag area specified in the fourth argument, atag. The decoded data length is output to the third argument, plain_data_length. If authentication fails, the return value will be TSIP_ERR_AUTHENTICATION. For the fourth argument, atag, input 16 bytes or less. If it is less than 16 bytes, it will be padded with zeros inside the function. For plain and atag, specify RAM addresses that are multiples of 4.

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in,out]	plain	plaintext data area (cipher_data_length byte)
[in,out]	plain_data_length	plaintext data length (0 always written here)
[in,out]	atag	authentication tag area (atag_length byte)
[in]	atag_length	authentication tag length (4,8,12,13,14,15,16 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_AUTHENTICATION	Authentication failed
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CCM_EncryptInit()

fsp_err_t R_SCE_AES128CCM_EncryptInit (sce_ccm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *nonce*, uint32_t *nonce_length*, uint8_t * *adata*, uint8_t *a_length*, uint32_t *payload_length*, uint32_t *mac_length*)

The R_SCE_AES128CCM_EncryptInit() function prepares to perform CCM computation and writes the result to the first argument, handle. The succeeding functions R_SCE_AES128CCM_EncryptUpdate() and R_SCE_AES128CCM_EncryptFinal() use handle as an argument.

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	wrapped_key	128-bit AES wrapped key
[in]	nonce	Nonce
[in]	nonce_length	Nonce data length (7 to 13 bytes)
[in]	adata	additional authentication data
[in]	a_length	additional authentication data length (0 to 110 bytes)
[in]	payload_length	Payload length (any number of bytes)
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CCM_EncryptUpdate()

fsp_err_t R_SCE_AES128CCM_EncryptUpdate (sce_ccm_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_length*)

The R_SCE_AES128CCM_EncryptUpdate() function encrypts the plaintext specified in the second argument, plain, in CCM mode using the values specified by wrapped_key, nonce, and adata in R_SCE_AES128CCM_EncryptInit(). This function buffers internally the data input by the user until the input value of plain exceeds 16 bytes. Once the amount of plain input data is 16 bytes or greater, the encrypted result is output to cipher, which is specified in the third argument. Use payload_length in R_SCE_AES128CCM_EncryptInit() to specify the total data length of plain that will be input. Use plain_length in this function to specify the data length to be input when the user calls this function. If the input value of plain is less than 16 bytes, the function performs padding internally.

Ensure that the areas allocated to plain and cipher do not overlap. Also, specify RAM addresses that are multiples of 4 for plain and cipher.

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in]	plain_length	plaintext data length

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CCM_EncryptFinal()

fsp_err_t R_SCE_AES128CCM_EncryptFinal (sce_ccm_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_length*, uint8_t * *mac*, uint32_t *mac_length*)

If the data length of plain input in R_SCE_AES128CCM_EncryptUpdate() results in leftover data after 16 bytes, the R_SCE_AES128CCM_EncryptFinal() function outputs the leftover encrypted data to cipher, which is specified in the second argument. The MAC value is output to the fourth argument, mac. Set the fifth argument, mac_length to the same value as that specified for the argument mac_length in Aes128CcmEncryptInit(). Also, specify RAM addresses that are multiples of 4 for cipher and mac.

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in,out]	cipher	ciphertext data area
[in,out]	cipher_length	ciphertext data length
[in,out]	mac	MAC area
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CCM_DecryptInit()

fsp_err_t R_SCE_AES128CCM_DecryptInit (sce_ccm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *nonce*, uint32_t *nonce_length*, uint8_t * *adata*, uint8_t *a_length*, uint32_t *payload_length*, uint32_t *mac_length*)

The R_SCE_AES128CCM_DecryptInit() function prepares to perform CCM computation and writes the result to the first argument, handle. The succeeding functions R_SCE_AES128CCM_DecryptUpdate() and R_SCE_AES128CCM_DecryptFinal() use handle as an argument.

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	wrapped_key	128-bit AES wrapped key
[in]	nonce	Nonce
[in]	nonce_length	Nonce data length (7 to 13 bytes)
[in]	adata	additional authentication data
[in]	a_length	additional authentication data length (0 to 110 bytes)
[in]	payload_length	Payload length (any number of bytes)
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CCM_DecryptUpdate()

fsp_err_t R_SCE_AES128CCM_DecryptUpdate (sce_ccm_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_length*)

The R_SCE_AES128CCM_DecryptUpdate() function decrypts the ciphertext specified by the second argument, cipher, in CCM mode using the values specified by wrapped_key, nonce, and adata in in R_SCE_AES128CCM_DecryptInit(). This function buffers internally the data input by the user until the input value of cipher exceeds 16 bytes. Once the amount of cipher input data is 16 bytes or greater, the decrypted result is output to plain, which is specified in the third argument. Use payload_length in R_SCE_AES128CCM_DecryptInit() to specify the total data length of cipher that will be input. Use cipher_length in this function to specify the data length to be input when the user calls this function. If the input value of cipher is less than 16 bytes, the function performs padding internally.

Ensure that the areas allocated to cipher and plain do not overlap. Also, specify RAM addresses that are multiples of 4 for cipher and plain.

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	cipher	ciphertext data area
[in,out]	plain	plaintext data area
[in]	cipher_length	ciphertext data length

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CCM_DecryptFinal()

fsp_err_t R_SCE_AES128CCM_DecryptFinal (sce_ccm_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_length*, uint8_t * *mac*, uint32_t *mac_length*)

If the data length of cipher input in R_SCE_AES128GCM_DecryptUpdate() results in leftover data after 16 bytes, the R_SCE_AES128GCM_DecryptFinal() function outputs the leftover decrypted data to cipher, which is specified in the second argument. In addition, the function verifies the fourth argument, mac. Set the fifth argument, mac_length, to the same value as that specified for the argument mac_length in Aes128CcmDecryptInit().

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in,out]	plain	plaintext data area
[in,out]	plain_length	plaintext data length
[in]	mac	MAC area
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error, or authentication failed.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CCM_EncryptInit()

fsp_err_t R_SCE_AES256CCM_EncryptInit (sce_ccm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *nonce*, uint32_t *nonce_length*, uint8_t * *adata*, uint8_t *a_length*, uint32_t *payload_length*, uint32_t *mac_length*)

The R_SCE_AES256CCM_EncryptInit() function prepares to perform CCM computation and writes the result to the first argument, handle. The succeeding functions R_SCE_AES256CCM_EncryptUpdate() and R_SCE_AES256CCM_EncryptFinal() use handle as an argument.

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	wrapped_key	256-bit AES wrapped key
[in]	nonce	Nonce
[in]	nonce_length	Nonce data length (7 to 13 bytes)
[in]	adata	additional authentication data
[in]	a_length	additional authentication data length (0 to 110 bytes)
[in]	payload_length	Payload length (any number of bytes)
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CCM_EncryptUpdate()

fsp_err_t R_SCE_AES256CCM_EncryptUpdate (sce_ccm_handle_t * *handle*, uint8_t * *plain*, uint8_t * *cipher*, uint32_t *plain_length*)

The R_SCE_AES256CCM_EncryptUpdate() function encrypts the plaintext specified in the second argument, plain, in CCM mode using the values specified by wrapped_key, nonce, and adata in R_SCE_AES256CCM_EncryptInit(). This function buffers internally the data input by the user until the input value of plain exceeds 16 bytes. Once the amount of plain input data is 16 bytes or greater, the encrypted result is output to cipher, which is specified in the third argument. Use payload_length in R_SCE_AES256CCM_EncryptInit() to specify the total data length of plain that will be input. Use plain_length in this function to specify the data length to be input when the user calls this function. If the input value of plain is less than 16 bytes, the function performs padding internally.

Ensure that the areas allocated to plain and cipher do not overlap. Also, specify RAM addresses that are multiples of 4 for plain and cipher.

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in]	plain_length	plaintext data length

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CCM_EncryptFinal()

fsp_err_t R_SCE_AES256CCM_EncryptFinal (sce_ccm_handle_t * *handle*, uint8_t * *cipher*, uint32_t * *cipher_length*, uint8_t * *mac*, uint32_t *mac_length*)

If the data length of plain input in R_SCE_AES256CCM_EncryptUpdate() results in leftover data after 16 bytes, the R_SCE_AES256CCM_EncryptFinal() function outputs the leftover encrypted data to cipher, which is specified in the second argument. The MAC value is output to the fourth argument, mac. Set the fifth argument, mac_length to the same value as that specified for the argument mac_length in Aes256CcmEncryptInit(). Also, specify RAM addresses that are multiples of 4 for cipher and mac.

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in,out]	cipher	ciphertext data area
[in,out]	cipher_length	ciphertext data length
[in,out]	mac	MAC area
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CCM_DecryptInit()

fsp_err_t R_SCE_AES256CCM_DecryptInit (sce_ccm_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*, uint8_t * *nonce*, uint32_t *nonce_length*, uint8_t * *adata*, uint8_t *a_length*, uint32_t *payload_length*, uint32_t *mac_length*)

The R_SCE_AES256CCM_DecryptInit() function prepares to perform CCM computation and writes the result to the first argument, handle. The succeeding functions R_SCE_AES256CCM_DecryptUpdate() and R_SCE_AES256CCM_DecryptFinal() use handle as an argument.

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	wrapped_key	256-bit AES wrapped key
[in]	nonce	Nonce
[in]	nonce_length	Nonce data length (7 to 13 bytes)
[in]	adata	additional authentication data
[in]	a_length	additional authentication data length (0 to 110 bytes)
[in]	payload_length	Payload length (any number of bytes)
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CCM_DecryptUpdate()

fsp_err_t R_SCE_AES256CCM_DecryptUpdate (sce_ccm_handle_t * *handle*, uint8_t * *cipher*, uint8_t * *plain*, uint32_t *cipher_length*)

The R_SCE_AES256CCM_DecryptUpdate() function decrypts the ciphertext specified by the second argument, cipher, in CCM mode using the values specified by wrapped_key, nonce, and adata in in R_SCE_AES256CCM_DecryptInit(). This function buffers internally the data input by the user until the input value of cipher exceeds 16 bytes. Once the amount of cipher input data is 16 bytes or greater, the decrypted result is output to plain, which is specified in the third argument. Use payload_length in R_SCE_AES256CCM_DecryptInit() to specify the total data length of cipher that will be input. Use cipher_length in this function to specify the data length to be input when the user calls this function. If the input value of cipher is less than 16 bytes, the function performs padding internally.

Ensure that the areas allocated to cipher and plain do not overlap. Also, specify RAM addresses that are multiples of 4 for cipher and plain.

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	cipher	ciphertext data area
[in,out]	plain	plaintext data area
[in]	cipher_length	ciphertext data length

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CCM_DecryptFinal()

fsp_err_t R_SCE_AES256CCM_DecryptFinal (sce_ccm_handle_t * *handle*, uint8_t * *plain*, uint32_t * *plain_length*, uint8_t * *mac*, uint32_t *mac_length*)

If the data length of cipher input in R_SCE_AES256GCM_DecryptUpdate() results in leftover data after 16 bytes, the R_SCE_AES256GCM_DecryptFinal() function outputs the leftover decrypted data to cipher, which is specified in the second argument. In addition, the function verifies the fourth argument, mac. Set the fifth argument, mac_length, to the same value as that specified for the argument mac_length in Aes256CcmDecryptInit().

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in,out]	plain	plaintext data area
[in,out]	plain_length	plaintext data length
[in]	mac	MAC area
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error, or authentication failed.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CMAC_GenerateInit()

fsp_err_t R_SCE_AES128CMAC_GenerateInit (sce_cmac_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*)

The R_SCE_AES128CMAC_GenerateInit() function performs preparations for the execution of an CMAC calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES128CMAC_GenerateUpdate() function and R_SCE_AES128CMAC_GenerateFinal() function.

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	wrapped_key	128-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CMAC_GenerateUpdate()

fsp_err_t R_SCE_AES128CMAC_GenerateUpdate (sce_cmac_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length*)

The R_SCE_AES128CMAC_GenerateUpdate() function performs MAC value generation based on the message specified in the second argument, message, using the value specified for wrapped_key in R_SCE_AES128CMAC_GenerateInit(). Inside this function, the data that is input by the user is buffered until the input value of message exceeds 16 bytes. The length of the message data to input is specified in the third argument, message_len. For these, input not the total byte count for message input data, but rather the message data length to input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it will be padded within the function. For message, specify a RAM address that are multiples of 4.

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	message	message data area (message_length byte)
[in]	message_length	message data length (0 or more bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CMAC_GenerateFinal()

fsp_err_t R_SCE_AES128CMAC_GenerateFinal (sce_cmac_handle_t * *handle*, uint8_t * *mac*)

The R_SCE_AES128CMAC_GenerateFinal() function outputs the MAC value to the MAC data area specified in the second argument, mac, and ends CMAC mode.

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in,out]	mac	MAC data area (16byte)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.
FSP_ERR_CRYPTO_SCE_AUTHENTICATION	Not used.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CMAC_VerifyInit()

fsp_err_t R_SCE_AES128CMAC_VerifyInit (sce_cmac_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*)

The R_SCE_AES128CMAC_VerifyInit() function performs preparations for the execution of a CMAC calculation, and writes the result to the first argument, handle. The value of handle is used as an argumentin the subsequent R_SCE_AES128CMAC_VerifyUpdate() function and R_SCE_AES128CMAC_VerifyFinal() function.

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	wrapped_key	128-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CMAC_VerifyUpdate()

fsp_err_t R_SCE_AES128CMAC_VerifyUpdate (sce_cmac_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length*)

The R_SCE_AES128CMAC_VerifyUpdate() function performs MAC value generation based on the message specified in the second argument, message, using the value specified for wrapped_key in R_SCE_AES128CMAC_VerifyInit(). Inside this function, the data that is input by the user is buffered until the input value of message exceeds 16 bytes. The length of the message data to input is specified in the third argument, message_len. For these, input not the total byte count for message input data, but rather the message data length to input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it will be padded within the function. For message, specify a RAM address that are multiples of 4.

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	message	message data area (message_length byte)
[in]	message_length	message data length (0 or more bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES128CMAC_VerifyFinal()

fsp_err_t R_SCE_AES128CMAC_VerifyFinal (sce_cmac_handle_t * *handle*, uint8_t * *mac*, uint32_t *mac_length*)

The R_SCE_AES128CMAC_VerifyFinal() function inputs the MAC value in the MAC data area specified in the second argument, mac, and verifies the MAC value. If authentication fails, the return value will be TSIP_ERR_AUTHENTICATION. If the MAC value is less than 16 bytes, it will be padded with zeros inside the function.

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in,out]	mac	MAC data area (mac_length byte)
[in,out]	mac_length	MAC data length (2 to 16 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_AUTHENTICATION	Authentication failed
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CMAC_GenerateInit()

fsp_err_t R_SCE_AES256CMAC_GenerateInit (sce_cmac_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*)

The R_SCE_AES256CMAC_GenerateInit() function performs preparations for the execution of an CMAC calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_AES256CMAC_GenerateUpdate() function and R_SCE_AES256CMAC_GenerateFinal() function.

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	wrapped_key	256-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CMAC_GenerateUpdate()

fsp_err_t R_SCE_AES256CMAC_GenerateUpdate (sce_cmac_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length*)

The R_SCE_AES256CMAC_GenerateUpdate() function performs MAC value generation based on the message specified in the second argument, message, using the value specified for wrapped_key in R_SCE_AES256CMAC_GenerateInit(). Inside this function, the data that is input by the user is buffered until the input value of message exceeds 16 bytes. The length of the message data to input is specified in the third argument, message_len. For these, input not the total byte count for message input data, but rather the message data length to input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it will be padded within the function. For message, specify a RAM address that are multiples of 4.

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	message	message data area (message_length byte)
[in]	message_length	message data length (0 or more bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CMAC_GenerateFinal()

fsp_err_t R_SCE_AES256CMAC_GenerateFinal (sce_cmac_handle_t * *handle*, uint8_t * *mac*)

The R_SCE_AES256CMAC_GenerateFinal() function outputs the MAC value to the MAC data area specified in the second argument, mac, and ends CMAC mode.

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in,out]	mac	MAC data area (16byte)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.
FSP_ERR_CRYPTO_SCE_AUTHENTICATION	Not used.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CMAC_VerifyInit()

fsp_err_t R_SCE_AES256CMAC_VerifyInit (sce_cmac_handle_t * *handle*, sce_aes_wrapped_key_t * *wrapped_key*)

The R_SCE_AES256CMAC_VerifyInit() function performs preparations for the execution of a CMAC calculation, and writes the result to the first argument, handle. The value of handle is used as an argumentin the subsequent R_SCE_AES256CMAC_VerifyUpdate() function and R_SCE_AES256CMAC_VerifyFinal() function.

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	wrapped_key	256-bit AES wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CMAC_VerifyUpdate()

fsp_err_t R_SCE_AES256CMAC_VerifyUpdate (sce_cmac_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length*)

The R_SCE_AES256CMAC_VerifyUpdate() function performs MAC value generation based on the message specified in the second argument, message, using the value specified for wrapped_key in R_SCE_AES256CMAC_VerifyInit(). Inside this function, the data that is input by the user is buffered until the input value of message exceeds 16 bytes. The length of the message data to input is specified in the third argument, message_len. For these, input not the total byte count for message input data, but rather the message data length to input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it will be padded within the function. For message, specify a RAM address that are multiples of 4.

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	message	message data area (message_length byte)
[in]	message_length	message data length (0 or more bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_AES256CMAC_VerifyFinal()

fsp_err_t R_SCE_AES256CMAC_VerifyFinal (sce_cmac_handle_t * *handle*, uint8_t * *mac*, uint32_t *mac_length*)

The R_SCE_AES256CMAC_VerifyFinal() function inputs the MAC value in the MAC data area specified in the second argument, mac, and verifies the MAC value. If authentication fails, the return value will be TSIP_ERR_AUTHENTICATION. If the MAC value is less than 16 bytes, it will be padded with zeros inside the function.

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in,out]	mac	MAC data area (mac_length byte)
[in,out]	mac_length	MAC data length (2 to 16 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_AUTHENTICATION	Authentication failed
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_SHA256_Init()

fsp_err_t R_SCE_SHA256_Init (sce_sha_md5_handle_t * *handle*)

The R_SCE_SHA256_Init() function performs preparations for the execution of an SHA-256 hash calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_SCE_SHA256_Update() function and R_SCE_SHA256_Final() function.

Parameters

[in,out]	handle	SHA handler (work area)

Return values

FSP_SUCCESS	Normal termination

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_SHA256_Update()

fsp_err_t R_SCE_SHA256_Update (sce_sha_md5_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length*)

The R_SCE_SHA256_Update() function calculates a hash value based on the second argument, message, and the third argument, message_length, and writes the ongoing status to the first argument, handle. After message input is completed, call R_SCE_SHA256_Final().

Parameters

[in,out]	handle	SHA handler (work area)
[in]	message	message data area
[in]	message_length	message data length

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_SHA256_Final()

fsp_err_t R_SCE_SHA256_Final (sce_sha_md5_handle_t * *handle*, uint8_t * *digest*, uint32_t * *digest_length*)

Using the handle specified in the first argument, handle, the R_SCE_SHA256_Final() function writes the calculation result to the second argument, digest, and writes the length of the calculation result to the third argument, digest_length.

Parameters

[in,out]	handle	SHA handler (work area)
[in,out]	digest	hasha data area
[in,out]	digest_length	hash data length (32bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_SHA256HMAC_GenerateInit()

fsp_err_t R_SCE_SHA256HMAC_GenerateInit (sce_hmac_sha_handle_t * *handle*, sce_hmac_sha_wrapped_key_t * *wrapped_key*)

The R_SCE_SHA256HMAC_GenerateInit() function uses the second argument wrapped_key to prepare for execution of SHA256-HMAC calculation, then writes the result to the first argument handle. The argument handle is used by the subsequent R_SCE_SHA256HMAC_GenerateUpdate() function or R_SCE_SHA256HMAC_GenerateFinal() function.

Parameters

[in,out]	handle	SHA-HMAC handler (work area)
[in]	wrapped_key	MAC wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	An invalid MAC wrapped key was input.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_SHA256HMAC_GenerateUpdate()

fsp_err_t R_SCE_SHA256HMAC_GenerateUpdate (sce_hmac_sha_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length*)

The R_SCE_SHA256HMAC_GenerateUpdate() function uses the handle specified by the first argument handle, calculates a hash value from the second argument message and third argument message_length, then writes the intermediate result to the first argument handle. After message input finishes, call the R_SCE_SHA256HMAC_GenerateFinal() function.

Parameters

[in,out]	handle	SHA-HMAC handle (work area)
[in]	message	Message area
[in]	message_length	Message length

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_SHA256HMAC_GenerateFinal()

fsp_err_t R_SCE_SHA256HMAC_GenerateFinal (sce_hmac_sha_handle_t * *handle*, uint8_t * *mac*)

The R_SCE_SHA256HMAC_GenerateFinal() function uses the handle specified by the first argument handle and writes the calculation result to the second argument mac.

Parameters

[in,out]	handle	SHA-HMAC handle (work area)
[in,out]	mac	HMAC area (32 bytes)

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_SHA256HMAC_VerifyInit()

fsp_err_t R_SCE_SHA256HMAC_VerifyInit (sce_hmac_sha_handle_t * *handle*, sce_hmac_sha_wrapped_key_t * *wrapped_key*)

The R_SCE_SHA256HMAC_VerifyInit() function uses the second argument wrapped_key to prepare for execution of SHA256-HMAC calculation, then writes the result to the first argument handle. The argument handle is used by the subsequent R_SCE_SHA256HMAC_VerifyUpdate() function or R_SCE_SHA256HMAC_VerifyFinal() function.

Parameters

[in,out]	handle	SHA-HMAC handler (work area)
[in]	wrapped_key	MAC wrapped key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	An invalid MAC wrapped key was input.
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_SHA256HMAC_VerifyUpdate()

fsp_err_t R_SCE_SHA256HMAC_VerifyUpdate (sce_hmac_sha_handle_t * *handle*, uint8_t * *message*, uint32_t *message_length*)

The R_SCE_SHA256HMAC_VerifyUpdate() function uses the handle specified by the first argument handle, calculates a hash value from the second argument message and third argument message_length, then writes the intermediate result to the first argument handle. After message input finishes, call the R_SCE_SHA256HMAC_VerifyFinal() function.

Parameters

[in,out]	handle	SHA-HMAC handle (work area)
[in]	message	Message area
[in]	message_length	Message length

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_SHA256HMAC_VerifyFinal()

fsp_err_t R_SCE_SHA256HMAC_VerifyFinal (sce_hmac_sha_handle_t * *handle*, uint8_t * *mac*, uint32_t *mac_length*)

The R_SCE_SHA256HMAC_VerifyFinal() function uses the handle specified by the first argument handle and verifies the mac value from the second argument mac and third argument mac_length. Input a value in bytes from 4 to 32 as mac_length.

Parameters

[in,out]	handle	SHA-HMAC handle (work area)
[in]	mac	HMAC area
[in]	mac_length	HMAC length

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSASSA_PKCS1024_SignatureGenerate()

fsp_err_t R_SCE_RSASSA_PKCS1024_SignatureGenerate (sce_rsa_byte_data_t * *message_hash*, sce_rsa_byte_data_t * *signature*, sce_rsa1024_private_wrapped_key_t * *wrapped_key*, uint8_t *hash_type*)

The R_SCE_RSASSA_PKCS1024_SignatureGenerate() function generates, in accordance with RSASSA-PKCS1-V1_5, a signature from the message text or hash value that is input in the first argument, message_hash, using the private wrapped key input to the third argument, wrapped_key, and writes the signature text to the second argument, signature. When a message is specified in the first argument, message_hash->data_type, a hash value is calculated for the message as specified by the fourth argument, hash_type. When specifying a hash value in the first argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

Parameters

[in]	message_hash	Message or hash value to which to attach signature • message_hash->pdata : Specifies pointer to array storing the message or hash value

		• message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1)
[in,out]	signature	Signature text storage destination information • signature->pdata : Specifies pointer to array storing the signature text • signature->data_length : data length
[in]	wrapped_key	Inputs the 1024-bit RSA private wrapped key.
[in]	hash_type	Only HW_SCE_RSA_HASH_SHA256 is supported

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSASSA_PKCS1024_SignatureVerify()

fsp_err_t R_SCE_RSASSA_PKCS1024_SignatureVerify (sce_rsa_byte_data_t * *signature*, sce_rsa_byte_data_t * *message_hash*, sce_rsa1024_public_wrapped_key_t * *wrapped_key*, uint8_t *hash_type*)

The R_SCE_RSASSA_PKCS1024_SignatureVerify() function verifies, in accordance with RSASSA-PKCS1-V1_5, the signature text input to the first argument signature, and the message text or hash value input to the second argument, message_hash, using the public wrapped key input to the third argument, wrapped_key. When a message is specified in the second argument, message_hash->data_type, a hash value is calculated using the public wrapped key input to the third argument, wrapped_key, and as specified by the fourth argument, hash_type. When specifying a hash value in the second argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

Parameters

[in]	signature	Signature text information to verify • signature->pdata : Specifies pointer to array storing the signature text • signature->data_length : Specifies effective data length of the array
[in]	message_hash	Message text or hash value to verify • message_hash->pdata : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1)
[in]	wrapped_key	Inputs the 1024-bit RSA public wrapped key.
[in]	hash_type	Only

		HW_SCE_RSA_HASH_SHA256 is supported

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_AUTHENTICATION	Authentication failed
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSASSA_PKCS2048_SignatureGenerate()

fsp_err_t R_SCE_RSASSA_PKCS2048_SignatureGenerate (sce_rsa_byte_data_t * *message_hash*, sce_rsa_byte_data_t * *signature*, sce_rsa2048_private_wrapped_key_t * *wrapped_key*, uint8_t *hash_type*)

The R_SCE_RSASSA_PKCS2048_SignatureGenerate() function generates, in accordance with RSASSA-PKCS1-V1_5, a signature from the message text or hash value that is input in the first argument, message_hash, using the private wrapped key input to the third argument, wrapped_key, and writes the signature text to the second argument, signature. When a message is specified in the first argument, message_hash->data_type, a hash value is calculated for the message as specified by the fourth argument, hash_type. When specifying a hash value in the first argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

Parameters

[in]	message_hash	Message or hash value to which to attach signature • message_hash->pdata : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the

		data type of message_hash (Message: 0 Hash value: 1)
[in,out]	signature	Signature text storage destination information • signature->pdata : Specifies pointer to array storing the signature text • signature->data_length : data length
[in]	wrapped_key	Inputs the 2048-bit RSA private wrapped key.
[in]	hash_type	Only HW_SCE_RSA_HASH_SHA256 is supported

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSASSA_PKCS2048_SignatureVerify()

fsp_err_t R_SCE_RSASSA_PKCS2048_SignatureVerify (sce_rsa_byte_data_t * *signature*, sce_rsa_byte_data_t * *message_hash*, sce_rsa2048_public_wrapped_key_t * *wrapped_key*, uint8_t *hash_type*)

The R_SCE_RSASSA_PKCS2048_SignatureVerify() function verifies, in accordance with RSASSA-PKCS1-V1_5, the signature text input to the first argument signature, and the message text or hash value input to the second argument, message_hash, using the public wrapped key input to the third argument, wrapped_key. When a message is specified in the second argument, message_hash->data_type, a hash value is calculated using the public wrapped key input to the third argument, wrapped_key, and as specified by the fourth argument, hash_type. When specifying a hash value in the second argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

Parameters

[in]	signature	Signature text information to verify • signature->pdata : Specifies pointer to array storing the signature text • signature->data_length : Specifies effective data length of the array
[in]	message_hash	Message text or hash value to verify • message_hash->pdata : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1)
[in]	wrapped_key	Inputs the 1024-bit RSA public wrapped key.
[in]	hash_type	Only

		HW_SCE_RSA_HASH_SHA256 is supported

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_AUTHENTICATION	Authentication failed
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSASSA_PKCS3072_SignatureVerify()

fsp_err_t R_SCE_RSASSA_PKCS3072_SignatureVerify (sce_rsa_byte_data_t * *signature*, sce_rsa_byte_data_t * *message_hash*, sce_rsa3072_public_wrapped_key_t * *wrapped_key*, uint8_t *hash_type*)

The R_SCE_RSASSA_PKCS3072_SignatureVerify() function verifies, in accordance with RSASSA-PKCS1-V1_5, the signature text input to the first argument signature, and the message text or hash value input to the second argument, message_hash, using the public wrapped key input to the third argument, wrapped_key. When a message is specified in the second argument, message_hash->data_type, a hash value is calculated using the public wrapped key input to the third argument, wrapped_key, and as specified by the fourth argument, hash_type. When specifying a hash value in the second argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

Parameters

[in]	signature	Signature text information to verify • signature->pdata : Specifies pointer to array storing the signature text • signature->data_length : Specifies effective data length of the array
[in]	message_hash	Message text or hash value to verify • message_hash->pda

		ta : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1)
[in]	wrapped_key	Inputs the 3072-bit RSA public wrapped key.
[in]	hash_type	Only HW_SCE_RSA_HASH_SHA256 is supported

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_AUTHENTICATION	Authentication failed
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSASSA_PKCS4096_SignatureVerify()

fsp_err_t R_SCE_RSASSA_PKCS4096_SignatureVerify (sce_rsa_byte_data_t * *signature*, sce_rsa_byte_data_t * *message_hash*, sce_rsa4096_public_wrapped_key_t * *wrapped_key*, uint8_t *hash_type*)

The R_SCE_RSASSA_PKCS4096_SignatureVerify() function verifies, in accordance with RSASSA-PKCS1-V1_5, the signature text input to the first argument signature, and the message text or hash value input to the second argument, message_hash, using the public wrapped key input to the third argument, wrapped_key. When a message is specified in the second argument, message_hash->data_type, a hash value is calculated using the public wrapped key input to the third argument, wrapped_key, and as specified by the fourth argument, hash_type. When specifying a hash value in the second argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

Parameters

[in]	signature	Signature text information to verify • signature->pdata : Specifies pointer to array storing the signature text • signature->data_length : Specifies effective data length of the array
[in]	message_hash	Message text or hash value to verify • message_hash->pdata : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1)
[in]	wrapped_key	Inputs the 1024-bit RSA public wrapped key.
[in]	hash_type	Only

		HW_SCE_RSA_HASH_SHA256 is supported

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_AUTHENTICATION	Authentication failed
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSAES_PKCS1024_Encrypt()

fsp_err_t R_SCE_RSAES_PKCS1024_Encrypt (sce_rsa_byte_data_t * *plain*, sce_rsa_byte_data_t * *cipher*, sce_rsa1024_public_wrapped_key_t * *wrapped_key*)

The R_SCE_RSAES_PKCS1024_Encrypt() function RSA-encrypts the plaintext input to the first argument, plain, according to RSAES-PKCS1-V1_5. It writes the encryption result to the second argument, cipher.

Parameters

[in]	plain	plaintext • plain->pdata : Specifies pointer to array containing plaintext. • plain->data_length : Specifies valid data length of plaintext array. data size <= public key n size - 11
[in,out]	cipher	ciphertext • cipher->pdata : Specifies pointer to array containing ciphertext. • cipher->data_length : Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size).
[in]	wrapped_key	Inputs the 1024-bit RSA public wrapped key.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Incorrect wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSAES_PKCS1024_Decrypt()

fsp_err_t R_SCE_RSAES_PKCS1024_Decrypt (sce_rsa_byte_data_t * *cipher*, sce_rsa_byte_data_t * *plain*, sce_rsa1024_private_wrapped_key_t * *wrapped_key*)

The R_SCE_RSAES_PKCS1024_Decrypt() function RSA-decrypts the ciphertext input to the first argument, cipher, according to RSAES-PKCS1-V1_5. It writes the decryption result to the second argument, plain.

Parameters

[in]	cipher	ciphertext • cipher->pdata : Specifies pointer to array containing ciphertext. • cipher->data_length : Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size).
[in,out]	plain	plaintext • plain->pdata : Specifies pointer to array containing plaintext. • plain->data_length : Inputs plaintext buffer size. The following size is required. Plaintext buffer size >= public key n size -11. Outputs valid data length after decryption (public key n size).
[in]	wrapped_key	Inputs the 1024-bit RSA private wrapped key.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Incorrect wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.

FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSAES_PKCS2048_Encrypt()

fsp_err_t R_SCE_RSAES_PKCS2048_Encrypt (sce_rsa_byte_data_t * *plain*, sce_rsa_byte_data_t * *cipher*, sce_rsa2048_public_wrapped_key_t * *wrapped_key*)

The R_SCE_RSAES_PKCS2048_Encrypt() function RSA-encrypts the plaintext input to the first argument, plain, according to RSAES-PKCS1-V1_5. It writes the encryption result to the second argument, cipher.

Parameters

[in]	plain	plaintext • plain->pdata : Specifies pointer to array containing plaintext. • plain->data_length : Specifies valid data length of plaintext array. data size <= public key n size - 11
[in,out]	cipher	ciphertext • cipher->pdata : Specifies pointer to array containing ciphertext. • cipher->data_length : Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size).
[in]	wrapped_key	Inputs the 2048-bit RSA public wrapped key.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Incorrect wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSAES_PKCS2048_Decrypt()

fsp_err_t R_SCE_RSAES_PKCS2048_Decrypt (sce_rsa_byte_data_t * *cipher*, sce_rsa_byte_data_t * *plain*, sce_rsa2048_private_wrapped_key_t * *wrapped_key*)

The R_SCE_RSAES_PKCS2048_Decrypt() function RSA-decrypts the ciphertext input to the first argument, cipher, according to RSAES-PKCS1-V1_5. It writes the decryption result to the second argument, plain.

Parameters

[in]	cipher	ciphertext • cipher->pdata : Specifies pointer to array containing ciphertext. • cipher->data_length : Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size).
[in,out]	plain	plaintext • plain->pdata : Specifies pointer to array containing plaintext. • plain->data_length : Inputs plaintext buffer size. The following size is required. Plaintext buffer size >= public key n size -11. Outputs valid data length after decryption (public key n size).
[in]	wrapped_key	Inputs the 1024-bit RSA private wrapped key.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Incorrect wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.

FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSAES_PKCS3072_Encrypt()

fsp_err_t R_SCE_RSAES_PKCS3072_Encrypt (sce_rsa_byte_data_t * *plain*, sce_rsa_byte_data_t * *cipher*, sce_rsa3072_public_wrapped_key_t * *wrapped_key*)

The R_SCE_RSAES_PKCS3072_Encrypt() function RSA-encrypts the plaintext input to the first argument, plain, according to RSAES-PKCS1-V1_5. It writes the encryption result to the second argument, cipher.

Parameters

[in]	plain	plaintext • plain->pdata : Specifies pointer to array containing plaintext. • plain->data_length : Specifies valid data length of plaintext array. data size <= public key n size - 11
[in,out]	cipher	ciphertext • cipher->pdata : Specifies pointer to array containing ciphertext. • cipher->data_length : Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size).
[in]	wrapped_key	Inputs the 3072-bit RSA public wrapped key.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Incorrect wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_RSAES_PKCS4096_Encrypt()

fsp_err_t R_SCE_RSAES_PKCS4096_Encrypt (sce_rsa_byte_data_t * *plain*, sce_rsa_byte_data_t * *cipher*, sce_rsa4096_public_wrapped_key_t * *wrapped_key*)

The R_SCE_RSAES_PKCS4096_Encrypt() function RSA-encrypts the plaintext input to the first argument, plain, according to RSAES-PKCS1-V1_5. It writes the encryption result to the second argument, cipher.

Parameters

[in]	plain	plaintext • plain->pdata : Specifies pointer to array containing plaintext. • plain->data_length : Specifies valid data length of plaintext array. data size <= public key n size - 11
[in,out]	cipher	ciphertext • cipher->pdata : Specifies pointer to array containing ciphertext. • cipher->data_length : Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size).
[in]	wrapped_key	Inputs the 4096-bit RSA public wrapped key.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Incorrect wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is illegal.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDSA_secp192r1_SignatureGenerate()

fsp_err_t R_SCE_ECDSA_secp192r1_SignatureGenerate (sce_ecdsa_byte_data_t * *message_hash*, sce_ecdsa_byte_data_t * *signature*, sce_ecc_private_wrapped_key_t * *wrapped_key*)

When a message is specified in the first argument, message_hash->data_type, a SHA-256 hash of the message text input as the first argument, message_hash->pdata, is calculated, and the signature text is written to the second argument, signature, in accordance with secp192r1 using the private wrapped key input as the third argument, wrapped_key.

When a hash value is specified in the first argument, message_hash->data_type, the signature text for the first 24 bytes of the SHA-256 hash value input to the first argument, message_hash->pdata, is written to the second argument, signature, in accordance with secp192r1 using the private wrapped key input as the third argument, wrapped_key.

Parameters

[in]	message_hash	Message or hash value to which to attach signature • message_hash->pdata : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1)
[in,out]	signature	Signature text storage destination information • signature->pdata : Specifies pointer to array storing signature text The signature format is "0 padding (64 bits) || signature r (192 bits) || 0 padding (64 bits) || signature s (192 bits)". • signature->data_length : Data length (byte units)

[in]	wrapped_key	Input wrapped key of secp192r1 private key.

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDSA_secp224r1_SignatureGenerate()

fsp_err_t R_SCE_ECDSA_secp224r1_SignatureGenerate (sce_ecdsa_byte_data_t * *message_hash*, sce_ecdsa_byte_data_t * *signature*, sce_ecc_private_wrapped_key_t * *wrapped_key*)

When a message is specified in the first argument, message_hash->data_type, a SHA-256 hash of the message text input as the first argument, message_hash->pdata, is calculated, and the signature text is written to the second argument, signature, in accordance with secp224r1 using the private wrapped key input as the third argument, wrapped_key.

When a hash value is specified in the first argument, message_hash->data_type, the signature text for the first 28 bytes of the SHA-256 hash value input to the first argument, message_hash->pdata, is written to the second argument, signature, in accordance with secp224r1 using the private wrapped key input as the third argument, wrapped_key.

Parameters

[in]	message_hash	Message or hash value to which to attach signature • message_hash->pdata : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash

		(Message: 0 Hash value: 1)
[in,out]	signature	Signature text storage destination information • signature->pdata : Specifies pointer to array storing signature text The signature format is "0 padding (32 bits) || signature r (224 bits) || 0 padding (32 bits) || signature s (224 bits)". • signature->data_length : Data length (byte units)
[in]	wrapped_key	Input wrapped key of secp224r1 private key.

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDSA_secp256r1_SignatureGenerate()

fsp_err_t R_SCE_ECDSA_secp256r1_SignatureGenerate (sce_ecdsa_byte_data_t * *message_hash*, sce_ecdsa_byte_data_t * *signature*, sce_ecc_private_wrapped_key_t * *wrapped_key*)

When a message is specified in the first argument, message_hash->data_type, a SHA-256 hash of the message text input as the first argument, message_hash->pdata, is calculated, and the signature text is written to the second argument, signature, in accordance with secp256r1 using the private wrapped key input as the third argument, wrapped_key.

When a hash value is specified in the first argument, message_hash->data_type, the signature text for the first 32 bytes of the SHA-256 hash value input to the first argument, message_hash->pdata, is written to the second argument, signature, in accordance with secp256r1 using the private wrapped key input as the third argument, wrapped_key.

Parameters

[in]	message_hash	Message or hash value to which to attach signature • message_hash->pdata : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1)
[in,out]	signature	Signature text storage destination information • signature->pdata : Specifies pointer to array storing signature text The signature format is "signature r (256 bits) || signature s (256 bits)". • signature->data_length : Data length (byte units)
[in]	wrapped_key	Input wrapped key of secp256r1 private key.

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDSA_secp384r1_SignatureGenerate()

fsp_err_t R_SCE_ECDSA_secp384r1_SignatureGenerate (sce_ecdsa_byte_data_t * *message_hash*, sce_ecdsa_byte_data_t * *signature*, sce_ecc_private_wrapped_key_t * *wrapped_key*)

When a message is specified in the first argument, message_hash->data_type, a SHA-384 hash of the message text input as the first argument, message_hash->pdata, is calculated, and the signature text is written to the second argument, signature, in accordance with secp384r1 using the private wrapped key input as the third argument, wrapped_key.

To use message input, prepare a user-defined function for SHA384.

When a hash value is specified in the first argument, message_hash->data_type, the signature text for the first 48 bytes of the SHA-384 hash value input to the first argument, message_hash->pdata, is written to the second argument, signature, in accordance with secp384r1 using the private wrapped key input as the third argument, wrapped_key.

Parameters

[in]	message_hash	Message or hash value to which to attach signature • message_hash->pdata : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash

		value: 1)
[in,out]	signature	Signature text storage destination information • signature->pdata : Specifies pointer to array storing signature text The signature format is "signature r (384 bits) || signature s (384 bits)". • signature->data_length : Data length (byte units)
[in]	wrapped_key	Input wrapped key of secp384r1 private key.

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDSA_secp192r1_SignatureVerify()

fsp_err_t R_SCE_ECDSA_secp192r1_SignatureVerify (sce_ecdsa_byte_data_t * *signature*, sce_ecdsa_byte_data_t * *message_hash*, sce_ecc_public_wrapped_key_t * *wrapped_key*)

When a message is specified in the second argument, message_hash->data_type, a SHA-256 hash of the message text input as the second argument, message_hash->pdata, is calculated, and the signature text input to the first argument, signature, is validated in accordance with secp192r1 using the public wrapped key input as the third argument, wrapped_key.

When a hash value is specified in the second argument, message_hash->data_type, the signature text for the first 24 bytes of the SHA-256 hash value input to the second argument, message_hash->pdata, input to the first argument, signature, is validated in accordance with secp192r1 using the public wrapped key input as the third argument, wrapped_key.

Parameters

[in]	signature	Signature text information to be verified • signature->pdata : Specifies pointer to array storing signature text The signature format is "0 padding (64 bits) || signature r (192 bits) || 0 padding (64 bits) || signature s (192 bits)". • signature->data_length : Specifies the data length (byte units) (nonuse)
[in,out]	message_hash	Message or hash value to be verified • message_hash->pdata : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1)

[in]	wrapped_key	Input wrapped key of secp192r1 public key.

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred or signature verification failed.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDSA_secp224r1_SignatureVerify()

fsp_err_t R_SCE_ECDSA_secp224r1_SignatureVerify (sce_ecdsa_byte_data_t * *signature*, sce_ecdsa_byte_data_t * *message_hash*, sce_ecc_public_wrapped_key_t * *wrapped_key*)

When a message is specified in the second argument, message_hash->data_type, a SHA-256 hash of the message text input as the second argument, message_hash->pdata, is calculated, and the signature text input to the first argument, signature, is validated in accordance with secp224r1 using the public wrapped key input as the third argument, wrapped_key.

When a hash value is specified in the second argument, message_hash->data_type, the signature text for the first 28 bytes of the SHA-256 hash value input to the second argument, message_hash->pdata, input to the first argument, signature, is validated in accordance with secp224r1 using the public wrapped key input as the third argument, wrapped_key.

Parameters

[in]	signature	Signature text information to be verified • signature->pdata : Specifies pointer to array storing signature text The signature format is "0 padding (32 bits) || signature r (224 bits) || 0 padding (32 bits) || signature s (224 bits)". • signature->data_length : Specifies the data length (byte units) (nonuse)

[in,out]	message_hash	Message or hash value to be verified • message_hash->pdata : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1)
[in]	wrapped_key	Input wrapped key of secp224r1 public key.

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred or signature verification failed.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDSA_secp256r1_SignatureVerify()

fsp_err_t R_SCE_ECDSA_secp256r1_SignatureVerify (sce_ecdsa_byte_data_t * *signature*, sce_ecdsa_byte_data_t * *message_hash*, sce_ecc_public_wrapped_key_t * *wrapped_key*)

When a message is specified in the second argument, message_hash->data_type, a SHA-256 hash of the message text input as the second argument, message_hash->pdata, is calculated, and the signature text input to the first argument, signature, is validated in accordance with secp256r1 using the public wrapped key input as the third argument, wrapped_key.

When a hash value is specified in the second argument, message_hash->data_type, the signature text for the first 32 bytes of the SHA-256 hash value input to the second argument, message_hash->pdata, input to the first argument, signature, is validated in accordance with secp256r1 using the public wrapped key input as the third argument, wrapped_key.

Parameters

[in]	signature	Signature text information to be verified • signature->pdata : Specifies pointer to array storing signature text The signature format is "signature r (256 bits) || signature s (256 bits)". • signature->data_length : Specifies the data length (byte units) (nonuse)
[in,out]	message_hash	Message or hash value to be verified • message_hash->pdata : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1)
[in]	wrapped_key	Input wrapped key of

		secp256r1 public key.

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred or signature verification failed.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDSA_secp384r1_SignatureVerify()

fsp_err_t R_SCE_ECDSA_secp384r1_SignatureVerify (sce_ecdsa_byte_data_t * *signature*, sce_ecdsa_byte_data_t * *message_hash*, sce_ecc_public_wrapped_key_t * *wrapped_key*)

When a message is specified in the second argument, message_hash->data_type, a SHA-256 hash of the message text input as the second argument, message_hash->pdata, is calculated, and the signature text input to the first argument, signature, is validated in accordance with secp384r1 using the public wrapped key input as the third argument, wrapped_key.

To use message input, prepare a user-defined function for SHA384.

When a hash value is specified in the second argument, message_hash->data_type, the signature text for the first 48 bytes of the SHA-256 hash value input to the second argument, message_hash->pdata, input to the first argument, signature, is validated in accordance with secp384r1 using the public wrapped key input as the third argument, wrapped_key.

Parameters

[in]	signature	Signature text information to be verified • signature->pdata : Specifies pointer to array storing signature text The signature format is "signature r (384 bits) || signature s (384 bits)". • signature->data_length : Specifies the data length (byte units) (nonuse)

[in,out]	message_hash	Message or hash value to be verified • message_hash->pdata : Specifies pointer to array storing the message or hash value • message_hash->data_length : Specifies effective data length of the array (Specify only when Message is selected) • message_hash->data_type : Selects the data type of message_hash (Message: 0 Hash value: 1)
[in]	wrapped_key	Input wrapped key of secp384r1 public key.

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred or signature verification failed.
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDH_secp256r1_Init()

fsp_err_t R_SCE_ECDH_secp256r1_Init (sce_ecdh_handle_t * *handle*, uint32_t *key_type*, uint32_t *use_key_id*)

The R_SCE_ECDH_secp256r1_Init() function prepares to perform ECDH key exchange computation and writes the result to the first argument, handle. The succeeding functions R_SCE_ECDH_secp256r1_PublicKeySign(), R_SCE_ECDH_secp256r1_PublicKeyVerify(), R_SCE_ECDH_secp256r1_SharedSecretCalculate(), and R_SCE_ECDH_secp256r1_KeyDerivation() use handle as an argument.

Use the second argument, key_type, to select the type of ECDH key exchange. When ECDHE is selected, the R_SCE_ECDH_secp256r1_PublicKeySign() function uses the SCE's random number generation functionality to generate an secp256r1 key pair. When ECDH is selected, keys installed beforehand are used for key exchange.

Input 1 as the third argument, use_key_id, to use key_id when key exchange is performed. key_id is for applications conforming to the DLMS/COSEM standard for smart meters.

Parameters

[in,out]	handle	ECDH handler (work area)
[in]	key_type	Key exchange type (0: ECDHE, 1: ECDH, 2:ECDH(AES-GCM-128 with IV))
[in]	use_key_id	0: key_id not used, 1: key_id used

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_PARAMETER	Input data is invalid.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDH_secp256r1_PublicKeySign()

fsp_err_t R_SCE_ECDH_secp256r1_PublicKeySign (sce_ecdh_handle_t * *handle*, sce_ecc_public_wrapped_key_t * *ecc_public_wrapped_key*, sce_ecc_private_wrapped_key_t * *ecc_private_wrapped_key*, uint8_t * *public_key*, sce_ecdsa_byte_data_t * *signature*, sce_ecc_private_wrapped_key_t * *wrapped_key*)

The R_SCE_ECDH_secp256r1_PublicKeySign() function calculates a signature for a public key user wrapped key used for ECDH key exchange.

If ECDHE is specified by the key_type argument of the R_SCE_ECDH_secp256r1_Init() function, the SCE's random number generation functionality is used to generate an secp256r1 key pair. The public key is output to public_key and the private key is output to wrapped_key.

If ECDH is specified by the key_type argument of the R_SCE_ECDH_secp256r1_Init() function, the public key input as ecc_public_wrapped_key is output to public_key and nothing is output to wrapped_key.

The succeeding function R_SCE_ECDH_secp256r1_SharedSecretCalculate() uses the first argument, handle, as an argument. R_SCE_ECDH_secp256r1_SharedSecretCalculate() function uses wrapped_key as input to calculate Z.

Parameters

[in,out]	handle	ECDH handler (work area) When using key_id, input handle->key_id after running R_SCE_ECDH_secp256r1_Init().
[in]	ecc_public_wrapped_key	For ECDHE, input a null pointer. For ECDH, input the wrapped key of a secp256r1 public key.
[in]	ecc_private_wrapped_key	secp256r1 private key for signature generation
[in,out]	public_key	User secp256r1 public key (512-bit) for key exchange. When using key_id, key_id (8-bit) || public key (512-bit) || 0 padding (24-bit)
[in,out]	signature	Signature text storage destination information • signature->pdata : Specifies pointer to array storing signature text. The signature format is "signature r (256 bits) || signature s (256 bits)" • signature->data_leng

		th : Data length (in byte units)
[in,out]	wrapped_key	For ECDHE, a private wrapped key generated from a random number. Not output for ECDH.

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDH_secp256r1_PublicKeyVerify()

fsp_err_t R_SCE_ECDH_secp256r1_PublicKeyVerify (sce_ecdh_handle_t * *handle*, sce_ecc_public_wrapped_key_t * *ecc_public_wrapped_key*, uint8_t * *public_key_data*, sce_ecdsa_byte_data_t * *signature*, sce_ecc_public_wrapped_key_t * *wrapped_key*)

The R_SCE_ECDH_secp256r1_PublicKeyVerify() function verifies the signature of the secp256r1 public key of the other ECDH key exchange party. If the signature is correct, it outputs the public wrapped key to the fifth argument. The first argument, handle, is used as an argument in the subsequent function R_SCE_ECDH_secp256r1_SharedSecretCalculate(). R_SCE_ECDH_secp256r1_SharedSecretCalculate() uses wrapped_key as input to calculate Z.

Parameters

[in,out]	handle	ECDH handler (work area)
[in]	ecc_public_wrapped_key	Public wrapped key area for signature verification
[in]	public_key_data	secp256r1 public key (512-bit). When key_id is used: key_id (8-bit) || public key (512-bit)
[in]	signature	ECDSA secp256r1 signature of ecc_public_wrapped_key
[in,out]	wrapped_key	wrapped key of ecc_public_wrapped_key

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred or signature verification failed.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDH_secp256r1_PublicKeyReadWithoutSignature()

fsp_err_t R_SCE_ECDH_secp256r1_PublicKeyReadWithoutSignature (sce_ecdh_handle_t * *handle*, uint8_t * *public_key_data*, sce_ecc_public_wrapped_key_t * *wrapped_key*)

The R_SCE_ECDH_secp256r1_PublicKeyReadWithoutSignature() function reads the secp256r1 public key of the other ECDH key exchange party and outputs the public wrapped key to the third argument. The first argument, handle, is used as an argument in the subsequent function R_SCE_ECDH_secp256r1_SharedSecretCalculate(). R_SCE_ECDH_secp256r1_SharedSecretCalculate() uses wrapped_key as input to calculate Z. This API does not verify signature of public_key_data, please protect this data by upper layer software.

Parameters

[in,out]	handle	ECDH handler (work area)
[in]	public_key_data	secp256r1 public key (512-bit). When key_id is used: key_id (8-bit) || public key (512-bit)
[in,out]	wrapped_key	wrapped key of public_key_data

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State. Please note that this is slightly contrary to the protected mode policy as it omits signature verification.

◆ R_SCE_ECDH_secp256r1_SharedSecretCalculate()

fsp_err_t R_SCE_ECDH_secp256r1_SharedSecretCalculate (sce_ecdh_handle_t * *handle*, sce_ecc_public_wrapped_key_t * *ecc_public_wrapped_key*, sce_ecc_private_wrapped_key_t * *ecc_private_wrapped_key*, sce_ecdh_wrapped_key_t * *shared_secret_wrapped_key*)

The R_SCE_ECDH_secp256r1_SharedSecretCalculate() function uses the ECDH key exchange algorithm to output the wrapped key of the shared secret Z derived from the public key of the other key exchange party and your own private key. Input as the second argument, ecc_public_wrapped_key, the public wrapped key whose signature was verified by R_SCE_ECDH_secp256r1_PublicKeyVerify(). When key_type of R_SCE_ECDH_secp256r1_Init() is 0, input as the third argument, ecc_private_wrapped_key, the private wrapped key generated from a random number by R_SCE_ECDH_secp256r1_PublicKeySign(), and when key_type is other than 0, input the private wrapped key that forms a pair with the second argument of R_SCE_ECDH_secp256r1_PublicKeySign(). The subsequent R_SCE_ECDH_secp256r1_KeyDerivation() function uses shared_secret_wrapped_key as key material for outputting the wrapped key.

Parameters

[in,out]	handle	ECDH handler (work area)
[in]	ecc_public_wrapped_key	Public wrapped key whose signature was verified by R_SCE_ECDH_secp256r1_PublicKeyVerify()
[in]	ecc_private_wrapped_key	Private wrapped key
[in,out]	shared_secret_wrapped_key	Wrapped key of shared secret Z calculated by ECDH key exchange

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_ECDH_secp256r1_KeyDerivation()

fsp_err_t R_SCE_ECDH_secp256r1_KeyDerivation (sce_ecdh_handle_t * *handle*, sce_ecdh_wrapped_key_t * *shared_secret_wrapped_key*, uint32_t *key_type*, uint32_t *kdf_type*, uint8_t * *other_info*, uint32_t *other_info_length*, sce_hmac_sha_wrapped_key_t * *salt_wrapped_key*, sce_aes_wrapped_key_t * *wrapped_key*)

The R_SCE_ECDH_secp256r1_KeyDerivation() function uses the shared secret "Z (shared_secret_index)" calculated by the R_SCE_ECDH_secp256r1_SharedSecretCalculate() function as the key material to derive the wrapped key specified by the third argument, key_type. The key derivation algorithm is one-step key derivation as defined in NIST SP800-56C. Either SHA-256 or SHA-256 HMAC is specified by the fourth argument, kdf_type. When SHA-256 HMAC is specified, the wrapped key output by the R_SCE_SHA256HMAC_EncryptedKeyWrap() function is specified as the seventh argument, salt_wrapped_key. Enter a fixed value for deriving a key shared with the key exchange partner in the fifth argument, other_info. A wrapped key corresponding to key_type is output as the eighth argument, wrapped_key. The correspondences between the types of derived wrapped_key and the functions with which they can be used as listed below.

- AES-128: All AES-128 Init functions
- AES-256: All AES-256 Init functions
- SHA256-HMAC: R_SCE_SHA256HMAC_GenerateInit() function and R_SCE_SHA256HMAC_VerifyInit() function

Parameters

[in,out]	handle	ECDH handler (work area)
[in]	shared_secret_wrapped_key	Z wrapped key calculated by R_SCE_ECDH_secp256r1_SharedSecretCalculate
[in]	key_type	Derived key type (0: AES-128, 1: AES-256, 2:SHA256-HMAC, 3: AES-GCM-128 with IV)
[in]	kdf_type	Algorithm used for key derivation calculation (0: SHA-256, 1:SHA256-HMAC)
[in]	other_info	Additional data used for key derivation calculation: AlgorithmID || PartyUInfo || PartyVInfo
[in]	other_info_length	Data length of other_info (up to 147 byte units)
[in]	salt_wrapped_key	Salt wrapped key (Input NULL when kdf_type is 0.)
[in,out]	wrapped_key	Wrapped key corresponding to key_type. When the value of key_type is 2, an SHA256-HMAC wrapped key is output. wrapped_key can be specified by casting the start address of the area

		reserved beforehand by the sce_hmac_sha_wrapped_key_t type with the (sce_aes_wrapped_key_t*) type.

Return values

FSP_SUCCESS	Normal end
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
FSP_ERR_CRYPTO_SCE_KEY_SET_FAIL	Invalid wrapped key was input.
FSP_ERR_CRYPTO_SCE_PARAMETER	An invalid handle was input.
FSP_ERR_CRYPTO_SCE_FAIL	Internal error occurred.
FSP_ERR_CRYPTO_SCE_PROHIBIT_FUNCTION	An invalid function was called.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_TLS_RootCertificateRSA2048PublicKeyInstall()

fsp_err_t R_SCE_TLS_RootCertificateRSA2048PublicKeyInstall (uint8_t * *encrypted_provisioning_key*, uint8_t * *initial_vector*, uint8_t * *encrypted_key*, sce_tls_ca_certification_public_wrapped_key_t * *wrapped_key*)

Generate TLS RSA Public key index data

Parameters

[in]	encrypted_provisioning_key	the provisioning key includes encrypted CBC/CBC-MAC key for user key
[in]	initial_vector	the initial_vector for user key CBC encrypt
[in]	encrypted_key	the user key encrypted with AES128-ECB mode
[out]	wrapped_key	the user Key Generation Information (141 words) of RSA2048 bit

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate()

fsp_err_t R_SCE_TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate (sce_tls_p256_ecc_wrapped_key_t * *tls_p256_ecc_wrapped_key*, uint8_t * *ephemeral_ecdh_public_key*)

Generate TLS ECC key pair

Parameters

[in]	tls_p256_ecc_wrapped_key	P256 ECC key index for TLS
[in]	ephemeral_ecdh_public_key	ephemeral ECDH public key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_TLS_RootCertificateVerify()

fsp_err_t R_SCE_TLS_RootCertificateVerify (uint32_t *public_key_type*, uint8_t * *certificate*, uint32_t *certificate_length*, uint32_t *public_key_n_start_position*, uint32_t *public_key_n_end_position*, uint32_t *public_key_e_start_position*, uint32_t *public_key_e_end_position*, uint8_t * *signature*, uint32_t * *encrypted_root_public_key*)

Verify root CA certificate.

Parameters

[in]	public_key_type	key type
[in]	certificate	certificates.
[in]	certificate_length	byte size of certificates.
[in]	public_key_n_start_position	start position of public key n.
[in]	public_key_n_end_position	end position of public key n.
[in]	public_key_e_start_position	start position of public key e.
[in]	public_key_e_end_position	end position of public key e.
[in]	signature	signature for certificates.
[out]	encrypted_root_public_key	public key for RSA 2048bit.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_TLS_CertificateVerify()

fsp_err_t R_SCE_TLS_CertificateVerify (uint32_t *public_key_type*, uint32_t * *encrypted_input_public_key*, uint8_t * *certificate*, uint32_t *certificate_length*, uint8_t * *signature*, uint32_t *public_key_n_start_position*, uint32_t *public_key_n_end_position*, uint32_t *public_key_e_start_position*, uint32_t *public_key_e_end_position*, uint32_t * *encrypted_output_public_key*)

Verify server certificate and intermediate certificate.

Parameters

[in]	public_key_type	key type
[in]	encrypted_input_public_key	public key.
[in]	certificate	certificates.
[in]	certificate_length	byte size of certificates.
[in]	signature	signature for certificates.
[in]	public_key_n_start_position	start position of public key n.
[in]	public_key_n_end_position	end position of public key n.
[in]	public_key_e_start_position	start position of public key e.
[in]	public_key_e_end_position	end position of public key e.
[out]	encrypted_output_public_key	public key for RSA 2048bit.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_TLS_PreMasterSecretEncryptWithRSA2048()

fsp_err_t R_SCE_TLS_PreMasterSecretEncryptWithRSA2048 (uint32_t * *encrypted_public_key*, uint32_t * *sce_pre_master_secret*, uint8_t * *encrypted_pre_master_secret*)

Output the result encrypted pre-master secret with RSA 2048bit

Parameters

[in]	encrypted_public_key	public key data.
[in]	sce_pre_master_secret	pre-master secret value.
[out]	encrypted_pre_master_secret	the value encrypted pre-master secret.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_TLS_PreMasterSecretGenerateForRSA2048()

fsp_err_t R_SCE_TLS_PreMasterSecretGenerateForRSA2048 (uint32_t * *sce_pre_master_secret*)

Generate encrypted pre-master secret.

Parameters

[out]	sce_pre_master_secret	pre-master secret value for SCE.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_TLS_MasterSecretGenerate()

fsp_err_t R_SCE_TLS_MasterSecretGenerate (uint32_t *select_cipher_suite*, uint32_t * *sce_pre_master_secret*, uint8_t * *client_random*, uint8_t * *server_random*, uint32_t * *sce_master_secret*)

Generate encrypted master secret.

Parameters

[in]	select_cipher_suite	cipher suite type
[in]	sce_pre_master_secret	pre-master secret value for SCE.
[in]	client_random	random value reported ClientHello.
[in]	server_random	random value reported ServerHello.
[out]	sce_master_secret	master secret value for SCE.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_TLS_SessionKeyGenerate()

fsp_err_t R_SCE_TLS_SessionKeyGenerate (uint32_t *select_cipher_suite*, uint32_t * *sce_master_secret*, uint8_t * *client_random*, uint8_t * *server_random*, uint8_t * *nonce_explicit*, sce_hmac_sha_wrapped_key_t * *client_mac_wrapped_key*, sce_hmac_sha_wrapped_key_t * *server_mac_wrapped_key*, sce_aes_wrapped_key_t * *client_crypto_wrapped_key*, sce_aes_wrapped_key_t * *server_crypto_wrapped_key*, uint8_t * *client_initial_vector*, uint8_t * *server_initial_vector*)

Output various key information.

Parameters

[in]	select_cipher_suite	Key suite information number.
[in]	sce_master_secret	master secret value.
[in]	client_random	random value reported ClientHello.
[in]	server_random	random value reported ServerHello.
[in]	nonce_explicit	nonce value
[out]	client_mac_wrapped_key	the mac key during communication from client to server.
[out]	server_mac_wrapped_key	the mac key during communication from server to client.
[out]	client_crypto_wrapped_key	the crypto key during communication from client to server.
[out]	server_crypto_wrapped_key	the crypto key during communication from server to client.
[in]	client_initial_vector	not use.
[in]	server_initial_vector	not use.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_TLS_VerifyDataGenerate()

fsp_err_t R_SCE_TLS_VerifyDataGenerate (uint32_t *select_verify_data*, uint32_t * *sce_master_secret*, uint8_t * *hand_shake_hash*, uint8_t * *verify_data*)

Generate verify data.

Parameters

[in]	select_verify_data	Select Client/Server data.
[in]	sce_master_secret	master secret data.
[in]	hand_shake_hash	TLS hand shake message SHA256 HASH value.
[out]	verify_data	verify data.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_TLS_ServerKeyExchangeVerify()

fsp_err_t R_SCE_TLS_ServerKeyExchangeVerify (uint32_t *public_key_type*, uint8_t * *client_random*, uint8_t * *server_random*, uint8_t * *server_ephemeral_ecdh_public_key*, uint8_t * *server_key_exchange_signature*, uint32_t * *encrypted_public_key*, uint32_t * *encrypted_ephemeral_ecdh_public_key*)

Retrives ECDH public key.

Parameters

[in]	public_key_type	key type
[in]	client_random	random value reported ClientHello.
[in]	server_random	random value reported ServerHello.
[in]	server_ephemeral_ecdh_public_key	Ephemeral ECDH public key from Server.
[in]	server_key_exchange_signature	Server Key Exchange sigunature.
[in]	encrypted_public_key	encrypted public key.
[out]	encrypted_ephemeral_ecdh_public_key	encrypted Ephemeral ECDH public key.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

◆ R_SCE_TLS_PreMasterSecretGenerateForECC_secp256r1()

fsp_err_t R_SCE_TLS_PreMasterSecretGenerateForECC_secp256r1 (uint32_t * *encrypted_public_key*, sce_tls_p256_ecc_wrapped_key_t * *tls_p256_ecc_wrapped_key*, uint32_t * *sce_pre_master_secret*)

Generate encrypted pre-master secret.

Parameters

[in]	encrypted_public_key	encrypted public key
[in]	tls_p256_ecc_wrapped_key	P-256 ECC key index.
[out]	sce_pre_master_secret	encrypted pre-master secret value for SCE.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

Note

The pre-run state is SCE Enabled State. After the function runs the state transitions to SCE Enabled State.

4.2.50 Serial Communications Interface (SCI) I2C (r_sci_b_i2c)

Modules

Functions

fsp_err_t	R_SCI_B_I2C_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg)
fsp_err_t	R_SCI_B_I2C_Close (i2c_master_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_B_I2C_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)
fsp_err_t	R_SCI_B_I2C_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)
fsp_err_t	R_SCI_B_I2C_Abort (i2c_master_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_B_I2C_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode)

fsp_err_t	R_SCI_B_I2C_CallbackSet (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void const *const p_context, i2c_master_callback_args_t *const p_callback_memory)
fsp_err_t	R_SCI_B_I2C_StatusGet (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status)

Detailed Description

Driver for the SCI_B peripheral on RA MCUs. This module implements the I2C Master Interface.

Overview

The Simple I2C master on SCI_B HAL module supports transactions with an I2C Slave device. Callbacks must be provided which would be invoked when a transmission or receive has been completed. The callback arguments will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100 kHz transaction rate.
  - Fast Mode Support with up to 400 kHz transaction rate.
- SDA Delay in nanoseconds can be specified as a part of the configuration.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
  - Optional (build time) DTC support for read and write respectively.
  - Optional (build time) support for 10-bit slave addressing.

Configuration

Build Time Configurations for r_sci_b_i2c

The following build time configurations are defined in fsp_cfg/r_sci_b_i2c_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DTC on Transmission and Reception	• Enabled • Disabled	Disabled	If enabled, DTC instances will be included in the build for both transmission and reception.
10-bit slave addressing	• Enabled	Disabled	If enabled, the driver

	• Disabled		will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode.

Configurations for Connectivity > I2C Master (r_sci_b_i2c)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Master (r_sci_b_i2c). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_i2c0	Module name.
Channel	Value must be an integer between 0 and 9	0	Select the SCI channel.
Slave Address	Value must be a hex value	0x00	Specify the slave address.
Address Mode	• 7-Bit • 10-Bit	7-Bit	Select the address mode.
Rate	• Standard • Fast-mode	Standard	Select the I2C data rate. If the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated transfer rate and SDA delay are printed in a comment in the generated sci_b_i2c_extended_cfg_t structure.
SDA Output Delay (nano seconds)	Must be a valid non-negative integer with maximum configurable value of 300	300	Specify the SDA output delay in nanoseconds.
Noise filter setting	• Use clock signal divided by 1 with noise filter • Use clock signal divided by 2 with noise filter	Use clock signal divided by 1 with noise filter	Select the sampling clock for the digital noise filter

	• Use clock signal divided by 4 with noise filter • Use clock signal divided by 8 with noise filter		
Clock Source	• PCLK • SCISPICLK	PCLK	Select the clock source for the SCI I2C module.
Bit Rate Modulation	• Enable • Disable	Enable	Enabling bitrate modulation reduces the percent error of the actual bitrate with respect to the requested baud rate. It does this by modulating the number of cycles per clock output pulse, so the clock is no longer a square wave.
Callback	Name must be a valid C symbol	sci_b_i2c_master_callback	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).
Interrupt Priority Level	MCU Specific Options		Select the interrupt priority level. This is set for TXI, RXI (if used), TEI interrupts.
RX Interrupt Priority Level [Only used when DTC is enabled]	MCU Specific Options		Select the interrupt priority level. This is set for RXI only when DTC is enabled.

Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

MCU Group	Peripheral Clock
RA2A1	PCLKB
RA2E1	PCLKB
RA2E2	PCLKB
RA2L1	PCLKB
RA4E1	PCLKA

RA4M1	PCLKA
RA4M2	PCLKA
RA4M3	PCLKA
RA4W1	PCLKA
RA6E1	PCLKA
RA6M1	PCLKA
RA6M2	PCLKA
RA6M3	PCLKA
RA6M4	PCLKA
RA6M5	PCLKA
RA6T1	PCLKA
RA6T2	PCLKA

The actual I2C transfer rate can be derived from either SCISPICLK or the peripheral clock (PCLK)[1], and will be calculated and set by the tooling depending on the selected transfer rate and the SDA delay. If the selected clock is configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

Note

1. See Figure 26.2 in the RA6T2 manual for more information.

Pin Configuration

The SCI_B I2C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

Usage Notes

Interrupt Configuration

- Receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel used must be enabled in the properties of the selected device.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

SCI_B I2C Master Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate and SDA Delay. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLK settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

Enabling DTC with the SCI_B I2C

- DTC transfer support is configurable and is disabled from the build by default. SCI_B I2C

driver provides two DTC instances for transmission and reception respectively.
- DTC is helpful for minimizing interrupts during large transactions. Many I2C applications have shorter transactions. These applications will likely not see any improvement with DTC. I2C often runs at a much slower speed than the CPU core clock. Some applications with longer transactions may prefer servicing the interrupts at the I2C bitrate to the overhead of bringing in the DTC driver.
- For further details on DTC please refer Data Transfer Controller (r_dtc)

Multiple Devices on the Bus

- A single SCI_B I2C instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet API.

Restart

- SCI_B_I2C can hold the the bus after an I2C transaction by issuing a repeated start condition.

Examples

Basic Example

This is a basic example of minimal use of the r_sci_b_i2c in an application. This example shows how this driver can be used for basic read and write operations.

```
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the I2C module */
    err = R_SCI_B_I2C_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SCI_B_I2C_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
```

```
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_B_I2C_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
       __BKPT(0);
    }
}
```

Multiple Slave devices on the same channel (bus)

This example demonstrates how a single SCI_B I2C driver can be used to communicate with different slave devices which are on the same channel.

```
void single_channel_multi_slave (void)
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_B_I2C_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Clear the recieve buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SCI_B_I2C_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
    err = R_SCI_B_I2C_SlaveAddressSet(&g_i2c_device_ctrl_2,
I2C_SLAVE_DISPLAY_ADAPTER, I2C_MASTER_ADDR_MODE_7BIT);
    assert(FSP_SUCCESS == err);
    g_i2c_tx_buffer[0]   = (uint8_t) I2C_EXAMPLE_DATA_1;
    g_i2c_tx_buffer[1]   = (uint8_t) I2C_EXAMPLE_DATA_2;
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms           = I2C_TRANSACTION_BUSY_DELAY;
```

```
    err = R_SCI_B_I2C_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
}
```

Data Structures

struct	sci_b_i2c_clock_settings_t
struct	sci_b_i2c_instance_ctrl_t
struct	sci_b_i2c_extended_cfg_t

Enumerations

enum	sci_b_i2c_clock_source_t

Data Structure Documentation

◆ sci_b_i2c_clock_settings_t

struct sci_b_i2c_clock_settings_t

I2C clock settings

Data Fields		
bool	bitrate_modulation	Bit-rate Modulation Function enable or disable.
uint8_t	brr_value	Bit rate register settings.
uint8_t	clk_divisor_value	Clock Select settings.
uint8_t	mddr_value	Modulation Duty Register settings.
uint8_t	cycles_value	SDA Delay Output Cycles Select.
uint8_t	snfr_value	Noise Filter Setting Register value.

sci_b_i2c_clock_source_t	clock_source	Clock source (PCLK or SCISPICLK)

◆ sci_b_i2c_instance_ctrl_t

struct sci_b_i2c_instance_ctrl_t
I2C control structure. DO NOT INITIALIZE.

◆ sci_b_i2c_extended_cfg_t

struct sci_b_i2c_extended_cfg_t		
SCI I2C extended configuration		
Data Fields		
sci_b_i2c_clock_settings_t	clock_settings	I2C Clock settings.

Enumeration Type Documentation

◆ sci_b_i2c_clock_source_t

enum sci_b_i2c_clock_source_t
SCI clock source

Function Documentation

◆ R_SCI_B_I2C_Open()

fsp_err_t R_SCI_B_I2C_Open (i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg*)

Opens the I2C device.

Return values

FSP_SUCCESS	Requested clock rate was set exactly.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_ASSERTION	Parameter check failure due to one or more reasons below: 1. p_api_ctrl or p_cfg is NULL. 2. extended parameter is NULL. 3. Callback parameter is NULL. 4. Clock rate requested is greater than 400KHz 5. Invalid IRQ number assigned

◆ R_SCI_B_I2C_Close()

fsp_err_t R_SCI_B_I2C_Close (i2c_master_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. Power down I2C peripheral.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Return values

FSP_SUCCESS	Device closed without issue.
FSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Device was not even opened.

◆ R_SCI_B_I2C_Read()

fsp_err_t R_SCI_B_I2C_Read (i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart*)

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	The parameter p_ctrl, p_dest is NULL, bytes is 0.
FSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer.
FSP_ERR_NOT_OPEN	Device was not even opened.

◆ R_SCI_B_I2C_Write()

fsp_err_t R_SCI_B_I2C_Write (i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart*)

Performs a write to the I2C device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C write operation will begin. When no callback is provided by the user, this function performs a blocking write. Otherwise, the write operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_TX_COMPLETE in the callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	p_ctrl, p_src is NULL.
FSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer.
FSP_ERR_NOT_OPEN	Device was not even opened.

◆ R_SCI_B_I2C_Abort()

fsp_err_t R_SCI_B_I2C_Abort (i2c_master_ctrl_t *const *p_api_ctrl*)

Aborts any in-progress transfer and forces the I2C peripheral into a ready state.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Return values

FSP_SUCCESS	Transaction was aborted without issue.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Device was not even opened.

◆ R_SCI_B_I2C_SlaveAddressSet()

fsp_err_t R_SCI_B_I2C_SlaveAddressSet (i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode*)

Sets address and addressing mode of the slave device.

This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

Return values

FSP_SUCCESS	Address of the slave is set correctly.
FSP_ERR_ASSERTION	p_ctrl or address is NULL.
FSP_ERR_NOT_OPEN	Device was not even opened.
FSP_ERR_IN_USE	An I2C Transaction is in progress.

◆ R_SCI_B_I2C_CallbackSet()

fsp_err_t R_SCI_B_I2C_CallbackSet (i2c_master_ctrl_t *const *p_api_ctrl*, void(*)(i2c_master_callback_args_t *) *p_callback*, void const *const *p_context*, i2c_master_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements i2c_master_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_SCI_B_I2C_StatusGet()

fsp_err_t R_SCI_B_I2C_StatusGet (i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_status_t * *p_status*)

Provides driver status.

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.

4.2.51 Serial Communications Interface (SCI) SPI (r_sci_b_spi)

Modules

Functions

fsp_err_t	R_SCI_B_SPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg)
fsp_err_t	R_SCI_B_SPI_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SCI_B_SPI_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SCI_B_SPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SCI_B_SPI_CallbackSet (spi_ctrl_t *const p_api_ctrl, void(*p_callback)(spi_callback_args_t *), void const *const p_context, spi_callback_args_t *const p_callback_memory)
fsp_err_t	R_SCI_B_SPI_Close (spi_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_B_SPI_CalculateBitrate (uint32_t bitrate, sci_b_spi_clock_source_t clock_source, sci_b_spi_div_setting_t *sclk_div)

Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the SPI Interface.

Overview

Features

- Standard SPI Modes
  - Master or Slave Mode
  - Clock Polarity (CPOL)
    - CPOL=0 SCLK is low when idle
    - CPOL=1 SCLK is high when idle
  - Clock Phase (CPHA)
    - CPHA=0 Select data sampling on leading edge, data change on trailing edge
    - CPHA=1 Select data change on leading edge, data sampling on trailing edge
  - MSB/LSB first
- Configurable bit rate
- DTC Support
- Callback Events
  - Transfer Complete
  - RX Overflow Error (The SCI shift register is copied to the data register before previous data was read)

Configuration

Build Time Configurations for r_sci_b_spi

The following build time configurations are defined in fsp_cfg/r_sci_b_spi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DTC Support	• Enabled • Disabled	Enabled	If support for transfering data using the DTC will be compiled in.

Configurations for Connectivity > SPI (r_sci_b_spi)

This module can be added to the Stacks tab via New Stack > Connectivity > SPI (r_sci_b_spi). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_spi0	Module name.
Channel	Value must be a non-negative integer	0	Select the SCI channel.
Operating Mode	• Master • Slave	Master	Select the SPI operating mode.

Clock Phase	• Data sampling on odd edge, data variation on even edge • Data sampling on even edge, data variation on odd edge	Data sampling on odd edge, data variation on even edge	Select the clock edge to sample data.
Clock Polarity	• Low when idle • High when idle	Low when idle	Select clock level when idle.
Mode Fault Error	• Enable • Disable	Disable	Detect master/slave mode conflicts.
Bit Order	• MSB First • LSB First	MSB First	Select the data bit order.
Clock Source	• PCLK • SCISPICLK	PCLK	Select whether the peripheral clock (PCLK) or SCISPICLK is used for generating the SCK frequency.
Callback	Name must be a valid C symbol	sci_b_spi_callback	A user callback function that is called from the sci_b_spi interrupts when a transfer is completed or an error has occurred.
Receive Interrupt Priority	MCU Specific Options		Select the receive interrupt priority.
Transmit Interrupt Priority	MCU Specific Options		Select the transmit interrupt priority.
Transmit End Interrupt Priority	MCU Specific Options		Select the transmit end interrupt priority.
Error Interrupt Priority	MCU Specific Options		Select the error interrupt priority.
Bitrate	Value must be an integer greater than 0	8000000	Enter the desired bitrate. If the requested bitrate cannot be achieved, the settings with the largest possible value that is less than or equal to the requested bitrate is used. The theoretical bitrate is printed in a comment in the generated sci_spi_extended_cfg_t structure.

Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

MCU Group	Peripheral Clock
RA2A1	PCLKB
RA2E1	PCLKB
RA2E2	PCLKB
RA2L1	PCLKB
RA4E1	PCLKA
RA4M1	PCLKA
RA4M2	PCLKA
RA4M3	PCLKA
RA4W1	PCLKA
RA6E1	PCLKA
RA6M1	PCLKA
RA6M2	PCLKA
RA6M3	PCLKA
RA6M4	PCLKA
RA6M5	PCLKA
RA6T1	PCLKA
RA6T2	PCLKA

Pin Configuration

This module uses SCIn_MOSI, SCIn_MISO, SCIn_SPCK, and SCIn_SS pins to communicate with on board devices.

Note

At high bit rates, it might be necessary to configure the pins with IOPORT_CFG_DRIVE_HIGH.

Usage Notes

Transfer Complete Event

The transfer complete event is triggered when all of the data has been transfered. In slave mode if the SS pin is de-asserted then no transfer complete event is generated until the SS pin is asserted and the remaining data is transferred.

Performance

At high bit rates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. In slave mode this could result in RX Overflow errors.

In order to improve performance at high bit rates, it is recommended that the instance be configured to service transfers using the DTC.

Transmit From RXI Interrupt

After every byte, the SCI SPI peripheral generates a transmit buffer empty interrupt and a receive buffer full interrupt. Whenever possible, the SCI SPI module handles both interrupts in the receive buffer full interrupt. This improves performance when the DTC is not being used.

Slave Select Pin

- In master mode the slave select pin must be driven in software.
- In slave mode the hardware handles the slave select pin and will only transfer data when the SS pin is low.

Examples

Basic Example

This is a basic example of minimal use of the SCI_B_SPI module in an application.

```
static volatile bool g_transfer_complete = false;
static void r_sci_b_spi_callback (spi_callback_args_t * p_args)
{
 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = true;
    }
}
void sci_b_spi_basic_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 /* Configure Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Configure Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SCI_B_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
```

```
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SCI_B_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Wait for minimum time required between transfers. */
 R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
 /* Assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SCI_B_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
}
```

Data Structures

struct	sci_b_spi_div_setting_t

Data Structure Documentation

◆ sci_b_spi_div_setting_t

struct sci_b_spi_div_setting_t
Settings for adjusting the SPI CLK.

Function Documentation

◆ R_SCI_B_SPI_Open()

fsp_err_t R_SCI_B_SPI_Open (spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg*)

Initialize a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enables the clock for the SCI channel.
- Initializes the associated registers with default value and the user-configurable options.
- Provides the channel handle for use with other API functions.

Parameters

p_api_ctrl	Pointer to the control structure.
p_cfg	Pointer to a configuration structure.

Return values

FSP_SUCCESS	Channel initialized successfully.
FSP_ERR_ASSERTION	An input parameter is invalid or NULL.
FSP_ERR_ALREADY_OPEN	The instance has already been opened.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel number is invalid.

◆ R_SCI_B_SPI_Read()

fsp_err_t R_SCI_B_SPI_Read (spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

Receive data from an SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission by writing data to the TXD register.
- Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination.
- Complete data reception via receive buffer full interrupt and transmitting dummy data.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Parameters

	p_api_ctrl	Pointer to the control structure.
	p_dest	Pointer to the destination buffer.
[in]	length	The number of bytes to transfer.
[in]	bit_width	Invalid for SCI_B_SPI (Set to SPI_BIT_WIDTH_8_BITS).

Return values

FSP_SUCCESS	Read operation successfully completed.
FSP_ERR_ASSERTION	One of the following invalid parameters passed: • Pointer p_api_ctrl is NULL • Bit width is not 8 bits • Length is equal to 0 • Pointer to destination is NULL
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_UNSUPPORTED	The given bit_width is not supported.
FSP_ERR_IN_USE	A transfer is already in progress.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

◆ R_SCI_B_SPI_Write()

fsp_err_t R_SCI_B_SPI_Write (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

Transmit data to a SPI device. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable interrupts.
- Start data transmission with data via transmit buffer empty interrupt.
- Copy data from source buffer to the SPI data register for transmission.
- Complete data transmission via transmit buffer empty interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Parameters

	p_api_ctrl	Pointer to the control structure.
	p_src	Pointer to the source buffer.
[in]	length	The number of bytes to transfer.
[in]	bit_width	Invalid for SCI_B_SPI (Set to SPI_BIT_WIDTH_8_BITS).

Return values

FSP_SUCCESS	Write operation successfully completed.
FSP_ERR_ASSERTION	One of the following invalid parameters passed: • Pointer p_api_ctrl is NULL • Pointer to source is NULL • Length is equal to 0 • Bit width is not equal to 8 bits
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_UNSUPPORTED	The given bit_width is not supported.
FSP_ERR_IN_USE	A transfer is already in progress.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

◆ R_SCI_B_SPI_WriteRead()

fsp_err_t R_SCI_B_SPI_WriteRead (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

Simultaneously transmit data to SPI device while receiving data from SPI device (full duplex). Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission using transmit buffer empty interrupt (or by writing to the TDR register).
- Copy data from source buffer to the SPI data register for transmission.
- Receive data from receive buffer full interrupt and copy data to the destination buffer.
- Complete data transmission and reception via transmit end interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Parameters

	p_api_ctrl	Pointer to the control structure.
	p_src	Pointer to the source buffer.
	p_dest	Pointer to the destination buffer.
[in]	length	The number of bytes to transfer.
[in]	bit_width	Invalid for SCI_B_SPI (Set to SPI_BIT_WIDTH_8_BITS).

Return values

FSP_SUCCESS	Write operation successfully completed.
FSP_ERR_ASSERTION	One of the following invalid parameters passed: • Pointer p_api_ctrl is NULL • Pointer to source is NULL • Pointer to destination is NULL • Length is equal to 0 • Bit width is not equal to 8 bits
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_UNSUPPORTED	The given bit_width is not supported.
FSP_ERR_IN_USE	A transfer is already in progress.

Returns
See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- transfer_api_t::reconfigure

◆ R_SCI_B_SPI_CallbackSet()

fsp_err_t R_SCI_B_SPI_CallbackSet (spi_ctrl_t *const *p_api_ctrl*, void(*)(spi_callback_args_t *) *p_callback*, void const *const *p_context*, spi_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements spi_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_SCI_B_SPI_Close()

fsp_err_t R_SCI_B_SPI_Close (spi_ctrl_t *const *p_api_ctrl*)

Disable the SCI channel and set the instance as not open. Implements spi_api_t::close.

Parameters

p_api_ctrl	Pointer to an opened instance.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

◆ R_SCI_B_SPI_CalculateBitrate()

fsp_err_t R_SCI_B_SPI_CalculateBitrate (uint32_t *bitrate*, sci_b_spi_clock_source_t *clock_source*, sci_b_spi_div_setting_t * *sclk_div*)

Calculate the register settings required to achieve the desired bitrate.

Parameters

[in]	bitrate	bitrate [bps]. For example, 250,000; 500,00; 2,500,000 (max), etc.
	clock_source	clock source (PCLKA or SCISPICLK) used for bit rate calculation.
	sclk_div	Pointer to sci_b_spi_div_setting_t used to configure baudrate settings.

Return values

FSP_SUCCESS	Baud rate is set successfully.
FSP_ERR_ASSERTION	Baud rate is not achievable.

Note

The application must pause for 1 bit time after the BRR register is loaded before transmitting/receiving to allow time for the clock to settle.

4.2.52 Serial Communications Interface (SCI) UART (r_sci_b_uart)

Modules

Functions

fsp_err_t	R_SCI_B_UART_Open (uart_ctrl_t *const p_api_ctrl, uart_cfg_t const *const p_cfg)
fsp_err_t	R_SCI_B_UART_Read (uart_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t	R_SCI_B_UART_Write (uart_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const bytes)
fsp_err_t	R_SCI_B_UART_BaudSet (uart_ctrl_t *const p_api_ctrl, void const *const p_baud_setting)

fsp_err_t	R_SCI_B_UART_InfoGet (uart_ctrl_t *const p_api_ctrl, uart_info_t *const p_info)
fsp_err_t	R_SCI_B_UART_Close (uart_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_B_UART_Abort (uart_ctrl_t *const p_api_ctrl, uart_dir_t communication_to_abort)
fsp_err_t	R_SCI_B_UART_BaudCalculate (uint32_t baudrate, bool bitrate_modulation, uint32_t baud_rate_error_x_1000, sci_b_baud_setting_t *const p_baud_setting)
fsp_err_t	R_SCI_B_UART_CallbackSet (uart_ctrl_t *const p_api_ctrl, void(*p_callback)(uart_callback_args_t *), void const *const p_context, uart_callback_args_t *const p_callback_memory)
fsp_err_t	R_SCI_B_UART_ReadStop (uart_ctrl_t *const p_api_ctrl, uint32_t *remaining_bytes)

Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the UART Interface.

Overview

Features

The SCI UART module supports the following features:

- Full-duplex UART communication
- Interrupt-driven data transmission and reception
- Invoking the user-callback function with an event code (RX/TX complete, TX data empty, RX char, error, etc)
- Baud-rate change at run-time
- Bit rate modulation and noise cancellation
- CTS/RTS hardware flow control (with an associated pin)
- Integration with the DTC transfer module
- Abort in-progress read/write operations
- FIFO support on supported channels

Configuration

Build Time Configurations for r_sci_b_uart

The following build time configurations are defined in fsp_cfg/r_sci_b_uart_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

FIFO Support	• Enable • Disable	Disable	Enable FIFO support for the SCI_UART module.
DTC Support	• Enable • Disable	Disable	Enable DTC support for the SCI_UART module.
Flow Control Support	• Enable • Disable	Disable	Enable RS232 and RS485 flow control support using a user provided pin.

Configurations for Connectivity > UART (r_sci_b_uart)

This module can be added to the Stacks tab via New Stack > Connectivity > UART (r_sci_b_uart). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_uart0	Module name.
General > Channel	Value must be a non-negative integer	0	Select the SCI channel.
General > Data Bits	• 8bits • 7bits • 9bits	8bits	Select the number of bits per word.
General > Parity	• None • Odd • Even	None	Select the parity mode.
General > Stop Bits	• 1bit • 2bits	1bit	Select the number of stop bits.
Baud > Baud Rate	Value must be an integer greater than 0	115200	Enter the desired baud rate. If the requested baud rate cannot be achieved, the settings with the smallest percent error are used. The theoretical calculated baud rate and percent error are printed in a comment in the generated sci_b_baud_setting_t structure.
Baud > Baud Rate Modulation	• Disabled • Enabled	Disabled	Enabling baud rate modulation reduces the percent error of the actual baud rate with respect to the requested baud rate. It

			does this by modulating the number of cycles per clock, so some bits are slightly longer than others.
Baud > Max Error (%)	Must be a valid non negative integer with a maximum configurable value of 100	5	Maximum percent error allowed during baud calculation. This is used by the algorithm to determine whether or not to consider using less accurate alternative register settings. NOTE: The baud calculation does not show an error in the tool if this percent error was not achieved. The calculated percent error is recorded in a comment in the generated sci_b_baud_setting_t structure.
Flow Control > CTS/RTS Selection	MCU Specific Options		Select either CTS or RTS function on the CTSn_RTSn pin of SCI channel n or select CTS function on CTSn pin and RTS function on CTSn_RTSn pin of SCI channel n (Available on selected MCUs and channels).
Flow Control > Software RTS Port	Refer to the RA Configuration tool for available options.	Disabled	Specify the flow control pin port for the MCU.
Flow Control > Software RTS Pin	Refer to the RA Configuration tool for available options.	Disabled	Specify the flow control pin for the MCU.
Extra > Clock Source	• Internal Clock • Internal Clock With Output on SCK • External Clock 8x baud rate • External Clock 16x baud rate	Internal Clock	Selection of the clock source to be used in the baud-rate clock generator. When internal clock is used the baud rate can be output on the SCK pin.
Extra > Start bit	• Falling Edge	Falling Edge	Start bit detected as

detection	• Low Level		falling edge or low level.
Extra > Noise Filter	• Enable • Disable	Disable	Enable the digital noise filter on RXDn pin. The digital noise filter block in SCI consists of two-stage flipflop circuits.
Extra > Receive FIFO Trigger Level	• One • Max	Max	Unused if the channel has no FIFO or if DTC is used for reception. Set to One to get a callback immediately when each byte is received. Set to Max to get a callback when FIFO is full or after 15 bit times with no data (fewer interrupts).
Interrupts > Callback	Name must be a valid C symbol	user_uart_callback	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).
Interrupts > Receive Interrupt Priority	MCU Specific Options		Select the receive interrupt priority.
Interrupts > Transmit Data Empty Interrupt Priority	MCU Specific Options		Select the transmit interrupt priority.
Interrupts > Transmit End Interrupt Priority	MCU Specific Options		Select the transmit end interrupt priority.
Interrupts > Error Interrupt Priority	MCU Specific Options		Select the error interrupt priority.

Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

MCU Group	Peripheral Clock
RA2A1	PCLKB
RA2E1	PCLKB
RA2E2	PCLKB
RA2L1	PCLKB
RA4E1	PCLKA

RA4M1	PCLKA
RA4M2	PCLKA
RA4M3	PCLKA
RA4W1	PCLKA
RA6E1	PCLKA
RA6M1	PCLKA
RA6M2	PCLKA
RA6M3	PCLKA
RA6M4	PCLKA
RA6M5	PCLKA
RA6T1	PCLKA
RA6T2	PCLKA

The clock source for the baud-rate clock generator can be selected from the internal clock, the external clock times 8 or the external clock times 16. The external clock is supplied to the SCK pin.

Pin Configuration

This module uses TXD and RXD to communicate to external devices. CTS or RTS can be controlled by the hardware. CTS or RTS or both (CTS and RTS) can be controlled by the hardware When the internal clock is the source for the baud-rate generator the SCK pin can be used to output a clock with the same frequency as the bit rate.

Usage Notes

- When configured for Hardware CTS and Software RTS the configured flow control pin will be used for RTS. The pin will be set high inside of the receive ISR while data is being read. It will be set low when all data is read.
- When configured for Hardware CTS and Hardware RTS the CSTn_RTSn pin will be used for RTS function and the CTSn pin will be used for CTS function on channel n.
- The driver will follow correct hardware flow control function when CTSn_RTSn pin is connected to CTSn pin when "Hardware CTS and Hardware RTS" flow control is selected. The data will still be transferred when CTSn_RTSn and CTSn are disconnected as the CTSn pin is internally pulled low on the hardware when CTSn pin is configured as a peripheral pin for SCI module. Do not configure CTSn pin if the hardware flow control is not desired.

Limitations

- Reception is still enabled after uart_api_t::communicationAbort API is called. Any characters received after abort and before the next call to read will arrive via the callback function with event UART_EVENT_RX_CHAR.

DTC Limitations

- DTC support is available for reception, but labeled as [Not recommended]. This is because the UART bytes are received asynchronously. Bytes can be received between calls to R_SCI_B_UART_Read(). The logic required to combine bytes received through

R_SCI_B_UART_Read() (UART_EVENT_RX_COMPLETE) and bytes received between calls (UART_EVENT_RX_CHAR) is complex. Reception length may also be unknown, and the driver will not issue an interrupt unless the entire DTC buffer is filled.
- Transfer size must be less than or equal to 64K bytes if DTC interface is used for transfer. uart_api_t::infoGet API can be used to get the max transfer size allowed.
- When using 9-bit reception with DTC, clear the upper 7 bits of data before processing the read data. The upper 7 bits contain status flags that are part of the register used to read data in 9-bit mode.

Examples

SCI UART Example

```
uint8_t g_dest[TRANSFER_LENGTH];
uint8_t g_src[TRANSFER_LENGTH];
uint8_t g_out_of_band_received[TRANSFER_LENGTH];
uint32_t g_transfer_complete = 0;
uint32_t g_receive_complete = 0;
uint32_t g_out_of_band_index = 0;
void r_sci_b_uart_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_SCI_B_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_SCI_B_UART_Read(&g_uart0_ctrl, g_dest, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
    err = R_SCI_B_UART_Write(&g_uart0_ctrl, g_src, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
    }
 while (!g_receive_complete)
    {
    }
```

```
}
void example_callback (uart_callback_args_t * p_args)
{
 /* Handle the UART event */
 switch (p_args->event)
    {
 /* Received a character */
 case UART_EVENT_RX_CHAR:
      {
 /* Only put the next character in the receive buffer if there is space for it */
 if (sizeof(g_out_of_band_received) > g_out_of_band_index)
      {
 /* Write either the next one or two bytes depending on the receive data size */
 if (UART_DATA_BITS_8 >= g_uart0_cfg.data_bits)
      {
                   g_out_of_band_received[g_out_of_band_index++] = (uint8_t)
p_args->data;
      }
 else
      {
                   uint16_t * p_dest = (uint16_t *)
&g_out_of_band_received[g_out_of_band_index];
                   *p_dest             = (uint16_t) p_args->data;
                   g_out_of_band_index += 2;
      }
      }
 break;
      }
 /* Receive complete */
 case UART_EVENT_RX_COMPLETE:
      {
            g_receive_complete = 1;
 break;
      }
```

```
 /* Transmit complete */
 case UART_EVENT_TX_COMPLETE:
      {
            g_transfer_complete = 1;
 break;
      }
 default:
      {
      }
    }
}
```

SCI UART Baud Set Example

```
#define SCI_B_UART_BAUDRATE_19200 (19200)
void r_sci_b_uart_baud_example (void)
{
 sci_b_baud_setting_t baud_setting;
    uint32_t             baud_rate                = SCI_B_UART_BAUDRATE_19200;
 bool                 enable_bitrate_modulation = false;
    uint32_t             error_rate_x_1000         = 5;
 fsp_err_t err = R_SCI_B_UART_BaudCalculate(baud_rate, enable_bitrate_modulation,
error_rate_x_1000, &baud_setting);
    assert(FSP_SUCCESS == err);
    err = R_SCI_B_UART_BaudSet(&g_uart0_ctrl, (void *) &baud_setting);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	sci_b_uart_instance_ctrl_t
struct	sci_b_baud_setting_t
struct	sci_b_uart_extended_cfg_t

Enumerations

enum	sci_b_clk_src_t

enum	sci_b_uart_flow_control_t
enum	sci_b_uart_rx_fifo_trigger_t
enum	sci_b_uart_start_bit_detect_t
enum	sci_b_uart_noise_cancellation_t

Data Structure Documentation

◆ sci_b_uart_instance_ctrl_t

struct sci_b_uart_instance_ctrl_t
UART instance control block.

◆ sci_b_baud_setting_t

struct sci_b_baud_setting_t
Register settings to acheive a desired baud rate and modulation duty.

◆ sci_b_uart_extended_cfg_t

struct sci_b_uart_extended_cfg_t

UART on SCI device Configuration

Data Fields		
sci_b_clk_src_t	clock	The source clock for the baud-rate generator. If internal optionally output baud rate on SCK.
sci_b_uart_start_bit_detect_t	rx_edge_start	Start reception on falling edge.
sci_b_uart_noise_cancellation_t	noise_cancel	Noise cancellation setting.
sci_b_baud_setting_t *	p_baud_setting	Register settings for a desired baud rate.
sci_b_uart_rx_fifo_trigger_t	rx_fifo_trigger	Receive FIFO trigger level, unused if channel has no FIFO or if DTC is used.
bsp_io_port_pin_t	flow_control_pin	UART Driver Enable pin.
sci_b_uart_flow_control_t	flow_control	CTS/RTS function of the SSn pin.

Enumeration Type Documentation

◆ sci_b_clk_src_t

enum sci_b_clk_src_t	
Enumeration for SCI clock source	
Enumerator	
SCI_B_UART_CLOCK_INT	Use internal clock for baud generation.
SCI_B_UART_CLOCK_INT_WITH_BAUDRATE_OUTPUT	Use internal clock for baud generation and output on SCK.
SCI_B_UART_CLOCK_EXT8X	Use external clock 8x baud rate.
SCI_B_UART_CLOCK_EXT16X	Use external clock 16x baud rate.

◆ sci_b_uart_flow_control_t

enum sci_b_uart_flow_control_t	
UART flow control mode definition	
Enumerator	
SCI_B_UART_FLOW_CONTROL_RTS	Use CTSn_RTSn pin for RTS.
SCI_B_UART_FLOW_CONTROL_CTS	Use CTSn_RTSn pin for CTS.
SCI_B_UART_FLOW_CONTROL_HARDWARE_CTSRTS	Use CTSn pin for CTS, CTSn_RTSn pin for RTS.
SCI_B_UART_FLOW_CONTROL_CTSRTS	Use SCI pin for CTS, external pin for RTS.

◆ sci_b_uart_rx_fifo_trigger_t

enum sci_b_uart_rx_fifo_trigger_t	
Receive FIFO trigger configuration.	
Enumerator	
SCI_B_UART_RX_FIFO_TRIGGER_1	Callback after each byte is received without buffering.
SCI_B_UART_RX_FIFO_TRIGGER_MAX	Callback when FIFO is full or after 15 bit times with no data (fewer interrupts)

◆ sci_b_uart_start_bit_detect_t

enum sci_b_uart_start_bit_detect_t

Asynchronous Start Bit Edge Detection configuration.

Enumerator	
SCI_B_UART_START_BIT_LOW_LEVEL	Detect low level on RXDn pin as start bit.
SCI_B_UART_START_BIT_FALLING_EDGE	Detect falling level on RXDn pin as start bit.

◆ sci_b_uart_noise_cancellation_t

enum sci_b_uart_noise_cancellation_t

Noise cancellation configuration.

Enumerator	
SCI_B_UART_NOISE_CANCELLATION_DISABLE	Disable noise cancellation.
SCI_B_UART_NOISE_CANCELLATION_ENABLE	Enable noise cancellation.

Function Documentation

◆ R_SCI_B_UART_Open()

fsp_err_t R_SCI_B_UART_Open (uart_ctrl_t *const *p_api_ctrl*, uart_cfg_t const *const *p_cfg*)

Configures the UART driver based on the input configurations. If reception is enabled at compile time, reception is enabled at the end of this function. Implements uart_api_t::open

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ASSERTION	Pointer to UART control block or configuration structure is NULL.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The requested channel does not exist on this MCU.
FSP_ERR_INVALID_ARGUMENT	Flow control is enabled but flow control pin is not defined.
FSP_ERR_ALREADY_OPEN	Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open

◆ R_SCI_B_UART_Read()

fsp_err_t R_SCI_B_UART_Read (uart_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Receives user specified number of bytes into destination buffer pointer. Implements uart_api_t::read

Return values

FSP_SUCCESS	Data reception successfully ends.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used
FSP_ERR_INVALID_ARGUMENT	Destination address or data size is not valid for 9-bit mode.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_IN_USE	A previous read operation is still in progress.
FSP_ERR_UNSUPPORTED	SCI_B_UART_CFG_RX_ENABLE is set to 0

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- transfer_api_t::reset

Note

If 9-bit data length is specified at R_SCI_B_UART_Open call, p_dest must be aligned 16-bit boundary.

◆ R_SCI_B_UART_Write()

fsp_err_t R_SCI_B_UART_Write (uart_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*)

Transmits user specified number of bytes from the source buffer pointer. Implements uart_api_t::write

Return values

FSP_SUCCESS	Data transmission finished successfully.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used
FSP_ERR_INVALID_ARGUMENT	Source address or data size is not valid for 9-bit mode.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_IN_USE	A UART transmission is in progress
FSP_ERR_UNSUPPORTED	SCI_B_UART_CFG_TX_ENABLE is set to 0

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

Note

If 9-bit data length is specified at R_SCI_B_UART_Open call, p_src must be aligned on a 16-bit boundary.

◆ R_SCI_B_UART_BaudSet()

fsp_err_t R_SCI_B_UART_BaudSet (uart_ctrl_t *const *p_api_ctrl*, void const *const *p_baud_setting*)

Updates the baud rate using the clock selected in Open. p_baud_setting is a pointer to a sci_b_baud_setting_t structure. Implements uart_api_t::baudSet

Warning

This terminates any in-progress transmission.

Return values

FSP_SUCCESS	Baud rate was successfully changed.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL or the UART is not configured to use the internal clock.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_SCI_B_UART_InfoGet()

fsp_err_t R_SCI_B_UART_InfoGet (uart_ctrl_t *const *p_api_ctrl*, uart_info_t *const *p_info*)

Provides the driver information, including the maximum number of bytes that can be received or transmitted at a time. Implements uart_api_t::infoGet

Return values

FSP_SUCCESS	Information stored in provided p_info.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_SCI_B_UART_Close()

fsp_err_t R_SCI_B_UART_Close (uart_ctrl_t *const *p_api_ctrl*)

Aborts any in progress transfers. Disables interrupts, receiver, and transmitter. Closes lower level transfer drivers if used. Removes power. Implements uart_api_t::close

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_SCI_B_UART_Abort()

fsp_err_t R_SCI_B_UART_Abort (uart_ctrl_t *const *p_api_ctrl*, uart_dir_t *communication_to_abort*)

Provides API to abort ongoing transfer. Transmission is aborted after the current character is transmitted. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::communicationAbort

Return values

FSP_SUCCESS	UART transaction aborted successfully.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_UNSUPPORTED	The requested Abort direction is unsupported.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

◆ R_SCI_B_UART_BaudCalculate()

fsp_err_t R_SCI_B_UART_BaudCalculate (uint32_t *baudrate*, bool *bitrate_modulation*, uint32_t *baud_rate_error_x_1000*, sci_b_baud_setting_t *const *p_baud_setting*)

Calculates baud rate register settings. Evaluates and determines the best possible settings set to the baud rate related registers.

Parameters

[in]	baudrate	Baud rate [bps]. For example, 19200, 57600, 115200, etc.
[in]	bitrate_modulation	Enable bitrate modulation
[in]	baud_rate_error_x_1000	<baud_rate_percent_error> x 1000 required for module to function. Absolute max baud_rate_error is 15000 (15%).
[out]	p_baud_setting	Baud setting information stored here if successful

Return values

FSP_SUCCESS	Baud rate is set successfully
FSP_ERR_ASSERTION	Null pointer
FSP_ERR_INVALID_ARGUMENT	Baud rate is '0', source clock frequency could not be read, or error in calculated baud rate is larger than 10%.

◆ R_SCI_B_UART_CallbackSet()

fsp_err_t R_SCI_B_UART_CallbackSet (uart_ctrl_t *const *p_api_ctrl*, void(*)(uart_callback_args_t *) *p_callback*, void const *const *p_context*, uart_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements uart_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_SCI_B_UART_ReadStop()

fsp_err_t R_SCI_B_UART_ReadStop (uart_ctrl_t *const *p_api_ctrl*, uint32_t * *remaining_bytes*)

Provides API to abort ongoing read. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::readStop

Return values

FSP_SUCCESS	UART transaction aborted successfully.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_UNSUPPORTED	The requested Abort direction is unsupported.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

4.2.53 Serial Communications Interface (SCI) I2C (r_sci_i2c)

Modules

Functions

fsp_err_t	R_SCI_I2C_Open (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_cfg_t const *const p_cfg)
fsp_err_t	R_SCI_I2C_Close (i2c_master_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_I2C_Read (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)
fsp_err_t	R_SCI_I2C_Write (i2c_master_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)
fsp_err_t	R_SCI_I2C_Abort (i2c_master_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_I2C_SlaveAddressSet (i2c_master_ctrl_t *const p_api_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode)
fsp_err_t	R_SCI_I2C_CallbackSet (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void const *const p_context, i2c_master_callback_args_t *const p_callback_memory)
fsp_err_t	R_SCI_I2C_StatusGet (i2c_master_ctrl_t *const p_api_ctrl,

i2c_master_status_t *p_status)

Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the I2C Master Interface.

Overview

The Simple I2C master on SCI HAL module supports transactions with an I2C Slave device. Callbacks must be provided which would be invoked when a transmission or receive has been completed. The callback arguments will contain information about the transaction status, bytes transferred and a pointer to the user defined context.

Features

- Supports multiple transmission rates
  - Standard Mode Support with up to 100 kHz transaction rate.
  - Fast Mode Support with up to 400 kHz transaction rate.
- SDA Delay in nanoseconds can be specified as a part of the configuration.
- I2C Master Read from a slave device.
- I2C Master Write to a slave device.
- Abort any in-progress transactions.
- Set the address of the slave device.
- Non-blocking behavior is achieved by the use of callbacks.
- Additional build-time features
  - Optional (build time) DTC support for read and write respectively.
  - Optional (build time) support for 10-bit slave addressing.

Configuration

Build Time Configurations for r_sci_i2c

The following build time configurations are defined in fsp_cfg/r_sci_i2c_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DTC on Transmission and Reception	• Enabled • Disabled	Disabled	If enabled, DTC instances will be included in the build for both transmission and reception.
10-bit slave addressing	• Enabled • Disabled	Disabled	If enabled, the driver will support 10-bit slave addressing mode along with the default 7-bit slave addressing mode.

Configurations for Connectivity > I2C Master (r_sci_i2c)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Master (r_sci_i2c). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_i2c0	Module name.
Channel	Value must be an integer between 0 and 9	0	Select the SCI channel.
Slave Address	Value must be a hex value	0x00	Specify the slave address.
Address Mode	• 7-Bit • 10-Bit	7-Bit	Select the address mode.
Rate	• Standard • Fast-mode	Standard	Select the I2C data rate. If the requested transfer rate cannot be achieved, the settings with the largest possible transfer rate that is less than or equal to the requested transfer rate are used. The theoretical calculated transfer rate and SDA delay are printed in a comment in the generated sci_i2c_extended_cfg_t structure.
SDA Output Delay (nano seconds)	Must be a valid non-negative integer with maximum configurable value of 300	300	Specify the SDA output delay in nanoseconds.
Noise filter setting	• Use clock signal divided by 1 with noise filter • Use clock signal divided by 2 with noise filter • Use clock signal divided by 4 with noise filter • Use clock signal divided by 8 with noise filter	Use clock signal divided by 1 with noise filter	Select the sampling clock for the digital noise filter

Bit Rate Modulation	• Enable • Disable	Enable	Enabling bitrate modulation reduces the percent error of the actual bitrate with respect to the requested baud rate. It does this by modulating the number of cycles per clock output pulse, so the clock is no longer a square wave.
Callback	Name must be a valid C symbol	sci_i2c_master_callback	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).
Interrupt Priority Level	MCU Specific Options		Select the interrupt priority level. This is set for TXI, RXI (if used), TEI interrupts.
RX Interrupt Priority Level [Only used when DTC is enabled]	MCU Specific Options		Select the interrupt priority level. This is set for RXI only when DTC is enabled.

Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

MCU Group	Peripheral Clock
RA2A1	PCLKB
RA2E1	PCLKB
RA2E2	PCLKB
RA2L1	PCLKB
RA4E1	PCLKA
RA4M1	PCLKA
RA4M2	PCLKA
RA4M3	PCLKA
RA4W1	PCLKA
RA6E1	PCLKA
RA6M1	PCLKA

RA6M2	PCLKA
RA6M3	PCLKA
RA6M4	PCLKA
RA6M5	PCLKA
RA6T1	PCLKA
RA6T2	PCLKA

The actual I2C transfer rate will be calculated and set by the tooling depending on the selected transfer rate and the SDA delay. If the PCLK is configured in such a manner that the selected internal rate cannot be achieved, an error will be returned.

Pin Configuration

The SCI I2C peripheral module uses pins on the MCU to communicate to external devices. I/O pins must be selected and configured as required by the external device. An I2C channel would consist of two pins - SDA and SCL for data/address and clock respectively.

Usage Notes

Interrupt Configuration

- Receive buffer full (RXI), transmit buffer empty (TXI) and transmit end (TEI) interrupts for the selected channel used must be enabled in the properties of the selected device.
- Set equal priority levels for all the interrupts mentioned above. Setting the interrupts to different priority levels could result in improper operation.

SCI I2C Master Rate Calculation

- The RA Configuration editor calculates the internal baud-rate setting based on the configured transfer rate and SDA Delay. The closest possible baud-rate that can be achieved (less than or equal to the requested rate) at the current PCLK settings is calculated and used.
- If a valid clock rate could not be calculated, an error is returned by the tool.

Enabling DTC with the SCI I2C

- DTC transfer support is configurable and is disabled from the build by default. SCI I2C driver provides two DTC instances for transmission and reception respectively.
- DTC is helpful for minimizing interrupts during large transactions. Many I2C applications have shorter transactions. These applications will likely not see any improvement with DTC. I2C often runs at a much slower speed than the CPU core clock. Some applications with longer transactions may prefer servicing the interrupts at the I2C bitrate to the overhead of bringing in the DTC driver.
- For further details on DTC please refer Data Transfer Controller (r_dtc)

Multiple Devices on the Bus

- A single SCI I2C instance can be used to communicate with multiple slave devices on the same channel by using the SlaveAddressSet API.

Restart

- SCI I2C master can hold the the bus after an I2C transaction by issuing Restart. This will mimic a stop followed by start condition.

Examples

Basic Example

This is a basic example of minimal use of the r_sci_i2c in an application. This example shows how this driver can be used for basic read and write operations.

```
void basic_example (void)
{
 fsp_err_t err;
    uint32_t i;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
 /* Initialize the I2C module */
    err = R_SCI_I2C_Open(&g_i2c_device_ctrl_1, &g_i2c_device_cfg_1);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Write some data to the transmit buffer */
 for (i = 0; i < I2C_BUFFER_SIZE_BYTES; i++)
    {
       g_i2c_tx_buffer[i] = (uint8_t) i;
    }
 /* Send data to I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SCI_I2C_Write(&g_i2c_device_ctrl_1, &g_i2c_tx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
```

```
        __BKPT(0);
    }
 /* Read data back from the I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_I2C_Read(&g_i2c_device_ctrl_1, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 /* Since there is nothing else to do, block until Callback triggers*/
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
        timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
        __BKPT(0);
    }
 /* Verify the read data */
 if (0U != memcmp(g_i2c_tx_buffer, g_i2c_rx_buffer, I2C_BUFFER_SIZE_BYTES))
    {
        __BKPT(0);
    }
}
```

Multiple Slave devices on the same channel (bus)

This example demonstrates how a single SCI I2C driver can be used to communicate with different slave devices which are on the same channel.

```
void single_channel_multi_slave (void)
{
 fsp_err_t err;
    uint32_t timeout_ms = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_I2C_Open(&g_i2c_device_ctrl_2, &g_i2c_device_cfg_2);
```

```
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Clear the recieve buffer */
    memset(g_i2c_rx_buffer, '0', I2C_BUFFER_SIZE_BYTES);
 /* Read data from I2C slave */
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    err = R_SCI_I2C_Read(&g_i2c_device_ctrl_2, &g_i2c_rx_buffer[0],
I2C_BUFFER_SIZE_BYTES, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_RX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
    {
       __BKPT(0);
    }
 /* Send data to I2C slave on the same channel */
    err = R_SCI_I2C_SlaveAddressSet(&g_i2c_device_ctrl_2, I2C_SLAVE_DISPLAY_ADAPTER,
I2C_MASTER_ADDR_MODE_7BIT);
    assert(FSP_SUCCESS == err);
    g_i2c_tx_buffer[0]   = (uint8_t) I2C_EXAMPLE_DATA_1;
    g_i2c_tx_buffer[1]   = (uint8_t) I2C_EXAMPLE_DATA_2;
    g_i2c_callback_event = I2C_MASTER_EVENT_ABORTED;
    timeout_ms          = I2C_TRANSACTION_BUSY_DELAY;
    err = R_SCI_I2C_Write(&g_i2c_device_ctrl_2, &g_i2c_tx_buffer[0], 2U, false);
    assert(FSP_SUCCESS == err);
 while ((I2C_MASTER_EVENT_TX_COMPLETE != g_i2c_callback_event) && timeout_ms)
    {
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
       timeout_ms--;;
    }
 if (I2C_MASTER_EVENT_ABORTED == g_i2c_callback_event)
```

```
    {
        __BKPT(0);
    }
}
```

Data Structures

struct	sci_i2c_clock_settings_t
struct	sci_i2c_instance_ctrl_t
struct	sci_i2c_extended_cfg_t

Data Structure Documentation

◆ sci_i2c_clock_settings_t

struct sci_i2c_clock_settings_t

I2C clock settings

Data Fields

bool	bitrate_modulation	Bit-rate Modulation Function enable or disable.
uint8_t	brr_value	Bit rate register settings.
uint8_t	clk_divisor_value	Clock Select settings.
uint8_t	mddr_value	Modulation Duty Register settings.
uint8_t	cycles_value	SDA Delay Output Cycles Select.
uint8_t	snfr_value	Noise Filter Setting Register value.

◆ sci_i2c_instance_ctrl_t

struct sci_i2c_instance_ctrl_t

I2C control structure. DO NOT INITIALIZE.

◆ sci_i2c_extended_cfg_t

struct sci_i2c_extended_cfg_t

SCI I2C extended configuration

Data Fields

sci_i2c_clock_settings_t	clock_settings	I2C Clock settings.

Function Documentation

◆ R_SCI_I2C_Open()

fsp_err_t R_SCI_I2C_Open (i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_cfg_t const *const *p_cfg*)

Opens the I2C device.

Return values

FSP_SUCCESS	Requested clock rate was set exactly.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_ASSERTION	Parameter check failure due to one or more reasons below: 1. p_api_ctrl or p_cfg is NULL. 2. extended parameter is NULL. 3. Callback parameter is NULL. 4. Clock rate requested is greater than 400KHz 5. Invalid IRQ number assigned

◆ R_SCI_I2C_Close()

fsp_err_t R_SCI_I2C_Close (i2c_master_ctrl_t *const *p_api_ctrl*)

Closes the I2C device. Power down I2C peripheral.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Return values

FSP_SUCCESS	Device closed without issue.
FSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Device was not even opened.

◆ R_SCI_I2C_Read()

fsp_err_t R_SCI_I2C_Read (i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*, bool const *restart*)

Performs a read from the I2C device. The caller will be notified when the operation has completed (successfully) by an I2C_MASTER_EVENT_RX_COMPLETE in the callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	The parameter p_ctrl, p_dest is NULL, bytes is 0.
FSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer.
FSP_ERR_NOT_OPEN	Device was not even opened.

◆ R_SCI_I2C_Write()

fsp_err_t R_SCI_I2C_Write (i2c_master_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*, bool const *restart*)

Performs a write to the I2C device.

This function will fail if there is already an in-progress I2C transfer on the associated channel. Otherwise, the I2C write operation will begin. When no callback is provided by the user, this function performs a blocking write. Otherwise, the write operation is non-blocking and the caller will be notified when the operation has finished by an I2C_EVENT_TX_COMPLETE in the callback.

Return values

FSP_SUCCESS	Function executed without issue.
FSP_ERR_ASSERTION	p_ctrl, p_src is NULL.
FSP_ERR_INVALID_SIZE	Provided number of bytes more than uint16_t size (65535) while DTC is used for data transfer.
FSP_ERR_NOT_OPEN	Device was not even opened.

◆ R_SCI_I2C_Abort()

fsp_err_t R_SCI_I2C_Abort (i2c_master_ctrl_t *const *p_api_ctrl*)

Aborts any in-progress transfer and forces the I2C peripheral into a ready state.

This function will safely terminate any in-progress I2C transfer with the device. If a transfer is aborted, the user will be notified via callback with an abort event. Since the callback is optional, this function will also return a specific error code in this situation.

Return values

FSP_SUCCESS	Transaction was aborted without issue.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Device was not even opened.

◆ R_SCI_I2C_SlaveAddressSet()

fsp_err_t R_SCI_I2C_SlaveAddressSet (i2c_master_ctrl_t *const *p_api_ctrl*, uint32_t const *slave*, i2c_master_addr_mode_t const *addr_mode*)

Sets address and addressing mode of the slave device.

This function is used to set the device address and addressing mode of the slave without reconfiguring the entire bus.

Return values

FSP_SUCCESS	Address of the slave is set correctly.
FSP_ERR_ASSERTION	p_ctrl or address is NULL.
FSP_ERR_NOT_OPEN	Device was not even opened.
FSP_ERR_IN_USE	An I2C Transaction is in progress.

◆ R_SCI_I2C_CallbackSet()

fsp_err_t R_SCI_I2C_CallbackSet (i2c_master_ctrl_t *const *p_api_ctrl*, void(*)(i2c_master_callback_args_t *) *p_callback*, void const *const *p_context*, i2c_master_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements i2c_master_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_SCI_I2C_StatusGet()

fsp_err_t R_SCI_I2C_StatusGet (i2c_master_ctrl_t *const *p_api_ctrl*, i2c_master_status_t * *p_status*)

Provides driver status.

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.

4.2.54 Serial Communications Interface (SCI) SPI (r_sci_spi)

Modules

Functions

fsp_err_t	R_SCI_SPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg)
fsp_err_t	R_SCI_SPI_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SCI_SPI_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SCI_SPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src,

	void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SCI_SPI_CallbackSet (spi_ctrl_t *const p_api_ctrl, void(*p_callback)(spi_callback_args_t *), void const *const p_context, spi_callback_args_t *const p_callback_memory)
fsp_err_t	R_SCI_SPI_Close (spi_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_SPI_CalculateBitrate (uint32_t bitrate, sci_spi_div_setting_t *sclk_div, bool use_mddr)

Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the SPI Interface.

Overview

Features

- Standard SPI Modes
  - Master or Slave Mode
  - Clock Polarity (CPOL)
    - CPOL=0 SCLK is low when idle
    - CPOL=1 SCLK is high when idle
  - Clock Phase (CPHA)
    - CPHA=0 Select data sampling on leading edge, data change on trailing edge
    - CPHA=1 Select data change on leading edge, data sampling on trailing edge
  - MSB/LSB first
- Configurable bit rate
- DTC Support
- Callback Events
  - Transfer Complete
  - RX Overflow Error (The SCI shift register is copied to the data register before previous data was read)

Configuration

Build Time Configurations for r_sci_spi

The following build time configurations are defined in fsp_cfg/r_sci_spi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DTC Support	• Enabled	Enabled	If support for

	• Disabled		transfering data using the DTC will be compiled in.

Configurations for Connectivity > SPI (r_sci_spi)

This module can be added to the Stacks tab via New Stack > Connectivity > SPI (r_sci_spi). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_spi0	Module name.
Channel	Value must be a non-negative integer	0	Select the SCI channel.
Operating Mode	• Master • Slave	Master	Select the SPI operating mode.
Clock Phase	• Data sampling on odd edge, data variation on even edge • Data sampling on even edge, data variation on odd edge	Data sampling on odd edge, data variation on even edge	Select the clock edge to sample data.
Clock Polarity	• Low when idle • High when idle	Low when idle	Select clock level when idle.
Mode Fault Error	• Enable • Disable	Disable	Detect master/slave mode conflicts.
Bit Order	• MSB First • LSB First	MSB First	Select the data bit order.
Callback	Name must be a valid C symbol	sci_spi_callback	A user callback function that is called from the sci spi interrupts when a transfer is completed or an error has occurred.
Receive Interrupt Priority	MCU Specific Options		Select the receive interrupt priority.
Transmit Interrupt Priority	MCU Specific Options		Select the transmit interrupt priority.
Transmit End Interrupt Priority	MCU Specific Options		Select the transmit end interrupt priority.
Error Interrupt Priority	MCU Specific Options		Select the error interrupt priority.

Bitrate	Value must be an integer greater than 0	8000000	Enter the desired bitrate. If the requested bitrate cannot be achieved, the settings with the largest possible value that is less than or equal to the requested bitrate is used. The theoretical bitrate is printed in a comment in the generated sci_spi_extended_cfg_t structure.
Bitrate Modulation	• Disabled • Enabled	Disabled	Enabling bitrate modulation reduces the percent error of the actual bitrate with respect to the requested baud rate. It does this by modulating the number of cycles per clock output pulse, so the clock is no longer a square wave.

Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

MCU Group	Peripheral Clock
RA2A1	PCLKB
RA2E1	PCLKB
RA2E2	PCLKB
RA2L1	PCLKB
RA4E1	PCLKA
RA4M1	PCLKA
RA4M2	PCLKA
RA4M3	PCLKA
RA4W1	PCLKA
RA6E1	PCLKA
RA6M1	PCLKA
RA6M2	PCLKA

RA6M3	PCLKA
RA6M4	PCLKA
RA6M5	PCLKA
RA6T1	PCLKA
RA6T2	PCLKA

Pin Configuration

This module uses SCIn_MOSI, SCIn_MISO, SCIn_SPCK, and SCIn_SS pins to communicate with on board devices.

Note

At high bit rates, it might be necessary to configure the pins with IOPORT_CFG_DRIVE_HIGH.

Usage Notes

Transfer Complete Event

The transfer complete event is triggered when all of the data has been transfered. In slave mode if the SS pin is de-asserted then no transfer complete event is generated until the SS pin is asserted and the remaining data is transferred.

Performance

At high bit rates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. In slave mode this could result in RX Overflow errors.

In order to improve performance at high bit rates, it is recommended that the instance be configured to service transfers using the DTC.

Transmit From RXI Interrupt

After every byte, the SCI SPI peripheral generates a transmit buffer empty interrupt and a receive buffer full interrupt. Whenever possible, the SCI_SPI module handles both interrupts in the receive buffer full interrupt. This improves performance when the DTC is not being used.

Slave Select Pin

- In master mode the slave select pin must be driven in software.
- In slave mode the hardware handles the slave select pin and will only transfer data when the SS pin is low.

Bit Rate Modulation

Depending on the peripheral clock frequency, the desired bit rate may not be achievable. With bit rate modulation, the device can remove a configurable number of input clock pulses to the internal bit rate counter in order to create the desired bit rate. This has the effect of changing the period of individual bits in order to achieve the desired average bit rate. For more information see section 34.9 Bit Rate Modulation Function in the RA6M3 manual.

Examples

Basic Example

This is a basic example of minimal use of the SCI_SPI in an application.

```
static volatile bool g_transfer_complete = false;
static void r_sci_spi_callback (spi_callback_args_t * p_args)
{
 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = true;
    }
}
void sci_spi_basic_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 /* Configure Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Configure Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SCI_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SCI_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
```

```
    {
        ;
    }
 /* De-assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Wait for minimum time required between transfers. */
 R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
 /* Assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err = R_SCI_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
        ;
    }
 /* De-assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
}
```

Data Structures

struct	sci_spi_div_setting_t

Data Structure Documentation

◆ sci_spi_div_setting_t

struct sci_spi_div_setting_t

Settings for adjusting the SPI CLK.

Data Fields		
uint8_t	brr	
uint8_t	cks: 2	
uint8_t	mddr	Set to 0 to disable MDDR.

Function Documentation

◆ R_SCI_SPI_Open()

fsp_err_t R_SCI_SPI_Open (spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg*)

Initialize a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enables the clock for the SCI channel.
- Initializes the associated registers with default value and the user-configurable options.
- Provides the channel handle for use with other API functions.

Parameters

p_api_ctrl	Pointer to the control structure.
p_cfg	Pointer to a configuration structure.

Return values

FSP_SUCCESS	Channel initialized successfully.
FSP_ERR_ASSERTION	An input parameter is invalid or NULL.
FSP_ERR_ALREADY_OPEN	The instance has already been opened.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel number is invalid.

◆ R_SCI_SPI_Read()

fsp_err_t R_SCI_SPI_Read (spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

Receive data from an SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission by writing data to the TXD register.
- Receive data from receive buffer full interrupt occurs and copy data to the buffer of destination.
- Complete data reception via receive buffer full interrupt and transmitting dummy data.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Parameters

	p_api_ctrl	Pointer to the control structure.
	p_dest	Pointer to the destination buffer.
[in]	length	The number of bytes to transfer.
[in]	bit_width	Invalid for SCI_SPI (Set to SPI_BIT_WIDTH_8_BITS).

Return values

FSP_SUCCESS	Read operation successfully completed.
FSP_ERR_ASSERTION	One of the following invalid parameters passed: • Pointer p_api_ctrl is NULL • Bit width is not 8 bits • Length is equal to 0 • Pointer to destination is NULL
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_UNSUPPORTED	The given bit_width is not supported.
FSP_ERR_IN_USE	A transfer is already in progress.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- ◦ transfer_api_t::reconfigure

◆ R_SCI_SPI_Write()

fsp_err_t R_SCI_SPI_Write (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

Transmit data to a SPI device. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable interrupts.
- Start data transmission with data via transmit buffer empty interrupt.
- Copy data from source buffer to the SPI data register for transmission.
- Complete data transmission via transmit buffer empty interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Parameters

	p_api_ctrl	Pointer to the control structure.
	p_src	Pointer to the source buffer.
[in]	length	The number of bytes to transfer.
[in]	bit_width	Invalid for SCI_SPI (Set to SPI_BIT_WIDTH_8_BITS).

Return values

FSP_SUCCESS	Write operation successfully completed.
FSP_ERR_ASSERTION	One of the following invalid parameters passed: • Pointer p_api_ctrl is NULL • Pointer to source is NULL • Length is equal to 0 • Bit width is not equal to 8 bits
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_UNSUPPORTED	The given bit_width is not supported.
FSP_ERR_IN_USE	A transfer is already in progress.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reconfigure

◆ R_SCI_SPI_WriteRead()

fsp_err_t R_SCI_SPI_WriteRead (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

Simultaneously transmit data to SPI device while receiving data from SPI device (full duplex). Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Enable transmitter.
- Enable receiver.
- Enable interrupts.
- Start data transmission using transmit buffer empty interrupt (or by writing to the TDR register).
- Copy data from source buffer to the SPI data register for transmission.
- Receive data from receive buffer full interrupt and copy data to the destination buffer.
- Complete data transmission and reception via transmit end interrupt.
- Disable transmitter.
- Disable receiver.
- Disable interrupts.

Parameters

	p_api_ctrl	Pointer to the control structure.
	p_src	Pointer to the source buffer.
	p_dest	Pointer to the destination buffer.
[in]	length	The number of bytes to transfer.
[in]	bit_width	Invalid for SCI_SPI (Set to SPI_BIT_WIDTH_8_BITS).

Return values

FSP_SUCCESS	Write operation successfully completed.
FSP_ERR_ASSERTION	One of the following invalid parameters passed: • Pointer p_api_ctrl is NULL • Pointer to source is NULL • Pointer to destination is NULL • Length is equal to 0 • Bit width is not equal to 8 bits
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_UNSUPPORTED	The given bit_width is not supported.
FSP_ERR_IN_USE	A transfer is already in progress.

Returns
See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- transfer_api_t::reconfigure

◆ R_SCI_SPI_CallbackSet()

fsp_err_t R_SCI_SPI_CallbackSet (spi_ctrl_t *const *p_api_ctrl*, void(*)(spi_callback_args_t *) *p_callback*, void const *const *p_context*, spi_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements spi_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_SCI_SPI_Close()

fsp_err_t R_SCI_SPI_Close (spi_ctrl_t *const *p_api_ctrl*)

Disable the SCI channel and set the instance as not open. Implements spi_api_t::close.

Parameters

p_api_ctrl	Pointer to an opened instance.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	The parameter p_api_ctrl is NULL.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

◆ R_SCI_SPI_CalculateBitrate()

fsp_err_t R_SCI_SPI_CalculateBitrate (uint32_t *bitrate*, sci_spi_div_setting_t * *sclk_div*, bool *use_mddr*)

Calculate the register settings required to achieve the desired bitrate.

Parameters

[in]	bitrate	bitrate [bps]. For example, 250,000; 500,00; 2,500,000 (max), etc.
	sclk_div	Pointer to sci_spi_div_setting_t used to configure baudrate settings.
[in]	use_mddr	Calculate the divider settings for use with MDDR.

Return values

FSP_SUCCESS	Baud rate is set successfully.
FSP_ERR_ASSERTION	Baud rate is not achievable.

Note

The application must pause for 1 bit time after the BRR register is loaded before transmitting/receiving to allow time for the clock to settle.

4.2.55 Serial Communications Interface (SCI) UART (r_sci_uart)

Modules

Functions

fsp_err_t	R_SCI_UART_Open (uart_ctrl_t *const p_api_ctrl, uart_cfg_t const *const p_cfg)
fsp_err_t	R_SCI_UART_Read (uart_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t	R_SCI_UART_Write (uart_ctrl_t *const p_api_ctrl, uint8_t const *const p_src, uint32_t const bytes)
fsp_err_t	R_SCI_UART_BaudSet (uart_ctrl_t *const p_api_ctrl, void const *const p_baud_setting)
fsp_err_t	R_SCI_UART_InfoGet (uart_ctrl_t *const p_api_ctrl, uart_info_t *const p_info)

fsp_err_t	R_SCI_UART_Close (uart_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SCI_UART_Abort (uart_ctrl_t *const p_api_ctrl, uart_dir_t communication_to_abort)
fsp_err_t	R_SCI_UART_BaudCalculate (uint32_t baudrate, bool bitrate_modulation, uint32_t baud_rate_error_x_1000, baud_setting_t *const p_baud_setting)
fsp_err_t	R_SCI_UART_CallbackSet (uart_ctrl_t *const p_api_ctrl, void(*p_callback)(uart_callback_args_t *), void const *const p_context, uart_callback_args_t *const p_callback_memory)
fsp_err_t	R_SCI_UART_ReadStop (uart_ctrl_t *const p_api_ctrl, uint32_t *remaining_bytes)

Detailed Description

Driver for the SCI peripheral on RA MCUs. This module implements the UART Interface.

Overview

Features

The SCI UART module supports the following features:

- Full-duplex UART communication
- Interrupt-driven data transmission and reception
- Invoking the user-callback function with an event code (RX/TX complete, TX data empty, RX char, error, etc)
- Baud-rate change at run-time
- Bit rate modulation and noise cancellation
- CTS/RTS hardware flow control (with an associated pin)
- Integration with the DTC transfer module
- Abort in-progress read/write operations
- FIFO support on supported channels

Configuration

Build Time Configurations for r_sci_uart

The following build time configurations are defined in fsp_cfg/r_sci_uart_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
FIFO Support	• Enable • Disable	Disable	Enable FIFO support for the SCI_UART module.

DTC Support	• Enable • Disable	Disable	Enable DTC support for the SCI_UART module.
Flow Control Support	• Enable • Disable	Disable	Enable RS232 and RS485 flow control support using a user provided pin.

Configurations for Connectivity > UART (r_sci_uart)

This module can be added to the Stacks tab via New Stack > Connectivity > UART (r_sci_uart). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_uart0	Module name.
General > Channel	Value must be a non-negative integer	0	Select the SCI channel.
General > Data Bits	• 8bits • 7bits • 9bits	8bits	Select the number of bits per word.
General > Parity	• None • Odd • Even	None	Select the parity mode.
General > Stop Bits	• 1bit • 2bits	1bit	Select the number of stop bits.
Baud > Baud Rate	Value must be an integer greater than 0	115200	Enter the desired baud rate. If the requested baud rate cannot be achieved, the settings with the smallest percent error are used. The theoretical calculated baud rate and percent error are printed in a comment in the generated baud_setting_t structure.
Baud > Baud Rate Modulation	• Disabled • Enabled	Disabled	Enabling baud rate modulation reduces the percent error of the actual baud rate with respect to the requested baud rate. It does this by modulating the number

			of cycles per clock, so some bits are slightly longer than others.
Baud > Max Error (%)	Must be a valid non negative integer with a maximum configurable value of 100	5	Maximum percent error allowed during baud calculation. This is used by the algorithm to determine whether or not to consider using less accurate alternative register settings. NOTE: The baud calculation does not show an error in the tool if this percent error was not achieved. The calculated percent error is recorded in a comment in the generated baud_setting_t structure.
Flow Control > CTS/RTS Selection	MCU Specific Options		Select either CTS or RTS function on the CTSn_RTSn pin of SCI channel n or select CTS function on CTSn pin and RTS function on CTSn_RTSn pin of SCI channel n (Available on selected MCUs and channels).
Flow Control > Software RTS Port	Refer to the RA Configuration tool for available options.	Disabled	Specify the flow control pin port for the MCU.
Flow Control > Software RTS Pin	Refer to the RA Configuration tool for available options.	Disabled	Specify the flow control pin for the MCU.
Extra > Clock Source	• Internal Clock • Internal Clock With Output on SCK • External Clock 8x baud rate • External Clock 16x baud rate	Internal Clock	Selection of the clock source to be used in the baud-rate clock generator. When internal clock is used the baud rate can be output on the SCK pin.
Extra > Start bit detection	• Falling Edge • Low Level	Falling Edge	Start bit detected as falling edge or low level.

Extra > Noise Filter	• Enable • Disable	Disable	Enable the digital noise filter on RXDn pin. The digital noise filter block in SCI consists of two-stage flipflop circuits.
Extra > Receive FIFO Trigger Level	• One • Max	Max	Unused if the channel has no FIFO or if DTC is used for reception. Set to One to get a callback immediately when each byte is received. Set to Max to get a callback when FIFO is full or after 15 bit times with no data (fewer interrupts).
Interrupts > Callback	Name must be a valid C symbol	user_uart_callback	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).
Interrupts > Receive Interrupt Priority	MCU Specific Options		Select the receive interrupt priority.
Interrupts > Transmit Data Empty Interrupt Priority	MCU Specific Options		Select the transmit interrupt priority.
Interrupts > Transmit End Interrupt Priority	MCU Specific Options		Select the transmit end interrupt priority.
Interrupts > Error Interrupt Priority	MCU Specific Options		Select the error interrupt priority.

Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

MCU Group	Peripheral Clock
RA2A1	PCLKB
RA2E1	PCLKB
RA2E2	PCLKB
RA2L1	PCLKB
RA4E1	PCLKA
RA4M1	PCLKA

RA4M2	PCLKA
RA4M3	PCLKA
RA4W1	PCLKA
RA6E1	PCLKA
RA6M1	PCLKA
RA6M2	PCLKA
RA6M3	PCLKA
RA6M4	PCLKA
RA6M5	PCLKA
RA6T1	PCLKA
RA6T2	PCLKA

The clock source for the baud-rate clock generator can be selected from the internal clock, the external clock times 8 or the external clock times 16. The external clock is supplied to the SCK pin.

Pin Configuration

This module uses TXD and RXD to communicate to external devices. CTS or RTS or both (CTS and RTS) can be controlled by the hardware. Some MCUs support hardware flow control for both CTS and RTS on some channels. Some MCUs and channels support hardware flow control for either CTS or RTS but not both. If both are desired a GPIO pin can be used for RTS. When the internal clock is the source for the baud-rate generator the SCK pin can be used to output a clock with the same frequency as the bit rate.

Usage Notes

- When configured for Hardware CTS and Software RTS the configured flow control pin will be used for RTS. The pin will be set high inside of the receive ISR while data is being read. It will be set low when all data is read.
- When configured for Hardware CTS and Hardware RTS the CSTn_RTSn pin will be used for RTS function and the CTSn pin will be used for CTS function on channel n.
- The driver will follow correct hardware flow control function when CTSn_RTSn pin is connected to CTSn pin when "Hardware CTS and Hardware RTS" flow control is selected. The data will still be transferred when CTSn_RTSn and CTSn are disconnected as the CTSn pin is internally pulled low on the hardware when CTSn pin is configured as a peripheral pin for SCI module. Do not configure CTSn pin if the hardware flow control is not desired.

Limitations

- Reception is still enabled after uart_api_t::communicationAbort API is called. Any characters received after abort and before the next call to read will arrive via the callback function with event UART_EVENT_RX_CHAR.

DTC Limitations

- DTC support is available for reception, but labeled as [Not recommended]. This is because the UART bytes are received asynchronously. Bytes can be received between calls to R_SCI_UART_Read(). The logic required to combine bytes received through

R_SCI_UART_Read() (UART_EVENT_RX_COMPLETE) and bytes received between calls (UART_EVENT_RX_CHAR) is complex. Reception length may also be unknown, and the driver will not issue an interrupt unless the entire DTC buffer is filled.
- Transfer size must be less than or equal to 64K bytes if DTC interface is used for transfer. uart_api_t::infoGet API can be used to get the max transfer size allowed.
- When using 9-bit reception with DTC, clear the upper 7 bits of data before processing the read data. The upper 7 bits contain status flags that are part of the register used to read data in 9-bit mode.

Examples

SCI UART Example

```
uint8_t g_dest[TRANSFER_LENGTH];
uint8_t g_src[TRANSFER_LENGTH];
uint8_t g_out_of_band_received[TRANSFER_LENGTH];
uint32_t g_transfer_complete = 0;
uint32_t g_receive_complete = 0;
uint32_t g_out_of_band_index = 0;
void r_sci_uart_basic_example (void)
{
 /* Initialize p_src to known data */
 for (uint32_t i = 0; i < TRANSFER_LENGTH; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the transfer instance with initial configuration. */
 fsp_err_t err = R_SCI_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
    assert(FSP_SUCCESS == err);
    err = R_SCI_UART_Read(&g_uart0_ctrl, g_dest, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
    err = R_SCI_UART_Write(&g_uart0_ctrl, g_src, TRANSFER_LENGTH);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
    }
 while (!g_receive_complete)
    {
    }
```

```
}
void example_callback (uart_callback_args_t * p_args)
{
 /* Handle the UART event */
 switch (p_args->event)
    {
 /* Received a character */
 case UART_EVENT_RX_CHAR:
      {
 /* Only put the next character in the receive buffer if there is space for it */
 if (sizeof(g_out_of_band_received) > g_out_of_band_index)
      {
 /* Write either the next one or two bytes depending on the receive data size */
 if (UART_DATA_BITS_8 >= g_uart0_cfg.data_bits)
      {
                   g_out_of_band_received[g_out_of_band_index++] = (uint8_t)
p_args->data;
      }
 else
      {
                   uint16_t * p_dest = (uint16_t *)
&g_out_of_band_received[g_out_of_band_index];
                   *p_dest             = (uint16_t) p_args->data;
                   g_out_of_band_index += 2;
      }
      }
 break;
      }
 /* Receive complete */
 case UART_EVENT_RX_COMPLETE:
      {
            g_receive_complete = 1;
 break;
      }
```

```
 /* Transmit complete */
 case UART_EVENT_TX_COMPLETE:
      {
            g_transfer_complete = 1;
 break;
      }
 default:
      {
      }
    }
}
```

SCI UART Baud Set Example

```
#define SCI_UART_BAUDRATE_19200 (19200)
void r_sci_uart_baud_example (void)
{
 baud_setting_t baud_setting;
    uint32_t       baud_rate                = SCI_UART_BAUDRATE_19200;
 bool          enable_bitrate_modulation = false;
    uint32_t       error_rate_x_1000         = 5;
 fsp_err_t err = R_SCI_UART_BaudCalculate(baud_rate, enable_bitrate_modulation,
error_rate_x_1000, &baud_setting);
    assert(FSP_SUCCESS == err);
    err = R_SCI_UART_BaudSet(&g_uart0_ctrl, (void *) &baud_setting);
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	sci_uart_instance_ctrl_t
struct	baud_setting_t
struct	sci_uart_extended_cfg_t

Enumerations

enum	sci_clk_src_t

enum	sci_uart_flow_control_t
enum	sci_uart_rx_fifo_trigger_t
enum	sci_uart_start_bit_detect_t
enum	sci_uart_noise_cancellation_t

Data Structure Documentation

◆ sci_uart_instance_ctrl_t

struct sci_uart_instance_ctrl_t
UART instance control block.

◆ baud_setting_t

struct baud_setting_t

Register settings to acheive a desired baud rate and modulation duty.

Data Fields

union baud_setting_t	__unnamed__	
uint8_t	cks: 2	CKS value to get divisor (CKS = N)
uint8_t	brr	Bit Rate Register setting.
uint8_t	mddr	Modulation Duty Register setting.

◆ sci_uart_extended_cfg_t

struct sci_uart_extended_cfg_t

UART on SCI device Configuration

Data Fields

sci_clk_src_t	clock	The source clock for the baud-rate generator. If internal optionally output baud rate on SCK.
sci_uart_start_bit_detect_t	rx_edge_start	Start reception on falling edge.
sci_uart_noise_cancellation_t	noise_cancel	Noise cancellation setting.
baud_setting_t *	p_baud_setting	Register settings for a desired baud rate.
sci_uart_rx_fifo_trigger_t	rx_fifo_trigger	Receive FIFO trigger level, unused if channel has no FIFO or if DTC is used.

bsp_io_port_pin_t	flow_control_pin	UART Driver Enable pin.
sci_uart_flow_control_t	flow_control	CTS/RTS function of the SSn pin.

Enumeration Type Documentation

◆ sci_clk_src_t

enum sci_clk_src_t	
Enumeration for SCI clock source	
Enumerator	
SCI_UART_CLOCK_INT	Use internal clock for baud generation.
SCI_UART_CLOCK_INT_WITH_BAUDRATE_OUTPUT	Use internal clock for baud generation and output on SCK.
SCI_UART_CLOCK_EXT8X	Use external clock 8x baud rate.
SCI_UART_CLOCK_EXT16X	Use external clock 16x baud rate.

◆ sci_uart_flow_control_t

enum sci_uart_flow_control_t	
UART flow control mode definition	
Enumerator	
SCI_UART_FLOW_CONTROL_RTS	Use SCI pin for RTS.
SCI_UART_FLOW_CONTROL_CTS	Use SCI pin for CTS.
SCI_UART_FLOW_CONTROL_CTSRTS	Use SCI pin for CTS, external pin for RTS.
SCI_UART_FLOW_CONTROL_HARDWARE_CTSRTS	Use CTSn_RTSn pin for RTS and CTSn pin for CTS. Available only for some channels on selected MCUs. See hardware manual for channel specific options.

◆ sci_uart_rx_fifo_trigger_t

enum sci_uart_rx_fifo_trigger_t	
Receive FIFO trigger configuration.	
Enumerator	
SCI_UART_RX_FIFO_TRIGGER_1	Callback after each byte is received without buffering.
SCI_UART_RX_FIFO_TRIGGER_MAX	Callback when FIFO is full or after 15 bit times with no data (fewer interrupts)

◆ sci_uart_start_bit_detect_t

enum sci_uart_start_bit_detect_t	
Asynchronous Start Bit Edge Detection configuration.	
Enumerator	
SCI_UART_START_BIT_LOW_LEVEL	Detect low level on RXDn pin as start bit.
SCI_UART_START_BIT_FALLING_EDGE	Detect falling level on RXDn pin as start bit.

◆ sci_uart_noise_cancellation_t

enum sci_uart_noise_cancellation_t	
Noise cancellation configuration.	
Enumerator	
SCI_UART_NOISE_CANCELLATION_DISABLE	Disable noise cancellation.
SCI_UART_NOISE_CANCELLATION_ENABLE	Enable noise cancellation.

Function Documentation

◆ R_SCI_UART_Open()

fsp_err_t R_SCI_UART_Open (uart_ctrl_t *const *p_api_ctrl*, uart_cfg_t const *const *p_cfg*)

Configures the UART driver based on the input configurations. If reception is enabled at compile time, reception is enabled at the end of this function. Implements uart_api_t::open

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ASSERTION	Pointer to UART control block or configuration structure is NULL.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The requested channel does not exist on this MCU.
FSP_ERR_INVALID_ARGUMENT	Flow control is enabled but flow control pin is not defined or selected channel does not support "Hardware CTS and Hardware RTS" flow control.
FSP_ERR_ALREADY_OPEN	Control block has already been opened or channel is being used by another instance. Call close() then open() to reconfigure.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open

◆ R_SCI_UART_Read()

fsp_err_t R_SCI_UART_Read (uart_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Receives user specified number of bytes into destination buffer pointer. Implements uart_api_t::read

Return values

FSP_SUCCESS	Data reception successfully ends.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used
FSP_ERR_INVALID_ARGUMENT	Destination address or data size is not valid for 9-bit mode.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_IN_USE	A previous read operation is still in progress.
FSP_ERR_UNSUPPORTED	SCI_UART_CFG_RX_ENABLE is set to 0

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

Note

If 9-bit data length is specified at R_SCI_UART_Open call, p_dest must be aligned 16-bit boundary.

◆ R_SCI_UART_Write()

fsp_err_t R_SCI_UART_Write (uart_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_src*, uint32_t const *bytes*)

Transmits user specified number of bytes from the source buffer pointer. Implements uart_api_t::write

Return values

FSP_SUCCESS	Data transmission finished successfully.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL. Number of transfers outside the max or min boundary when transfer instance used
FSP_ERR_INVALID_ARGUMENT	Source address or data size is not valid for 9-bit mode.
FSP_ERR_NOT_OPEN	The control block has not been opened
FSP_ERR_IN_USE	A UART transmission is in progress
FSP_ERR_UNSUPPORTED	SCI_UART_CFG_TX_ENABLE is set to 0

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

Note

If 9-bit data length is specified at R_SCI_UART_Open call, p_src must be aligned on a 16-bit boundary.

◆ R_SCI_UART_BaudSet()

fsp_err_t R_SCI_UART_BaudSet (uart_ctrl_t *const *p_api_ctrl*, void const *const *p_baud_setting*)

Updates the baud rate using the clock selected in Open. p_baud_setting is a pointer to a baud_setting_t structure. Implements uart_api_t::baudSet

Warning

This terminates any in-progress transmission.

Return values

FSP_SUCCESS	Baud rate was successfully changed.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL or the UART is not configured to use the internal clock.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_SCI_UART_InfoGet()

fsp_err_t R_SCI_UART_InfoGet (uart_ctrl_t *const *p_api_ctrl*, uart_info_t *const *p_info*)

Provides the driver information, including the maximum number of bytes that can be received or transmitted at a time. Implements uart_api_t::infoGet

Return values

FSP_SUCCESS	Information stored in provided p_info.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_SCI_UART_Close()

fsp_err_t R_SCI_UART_Close (uart_ctrl_t *const *p_api_ctrl*)

Aborts any in progress transfers. Disables interrupts, receiver, and transmitter. Closes lower level transfer drivers if used. Removes power. Implements uart_api_t::close

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened

◆ R_SCI_UART_Abort()

fsp_err_t R_SCI_UART_Abort (uart_ctrl_t *const *p_api_ctrl*, uart_dir_t *communication_to_abort*)

Provides API to abort ongoing transfer. Transmission is aborted after the current character is transmitted. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::communicationAbort

Return values

FSP_SUCCESS	UART transaction aborted successfully.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_UNSUPPORTED	The requested Abort direction is unsupported.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::disable

◆ R_SCI_UART_BaudCalculate()

fsp_err_t R_SCI_UART_BaudCalculate (uint32_t *baudrate*, bool *bitrate_modulation*, uint32_t *baud_rate_error_x_1000*, baud_setting_t *const *p_baud_setting*)

Calculates baud rate register settings. Evaluates and determines the best possible settings set to the baud rate related registers.

Parameters

[in]	baudrate	Baud rate [bps]. For example, 19200, 57600, 115200, etc.
[in]	bitrate_modulation	Enable bitrate modulation
[in]	baud_rate_error_x_1000	<baud_rate_percent_error> x 1000 required for module to function. Absolute max baud_rate_error is 15000 (15%).
[out]	p_baud_setting	Baud setting information stored here if successful

Return values

FSP_SUCCESS	Baud rate is set successfully
FSP_ERR_ASSERTION	Null pointer
FSP_ERR_INVALID_ARGUMENT	Baud rate is '0', source clock frequency could not be read, or error in calculated baud rate is larger than 10%.

◆ R_SCI_UART_CallbackSet()

fsp_err_t R_SCI_UART_CallbackSet (uart_ctrl_t *const *p_api_ctrl*, void(*)(uart_callback_args_t *) *p_callback*, void const *const *p_context*, uart_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements uart_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_SCI_UART_ReadStop()

fsp_err_t R_SCI_UART_ReadStop (uart_ctrl_t *const *p_api_ctrl*, uint32_t * *remaining_bytes*)

Provides API to abort ongoing read. Reception is still enabled after abort(). Any characters received after abort() and before the transfer is reset in the next call to read(), will arrive via the callback function with event UART_EVENT_RX_CHAR. Implements uart_api_t::readStop

Return values

FSP_SUCCESS	UART transaction aborted successfully.
FSP_ERR_ASSERTION	Pointer to UART control block is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_UNSUPPORTED	The requested Abort direction is unsupported.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- transfer_api_t::disable

4.2.56 Sigma Delta Analog to Digital Converter (r_sdadc)

Modules

Functions

fsp_err_t	R_SDADC_Open (adc_ctrl_t *p_ctrl, adc_cfg_t const *const p_cfg)
fsp_err_t	R_SDADC_ScanCfg (adc_ctrl_t *p_ctrl, void const *const p_extend)
fsp_err_t	R_SDADC_InfoGet (adc_ctrl_t *p_ctrl, adc_info_t *p_adc_info)
fsp_err_t	R_SDADC_ScanStart (adc_ctrl_t *p_ctrl)
fsp_err_t	R_SDADC_ScanGroupStart (adc_ctrl_t *p_ctrl, adc_group_mask_t group_id)
fsp_err_t	R_SDADC_ScanStop (adc_ctrl_t *p_ctrl)
fsp_err_t	R_SDADC_StatusGet (adc_ctrl_t *p_ctrl, adc_status_t *p_status)
fsp_err_t	R_SDADC_Read (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data)
fsp_err_t	R_SDADC_Read32 (adc_ctrl_t *p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data)

fsp_err_t	R_SDADC_OffsetSet (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset)
fsp_err_t	R_SDADC_Calibrate (adc_ctrl_t *const p_ctrl, void const *p_extend)
fsp_err_t	R_SDADC_Close (adc_ctrl_t *p_ctrl)

Detailed Description

Driver for the SDADC24 peripheral on RA MCUs. This module implements the ADC Interface.

Overview

Features

The SDADC module supports the following features:

- 24 bit maximum resolution
- Configure scans to include:
  - Multiple analog channels
  - Outputs of OPAMP0 (P side) and OPAMP1 (N side) of SDADC channel 4
- Configurable scan start trigger:
  - Software scan triggers
  - Hardware scan triggers (timer expiration, for example)
- Configurable scan mode:
  - Single scan mode, where each trigger starts a single scan
  - Continuous scan mode, where all channels are scanned continuously
- Supports averaging converted samples
- Optional callback when single conversion, entire scan, or calibration completes
- Supports reading converted data
- Sample and hold support

Selecting an ADC

All RA MCUs have an Analog to Digital Converter (r_adc). Only select RA MCUs have an SDADC. When selecting between them, consider these factors. Refer to the hardware manual for details.

	ADC	SDADC
Availability	Available on all RA MCUs.	Available on select RA MCUs.
Resolution	The ADC has a maximum resolution of 12, 14, or 16 bits depending on the MCU.	The SDADC has a maximum accuracy of 24 bits.
Number of Channels	The ADC has more channels than the SDADC.	The SDADC 5 channels, one of which is tied to OPAMP0 and OPAMP1.
Frequency	The ADC sampling time is shorter (more samples per second).	The SDADC sampling time is longer (fewer samples per second).

Settling Time	The ADC does not have a settling time when switching between channels.	The SDADC requires a settling time when switching between channels.

Configuration

Build Time Configurations for r_sdadc

The following build time configurations are defined in fsp_cfg/r_sdadc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Analog > ADC (r_sdadc)

This module can be added to the Stacks tab via New Stack > Analog > ADC (r_sdadc).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_adc0	Module name.
Mode	• Single Scan • Continuous Scan	Continuous Scan	In single scan mode, all channels are converted once per start trigger, and conversion stops after all enabled channels are scanned. In continuous scan mode, conversion starts after a start trigger, then continues until stopped in software.
Resolution	• 16 Bit • 24 Bit	24 Bit	Select 24-bit or 16-bit resolution.
Alignment	• Right • Left	Right	Select left or right alignment.
Trigger	MCU Specific Options		Select conversion start trigger. Conversion can be started in software, or conversion can be started when a hardware event occurs if the hardware event is linked to the SDADC peripheral using the ELC API.

Vref Source	• Internal • External	Internal	Vref can be source internally and output on the SBIAS pin, or Vref can be input from VREFI.
Vref Voltage	• 0.8 V • 1.0 V • 1.2 V • 1.4 V • 1.6 V • 1.8 V • 2.0 V • 2.2 V • 2.4 V	1.0 V	Select Vref voltage. If Vref is input externally, the voltage on VREFI must match the voltage selected within 3%.
Callback	Name must be a valid C symbol	NULL	Enter the name of the callback function to be called when conversion completes or a scan ends.
Conversion End Interrupt Priority	MCU Specific Options		[Required] Select the interrupt priority for the conversion end interrupt.
Scan End Interrupt Priority	MCU Specific Options		[Optional] Select the interrupt priority for the scan end interrupt.
Calibration End Interrupt Priority	MCU Specific Options		[Optional] Select the interrupt priority for the calibration end interrupt.

Configurations for Analog > SDADC Channel Configuration (r_sdadc)

Configuration	Options	Default	Description
Input	• Differential • Single Ended	Differential	Select differential or single-ended input.
Stage 1 Gain	• 1 • 2 • 3 • 4 • 8	1	Select the gain for stage 1 of the PGA. Must be 1 for single-ended input.
Stage 2 Gain	• 1 • 2 • 4 • 8	1	Select the gain for stage 2 of the PGA. Must be 1 for single-ended input.
Oversampling Ratio	• 64 • 128 • 256 • 512	256	Select the oversampling ratio for the PGA. Must be 256 for single-ended input.

	• 1024 • 2048		
Polarity (Valid for Single-Ended Input Only)	• Positive • Negative	Positive	Select positive or negative polarity for single-ended input. VBIAS (1.0 V typical) is connected on the opposite input.
Conversions to Average per Result	• Do Not Average (Interrupt after Each Conversion) • Average 8 • Average 16 • Average 32 • Average 64	Do Not Average (Interrupt after Each Conversion)	Select the number of conversions to average for each result. The ADC_EVENT_CONVERSION_END event occurs after each average, or after each individual conversion if averaging is disabled.
Invert (Valid for Negative Single-Ended Input Only)	• Result Not Inverted • Result Inverted	Result Not Inverted	Select whether to invert negative single-ended input. When the result is inverted, the lowest measurable voltage gives a result of 0, and the highest measurable voltage gives a result of 2^resolution - 1.
Number of Conversions Per Scan	Refer to the RA Configuration tool for available options.	1	Number of conversions on this channel before AUTOSCAN moves to the next channel. When all conversions of all channels are complete, the ADC_EVENT_SCAN_END event occurs.

Clock Configuration

The SDADC clock clock is configurable on the clocks tab.

The SDADC clock must be 4 MHz when the SDADC is used.

Pin Configuration

The ANSDnP (n = 0-3) pins are analog input channels that can be used with the SDADC.

Usage Notes

Scan Procedure

In this document, the term "scan" refers to the AUTOSCAN feature of the SDADC, which works as follows:

1. Conversions are performed on enabled channels in ascending order of channel number. All conversions required for a single channel are completed before the sequencer moves to the next channel.
2. Conversions are performed at the rate (in Hz) of the SDADC oversampling clock frequency / oversampling ratio (configured per channel). The FSP uses the normal mode SDADC oversampling clock frequency.
3. If averaging is enabled for the channel, the number of conversions to average are performed before each conversion end interrupt occurs.
4. If the number of conversions for the channel is more than 1, SDADC performs the number of conversions requested. These are performed consecutively. There is a settling time associated with switching channels. Performing all of the requested conversions for each channel at a time avoids this settling time after the first conversion.

   If averaging is enabled for the channel, each averaged result counts as a single conversion.
5. Continues to the next enabled channel only after completing all conversions requested.
6. After all enabled channels are scanned, a scan end interrupt occurs. The driver supports single-scan and continuous scan operation modes.
   - Single-scan mode performs one scan per trigger (hardware trigger or software start using R_SDADC_ScanStart).
   - In continuous scan mode, the scan is restarted after each scan completes. A single trigger is required to start continuous operation of the SDADC.

When Interrupts Are Not Enabled

If interrupts are not enabled, the R_SDADC_StatusGet() API can be used to poll the SDADC to determine when the scan has completed. The R_SDADC_Read() API function is used to access the converted SDADC result. This applies to both normal scans and calibration scans.

Calibration

Calibration is required to use the SDADC if any channel is configured for differential mode. Call R_SDADC_Calibrate() after open, and prior to any other function, then wait for a calibration complete event before using the SDADC. R_SDADC_Calibrate() should not be called if all channels are configured for single-ended mode.

Examples

Basic Example

This is a basic example of minimal use of the SDADC in an application.

```
void sdadc_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = R_SDADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
```

```
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Calibrate all differential channels. */
sdadc_calibrate_args_t calibrate_args;
   calibrate_args.mode    = SDADC_CALIBRATION_INTERNAL_GAIN_OFFSET;
   calibrate_args.channel = ADC_CHANNEL_0;
   err = R_SDADC_Calibrate(&g_adc0_ctrl, &calibrate_args);
   assert(FSP_SUCCESS == err);
/* Wait for calibration to complete. */
adc_status_t status;
   status.state = ADC_STATE_SCAN_IN_PROGRESS;
while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
   {
R_SDADC_StatusGet(&g_adc0_ctrl, &status);
   }
/* In software trigger mode, start a scan by calling R_SDADC_ScanStart(). In other
modes, enable external
 * triggers by calling R_SDADC_ScanStart(). */
   (void) R_SDADC_ScanStart(&g_adc0_ctrl);
/* Wait for conversion to complete. */
   status.state = ADC_STATE_SCAN_IN_PROGRESS;
while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
   {
R_SDADC_StatusGet(&g_adc0_ctrl, &status);
   }
/* Read converted data. */
   uint32_t channel1_conversion_result;
R_SDADC_Read32(&g_adc0_ctrl, ADC_CHANNEL_1, &channel1_conversion_result);
}
```

Using DTC or DMAC with the SDADC

If desired, the DTC or DMAC can be used to store each conversion result in a circular buffer. An example configuration is below.

```
/* Example DTC transfer settings to used with SDADC. */
/* The transfer length should match the total number of conversions per scan. This
example assumes the SDADC is
 * configured to scan channel 1 three times, then channel 2 and channel 4 once, for a
total of 5 conversions. */
#define SDADC_EXAMPLE_TRANSFER_LENGTH (5)
uint32_t g_sdadc_example_buffer[SDADC_EXAMPLE_TRANSFER_LENGTH];
transfer_info_t g_sdadc_transfer_info =
{
    .dest_addr_mode = TRANSFER_ADDR_MODE_INCREMENTED,
    .repeat_area    = TRANSFER_REPEAT_AREA_DESTINATION,
    .irq            = TRANSFER_IRQ_END,
    .chain_mode    = TRANSFER_CHAIN_MODE_DISABLED,
    .src_addr_mode = TRANSFER_ADDR_MODE_FIXED,
    .mode          = TRANSFER_MODE_REPEAT,
 /* NOTE: The data transferred will contain a 24-bit converted value in bits 23:0.
Bit 24 contains a status flag
  * indicating if the result overflowed or not. Bits 27:25 contain the channel number
+ 1. The settings for
  * resolution and alignment and ignored when DTC or DMAC is used. */
    .size          = TRANSFER_SIZE_4_BYTE,
 /* NOTE: It is strongly recommended to enable averaging on all channels or no
channels when using DTC with SDADC
  * because the result register is different when averaging is used. If averaging is
enabled on all channels,
  * set transfer_info_t::p_src to &R_SDADC->ADAR. */
    .p_src = (void const *) &R_SDADC0->ADCR,
    .p_dest = &g_sdadc_example_buffer[0],
    .length = SDADC_EXAMPLE_TRANSFER_LENGTH,
};
void sdadc_dtc_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
```

```
    err = R_SDADC_Open(&g_adc0_ctrl, &g_adc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Calibrate all differential channels. */
 sdadc_calibrate_args_t calibrate_args;
    calibrate_args.mode    = SDADC_CALIBRATION_INTERNAL_GAIN_OFFSET;
    calibrate_args.channel = ADC_CHANNEL_0;
    err = R_SDADC_Calibrate(&g_adc0_ctrl, &calibrate_args);
    assert(FSP_SUCCESS == err);
 /* Wait for calibration to complete. */
 adc_status_t status;
    status.state = ADC_STATE_SCAN_IN_PROGRESS;
 while (ADC_STATE_SCAN_IN_PROGRESS == status.state)
    {
 R_SDADC_StatusGet(&g_adc0_ctrl, &status);
    }
 /* In software trigger mode, start a scan by calling R_SDADC_ScanStart(). In other
modes, enable external
  * triggers by calling R_SDADC_ScanStart(). */
    (void) R_SDADC_ScanStart(&g_adc0_ctrl);
 /* After each conversion, the converted data is transferred to the next index in
g_sdadc_example_buffer. After
  * the entire scan completes, the index in g_sdadc_example_buffer resets. The data
in g_sdadc_example_buffer
  * is:
  * - g_sdadc_example_buffer[0] = SDADC channel 1 conversion 0
  * - g_sdadc_example_buffer[1] = SDADC channel 1 conversion 1
  * - g_sdadc_example_buffer[2] = SDADC channel 1 conversion 2
  * - g_sdadc_example_buffer[3] = SDADC channel 2 conversion 0
  * - g_sdadc_example_buffer[4] = SDADC channel 4 conversion 0
  *//* At any point in the application after the first scan completes, the most
recent data for channel 2 can be read
  * from the buffer like this. Shifting removes the unrelated bits in the result
register and propagates the sign
```

```
  * bit so the value can be interpreted as a signed result. This assumes channel 2 is
configured in differential
  * mode. */
    int32_t channel_2_data = (int32_t) (g_sdadc_example_buffer[3] << 8) >> 8;
 FSP_PARAMETER_NOT_USED(channel_2_data);
}
```

Data Structures

struct	sdadc_calibrate_args_t
struct	sdadc_channel_cfg_t
struct	sdadc_scan_cfg_t
struct	sdadc_extended_cfg_t
struct	sdadc_instance_ctrl_t

Enumerations

enum	sdadc_vref_src_t
enum	sdadc_vref_voltage_t
enum	sdadc_channel_input_t
enum	sdadc_channel_stage_1_gain_t
enum	sdadc_channel_stage_2_gain_t
enum	sdadc_channel_oversampling_t
enum	sdadc_channel_polarity_t
enum	sdadc_channel_average_t
enum	sdadc_channel_inversion_t
enum	sdadc_channel_count_formula_t
enum	sdadc_calibration_t

Data Structure Documentation

◆ sdadc_calibrate_args_t

struct sdadc_calibrate_args_t		
Structure to pass to the adc_api_t::calibrate p_extend argument.		
Data Fields		
adc_channel_t	channel	Which channel to calibrate.
sdadc_calibration_t	mode	Calibration mode.

◆ sdadc_channel_cfg_t

struct sdadc_channel_cfg_t
SDADC per channel configuration.

◆ sdadc_scan_cfg_t

struct sdadc_scan_cfg_t		
SDADC active channel configuration		
Data Fields		
uint32_t	scan_mask	Channels/bits: bit 0 is ch0; bit 15 is ch15.

◆ sdadc_extended_cfg_t

struct sdadc_extended_cfg_t		
SDADC configuration extension. This extension is required and must be provided in adc_cfg_t::p_extend.		
Data Fields		
uint8_t	conv_end_ipl	Conversion end interrupt priority.
IRQn_Type	conv_end_irq	
sdadc_vref_src_t	vref_src	Source of Vref (internal or external)
sdadc_vref_voltage_t	vref_voltage	Voltage of Vref, required for both internal and external Vref. If Vref is from an external source, the voltage must match the specified voltage within 3%.
sdadc_channel_cfg_t const *	p_channel_cfgs[SDADC_MAX_NUM_CHANNELS]	Configuration for each channel, set to NULL if unused.

◆ sdadc_instance_ctrl_t

struct sdadc_instance_ctrl_t
ADC instance control block. DO NOT INITIALIZE. Initialized in adc_api_t::open().

Enumeration Type Documentation

◆ sdadc_vref_src_t

enum sdadc_vref_src_t	
Source of Vref.	
Enumerator	
SDADC_VREF_SRC_INTERNAL	Vref is internally sourced, can be output as SBIAS.
SDADC_VREF_SRC_EXTERNAL	Vref is externally sourced from the VREFI pin.

◆ sdadc_vref_voltage_t

enum sdadc_vref_voltage_t	
Voltage of Vref.	
Enumerator	
SDADC_VREF_VOLTAGE_800_MV	Vref is 0.8 V.
SDADC_VREF_VOLTAGE_1000_MV	Vref is 1.0 V.
SDADC_VREF_VOLTAGE_1200_MV	Vref is 1.2 V.
SDADC_VREF_VOLTAGE_1400_MV	Vref is 1.4 V.
SDADC_VREF_VOLTAGE_1600_MV	Vref is 1.6 V.
SDADC_VREF_VOLTAGE_1800_MV	Vref is 1.8 V.
SDADC_VREF_VOLTAGE_2000_MV	Vref is 2.0 V.
SDADC_VREF_VOLTAGE_2200_MV	Vref is 2.2 V.
SDADC_VREF_VOLTAGE_2400_MV	Vref is 2.4 V (only valid for external Vref)

◆ sdadc_channel_input_t

enum sdadc_channel_input_t	
Per channel input mode.	
Enumerator	
SDADC_CHANNEL_INPUT_DIFFERENTIAL	Differential input.
SDADC_CHANNEL_INPUT_SINGLE_ENDED	Single-ended input.

◆ sdadc_channel_stage_1_gain_t

enum sdadc_channel_stage_1_gain_t	
Per channel stage 1 gain options.	
Enumerator	
SDADC_CHANNEL_STAGE_1_GAIN_1	Gain of 1.
SDADC_CHANNEL_STAGE_1_GAIN_2	Gain of 2.
SDADC_CHANNEL_STAGE_1_GAIN_3	Gain of 3 (only valid for stage 1)
SDADC_CHANNEL_STAGE_1_GAIN_4	Gain of 4.
SDADC_CHANNEL_STAGE_1_GAIN_8	Gain of 8.

◆ sdadc_channel_stage_2_gain_t

enum sdadc_channel_stage_2_gain_t	
Per channel stage 2 gain options.	
Enumerator	
SDADC_CHANNEL_STAGE_2_GAIN_1	Gain of 1.
SDADC_CHANNEL_STAGE_2_GAIN_2	Gain of 2.
SDADC_CHANNEL_STAGE_2_GAIN_4	Gain of 4.
SDADC_CHANNEL_STAGE_2_GAIN_8	Gain of 8.

◆ sdadc_channel_oversampling_t

enum sdadc_channel_oversampling_t	
Per channel oversampling ratio.	
Enumerator	
SDADC_CHANNEL_OVERSAMPLING_64	Oversampling ratio of 64.
SDADC_CHANNEL_OVERSAMPLING_128	Oversampling ratio of 128.
SDADC_CHANNEL_OVERSAMPLING_256	Oversampling ratio of 256.
SDADC_CHANNEL_OVERSAMPLING_512	Oversampling ratio of 512.
SDADC_CHANNEL_OVERSAMPLING_1024	Oversampling ratio of 1024.
SDADC_CHANNEL_OVERSAMPLING_2048	Oversampling ratio of 2048.

◆ sdadc_channel_polarity_t

enum sdadc_channel_polarity_t	
Per channel polarity, valid for single-ended input only.	
Enumerator	
SDADC_CHANNEL_POLARITY_POSITIVE	Positive-side single-ended input.
SDADC_CHANNEL_POLARITY_NEGATIVE	Negative-side single-ended input.

◆ sdadc_channel_average_t

enum sdadc_channel_average_t	
Per channel number of conversions to average before conversion end callback.	
Enumerator	
SDADC_CHANNEL_AVERAGE_NONE	Do not average (callback for each conversion)
SDADC_CHANNEL_AVERAGE_8	Average 8 samples for each conversion end callback.
SDADC_CHANNEL_AVERAGE_16	Average 16 samples for each conversion end callback.
SDADC_CHANNEL_AVERAGE_32	Average 32 samples for each conversion end callback.
SDADC_CHANNEL_AVERAGE_64	Average 64 samples for each conversion end callback.

◆ sdadc_channel_inversion_t

enum sdadc_channel_inversion_t	
Per channel polarity, valid for negative-side single-ended input only.	
Enumerator	
SDADC_CHANNEL_INVERSION_OFF	Do not invert conversion result.
SDADC_CHANNEL_INVERSION_ON	Invert conversion result.

◆ sdadc_channel_count_formula_t

enum sdadc_channel_count_formula_t

Select a formula to specify the number of conversions. The following symbols are used in the formulas:

- N: Number of conversions
- n: sdadc_channel_cfg_t::coefficient_n, do not set to 0 if m is 0
- m: sdadc_channel_cfg_t::coefficient_m, do not set to 0 if n is 0

Either m or n must be non-zero.

Enumerator	
SDADC_CHANNEL_COUNT_FORMULA_EXPONENTIAL	N = 32 * (2 ^ n - 1) + m * 2 ^ n.
SDADC_CHANNEL_COUNT_FORMULA_LINEAR	N = (32 * n) + m.

◆ sdadc_calibration_t

enum sdadc_calibration_t

Calibration mode.

Enumerator	
SDADC_CALIBRATION_INTERNAL_GAIN_OFFSET	Use internal reference to calibrate offset and gain.
SDADC_CALIBRATION_EXTERNAL_OFFSET	Use external reference to calibrate offset.
SDADC_CALIBRATION_EXTERNAL_GAIN	Use external reference to calibrate gain.

Function Documentation

◆ R_SDADC_Open()

fsp_err_t R_SDADC_Open (adc_ctrl_t * *p_ctrl*, adc_cfg_t const *const *p_cfg*)

Applies power to the SDADC and initializes the hardware based on the user configuration. As part of this initialization, the SDADC clock is configured and enabled. If an interrupt priority is non-zero, enables an interrupt which will call a callback to notify the user when a conversion, scan, or calibration is complete. R_SDADC_Calibrate() must be called after this function before using the SDADC if any channels are used in differential mode. Implements adc_api_t::open().

Note

This function delays at least 2 ms as required by the SDADC power on procedure.

Return values

FSP_SUCCESS	Configuration successful.
FSP_ERR_ASSERTION	An input pointer is NULL or an input parameter is invalid.
FSP_ERR_ALREADY_OPEN	Control block is already open.
FSP_ERR_IRQ_BSP_DISABLED	A required interrupt is disabled

◆ R_SDADC_ScanCfg()

fsp_err_t R_SDADC_ScanCfg (adc_ctrl_t * *p_ctrl*, void const *const *p_extend*)

Configures the enabled channels of the ADC. Pass a pointer to sdadc_scan_cfg_t to p_extend. Implements adc_api_t::scanCfg().

Return values

FSP_SUCCESS	Information stored in p_adc_info.
FSP_ERR_ASSERTION	An input pointer is NULL or an input parameter is invalid.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_SDADC_InfoGet()

fsp_err_t R_SDADC_InfoGet (adc_ctrl_t * *p_ctrl*, adc_info_t * *p_adc_info*)

Returns the address of the lowest number configured channel, the total number of results to be read in order to read the results of all configured channels, the size of each result, and the ELC event enumerations. Implements adc_api_t::infoGet().

Return values

FSP_SUCCESS	Information stored in p_adc_info.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_SDADC_ScanStart()

fsp_err_t R_SDADC_ScanStart (adc_ctrl_t * *p_ctrl*)

If the SDADC is configured for hardware triggers, enables hardware triggers. Otherwise, starts a scan. Implements adc_api_t::scanStart().

Return values

FSP_SUCCESS	Scan started or hardware triggers enabled successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.
FSP_ERR_IN_USE	A conversion or calibration is in progress.

◆ R_SDADC_ScanGroupStart()

fsp_err_t R_SDADC_ScanGroupStart (adc_ctrl_t * *p_ctrl*, adc_group_mask_t *group_id*)

adc_api_t::scanStart is not supported on the SDADC. Use scanStart instead.

Return values

FSP_ERR_UNSUPPORTED	Function not supported in this implementation.

◆ R_SDADC_ScanStop()

fsp_err_t R_SDADC_ScanStop (adc_ctrl_t * *p_ctrl*)

If the SDADC is configured for hardware triggers, disables hardware triggers. Otherwise, stops any in-progress scan started by software. Implements adc_api_t::scanStop().

Return values

FSP_SUCCESS	Scan stopped or hardware triggers disabled successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_SDADC_StatusGet()

fsp_err_t R_SDADC_StatusGet (adc_ctrl_t * *p_ctrl*, adc_status_t * *p_status*)

Returns the status of a scan started by software, including calibration scans. It is not possible to determine the status of a scan started by a hardware trigger. Implements adc_api_t::scanStatusGet().

Return values

FSP_SUCCESS	No software scan or calibration is in progress.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_SDADC_Read()

fsp_err_t R_SDADC_Read (adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint16_t *const *p_data*)

Reads the most recent conversion result from a channel. Truncates 24-bit results to the upper 16 bits. Implements adc_api_t::read().

Note

The result stored in p_data is signed when the SDADC channel is configured in differential mode.
Do not use this API if the conversion end interrupt (SDADC0_ADI) is used to trigger the DTC unless the interrupt mode is set to TRANSFER_IRQ_EACH.

Return values

FSP_SUCCESS	Conversion result in p_data.
FSP_ERR_ASSERTION	An input pointer was NULL or an input parameter was invalid.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_SDADC_Read32()

fsp_err_t R_SDADC_Read32 (adc_ctrl_t * *p_ctrl*, adc_channel_t const *reg_id*, uint32_t *const *p_data*)

Reads the most recent conversion result from a channel. Implements adc_api_t::read32().

Note

The result stored in p_data is signed when the SDADC channel is configured in differential mode. When the SDADC is configured for 24-bit resolution and right alignment, the sign bit is bit 23, and the upper 8 bits are 0. When the SDADC is configured for 16-bit resolution and right alignment, the sign bit is bit 15, and the upper 16 bits are 0.
Do not use this API if the conversion end interrupt (SDADC0_ADI) is used to trigger the DTC unless the interrupt mode is set to TRANSFER_IRQ_EACH.

Return values

FSP_SUCCESS	Conversion result in p_data.
FSP_ERR_ASSERTION	An input pointer was NULL or an input parameter was invalid.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_SDADC_OffsetSet()

fsp_err_t R_SDADC_OffsetSet (adc_ctrl_t *const *p_ctrl*, adc_channel_t const *reg_id*, int32_t const *offset*)

Sets the offset. Offset is applied after stage 1 of the input channel. Offset can only be applied when the channel is configured for differential input. Implements adc_api_t::offsetSet().

Note: The offset is cleared if adc_api_t::calibrate() is called. The offset can be re-applied if necessary after the the callback with event ADC_EVENT_CALIBRATION_COMPLETE is called.

Parameters

[in]	p_ctrl	See p_instance_ctrl in adc_api_t::offsetSet().
[in]	reg_id	See reg_id in adc_api_t::offsetSet().
[in]	offset	Must be between -15 and 15, offset (mV) = 10.9376 mV * offset_steps / stage 1 gain.

Return values

FSP_SUCCESS	Offset updated successfully.
FSP_ERR_ASSERTION	An input pointer was NULL or an input parameter was invalid.
FSP_ERR_IN_USE	A conversion or calibration is in progress.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_SDADC_Calibrate()

fsp_err_t R_SDADC_Calibrate (adc_ctrl_t *const *p_ctrl*, void const * *p_extend*)

Requires sdadc_calibrate_args_t passed to p_extend. Calibrates the specified channel. Calibration is not required or supported for single-ended mode. Calibration must be completed for differential mode before using the SDADC. A callback with the event ADC_EVENT_CALIBRATION_COMPLETE is called when calibration completes. Implements adc_api_t::calibrate().

During external offset calibration, apply a differential voltage of 0 to ANSDnP - ANSDnN, where n is the input channel and ANSDnP is OPAMP0 for channel 4 and ANSDnN is OPAMP1 for channel 4. Complete external offset calibration before external gain calibration.

During external gain calibration apply a voltage between 0.4 V / total_gain and 0.8 V / total_gain. The differential voltage applied during calibration is corrected to a conversion result of 0x7FFFFF, which is the maximum possible positive differential measurement.

This function clears the offset value. If offset is required after calibration, it must be reapplied after calibration is complete using adc_api_t::offsetSet.

Return values

FSP_SUCCESS	Calibration began successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_IN_USE	A conversion or calibration is in progress.
FSP_ERR_NOT_OPEN	Instance control block is not open.

◆ R_SDADC_Close()

fsp_err_t R_SDADC_Close (adc_ctrl_t * *p_ctrl*)

Stops any scan in progress, disables interrupts, and powers down the SDADC peripheral. Implements adc_api_t::close().

Note

This function delays at least 3 us as required by the SDADC24 stop procedure.

Return values

FSP_SUCCESS	Instance control block closed successfully.
FSP_ERR_ASSERTION	An input pointer was NULL.
FSP_ERR_NOT_OPEN	Instance control block is not open.

4.2.57 SD/MMC Host Interface (r_sdhi)

Modules

Functions

fsp_err_t	R_SDHI_Open (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_cfg_t const *const p_cfg)
fsp_err_t	R_SDHI_MediaInit (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_device_t *const p_device)
fsp_err_t	R_SDHI_Read (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t	R_SDHI_Write (sdmmc_ctrl_t *const p_api_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t	R_SDHI_ReadIo (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)
fsp_err_t	R_SDHI_WriteIo (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)
fsp_err_t	R_SDHI_ReadIoExt (sdmmc_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
fsp_err_t	R_SDHI_WriteIoExt (sdmmc_ctrl_t *const p_api_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
fsp_err_t	R_SDHI_IoIntEnable (sdmmc_ctrl_t *const p_api_ctrl, bool enable)
fsp_err_t	R_SDHI_StatusGet (sdmmc_ctrl_t *const p_api_ctrl, sdmmc_status_t *const p_status)
fsp_err_t	R_SDHI_Erase (sdmmc_ctrl_t *const p_api_ctrl, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t	R_SDHI_CallbackSet (sdmmc_ctrl_t *const p_api_ctrl, void(*p_callback)(sdmmc_callback_args_t *), void const *const p_context, sdmmc_callback_args_t *const p_callback_memory)
fsp_err_t	R_SDHI_Close (sdmmc_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the SD/MMC Host Interface (SDHI) peripheral on RA MCUs. This module implements the

SD/MMC Interface.

Overview

Features

- Supports the following memory devices: SDSC (SD Standard Capacity), SDHC (SD High Capacity), SDXC (SD Extended Capacity) and eMMC (embedded Multi Media Card)
  - Supports reading, writing and erasing SD memory devices
  - Supports 1, 4 or 8-bit data bus (8-bit bus is supported for eMMC only)
  - Supports detection of device write protection (SD cards only)
  - Supports high speed mode
- Automatically configures the clock to the maximum clock rate supported by both host (MCU) and device
- Supports hardware acceleration using DMAC or DTC
- Supports callback notification when an operation completes or an error occurs

Configuration

Build Time Configurations for r_sdhi

The following build time configurations are defined in fsp_cfg/r_sdhi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Unaligned Access Support	• Disabled • Enabled	Enabled	If enabled, code for supporting buffers that are not aligned on a 4-byte boundary is included in the build. Only disable this if all buffers passed to the driver are 4-byte aligned.
SD Support	• Disabled • Enabled	Enabled	If selected code for SD card support is included in the build.
eMMC Support	• Disabled • Enabled	Disabled	If selected code for eMMC device support is included in the build.

Configurations for Storage > SD/MMC (r_sdhi)

This module can be added to the Stacks tab via New Stack > Storage > SD/MMC (r_sdhi). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description

Name	Name must be a valid C symbol	g_sdmmc0	Module name.
Channel	Value must be a non-negative integer	0	Select the channel.
Bus Width	MCU Specific Options		Select the bus width.
Block Size	Value must be an integer between 1 and 512	512	Select the media block size. Must be 512 for SD cards or eMMC devices. Must be 1-512 for SDIO.
Card Detection	• Not Used • CD Pin	CD Pin	Select the card detection method.
Write Protection	• Not Used • WP Pin	WP Pin	Select whether or not to use the write protect pin. Select Not Used if the MCU or device does not have a write protect pin.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).
Access Interrupt Priority	MCU Specific Options		Select the access interrupt priority.
Card Interrupt Priority	MCU Specific Options		Select the card interrupt priority.
DTC Interrupt Priority	MCU Specific Options		Select the DTC interrupt priority.

Interrupt Configurations:

The following interrupts are required to use the r_sdhi module:

Using SD/MMC with DTC:

- Access Interrupt
- DTC Interrupt

Using SD/MMC with DMAC:

- Access Interrupt
- DMAC Interrupt (in DMAC instance)

The Card interrupt is optional and only available on MCU packages that have the SDnCD pin (n = channel number).

Clock Configuration

The SDMMC MCU peripheral (SDHI) uses the PCLKA for its clock source. The SDMMC driver selects the optimal built-in divider based on the PCLKA frequency and the maximum clock rate allowed by the device obtained at media initialization.

Pin Configuration

The SDMMC driver supports the following pins (n = channel number):

- SDnCLK
- SDnCMD
- SDnDAT0
- SDnDAT1
- SDnDAT2
- SDnDAT3
- SDnDAT4 (not available on all MCUs)
- SDnDAT5 (not available on all MCUs)
- SDnDAT6 (not available on all MCUs)
- SDnDAT7 (not available on all MCUs)
- SDnCD (not available on all MCUs)
- SDnWP

The drive capacity for each pin should be set to "Medium" or "High" for most hardware designs. This can be configured in the **Pins** tab of the RA Configuration editor by selecting the pin under Pin Selection -> Ports.

Usage Notes

Card Detection

When Card Detection is configured to "CD Pin" in the RA Configuration editor, interrupt flags are cleared and card detection is enabled during R_SDHI_Open().

R_SDHI_StatusGet() can be called to retrieve the current status of the card (including whether a card is present). If the Card Interrupt Priority is enabled, a callback is called when a card is inserted or removed.

If a card is removed and reinserted, R_SDHI_MediaInit() must be called before reading from the card or writing to the card.

Note

R_SDHI_StatusGet() should be used to initially determine the card state after opening the interface.

DMA Request Interrupt Priority

When data transfers are not 4-byte aligned or not a multiple of 4 bytes, a software copy of the block size (up to 512 bytes) is done in the DMA Request interrupt. This blocks all other interrupts that are a lower or equal priority to the access interrupt until the software copy is complete.

Timing Notes for R_SDHI_MediaInit

The R_SDHI_MediaInit() API completes the entire device identification and configuration process. This involves several command-response cycles at a bus width of 1 bit and a bus speed of 400 kHz or less.

Limitations

Developers should be aware of the following limitations when using the SDHI:

Blocking Calls

The following functions block execution until the response is received for at least one command:

- R_SDHI_MediaInit
- R_SDHI_Erase

Once the function returns the status of the operation can be determined via R_SDHI_StatusGet or through receipt of a callback.

Note

Due to the variability in clocking configurations it is recommended to determine blocking delays experimentally on the target system.

Data Alignment and Size

Data transfers should be 4-byte aligned and a multiple of 4 bytes in size whenever possible. This recommendation applies to the read(), write(), readIoExt(), and writeIoExt() APIs. When data transfers are 4-byte aligned and a multiple of 4-bytes, the r_sdhi driver is zero copy and takes full advantage of hardware acceleration by the DMAC or DTC. When data transfers are not 4-byte aligned or not a multiple of 4 bytes an extra CPU interrupt is required for each block transferred and a software copy is used to move data to the destination buffer.

Examples

Basic Example

This is a basic example of minimal use of the r_sdhi in an application.

```
uint8_t g_dest[SDHI_MAX_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t g_src[SDHI_MAX_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint32_t g_transfer_complete = 0;
void r_sdhi_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < SDHI_MAX_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the SDHI driver. */
 fsp_err_t err = R_SDHI_Open(&g_sdmmc0_ctrl, &g_sdmmc0_cfg);
```

```
    assert(FSP_SUCCESS == err);
 /* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card. This should not be done until the card is plugged in for
SD devices. */
    err = R_SDHI_MediaInit(&g_sdmmc0_ctrl, NULL);
    assert(FSP_SUCCESS == err);
    err = R_SDHI_Write(&g_sdmmc0_ctrl, g_src, 3, 1);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
 /* Wait for transfer. */
    }
    err = R_SDHI_Read(&g_sdmmc0_ctrl, g_dest, 3, 1);
    assert(FSP_SUCCESS == err);
 while (!g_transfer_complete)
    {
 /* Wait for transfer. */
    }
}
/* The callback is called when a transfer completes. */
void r_sdhi_example_callback (sdmmc_callback_args_t * p_args)
{
 if (SDMMC_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = 1;
    }
}
```

Card Detection Example

This is an example of using SDHI when the card may not be plugged in. The card detection interrupt must be enabled to use this example.

```
bool g_card_inserted = false;
void r_sdhi_card_detect_example (void)
{
 /* Open the SDHI driver. This enables the card detection interrupt. */
 fsp_err_t err = R_SDHI_Open(&g_sdmmc0_ctrl, &g_sdmmc0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Check if card is inserted. */
 sdmmc_status_t status;
    err = R_SDHI_StatusGet(&g_sdmmc0_ctrl, &status);
    assert(FSP_SUCCESS == err);
 if (!status.card_inserted)
    {
 while (!g_card_inserted)
      {
 /* Wait for a card insertion interrupt. */
      }
    }
 /* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card after card insertion is detected. */
    err = R_SDHI_MediaInit(&g_sdmmc0_ctrl, NULL);
    assert(FSP_SUCCESS == err);
}
/* The callback is called when a card detection event occurs if the card detection
interrupt is enabled. */
void r_sdhi_card_detect_example_callback (sdmmc_callback_args_t * p_args)
{
 if (SDMMC_EVENT_CARD_INSERTED == p_args->event)
    {
```

```
        g_card_inserted = true;
    }
 if (SDMMC_EVENT_CARD_REMOVED == p_args->event)
    {
        g_card_inserted = false;
    }
}
```

Function Documentation

◆ R_SDHI_Open()

fsp_err_t R_SDHI_Open (sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_cfg_t const *const *p_cfg*)

Opens the driver. Resets SDHI, and enables card detection interrupts if card detection is enabled. R_SDHI_MediaInit must be called after this function before any other functions can be used.

Implements sdmmc_api_t::open().

Example:

```
/* Open the SDHI driver. */
fsp_err_t err = R_SDHI_Open(&g_sdmmc0_ctrl, &g_sdmmc0_cfg);
```

Return values

FSP_SUCCESS	Module is now open.
FSP_ERR_ASSERTION	Null Pointer or block size is not in the valid range of 1-512. Block size must be 512 bytes for SD cards and eMMC devices. It is configurable for SDIO only.
FSP_ERR_ALREADY_OPEN	Driver has already been opened with this instance of the control structure.
FSP_ERR_IRQ_BSP_DISABLED	Access interrupt is not enabled.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Requested channel does not exist on this MCU.

◆ R_SDHI_MediaInit()

fsp_err_t R_SDHI_MediaInit (sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_device_t *const *p_device*)

Initializes the SDHI hardware and completes identification and configuration for the SD or eMMC device. This procedure requires several sequential commands. This function blocks until all identification and configuration commands are complete.

Implements sdmmc_api_t::mediaInit().

Example:

```
/* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
 * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
/* Initialize the SD card. This should not be done until the card is plugged in for
SD devices. */
    err = R_SDHI_MediaInit(&g_sdmmc0_ctrl, NULL);
```

Return values

FSP_SUCCESS	Module is now ready for read/write access.
FSP_ERR_ASSERTION	Null Pointer or block size is not in the valid range of 1-512. Block size must be 512 bytes for SD cards and eMMC devices. It is configurable for SDIO only.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_CARD_INIT_FAILED	Device was not identified as an SD card, eMMC device, or SDIO card.
FSP_ERR_RESPONSE	Device did not respond or responded with an error.
FSP_ERR_DEVICE_BUSY	Device is holding DAT0 low (device is busy) or another operation is ongoing.

◆ R_SDHI_Read()

fsp_err_t R_SDHI_Read (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *start_sector*, uint32_t const *sector_count*)

Reads data from an SD or eMMC device. Up to 0x10000 sectors can be read at a time. Implements sdmmc_api_t::read().

A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the read data is available.

Example:

```
err = R_SDHI_Read(&g_sdmmc0_ctrl, g_dest, 3, 1);
```

Return values

FSP_SUCCESS	Data read successfully.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_CARD_NOT_INITIALIZED	Card was unplugged.
FSP_ERR_DEVICE_BUSY	Driver is busy with a previous operation.

◆ R_SDHI_Write()

fsp_err_t R_SDHI_Write (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_source*, uint32_t const *start_sector*, uint32_t const *sector_count*)

Writes data to an SD or eMMC device. Up to 0x10000 sectors can be written at a time. Implements sdmmc_api_t::write().

A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the all data has been written and the device is no longer holding DAT0 low to indicate it is busy.

Example:

```
err = R_SDHI_Write(&g_sdmmc0_ctrl, g_src, 3, 1);
```

Return values

FSP_SUCCESS	Card write finished successfully.
FSP_ERR_ASSERTION	Handle or Source address is NULL.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_CARD_NOT_INITIALIZED	Card was unplugged.
FSP_ERR_DEVICE_BUSY	Driver is busy with a previous operation.
FSP_ERR_CARD_WRITE_PROTECTED	SD card is Write Protected.
FSP_ERR_WRITE_FAILED	Write operation failed.

◆ R_SDHI_ReadIo()

fsp_err_t R_SDHI_ReadIo (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t const *function*, uint32_t const *address*)

The Read function reads a one byte register from an SDIO card. Implements sdmmc_api_t::readIo().

This function blocks until the command is sent and the response is received. p_data contains the register value read when this function returns.

Return values

FSP_SUCCESS	Data read successfully.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_CARD_NOT_INITIALIZED	Card was unplugged.
FSP_ERR_UNSUPPORTED	SDIO support disabled in SDHI_CFG_SDIO_SUPPORT_ENABLE.
FSP_ERR_RESPONSE	Device did not respond or responded with an error.
FSP_ERR_DEVICE_BUSY	Device is holding DAT0 low (device is busy) or another operation is ongoing.

◆ R_SDHI_WriteIo()

fsp_err_t R_SDHI_WriteIo (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_data*, uint32_t const *function*, uint32_t const *address*, sdmmc_io_write_mode_t const *read_after_write*)

Writes a one byte register to an SDIO card. Implements sdmmc_api_t::writeIo().

This function blocks until the command is sent and the response is received. The register has been written when this function returns. If read_after_write is true, p_data contains the register value read when this function returns.

Return values

FSP_SUCCESS	Card write finished successfully.
FSP_ERR_ASSERTION	Handle or Source address is NULL.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_CARD_NOT_INITIALIZED	Card was unplugged.
FSP_ERR_WRITE_FAILED	Write operation failed.
FSP_ERR_UNSUPPORTED	SDIO support disabled in SDHI_CFG_SDIO_SUPPORT_ENABLE.
FSP_ERR_RESPONSE	Device did not respond or responded with an error.
FSP_ERR_DEVICE_BUSY	Device is holding DAT0 low (device is busy) or another operation is ongoing.

◆ R_SDHI_ReadIoExt()

fsp_err_t R_SDHI_ReadIoExt (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *function*, uint32_t const *address*, uint32_t *const *count*, sdmmc_io_transfer_mode_t *transfer_mode*, sdmmc_io_address_mode_t *address_mode*)

Reads data from an SDIO card function. Implements sdmmc_api_t::readIoExt().

This function blocks until the command is sent and the response is received. A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the read data is available.

Return values

FSP_SUCCESS	Data read successfully.
FSP_ERR_ASSERTION	NULL pointer, or count is not in the valid range of 1-512 for byte mode or 1-511 for block mode.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_CARD_NOT_INITIALIZED	Card was unplugged.
FSP_ERR_DEVICE_BUSY	Driver is busy with a previous operation.
FSP_ERR_UNSUPPORTED	SDIO support disabled in SDHI_CFG_SDIO_SUPPORT_ENABLE.

◆ R_SDHI_WriteIoExt()

fsp_err_t R_SDHI_WriteIoExt (sdmmc_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_source*, uint32_t const *function*, uint32_t const *address*, uint32_t const *count*, sdmmc_io_transfer_mode_t *transfer_mode*, sdmmc_io_address_mode_t *address_mode*)

Writes data to an SDIO card function. Implements sdmmc_api_t::writeIoExt().

This function blocks until the command is sent and the response is received. A callback with the event SDMMC_EVENT_TRANSFER_COMPLETE is called when the all data has been written.

Return values

FSP_SUCCESS	Card write finished successfully.
FSP_ERR_ASSERTION	NULL pointer, or count is not in the valid range of 1-512 for byte mode or 1-511 for block mode.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_CARD_NOT_INITIALIZED	Card was unplugged.
FSP_ERR_DEVICE_BUSY	Driver is busy with a previous operation.
FSP_ERR_WRITE_FAILED	Write operation failed.
FSP_ERR_UNSUPPORTED	SDIO support disabled in SDHI_CFG_SDIO_SUPPORT_ENABLE.

◆ R_SDHI_IoIntEnable()

fsp_err_t R_SDHI_IoIntEnable (sdmmc_ctrl_t *const *p_api_ctrl*, bool *enable*)

Enables or disables the SDIO Interrupt. Implements sdmmc_api_t::ioIntEnable().

Return values

FSP_SUCCESS	Card enabled or disabled SDIO interrupts successfully.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_DEVICE_BUSY	Driver is busy with a previous operation.
FSP_ERR_UNSUPPORTED	SDIO support disabled in SDHI_CFG_SDIO_SUPPORT_ENABLE.

◆ R_SDHI_StatusGet()

fsp_err_t R_SDHI_StatusGet (sdmmc_ctrl_t *const *p_api_ctrl*, sdmmc_status_t *const *p_status*)

Provides driver status. Implements sdmmc_api_t::statusGet().

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_NOT_OPEN	Driver has not been initialized.

◆ R_SDHI_Erase()

fsp_err_t R_SDHI_Erase (sdmmc_ctrl_t *const *p_api_ctrl*, uint32_t const *start_sector*, uint32_t const *sector_count*)

Erases sectors of an SD card or eMMC device. Implements sdmmc_api_t::erase().

This function blocks until the erase command is sent. Poll the status to determine when erase is complete.

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	A required pointer is NULL or an argument is invalid.
FSP_ERR_NOT_OPEN	Driver has not been initialized.
FSP_ERR_CARD_NOT_INITIALIZED	Card was unplugged.
FSP_ERR_CARD_WRITE_PROTECTED	SD card is Write Protected.
FSP_ERR_RESPONSE	Device did not respond or responded with an error.
FSP_ERR_DEVICE_BUSY	Device is holding DAT0 low (device is busy) or another operation is ongoing.

◆ R_SDHI_CallbackSet()

fsp_err_t R_SDHI_CallbackSet (sdmmc_ctrl_t *const *p_api_ctrl*, void(*)(sdmmc_callback_args_t *) *p_callback*, void const *const *p_context*, sdmmc_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements sdmmc_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ R_SDHI_Close()

fsp_err_t R_SDHI_Close (sdmmc_ctrl_t *const *p_api_ctrl*)

Closes an open SD/MMC device. Implements sdmmc_api_t::close().

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Driver has not been initialized.

4.2.58 Segment LCD Controller (r_slcdc)

Modules

Functions

fsp_err_t	R_SLCDC_Open (slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg)
fsp_err_t	R_SLCDC_Write (slcdc_ctrl_t *const p_ctrl, uint8_t const start_segment, uint8_t const *p_data, uint8_t const segment_count)
fsp_err_t	R_SLCDC_Modify (slcdc_ctrl_t *const p_ctrl, uint8_t const segment_number, uint8_t const data_mask, uint8_t const data)

fsp_err_t	R_SLCDC_Start (slcdc_ctrl_t *const p_ctrl)
fsp_err_t	R_SLCDC_Stop (slcdc_ctrl_t *const p_ctrl)
fsp_err_t	R_SLCDC_SetContrast (slcdc_ctrl_t *const p_ctrl, slcdc_contrast_t const contrast)
fsp_err_t	R_SLCDC_SetDisplayArea (slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area)
fsp_err_t	R_SLCDC_Close (slcdc_ctrl_t *const p_ctrl)

Detailed Description

Driver for the SLCDC peripheral on RA MCUs. This module implements the SLCDC Interface.

Overview

The segment LCD controller (SLCDC) utilizes two to four reference voltages to provide AC signals for driving traditional segment LCD panels. Depending on the LCD and MCU package, up to 272 segments can be driven. A built-in link to the RTC allows for up to 152 segments to switch between two patterns at regular intervals. An on-chip boost driver can be used to provide configurable reference voltages up to 5.25V allowing for simple contrast adjustment.

Features

The SLCDC module can perform the following functions:

- Initialize, start and stop the SLCDC
- Set and modify the output pattern
- Blink between two patterns based on a periodic RTC interrupt signal
- Adjust display contrast (only when using internal voltage boosting)

Configuration

Build Time Configurations for r_slcdc

The following build time configurations are defined in fsp_cfg/r_slcdc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Graphics > Segment LCD (r_slcdc)

This module can be added to the Stacks tab via New Stack > Graphics > Segment LCD (r_slcdc).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_slcdc0	Module Name
Clock > Source	• LOCO • SOSC • MOSC • HOCO	HOCO	Select the clock source.
Clock > Divisor	Refer to the RA Configuration tool for available options.	(HOCO/MOSC) 16384	Select the clock divisor.
Output > Bias method	• 1/2 bias • 1/3 bias • 1/4 bias	1/2 bias	Select the bias method. This determines the number of voltage levels used to create the waveforms.
Output > Timeslice	• Static • 2-slice • 3-slice • 4-slice • 8-slice	Static	Select the LCD time slice. The number of slices should match the number of common (COM) pins for your LCD panel.
Output > Waveform	• Waveform A • Waveform B	Waveform A	Select the LCD waveform.
Output > Drive method	• External resistance division • Internal voltage boosting • Capacitor split	External resistance division	Select the LCD drive method.
Output > Default contrast	Refer to the RA Configuration tool for available options.	0	Select the default contrast level.

Valid Configurations

Though there are many setting combinations only a limited subset are supported by the SLCDC peripheral hardware:

Waveform	Slices	Bias	External Resistance	Internal Boost	Capacitor Split
A	8	1/4	Available	Available	—
A	4	1/3	Available	Available	Available
A	3	1/3	Available	Available	Available
A	3	1/2	Available	—	—
A	2	1/2	Available	—	—
A	Static	—	Available	—	—

B	8	1/4	Available	Available	Available
B	4	1/3	Available	Available	—

Clock Configuration

The SLCDC clock can be sourced from the main clock (MOSC), sub-clock (SOSC), HOCO or LOCO. Dividers of 4 to 1024 are available for SOSC/LOCO and 256 to 524288 for MOSC/HOCO. It is recommended to adjust the divisor such that the resulting clock provides a frame frequency of 32-128 Hz.

Note

Make sure your desired source clock is enabled and running before starting SLCDC output.
Do not set the segment LCD clock over 512 Hz when using internal boost or capacitor split modes.

Pin Configuration

This module controls a variety of pins necessary for segment LCD voltage generation and signal output:

Pin Name	Function	Notes
SEGn	Segment data output	Connect these signals to the segment pins of the LCD.
COMn	Common signal output	Connect these signals to the common pins of the LCD.
VLn	Voltage reference	These pins should be connected to passive components based on the selected drive method (see section 45.7 "Supplying LCD Drive Voltages VL1, VL2, VL3, and VL4" in the RA4M1 User's Manual (R01UH0887EJ0100)).
CAPH, CAPL	Drive voltage generator capacitor	Connect a nonpolar 0.47uF capacitor across these pins when using internal boost or capacitor split modes. This pin is not needed when using resistance division.

Interrupt Configuration

The SLCDC provides no interrupt signals.

Note

Blinking output timing is driven directly from the RTC periodic interrupt. Once the interrupt is enabled setting the display to SLCDC_DISP_BLINK will swap between A- and B-pattern each time it occurs. The ELC is not required for this functionality.

Usage Notes

Limitations

Developers should be aware of the following limitations when using the SLCDC:

- Different packages provide different numbers of segment pins. Check the User's Manual for your device to confirm availability and mapping of segment signals.
- When using internal boost mode a delay of 5ms is required between calling R_SLCDC_Open and R_SLCDC_Start to allow the boost circuit to charge.
- When using the internal boost or capacitor split method do not set the segment LCD clock higher than 512 Hz.

Examples

Basic Example

Below is a basic example of minimal use of the SLCDC in an application. The SLCDC driver is initialized, output is started and a pattern is written to the segment registers.

```
void slcdc_init (void)
{
 fsp_err_t err;
 /* Open SLCDC driver */
    err = R_SLCDC_Open(&g_slcdc_ctrl, &g_slcdc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* When using internal boost mode this delay is required to allow the boost circuit
to charge. See RA4M1 User's
  * Manual (R01UH0887EJ0100) 8.2.18 "Segment LCD Source Clock Control Register
(SLCDSCKCR)" for details. */
 R_BSP_SoftwareDelay(5, BSP_DELAY_UNITS_MILLISECONDS);
 /* Start SLCDC output */
    err = R_SLCDC_Start(&g_slcdc_ctrl);
    assert(FSP_SUCCESS == err);
 /* Write pattern to display */
    err = R_SLCDC_Write(&g_slcdc_ctrl, 0, segment_data, NUM_SEGMENTS);
    assert(FSP_SUCCESS == err);
}
```

Note

While the SLCDC is running, pattern data is constantly being output. No latching or buffering is required when writing or reading segment data.

Blinking Output

This example demonstrates how to set up blinking output using the RTC periodic interrupt. In this example it is assumed that the SLCDC has already been started.

```
void slcdc_blink (void)
{
 fsp_err_t err;
 /* Open RTC and set time/date */
    err = R_RTC_Open(&r_rtc_ctrl, &r_rtc_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
    err = R_RTC_CalendarTimeSet(&r_rtc_ctrl, &g_rtc_time);
    assert(FSP_SUCCESS == err);
 /* Set RTC periodic interrupt to 2 Hz (display blink cycle will be 1 Hz) */
    err = R_RTC_PeriodicIrqRateSet(&r_rtc_ctrl,
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND);
    assert(FSP_SUCCESS == err);
 /* Set display to blink */
    err = R_SLCDC_SetDisplayArea(&g_slcdc_ctrl, SLCDC_DISP_BLINK);
    assert(FSP_SUCCESS == err);
 /* Display will now continuously blink */
}
```

Data Structures

struct	slcdc_instance_ctrl_t

Data Structure Documentation

◆ slcdc_instance_ctrl_t

struct slcdc_instance_ctrl_t
SLCDC control block. DO NOT INITIALIZE. Initialization occurs when slcdc_api_t::open is called

Function Documentation

◆ R_SLCDC_Open()

fsp_err_t R_SLCDC_Open (slcdc_ctrl_t *const *p_ctrl*, slcdc_cfg_t const *const *p_cfg*)

Opens the SLCDC driver. Implements slcdc_api_t::open.

Return values

FSP_SUCCESS	Device was opened successfully.
FSP_ERR_ASSERTION	Pointer to the control block or the configuration structure is NULL.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_UNSUPPORTED	Invalid display mode.

◆ R_SLCDC_Write()

fsp_err_t R_SLCDC_Write (slcdc_ctrl_t *const *p_ctrl*, uint8_t const *start_segment*, uint8_t const * *p_data*, uint8_t const *segment_count*)

Writes a sequence of display data to the segment data registers. Implements slcdc_api_t::write.

Return values

FSP_SUCCESS	Data was written successfully.
FSP_ERR_ASSERTION	Pointer to the control block or data is NULL.
FSP_ERR_INVALID_ARGUMENT	Segment index is (or will be) out of range.
FSP_ERR_NOT_OPEN	Device is not opened or initialized.

◆ R_SLCDC_Modify()

fsp_err_t R_SLCDC_Modify (slcdc_ctrl_t *const *p_ctrl*, uint8_t const *segment*, uint8_t const *data*, uint8_t const *data_mask*)

Modifies a single segment register based on a mask and the desired data. Implements slcdc_api_t::modify.

Return values

FSP_SUCCESS	Device was opened successfully.
FSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
FSP_ERR_INVALID_ARGUMENT	Invalid parameter in the argument.
FSP_ERR_NOT_OPEN	Device is not opened or initialized

◆ R_SLCDC_Start()

fsp_err_t R_SLCDC_Start (slcdc_ctrl_t *const *p_ctrl*)

Starts output of LCD signals. Implements slcdc_api_t::start.

Return values

FSP_SUCCESS	Device was opened successfully.
FSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
FSP_ERR_NOT_OPEN	Device is not opened or initialized

◆ R_SLCDC_Stop()

fsp_err_t R_SLCDC_Stop (slcdc_ctrl_t *const *p_ctrl*)

Stops output of LCD signals. Implements slcdc_api_t::stop.

Return values

FSP_SUCCESS	Device was opened successfully.
FSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
FSP_ERR_NOT_OPEN	Device is not opened or initialized

◆ R_SLCDC_SetContrast()

fsp_err_t R_SLCDC_SetContrast (slcdc_ctrl_t *const *p_ctrl*, slcdc_contrast_t const *contrast*)

Sets contrast to the specified level. Implements slcdc_api_t::setContrast.

Note

Contrast can be adjusted when the SLCDC is operating in internal boost mode only. The range of values is 0-5 when 1/4 bias setting is used and 0-15 otherwise. See RA4M1 User's Manual (R01UH0887EJ0100) section 45.2.4 "LCD Boost Level Control Register (VLCD)" for voltage levels at each setting.

Return values

FSP_SUCCESS	Device was opened successfully.
FSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
FSP_ERR_NOT_OPEN	Device is not opened or initialized
FSP_ERR_UNSUPPORTED	Unsupported operation

◆ R_SLCDC_SetDisplayArea()

fsp_err_t R_SLCDC_SetDisplayArea (slcdc_ctrl_t *const *p_ctrl*, slcdc_display_area_t const *display_area*)
Sets output to Waveform A, Waveform B or blinking output. Implements slcdc_api_t::setDisplayArea .

Return values

FSP_SUCCESS	Device was opened successfully.
FSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
FSP_ERR_UNSUPPORTED	Pattern selection has no effect in 8-time-slice mode.
FSP_ERR_NOT_OPEN	Device is not opened or initialized.

◆ R_SLCDC_Close()

fsp_err_t R_SLCDC_Close (slcdc_ctrl_t *const *p_ctrl*)
Closes the SLCDC driver. Implements slcdc_api_t::close.

Return values

FSP_SUCCESS	Device was closed successfully.
FSP_ERR_ASSERTION	Pointer to the control block structure is NULL.
FSP_ERR_NOT_OPEN	Device is not opened or initialized

4.2.59 Serial Peripheral Interface (r_spi)

Modules

Functions

fsp_err_t	R_SPI_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg)
fsp_err_t	R_SPI_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SPI_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)

fsp_err_t	R_SPI_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SPI_Close (spi_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SPI_CalculateBitrate (uint32_t bitrate, rspck_div_setting_t *spck_div)
fsp_err_t	R_SPI_CallbackSet (spi_ctrl_t *const p_api_ctrl, void(*p_callback)(spi_callback_args_t *), void const *const p_context, spi_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the SPI peripheral on RA MCUs. This module implements the SPI Interface.

Overview

Features

- Standard SPI Modes
  - Master or Slave Mode
  - Clock Polarity (CPOL)
    - CPOL=0 SCLK is low when idle
    - CPOL=1 SCLK is high when idle
  - Clock Phase (CPHA)
    - CPHA=0 Data Sampled on the even edge of SCLK (Master Mode Only)
    - CPHA=1 Data Sampled on the odd edge of SCLK
  - MSB/LSB first
  - 8-Bit, 9-bit, 10-bit, 11-bit, 12-bit, 13-bit, 14-bit, 15-bit, 16-Bit, 20-bit, 24-bit, and 32-Bit data frames
    - Hardware endian swap in 16-Bit and 32-Bit mode
  - 3-Wire (clock synchronous) or 4-Wire (SPI) Mode
- Configurable bitrate
- Supports Full Duplex or Transmit Only Mode
- DTC Support
- Callback Events
  - Transfer Complete
  - RX Overflow Error (The SPI shift register is copied to the data register before previous data was read)
  - TX Underrun Error (No data to load into shift register for transmitting)
  - Parity Error (When parity is enabled and a parity error is detected)

Configuration

Build Time Configurations for r_spi

The following build time configurations are defined in fsp_cfg/r_spi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Enable Support for using DTC	• Enabled • Disabled	Enabled	If enabled, DTC instances will be included in the build for both transmission and reception.
Enable Transmitting from RXI Interrupt	• Enabled • Disabled	Disabled	If enabled, all operations will be handled from the RX (receive) interrupt. This setting only provides a performance boost when DTC is not used. In addition, Transmit Only mode is not supported when this configuration is enabled.

Configurations for Connectivity > SPI (r_spi)

This module can be added to the Stacks tab via New Stack > Connectivity > SPI (r_spi). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_spi0	Module name.
Channel	Value must be a non-negative integer	0	Select the SPI channel.
Receive Interrupt Priority	MCU Specific Options		Select the interrupt priority for all SPI interrupts.
Transmit Buffer Empty Interrupt Priority	MCU Specific Options		Select the interrupt priority for all SPI interrupts.
Transfer Complete Interrupt Priority	MCU Specific Options		Select the interrupt priority for all SPI interrupts.
Error Interrupt Priority	MCU Specific Options		Select the interrupt priority for all SPI interrupts.
Operating Mode	• Master • Slave	Master	Select the SPI operating mode.
Clock Phase	• Data sampling	Data sampling on odd	Select the clock edge

	on odd edge, data variation on even edge • Data sampling on even edge, data variation on odd edge	edge, data variation on even edge	to sample data.
Clock Polarity	• Low when idle • High when idle	Low when idle	Select clock level when idle.
Mode Fault Error	• Enable • Disable	Disable	Detect master/slave mode conflicts.
Bit Order	• MSB First • LSB First	MSB First	Select the data bit order.
Callback	Name must be a valid C symbol	spi_callback	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).
SPI Mode	• SPI Operation • Clock Synchronous Operation	Clock Synchronous Operation	Select the clock sync mode.
Full or Transmit Only Mode	• Full Duplex • Transmit Only	Full Duplex	Select Full Duplex or Transmit Only Mode.
Slave Select Polarity	• Active Low • Active High	Active Low	Select the slave select active level.
Select SSL(Slave Select)	• SSL0 • SSL1 • SSL2 • SSL3	SSL0	Select which slave to use.
MOSI Idle State	• MOSI Idle Value Fixing Disable • MOSI Idle Value Fixing Low • MOSI Idle Value Fixing High	MOSI Idle Value Fixing Disable	Select the MOSI idle level if MOSI idle is enabled.
Parity Mode	• Disabled • Odd • Even	Disabled	Select the parity mode if parity is enabled.
Byte Swapping	• Disable • Enable	Disable	Select the byte swap mode for 16/32-Bit Data Frames.
Bitrate	Value must be an integer greater than 0	16000000	Enter the desired bitrate, change the

			bitrate to a value supported by MCU. If the requested bitrate cannot be achieved, the settings with the largest possible value that is less than or equal to the requested bitrate is used. The theoretical bitrate is printed in a comment in the generated spi_extended_cfg_t structure.
Clock Delay	• 1 Clock • 2 Clocks • 3 Clocks • 4 Clocks • 5 Clocks • 6 Clocks • 7 Clocks • 8 Clocks	1 Clock	Configure the number of SPI clock cycles before each data frame.
SSL Negation Delay	• 1 Clock • 2 Clocks • 3 Clocks • 4 Clocks • 5 Clocks • 6 Clocks • 7 Clocks • 8 Clocks	1 Clock	Configure the number of SPI clock cycles after each data frame.
Next Access Delay	• 1 Clock • 2 Clocks • 3 Clocks • 4 Clocks • 5 Clocks • 6 Clocks • 7 Clocks • 8 Clocks	1 Clock	Configure the number of SPI clock cycles between each data frame.

Clock Configuration

The clock for this module is derived from the following peripheral clock for each MCU group:

MCU Group	Peripheral Clock
RA2A1	PCLKB
RA2E1	PCLKB
RA2E2	PCLKB
RA2L1	PCLKB
RA4E1	PCLKA

RA4M1	PCLKA
RA4M2	PCLKA
RA4M3	PCLKA
RA4W1	PCLKA
RA6E1	PCLKA
RA6M1	PCLKA
RA6M2	PCLKA
RA6M3	PCLKA
RA6M4	PCLKA
RA6M5	PCLKA
RA6T1	PCLKA
RA6T2	PCLKA

Pin Configuration

This module uses MOSI, MISO, RSPCK, and SSL pins to communicate with on board devices.

Note

At high bitrates, it might be nessecary to configure the pins with IOPORT_CFG_DRIVE_HIGH.

Usage Notes

Performance

At high bitrates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. In slave mode this could result in TX Underrun and RX Overflow errors.

In order to improve performance at high bitrates, it is recommended that the instance be configured to service transfers using the DTC.

Another way to improve performance is to transfer the data in 16/32 bit wide data frames when possible. A typical use-case where this is possible is when reading/writing to a block device.

Transmit From RXI Interrupt

After every data frame the SPI peripheral generates a transmit buffer empty interrupt and a receive buffer full interrupt. It is possible to configure the driver to handle transmit buffer empty interrupts in the receive buffer full isr. This only improves performance when the DTC is not being used.

Note

Configuring the module to use RX DTC instance without also providing a TX DTC instance results in an invalid configuration when RXI transmit is enabled.
Transmit Only mode is not supported when Transmit from RXI is enabled.

Clock Auto-Stopping

In master mode, if the Receive Buffer Full Interrupts are not handled fast enough, instead of generating a RX Overflow error, the last clock cycle will be stretched until the receive buffer is read.

Parity Mode

When parity mode is configured, the LSB of each data frame is used as a parity bit. When odd parity is selected, the LSB is set such that there are an odd number of ones in the data frame. When even parity is selected, the LSB is set such that there are an even number of ones in the data frame.

Limitations

Developers should be aware of the following limitations when using the SPI:

- In master mode, the driver will only configure 4-Wire mode if the device supports SSL Level Keeping (SSLKP bit in SPCMD0) and will return FSP_ERR_UNSUPPORTED if configured for 4-Wire mode on devices without SSL Level Keeping. Without SSL Level Keeping, the SSL pin is toggled after every data frame. In most cases this is not desirable behavior so it is recommended that the SSL pin be driven in software if SSL Level Keeping is not present on the device.
- In order to use CPHA=0 setting in slave mode, the master must toggle the SSL pin after every data frame (Even if the device supports SSL Level Keeping). Because of this hardware limitation, the module will return FSP_ERR_UNSUPPORTED when it is configured to use CPHA=0 setting in slave mode.
- The module does not support communicating with multiple slaves using different SSL pins. In order to achieve this, the module must either be closed and re-opened to change the SSL pin or drive SSL in software. It is recommended that SSL be driven in software when controlling multiple slave devices.
- The SPI peripheral has a minimum 3 SPI CLK delay between each data frame.
- The behavior for Byte Swap operation is not guaranteed for data frames other than 8-bit, 16-bit and 32bit.

Examples

Basic Example

This is a basic example of minimal use of the SPI in an application.

```
static volatile bool g_transfer_complete = false;
void spi_basic_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start a write/read transfer */
```

```
    err = R_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
}
static void r_spi_callback (spi_callback_args_t * p_args)
{
 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = true;
    }
}
```

Driving Software Slave Select Line

This is an example of communicating with multiple slave devices by asserting SSL in software.

```
void spi_software_ssl_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 /* Configure Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Configure Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Assert Slave Select Line 1 */
```

```
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err                = R_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer,
TRANSFER_SIZE, SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Wait for minimum time required between transfers. */
 R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
 /* Assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err                = R_SPI_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer,
TRANSFER_SIZE, SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
}
```

Configuring the SPI Clock Divider Registers

This example demonstrates how to set the SPI clock divisors at runtime.

```
void spi_bitrate_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_spi_cfg.p_extend = &g_spi_extended_cfg;
 /* Configure SPI Clock divider to achieve largest bitrate less than or equal to the
desired bitrate. */
    err = R_SPI_CalculateBitrate(BITRATE, &(g_spi_extended_cfg.spck_div));
    assert(FSP_SUCCESS == err);
 /* Initialize the SPI module. */
    err = R_SPI_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	rspck_div_setting_t
struct	spi_extended_cfg_t
struct	spi_instance_ctrl_t

Enumerations

enum	spi_ssl_mode_t
enum	spi_communication_t
enum	spi_ssl_polarity_t
enum	spi_ssl_select_t
enum	spi_mosi_idle_value_fixing_t
enum	spi_parity_t
enum	spi_byte_swap_t
enum	spi_delay_count_t

Data Structure Documentation

◆ rspck_div_setting_t

struct rspck_div_setting_t		
SPI Clock Divider settings.		
Data Fields		
uint8_t	spbr	SPBR register setting.
uint8_t	brdv: 2	BRDV setting in SPCMD0.

◆ spi_extended_cfg_t

struct spi_extended_cfg_t		
Extended SPI interface configuration		
Data Fields		
spi_ssl_mode_t	spi_clksyn	Select spi or clock syn mode operation.
spi_communication_t	spi_comm	Select full-duplex or transmit-only communication.
spi_ssl_polarity_t	ssl_polarity	Select SSLn signal polarity.
spi_ssl_select_t	ssl_select	Select which slave to use: 0-SSL0, 1-SSL1, 2-SSL2, 3-SSL3.
spi_mosi_idle_value_fixing_t	mosi_idle	Select MOSI idle fixed value and selection.
spi_parity_t	parity	Select parity and enable/disable parity.
spi_byte_swap_t	byte_swap	Select byte swap mode.
rspck_div_setting_t	spck_div	Register values for configuring the SPI Clock Divider.
spi_delay_count_t	spck_delay	SPI Clock Delay Register Setting.
spi_delay_count_t	ssl_negation_delay	SPI Slave Select Negation Delay Register Setting.
spi_delay_count_t	next_access_delay	SPI Next-Access Delay Register Setting.

◆ spi_instance_ctrl_t

struct spi_instance_ctrl_t	
Channel control block. DO NOT INITIALIZE. Initialization occurs when spi_api_t::open is called.	
Data Fields	
uint32_t	open
	Indicates whether the open() API has been successfully called.

spi_cfg_t const *	p_cfg
	Pointer to instance configuration.
R_SPI0_Type *	p_regs
	Base register for this channel.
void const *	p_tx_data
	Buffer to transmit.
void *	p_rx_data
	Buffer to receive.
uint32_t	tx_count
	Number of Data Frames to transfer (8-bit, 16-bit, 32-bit)
uint32_t	rx_count
	Number of Data Frames to transfer (8-bit, 16-bit, 32-bit)
uint32_t	count
	Number of Data Frames to transfer (8-bit, 16-bit, 32-bit)
spi_bit_width_t	bit_width
	Bits per Data frame (8-bit, 16-bit, 32-bit)

Enumeration Type Documentation

◆ spi_ssl_mode_t

enum spi_ssl_mode_t	
3-Wire or 4-Wire mode.	
Enumerator	
SPI_SSL_MODE_SPI	SPI operation (4-wire method)
SPI_SSL_MODE_CLK_SYN	Clock Synchronous operation (3-wire method)

◆ spi_communication_t

enum spi_communication_t	
Transmit Only (Half Duplex), or Full Duplex.	
Enumerator	
SPI_COMMUNICATION_FULL_DUPLEX	Full-Duplex synchronous serial communication.
SPI_COMMUNICATION_TRANSMIT_ONLY	Transit only serial communication.

◆ spi_ssl_polarity_t

enum spi_ssl_polarity_t	
Slave Select Polarity.	
Enumerator	
SPI_SSLP_LOW	SSLP signal polarity active low.
SPI_SSLP_HIGH	SSLP signal polarity active high.

◆ spi_ssl_select_t

enum spi_ssl_select_t	
The Slave Select Line	
Enumerator	
SPI_SSL_SELECT_SSL0	Select SSL0.
SPI_SSL_SELECT_SSL1	Select SSL1.
SPI_SSL_SELECT_SSL2	Select SSL2.
SPI_SSL_SELECT_SSL3	Select SSL3.

◆ spi_mosi_idle_value_fixing_t

enum spi_mosi_idle_value_fixing_t	
MOSI Idle Behavior.	
Enumerator	
SPI_MOSI_IDLE_VALUE_FIXING_DISABLE	MOSI output value=value set in MOIFV bit.
SPI_MOSI_IDLE_VALUE_FIXING_LOW	MOSIn level low during MOSI idling.
SPI_MOSI_IDLE_VALUE_FIXING_HIGH	MOSIn level high during MOSI idling.

◆ spi_parity_t

enum spi_parity_t	
Parity Mode	
Enumerator	
SPI_PARITY_MODE_DISABLE	Disable parity.
SPI_PARITY_MODE_ODD	Select even parity.
SPI_PARITY_MODE_EVEN	Select odd parity.

◆ spi_byte_swap_t

enum spi_byte_swap_t	
Byte Swapping Enable/Disable.	
Enumerator	
SPI_BYTE_SWAP_DISABLE	Disable Byte swapping for 16/32-Bit transfers.
SPI_BYTE_SWAP_ENABLE	Enable Byte swapping for 16/32-Bit transfers.

◆ spi_delay_count_t

enum spi_delay_count_t	
Delay count for SPI delay settings.	
Enumerator	
SPI_DELAY_COUNT_1	Set RSPCK delay count to 1 RSPCK.
SPI_DELAY_COUNT_2	Set RSPCK delay count to 2 RSPCK.
SPI_DELAY_COUNT_3	Set RSPCK delay count to 3 RSPCK.
SPI_DELAY_COUNT_4	Set RSPCK delay count to 4 RSPCK.
SPI_DELAY_COUNT_5	Set RSPCK delay count to 5 RSPCK.
SPI_DELAY_COUNT_6	Set RSPCK delay count to 6 RSPCK.
SPI_DELAY_COUNT_7	Set RSPCK delay count to 7 RSPCK.
SPI_DELAY_COUNT_8	Set RSPCK delay count to 8 RSPCK.

Function Documentation

◆ R_SPI_Open()

fsp_err_t R_SPI_Open (spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg*)

This functions initializes a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Configures the pperipheral registers acording to the configuration.
- Initialize the control structure for use in other SPI Interface functions.

Return values

FSP_SUCCESS	Channel initialized successfully.
FSP_ERR_ALREADY_OPEN	Instance was already initialized.
FSP_ERR_ASSERTION	An invalid argument was given in the configuration structure.
FSP_ERR_UNSUPPORTED	A requested setting is not possible on this device with the current build configuration.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel number is invalid.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls: transfer_api_t::open

Note

This function is reentrant.

◆ R_SPI_Read()

fsp_err_t R_SPI_Read (spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

This function receives data from a SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI read operation.

Return values

FSP_SUCCESS	Read operation successfully completed.
FSP_ERR_ASSERTION	NULL pointer to control or destination parameters or transfer length is zero.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open channel first.
FSP_ERR_IN_USE	A transfer is already in progress.

◆ R_SPI_Write()

fsp_err_t R_SPI_Write (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

This function transmits data to a SPI device using the TX Only Communications Operation Mode. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI write operation.

Return values

FSP_SUCCESS	Write operation successfully completed.
FSP_ERR_ASSERTION	NULL pointer to control or source parameters or transfer length is zero.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_IN_USE	A transfer is already in progress.

◆ R_SPI_WriteRead()

fsp_err_t R_SPI_WriteRead (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

This function simultaneously transmits and receive data. Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI writeRead operation.

Return values

FSP_SUCCESS	Write operation successfully completed.
FSP_ERR_ASSERTION	NULL pointer to control, source or destination parameters or transfer length is zero.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_IN_USE	A transfer is already in progress.

◆ R_SPI_Close()

fsp_err_t R_SPI_Close (spi_ctrl_t *const *p_api_ctrl*)

This function manages the closing of a channel by the following task. Implements spi_api_t::close.

Disables SPI operations by disabling the SPI bus.

- Disables the SPI peripheral.
- Disables all the associated interrupts.
- Update control structure so it will not work with SPI Interface functions.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	A required pointer argument is NULL.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

◆ R_SPI_CalculateBitrate()

fsp_err_t R_SPI_CalculateBitrate (uint32_t *bitrate*, rspck_div_setting_t * *spck_div*)

Calculates the SPBR register value and the BRDV bits for a desired bitrate. If the desired bitrate is faster than the maximum bitrate, than the bitrate is set to the maximum bitrate. If the desired bitrate is slower than the minimum bitrate, than an error is returned.

Parameters

[in]	bitrate	Desired bitrate
[out]	spck_div	Memory location to store bitrate register settings.

Return values

FSP_SUCCESS	Valid spbr and brdv values were calculated
FSP_ERR_UNSUPPORTED	Bitrate is not achievable

◆ R_SPI_CallbackSet()

fsp_err_t R_SPI_CallbackSet (spi_ctrl_t *const *p_api_ctrl*, void(*)(spi_callback_args_t *) *p_callback*, void const *const *p_context*, spi_callback_args_t *const *p_callback_memory*)
Updates the user callback and has option of providing memory for callback structure. Implements spi_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.60 Serial Peripheral Interface (r_spi_b)

Modules

Functions

fsp_err_t	R_SPI_B_Open (spi_ctrl_t *p_api_ctrl, spi_cfg_t const *const p_cfg)
fsp_err_t	R_SPI_B_Read (spi_ctrl_t *const p_api_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SPI_B_Write (spi_ctrl_t *const p_api_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SPI_B_WriteRead (spi_ctrl_t *const p_api_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t	R_SPI_B_Close (spi_ctrl_t *const p_api_ctrl)
fsp_err_t	R_SPI_B_CalculateBitrate (uint32_t bitrate, spi_b_clock_source_t clock_source, rspck_div_setting_t *spck_div)
fsp_err_t	R_SPI_B_CallbackSet (spi_ctrl_t *const p_api_ctrl, void(*p_callback)(spi_callback_args_t *), void const *const p_context, spi_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the SPI peripheral on RA MCUs. This module implements the SPI Interface.

Overview

Features

- Standard SPI Modes
  - Master or Slave Mode
  - Clock Polarity (CPOL)
    - CPOL=0 SCLK is low when idle
    - CPOL=1 SCLK is high when idle
  - Clock Phase (CPHA)
    - CPHA=0 Data Sampled on the even edge of SCLK (Master Mode Only)
    - CPHA=1 Data Sampled on the odd edge of SCLK
  - MSB/LSB first
  - 8- to 32-Bit data frames
    - Hardware endian swap in 16-Bit and 32-Bit mode
  - 3-Wire (clock synchronous) or 4-Wire (SPI) Mode
- Configurable bitrate
- Supports Full Duplex or Transmit Only Mode
- DTC Support
- Callback Events
  - Transfer Complete
  - RX Overflow Error (The SPI shift register is copied to the data register before previous data was read)
  - TX Underrun Error (No data to load into shift register for transmitting)
  - Parity Error (When parity is enabled and a parity error is detected)

Configuration

Build Time Configurations for r_spi_b

The following build time configurations are defined in fsp_cfg/r_spi_b_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Enable Support for using DTC	• Enabled • Disabled	Enabled	If enabled, DTC instances will be included in the build for both transmission and reception.
Enable Transmitting from RXI Interrupt	• Enabled • Disabled	Disabled	If enabled, all operations will be handled from the RX (receive) interrupt. This setting only provides a

performance boost when DTC is not used. In addition, Transmit Only mode is not supported when this configuration is enabled.

Configurations for Connectivity > SPI (r_spi_b)

This module can be added to the Stacks tab via New Stack > Connectivity > SPI (r_spi_b). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_spi0	Module name.
Channel	Value must be a non-negative integer	0	Select the SPI channel.
Receive Interrupt Priority	MCU Specific Options		Select the interrupt priority for all SPI interrupts.
Transmit Buffer Empty Interrupt Priority	MCU Specific Options		Select the interrupt priority for all SPI interrupts.
Transfer Complete Interrupt Priority	MCU Specific Options		Select the interrupt priority for all SPI interrupts.
Error Interrupt Priority	MCU Specific Options		Select the interrupt priority for all SPI interrupts.
Operating Mode	• Master • Slave	Master	Select the SPI operating mode.
Clock Phase	• Data sampling on odd edge, data variation on even edge • Data sampling on even edge, data variation on odd edge	Data sampling on odd edge, data variation on even edge	Select the clock edge to sample data.
Clock Polarity	• Low when idle • High when idle	Low when idle	Select clock level when idle.
Mode Fault Error	• Enable • Disable	Disable	Detect master/slave mode conflicts.
Bit Order	• MSB First • LSB First	MSB First	Select the data bit order.

Callback	Name must be a valid C symbol	spi_callback	A user callback function can be provided. If this callback function is provided, it will be called from the interrupt service routine (ISR).
SPI Mode	• SPI Operation • Clock Synchronous Operation	Clock Synchronous Operation	Select the clock sync mode.
Full or Transmit Only Mode	• Full Duplex • Transmit Only	Full Duplex	Select Full Duplex or Transmit Only Mode.
Slave Select Polarity	• Active Low • Active High	Active Low	Select the slave select active level.
Select SSL(Slave Select)	• SSL0 • SSL1 • SSL2 • SSL3	SSL0	Select which slave to use.
MOSI Idle State	• MOSI Idle Value Fixing Disable • MOSI Idle Value Fixing Low • MOSI Idle Value Fixing High	MOSI Idle Value Fixing Disable	Select the MOSI idle level if MOSI idle is enabled.
Parity Mode	• Disabled • Odd • Even	Disabled	Select the parity mode if parity is enabled.
Byte Swapping	• Disable • Enable	Disable	Select the byte swap mode for 16/32-Bit Data Frames.
Clock Source	• SCISPICLK • PCLK	PCLK	Select the clock source for communication.
Bitrate	Value must be an integer greater than 0	16000000	Enter the desired bitrate, change the bitrate to a value supported by MCU. If the requested bitrate cannot be achieved, the settings with the largest possible value that is less than or equal to the requested bitrate is used. The theoretical bitrate is printed in a comment in the generated spi_extended_cfg_t

			structure.
Clock Delay	• 1 Clock • 2 Clocks • 3 Clocks • 4 Clocks • 5 Clocks • 6 Clocks • 7 Clocks • 8 Clocks	1 Clock	Configure the number of SPI clock cycles before each data frame.
SSL Negation Delay	• 1 Clock • 2 Clocks • 3 Clocks • 4 Clocks • 5 Clocks • 6 Clocks • 7 Clocks • 8 Clocks	1 Clock	Configure the number of SPI clock cycles after each data frame.
Next Access Delay	• 1 Clock • 2 Clocks • 3 Clocks • 4 Clocks • 5 Clocks • 6 Clocks • 7 Clocks • 8 Clocks	1 Clock	Configure the number of SPI clock cycles between each data frame.

Clock Configuration

The SPI peripheral uses the SCISPICLK for communication and PCLKB for internal operations. Both can be configured via the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time.

Pin Configuration

This module uses MOSI, MISO, RSPCK, and SSL pins to communicate with on board devices.

Note

At high bitrates it may be necessary to configure the pins with IOPORT_CFG_DRIVE_HIGH to maintain signal integrity.

Usage Notes

Performance

At high bitrates, interrupts may not be able to service transfers fast enough. In master mode this means there will be a delay between each data frame. In slave mode this could result in TX Underrun and RX Overflow errors.

In order to improve performance at high bitrates, it is recommended that the instance be configured to service transfers using the DTC.

Another way to improve performance is to transfer the data in 16/32 bit wide data frames when possible. A typical use-case where this is possible is when reading/writing to a block device.

Transmit From RXI Interrupt

After every data frame the SPI peripheral generates a transmit buffer empty interrupt and a receive buffer full interrupt. It is possible to configure the driver to handle transmit buffer empty interrupts in the receive buffer full ISR. This only improves performance when the DTC is not being used.

Note

Configuring the module to use RX DTC instance without also providing a TX DTC instance results in an invalid configuration when RXI transmit is enabled.
Transmit Only mode is not supported when Transmit from RXI is enabled.

Clock Auto-Stopping

In master mode, if the Receive Buffer Full Interrupts are not handled fast enough, instead of generating a RX Overflow error, the last clock cycle will be stretched until the receive buffer is read.

Parity Mode

When parity mode is configured, the LSB of each data frame is used as a parity bit. When odd parity is selected, the LSB is set such that there are an odd number of ones in the data frame. When even parity is selected, the LSB is set such that there are an even number of ones in the data frame.

Limitations

Developers should be aware of the following limitations when using the SPI:

- In master mode, the driver will only configure 4-Wire mode if the device supports SSL Level Keeping (SSLKP bit in SPCMD0) and will return FSP_ERR_UNSUPPORTED if configured for 4-Wire mode on devices without SSL Level Keeping. Without SSL Level Keeping, the SSL pin is toggled after every data frame. In most cases this is not desirable behavior so it is recommended that the SSL pin be driven in software if SSL Level Keeping is not present on the device.
- The module does not support communicating with multiple slaves using different SSL pins. In order to achieve this, the module must either be closed and re-opened to change the SSL pin or drive SSL in software. It is recommended that SSL be driven in software when controlling multiple slave devices.
- The SPI peripheral has a minimum 3 SPI CLK delay between each data frame.
- The behavior for Byte Swap operation is not guaranteed for data frames other than 8-bit, 16-bit and 32bit.

Examples

Basic Example

This is a basic example of minimal use of the SPI in an application.

```
static volatile bool g_transfer_complete = false;
void spi_basic_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
```

```
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SPI_B_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start a write/read transfer */
    err = R_SPI_B_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer, TRANSFER_SIZE,
SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
}
static void r_spi_callback (spi_callback_args_t * p_args)
{
 if (SPI_EVENT_TRANSFER_COMPLETE == p_args->event)
    {
       g_transfer_complete = true;
    }
}
```

Driving Software Slave Select Line

This is an example of communicating with multiple slave devices by asserting SSL in software.

```
void spi_software_ssl_example (void)
{
    uint8_t tx_buffer[TRANSFER_SIZE];
    uint8_t rx_buffer[TRANSFER_SIZE];
 /* Configure Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Configure Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
```

```
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the SPI module. */
    err = R_SPI_B_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err                = R_SPI_B_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer,
TRANSFER_SIZE, SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 1 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_1, BSP_IO_LEVEL_HIGH);
 /* Wait for minimum time required between transfers. */
 R_BSP_SoftwareDelay(SSL_NEXT_ACCESS_DELAY, BSP_DELAY_UNITS_MICROSECONDS);
 /* Assert Slave Select Line 2 */
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_LOW);
 /* Start a write/read transfer */
    g_transfer_complete = false;
    err                = R_SPI_B_WriteRead(&g_spi_ctrl, tx_buffer, rx_buffer,
TRANSFER_SIZE, SPI_BIT_WIDTH_8_BITS);
    assert(FSP_SUCCESS == err);
 /* Wait for SPI_EVENT_TRANSFER_COMPLETE callback event. */
 while (false == g_transfer_complete)
    {
       ;
    }
 /* De-assert Slave Select Line 2 */
```

```
 R_BSP_PinWrite(SLAVE_SELECT_LINE_2, BSP_IO_LEVEL_HIGH);
}
```

Configuring the SPI Clock Divider Registers

This example demonstrates how to set the SPI clock divisors at runtime.

```
void spi_bitrate_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    g_spi_cfg.p_extend = &g_spi_extended_cfg;
 /* Configure SPI Clock divider to achieve largest bitrate less than or equal to the
desired bitrate. */
    err = R_SPI_B_CalculateBitrate(BITRATE, SPI_B_CLOCK_SOURCE_SCISPICLK,
&(g_spi_extended_cfg.spck_div));
    assert(FSP_SUCCESS == err);
 /* Initialize the SPI module. */
    err = R_SPI_B_Open(&g_spi_ctrl, &g_spi_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	rspck_div_setting_t
struct	spi_b_extended_cfg_t
struct	spi_b_instance_ctrl_t

Enumerations

enum	spi_b_ssl_mode_t
enum	spi_b_communication_t
enum	spi_b_ssl_polarity_t
enum	spi_b_ssl_select_t
enum	spi_b_mosi_idle_value_fixing_t

enum	spi_b_parity_t
enum	spi_b_byte_swap_t
enum	spi_b_delay_count_t
enum	spi_b_clock_source_t

Data Structure Documentation

◆ rspck_div_setting_t

struct rspck_div_setting_t

SPI Clock Divider settings.

Data Fields

uint8_t	spbr	SPBR register setting.
uint8_t	brdv: 2	BRDV setting in SPCMD0.

◆ spi_b_extended_cfg_t

struct spi_b_extended_cfg_t

Extended SPI interface configuration

Data Fields

spi_b_ssl_mode_t	spi_clksyn	Select SPI or Clock Synchronous mode operation.
spi_b_communication_t	spi_comm	Select full-duplex or transmit-only communication.
spi_b_ssl_polarity_t	ssl_polarity	Select SSLn signal polarity.
spi_b_ssl_select_t	ssl_select	Select which slave to use: 0-SSL0, 1-SSL1, 2-SSL2, 3-SSL3.
spi_b_mosi_idle_value_fixing_t	mosi_idle	Select MOSI idle fixed value and selection.
spi_b_parity_t	parity	Select parity and enable/disable parity.
spi_b_byte_swap_t	byte_swap	Select byte swap mode.
spi_b_clock_source_t	clock_source	Communication clock source (TCLK).
rspck_div_setting_t	spck_div	Register values for configuring the SPI Clock Divider.
spi_b_delay_count_t	spck_delay	SPI Clock Delay Register Setting.
spi_b_delay_count_t	ssl_negation_delay	SPI Slave Select Negation Delay Register Setting.

spi_b_delay_count_t	next_access_delay	SPI Next-Access Delay Register Setting.

◆ spi_b_instance_ctrl_t

struct spi_b_instance_ctrl_t	
Channel control block. DO NOT INITIALIZE. Initialization occurs when spi_api_t::open is called.	
Data Fields	
uint32_t	open
	Indicates whether the open() API has been successfully called.
spi_cfg_t const *	p_cfg
	Pointer to instance configuration.
R_SPI_B0_Type *	p_regs
	Base register for this channel.
void const *	p_tx_data
	Buffer to transmit.
void *	p_rx_data
	Buffer to receive.
uint32_t	tx_count
	Number of Data Frames to transfer (8-bit, 16-bit, 32-bit)
uint32_t	rx_count
	Number of Data Frames to transfer (8-bit, 16-bit, 32-bit)
uint32_t	count
	Number of Data Frames to transfer (8-bit, 16-bit, 32-bit)

spi_bit_width_t	bit_width
	Bits per Data frame (8-bit, 16-bit, 32-bit)

Enumeration Type Documentation

◆ spi_b_ssl_mode_t

enum spi_b_ssl_mode_t

3-Wire or 4-Wire mode.

Enumerator	
SPI_B_SSL_MODE_SPI	SPI operation (4-wire method)
SPI_B_SSL_MODE_CLK_SYN	Clock Synchronous operation (3-wire method)

◆ spi_b_communication_t

enum spi_b_communication_t

Transmit Only (Half Duplex), or Full Duplex.

Enumerator	
SPI_B_COMMUNICATION_FULL_DUPLEX	Full-Duplex synchronous serial communication.
SPI_B_COMMUNICATION_TRANSMIT_ONLY	Transit only serial communication.

◆ spi_b_ssl_polarity_t

enum spi_b_ssl_polarity_t

Slave Select Polarity.

Enumerator	
SPI_B_SSLP_LOW	SSLP signal polarity active low.
SPI_B_SSLP_HIGH	SSLP signal polarity active high.

◆ spi_b_ssl_select_t

enum spi_b_ssl_select_t	
The Slave Select Line	
Enumerator	
SPI_B_SSL_SELECT_SSL0	Select SSL0.
SPI_B_SSL_SELECT_SSL1	Select SSL1.
SPI_B_SSL_SELECT_SSL2	Select SSL2.
SPI_B_SSL_SELECT_SSL3	Select SSL3.

◆ spi_b_mosi_idle_value_fixing_t

enum spi_b_mosi_idle_value_fixing_t	
MOSI Idle Behavior.	
Enumerator	
SPI_B_MOSI_IDLE_VALUE_FIXING_DISABLE	MOSI output value=value set in MOIFV bit.
SPI_B_MOSI_IDLE_VALUE_FIXING_LOW	MOSIn level low during MOSI idling.
SPI_B_MOSI_IDLE_VALUE_FIXING_HIGH	MOSIn level high during MOSI idling.

◆ spi_b_parity_t

enum spi_b_parity_t	
Parity Mode	
Enumerator	
SPI_B_PARITY_MODE_DISABLE	Disable parity.
SPI_B_PARITY_MODE_ODD	Select even parity.
SPI_B_PARITY_MODE_EVEN	Select odd parity.

◆ spi_b_byte_swap_t

enum spi_b_byte_swap_t	
Byte Swapping Enable/Disable.	
Enumerator	
SPI_B_BYTE_SWAP_DISABLE	Disable Byte swapping for 16/32-Bit transfers.
SPI_B_BYTE_SWAP_ENABLE	Enable Byte swapping for 16/32-Bit transfers.

◆ spi_b_delay_count_t

enum spi_b_delay_count_t	
Delay count for SPI delay settings.	
Enumerator	
SPI_B_DELAY_COUNT_1	Set RSPCK delay count to 1 RSPCK.
SPI_B_DELAY_COUNT_2	Set RSPCK delay count to 2 RSPCK.
SPI_B_DELAY_COUNT_3	Set RSPCK delay count to 3 RSPCK.
SPI_B_DELAY_COUNT_4	Set RSPCK delay count to 4 RSPCK.
SPI_B_DELAY_COUNT_5	Set RSPCK delay count to 5 RSPCK.
SPI_B_DELAY_COUNT_6	Set RSPCK delay count to 6 RSPCK.
SPI_B_DELAY_COUNT_7	Set RSPCK delay count to 7 RSPCK.
SPI_B_DELAY_COUNT_8	Set RSPCK delay count to 8 RSPCK.

◆ spi_b_clock_source_t

enum spi_b_clock_source_t
SPI communication clock source.

Function Documentation

◆ R_SPI_B_Open()

fsp_err_t R_SPI_B_Open (spi_ctrl_t * *p_api_ctrl*, spi_cfg_t const *const *p_cfg*)

This functions initializes a channel for SPI communication mode. Implements spi_api_t::open.

This function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Configures the peripheral registers according to the configuration.
- Initialize the control structure for use in other SPI Interface functions.

Return values

FSP_SUCCESS	Channel initialized successfully.
FSP_ERR_ALREADY_OPEN	Instance was already initialized.
FSP_ERR_ASSERTION	An invalid argument was given in the configuration structure.
FSP_ERR_UNSUPPORTED	A requested setting is not possible on this device with the current build configuration.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	The channel number is invalid.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls: transfer_api_t::open

Note

This function is reentrant.

◆ R_SPI_B_Read()

fsp_err_t R_SPI_B_Read (spi_ctrl_t *const *p_api_ctrl*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

This function receives data from a SPI device. Implements spi_api_t::read.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI read operation.

Return values

FSP_SUCCESS	Read operation successfully completed.
FSP_ERR_ASSERTION	NULL pointer to control or destination parameters or transfer length is zero.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open channel first.
FSP_ERR_IN_USE	A transfer is already in progress.

◆ R_SPI_B_Write()

fsp_err_t R_SPI_B_Write (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

This function transmits data to a SPI device using the TX Only Communications Operation Mode. Implements spi_api_t::write.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI write operation.

Return values

FSP_SUCCESS	Write operation successfully completed.
FSP_ERR_ASSERTION	NULL pointer to control or source parameters or transfer length is zero.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_IN_USE	A transfer is already in progress.

◆ R_SPI_B_WriteRead()

fsp_err_t R_SPI_B_WriteRead (spi_ctrl_t *const *p_api_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t const *length*, spi_bit_width_t const *bit_width*)

This function simultaneously transmits and receive data. Implements spi_api_t::writeRead.

The function performs the following tasks:

- Performs parameter checking and processes error conditions.
- Sets up the instance to complete a SPI writeRead operation.

Return values

FSP_SUCCESS	Write operation successfully completed.
FSP_ERR_ASSERTION	NULL pointer to control, source or destination parameters or transfer length is zero.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.
FSP_ERR_IN_USE	A transfer is already in progress.

◆ R_SPI_B_Close()

fsp_err_t R_SPI_B_Close (spi_ctrl_t *const *p_api_ctrl*)

This function manages the closing of a channel by the following task. Implements spi_api_t::close.

Disables SPI operations by disabling the SPI bus.

- Disables the SPI peripheral.
- Disables all the associated interrupts.
- Update control structure so it will not work with SPI Interface functions.

Return values

FSP_SUCCESS	Channel successfully closed.
FSP_ERR_ASSERTION	A required pointer argument is NULL.
FSP_ERR_NOT_OPEN	The channel has not been opened. Open the channel first.

◆ R_SPI_B_CalculateBitrate()

fsp_err_t R_SPI_B_CalculateBitrate (uint32_t *bitrate*, spi_b_clock_source_t *clock_source*, rspck_div_setting_t * *spck_div*)

Calculates the SPBR register value and the BRDV bits for a desired bitrate. If the desired bitrate is faster than the maximum bitrate, than the bitrate is set to the maximum bitrate. If the desired bitrate is slower than the minimum bitrate, than an error is returned.

Parameters

[in]	bitrate	Desired bitrate
[in]	clock_source	SPI communication clock source to be used
[out]	spck_div	Memory location to store bitrate register settings.

Return values

FSP_SUCCESS	Valid spbr and brdv values were calculated
FSP_ERR_UNSUPPORTED	Bitrate is not achievable

◆ R_SPI_B_CallbackSet()

fsp_err_t R_SPI_B_CallbackSet (spi_ctrl_t *const *p_api_ctrl*, void(*)(spi_callback_args_t *) *p_callback*, void const *const *p_context*, spi_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements spi_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.61 Serial Sound Interface (r_ssi)

Modules

Functions

fsp_err_t	R_SSI_Open (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)
fsp_err_t	R_SSI_Stop (i2s_ctrl_t *const p_ctrl)
fsp_err_t	R_SSI_StatusGet (i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status)
fsp_err_t	R_SSI_Write (i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes)
fsp_err_t	R_SSI_Read (i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)
fsp_err_t	R_SSI_WriteRead (i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes)
fsp_err_t	R_SSI_Mute (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)
fsp_err_t	R_SSI_Close (i2s_ctrl_t *const p_ctrl)
fsp_err_t	R_SSI_CallbackSet (i2s_ctrl_t *const p_api_ctrl,

	void(*p_callback)(i2s_callback_args_t *), void const *const p_context, i2s_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the SSIE peripheral on RA MCUs. This module implements the I2S Interface.

Overview

Features

The SSI module supports the following features:

- Transmission and reception of uncompressed audio data using the standard I2S protocol in master mode
- Full-duplex I2S communication (channel 0 only)
- Integration with the DTC transfer module
- Internal connection to GPT timer output to generate the audio clock
- Callback function notification when all data is loaded into the SSI FIFO

Configuration

Build Time Configurations for r_ssi

The following build time configurations are defined in fsp_cfg/r_ssi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DTC Support	• Enabled • Disabled	Enabled	If code for DTC transfer support is included in the build.

Configurations for Connectivity > I2S (r_ssi)

This module can be added to the Stacks tab via New Stack > Connectivity > I2S (r_ssi). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_i2s0	Module name.
Channel	Value must be an integer between 0 and 1	0	Specify the I2S channel.
Bit Depth	• 8 Bits	16 Bits	Select the bit depth of

	• 16 Bits • 18 Bits • 20 Bits • 22 Bits • 24 Bits • 32 Bits		one sample of audio data.
Word Length	• 8 Bits • 16 Bits • 24 Bits • 32 Bits • 48 Bits • 64 Bits • 128 Bits • 256 Bits	16 Bits	Select the word length of audio data. Must be at least as large as Data bits.
WS Continue Mode	• Enabled • Disabled	Disabled	Enable WS continue mode to output the word select (WS) pin even when transmission is idle.
Bit Clock Source	• External AUDIO_CLK • Internal AUDIO_CLK	External AUDIO_CLK	Select External AUDIO_CLK for external signal to AUDIO_CLK input pin or Internal AUDIO_CLK for internal connection to MCU specific GPT channel. Please refer to the hardware manual for which GPT channel is connected to the internal signal
Bit Clock Divider	Refer to the RA Configuration tool for available options.	Audio Clock / 1	Select divider used to generate bit clock from audio clock.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called from all three interrupt service routines (ISR).
Transmit Interrupt Priority	MCU Specific Options		Select the transmit interrupt priority.
Receive Interrupt Priority	MCU Specific Options		Select the receive interrupt priority.
Idle/Error Interrupt Priority	MCU Specific Options		Select the Idle/Error interrupt priority.

Clock Configuration

The SSI peripheral runs on PCLKB. The PCLKB frequency can be configured on the **Clocks** tab of the RA Configuration editor. The SSI audio clock can optionally be supplied from an external source through the AUDIO_CLK pin in master mode.

Pin Configuration

The SSI uses the following pins:

- AUDIO_CLK (optional, master mode only): The AUDIO_CLK pin is used to supply the audio clock from an external source.
- SSIBCKn: Bit clock pin for channel n
- SSILRCKn/SSIFSn: Channel selection pin for channel n
- SSIRXD0: Reception pin for channel 0
- SSITXD0: Transmission pin for channel 0
- SSIDATA1: Transmission or reception pin for channel 1

Usage Notes

SSI Frames

An SSI frame is 2 samples worth of data. The frame boundary (end of previous frame, start of next frame) is on the falling edge of the SSILRCKn signal.

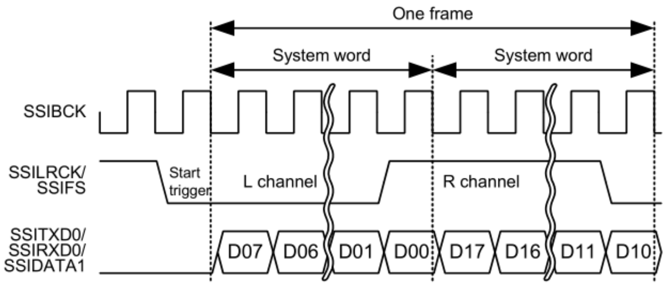


Figure 174: SSI Frame Diagram (8-bit word, 8-bit samples)

Note

If the word length is longer than the sample bit depth, padding bits (0) will be added after the sample.

Audio Data

Only uncompressed PCM data is supported.

Data arrays have the following size, alignment, and length based on the "Bit Depth" setting:

Bit Depth	Array Data Type	Required Alignment	Required Length (bytes)
8 Bits	8-bit integer	1 byte alignment	Multiple of 2
16 Bits	16-bit integer	2 byte alignment	Multiple of 4
18 Bits	32-bit integer, right justified	4 byte alignment	Multiple of 8

20 Bits	32-bit integer, right justified	4 byte alignment	Multiple of 8
22 Bits	32-bit integer, right justified	4 byte alignment	Multiple of 8
24 Bits	32-bit integer, right justified	4 byte alignment	Multiple of 8
32 Bits	32-bit integer	4 byte alignment	Multiple of 8

Note

The length of the array must be a multiple of 2 when the data type is the recommended data type. The 2 represents the frame size (left and right channel) of I2S communication. The SSIE peripheral does not support odd read/write lengths in I2S mode.

Audio Clock

The audio clock is only required for master mode.

Audio Clock Frequency

The bit clock frequency is the product of the sampling frequency and channels and bits per system word:

bit_clock (Hz) = sampling_frequency (Hz) * channels * system_word_bits

I2S data always has 2 channels.

For example, the bit clock for transmitting 2 channels of 16-bit data (using a 16-bit system word) at 44100 Hz would be:

44100 * 2 * 16 = 1,411,200 Hz

The audio clock frequency is used to generate the bit clock frequency. It must be a multiple of the bit clock frequency. Refer to the Bit Clock Divider configuration for divider options. The input audio clock frequency must be:

audio_clock (Hz) = desired_bit_clock (Hz) * bit_clock_divider

To get a bit clock of 1.4 MHz from an audio clock of 2.8 MHz, select the divider Audio Clock / 2.

Audio Clock Source

The audio clock source can come from:

- An external source input to the AUDIO_CLK pin
- An internal connection to the GPT timer output

Note

When using the internal GPT timer output, Pin Output Support must be Enabled, and GTIOCA Output Enabled must be True.
See the SSIE section in the MCU hardware manual for information about which GPT channel may be used.

Limitations

Developers should be aware of the following limitations when using the SSI:

- When using channel 1, full duplex communication is not possible. Only tranmission or reception is possible.
- SSI must go idle before changing the communication mode (between read only, write only, and full duplex)

Examples

Basic Example

This is a basic example of minimal use of the SSI in an application.

```
#define SSI_EXAMPLE_SAMPLES_TO_TRANSFER (1024)
#define SSI_EXAMPLE_TONE_FREQUENCY_HZ (800)
int16_t g_src[SSI_EXAMPLE_SAMPLES_TO_TRANSFER];
int16_t g_dest[SSI_EXAMPLE_SAMPLES_TO_TRANSFER];
void ssi_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Create a stereo sine wave. Using formula sample = sin(2 * pi * tone_frequency * t
/ sampling_frequency) */
    uint32_t freq = SSI_EXAMPLE_TONE_FREQUENCY_HZ;
 for (uint32_t t = 0; t < SSI_EXAMPLE_SAMPLES_TO_TRANSFER / 2; t += 1)
    {
 float input = (((float) (freq * t)) * (M_TWOPI)) /
SSI_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ;
       g_src[2 * t]    = (int16_t) ((INT16_MAX * sinf(input)));
       g_src[2 * t + 1] = (int16_t) ((INT16_MAX * sinf(input)));
    }
 /* Initialize the module. */
    err = R_SSI_Open(&g_i2s_ctrl, &g_i2s_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Transfer data. */
    (void) R_SSI_WriteRead(&g_i2s_ctrl,
                           (uint8_t *) &g_src[0],
                           (uint8_t *) &g_dest[0],
                           SSI_EXAMPLE_SAMPLES_TO_TRANSFER * sizeof(int16_t));
```

```
}
```

Streaming Example

This is an example of using SSI to stream audio data. This application uses a double buffer to store PCM sine wave data. It starts transmitting in the main loop, then loads the next buffer if it is ready in the callback. If the next buffer is not ready, a flag is set in the callback so the application knows to restart transmission in the main loop.

This example also checks the return code of R_SSI_Write() because R_SSI_Write() can return an error if a transmit overflow occurs before the FIFO is reloaded. If a transmit overflow occurs before the FIFO is reloaded, the SSI will be stopped in the error interrupt, and it cannot be restarted until the I2S_EVENT_IDLE callback is received.

```
#define SSI_STREAMING_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ (22050)
#define SSI_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK (1024)
#define SSI_STREAMING_EXAMPLE_TONE_FREQUENCY_HZ (800)
int16_t       g_stream_src[2][SSI_EXAMPLE_SAMPLES_TO_TRANSFER];
uint32_t      g_buffer_index          = 0;
volatile bool g_send_data_in_main_loop = true;
volatile bool g_data_ready = false;
/* Example callback called when SSI is ready for more data. */
void ssi_example_callback (i2s_callback_args_t * p_args)
{
 /* Reload the FIFO if we hit the transmit watermark or restart transmission if the
SSI is idle because it was
  * stopped after a transmit FIFO overflow. */
 if ((I2S_EVENT_TX_EMPTY == p_args->event) || (I2S_EVENT_IDLE == p_args->event))
    {
 if (g_data_ready)
      {
 /* Reload FIFO and handle errors. */
            ssi_example_write();
      }
 else
      {
 /* Data was not ready yet, send it in the main loop. */
            g_send_data_in_main_loop = true;
```

```
      }
    }
}
/* Load the transmit FIFO and check for error conditions. */
void ssi_example_write (void)
{
 /* Transfer data. This call is non-blocking. */
 fsp_err_t err = R_SSI_Write(&g_i2s_ctrl,
                               (uint8_t *) &g_stream_src[g_buffer_index][0],
                               SSI_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK * sizeof
(int16_t));
 if (FSP_SUCCESS == err)
    {
 /* Switch the buffer after data is sent. */
       g_buffer_index = !g_buffer_index;
 /* Allow loop to calculate next buffer only if transmission was successful. */
       g_data_ready = false;
    }
 else
    {
 /* Getting here most likely means a transmit overflow occurred before the FIFO could
be reloaded. The
  * application must wait until the SSI is idle, then restart transmission. In this
example, the idle
  * callback transmits data or resets the flag g_send_data_in_main_loop. */
    }
}
/* Calculate samples. This example is just a sine wave. For this type of data, it
would be better to calculate
 * one period and loop it. This example should be updated for the audio data used by
the application. */
void ssi_example_calculate_samples (uint32_t buffer_index)
{
 static uint32_t t = 0U;
```

```
 /* Create a stereo sine wave. Using formula sample = sin(2 * pi * tone_frequency * t
/ sampling_frequency) */
    uint32_t freq = SSI_STREAMING_EXAMPLE_TONE_FREQUENCY_HZ;
 for (uint32_t i = 0; i < SSI_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK / 2; i += 1)
    {
 float input = (((float) (freq * t)) * M_TWOPI) /
SSI_STREAMING_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ;
       t++;
 /* Store sample twice, once for left channel and once for right channel. */
       int16_t sample = (int16_t) ((INT16_MAX * sinf(input)));
       g_stream_src[buffer_index][2 * i]    = sample;
       g_stream_src[buffer_index][2 * i + 1] = sample;
    }
 /* Data is ready to be sent in the interrupt. */
    g_data_ready = true;
}
void ssi_streaming_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the module. */
    err = R_SSI_Open(&g_i2s_ctrl, &g_i2s_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Prepare data in a buffer that is not currently used for transmission. */
       ssi_example_calculate_samples(g_buffer_index);
 /* Send data in main loop the first time, and if it was not ready in the interrupt.
*/
 if (g_send_data_in_main_loop)
      {
 /* Clear flag. */
            g_send_data_in_main_loop = false;
 /* Reload FIFO and handle errors. */
```

```
            ssi_example_write();
      }
 /* If the next buffer is ready, wait for the data to be sent in the interrupt. */
 while (g_data_ready)
      {
 /* Do nothing. */
      }
    }
}
```

Data Structures

struct	ssi_instance_ctrl_t
struct	ssi_extended_cfg_t

Enumerations

enum	ssi_audio_clock_t
enum	ssi_clock_div_t

Data Structure Documentation

◆ ssi_instance_ctrl_t

struct ssi_instance_ctrl_t
Channel instance control block. DO NOT INITIALIZE. Initialization occurs when i2s_api_t::open is called.

◆ ssi_extended_cfg_t

struct ssi_extended_cfg_t

SSI configuration extension. This extension is optional.

Data Fields

ssi_audio_clock_t	audio_clock	Audio clock source, default is SSI_AUDIO_CLOCK_EXTERNAL.
ssi_clock_div_t	bit_clock_div	Select bit clock division ratio.

Enumeration Type Documentation

◆ ssi_audio_clock_t

enum ssi_audio_clock_t	
Audio clock source.	
Enumerator	
SSI_AUDIO_CLOCK_EXTERNAL	Audio clock source is the AUDIO_CLK input pin.
SSI_AUDIO_CLOCK_INTERNAL	Audio clock source is internal connection to a MCU specific GPT channel output.

◆ ssi_clock_div_t

enum ssi_clock_div_t	
Bit clock division ratio. Bit clock frequency = audio clock frequency / bit clock division ratio.	
Enumerator	
SSI_CLOCK_DIV_1	Clock divisor 1.
SSI_CLOCK_DIV_2	Clock divisor 2.
SSI_CLOCK_DIV_4	Clock divisor 4.
SSI_CLOCK_DIV_6	Clock divisor 6.
SSI_CLOCK_DIV_8	Clock divisor 8.
SSI_CLOCK_DIV_12	Clock divisor 12.
SSI_CLOCK_DIV_16	Clock divisor 16.
SSI_CLOCK_DIV_24	Clock divisor 24.
SSI_CLOCK_DIV_32	Clock divisor 32.
SSI_CLOCK_DIV_48	Clock divisor 48.
SSI_CLOCK_DIV_64	Clock divisor 64.
SSI_CLOCK_DIV_96	Clock divisor 96.
SSI_CLOCK_DIV_128	Clock divisor 128.

Function Documentation

◆ R_SSI_Open()

fsp_err_t R_SSI_Open (i2s_ctrl_t *const *p_ctrl*, i2s_cfg_t const *const *p_cfg*)

Opens the SSI. Implements i2s_api_t::open.

This function sets this clock divisor and the configurations specified in i2s_cfg_t. It also opens the timer and transfer instances if they are provided.

Return values

FSP_SUCCESS	Ready for I2S communication.
FSP_ERR_ASSERTION	The pointer to p_ctrl or p_cfg is null.
FSP_ERR_ALREADY_OPEN	The control block has already been opened.
FSP_ERR_IP_CHANNEL_NOT_PRESENT	Channel number is not available on this MCU.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::open

◆ R_SSI_Stop()

fsp_err_t R_SSI_Stop (i2s_ctrl_t *const *p_ctrl*)

Stops SSI. Implements i2s_api_t::stop.

This function disables both transmission and reception, and disables any transfer instances used.

The SSI will stop on the next frame boundary. Do not restart SSI until it is idle.

Return values

FSP_SUCCESS	I2S communication stop request issued.
FSP_ERR_ASSERTION	The pointer to p_ctrl was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

Returns

See Common Error Codes or lower level drivers for other possible return codes.

◆ R_SSI_StatusGet()

fsp_err_t R_SSI_StatusGet (i2s_ctrl_t *const *p_ctrl*, i2s_status_t *const *p_status*)

Gets SSI status and stores it in provided pointer p_status. Implements i2s_api_t::statusGet.

Return values

FSP_SUCCESS	Information stored successfully.
FSP_ERR_ASSERTION	The p_instance_ctrl or p_status parameter was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

◆ R_SSI_Write()

fsp_err_t R_SSI_Write (i2s_ctrl_t *const *p_ctrl*, void const *const *p_src*, uint32_t const *bytes*)

Writes data buffer to SSI. Implements i2s_api_t::write.

This function resets the transfer if the transfer interface is used, or writes the length of data that fits in the FIFO then stores the remaining write buffer in the control block to be written in the ISR.

Write() cannot be called if another write(), read() or writeRead() operation is in progress. Write can be called when the SSI is idle, or after the I2S_EVENT_TX_EMPTY event.

Return values

FSP_SUCCESS	Write initiated successfully.
FSP_ERR_ASSERTION	The pointer to p_ctrl or p_src was null, or bytes requested was 0.
FSP_ERR_IN_USE	Another transfer is in progress, data was not written.
FSP_ERR_NOT_OPEN	The channel is not opened.
FSP_ERR_UNDERFLOW	A transmit underflow error is pending. Wait for the SSI to go idle before resuming communication.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- transfer_api_t::reset

◆ R_SSI_Read()

fsp_err_t R_SSI_Read (i2s_ctrl_t *const *p_ctrl*, void *const *p_dest*, uint32_t const *bytes*)

Reads data into provided buffer. Implements i2s_api_t::read.

This function resets the transfer if the transfer interface is used, or reads the length of data available in the FIFO then stores the remaining read buffer in the control block to be filled in the ISR.

Read() cannot be called if another write(), read() or writeRead() operation is in progress. Read can be called when the SSI is idle, or after the I2S_EVENT_RX_FULL event.

Return values

FSP_SUCCESS	Read initiated successfully.
FSP_ERR_IN_USE	Peripheral is in the wrong mode or not idle.
FSP_ERR_ASSERTION	The pointer to p_ctrl or p_dest was null, or bytes requested was 0.
FSP_ERR_NOT_OPEN	The channel is not opened.
FSP_ERR_OVERFLOW	A receive overflow error is pending. Wait for the SSI to go idle before resuming communication.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- transfer_api_t::reset

◆ R_SSI_WriteRead()

fsp_err_t R_SSI_WriteRead (i2s_ctrl_t *const *p_ctrl*, void const *const *p_src*, void *const *p_dest*, uint32_t const *bytes*)

Writes from source buffer and reads data into destination buffer. Implements i2s_api_t::writeRead.

This function calls R_SSI_Write and R_SSI_Read.

writeRead() cannot be called if another write(), read() or writeRead() operation is in progress. writeRead() can be called when the SSI is idle, or after the I2S_EVENT_RX_FULL event.

Return values

FSP_SUCCESS	Write and read initiated successfully.
FSP_ERR_IN_USE	Peripheral is in the wrong mode or not idle.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	The channel is not opened.
FSP_ERR_UNDERFLOW	A transmit underflow error is pending. Wait for the SSI to go idle before resuming communication.
FSP_ERR_OVERFLOW	A receive overflow error is pending. Wait for the SSI to go idle before resuming communication.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- transfer_api_t::reset

◆ R_SSI_Mute()

fsp_err_t R_SSI_Mute (i2s_ctrl_t *const *p_ctrl*, i2s_mute_t const *mute_enable*)

Mutes SSI on the next frame boundary. Implements i2s_api_t::mute.

Data is still written while mute is enabled, but the transmit line outputs zeros.

Return values

FSP_SUCCESS	Transmission is muted.
FSP_ERR_ASSERTION	The pointer to p_ctrl was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

◆ R_SSI_Close()

fsp_err_t R_SSI_Close (i2s_ctrl_t *const *p_ctrl*)

Closes SSI. Implements i2s_api_t::close.

This function powers down the SSI and closes the lower level timer and transfer drivers if they are used.

Return values

FSP_SUCCESS	Device closed successfully.
FSP_ERR_ASSERTION	The pointer to p_ctrl was null.
FSP_ERR_NOT_OPEN	The channel is not opened.

◆ R_SSI_CallbackSet()

fsp_err_t R_SSI_CallbackSet (i2s_ctrl_t *const *p_api_ctrl*, void(*)(i2s_callback_args_t *) *p_callback*, void const *const *p_context*, i2s_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements i2s_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.62 USB (r_usb_basic)

Modules

Functions

fsp_err_t	R_USB_Open (usb_ctrl_t *const p_api_ctrl, usb_cfg_t const *const p_cfg) Applies power to the USB module specified in the argument (p_ctrl). More...

fsp_err_t	R_USB_Close (usb_ctrl_t *const p_api_ctrl) Terminates power to the USB module specified in argument (p_ctrl). USB0 module stops when USB_IP0 is specified to the member (module), USB1 module stops when USB_IP1 is specified to the member (module). More...
fsp_err_t	R_USB_Read (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t destination) Bulk/interrupt data transfer and control data transfer. More...
fsp_err_t	R_USB_Write (usb_ctrl_t *const p_api_ctrl, uint8_t const *const p_buf, uint32_t size, uint8_t destination) Bulk/Interrupt data transfer and control data transfer. More...
fsp_err_t	R_USB_Stop (usb_ctrl_t *const p_api_ctrl, usb_transfer_t direction, uint8_t destination) Requests a data read/write transfer be terminated when a data read/write transfer is being performed. More...
fsp_err_t	R_USB_Suspend (usb_ctrl_t *const p_api_ctrl) Sends a SUSPEND signal from the USB module assigned to the member (module) of the usb_crtl_t structure. More...
fsp_err_t	R_USB_Resume (usb_ctrl_t *const p_api_ctrl) Sends a RESUME signal from the USB module assigned to the member (module) of the usb_ctrl_tstructure. More...
fsp_err_t	R_USB_VbusSet (usb_ctrl_t *const p_api_ctrl, uint16_t state) Specifies starting or stopping the VBUS supply. More...
fsp_err_t	R_USB_InfoGet (usb_ctrl_t *const p_api_ctrl, usb_info_t *p_info, uint8_t destination) Obtains completed USB-related events. More...
fsp_err_t	R_USB_PipeRead (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number) Requests a data read (bulk/interrupt transfer) via the pipe specified in the argument. More...

fsp_err_t	R_USB_PipeWrite (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number) Requests a data write (bulk/interrupt transfer). More...
fsp_err_t	R_USB_PipeStop (usb_ctrl_t *const p_api_ctrl, uint8_t pipe_number) Terminates a data read/write operation. More...
fsp_err_t	R_USB_UsedPipesGet (usb_ctrl_t *const p_api_ctrl, uint16_t *p_pipe, uint8_t destination) Gets the selected pipe number (number of the pipe that has completed initalization) via bit map information. More...
fsp_err_t	R_USB_PipeInfoGet (usb_ctrl_t *const p_api_ctrl, usb_pipe_t *p_info, uint8_t pipe_number) Gets the following pipe information regarding the pipe specified in the argument (p_ctrl) member (pipe): endpoint number, transfer type, transfer direction and maximum packet size. More...
fsp_err_t	R_USB_PullUp (usb_ctrl_t *const p_api_ctrl, uint8_t state) This API enables or disables pull-up of D+/D- line. More...
fsp_err_t	R_USB_EventGet (usb_ctrl_t *const p_api_ctrl, usb_status_t *event) Obtains completed USB related events. (OS-less Only) More...
fsp_err_t	R_USB_Callback (usb_callback_t *p_callback) Register a callback function to be called upon completion of a USB related event. (RTOS only) More...
fsp_err_t	R_USB_HostControlTransfer (usb_ctrl_t *const p_api_ctrl, usb_setup_t *p_setup, uint8_t *p_buf, uint8_t device_address) Performs settings and transmission processing when transmitting a setup packet. More...
fsp_err_t	R_USB_PeriControlDataGet (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size) Receives data sent by control transfer. More...

fsp_err_t	R_USB_PeriControlDataSet (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size) Performs transfer processing for control transfer. More...
fsp_err_t	R_USB_PeriControlStatusSet (usb_ctrl_t *const p_api_ctrl, usb_setup_status_t status) Set the response to the setup packet. More...
fsp_err_t	R_USB_RemoteWakeup (usb_ctrl_t *const p_api_ctrl) Sends a remote wake-up signal to the connected Host. More...
fsp_err_t	R_USB_ModuleNumberGet (usb_ctrl_t *const p_api_ctrl, uint8_t *module_number) This API gets the module number. More...
fsp_err_t	R_USB_ClassTypeGet (usb_ctrl_t *const p_api_ctrl, usb_class_t *class_type) This API gets the class type. More...
fsp_err_t	R_USB_DeviceAddressGet (usb_ctrl_t *const p_api_ctrl, uint8_t *device_address) This API gets the device address. More...
fsp_err_t	R_USB_PipeNumberGet (usb_ctrl_t *const p_api_ctrl, uint8_t *pipe_number) This API gets the pipe number. More...
fsp_err_t	R_USB_DeviceStateGet (usb_ctrl_t *const p_api_ctrl, uint16_t *state) This API gets the state of the device. More...
fsp_err_t	R_USB_DataSizeGet (usb_ctrl_t *const p_api_ctrl, uint32_t *data_size) This API gets the data size. More...
fsp_err_t	R_USB_SetupGet (usb_ctrl_t *const p_api_ctrl, usb_setup_t *setup) This API gets the setup type. More...

Detailed Description

Driver for the USB peripheral on RA MCUs. This module implements the USB Interface.

Overview

The USB module operates in combination with the device class drivers provided by Renesas to form a complete USB stack.

Features

The USB module has the following key features:

- USB Host mode
  - Enumerates Low/Full/High-speed devices (see note below)
  - Automatic transfer error determination and retry
- USB Peripheral mode
  - Supports USB1.1/2.0/3.0 hosts
- Automatic processing of device connect/disconnect, suspend/resume, and USB bus reset
- Up to 10 pipes
  - Control transfers supported on pipe 0
  - Data transfer on pipes 1 to 9 (Bulk or Interrupt)
- Functions with or without an RTOS

Note

Supported speeds are dependent on the MCU.

Configuration

Build Time Configurations for r_usb_basic

The following build time configurations are defined in fsp_cfg/r_usb_basic_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
PLL Frequency	• 24MHz • 20MHz • 12MHz • Other	24MHz	Specify the PLL frequency supplied to the USB module. This setting only applies to USB1 (not USB0).
CPU Bus Access Wait Cycles	Refer to the RA Configuration tool for available options.	9 cycles	This setting controls the delay for consecutive USB peripheral register access. Set this value to a number of CPU cycles that is equivalent to 40.8ns or more.

Battery Charging	• Enabled • Disabled	Enabled	Specify whether or not to include battery charging functionality.
Power IC Shutdown Polarity	• Active High • Active Low	Active High	Select the polarity of the Shutdown signal on the power supply IC (if provided).
Dedicated Charging Port (DCP) Mode	• Enabled • Disabled	Disabled	When enabled, USB communication is disabled and the port is used for charging only.
Notifications for SET_INTERFACE/SET_FEATURE/CLEAR_FEATURE	• Disabled • Enabled	Enabled	When enabled, the application will receive notifications for SET_INTERFACE, SET_FEATURE and CLEAR_FEATURE messages.
Double Buffering	• Disabled • Enabled	Enabled	When enabled, the FIFOs for Pipes 1-5 are double-buffered.
Continuous Transfer Mode	• Disabled • Enabled	Disabled	Enable or disable continuous transfer mode.
LDO Regulator	• Disable • Enable	Disable	Enable or disable LDO regulator.
DMA Support	• Disable • Enable	Disable	Enable or disable DMA support for the USB module.
DMA Source Address	• DMA Disabled • FS Address • HS Address	DMA Disabled	Set this to match the speed mode when DMA is enabled. Otherwise, set to 'DMA Disabled'.
DMA Destination Address	• DMA Disabled • FS Address • HS Address	DMA Disabled	Set this to match the speed mode when DMA is enabled. Otherwise, set to 'DMA Disabled'.
USB Compliance Test mode	• Enabled • Disabled	Disabled	Display the information required to take the compliance test.
USB TPL table name	Enter the TPL table name.	NULL	Enter the name of the TPL Table.

Configurations for Connectivity > USB (r_usb_basic)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_basic0	Module name.

USB Mode	• Host mode • Peri mode	Host mode	Select the usb mode.
USB Speed	• Full Speed • Hi Speed • Low Speed	Full Speed	Select the USB speed.
USB Module Number	• USB_IP0 Port • USB_IP1 Port	USB_IP0 Port	Specify the USB module number to be used.
USB Device Class	• Peripheral Communications Device Class • Peripheral Human Interface Device Class • Peripheral Mass Storage Class • Peripheral Vendor Class • Peripheral Audio Class • Host Communications Device Class • Host Human Interface Device Class • Host Mass Storage Class • Host Vendor Class	Peripheral Communications Device Class	Select the USB device class.
USB Descriptor	USB Descriptor must be a valid C symbol.	g_usb_descriptor	Enter the name of the descriptor to be used. For how to create a descriptor structure, refer to the Descriptor definition chapter in the usb_basic manual. Specify NULL when using the Host class.
USB Compliance Callback	Compliance Callback must be a valid C symbol.	NULL	Set the callback for compliance tests here.
USBFS Interrupt Priority	MCU Specific Options		Select the interrupt priority used by the main USBFS ISR.
USBFS Resume Priority	MCU Specific Options		Select the interrupt priority used by the USBFS Resume ISR.
USBFS D0FIFO	MCU Specific Options		Select the interrupt

Interrupt Priority			priority used by the USBFS D0FIFO.
USBFS D1FIFO Interrupt Priority	MCU Specific Options		Select the interrupt priority used by the USBFS D1FIFO.
USBHS Interrupt Priority	MCU Specific Options		Select the interrupt priority used by the main USBHS ISR.
USBHS D0FIFO Interrupt Priority	MCU Specific Options		Select the interrupt priority used by the USBHS D0FIFO ISR.
USBHS D1FIFO Interrupt Priority	MCU Specific Options		Select the interrupt priority used by the USBHS D1FIFO ISR.
USB RTOS Callback	Enter the address of the function.	NULL	If an FreeRTOS is used, set the callback function here.
USB Callback Context	Enter the address of the context.	NULL	Set the callback context here.

Clock Configuration

The USB module uses PLL as the clock source. The PLL frequency can be set in the **Clocks** tab of the configuration editor or by using the CGC Interface at run-time.

Note

When using HOCO as the PLL source on Cortex M33 parts the FLL function must be enabled for correct USB operation. Refer to the MCU Family -> Clocks group of the BSP properties in the RA configuration tool to adjust FLL settings.

Pin Configuration

In peripheral mode the USB_VBUS and/or USBHS_VBUS pins are used to detect the USB connection status (connected or disconnected) and should be connected to the USB VBUS signal.

Note

USB_VBUS and USBHS_VBUS are 5V-tolerant pins.

In host mode the USBHS_VBUSEN, USBHS_OVRCURA and USBHS_OVRCURB pins should be connected to the relevant pins of an external power supply IC, if available. These pins will be used to manage the USB VBUS supply.

DMA Configuration

When using DMA with USB the following properties must be configured for each DMAC module:

Config Name	Select Name	Description
Transfer Size	2 Bytes 4 Bytes	In FS mode, select "2 Bytes" In HS mode, select "4 Bytes"
Activation source	USBFS FIFO 0	USB FS Reception

USBFS FIFO 1	USB FS Transmission
USBHS FIFO 0	USB HS Reception
USBHS FIFO 1	USB HS Transmission

Descriptor definition

In Peripheral mode, the usb_descriptor_t structure stores descriptor information including the device and configuration descriptors. The values set in this structure are sent to the USB host as part of enumeration.

```
typedef struct usb_descriptor
{
    uint8_t *p_device;    /* Pointer to device descriptor */
    uint8_t *p_config_f; /* Pointer to full-speed configuration descriptor */
    uint8_t *p_config_h; /* Pointer to high-speed configuration descriptor (HS only)
*/
    uint8_t *p_qualifier; /* Pointer to device qualifier descriptor (HS only) */
    uint8_t **pp_string; /* Pointer to string descriptor table */
    uint8_t num_string;   /* Number of strings in table */
} usb_descriptor_t;
```

Note

Even in high-speed mode the full-speed configuration must be made available:

```
/* Example USB FS descriptor struct */
usb_descriptor_t g_usb_descriptor =
{
    smp_device,
    smp_config_f,
    NULL,
    NULL,
    smp_str_table,
    3,
};
/* Example USB HS descriptor struct */
usb_descriptor_t g_usb_descriptor =
{
    smp_device,
```

```
    smp_config_f,
    smp_config_h,
    smp_qualifier,
    smp_str_table,
    3,
};
```

String Descriptor

This USB driver requires string descriptors to be registered in the string descriptor table. Use the following format to define the elements:

```
/* String descriptor 0 is reserved for language ID information */
uint8_t str_descriptor_0[]
{
    0x04,      /* Length */
    0x03,      /* Descriptor type */
    0x09, 0x04 /* Language ID */
};
uint8_t str_descriptor_manufacturer[] =
{
    0x10,      /* Length */
    0x03,      /* Descriptor type */
 'R', 0x00,
 'E', 0x00,
 'N', 0x00,
 'E', 0x00,
 'S', 0x00,
 'A', 0x00,
 'S', 0x00
};
uint8_t str_descriptor_product[] =
{
    0x12,      /* Length */
    0x03,      /* Descriptor type */
```

```
 'C', 0x00,
 'D', 0x00,
 'C', 0x00,
 '_', 0x00,
 'D', 0x00,
 'E', 0x00,
 'M', 0x00,
 'O', 0x00
};
/* String descriptor table */
uint8_t * smp_str_table[] =
{
    str_descriptor_0,            /* Index: 0 */
    str_descriptor_manufacturer, /* Index: 1 */
    str_descriptor_product,      /* Index: 2 */
};
```

Note

*Set the string index values in the device/configuration descriptors (iManufacturer, iConfiguration etc.) to the index of the desired string in the string descriptor table. For example, in the table below, the manufacturer is described in **str_descriptor_manufacturer** and the value of iManufacturer in the device descriptor is **1**.*

Other Descriptors

Refer to the Universal Serial Bus Revision 2.0 specification (http://www.usb.org/developers/docs/) for details on how to construct the device, configuration and qualifier descriptors.

Usage Notes

Program Structure

USB applications (whether using an RTOS or not) should be written as an event-handling loop. Either a callback function (RTOS only) or R_USB_EventGet should be used to provide event data to the application loop where a switch statement handles the event.

Note

1.The USB_STATUS_CONFIGURED event should be confirmed before calling R_USB_Read or R_USB_Write.
2.When attaching to USB Host, Suspend event is notified to the application program in USB peripheral mode. Notification of this event to the application program does not affect the operation.

Limitations

Developers should be aware of the following limitations when using the USB driver:

- The current USB driver does not support hub.
- In USB host mode, the module does not support suspend during data transfers. Execute suspend only after confirming that all transfers are complete.
- Multiconfigurations are not supported.
- This driver does not support CPU transfers using the D0FIFO/D1FIFO register.
- Only one device-class driver may be used at a time.
- The USB Hi-Speed module only supports Hi-Speed operation.
- In USB host mode, this USB driver does not support the composite device.
- The user can not specify DMA transfer to USB IP0 and IP1 modules at the same time when using USB multi-port feature. USB multi-port function: Simultaneous operation feature of USB Host and Peripheral.

Compliance Test

Please set as follows to the following items in RA configuration (r_usb_basic) when doing the compliance test.

1.USB Compliance Test mode Set "Enabled" in this item.

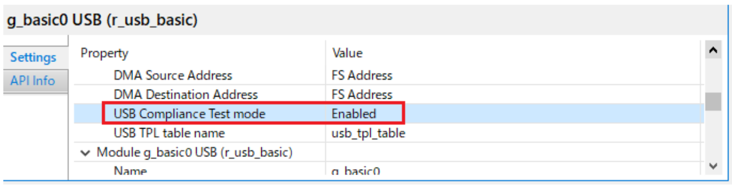


Figure 175: Compliance Test Setting

2.USB TPL table name. Set the start address of TPL(Target Peripheral List) defined in the application program.

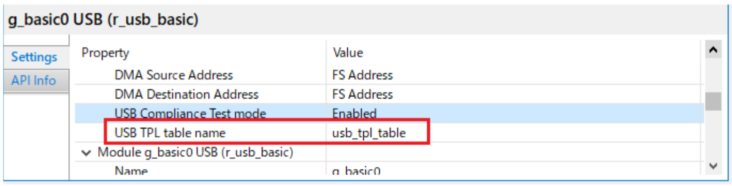


Figure 176: TPL Start Address Setting

Please refer to the following about how to define for TPL.
The following example is when two devices are set in TPL.

```
const uint16_t usb_tpl_table[] =
{
    2,                  /* Number of tpl */
    0,                  /* Reserved */
    0x0123, 0x4567,     /* Vendor ID, Product ID (1st device) */
```

```
    0x89ab, 0xcdef    /* Vendor ID, Product ID (2nd device) */
};
```

3.USB Compliance Callback Set the start address of the callback function defined in the application program.

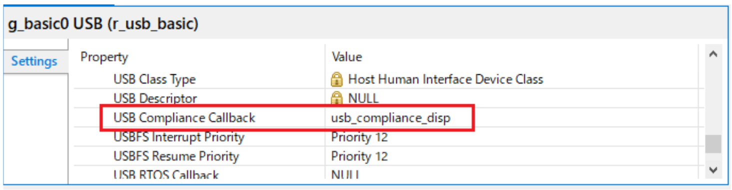


Figure 177: Compliance Callback Setting

The user needs to create this callback function by referring to the following.

```
void usb_compliance_disp (void * param)
{
    usb_compliance_t    *disp_data;
    uint8_t             print_data[32];
    disp_data = (usb_compliance_t*)param;
 switch(disp_data->status)
    {
 case USB_COMPLIANCETEST_ATTACH:             /* Device Attach Detection */
            display("\nATTACH \n");
 break;
 case USB_COMPLIANCETEST_DETACH:             /* Device Detach Detection */
            display("\nDETACH \n");
 break;
 case USB_COMPLIANCETEST_TPL:                /* TPL device connect */
            display("\nTPL PID:%04x VID:%04x \n",disp_data->pid, disp_data->vid);
 break;
 case USB_COMPLIANCETEST_NOTTPL:             /* Not TPL device connect */
            display("\nNOTPL PID:%04x VID:%04x \n",disp_data->pid, disp_data->vid);
 break;
 case USB_COMPLIANCETEST_HUB:                /* USB Hub connect */
            display("\nHub    \n");
```

```
 break;
 case USB_COMPLIANCETEST_OVRC:             /* over current */
 break;
 case USB_COMPLIANCETEST_NORES:             /* Responce Time out for Control Read
Transfer */
            display("\nNOTRESP \n");
 break;
 case USB_COMPLIANCETEST_SETUP_ERR:         /* Setup Transaction Error */
 break;
 default:
 break;
    }
} /* End of function usb_compliance_disp() */
```

Please replace the display function described in the example with the display function created by the customer.

TrustZone

The USB driver for FreeRTOS cannot be allocated in Secure region.

UCLK setting

Enable UCLK in "Clocks" tab on e2 studio when using the following MCU.

1. RA6M4

Examples

USB Basic Example

This is a basic example of minimal use of the USB in an application.

```
void usb_basic_example (void)
{
    usb_event_info_t event_info;
 usb_status_t    event;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
```

```
    {
       g_usb_on_usb.eventGet(&event_info, &event);
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
 case USB_STATUS_WRITE_COMPLETE:
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
 break;
 case USB_STATUS_READ_COMPLETE:
                g_usb_on_usb.write(&g_basic0_ctrl, g_buf, event_info.data_size,
USB_CLASS_PCDC);
 break;
 case USB_STATUS_REQUEST:   /* Receive Class Request */
 if (USB_PCDC_SET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlDataGet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
      }
 else if (USB_PCDC_GET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
      }
 else
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
      }
 break;
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
 break;
 default:
 break;
```

```
        }
    }
} /* End of function usb_main() */
```

Typedefs

typedef usb_event_info_t usb_instance_ctrl_t

Typedef Documentation

◆ usb_instance_ctrl_t

typedef usb_event_info_t usb_instance_ctrl_t

ICU private control block. DO NOT MODIFY. Initialization occurs when R_ICU_ExternalIrqOpen is called.

Function Documentation

◆ R_USB_Open()

fsp_err_t R_USB_Open (usb_ctrl_t *const *p_api_ctrl*, usb_cfg_t const *const *p_cfg*)

Applies power to the USB module specified in the argument (p_ctrl).

Return values

FSP_SUCCESS	Success in open.
FSP_ERR_USB_BUSY	Specified USB module now in use.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_Close()

fsp_err_t R_USB_Close (usb_ctrl_t *const *p_api_ctrl*)

Terminates power to the USB module specified in argument (p_ctrl). USB0 module stops when USB_IP0 is specified to the member (module), USB1 module stops when USB_IP1 is specified to the member (module).

Return values

FSP_SUCCESS	Success.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_NOT_OPEN	USB module is not open.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_Read()

fsp_err_t R_USB_Read (usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *destination*)

Bulk/interrupt data transfer and control data transfer.

1. Bulk/interrupt data transfer

   Requests USB data read (bulk/interrupt transfer). The read data is stored in the area specified by argument (p_buf). After data read is completed, confirm the operation by checking the return value (USB_STATUS_READ_COMPLETE) of the R_USB_GetEvent function. The received data size is set in member (size) of the usb_ctrl_t structure. To figure out the size of the data when a read is complete, check the return value (USB_STATUS_READ_COMPLETE) of the R_USB_GetEvent function, and then refer to the member (size) of the usb_crtl_t structure.

2. Control data transfer

   The R_USB_Read function is used to receive data in the data stage and the R_USB_Write function is used to send data to the USB host.

Return values

FSP_SUCCESS	Successfully completed (Data read request completed).
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_BUSY	Data receive request already in process for USB device with same device address.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_Write()

fsp_err_t R_USB_Write (usb_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_buf*, uint32_t *size*, uint8_t *destination*)

Bulk/Interrupt data transfer and control data transfer.

1. Bulk/Interrupt data transfer

   Requests USB data write (bulk/interrupt transfer). Stores write data in area specified by argument (p_buf). Set the device class type in usb_ctrl_t structure member (type). Confirm after data write is completed by checking the return value (USB_STATUS_WRITE_COMPLETE) of the R_USB_GetEvent function. For sending a zero-length packet, please refer the following Note.

2. Control data transfer

   The R_USB_Read function is used to receive data in the data stage and the R_USB_Write function is used to send data to the USB host.

Return values

FSP_SUCCESS	Successfully completed. (Data write request completed)
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_BUSY	Data write request already in process for USB device with same device address.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

1.The user needs to send the zero-length packet(ZLP) since this USB driver does not send the ZLP automatically. When sending a ZLP, the user sets USB_NULL in the third argument (size) of R_USB_Write function as follow.
e.g)
R_USB_Write (&g_basic0_ctrl, &g_buf, USB_NULL);
2.Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_Stop()

fsp_err_t R_USB_Stop (usb_ctrl_t *const *p_api_ctrl*, usb_transfer_t *direction*, uint8_t *destination*)

Requests a data read/write transfer be terminated when a data read/write transfer is being performed.

To stop a data read, set USB_TRANSFER_READ as the argument (type); to stop a data write, specify USB_WRITE as the argument (type).

Return values

FSP_SUCCESS	Successfully completed. (stop completed)
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_BUSY	Stop processing is called multiple times.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_Suspend()

fsp_err_t R_USB_Suspend (usb_ctrl_t *const *p_api_ctrl*)

Sends a SUSPEND signal from the USB module assigned to the member (module) of the usb_crtl_t structure.

After the suspend request is completed, confirm the operation with the return value (USB_STATUS_SUSPEND) of the R_USB_EventGet function.

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_BUSY	During a suspend request to the specified USB module, or when the USB module is already in the suspended state.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_Resume()

fsp_err_t R_USB_Resume (usb_ctrl_t *const *p_api_ctrl*)

Sends a RESUME signal from the USB module assigned to the member (module) of the usb_ctrl_tstructure.

After the resume request is completed, confirm the operation with the return value (USB_STATUS_RESUME) of the R_USB_EventGet function

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_BUSY	Resume already requested for same device address. (USB host mode only)
FSP_ERR_USB_NOT_SUSPEND	USB device is not in the SUSPEND state.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_VbusSet()

fsp_err_t R_USB_VbusSet (usb_ctrl_t *const *p_api_ctrl*, uint16_t *state*)

Specifies starting or stopping the VBUS supply.

Return values

FSP_SUCCESS	Successful completion. (VBUS supply start/stop completed)
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_InfoGet()

fsp_err_t R_USB_InfoGet (usb_ctrl_t *const *p_api_ctrl*, usb_info_t * *p_info*, uint8_t *destination*)

Obtains completed USB-related events.

Return values

FSP_SUCCESS	Successful completion. (VBUS supply start/stop completed)
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_USB_PARAMETER	Parameter error.

◆ R_USB_PipeRead()

fsp_err_t R_USB_PipeRead (usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *pipe_number*)

Requests a data read (bulk/interrupt transfer) via the pipe specified in the argument.

The read data is stored in the area specified in the argument (p_buf). After the data read is completed, confirm the operation with the R_USB_GetEvent function return value(USB_STATUS_READ_COMPLETE). To figure out the size of the data when a read is complete, check the return value (USB_STATUS_READ_COMPLETE) of the R_USB_GetEvent function, and then refer to the member (size) of the usb_crtl_t structure.

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_BUSY	Specified pipe now handling data receive/send request.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_PipeWrite()

fsp_err_t R_USB_PipeWrite (usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*, uint8_t *pipe_number*)

Requests a data write (bulk/interrupt transfer).

The write data is stored in the area specified in the argument (p_buf). After data write is completed, confirm the operation with the return value (USB_STATUS_WRITE_COMPLETE) of the EventGet function. For sending a zero-length packet, please refer the following Note.

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_BUSY	Specified pipe now handling data receive/send request.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

1.The user needs to send the zero-length packet(ZLP) since this USB driver does not send the ZLP automatically. When sending a ZLP, the user sets USB_NULL in the third argument (size) of R_USB_PipeWrite function as follow.
e.g)
R_USB_PipeWrite (&g_basic0_ctrl, &g_buf, USB_NULL, pipe_number);
2.Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_PipeStop()

fsp_err_t R_USB_PipeStop (usb_ctrl_t *const *p_api_ctrl*, uint8_t *pipe_number*)

Terminates a data read/write operation.

Return values

FSP_SUCCESS	Successfully completed. (Stop request completed)
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_UsedPipesGet()

fsp_err_t R_USB_UsedPipesGet (usb_ctrl_t *const *p_api_ctrl*, uint16_t * *p_pipe*, uint8_t *destination*)

Gets the selected pipe number (number of the pipe that has completed initalization) via bit map information.

The bit map information is stored in the area specified in argument (p_pipe). Based on the information (module member and address member) assigned to the usb_ctrl_t structure, obtains the PIPE information of that USB device.

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

◆ R_USB_PipeInfoGet()

fsp_err_t R_USB_PipeInfoGet (usb_ctrl_t *const *p_api_ctrl*, usb_pipe_t * *p_info*, uint8_t *pipe_number*)

Gets the following pipe information regarding the pipe specified in the argument (p_ctrl) member (pipe): endpoint number, transfer type, transfer direction and maximum packet size.

The obtained pipe information is stored in the area specified in the argument (p_info).

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

◆ R_USB_PullUp()

fsp_err_t R_USB_PullUp (usb_ctrl_t *const *p_api_ctrl*, uint8_t *state*)

This API enables or disables pull-up of D+/D- line.

Return values

FSP_SUCCESS	Successful completion. (Pull-up enable/disable setting completed)
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_EventGet()

fsp_err_t R_USB_EventGet (usb_ctrl_t *const *p_api_ctrl*, usb_status_t * *event*)

Obtains completed USB related events. (OS-less Only)

In USB host mode, the device address value of the USB device that completed an event is specified in the usb_ctrl_t structure member (address) specified by the event's argument. In USB peripheral mode, USB_NULL is specified in member (address). If this function is called in the RTOS execution environment, a failure is returned.

Return values

FSP_SUCCESS	Event Get Success.
FSP_ERR_USB_FAILED	If called in the RTOS environment, an error is returned.

Note

Do not use the same variable as the first argument of R_USB_Open for the first argument.
Do not call this API in the interrupt function.

◆ R_USB_Callback()

fsp_err_t R_USB_Callback (usb_callback_t * *p_callback*)

Register a callback function to be called upon completion of a USB related event. (RTOS only)

This function registers a callback function to be called when a USB-related event has completed. If this function is called in the OS-less execution environment, a failure is returned.

Return values

FSP_SUCCESS	Successfully completed.
FSP_ERR_USB_FAILED	If this function is called in the OS-less execution environment, a failure is returned.
FSP_ERR_ASSERTION	Parameter is NULL error.

Note

Do not call this API in the interrupt function.

◆ R_USB_HostControlTransfer()

fsp_err_t R_USB_HostControlTransfer (usb_ctrl_t *const *p_api_ctrl*, usb_setup_t * *p_setup*, uint8_t * *p_buf*, uint8_t *device_address*)

Performs settings and transmission processing when transmitting a setup packet.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.
FSP_ERR_USB_BUSY	Specified pipe now handling data receive/send request.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_PeriControlDataGet()

fsp_err_t R_USB_PeriControlDataGet (usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*)

Receives data sent by control transfer.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_BUSY	Specified pipe now handling data receive/send request.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_PeriControlDataSet()

fsp_err_t R_USB_PeriControlDataSet (usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_buf*, uint32_t *size*)

Performs transfer processing for control transfer.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_BUSY	Specified pipe now handling data receive/send request.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_PeriControlStatusSet()

fsp_err_t R_USB_PeriControlStatusSet (usb_ctrl_t *const *p_api_ctrl*, usb_setup_status_t *status*)

Set the response to the setup packet.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_RemoteWakeup()

fsp_err_t R_USB_RemoteWakeup (usb_ctrl_t *const *p_api_ctrl*)

Sends a remote wake-up signal to the connected Host.

Return values

FSP_SUCCESS	Successful completion.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter is NULL error.
FSP_ERR_USB_NOT_SUSPEND	Device is not suspended.
FSP_ERR_USB_BUSY	The device is in resume operation.

Note

Do not call this API in the following function.
(1). Interrupt function.
(2). Callback function (for RTOS).

◆ R_USB_ModuleNumberGet()

fsp_err_t R_USB_ModuleNumberGet (usb_ctrl_t *const *p_api_ctrl*, uint8_t * *module_number*)

This API gets the module number.

Return values

FSP_SUCCESS	Successful completion.

◆ R_USB_ClassTypeGet()

fsp_err_t R_USB_ClassTypeGet (usb_ctrl_t *const *p_api_ctrl*, usb_class_t * *class_type*)

This API gets the class type.

Return values

FSP_SUCCESS	Successful completion.

◆ R_USB_DeviceAddressGet()

fsp_err_t R_USB_DeviceAddressGet (usb_ctrl_t *const *p_api_ctrl*, uint8_t * *device_address*)

This API gets the device address.

Return values

FSP_SUCCESS	Successful completion.

◆ R_USB_PipeNumberGet()

fsp_err_t R_USB_PipeNumberGet (usb_ctrl_t *const *p_api_ctrl*, uint8_t * *pipe_number*)

This API gets the pipe number.

Return values

FSP_SUCCESS	Successful completion.

◆ R_USB_DeviceStateGet()

fsp_err_t R_USB_DeviceStateGet (usb_ctrl_t *const *p_api_ctrl*, uint16_t * *state*)

This API gets the state of the device.

Return values

FSP_SUCCESS	Successful completion.

◆ R_USB_DataSizeGet()

fsp_err_t R_USB_DataSizeGet (usb_ctrl_t *const *p_api_ctrl*, uint32_t * *data_size*)

This API gets the data size.

Return values

FSP_SUCCESS	Successful completion.

◆ R_USB_SetupGet()

fsp_err_t R_USB_SetupGet (usb_ctrl_t *const *p_api_ctrl*, usb_setup_t * *setup*)
This API gets the setup type.

Return values

FSP_SUCCESS	Successful completion.

4.2.63 USB Composite Class (r_usb_composite)

Modules

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Overview

USB composite device works as a USB Peripheral by combining two peripheral device classes and r_usb_basic module.
This USB driver supports the following composite devices:

1. PCDC + PMSC
2. PCDC + PHID
3. PHID + PMSC
4. PCDC + PCDC

How to Configuration

The following shows FSP configuration procedure for USB composite device.

- Select [New Stack]->[USB]->[USB Composite]

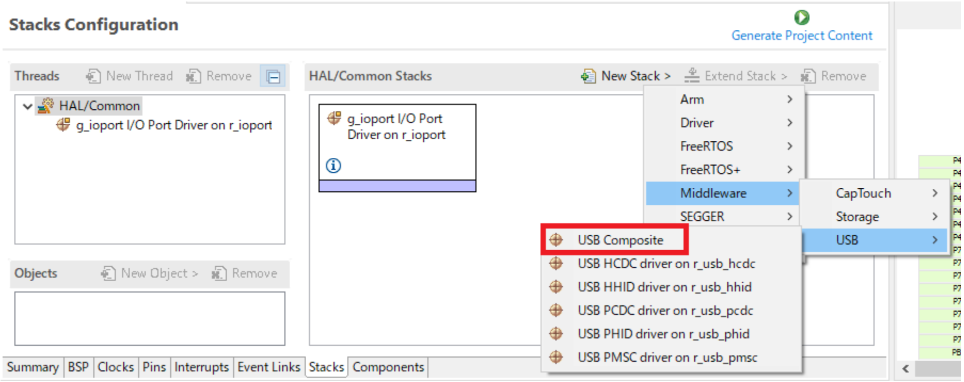


Figure 178: Select USB Composite

- The following is displayed when selecting [USB Composite].

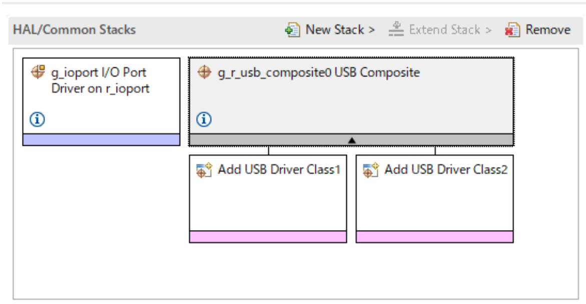


Figure 179: USB Composite Stack

- Select the supported 2 device classes as follows.

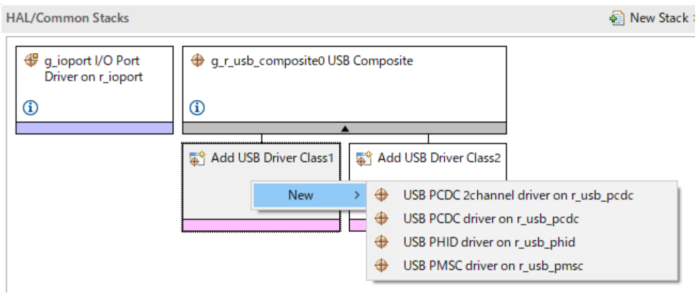


Figure 180: Select Device Classes

Note

1. *Be sure to select "USB PCDC driver on r_usb_pcdc" and "USB PCDC 2channel driver on r_usb_pcdc" when configurating for "PCDC + PCDC".*

- Select the supported 2 device classes as follows. The following is displayed when selecting 2 device classes.

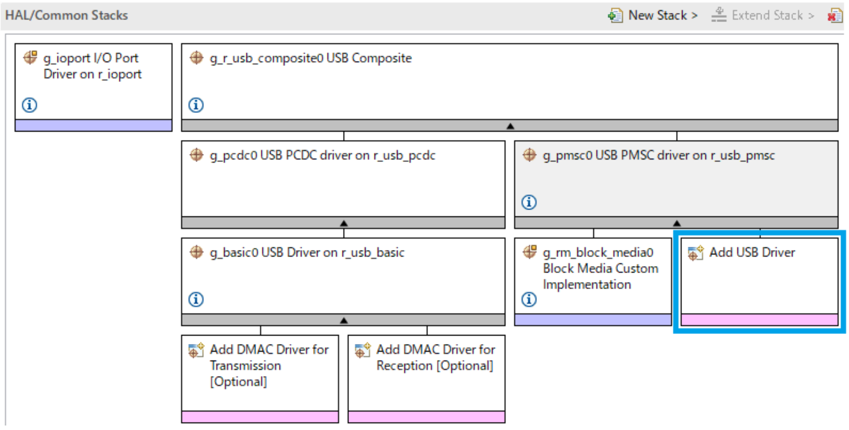


Figure 181: Delete USB Basic Instance

Note

1. *Delete the "g_basic1" instance manually since this instance is not used in composite device. (Refer to the blue frame in the above figure.)*
2. *The error is output when selecting the following device classes.*
   a. *PMSC + PMSC*
   b. *PHID + PHID*

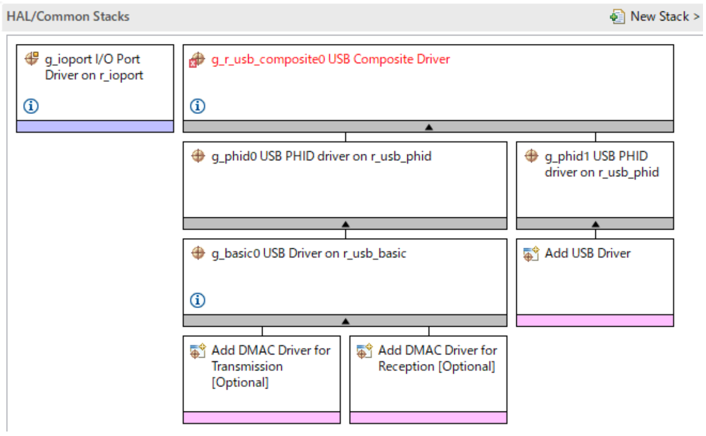


Figure 182: Device Class Selection Error

Limitations

1.The following composite device is not suppored when using RA2A1(MCU).

1. PMSC + PCDC
2. PCDC + PCDC

2.If you use PMSC, make sure to use usb_basic module with PMSC. There is a risk that the information on the PMSC storage media cannot be registered normally in the "USB Callback Context".

Notes

Please determine by the member "pipe" in "usb_event_info" structure when getting PCDC channel number which the write event is completed in PCDC + PCDC.
Don't refer to the member "type" in "usb_event_info" structure.

Descriptor

Templates for composite device descriptors can be found in ra/fsp/src/r_usb_composite folder. Also, please be sure to use your vendor ID.

1. r_usb_pcdc_pmsc_descriptor.c.template (for PCDC + PMSC)
2. r_usb_pcdc_phid_descriptor.c.template (for PCDC + PHID)
3. r_usb_phid_pmsc_descriptor.c.template (for PHID + PMSC)
4. r_usb_pcdc_pcdc_descriptor.c.template (for PCDC + PCDC)

Examples

USB COMPOSITE Example

- PCDC + PHID

```
void main_task (void)
{
  #if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
  #endif
    usb_event_info_t usb_event;
 usb_status_t    event;
    uint8_t       * p_idle_value;
    uint8_t          sw_data;
    usb_info_t       info;
 fsp_err_t       ret_code = FSP_SUCCESS;
    uint8_t          send_data[16] BSP_ALIGN_VARIABLE(4);
    uint8_t          req_comp_flag = 0;
    uint8_t          count         = 0;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    set_key_data(g_buf_phid);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
  #if (BSP_CFG_RTOS == 2)
       USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **) &p_mess);
```

```
      usb_event = *p_mess;
event    = usb_event.event;
 #else  /* (BSP_CFG_RTOS == 2) */
R_USB_EventGet(&usb_event, &event);
 #endif /* (BSP_CFG_RTOS == 2) */
switch (event)
     {
case USB_STATUS_CONFIGURED:
     {
              g_status = NO_WRITING;
              g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
break;
           }
case USB_STATUS_WRITE_COMPLETE:
     {
if (usb_event.type == USB_CLASS_PCDC)
     {
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
             }
else if (usb_event.type == USB_CLASS_PHID)
     {
if (DATA_WRITING == g_status)
     {
                      g_status = ZERO_WRITING;
                      g_usb_on_usb.write(&g_basic0_ctrl, (uint8_t *) g_zero_data,
DATA_LEN_PHID, USB_CLASS_PHID); /* Sending the zero data (8 bytes) */
             }
else if (g_status == ZERO_WRITING)
     {
                g_status = NO_WRITING;
             }
             }
break;
           }
```

```
case USB_STATUS_READ_COMPLETE:
     {
if (usb_event.type == USB_CLASS_PCDC)
     {
                  g_usb_on_usb.write(&g_basic0_ctrl, g_buf, usb_event.data_size,
USB_CLASS_PCDC);
if (req_comp_flag == 1)
     {
if (g_status == NO_WRITING)
     {
                           count++;
                           g_status = DATA_WRITING;
                           g_usb_on_usb.write(&g_basic0_ctrl, g_buf_phid,
DATA_LEN_PHID, USB_CLASS_PHID);
             }
             }
             }
break;
           }
case USB_STATUS_REQUEST:   /* Receive Class Request */
     {
if (USB_PCDC_SET_LINE_CODING == (usb_event.setup.request_type & USB_BREQUEST))
     {
R_USB_PeriControlDataGet(&g_basic0_ctrl, (uint8_t *) &g_line_coding,
LINE_CODING_LENGTH);
             }
else if (USB_PCDC_GET_LINE_CODING == (usb_event.setup.request_type & USB_BREQUEST))
     {
R_USB_PeriControlDataSet(&g_basic0_ctrl, (uint8_t *) &g_line_coding,
LINE_CODING_LENGTH);
             }
else if (USB_SET_REPORT == (usb_event.setup.request_type & USB_BREQUEST))
     {
                  g_usb_on_usb.read(&g_basic0_ctrl, (uint8_t *) &g_numlock, 2,
```

```
USB_CLASS_PHID); /* Get the NumLock data (NumLock data is not used) */
              }
 else if (USB_GET_DESCRIPTOR == (usb_event.setup.request_type & USB_BREQUEST))
      {
 if (USB_GET_REPORT_DESCRIPTOR == usb_event.setup.request_value)
      {
                        g_usb_on_usb.periControlDataSet(&g_basic0_ctrl,
                                                               (uint8_t *) g_apl_report,

USB_RECEIVE_REPORT_DESCRIPTOR);
              }
 else if (USB_GET_HID_DESCRIPTOR == usb_event.setup.request_value)
      {
 for (uint8_t i = 0; i < USB_RECEIVE_HID_DESCRIPTOR; i++)
      {
                            send_data[i] = g_apl_configuration[84 + i];
              }
 /* Configuration Descriptor address set. */
                        g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, send_data,
USB_RECEIVE_HID_DESCRIPTOR);
              }
 else
      {
                        g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
              }
              }
 else if (USB_SET_IDLE == (usb_event.setup.request_type & USB_BREQUEST))
      {
 /* Get SetIdle value */
                   p_idle_value = (uint8_t *) &usb_event.setup.request_value;
                   g_idle       = p_idle_value[1];
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
```

```
              }
 else if (USB_GET_IDLE == (usb_event.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, &g_idle, 1);
              }
 else if (USB_SET_PROTOCOL == (usb_event.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
              }
 else if (USB_GET_PROTOCOL == (usb_event.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
              }
 else
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
              }
 break;
            }
 case USB_STATUS_REQUEST_COMPLETE: /* Complete Class Request */
      {
 if (USB_SET_IDLE == (usb_event.setup.request_type & USB_BREQUEST))
      {
                   p_idle_value = (uint8_t *) &usb_event.setup.request_value;
                   g_idle = p_idle_value[1];
              }
 else if (USB_SET_PROTOCOL == (usb_event.setup.request_type & USB_BREQUEST))
      {
 /* None */
 /* g_protocol = event_info.setup.value; */
              }
```

```
else
     {
                    req_comp_flag = 1;
              }
break;
            }
case USB_STATUS_SUSPEND:
case USB_STATUS_DETACH:
     {
break;
            }
default:
     {
break;
            }
       }
   }
} /* End of function usb_main() */
void set_key_data (uint8_t * p_buf)
{
 static uint8_t key_data;
   key_data = KBD_CODE_A;
   *(p_buf + 2) = key_data;
}
  #if (BSP_CFG_RTOS == 2)
/******************************************************************************
 * Function Name : usb_apl_rec_msg
 * Description : Receive a message to the specified id (mailbox).
 * Argument : uint8_t id : ID number (mailbox).
 * : usb_msg_t** mess : Message pointer
 * : usb_tm_t tm : Timeout Value
 * Return : uint16_t : USB_OK / USB_ERROR
 ******************************************************************************/
usb_er_t usb_apl_rec_msg (uint8_t id, usb_msg_t ** mess, usb_tm_t tm)
```

```
{
    BaseType_t    err;
    QueueHandle_t handle;
    usb_er_t      result;
    (void) tm;
 if (NULL == mess)
    {
 return USB_APL_ERROR;
    }
    handle = (*(g_apl_mbx_table[id]));
    *mess = NULL;
    err = xQueueReceive(handle, (void *) mess, (portMAX_DELAY));
 if ((pdTRUE == err) && (NULL != (*mess)))
    {
       result = USB_APL_OK;
    }
 else
    {
       result = USB_APL_ERROR;
    }
 return result;
}
/******************************************************************************
 * End of function usb_apl_rec_msg
 ******************************************************************************/
/******************************************************************************
 * Function Name : usb_apl_snd_msg
 * Description : Send a message to the specified id (mailbox).
 * Argument : uint8_t id : ID number (mailbox).
 * : usb_msg_t* mess : Message pointer
 * Return : usb_er_t : USB_OK / USB_ERROR
 ******************************************************************************/
usb_er_t usb_apl_snd_msg (uint8_t id, usb_msg_t * mess)
{
```

```
    BaseType_t    err;
    QueueHandle_t handle;
    usb_er_t      result;
 if (NULL == mess)
    {
 return USB_APL_ERROR;
    }
    handle = (*(g_apl_mbx_table[id]));
    err = xQueueSend(handle, (const void *) &mess, (TickType_t) (0));
 if (pdTRUE == err)
    {
       result = USB_APL_OK;
    }
 else
    {
       result = USB_APL_ERROR;
    }
 return result;
}
/*******************************************************************************
 * End of function usb_apl_snd_msg
 ******************************************************************************/
  #endif  /* #if (BSP_CFG_RTOS == 2) */
```

4.2.64 USB Host Communications Device Class Driver (r_usb_hcdc)

Modules

This module provides a USB Host Communications Device Class (HCDC) driver. It implements the USB HCDC Interface.

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Detailed Description

Overview

The r_usb_hcdc module, when used in combination with the r_usb_basic module, operates as a USB Host Communications Device Class (HCDC) driver. The HCDC conforms to the PSTN device subclass abstract control model of the USB Communications Device Class (CDC) specification and enables communication with a CDC peripheral device.

Features

The r_usb_hcdc module has the following key features:

- Checks for connected devices
- Implementation of communication line settings
- Acquisition of the communication line state
- Data transfer to and from a CDC peripheral device

Configuration

Build Time Configurations for r_usb_hcdc

The following build time configurations are defined in fsp_cfg/r_usb_hcdc_cfg.h:

Configuration	Options	Default	Description
Target Peripheral Device Class ID	• CDC class supported device • Vendor class device	CDC class supported device	Specify the device class ID of the CDC device to be connected.
Bulk Input Transfer Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE1	Select the USB pipe to use for bulk input transfers.
Bulk Output Transfer Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE2	Select the USB pipe to use for bulk output transfers.
Interrupt In Pipe	• USB PIPE6 • USB PIPE7 • USB PIPE8 • USB PIPE9	USB PIPE6	Select the USB pipe to use for interrupts.

Configurations for Connectivity > USB HCDC (r_usb_hcdc)

This module can be added to the Stacks tab via New Stack > Connectivity > USB HCDC (r_usb_hcdc).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_hcdc0	Module name.

Note

Refer to the USB (r_usb_basic) module for hardware configuration options.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Communications Device Class (CDC), PSTN and ACM

This software conforms to the Abstract Control Model (ACM) subclass of the Communications Device Class specification as defined in the "USB Communications Class Subclass Specification for PSTN Devices", Revision 1.2. The Abstract Control Model subclass is a technology that bridges the gap between USB devices and earlier modems (employing RS-232C connections) enabling use of application programs designed for older modems.

Basic Functions

The main functions of HCDC are the following:

- Verify connected devices
- Make communication line settings
- Acquire the communication line state
- Transfer data to and from the CDC peripheral device

Abstract Control Model Class Requests - Host to Device

This driver supports the following class requests:

Request	Code	Description
SendEncapsulatedCommand	0x00	Transmits an AT command as defined by the protocol used by the device (normally 0 for USB).
GetEncapsulatedResponse	0x01	Requests a response to a command transmitted by SendEncapsulatedCommand.
SetCommFeature	0x02	Enables or disables features such as device-specific 2-byte code and country setting.
GetCommFeature	0x03	Acquires the enabled/disabled state of features such as device-specific 2-byte code and country setting.
ClearCommFeature	0x04	Restores the default enabled/disabled settings of

		features such as device-specific 2-byte code and country setting.
SetLineCoding	0x20	Makes communication line settings (communication speed, data length, parity bit, and stop bit length).
GetLineCoding	0x21	Acquires the communication line setting state.
SetControlLineState	0x22	Makes communication line control signal (RTS, DTR) settings.
SendBreak	0x23	Transmits a break signal.

Note

For more information about Abstract Control Model requests, refer to Table 11 "Requests - Abstract Control Model" in the "USB Communications Class Subclass Specification for PSTN Devices", Revision 1.2.

The expected data format for each command is shown below followed by dependent structures.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SEND_ENCAPSULATED_COMMAND (0x00)	0x0000	0x0000	Data length	usb_hcdc_encapsulated_t
0x21	GET_ENCAPSULATED_RESPONSE (0x01)	0x0000	0x0000	Data length	usb_hcdc_encapsulated_t
0x21	SET_COMM_FEATURE (0x02)	usb_hcdc_feature_selector_t	0x0000	Data length	usb_hcdc_commfeature_t
0x21	GET_COMM_FEATURE (0x03)	usb_hcdc_feature_selector_t	0x0000	Data length	usb_hcdc_commfeature_t
0x21	CLEAR_COMM_FEATURE (0x04)	usb_hcdc_feature_selector_t	0x0000	Data length	None
0x21	SET_LINE_CODING (0x20)	0x0000	0x0000	0x0000	usb_hcdc_linecoding_t
0xA1	GET_LINE_CODING (0x21)	0x0000	0x0000	0x0007	usb_hcdc_linecoding_t
0x21	SET_CONTROL_LINE_STATE (0x22)	usb_hcdc_controllinestate_t	0x0000	0x0000	None
0x21	SEND_BREAK (0x23)	usb_hcdc_breakduration_t	0x0000	0x0000	None

ACM Notifications from Device to Host

The following class notifications are supported:

Notification	Code	Description
RESPONSE_AVAILABLE	0x01	Response to GET_ENCAPSULATED_RESPONSE
SERIAL_STATE	0x20	Notification of serial line state

The data types returned are as follows:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	RESPONSE_AVAILABLE (0x01)	0x0000	0x0000	0x0000	None
0xA1	SERIAL_STATE (0x20)	0x0000	0x0000	0x0002	usb_hcdc_serialstate_t

Note

The host is notified with SERIAL_STATE whenever a change in the UART port state is detected.

Limitations

This driver is subject to the following limitations:

- Suspend is not supported when a data transfer is in progress. Confirm that data transfer has completed before executing suspend.
- Use of compound USB devices with CDC class support is not supported.
- This module must be incorporated into a project using r_usb_basic and does not provide any public APIs.
- This driver does not support Low-speed.
- This driver does not support simultaneous operation with the other device class.

Examples

USB HCDC Loopback Example

The main functions of the HCDC loopback example are as follows:

1. Virtual UART control settings are configured by transmitting the class request SET_LINE_CODING to the CDC device.
2. Sends receive (Bulk In transfer) requests to a CDC peripheral device and receives data.
3. Loops received data back to the peripheral by means of Bulk Out transfers.

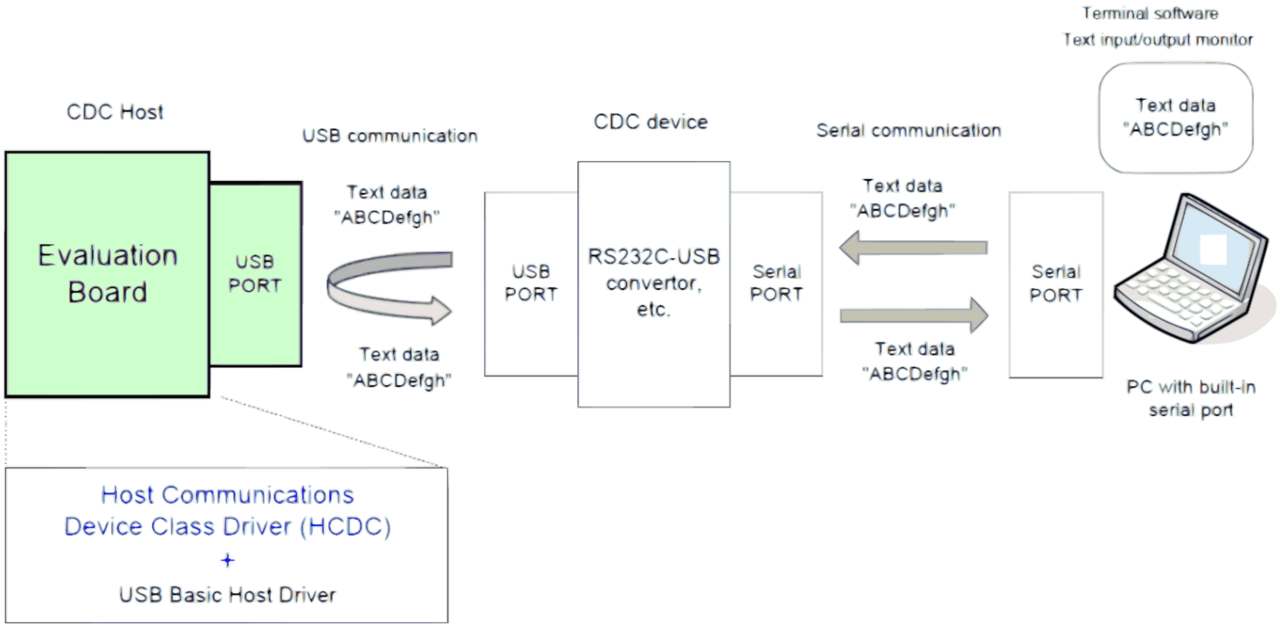


Figure 183: Data Transfer (Loopback)

The main loop performs loopback processing in which data received from a CDC peripheral device is transmitted unaltered back to the peripheral.

```
#define SET_LINE_CODING (USB_CDC_SET_LINE_CODING | USB_HOST_TO_DEV | USB_CLASS |
USB_INTERFACE)
#define GET_LINE_CODING (USB_CDC_GET_LINE_CODING | USB_DEV_TO_HOST | USB_CLASS |
USB_INTERFACE)
#define SET_CONTROL_LINE_STATE (USB_CDC_SET_CONTROL_LINE_STATE | USB_HOST_TO_DEV |
USB_CLASS | USB_INTERFACE)
#define COM_SPEED (9600U)
#define COM_DATA_BIT (8U)
#define COM_STOP_BIT (0)
#define COM_PARITY_BIT (0)
#define LINE_CODING_LENGTH (7)
void usb_basic_example (void)
{
 usb_status_t    event;
    usb_event_info_t event_info;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
```

```
 /* Get USB event data */
       g_usb_on_usb.eventGet(&event_info, &event);
 /* Handle the received event (if any) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
 /* Configure virtual UART settings */
                set_line_coding(&g_basic0_ctrl, event_info.device_address); /* CDC
Class request "SetLineCoding" */
 break;
 case USB_STATUS_READ_COMPLETE:
 if (USB_CLASS_HCDC == event_info.type)
      {
 if (event_info.data_size > 0)
      {
 /* Send the received data back to the connected peripheral */
                        g_usb_on_usb.write(&g_basic0_ctrl, g_snd_buf,
event_info.data_size, USB_DEVICE_ADDRESS_1);
                   }
 else
      {
 /* Send the data reception request when the zero-length packet is received. */
                        g_usb_on_usb.read(&g_basic0_ctrl, g_rcv_buf, CDC_DATA_LEN,
USB_DEVICE_ADDRESS_1);
                   }
                }
 else /* USB_HCDCC */
      {
 /* Control Class notification "SerialState" receive start */
                   g_usb_on_usb.read(&g_basic0_ctrl,
                                     (uint8_t *) &g_serial_state,
                                     USB_HCDC_SERIAL_STATE_MSG_LEN,
                                     USB_DEVICE_ADDRESS_1);
                }
```

```
 break;
 case USB_STATUS_WRITE_COMPLETE:
 /* Start receive operation */
                g_usb_on_usb.read(&g_basic0_ctrl, g_rcv_buf, CDC_DATA_LEN,
USB_DEVICE_ADDRESS_1);
 break;
 case USB_STATUS_REQUEST_COMPLETE:
 if (USB_CDC_SET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Set virtual RTS/DTR signal state */
                   set_control_line_state(&g_basic0_ctrl, event_info.device_address);
/* CDC Class request "SetControlLineState" */
                }
 /* Check Complete request "SetControlLineState" */
 else if (USB_CDC_SET_CONTROL_LINE_STATE == (event_info.setup.request_type &
USB_BREQUEST))
      {
 /* Read back virtual UART settings */
                   get_line_coding(&g_basic0_ctrl, event_info.device_address); /* CDC
Class request "SetLineCoding" */
                }
 else if (USB_CDC_GET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Now that setup is complete, start loopback operation */
                   g_usb_on_usb.read(&g_basic0_ctrl, g_snd_buf, CDC_DATA_LEN,
USB_DEVICE_ADDRESS_1);
                }
 else
      {
 /* Unsupported request */
                }
 break;
 default:
 /* Other event */
```

```
 break;
      }
    }
}
void set_control_line_state (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
    usb_setup_t setup;
    setup.request_type   = SET_CONTROL_LINE_STATE; /*
bRequestCode:SET_CONTROL_LINE_STATE, bmRequestType */
    setup.request_value = 0x0000;                /* wValue:Zero */
    setup.request_index = 0x0000;                /* wIndex:Interface */
    setup.request_length = 0x0000;                /* wLength:Zero */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_usb_dummy,
device_address);
}
void set_line_coding (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
    usb_setup_t setup;
    g_com_parm.dwdte_rate   = (usb_hcdc_line_speed_t) COM_SPEED;
    g_com_parm.bchar_format = (usb_hcdc_stop_bit_t) COM_STOP_BIT;
    g_com_parm.bparity_type = (usb_hcdc_parity_bit_t) COM_PARITY_BIT;
    g_com_parm.bdata_bits   = (usb_hcdc_data_bit_t) COM_DATA_BIT;
    setup.request_type   = SET_LINE_CODING;    /* bRequestCode:SET_LINE_CODING,
bmRequestType */
    setup.request_value = 0x0000;             /* wValue:Zero */
    setup.request_index = 0x0000;             /* wIndex:Interface */
    setup.request_length = LINE_CODING_LENGTH; /* Data:Line Coding Structure */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_com_parm,
device_address);
}
void get_line_coding (usb_instance_ctrl_t * p_ctrl, uint8_t device_address)
{
    usb_setup_t setup;
```

```
    setup.request_type   = GET_LINE_CODING;    /* bRequestCode:GET_LINE_CODING,
bmRequestType */
    setup.request_value = 0x0000;             /* wValue:Zero */
    setup.request_index = 0x0000;             /* wIndex:Interface */
    setup.request_length = LINE_CODING_LENGTH; /* Data:Line Coding Structure */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_com_parm,
device_address);
}
```

4.2.65 USB Host Human Interface Device Class Driver (r_usb_hhid)

Modules

Functions

fsp_err_t	R_USB_HHID_TypeGet (usb_ctrl_t *const p_api_ctrl, uint8_t *p_type, uint8_t device_address) Get HID protocol.(USB Mouse/USB Keyboard/Other Type.) More...
fsp_err_t	R_USB_HHID_MaxPacketSizeGet (usb_ctrl_t *const p_api_ctrl, uint16_t *p_size, uint8_t direction, uint8_t device_address) Obtains max packet size for the connected HID device. The max packet size is set to the area. Set the direction (USB_HID_IN/USB_HID_OUT). More...

Detailed Description

This module provides a USB Host Human Interface Device Class Driver (HHID). It implements the USB HHID Interface.

Overview

The r_usb_hhid module combines with the r_usb_basic module to provide a USB Host Human Interface Device Class (HHID) driver. The HHID driver conforms to the USB Human Interface Device class specifications and implements communication with a HID device.

Features

The r_usb_hhid module has the following key features:

- Data communication with a connected HID device (USB mouse, keyboard etc.)
- Issuing of HID class requests to a connected HID device
- Supports Interrupt OUT transfer

Configuration

Build Time Configurations for r_usb_hhid

The following build time configurations are defined in fsp_cfg/r_usb_hhid_cfg.h:

Configuration	Options	Default	Description
Interrupt In Pipe	• USB PIPE6 • USB PIPE7 • USB PIPE8 • USB PIPE9	USB PIPE6	Select the pipe number to use for input interrupt events.
Interrupt Out Pipe	• USB PIPE6 • USB PIPE7 • USB PIPE8 • USB PIPE9	USB PIPE9	Select the pipe number to use for output interrupt events.

Configurations for Connectivity > USB HHID (r_usb_hhid)

This module can be added to the Stacks tab via New Stack > Connectivity > USB HHID (r_usb_hhid). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_hhid0	Module name.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Note

This driver is not guaranteed to provide USB HID operation in all scenarios. The developer must verify correct operation when connected to the targeted USB peripherals.

Class Requests

The class requests supported by this driver are shown below:

Request	Code	Description
USB_GET_REPORT	0x01	Receives a report from the HID device.

USB_SET_REPORT	0x09	Sends a report to the HID device.
USB_GET_IDLE	0x02	Receives a duration (time) from the HID device.
USB_SET_IDLE	0x0A	Sends a duration (time) to the HID device.
USB_GET_PROTOCOL	0x03	Reads a protocol from the HID device.
USB_SET_PROTOCOL	0x0B	Sends a protocol to the HID device.
USB_GET_REPORT_DESCRIPTOR	0x06	Requests a report descriptor.
USB_GET_HID_DESCRIPTOR	0x06	Requests a HID descriptor.

Data Format

The boot protocol data format of data received from the keyboard or mouse through interrupt-IN transfers is shown below:

offset	Keyboard (8 Bytes)	Mouse (3 Bytes)
0 (Top Byte)	Modifier keys	b0 : Button 1 b1 : Button 2 b2 : Button 3 b3-b7 : Reserved
+1	Reserved	X displacement
+2	Keycode 1	Y displacement
+3	Keycode 2	-
+4	Keycode 3	-
+5	Keycode 4	-
+6	Keycode 5	-
+7	Keycode 6	-

Limitations

- The HID driver does not analyze the report descriptor. This driver determines the report format from the interface protocol.
- This driver does not support DMA transfers.
- This driver does not support High-speed.
- The transfer rates of Full-speed and Low-speed are the same when the max packet sizes of Full-speed and Low-speed are the same.
- This driver does not support simultaneous operation with the other device class.

Examples

USB HHID Example

The main functions of the application are as follows:

1. Performs enumeration and initialization of HID devices.
2. Transfers data to and from a connected HID device (mouse or keyboard). Data received from the device is read and discarded.
3. When an RTOS is used, the USB driver calls the callback (usb_apl_callback) in order to pass events to the main loop through a queue.

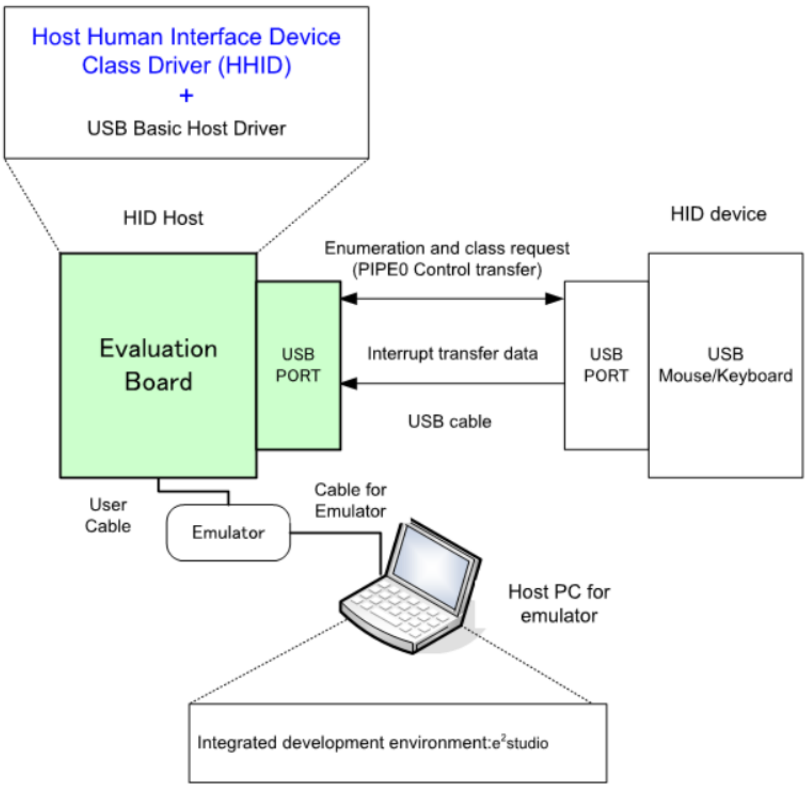


Figure 184: Example Operating Environment

Application Processing (for RTOS)

The main loop performs processing to receive data from the HID device as part of the main routine. An overview of the processing performed by the loop is shown below.

1. When a USB-related event has completed, the USB driver calls the callback function (usb_apl_callback). In the callback function (usb_apl_callback), the application task (APL) is notified of the USB completion event using the real-time OS functionality.
2. In APL, information regarding the USB completion event was notified from the callback function is retrieved using the real-time OS functionality.
3. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STATUS_CONFIGURED, APL sends the class request (SET_PROTOCOL) to the HID device.
4. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STATUS_REQUEST_COMPLETE, APL performs a data reception request to receive data transmitted from the HID device by calling the R_USB_Read function.
5. The above processing is repeated.

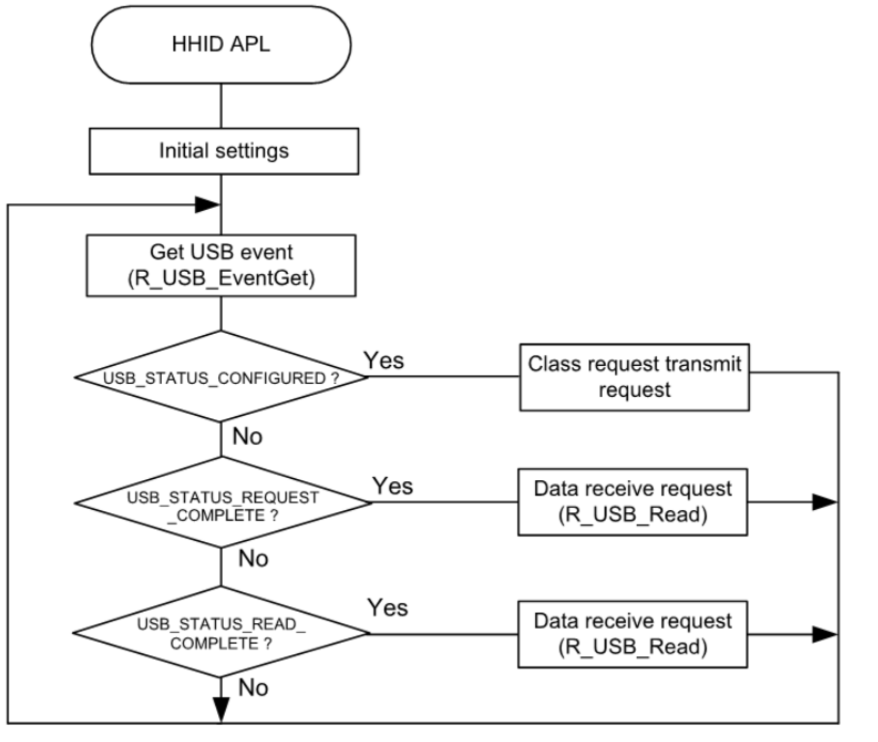


Figure 185: Main Loop (Normal mode)

Application Processing (for Non-OS)

The main loop performs processing to receive data from the HID device as part of the main routine. An overview of the processing of the main loop is presented below.

1. When the R_USB_GetEvent function is called after an HID device attaches to the USB host and enumeration completes, USB_STATUS_CONFIGURED is set as the return value. When the APL confirms USB_STATUS_CONFIGURED, it calls the R_USB_Write function to request transmission of data to the HID device.
2. When the R_USB_GetEvent function is called after sending of class request SET_PROTOCOL to the HID device has completed, USB_STATUS_REQUEST_COMPLETE is set as the return value. When the APL confirms USB_STATUS_REQUEST_COMPLETE, it calls the R_USB_Read function to make a data receive request for data sent by the HID device.
3. When the R_USB_GetEvent function is called after reception of data from the HID device has completed, USB_STATUS_READ_COMPLETE is set as the return value. When the APL confirms USB_STATUS_READ_COMPLETE, it calls the R_USB_Read function to make a data receive request for data sent by the HID device.
4. The processing in step 3, above, is repeated.

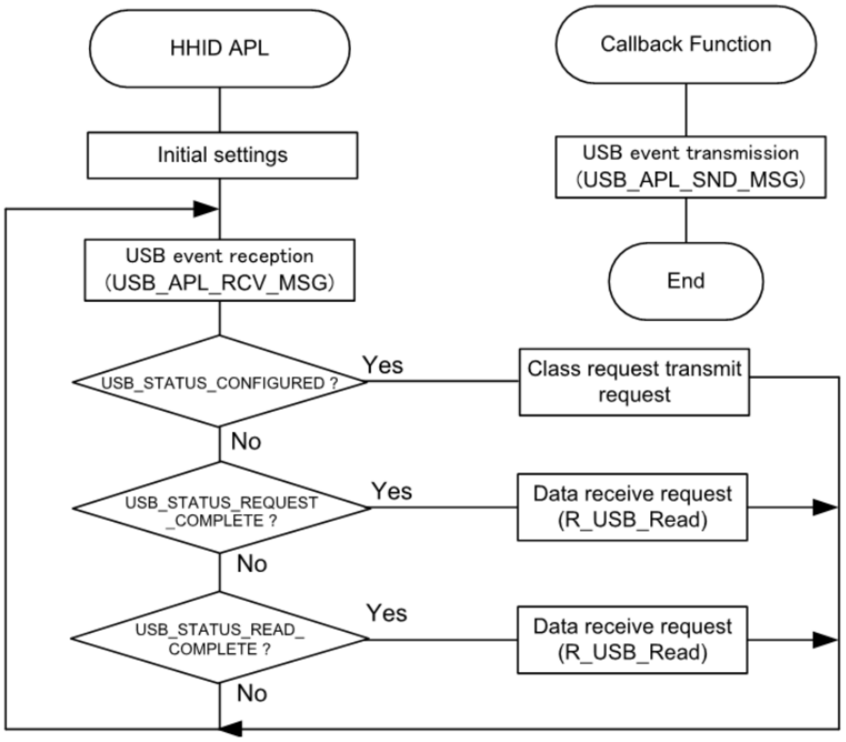


Figure 186: Main Loop (Normal mode)

```
/******************************************************************************
 * Macro definitions
 ******************************************************************************/
#define SET_PROTOCOL (USB_HID_SET_PROTOCOL | USB_HOST_TO_DEV | USB_CLASS |
USB_INTERFACE)
#define BOOT_PROTOCOL (0)
#define USB_FS_DEVICE_ADDRESS_1 (1)
/******************************************************************************
 * Private global variables and functions
 ******************************************************************************/
static const usb_hhid_api_t g_hhid_on_usb =
{
    .typeGet           = R_USB_HHID_TypeGet,
    .maxPacketSizeGet = R_USB_HHID_MaxPacketSizeGet,
};
/******************************************************************************
 * Function Name : r_usb_hhid_example
 * Description : Host HID application main process
```

```
 * Arguments : none
 * Return value : none
 ******************************************************************************/
static void r_usb_hhid_example (void)
{
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif  /* (BSP_CFG_RTOS == 2) */
 usb_status_t    event;
    usb_event_info_t event_info;
    uint16_t         offset = 0;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
#if (BSP_CFG_RTOS == 2)
       USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **) &p_mess);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&event_info, &event); /* Get event code */
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
      {
                g_hhid_on_usb.typeGet(&g_basic0_ctrl, &g_hid_type,
USB_FS_DEVICE_ADDRESS_1);
                g_hhid_on_usb.maxPacketSizeGet(&g_basic0_ctrl, &g_mxps, USB_HID_IN,
USB_FS_DEVICE_ADDRESS_1);
 /* Send the HID request (SetProtocol) to HID device */
                set_protocol(&g_basic0_ctrl, BOOT_PROTOCOL, USB_FS_DEVICE_ADDRESS_1);
 break;
            }
 case USB_STATUS_READ_COMPLETE:
```

```
      {
                offset = hid_memcpy(g_store_buf, g_buf, offset, g_mxps);
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, (uint32_t) g_mxps,
USB_FS_DEVICE_ADDRESS_1);
 break;
            }
 case USB_STATUS_REQUEST_COMPLETE:
      {
 if (USB_HID_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
                  g_usb_on_usb.read(&g_basic0_ctrl, g_buf, (uint32_t) g_mxps,
USB_FS_DEVICE_ADDRESS_1);
              }
 break;
            }
 default:
      {
 break;
            }
       }
    }
} /* End of function usb_main */
/*******************************************************************************
 * Function Name : set_protocol
 * Description : Sending SetProtocol request to HID device
 * Arguments : usb_ctrl_t *p_ctrl : Pointer to usb_instance_ctrl_t structure.
 * : uint8_t ptorocol: Protocol Type
 * : uint8_t device_address: Device address that sends this request
 * Return value : none
 *******************************************************************************/
static void set_protocol (usb_instance_ctrl_t * p_ctrl, uint8_t protocol, uint8_t
device_address)
{
    usb_setup_t setup;
```

```
    setup.request_type   =
SET_PROTOCOL;                                                        /*
bRequestCode:SET_PROTOCOL, bmRequestType */
    setup.request_value =
protocol;                                                             /* wValue: Protocol
Type */
    setup.request_index =
0x0000;                                                              /* wIndex:Interface */
    setup.request_length =
0x0000;                                                              /* wLength:Zero */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, (uint8_t *) &g_setup_data,
device_address); /* Request Control transfer */
} /* End of function set_protocol */
/******************************************************************************
 * Function Name : hid_memcpy
 * Description : Copy received hhid data to the application buffer
 * Arguments : uint8_t *p_dest : Pointer to application buffer
 * : uint8_t *p_src : Pointer to received buffer
 * : uint16_t offset : Application buffer offset
 * : uint16_t size : Size of receiced hhid data
 * Return value : uint16_t offset + i: Offset
 ******************************************************************************/
static uint16_t hid_memcpy (uint8_t * p_dest, uint8_t * p_src, uint16_t offset,
uint16_t size)
{
    uint16_t i;
 for (i = 0; i < size; i++)
    {
 if ((offset + i) == BUFSIZE)
      {
            offset = 0;
       }
       *(p_dest + offset + i) = *(p_src + i);
    }
```

```
 return (uint16_t) (offset + i);
} /* End of function hid_memcpy */
```

Function Documentation

◆ R_USB_HHID_TypeGet()

fsp_err_t R_USB_HHID_TypeGet (usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_type*, uint8_t *device_address*)

Get HID protocol.(USB Mouse/USB Keyboard/Other Type.)

Return values

FSP_SUCCESS	Success.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter Null pointer error.
FSP_ERR_USB_PARAMETER	Parameter error.

◆ R_USB_HHID_MaxPacketSizeGet()

fsp_err_t R_USB_HHID_MaxPacketSizeGet (usb_ctrl_t *const *p_api_ctrl*, uint16_t * *p_size*, uint8_t *direction*, uint8_t *device_address*)

Obtains max packet size for the connected HID device. The max packet size is set to the area. Set the direction (USB_HID_IN/USB_HID_OUT).

Return values

FSP_SUCCESS	Success.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter Null pointer error.
FSP_ERR_USB_PARAMETER	Parameter error.

4.2.66 USB Host Mass Storage Class Driver (r_usb_hmsc)

Modules

Functions

fsp_err_t	R_USB_HMSC_StorageCommand (usb_ctrl_t *const p_api_ctrl, uint8_t

	*buf, uint8_t command, uint8_t destination) Processing for MassStorage(ATAPI) command. More...
fsp_err_t	R_USB_HMSC_DriveNumberGet (usb_ctrl_t *const p_api_ctrl, uint8_t *p_drive, uint8_t destination) Get number of Storage drive. More...
fsp_err_t	R_USB_HMSC_SemaphoreGet (void) Get a semaphore. (RTOS only) More...
fsp_err_t	R_USB_HMSC_SemaphoreRelease (void) Release a semaphore. (RTOS only) More...
fsp_err_t	R_USB_HMSC_StorageReadSector (uint16_t drive_number, uint8_t *const buff, uint32_t sector_number, uint16_t sector_count) Read sector information. More...
fsp_err_t	R_USB_HMSC_StorageWriteSector (uint16_t drive_number, uint8_t const *const buff, uint32_t sector_number, uint16_t sector_count) Write sector information. More...

Detailed Description

This module provides a USB Host Mass Storage Class (HMSC) driver. It implements the USB HMSC Interface.

Overview

The r_usb_hmsc module, when used in combination with the r_usb_basic module, operates as a USB Host Mass Storage Class (HMSC) driver. It is built on the USB Mass Storage Class Bulk-Only Transport (BOT) protocol. It is possible to communicate with BOT-compatible USB storage devices by combining this module with a file system and storage device driver.

Note

This module should be used in combination with the FreeRTOS+FAT File System.

Features

The r_usb_hmsc module has the following key features:

- Checking of connected USB storage devices to determine whether or not operation is supported

- Storage command communication using the BOT protocol
- Support for SFF-8070i (ATAPI) USB mass storage subclass
- Sharing of a single pipe for IN/OUT directions or multiple devices
- Supports up to 4 connected USB storage devices

Class Requests

The class requests supported by this driver are shown below.

Request	Description
GetMaxLun	Gets the maximum number of units that are supported.
MassStorageReset	Cancels a protocol error.

Storage Commands

This driver supports the following storage commands:

- TEST_UNIT_READY
- MODE_SELECT10
- MODE_SENSE10
- PREVENT_ALLOW
- READ_FORMAT_CAPACITY
- READ10
- WRITE10

Configuration

Refer to the USB (r_usb_basic) module.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Warning
Due to the wide variety of USB mass storage device implementations, this driver is not guaranteed to work with all devices. When implementing the driver it is important to verify correct operation with the mass storage devices that the end user is expected to use.

Multi Port

This driver supports simultaneous operation with Peripheral Communication Device Class(PCDC). If the user are using MCU that supports 2 USB modules, such as RA6M3, the user can run HMSC on one USB module and PCDC on the other. This driver does not support simultaneous operation using device classes other than PCDC.

For Bare Metal

1. To use FreeRTOS+FAT without FreeRTOS, copy FreeRTOSConfigMinimal.h to one of your project's include paths and rename it FreeRTOSConfig.h.
2. In RA configurator, enter the appropriate values in the Main stack size and Heap size fields. The figure below is an example of the RA6M3-EK board.

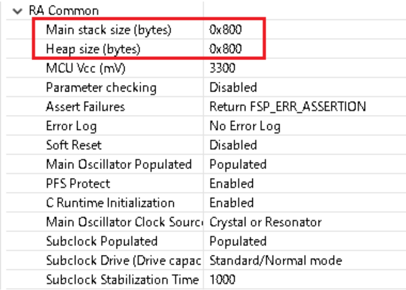


Figure 187: BSP Setting

1. In the Bare Metal version, specify "NULL" in the Callback item.

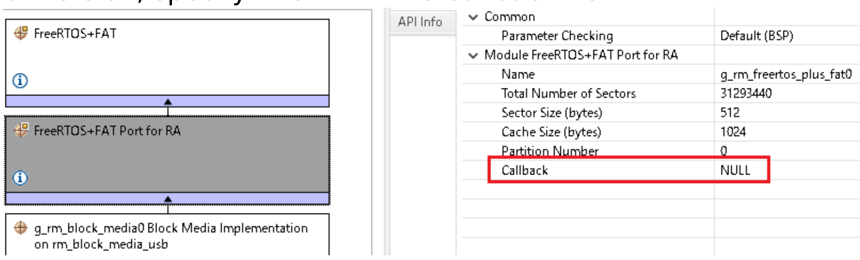


Figure 188: For Bare Metal Setting

Limitations

1. Some MSC devices may be unable to connect because they are not recognized as storage devices.
2. MSC devices that return values of 1 or higher in response to the GetMaxLun command (mass storage class command) are not supported.
3. A maximum of 4 USB storage devices can be connected.
4. Only USB storage devices with a sector size of 512 bytes can be connected.
5. A device that does not respond to the READ_CAPACITY command operates as a device with a sector size of 512 bytes.
6. The continuous transfer mode cannot be used when using DMA.
7. This module must be incorporated into a project using r_usb_basic and does not provide any public APIs.
8. This driver does not support Low-speed.

Examples

USB HMSC Example

Example Operating Environment

The following shows an example operating environment for the HMSC.

Refer to the associated instruction manuals for details on setting up the evaluation board and using the emulator, etc.

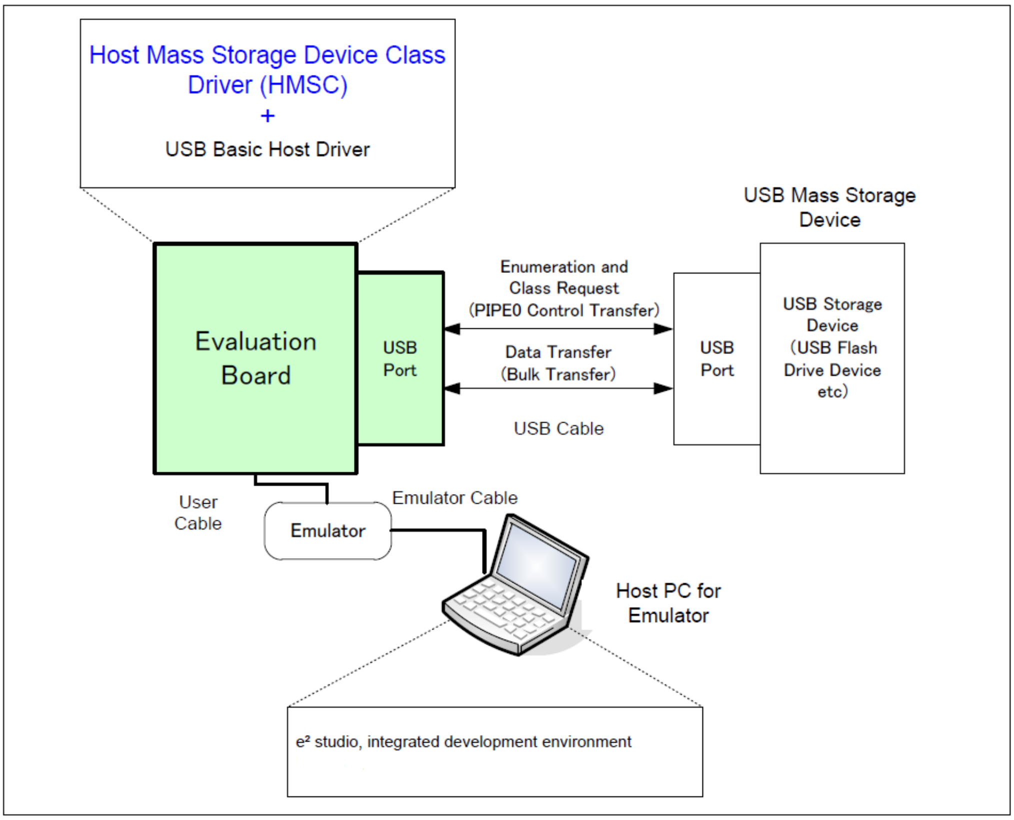


Figure 189: Example Operating Environment

Application Specifications

The main functions of the application are as follows:

1. Performs enumeration and drive recognition processing on MSC devices.
2. After the above processsing finisihes, the application writes the file to the MSC device once.
3. After writing the above file, the APL repeatedly reads the file. It continues to read the file repeatedly until the switch is pressed again.

Application Processing (for RTOS)

This application has two tasks. An overview of the processing in these two tasks is provided below.

usb_apl_task

1. After start up, MCU pin setting, USB controller initialization, and application program initialization are performed.
2. The MSC device is attached to the kit. When enumeration and drive recognition processing have completed, the USB driver calls the callback function (usb_apl_callback). In the callback function (usb_apl_callback), the application task is notified of the USB completion event using the FreeRTOS functionality.
3. In the application task, information regarding the USB completion event about which notification was received from the callback function is retrieved using the real-time OS

functionality.
4. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_CONFIGURED then, based on the USB completion event, the MSC device is mounted and the file is written to the MSC device.
5. If the USB completion event (the event member of the usb_ctrl_t structure) retrieved in step 2 above is USB_STS_DETACH, the application initializes the variables for state management.

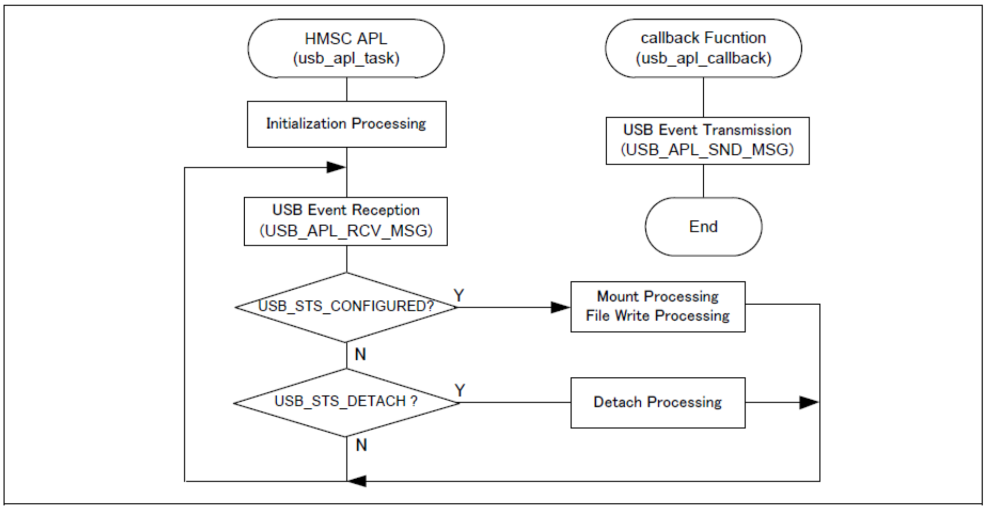


Figure 190: usb_apl_task

file_read_task

Of the application tasks usb_apl_task and file_read_task, file_read_task is processed while usb_apl_task is in the wait state. This task performs file read processing on the file that was written to the MSC device.

Example Code

```
 #define RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME "TEST_FILE.txt"
 #define RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES (10240)
 #define RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER (0)
 #define RM_FREERTOS_PLUS_FAT_EXAMPLE_SUPPORT_USB
typedef enum
{
    STATE_ATTACH, STATE_DATA_READY, STATE_DATA_WRITE, STATE_FILE_READ, STATE_DETACH,
STATE_ERROR,
} state_t;
extern rm_freertos_plus_fat_instance_ctrl_t g_rm_freertos_plus_fat0_ctrl;
extern const rm_freertos_plus_fat_cfg_t g_rm_freertos_plus_fat0_cfg;
// @@extern const rm_freertos_plus_fat_disk_cfg_t g_rm_freertos_plus_fat0_disk_cfg;
uint8_t g_file_data[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
```

```
uint8_t g_read_buffer[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
static uint16_t g_state = STATE_DETACH;
void usb_hmsc_baremetal_example (void)
{
    uint16_t i;
    uint16_t k;
 fsp_err_t err;
    FF_FILE * pxSourceFile;
    FF_Disk_t disk;
 rm_freertos_plus_fat_device_t device;
 usb_status_t    event;
    usb_event_info_t event_info;
    FF_Error_t       ff_err;
 size_t          size_return;
 int             close_err;
 rm_block_media_usb_instance_ctrl_t * p_instance_ctrl;
 for (i = 0; i < RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES; i++)
    {
       g_file_data[i] = (uint8_t) i;
    }
 /* Open media driver.*/
 RM_FREERTOS_PLUS_FAT_Open(&g_rm_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat0_cfg);
/* When using USB media, enable RM_FREERTOS_PLUS_FAT_EXAMPLE_SUPPORT_USB macro. */
 #ifdef RM_FREERTOS_PLUS_FAT_EXAMPLE_SUPPORT_USB
 while (1)
    {
       g_usb_on_usb.eventGet(&event_info, &event);
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
      {
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
```

```
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
                p_instance_ctrl                = event_info.p_context;
                p_instance_ctrl->device_address = event_info.device_address;
 RM_FREERTOS_PLUS_FAT_MediaInit(&g_rm_freertos_plus_fat0_ctrl, &device);
 /* Initialize one disk for each partition used in the application. */
 RM_FREERTOS_PLUS_FAT_DiskInit(&g_rm_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat0_disk_cfg, &disk);
 /* Mount each disk. This assumes the disk is already partitioned and formatted. */
                FF_Mount(&disk, RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
 /* Add the disk to the file system. */
                FF_FS_Add("/", &disk);
 /* Open a source file for writing. */
                pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "w");
 /* Write file data. */
                ff_fwrite(g_file_data, sizeof(g_file_data), 1, pxSourceFile);
 /* Close the file. */
                ff_fclose(pxSourceFile);
                g_state = STATE_FILE_READ;
 break;
     }
 case USB_STATUS_DETACH:
     {
                g_state = STATE_DETACH;
 RM_FREERTOS_PLUS_FAT_DiskDeinit(&g_rm_freertos_plus_fat0_ctrl, &disk);
 break;
     }
 default:
     {
 break;
     }
     }
 if (STATE_FILE_READ == g_state)
     {
```

```
            pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "r");
 for (k = 0; k < RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES; k++)
      {
                g_read_buffer[k] = (uint8_t) 0;
      }
 /* Read file data. */
            size_return = ff_fread(g_read_buffer, sizeof(g_file_data), 1,
pxSourceFile);
 /* Close the file. */
            close_err = ff_fclose(pxSourceFile);
      }
    }
 #endif
}
```

Function Documentation

◆ R_USB_HMSC_StorageCommand()

fsp_err_t R_USB_HMSC_StorageCommand (usb_ctrl_t *const *p_api_ctrl*, uint8_t * *buf*, uint8_t *command*, uint8_t *destination*)

Processing for MassStorage(ATAPI) command.

Return values

FSP_SUCCESS	Success.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter Null pointer error.
FSP_ERR_USB_PARAMETER	Parameter error.

◆ R_USB_HMSC_DriveNumberGet()

fsp_err_t R_USB_HMSC_DriveNumberGet (usb_ctrl_t *const *p_api_ctrl*, uint8_t * *p_drive*, uint8_t *destination*)

Get number of Storage drive.

Return values

FSP_SUCCESS	Success.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter Null pointer error.
FSP_ERR_USB_PARAMETER	Parameter error.

◆ R_USB_HMSC_SemaphoreGet()

fsp_err_t R_USB_HMSC_SemaphoreGet (void)

Get a semaphore. (RTOS only)

If this function is called in the OS less execution environment, a failure is returned.

Return values

FSP_SUCCESS	Success.
FSP_ERR_USB_FAILED	The function could not be completed successfully.

◆ R_USB_HMSC_SemaphoreRelease()

fsp_err_t R_USB_HMSC_SemaphoreRelease (void)

Release a semaphore. (RTOS only)

If this function is called in the OS less execution environment, a failure is returned.

Return values

FSP_SUCCESS	Success.
FSP_ERR_USB_FAILED	The function could not be completed successfully.

◆ R_USB_HMSC_StorageReadSector()

fsp_err_t R_USB_HMSC_StorageReadSector (uint16_t *drive_number*, uint8_t *const *buff*, uint32_t *sector_number*, uint16_t *sector_count*)

Read sector information.

Return values

FSP_SUCCESS	Success.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter Null pointer error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

The address specified in the argument buff must be 4-byte aligned.

◆ R_USB_HMSC_StorageWriteSector()

fsp_err_t R_USB_HMSC_StorageWriteSector (uint16_t *drive_number*, uint8_t const *const *buff*, uint32_t *sector_number*, uint16_t *sector_count*)

Write sector information.

Return values

FSP_SUCCESS	Success.
FSP_ERR_USB_FAILED	The function could not be completed successfully.
FSP_ERR_ASSERTION	Parameter Null pointer error.
FSP_ERR_USB_PARAMETER	Parameter error.

Note

The address specified in the argument buff must be 4-byte aligned.

4.2.67 USB Host Vendor Class (r_usb_hvnd)

Modules

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Overview

USB Host Vendor class works by combining r_usb_basic module.

How to Configuration

The following shows FSP configuration procedure for USB Host Vendor class.

- Select [New Stack]->[Middleware]->[USB]->[USB Host Vendor class driver on r_usb_hvnd].

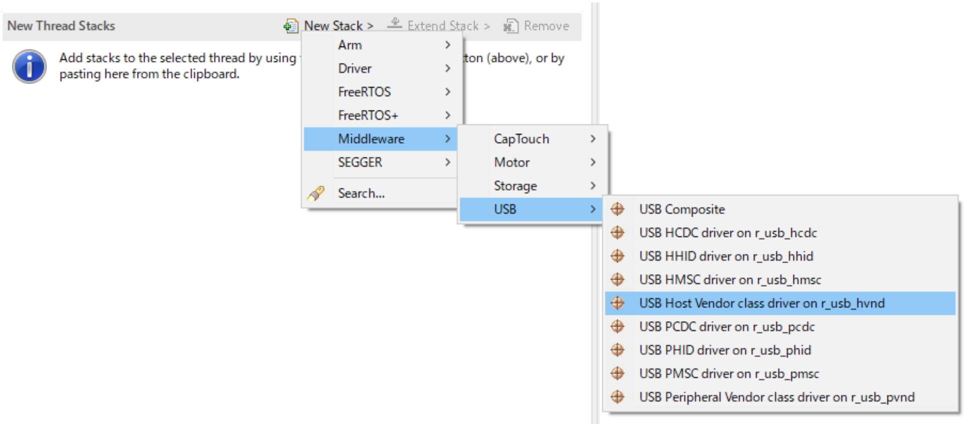


Figure 191: Select USB Host Vendor Class

- The following is displayed when selecting [USB Host Vendor class driver on r_usb_hvnd].
  The user does not specify USB pipe number in Vendor class.

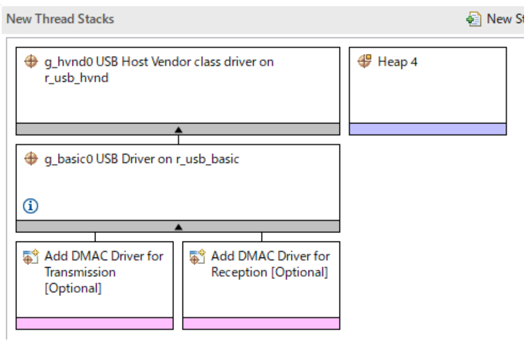


Figure 192: USB Host Vendor Class Stack

API

Use the following APIs in Host Vendor class application program.

- For Data Transfer
  Use the following APIs for data transfer for Bulk transfer or Interrupt transfer.

  1. R_USB_PipeRead()
  2. R_USB_PipeWrite()

3. R_USB_PipeStop()

- For Control Transfer
  Use the following API for the class request processing.

  1. R_USB_HostControlTransfer()

- For USB Pipe Information
  The USB driver allocates USB PIPE by analyzing the descriptor of USB device in Vendor class. Use the following APIs to get the allocated USB pipe information.

  1. R_USB_UsedPipesGet()
  2. R_USB_PipeInfoGet()

USB PIPE Allocation

The USB driver allocates USB PIPE by analyzing the descriptor of USB device in Vendor class.
The USB PIPE related to the Endpoint Descriptor are allocated in order from USB PIPE1 according to the description order of the Endpoint Descriptor.

Examples

This application program processes the follwoing after the enumeration completes with USB device.

1. Getting USB Pipe Infomattion
2. Vendor Class Request Processing
3. Loopback processing of bulk transfer and interrupt transfer.

```
/***********************************************************************************
**
 * Macro definitions
 ***********************************************************************************
**/
/* for Vendor Class Request */
#define USB_SET_VENDOR_NO_DATA (0x0000U)
#define USB_SET_VENDOR (0x0100U)
#define USB_GET_VENDOR (0x0200U)
#define SET_VENDOR_NO_DATA (USB_SET_VENDOR_NO_DATA | USB_HOST_TO_DEV |
USB_VENDOR | USB_INTERFACE)
#define SET_VENDOR (USB_SET_VENDOR | USB_HOST_TO_DEV | USB_VENDOR |
USB_INTERFACE)
#define GET_VENDOR (USB_GET_VENDOR | USB_DEV_TO_HOST | USB_VENDOR |
USB_INTERFACE)
/***********************************************************************************
**
```

```
 * Function Name : usb_main
 * Description : main routine or task for host vendor class application.
 * Arguments : none:
 * Return value : none
 *****************************************************************************
**/
void main_task (void)
{
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif
 usb_status_t    event;
    usb_event_info_t event_info;
    uint8_t          bulk_out_pipe = 0; /* Bulk Out Pipe       */
    uint8_t          bulk_in_pipe = 0; /* Bulk In Pipe       */
    uint8_t          int_out_pipe = 0; /* Interrupt Out Pipe   */
    uint8_t          int_in_pipe   = 0; /* Interrupt In Pipe   */
    uint16_t   buf_type = 0;
    uint8_t    pipe      = 0;
    uint8_t    is_zlp[2] = {0, 0};
    uint16_t   used_pipe = 0;
    usb_pipe_t pipe_info;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
#if (BSP_CFG_RTOS == 2)
       USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **) &p_mess);
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&event_info, &event);
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
      {
```

```
 case USB_STATUS_CONFIGURED:
      {
                buffer_init();
                is_zlp[0] = 0;
                is_zlp[1] = 0;
 /* Get USB Pipe Information */
                g_usb_on_usb.usedPipesGet(&g_basic0_ctrl, &used_pipe,
ADDRESS1);
 for (pipe = START_PIPE; pipe < END_PIPE; pipe++)
      {
 if ((used_pipe & (1 << pipe)) != 0)
      {
                        g_usb_on_usb.pipeInfoGet(&g_basic0_ctrl, &pipe_info,
pipe);
 if (USB_EP_DIR_IN != (pipe_info.endpoint & USB_EP_DIR_IN))
      {
 /* Out Transfer */
 if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
      {
                               buf_type      = BUF_BULK;
                               bulk_out_pipe = pipe;
                            }
 else
      {
                               buf_type    = BUF_INT;
                               int_out_pipe = pipe;
                            }
                        }
 else
      {
 /* In Transfer */
 if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
      {
                               buf_type    = BUF_BULK;
```

```
                              bulk_in_pipe = pipe;
                           }
 else
      {
                              buf_type    = BUF_INT;
                              int_in_pipe = pipe;
                           }
                       }
                  }
               }
 /* Send Vendor Class Request */
               class_request_set_vendor_no_data(&g_basic0_ctrl,
event_info.device_address);
 break;
           }
 case USB_STATUS_READ_COMPLETE:
      {
 if (FSP_ERR_USB_FAILED != event_info.status)
      {
 if (bulk_in_pipe == event_info.pipe)
      {
                        buf_type = BUF_BULK;
                        pipe    = bulk_out_pipe;
                   }
 else if (int_in_pipe == event_info.pipe)
      {
                        buf_type = BUF_INT;
                        pipe    = int_out_pipe;
                   }
 else
      {
 while (1)
      {
                           ;
```

```
                        }
                   }
                   buffer_check(buf_type, event_info.data_size);
                   g_usb_on_usb.pipeWrite(&g_basic0_ctrl,
&g_buf[buf_type][0], event_info.data_size, pipe);
                }
 break;
            }
 case USB_STATUS_WRITE_COMPLETE:
     {
 if (bulk_out_pipe == event_info.pipe)
     {
                   buf_type = BUF_BULK;
 if (1 == is_zlp[buf_type])
     {
                        pipe = bulk_in_pipe;
                   }
                }
 else if (int_out_pipe == event_info.pipe)
     {
                   buf_type = BUF_INT;
 if (1 == is_zlp[buf_type])
     {
                        pipe = int_in_pipe;
                   }
                }
 else
     {
 /* Nothing */
                }
 if (1 == is_zlp[buf_type])
     {
                   is_zlp[buf_type] = 0;
                   buffer_clear(buf_type);
```

```
                   g_usb_on_usb.pipeRead(&g_basic0_ctrl, &g_buf[buf_type][0],
BUF_SIZE, pipe);
                }
 else
      {
                   is_zlp[buf_type] = 1;
                   g_usb_on_usb.pipeWrite(&g_basic0_ctrl, 0, 0,
event_info.pipe); /* Send ZLP */
                }
 break;
            }
 case USB_STATUS_REQUEST_COMPLETE:
      {
 if (USB_SET_VENDOR_NO_DATA == (event_info.setup.request_type & USB_BREQUEST
))
      {
                   class_request_set_vendor(&g_basic0_ctrl,
event_info.device_address);
                }
 else if (USB_SET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
      {
                   class_request_get_vendor(&g_basic0_ctrl,
event_info.device_address);
                }
 else if (USB_GET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
      {
                  buffer_init();
 /* Bulk Out Transfer */
                   g_usb_on_usb.pipeWrite(&g_basic0_ctrl,
&g_buf[BUF_BULK][0], (BUF_SIZE - USB_APL_MXPS),
                                                 bulk_out_pipe);
 /* Interrupt Out Transfer */
                   g_usb_on_usb.pipeWrite(&g_basic0_ctrl, &g_buf[BUF_INT][0],
(BUF_SIZE - USB_APL_MXPS), int_out_pipe);
```

```
                }
 else
     {
 /* Unsupported request */
                }
 break;
            }
 case USB_STATUS_DETACH:
     {
 break;
            }
 default:
     {
 break;
            }
      }
    }
} /* End of function usb_main */
static void class_request_set_vendor (usb_instance_ctrl_t * p_ctrl, uint8_t
device_address)
{
    usb_setup_t setup;
    uint16_t    i;
 for (i = 0; i < REQ_SIZE; i++)
    {
       g_request_buf[i] = (uint8_t) i;
    }
    setup.request_type   = SET_VENDOR; /* bRequestCode:SET_VENDOR,
bmRequestType */
    setup.request_value = 0x0000;    /* wValue:Zero */
    setup.request_index = 0x0000;    /* wIndex:Interface */
    setup.request_length = REQ_SIZE;   /* wLength: Data Length */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, &g_request_buf[0],
```

```
device_address);
}
static void class_request_set_vendor_no_data (usb_instance_ctrl_t * p_ctrl,
uint8_t device_address)
{
    usb_setup_t setup;
    uint16_t    i;
 for (i = 0; i < REQ_SIZE; i++)
    {
       g_request_buf[i] = (uint8_t) i;
    }
    setup.request_type   = SET_VENDOR_NO_DATA; /*
bRequestCode:SET_VENDOR_NO_DATA, bmRequestType */
    setup.request_value = 0x0000;             /* wValue:Zero */
    setup.request_index = 0x0000;             /* wIndex:Interface */
    setup.request_length = 0x0000;             /* wLength: Data Length */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, &g_request_buf[0],
device_address);
}
/******************************************************************************
**
 * Function Name : class_request_get_vendor
 * Description : Send Vendor Class Request (GET_VENDOR) to USB device.
 * Arguments : none
 * Return value : none
 ******************************************************************************
**/
static void class_request_get_vendor (usb_instance_ctrl_t * p_ctrl, uint8_t
device_address)
{
    usb_setup_t setup;
    uint16_t    i;
 for (i = 0; i < REQ_SIZE; i++)
```

```
    {
       g_request_buf[i] = 0;
    }
    setup.request_type   = GET_VENDOR; /* bRequestCode:GET_VENDOR,
bmRequestType */
    setup.request_value = 0x0000;    /* wValue:Zero */
    setup.request_index = 0x0000;    /* wIndex:Interface */
    setup.request_length = REQ_SIZE;   /* wLength: Data Length */
 /* Request Control transfer */
    g_usb_on_usb.hostControlTransfer(p_ctrl, &setup, &g_request_buf[0],
device_address);
}
/*****************************************************************************
**
 * Function Name : buffer_init
 * Description : buffer initialization
 * Arguments : none
 * Return value : none
 *****************************************************************************
**/
static void buffer_init (void)
{
    uint16_t i;
    uint16_t j;
 for (j = 0; j < 2; j++)
    {
 for (i = 0; i < BUF_SIZE; i++)
      {
            g_buf[j][i] = (uint8_t) i;
       }
    }
}
/*****************************************************************************
**
```

```
 * Function Name : buffer_check
 * Description : buffer check
 * Arguments : buf_type : buffer number
 * Return value : none
 *****************************************************************************
**/
static void buffer_check (uint16_t buf_type, uint32_t size)
{
    uint16_t i;
 for (i = 0; i < (uint16_t) size; i++)
    {
 if ((uint8_t) (i & USB_VALUE_FF) != g_buf[buf_type][i])
      {
 while (1)
      {
                ;
           }
       }
    }
}
/*****************************************************************************
**
 * Function Name : buffer_clear
 * Description : buffer clear
 * Arguments : buf_type : buffer number
 * Return value : none
 *****************************************************************************
**/
static void buffer_clear (uint16_t buf_type)
{
    uint16_t i;
 for (i = 0; i < BUF_SIZE; i++)
    {
       g_buf[buf_type][i] = 0;
```

```
    }
}
/*****************************************************************************
**
 * End of function usb_mcu_init
 ****************************************************************************
**/
#if (BSP_CFG_RTOS == 2)
/*****************************************************************************
**
 * Function Name : usb_apl_rec_msg
 * Description : Receive a message to the specified id (mailbox).
 * Argument : uint8_t id : ID number (mailbox).
 * : usb_msg_t** mess : Message pointer
 * : usb_tm_t tm : Timeout Value
 * Return : uint16_t : USB_OK / USB_ERROR
 ****************************************************************************
**/
usb_er_t usb_apl_rec_msg (uint8_t id, usb_msg_t ** mess, usb_tm_t tm)
{
    BaseType_t    err;
    QueueHandle_t handle;
    usb_er_t      result;
    (void) tm;
 if (NULL == mess)
    {
 return USB_APL_ERROR;
    }
    handle = (*(g_apl_mbx_table[id]));
    *mess = NULL;
    err = xQueueReceive(handle, (void *) mess, (portMAX_DELAY));
 if ((pdTRUE == err) && (NULL != (*mess)))
    {
       result = USB_APL_OK;
```

```
    }
 else
    {
       result = USB_APL_ERROR;
    }
 return result;
}
/*****************************************************************************
**
 * End of function usb_apl_rec_msg
 *****************************************************************************
**/
/*****************************************************************************
**
 * Function Name : usb_apl_snd_msg
 * Description : Send a message to the specified id (mailbox).
 * Argument : uint8_t id : ID number (mailbox).
 * : usb_msg_t* mess : Message pointer
 * Return : usb_er_t : USB_OK / USB_ERROR
 *****************************************************************************
**/
usb_er_t usb_apl_snd_msg (uint8_t id, usb_msg_t * mess)
{
    BaseType_t    err;
    QueueHandle_t handle;
    usb_er_t      result;
 if (NULL == mess)
    {
 return USB_APL_ERROR;
    }
    handle = (*(g_apl_mbx_table[id]));
    err = xQueueSend(handle, (const void *) &mess, (TickType_t) (0));
 if (pdTRUE == err)
    {
```

```
      result = USB_APL_OK;
   }
 else
   {
      result = USB_APL_ERROR;
   }
 return result;
}
/*****************************************************************************
**
 * End of function usb_apl_snd_msg
 ****************************************************************************
**/
#endif  /* #if (BSP_CFG_RTOS == 2) */
```

4.2.68 USB Peripheral Communications Device Class (r_usb_pcdc)

Modules

This module provides a USB Peripheral Communications Device Class Driver (PCDC). It implements the USB PCDC Interface.

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Detailed Description

Overview

The r_usb_pcdc module combines with the r_usb_basic module to provide a USB Peripheral Communications Device Class (PCDC) driver. The PCDC driver conforms to Abstract Control Model of the USB Communications Device Class (CDC) specification and enables communication with a CDC host device.

Features

The r_usb_pcdc module has the following key features:

- Data transfer to and from a USB host
- Response to CDC class requests
- Supports CDC notifications

Configuration

Build Time Configurations for r_usb_pcdc

The following build time configurations are defined in fsp_cfg/r_usb_pcdc_cfg.h:

Configuration	Options	Default	Description
Bulk In Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE1	Select the USB pipe to use for bulk input transfers.
Bulk Out Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE2	Select the USB pipe to use for bulk output transfers.
Interrupt Out Pipe	• USB PIPE6 • USB PIPE7 • USB PIPE8 • USB PIPE9	USB PIPE6	Select the USB pipe to use for interrupts.

Configurations for Connectivity > USB PCDC (r_usb_pcdc)

This module can be added to the Stacks tab via New Stack > Connectivity > USB PCDC (r_usb_pcdc).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_pcdc0	Module name.

Note

Refer to the USB (r_usb_basic) module for hardware configuration options.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Abstract Control Model Overview

The Abstract Control Model subclass of CDC is a technology that bridges the gap between USB devices and earlier modems (employing RS-232C connections), enabling use of application programs designed for older modems.

Class Requests (Host to Peripheral)

This driver notifies the application when receiving the following class requests:

Request	Code	Description
SetLineCoding	0x20	Sets communication line settings (bitrate, data length, parity, and stop bit length)
GetLineCoding	0x21	Acquires the communication line setting state
SetControlLineState	0x22	Set communication line control signals (RTS, DTR)

Note

For details concerning the Abstract Control Model requests, refer to Table 11 "Requests - Abstract Control Model" in the "USB Communications Class Subclass Specification for PSTN Devices", Revision 1.2.

Data Format of Class Requests

The data format of supported class requests is described below:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0x21	SET_LINE_CODING (0x20)	0x0000	0x0000	0x0007	usb_pcdc_linecoding_t
0xA1	GET_LINE_CODING (0x21)	0x0000	0x0000	0x0007	usb_pcdc_linecoding_t
0x21	SET_CONTROL_LINE_STATE (0x22)	usb_pcdc_ctrllinestate_t	0x0000	0x0000	None

Class Notifications (Peripheral to Host)

The following class notifications are supported:

Notification	Code	Description
SERIAL_STATE	0x20	Notification of serial line state

The data types returned are as follows:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	SERIAL_STATE (0x20)	0x0000	0x0000	0x0002	usb_serial_state_bitmap_t

Note

The host is notified with SERIAL_STATE whenever a change in the UART port state is detected. This driver will automatically detect overrun, parity and framing errors. A state notification is performed when a transition from normal to error state is detected.

Virtual COM-port Usage

When connected to a PC the CDC device can be used as a virtual COM port. After enumeration, the CDC class requests GetLineCoding and SetControlLineState are executed by the target, and the CDC device is registered in Windows Device Manager as a virtual COM device.

Registering the CDC device as a virtual COM-port in Windows Device Manager enables data communication with the CDC device via a terminal app such as PuTTY. When changing settings of the serial port in the terminal application, the UART setting is propagated to the firmware via the class request SetLineCoding.

Data input (or file transmission) from the terminal app window is transmitted to the board using endpoint 2 (EP2); data from the board side is transmitted to the PC using EP1.

When the last packet of data received is the maximum packet size, and the terminal determines that there is continuous data, the received data may not be displayed in the terminal. If the received data is smaller than the maximum packet size, the data received up to that point is displayed in the terminal.

Multi Port

This driver supports simultaneous operation with Host Mass Storage Class(HMSC). If the user are using MCU that supports 2 USB modules, such as RA6M3, the user can run PCDC on one USB module and HMSC on the other. This driver does not support simultaneous operation using device classes other than HMSC.

Limitations

- This module must be incorporated into a project using r_usb_basic and does not provide any public APIs.
- This driver does not support Low-speed.

Examples

USB PCDC Loopback Example

The main functions of the PCDC loopback example are as follows:

1. Receives virtual UART configuration data from the host terminal
2. Loops all other received data back to the host terminal

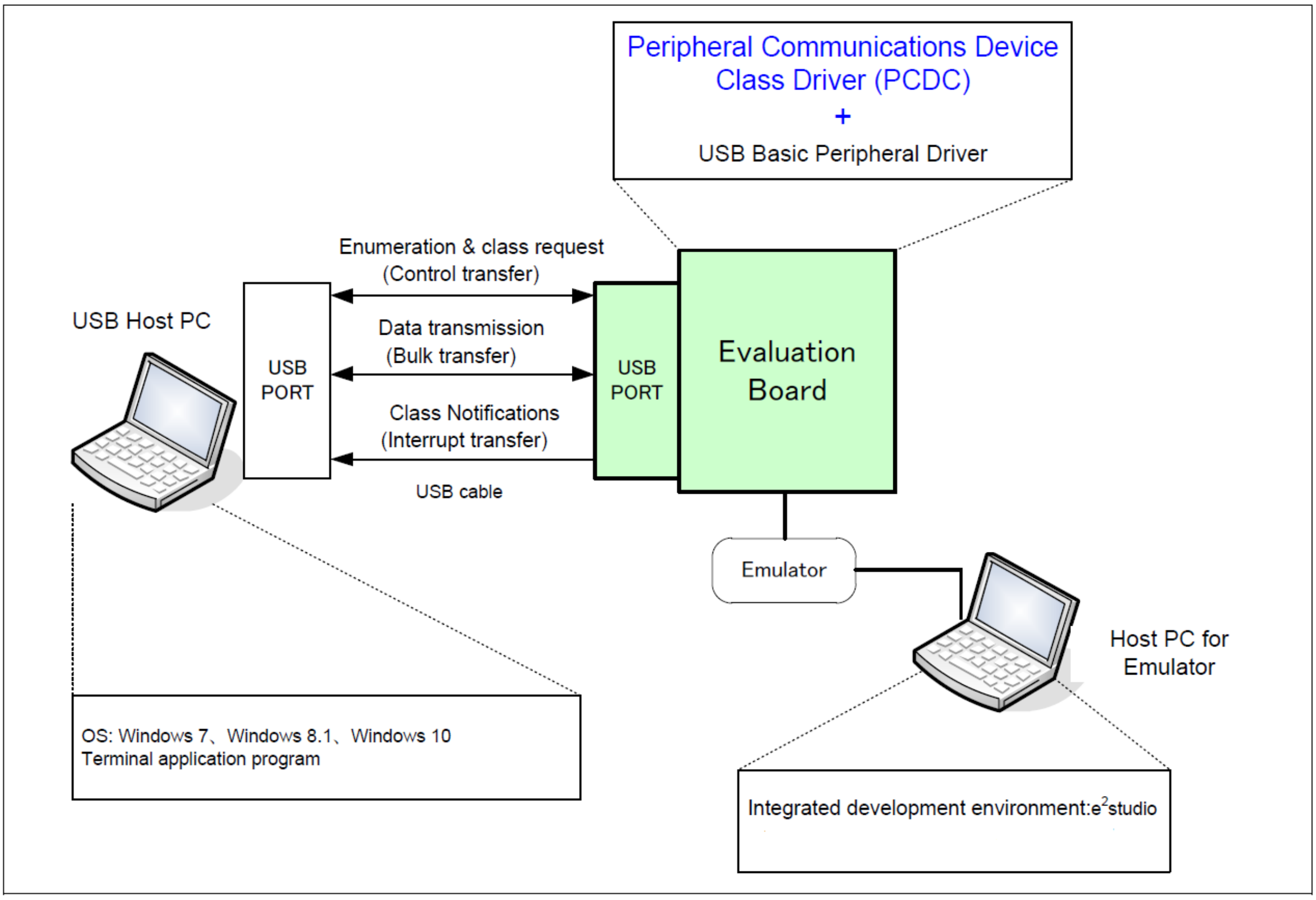


Figure 193: Example Operating Environment

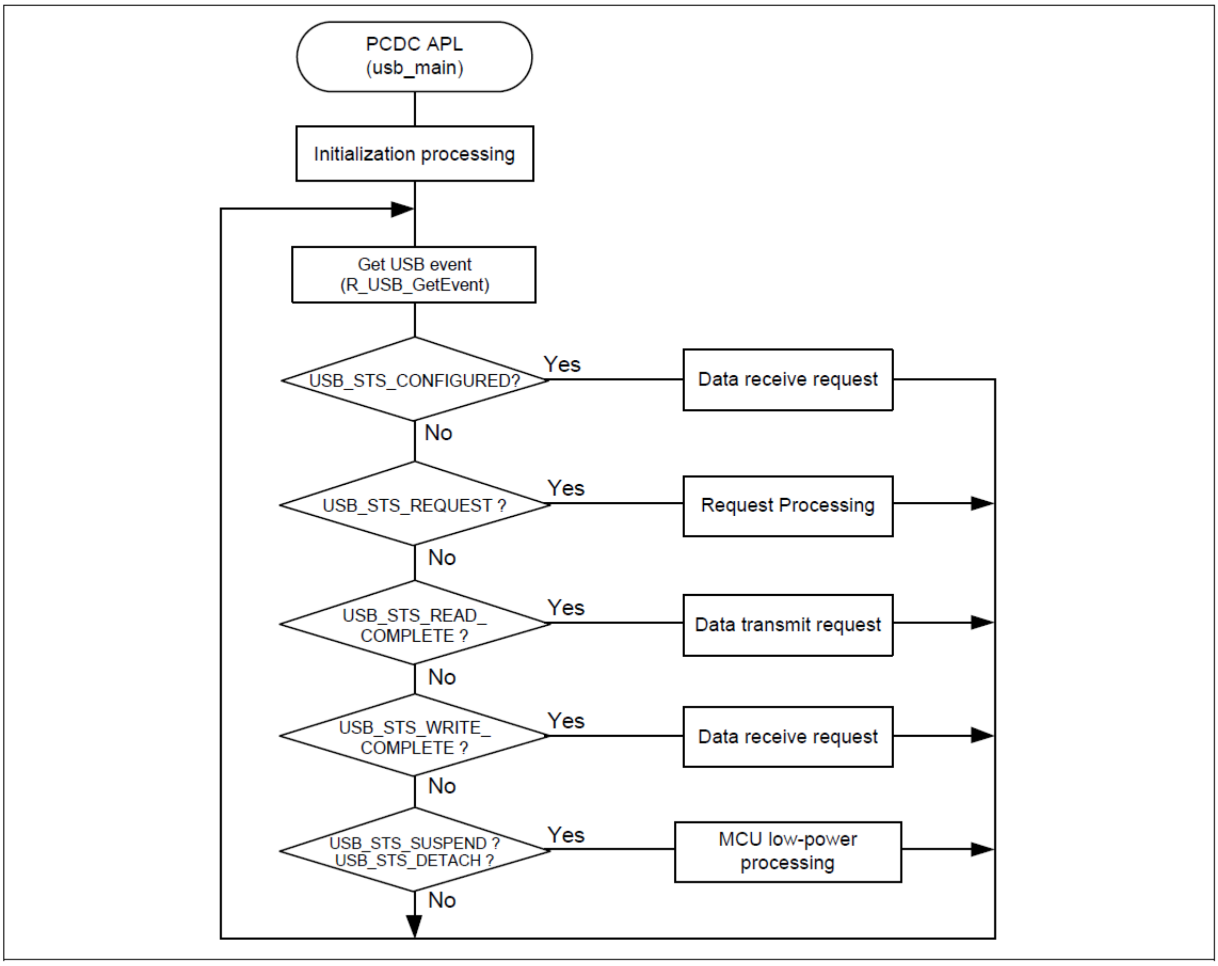


Figure 194: Main Loop processing (Echo mode)

```
void usb_basic_example (void)
{
    usb_event_info_t event_info;
 usb_status_t    event;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
 /* Get USB event data */
       g_usb_on_usb.eventGet(&event_info, &event);
 /* Handle the received event (if any) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
```

```
 case USB_STATUS_WRITE_COMPLETE:
 /* Initialization complete; get data from host */
                g_usb_on_usb.read(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PCDC);
 break;
 case USB_STATUS_READ_COMPLETE:
 /* Loop back received data to host */
                g_usb_on_usb.write(&g_basic0_ctrl, g_buf, event_info.data_size,
USB_CLASS_PCDC);
 break;
 case USB_STATUS_REQUEST:   /* Receive Class Request */
 if (USB_PCDC_SET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Configure virtual UART settings */
                   g_usb_on_usb.periControlDataGet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
                }
 else if (USB_PCDC_GET_LINE_CODING == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Send virtual UART settings back to host */
                   g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, (uint8_t *)
&g_line_coding, LINE_CODING_LENGTH);
                }
 else
      {
 /* ACK all other status requests */
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
                }
 break;
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
 break;
 default:
 break;
```

```
        }
    }
}
```

Descriptor

A template for PCDC descriptors can be found in ra/fsp/src/r_usb_pcdc/r_usb_pcdc_descriptor.c.template. Also, please be sure to use your vendor ID.

4.2.69 USB Peripheral Human Interface Device Class (r_usb_phid)

Modules

This module is USB Peripheral Human Interface Device Class Driver (PHID). It implements the USB PHID Interface.

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Detailed Description

Overview

The r_usb_phid module combines with the r_usb_basic module to provide a USB Peripheral Human Interface Device Class (PHID) driver. The PHID driver conforms to the USB Human Interface Device class specifications and implements communication with a HID host.

Features

The r_usb_phid module has the following functions:

- Data transfer to and from a USB host
- Response to HID class requests
- Response to function references from the HID host

Note

This driver is not guaranteed to provide USB HID operation in all scenarios. The developer must verify correct operation when connected to the targeted USB hosts.

Configuration

Build Time Configurations for r_usb_phid

The following build time configurations are defined in fsp_cfg/r_usb_phid_cfg.h:

Configuration	Options	Default	Description
Interrupt In Pipe	• USB PIPE6 • USB PIPE7 • USB PIPE8 • USB PIPE9	USB PIPE6	Select the pipe number for input interrupt events.
Interrupt Out Pipe	• USB PIPE6 • USB PIPE7 • USB PIPE8 • USB PIPE9	USB PIPE7	Select the pipe number for output interrupt events.

Configurations for Connectivity > USB PHID (r_usb_phid)

This module can be added to the Stacks tab via New Stack > Connectivity > USB PHID (r_usb_phid).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_phid0	Module name.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Class Requests (Host to Peripheral)

This driver notifies the application when receiving the following class requests:

Request	Code	Description
Get_Report	0x01	Receives a report from the HID host
Set_Report	0x09	Sends a report to the HID host
Get_Idle	0x02	Receives a duration (time) from the HID host
Set_Idle	0x0A	Sends a duration (time) to the HID host
Get_Protocol	0x03	Reads a protocol from the HID host
Set_Protocol	0x0B	Sends a protocol to the HID host
Get_Descriptor	0x06	Transmits a report or HID

descriptor

The data format of supported class requests is described below:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0xA1	GET_REPORT (0x01)	ReportType & ReportID	Interface	ReportLength	Report
0x21	SET_REPORT (0x09)	ReportType & ReportID	Interface	ReportLength	Report
0xA1	GET_IDLE (0x02)	0 & ReportID	Interface	1	Idle rate
0x21	SET_IDLE (0x0A)	Duration & ReportID	Interface	0	Idle rate
0xA1	GET_PROTOCOL (0x03)	0	Interface	0	0 (Boot) or 1 (Report)
0x21	SET_PROTOCOL (0x0B)	0 (Boot) or 1 (Report)	Interface	0	Not applicable

Limitations

- This driver does not support USB Hi-speed mode.
- This driver does not support USB Low-speed mode.
- This driver does not support DMA transfers.
- This driver does not support simultaneous operation with USB Host device class.

Examples

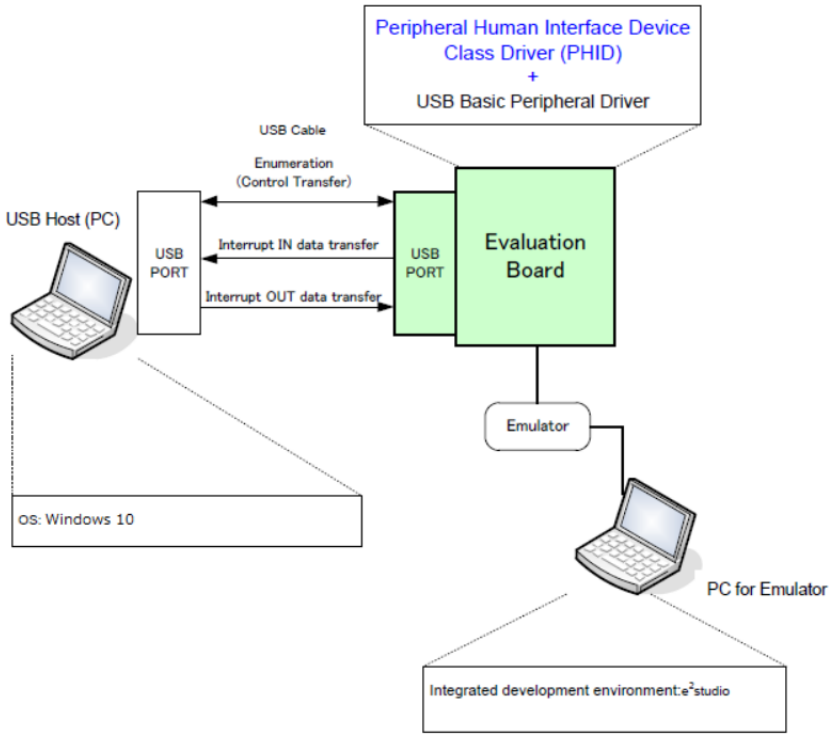


Figure 195: Example Operating Environment

USB PHID Example (no RTOS)

This is a minimal example for implementing PHID in a non-RTOS application.

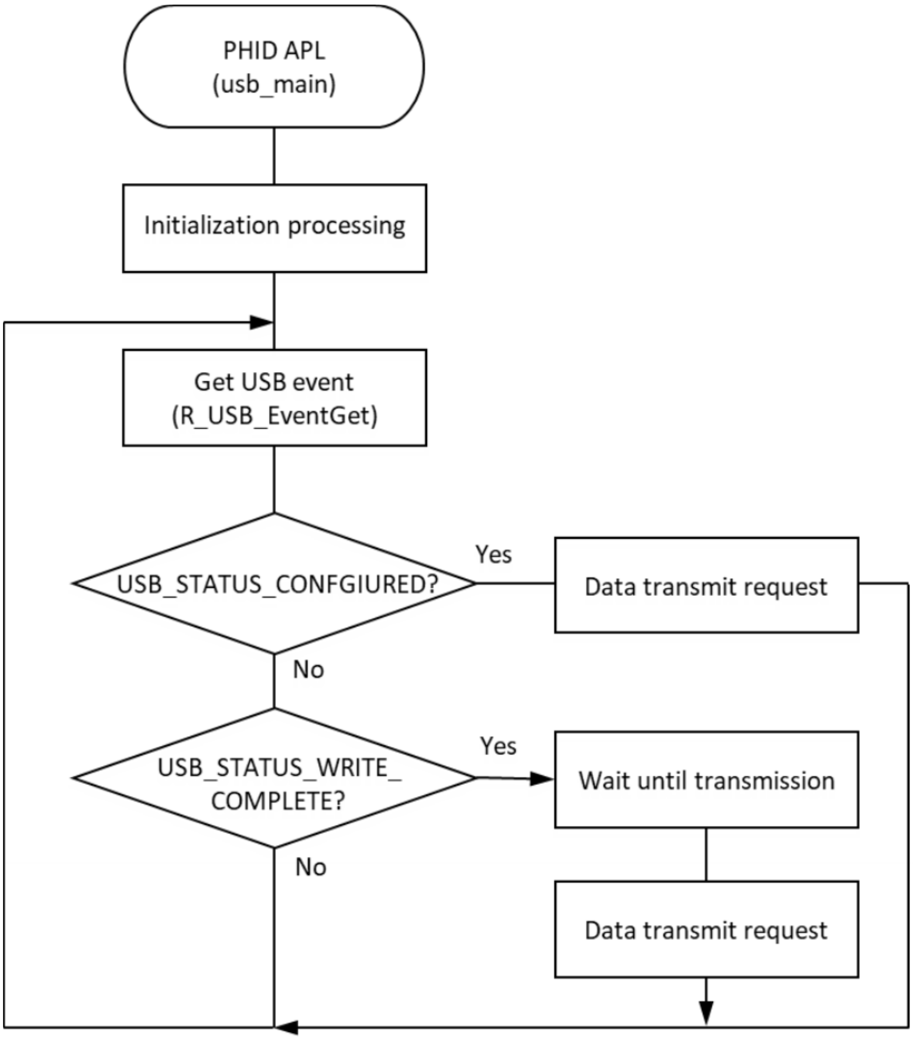


Figure 196: Main Loop processing for non-RTOS example

```
#define USB_RECEIVE_REPORT_DESCRIPTOR (76)
#define USB_RECEIVE_HID_DESCRIPTOR (9)
#define USB_WAIT_1000MS (1000)
#define SW_ACTIVE 0
#define SW R_PFS->PORT[0].PIN[8].PmnPFS_b.PIDR
#define SW_PDR R_PFS->PORT[0].PIN[8].PmnPFS_b.PDR
#define SW_PMR R_PFS->PORT[0].PIN[8].PmnPFS_b.PMR
static uint8_t g_buf[] = {0, 0, 0, 0, 0, 0, 0, 0}; /* HID data */
static const uint8_t g_zero_data[] = {0, 0, 0, 0, 0, 0, 0, 0}; /* zero data */
static uint16_t g_numlock = 0;
static uint8_t g_idle = 0;
uint8_t             g_remote_wakeup_enable = USB_OFF;
uint8_t             g_status               = NO_WRITING;
```

```
/*****************************************************************************
 * Function Name : usb_cpu_getkeyno
 * Description : input key port
 * Arguments : none
 * Return value : uint16_t : key_no
 ******************************************************************************/
uint8_t usb_cpu_getkeyno (void)
{
    uint8_t key_buf = 0;
 if (SW_ACTIVE == SW)
    {
 if (sw_on_count[0] < SW_ON_THRESHOLD)
      {
            sw_on_count[0]++;
       }
    }
 else
    {
 if (sw_on_count[0] >= SW_ON_THRESHOLD)
      {
            key_buf |= SW_PUSH;
       }
       sw_on_count[0] = 0;
    }
 return key_buf;
}
void set_key_data (uint8_t * p_buf)
{
 static uint8_t key_data;
    key_data = KBD_CODE_A;
    *(p_buf + 2) = key_data;
}
void usb_basic_example (void)
{
```

```
    usb_event_info_t event_info;
 usb_status_t    event;
    uint8_t       * p_idle_value;
    uint8_t          sw_data;
    usb_info_t       info;
 fsp_err_t       ret_code = FSP_SUCCESS;
    uint8_t          send_data[16] BSP_ALIGN_VARIABLE(4);
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    set_key_data(g_buf);
 while (1)
    {
       g_usb_on_usb.eventGet(&event_info, &event);
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
 break;
 case USB_STATUS_WRITE_COMPLETE:
 if (DATA_WRITING == g_status)
      {
                   g_status = ZERO_WRITING;
                   g_usb_on_usb.write(&g_basic0_ctrl, (uint8_t *) g_zero_data,
DATA_LEN, USB_CLASS_PHID); /* Sending the zero data (8 bytes) */
         }
 else
      {
                   g_status = DATA_WRITING;
                   usb_cpu_delay_xms(USB_WAIT_1000MS);
                   g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
         }
 break;
 case USB_STATUS_REQUEST:
/* Receive Class Request */
 if (USB_SET_REPORT == (event_info.setup.request_type & USB_BREQUEST))
```

```
     {
                   g_usb_on_usb.read(&g_basic0_ctrl, (uint8_t *) &g_numlock, 2,
USB_CLASS_PHID); /* Get the NumLock data (NumLock data is not used) */
        }
 else if (USB_GET_DESCRIPTOR == (event_info.setup.request_type & USB_BREQUEST))
     {
 if (USB_GET_REPORT_DESCRIPTOR == event_info.setup.request_value)
     {
                        g_usb_on_usb.periControlDataSet(&g_basic0_ctrl,
                                                               (uint8_t *) g_apl_report,

USB_RECEIVE_REPORT_DESCRIPTOR);
        }
 else if (USB_GET_HID_DESCRIPTOR == event_info.setup.request_value)
     {
 for (uint8_t i = 0; i < USB_RECEIVE_HID_DESCRIPTOR; i++)
     {
                            send_data[i] = g_apl_configuration[18 + i];
        }
 /* Configuration Descriptor address set. */
                        g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, send_data,
USB_RECEIVE_HID_DESCRIPTOR);
        }
 else
     {
                        g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
        }
        }
 else if (USB_SET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
     {
 /* Get SetIdle value */
                   p_idle_value = (uint8_t *) &event_info.setup.request_value;
                   g_idle = p_idle_value[1];
```

```
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
         }
 else if (USB_GET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, &g_idle, 1);
         }
 else if (USB_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
                   g_status = DATA_WRITING;
                   g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
         }
 else if (USB_GET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
         }
 else
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
         }
 break;
 case USB_STATUS_REQUEST_COMPLETE: /* Complete Class Request */
 if (USB_SET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
      {
                   p_idle_value = (uint8_t *) &event_info.setup.request_value;
                   g_idle = p_idle_value[1];
         }
 else if (USB_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
```

```
/* None */
        }
else
     {
                 g_status = DATA_WRITING;
                 g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
        }
break;
case USB_STATUS_SUSPEND:
break;
case USB_STATUS_DETACH:
              g_remote_wakeup_enable = USB_OFF;
break;
default:
break;
      }
      ret_code = g_usb_on_usb.infoGet(&g_basic0_ctrl, &info, NULL);
if (FSP_SUCCESS == ret_code)
     {
           sw_data = usb_cpu_getkeyno();
if (USB_STATUS_SUSPEND == info.device_status)
     {
if (0 != (sw_data & SW_PUSH))
     {
                 g_usb_on_usb.remoteWakeup(&g_basic0_ctrl);
        }
        }
      }
   }
} /* End of function usb_basic_example() */
```

USB PHID Example (RTOS)

This is a minimal example for implementing PHID in an RTOS application.

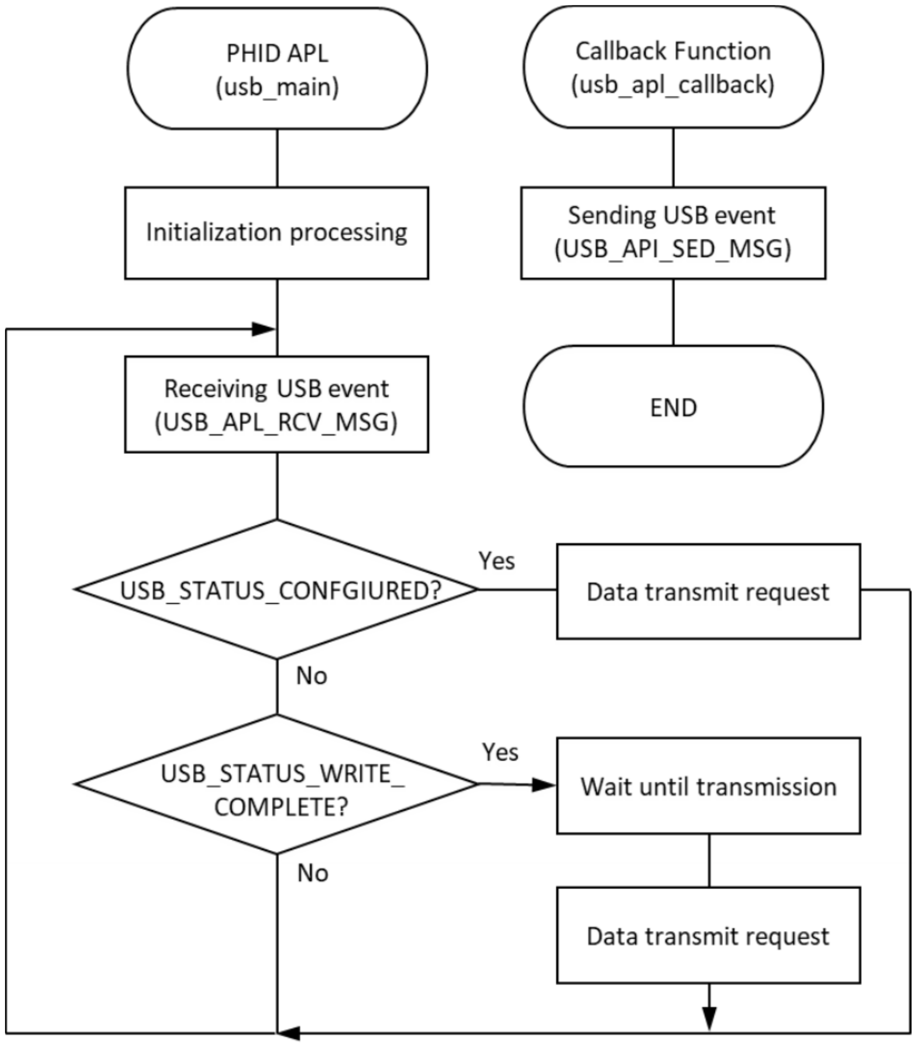


Figure 197: Main Loop processing for RTOS example

```
 #define USB_APL_MBX (0)
void usb_apl_callback (usb_event_info_t * p_api_event, usb_hdl_t cur_task,
usb_onoff_t usb_state)
{
    (void) usb_state;
    (void) cur_task;
    USB_APL_SND_MSG(USB_APL_MBX, (usb_msg_t *) p_api_event);
} /* End of function usb_apl_callback */
/*******************************************************************************
 * Function Name : usb_apl_rec_msg
 * Description : Receive a message to the specified id (mailbox).
 * Argument : uint8_t id : ID number (mailbox).
 * : usb_msg_t** mess : Message pointer
```

```
 * : usb_tm_t tm : Timeout Value
 * Return : uint16_t : USB_OK / USB_ERROR
 ******************************************************************************/
usb_er_t usb_apl_rec_msg (uint8_t id, usb_msg_t ** mess, usb_tm_t tm)
{
    BaseType_t    err;
    QueueHandle_t handle;
    usb_er_t      result;
 if (NULL == mess)
    {
 return USB_APL_ERROR;
    }
    handle = (*(g_apl_mbx_table[id]));
    *mess = NULL;
    err = xQueueReceive(handle, (void *) mess, (tm));
 if ((pdTRUE == err) && (NULL != (*mess)))
    {
       result = USB_APL_OK;
    }
 else
    {
       result = USB_APL_ERROR;
    }
 return result;
}
/******************************************************************************
 * Function Name : usb_apl_snd_msg
 * Description : Send a message to the specified id (mailbox).
 * Argument : uint8_t id : ID number (mailbox).
 * : usb_msg_t* mess : Message pointer
 * Return : usb_er_t : USB_OK / USB_ERROR
 ******************************************************************************/
usb_er_t usb_apl_snd_msg (uint8_t id, usb_msg_t * mess)
{
```

```
    BaseType_t    err;
    QueueHandle_t handle;
    usb_er_t      result;
 if (NULL == mess)
    {
 return USB_APL_ERROR;
    }
    handle = (*(g_apl_mbx_table[id]));
    err = xQueueSend(handle, (const void *) &mess, (TickType_t) (0));
 if (pdTRUE == err)
    {
       result = USB_APL_OK;
    }
 else
    {
       result = USB_APL_ERROR;
    }
 return result;
}
/* RTOS-enabled HID example */
void usb_basic_example_rtos (void)
{
    usb_event_info_t * p_mess;
    usb_event_info_t   event_info;
    uint8_t          * p_idle_value;
    uint8_t            sw_data;
    usb_info_t         info;
 fsp_err_t           ret_code = FSP_SUCCESS;
    uint8_t            send_data[16] BSP_ALIGN_VARIABLE(4);
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    set_key_data(g_buf);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
```

```
      USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **) &p_mess);
      event_info = *p_mess;
 switch (event_info.event)
      {
 case USB_STATUS_CONFIGURED:
 break;
 case USB_STATUS_WRITE_COMPLETE:
 if (DATA_WRITING == g_status)
      {
                  g_status = ZERO_WRITING;
                  g_usb_on_usb.write(&g_basic0_ctrl, (uint8_t *) g_zero_data,
DATA_LEN, USB_CLASS_PHID); /* Sending the zero data (8 bytes) */
         }
 else
      {
                  g_status = DATA_WRITING;
                  usb_cpu_delay_xms(USB_WAIT_1000MS);
                  g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
         }
 break;
 case USB_STATUS_REQUEST:
/* Receive Class Request */
 if (USB_SET_REPORT == (event_info.setup.request_type & USB_BREQUEST))
      {
                  g_usb_on_usb.read(&g_basic0_ctrl, (uint8_t *) &g_numlock, 2,
USB_CLASS_PHID); /* Get the NumLock data (NumLock data is not used) */
         }
 else if (USB_GET_DESCRIPTOR == (event_info.setup.request_type & USB_BREQUEST))
      {
 if (USB_GET_REPORT_DESCRIPTOR == event_info.setup.request_value)
      {
                       g_usb_on_usb.periControlDataSet(&g_basic0_ctrl,
                                                               (uint8_t *) g_apl_report,
```

```
USB_RECEIVE_REPORT_DESCRIPTOR);
         }
 else if (USB_GET_HID_DESCRIPTOR == event_info.setup.request_value)
      {
 for (uint8_t i = 0; i < USB_RECEIVE_HID_DESCRIPTOR; i++)
      {
                            send_data[i] = g_apl_configuration[18 + i];
         }
 /* Configuration Descriptor address set. */
                        g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, send_data,
USB_RECEIVE_HID_DESCRIPTOR);
         }
 else
      {
                        g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
         }
         }
 else if (USB_SET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* Get SetIdle value */
                   p_idle_value = (uint8_t *) &event_info.setup.request_value;
                   g_idle = p_idle_value[1];
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
         }
 else if (USB_GET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlDataSet(&g_basic0_ctrl, &g_idle, 1);
         }
 else if (USB_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
```

```
USB_SETUP_STATUS_ACK);
                   g_status = DATA_WRITING;
                   g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
         }
 else if (USB_GET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
         }
 else
      {
                   g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_STALL);
         }
 break;
 case USB_STATUS_REQUEST_COMPLETE: /* Complete Class Request */
 if (USB_SET_IDLE == (event_info.setup.request_type & USB_BREQUEST))
      {
                   p_idle_value = (uint8_t *) &event_info.setup.request_value;
                   g_idle = p_idle_value[1];
         }
 else if (USB_SET_PROTOCOL == (event_info.setup.request_type & USB_BREQUEST))
      {
 /* None */
         }
 else
      {
                   g_status = DATA_WRITING;
                   g_usb_on_usb.write(&g_basic0_ctrl, g_buf, DATA_LEN, USB_CLASS_PHID
);
         }
 break;
 case USB_STATUS_SUSPEND:
```

```
break;
case USB_STATUS_DETACH:
               g_remote_wakeup_enable = USB_OFF;
break;
default:
break;
      }
      ret_code = g_usb_on_usb.infoGet(&g_basic0_ctrl, &info, NULL);
if (FSP_SUCCESS == ret_code)
     {
          sw_data = usb_cpu_getkeyno();
if (USB_STATUS_SUSPEND == info.device_status)
     {
if (0 != (sw_data & SW_PUSH))
     {
                  g_usb_on_usb.remoteWakeup(&g_basic0_ctrl);
        }
        }
      }
   }
} /* End of function usb_basic_example_rtos() */
```

Descriptors

A template for PHID descriptors can be found in ra/fsp/src/r_usb_phid. Be sure to replace the vendor ID with your own.
Keyboard templates should be referred to r_usb_phid_descriptor_keyboard.c.template.
Mouse templates should be referred to r_usb_phid_descriptor_mouse.c.template.

4.2.70 USB Peripheral Mass Storage Class (r_usb_pmsc)

Modules

This module provides a USB Peripheral Mass Storage Class (PMSC) driver. It implements the USB PMSC Interface.

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Detailed Description

Overview

The r_usb_pmsc module combines with the r_usb_basic module to provide USB Peripheral It operates as a Mass Storage class driver (hereinafter referred to as PMSC).
The USB peripheral mass storage class driver (PMSC) comprises a USB mass storage class bulk-only transport (BOT) protocol.
When combined with a USB peripheral control driver and media driver, it enables communication with a USB host as a BOT-compatible storage device.

Features

The r_usb_pmsc module has the following key features:

- Storage command control using the BOT protocol
- Supports SFF-8070i (ATAPI)
- Response to mass storage device class requests from a USB host

Configuration

Build Time Configurations for r_usb_pmsc

The following build time configurations are defined in fsp_cfg/r_usb_pmsc_cfg.h:

Configuration	Options	Default	Description
Bulk Input Transfer Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE1	Select the USB pipe to use for bulk input transfers.
Bulk Output Transfer Pipe	• USB PIPE1 • USB PIPE2 • USB PIPE3 • USB PIPE4 • USB PIPE5	USB PIPE2	Select the USB pipe to use for bulk output transfers.
Vendor Information	Vendor Information must be 8 bytes long; pad with spaces if shorter.	Vendor	Specify the vendor information field (part of the Inquiry command response).
Product Information	Product Information must be 16 bytes long; pad with spaces if shorter.	Mass Storage	Specify the product information field (part of the Inquiry command response).
Product Revision Level	Product Revision Level	1.00	Specify the product

	must be 4 bytes long; pad with spaces if shorter.		revision level field (part of the Inquiry command response).
Sector size	• 512 • 4096	512	Specifies the sector size.
Number of Transfer Sectors	Please enter a number between 1 and 255.	8	Specify the maximum sector size to request with one data transfer.

Configurations for Connectivity > USB PMSC (r_usb_pmsc)

This module can be added to the Stacks tab via New Stack > Connectivity > USB PMSC (r_usb_pmsc).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_pmsc0	Module name.

Refer to the USB (r_usb_basic) module for hardware configuration options.

Clock Configuration

Refer to the USB (r_usb_basic) module.

Pin Configuration

Refer to the USB (r_usb_basic) module.

Usage Notes

Class Requests

The class requests supported by this driver are shown below.

Request	Code	Description
Bulk-Only Mass Storage Reset	0xFF	Resets the connection interface to the mass storage device.
Get Max Logical Unit Number	0xFE	Reports the logical numbers supported by the device.

Storage Commands

This driver supports the following storage commands.

Command	Code	Description
TEST_UNIT_READY	0x00	Checks the state of the peripheral device.
REQUEST_SENSE	0x03	Gets the error information of the previous storage command

		execution result.
INQUIRY	0x12	Gets the parameter information of the logical unit.
READ_FORMAT_CAPACITY	0x23	Gets the formattable capacity.
READ_CAPACITY	0x25	Gets the capacity information of the logical unit.
READ10	0x28	Reads data.
WRITE10	0x1A	Writes data.
MODE_SENSE10	0x5A	Gets the parameters of the logical unit.

Note

A STALL or FAIL error is sent to the host upon receipt of any command not listed in the above table.

BOT Protocol Overview

BOT (USB MSC Bulk-Only Transport) is a transfer protocol that encapsulates command, data, and status (results of commands) using only two endpoints (one bulk in and one bulk out). The ATAPI storage commands and the response status are embedded in a Command Block Wrapper (CBW) and a Command Status Wrapper (CSW). The below image shows an overview of how the BOT protocol progresses with command and status data flowing between USB host and peripheral.

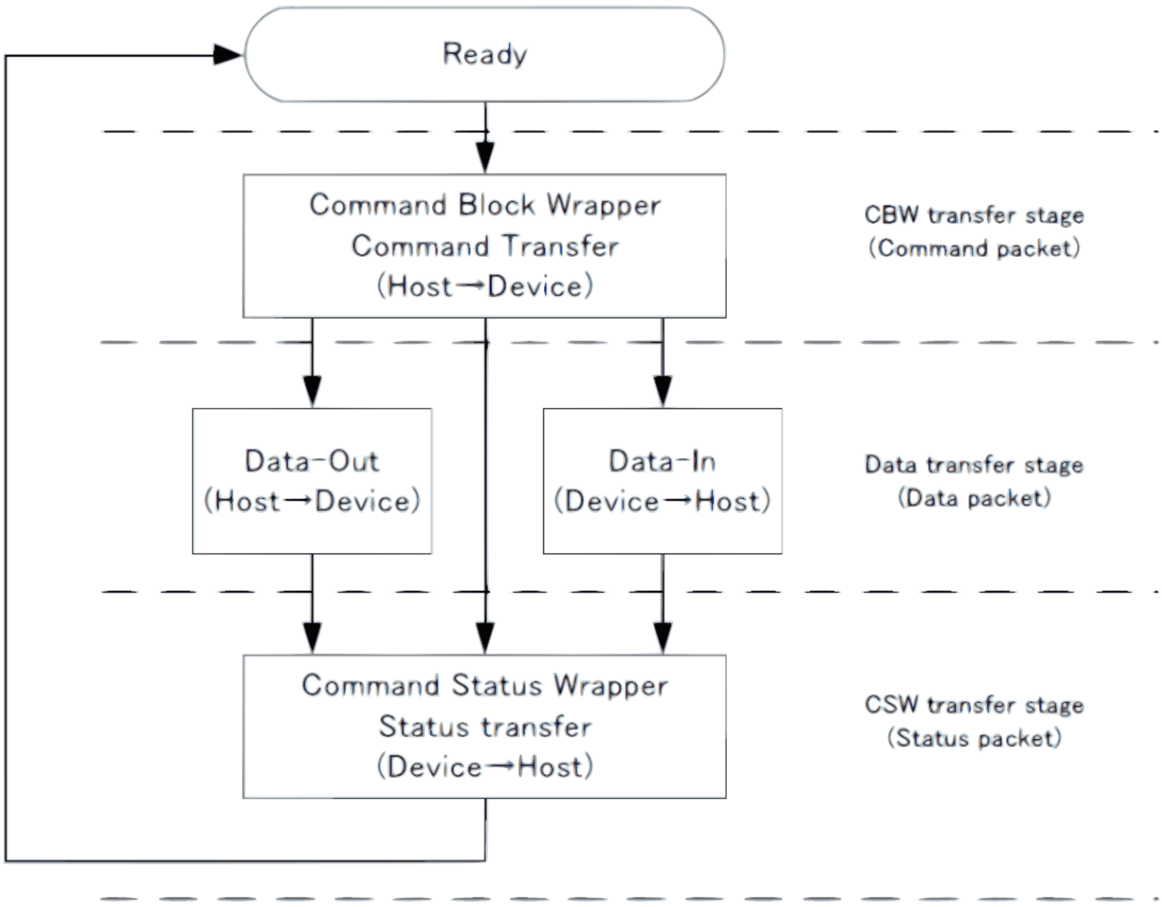


Figure 198: BOT protocol Overview

Block Media Interface

PMSC implements a block media interface to enable access to higher-level modules. If the block media interface supports multiple media, users can select any media to access.

Note

When the user develops the storage media driver, be sure to define the instance named "g_rm_block_media0".

Limitations

1. The driver always returns 0 in response to the GetMaxLun command.
2. The driver supports a sector size of 512 bytes only.
3. The only media currently supported by the block media interface is an SD card. The card must be inserted before initializing the driver.
4. When using DMA for Hi-Speed transfers continuous transfer mode must not be used in the USB Basic driver.
5. The storage area must be formatted before use.
6. When using the SD/MMC Block Media Implementation (rm_block_media_sdmmc), "Card Detection" must be set to "Not Used" in the SD/MMC Host Interface (r_sdhi) settings.
7. The driver does not support Low-speed.
8. This driver does not support simultaneous operation with USB Host device class.

Examples

USB PMSC Example

In this example, when the evaluation board is connected to the host PC it is recognized as a removable disk and reading/writing files is possible. The FAT type is either FAT12, FAT16, or FAT32 depending on the size of the media used.

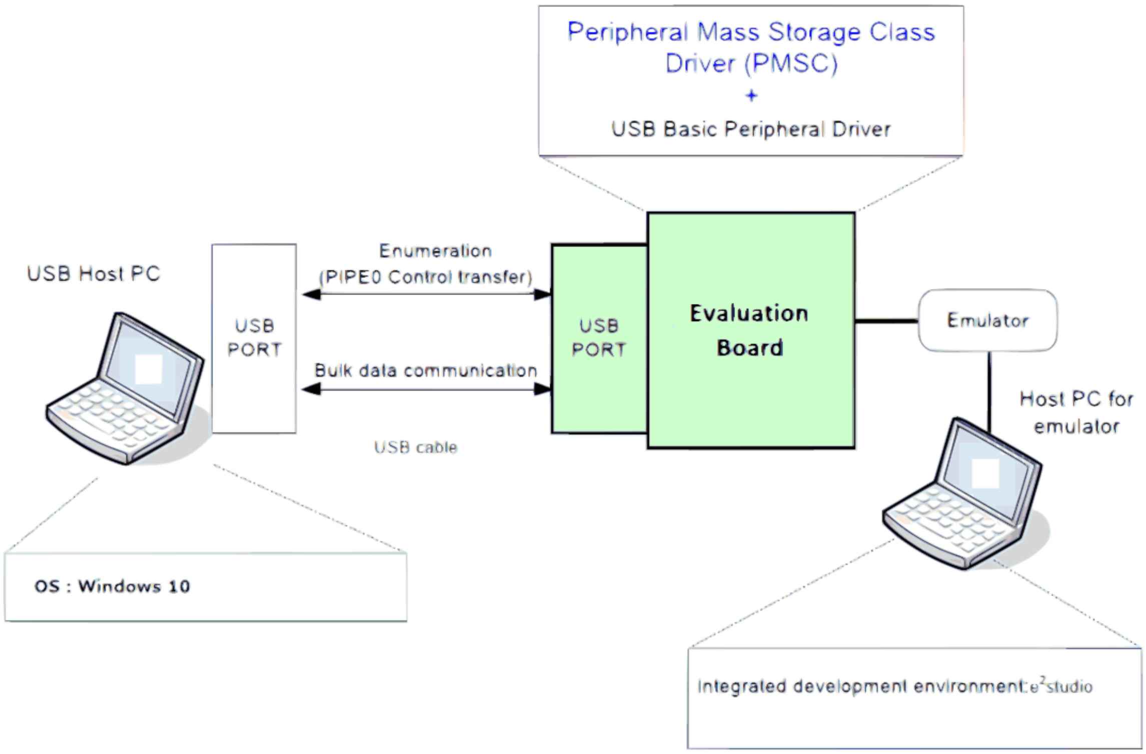


Figure 199: Example Operating Environment

```
void usb_pmsc_example (void)
{
    usb_event_info_t usb_event;
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#else
 usb_status_t event;
#endif
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 /* Loop back between PC(TerminalSoft) and USB MCU */
 while (1)
    {
#if (BSP_CFG_RTOS == 2)
       USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **) &p_mess);
       usb_event = *p_mess;
 /* Analyzing the received message */
 switch (usb_event.event)
#else  /* (BSP_CFG_RTOS == 2) */
```

```
        g_usb_on_usb.eventGet(&usb_event, &event);
 switch (event)
#endif  /* (BSP_CFG_RTOS == 2) */
      {
 case USB_STATUS_CONFIGURED:
      {
 break;
            }
 case USB_STATUS_SUSPEND:
 case USB_STATUS_DETACH:
      {
#if USB_SUPPORT_LPW == USB_APL_ENABLE
// @@ low_power_mcu();
#endif  /* USB_SUPPORT_LPW == USB_APL_ENABLE */
 break;
            }
 default:
      {
 break;
            }
       }
    }
} /* End of function usb_main() */
```

Descriptor

A template for PMSC descriptors can be found in ra/fsp/src/r_usb_pmsc/r_usb_pmsc_descriptor.c.template. Also, please be sure to use your vendor ID.

4.2.71 USB Peripheral Vendor Class (r_usb_pvnd)

Modules

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Overview

USB Peripheral Vendor class works by combining r_usb_basic module.

How to Configuration

The following shows FSP configuration procedure for USB Peripheral Vendor class.

- Select [New Stack]->[Middleware]->[USB]->[USB Peripheral Vendor class driver on r_usb_pvnd].

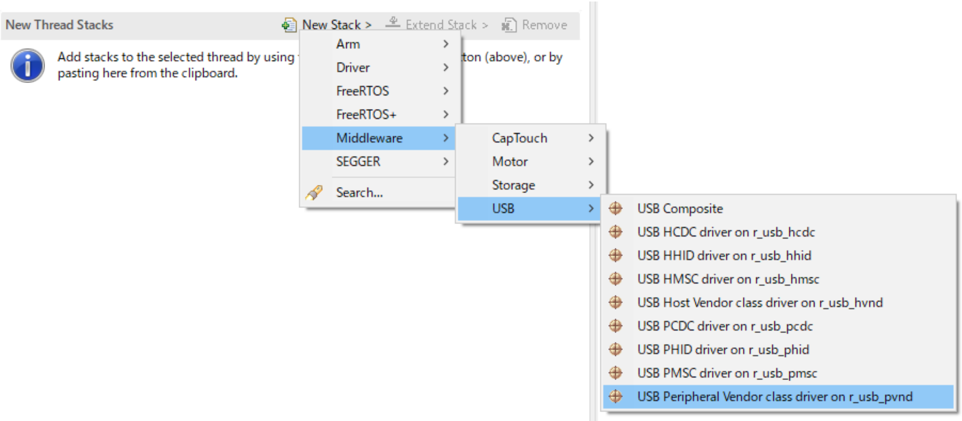


Figure 200: Select USB Peripheral Vendor Class

- The following is displayed when selecting [USB Peripheral Vendor class driver on r_usb_pvnd]. The user does not specify USB pipe number in Vendor class.

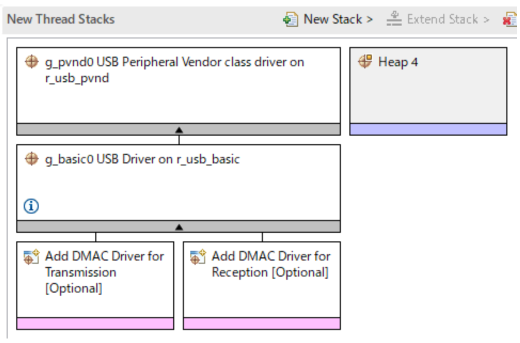


Figure 201: USB Peripheral Vendor Class Stack

API

Use the following APIs in Peripheral Vendor class application program.

- For Data Transfer
  Use the following APIs for data transfer for Bulk transfer or Interrupt transfer.

  1. R_USB_PipeRead()
  2. R_USB_PipeWrite()
  3. R_USB_PipeStop()

- For Control Transfer
  Use the following API for the class request processing.

  1. R_USB_PeriControlDataGet()
  2. R_USB_PeriControlDataSet()
  3. R_USB_PeriControlStatusSet()

- For USB Pipe Information
  The USB driver allocates USB PIPE by analyzing the descriptor of USB device in Vendor class. Use the following APIs to get the allocated USB pipe information.

  1. R_USB_UsedPipesGet()
  2. R_USB_PipeInfoGet()

USB PIPE Allocation

The USB driver allocates USB PIPE by analyzing the descriptor of USB device in Vendor class.
The USB PIPE related to the Endpoint Descriptor are allocated in order from USB PIPE1 according to the description order of the Endpoint Descriptor.

Limitations

This Peripheral Vendor class can not be included in composite device.

Descriptor

Template for Vendor class descriptor can be found in ra/fsp/src/r_usb_pvnd folder. Also, please be sure to use your vendor ID.

Examples

This application program processes the follwoing after the enumeration completes with USB device.

1. Getting USB Pipe Infomattion
2. Vendor Class Request Processing
3. Loopback processing of bulk transfer and interrupt transfer.

```
/***************************************************************************
**
 * Macro definitions
 ***************************************************************************
**/
/* for Vendor Class Request */
```

```
#define USB_SET_VENDOR_NO_DATA (0x0000U)
#define USB_SET_VENDOR (0x0100U)
#define USB_GET_VENDOR (0x0200U)
/*****************************************************************************
**
 * Function Name : usb_main
 * Description : main routine or task for peripheral vendor class
application.
 * Arguments : none:
 * Return value : none
 *****************************************************************************
**/
void main_task (void)
{
#if (BSP_CFG_RTOS == 2)
    usb_event_info_t * p_mess;
#endif
 usb_status_t    event;
    usb_event_info_t event_info;
    uint8_t          bulk_out_pipe = 0; /* Bulk Out Pipe       */
    uint8_t          bulk_in_pipe = 0; /* Bulk In Pipe       */
    uint8_t          int_out_pipe = 0; /* Interrupt Out Pipe   */
    uint8_t          int_in_pipe   = 0; /* Interrupt In Pipe   */
    uint16_t   buf_type       = 0;
    uint8_t    pipe          = 0;
    uint8_t    is_zlp[2]      = {0, 0};
    uint32_t   request_length = 0;
    uint16_t   used_pipe      = 0;
    usb_pipe_t pipe_info;
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
#if (BSP_CFG_RTOS == 2)
       USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **) &p_mess);
```

```
       event_info = *p_mess;
 event      = event_info.event;
#else  /* (BSP_CFG_RTOS == 2) */
       g_usb_on_usb.eventGet(&event_info, &event);
#endif  /* (BSP_CFG_RTOS == 2) */
 switch (event)
      {
 case USB_STATUS_CONFIGURED:
      {
                buffer_init();
                is_zlp[0] = 0;
                is_zlp[1] = 0;
                g_usb_on_usb.usedPipesGet(&g_basic0_ctrl, &used_pipe,
USB_CLASS_PVND);
 for (pipe = START_PIPE; pipe < END_PIPE; pipe++)
      {
 if ((used_pipe & (1 << pipe)) != 0)
      {
                        g_usb_on_usb.pipeInfoGet(&g_basic0_ctrl, &pipe_info,
pipe);
 if (USB_EP_DIR_IN != (pipe_info.endpoint & USB_EP_DIR_IN))
      {
 /* Out Transfer */
 if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
      {
                               buf_type      = BUF_BULK;
                               bulk_out_pipe = pipe;
                            }
 else
      {
                               buf_type    = BUF_INT;
                               int_out_pipe = pipe;
                            }
                        }
```

```
else
     {
/* In Transfer */
if (USB_TRANSFER_TYPE_BULK == pipe_info.transfer_type)
     {
                              buf_type    = BUF_BULK;
                              bulk_in_pipe = pipe;
                           }
else
     {
                              buf_type    = BUF_INT;
                              int_in_pipe = pipe;
                           }
                       }
                  }
               }
break;
           }
case USB_STATUS_READ_COMPLETE:
     {
if (FSP_ERR_USB_FAILED != event_info.status)
     {
if (bulk_out_pipe == event_info.pipe)
     {
                       buf_type = BUF_BULK;
                       pipe    = bulk_in_pipe;
                  }
else if (int_out_pipe == event_info.pipe)
     {
                       buf_type = BUF_INT;
                       pipe    = int_in_pipe;
                  }
else
     {
```

```
while (1)
     {
                           ;
                       }
                  }
                  buffer_check(buf_type, event_info.data_size);
                  g_usb_on_usb.pipeWrite(&g_basic0_ctrl,
&g_buf[buf_type][0], event_info.data_size, pipe);
               }
break;
           }
case USB_STATUS_WRITE_COMPLETE:
     {
if (bulk_in_pipe == event_info.pipe)
     {
                  buf_type = BUF_BULK;
if (1 == is_zlp[buf_type])
     {
                       pipe = bulk_out_pipe;
                  }
               }
else if (int_in_pipe == event_info.pipe)
     {
                  buf_type = BUF_INT;
if (1 == is_zlp[buf_type])
     {
                       pipe = int_out_pipe;
                  }
               }
else
     {
/* Nothing */
               }
if (1 == is_zlp[buf_type])
```

```
     {
                    is_zlp[buf_type] = 0;
                    buffer_clear(buf_type);
                    g_usb_on_usb.pipeRead(&g_basic0_ctrl, &g_buf[buf_type][0],
BUF_SIZE, pipe);
                }
 else
     {
                    is_zlp[buf_type] = 1;
                    g_usb_on_usb.pipeWrite(&g_basic0_ctrl, 0, 0,
event_info.pipe); /* Send ZLP */
                }
 break;
            }
 case USB_STATUS_REQUEST:
     {
 if (USB_SET_VENDOR_NO_DATA == (event_info.setup.request_type & USB_BREQUEST
))
     {
                    g_usb_on_usb.periControlStatusSet(&g_basic0_ctrl,
USB_SETUP_STATUS_ACK);
                }
 else if (USB_SET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
     {
                    request_length = event_info.setup.request_length;
                    g_usb_on_usb.periControlDataGet(&g_basic0_ctrl,
&g_request_buf[0], request_length);
                }
 else if (USB_GET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
     {
                    g_usb_on_usb.periControlDataSet(&g_basic0_ctrl,
&g_request_buf[0], request_length);
                }
 else
```

```
      {
 /* Nothing */
                }
 break;
            }
 case USB_STATUS_REQUEST_COMPLETE:
      {
 if (USB_GET_VENDOR == (event_info.setup.request_type & USB_BREQUEST))
      {
                   g_usb_on_usb.pipeRead(&g_basic0_ctrl, &g_buf[BUF_BULK][0],
BUF_SIZE, bulk_out_pipe);
                   g_usb_on_usb.pipeRead(&g_basic0_ctrl, &g_buf[BUF_INT][0],
BUF_SIZE, int_out_pipe);
                }
 break;
            }
 case USB_STATUS_DETACH:
      {
 break;
            }
 default:
      {
 break;
            }
       }
    }
} /* End of function usb_main */
/******************************************************************************
**
 * Function Name : buffer_init
 * Description : buffer initialization
 * Arguments : none
 * Return value : none
 *****************************************************************************
```

```
**/
static void buffer_init (void)
{
    uint16_t i;
    uint16_t j;
 for (j = 0; j < 2; j++)
    {
 for (i = 0; i < BUF_SIZE; i++)
      {
            g_buf[j][i] = (uint8_t) i;
       }
    }
}
/****************************************************************************
**
 * Function Name : buffer_check
 * Description : buffer check
 * Arguments : buf_type : buffer number
 * Return value : none
 ****************************************************************************
**/
static void buffer_check (uint16_t buf_type, uint32_t size)
{
    uint16_t i;
 for (i = 0; i < (uint16_t) size; i++)
    {
 if ((uint8_t) (i & USB_VALUE_FF) != g_buf[buf_type][i])
      {
 while (1)
      {
                ;
            }
       }
    }
```

```
}
/****************************************************************************
**
 * Function Name : buffer_clear
 * Description : buffer clear
 * Arguments : buf_type : buffer number
 * Return value : none
 ****************************************************************************
**/
static void buffer_clear (uint16_t buf_type)
{
    uint16_t i;
 for (i = 0; i < BUF_SIZE; i++)
    {
       g_buf[buf_type][i] = 0;
    }
}
/****************************************************************************
**
 * End of function usb_mcu_init
 ****************************************************************************
**/
#if (BSP_CFG_RTOS == 2)
/****************************************************************************
**
 * Function Name : usb_apl_rec_msg
 * Description : Receive a message to the specified id (mailbox).
 * Argument : uint8_t id : ID number (mailbox).
 * : usb_msg_t** mess : Message pointer
 * : usb_tm_t tm : Timeout Value
 * Return : uint16_t : USB_OK / USB_ERROR
 ****************************************************************************
**/
usb_er_t usb_apl_rec_msg (uint8_t id, usb_msg_t ** mess, usb_tm_t tm)
```

```
{
    BaseType_t    err;
    QueueHandle_t handle;
    usb_er_t      result;
    (void) tm;
 if (NULL == mess)
    {
 return USB_APL_ERROR;
    }
    handle = (*(g_apl_mbx_table[id]));
    *mess = NULL;
    err = xQueueReceive(handle, (void *) mess, (portMAX_DELAY));
 if ((pdTRUE == err) && (NULL != (*mess)))
    {
       result = USB_APL_OK;
    }
 else
    {
       result = USB_APL_ERROR;
    }
 return result;
}
/****************************************************************************
**
 * End of function usb_apl_rec_msg
 ****************************************************************************
**/
/****************************************************************************
**
 * Function Name : usb_apl_snd_msg
 * Description : Send a message to the specified id (mailbox).
 * Argument : uint8_t id : ID number (mailbox).
 * : usb_msg_t* mess : Message pointer
 * Return : usb_er_t : USB_OK / USB_ERROR
```

```
 ***************************************************************************
**/
usb_er_t usb_apl_snd_msg (uint8_t id, usb_msg_t * mess)
{
    BaseType_t    err;
    QueueHandle_t handle;
    usb_er_t      result;
 if (NULL == mess)
    {
 return USB_APL_ERROR;
    }
    handle = (*(g_apl_mbx_table[id]));
    err = xQueueSend(handle, (const void *) &mess, (TickType_t) (0));
 if (pdTRUE == err)
    {
       result = USB_APL_OK;
    }
 else
    {
       result = USB_APL_ERROR;
    }
 return result;
}
/***************************************************************************
**
 * End of function usb_apl_snd_msg
 ***************************************************************************
**/
#endif  /* #if (BSP_CFG_RTOS == 2) */
```

4.2.72 Watchdog Timer (r_wdt)

Modules

Functions

fsp_err_t	R_WDT_Refresh (wdt_ctrl_t *const p_ctrl)
fsp_err_t	R_WDT_Open (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)
fsp_err_t	R_WDT_StatusClear (wdt_ctrl_t *const p_ctrl, const wdt_status_t status)
fsp_err_t	R_WDT_StatusGet (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)
fsp_err_t	R_WDT_CounterGet (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)
fsp_err_t	R_WDT_TimeoutGet (wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)
fsp_err_t	R_WDT_CallbackSet (wdt_ctrl_t *const p_ctrl, void(*p_callback)(wdt_callback_args_t *), void const *const p_context, wdt_callback_args_t *const p_callback_memory)

Detailed Description

Driver for the WDT peripheral on RA MCUs. This module implements the WDT Interface.

Overview

The watchdog timer is used to recover from unexpected errors in an application. The watchdog timer must be refreshed periodically in the permitted count window by the application. If the count is allowed to underflow or refresh occurs outside of the valid refresh period, the WDT resets the device or generates an NMI.

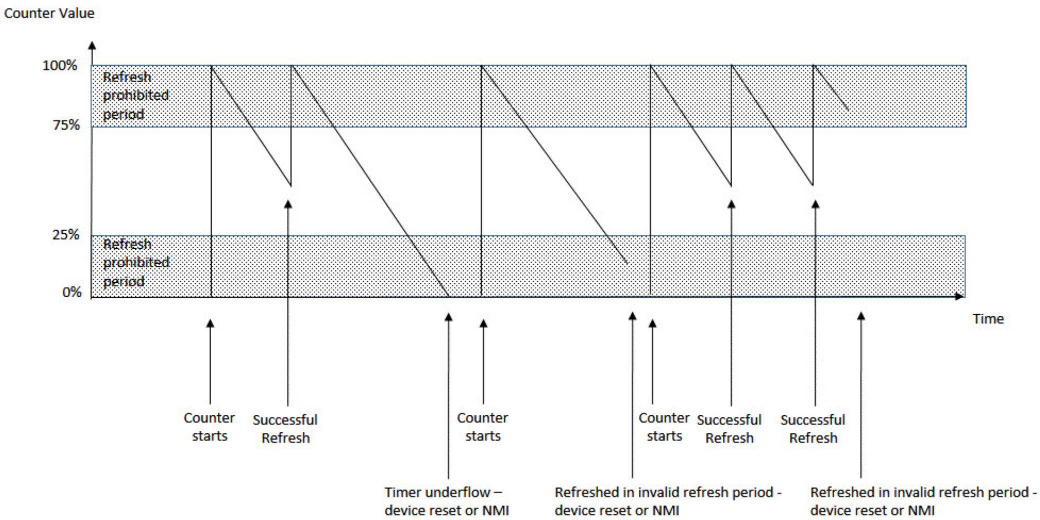


Figure 202: Watchdog Timer Operation Example

Features

The WDT HAL module has the following key features:

- When the WDT underflows or is refreshed outside of the permitted refresh window, one of the following events can occur:
  - Resetting of the device
  - Generation of an NMI
- The WDT has two supported modes:
  - In auto start mode, the WDT begins counting at reset.
  - In register start mode, the WDT can be started from the application.

Selecting a Watchdog

RA MCUs have two watchdog peripherals: the watchdog timer (WDT) and the independent watchdog timer (IWDT). When selecting between them, consider these factors:

	WDT	IWDT
Start Mode	The WDT can be started from the application (register start mode) or configured by hardware to start automatically (auto start mode).	The IWDT can only be configured by hardware to start automatically.
Clock Source	The WDT runs off a peripheral clock.	The IWDT has its own clock source which improves safety.

Configuration

When using register start mode, configure the watchdog timer on the Stacks tab.

Note

*When using auto start mode, configurations on the **Stacks** tab are ignored. Configure the watchdog using the OFS settings on the BSP tab.*

Build Time Configurations for r_wdt

The following build time configurations are defined in fsp_cfg/r_wdt_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Register Start NMI Support	• Enabled • Disabled	Disabled	If enabled, code for NMI support in register start mode is included in the build.

Configurations for Monitoring > Watchdog (r_wdt)

This module can be added to the Stacks tab via New Stack > Monitoring > Watchdog (r_wdt). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_wdt0	Module name.
Timeout	• 1,024 Cycles • 4,096 Cycles • 8,192 Cycles • 16,384 Cycles	16,384 Cycles	Select the watchdog timeout in cycles.
Clock Division Ratio	• PCLK/4 • PCLK/64 • PCLK/128 • PCLK/512 • PCLK/2048 • PCLK/8192	PCLK/8192	Select the watchdog clock divisor.
Window Start Position	• 100% (Window Position Not Specified) • 75% • 50% • 25	100% (Window Position Not Specified)	Select the allowed watchdog refresh start point.
Window End Position	• 0% (Window Position Not	0% (Window Position Not Specified)	Select the allowed watchdog refresh end

	Specified) • 25% • 50% • 75%		point.
Reset Control	• Reset Output • NMI Generated	Reset Output	Select what happens when the watchdog timer expires.
Stop Control	• WDT Count Enabled in Low Power Mode • WDT Count Disabled in Low Power Mode	WDT Count Disabled in Low Power Mode	Select the watchdog state in low power mode.
NMI Callback	Name must be a valid C symbol	NULL	A user callback function must be provided if the WDT is configured to generate an NMI when the timer underflows or a refresh error occurs. If this callback function is provided, it will be called from the NMI handler each time the watchdog triggers.

Clock Configuration

The WDT clock is based on the PCLKB frequency. You can set the PCLKB frequency using the **Clocks** tab of the RA Configuration editor or by using the CGC Interface at run-time. The maximum timeout period with PCLKB running at 60 MHz is approximately 2.2 seconds.

Pin Configuration

This module does not use I/O pins.

Usage Notes

NMI Interrupt

The watchdog timer uses the NMI, which is enabled by default. No special configuration is required. When the NMI is triggered, the callback function registered during open is called.

Note

When using the WDT in software start mode with NMI and the timer underflows, the WDT status must be reset by calling R_WDT_StatusClear before restarting the timer via R_WDT_Refresh.

Period Calculation

The WDT operates from PCLKB. With a PCLKB of 60 MHz, the maximum time from the last refresh to device reset or NMI generation will be just over 2.2 seconds as detailed below.

PLCKB = 60 MHz

Clock division ratio = PCLKB / 8192
Timeout period = 16384 cycles
WDT clock frequency = 60 MHz / 8192 = 7.324 kHz
Cycle time = 1 / 7.324 kHz = 136.53 us
Timeout = 136.53 us x 16384 cycles = 2.23 seconds

Limitations

Developers should be aware of the following limitations when using the WDT:

- When using a J-Link debugger the WDT counter does not count and therefore will not reset the device or generate an NMI. To enable the watchdog to count and generate a reset or NMI while debugging, add this line of code in the application:

```
/* (Optional) Enable the WDT to count and generate NMI or reset when the
 * debugger is connected. */
   R_DEBUG->DBGSTOPCR_b.DBGSTOP_WDT = 0;
```

- If the WDT is configured to stop the counter in low power mode, then your application must restart the watchdog by calling R_WDT_Refresh() after the MCU wakes from low power mode.

Examples

WDT Basic Example

This is a basic example of minimal use of the WDT in an application.

```
void wdt_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* In auto start mode, the WDT starts counting immediately when the MCU is powered
on. */
 /* Initializes the module. */
    err = R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* In register start mode, start the watchdog by calling R_WDT_Refresh. */
    err = R_WDT_Refresh(&g_wdt0_ctrl);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Application work here. */
```

```
 /* Refresh before the counter underflows to prevent reset or NMI. */
      err = R_WDT_Refresh(&g_wdt0_ctrl);
      assert(FSP_SUCCESS == err);
   }
}
```

WDT Advanced Example

This example demonstrates using a start window and gives an example callback to handle an NMI generated by an underflow or refresh error.

```
#define WDT_TIMEOUT_COUNTS (16384U)
#define WDT_MAX_COUNTER (WDT_TIMEOUT_COUNTS - 1U)
#define WDT_START_WINDOW_75 ((WDT_MAX_COUNTER * 3) / 4)
/* Example callback called when a watchdog NMI occurs. */
void wdt_callback (wdt_callback_args_t * p_args)
{
 FSP_PARAMETER_NOT_USED(p_args);
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) Determine the source of the NMI. */
 wdt_status_t status = WDT_STATUS_NO_ERROR;
    err = R_WDT_StatusGet(&g_wdt0_ctrl, &status);
    assert(FSP_SUCCESS == err);
 /* (Optional) Log source of NMI and any other debug information. */
 /* (Optional) Clear the error flags. */
    err = R_WDT_StatusClear(&g_wdt0_ctrl, status);
    assert(FSP_SUCCESS == err);
 /* (Register start mode) In register start mode, call R_WDT_Refresh() to
  * continue using the watchdog after an error. */
    err = R_WDT_Refresh(&g_wdt0_ctrl);
    assert(FSP_SUCCESS == err);
 /* (Optional) Issue a software reset to reset the MCU. */
    __NVIC_SystemReset();
}
void wdt_advanced_example (void)
```

```
{
 fsp_err_t err = FSP_SUCCESS;
 /* (Optional) Enable the WDT to count and generate NMI or reset when the
  * debugger is connected. */
    R_DEBUG->DBGSTOPCR_b.DBGSTOP_WDT = 0;
 /* (Optional) Check if the WDTRF flag is set to know if the system is
  * recovering from a WDT reset. */
 if (R_SYSTEM->RSTSR1_b.WDTRF)
    {
 /* Clear the flag. */
       R_SYSTEM->RSTSR1 = 0U;
    }
 /* Open the module. */
    err = R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize other application code. */
 /* (Register start mode) Call R_WDT_Refresh() to start the WDT in register
  * start mode. Do not call R_WDT_Refresh() in auto start mode unless the
  * counter is in the acceptable refresh window. */
    err = R_WDT_Refresh(&g_wdt0_ctrl);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Application work here. */
 /* (Optional) If there is a chance the application takes less time than
  * the start window, verify the WDT counter is past the start window
  * before refreshing the WDT. */
       uint32_t wdt_counter = 0U;
 do
      {
 /* Read the current WDT counter value. */
           err = R_WDT_CounterGet(&g_wdt0_ctrl, &wdt_counter);
      assert(FSP_SUCCESS == err);
```

```
        } while (wdt_counter >= WDT_START_WINDOW_75);
 /* Refresh before the counter underflows to prevent reset or NMI. */
        err = R_WDT_Refresh(&g_wdt0_ctrl);
        assert(FSP_SUCCESS == err);
    }
}
```

Data Structures

struct	wdt_instance_ctrl_t

Data Structure Documentation

◆ wdt_instance_ctrl_t

struct wdt_instance_ctrl_t
WDT private control block. DO NOT MODIFY. Initialization occurs when R_WDT_Open() is called.

Function Documentation

◆ R_WDT_Refresh()

fsp_err_t R_WDT_Refresh (wdt_ctrl_t *const *p_ctrl*)

Refresh the watchdog timer. Implements wdt_api_t::refresh.

In addition to refreshing the watchdog counter this function can be used to start the counter in register start mode.

Example:

```
 /* Refresh before the counter underflows to prevent reset or NMI. */
        err = R_WDT_Refresh(&g_wdt0_ctrl);
        assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	WDT successfully refreshed.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Instance control block is not initialized.

Note

This function only returns FSP_SUCCESS. If the refresh fails due to being performed outside of the permitted refresh period the device will either reset or trigger an NMI ISR to run.

◆ R_WDT_Open()

fsp_err_t R_WDT_Open (wdt_ctrl_t *const *p_ctrl*, wdt_cfg_t const *const *p_cfg*)

Configure the WDT in register start mode. In auto-start_mode the NMI callback can be registered. Implements wdt_api_t::open.

This function should only be called once as WDT configuration registers can only be written to once so subsequent calls will have no effect.

Example:

```
/* Initializes the module. */
    err = R_WDT_Open(&g_wdt0_ctrl, &g_wdt0_cfg);
```

Return values

FSP_SUCCESS	WDT successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_STATE	The security state of the NMI and the module do not match.

Note

In auto start mode the only valid configuration option is for registering the callback for the NMI ISR if NMI output has been selected.

◆ R_WDT_StatusClear()

fsp_err_t R_WDT_StatusClear (wdt_ctrl_t *const *p_ctrl*, const wdt_status_t *status*)

Clear the WDT status and error flags. Implements wdt_api_t::statusClear.

Example:

```
/* (Optional) Clear the error flags. */
   err = R_WDT_StatusClear(&g_wdt0_ctrl, status);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	WDT flag(s) successfully cleared.
FSP_ERR_ASSERTION	Null pointer as a parameter.
FSP_ERR_NOT_OPEN	Instance control block is not initialized.
FSP_ERR_UNSUPPORTED	This function is only valid if the watchdog generates an NMI when an error occurs.

Note

When the WDT is configured to output a reset on underflow or refresh error reading the status and error flags serves no purpose as they will always indicate that no underflow has occurred and there is no refresh error. Reading the status and error flags is only valid when interrupt request output is enabled.

◆ R_WDT_StatusGet()

fsp_err_t R_WDT_StatusGet (wdt_ctrl_t *const *p_ctrl*, wdt_status_t *const *p_status*)

Read the WDT status flags. Implements wdt_api_t::statusGet.

Indicates both status and error conditions.

Example:

```
/* (Optional) Determine the source of the NMI. */
wdt_status_t status = WDT_STATUS_NO_ERROR;
   err = R_WDT_StatusGet(&g_wdt0_ctrl, &status);
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	WDT status successfully read.
FSP_ERR_ASSERTION	Null pointer as a parameter.
FSP_ERR_NOT_OPEN	Instance control block is not initialized.
FSP_ERR_UNSUPPORTED	This function is only valid if the watchdog generates an NMI when an error occurs.

Note

When the WDT is configured to output a reset on underflow or refresh error reading the status and error flags serves no purpose as they will always indicate that no underflow has occurred and there is no refresh error. Reading the status and error flags is only valid when interrupt request output is enabled.

◆ R_WDT_CounterGet()

fsp_err_t R_WDT_CounterGet (wdt_ctrl_t *const *p_ctrl*, uint32_t *const *p_count*)

Read the current count value of the WDT. Implements wdt_api_t::counterGet.

Example:

```
/* Read the current WDT counter value. */
        err = R_WDT_CounterGet(&g_wdt0_ctrl, &wdt_counter);
     assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	WDT current count successfully read.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Instance control block is not initialized.

◆ R_WDT_TimeoutGet()

fsp_err_t R_WDT_TimeoutGet (wdt_ctrl_t *const *p_ctrl*, wdt_timeout_values_t *const *p_timeout*)

Read timeout information for the watchdog timer. Implements wdt_api_t::timeoutGet.

Return values

FSP_SUCCESS	WDT timeout information retrieved successfully.
FSP_ERR_ASSERTION	Null Pointer.
FSP_ERR_NOT_OPEN	Instance control block is not initialized.

◆ R_WDT_CallbackSet()

fsp_err_t R_WDT_CallbackSet (wdt_ctrl_t *const *p_ctrl*, void(*)(wdt_callback_args_t *) *p_callback*, void const *const *p_context*, wdt_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements wdt_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

4.2.73 ADPCM Decoder (rm_adpcm_decoder)

Modules

Functions

fsp_err_t	RM_ADPCM_DECODER_Open (adpcm_decoder_ctrl_t *p_ctrl, adpcm_decoder_cfg_t const *const p_cfg)
fsp_err_t	RM_ADPCM_DECODER_Decode (adpcm_decoder_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t src_len_bytes)
fsp_err_t	RM_ADPCM_DECODER_Reset (adpcm_decoder_ctrl_t *p_ctrl)

fsp_err_t RM_ADPCM_DECODER_Close (adpcm_decoder_ctrl_t *p_ctrl)

Detailed Description

Middleware to implement the ADPCM Audio Decoder. This module implements the ADPCM Decoder Interface.

Overview

Features

The ADPCM Audio Decoder has the following key features:

- Decodes 4-bit ADPCM input to 16-bit PCM output

Configuration

Build Time Configurations for rm_adpcm_decoder

The following build time configurations are defined in fsp_cfg/rm_adpcm_decoder_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Audio > ADPCM Decoder (rm_adpcm_decoder)

This module can be added to the Stacks tab via New Stack > Audio > ADPCM Decoder (rm_adpcm_decoder).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_adpcm_decoder0	Module name.

Clock Configuration

This module has no required clock configurations.

Pin Configuration

This module does not use I/O pins.

Examples

Basic Example

This is a basic example of minimal use of the ADPCM Audio Decoder implementation in an application.

```
void rm_adpcm_decoder_example ()
{
 /* Open the ADPCM audio decoder instance. */
 fsp_err_t err = RM_ADPCM_DECODER_Open(&g_adpcmdec_ctrl, &g_adpcmdec_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Decode the data */
    err = RM_ADPCM_DECODER_Decode(&g_adpcmdec_ctrl, g_adpcm_stream1, g_pcm_stream,
ADPCM_BUFFER_SIZE_BYTES);
    assert(FSP_SUCCESS == err);
 /* Reset the ADPCM audio decoder instance before decoding a new stream. */
    err = RM_ADPCM_DECODER_Reset(&g_adpcmdec_ctrl);
    assert(FSP_SUCCESS == err);
 /* Decode the first chunk of ADPCM data */
    err = RM_ADPCM_DECODER_Decode(&g_adpcmdec_ctrl, g_adpcm_stream2, g_pcm_stream,
(ADPCM_BUFFER_SIZE_BYTES/2));
    assert(FSP_SUCCESS == err);
 /* Decode the second chunk of ADPCM data */
    err = RM_ADPCM_DECODER_Decode(&g_adpcmdec_ctrl,
&g_adpcm_stream2[ADPCM_BUFFER_SIZE_BYTES/2],
                                  g_pcm_stream, (ADPCM_BUFFER_SIZE_BYTES/2));
    assert(FSP_SUCCESS == err);
}
```

Data Structures

struct	adpcm_decoder_instance_ctrl_t

Data Structure Documentation

◆ adpcm_decoder_instance_ctrl_t

struct adpcm_decoder_instance_ctrl_t
RM_ADPCM_DECODER instance control block. DO NOT INITIALIZE. Initialized in adpcm_decoder_api_t::open().

Function Documentation

◆ RM_ADPCM_DECODER_Open()

fsp_err_t RM_ADPCM_DECODER_Open (adpcm_decoder_ctrl_t * *p_ctrl*, adpcm_decoder_cfg_t const *const *p_cfg*)

Initializes ADPCM audio decoder device.

Implements adpcm_decoder_api_t::open().

Return values

FSP_SUCCESS	Module is ready for use.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_ALREADY_OPEN	The instance control structure has already been opened.

◆ RM_ADPCM_DECODER_Decode()

fsp_err_t RM_ADPCM_DECODER_Decode (adpcm_decoder_ctrl_t *const *p_ctrl*, void const * *p_src*, void * *p_dest*, uint32_t *src_len_bytes*)

Decodes 4bit ADPCM data to 16bit PCM data. It reads ADPCM data from area pointed by inputAddr pointer, decodes the number of samples specified and stores the decoded data in buffer pointed with outputAddr pointer.

Implements adpcm_decoder_api_t::decode().

Return values

FSP_SUCCESS	Decode operation successfully completed.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ RM_ADPCM_DECODER_Reset()

fsp_err_t RM_ADPCM_DECODER_Reset (adpcm_decoder_ctrl_t * *p_ctrl*)

This function resets the ADPCM decoder device.

Implements adpcm_decoder_api_t::reset().

Return values

FSP_SUCCESS	Module closed.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

◆ RM_ADPCM_DECODER_Close()

fsp_err_t RM_ADPCM_DECODER_Close (adpcm_decoder_ctrl_t * *p_ctrl*)

This function closes the ADPCM decoder device.

Implements adpcm_decoder_api_t::close().

Return values

FSP_SUCCESS	Module closed.
FSP_ERR_ASSERTION	An input argument is invalid.
FSP_ERR_NOT_OPEN	Unit is not open.

4.2.74 Audio Playback with PWM (rm_audio_playback_pwm)

Modules

Functions

fsp_err_t	RM_AUDIO_PLAYBACK_PWM_Open (audio_playback_ctrl_t *const p_api_ctrl, audio_playback_cfg_t const *const p_cfg)
fsp_err_t	RM_AUDIO_PLAYBACK_PWM_Start (audio_playback_ctrl_t *const p_api_ctrl)
fsp_err_t	RM_AUDIO_PLAYBACK_PWM_Stop (audio_playback_ctrl_t *const p_api_ctrl)
fsp_err_t	RM_AUDIO_PLAYBACK_PWM_Play (audio_playback_ctrl_t *const p_api_ctrl, void const *const p_buffer, uint32_t length)
fsp_err_t	RM_AUDIO_PLAYBACK_PWM_Close (audio_playback_ctrl_t *const p_api_ctrl)

Detailed Description

Driver for the Audio Playback middleware on RA MCUs. This module implements the AUDIO PLAYBACK Interface.

Overview

Features

The Audio Playback with PWM middleware is used to play audio streams at user selected playback rate using Pulse Width Modulation hardware on GPT or AGT timers. This module can play the 16 bit uncompressed, unsigned PCM audio stream when AGT is selected as PWM interface and can play 32 bit uncompressed, unsigned PCM audio stream when GPT is used as PWM interface. The application code is expected to convert the signed PCM data to unsigned PCM data and scale it with the playback rate before starting the playback.

Configuration

Build Time Configurations for rm_audio_playback_pwm

The following build time configurations are defined in fsp_cfg/rm_audio_playback_pwm_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
DMAC Support	• Enabled • Disabled	Disabled	Select if DMAC will be used.

Configurations for Audio > Audio Playback PWM (rm_audio_playback_pwm)

This module can be added to the Stacks tab via New Stack > Audio > Audio Playback PWM (rm_audio_playback_pwm).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_rm_audio_playback0	Module name.
General > Playback Speed (Hz)	Manual Entry	44100	Enter playback sample rate in Hz.
Interrupts > Callback	Name must be a valid C symbol	g_rm_audio_playback0_callback	A user callback function. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the playback completes.
PWM Output Pin	• Pin A • Pin B	Pin A	Select which timer output pin should be used for audio output.

Clock Configuration

The Audio Playback with PWM module does not require a specific clock configuration.

Pin Configuration

Configure the PWM output pins for selected PWM HAL layer peripheral (AGT/GPT). One of the following pins needs to be selected and enabled as PWM output for selected channel n, If GPT is used as PWM interface,

- GTIOCAn
- GTIOCBn

If AGT is used as PWM interface,

- AGTOAn
- AGTOBn

Usage Notes

DMAC/DTC Integration

DMAC/DTC is used as a lower level transfer instance with this module and is operated in Normal mode to transfer 16 bit or 32 bit data from the audio stream buffer to the PWM peripheral AGT or GPT respectively. Destination address for transfer instance needs to be the Duty Cycle setting register GTCCR for GPT as PWM driver or AGTMA/AGTCMB in case of AGT as PWM driver. The Audio Playback with PWM module internally configures 'Transfer Size' as 2 Bytes if AGT is used for PWM generation, otherwise it configures 'Transfer Size" as 4 Bytes if GPT is used for PWM generation.

Examples

Basic Example

This is a basic example of minimal use of the RM_AUDIO_PLAYBACK_PWM in an application. This example shows how this driver can be used for playing a 16 bit uncompressed PCM audio from a single input buffer.

```
int16_t play_buffer[AUDIO_EXAMPLE_LENGTH];
uint32_t g_audio_callback_counter = 0;
void g_audio_example_counter_callback (audio_playback_callback_args_t * p_args)
{
 if (AUDIO_PLAYBACK_EVENT_PLAYBACK_COMPLETE == (p_args->event))
    {
       g_audio_callback_counter++;
    }
}
void basic_example (void)
{
 fsp_err_t err;
 /* Initialize the Audio Playback module for playing an audio stream. */
    err = RM_AUDIO_PLAYBACK_PWM_Open(&g_audio_playback_pwm_ctrl,
&g_audio_playback_pwm_cfg);
```

```
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Set the 16 Bit PCM audio stream to play next */
    err = RM_AUDIO_PLAYBACK_PWM_Play(&g_audio_playback_pwm_ctrl, play_buffer,
AUDIO_EXAMPLE_LENGTH);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start to the play the selected audio stream*/
    err = RM_AUDIO_PLAYBACK_PWM_Start(&g_audio_playback_pwm_ctrl);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait till the playback is completed */
 while (g_audio_callback_counter == 0)
    {
       ;
    }
 /* Stop playing. */
    err = RM_AUDIO_PLAYBACK_PWM_Stop(&g_audio_playback_pwm_ctrl);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

Streaming Example

This is an example of using Audio Playback module to play audio stream. This application uses a double buffer to store PCM sine wave data. It starts playing in the main loop, then loads the next buffer if it is ready in the callback. If the next buffer is not ready, a flag is set in the callback so the application knows to restart playing in the main loop. This example also demonstrates conversion of signed PCM format data to unsigned PWM format data along with scaling the data samples for optimum PWM wave generation.

```
#define AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ (22050U)
#define AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_PERIOD_VALUE_AT_22050HZ (0x11B7U)
#define AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK (1024U)
#define AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_TONE_FREQUENCY_HZ (800U)
#define AUDIO_PLAYBACK_PWM_EXAMPLE_SAMPLES_TO_TRANSFER (1024U)
#define AUDIO_PLAYBACK_PWM_EXAMPLE_CONVERT_TO_PWM_SAMPLES (32768U)
```

```
#define AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_DATA_BIT_SIZE (16U)
int16_t       g_stream_src[2][AUDIO_PLAYBACK_PWM_EXAMPLE_SAMPLES_TO_TRANSFER];
q15_t         g_pwm_sample[2][AUDIO_PLAYBACK_PWM_EXAMPLE_SAMPLES_TO_TRANSFER];
q15_t         g_pwm_scaled_sample[2][AUDIO_PLAYBACK_PWM_EXAMPLE_SAMPLES_TO_TRANSFER];
uint32_t      g_buffer_index          = 0;
volatile bool g_send_data_in_main_loop = true;
volatile bool g_data_ready = false;
/* Example callback called when Audio Playback is ready for more data. */
void rm_audio_playback_example_callback (audio_playback_callback_args_t * p_args)
{
 /* Start playing next stream if data is ready. */
 if (AUDIO_PLAYBACK_EVENT_PLAYBACK_COMPLETE == (p_args->event))
    {
 if (g_data_ready)
      {
 /* Reload data and handle errors. */
            rm_audio_playback_example_play();
      }
 else
      {
 /* Data was not ready yet, send it in the main loop. */
            g_send_data_in_main_loop = true;
      }
    }
}
/* Load the next stream and check for error condition. */
void rm_audio_playback_example_play (void)
{
 /* Set the playback stream */
 fsp_err_t err;
    err =
 RM_AUDIO_PLAYBACK_PWM_Play(&g_audio_playback_pwm_ctrl, (int16_t *)
&g_pwm_scaled_sample[g_buffer_index][0],
```

```
(AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK * sizeof(int16_t)));
 if (FSP_SUCCESS == err)
    {
 /* Switch the buffer after data is sent. */
       g_buffer_index = !g_buffer_index;
 /* Allow loop to calculate next buffer. */
       g_data_ready = false;
    }
 else
    {
 /* The
  * application must wait until the audio playback is completed. In this example, the
  * callback sets data or resets the flag g_send_data_in_main_loop. */
    }
}
/* Calculate samples. This example is just a sine wave. For this type of data, it
would be better to calculate
 * one period and loop it. This example should be updated for the audio data used by
the application. */
void rm_audio_playback_example_calculate_samples (uint32_t buffer_index)
{
 static uint32_t t = 0U;
 /* Create a sine wave. Using formula sample = sin(2 * pi * tone_frequency * t /
sampling_frequency) */
    uint32_t freq = AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_TONE_FREQUENCY_HZ;
 for (uint32_t i = 0; i < AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK; i
+= 1)
    {
 float input = (((float) (freq * t)) * M_TWOPI) /

AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_AUDIO_SAMPLING_FREQUENCY_HZ;
       t++;
 /* Store sample. */
       int16_t sample = (int16_t) ((INT16_MAX * sinf(input)));
```

```
        g_stream_src[buffer_index][i] = sample;
    }
 /* Convert signed PCM data to unsigned PCM data as PWM needs unsigned input. */
    arm_offset_q15(&g_stream_src[buffer_index][0],
                   (q15_t) (INT16_MAX + 1),
                   &g_pwm_sample[buffer_index][0],
                   AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK);
 /* Scale the data by the selected period for the timer (calculated for equivalent
playback rate) */
    arm_scale_q15(&g_pwm_sample[buffer_index][0],
                  AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_PERIOD_VALUE_AT_22050HZ,
                  AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_DATA_BIT_SIZE,
                  &g_pwm_scaled_sample[buffer_index][0],
                  AUDIO_PLAYBACK_PWM_STREAMING_EXAMPLE_SAMPLES_PER_CHUNK);
 /* Data is ready to be sent in the interrupt. */
    g_data_ready = true;
}
void rm_audio_playback_streaming_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initialize the module.
  * Configure the following pins in the pin configurator for PWM output:
  * - If the GPT timer is used for generation of PWM waves configure GTIOCAn or
GTIOCBn pin and enable the output
  * to these pins through the GPT module properties for desired channel n.
  * - Otherwise, if AGT is used for generation of PWM waves configure AGTOAn or
AGTOBn pin and enable the output to
  * to these pins through the AGT module properties for desired channel n.
  * Configure the DMAC/DTC destination address as following:
  * - If the GPT timer is used for generation of PWM waves, configure DMAC/DTC
destination address to the address of
  * GTCCRC register (&R_GPTn->GTCCR[2]) if PWM output pin is GTIOCA otherwise
configure to the address of GTCCRD
  * register (&R_GPTn->GTCCR[3]) if PWM output pin is GTIOCB for desired GPT channel
```

```
n.
  * - If the AGT timer is used for generation of PWM waves, configure DMAC/DTC
destination address as the address of
  * AGTCMA register (&R_AGTn->AGTCMA) if PWM output pin is AGTOA otherwise the
address of AGTCMB register
  * (&R_AGTn->AGTCMB) if the PWM output pin is AGTOB for desired AGT channel n.
  * Configure the DMAC/DTC transfer size as 4 Bytes if PWM interface is GPT timer
otherwise configure transfer size as 2 Bytes if
  * PWM interface is AGT timer. */
    err = RM_AUDIO_PLAYBACK_PWM_Open(&g_audio_playback_pwm_ctrl,
&g_audio_playback_pwm_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start timer and transfer modules. */
    err = RM_AUDIO_PLAYBACK_PWM_Start(&g_audio_playback_pwm_ctrl);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 /* Prepare data in a buffer that is not currently used for transmission. */
       rm_audio_playback_example_calculate_samples(g_buffer_index);
 /* Send data in main loop the first time, and if it was not ready in the interrupt.
*/
 if (g_send_data_in_main_loop)
      {
 /* Clear flag. */
            g_send_data_in_main_loop = false;
 /* Reload data and handle errors. */
            rm_audio_playback_example_play();
      }
 /* If the next buffer is ready, wait for the data to be sent in the interrupt. */
 while (g_data_ready)
      {
 /* Do nothing. */
```

```
        }
    }
}
```

Data Structures

struct	audio_playback_pwm_instance_ctrl_t

Data Structure Documentation

◆ audio_playback_pwm_instance_ctrl_t

struct audio_playback_pwm_instance_ctrl_t

AUDIO_PLAYBACK_PWM instance control block. DO NOT MODIFY. Initialization occurs when RM_AUDIO_PLAYBACK_PWM_Open() is called.

Data Fields

void(*	p_callback)(audio_playback_callback_args_t *p_args)
void *	p_context
audio_playback_cfg_t const *	p_cfg
	Pointer to the configuration structure.
uint32_t	open
	Used by driver to check if the control structure is valid.
timer_instance_t const *	p_lower_lvl_timer
	Timer API used to generate sampling frequency and GPT/AGT API used to access PWM hardware.
transfer_instance_t const *	p_lower_lvl_transfer
	Transfer API used to transfer data each sampling frequency.

Field Documentation

◆ **p_callback**

void(* audio_playback_pwm_instance_ctrl_t::p_callback) (audio_playback_callback_args_t *p_args)

Callback called when play is complete.

◆ **p_context**

void* audio_playback_pwm_instance_ctrl_t::p_context

Placeholder for user data. Passed to the user callback in audio_playback_callback_args_t.

Function Documentation

◆ RM_AUDIO_PLAYBACK_PWM_Open()

fsp_err_t RM_AUDIO_PLAYBACK_PWM_Open (audio_playback_ctrl_t *const *p_api_ctrl*, audio_playback_cfg_t const *const *p_cfg*)

Opens and configures the Audio Playback with PWM driver. Sets playback speed and transfer rate to read the audio buffer.

Example:

```
/* Initialize the Audio Playback module for playing an audio stream. */
   err = RM_AUDIO_PLAYBACK_PWM_Open(&g_audio_playback_pwm_ctrl,
&g_audio_playback_pwm_cfg);
```

Return values

FSP_SUCCESS	Audio Playback module successfully configured.
FSP_ERR_ALREADY_OPEN	Module already open.
FSP_ERR_ASSERTION	One or more pointers point to NULL or callback is NULL.

◆ RM_AUDIO_PLAYBACK_PWM_Start()

fsp_err_t RM_AUDIO_PLAYBACK_PWM_Start (audio_playback_ctrl_t *const *p_api_ctrl*)

Start the PWM HAL driver (AGT or GPT) and timer HAL (AGT or GPT) drivers.

- Example:

```
/* Start to the play the selected audio stream*/
   err = RM_AUDIO_PLAYBACK_PWM_Start(&g_audio_playback_pwm_ctrl);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Audio playback hardware started successfully.
FSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Driver not open. This function calls • timer_api_t::start

◆ RM_AUDIO_PLAYBACK_PWM_Stop()

fsp_err_t RM_AUDIO_PLAYBACK_PWM_Stop (audio_playback_ctrl_t *const *p_api_ctrl*)

Stop the PWM HAL driver (AGT or GPT) and timer HAL driver (AGT or GPT).

- Example:

```
/* Stop playing. */
   err = RM_AUDIO_PLAYBACK_PWM_Stop(&g_audio_playback_pwm_ctrl);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Audio playback hardware stopped successfully.
FSP_ERR_ASSERTION	The parameter p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Driver not open. This function calls • timer_api_t::stop

◆ RM_AUDIO_PLAYBACK_PWM_Play()

fsp_err_t RM_AUDIO_PLAYBACK_PWM_Play (audio_playback_ctrl_t *const *p_api_ctrl*, void const *const *p_buffer*, uint32_t *length*)

Play a single audio buffer by input samples to the PWM HAL (AGT or GPT) at the sampling frequency configured by the timer.

- Example:

```
/* Set the 16 Bit PCM audio stream to play next */
   err = RM_AUDIO_PLAYBACK_PWM_Play(&g_audio_playback_pwm_ctrl, play_buffer,
AUDIO_EXAMPLE_LENGTH);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
```

Return values

FSP_SUCCESS	Buffer playback began successfully.
FSP_ERR_ASSERTION	The parameter p_ctrl or p_buffer is NULL or buffer length is greater than 0x10000.
FSP_ERR_NOT_OPEN	Driver not open.. This function calls • transfer_api_t::reset

◆ RM_AUDIO_PLAYBACK_PWM_Close()

fsp_err_t RM_AUDIO_PLAYBACK_PWM_Close (audio_playback_ctrl_t *const *p_api_ctrl*)

Closes the module driver. Enables module stop mode.

Return values

FSP_SUCCESS	Module successfully closed.
FSP_ERR_NOT_OPEN	Driver not open.
FSP_ERR_ASSERTION	Pointer pointing to NULL.

Note

This function will close all the lower level HAL drivers as well.

4.2.75 AWS PKCS11 PAL (rm_aws_pkcs11_pal)

Modules

PKCS#11 PAL layer implementation for use by FreeRTOS TLS.

Overview

Note

> ***The PKCS#11 PAL Interface does not provide any interfaces to the user. Consult the AWS documentation for more info: https://docs.aws.amazon.com/freertos/latest/portingguide/afr-porting-pkcs.html.***

Configuration

There is no user configuration for this module

Data Flash Usage

The current implementation utilizes 16K of Data flash of which 8K is used for storage and the other 8K is used for backup.

Usage Notes

Limitations

- Interrupts are disabled while write or erase operations are being performed.
- Credentials are stored on data flash with no tamper protection other than SHA256 for integrity.
- Credential access is not limited in any way. The credential access and tamper issues can be resolved by updating the implementation to use code flash instead of data flash and using the Secure MPU to control access to it.

4.2.76 AWS PKCS11 PAL LITTLEFS (rm_aws_pkcs11_pal_littlefs)

Modules

PKCS#11 PAL LittleFS layer implementation for use by FreeRTOS TLS.

Overview

Note

> ***The PKCS#11 PAL LittleFS Interface does not provide any interfaces to the user. Consult the AWS documentation for more info: https://docs.aws.amazon.com/freertos/latest/portingguide/afr-porting-pkcs.html.***

Configuration

There is no user configuration for this module

Usage Notes

The current implementation utilizes LittleFS Flash Port (rm_littlefs_flash) for storage.

Limitations

- Credential access is not limited in any way.

4.2.77 Bluetooth Low Energy Abstraction (rm_ble_abs)

Modules

Functions

fsp_err_t	RM_BLE_ABS_Open (ble_abs_ctrl_t *const p_ctrl, ble_abs_cfg_t const *const p_cfg)
fsp_err_t	RM_BLE_ABS_Close (ble_abs_ctrl_t *const p_ctrl) Close the BLE channel. Implements ble_abs_api_t::close. More...
fsp_err_t	RM_BLE_ABS_Reset (ble_abs_ctrl_t *const p_ctrl, ble_event_cb_t init_callback)
fsp_err_t	RM_BLE_ABS_StartLegacyAdvertising (ble_abs_ctrl_t *const p_ctrl, ble_abs_legacy_advertising_parameter_t const *const p_advertising_parameter)
fsp_err_t	RM_BLE_ABS_StartExtendedAdvertising (ble_abs_ctrl_t *const p_ctrl, ble_abs_extend_advertising_parameter_t const *const p_advertising_parameter)
fsp_err_t	RM_BLE_ABS_StartNonConnectableAdvertising (ble_abs_ctrl_t *const p_ctrl, ble_abs_non_connectable_advertising_parameter_t const *const p_advertising_parameter)
fsp_err_t	RM_BLE_ABS_StartPeriodicAdvertising (ble_abs_ctrl_t *const p_ctrl, ble_abs_periodic_advertising_parameter_t const *const p_advertising_parameter)
fsp_err_t	RM_BLE_ABS_StartScanning (ble_abs_ctrl_t *const p_ctrl, ble_abs_scan_parameter_t const *const p_scan_parameter)
fsp_err_t	RM_BLE_ABS_CreateConnection (ble_abs_ctrl_t *const p_ctrl,

	ble_abs_connection_parameter_t const *const p_connection_parameter)
fsp_err_t	RM_BLE_ABS_SetLocalPrivacy (ble_abs_ctrl_t *const p_ctrl, uint8_t const *const p_lc_irk, uint8_t privacy_mode)
fsp_err_t	RM_BLE_ABS_StartAuthentication (ble_abs_ctrl_t *const p_ctrl, uint16_t connection_handle)
fsp_err_t	RM_BLE_ABS_DeleteBondInformation (ble_abs_ctrl_t *const p_ctrl, ble_abs_bond_information_parameter_t const *const p_bond_information_parameter)
fsp_err_t	RM_BLE_ABS_ImportKeyInformation (ble_abs_ctrl_t *const p_ctrl, ble_device_address_t *p_local_identity_address, uint8_t *p_local_irk, uint8_t *p_local_csrk)
fsp_err_t	RM_BLE_ABS_ExportKeyInformation (ble_abs_ctrl_t *const p_ctrl, ble_device_address_t *p_local_identity_address, uint8_t *p_local_irk, uint8_t *p_local_csrk)

Detailed Description

Middleware for the Bluetooth peripheral on RA MCUs. This module implements the BLE ABS Interface .

Overview

This module provides BLE GAP functionality that complies with the Bluetooth Core Specification version 5.0 specified by the Bluetooth SIG. This module is configured via the QE for BLE. QE for BLE provides standard services defined by standardization organization and custom services defined by user. Bluetooth LE Profile API Document User's Manual describes the APIs for standard services.

Features

The Bluetooth Low Energy Abstraction module supports the following features:

- following GAP Role support
  - Central: The device that sends a connection request to the Peripheral device.
  - Peripheral: The device that accepts a connection request from Central and establishes a connection.
  - Observer : The device that scans for advertising.
  - Broadcaster : The device that sends advertising.
- LE 2M PHY
  - BLE communication is supported on the 2 Msym/s PHY.
- LE Coded PHY -Supports BLE communication on the Coded PHY. This enables communication over longer distances than 1M PHY and 2M PHY.
- LE Advertising Extensions
  - Up to four independent adverts can be executed simultaneously.
  - The size of Advertising Data/Scan Response Data has been expanded to a maximum of 1650 bytes.
  - Periodic Advertising is available.

- LE Channel Selection Algorithm #2
  - With the hopping channel selection algorithm added in Version 5.0, the machine that selects the channel It is possible.
- High Duty Cycle Non-Connectable Advertising
  - The ability to support non-connectable advertising with a minimum interval of up to 20 msec.
- LE Secure Connections
  - Elliptic curve Diffie-Hellman key sharing (ECDH) for pairing with passive eavesdropping support.
- Link Layer privacy
  - This feature avoids being tracked by other BLE devices by periodically changing the Bluetooth device address.
- Link Layer Extended Scanner Filter policies
  - Scan Filter support for Resolvable private addresses.
- LE Data Packet Length Extension
  - This function expands the packet size of BLE data communications. It is possible to scale up to 251 bytes.
- LE L2CAP Connection Oriented Channel Support
  - The ability to support communication using the L2CAP credit based flow control channel.
- Low Duty Cycle Directed Advertising
  - The ability to support the advertising of the Low Duty Cycle for reconnecting to a known device.
- LE Link Layer Topology
  - It supports both Master and Slave roles and can operate as Master when connected to one remote device and as Slave when connected to another remote device.
- LE Ping
  - This function checks whether the link is maintained or not by requesting the transmission of packets containing MIC after link encryption.

BLE Library Configuration

There are three types of BLE Protocol Stacks, and the functions provided are different depending on the type of BLE Protocol Stack you select.

BLE library feature	Extended	Balance	Compact
GAP Role	Central Peripheral Observer Broadcaster	Central Peripheral Observer Broadcaster	Peripheral Broadcaster
LE 2M PHY	Yes	Yes	No
LE Coded PHY	Yes	Yes	No
LE Advertising Extensions	Yes	No	No
LE Channel Selection Algorithm #2	Yes	Yes	No
High Duty Cycle Non-Connectable Advertising	Yes	Yes	Yes
LE Secure Connections	Yes	Yes	Yes
Link Layer privacy	Yes	Yes	Yes

Link Layer Extended Scanner Filter policies	Yes	Yes	No
LE Data Packet Length Extension	Yes	Yes	Yes
LE L2CAP Connection Oriented Channel Support	Yes	No	No
Low Duty Cycle Directed Advertising	Yes	Yes	Yes
LE Link Layer Topology	Yes	Yes	No
LE Ping	Yes	Yes	Yes
32-bit UUID Support in LE	Yes	Yes	Yes

Target Devices

The Bluetooth Low Energy Abstraction module supports the following devices.

- RA4W1

Configuration

Build Time Configurations for rm_ble_abs

The following build time configurations are defined in fsp_cfg/rm_ble_abs_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enable • Disable	Default (BSP)	Specify whether to include code for API parameter checking. Valid settings include.
Debug Public Address	Must be a valid device address	FF:FF:FF:50:90:74	Public Address of firmware initial value.
Debug Random Address	Must be a valid device address	FF:FF:FF:FF:FF:FF	Random Address of firmware initial value.
Maximum number of connections	Value must be an integer between 1 and 7	7	Maximum number of connections.
Maximum connection data length	Value must be an integer between 27 and 251	251	Maximum connection data length.
Maximum advertising data length	Value must be an integer between 31 and 1650	1650	Maximum advertising data length.
Maximum advertising set number	Value must be an integer between 1 and	4	Maximum advertising set number.

	4		
Maximum periodic sync set number.	Value must be an integer between 1 and 2	2	Maximum periodic sync set number.
Store Security Data	• Disable • Enable	Disable	Store Security Data in DataFlash.
Data Flash Block for Security Data	Value must be an integer between 0 and 7	0	Data Flash Block for Security Data Management.
Remote Device Bonding Number	Value must be an integer between 1 and 7	7	Number of remote device bonding information.
Connection Event Start Notify	• Disable • Enable	Disable	Set Connection event start notify enable/disable.
Connection Event Close Notify	• Disable • Enable	Disable	Set Connection event close notify enable/disable.
Advertising Event Start Notify	• Disable • Enable	Disable	Set Advertising event start notify enable/disable.
Advertising Event Close Notify	• Disable • Enable	Disable	Set Advertising event close notify enable/disable.
Scanning Event Start Notify	• Disable • Enable	Disable	Set Scanning event start notify enable/disable.
Scanning Event Close Notify	• Disable • Enable	Disable	Set Scanning event close notify enable/disable.
Initiating Event Start Notify	• Disable • Enable	Disable	Set Initiating event start notify enable/disable.
Initiating Event Close Notify	• Disable • Enable	Disable	Set Initiating event close notify enable/disable.
RF Deep Sleep Start Notify	• Disable • Enable	Disable	Set RF_DEEP_SLEEP start notify enable/disable.
RF Deep Sleep Wakeup Notify	• Disable • Enable	Disable	Set RF_DEEP_SLEEP wakeup notify enable/disable.
Bluetooth dedicated clock	Value must be an integer between 0 and 15	6	Load capacitance adjustment.

DC-DC converter	• Disable • Enable	Disable	Set DC-DC converter for RF part.
Slow Clock Source	• Use RF_LOCO • Use External 32.768kHz	Use RF_LOCO	Set slow clock source for RF part.
MCU CLKOUT Port	• P109 • P205	P109	When BLE_ABS_CFG_RF_EXTERNAL_32K_ENABLE = 1, Set port of MCU CLKOUT.
MCU CLKOUT Frequency Output	• MCU CLKOUT frequency 32.768kHz • MCU CLKOUT frequency 16.384kHz	MCU CLKOUT frequency 32.768kHz	When BLE_ABS_CFG_RF_EXTERNAL_32K_ENABLE = 1, set frequency output from CLKOUT of MCU part.
Sleep Clock Accuracy(SCA)	Value must be an integer between 0 and 500	250	When BLE_ABS_CFG_RF_EXTERNAL_32K_ENABLE = 1, set Sleep Clock Accuracy(SCA) for RF slow clock.
Transmission Power Maximum Value	• max +0dBm • max +4dBm	max +4dBm	Set transmission power maximum value.
Transmission Power Default Value	• High 0dBm(Transmission Power Maximum Value = +0dBm) / +4dBm(Transmission Power Maximum Value = +4dBm) • Mid 0dBm(Transmission Power Maximum Value = +0dBm) / 0dBm(Transmission Power Maximum Value = +4dBm) • Low -18dBm(Transmission Power Maximum Value = +0dBm) / -20dBm(Transmission Power Maximum Value = +4dBm)	High 0dBm(Transmission Power Maximum Value = +0dBm) / +4dBm(Transmission Power Maximum Value = +4dBm)	Set default transmit power. Default transmit power is dependent on the configuration of Maximum transmission power(BLE_ABS_CFG_RF_DEF_TX_POW).
CLKOUT_RF Output	• No output • 4MHz output • 2MHz output	No output	Set CLKOUT_RF output setting.

	• 1MHz output		
RF_DEEP_SLEEP Transition	• Disable • Enable	Enable	Set RF_DEEP_SLEEP transition.
MCU Main Clock Frequency	Value must be an integer between 1000 and 20000	8000	Set MCU Main Clock Frequency (kHz). Set clock source according to your board environment. HOCO: don't care. / Main Clock: 1000 to 20000 kHz / PLL Circuit: 4000 to 12500 kHz
Code Flash(ROM) Device Data Block	Value must be an integer between -1 and 255	255	Device specific data block on Code Flash (ROM).
Device Specific Data Flash Block	Value must be an integer between -1 and 7	-1	Device specific data block on E2 Data Flash.
MTU Size Configured	Value must be an integer between 23 and 247	247	MTU Size configured by GATT MTU exchange procedure.
Timer Slot Maximum Number	Value must be an integer between 1 and 10	10	The maximum number of timer slot.

Configurations for Networking > BLE Abstraction (rm_ble_abs)

This module can be added to the Stacks tab via New Stack > Networking > BLE Abstraction (rm_ble_abs).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_ble_abs0	Module name.
General > Gap callback	Name must be a valid C symbol	gap_cb	A user callback function must be provided if the BLE_ABS is configured to generate a GAP. If QE is used, set to NULL.
General > Vendor specific callback	Name must be a valid C symbol	vs_cb	A user callback function must be provided if the BLE_ABS is configured to generate a Vendor Specific. If QE is used, set to NULL.
General > GATT server	Name must be a valid	gs_abs_gatts_cb_param	Set GATT server

callback parameter	C symbol		callback parameter. If QE is used, set to NULL.
General > GATT server callback number	Must be a valid number	2	The number of GATT Server callback functions.
General > GATT client callback parameter	Name must be a valid C symbol	gs_abs_gattc_cb_param	Set GATT client callback parameter. If QE is used, set to NULL.
General > GATT client callback number	Must be a valid number	2	The number of GATT Server callback functions.
Security > Pairing parameters	Name must be a valid C symbol	gs_abs_pairing_param	Set pairing parameters.
Security > IO capabilities of local device.	• BLE_GAP_IOCAP_DISPLAY_ONLY • BLE_GAP_IOCAP_DISPLAY_YESNO • BLE_GAP_IOCAP_KEYBOARD_ONLY • BLE_GAP_IOCAP_NOINPUT_NOOUTPUT • BLE_GAP_IOCAP_KEYBOARD_DISPLAY	BLE_GAP_IOCAP_NOINPUT_NOOUTPUT	Select IO capabilities of local device.
Security > MITM protection policy.	• BLE_GAP_SEC_MITM_BEST_EFFORT • BLE_GAP_SEC_MITM_STRICT	BLE_GAP_SEC_MITM_BEST_EFFORT	Select MITM protection policy.
Security > Determine whether to accept only Secure Connections or not.	• BLE_GAP_SC_BEST_EFFORT • BLE_GAP_SC_STRICT	BLE_GAP_SC_BEST_EFFORT	Select determine whether to accept only Secure Connections or not.
Security > Type of keys to be distributed from local device.	• BLE_GAP_KEY_DIST_ENCKEY • BLE_GAP_KEY_DIST_IDKEY • BLE_GAP_KEY_DIST_SIGNKEY		Select type of keys to be distributed from local device.
Security > Type of keys which local device requests a remote device to distribute.	• BLE_GAP_KEY_DIST_ENCKEY • BLE_GAP_KEY_DIST_IDKEY • BLE_GAP_KEY_D		Set type of keys which local device requests a remote device to distribute.

	IST_SIGNKEY		
Security > Maximum LTK size.	Valid range is 7 - 16	16	Set Maximum LTK size.
Interrupts > Callback provided when an ISR occurs	Name must be a valid C symbol	NULL	Callback provided when BLE ABS ISR occurs

Clock Configuration

Note

System clock (ICLK): 8 MHz or more
Peripheral module clock A (PCLKA): 8MHz or more
The BLE Protocol Stack is optimized for ICLK and PCLKA frequencies of 32 MHz.
It is recommended that the clock be set so that the ICLK and PCLKA frequencies are 32MHz in order to get the best performance from the BLE.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

Developers should be aware of the following limitations when using the BLE_ABS:

Examples

BLE_ABS Basic Example

This is a basic example of minimal use of the BLE_ABS in an application.

```
#define BLE_ABS_EVENT_FLAG_STACK_ON (0x01 << 0)
#define BLE_ABS_EVENT_FLAG_CONN_IND (0x01 << 1)
#define BLE_ABS_EVENT_FLAG_ADV_ON (0x01 << 2)
#define BLE_ABS_EVENT_FLAG_ADV_OFF (0x01 << 3)
#define BLE_ABS_EVENT_FLAG_DISCONN_IND (0x01 << 4)
#define BLE_ABS_EVENT_FLAG_RSLV_LIST_CONF_COMP (0x01 << 5)
#define BLE_ABS_EXAMPLE_SHORTENED_LOCAL_NAME 'E', 'x', 'a', 'm', 'p', 'l', 'e'
#define BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME 'T', 'E', 'S', 'T', '_', 'E', 'x', 'a',
'm', 'p', 'l', 'e'
#define BLE_ABS_EXAMPLE_SLOW_ADVERTISING_INTERVAL (0x00000640)
void ble_abs_peripheral_example (void)
{
 fsp_err_t          err    = FSP_SUCCESS;
```

```
volatile uint32_t timeout = UINT16_MAX * UINT8_MAX * 8;
ble_device_address_t local_identity_address;
   uint8_t             local_irk[BLE_GAP_IRK_SIZE];
   uint8_t             local_csrk[BLE_GAP_CSRK_SIZE];
   uint8_t           * p_local_irk = NULL;
   uint8_t             privacy_mode = BLE_GAP_NET_PRIV_MODE;
   uint8_t advertising_data[] =
   {
/* Flags */
      0x02,
      0x01,
      (0x1a),
/* Shortened Local Name */
      0x08,
      0x08,
      BLE_ABS_EXAMPLE_SHORTENED_LOCAL_NAME,
   };
/* Scan Response Data */
   uint8_t scan_response_data[] =
   {
/* Complete Local Name */
      0x0D,
      0x09,
      BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME,
   };
ble_abs_legacy_advertising_parameter_t legacy_advertising_parameter =
   {
      .p_peer_address             =
NULL,
      .slow_advertising_interval =
BLE_ABS_EXAMPLE_SLOW_ADVERTISING_INTERVAL,
      .slow_advertising_period    =
0x0000,
      .p_advertising_data         =
```

```
advertising_data,
       .advertising_data_length    = sizeof
(advertising_data),
       .p_scan_response_data       =
scan_response_data,
       .scan_response_data_length = sizeof
(scan_response_data),
       .advertising_filter_policy = BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY
,
       .advertising_channel_map    = (BLE_GAP_ADV_CH_37 | BLE_GAP_ADV_CH_38 |
BLE_GAP_ADV_CH_39),
       .own_bluetooth_address_type = BLE_GAP_ADDR_PUBLIC
,
       .own_bluetooth_address      = {0},
    };
    g_ble_event_flag = 0;
 /* Open the module. */
    err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait BLE_GAP_EVENT_STACK_ON event is notified. */
 while (!(BLE_ABS_EVENT_FLAG_STACK_ON & g_ble_event_flag) && (--timeout > 0U))
    {
 R_BLE_Execute();
    }
 /* Set local privacy. */
    err = RM_BLE_ABS_SetLocalPrivacy(&g_ble_abs0_ctrl, p_local_irk, privacy_mode);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait BLE_GAP_EVENT_RSLV_LIST_CONF_COMP event is notified. */
 while (!(BLE_ABS_EVENT_FLAG_RSLV_LIST_CONF_COMP & g_ble_event_flag) && (--timeout >
0U))
    {
 R_BLE_Execute();
```

```
    }
    time_out_handle_error(timeout);
    g_ble_event_flag = 0;
    timeout = UINT16_MAX * UINT8_MAX * 8;
 /* Start advertising. */
    err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (!(BLE_ABS_EVENT_FLAG_CONN_IND & g_ble_event_flag) && (--timeout > 0U))
    {
 if (BLE_ABS_EVENT_FLAG_ADV_OFF & g_ble_event_flag)
      {
 /* Restart advertise, when stop advertising. */
      err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
 if (FSP_SUCCESS == err)
      {
                g_ble_event_flag &= (uint16_t) ~BLE_ABS_EVENT_FLAG_ADV_OFF;
      }
 else if (FSP_ERR_INVALID_STATE == err)
      {
 /* BLE driver state is busy. */
                ;
      }
 else
      {
 /* Handle any errors. This function should be defined by the user. */
      assert(FSP_SUCCESS == err);
      }
      }
 else if ((timeout % BLE_ABS_RETRY_INTERVAL) == 0U)
      {
 /* Stop advertising after a certain amount of time */
```

```
 R_BLE_GAP_StopAdv(g_advertising_handle);
      }
 else
      {
            ;
      }
 R_BLE_Execute();
    }
    time_out_handle_error(timeout);
 /* Export local key information. */
    err = RM_BLE_ABS_ExportKeyInformation(&g_ble_abs0_ctrl, &local_identity_address,
local_irk, local_csrk);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Clean up & Close BLE driver */
    g_ble_event_flag = 0;
 /* Close BLE driver */
    err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
```

```
#define BLE_ABS_EVENT_FLAG_STACK_ON (0x01 << 0)
#define BLE_ABS_EVENT_FLAG_CONN_IND (0x01 << 1)
#define BLE_ABS_EVENT_FLAG_ADV_REPT_IND (0x01 << 2)
#define BLE_ABS_EVENT_FLAG_ADV_OFF (0x01 << 3)
#define BLE_ABS_EVENT_FLAG_PAIRING_COMP (0x01 << 4)
#define BLE_ABS_EVENT_FLAG_SCAN_TIMEOUT (0x01 << 5)
#define BLE_ABS_EVENT_FLAG_DELETE_BOND_COMP (0x01 << 6)
#define BLE_ABS_EXAMPLE_FAST_SCAN_INTERVAL (0x0060)
#define BLE_ABS_EXAMPLE_FAST_SCAN_WINDOW (0x0030)
#define BLE_ABS_EXAMPLE_SLOW_SCAN_INTERVAL (0x0800)
#define BLE_ABS_EXAMPLE_SLOW_SCAN_WINDOW (0x0012)
#define BLE_ABS_EXAMPLE_FAST_SCAN_PERIOD (0x0BB8)
```

```
#define BLE_ABS_EXAMPLE_SLOW_SCAN_PERIOD (0x0000)
#define BLE_ABS_EXAMPLE_CONNECTION_INTERVAL (0x0028)
#define BLE_ABS_EXAMPLE_SUPERVISION_TIMEOUT (0x0200)
#define BLE_ABS_EXAMPLE_DEVICE_ADDRESS 0x88, 0x88, 0x88, 0x88, 0x88, 0x88
#define BLE_ABS_EXAMPLE_IRK 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5,
0xA5
#define BLE_ABS_EXAMPLE_CSRK 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5, 0xA5,
0xA5
#define BLE_ABS_SCAN_FILTER_DATA_LENGTH (12)
/* Scan filter data (data type: Complete Local Name ) */
static uint8_t g_filter_data[] =
{
    BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME
};
/* Connection phy parameters */
ble_abs_connection_phy_parameter_t g_connection_phy_parameter =
{
    .connection_interval      = BLE_ABS_EXAMPLE_CONNECTION_INTERVAL, /* 50.0(ms) */
    .supervision_timeout      = BLE_ABS_EXAMPLE_SUPERVISION_TIMEOUT, /* 5,120(ms) */
    .connection_slave_latency = 0x0000,
};
/* Connection device address */
ble_device_address_t g_connection_device_address;
/* Connection parameters */
ble_abs_connection_parameter_t g_connection_parameter =
{
    .p_connection_phy_parameter_1M = &g_connection_phy_parameter,
    .p_device_address             = &g_connection_device_address,
    .filter_parameter             = BLE_GAP_INIT_FILT_USE_ADDR,
    .connection_timeout            = 0x05, /* 5(s) */
};
ble_abs_bond_information_parameter_t g_bond_information_parameter =
{
    .local_bond_information   = BLE_ABS_LOCAL_BOND_INFORMATION_ALL,
```

```
    .remote_bond_information = BLE_ABS_REMOTE_BOND_INFORMATION_ALL,
    .delete_non_volatile_area = BLE_ABS_DELETE_NON_VOLATILE_AREA_ENABLE,
    .p_address                = NULL,
    .abs_delete_bond_callback = delete_bond_cb,
};
void ble_abs_central_example (void)
{
 fsp_err_t         err    = FSP_SUCCESS;
 volatile uint32_t timeout = UINT16_MAX * UINT8_MAX * 8;
    g_connection_handle = BLE_GAP_INVALID_CONN_HDL;
 ble_device_address_t local_identity_address =
    {
       .addr = {BLE_ABS_EXAMPLE_DEVICE_ADDRESS},
       .type = BLE_GAP_ADDR_PUBLIC
    };
    uint8_t local_irk[BLE_GAP_IRK_SIZE]   = {BLE_ABS_EXAMPLE_IRK};
    uint8_t local_csrk[BLE_GAP_CSRK_SIZE] = {BLE_ABS_EXAMPLE_CSRK};
 static ble_abs_scan_phy_parameter_t scan_phy_parameter =
    {
       .fast_scan_interval = BLE_ABS_EXAMPLE_FAST_SCAN_INTERVAL, /* 60.0(ms) */
       .fast_scan_window   = BLE_ABS_EXAMPLE_FAST_SCAN_WINDOW,   /* 30.0(ms) */
       .slow_scan_interval = BLE_ABS_EXAMPLE_SLOW_SCAN_INTERVAL, /* 1,280.0(ms) */
       .slow_scan_window   = BLE_ABS_EXAMPLE_SLOW_SCAN_WINDOW,   /* 11.25(ms) */
       .scan_type          = BLE_GAP_SCAN_ACTIVE
    };
 /* Scan parameters */
 ble_abs_scan_parameter_t scan_parameter =
    {
       .p_phy_parameter_1M       = &scan_phy_parameter,
       .fast_scan_period          = BLE_ABS_EXAMPLE_FAST_SCAN_PERIOD, /* 30,000(ms)
*/
       .slow_scan_period          = BLE_ABS_EXAMPLE_SLOW_SCAN_PERIOD,
       .p_filter_data             = g_filter_data,
       .filter_data_length       = (uint16_t) BLE_ABS_SCAN_FILTER_DATA_LENGTH,
```

```
        .filter_ad_type             = 0x09,                            /* Data type:
Complete Local Name */
        .device_scan_filter_policy = BLE_GAP_SCAN_ALLOW_ADV_ALL,
        .filter_duplicate           = BLE_GAP_SCAN_FILT_DUPLIC_ENABLE,
    };
    g_ble_event_flag = 0;
 /* Open the module. */
    err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Connection parameters */
 while (!(BLE_ABS_EVENT_FLAG_STACK_ON & g_ble_event_flag) && (--timeout > 0U))
    {
 R_BLE_Execute();
    }
 /* Import local key information. */
    err = RM_BLE_ABS_ImportKeyInformation(&g_ble_abs0_ctrl, &local_identity_address,
local_irk, local_csrk);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Start scanning. */
    err = RM_BLE_ABS_StartScanning(&g_ble_abs0_ctrl, &scan_parameter);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while ((BLE_ABS_EVENT_FLAG_ADV_REPT_IND & g_ble_event_flag) && (--timeout > 0U))
    {
 if ((BLE_ABS_EVENT_FLAG_SCAN_TIMEOUT & g_ble_event_flag) || (BLE_GAP_EVENT_SCAN_OFF
& g_ble_event_flag))
      {
            g_ble_event_flag &= (uint16_t) ~BLE_ABS_EVENT_FLAG_ADV_OFF;
            g_ble_event_flag &= (uint16_t) ~BLE_ABS_EVENT_FLAG_SCAN_TIMEOUT;
 /* Start scanning. */
      err = RM_BLE_ABS_StartScanning(&g_ble_abs0_ctrl, &scan_parameter);
 /* Handle any errors. This function should be defined by the user. */
```

```
     assert(FSP_SUCCESS == err);
     }
else if ((timeout % BLE_ABS_RETRY_INTERVAL) == 0U)
     {
/* Stop scanning after a certain amount of time */
R_BLE_GAP_StopScan();
     }
else
     {
           ;
     }
R_BLE_Execute();
   }
   g_ble_event_flag = 0;
   time_out_handle_error(timeout);
   timeout = UINT16_MAX * UINT8_MAX * 8;
/* Create connection with remote device. */
   err = RM_BLE_ABS_CreateConnection(&g_ble_abs0_ctrl, &g_connection_parameter);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Wait BLE_GAP_EVENT_CONN_IND event is notified. */
while (!(BLE_ABS_EVENT_FLAG_CONN_IND & g_ble_event_flag) && (--timeout > 0U))
   {
R_BLE_Execute();
   }
   time_out_handle_error(timeout);
   g_ble_event_flag = 0;
   timeout = UINT16_MAX * UINT8_MAX * 8;
/* Start authentication with remote device. */
   err = RM_BLE_ABS_StartAuthentication(&g_ble_abs0_ctrl, g_connection_handle);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
/* Wait BLE_GAP_EVENT_PAIRING_COMP event is notified. */
while (!(BLE_ABS_EVENT_FLAG_PAIRING_COMP & g_ble_event_flag) && (--timeout > 0U))
```

```
    {
 R_BLE_Execute();
    }
    time_out_handle_error(timeout);
    g_ble_event_flag = 0;
    timeout = UINT16_MAX * UINT8_MAX * 8;
 /* Delete bonding information. */
    err = RM_BLE_ABS_DeleteBondInformation(&g_ble_abs0_ctrl,
&g_bond_information_parameter);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait delete_bond_cb application callback function is called. */
 while (!(BLE_ABS_EVENT_FLAG_DELETE_BOND_COMP & g_ble_event_flag) && (--timeout >
0U))
    {
 R_BLE_Execute();
    }
    time_out_handle_error(timeout);
 /* Clean up & Close BLE driver */
    g_ble_event_flag = 0;
    err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
}
void delete_bond_cb (st_ble_dev_addr_t * p_addr) {
    (void) p_addr;
    g_ble_event_flag = g_ble_event_flag | BLE_ABS_EVENT_FLAG_DELETE_BOND_COMP;
}
```

Data Structures

struct	abs_advertising_parameter_t
struct	abs_scan_parameter_t
struct	ble_abs_instance_ctrl_t

struct	st_ble_rf_notify_t This structure is RF event notify management. More...

Typedefs

typedef void(*	ble_abs_timer_cb_t) (uint32_t timer_hdl)
typedef void(*	ble_mcu_clock_change_cb_t) (void) ble_mcu_clock_change_cb_t is the callback function type to use CLKOUT_RF as the MCU main clock source. More...
typedef void(*	ble_rf_notify_cb_t) (uint32_t) ble_rf_notify_cb_t is the RF event notify callback function type. More...

Enumerations

enum	e_ble_timer_type_t

Data Structure Documentation

◆ abs_advertising_parameter_t

struct abs_advertising_parameter_t

advertising set parameters structure

Data Fields		
union abs_advertising_parameter_t	advertising_parameter	Advertising parameters.
uint32_t	advertising_status	Advertising status.
ble_device_address_t	remote_device_address	Remote device address for direct advertising.

◆ abs_scan_parameter_t

struct abs_scan_parameter_t

scan parameters structure

Data Fields		
ble_abs_scan_parameter_t	scan_parameter	Scan parameters.
ble_abs_scan_phy_parameter_t	scan_phy_parameter_1M	1M phy parameters for scan.
ble_abs_scan_phy_parameter_t	scan_phy_parameter_coded	Coded phy parameters for scan. */.
uint32_t	scan_status	

◆ ble_abs_instance_ctrl_t

struct ble_abs_instance_ctrl_t		
BLE ABS private control block. DO NOT MODIFY. Initialization occurs when RM_BLE_ABS_Open() is called.		
Data Fields		
uint32_t	open	Indicates whether the open() API has been successfully called.
void const *	p_context	Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t.
ble_gap_application_callback_t	abs_gap_callback	GAP callback function.
ble_vendor_specific_application_callback_t	abs_vendor_specific_callback	Vendor specific callback function.
ble_abs_delete_bond_application_callback_t	abs_delete_bond_callback	Delete bond information callback function.
uint32_t	connection_timer_handle	Cancel a request for connection timer.
uint32_t	advertising_timer_handle	Advertising timer for legacy advertising.
uint32_t	scan_timer_handle	Scan interval timer.
abs_advertising_parameter_t	advertising_sets[BLE_MAX_NO_OF_ADV_SETS_SUPPORTED]	Advertising set information.
abs_scan_parameter_t	abs_scan	Scan information.
st_ble_dev_addr_t	loc_bd_addr	Local device address.
uint8_t	privacy_mode	Privacy mode.
uint32_t	set_privacy_status	Local privacy status.
ble_abs_timer_t	timer[BLE_ABS_CFG_TIMER_NUMBER_OF_SLOT]	
uint8_t	local_irk[BLE_GAP_IRK_SIZE]	
ble_abs_identity_address_info_t	identity_address_info	
uint32_t	current_timeout_ms	Current timeout.
uint32_t	elapsed_timeout_ms	Elapsed timeout.
ble_abs_cfg_t const *	p_cfg	Pointer to the BLE ABS configuration block.

◆ st_ble_rf_notify_t

struct st_ble_rf_notify_t

This structure is RF event notify management.

Data Fields		
uint32_t	enable	Set enable/disable of each RF event notification. Bit0 Notify Connection event start(0:Disable/1:Enable) Bit1 Notify Advertising event start(0:Disable/1:Enable) Bit2 Notify Scanning event start(0:Disable/1:Enable) Bit3 Notify Initiating event start(0:Disable/1:Enable) Bit4 Notify Connection event close(0:Disable/1:Enable) Bit5 Notify Advertising event close(0:Disable/1:Enable) Bit6 Notify Scanning event close(0:Disable/1:Enable) Bit7 Notify Initiating event close(0:Disable/1:Enable) Bit8 Notify RF_DEEP_SLEEP event start(0:Disable/1:Enable) Bit9 Notify RF_DEEP_SLEEP event close(0:Disable/1:Enable) Other Bit: Reserved for future use.
ble_rf_notify_cb_t	start_cb	Set callback function pointer for RF event start.
ble_rf_notify_cb_t	close_cb	Set callback function pointer for RF event close.
ble_rf_notify_cb_t	dsleep_cb	Set callback function pointer for RF_DEEP_SLEEP.

Typedef Documentation

◆ ble_abs_timer_cb_t

typedef void(* ble_abs_timer_cb_t) (uint32_t timer_hdl)
The timer callback invoked when the timer expired.

◆ ble_mcu_clock_change_cb_t

ble_mcu_clock_change_cb_t

ble_mcu_clock_change_cb_t is the callback function type to use CLKOUT_RF as the MCU main clock source.

Parameters

none	

Returns
none

◆ ble_rf_notify_cb_t

ble_rf_notify_cb_t

ble_rf_notify_cb_t is the RF event notify callback function type.

Parameters

[in]	uint32_t	The infomation of RF event notification.

Returns
none

Enumeration Type Documentation

◆ e_ble_timer_type_t

enum e_ble_timer_type_t

The timer type.

Enumerator	
BLE_TIMER_ONE_SHOT	One shot timer type
BLE_TIMER_PERIODIC	Periodic timer type

Function Documentation

◆ RM_BLE_ABS_Open()

fsp_err_t RM_BLE_ABS_Open (ble_abs_ctrl_t *const *p_ctrl*, ble_abs_cfg_t const *const *p_cfg*)

Host stack is initialized with this function. Before using All the R_BLE APIs, it's necessary to call this function. A callback functions are registered with this function. In order to receive the GAP, GATT, Vendor specific event, it's necessary to register a callback function. The result of this API call is notified in BLE_GAP_EVENT_STACK_ON event. Implements ble_abs_api_t::open.

Example:

```
/* Open the module. */
   err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
```

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ASSERTION	Null pointer presented.
FSP_ERR_INVALID_CHANNEL	The channel number is invalid.
FSP_ERR_ALREADY_OPEN	Requested channel is already open in a different configuration.
FSP_ERR_INVALID_ARGUMENT	Invalid input parameter.

Host stack is initialized with this function. Before using All the R_BLE APIs, it's necessary to call this function. A callback functions are registered with this function. In order to receive the GAP, GATT, Vendor specific event, it's necessary to register a callback function. The result of this API call is notified in BLE_GAP_EVENT_STACK_ON event. Implements ble_abs_api_t::open.

Example:

```
/* Open the module. */
   err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
```

Return values

FSP_SUCCESS	Channel opened successfully.
FSP_ERR_ASSERTION	Null pointer presented.
FSP_ERR_ALREADY_OPEN	Requested channel is already open in a different configuration.
FSP_ERR_INVALID_ARGUMENT	Invalid input parameter.
FSP_ERR_INVALID_MODE	Invalid mode during open call

◆ RM_BLE_ABS_Close()

fsp_err_t RM_BLE_ABS_Close (ble_abs_ctrl_t *const *p_ctrl*)

Close the BLE channel. Implements ble_abs_api_t::close.

Example:

```
/* Close BLE driver */
   err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
```

Return values

FSP_SUCCESS	Channel closed successfully.
FSP_ERR_ASSERTION	Null pointer presented.
FSP_ERR_NOT_OPEN	Control block not open.

Example:

```
/* Close BLE driver */
   err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
```

Return values

FSP_SUCCESS	Channel closed successfully.
FSP_ERR_ASSERTION	Null pointer presented.
FSP_ERR_NOT_OPEN	Control block not open.

◆ RM_BLE_ABS_Reset()

fsp_err_t RM_BLE_ABS_Reset (ble_abs_ctrl_t *const *p_ctrl*, ble_event_cb_t *init_callback*)

BLE is reset with this function. The process is carried out in the following order. R_BLE_Close() -> R_BLE_GAP_Terminate() -> R_BLE_Open() -> R_BLE_SetEvent(). The init_cb callback initializes the others (Host Stack, timer, etc...). Implements ble_abs_api_t::reset.

Return values

FSP_SUCCESS	Channel closed successfully.
FSP_ERR_ASSERTION	Null pointer presented.
FSP_ERR_NOT_OPEN	Control block not open.

This function is not implemented. To perform this function call R_BLE_Close followed by R_BLE_Open. Implements ble_abs_api_t::reset.

Return values

FSP_ERR_UNSUPPORTED	Function is not supported

◆ RM_BLE_ABS_StartLegacyAdvertising()

fsp_err_t RM_BLE_ABS_StartLegacyAdvertising (ble_abs_ctrl_t *const *p_ctrl*, ble_abs_legacy_advertising_parameter_t const *const *p_advertising_parameter*)

Start Legacy Advertising after setting advertising parameters, advertising data and scan response data. The legacy advertising uses the advertising set whose advertising handle is 0. The advertising type is connectable and scannable(ADV_IND). The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Implements ble_abs_api_t::startLegacyAdvertising

Example:

```
/* Start advertising. */
    err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
```

Return values

FSP_SUCCESS	Operation succeeded
FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_STATE	Host stack hasn't been initialized.
FSP_ERR_INVALID_POINTER	p_advertising_parameter is specified as NULL.

FSP_ERR_INVALID_ARGUMENT	The advertising parameter is out of range.

Start Legacy Advertising after setting advertising parameters, advertising data and scan response data. The legacy advertising uses the advertising set whose advertising handle is 0. The advertising type is connectable and scannable(ADV_IND). The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Implements ble_abs_api_t::startLegacyAdvertising

Example:

```
/* Start advertising. */
    err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
```

Return values

FSP_SUCCESS	Operation succeeded
FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_STATE	Host stack hasn't been initialized.
FSP_ERR_INVALID_POINTER	p_advertising_parameter is specified as NULL.
FSP_ERR_INVALID_ARGUMENT	The advertising parameter is out of range.

◆ RM_BLE_ABS_StartExtendedAdvertising()

fsp_err_t RM_BLE_ABS_StartExtendedAdvertising (ble_abs_ctrl_t *const *p_ctrl*, ble_abs_extend_advertising_parameter_t const *const *p_advertising_parameter*)

Start Extended Advertising after setting advertising parameters, advertising data. The extended advertising uses the advertising set whose advertising handle is 1. The advertising type is connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Implements ble_abs_api_t::startExtendedAdvertising

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_POINTER	p_advertising_parameter is specified as NULL.
FSP_ERR_INVALID_STATE	Host stack hasn't been initialized.
FSP_ERR_INVALID_ARGUMENT	The advertising parameter is out of range.
FSP_ERR_UNSUPPORTED	Subordinate modules do not support this feature.

Start Extended Advertising after setting advertising parameters, advertising data. The extended advertising uses the advertising set whose advertising handle is 1. The advertising type is connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Implements ble_abs_api_t::startExtendedAdvertising

Return values

FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_POINTER	p_advertising_parameter is specified as NULL.
FSP_ERR_UNSUPPORTED	Subordinate modules do not support this feature.

◆ RM_BLE_ABS_StartNonConnectableAdvertising()

fsp_err_t RM_BLE_ABS_StartNonConnectableAdvertising (ble_abs_ctrl_t *const *p_ctrl*, ble_abs_non_connectable_advertising_parameter_t const *const *p_advertising_parameter*)

Start Non-Connectable Advertising after setting advertising parameters, advertising data. The non-connectable advertising uses the advertising set whose advertising handle is 2. The advertising type is non-connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Secondary Advertising Max Skip is 0. Implements ble_abs_api_t::startNonConnectableAdvertising.

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_POINTER	p_advertising_parameter is specified as NULL.
FSP_ERR_INVALID_STATE	Host stack hasn't been initialized.
FSP_ERR_INVALID_ARGUMENT	The advertising parameter is out of range.

Start Non-Connectable Advertising after setting advertising parameters, advertising data. The non-connectable advertising uses the advertising set whose advertising handle is 2. The advertising type is non-connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Secondary Advertising Max Skip is 0. Implements ble_abs_api_t::startNonConnectableAdvertising.

Return values

FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_POINTER	p_advertising_parameter is specified as NULL.
FSP_ERR_UNSUPPORTED	Feature not yet supported.

◆ RM_BLE_ABS_StartPeriodicAdvertising()

fsp_err_t RM_BLE_ABS_StartPeriodicAdvertising (ble_abs_ctrl_t *const *p_ctrl*, ble_abs_periodic_advertising_parameter_t const *const *p_advertising_parameter*)

Start Periodic Advertising after setting advertising parameters, periodic advertising parameters, advertising data and periodic advertising data. The periodic advertising uses the advertising set whose advertising handle is 3. The advertising type is non-connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Secondary Advertising Max Skip is 0. Implements ble_abs_api_t::startPeriodicAdvertising

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_POINTER	p_advertising_parameter is specified as NULL.
FSP_ERR_INVALID_ARGUMENT	The advertising parameter is out of range.
FSP_ERR_UNSUPPORTED	This feature is not supported in this configuration.

Start Periodic Advertising after setting advertising parameters, periodic advertising parameters, advertising data and periodic advertising data. The periodic advertising uses the advertising set whose advertising handle is 3. The advertising type is non-connectable and non-scannable. The address type of local device is Public Identity Address or RPA(If the resolving list contains no matching entry, use the public address.). Scan request event(BLE_GAP_EVENT_SCAN_REQ_RECV) is not notified. Secondary Advertising Max Skip is 0. Implements ble_abs_api_t::startPeriodicAdvertising

Return values

FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_POINTER	p_advertising_parameter is specified as NULL.
FSP_ERR_UNSUPPORTED	Subordinate modules do not support this feature.

◆ RM_BLE_ABS_StartScanning()

fsp_err_t RM_BLE_ABS_StartScanning (ble_abs_ctrl_t *const *p_ctrl*, ble_abs_scan_parameter_t const *const *p_scan_parameter*)

Start scanning after setting scan parameters. The scanner address type is Public Identity Address. Fast scan is followed by slow scan. The end of fast scan or slow scan is notified with BLE_GAP_EVENT_SCAN_TO event. If fast_period is 0, only slow scan is carried out. If scan_period is 0, slow scan continues. Implements ble_abs_api_t::startScanning.

Example:

```
/* Start scanning. */
    err = RM_BLE_ABS_StartScanning(&g_ble_abs0_ctrl, &scan_parameter);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_POINTER	p_scan_parameter is specified as NULL.
FSP_ERR_INVALID_ARGUMENT	The scan parameter is out of range.
FSP_ERR_IN_USE	This API is called in scanning.
FSP_ERR_BLE_ABS_NOT_FOUND	Usable timer slot not found.
FSP_ERR_UNSUPPORTED	This feature is not supported in this configuration.

Start scanning after setting scan parameters. The scanner address type is Public Identity Address. Fast scan is followed by slow scan. The end of fast scan or slow scan is notified with BLE_GAP_EVENT_SCAN_TO event. If fast_period is 0, only slow scan is carried out. If scan_period is 0, slow scan continues. Implements ble_abs_api_t::startScanning.

Example:

```
/* Start scanning. */
    err = RM_BLE_ABS_StartScanning(&g_ble_abs0_ctrl, &scan_parameter);
```

Return values

FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_POINTER	p_scan_parameter is specified as NULL.
FSP_ERR_UNSUPPORTED	Function is not supported

◆ RM_BLE_ABS_CreateConnection()

fsp_err_t RM_BLE_ABS_CreateConnection (ble_abs_ctrl_t *const *p_ctrl*, ble_abs_connection_parameter_t const *const *p_connection_parameter*)

Request create connection. The initiator address type is Public Identity Address. The scan interval is 60ms and the scan window is 30ms in case of 1M PHY or 2M PHY. The scan interval is 180ms and the scan window is 90ms in case of coded PHY. The Minimum CE Length and the Maximum CE Length are 0xFFFF. When the request for a connection has been received by the Controller, BLE_GAP_EVENT_CREATE_CONN_COMP event is notified. When a link has beens established, BLE_GAP_EVENT_CONN_IND event is notified. Implements ble_abs_api_t::createConnection.

Example:

```
/* Create connection with remote device. */
   err = RM_BLE_ABS_CreateConnection(&g_ble_abs0_ctrl, &g_connection_parameter);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_POINTER	p_connection_parameter is specified as NULL.
FSP_ERR_INVALID_ARGUMENT	The create connection parameter is out of range.
FSP_ERR_IN_USE	This API is called while creating a link by previous API call.
FSP_ERR_BLE_ABS_NOT_FOUND	Couldn't find a valid timer.
FSP_ERR_UNSUPPORTED	This feature is not supported in this configuration.

Request create connection. The initiator address type is Public Identity Address. The scan interval is 60ms and the scan window is 30ms in case of 1M PHY or 2M PHY. The scan interval is 180ms and the scan window is 90ms in case of coded PHY. The Minimum CE Length and the Maximum CE Length are 0xFFFF. When the request for a connection has been received by the Controller, BLE_GAP_EVENT_CREATE_CONN_COMP event is notified. When a link has beens established, BLE_GAP_EVENT_CONN_IND event is notified. Implements ble_abs_api_t::createConnection.

Example:

```
/* Create connection with remote device. */
   err = RM_BLE_ABS_CreateConnection(&g_ble_abs0_ctrl, &g_connection_parameter);
```

Return values

FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_POINTER	p_connection_parameter is specified as

	NULL.
FSP_ERR_INVALID_ARGUMENT	The create connection parameter is out of range.
FSP_ERR_UNSUPPORTED	Function is not supported

◆ RM_BLE_ABS_SetLocalPrivacy()

fsp_err_t RM_BLE_ABS_SetLocalPrivacy (ble_abs_ctrl_t *const *p_ctrl*, uint8_t const *const *p_lc_irk*, uint8_t *privacy_mode*)

Generate a IRK, add it to the resolving list, set privacy mode and enable RPA function. Register vendor specific callback function, if IRK is generated by this function. After configuring local device privacy, BLE_GAP_ADDR_RPA_ID_PUBLIC is specified as own device address in theadvertising/scan/create connection API. Implements ble_abs_api_t::setLocalPrivacy

Example:

```
/* Set local privacy. */
    err = RM_BLE_ABS_SetLocalPrivacy(&g_ble_abs0_ctrl, p_local_irk, privacy_mode);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_ARGUMENT	The privacy_mode parameter is out of range.

Generate a IRK, add it to the resolving list, set privacy mode and enable RPA function. Register vendor specific callback function, if IRK is generated by this function. After configuring local device privacy, BLE_GAP_ADDR_RPA_ID_PUBLIC is specified as own device address in theadvertising/scan/create connection API. Implements ble_abs_api_t::setLocalPrivacy

Return values

FSP_ERR_ASSERTION	p_instance_ctrl is specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_ARGUMENT	The privacy_mode parameter is out of range.
FSP_ERR_UNSUPPORTED	Function is not supported

◆ RM_BLE_ABS_StartAuthentication()

fsp_err_t RM_BLE_ABS_StartAuthentication (ble_abs_ctrl_t *const *p_ctrl*, uint16_t *connection_handle*)

Start pairing or encryption. If pairing has been done, start encryption. The pairing parameters are configured by RM_BLE_ABS_Open() or R_BLE_GAP_SetPairingParams(). If the pairing parameters are configure by RM_BLE_ABS_Open(),

- bonding policy is that bonding information is stored.
- Key press notification is not supported. Implements ble_abs_api_t::startAuthentication.

Example:

```
/* Start authentication with remote device. */
   err = RM_BLE_ABS_StartAuthentication(&g_ble_abs0_ctrl, g_connection_handle);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_ASSERTION	p_instance_ctrl or connection_handle are specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_ARGUMENT	The connection handle parameter is out of range.

Start pairing or encryption. If pairing has been done, start encryption. The pairing parameters are configured by RM_BLE_ABS_Open() or R_BLE_GAP_SetPairingParams(). If the pairing parameters are configure by RM_BLE_ABS_Open(),

- bonding policy is that bonding information is stored.
- Key press notification is not supported. Implements ble_abs_api_t::startAuthentication.

Example:

```
/* Start authentication with remote device. */
   err = RM_BLE_ABS_StartAuthentication(&g_ble_abs0_ctrl, g_connection_handle);
```

Return values

FSP_ERR_ASSERTION	p_instance_ctrl or connection_handle are specified as NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_UNSUPPORTED	Function is not supported

◆ RM_BLE_ABS_DeleteBondInformation()

fsp_err_t RM_BLE_ABS_DeleteBondInformation (ble_abs_ctrl_t *const *p_ctrl*, ble_abs_bond_information_parameter_t const *const *p_bond_information_parameter*)

Delete bonding information from BLE stack and storage. Implements ble_abs_api_t::deleteBondInformation.

Example:

```
/* Delete bonding information. */
    err = RM_BLE_ABS_DeleteBondInformation(&g_ble_abs0_ctrl,
&g_bond_information_parameter);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_ASSERTION	The parameter p_instance_ctrl is NULL.
FSP_ERR_INVALID_POINTER	The parameter p_bond_information_parameter is NULL.
FSP_ERR_NOT_OPEN	Control block not open.

Delete bonding information from BLE stack and storage. Implements ble_abs_api_t::deleteBondInformation.

Example:

```
/* Delete bonding information. */
    err = RM_BLE_ABS_DeleteBondInformation(&g_ble_abs0_ctrl,
&g_bond_information_parameter);
```

Return values

FSP_SUCCESS	Operation was successful
FSP_ERR_ASSERTION	The parameter p_instance_ctrl is NULL.
FSP_ERR_INVALID_POINTER	The parameter p_bond_information_parameter is NULL.
FSP_ERR_NOT_OPEN	Control block not open.

◆ RM_BLE_ABS_ImportKeyInformation()

fsp_err_t RM_BLE_ABS_ImportKeyInformation (ble_abs_ctrl_t *const *p_ctrl*, ble_device_address_t * *p_local_identity_address*, uint8_t * *p_local_irk*, uint8_t * *p_local_csrk*)

Import key information to BLE stack and storage. Implements ble_abs_api_t::importKeyInformation.

Example:

```
 /* Import local key information. */
    err = RM_BLE_ABS_ImportKeyInformation(&g_ble_abs0_ctrl, &local_identity_address,
local_irk, local_csrk);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_ASSERTION	The parameter p_instance_ctrl is NULL.
FSP_ERR_INVALID_POINTER	The parameter p_local_identity_address, p_local_irk or p_local_csrk is NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_INVALID_HW_CONDITION	Failure to access internal storage.
FSP_ERR_UNSUPPORTED	Not supported in this configuration.

◆ RM_BLE_ABS_ExportKeyInformation()

fsp_err_t RM_BLE_ABS_ExportKeyInformation (ble_abs_ctrl_t *const *p_ctrl*, ble_device_address_t * *p_local_identity_address*, uint8_t * *p_local_irk*, uint8_t * *p_local_csrk*)

Export key information to BLE stack and storage. Implements ble_abs_api_t::exportKeyInformation.

Example:

```
 /* Export local key information. */
    err = RM_BLE_ABS_ExportKeyInformation(&g_ble_abs0_ctrl, &local_identity_address,
local_irk, local_csrk);
```

Return values

FSP_SUCCESS	Operation succeeded.
FSP_ERR_ASSERTION	The parameter p_instance_ctrl is NULL.
FSP_ERR_INVALID_POINTER	The parameter p_local_identity_address, p_local_irk or p_local_csrk is NULL.
FSP_ERR_NOT_OPEN	Control block not open.
FSP_ERR_BUFFER_EMPTY	Dynamic memory allocation failed.
FSP_ERR_OUT_OF_MEMORY	Failure to access internal storage.
FSP_ERR_NOT_INITIALIZED	Not initialized internal storage.
FSP_ERR_UNSUPPORTED	Not supported in this configuration.

4.2.78 Bluetooth Low Energy Abstraction with RYZ012 (rm_ble_abs_spp)

Modules

Middleware for the Bluetooth peripheral on RA MCUs. This module implements the BLE ABS Interface .

Overview

This module provides BLE GAP functionality that complies with the Bluetooth Core Specification version 5.0 specified by the Bluetooth SIG. This module is configured via the QE for BLE. QE for BLE provides standard and custom services defined by the user.

The module supports the Renesas Electronics RYZ012 Bluetooth® Low Energy 5 Module. The connection to the host is done through connecting the module to a PMOD on the development board. The rm_ble_abs_spp driver interfaces with the RYZ012 module over UART.

Features

The Bluetooth Low Energy Abstraction module with SPP supports the following features:

- Common functionality
    - Open/Close the BLE protocol stack.
    - Setting Transmit Power Level.
- The following GAP Role support
    - Peripheral: The device that accepts a connection request from Central and establishes a connection.
- GAP functionality
    - Initialize the Host stack.
    - Setting address.
    - Start/Stop Advertising.
    - Connect/Disconnect a link.
- GATT Common functionality
    - Get MTU Size.
- GATT Server functionality
    - Initialization of GATT Server.
    - Loading of Profile definition.
    - Notification of characteristics modification.
    - Read/Write of GATT Profile from host.

Target Devices

The Bluetooth Low Energy Abstraction module supports the following devices.

- RA6 line of devices using a PMOD connector for the RYZ012 module
- RA4 Line of devices using a PMOD connector for the RYZ012 module

Configuration

Build Time Configurations for rm_ble_abs_spp

The following build time configurations are defined in fsp_cfg/rm_ble_abs_spp_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enable • Disable	Default (BSP)	Specify whether to include code for API parameter checking.
Reset Port	Refer to the RA Configuration tool for available options.	03	Specify the module reset pin port for the MCU.
Reset Pin	Refer to the RA Configuration tool for available options.	10	Specify the module reset pin for the MCU.
UART/SPI Select Port (PB5)	Refer to the RA Configuration tool for available options.	03	Specify the module PB5 port for the MCU.
UART/SPI Select Pin (PB5)	Refer to the RA Configuration tool for available options.	11	Specify the module PB5 pin for the MCU.

Transmit Power Level (in dBm)	Refer to the RA Configuration tool for available options.	4.57	Specify the module transmit power in dBm.

Configurations for Networking > SPP BLE Abstraction (rm_ble_abs_spp)

This module can be added to the Stacks tab via New Stack > Networking > SPP BLE Abstraction (rm_ble_abs_spp).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_ble_abs0	Module name.
General > GAP callback	Name must be a valid C symbol	gap_cb	If QE is used, set to NULL.
General > Vendor specific callback	Name must be a valid C symbol	vs_cb	If QE is used, set to NULL.
General > GATT server callback parameter	Name must be a valid C symbol	gs_abs_gatts_cb_param	If QE is used, set to NULL.
General > GATT server callback number	Must be a valid number	2	The number of GATT Server callback functions.
General > GATT client callback parameter	Name must be a valid C symbol	gs_abs_gattc_cb_param	Set GATT client callback parameter. If QE is used, set to NULL.
General > GATT client callback number	Must be a valid number	2	The number of GATT Server callback functions.

Pin Configuration

Refer to Serial Communications Interface (SCI) UART (r_sci_uart). R_SCI_UART_Open() is called by Bluetooth Low Energy Abstraction with RYZ012 (rm_ble_abs_spp)

GPIO pins used by the BLE driver to control RYZ012:

- Reset Pin : Active low reset line for RYZ012 module
- UART/SPI Select Pin : Allows selection between UART or SPI from module communication. UART is selected with output low and SPI with output high.

Usage Notes

Limitations

Developers should be aware of the following limitations when using the BLE_ABS:

Currently supported rm_ble_abs_spp interface functions:

- RM_BLE_ABS_Open
- RM_BLE_ABS_Close

- RM_BLE_ABS_StartLegacyAdvertising

Currently supported r_ble_spp_api interface functions:

- R_BLE_Open
- R_BLE_Close
- R_BLE_SetEvent
- R_BLE_GAP_Init
- R_BLE_GAP_StopAdv
- R_BLE_GAP_SetAdvSresData
- R_BLE_GAP_StartAdv
- R_BLE_GAP_SetAdvParam
- R_BLE_GATT_GetMtu
- R_BLE_GATTS_SetDbInst
- R_BLE_GATTS_Init
- R_BLE_GATTS_GetAttr
- R_BLE_GATTS_SetAttr
- R_BLE_GATTS_Notification
- R_BLE_GATTS_Indication
- R_BLE_GATTS_RegisterCb
- R_BLE_GATTC_Init
- R_BLE_GATTC_RegisterCb
- R_BLE_VS_SetBdAddr
- R_BLE_VS_SetTxPower

Currently supported callback events:

- BLE_GAP_EVENT_CONN_IND
- BLE_GAP_EVENT_DISCONN_IND
- BLE_GAP_EVENT_ADV_ON
- BLE_GAP_EVENT_ADV_OFF
- BLE_GATTC_EVENT_CONN_IND
- BLE_GATTS_EVENT_CONN_IND
- BLE_GATTC_EVENT_DISCONN_IND
- BLE_GATTS_EVENT_DISCONN_IND
- BLE_GATTS_EVENT_WRITE_RSP_COMP

Transmit Power

Specific Operational Use Conditions for RYZ012 BLE module:

Please consult the laws and regulations of the region(s) in which the device will be used to verify the information below before using this device.

North America (FCC) The module must not be operated at power levels above 10.0 dBm. Host devices that need higher output power may not be marketed without prior re-certification.

Europe (RED) The module must not be operated at power levels above 8.4 dBm. Host devices that need higher output power may not be marketed in regions covered by RED regulation without prior re-certification.

Japan (MIC) The module must be operated at power level 4.5 dBm. Host devices that need to change output power may not be marketed without prior re-certification.

The current default level used in the E2 Studio Configurator is 4.5 dBm for the transmit power level.

Examples

BLE_ABS_SPP Basic Example

This is a basic example of minimal use of the BLE_ABS_SPP in an application.

```
/* The callback is called when peripheral event occurs. */
void gap_peripheral_cb (uint16_t type, ble_status_t result, st_ble_evt_data_t *
p_data)
{
 FSP_PARAMETER_NOT_USED(result);
 FSP_PARAMETER_NOT_USED(p_data);
 switch (type)
    {
 case BLE_GAP_EVENT_STACK_ON:
      {
            g_ble_event_flag = g_ble_event_flag | BLE_ABS_EVENT_FLAG_STACK_ON;
 break;
       }
 case BLE_GAP_EVENT_ADV_ON:
      {
 st_ble_gap_adv_set_evt_t * p_gap_adv_set_evt_param = (st_ble_gap_adv_set_evt_t *)
p_data->p_param;
            g_advertising_handle = p_gap_adv_set_evt_param->adv_hdl;
            g_ble_event_flag    |= BLE_ABS_EVENT_FLAG_ADV_ON;
 break;
       }
 case BLE_GAP_EVENT_ADV_OFF:
      {
            g_ble_event_flag |= BLE_ABS_EVENT_FLAG_ADV_OFF;
 break;
       }
 case BLE_GAP_EVENT_CONN_IND:
      {
            g_ble_event_flag |= BLE_ABS_EVENT_FLAG_CONN_IND;
```

```
 break;
      }
     {
 /* Do nothing. */
 break;
        }
 default:
 break;
    }
}
```

```
#define BLE_ABS_EVENT_FLAG_STACK_ON (0x01 << 0)
#define BLE_ABS_EVENT_FLAG_CONN_IND (0x01 << 1)
#define BLE_ABS_EVENT_FLAG_ADV_ON (0x01 << 2)
#define BLE_ABS_EVENT_FLAG_ADV_OFF (0x01 << 3)
#define BLE_ABS_EVENT_FLAG_DISCONN_IND (0x01 << 4)
#define BLE_ABS_EVENT_FLAG_RSLV_LIST_CONF_COMP (0x01 << 5)
#define BLE_ABS_EXAMPLE_SHORTENED_LOCAL_NAME 'E', 'x', 'a', 'm', 'p', 'l', 'e'
#define BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME 'T', 'E', 'S', 'T', '_', 'E', 'x', 'a',
'm', 'p', 'l', 'e'
#define BLE_ABS_EXAMPLE_SLOW_ADVERTISING_INTERVAL (0x00000640)
void ble_abs_peripheral_example (void)
{
 fsp_err_t         err    = FSP_SUCCESS;
 volatile uint32_t timeout = UINT16_MAX * UINT8_MAX * 8;
    uint8_t advertising_data[] =
    {
 /* Flags */
       0x02,
       0x01,
       (0x1a),
 /* Shortened Local Name */
       0x08,
       0x08,
```

```
        BLE_ABS_EXAMPLE_SHORTENED_LOCAL_NAME,
    };
 /* Scan Response Data */
    uint8_t scan_response_data[] =
    {
 /* Complete Local Name */
        0x0D,
        0x09,
        BLE_ABS_EXAMPLE_COMPLETE_LOCAL_NAME,
    };
 ble_abs_legacy_advertising_parameter_t legacy_advertising_parameter =
    {
        .p_peer_address             =
NULL,
        .slow_advertising_interval =
BLE_ABS_EXAMPLE_SLOW_ADVERTISING_INTERVAL,
        .slow_advertising_period    =
0x0000,
        .p_advertising_data         =
advertising_data,
        .advertising_data_length    = sizeof
(advertising_data),
        .p_scan_response_data       =
scan_response_data,
        .scan_response_data_length = sizeof
(scan_response_data),
        .advertising_filter_policy = BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY
,
        .advertising_channel_map    = (BLE_GAP_ADV_CH_37 | BLE_GAP_ADV_CH_38 |
BLE_GAP_ADV_CH_39),
        .own_bluetooth_address_type = BLE_GAP_ADDR_PUBLIC
,
        .own_bluetooth_address      = {0},
    };
```

```
    g_ble_event_flag = 0;
 /* Open the module. */
    err = RM_BLE_ABS_Open(&g_ble_abs0_ctrl, &g_ble_abs0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Configure the Transmit Level */
    err = R_BLE_VS_SetTxPower(0, BLE_ABS_TRANSMIT_POWER);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait BLE_GAP_EVENT_STACK_ON event is notified. */
 while (!(BLE_ABS_EVENT_FLAG_STACK_ON & g_ble_event_flag) && (--timeout > 0U))
    {
 R_BLE_Execute();
    }
    time_out_handle_error(timeout);
    g_ble_event_flag = 0;
    timeout = UINT16_MAX * UINT8_MAX * 8;
 /* Start advertising. */
    err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (!(BLE_ABS_EVENT_FLAG_CONN_IND & g_ble_event_flag) && (--timeout > 0U))
    {
 if (BLE_ABS_EVENT_FLAG_ADV_OFF & g_ble_event_flag)
      {
 /* Restart advertise, when stop advertising. */
      err = RM_BLE_ABS_StartLegacyAdvertising(&g_ble_abs0_ctrl,
&legacy_advertising_parameter);
 if (FSP_SUCCESS == err)
      {
                g_ble_event_flag &= (uint16_t) ~BLE_ABS_EVENT_FLAG_ADV_OFF;
         }
 else if (FSP_ERR_INVALID_STATE == err)
```

```
     {
/* BLE driver state is busy. */
               ;
        }
else
     {
/* Handle any errors. This function should be defined by the user. */
     assert(FSP_SUCCESS == err);
        }
      }
else if ((timeout % BLE_ABS_RETRY_INTERVAL) == 0U)
     {
/* Stop advertising after a certain amount of time */
R_BLE_GAP_StopAdv(g_advertising_handle);
      }
else
     {
           ;
      }
R_BLE_Execute();
   }
   time_out_handle_error(timeout);
/* Clean up & Close BLE driver */
   g_ble_event_flag = 0;
/* Close BLE driver */
   err = RM_BLE_ABS_Close(&g_ble_abs0_ctrl);
/* Handle any errors. This function should be defined by the user. */
   assert(FSP_SUCCESS == err);
}
```

4.2.79 SD/MMC Block Media Implementation (rm_block_media_sdmmc)

Modules

Functions

fsp_err_t	RM_BLOCK_MEDIA_SDMMC_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_MediaInit (rm_block_media_ctrl_t *const p_ctrl)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_Write (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_CallbackSet (rm_block_media_ctrl_t *const p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void const *const p_context, rm_block_media_callback_args_t *const p_callback_memory)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_StatusGet (rm_block_media_ctrl_t *const p_api_ctrl, rm_block_media_status_t *const p_status)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info)
fsp_err_t	RM_BLOCK_MEDIA_SDMMC_Close (rm_block_media_ctrl_t *const p_ctrl)

Detailed Description

Middleware to implement the block media interface on SD cards. This module implements the Block Media Interface.

Overview

Features

The SD/MMC implementation of the block media interface has the following key features:

- Reading, writing, and erasing data from an SD card
- Callback called when card insertion or removal is detected
- Provides media information such as sector size and total number of sectors.

Configuration

Build Time Configurations for rm_block_media_sdmmc

The following build time configurations are defined in driver/rm_block_media_sdmmc_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Storage > Block Media SD/MMC (rm_block_media_sdmmc)

This module can be added to the Stacks tab via New Stack > Storage > Block Media SD/MMC (rm_block_media_sdmmc). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_block_media0	Module name.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called when a card is inserted or removed.

Clock Configuration

This module has no required clock configurations.

Pin Configuration

This module does not use I/O pins.

Examples

Basic Example

This is a basic example of minimal use of the SD/MMC block media implementation in an application.

```
#define RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE (512)
uint8_t g_dest[RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t g_src[RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint32_t g_transfer_complete = 0;
void rm_block_media_sdmmc_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_SDMMC_BLOCK_SIZE; i++)
    {
```

```
        g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_SDMMC driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_SDMMC_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* A device shall be ready to accept the first command within 1ms from detecting VDD
min. Reference section 6.4.1.1
  * "Power Up Time of Card" in the SD Physical Layer Simplified Specification Version
6.00. */
 R_BSP_SoftwareDelay(1U, BSP_DELAY_UNITS_MILLISECONDS);
 /* Initialize the SD card. This should not be done until the card is plugged in for
SD devices. */
    err = RM_BLOCK_MEDIA_SDMMC_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Write a block of data to sector 3 of an SD card. */
    err = RM_BLOCK_MEDIA_SDMMC_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
    assert(FSP_SUCCESS == err);
 /* Read a block of data from sector 3 of an SD card. */
    err = RM_BLOCK_MEDIA_SDMMC_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
    assert(FSP_SUCCESS == err);
}
```

Function Documentation

◆ RM_BLOCK_MEDIA_SDMMC_Open()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Open (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_cfg_t const *const *p_cfg*)

Opens the module.

Implements rm_block_media_api_t::open().

Return values

FSP_SUCCESS	Module is available and is now open.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_ALREADY_OPEN	Module has already been opened with this instance of the control structure.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::open

◆ RM_BLOCK_MEDIA_SDMMC_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_MediaInit (rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the SD or eMMC device. This procedure requires several sequential commands. This function blocks until all identification and configuration commands are complete.

Implements rm_block_media_api_t::mediaInit().

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::mediaInit

◆ RM_BLOCK_MEDIA_SDMMC_Read()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Read (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest_address*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Reads data from an SD or eMMC device. Up to 0x10000 sectors can be read at a time. Implements rm_block_media_api_t::read().

Return values

FSP_SUCCESS	Data read successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::read

◆ RM_BLOCK_MEDIA_SDMMC_Write()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Write (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src_address*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Writes data to an SD or eMMC device. Up to 0x10000 sectors can be written at a time. Implements rm_block_media_api_t::write().

Return values

FSP_SUCCESS	Write finished successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::write

◆ RM_BLOCK_MEDIA_SDMMC_Erase()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Erase (rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Erases sectors of an SD card or eMMC device. Implements rm_block_media_api_t::erase().

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- sdmmc_api_t::erase
- sdmmc_api_t::statusGet

◆ RM_BLOCK_MEDIA_SDMMC_CallbackSet()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_CallbackSet (rm_block_media_ctrl_t *const *p_ctrl*, void(*)(rm_block_media_callback_args_t *) *p_callback*, void const *const *p_context*, rm_block_media_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements rm_block_media_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ RM_BLOCK_MEDIA_SDMMC_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_StatusGet (rm_block_media_ctrl_t *const *p_api_ctrl*, rm_block_media_status_t *const *p_status*)

Provides driver status. Implements rm_block_media_api_t::statusGet().

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_BLOCK_MEDIA_SDMMC_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_InfoGet (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info*)

Retrieves module information. Implements rm_block_media_api_t::infoGet().

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

◆ RM_BLOCK_MEDIA_SDMMC_Close()

fsp_err_t RM_BLOCK_MEDIA_SDMMC_Close (rm_block_media_ctrl_t *const *p_ctrl*)

Closes an open SD/MMC device. Implements rm_block_media_api_t::close().

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

4.2.80 SPI Block Media Implementation (rm_block_media_spi)

Modules

Functions

fsp_err_t	RM_BLOCK_MEDIA_SPI_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)
fsp_err_t	RM_BLOCK_MEDIA_SPI_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info)
fsp_err_t	RM_BLOCK_MEDIA_SPI_MediaInit (rm_block_media_ctrl_t *const p_ctrl)
fsp_err_t	RM_BLOCK_MEDIA_SPI_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_block, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_SPI_StatusGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_status_t *const p_status)
fsp_err_t	RM_BLOCK_MEDIA_SPI_Write (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const start_block, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_SPI_CallbackSet (rm_block_media_ctrl_t *const p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void const *const p_context, rm_block_media_callback_args_t *const p_callback_memory)
fsp_err_t	RM_BLOCK_MEDIA_SPI_Close (rm_block_media_ctrl_t *const p_ctrl)
fsp_err_t	RM_BLOCK_MEDIA_SPI_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const start_block, uint32_t const num_blocks)

Detailed Description

Middleware to implement the block media interface on SPI flash memory. This module implements the Block Media Interface.

Overview

Features

The SPI implementation of the block media interface has the following key features:

- Reading, writing, and erasing data from SPI flash memory
- Provides media information such as sector size and total number of sectors.

  Note

  By default, Block Media SPI Read, Write, and Erase are blocking operations. Non-blocking operation may be achieved by yielding control within the optional callback function.

Configuration

Build Time Configurations for rm_block_media_spi

The following build time configurations are defined in driver/rm_block_media_spi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected, code for parameter checking is included in the build

Configurations for Storage > Block Media SPI Flash (rm_block_media_spi)

This module can be added to the Stacks tab via New Stack > Storage > Block Media SPI Flash (rm_block_media_spi).

Configuration	Options	Default	Description
Module Instance Name	Name must be a valid C symbol	g_rm_block_media0	Module name
Block size (bytes)	Manual Entry	4096	Specify the size of a block in bytes.
Block count	Minimum block count is 1, maximum is defined by hardware and software design.	8192	Number of blocks available for use by this driver instance.
Base Address	Manual Entry	0	Base address offset (bytes) for instance memory region.
Callback Function	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback is provided, it will be called after the completion of Read, Write, and Erase operations, or anytime these functions are waiting on hardware.
[DEPRECATED] Callback Context	Name must be a valid C symbol	NULL	A user specified context that will be provided back to the user when a callback occurs.

Clock Configuration

This module has no required clock configurations.

Pin Configuration

This module does not use I/O pins.

Limitations

Developers should be aware of the following limitations when using RM_BLOCK_MEDIA_SPI:

- Getting and setting Block Protection or Advanced Sector Protection modes is not supported.
- Addressing QSPI memory address ranges greater than 64 MB (one bank) is not supported.

Examples

Basic Example

This is a basic example of minimal use of the SPI block media implementation in an application.

```
#define RM_BLOCK_MEDIA_SPI_BLOCK_SIZE (256U)
uint8_t g_dest[RM_BLOCK_MEDIA_SPI_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t g_src[RM_BLOCK_MEDIA_SPI_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
void rm_block_media_spi_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_SPI_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_SPI driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_SPI_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the SPI flash memory. */
    err = RM_BLOCK_MEDIA_SPI_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Write a block of data to block 3 of the SPI flash memory. */
    err = RM_BLOCK_MEDIA_SPI_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
    assert(FSP_SUCCESS == err);
 /* Read a block of data from block 3 of the SPI flash memory. */
    err = RM_BLOCK_MEDIA_SPI_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
    assert(FSP_SUCCESS == err);
```

```
}
```

Non-Blocking Example

This is a basic example of using the optional SPI callback to impliment non-blocking operation.

```
#define RM_BLOCK_MEDIA_EXAMPLE_DEVICE_BLOCK_COUNT 0x1000
void rm_block_media_spi_non_blocking_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_SPI_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_SPI driver. This enables the card detection interrupt. */
 fsp_err_t err = RM_BLOCK_MEDIA_SPI_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the Block Media SPI driver. */
    err = RM_BLOCK_MEDIA_SPI_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Erase a large quantity of data from SPI Flash Memory */
    err = RM_BLOCK_MEDIA_SPI_Erase(&g_rm_block_media0_ctrl, 0,
RM_BLOCK_MEDIA_EXAMPLE_DEVICE_BLOCK_COUNT);
    assert(FSP_SUCCESS == err);
}
/* The optional callback is invoked for Read, Write, and Erase operations, whenever
the operation completes or has
 * been blocked by the lower level SPI driver busy indication.
 */
void rm_block_media_spi_example_callback (rm_block_media_callback_args_t * p_args)
{
 if (RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE == p_args->event)
    {
```

```
/* TODO: Process operation complete. */
   }
else if (RM_BLOCK_MEDIA_EVENT_POLL_STATUS == p_args->event)
   {
rm_block_media_status_t status;
rm_block_media_ctrl_t * p_ctrl = (rm_block_media_ctrl_t *) p_args->p_context;
fsp_err_t err = RM_BLOCK_MEDIA_SPI_StatusGet(p_ctrl, &status);
     assert(FSP_SUCCESS == err);
if (true == status.busy)
     {
/* Run waiting tasks */
           vTaskSuspend(xTaskGetCurrentTaskHandle());
     }
   }
else
   {
      assert(RM_BLOCK_MEDIA_EVENT_ERROR == p_args->event);
/* TODO: Process Read, Write, or Erase error. */
   }
}
```

Function Documentation

◆ RM_BLOCK_MEDIA_SPI_Open()

fsp_err_t RM_BLOCK_MEDIA_SPI_Open (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_cfg_t const *const *p_cfg*)

Parameter checking and Acquires mutex, then handles driver initialization at the HAL SPI layer and marking the open flag in control block.

Implements rm_block_media_api_t::open.

Return values

FSP_SUCCESS	Block media for SPI framework is successfully opened.
FSP_ERR_ASSERTION	One of the input parameters or their data references may be null.
FSP_ERR_ALREADY_OPEN	The channel specified has already been opened. See HAL driver for other possible causes.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- spi_flash_api_t::open

◆ RM_BLOCK_MEDIA_SPI_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_SPI_InfoGet (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info*)

Retrieves module information.

Implements rm_block_media_api_t::infoGet.

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

◆ RM_BLOCK_MEDIA_SPI_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_SPI_MediaInit (rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the Block Media SPI Flash device.

Implements rm_block_media_api_t::mediaInit.

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_BLOCK_MEDIA_SPI_Read()

fsp_err_t RM_BLOCK_MEDIA_SPI_Read (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest*, uint32_t const *start_block*, uint32_t const *num_blocks*)

Reads a number of blocks from spi flash memory. By default, this is a function is blocking. Non-blocking operation may be achieved by yielding control within the optional callback function.

Implements rm_block_media_api_t::read.

Return values

FSP_SUCCESS	SPI data read successfully
FSP_ERR_ASSERTION	p_ctrl or p_dest is NULL, or num_blocks is zero
FSP_ERR_NOT_OPEN	Block Media SPI module is not yet open
FSP_ERR_INVALID_ADDRESS	Invalid address range for read operation
FSP_ERR_NOT_INITIALIZED	Block Media SPI module is not yet initialized

◆ RM_BLOCK_MEDIA_SPI_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_SPI_StatusGet (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_status_t *const *p_status*)

Provides driver status.

Implements rm_block_media_api_t::statusGet.

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls
- spi_flash_api_t::statusGet

◆ RM_BLOCK_MEDIA_SPI_Write()

fsp_err_t RM_BLOCK_MEDIA_SPI_Write (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src*, uint32_t const *start_block*, uint32_t const *num_blocks*)

Writes provided data to a number of blocks of spi flash memory. By default, this is a function is blocking. Non-blocking operation may be achieved by yielding control within the optional callback function.

Implements rm_block_media_api_t::write.

Return values

FSP_SUCCESS	Flash write finished successfully.
FSP_ERR_ASSERTION	p_ctrl or p_src is NULL. Or num_blocks is zero.
FSP_ERR_NOT_OPEN	Block media SPI Framework module is not yet initialized.
FSP_ERR_INVALID_ADDRESS	Invalid address range
FSP_ERR_NOT_INITIALIZED	Block Media SPI module is not yet initialized

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls
- spi_flash_api_t::write

◆ RM_BLOCK_MEDIA_SPI_CallbackSet()

fsp_err_t RM_BLOCK_MEDIA_SPI_CallbackSet (rm_block_media_ctrl_t *const *p_ctrl*, void(*)(rm_block_media_callback_args_t *) *p_callback*, void const *const *p_context*, rm_block_media_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. API not supported.

Implements rm_block_media_api_t::callbackSet.

Return values

FSP_ERR_UNSUPPORTED	API not supported by RM_BLOCK_MEDIA_SPI.

◆ RM_BLOCK_MEDIA_SPI_Close()

fsp_err_t RM_BLOCK_MEDIA_SPI_Close (rm_block_media_ctrl_t *const *p_ctrl*)

Closes the Block Media SPI device. Implements rm_block_media_api_t::close.

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	One of the following parameters may be null: p_ctrl.
FSP_ERR_NOT_OPEN	Block media SPI Framework module is not yet initialized.

Returns

See Common Error Codes or HAL driver for other possible return codes or causes. This function calls

- spi_flash_api_t::close

◆ RM_BLOCK_MEDIA_SPI_Erase()

fsp_err_t RM_BLOCK_MEDIA_SPI_Erase (rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *start_block*, uint32_t const *num_blocks*)

This function erases blocks of the SPI device. By default, this is a function is blocking. Non-blocking operation may be achieved by yielding control within the optional callback function.

Implements rm_block_media_api_t::erase.

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.
FSP_ERR_INVALID_ADDRESS	Invalid address range

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- spi_flash_api_t::erase
- spi_flash_api_t::statusGet

4.2.81 USB HMSC Block Media Implementation (rm_block_media_usb)

Modules

Functions

fsp_err_t	RM_BLOCK_MEDIA_USB_Open (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)
fsp_err_t	RM_BLOCK_MEDIA_USB_MediaInit (rm_block_media_ctrl_t *const p_ctrl)
fsp_err_t	RM_BLOCK_MEDIA_USB_Read (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_USB_Write (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_USB_Erase (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t	RM_BLOCK_MEDIA_USB_CallbackSet (rm_block_media_ctrl_t *const

	p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void const *const p_context, rm_block_media_callback_args_t *const p_callback_memory)
fsp_err_t	RM_BLOCK_MEDIA_USB_StatusGet (rm_block_media_ctrl_t *const p_api_ctrl, rm_block_media_status_t *const p_status)
fsp_err_t	RM_BLOCK_MEDIA_USB_InfoGet (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info)
fsp_err_t	RM_BLOCK_MEDIA_USB_Close (rm_block_media_ctrl_t *const p_ctrl)

Detailed Description

Middleware to implement the block media interface on USB mass storage devices. This module implements the Block Media Interface.

Overview

Features

The USB implementation of the block media interface has the following key features:

- Reading, writing, and erasing data from a USB mass storage device
- Callback called when device insertion or removal is detected
- Provides media information such as sector size and total number of sectors.

Configuration

Build Time Configurations for rm_block_media_usb

The following build time configurations are defined in driver/rm_block_media_usb_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Storage > Block Media USB (rm_block_media_usb)

This module can be added to the Stacks tab via New Stack > Storage > Block Media USB (rm_block_media_usb).

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_block_media0	Module name.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this

			callback function is provided, it will be called when a device is attached or removed.
Pointer to user context	Name must be a valid C symbol	NULL	A user context can be provided. If this context is provided, it will be passed to callback function when a device is attached or removed.

Note

RM_BLOCK_MEDIA_USB_MediaInit function must be called after receiving the insert event notification.

Clock Configuration

This module has no required clock configurations.

Pin Configuration

This module does not use I/O pins.

Examples

Basic Example

This is a basic example of minimal use of the USB mass storage block media implementation in an application.

```
#define RM_BLOCK_MEDIA_USB_BLOCK_SIZE (512)
volatile bool g_usb_inserted = false;
uint8_t       g_dest[RM_BLOCK_MEDIA_USB_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
uint8_t       g_src[RM_BLOCK_MEDIA_USB_BLOCK_SIZE] BSP_ALIGN_VARIABLE(4);
void rm_block_media_usb_basic_example (void)
{
 /* Initialize g_src to known data */
 for (uint32_t i = 0; i < RM_BLOCK_MEDIA_USB_BLOCK_SIZE; i++)
    {
       g_src[i] = (uint8_t) ('A' + (i % 26));
    }
 /* Open the RM_BLOCK_MEDIA_USB driver. */
 fsp_err_t err = RM_BLOCK_MEDIA_USB_Open(&g_rm_block_media0_ctrl,
&g_rm_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
```

```
    assert(FSP_SUCCESS == err);
 while (!g_usb_inserted)
    {
 /* Wait for a card insertion interrupt. */
    }
 /* Initialize the mass storage device. This should not be done until the device is
plugged in and initialized. */
    err = RM_BLOCK_MEDIA_USB_MediaInit(&g_rm_block_media0_ctrl);
    assert(FSP_SUCCESS == err);
 /* Write a block of data to sector 3 of an USB mass storage device. */
    err = RM_BLOCK_MEDIA_USB_Write(&g_rm_block_media0_ctrl, g_src, 3, 1);
    assert(FSP_SUCCESS == err);
 /* Read a block of data from sector 3 of an USB mass storage device. */
    err = RM_BLOCK_MEDIA_USB_Read(&g_rm_block_media0_ctrl, g_dest, 3, 1);
    assert(FSP_SUCCESS == err);
}
```

Device Insertion

This is an example of using a callback to determine when a mass storage device is plugged in and enumerated.

```
/* The callback is called when a media insertion event occurs. */
void rm_block_media_usb_media_insertion_example_callback
(rm_block_media_callback_args_t * p_args)
{
 if (RM_BLOCK_MEDIA_EVENT_MEDIA_INSERTED == p_args->event)
    {
       g_usb_inserted = true;
    }
 if (RM_BLOCK_MEDIA_EVENT_MEDIA_REMOVED == p_args->event)
    {
       g_usb_inserted = false;
    }
}
```

Function Documentation

◆ RM_BLOCK_MEDIA_USB_Open()

fsp_err_t RM_BLOCK_MEDIA_USB_Open (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_cfg_t const *const *p_cfg*)

Opens the module.

Implements rm_block_media_api_t::open().

Return values

FSP_SUCCESS	Module is available and is now open.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_ALREADY_OPEN	Module has already been opened with this instance of the control structure.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_USB_MediaInit()

fsp_err_t RM_BLOCK_MEDIA_USB_MediaInit (rm_block_media_ctrl_t *const *p_ctrl*)

Initializes the USB device.

Implements rm_block_media_api_t::mediaInit().

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_BLOCK_MEDIA_USB_Read()

fsp_err_t RM_BLOCK_MEDIA_USB_Read (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t *const *p_dest_address*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Reads data from an USB device. Implements rm_block_media_api_t::read().

This function blocks until the data is read into the destination buffer.

Return values

FSP_SUCCESS	Data read successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.
FSP_ERR_USB_FAILED	The message could not received completed successfully.

Returns
See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_USB_Write()

fsp_err_t RM_BLOCK_MEDIA_USB_Write (rm_block_media_ctrl_t *const *p_ctrl*, uint8_t const *const *p_src_address*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Writes data to an USB device. Implements rm_block_media_api_t::write().

This function blocks until the write operation completes.

Return values

FSP_SUCCESS	Write finished successfully.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.
FSP_ERR_USB_FAILED	The message could not received completed successfully.

Returns
See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_USB_Erase()

fsp_err_t RM_BLOCK_MEDIA_USB_Erase (rm_block_media_ctrl_t *const *p_ctrl*, uint32_t const *block_address*, uint32_t const *num_blocks*)

Erases sectors of an USB device. Implements rm_block_media_api_t::erase().

This function blocks until erase is complete.

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

Returns
See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_USB_CallbackSet()

fsp_err_t RM_BLOCK_MEDIA_USB_CallbackSet (rm_block_media_ctrl_t *const *p_ctrl*, void(*)(rm_block_media_callback_args_t *) *p_callback*, void const *const *p_context*, rm_block_media_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure. Implements rm_block_media_api_t::callbackSet.

Note
This function is currently unsupported for Block Media over USB.

Return values

FSP_ERR_UNSUPPORTED	CallbackSet is not currently supported for Block Media over USB.

◆ RM_BLOCK_MEDIA_USB_StatusGet()

fsp_err_t RM_BLOCK_MEDIA_USB_StatusGet (rm_block_media_ctrl_t *const *p_api_ctrl*, rm_block_media_status_t *const *p_status*)

Provides driver status. Implements rm_block_media_api_t::statusGet().

Return values

FSP_SUCCESS	Status stored in p_status.
FSP_ERR_ASSERTION	NULL pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_BLOCK_MEDIA_USB_InfoGet()

fsp_err_t RM_BLOCK_MEDIA_USB_InfoGet (rm_block_media_ctrl_t *const *p_ctrl*, rm_block_media_info_t *const *p_info*)

Retrieves module information. Implements rm_block_media_api_t::infoGet().

Return values

FSP_SUCCESS	Erase operation requested.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_NOT_INITIALIZED	Module has not been initialized.

◆ RM_BLOCK_MEDIA_USB_Close()

fsp_err_t RM_BLOCK_MEDIA_USB_Close (rm_block_media_ctrl_t *const *p_ctrl*)

Closes an open USB device. Implements rm_block_media_api_t::close().

Return values

FSP_SUCCESS	Successful close.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

4.2.82 User Block Media Implementation (rm_block_media_user)

Modules

Middleware that implements a block media interface on the media of your choice. This module implements the Block Media Interface.

Overview

Features

This module is for using Block media with user-selected media.

Configuration

Block Media User has no output config settings.
The user is required to create the config settings etc. in the application.
The figure below is an example of the config definition when the user media in USB PMSC is RAM.

```
const rm_block_media_api_t g_rm_block_media_on_user_media =
{
    .open      = RM_BLOCK_MEDIA_RAM_Open,
    .mediaInit = RM_BLOCK_MEDIA_RAM_MediaInit,
    .read      = RM_BLOCK_MEDIA_RAM_Read,
    .write    = RM_BLOCK_MEDIA_RAM_Write,
    .erase    = RM_BLOCK_MEDIA_RAM_Erase,
    .infoGet   = RM_BLOCK_MEDIA_RAM_InfoGet,
    .statusGet = RM_BLOCK_MEDIA_RAM_StatusGet,
    .close    = RM_BLOCK_MEDIA_RAM_Close,
};
extern void r_usb_pmsc_block_media_event_callback(rm_block_media_callback_args_t *
p_args);
const rm_block_media_cfg_t g_rm_block_media0_cfg =
{.p_extend = NULL, .p_callback = r_usb_pmsc_block_media_event_callback, .p_context =
NULL, };
rm_block_media_instance_t g_rm_block_media0 =
{.p_api = &g_rm_block_media_on_user_media, .p_ctrl = NULL, .p_cfg =
&g_rm_block_media0_cfg, };
```

Note

If you use block_media_user, you need to create a function that matches the media you are using.
In the above example, this is the function with RM_BLOCK_MEDIA_.
Register the created function in rm_block_media_api_t.
The registered rm_block_media_api_t is registered in p_api, which is a member of rm_block_media_instance_t.

Clock Configuration

This module has no required clock configurations.

Pin Configuration

This module does not use I/O pins.

Examples

Basic Example

Please refer to USB Peripheral Mass Storage Class (r_usb_pmsc) for the PMSC application given as an example.

4.2.83 I2C Communicatons Middleware (rm_comms_i2c)

Modules

Functions

fsp_err_t	RM_COMMS_I2C_Open (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_cfg_t const *const p_cfg) Opens and configures the Communications Middle module. Implements rm_comms_api_t::open. More...
fsp_err_t	RM_COMMS_I2C_Close (rm_comms_ctrl_t *const p_api_ctrl) Disables specified Communications Middle module. Implements rm_comms_api_t::close. More...
fsp_err_t	RM_COMMS_I2C_Read (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_dest, uint32_t const bytes) Performs a read from the I2C device. Implements rm_comms_api_t::read. More...
fsp_err_t	RM_COMMS_I2C_Write (rm_comms_ctrl_t *const p_api_ctrl, uint8_t *const p_src, uint32_t const bytes)

	Performs a write from the I2C device. Implements rm_comms_api_t::write. More...
fsp_err_t	RM_COMMS_I2C_WriteRead (rm_comms_ctrl_t *const p_api_ctrl, rm_comms_write_read_params_t const write_read_params) Performs a write to, then a read from the I2C device. Implements rm_comms_api_t::writeRead. More...
void	rm_comms_i2c_callback (i2c_master_callback_args_t *p_args) Common callback function called in the I2C driver callback function.

Detailed Description

Middleware to implement the I2C communications interface. This module implements the Communicatons Middleware Interface.

Overview

Features

The implementation of the I2C communications interface has the following key features:

- Reading data from, writing data to I2C bus
- Writes to I2C bus, then reads with restart
- A single I2C bus used by multiple I2C devices

Configuration

Build Time Configurations for rm_comms_i2c

The following build time configurations are defined in fsp_cfg/rm_comms_i2c_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Connectivity > I2C Shared Bus (rm_comms_i2c)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Shared Bus (rm_comms_i2c).

Configuration	Options	Default	Description
Name	Manual Entry	g_comms_i2c_bus0	Module name.
Bus Timeout	Value must be a non-	0xFFFFFFFF	Set timeout for locking

	negative integer		bus in using RTOS.
Semaphore for Blocking	• Unuse • Use	Unuse	Set Semaphore for blocking in using RTOS.
Recursive Mutex for Bus	• Unuse • Use	Unuse	Set Mutex for locking bus in using RTOS.

Configurations for Connectivity > I2C Communication Device (rm_comms_i2c)

This module can be added to the Stacks tab via New Stack > Connectivity > I2C Communication Device (rm_comms_i2c).

Configuration	Options	Default	Description
Name	Manual Entry	g_comms_i2c_device0	Module name.
Semaphore Timeout	Value must be a non-negative integer	0xFFFFFFFF	Set timeout for blocking in using RTOS.
Slave Address	Value must be non-negative	0x00	Specify the slave address.
Address Mode	• 7-Bit • 10-Bit	7-Bit	Select the I2C address mode.
Callback	Name must be a valid C symbol	comms_i2c_callback	A user callback function can be provided.

Pin Configuration

This module uses SDA and SCL pins of I2C Master and SCI I2C.

Usage Notes

If an RTOS is used, blocking and bus lock is available.

- If blocking of an I2C bus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.

Bus Initialization

The I2C communications interface expects a bus instance to be opened before opening any specific I2C comms device. The communications interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate I2C Master Interface open call.

Examples

Basic Example

This is a basic example of minimal use of I2C communications implementation in an application.

```
void rm_comms_i2c_basic_example (void)
```

```
{
 fsp_err_t err = FSP_SUCCESS;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend          =
(rm_comms_i2c_bus_extended_cfg_t *) g_comms_i2c_cfg.p_extend;
 i2c_master_instance_t          * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                            p_extend->p_blocking_semaphore->p_semaphore_name,
                            (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                        p_extend->p_bus_recursive_mutex->p_mutex_name,
                        TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
```

```
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_COMMS_I2C_Open(&g_comms_i2c_ctrl, &g_comms_i2c_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (true)
    {
       g_flag = 0;
 /* Send data to an I2C device. */
 RM_COMMS_I2C_Write(&g_comms_i2c_ctrl, &write_data, 1);
 while (0 == g_flag)
      {
 /* Wait callback */
      }
       g_flag = 0;
 /* Receive data from an I2C device. */
 RM_COMMS_I2C_Read(&g_comms_i2c_ctrl, &read_data, 1);
 while (0 == g_flag)
      {
 /* Wait callback */
      }
    }
}
```

Data Structures

struct	rm_comms_i2c_instance_ctrl_t

Data Structure Documentation

◆ rm_comms_i2c_instance_ctrl_t

struct rm_comms_i2c_instance_ctrl_t
Communications middleware control structure.
Data Fields

rm_comms_cfg_t const *	p_cfg
	middleware configuration.
rm_comms_i2c_bus_extended_cfg_t *	p_bus
	Bus using this device;.
void *	p_lower_level_cfg
	Used to reconfigure I2C driver.
uint32_t	open
	Open flag.
uint32_t	transfer_data_bytes
	Size of transfer data.
uint8_t *	p_transfer_data
	Pointer to transfer data buffer.
void const *	p_context
	Pointer to the user-provided context.

Function Documentation

◆ RM_COMMS_I2C_Open()

fsp_err_t RM_COMMS_I2C_Open (rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_cfg_t const *const *p_cfg*)

Opens and configures the Communications Middle module. Implements rm_comms_api_t::open.

Example:

```
err = RM_COMMS_I2C_Open(&g_comms_i2c_ctrl, &g_comms_i2c_cfg);
```

Return values

FSP_SUCCESS	Communications Middle module successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_COMMS_BUS_NOT_OPEN	I2C driver is not open.

◆ RM_COMMS_I2C_Close()

fsp_err_t RM_COMMS_I2C_Close (rm_comms_ctrl_t *const *p_api_ctrl*)

Disables specified Communications Middle module. Implements rm_comms_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_COMMS_I2C_Read()

fsp_err_t RM_COMMS_I2C_Read (rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_dest*, uint32_t const *bytes*)

Performs a read from the I2C device. Implements rm_comms_api_t::read.

Return values

FSP_SUCCESS	Successfully data decoded.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_COMMS_I2C_Write()

fsp_err_t RM_COMMS_I2C_Write (rm_comms_ctrl_t *const *p_api_ctrl*, uint8_t *const *p_src*, uint32_t const *bytes*)
Performs a write from the I2C device. Implements rm_comms_api_t::write. **Return values**

FSP_SUCCESS	Successfully writing data .
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_COMMS_I2C_WriteRead()

fsp_err_t RM_COMMS_I2C_WriteRead (rm_comms_ctrl_t *const *p_api_ctrl*, rm_comms_write_read_params_t const *write_read_params*)
Performs a write to, then a read from the I2C device. Implements rm_comms_api_t::writeRead. **Return values**

FSP_SUCCESS	Successfully data decoded.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

4.2.84 SEGGER emWin Port (rm_emwin_port)

Modules

SEGGER emWin port for RA MCUs.

Overview

The SEGGER emWin RA Port module provides the configuration and hardware acceleration support necessary for use of emWin on RA products. The port provides full integration with the graphics peripherals (GLCDC, DRW and JPEG) as well as FreeRTOS.

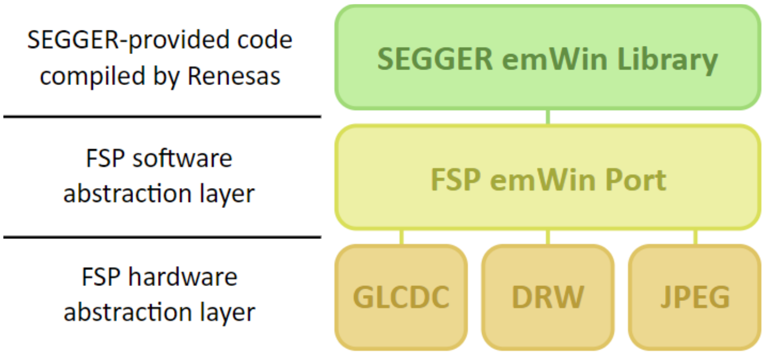


Figure 203: SEGGER emWin FSP Port Block Diagram

Note

> *This port layer primarily enables hardware acceleration and background handling of many display operations and does not contain code intended to be directly called by the user. Please consult the SEGGER emWin User Guide (UM03001) for details on how to use emWin in your project.*

Hardware Acceleration

The following functions are currently performed with hardware acceleration:

- DRW Engine (r_drw)
  - Drawing bitmaps (ARGB8888 and RGB565)
  - 4bpp font rendering
  - Rectangle fill
  - Line and shape drawing
  - Anti-aliased operations
    - Circle stroke and fill
    - Polygon stroke and fill
    - Lines and arcs
- JPEG Codec (r_jpeg)
  - JPEG decoding
- Graphics LCD Controller (r_glcdc)
  - Brightness, contrast and gamma correction
  - Pixel format conversion (framebuffer to LCD)

Configuration

Build Time Configurations for rm_emwin_port

The following build time configurations are defined in fsp_cfg/rm_emwin_port_cfg.h:

Configuration	Options	Default	Description
Memory Allocation > GUI Heap Size	Value must be a non-negative integer	32768	Set the size of the heap to be allocated for use exclusively by emWin.
Memory Allocation > Section for GUI Heap	Manual Entry	.noinit	Specify the section in which to allocate the GUI heap. When Arm Compiler 6 is used to

			place this memory in on-chip SRAM, the section name must be .bss or start with .bss. to avoid consuming unnecessary ROM space.
Memory Allocation > Maximum Layers	Value must be a non-negative integer	16	Set the maximum number of available display layers in emWin. This setting is not related to GLCDC Layer 1 or 2.
Memory Allocation > AA Font Conversion Buffer Size	Value must be a non-negative integer	400	Set the size of the conversion buffer for anti-aliased font glyphs. This should be set to the size (in bytes) of the largest AA character to be rendered.
Configuration > Multi-thread Support	• Enabled • Disabled	Enabled	Enable or disable multithreading support.
Configuration > Number of emWin-supported threads	Manual Entry	5	If multithreading support is enabled this configuration specifies the number of different tasks that can call emWin functions.
Configuration > Touch Panel Support	• Enabled • Disabled	Enabled	Enable or disable touch panel support.
Configuration > Mouse Support	• Enabled • Disabled	Disabled	Enable or disable support for mouse input.
Configuration > Memory Devices	• Enabled • Disabled	Enabled	Enable or disable support for memory devices, which allow the user to allocate their own memory in the GUI heap.
Configuration > Text Rotation	• Enabled • Disabled	Enabled	Enable or disable support for displaying rotated text.
Configuration > Window Manager	• Enabled • Disabled	Enabled	Enable or disable the emWin Window Manager (WM).
Configuration >	• Enabled	Disabled	Enable or disable

Bidirectional Text	• Disabled		support for bidirectional text (such as Arabic or Hebrew).
Configuration > Debug Logging Level	• None (0) • Parameter checking only (1) • All checks enabled (2) • Log errors (3) • Log warnings (4) • Log all messages (5)	All checks enabled (2)	Set the debug logging level.
LCD Settings > Wait for Vertical Sync	• Enabled • Disabled	Enabled	When enabled emWin will wait for a vertical sync event each time the display is updated. If an RTOS is used the thread will yield; otherwise each frame will block until Vsync. WARNING: Disabling vertical sync will result in tearing. It is recommended to always leave this setting Enabled if an RTOS is used.
JPEG Decoding > General > Input Alignment	• 8-byte aligned (faster) • Unaligned (slower)	8-byte aligned (faster)	Setting this option to 8-bit alignment can allow the hardware JPEG Codec to directly read JPEG data. This speeds up JPEG decoding operations and reduces RAM overhead, but all JPEG images must reside on an 8-byte boundary. When this option is enabled the input buffer is not allocated.
JPEG Decoding > General > Double-Buffer Output	• Enabled • Disabled	Disabled	Enable this option to configure JPEG decoding operations to use a double-buffered output pipeline. This allows the JPEG to be rendered to the display

			at the same time as decoding at the cost of additional RAM usage. Enabling this option automatically allocates double the output buffer size.
JPEG Decoding > General > Error Timeout	Value must be a non-negative integer	50	Set the timeout for JPEG decoding operations (in RTOS ticks) in the event of a decode error.
JPEG Decoding > Buffers > Input Buffer Size	Value must be a non-negative integer	0x1000	Set the size of the JPEG decode input buffer (in bytes). This buffer is used to ensure 8-byte alignment of input data. Specifying a size smaller than the size of the JPEG to decode will use additional interrupts to stream data in during the decoding process.
JPEG Decoding > Buffers > Output Buffer Size	Value must be a non-negative integer	0x3C00	Set the size of the JPEG decode output buffer (in bytes). An output buffer smaller than the size of a decoded image will use additional interrupts to stream the data into a framebuffer. Unless you are sure of the subsampling used in and the size of the input JPEG images it is recommended to allocate at least 16 framebuffer lines of memory.
JPEG Decoding > Buffers > Section for Buffers	Manual Entry	.noinit	Specify the section in which to allocate the JPEG work buffers. When Arm Compiler 6 is used to place this memory in on-chip SRAM, the section name must be .bss or start with .bss. to avoid

consuming unnecessary ROM space.

Hardware Configuration

No clocks or pins are directly required by this module. Please consult the submodules' documentation for their requirements.

Usage Notes

Getting Started

To get started with emWin in an RA project the following must be performed:

1. Open the RA Configuration editor for your project
2. Add emWin to your project in the Stacks view by clicking **New Stack -> SEGGER -> SEGGER emWin**
3. Ensure the configuration options for emWin are set as necessary for your application
4. Set the proporties for the GLCDC module to match the timing and memory requirements of your display panel
5. Set the JPEG decode color depth to the desired value (if applicable)
6. Ensure interrupts on all modules are enabled:
    - GLCDC Vertical Position (Vpos) Interrupt
    - DRW Interrupt (if applicable)
    - JPEG Encode/Decode and Data Transfer Interrupts (if applicable)
7. Confirm stack and heap are configured as needed
    - When starting development a minimum stack of 0x1000 (4K) and heap of 0x4000 (16K) are recommended until actual usage can be characterized
8. Click Generate Project Content to save and commit configuration changes

At this point the project is now ready to build with emWin. Please refer to the SEGGER emWin User Guide (UM03001) as well as demo and sample code for details on how to create a GUI application.

Using Hardware Acceleration

In most cases there is no need to perform additional configuration to ensure the DRW Engine is used. However, there are some guidelines that should be followed depending on the item in question:

- Bitmaps:
    - ARGB8888, RGB888 and RGB565 bitmaps require no additional settings.
- Anti-aliased shapes:
    - Anti-aliased lines, circles, polygons, polygon outlines and arcs are rendered with the DRW Engine.
- Anti-aliased (4bpp) fonts:
    - Set the text mode to GUI_TM_TRANS or create the relevant widget with WM_CF_HASTRANS set.
    - Ensure the "AA Font Conversion Buffer Size" configuration option is set to a size equal to or greater than the size (in bytes) of the largest glyph.
- 8bpp palletized images:
    - When creating these images ensure transparency is not enabled as the SEGGER method for this is not compatible with the DRW Engine.
- RLE-encoded images:
    - Hardware acceleration is not available for SEGGER's RLE format.
- JPEG images:

- Align any user-declared JPEG data to an 8-byte boundary. If 8-byte alignment cannot be guaranteed disable the **JPEG Decoding -> General -> Input Alignment** option in the RA Configuration.

Multi-thread Support

When the "Multi-thread Support" configuration is enabled, emWin can be called from multiple threads. This comes with advantages and disadvantages:

Advantages:

- High flexibility in development of applications
- Threads can pend and post on emWin events

Disadvantages:

- Slightly higher RAM/ROM use
- Large GUI projects can become difficult to debug

Note

Multi-thread support is independent of RTOS support. RTOS support is managed internally and cannot be manually configured.

Limitations

Developers should be aware of the following limitations when using SEGGER emWin with FSP:

- Hardware acceleration is not available when using color modes lower than 16 bits.
- Hardware acceleration is not available for SEGGER's RLE image format.
- Rotated screen modes are not supported.

Examples

Basic Example

This is a basic example demonstrating a very simple emWin application. The screen is cleared to white and "Hello World!" is printed in the center.

Note

emWin manages the GLCDC, DRW and JPEG Codec submodules internally; they do not need to be opened directly.

```
#include "DIALOG.h"

#define COLOR_WHITE 0x00FFFFFFU

#define COLOR_BLACK 0x00000000U

#define GUI_DRAW_DELAY 100

static void _cbMain (WM_MESSAGE * pMsg)

{

    GUI_RECT Rect;

 switch (pMsg->MsgId)
```

```
    {
 case WM_CREATE:
      {
 break;
       }
 case WM_PAINT:
      {
 /* Clear background to white */
            GUI_SetBkColor(COLOR_WHITE);
            GUI_Clear();
 /* Draw "Hello World!" in black in the center */
            WM_GetClientRect(&Rect);
            GUI_SetColor(COLOR_BLACK);
            GUI_DispStringInRect("Hello World!", &Rect, GUI_TA_VCENTER |
GUI_TA_HCENTER);
 break;
       }
 default:
      {
            WM_DefaultProc(pMsg);
 break;
       }
    }
}
void emWinTask (void)
{
    int32_t xSize;
    int32_t ySize;
 /* Initialize emWin */
    GUI_Init();
 /* Get screen dimensions */
    xSize = LCD_GetXSize();
    ySize = LCD_GetYSize();
 /* Create main window */
```

```
   WM_CreateWindowAsChild(0, 0, xSize, ySize, WM_HBKWIN, WM_CF_SHOW, _cbMain, 0);
/* Enter main drawing loop */
while (1)
   {
      GUI_Delay(GUI_DRAW_DELAY);
   }
}
```

Note

For further example code please consult SEGGER emWin documentation, which can be downloaded here, as well as the Quick Start Guide and example project(s) provided with your Evaluation Kit (if applicable).

4.2.85 Azure RTOS FileX Block Media I/O Driver (rm_filex_block_media)

Modules

Functions

fsp_err_t	RM_FILEX_BLOCK_MEDIA_Open (rm_filex_block_media_ctrl_t *const p_ctrl, rm_filex_block_media_cfg_t const *const p_cfg)
fsp_err_t	RM_FILEX_BLOCK_MEDIA_Close (rm_filex_block_media_ctrl_t *const p_ctrl)
void	RM_FILEX_BLOCK_MEDIA_BlockDriver (FX_MEDIA *p_fx_media) Access Block Media device functions open, close, read, write and control. More...

Detailed Description

Middleware for the Azure RTOS FileX File System control using Block Media on RA MCUs.

Overview

This module provides the hardware port layer for FileX file system. After initializing this module, refer to the FileX API reference to use the file system: https://docs.microsoft.com/en-us/azure/rtos/filex/

Features

The FileX Block Media module supports the following features:

- Callbacks for insertion and removal for removable devices.

- ThreadX is typically required for FileX. To use FileX without ThreadX FX_STANDALONE_ENABLE must be defined.
- Unless FX_SINGLE_THREAD or FX_STANDALONE_ENABLE are defined, all FileX operations are thread safe.

Configuration

Build Time Configurations for rm_filex_block_media

The following build time configurations are defined in fsp_cfg/middleware/rm_filex_block_media_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	Selects if code for parameter checking is to be included in the build.

Configurations for Storage > FileX I/O (rm_filex_block_media)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_filex_block_media_0	Module name.
Callback	Name must be a valid C symbol	g_rm_filex_block_media_0_callback	A user callback function can be provided. If this callback function is provided, it will be called when media is inserted or removed. It will also be called during operations by the lower level block media as a way for the user to provide their desired waiting functionality.
Partition Number	• 0 • 1 • 2 • 3	0	The partition to use for partitioned media. This partition will only be used if a Master Boot Record with partition table exists at block 0 of the media, otherwise the FileX FAT boot record should exist or be formatted to block 0.

Build Time Configurations for fx

The following build time configurations are defined in fsp_cfg/azure/fx/fx_user.h:

Configuration	Options	Default	Description
Common > Max Long Name Len	Value must be greater than or equal to 13 and less than or equal to 256, or empty		Specifies the maximum file name size for FileX. If left blank the default value is 256. Legal values range between 13 and 256.
Common > Max Last Name Len	Value must be greater than or equal to 13 and less than or equal to 256, or empty		This value defines the maximum file name length, which includes full path name. If left blank the default value is 256. Legal values range between 13 and 256.
Common > Max Sector Cache	Value must be greater than 0 or empty		Specifies the maximum number of logical sectors that can be cached by FileX. The actual number of sectors that can be cached is lesser of this constant and how many sectors can fit in the amount of memory supplied at fx_media_open. The default value if left blank is 256. All values must be a power of 2.
Common > Fat Map Size	Value must be greater than 0 or empty		Specifies the number of sectors that can be represented in the FAT update map. The default value if left blank is 256. Larger values help reduce unneeded updates of secondary FAT sectors.
Common > Max Fat Cache	Value must be greater than 0 or empty		Specifies the number of entries in the internal FAT cache. The default value if left blank is 16. All values must be a power of 2.
Threading > Update Rate (Seconds)	Value must be greater than 0 or empty		Specifies rate at which system time in FileX is adjusted. Default value if left blank is 10, specifying that the FileX system time is

			updated every 10 seconds.
Threading > No Timer	• Enabled • Disabled (default)	Disabled (default)	Eliminates the ThreadX timer setup to update the FileX system time and date. Doing so causes default time and date to be placed on all file operations.
Threading > Single Thread	• Enabled • Disabled (default)	Disabled (default)	Eliminates ThreadX protection logic from the FileX source. It should be used if FileX is being used only from one thread.
Threading > Standalone	• Enabled • Disabled (default)	Disabled (default)	Enables FileX to be used in standalone mode (without Azure RTOS).
Extra Features > Don't Update Open Files	• Enabled • Disabled (default)	Disabled (default)	When enabled, FileX does not update already opened files.
Extra Features > Media Search Cache	• Enabled (default) • Disabled	Enabled (default)	When disabled, the file search cache optimization is disabled.
Extra Features > Direct Data Read Cache Fill	• Enabled (default) • Disabled	Enabled (default)	When disabled, the direct read sector update of cache is disabled.
Extra Features > Media Statistics	• Enabled (default) • Disabled	Enabled (default)	When disabled, gathering of media statistics is disabled.
Extra Features > Single Open Legacy	• Enabled • Disabled (default)	Disabled (default)	When enabled, legacy single open logic for the same file is enabled.
Extra Features > Rename Path Inherit	• Enabled • Disabled (default)	Disabled (default)	When enabled, renaming inherits path information.
Extra Features > No Local Path	• Enabled • Disabled (default)	Disabled (default)	When enabled, removes local path logic from FileX, resulting in smaller code size.
Extra Features > 64-bit LBA	• Enabled • Disabled (default)	Disabled (default)	When enabled, 64-bits sector addresses are used in I/O driver.

Extra Features > Cache	• Enabled (default) • Disabled	Enabled (default)	Enables or disables the cache, default is enabled.
Extra Features > File Close	• Enabled (default) • Disabled	Enabled (default)	Enables or disables file close, default is enabled.
Extra Features > Fast Close	• Enabled (default) • Disabled	Enabled (default)	Enables or disables fast open, default is enabled.
Extra Features > Force Memory Operation	• Enabled (default) • Disabled	Enabled (default)	Enables or disables force memory operation, default is enabled.
Extra Features > Build Options	• Enabled (default) • Disabled	Enabled (default)	Enables or disables build options, default is enabled.
Extra Features > One Line Function	• Enabled (default) • Disabled	Enabled (default)	Enables or disables one line function, default is enabled.
Extra Features > FAT Entry Refresh	• Enabled (default) • Disabled	Enabled (default)	Enables or disables FAT entry refresh, default is enabled.
Extra Features > Consecutive Detect	• Enabled (default) • Disabled	Enabled (default)	Enables or disables consecutive detect, default is enabled.
Extra Features > Enable exFAT	• Enabled • Disabled (default)	Disabled (default)	Enables exFAT support in FileX.
Fault Tolerant > Fault Tolerant Service	• Enabled • Disabled (default)	Disabled (default)	When enabled, enables the FileX Fault Tolerant Module. Enabling Fault Tolerant automatically defines the symbol FX_FAULT_TOLERANT and FX_FAULT_TOLERA NT_DATA.
Fault Tolerant > Fault Tolerant Data	• Enabled • Disabled (default)	Disabled (default)	When enabled, FileX immediately passes all file data write requests to the media's driver. This potentially decreases performance, but helps limit lost file data. Note that enabling this feature does not automatically enable FileX Fault Tolerant Module, which should

			be enabled separately.
Fault Tolerant > Fault Tolerant	• Enabled • Disabled (default)	Disabled (default)	When enabled, FileX immediately passes write requests of all system sectors (boot, FAT, and directory sectors) to the media's driver. This potentially decreases performance, but helps limit corruption to lost clusters. Note that enabling this feature does not automatically enable FileX Fault Tolerant Module, which should be enabled separately.
Fault Tolerant > Fault Tolerant Boot Index	Value must be greater than or equal to 116 and less than or equal to 119		Defines byte offset in the boot sector where the cluster for the fault tolerant log is. By default if left blank this value is 116. This field takes 4 bytes. Bytes 116 through 119 are chosen because they are marked as reserved by FAT 12/16/32/exFAT specification.
Error Checking	• Enabled (default) • Disabled	Enabled (default)	

Configurations for Storage > Azure RTOS FileX on Block Media

This module can be added to the Stacks tab via New Stack > Storage > Azure RTOS FileX on Block Media.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_fx_media0	Symbol used for media_ptr parameter in FileX APIs
Volume Name	Name must be a valid C symbol	Volume 1	Volume name string, which is a maximum of 11 characters.
Number of FATs	Number of FATs must be an integer greater than 0	1	Number of FATs in the media. The minimal value is 1 for the primary FAT. Values

			greater than 1 result in additional FAT copies being maintained at run-time.
Directory Entries	Number of Directory Entries must be an integer greater than 0	256	Number of directory entries in the root directory.
Hidden Sectors	Number of Hidden Sectors must be an integer	0	Number of sectors hidden before this media's boot sector. If using media formatted with multiple partitions this number should correspond to the starting block number for the desired partition.
Total Sectors	Total Sectors must be an integer greater than 0	3751936	Total number of sectors in the media. When using a Renesas provided block media implementation, total sectors can be fetched by the infoGet from the block media API. Any removable media must be inserted and initialized first to retrieve this info.
Bytes per Sector	Bytes per Sector must be multiple of 32	512	Number of bytes per sector, which is typically 512. FileX requires this to be a multiple of 32. When using a Renesas provided block media implementation, bytes per sector can be fetched by the infoGet from the block media API. Any removable media must be inserted and initialized first to retrieve this info.
Sectors per Cluster	Sectors per Cluster must be an integer greater than 0	1	Number of sectors in each cluster. The cluster is the minimum allocation unit in a FAT file system.
Volume Serial Number (exFAT only)	Volume Serial Number must be an integer	12345	Serial number to be used for this volume.

	greater than 0		exFAT only.
Boundary Unit (exFAT only)	Boundary unit must be an integer greater than 0	128	Physical data area alignment size, in number of sectors. exFAT only.
Working media memory size	Memory size must be an integer greater than or equal to the size of one sector	512	Memory allocated for file system. Memory size must be an integer greater than or equal to the size of one sector.

Usage Notes

Pending during Read/Write

The FileX Block Media driver provides a number of events in the user callback to handle waiting or pending while it is doing blocking operations. The events received in the callback will differ depending on the lower level block media driver in use.

If the lower level block media driver is rm_block_media_spi (SPI blocks on read/write operations):

- The user will receive RM_BLOCK_MEDIA_EVENT_POLL_STATUS in the user callback while the lower level driver is polling for the read/write operation to be complete. The user can choose to do a thread sleep or software delay upon receiving this event in the callback.
- Once the operation is complete no other callbacks will be received.

If the lower level block media driver is rm_block_media_sdmmc (SDMMC is interrupt based, the FileX Block Media driver will still block while waiting for interrupts from SDMMC):

- The user will receive RM_BLOCK_MEDIA_EVENT_WAIT in the user callback when the FileX Block Media driver begins waiting for an interrupt event from SDMMC. This is sent from a thread context. The user can choose to pend on a semaphore, sleep the thread, or do a software delay upon receiving this event in the callback. The FileX Block Media driver thread will block until an interrupt is received.
- Once an SDMMC interrupt is received the user will receive RM_BLOCK_MEDIA_EVENT_WAIT_END in the user callback. This is sent from an interrupt context. The user can choose to give a semaphore on this event or do nothing.
- If SDMMC is busy on a long erase after receiving the interrupt the FileX Block Media driver will send RM_BLOCK_MEDIA_EVENT_POLL_STATUS to the user callback and proceed to do a blocking poll on SDMMC status. The user can choose to do a thread sleep or software delay upon receiving this event in the callback. This event will not be received by the user on typical operations by FileX.

Partitioned Media

When using fx_format to format a partition the number of hidden sectors should match the starting block number of the partition and the total number of sectors should be equal to the number of sectors in the partition.

Unused User Callback Events

Certain events are defined in rm_block_media_event_t but not returned by the FileX Block Media user callback:

- RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE: This event is handled internally and operation success is indicated by FileX API calls returning FX_SUCCESS.
- RM_BLOCK_MEDIA_EVENT_ERROR: This event is handled internally and operation failure will be indicated by an error return code from FileX API calls.

Examples

Basic Example

This is a basic example of FileX Block Media in an application.

```
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_FILE_NAME "TEST_FILE.txt"
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_BUFFER_SIZE_BYTES (10240)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_PARTITION_NUMBER (0)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_BLOCK_SIZE (512)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_DIRECTORY_ENTRIES (128)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_FATS (1)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_HIDDEN_SECTORS (0)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_TOTAL_SECTORS (1073741824)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTOR_SIZE (512)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTORS_PER_CLUSTER (1)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_HEADS (1)
#define RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTORS_PER_TRACK (1)
extern rm_filex_block_media_instance_t g_filex_block_media0;
extern rm_filex_block_media_instance_ctrl_t g_filex_block_media0_ctrl;
extern rm_filex_block_media_cfg_t g_filex_block_media0_cfg;
extern FX_MEDIA g_fx_media0;
extern uint8_t g_fx_media0_memory[RM_FILEX_BLOCK_MEDIA_EXAMPLE_BLOCK_SIZE];
extern uint8_t g_file_data[RM_FILEX_BLOCK_MEDIA_EXAMPLE_BUFFER_SIZE_BYTES];
extern uint8_t g_read_buffer[RM_FILEX_BLOCK_MEDIA_EXAMPLE_BUFFER_SIZE_BYTES];
void rm_filex_block_media_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FILEX_BLOCK_MEDIA_Open(&g_filex_block_media0_ctrl,
&g_filex_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
```

```
 /* Initialize FileX */
    fx_system_initialize();
 /* Open the media. If the media is removable, it must be inserted before calling
  * fx_media_open. This assumes the disk is already partitioned and formatted. */
    UINT fx_err = fx_media_open(&g_fx_media0,
 "filex_example_media",
 RM_FILEX_BLOCK_MEDIA_BlockDriver,
                               &g_filex_block_media0,
                               g_fx_media0_memory,
 sizeof(g_fx_media0_memory));
    handle_fx_error(fx_err);
 /* Create a file */
    fx_err = fx_file_create(&g_fx_media0, RM_FILEX_BLOCK_MEDIA_EXAMPLE_FILE_NAME);
    handle_fx_error(fx_err);
 /* Open source file for writing. */
    FX_FILE sourceFile;
    fx_err = fx_file_open(&g_fx_media0, &sourceFile,
RM_FILEX_BLOCK_MEDIA_EXAMPLE_FILE_NAME, FX_OPEN_FOR_WRITE);
    handle_fx_error(fx_err);
 /* Write file data. */
    fx_err = fx_file_write(&sourceFile, g_file_data, sizeof(g_file_data));
    handle_fx_error(fx_err);
 /* Close the file. */
    fx_err = fx_file_close(&sourceFile);
    handle_fx_error(fx_err);
 /* Open the source file in read mode. */
    fx_err = fx_file_open(&g_fx_media0, &sourceFile,
RM_FILEX_BLOCK_MEDIA_EXAMPLE_FILE_NAME, FX_OPEN_FOR_READ);
    handle_fx_error(fx_err);
 /* Read file data. */
    ULONG actual_size_read;
    fx_err = fx_file_read(&sourceFile, g_read_buffer, sizeof(g_file_data),
&actual_size_read);
    handle_fx_error(fx_err);
```

```
    assert(sizeof(g_file_data) == actual_size_read);
 /* Close the file. */
    fx_err = fx_file_close(&sourceFile);
    handle_fx_error(fx_err);
 /* Verify the file data read matches the file written. */
    assert(0U == memcmp(g_file_data, g_read_buffer, sizeof(g_file_data)));
 /* Close the Media */
    fx_err = fx_media_close(&g_fx_media0);
    handle_fx_error(fx_err);
}
```

Format Example

This shows how to partition and format a disk if it is not already partitioned and formatted.

```
void rm_filex_block_media_format_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FILEX_BLOCK_MEDIA_Open(&g_filex_block_media0_ctrl,
&g_filex_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Format the media */
    UINT fx_err =
fx_media_format(&g_fx_media0,                                      // Pointer to
FileX media control block.
 RM_FILEX_BLOCK_MEDIA_BlockDriver,                   // Driver entry
                                 &g_filex_block_media0, // Pointer to Block Media
Driver
                                 g_fx_media0_memory, // Media buffer pointer
 sizeof(g_fx_media0_memory),                         // Media buffer size
 "EXAMPLE_VOLUME",                                  // Volume Name
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_FATS,
// Number of FATs
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_DIRECTORY_ENTRIES,
```

```
// Directory Entries
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_HIDDEN_SECTORS,
// Hidden sectors

RM_FILEX_BLOCK_MEDIA_EXAMPLE_TOTAL_SECTORS,         // Total sectors
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTOR_SIZE,
// Sector size

RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTORS_PER_CLUSTER,   // Sectors per cluster
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_HEADS,
// Heads
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTORS_PER_TRACK);
// Sectors per track
    handle_fx_error(fx_err);
}
```

Callback Pend Example

This shows how to use the I/O driver callback with ThreadX in order to wait/pend for operations to complete.

```
TX_SEMAPHORE g_operation_wait_semaphore;
/* Callback called by FileX block media I/O driver needs to pend on operation. */
void rm_filex_block_media_test_callback_pend (rm_filex_block_media_callback_args_t *
p_args)
{
 if (p_args->event & RM_BLOCK_MEDIA_EVENT_WAIT)
    {
 /* Interrupt has not happened for operation, get semaphore to wait for it. This will
be called from the FileX I/O driver thread. */
       tx_semaphore_get(&g_operation_wait_semaphore, TX_WAIT_FOREVER);
    }
 if (p_args->event & RM_BLOCK_MEDIA_EVENT_WAIT_END)
    {
 /* Interrupt has occurred for operation, post semaphore so that wait will end. This
```

```
will be called from an interrupt context. */
       tx_semaphore_put(&g_operation_wait_semaphore);
    }
 if (p_args->event & RM_BLOCK_MEDIA_EVENT_POLL_STATUS)
    {
 /* Interrupt has been received from block media device but operation is still
ongoing. The FileX I/O driver will wait on the driver busy status.
  * This event can be used to put the thread to sleep while waiting. This will be
called from the FileX I/O driver thread. */
       tx_thread_sleep(1);
    }
}
void rm_filex_block_media_callback_pend_example (void)
{
 /* Create semaphore for driver use */
    tx_semaphore_create(&g_operation_wait_semaphore, "operation_wait_semaphore", 0);
 /* Open media driver.*/
 fsp_err_t err = RM_FILEX_BLOCK_MEDIA_Open(&g_filex_block_media0_ctrl,
&g_filex_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Format the media */
    UINT fx_err =
fx_media_format(&g_fx_media0,                                      // Pointer to
FileX media control block.
 RM_FILEX_BLOCK_MEDIA_BlockDriver,                  // Driver entry
                                 &g_filex_block_media0, // Pointer to Block Media
Driver
                                 g_fx_media0_memory, // Media buffer pointer
 sizeof(g_fx_media0_memory),                        // Media buffer size
 "EXAMPLE_VOLUME",                                 // Volume Name
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_FATS,
// Number of FATs
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_DIRECTORY_ENTRIES,
```

```
// Directory Entries
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_HIDDEN_SECTORS,
// Hidden sectors

RM_FILEX_BLOCK_MEDIA_EXAMPLE_TOTAL_SECTORS,         // Total sectors
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTOR_SIZE,
// Sector size

RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTORS_PER_CLUSTER,   // Sectors per cluster
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_NUM_HEADS,
// Heads
                                  RM_FILEX_BLOCK_MEDIA_EXAMPLE_SECTORS_PER_TRACK);
// Sectors per track
    handle_fx_error(fx_err);
}
```

Media Insertion Example

This shows how to use the callback to wait for media insertion.

```
volatile uint32_t g_rm_filex_block_media_insertion_events = 0;
volatile uint32_t g_rm_filex_block_media_removal_events = 0;
/* Callback called by media driver when a removable device is inserted or removed. */
void rm_filex_block_media_test_callback (rm_filex_block_media_callback_args_t *
p_args)
{
 if (p_args->event & RM_BLOCK_MEDIA_EVENT_MEDIA_INSERTED)
    {
       g_rm_filex_block_media_insertion_events++;
    }
 if (p_args->event & RM_BLOCK_MEDIA_EVENT_MEDIA_REMOVED)
    {
       g_rm_filex_block_media_removal_events++;
    }
}
```

```
void rm_filex_block_media_media_insertion_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FILEX_BLOCK_MEDIA_Open(&g_filex_block_media0_ctrl,
&g_filex_block_media0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait for media insertion. */
 while (0U == g_rm_filex_block_media_insertion_events)
    {
 /* Wait for media insertion. */
    }
 /* Open the media. If the media is removable, it must be inserted before calling
  * fx_media_open. This assumes the disk is already partitioned and formatted. */
    UINT fx_err = fx_media_open(&g_fx_media0,
 "filex_example_media",
 RM_FILEX_BLOCK_MEDIA_BlockDriver,
                               &g_filex_block_media0,
                               g_fx_media0_memory,
 sizeof(g_fx_media0_memory));
    handle_fx_error(fx_err);
}
```

Using FileX with Custom Block Media Implementations

When using a Custom Block Media implementation with rm_filex_block_media the custom implementation must call rm_filex_block_media_memory_callback upon the completion of a read/write operation. This callback should be called with an event of RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE and p_context of rm_filex_block_media_instance_ctrl_t *. The following example shows how this should be done in the context of a demo RAM block media read function.

```
#define EXAMPLE_BLOCK_MEDIA_RAM_START_ADDR (0x20004AFE)
#define EXAMPLE_BLOCK_MEDIA_RAM_BLOCK_SIZE_BYTES (512)
/* Example implementation of rm_block_media_api_t::read(), user should define custom
block media RAM implementation. */
fsp_err_t RM_BLOCK_MEDIA_CUSTOM_RAM_Read (rm_block_media_ctrl_t * const p_ctrl,
```

```
                                          uint8_t * const
               p_dest_address,
                                          uint32_t const
block_address,
                                          uint32_t const                num_blocks)
{
 FSP_PARAMETER_NOT_USED(p_ctrl);
    memcpy(p_dest_address,
          (void *) (EXAMPLE_BLOCK_MEDIA_RAM_START_ADDR + (block_address *
EXAMPLE_BLOCK_MEDIA_RAM_BLOCK_SIZE_BYTES)),
          (EXAMPLE_BLOCK_MEDIA_RAM_BLOCK_SIZE_BYTES * num_blocks));
 /* Notify FileX port of operation complete through calling the callback, this is
required for custom block media/FileX port integration */
 rm_block_media_callback_args_t args;
    args.event     = RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE;
    args.p_context = (void *) &g_filex_block_media0_ctrl;
    rm_filex_block_media_memory_callback(&args);
 return FSP_SUCCESS;
}
```

Data Structures

struct	rm_filex_block_media_instance_ctrl_t

Data Structure Documentation

◆ rm_filex_block_media_instance_ctrl_t

struct rm_filex_block_media_instance_ctrl_t
Common macro for FSP header files. There is also a corresponding FSP_FOOTER macro at the end of this file. FileX block media private control block. DO NOT MODIFY. Initialization occurs when RM_FILEX_BLOCK_MEDIA_Open is called.

Function Documentation

◆ RM_FILEX_BLOCK_MEDIA_Open()

fsp_err_t RM_FILEX_BLOCK_MEDIA_Open (rm_filex_block_media_ctrl_t *const *p_ctrl*, rm_filex_block_media_cfg_t const *const *p_cfg*)

The file system relies on the media to be formatted prior to creating directories and files The sector size and sector count will change depending on the media type and size.

The File Allocation Table (FAT) starts after the reserved sectors in the media. The FAT area is basically an array of 12-bit, 16-bit, or 32-bit entries that determine if that cluster is allocated or part of a chain of clusters comprising a subdirectory or a file. The size of each FAT entry is determined by the number of clusters that need to be represented. If the number of clusters (derived from the total sectors divided by the sectors per cluster) is less than 4,086, 12-bit FAT entries are used. If the total number of clusters is greater than 4,086 and less than or equal to 65,525, 16-bit FAT entries are used. Otherwise, if the total number of clusters is greater than 65,525, 32-bit FAT entries are used. Initializes callback and configuration for FileX Block Media interface. Call this before calling any FileX functions.

Implements rm_filex_block_media_api_t::open().

Return values

FSP_SUCCESS	Success.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_ALREADY_OPEN	Module is already open.

Returns
See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_FILEX_BLOCK_MEDIA_Close()

fsp_err_t RM_FILEX_BLOCK_MEDIA_Close (rm_filex_block_media_ctrl_t *const *p_ctrl*)

Closes media device.

Implements rm_filex_block_media_api_t::close().

Return values

FSP_SUCCESS	Media device closed.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Returns
See Common Error Codes or functions called by this function for other possible return codes.

◆ RM_FILEX_BLOCK_MEDIA_BlockDriver()

void RM_FILEX_BLOCK_MEDIA_BlockDriver (FX_MEDIA * *p_fx_media*)

Access Block Media device functions open, close, read, write and control.

The RM_FILEX_BLOCK_MEDIA_BlockDriver function is called from the FileX file system driver and issues requests to a Block Media device through the FSP Block Media Interface. Uses block media driver for accesses.

Parameters

[in,out]	p_fx_media	FileX media control block. All information about each open media device are maintained in the FX_MEDIA data type. The I/O driver communicates the success or failure of the request through the fx_media_driver_status member of FX_MEDIA (p_fx_media->fx_media_driver_status). Possible values are documented in the FileX User Guide.

Return values

None	

Returns

Nothing, but updates FileX media control block.

4.2.86 Azure RTOS FileX LevelX I/O Driver (rm_filex_levelx_nor)

Modules

Functions

void	RM_FILEX_LEVELX_NOR_DeviceDriver (FX_MEDIA *p_fx_media) Access LevelX NOR device functions open, close, read, write and control. More...

Detailed Description

Middleware for the Azure RTOS FileX File System control using LevelX NOR on RA MCUs.

Overview

This module provides the hardware port layer for FileX file system. After initializing this module, refer to the FileX API reference to use the file system: https://docs.microsoft.com/en-us/azure/rtos/filex/

Features

The FileX LevelX NOR module supports the following features:

- ThreadX is typically required for FileX. To use FileX without ThreadX FX_STANDALONE_ENABLE must be defined.
- Unless FX_SINGLE_THREAD or FX_STANDALONE_ENABLE are defined, all FileX operations are thread safe.

Configuration

Build Time Configurations for rm_filex_levelx_nor

The following build time configurations are defined in fsp_cfg/middleware/rm_filex_levelx_nor_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	Selects if code for parameter checking is to be included in the build.

Configurations for Storage > FileX I/O (rm_filex_levelx_nor)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_filex_levelx_nor0	Module name.
Callback	Name must be a valid C symbol	g_rm_filex_levelx_nor_0_callback	A user callback function can be provided. If this callback function is provided, it will be called during operations by the lower level block media as a way for the user to provide their desired waiting functionality.
LevelX NOR Name (String)	Manual Entry	g_rm_filex_levelx_nor_0	String name to be input into LevelX API.

Build Time Configurations for fx

The following build time configurations are defined in fsp_cfg/azure/fx/fx_user.h:

Configuration	Options	Default	Description
Common > Max Long Name Len	Value must be greater than or equal to 13 and less than or equal to		Specifies the maximum file name size for FileX. If left blank the default

	256, or empty		value is 256. Legal values range between 13 and 256.
Common > Max Last Name Len	Value must be greater than or equal to 13 and less than or equal to 256, or empty		This value defines the maximum file name length, which includes full path name. If left blank the default value is 256. Legal values range between 13 and 256.
Common > Max Sector Cache	Value must be greater than 0 or empty		Specifies the maximum number of logical sectors that can be cached by FileX. The actual number of sectors that can be cached is lesser of this constant and how many sectors can fit in the amount of memory supplied at fx_media_open. The default value if left blank is 256. All values must be a power of 2.
Common > Fat Map Size	Value must be greater than 0 or empty		Specifies the number of sectors that can be represented in the FAT update map. The default value if left blank is 256. Larger values help reduce unneeded updates of secondary FAT sectors.
Common > Max Fat Cache	Value must be greater than 0 or empty		Specifies the number of entries in the internal FAT cache. The default value if left blank is 16. All values must be a power of 2.
Threading > Update Rate (Seconds)	Value must be greater than 0 or empty		Specifies rate at which system time in FileX is adjusted. Default value if left blank is 10, specifying that the FileX system time is updated every 10 seconds.
Threading > No Timer	• Enabled • Disabled	Disabled (default)	Eliminates the ThreadX timer setup to update

	(default)		the FileX system time and date. Doing so causes default time and date to be placed on all file operations.
Threading > Single Thread	• Enabled • Disabled (default)	Disabled (default)	Eliminates ThreadX protection logic from the FileX source. It should be used if FileX is being used only from one thread.
Threading > Standalone	• Enabled • Disabled (default)	Disabled (default)	Enables FileX to be used in standalone mode (without Azure RTOS).
Extra Features > Don't Update Open Files	• Enabled • Disabled (default)	Disabled (default)	When enabled, FileX does not update already opened files.
Extra Features > Media Search Cache	• Enabled (default) • Disabled	Enabled (default)	When disabled, the file search cache optimization is disabled.
Extra Features > Direct Data Read Cache Fill	• Enabled (default) • Disabled	Enabled (default)	When disabled, the direct read sector update of cache is disabled.
Extra Features > Media Statistics	• Enabled (default) • Disabled	Enabled (default)	When disabled, gathering of media statistics is disabled.
Extra Features > Single Open Legacy	• Enabled • Disabled (default)	Disabled (default)	When enabled, legacy single open logic for the same file is enabled.
Extra Features > Rename Path Inherit	• Enabled • Disabled (default)	Disabled (default)	When enabled, renaming inherits path information.
Extra Features > No Local Path	• Enabled • Disabled (default)	Disabled (default)	When enabled, removes local path logic from FileX, resulting in smaller code size.
Extra Features > 64-bit LBA	• Enabled • Disabled (default)	Disabled (default)	When enabled, 64-bits sector addresses are used in I/O driver.
Extra Features > Cache	• Enabled (default) • Disabled	Enabled (default)	Enables or disables the cache, default is enabled.

Extra Features > File Close	• Enabled (default) • Disabled	Enabled (default)	Enables or disables file close, default is enabled.
Extra Features > Fast Close	• Enabled (default) • Disabled	Enabled (default)	Enables or disables fast open, default is enabled.
Extra Features > Force Memory Operation	• Enabled (default) • Disabled	Enabled (default)	Enables or disables force memory operation, default is enabled.
Extra Features > Build Options	• Enabled (default) • Disabled	Enabled (default)	Enables or disables build options, default is enabled.
Extra Features > One Line Function	• Enabled (default) • Disabled	Enabled (default)	Enables or disables one line function, default is enabled.
Extra Features > FAT Entry Refresh	• Enabled (default) • Disabled	Enabled (default)	Enables or disables FAT entry refresh, default is enabled.
Extra Features > Consecutive Detect	• Enabled (default) • Disabled	Enabled (default)	Enables or disables consecutive detect, default is enabled.
Extra Features > Enable exFAT	• Enabled • Disabled (default)	Disabled (default)	Enables exFAT support in FileX.
Fault Tolerant > Fault Tolerant Service	• Enabled • Disabled (default)	Disabled (default)	When enabled, enables the FileX Fault Tolerant Module. Enabling Fault Tolerant automatically defines the symbol FX_FAULT_TOLERANT and FX_FAULT_TOLERANT_DATA.
Fault Tolerant > Fault Tolerant Data	• Enabled • Disabled (default)	Disabled (default)	When enabled, FileX immediately passes all file data write requests to the media's driver. This potentially decreases performance, but helps limit lost file data. Note that enabling this feature does not automatically enable FileX Fault Tolerant Module, which should be enabled separately.
Fault Tolerant > Fault Tolerant	• Enabled • Disabled	Disabled (default)	When enabled, FileX immediately passes

	(default)		write requests of all system sectors (boot, FAT, and directory sectors) to the media's driver. This potentially decreases performance, but helps limit corruption to lost clusters. Note that enabling this feature does not automatically enable FileX Fault Tolerant Module, which should be enabled separately.
Fault Tolerant > Fault Tolerant Boot Index	Value must be greater than or equal to 116 and less than or equal to 119		Defines byte offset in the boot sector where the cluster for the fault tolerant log is. By default if left blank this value is 116. This field takes 4 bytes. Bytes 116 through 119 are chosen because they are marked as reserved by FAT 12/16/32/exFAT specification.
Error Checking	• Enabled (default) • Disabled	Enabled (default)	

Configurations for Storage > Azure RTOS FileX on LevelX NOR

This module can be added to the Stacks tab via New Stack > Storage > Azure RTOS FileX on LevelX NOR.

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_fx_media0	Symbol used for media_ptr parameter in FileX APIs
Volume Name	Name must be a valid C symbol	Volume 1	Volume name string, which is a maximum of 11 characters.
Number of FATs	Number of FATs must be an integer greater than 0	1	Number of FATs in the media. The minimal value is 1 for the primary FAT. Values greater than 1 result in additional FAT copies being maintained at run-time.

Directory Entries	Number of Directory Entries must be an integer greater than 0	256	Number of directory entries in the root directory.
Hidden Sectors	Number of Hidden Sectors must be an integer	0	Number of sectors hidden before this media's boot sector. If using media formatted with multiple partitions this number should correspond to the starting block number for the desired partition.
Total Sectors	Total Sectors must be an integer greater than 0	3751936	Total number of sectors in the media. When using a Renesas provided block media implementation, total sectors can be fetched by the infoGet from the block media API. Any removable media must be inserted and initialized first to retrieve this info.
Sectors per Cluster	Sectors per Cluster must be an integer greater than 0	1	Number of sectors in each cluster. The cluster is the minimum allocation unit in a FAT file system.
Volume Serial Number (exFAT only)	Volume Serial Number must be an integer greater than 0	12345	Serial number to be used for this volume. exFAT only.
Boundary Unit (exFAT only)	Boundary unit must be an integer greater than 0	128	Physical data area alignment size, in number of sectors. exFAT only.
Working media memory size	Memory size must be an integer greater than or equal to the size of one sector	512	Memory allocated for file system. Memory size must be an integer greater than or equal to the size of one sector.

Usage Notes

Pending during Write/Erase

If the underlying LevelX NOR driver performs a blocking operation that requires waiting to complete (such as a long write/erase on NOR SPI), a callback can be provided to provide a way to wait with an

OS-specific thread wait. This callback will also pass up block erase events.

Partitioned Media

Partitioned media is not supported directly by the FileX LevelX NOR port.

Examples

Basic Example

This is a basic example of FileX Block Media in an application.

```
#define RM_FILEX_LEVELX_NOR_EXAMPLE_FILE_NAME "TEST_FILE.txt"
#define RM_FILEX_LEVELX_NOR_EXAMPLE_BUFFER_SIZE_BYTES (10240)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_PARTITION_NUMBER (0)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_BLOCK_SIZE (512)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_DIRECTORY_ENTRIES (128)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_FATS (1)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_HIDDEN_SECTORS (0)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_TOTAL_SECTORS (512)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_SECTOR_SIZE (512)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_SECTORS_PER_CLUSTER (1)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_HEADS (1)
#define RM_FILEX_LEVELX_NOR_EXAMPLE_SECTORS_PER_TRACK (1)
extern rm_filex_levelx_nor_instance_t g_filex_levelx_nor0;
extern rm_filex_levelx_nor_instance_ctrl_t g_filex_levelx_nor0_ctrl;
extern rm_filex_levelx_nor_cfg_t g_filex_levelx_nor0_cfg;
extern FX_MEDIA g_fx_media0;
extern uint8_t g_fx_media0_memory[RM_FILEX_LEVELX_NOR_EXAMPLE_BLOCK_SIZE];
extern uint8_t g_file_data[RM_FILEX_LEVELX_NOR_EXAMPLE_BUFFER_SIZE_BYTES];
extern uint8_t g_read_buffer[RM_FILEX_LEVELX_NOR_EXAMPLE_BUFFER_SIZE_BYTES];
void rm_filex_levelx_nor_example (void)
{
 /* Initialize FileX */
    fx_system_initialize();
 /* Initialize LevelX */
    lx_nor_flash_initialize();
 /* Open the media. This assumes the flash is already formatted. */
```

```
    UINT fx_err = fx_media_open(&g_fx_media0,
 "filex_example_media",
 RM_FILEX_LEVELX_NOR_DeviceDriver,
                              &g_filex_levelx_nor0,
                              g_fx_media0_memory,
 sizeof(g_fx_media0_memory));
    handle_fx_error(fx_err);
 /* Create a file */
    fx_err = fx_file_create(&g_fx_media0, RM_FILEX_LEVELX_NOR_EXAMPLE_FILE_NAME);
    handle_fx_error(fx_err);
 /* Open source file for writing. */
    FX_FILE sourceFile;
    fx_err = fx_file_open(&g_fx_media0, &sourceFile,
RM_FILEX_LEVELX_NOR_EXAMPLE_FILE_NAME, FX_OPEN_FOR_WRITE);
    handle_fx_error(fx_err);
 /* Write file data. */
    fx_err = fx_file_write(&sourceFile, g_file_data, sizeof(g_file_data));
    handle_fx_error(fx_err);
 /* Close the file. */
    fx_err = fx_file_close(&sourceFile);
    handle_fx_error(fx_err);
 /* Open the source file in read mode. */
    fx_err = fx_file_open(&g_fx_media0, &sourceFile,
RM_FILEX_LEVELX_NOR_EXAMPLE_FILE_NAME, FX_OPEN_FOR_READ);
    handle_fx_error(fx_err);
 /* Read file data. */
    ULONG actual_size_read;
    fx_err = fx_file_read(&sourceFile, g_read_buffer, sizeof(g_file_data),
&actual_size_read);
    handle_fx_error(fx_err);
    assert(sizeof(g_file_data) == actual_size_read);
 /* Close the file. */
    fx_err = fx_file_close(&sourceFile);
    handle_fx_error(fx_err);
```

```
 /* Verify the file data read matches the file written. */
    assert(0U == memcmp(g_file_data, g_read_buffer, sizeof(g_file_data)));
 /* Close the Media */
    fx_err = fx_media_close(&g_fx_media0);
    handle_fx_error(fx_err);
}
```

Format Example

This shows how to partition and format a disk if it is not already partitioned and formatted.

```
extern rm_levelx_nor_spi_cfg_t g_levelx_nor_spi0_cfg;
#define RM_FILEX_LEVELX_NOR_EXAMPLE_SPI_SECTOR_SIZE (4096)
void rm_filex_levelx_nor_format_example (void)
{
 spi_flash_instance_t * p_spi_flash_instance = (spi_flash_instance_t *)
g_levelx_nor_spi0_cfg.p_lower_lvl;
 spi_flash_status_t    status;
 /* Erase flash prior to usage */
 fsp_err_t err = p_spi_flash_instance->p_api->open(p_spi_flash_instance->p_ctrl,
p_spi_flash_instance->p_cfg);
    assert(FSP_SUCCESS == err);
 for (uint32_t i = g_levelx_nor_spi0_cfg.address_offset;
         i < g_levelx_nor_spi0_cfg.size;
         i += RM_FILEX_LEVELX_NOR_EXAMPLE_SPI_SECTOR_SIZE)
    {
       err = p_spi_flash_instance->p_api->erase(p_spi_flash_instance->p_ctrl,
                                                 (uint8_t *)
g_levelx_nor_spi0_cfg.base_address + i,

RM_FILEX_LEVELX_NOR_EXAMPLE_SPI_SECTOR_SIZE);
      assert(FSP_SUCCESS == err);
       status.write_in_progress = true;
 while (status.write_in_progress)
      {
```

```
            err =
p_spi_flash_instance->p_api->statusGet(p_spi_flash_instance->p_ctrl, &status);
      assert(FSP_SUCCESS == err);
       }
    }
    err = p_spi_flash_instance->p_api->close(p_spi_flash_instance->p_ctrl);
    assert(FSP_SUCCESS == err);
 /* Format the media */
    UINT fx_err = fx_media_format(&g_fx_media0,
// Pointer to FileX media control block.
 RM_FILEX_LEVELX_NOR_DeviceDriver,                   // Driver entry
                                 &g_filex_levelx_nor0, // Pointer to Block Media
Driver
                                 g_fx_media0_memory, // Media buffer pointer
 sizeof(g_fx_media0_memory),                         // Media buffer size
 "EXAMPLE_VOLUME",                                   // Volume Name
                                  RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_FATS,
// Number of FATs
                                  RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_DIRECTORY_ENTRIES,
// Directory Entries
                                  RM_FILEX_LEVELX_NOR_EXAMPLE_HIDDEN_SECTORS,
// Hidden sectors

RM_FILEX_LEVELX_NOR_EXAMPLE_TOTAL_SECTORS,          // Total sectors
                                  RM_FILEX_LEVELX_NOR_EXAMPLE_SECTOR_SIZE,
// Sector size

RM_FILEX_LEVELX_NOR_EXAMPLE_SECTORS_PER_CLUSTER,    // Sectors per cluster
                                  RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_HEADS,
// Heads
                                  RM_FILEX_LEVELX_NOR_EXAMPLE_SECTORS_PER_TRACK);
// Sectors per track
    handle_fx_error(fx_err);
}
```

Callback Wait Example

This shows how to use the I/O driver callback with ThreadX in order to wait for operations to complete.

```
/* Callback called by FileX block media I/O driver needs to wait on operation. */
void rm_filex_levelx_nor_test_callback_wait (rm_filex_levelx_nor_callback_args_t *
p_args)
{
 if (p_args->event & RM_FILEX_LEVELX_NOR_EVENT_BUSY)
    {
 /* Put the thread to sleep while waiting for operation to complete. */
       tx_thread_sleep(1);
    }
}
void rm_filex_levelx_nor_callback_wait_example (void)
{
 /* Format the media */
    UINT fx_err = fx_media_format(&g_fx_media0,
// Pointer to FileX media control block.
 RM_FILEX_LEVELX_NOR_DeviceDriver,                  // Driver entry
                                 &g_filex_levelx_nor0, // Pointer to Block Media
Driver
                                 g_fx_media0_memory, // Media buffer pointer
 sizeof(g_fx_media0_memory),                        // Media buffer size
 "EXAMPLE_VOLUME",                                  // Volume Name
                                  RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_FATS,
// Number of FATs
                                  RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_DIRECTORY_ENTRIES,
// Directory Entries
                                  RM_FILEX_LEVELX_NOR_EXAMPLE_HIDDEN_SECTORS,
// Hidden sectors

RM_FILEX_LEVELX_NOR_EXAMPLE_TOTAL_SECTORS,         // Total sectors
```

```
                                   RM_FILEX_LEVELX_NOR_EXAMPLE_SECTOR_SIZE,
// Sector size

RM_FILEX_LEVELX_NOR_EXAMPLE_SECTORS_PER_CLUSTER,   // Sectors per cluster
                                   RM_FILEX_LEVELX_NOR_EXAMPLE_NUM_HEADS,
// Heads
                                   RM_FILEX_LEVELX_NOR_EXAMPLE_SECTORS_PER_TRACK);
// Sectors per track
    handle_fx_error(fx_err);
}
```

Data Structures

struct	rm_filex_levelx_nor_callback_args_t
struct	rm_filex_levelx_nor_cfg_t
struct	rm_filex_levelx_nor_instance_ctrl_t
struct	rm_filex_levelx_nor_instance_t

Enumerations

enum	rm_filex_levelx_nor_event_t

Data Structure Documentation

◆ rm_filex_levelx_nor_callback_args_t

struct rm_filex_levelx_nor_callback_args_t

Callback function parameter data

Data Fields		
rm_filex_levelx_nor_event_t	event	The event can be used to identify what caused the callback.
void const *	p_context	Placeholder for user data.

◆ rm_filex_levelx_nor_cfg_t

struct rm_filex_levelx_nor_cfg_t

FileX LevelX configuration

Data Fields

UINT(*	nor_driver_initialize)(LX_NOR_FLASH *)

	Pointer to the initialization function.
LX_NOR_FLASH *	p_nor_flash
	NOR Flash instance.
CHAR *	p_nor_flash_name
	NOR Flash instance name.
fsp_err_t(*	close)()
	Pointer to underlying driver close.
void(*	p_callback)(rm_filex_levelx_nor_callback_args_t *p_args)
	Pointer to callback function.
void const *	p_context
	Placeholder for user data.

◆ **rm_filex_levelx_nor_instance_ctrl_t**

struct rm_filex_levelx_nor_instance_ctrl_t		
FileX block media private control block. DO NOT MODIFY. Initialization occurs when RM_FILEX_LEVELX_NOR_Open is called.		
Data Fields		
rm_filex_levelx_nor_cfg_t const *	p_cfg	Pointer to instance configuration.

◆ **rm_filex_levelx_nor_instance_t**

struct rm_filex_levelx_nor_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
rm_filex_levelx_nor_instance_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
rm_filex_levelx_nor_cfg_t *const	p_cfg	Pointer to the configuration

		structure for this instance.

Enumeration Type Documentation

◆ rm_filex_levelx_nor_event_t

enum rm_filex_levelx_nor_event_t

Common macro for FSP header files. There is also a corresponding FSP_FOOTER macro at the end of this file. Options for the callback events.

Enumerator	
RM_FILEX_LEVELX_NOR_EVENT_BUSY	Pending operation, user can define their own wait functionality.

Function Documentation

◆ RM_FILEX_LEVELX_NOR_DeviceDriver()

void RM_FILEX_LEVELX_NOR_DeviceDriver (FX_MEDIA * *p_fx_media*)

Access LevelX NOR device functions open, close, read, write and control.

The RM_FILEX_LEVELX_NOR_DeviceDriver function is called from the FileX file system driver and issues requests to a LevelX NOR device through the LevelX API.

Parameters

[in,out]	p_fx_media	FileX media control block. All information about each open media device are maintained in the FX_MEDIA data type. The I/O driver communicates the success or failure of the request through the fx_media_driver_status member of FX_MEDIA (p_fx_media->fx_media_driver_status). Possible values are documented in the FileX User Guide.

Return values

None	

Returns
Nothing, but updates FileX media control block.

4.2.87 FreeRTOS+FAT Port (rm_freertos_plus_fat)

Modules

Functions

fsp_err_t	RM_FREERTOS_PLUS_FAT_Open (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_cfg_t const *const p_cfg)
fsp_err_t	RM_FREERTOS_PLUS_FAT_MediaInit (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_device_t *const p_device)
fsp_err_t	RM_FREERTOS_PLUS_FAT_DiskInit (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_disk_cfg_t const *const p_disk_cfg, FF_Disk_t *const p_disk)
fsp_err_t	RM_FREERTOS_PLUS_FAT_DiskDeinit (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk)
fsp_err_t	RM_FREERTOS_PLUS_FAT_InfoGet (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk, rm_freertos_plus_fat_info_t *const p_info)
fsp_err_t	RM_FREERTOS_PLUS_FAT_Close (rm_freertos_plus_fat_ctrl_t *const p_ctrl)

Detailed Description

Middleware for the FAT File System control on RA MCUs.

Overview

This module provides the hardware port layer for FreeRTOS+FAT file system. After initializing this module, refer to the FreeRTOS+FAT API reference to use the file system: https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_FAT/index.html

Features

The FreeRTOS+FAT port module supports the following features:

- Callbacks for insertion and removal for removable devices.
- Helper function to initialize FF_Disk_t
- Blocking read and write port functions that use FreeRTOS task notification to pend if FreeRTOS is used
- FreeRTOS is optional

Configuration

Build Time Configurations for rm_freertos_plus_fat

The following build time configurations are defined in fsp_cfg/middleware/rm_freertos_plus_fat_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Storage > FreeRTOS+FAT Port for RA (rm_freertos_plus_fat)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_freertos_plus_fat0	Module name.
Total Number of Sectors	Must be a non-negative integer	31293440	Enter the total number of sectors on the device. If this is not known, update rm_freertos_plus_fat_disk_cfg_t::num_blocks after calling RM_FREERTOS_PLUS_FAT_MediaInit().
Sector Size (bytes)	Must be a power of 2 multiple of 512	512	Select the sector size. Must match the underlying media sector size and at least 512. If this is not known, update rm_freertos_plus_fat_disk_cfg_t::num_blocks after calling RM_FREERTOS_PLUS_FAT_MediaInit().
Cache Size (bytes)	Must be a power of 2 multiple of 512	1024	Select the cache size. Must be a multiple of the sector size and at least 2 times the sector size.
Partition Number	Must be a non-negative integer	0	Select the partition number for this disk.
Callback	Name must be a valid C symbol	NULL	A user callback function can be provided. If this callback function is provided, it will be called when a card is inserted or removed.

Usage Notes

Pending during Read/Write

If the underlying driver supports non-blocking operations, the FreeRTOS+FAT port pends the active

FreeRTOS task during read and write operations so other tasks can run in the background.

If FreeRTOS is not used, the FreeRTOS+FAT port spins in a while loop waiting for read and write operations to complete.

FreeRTOS+FAT without FreeRTOS

To use FreeRTOS+FAT without FreeRTOS, copy FreeRTOSConfigMinimal.h to one of your project's include paths and rename it FreeRTOSConfig.h.

Also, update the Malloc function to malloc and the Free function to free in the Common configurations.

Examples

Basic Example

This is a basic example of FreeRTOS+FAT in an application.

```
#define RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME "TEST_FILE.txt"
#define RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES (10240)
#define RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER (0)
extern rm_freertos_plus_fat_instance_ctrl_t g_freertos_plus_fat0_ctrl;
extern const rm_freertos_plus_fat_cfg_t g_freertos_plus_fat0_cfg;
extern rm_freertos_plus_fat_disk_cfg_t g_rm_freertos_plus_fat_disk_cfg;
extern uint8_t g_file_data[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
extern uint8_t g_read_buffer[RM_FREERTOS_PLUS_FAT_EXAMPLE_BUFFER_SIZE_BYTES];
void rm_freertos_plus_fat_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg.device);
    assert(FSP_SUCCESS == err);
 /* Initialize one disk for each partition used in the application. */
```

```
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    assert(FSP_SUCCESS == err);
 /* Mount each disk. This assumes the disk is already partitioned and formatted. */
    FF_Error_t ff_err = FF_Mount(&disk,
RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
    handle_ff_error(ff_err);
 /* Add the disk to the file system. */
    FF_FS_Add("/", &disk);
 /* Open a source file for writing. */
    FF_FILE * pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "w");
    assert(NULL != pxSourceFile);
 /* Write file data. */
 size_t size_return = ff_fwrite(g_file_data, sizeof(g_file_data), 1, pxSourceFile);
    assert(1 == size_return);
 /* Close the file. */
 int close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Open the source file in read mode. */
    pxSourceFile = ff_fopen((const char *) RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME,
"r");
    assert(NULL != pxSourceFile);
 /* Read file data. */
    size_return = ff_fread(g_read_buffer, sizeof(g_file_data), 1, pxSourceFile);
    assert(1 == size_return);
 /* Close the file. */
    close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Verify the file data read matches the file written. */
    assert(0U == memcmp(g_file_data, g_read_buffer, sizeof(g_file_data)));
}
```

Format Example

This shows how to partition and format a disk if it is not already partitioned and formatted.

```
void rm_freertos_plus_fat_format_example (void)
{
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg.device);
    assert(FSP_SUCCESS == err);
 /* Initialize one disk for each partition used in the application. */
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    assert(FSP_SUCCESS == err);
 /* Try to mount the disk. If the disk is not formatted, mount will fail. */
    FF_Error_t ff_err = FF_Mount(&disk,
RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
 if (FF_isERR((uint32_t) ff_err))
    {
 /* The disk is likely not formatted. Partition and format the disk, then mount
again. */
       FF_PartitionParameters_t partition_params;
       partition_params.ulSectorCount   =
g_rm_freertos_plus_fat_disk_cfg.device.sector_count;
       partition_params.ulHiddenSectors = 1;
       partition_params.ulInterSpace    = 0;
       memset(partition_params.xSizes, 0, sizeof(partition_params.xSizes));
       partition_params.xSizes[RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER] =
```

```
            (BaseType_t) partition_params.ulSectorCount - 1;
       partition_params.xPrimaryCount = 1;
       partition_params.eSizeType    = eSizeIsSectors;
       ff_err = FF_Partition(&disk, &partition_params);
      handle_ff_error(ff_err);
       ff_err = FF_Format(&disk, RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER,
pdFALSE, pdFALSE);
      handle_ff_error(ff_err);
       ff_err = FF_Mount(&disk, RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
      handle_ff_error(ff_err);
    }
}
```

Media Insertion Example

This shows how to use the callback to wait for media insertion.

```
#if 2 == BSP_CFG_RTOS
static EventGroupHandle_t xUSBEventGroupHandle = NULL;
#else
volatile uint32_t g_rm_freertos_plus_fat_insertion_events = 0;
volatile uint32_t g_rm_freertos_plus_fat_removal_events = 0;
#endif
/* Callback called by media driver when a removable device is inserted or removed. */
void rm_freertos_plus_fat_test_callback (rm_freertos_plus_fat_callback_args_t *
p_args)
{
#if 2 == BSP_CFG_RTOS
 /* Post an event if FreeRTOS is available. */
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    xEventGroupSetBitsFromISR(xUSBEventGroupHandle, p_args->event,
&xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
#else
 /* If FreeRTOS is not used, set a global flag. */
```

```
 if (p_args->event & RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED)
    {
       g_rm_freertos_plus_fat_insertion_events++;
    }
 if (p_args->event & RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_REMOVED)
    {
       g_rm_freertos_plus_fat_removal_events++;
    }
#endif
}
void rm_freertos_plus_fat_media_insertion_example (void)
{
#if 2 == BSP_CFG_RTOS
 /* Create event flags if FreeRTOS is used. */
    xUSBEventGroupHandle = xEventGroupCreate();
    TEST_ASSERT_NOT_EQUAL(NULL, xUSBEventGroupHandle);
#endif
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait for media insertion. */
#if 2 == BSP_CFG_RTOS
    EventBits_t xEventGroupValue = xEventGroupWaitBits(xUSBEventGroupHandle,
 RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED,
                                                       pdTRUE,
                                                       pdFALSE,
                                                       portMAX_DELAY);
    assert(RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED ==
          (RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED & xEventGroupValue));
#else
 while (0U == g_rm_freertos_plus_fat_insertion_events)
    {
```

```
 /* Wait for media insertion. */
    }
#endif
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg.device);
    assert(FSP_SUCCESS == err);
 /* Initialize one disk for each partition used in the application. */
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    assert(FSP_SUCCESS == err);
}
```

Media Insertion Example for USB

This shows how to use the callback to read and write to USB media.

```
void rm_freertos_plus_fat_usb_example (void)
{
#if 2 == BSP_CFG_RTOS
 /* Create event flags if FreeRTOS is used. */
    xUSBEventGroupHandle = xEventGroupCreate();
#endif
 /* Open media driver.*/
 fsp_err_t err = RM_FREERTOS_PLUS_FAT_Open(&g_freertos_plus_fat0_ctrl,
&g_freertos_plus_fat0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Wait for the USB media to be attached. */
#if 2 == BSP_CFG_RTOS
    EventBits_t xEventGroupValue = xEventGroupWaitBits(xUSBEventGroupHandle,
 RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED,
```

```
                                                         pdTRUE,
                                                         pdFALSE,
                                                         portMAX_DELAY);
    assert(RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED ==
          (RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED & xEventGroupValue));
#else
 while (0U == g_rm_freertos_plus_fat_insertion_events)
    {
 /* Wait for the USB media to be attached. */
    }
#endif
 /* Initialize the media and the disk. If the media is removable, it must be inserted
before calling
  * RM_FREERTOS_PLUS_FAT_MediaInit. */
    err = RM_FREERTOS_PLUS_FAT_MediaInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg.device);
    assert(FSP_SUCCESS == err);
 /* Initialize one disk for each partition used in the application. */
    FF_Disk_t disk;
    err = RM_FREERTOS_PLUS_FAT_DiskInit(&g_freertos_plus_fat0_ctrl,
&g_rm_freertos_plus_fat_disk_cfg, &disk);
    assert(FSP_SUCCESS == err);
 /* Mount each disk. This assumes the disk is already partitioned and formatted. */
    FF_Error_t ff_err = FF_Mount(&disk,
RM_FREERTOS_PLUS_FAT_EXAMPLE_PARTITION_NUMBER);
    handle_ff_error(ff_err);
 /* Add the disk to the file system. */
    FF_FS_Add("/", &disk);
 /* Open a source file for writing. */
    FF_FILE * pxSourceFile = ff_fopen((const char *)
RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME, "w");
    assert(NULL != pxSourceFile);
 /* Write file data. */
 size_t size_return = ff_fwrite(g_file_data, sizeof(g_file_data), 1, pxSourceFile);
```

```
    assert(1 == size_return);
 /* Close the file. */
 int close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Open the source file in read mode. */
    pxSourceFile = ff_fopen((const char *) RM_FREERTOS_PLUS_FAT_EXAMPLE_FILE_NAME,
"r");
    assert(NULL != pxSourceFile);
 /* Read file data. */
    size_return = ff_fread(g_read_buffer, sizeof(g_file_data), 1, pxSourceFile);
    assert(1 == size_return);
 /* Close the file. */
    close_err = ff_fclose(pxSourceFile);
    assert(0 == close_err);
 /* Verify the file data read matches the file written. */
    assert(0U == memcmp(g_file_data, g_read_buffer, sizeof(g_file_data)));
}
```

Data Structures

struct	rm_freertos_plus_fat_instance_ctrl_t

Data Structure Documentation

◆ rm_freertos_plus_fat_instance_ctrl_t

struct rm_freertos_plus_fat_instance_ctrl_t
FreeRTOS plus FAT private control block. DO NOT MODIFY. Initialization occurs when RM_FREERTOS_PLUS_FAT_Open is called.

Function Documentation

◆ RM_FREERTOS_PLUS_FAT_Open()

fsp_err_t RM_FREERTOS_PLUS_FAT_Open (rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, rm_freertos_plus_fat_cfg_t const *const *p_cfg*)

Initializes lower layer media device.

Implements rm_freertos_plus_fat_api_t::open().

Return values

FSP_SUCCESS	Success.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_OUT_OF_MEMORY	Not enough memory to create semaphore.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- rm_block_media_api_t::open

◆ RM_FREERTOS_PLUS_FAT_MediaInit()

fsp_err_t RM_FREERTOS_PLUS_FAT_MediaInit (rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, rm_freertos_plus_fat_device_t *const *p_device*)

Initializes the media device. This function blocks until all identification and configuration commands are complete.

Implements rm_freertos_plus_fat_api_t::mediaInit().

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module has not been initialized.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- rm_block_media_api_t::mediaInit
- rm_block_media_api_t::infoGet

◆ RM_FREERTOS_PLUS_FAT_DiskInit()

fsp_err_t RM_FREERTOS_PLUS_FAT_DiskInit (rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, rm_freertos_plus_fat_disk_cfg_t const *const *p_disk_cfg*, FF_Disk_t *const *p_disk*)

Initializes a FreeRTOS+FAT disk structure. This function calls FF_CreateIOManger.

Implements rm_freertos_plus_fat_api_t::diskInit().

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module has not been initialized.
FSP_ERR_INTERNAL	Call to FF_CreateIOManger failed.

◆ RM_FREERTOS_PLUS_FAT_DiskDeinit()

fsp_err_t RM_FREERTOS_PLUS_FAT_DiskDeinit (rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, FF_Disk_t *const *p_disk*)

Deinitializes a FreeRTOS+FAT disk structure. This function calls FF_DeleteIOManger.

Implements rm_freertos_plus_fat_api_t::diskDeinit().

Return values

FSP_SUCCESS	Module is initialized and ready to access the memory device.
FSP_ERR_ASSERTION	An input parameter is invalid.
FSP_ERR_NOT_OPEN	Module has not been initialized.

◆ RM_FREERTOS_PLUS_FAT_InfoGet()

fsp_err_t RM_FREERTOS_PLUS_FAT_InfoGet (rm_freertos_plus_fat_ctrl_t *const *p_ctrl*, FF_Disk_t *const *p_disk*, rm_freertos_plus_fat_info_t *const *p_info*)

Get partition information. This function can only be called after rm_freertos_plus_fat_api_t::diskInit()
.

Implements rm_freertos_plus_fat_api_t::infoGet().

Return values

FSP_SUCCESS	Information stored in p_info.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.
FSP_ERR_NOT_FOUND	The value of p_iomanager is NULL.

◆ RM_FREERTOS_PLUS_FAT_Close()

fsp_err_t RM_FREERTOS_PLUS_FAT_Close (rm_freertos_plus_fat_ctrl_t *const *p_ctrl*)

Closes media device.

Implements rm_freertos_plus_fat_api_t::close().

Return values

FSP_SUCCESS	Media device closed.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- rm_block_media_api_t::close

4.2.88 FreeRTOS Plus TCP (rm_freertos_plus_tcp)

Modules

Middleware for using TCP on RA MCUs.

Overview

FreeRTOS Plus TCP is a TCP stack created for use with FreeRTOS.

This module provides the NetworkInterface required to use FreeRTOS Plus TCP with the Ethernet (r_ether) driver.

Please refer to the FreeRTOS Plus TCP documentation for further details.

Configuration

Build Time Configurations for FreeRTOS_Plus_TCP

The following build time configurations are defined in aws/FreeRTOSIPConfig.h:

Configuration	Options	Default	Description
Print debug messages	• Disable • Enable	Disable	If ipconfigHAS_DEBUG_PRINTF is set to 1 then FreeRTOS_debug_printf should be defined to the function used to print out the debugging messages.
Print info messages	• Disable • Enable	Disable	Set to 1 to print out non debugging messages, for example the output of the FreeRTOS_netstat() command, and ping replies. If ipconfigHAS_PRINTF is set to 1 then FreeRTOS_printf should be set to the function used to print out the messages.
Byte order of the target MCU	pdFREERTOS_LITTLE_ENDIAN	pdFREERTOS_LITTLE_ENDIAN	Define the byte order of the target MCU
IP/TCP/UDP checksums	• Disable • Enable	Enable	If the network card/driver includes checksum offloading (IP/TCP/UDP checksums) then set ipconfigDRIVER_INCLUDED_RX_IP_CHECKSUM to 1 to prevent the software stack repeating the checksum calculations.
Receive Block Time	Value must be a non-negative integer	10000	Amount of time FreeRTOS_recv() will

			block for. The timeouts can be set per socket, using setsockopt().
Send Block Time	Value must be a non-negative integer	10000	Amount of time FreeRTOS_send() will block for. The timeouts can be set per socket, using setsockopt().
DNS caching	• Disable • Enable	Enable	DNS caching
DNS Request Attempts	Value must be an integer	2	When a cache is present, ipconfigDNS_REQUEST_ATTEMPTS can be kept low and also DNS may use small timeouts.
IP stack task priority	Manual Entry	configMAX_PRIORITIES - 2	Set the priority of the task that executes the IP stack.
Stack size in words (not bytes)	Manual Entry	configMINIMAL_STACK_SIZE * 5	The size, in words (not bytes), of the stack allocated to the FreeRTOS+TCP stack.
Network Events call vApplicationIPNetworkEventHook	• Disable • Enable	Enable	vApplicationIPNetworkEventHook is called when the network connects or disconnects.
Max UDP send block time	Manual Entry	15000 / portTICK_PERIOD_MS	Max UDP send block time
Use DHCP	• Disable • Enable	Enable	If ipconfigUSE_DHCP is 1 then FreeRTOS+TCP will attempt to retrieve an IP address, netmask, DNS server address and gateway address from a DHCP server.
DHCP Register Hostname	• Disable • Enable	Enable	Register hostname when using DHCP
DHCP Uses Unicast	• Disable • Enable	Enable	DHCP uses unicast.
DHCP Send Discover After Auto IP	• Disable • Enable	Disable	DHCP Send Discover After Auto IP
DHCP callback function	• Disable • Enable	Disable	Provide an implementation of the DHCP callback function

			(xApplicationDHCPHook)
Interval between transmissions	Manual Entry	120000 / portTICK_PERIOD_MS	When ipconfigUSE_DHCP is set to 1, DHCP requests will be sent out at increasing time intervals until either a reply is received from a DHCP server and accepted, or the interval between transmissions reaches ipconfigMAXIMUM_DISCOVER_TX_PERIOD.
ARP Cache Entries	Value must be an integer	6	The maximum number of entries that can exist in the ARP table at any one time
ARP Request Retransmissions	Value must be an integer	5	ARP requests that do not result in an ARP response will be re-transmitted a maximum of ipconfigMAX_ARP_RETRANSMISSIONS times before the ARP request is aborted.
Maximum time before ARP table entry becomes stale	Value must be an integer	150	The maximum time between an entry in the ARP table being created or refreshed and the entry being removed because it is stale
Use string for IP Address	• Disable • Enable	Enable	Take an IP in decimal dot format (for example, "192.168.0.1") as its parameter FreeRTOS_inet_addr_quick() takes an IP address as four separate numerical octets (for example, 192, 168, 0, 1) as its parameters
Total number of available network buffers	Value must be an integer	10	Define the total number of network buffer that are available to the IP stack
Set the maximum	Please enter a valid	ipconfigNUM_NETWORK	Set the maximum

number of events	function name without spaces or funny characters	_BUFFER_DESCRIPTORS + 5	number of events that can be queued for processing at any one time. The event queue must be a minimum of 5 greater than the total number of network buffers
Enable FreeRTOS_sendto() without calling Bind	• Enable • Disable	Disable	Set to 1 then calling FreeRTOS_sendto() on a socket that has not yet been bound will result in the IP stack automatically binding the socket to a port number from the range socketAUTO_PORT_ALLOCATION_START_NUMBER to 0xffff. If ipconfigALLOW_SOCKET_SEND_WITHOUT_BIND is set to 0 then calling FreeRTOS_sendto() on a socket that has not yet been bound will result in the send operation being aborted.
TTL values for UDP packets	Value must be an integer	128	Define the Time To Live (TTL) values used in outgoing UDP packets
TTL values for TCP packets	Value must be an integer	128	Defines the Time To Live (TTL) values used in outgoing TCP packets
Use TCP and all its features	• Disable • Enable	Enable	Use TCP and all its features
Let TCP use windowing mechanism	• Disable • Enable	Disable	Let TCP use windowing mechanism
Maximum number of bytes the payload of a network frame can contain	Value must be an integer	1500	Maximum number of bytes the payload of a network frame can contain
Basic DNS client or resolver	• Disable • Enable	Enable	Set ipconfigUSE_DNS to 1 to include a basic DNS client/resolver. DNS is used through the FreeRTOS_gethostbyname() API function.
Reply to incoming ICMP	• Disable	Enable	If ipconfigREPLY_TO_IN

echo (ping) requests	• Enable		COMING_PINGS is set to 1 then the IP stack will generate replies to incoming ICMP echo (ping) requests.
FreeRTOS_SendPingRequest() is available	• Disable • Enable	Disable	If ipconfigSUPPORT_OUTGOING_PINGS is set to 1 then the FreeRTOS_SendPingRequest() API function is available.
FreeRTOS_select() (and associated) API function is available	• Disable • Enable	Disable	If ipconfigSUPPORT_SELECT_FUNCTION is set to 1 then the FreeRTOS_select() (and associated) API function is available
Filter out non Ethernet II frames.	• Disable • Enable	Enable	If ipconfigFILTER_OUT_NON_ETHERNET_II_FRAMES is set to 1 then Ethernet frames that are not in Ethernet II format will be dropped. This option is included for potential future IP stack developments
Responsibility of the Ethernet interface to filter out packets	• Disable • Enable	Disable	If ipconfigETHERNET_DRIVER_FILTERS_FRAME_TYPES is set to 1 then it is the responsibility of the Ethernet interface to filter out packets that are of no interest.
Send RST packets, when receive unknown packets.	• Disable • Enable	Enable	TCP will not send RST packets in reply to TCP unknown or out-of-order packets
Block time to simulate MAC interrupts	Please enter a valid function name without spaces or funny characters	20 / portTICK_PERIOD_MS	The windows simulator cannot really simulate MAC interrupts, and needs to block occasionally to allow other tasks to run
Access 32-bit fields in the IP packets	Value must be an integer	2	To access 32-bit fields in the IP packets with 32-bit memory instructions, all packets will be stored 32-bit-aligned, plus 16-bits. This has to do with the contents of the IP-packets: all 32-bit

			fields are 32-bit-aligned, plus 16-bit
Size of the pool of TCP window descriptors	Value must be an integer	240	Define the size of the pool of TCP window descriptors
Size of Rx buffer for TCP sockets	Value must be an integer	3000	Define the size of Rx buffer for TCP sockets
Size of Tx buffer for TCP sockets	Value must be an integer	3000	Define the size of Tx buffer for TCP sockets
TCP keep-alive	• Disable • Enable	Enable	TCP keep-alive is avaiable or not
TCP keep-alive interval	Value must be an integer	120	TCP keep-alive interval in second
The socket semaphore to unblock the MQTT task (USER_SEMAPHORE)	• Disable • Enable	Disable	The socket semaphore is used to unblock the MQTT task
The socket semaphore to unblock the MQTT task (WAKE_CALLBACK)	• Disable • Enable	Enable	The socket semaphore is used to unblock the MQTT task
The socket semaphore to unblock the MQTT task (USE_CALLBACKS)	• Disable • Enable	Disable	The socket semaphore is used to unblock the MQTT task
The socket semaphore to unblock the MQTT task (TX_DRIVER)	• Disable • Enable	Disable	The socket semaphore is used to unblock the MQTT task
The socket semaphore to unblock the MQTT task (RX_DRIVER)	• Disable • Enable	Disable	The socket semaphore is used to unblock the MQTT task
Possible optimisation for expert users	• Disable • Enable	Disable	Possible optimisation for expert users - requires network driver support. It is is useful when there is high network traffic. If non-zero value then instead of passing received packets into the IP task one at a time the network interface can chain received packets together and pass them into the IP task in one go. If set to 0 then only one buffer will be sent at a time.

Usage Notes

In order to use the NetworkInterface implementation provided by Renesas for RA devices:

- Configure an r_ether instance and provide a pointer to the instance of the NetworkInterface as follows:

```
/* Reference used by the NetworkInterface to access the ethernet instance. */
extern ether_instance_t const * gp_freertos_ether;
...
/* Make it reference the configured ether instance. */
ether_instance_t const * gp_freertos_ether = &g_ether_instance;
```

- Follow the TCP stack initialization procedure as described here: FreeRTOS+TCP Networking Tutorial: Initializing the TCP/IP Stack

Note

The MAC address passed to FreeRTOS_IPInit must match the MAC address configured in the r_ether instance.
g_ether_instance must have vEtherISRCallback configured as the callback.
The xApplicationGetRandomNumber and ulApplicationGetNextSequenceNumber functions should be implemented in systems using FreeRTOS Plus TCP without Secure Sockets.
To connect to a server using an IP address the macro ipconfigINCLUDE_FULL_INET_ADDR must be set to 1.

Limitations

- Zero-copy is not currently supported by the NetworkInterface.

4.2.89 FreeRTOS Port (rm_freertos_port)

Modules

FreeRTOS port for RA MCUs.

Overview

Note

The FreeRTOS Port does not provide any interfaces to the user. Consult the FreeRTOS documentation at https://www.freertos.org/Documentation for further information.

Features

The RA FreeRTOS port supports the following features:

- Standard FreeRTOS configurations
- Hardware stack monitor

Configuration

Build Time Configurations for all

The following build time configurations are defined in aws/FreeRTOSConfig.h:

Configuration	Options	Default	Description
General > Custom FreeRTOSConfig.h	Manual Entry		Add a path to your custom FreeRTOSConfig.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations.
General > Use Preemption	• Enabled • Disabled	Enabled	Set to Enabled to use the preemptive RTOS scheduler, or Disabled to use the cooperative RTOS scheduler.
General > Use Port Optimised Task Selection	• Enabled • Disabled	Disabled	Some FreeRTOS ports have two methods of selecting the next task to execute - a generic method, and a method that is specific to that port. The Generic method: Is used when Use Port Optimized Task Selection is set to 0, or when a port specific method is not implemented. Can be used with all FreeRTOS ports. Is completely written in C, making it less efficient than a port specific method. Does not impose a limit on the maximum number of available priorities. A port specific method: Is not available for all ports. Is used when Use Port Optimized Task

			Selection is Enabled. Relies on one or more architecture specific assembly instructions (typically a Count Leading Zeros [CLZ] or equivalent instruction) so can only be used with the architecture for which it was specifically written. Is more efficient than the generic method. Typically imposes a limit of 32 on the maximum number of available priorities.
General > Use Tickless Idle	• Enabled • Disabled	Disabled	Set Use Tickless Idle to Enabled to use the low power tickless mode, or Disabled to keep the tick interrupt running at all times. Low power tickless implementations are not provided for all FreeRTOS ports.
General > Cpu Clock Hz	Manual Entry	SystemCoreClock	Enter the frequency in Hz at which the internal clock that drives the peripheral used to generate the tick interrupt will be executing - this is normally the same clock that drives the internal CPU clock. This value is required in order to correctly configure timer peripherals.
General > Tick Rate Hz	Must be an integer and greater than 0	1000	The frequency of the RTOS tick interrupt. The tick interrupt is used to measure time. Therefore a higher tick frequency means time can be measured to a higher resolution. However, a high tick frequency also means that the RTOS kernel will use more CPU time

			so be less efficient. The RTOS demo applications all use a tick rate of 1000Hz. This is used to test the RTOS kernel and is higher than would normally be required. More than one task can share the same priority. The RTOS scheduler will share processor time between tasks of the same priority by switching between the tasks during each RTOS tick. A high tick rate frequency will therefore also have the effect of reducing the 'time slice' given to each task.
General > Max Priorities	Must be an integer and greater than 0	5	The number of priorities available to the application tasks. Any number of tasks can share the same priority. Each available priority consumes RAM within the RTOS kernel so this value should not be set any higher than actually required by your application.
General > Minimal Stack Size	Must be an integer and greater than 0	128	The size of the stack used by the idle task. Generally this should not be reduced from the value set in the FreeRTOSConfig.h file provided with the demo application for the port you are using. Like the stack size parameter to the xTaskCreate() and xTaskCreateStatic() functions, the stack size is specified in words, not bytes. If each item placed on

			the stack is 32-bits, then a stack size of 100 means 400 bytes (each 32-bit stack item consuming 4 bytes).
General > Max Task Name Len	Must be an integer and greater than 0	16	The maximum permissible length of the descriptive name given to a task when the task is created. The length is specified in the number of characters including the NULL termination byte.
General > Use 16-bit Ticks	Disabled	Disabled	Time is measured in 'ticks' - which is the number of times the tick interrupt has executed since the RTOS kernel was started. The tick count is held in a variable of type TickType_t. Defining configUSE_16_BIT_TICKS as 1 causes TickType_t to be defined (typedef'ed) as an unsigned 16bit type. Defining configUSE_16_BIT_TICKS as 0 causes TickType_t to be defined (typedef'ed) as an unsigned 32bit type. Using a 16-bit type will greatly improve performance on 8- and 16-bit architectures, but limits the maximum specifiable time period to 65535 'ticks'. Therefore, assuming a tick frequency of 250Hz, the maximum time a task can delay or block when a 16bit counter is used is 262 seconds, compared to 17179869 seconds when using a 32-bit counter.

General > Idle Should Yield	• Enabled • Disabled	Enabled	This parameter controls the behaviour of tasks at the idle priority. It only has an effect if: The preemptive scheduler is being used. The application creates tasks that run at the idle priority. If Use Time Slicing is Enabled then tasks that share the same priority will time slice. If none of the tasks get preempted then it might be assumed that each task at a given priority will be allocated an equal amount of processing time - and if the priority is above the idle priority then this is indeed the case. When tasks share the idle priority the behaviour can be slightly different. If Idle Should Yield is Enabled then the idle task will yield immediately if any other task at the idle priority is ready to run. This ensures the minimum amount of time is spent in the idle task when application tasks are available for scheduling. This behaviour can however have undesirable effects (depending on the needs of your application) as depicted below: The diagram above shows the execution pattern of four tasks that are all running at the idle priority. Tasks A, B and C are

			application tasks. Task I is the idle task. A context switch occurs with regular period at times T0, T1, ..., T6. When the idle task yields task A starts to execute - but the idle task has already consumed some of the current time slice. This results in task I and task A effectively sharing the same time slice. The application tasks B and C therefore get more processing time than the application task A. This situation can be avoided by: If appropriate, using an idle hook in place of separate tasks at the idle priority. Creating all application tasks at a priority greater than the idle priority. Setting Idle Should Yield to Disabled. Setting Idle Should Yield to Disabled prevents the idle task from yielding processing time until the end of its time slice. This ensure all tasks at the idle priority are allocated an equal amount of processing time (if none of the tasks get pre-empted) - but at the cost of a greater proportion of the total processing time being allocated to the idle task.
General > Use Task Notifications	• Enabled • Disabled	Enabled	Setting Use Task Notifications to Enabled will include direct to

			task notification functionality and its associated API in the build. Setting Use Task Notifications to Disabled will exclude direct to task notification functionality and its associated API from the build. Each task consumes 8 additional bytes of RAM when direct to task notifications are included in the build.
General > Use Mutexes	• Enabled • Disabled	Disabled	Set to Enabled to include mutex functionality in the build, or Disabled to omit mutex functionality from the build. Readers should familiarise themselves with the differences between mutexes and binary semaphores in relation to the FreeRTOS functionality.
General > Use Recursive Mutexes	• Enabled • Disabled	Disabled	Set to Enabled to include recursive mutex functionality in the build, or Disabled to omit recursive mutex functionality from the build.
General > Use Counting Semaphores	• Enabled • Disabled	Enabled	Set to Enabled to include counting semaphore functionality in the build, or Disabled to omit counting semaphore functionality from the build.
General > Queue Registry Size	Must be an integer and greater than 0	10	The queue registry has two purposes, both of which are associated with RTOS kernel aware debugging:

			It allows a textual name to be associated with a queue for easy queue identification within a debugging GUI. It contains the information required by a debugger to locate each registered queue and semaphore. The queue registry has no purpose unless you are using a RTOS kernel aware debugger. Registry Size defines the maximum number of queues and semaphores that can be registered. Only those queues and semaphores that you want to view using a RTOS kernel aware debugger need be registered. See the API reference documentation for vQueueAddToRegistry() and vQueueUnregisterQueue() for more information.
General > Use Queue Sets	• Enabled • Disabled	Disabled	Set to Enabled to include queue set functionality (the ability to block, or pend, on multiple queues and semaphores), or Disabled to omit queue set functionality.
General > Use Time Slicing	• Enabled • Disabled	Disabled	If Use Time Slicing is Enabled, FreeRTOS uses prioritised preemptive scheduling with time slicing. That means the RTOS scheduler will always run the highest priority task that is in the Ready state, and will switch between tasks of equal priority on

			every RTOS tick interrupt. If Use Time Slicing is Disabled then the RTOS scheduler will still run the highest priority task that is in the Ready state, but will not switch between tasks of equal priority just because a tick interrupt has occurred.
General > Use Newlib Reentrant	• Enabled • Disabled	Disabled	If Use Newlib Reentrant is Enabled then a newlib reent structure will be allocated for each created task. Note Newlib support has been included by popular demand, but is not used by the FreeRTOS maintainers themselves. FreeRTOS is not responsible for resulting newlib operation. User must be familiar with newlib and must provide system-wide implementations of the necessary stubs. Be warned that (at the time of writing) the current newlib design implements a system-wide malloc() that must be provided with locks.
General > Enable Backward Compatibility	• Enabled • Disabled	Disabled	The FreeRTOS.h header file includes a set of #define macros that map the names of data types used in versions of FreeRTOS prior to version 8.0.0 to the names used in FreeRTOS version 8.0.0. The macros allow application code to update the version of FreeRTOS they are built against from a pre 8.0.0 version to a post 8.0.0 version without modification. Setting Enable Backward

			Compatibility to Disabled in FreeRTOSConfig.h excludes the macros from the build, and in so doing allowing validation that no pre version 8.0.0 names are being used.
General > Num Thread Local Storage Pointers	Must be an integer and greater than 0	5	Sets the number of indexes in each task's thread local storage array.
General > Stack Depth Type	Manual Entry	uint32_t	Sets the type used to specify the stack depth in calls to xTaskCreate(), and various other places stack sizes are used (for example, when returning the stack high water mark). Older versions of FreeRTOS specified stack sizes using variables of type UBaseType_t, but that was found to be too restrictive on 8-bit microcontrollers. Stack Depth Type removes that restriction by enabling application developers to specify the type to use.
General > Message Buffer Length Type	Manual Entry	size_t	FreeRTOS Message buffers use variables of type Message Buffer Length Type to store the length of each message. If Message Buffer Length Type is not defined then it will default to size_t. If the messages stored in a message buffer will never be larger than 255 bytes then defining Message Buffer Length Type to uint8_t will save 3 bytes per message on a 32-bit microcontroller.

			Likewise if the messages stored in a message buffer will never be larger than 65535 bytes then defining Message Buffer Length Type to uint16_t will save 2 bytes per message on a 32-bit microcontroller.
General > Library Max Syscall Interrupt Priority	MCU Specific Options		The highest interrupt priority that can be used by any interrupt service routine that makes calls to interrupt safe FreeRTOS API functions. DO NOT CALL INTERRUPT SAFE FREERTOS API FUNCTIONS FROM ANY INTERRUPT THAT HAS A HIGHER PRIORITY THAN THIS! (higher priorities are lower numeric values) Below is explanation for macros that are set based on this value from FreeRTOS website. In the RA port, configKERNEL_INTERRUPT_PRIORITY is not used and the kernel runs at the lowest priority. Note in the following discussion that only API functions that end in "FromISR" can be called from within an interrupt service routine. configMAX_SYSCALL_INTERRUPT_PRIORITY sets the highest interrupt priority from which interrupt safe FreeRTOS API functions can be called.

A full interrupt nesting model is achieved by setting configMAX_SYSCALL_INTERRUPT_PRIORITY above (that is, at a higher priority level) than configKERNEL_INTERRUPT_PRIORITY. This means the FreeRTOS kernel does not completely disable interrupts, even inside critical sections. Further, this is achieved without the disadvantages of a segmented kernel architecture.

Interrupts that do not call API functions can execute at priorities above configMAX_SYSCALL_INTERRUPT_PRIORITY and therefore never be delayed by the RTOS kernel execution.

A special note for ARM Cortex-M users: Please read the page dedicated to interrupt priority settings on ARM Cortex-M devices. As a minimum, remember that ARM Cortex-M cores use numerically low priority numbers to represent HIGH priority interrupts, which can seem counter-intuitive and is easy to forget! If you wish to assign an interrupt a low priority do NOT assign it a priority of 0 (or other low numeric value) as this can result in the interrupt actually having the highest priority in the system - and therefore potentially make your

			system crash if this priority is above config MAX_SYSCALL_INTERRUPT_PRIORITY. The lowest priority on a ARM Cortex-M core is in fact 255 - however different ARM Cortex-M vendors implement a different number of priority bits and supply library functions that expect priorities to be specified in different ways. For example, on the RA6M3 the lowest priority you can specify is 15 - and the highest priority you can specify is 0.
General > Assert	Manual Entry	assert(x)	The semantics of the configASSERT() macro are the same as the standard C assert() macro. An assertion is triggered if the parameter passed into configASSERT() is zero. configASSERT() is called throughout the FreeRTOS source files to check how the application is using FreeRTOS. It is highly recommended to develop FreeRTOS applications with configASSERT() defined. The example definition (shown at the top of the file and replicated below) calls vAssertCalled(), passing in the file name and line number of the triggering configASSERT() call (__FILE__ and __LINE__ are standard macros provided by most compilers). This is just

for demonstration as vAssertCalled() is not a FreeRTOS function, configASSERT() can be defined to take whatever action the application writer deems appropriate.

It is normal to define configASSERT() in such a way that it will prevent the application from executing any further. This if for two reasons; stopping the application at the point of the assertion allows the cause of the assertion to be debugged, and executing past a triggered assertion will probably result in a crash anyway.

Note defining configASSERT() will increase both the application code size and execution time. When the application is stable the additional overhead can be removed by simply commenting out the configASSERT() definition in FreeRTOSConfig.h.

/* Define configASSERT() to call vAssertCalled() if the assertion fails. The assertion
has failed if the value of the parameter passed into configASSERT() equals zero. */
#define configASSERT((x)) if((x) == 0) vAssertCalled(__FILE__, __LINE__)
If running FreeRTOS

			under the control of a debugger, then configASSERT() can be defined to just disable interrupts and sit in a loop, as demonstrated below. That will have the effect of stopping the code on the line that failed the assert test - pausing the debugger will then immediately take you to the offending line so you can see why it failed. /* Define configASSERT() to disable interrupts and sit in a loop. */ #define configASSERT((x)) if((x) == 0) { taskDISABLE_INTERRUPTS(); for(;;); }
General > Include Application Defined Privileged Functions	• Enabled • Disabled	Disabled	Include Application Defined Privileged Functions is only used by FreeRTOS MPU. If Include Application Defined Privileged Functions is Enabled then the application writer must provide a header file called "application_defined_privileged_functions.h", in which functions the application writer needs to execute in privileged mode can be implemented. Note that, despite having a .h extension, the header file should contain the implementation of the C functions, not just the functions' prototypes. Functions implemented in "application_defined_privileged_functions.h

			" must save and restore the processor's privilege state using the prvRaisePrivilege() function and portRESET_PRIVILEGE() macro respectively. For example, if a library provided print function accesses RAM that is outside of the control of the application writer, and therefore cannot be allocated to a memory protected user mode task, then the print function can be encapsulated in a privileged function using the following code:
			
			void MPU_debug_printf(const char *pcMessage)
			{
			/* State the privilege level of the processor when the function was called. */
			BaseType_t xRunningPrivileged = prvRaisePrivilege();
			
			/* Call the library function, which now has access to all RAM. */
			debug_printf(pcMessage);
			
			/* Reset the processor privilege level to its original value. */
			portRESET_PRIVILEGE(xRunningPrivileged);
			}
			This technique should only be use during development, and not deployment, as it circumvents the memory protection.

Hooks > Use Idle Hook	• Enabled	Enabled	Set to Enabled if you

	• Disabled		wish to use an idle hook, or Disabled to omit an idle hook.
Hooks > Use Malloc Failed Hook	• Enabled • Disabled	Disabled	The kernel uses a call to pvPortMalloc() to allocate memory from the heap each time a task, queue or semaphore is created. The official FreeRTOS download includes four sample memory allocation schemes for this purpose. The schemes are implemented in the heap_1.c, heap_2.c, heap_3.c, heap_4.c and heap_5.c source files respectively. Use Malloc Failed Hook is only relevant when one of these three sample schemes is being used. The malloc() failed hook function is a hook (or callback) function that, if defined and configured, will be called if pvPortMalloc() ever returns NULL. NULL will be returned only if there is insufficient FreeRTOS heap memory remaining for the requested allocation to succeed. If Use Malloc Failed Hook is Enabled then the application must define a malloc() failed hook function. If Use Malloc Failed Hook is set to Dosab;ed then the malloc() failed hook function will not be called, even if one is defined. Malloc() failed hook functions must have the name and prototype shown below.

			void vApplicationMallocFailedHook(void);
Hooks > Use Daemon Task Startup Hook	• Enabled • Disabled	Disabled	If Use Timers and Use Daemon Task Startup Hook are both Enabled then the application must define a hook function that has the exact name and prototype as shown below. The hook function will be called exactly once when the RTOS daemon task (also known as the timer service task) executes for the first time. Any application initialisation code that needs the RTOS to be running can be placed in the hook function. void void vApplicationDaemonTaskStartupHook(void);
Hooks > Use Tick Hook	• Enabled • Disabled	Disabled	Set to Enabled if you wish to use an tick hook, or Disabled to omit an tick hook.
Hooks > Check For Stack Overflow	• Enabled • Disabled	Disabled	The stack overflow detection page describes the use of this parameter. This is not recommended for RA MCUs with hardware stack monitor support. RA MCU designs should enable the RA hardware stack monitor instead.
Stats > Use Trace Facility	• Enabled • Disabled	Disabled	Set to Enabled if you wish to include additional structure members and functions to assist with execution visualisation and tracing.
Stats > Use Stats Formatting Functions	• Enabled • Disabled	Disabled	Set Use Trace Facility and Use Stats Formatting Functions to Enabled to include

			the vTaskList() and vTa skGetRunTimeStats() functions in the build. Setting either to Disabled will omit vTaskList() and vTaskG etRunTimeStates() from the build.
Stats > Generate Run Time Stats	• Enabled • Disabled	Disabled	The Run Time Stats page describes the use of this parameter.
Memory Allocation > Support Static Allocation	• Enabled • Disabled	Enabled	If Support Static Allocation is Enabled then RTOS objects can be created using RAM provided by the application writer. If Support Static Allocation is Disabled then RTOS objects can only be created using RAM allocated from the FreeRTOS heap. If Support Static Allocation is left undefined it will default to 0. If Support Static Allocation is Enabled then the application writer must also provide two callback functions: vApplication GetIdleTaskMemory() to provide the memory for use by the RTOS Idle task, and (if Use Timers is Enabled) vAp plicationGetTimerTask Memory() to provide memory for use by the RTOS Daemon/Timer Service task. Examples are provided below. /* Support Static Allocation is Enabled, so the application must provide an implementation of vAp plicationGetIdleTaskMe

mory() to provide the
memory that is
used by the Idle task. */
void vApplicationGetIdl
eTaskMemory(
StaticTask_t **ppxIdleT
askTCBBuffer,
StackType_t **ppxIdleT
askStackBuffer,
uint32_t
*pulIdleTaskStackSize)
{
/* If the buffers to be
provided to the Idle
task are declared
inside this
function then they
must be declared static
- otherwise they will be
allocated on
the stack and so not
exists after this
function exits. */
static StaticTask_t
xIdleTaskTCB;
static StackType_t
uxIdleTaskStack[config
MINIMAL_STACK_SIZE];

/* Pass out a pointer to
the StaticTask_t
structure in which the
Idle task's
state will be stored. */
*ppxIdleTaskTCBBuffer
=

/* Pass out the array
that will be used as the
Idle task's stack. */
*ppxIdleTaskStackBuffe
r = uxIdleTaskStack;

/* Pass out the size of
the array pointed to by
*ppxIdleTaskStackBuffe
r.
Note that, as the array
is necessarily of type
StackType_t,
configMINIMAL_STACK_
SIZE is specified in
words, not bytes. */
*pulIdleTaskStackSize
= configMINIMAL_STAC

K_SIZE;
}
/*---*/

/* Support Static Allocation and Use Timers are both Enabled, so the application must provide an implementation of vApplicationGetTimerTaskMemory()
to provide the memory that is used by the Timer service task. */
void vApplicationGetTimerTaskMemory(StaticTask_t **ppxTimerTaskTCBBuffer,
StackType_t **ppxTimerTaskStackBuffer,
uint32_t *pulTimerTaskStackSize)
{
/* If the buffers to be provided to the Timer task are declared inside this
function then they must be declared static - otherwise they will be allocated on
the stack and so not exists after this function exits. */
static StaticTask_t xTimerTaskTCB;
static StackType_t uxTimerTaskStack[configTIMER_TASK_STACK_DEPTH];

/* Pass out a pointer to the StaticTask_t structure in which the Timer
task's state will be stored. */
*ppxTimerTaskTCBBuffer =

/* Pass out the array

			that will be used as the Timer task's stack. */ *ppxTimerTaskStackBuffer = uxTimerTaskStack; /* Pass out the size of the array pointed to by *ppxTimerTaskStackBuffer. Note that, as the array is necessarily of type StackType_t, configTIMER_TASK_STACK_DEPTH is specified in words, not bytes. */ *pulTimerTaskStackSize = configTIMER_TASK_STACK_DEPTH; } Examples of the callback functions that must be provided by the application to supply the RAM used by the Idle and Timer Service tasks if Support Static Allocation is Enabled. See the Static Vs Dynamic Memory Allocation page for more information.
Memory Allocation > Support Dynamic Allocation	• Enabled • Disabled	Disabled	If Support Dynamic Allocation is Enabled then RTOS objects can be created using RAM that is automatically allocated from the FreeRTOS heap. If Support Dynamic Allocation is set to 0 then RTOS objects can only be created using RAM provided by the application writer. See the Static Vs Dynamic Memory Allocation page for more information.
Memory Allocation >	Must be an integer and	1024	The total amount of

Total Heap Size	greater than 0		RAM available in the FreeRTOS heap. This value will only be used if Support Dynamic Allocation is Enabled and the application makes use of one of the sample memory allocation schemes provided in the FreeRTOS source code download. See the memory configuration section for further details.
Memory Allocation > Application Allocated Heap	• Enabled • Disabled	Disabled	By default the FreeRTOS heap is declared by FreeRTOS and placed in memory by the linker. Setting Application Allocated Heap to Enabled allows the heap to instead be declared by the application writer, which allows the application writer to place the heap wherever they like in memory. If heap_1.c, heap_2.c or heap_4.c is used, and Application Allocated Heap is Enabled, then the application writer must provide a uint8_t array with the exact name and dimension as shown below. The array will be used as the FreeRTOS heap. How the array is placed at a specific memory location is dependent on the compiler being used - refer to your compiler's documentation. uint8_t ucHeap[configTOTAL_HEAP_SIZE];
Timers > Use Timers	• Enabled • Disabled	Enabled	Set to Enabled to include software timer

			functionality, or Disabled to omit software timer functionality. See the FreeRTOS software timers page for a full description.
Timers > Timer Task Priority	Must be an integer and greater than 0	3	Sets the priority of the software timer service/daemon task. See the FreeRTOS software timers page for a full description.
Timers > Timer Queue Length	Must be an integer and greater than 0	10	Sets the length of the software timer command queue. See the FreeRTOS software timers page for a full description.
Timers > Timer Task Stack Depth	Must be an integer and greater than 0	128	Sets the stack depth allocated to the software timer service/daemon task. See the FreeRTOS software timers page for a full description.
Optional Functions > vTaskPrioritySet() Function	• Enabled • Disabled	Enabled	Include vTaskPrioritySet() function in build
Optional Functions > uxTaskPriorityGet() Function	• Enabled • Disabled	Enabled	Include uxTaskPriorityGet() function in build
Optional Functions > vTaskDelete() Function	• Enabled • Disabled	Enabled	Include vTaskDelete() function in build
Optional Functions > vTaskSuspend() Function	• Enabled • Disabled	Enabled	Include vTaskSuspend() function in build
Optional Functions > xResumeFromISR() Function	• Enabled • Disabled	Enabled	Include xResumeFromISR() function in build
Optional Functions > vTaskDelayUntil() Function	• Enabled • Disabled	Enabled	Include vTaskDelayUntil() function in build
Optional Functions > vTaskDelay() Function	• Enabled • Disabled	Enabled	Include vTaskDelay() function in build
Optional Functions > x TaskGetSchedulerState () Function	• Enabled • Disabled	Enabled	Include xTaskGetSched ulerState() function in build

Optional Functions > xTaskGetCurrentTaskHandle() Function	• Enabled • Disabled	Enabled	Include xTaskGetCurrentTaskHandle() function in build
Optional Functions > uxTaskGetStackHighWaterMark() Function	• Enabled • Disabled	Disabled	Include uxTaskGetStackHighWaterMark() function in build
Optional Functions > xTaskGetIdleTaskHandle() Function	• Enabled • Disabled	Disabled	Include xTaskGetIdleTaskHandle() function in build
Optional Functions > eTaskGetState() Function	• Enabled • Disabled	Disabled	Include eTaskGetState() function in build
Optional Functions > xEventGroupSetBitFromISR() Function	• Enabled • Disabled	Enabled	Include xEventGroupSetBitFromISR() function in build
Optional Functions > xTimerPendFunctionCall() Function	• Enabled • Disabled	Disabled	Include xTimerPendFunctionCall() function in build
Optional Functions > xTaskAbortDelay() Function	• Enabled • Disabled	Disabled	Include xTaskAbortDelay() function in build
Optional Functions > xTaskGetHandle() Function	• Enabled • Disabled	Disabled	Include xTaskGetHandle() function in build
Optional Functions > xTaskResumeFromISR() Function	• Enabled • Disabled	Enabled	Include xTaskResumeFromISR() function in build
RA > Hardware Stack Monitor	• Enabled • Disabled	Disabled	Include RA stack monitor
Logging > Print String Function	Manual Entry	printf(x)	
Logging > Logging Max Message Length	Manual Entry	192	
Logging > Logging Include Time and Task Name	• Disabled • Enabled	Disabled	

Clock Configuration

The FreeRTOS port uses the SysTick timer as the system clock. The timer rate is configured in the FreeRTOS component under General > Tick Rate Hz.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Hardware Stack Monitor (PSPLIM)

A UsageFault is generated if PSP goes out of the memory area for the stack allocated for the current task. If UsageFault is not enabled, it is escalated to HardFault.

Hardware Stack Monitor (SPMON)

The hardware stack monitor generates an NMI if the PSP goes out of the memory area for the stack allocated for the current task. A callback can be registered using R_BSP_GroupIrqWrite() to be called whenever a stack overflow or underflow of the PSP for a particular thread is detected.

Stack Monitor Underflow Detection

By default the hardware stack monitor only checks for overflow of the process stack. To check for underflow define configRECORD_STACK_HIGH_ADDRESS as 1 on the command line.

Low Power Modes

When FreeRTOS is configured to use tickless idle, the idle task executes WFI() when no task is ready to run. If the MCU is configured to enter software standby mode or deep software standby mode when the idle task executes WFI(), the RA FreeRTOS port changes the low power mode to sleep mode so the idle task can wake from SysTick. The low power mode settings are restored when the MCU wakes from sleep mode.

TrustZone Integration

When using an RTOS in a TrustZone project, ARM recommends keeping the RTOS in the non-secure project. Tasks may call non-secure callable functions if the task has allocated a secure context (using portALLOCATE_SECURE_CONTEXT).

The secure context can be freed by deleting the thread or using the portCLEAN_UP_TCB(pxTCB) macro.

Examples

Stack Monitor Example

This is an example of using the stack monitor in an application.

```
#if BSP_FEATURE_BSP_HAS_SP_MON
void stack_monitor_callback (bsp_grp_irq_t irq)
{
 FSP_PARAMETER_NOT_USED(irq);
 if (1U == R_MPU_SPMON->SP[0].CTL_b.ERROR)
    {
 /* Handle main stack monitor error here. */
    }
 if (1U == R_MPU_SPMON->SP[1].CTL_b.ERROR)
```

```
    {
 /* Handle process stack monitor error here. */
    }
}
void rm_freertos_port_stack_monitor_example (void)
{
 /* Register a callback to be called when the stack goes outside the allocated stack
area. */
 R_BSP_GroupIrqWrite(BSP_GRP_IRQ_MPU_STACK, stack_monitor_callback);
}
#else
/* Allocate stack space to return from UsageFault. */
uint32_t g_stack_overflow_exception_stack[8] BSP_ALIGN_VARIABLE(BSP_STACK_ALIGNMENT)
BSP_PLACE_IN_SECTION(
    BSP_SECTION_STACK);
/* MCUs that do not have an SPMON stack monitor use PSPLIM to detect stack overflows.
When a stack overflow error
 * occurs, the UsageFault_Handler fires if it has been enabled. */
void UsageFault_Handler (void)
{
 register uint32_t cfsr = SCB->CFSR;
 if (cfsr & SCB_CFSR_STKOF_Msk)
    {
 /* Update PSP and PSPLIM to point to an exception stack frame allocated for stack
overflows. */
 register uint32_t * p_exception_stack_frame = (uint32_t *)
(&g_stack_overflow_exception_stack);
       __set_PSP((uint32_t) p_exception_stack_frame);
       __set_PSPLIM((uint32_t) p_exception_stack_frame);
 /* Clear XPSR, only set T-bit. */
       p_exception_stack_frame[7] = 1U << 24;
 /* Set PC to stack overflow error while loop. When execution returns from the
UsageFault, it will go to the
  * stack_overflow_error_occurred function. It cannot return to the location where
```

```
the fault occurred because
  * the MCU does not save the exception stack frame to the stack when a stack
overflow error occurs. */
       p_exception_stack_frame[6] = (uint32_t) stack_overflow_error_occurred;
    }
 /* Clear flags. */
    SCB->CFSR = cfsr;
}
/* This function is called from UsageFault_Handler after a stack overflow occurs. */
void stack_overflow_error_occurred (void)
{
 /* When recovering from a stack overflow, move the task to a while(1) loop. */
 while (1)
    {
 /* Do nothing. */
    }
}
void rm_freertos_port_stack_monitor_example (void)
{
 /* Enable usage fault. */
    SCB->SHCSR |= SCB_SHCSR_USGFAULTENA_Msk;
}
#endif
```

TrustZone Example

This is an example of calling portALLOCATE_SECURE_CONTEXT before calling any non-secure callable functions in a task.

```
void rm_freertos_port_trustzone_task_example (void)
{
 /* When FreeRTOS is used in a non-secure TrustZone application,
portALLOCATE_SECURE_CONTEXT must be called prior
  * to calling any non-secure callable function in a task. The parameter is unused in
the FSP implementation. */
```

```
    portALLOCATE_SECURE_CONTEXT(0);
    rm_freertos_port_nsc_function();
}
```

4.2.90 RTOS Context Management (rm_tz_context)

Modules

RTOS Context Management for RA MCUs.

Overview

Add this module to a secure TrustZone project to allow the associated non-secure project to use an RTOS. It is used by an RTOS port for RA MCUs (for example, the FreeRTOS Port (rm_freertos_port), which is automatically added to RA projects when FreeRTOS is selected during project creation).

Note

The RTOS Context Management module does not provide any interfaces to the user. To use this module to port an RTOS, consult the Arm documentation at https://arm-software.github.io/CMSIS_5/Core/html/group__context__trustzone__functions.html for further information.

Configuration

Build Time Configurations for rm_tz_context

The following build time configurations are defined in fsp_cfg/rm_tz_context_cfg.h:

Configuration	Options	Default	Description
Process Stack Slots	Value must be a non-negative integer greater than 0	8	The maximum number of threads that can allocate a secure context. For applications using FreeRTOS, the Idle task requires 1 context as well.
Process Stack Size	Value must be a non-negative multiple of 8	256	The maximum stack size of all non-secure callable functions.

Clock Configuration

This module does not use peripheral clocks.

Pin Configuration

This module does not use I/O pins.

Usage Notes

TrustZone Integration

When using an RTOS in a TrustZone project, ARM recommends keeping the RTOS in the non-secure project. Tasks may call non-secure callable functions if the task has allocated a secure context. To allocate a secure context, reference the documentation for the RTOS port used. For example, reference TrustZone Integration when FreeRTOS is used.

Sealing the Process Stack

This module seals each process stack by placing the value 0xFEF5EDA5 above the stack top. For more information, refer to section 3.5 "Sealing a Stack" in "Secure software guidelines for ARMv8-M": https://developer.arm.com/documentation/100720/0300.

4.2.91 FS2012 Sensor Middleware (rm_fs2012)

Modules

Functions

fsp_err_t	RM_FS2012_Open (rm_fsxxxx_ctrl_t *const p_api_ctrl, rm_fsxxxx_cfg_t const *const p_cfg) Opens and configures the FS2012 Middle module. Implements rm_fsxxxx_api_t::open. More...
fsp_err_t	RM_FS2012_Close (rm_fsxxxx_ctrl_t *const p_api_ctrl) Disables specified FS2012 control block. Implements rm_fsxxxx_api_t::close. More...
fsp_err_t	RM_FS2012_Read (rm_fsxxxx_ctrl_t *const p_api_ctrl, rm_fsxxxx_raw_data_t *const p_raw_data) Reads ADC data from FS2012. Implements rm_fsxxxx_api_t::read. More...
fsp_err_t	RM_FS2012_DataCalculate (rm_fsxxxx_ctrl_t *const p_api_ctrl, rm_fsxxxx_raw_data_t *const p_raw_data, rm_fsxxxx_data_t *const p_fs2012_data) Calculates flow from ADC data. Unit of Gas flow is SLPM (Standard liter per minute) Unit of Liquid flow is SCCM (Standard cubic centimeter per minute) Implements rm_fsxxxx_api_t::dataCalculate. More...

Detailed Description

Middleware to implement the FS2012 sensor interface. This module implements the FSXXXX Middleware Interface.

Overview

Features

The FS2012 sensor interface implementation has the following key features:

- Getting ADC data from the sensor
- Calculating flow value from ADC data

Configuration

Build Time Configurations for rm_fs2012

The following build time configurations are defined in fsp_cfg/rm_fs2012_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Device Type	• FS2012-1020-NG • FS2012-1100-NG	FS2012-1100-NG	Select FS2012 device type.

Configurations for Sensor > FS2012 Flow Sensor (rm_fs2012)

This module can be added to the Stacks tab via New Stack > Sensor > FS2012 Flow Sensor (rm_fs2012).

Configuration	Options	Default	Description
Name	Manual Entry	g_fs2012_sensor0	Module name.
Callback	Name must be a valid C symbol	fs2012_callback	A user callback function can be provided.

Pin Configuration

This module uses SDA and SCL pins of I2C Master and SCI I2C.

Usage Notes

FS2012 datasheet is here. The module only supports FS2012-1020-NG and FS2012-1100-NG.

If an RTOS is used, blocking and bus lock is available.

- If blocking of an I2C bus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.

Bus Initialization

The FS2012 interface expects a bus instance to be opened before opening any FS2012 device. The interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate I2C Master Interface open call.

Examples

Basic Example

This is a basic example of minimal use of FS2012 sensor implementation in an application.

```
void rm_fs2012_basic_example (void)
{
 fsp_err_t            err = FSP_SUCCESS;
 rm_fsxxxx_raw_data_t fs2012_raw_data;
 rm_fsxxxx_data_t    fs2012_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *) g_fs2012_cfg.p_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                            p_extend->p_blocking_semaphore->p_semaphore_name,
                            (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
```

```
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                        p_extend->p_bus_recursive_mutex->p_mutex_name,
                        TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_FS2012_Open(&g_fs2012_ctrl, &g_fs2012_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (true)
    {
       g_flag = 0;
 /* Read ADC Data from FS2012 */
 RM_FS2012_Read(&g_fs2012_ctrl,
                      &fs2012_raw_data);
 while (0 == g_flag)
      {
 /* Wait callback */
      }
 /* Calculate Flow value from ADC data */
 RM_FS2012_DataCalculate(&g_fs2012_ctrl, &fs2012_raw_data, &fs2012_data);
```

```
/* FS2012 sample rate. See table 4 on the page 5 of the datasheet. */
/* Gas : 409.6ms, Liquid : 716.8ms */
R_BSP_SoftwareDelay(FS2012_GAS_SAMPLE_RATE, BSP_DELAY_UNITS_MICROSECONDS);
   }
}
```

Data Structures

struct	rm_fs2012_instance_ctrl_t

Data Structure Documentation

◆ rm_fs2012_instance_ctrl_t

struct rm_fs2012_instance_ctrl_t

FS2012 Control Block

Data Fields

uint32_t	open
	Open flag.
rm_fsxxxx_cfg_t const *	p_cfg
	Pointer to FS2012 Configuration.
rm_comms_instance_t const *	p_comms_i2c_instance
	Pointer of I2C Communications Middleware instance structure.
void const *	p_context
	Pointer to the user-provided context.

Function Documentation

◆ RM_FS2012_Open()

fsp_err_t RM_FS2012_Open (rm_fsxxxx_ctrl_t *const *p_api_ctrl*, rm_fsxxxx_cfg_t const *const *p_cfg*)

Opens and configures the FS2012 Middle module. Implements rm_fsxxxx_api_t::open.

Example:

```
err = RM_FS2012_Open(&g_fs2012_ctrl, &g_fs2012_cfg);
```

Return values

FSP_SUCCESS	FS2012 successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.

◆ RM_FS2012_Close()

fsp_err_t RM_FS2012_Close (rm_fsxxxx_ctrl_t *const *p_api_ctrl*)

Disables specified FS2012 control block. Implements rm_fsxxxx_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_FS2012_Read()

fsp_err_t RM_FS2012_Read (rm_fsxxxx_ctrl_t *const *p_api_ctrl*, rm_fsxxxx_raw_data_t *const *p_raw_data*)

Reads ADC data from FS2012. Implements rm_fsxxxx_api_t::read.

Return values

FSP_SUCCESS	Successfully data decoded.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_FS2012_DataCalculate()

fsp_err_t RM_FS2012_DataCalculate (rm_fsxxxx_ctrl_t *const *p_api_ctrl*, rm_fsxxxx_raw_data_t *const *p_raw_data*, rm_fsxxxx_data_t *const *p_fs2012_data*)

Calculates flow from ADC data. Unit of Gas flow is SLPM (Standard liter per minute) Unit of Liquid flow is SCCM (Standard cubic centimeter per minute) Implements rm_fsxxxx_api_t::dataCalculate.

Return values

FSP_SUCCESS	Successfully data decoded.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

4.2.92 Azure RTOS GUIX Port (rm_guix_port)

Modules

Functions

UINT	rm_guix_port_hw_initialize (GX_DISPLAY *p_display)

Detailed Description

Overview

The Azure RTOS GUIX Port module provides the configuration and hardware acceleration support necessary for use of GUIX on RA products. The port provides full integration with the graphics peripherals (GLCDC, DRW and JPEG).

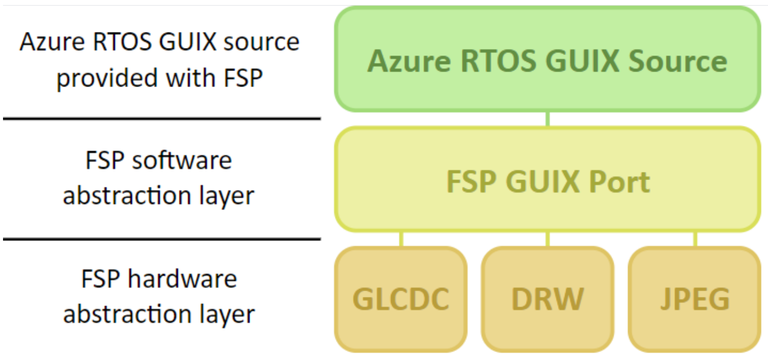


Figure 204: Azure RTOS GUIX Port Block Diagram

Note

This port layer primarily enables hardware acceleration and background handling of many display operations and does not contain code intended to be directly called by the user. For information about how to use GUIX and GUIX Studio (including example code) please consult the Azure RTOS GUIX documentation.

Hardware Acceleration

The following functions are currently performed with hardware acceleration:

- DRW Engine (D/AVE 2D Port Interface (r_drw))
  - Drawing bitmaps
  - 8, 4 and 1bpp uncompressed and compressed (RLE) font rendering
  - Line and shape drawing
  - Anti-aliased operations
    - Circle stroke and fill
    - Polygon stroke and fill
    - Lines and arcs
- JPEG Codec (r_jpeg)
  - JPEG decoding
- Graphics LCD Controller (r_glcdc)
  - Brightness, contrast and gamma correction
  - Pixel format conversion (framebuffer to LCD)

Configuration

Build Time Configurations for gx

The following build time configurations are defined in fsp_cfg/azure/gx/gx_user.h:

Configuration	Options	Default	Description
Hardware Acceleration > JPEG Codec Support	• Enabled • Disabled	Enabled	Select whether or not to use the JPEG Codec for hardware acceleration.
Hardware Acceleration > DRW Engine Support	• Enabled • Disabled	Enabled	Select whether or not to use the DRW Engine for hardware acceleration.
Hardware Acceleration > Max DRW Operations	Value must be a positive integer	85	Specifies the maximum number of DRW operations before flushing the display list. Reducing this value may reduce the peak heap used by the application but may reduce performance.
Internal Thread > Stack Size	Value must be greater than zero	4096	GUIX internal thread stack size in bytes. Must be greater than zero.
Internal Thread >	Value must be between	30	Priority of the GUIX

Priority	0 to 31		Internal Thread. The value must be between 0 to 31.
Internal Thread > Time Slice	Value must be a non-negative integer	10	Time Slice value of the GUIX Internal Thread. The value must be between 0 (TX_NO_TIME_SLICE) to 0xFFFFFFFF.
System Timer (ms)	Value must be greater than or equal to 10	20	GUIX system timer period (GX_SYSTEM_TIMER_MS). This value will be internally converted to RTOS ticks and will be rounded down to the next smallest multiple of the RTOS tick period (1000 / TX_TIMER_TICKS_PER_SECOND).
Multithread Support	• Disabled • Enabled	Enabled	Must be enabled if GUIX functions are called from multiple threads. Set to Disabled when calling GUIX from only one thread to reduce system overhead.
UTF8 Support	• Disabled • Enabled	Enabled	Select whether to enable or disable support for UTF8 characters.
Event Queue Size	Value must be greater than zero	48	Maximum number of events in the GUIX event queue.
Enable GX_WIDGET User Data	• Enabled • Disabled	Disabled	Set to Enabled to to use the gx_widget_user_data member in the GX_WIDGET structure.

Build Time Configurations for rm_guix_port

The following build time configurations are defined in fsp_cfg/middleware/rm_guix_port_cfg.h:

Configuration	Options	Default	Description
DRW Buffer Cache	• Enabled • Disabled	Enabled	Enabling this option significantly improves DRW Engine performance. Set to

			Disabled only if Display underflow events are triggered under high graphics load.

Configurations for Graphics > Azure RTOS GUIX Port (rm_guix_port)

Configuration	Options	Default	Description
Display Rotation > Screen Orientation	• None • CW (90 degrees) • FLIP (180 degrees) • CCW (270 degrees)	None	Select the display orientation specified in the GUIX Studio project. The Canvas Buffer must be enabled when rotating 180 degrees (FLIP).
Display Rotation > Use Canvas Buffer	• Enabled • Disabled	Disabled	When screen rotation is set to a value other than 0 a canvas buffer must be used. The canvas buffer size will be the same as a frame buffer for the display module.
Display Rotation > Canvas Buffer Memory Section	This property must be a valid section name	bss	Specify the memory section where the GUIX Canvas Buffer will be allocated.
JPEG Decoding > Work Buffer Size	Must be a valid integer	0xC800	Specify the JPEG work buffer size in bytes. A larger buffer can reduce JPEG decode/draw times. The buffer will not be allocated if JPEG Codec support is disabled. Unless you are sure of the subsampling used in and the size of the input JPEG images it is recommended to allocate at least 16 framebuffer lines of memory.
JPEG Decoding > Buffer Memory Section	This property must be a valid section name	bss	Specify the memory section where the JPEG Work Buffer will be allocated.
Name	Name must be a valid C symbol	g_rm_guix_port0	Module name.
Target Display Layer	• Graphics Layer	Graphics Layer 1	Specify which graphics

	1 • Graphics Layer 2		screen to inherit the buffer and display dimensions from.
Callback Function	Must be a valid C symbol	NULL	If a callback function is provided it will be called when Display events occur.

Hardware Configuration

No clocks or pins are directly required by this module. Please consult the submodules' documentation for their requirements.

Usage Notes

Getting Started

To get started with GUIX in an RA project the following must be performed:

1. In e2 studio, open the RA Configuration editor for your GUIX project
2. Select or create a thread
3. Add GUIX to your project in the Stacks view by clicking **New Stack -> Azure RTOS -> GUIX**
4. Ensure the configuration options for GUIX and the port layer are set as necessary for your application
5. Set the proporties for the GLCDC module to match the timing and memory requirements of your display panel
6. Set the input color format in the GLCDC module (Input -> Graphics Layer * -> General -> Color format) and the output color format in the JPEG Codec module if applicable (Decode -> Output color format) per your project specification
7. Click the BSP tab in the configuration editor and confirm the heap size in the Properties pane is sufficient (see Note below)
8. Click Generate Project Content to save and commit configuration changes
9. Drop the Quick Setup entry in Developer Assistance into the desired thread entry C file and update the items marked with TODO as necessary
10. Call the Quick Setup function from the thread entry function (or where desired)

At this point the project is now ready to build and run your GUIX Studio project. Please refer to the documentation for Azure RTOS GUIX and GUIX Studio for details on how to create and edit a GUI application.

Note

It is recommended to start with 8K-32K of heap to begin development. Actual heap use is typically far lower than this but must be characterized by the developer.

Using Hardware Acceleration

In most cases there is no need to perform additional configuration to ensure the DRW Engine is used. However, there are some guidelines that should be followed:

- Avoid transparent pixelmaps in 8-bit display mode as they are rendered in software. In particular, ensure PNGs to be used in 8bpp GUIX Studio projects are saved without transparency data if no transparency is needed.
- The following items may require a large heap to draw successfully:

  - Polygons (more sides = more heap)
  - Filled arcs and ellipses (more framebuffer lines occupied = more heap)
  - gx_canvas_pixelmap_tile (more tiles = more heap)
- When using hardware acceleration, images used for tile fill of shapes must have dimensions that are a power of 2. This limitation does not apply to gx_canvas_pixelmap_tile as well as certain arc/ellipse fill functions as GUIX manually draws pixelmaps to fill these shapes (at the expense of heap space).

Examples

Basic Example

This is a basic example demonstrating how to get GUIX up and running given an existing GUIX Studio project. A template for this code is available in Developer Assistance for the GUIX Port module.

Note

GUIX manages the GLCDC, DRW and JPEG Codec submodules internally; they do not need to be opened directly.

```
GX_WINDOW_ROOT * p_window_root;
void guix_user_start (void)
{
 /* Initialize GUIX */
    gx_system_initialize();
 /* Configure GUIX Studio project main display and get a pointer to the root window
*/
    gx_studio_display_configure(MAIN_DISPLAY,
 rm_guix_port_hw_initialize,
                               MAIN_DISPLAY_LANGUAGE_ENGLISH,
                               MAIN_DISPLAY_THEME,
                               &p_window_root);
 /* Set pointer to the first buffer generated by the configuration
(rm_guix_port_canvas) */
    gx_canvas_memory_define(p_window_root->gx_window_root_canvas,
                            rm_guix_port_canvas,

p_window_root->gx_window_root_canvas->gx_canvas_memory_size);
 /* Create and show the root window in the GUIX Studio project */
    gx_studio_named_widget_create("root_widget_name", (GX_WIDGET *) p_window_root,
GX_NULL);
```

```
    gx_widget_show(p_window_root);
 /* Start GUIX */
    gx_system_start();
 /* GUIX will continue to run in its own thread */
}
```

Data Structures

struct	rm_guix_port_callback_args_t

Enumerations

enum	rm_guix_port_device_t
enum	rm_guix_port_event_t

Data Structure Documentation

◆ rm_guix_port_callback_args_t

struct rm_guix_port_callback_args_t		
Callback arguments for the FSP GUIX Port		
Data Fields		
rm_guix_port_device_t	device	Device code.
rm_guix_port_event_t	event	Event code of the low level hardware.
uint32_t	error	Error code if RM_GUIX_PORT_EVENT_ERROR.

Enumeration Type Documentation

◆ rm_guix_port_device_t

enum rm_guix_port_device_t	
Low level device code for the GUIX	
Enumerator	
RM_GUIX_PORT_DEVICE_NONE	Non hardware.
RM_GUIX_PORT_DEVICE_DISPLAY	Display device.
RM_GUIX_PORT_DEVICE_DRW	2D Graphics Engine
RM_GUIX_PORT_DEVICE_JPEG	JPEG Codec.

◆ **rm_guix_port_event_t**

enum rm_guix_port_event_t

Display event codes

Enumerator	
RM_GUIX_PORT_EVENT_ERROR	Low level driver error occurs.
RM_GUIX_PORT_EVENT_DISPLAY_VSYNC	Display interface VSYNC.
RM_GUIX_PORT_EVENT_UNDERFLOW	Display interface underflow.

Function Documentation

◆ **rm_guix_port_hw_initialize()**

UINT rm_guix_port_hw_initialize (GX_DISPLAY * *p_display*)

Callback function to be passed to gx_studio_display_configure in order to start hardware modules.

Example:

```
 /* Configure GUIX Studio project main display and get a pointer to the root window
*/
    gx_studio_display_configure(MAIN_DISPLAY,
 rm_guix_port_hw_initialize,
                                MAIN_DISPLAY_LANGUAGE_ENGLISH,
                                MAIN_DISPLAY_THEME,
                                &p_window_root);
```

Note

This function should only be called by GUIX.

Return values

GX_SUCCESS	Device driver setup is successfully done.
GX_FAILURE	Device driver setup failed.

4.2.93 HS300X Sensor Middleware (rm_hs300x)

Modules

Functions

fsp_err_t	RM_HS300X_Open (rm_hs300x_ctrl_t *const p_api_ctrl, rm_hs300x_cfg_t const *const p_cfg) Opens and configures the HS300X Middle module. Implements rm_hs300x_api_t::open. More...
fsp_err_t	RM_HS300X_Close (rm_hs300x_ctrl_t *const p_api_ctrl) Disables specified HS300X control block. Implements rm_hs300x_api_t::close. More...
fsp_err_t	RM_HS300X_MeasurementStart (rm_hs300x_ctrl_t *const p_api_ctrl) This function should be called when start a measurement and when measurement data is stale data. Sends the slave address to the hs300x and start a measurement. Implements rm_hs300x_api_t::measurementStart. More...
fsp_err_t	RM_HS300X_Read (rm_hs300x_ctrl_t *const p_api_ctrl, rm_hs300x_raw_data_t *const p_raw_data) Reads ADC data from HS300X. Implements rm_hs300x_api_t::read. More...
fsp_err_t	RM_HS300X_DataCalculate (rm_hs300x_ctrl_t *const p_api_ctrl, rm_hs300x_raw_data_t *const p_raw_data, rm_hs300x_data_t *const p_hs300x_data) Calculates humidity [RH] and temperature [Celsius] from ADC data. Implements rm_hs300x_api_t::dataCalculate. More...
fsp_err_t	RM_HS300X_ProgrammingModeEnter (rm_hs300x_ctrl_t *const p_api_ctrl) This function must be called within 10ms after applying power to the sensor. Sends the commands to enter the programming mode. After calling this function, please wait 120us. Implements rm_hs300x_api_t::programmingModeEnter. More...
fsp_err_t	RM_HS300X_ResolutionChange (rm_hs300x_ctrl_t *const p_api_ctrl, rm_hs300x_data_type_t const data_type, rm_hs300x_resolution_t const resolution) This function must be called after calling the RM_HS300X_ProgrammingModeEnter function. Changes the sensor resolution. This function blocks for 120 us software delay plus 9 bytes on the I2C bus. After calling this function, 14ms must be waited. Failure to comply with these times may result in data corruption and introduce errors in sensor measurements. Implements rm_hs300x_api_t::resolutionChange. More...

fsp_err_t	RM_HS300X_SensorIdGet (rm_hs300x_ctrl_t *const p_api_ctrl, uint32_t *const p_sensor_id) This function must be called after calling the RM_HS300X_ProgrammingModeEnter function. Gets the sensor ID. This function blocks for 240 us software delay plus 12 bytes on the I2C bus. Implements rm_hs300x_api_t::sensorIdGet. More...
fsp_err_t	RM_HS300X_ProgrammingModeExit (rm_hs300x_ctrl_t *const p_api_ctrl) This function must be called after calling the RM_HS300X_ProgrammingModeEnter function. This function must be called to return to normal sensor operation and perform measurements. Sends the commands to exit the programming mode. Implements rm_hs300x_api_t::programmingModeExit. More...

Detailed Description

Middleware to implement the HS300X sensor interface. This module implements the HS300X Middleware Interface.

Overview

Features

The HS300X sensor interface implementation has the following key features:

- Starting a measurement at any time
- Getting ADC data from the sensor
- Calculating humidity and temperature value from getting ADC data
- Changing the sensor resolution
- Getting the sensor ID

Configuration

Build Time Configurations for rm_hs300x

The following build time configurations are defined in fsp_cfg/rm_hs300x_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Data type	• Both humidity and temperature • Humidity only	Both humidity and temperature	Select Getting humidity only and both humidity and temperature.

Programming Mode	• ON • OFF	OFF	If selected the programming mode can be entered.

Configurations for Sensor > HS300X Temperature/Humidity Sensor (rm_hs300x)

This module can be added to the Stacks tab via New Stack > Sensor > HS300X Temperature/Humidity Sensor (rm_hs300x).

Configuration	Options	Default	Description
Name	Manual Entry	g_hs300x_sensor0	Module name.
Callback	Name must be a valid C symbol	hs300x_callback	A user callback function can be provided.

Pin Configuration

This module use SDA and SCL pins of I2C Master and SCI I2C.

Usage Notes

HS300x datasheet is here.
If ADC data is valid and calculating humidity and temperature is finished, it is needed to start a measurement again. If ADC data is invalid, it is needed to read ADC data from HS300x again.

If changing the sensor resolution and getting the sensor ID, RM_HS300X_ProgrammingModeEnter function must be called within 10ms after applying power to the sensor. Entering the programming mode takes 120us. Thresore, after calling RM_HS300X_ProgrammingModeEnter function, please wait 120us. After calling RM_HS300X_ResolutionChange function, 14ms must be waited because failure to comply with these times may result in data corruption and introduce errors in sensor measurements.

If an RTOS is used, blocking and bus lock is available.

- If blocking of an I2C bus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.

Bus Initialization

The HS300X interface expects a bus instance to be opened before opening any HS300X device. The interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate I2C Master Interface open call.

Examples

Basic Example

This is a basic example of minimal use of HS300X sensor implementation in an application.

```
void rm_hs300x_basic_example (void)
{
```

```
 fsp_err_t            err = FSP_SUCCESS;
 rm_hs300x_raw_data_t hs300x_raw_data;
 rm_hs300x_data_t    hs300x_data;
    uint8_t             calculated_flag = 0;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *) g_hs300x_cfg.p_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                       p_extend->p_bus_recursive_mutex->p_mutex_name,
                       TX_INHERIT);
```

```
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_HS300X_Open(&g_hs300x_ctrl, &g_hs300x_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
#if RM_HS300X_CFG_PROGRAMMING_MODE
    uint32_t sensor_id;
    g_flag = 0;
 /* Enter the programming mode. This must be called within 10ms after applying power.
*/
 RM_HS300X_ProgrammingModeEnter(&g_hs300x_ctrl);
 while (0 == g_flag)
    {
 /* Wait callback */
    }
 /* Delay 120us. Entering the programming mode takes 120us. */
 R_BSP_SoftwareDelay(120, BSP_DELAY_UNITS_MICROSECONDS);
 /* Get the sensor ID */
 RM_HS300X_SensorIdGet(&g_hs300x_ctrl, &sensor_id);
    g_flag = 0;
 /* Change the humidity resolution */
 RM_HS300X_ResolutionChange(&g_hs300x_ctrl, RM_HS300X_HUMIDITY_DATA,
RM_HS300X_RESOLUTION_8BIT);
 while (0 == g_flag)
    {
 /* Wait callback */
    }
 /* Delay 14ms. Failure to comply with these times may result in data corruption and
introduce errors in sensor measurements. */
```

```
 R_BSP_SoftwareDelay(14, BSP_DELAY_UNITS_MILLISECONDS);
    g_flag = 0;
 /* Change the temperature resolution */
 RM_HS300X_ResolutionChange(&g_hs300x_ctrl, RM_HS300X_TEMPERATURE_DATA,
RM_HS300X_RESOLUTION_8BIT);
 while (0 == g_flag)
    {
 /* Wait callback */
    }
 /* Delay 14ms. Failure to comply with these times may result in data corruption and
introduce errors in sensor measurements. */
 R_BSP_SoftwareDelay(14, BSP_DELAY_UNITS_MILLISECONDS);
    g_flag = 0;
 /* Exit the programming mode */
 RM_HS300X_ProgrammingModeExit(&g_hs300x_ctrl);
 while (0 == g_flag)
    {
 /* Wait callback */
    }
#endif
 while (true)
    {
      g_flag = 0;
 /* Start Measurement */
 RM_HS300X_MeasurementStart(&g_hs300x_ctrl);
 while (0 == g_flag)
      {
 /* Wait callback */
      }
 do
      {
      g_flag = 0;
 /* Read ADC Data from HS300X */
 RM_HS300X_Read(&g_hs300x_ctrl, &hs300x_raw_data);
```

```
 while (0 == g_flag)
      {
 /* Wait callback */
      }
 /* Calculate Humidity and Temperatuere values from ADC data */
            err = RM_HS300X_DataCalculate(&g_hs300x_ctrl, &hs300x_raw_data,
&hs300x_data);
 if (FSP_SUCCESS == err)
      {
                calculated_flag = 1;
      }
 else if (FSP_ERR_SENSOR_INVALID_DATA == err)
      {
 /* Stale data */
                calculated_flag = 0;
      }
 else
      {
      handle_error(err);
      }
      } while (0 == calculated_flag);
 /* Wait 4 seconds. See table 4 on the page 6 of the datasheet. */
 R_BSP_SoftwareDelay(4, BSP_DELAY_UNITS_SECONDS);
    }
}
```

Data Structures

struct	rm_hs300x_programmnig_mode_params_t
struct	rm_hs300x_instance_ctrl_t

Data Structure Documentation

◆ rm_hs300x_programmnig_mode_params_t

struct rm_hs300x_programmnig_mode_params_t
HS300X programming mode process block

Data Fields		
volatile bool	enter	Enter flag.
volatile bool	blocking	Blocking flag.
volatile bool	communication_finished	Communication flag for blocking.
volatile rm_hs300x_event_t	event	Callback event.

◆ rm_hs300x_instance_ctrl_t

struct rm_hs300x_instance_ctrl_t

HS300x Control Block

Data Fields

uint32_t	open
	Open flag.
rm_hs300x_cfg_t const *	p_cfg
	Pointer to HS300X Configuration.
rm_comms_instance_t const *	p_comms_i2c_instance
	Pointer of I2C Communications Middleware instance structure.
void const *	p_context
	Pointer to the user-provided context.
rm_hs300x_programmnig_mode_params_t	programming_mode
	Programming mode flag.

Function Documentation

◆ RM_HS300X_Open()

fsp_err_t RM_HS300X_Open (rm_hs300x_ctrl_t *const *p_api_ctrl*, rm_hs300x_cfg_t const *const *p_cfg*)

Opens and configures the HS300X Middle module. Implements rm_hs300x_api_t::open.

Example:

```
err = RM_HS300X_Open(&g_hs300x_ctrl, &g_hs300x_cfg);
```

Return values

FSP_SUCCESS	HS300X successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.

◆ RM_HS300X_Close()

fsp_err_t RM_HS300X_Close (rm_hs300x_ctrl_t *const *p_api_ctrl*)

Disables specified HS300X control block. Implements rm_hs300x_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_HS300X_MeasurementStart()

fsp_err_t RM_HS300X_MeasurementStart (rm_hs300x_ctrl_t *const *p_api_ctrl*)

This function should be called when start a measurement and when measurement data is stale data. Sends the slave address to the hs300x and start a measurement. Implements rm_hs300x_api_t::measurementStart.

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_HS300X_Read()

fsp_err_t RM_HS300X_Read (rm_hs300x_ctrl_t *const *p_api_ctrl*, rm_hs300x_raw_data_t *const *p_raw_data*)

Reads ADC data from HS300X. Implements rm_hs300x_api_t::read.

Return values

FSP_SUCCESS	Successfully data decoded.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_HS300X_DataCalculate()

fsp_err_t RM_HS300X_DataCalculate (rm_hs300x_ctrl_t *const *p_api_ctrl*, rm_hs300x_raw_data_t *const *p_raw_data*, rm_hs300x_data_t *const *p_hs300x_data*)

Calculates humidity [RH] and temperature [Celsius] from ADC data. Implements rm_hs300x_api_t::dataCalculate.

Return values

FSP_SUCCESS	Successfully data decoded.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_SENSOR_INVALID_DATA	Data is invalid.

◆ RM_HS300X_ProgrammingModeEnter()

fsp_err_t RM_HS300X_ProgrammingModeEnter (rm_hs300x_ctrl_t *const *p_api_ctrl*)

This function must be called within 10ms after applying power to the sensor. Sends the commands to enter the programming mode. After calling this function, please wait 120us. Implements rm_hs300x_api_t::programmingModeEnter.

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_UNSUPPORTED	Programming mode is not supported.

◆ RM_HS300X_ResolutionChange()

fsp_err_t RM_HS300X_ResolutionChange (rm_hs300x_ctrl_t *const *p_api_ctrl*, rm_hs300x_data_type_t const *data_type*, rm_hs300x_resolution_t const *resolution*)

This function must be called after calling the RM_HS300X_ProgrammingModeEnter function. Changes the sensor resolution. This function blocks for 120 us software delay plus 9 bytes on the I2C bus. After calling this function, 14ms must be waited. Failure to comply with these times may result in data corruption and introduce errors in sensor measurements. Implements rm_hs300x_api_t::resolutionChange.

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_MODE	Module is not entering the programming mode.
FSP_ERR_ABORTED	Communication is aborted.
FSP_ERR_TIMEOUT	Communication is timeout.
FSP_ERR_UNSUPPORTED	Programming mode is not supported.

◆ RM_HS300X_SensorIdGet()

fsp_err_t RM_HS300X_SensorIdGet (rm_hs300x_ctrl_t *const *p_api_ctrl*, uint32_t *const *p_sensor_id*)

This function must be called after calling the RM_HS300X_ProgrammingModeEnter function. Gets the sensor ID. This function blocks for 240 us software delay plus 12 bytes on the I2C bus. Implements rm_hs300x_api_t::sensorIdGet.

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_MODE	Module is not entering the programming mode.
FSP_ERR_ABORTED	Communication is aborted.
FSP_ERR_TIMEOUT	Communication is timeout.
FSP_ERR_UNSUPPORTED	Programming mode is not supported.

◆ RM_HS300X_ProgrammingModeExit()

fsp_err_t RM_HS300X_ProgrammingModeExit (rm_hs300x_ctrl_t *const *p_api_ctrl*)

This function must be called after calling the RM_HS300X_ProgrammingModeEnter function. This function must be called to return to normal sensor operation and perform measurements. Sends the commands to exit the programming mode. Implements rm_hs300x_api_t::programmingModeExit.

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_MODE	Module is not entering the programming mode.
FSP_ERR_UNSUPPORTED	Programming mode is not supported.

4.2.94 Azure RTOS LevelX NOR SPI Driver (rm_levelx_nor_spi)

Modules

Functions

fsp_err_t	RM_LEVELX_NOR_SPI_Open (rm_levelx_nor_spi_instance_ctrl_t *const p_ctrl, rm_levelx_nor_spi_cfg_t const *const p_cfg) Initializes LevelX NOR SPI port read/write and control. More...
fsp_err_t	RM_LEVELX_NOR_SPI_Read (rm_levelx_nor_spi_instance_ctrl_t *const p_ctrl, ULONG *const p_flash_addr, ULONG *const p_dest, ULONG word_count) LevelX NOR driver "read sector" service. More...
fsp_err_t	RM_LEVELX_NOR_SPI_Write (rm_levelx_nor_spi_instance_ctrl_t *const p_ctrl, ULONG *const p_flash_addr, ULONG *const p_src, ULONG word_count) LevelX NOR driver "write sector" service. More...
fsp_err_t	RM_LEVELX_NOR_SPI_BlockErase (rm_levelx_nor_spi_instance_ctrl_t *const p_ctrl, ULONG block, ULONG erase_count) LevelX NOR driver "block erase" service. More...

fsp_err_t	RM_LEVELX_NOR_SPI_BlockErasedVerify (rm_levelx_nor_spi_instance_ctrl_t *const p_ctrl, ULONG block) LevelX NOR driver "block erased verify" service. More...
fsp_err_t	RM_LEVELX_NOR_SPI_Close (rm_levelx_nor_spi_instance_ctrl_t *const p_ctrl) LevelX NOR driver close service. More...

Detailed Description

Middleware for using Azure RTOS LevelX on NOR SPI memory.

Overview

This module provides the hardware port layer for LevelX on NOR SPI flash memory. Setup for this module is done solely through calling LevelX APIs. Please refer to the LevelX API reference: https://docs.microsoft.com/en-us/azure/rtos/levelx/

Configuration

Build Time Configurations for rm_levelx_nor_spi

The following build time configurations are defined in fsp_cfg/middleware/rm_levelx_nor_spi_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	Selects if code for parameter checking is to be included in the build.
Write Verify	• Enabled • Disabled	Disabled	When enabled reads back data written to SPI memory in order to verify it.
Page Buffer Size (bytes)	Size should be greater than zero	256	When direct read is enabled in LevelX a situation can occur where the driver has to write to SPI memory with the source locaiton also being within the SPI memory address range. In this situation the driver needs a buffer that is at least the same size as a page in order to

			temporarily store data to write out.

Configurations for Storage > LevelX NOR Port (rm_levelx_nor_spi)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_levelx_nor_spi0	Module name.
Memory Start Adress Offset (bytes)	Offset should be greater than or equal to zero	0	Enter the starting offset to use in the SPI memory. The starting address for LevelX memory will be the SPI memory base address plus this offset.
Memory Size (bytes)	Size should be greater than zero	33554432	Enter the size that the LevelX Memory should be. This can be smaller than the SPI memory size in order to use a subset of SPI memory.
Poll Status Count	Poll Status Count should be greater than or equal to zero	0xFFFFFFFF	Number of times to poll for operation complete status for blocking memory operations.

Build Time Configurations for lx

The following build time configurations are defined in fsp_cfg/azure/lx/lx_user.h:

Configuration	Options	Default	Description
NOR > Direct Read	• Enabled (default) • Disabled	Enabled (default)	When enabled, this option bypasses the NOR flash driver read routine in favor or reading the NOR memory directly, resulting in a significant performance increase.
NOR > Free Sector Data Verify	• Enabled • Disabled (default)	Disabled (default)	When enabled, this causes the LevelX NOR instance open logic to verify free NOR sectors are all ones.
NOR > Extended Cache	• Enabled (default) • Disabled	Enabled (default)	Enables the extended NOR cache.
NOR > Extended Cache Size	Manual Entry		If not set this value defaults to 8, which

			represents a maximum of 8 sectors that can be cached in a NOR instance.
NOR > Sector Mapping Cache Size	Value must be greater than or equal to 8 and a power of 2, or empty		If not set this value defaults to 16 and defines the logical sector mapping cache size. Large values improve performance, but cost memory. The minimum size is 8 and all values must be a power of 2.
NAND > Sector Mapping Cache Size	Value must be greater than or equal to 8 and a power of 2, or empty		If not set this value defaults to 128 and defines the logical sector mapping cache size. Large values improve performance, but cost memory. The minimum size is 8 and all values must be a power of 2.
NAND > Flash Direct Mapping Cache	• Enabled • Disabled (default)	Disabled (default)	When enabled, this creates a direct mapping cache, such that there are no cache misses. It also required that LX_NAND_SECTOR_MAPPING_CACHE_SIZE represents the exact number of total pages in your flash device.
Thread Safe	• Enabled • Disabled (default)	Disabled (default)	When enabled, this makes LevelX thread-safe by using a ThreadX mutex object throughout the API.
Standalone Mode	• Enabled • Disabled (default)	Disabled (default)	When enabled, allows LevelX to be used in standalone mode (without Azure RTOS).

Usage Notes

Pending during Erase/Write

The LevelX NOR SPI driver is blocking on all SPI operations and will poll device status for operation completion on Writes and Erases. A callback can be provided by the user to wait with an OS-specific thread wait in these instances.

Closing the driver

When lx_nor_flash_close is called to close the LevelX instance it does not call any services within the LevelX NOR SPI driver to close out the driver instance. The user should call the generated close function (i.e. g_rm_levelx_nor_spi0_close) in order to close out the driver instance.

Erasing Flash Memory Prior to Usage

The area of the flash memory being used for the LevelX instance should be erased using the lower level flash API prior to usage. Otherwise, LevelX API may fail on lx_nor_flash_open due to any areas in flash memory that have been written/set.

Examples

Basic Example

This is a basic example of using the LevelX NOR SPI driver with the LevelX API in an application.

```
#define RM_LEVELX_NOR_SPI_EXAMPLE_SECTOR_SIZE (512)
#define RM_LEVELX_NOR_SPI_EXAMPLE_BUFFER_FILL_VALUE (0xA5)
#define RM_LEVELX_NOR_SPI_EXAMPLE_SPI_SECTOR_SIZE (4096)
extern rm_levelx_nor_spi_instance_ctrl_t g_levelx_nor_spi0_ctrl;
extern rm_levelx_nor_spi_cfg_t g_levelx_nor_spi0_cfg;
extern LX_NOR_FLASH g_lx_nor_flash0;
void rm_levelx_nor_spi_example (void)
{
    uint8_t                read_buffer[RM_LEVELX_NOR_SPI_EXAMPLE_SECTOR_SIZE];
    uint8_t                write_buffer[RM_LEVELX_NOR_SPI_EXAMPLE_SECTOR_SIZE];
 spi_flash_instance_t * p_spi_flash_instance = (spi_flash_instance_t *)
g_levelx_nor_spi0_cfg.p_lower_lvl;
 spi_flash_status_t    status;
    memset(write_buffer, RM_LEVELX_NOR_SPI_EXAMPLE_BUFFER_FILL_VALUE, sizeof
(write_buffer));
 /* Erase flash prior to usage */
 fsp_err_t err = p_spi_flash_instance->p_api->open(p_spi_flash_instance->p_ctrl,
p_spi_flash_instance->p_cfg);
    assert(FSP_SUCCESS == err);
 for (uint32_t i = g_levelx_nor_spi0_cfg.address_offset;
         i < g_levelx_nor_spi0_cfg.size;
         i += RM_LEVELX_NOR_SPI_EXAMPLE_SPI_SECTOR_SIZE)
    {
```

```
       err = p_spi_flash_instance->p_api->erase(p_spi_flash_instance->p_ctrl,
                                                         (uint8_t *)
g_levelx_nor_spi0_cfg.base_address + i,

RM_LEVELX_NOR_SPI_EXAMPLE_SPI_SECTOR_SIZE);
      assert(FSP_SUCCESS == err);
       status.write_in_progress = true;
 while (status.write_in_progress)
      {
            err =
p_spi_flash_instance->p_api->statusGet(p_spi_flash_instance->p_ctrl, &status);
      assert(FSP_SUCCESS == err);
       }
    }
    err = p_spi_flash_instance->p_api->close(p_spi_flash_instance->p_ctrl);
    assert(FSP_SUCCESS == err);
 /* Initialize LevelX */
    lx_nor_flash_initialize();
    UINT lx_err = lx_nor_flash_open(&g_lx_nor_flash0, "LX_NOR_SPI_EXAMPLE",
g_levelx_nor_spi0_initialize);
    handle_lx_error(lx_err);
 /* Write test value to sector 0 then read back to verify */
    lx_err = lx_nor_flash_sector_write(&g_lx_nor_flash0, 0, write_buffer);
    handle_lx_error(lx_err);
    lx_err = lx_nor_flash_sector_read(&g_lx_nor_flash0, 0, read_buffer);
    handle_lx_error(lx_err);
    assert(0 == memcmp(read_buffer, write_buffer, sizeof(read_buffer)));
}
```

Callback Wait Example

This shows how to use the LevelX NOR SPI driver callback with ThreadX in order to wait for operations to complete.

```
/* Callback called by LevelX NOR SPI driver needs to wait on operation. */
```

```
void rm_levelx_nor_spi_callback_wait_example (rm_levelx_nor_spi_callback_args_t *
p_args)
{
 if (p_args->event & RM_LEVELX_NOR_SPI_EVENT_BUSY)
    {
 /* Put the thread to sleep while waiting for operation to complete. */
       tx_thread_sleep(1);
    }
}
```

Data Structures

struct	rm_levelx_nor_spi_callback_args_t
struct	rm_levelx_nor_spi_cfg_t
struct	rm_levelx_nor_spi_instance_ctrl_t

Enumerations

enum	rm_levelx_nor_spi_event_t

Data Structure Documentation

◆ rm_levelx_nor_spi_callback_args_t

struct rm_levelx_nor_spi_callback_args_t

RM_LEVELX_NOR_SPI callback arguments definitions

Data Fields		
rm_levelx_nor_spi_event_t	event	LevelX NOR driver callback event.
void const *	p_context	Placeholder for user data.

◆ rm_levelx_nor_spi_cfg_t

struct rm_levelx_nor_spi_cfg_t

SF_EL_LX_NOR Config Block Type

Data Fields	
spi_flash_instance_t const *	p_lower_lvl
	Lower level memory pointer.
LX_NOR_FLASH *	p_lx_nor_flash

	Pointer to the LevelX nor flash instance.
uint32_t	base_address
	Base address of memory mapped region.
uint32_t	address_offset
	Offset to use subset of available flash size if desired.
uint32_t	size
	Size of the partitioned region.
uint32_t	poll_status_count
	Number of times to poll for operation complete status before returning an error.
void const *	p_context
	Placeholder for user data. Passed to the user callback.
void(*	p_callback)(rm_levelx_nor_spi_callback_args_t *p_args)
	Callback function.

◆ rm_levelx_nor_spi_instance_ctrl_t

struct rm_levelx_nor_spi_instance_ctrl_t

SF_EL_LX_NOR Control Block Type

Data Fields

rm_levelx_nor_spi_cfg_t const *	p_cfg	Pointer to instance configuration.
uint32_t	start_address	Start address of partition to use within memory mapped region.
uint32_t	minimum_erase_size	Minimum erase size of SPI memory.

uint8_t	page_buffer[RM_LEVELX_NOR_SPI_CFG_BUFFER_SIZE]	Page buffer for situations when writing to SPI memory from a source within SPI memory.
uint32_t	open	Used to determine if module is initialized.

Enumeration Type Documentation

◆ rm_levelx_nor_spi_event_t

enum rm_levelx_nor_spi_event_t

Common macro for FSP header files. There is also a corresponding FSP_FOOTER macro at the end of this file. Options for the callback events.

Enumerator	
RM_LEVELX_NOR_SPI_EVENT_BUSY	Pending operation, user can define their own wait functionality.

Function Documentation

◆ RM_LEVELX_NOR_SPI_Open()

fsp_err_t RM_LEVELX_NOR_SPI_Open (rm_levelx_nor_spi_instance_ctrl_t *const *p_ctrl*, rm_levelx_nor_spi_cfg_t const *const *p_cfg*)

Initializes LevelX NOR SPI port read/write and control.

Calls lower level SPI memory functions.

Parameters

[in,out]	p_ctrl	Control block for the LevelX NOR SPI instance.
[in,out]	p_cfg	Configuration for LevelX NOR SPI port.

Return values

FSP_SUCCESS	LevelX NOR driver is successfully opened.
FSP_ERR_ASSERTION	p_ctrl or p_cfg is NULL.
FSP_ERR_ALREADY_OPEN	Driver is already in OPEN state.

Returns

See Common_Error_Codes or lower level drivers for other possible return codes. This function calls

- spi_flash_api_t:open

◆ RM_LEVELX_NOR_SPI_Read()

fsp_err_t RM_LEVELX_NOR_SPI_Read (rm_levelx_nor_spi_instance_ctrl_t *const *p_ctrl*, ULONG *const *p_flash_addr*, ULONG *const *p_dest*, ULONG *word_count*)

LevelX NOR driver "read sector" service.

This is responsible for reading a specific sector in a specific block of the NOR flash. All error checking and correcting logic is the responsibility of this service.

Parameters

[in]	p_ctrl	Control block for the LevelX NOR SPI instance.
[in]	p_flash_addr	Specifies the address of a logical sector within a NOR flash block of memory.
[in,out]	p_dest	Specifies where to place the sector contents.
[in]	word_count	Specifies how many 32-bit words to read.

Return values

FSP_SUCCESS	LevelX NOR flash sector read successful.
FSP_ERR_ASSERTION	p_ctrl, p_flash_addr or p_dest is NULL.
FSP_ERR_NOT_OPEN	Driver not in OPEN state for reading.

Returns

See Common_Error_Codes or lower level drivers for other possible return codes.

◆ RM_LEVELX_NOR_SPI_Write()

fsp_err_t RM_LEVELX_NOR_SPI_Write (rm_levelx_nor_spi_instance_ctrl_t *const *p_ctrl*, ULONG *const *p_flash_addr*, ULONG *const *p_src*, ULONG *word_count*)

LevelX NOR driver "write sector" service.

This is responsible for writing a specific sector into a block of the NOR flash. All error checking is the responsibility of the this service.

Parameters

[in]	p_ctrl	Control block for the LevelX NOR SPI instance.
[in,out]	p_flash_addr	Specifies the address of a logical sector within a NOR flash block of memory.
[in]	p_src	Specifies the source of the write.
[in]	word_count	Specifies how many 32-bit words to write.

Return values

FSP_SUCCESS	LevelX NOR flash sector write successful.
FSP_ERR_ASSERTION	p_ctrl, p_flash_addr or p_src is NULL.
FSP_ERR_NOT_OPEN	Driver not in OPEN state for writing.
FSP_ERR_TIMEOUT	Timeout occurred while waiting for operation to complete.
FSP_ERR_WRITE_FAILED	Verification of Write operation failed.
FSP_ERR_INVALID_ADDRESS	Write address or size falls outside of flash memory range.

Returns

See Common_Error_Codes or lower level drivers for other possible return codes. This function calls

- spi_flash_api_t:write

◆ RM_LEVELX_NOR_SPI_BlockErase()

fsp_err_t RM_LEVELX_NOR_SPI_BlockErase (rm_levelx_nor_spi_instance_ctrl_t *const *p_ctrl*, ULONG *block*, ULONG *erase_count*)

LevelX NOR driver "block erase" service.

This is responsible for erasing the specified block of the NOR flash.

Parameters

[in]	p_ctrl	Control block for the LevelX NOR SPI instance.
[in]	block	Specifies which NOR block to erase.
[in]	erase_count	Provided for diagnostic purposes(currently unused).

Return values

FSP_SUCCESS	LevelX NOR flash block erase successful.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Driver not in OPEN state for erasing.
FSP_ERR_TIMEOUT	Timeout occurred while waiting for operation to complete.

Returns

See Common_Error_Codes or lower level drivers for other possible return codes. This function calls

- spi_flash_api_t:erase

◆ RM_LEVELX_NOR_SPI_BlockErasedVerify()

fsp_err_t RM_LEVELX_NOR_SPI_BlockErasedVerify (rm_levelx_nor_spi_instance_ctrl_t *const *p_ctrl*, ULONG *block*)

LevelX NOR driver "block erased verify" service.

This is responsible for verifying the specified block of the NOR flash is erased.

Parameters

[in]	p_ctrl	Control block for the LevelX NOR SPI instance.
[in]	block	Specifies which block to verify that it is erased.

Return values

FSP_SUCCESS	LevelX flash block erase verification successful.
FSP_ERR_ASSERTION	p_ctrl or lower level driver is NULL.
FSP_ERR_NOT_OPEN	Driver not in OPEN state for verifying.
FSP_ERR_NOT_ERASED	The block is not erased properly.

Returns

See Common_Error_Codes or lower level drivers for other possible return codes.

◆ RM_LEVELX_NOR_SPI_Close()

fsp_err_t RM_LEVELX_NOR_SPI_Close (rm_levelx_nor_spi_instance_ctrl_t *const *p_ctrl*)

LevelX NOR driver close service.

This is responsible for closing the driver properly.

Parameters

[in]	p_ctrl	Control block for the LevelX NOR SPI instance.

Return values

FSP_SUCCESS	LevelX flash is available and is now open for read, write, and control access.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Driver not in OPEN state for closing.

Returns

See Common_Error_Codes or lower level drivers for other possible return codes. This function calls

- spi_flash_api_t:close

4.2.95 LittleFS Flash Port (rm_littlefs_flash)

Modules

Functions

fsp_err_t	RM_LITTLEFS_FLASH_Open (rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg)
fsp_err_t	RM_LITTLEFS_FLASH_Close (rm_littlefs_ctrl_t *const p_ctrl)

Detailed Description

Middleware for the LittleFS File System control on RA MCUs.

Overview

This module provides the hardware port layer for the LittleFS file system. After initializing this module, refer to the LittleFS documentation to use the file system:
https://github.com/ARMmbed/littlefs

Configuration

Build Time Configurations for rm_littlefs_flash

The following build time configurations are defined in fsp_cfg/rm_littlefs_flash_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Storage > LittleFS on Flash (rm_littlefs_flash)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_rm_littlefs0	Module name.
Read Size	Must be a non-negative integer	1	Minimum size of a block read. All read operations will be a multiple of this value.
Program Size	Must be a non-negative integer	4	Minimum size of a block program. All program operations will be a multiple of this value.

Block Size (bytes)	Must be a multiple of 64	128	Size of an erasable block. This does not impact RAM consumption and may be larger than the physical erase size. However, non-inlined files take up at minimum one block. Must be a multiple of the read and program sizes.
Block Count	Manual Entry	(BSP_DATA_FLASH_SIZE_BYTES/128)	Number of erasable blocks on the device.
Block Cycles	Must be an integer	1024	Number of erase cycles before LittleFS evicts metadata logs and moves the metadata to another block. Suggested values are in the range 100-1000, with large values having better performance at the cost of less consistent wear distribution. Set to -1 to disable block-level wear-leveling.
Cache Size	Must be a non-negative integer	64	Size of block caches. Each cache buffers a portion of a block in RAM. The LittleFS needs a read cache, a program cache, and one additional cache per file. Larger caches can improve performance by storing more data and reducing the number of disk accesses. Must be a multiple of the read and program sizes, and a factor of the block size.
Lookahead Size	Must be a non-negative multiple of 8	16	Size of the lookahead buffer in bytes. A larger lookahead buffer increases the number of blocks found during an allocation pass. The lookahead buffer is

			stored as a compact bitmap, so each byte of RAM can track 8 blocks. Must be a multiple of 8.

Common LittleFS Configuration

Build Time Configurations for LittleFS

The following build time configurations are defined in arm/littlefs/lfs_util.h:

Configuration	Options	Default	Description
Custom lfs_util.h	Manual Entry		Add a path to your custom lfs_util.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations.
Thread Safe	• Enabled • Disabled	Disabled	Enables thread safety in LittleFS.
Use Malloc	• Enabled • Disabled	Enabled	Configures the use of malloc by LittleFS.
Use Assert	• Enabled • Disabled	Enabled	Configures the use of assert by LittleFS.
Debug Messages	• Enabled • Disabled	Disabled	Configures debug messages.
Warning Messages	• Enabled • Disabled	Disabled	Configures warning messages.
Error Messages	• Enabled • Disabled	Disabled	Configures error messages.
Trace Messages	• Enabled • Disabled	Disabled	Configures trace messages.
Intrinsics	• Enabled • Disabled	Enabled	Configures intrinsic functions such as __builtin_clz.
Instance Name for STDIO wrapper	Name must be a valid C symbol	g_rm_littlefs0	The rm_littlefs instance name to use with the STDIO wrapper.

Usage Notes

Blocking Read/Write/Erase

The LittleFS port blocks on Read/Write/Erase calls until the operation has completed.

Memory Constraints

The block size defined in the LittleFS configuration must be a multiple of the data flash erase size of the MCU. It must be greater than 104bytes which is the minimum block size of a LittleFS block. For information about data flash erase sizes refer to the "Specifications of the code flash memory and data flash memory" table of the "Flash Memory" chapter's "Overview" section.

Limitations

This module is not thread safe.

Examples

Basic Example

This is a basic example of LittleFS on Flash in an application.

```
extern const rm_littlefs_cfg_t g_rm_littlefs_flash0_cfg;
#ifdef LFS_NO_MALLOC
static uint8_t g_file_buffer[LFS_CACHE_SIZE];
static struct lfs_file_config g_file_cfg =
{
    .buffer = g_file_buffer
};
#endif
void rm_littlefs_example (void)
{
    uint8_t    buffer[30];
    lfs_file_t file;
 /* Open LittleFS Flash port.*/
 fsp_err_t err = RM_LITTLEFS_FLASH_Open(&g_rm_littlefs_flash0_ctrl,
&g_rm_littlefs_flash0_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 /* Format the filesystem. */
 int lfs_err = lfs_format(&g_rm_littlefs_flash0_lfs, &g_rm_littlefs_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Mount the filesystem. */
    lfs_err = lfs_mount(&g_rm_littlefs_flash0_lfs, &g_rm_littlefs_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
```

```
 /* Create a breakfast directory. */
    lfs_err = lfs_mkdir(&g_rm_littlefs_flash0_lfs, "breakfast");
    handle_lfs_error(lfs_err);
 /* Create a file toast in the breakfast directory. */
 const char * path = "breakfast/toast";
#ifdef LFS_NO_MALLOC
 /*************************************************************************************
************************************
  * By default LittleFS uses malloc to allocate buffers. This can be disabled in the
RA Configuration editor.
  * Buffers will be generated from the configuration for the read, program and
lookahead buffers.
  * When opening a file a unique buffer must be passed in for use as a file buffer.
  * The buffer size must be equal to the cache size.
  ************************************************************************************
***********************************/
    lfs_err = lfs_file_opencfg(&g_rm_littlefs_flash0_lfs,
                               &file,
                               path,
                               LFS_O_WRONLY | LFS_O_CREAT | LFS_O_APPEND,
                               &g_file_cfg);
    handle_lfs_error(lfs_err);
#else
    lfs_err = lfs_file_open(&g_rm_littlefs_flash0_lfs, &file, path, LFS_O_WRONLY |
LFS_O_CREAT | LFS_O_APPEND);
    handle_lfs_error(lfs_err);
#endif
 const char * contents = "butter";
    lfs_size_t   len      = strlen(contents);
 /* Apply butter to toast 10 times. */
 for (uint32_t i = 0; i < 10; i++)
    {
       lfs_err = lfs_file_write(&g_rm_littlefs_flash0_lfs, &file, contents, len);
 if (lfs_err < 0)
```

```
      {
      handle_lfs_error(lfs_err);
       }
    }
 /* Close the file. */
    lfs_err = lfs_file_close(&g_rm_littlefs_flash0_lfs, &file);
    handle_lfs_error(lfs_err);
 /* Unmount the filesystem. */
    lfs_err = lfs_unmount(&g_rm_littlefs_flash0_lfs);
    handle_lfs_error(lfs_err);
 /* Remount the filesystem. */
    lfs_err = lfs_mount(&g_rm_littlefs_flash0_lfs, &g_rm_littlefs_flash0_lfs_cfg);
    handle_lfs_error(lfs_err);
 /* Open breakfast/toast. */
#ifdef LFS_NO_MALLOC
    lfs_err = lfs_file_opencfg(&g_rm_littlefs_flash0_lfs, &file, path, LFS_O_RDONLY,
&g_file_cfg);
    handle_lfs_error(lfs_err);
#else
    lfs_err = lfs_file_open(&g_rm_littlefs_flash0_lfs, &file, path, LFS_O_RDONLY);
    handle_lfs_error(lfs_err);
#endif
    handle_lfs_error(lfs_err);
 /* Verify the toast is buttered the correct amount. */
 for (uint32_t i = 0; i < 10; i++)
    {
       lfs_err = lfs_file_read(&g_rm_littlefs_flash0_lfs, &file, buffer, len);
 if (lfs_err < 0)
      {
      handle_lfs_error(lfs_err);
       }
       assert(0 == memcmp(buffer, contents, len));
    }
 /* Close the file. */
```

```
    lfs_err = lfs_file_close(&g_rm_littlefs_flash0_lfs, &file);
    handle_lfs_error(lfs_err);
}
```

Function Documentation

◆ RM_LITTLEFS_FLASH_Open()

fsp_err_t RM_LITTLEFS_FLASH_Open (rm_littlefs_ctrl_t *const *p_ctrl*, rm_littlefs_cfg_t const *const *p_cfg*)

Opens the driver and initializes lower layer driver.

Implements rm_littlefs_api_t::open().

Return values

FSP_SUCCESS	Success.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_INVALID_SIZE	The provided block size is invalid.
FSP_ERR_INVALID_ARGUMENT	Flash BGO mode must be disabled.
FSP_ERR_INTERNAL	Failed to create the semaphore.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- flash_api_t::open

◆ RM_LITTLEFS_FLASH_Close()

fsp_err_t RM_LITTLEFS_FLASH_Close (rm_littlefs_ctrl_t *const *p_ctrl*)

Closes the lower level driver.

Implements rm_littlefs_api_t::close().

Return values

FSP_SUCCESS	Media device closed.
FSP_ERR_ASSERTION	An input parameter was invalid.
FSP_ERR_NOT_OPEN	Module not open.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- flash_api_t::close

4.2.96 MCUboot Port (rm_mcuboot_port)

Modules

MCUboot Port for RA MCUs.

Overview

Note

The MCUboot Port does not provide any interfaces to the user. Consult the MCUboot documentation at https://mcu-tools.github.io/mcuboot/ for further information.

Configuration

Build Time Configurations for MCUboot

The following build time configurations are defined in mcu-tools/include/mcuboot_config/mcuboot_config.h:

Configuration	Options	Default	Description
General > Custom mcuboot_config.h	Manual Entry		Add a path to your custom mcuboot_config.h file. It can be used to override some or all of the configurations defined here, and to define additional configurations.
General > Upgrade Mode	• Swap • Overwrite Only • Overwrite Only Fast	Overwrite Only	Swap supports A/B image swapping with rollback. Other modes with simpler code path, which only supports overwriting the existing image with the update image or running the newest image directly from its flash partition, are also available.
General > Validate Primary Image	• Enabled • Disabled	Enabled	Always check the signature of the image in the primary slot before booting, even if no upgrade was performed. This is recommended if the

			boot time penalty is acceptable.
General > Downgrade Prevention (Overwrite Only)	• Enabled • Disabled	Disabled	Prevent downgrades by enforcing incrementing version numbers. When this option is set, any upgrade must have greater major version or greater minor version with equal major version. This mechanism only protects against some attacks against version downgrades (for example, a JTAG could be used to write an older version).
General > Number of Images Per Application	• 1 • 2 (TrustZone)	1	Number of separately updateable images.
General > Watchdog Feed	Manual Entry		This function might be implemented if the OS / HW watchdog is enabled while doing a swap upgrade and the time it takes for a swapping is long enough to cause an unwanted reset. If implementing this, the OS main.c must also enable the watchdog (if required)!
General > Measured Boot	• Enabled • Disabled	Disabled	Copies the boot data into the secure RAM, intended to be used by the secure App.
General > Data Sharing	• Enabled • Disabled	Disabled	Copies the user data into the secure RAM, intended to be used by the secure App.
Signing Options > TrustZone > Boot Record (Image 2)	String length must be 12 characters or less.		Create CBOR encoded boot record TLV for Image 2. Represents the role of the software component (e.g. CoFM for coprocessor firmware). [max. 12 characters]
Signing Options > TrustZone > Custom	Manual Entry	--confirm	Add any custom options to pass to

(Image 2)			imgtool.py here. --pad places a trailer on the image that indicates that the image should be considered an upgrade. Writing this image in the secondary slot will then cause the bootloader to upgrade to it. --confirm marks the image as confirmed, which causes the upgrade to be permanent.
Signing Options > Signature Type	• None • ECDSA P-256 • RSA 2048 • RSA 3072	ECDSA P-256	Configure the signature type.
Signing Options > Boot Record	String length must be 12 characters or less.		Create CBOR encoded boot record TLV. Represents the role of the software component (e.g. CoFM for coprocessor firmware). [max. 12 characters]
Signing Options > Custom	Manual Entry	--confirm	Add any custom options to pass to imgtool.py here. --pad places a trailer on the image that indicates that the image should be considered an upgrade. Writing this image in the secondary slot will then cause the bootloader to upgrade to it. --confirm marks the image as confirmed, which causes the upgrade to be permanent.
Signing Options > Python	Manual Entry	python	Name of the python command to use. Default is python, but can be updated to python3 for Linux or an absolute path if needed.
Debugging > Log Level	• Off • Error • Warning	Off	Configure log level.

	• Info • Debug		
Flash Layout > TrustZone > Non-Secure Callable Region Size (Bytes)	Value must be an integer multiple of the 1024.	0x0	Size of the Non-Secure Callable region of the Secure image.
Flash Layout > TrustZone > Non-Secure Flash Area Size (Bytes) (TrustZone Non-Secure)	Value must be an integer multiple of the largest erase size on the mcu.	0x0	Size of the Non-Secure region. This must be non-zero for all TrustZone projects to ensure memory is partitioned correctly, even if the Secure and Non-Secure regions are treated as a single image. If the Non-Secure region can be updated separately, this size must account for the header and trailer.
Flash Layout > TrustZone > Non-Secure Callable RAM Region Size (Bytes)	Value must be an integer multiple of the 1024.	0x0	Size of the Non-Secure Callable RAM region of the Secure image.
Flash Layout > TrustZone > Non-Secure RAM Region Size (Bytes) (TrustZone Non-Secure)	Value must be an integer multiple of the 8192.	0x0	Size of the Non-Secure RAM region. This must be non-zero for all TrustZone projects to ensure memory is partitioned correctly, even if the Secure and Non-Secure regions are treated as a single image.
Flash Layout > TrustZone > Image 2 Header Size (Bytes)	Value must be an integer multiple of 0x80 (alignment required by VTOR).	0x80	Size of the flash reserved for the application image header for Image 2.
Flash Layout > Bootloader Flash Area Size (Bytes)	Value must be an integer multiple of the largest erase size on the mcu.	0x20000	Size of the flash reserved for the bootloader.
Flash Layout > Image 1 Header Size (Bytes)	Value must be an integer multiple of 0x80 (alignment required by VTOR).	0x80	Size of the flash reserved for the application image header.
Flash Layout > Image 1 Flash Area Size (Bytes)	Value must be an integer multiple of the largest erase size on	0x20000	Size of the application image 1, including the header and trailer. For

	the mcu.		TrustZone projects, enter the combined size of the Secure and Non-Secure Callable regions if the Non-Secure image can be updated separately, or enter the size of the entire image slot if Secure, Non-Secure Callable, and Non-Secure regions are updated as a single image.
Flash Layout > Scratch Flash Area Size (Bytes)	Value must be an integer multiple of the largest erase size on the mcu.	0x0	Size of the scratch area. Only required for swap update method.
Data Sharing > Maximum Measured Boot Record Size (Bytes)	Value must be an integer.	0x64	Maximum size of the boot record.
Data Sharing > Shared Data Size (Bytes)	Value must be an integer.	0x380	Size of the shared RAM area. Required for Measured Boot.
Data Sharing > Shared Data Address	Value must be an integer	0x20000000	Shared RAM start address. Required for Measured Boot.

Clock Configuration

This module does not use peripheral clocks.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Getting Started: Creating an MCUboot Project

Start by creating a new project in e2 studio or RA SC. If the MCU supports TrustZone, select a Flat project. The bootloader resides entirely in Secure memory, but it is considered a flat project because it does not provide any Non-Secure Callable functions. On the Stacks tab, add New > Bootloader > MCUboot. Resolve any constraint errors and edit configurations as desired. Add either the example keys or generate your own key. The MCUboot key generation tool is provided at ra/mcu-tools/MCUboot/scripts/imgtool.py and documented at https://github.com/mcu-tools/mcuboot/blob/master/docs/imgtool.md. Install the following required python packages to use imgtool.py: https://github.com/mcu-tools/mcuboot/blob/master/scripts/requirements.txt.

In src/hal_entry.c, drag in Developer Assistance > HAL/Common > MCUboot > Quick Setup > Call Quick Setup. Add a call to mcuboot_quick_setup() in the application and make any desired updates.

Note

MCUboot will contain either the verificaton public key or its hash. During production it is necessary to permanently lock the flash region where MCUboot is programmed to prevent the keys or the code from being modified.

Getting Started: Signing Tool Prerequisite

To use the MCUboot signing tool, ensure you have Python 3.x installed on your system. Then install the Python packages required for the signing tool with the following command:

pip3 install --user -r ra/mcu-tools/MCUboot/scripts/requirements.txt

Getting Started: Converting a Project to an MCUboot Image

MCUboot application images must execute from the image slot defined by the MCUboot project. They are also limited to a single downloadable flash region. All of this is handled by specifying a BootloaderDataFile in the FSP Configuration tool.

Any existing project can be converted to an MCUboot image.

1. If the project was created with a version prior to FSP v3.0.0, update the linker script to the v3.0.0 version before using it as an MCUboot application image.
2. Right click the project to convert in e2 studio or RA SC and select Properties.
3. Open C/C++ Build and select Build Variables.
4. Click Add...
5. For Variable Name, enter BootloaderDataFile. For Type, select File. Browse to the *.bld file created alongside the *.elf file for the associated MCUboot project.
6. Click OK, then Apply and Close.

To convert a TrustZone image, follow the steps above for both the Secure project and the Non-Secure project.

MCUboot application images must also be signed to work with MCUboot. At a minimum, this involves adding a SHA and MCUboot specific constant data called boot magic in the image trailer.

Signing can be done on the as a post-build step in e2 studio. To sign the image as a post-build step:

1. If Linux is used to develop the application image, change the MCUboot property Signing > Python to python3.
2. Build the bootloader project to generate the *.bld file. Make sure to build the bootloader project on the same computer as the application image to ensure the path to the signing script is correct.
3. Define environment variables in the Properties of the application image project in e2 studio.
    a. Right click the application image project, and select **Properties**.
    b. Select **C/C++ Build > Environment** on the left.
    c. Click **Add...**
    d. Define the following environment variables one at a time:
        - MCUBOOT_IMAGE_VERSION: Set to the version of the application image.
        - MCUBOOT_IMAGE_SIGNING_KEY: Set the path to the key used for signing. If signing is not required, do not set this variable. If example keys are used, set MCUBOOT_IMAGE_SIGNING_KEY as follows (replace <boot_project> with the bootloader project path):
            - ECC: <boot_project>/ra/mcu-tools/MCUboot/root-ec-p256.pem
            - RSA 2K: <boot_project>/ra/mcu-tools/MCUboot/root-rsa-2048.pem

        - RSA 3K: <boot_project>/ra/mcu-tools/MCUboot/root-rsa-3072.pem
    - MCUBOOT_APP_BIN_CONVERTER: Optional. Set to path to objcopy, arm-none-eabi-objcopy, fromelf, or ielftool. Not required if one of these tools is on the path.
4. Build the project.
5. The signed image is output next to the application <project>.elf file with the name <project>bin.signed.

Getting Started: Download and Debug

For projects that do not use TrustZone, debug the MCUboot project using the default configuration. Before running, load the signed image to the address specified in the signing comment in ra_cfg/mcu-tools/include/mcuboot_config/mcuboot_config.h. This can be done with the Load Ancillary File button when debugging in e2 studio. Upgrade images can be loaded to the upgrade image slots using the same method.

For TrustZone projects, debug using the Secure project to ensure the IDAU is partitioned correctly when debugging in e2 studio. Make the following modifications before debugging in e2 studio:

1. In the Debug Configurations for your project, on the Startup tab, click Add... to add the MCUboot project *.elf file (Image and Symbols), and optionally the Non-Secure project *.elf file.
2. For the Secure and Non-Secure project *.elf file, load Symbols Only.
3. After starting to debug, load the signed Secure image and the signed Non-Secure image into the addresses specified in the signing comment in ra_cfg/mcu-tools/include/mcuboot_config/mcuboot_config.h. This can be done with the Load Ancillary File button when debugging in e2 studio. Upgrade images can be loaded to the upgrade image slots using the same method.

Confirming Upgrade in Swap Mode

In Swap Mode operation, if the upgrade image is signed with the –pad option, MCUboot will install that image as a temporary update where if nothing else is done, a reboot will cause MCUboot to revert to the image version that was swapped out during the upgrade. In order for the updated image to prevent this reversion and make the update permanent, the boot_set_confirmed() must be called from the application.

To avail this capability in the application image, from the Stacks tab, add New > Bootloader > MCUboot Image Utilities (Swap Mode). Resolve any constraint errors and edit configurations as desired.

In src/hal_entry.c, drag in Developer Assistance > HAL/Common > MCUboot Image Utilities > Quick Setup > Confirm Primary Image. Add a call to boot_set_confirmed() in the application and confirm the image in the primary slot.

External Memory Support

QSPI support for secondary image storage can be enabled in the configurator. The booloader expectes the QSPI memory to be pre-configured by user code in the bootloader for read/write operation. The bootloader code operates under the assumption that the user has configured the QSPI in Extended-SPI mode and that R_QSPI_Open() has been called prior to invoking boot_go();. For an example on how to initialize the QSPI device, refer to the QSPI module. The QSPI sector size must be the same as that of the MCU internal flash (BSP_FEATURE_FLASH_HP_CF_REGION1_BLOCK_SIZE) for swap mode operation.

MCUboot Memory Map

For single image projects with no external memory support, the default memory map looks like:

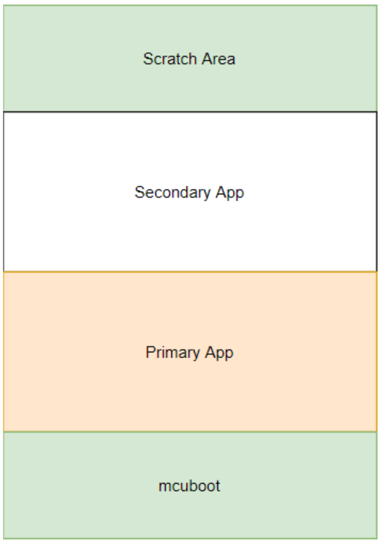


Figure 205: MCUboot Memory Map

For projects with 2 separately updateable images (used for TrustZone applications where the Secure and Non-Secure images can be updated separately), the default memory map with no external memory support looks like:

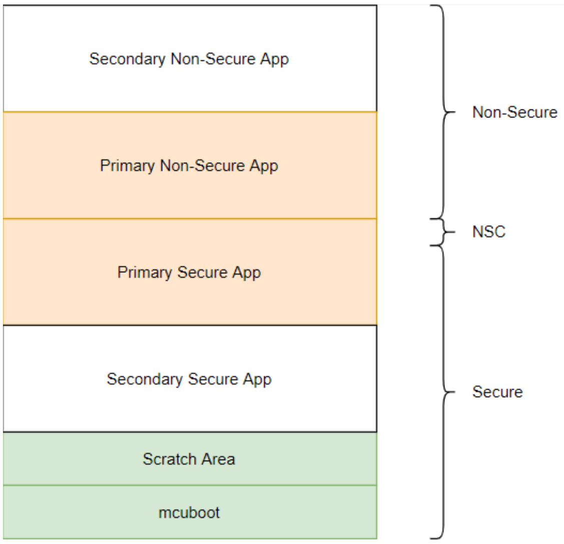


Figure 206: MCUboot Memory Map (TrustZone)

For single image projects with QSPI, the default memory map looks like:

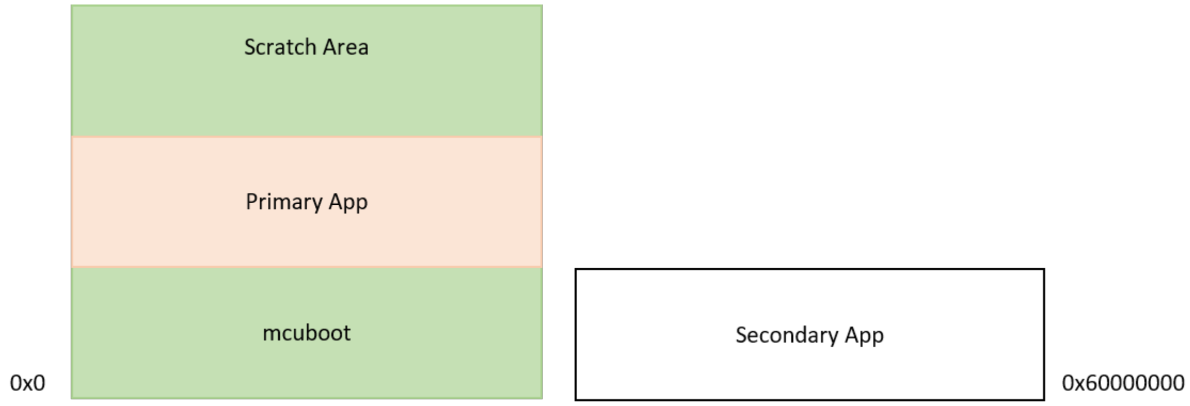


Figure 207: MCUboot Memory Map with QSPI

For projects with 2 separately updateable images (used for TrustZone applications where the Secure and Non-Secure images can be updated separately), the default memory map with QSPI support looks like:

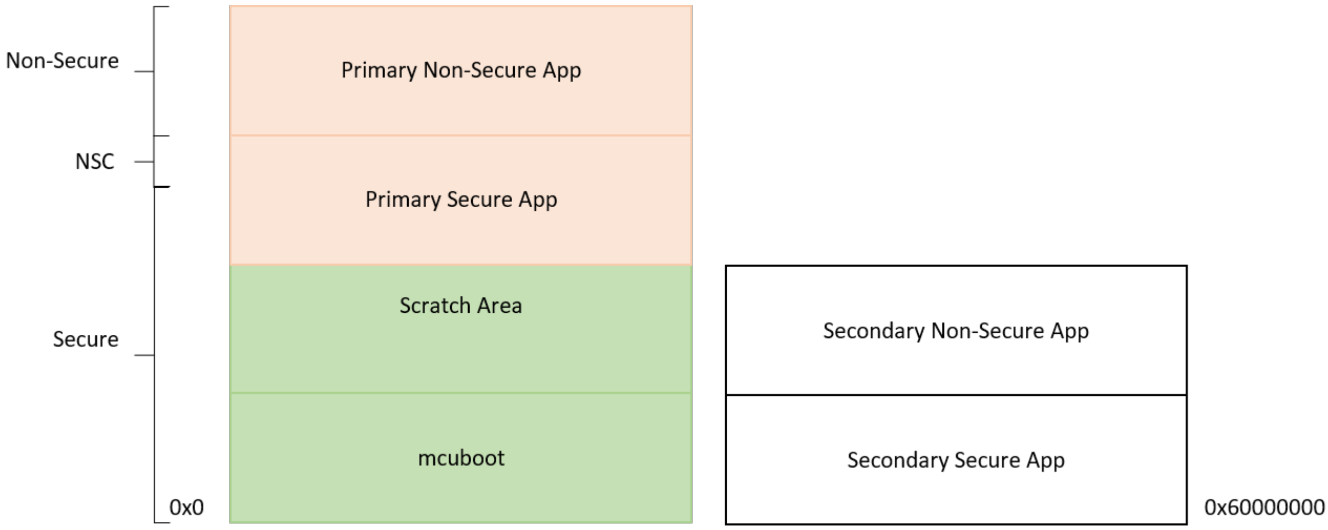


Figure 208: MCUboot Memory Map (TrustZone) with QSPI

MCUboot Crypto Stack Options

The following crypto stacks can be used with MCUboot in FSP:

1. MbedTLS, which is hardware accelerated on all RA devices. On the RA2 which has an AES engine only, ECC/RSA/SHA operations are in software.
2. TinyCrypt (S/W Only) can be used with all devices.
3. TinyCrypt (H/W Accelerated) has AES operations accelerated for the RA2 family only. When using MCUboot without encryption there is no difference between using this or the S/W only version.
4. SCE9 Protected Mode on devices that have the SCE9 (eg: RA6M4, RA4M3, RA4M2)

MbedTLS provides the best performance for MCUBoot signature verification on the RA6 and RA4 devices but has a much larger code footprint compared to TinyCrypt. For RA2 devices TinyCrypt is the best option.

Using SCE9 Protected Mode Crypto Stack

Using this crypto stack with MCUBoot provides additional security by ensuring that any keys that are used were securely provisioned for the specific device. The Application Note R11AN0496 provides detailed steps on how to go about installing these keys. Since the section "Preparing Keys for Installation and Update Using RFP" document currently only provides information on how to install an AES key, this section will provide information on how to install an ECC public key.

Note

> ***When using the SCE9 Protected Mode Stack with MCUboot it is required that the public keys in the format described in the "MCUboot Example Keys" module in the stack is also provided in the project.***

1. Generate an ECC key pair. There are various ways to do this but you can use openSSL to do so: "openssl ecparam -name secp256k1 -genkey -noout -out my_ecc_secp256k1_key.pem".
2. Once the key is generated, in order to install the public key using RFP (Renesas Flash Programmer) the user needs to have their own UFPK (User Factory Programming Key) and W-UFPK (Wrapped User Factory Programming Key). Refer to R11AN0496 on how to obtain these keys.
3. Once the UFPK and W-UFPK are available, we need to extract the public key from the pem file. The public key can be viewed by using "openssl ec -noout -text -in my_ecc_secp256k1_key.pem". Note that when the ECC public key is printed out this way, it will contain a 0x04 ASN.1 demarcator at the start which should be discarded.
4. Use the rfp-util.exe utility from the RFP installation folder to wrap the public key using the UFPK and W-UFPK into a format that can be installed by RFP and the factory bootloader on the MCU.
5. Use RFP as described in R11AN0496 to install the key to the location of mcuboot_sce9_key section.

These are examples that install the default keys provided with MCUboot in ra/mcu-tools/MCUboot/. The examples assume that UFPK nd W-UFPK are already available.

```
  //Print out the EC-P256 Public Key using openSSL
C:\ openssl ec -noout -text -in root-ec-p256.pem
read EC key
Private-Key: (256 bit)
priv:
    d7:98:d5:2f:83:01:24:3b:d3:54:2b:7e:55:ed:4c:
    74:61:19:00:b0:f9:50:5a:82:4f:e1:e8:ec:06:3b:
    cf:f1
pub:
    04:2a:cb:40:3c:e8:fe:ed:5b:a4:49:95:a1:a9:1d:
    ae:e8:db:be:19:37:cd:14:fb:2f:24:57:37:e5:95:
    39:88:d9:94:b9:d6:5a:eb:d7:cd:d5:30:8a:d6:fe:
    48:b2:4a:6a:81:0e:e5:f0:7d:8b:68:34:cc:3a:6a:
    fc:53:8e:fa:c1
ASN1 OID: prime256v1
```

```
NIST CURVE: P-256
//Use the public key (ignore the 0x04 ASN.1 demarcator) in the RFP command line to
convert the public key into an installable format
C:\ "C:\Program Files (x86)\Renesas Electronics\Programming Tools\Renesas Flash
Programmer V3.08\rfp-util.exe" /genkey /ufpk "C:\ufpk.key" /wufpk
"C:\ufpk.key_enc.key" /key "2acb403ce8feed5ba44995a1a91daee8dbbe1937cd14fb2f245737e59
53988d994b9d65aebd7cdd5308ad6fe48b24a6a810ee5f07d8b6834cc3a6afc538efac1" /userkey
"16" /output "C:\ECC_pub_install.rkey"
// From the bootloader map file determine the address of mcuboot_sce9_key section
Use RFP to install "ECC_pub_install.rkey" as described in R11AN0496 to the address
where the mcuboot_sce9_key section is located.
```

Limitations

MCUboot tooling updates to adjust available flash in application images are currently only supported for the GCC compiler. IAR and AC6 support will be added in a future release.

Examples

Basic Example

This is an example of using MCUboot in an application.

```
void rm_mcuboot_port_example (void)
{
#ifdef MCUBOOT_USE_MBED_TLS
 /* Initialize mbedtls. */
 mbedtls_platform_context ctx = {0};
    assert(0 == mbedtls_platform_setup(&ctx));
#elif defined(MCUBOOT_USE_TINYCRYPT)
 /* Initialize TinyCrypt port. */
    assert(FSP_SUCCESS == RM_TINCYRYPT_PORT_Init());
#else
 /* Initialize SCE9 Protected Mode driver. */
 sce_instance_ctrl_t sce_ctrl;
 const sce_cfg_t    sce_cfg =
    {.lifecycle = SCE_SSD};
```

```
    assert(FSP_SUCCESS == R_SCE_Open(&sce_ctrl, &sce_cfg));
#endif
 /* (Optional) To check for updates, call boot_set_pending. */
 bool update = 0;
 if (update)
    {
       boot_set_pending(0);
    }
 /* Verify the boot image and get its location. */
 struct boot_rsp rsp;
    assert(0 == boot_go(&rsp));
 /* Enter the application. */
    RM_MCUBOOT_PORT_BootApp(&rsp);
}
```

4.2.97 Motor 120 Control Hall (motor_120_control_hall)

Modules

Functions

fsp_err_t	RM_MOTOR_120_CONTROL_HALL_Open (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg) Opens and configures the motor hall 120 detection module. Implements motor_120_control_api_t::open. More...
fsp_err_t	RM_MOTOR_120_CONTROL_HALL_Close (motor_120_control_ctrl_t *const p_ctrl) Disables specified motor hall 120 detection module. Implements motor_120_control_api_t::close. More...
fsp_err_t	RM_MOTOR_120_CONTROL_HALL_Run (motor_120_control_ctrl_t *const p_ctrl) Run motor (Start motor rotation). Implements motor_120_control_api_t::run. More...
fsp_err_t	RM_MOTOR_120_CONTROL_HALL_Stop (motor_120_control_ctrl_t *const p_ctrl)

	Stop motor (Stop motor rotation). Implements motor_120_control_api_t::stop. More...
fsp_err_t	RM_MOTOR_120_CONTROL_HALL_Reset (motor_120_control_ctrl_t *const p_ctrl) Reset variables of motor hall 120 detection module. Implements motor_120_control_api_t::reset. More...
fsp_err_t	RM_MOTOR_120_CONTROL_HALL_SpeedSet (motor_120_control_ctrl_t *const p_ctrl, float const speed_rpm) Set speed[rpm]. Implements motor_120_control_api_t::speedSet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_HALL_SpeedGet (motor_120_control_ctrl_t *const p_ctrl, float *const p_speed_rpm) Get speed. Implements motor_120_control_api_t::speedGet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_HALL_CurrentGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status) Get current. Implements motor_120_control_api_t::currentGet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_HALL_WaitStopFlagGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_wait_stop_flag_t *const p_flag) Get wait stop flag. Implements motor_120_control_api_t::waitStopFlagGet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_HALL_TimeoutErrorFlagGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_timeout_error_flag_t *const p_timeout_error_flag) Get timeout error flag. Implements motor_120_control_api_t::timeoutErrorFlagGet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_HALL_PatternErrorFlagGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_pattern_error_flag_t *const p_pattern_error_flag) Get pattern error flag. Implements motor_120_control_api_t::patternErrorFlagGet. More...

fsp_err_t	RM_MOTOR_120_CONTROL_HALL_VoltageRefGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_voltage_ref_t *const p_voltage_ref) Get voltage ref. Implements motor_120_control_api_t::voltageRefGet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_HALL_ParameterUpdate (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg) Update the parameters of hall 120 detection module. Implements motor_120_control_api_t::parameterUpdate. More...

Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Interface.

Overview

The motor current is used to control the electric current of motor rotation in an appication. This module should be called cyclically after the A/D conversion of electric current of each phase in an application. This module calculates each phase voltage with input current reference, electric current and rotor angle.

Features

The motor 120 control hall module has below features.

- Calculate each phase(U/V/W) voltage.
- Decoupling control.
- Voltage error compensation.

Configuration

Build Time Configurations for rm_motor_120_control_hall

The following build time configurations are defined in fsp_cfg/rm_motor_120_control_hall_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > 120-degree conduction control with Hall sensors (rm_motor_120_control_hall)

This module can be added to the Stacks tab via New Stack > Motor > 120-degree conduction control with Hall sensors (rm_motor_120_control_hall).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_120_control_hall0	Module name.
General > Conduction type	• First 60 degree PWM • Complementary First 60 degree PWM	First 60 degree PWM	Select conduction type
General > Timeout counts (msec)	Must be a valid non-negative value.	200	Counts to judge rotor unrotate
General > Maximum voltage (V)	Must be a valid non-negative value.	20.0	Maximum output voltage (V)
General > Minimum voltage (V)	Must be a valid non-negative value.	3.0	Minimum output voltage (V)
General > Speed PI decimation	Must be a valid non-negative value.	0	Speed PI control decimation count
General > Freerun timer frequency (MHz)	Must be a valid non-negative value.	120	Freerun timer frequency (MHz)
General > Speed LPF	Must be a valid non-negative value.	1.0	Speed LPF parameter
General > Step of speed reference change	Must be a valid non-negative value.	0.2	Speed ref change step
General > Start reference voltage (V)	Must be a valid non-negative value.	5.8	Reference voltage for boot mode (V)
General > Hall wait counts	Must be a valid non-negative value.	12	Wait counts of hall interrupts to start speed calculation
General > Stop judge time	Must be a valid non-negative value.	1000	Stop judge time
General > Minimum limit speed (rpm)	Must be a valid non-negative value.	550	Minimum limit speed (rpm) (mechanical angle)
General > PI control KP	Must be a valid non-negative value.	0.02	PI control gain of proportional term
General > PI control KI	Must be a valid non-negative value.	0.0005	PI control gain of integral term
General > PI control limit	Must be a valid non-negative value.	24.0	PI control limit of integral term
General > Hall interrupt mask value	Must be a valid non-negative value.	15	For limiting hall interrupt processing. Limited by the number of ADC interrupts

Motor Parameter > Pole pairs	Must be a valid non-negative value.	2	Pole pairs
Motor Parameter > Resistance (ohm)	Must be a valid non-negative value.	6.447	Resistance
Motor Parameter > Inductance of d-axis (H)	Must be a valid non-negative value.	0.0045	Inductance of d-axis
Motor Parameter > Inductance of q-axis (H)	Must be a valid non-negative value.	0.0045	Inductance of q-axis
Motor Parameter > Permanent magnetic flux (Wb)	Must be a valid non-negative value.	0.02159	Permanent magnetic flux
Motor Parameter > Rotor inertia (kgm^2)	Must be a valid non-negative value.	1.8	Rotor inertia
Interrupts > Callback	Name must be a valid C symbol	NULL	Callback function
Hall sensor port U	Manual Entry	BSP_IO_PORT_04_PIN_11	Hall sensor port U
Hall sensor port V	Manual Entry	BSP_IO_PORT_04_PIN_10	Hall sensor port V
Hall sensor port W	Manual Entry	BSP_IO_PORT_04_PIN_09	Hall sensor port W

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

Pin Configuration

Depend on included ICU module.

Usage Notes

Limitations

- Set the period of current control with none-negative value.
- Set the reference voltage with none-negative value.

Examples

Basic Example

This is a basic example of minimal use of the motor 120 control hall in an application.

```
void motor_120_control_hall_basic_example (void)
{
```

```
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_120_CONTROL_HALL_Open(g_motor_120_control_hall0.p_ctrl,
g_motor_120_control_hall0.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Set speed reference before motor run */
    (void) RM_MOTOR_120_CONTROL_HALL_SpeedSet(g_motor_120_control_hall0.p_ctrl,
DEF_120_CONTROL_HALL_TEST_SPEED_REF);
 /* Start motor rotation */
    (void) RM_MOTOR_120_CONTROL_HALL_Run(g_motor_120_control_hall0.p_ctrl);
 /* Get current motor speed */
    (void) RM_MOTOR_120_CONTROL_HALL_SpeedGet(g_motor_120_control_hall0.p_ctrl,
&smpl_speed);
 /* Get current */
    (void) RM_MOTOR_120_CONTROL_HALL_CurrentGet(g_motor_120_control_hall0.p_ctrl,
&smpl_current_status);
 /* Get wait stop flag */
    (void)
RM_MOTOR_120_CONTROL_HALL_WaitStopFlagGet(g_motor_120_control_hall0.p_ctrl,
&smpl_wait_stop_flag);
 /* Get timeout error flag */
    (void)
RM_MOTOR_120_CONTROL_HALL_TimeoutErrorFlagGet(g_motor_120_control_hall0.p_ctrl,
&smpl_timeout_error_flag);
 /* Get pattern error flag */
    (void)
RM_MOTOR_120_CONTROL_HALL_PatternErrorFlagGet(g_motor_120_control_hall0.p_ctrl,
&smpl_pattern_error_flag);
 /* Get voltage ref */
    (void) RM_MOTOR_120_CONTROL_HALL_VoltageRefGet(g_motor_120_control_hall0.p_ctrl,
&smpl_voltage_ref);
    (void)
RM_MOTOR_120_CONTROL_HALL_ParameterUpdate(g_motor_120_control_hall0.p_ctrl,
g_motor_120_control_hall0.p_cfg);
```

```
/* Stop motor rotation */
   (void) RM_MOTOR_120_CONTROL_HALL_Stop(g_motor_120_control_hall0.p_ctrl);
/* Reset the process. */
   (void) RM_MOTOR_120_CONTROL_HALL_Reset(g_motor_120_control_hall0.p_ctrl);
/* Close the module. */
   (void) RM_MOTOR_120_CONTROL_HALL_Close(g_motor_120_control_hall0.p_ctrl);
}
```

Data Structures

struct	motor_120_control_hall_extended_cfg_t
struct	motor_120_control_hall_instance_ctrl_t

Data Structure Documentation

◆ motor_120_control_hall_extended_cfg_t

struct motor_120_control_hall_extended_cfg_t

Extended configurations for motor 120 control hall

Data Fields		
bsp_io_port_pin_t	port_hall_sensor_u	Hall sensor port U.
bsp_io_port_pin_t	port_hall_sensor_v	Hall sensor port V.
bsp_io_port_pin_t	port_hall_sensor_w	Hall sensor port W.
float	f4_start_refv	Reference voltage for boot mode.
uint32_t	u4_hall_wait_cnt	Wait counts of hall interrupts for speed calculation.
uint32_t	u4_stop_judge_time	Stop judge time.
uint32_t	u4_min_speed_rpm	Minimum limit speed (rpm) (mechanical angle)
uint32_t	u4_hall_interrupt_mask_value	For limiting hall interrupt processing. Limited by the number of ADC interrupts.
motor_120_driver_instance_t const *	p_motor_120_driver_instance	Motor 120 driver access module.
timer_instance_t const *	p_speed_cyclic_timer_instance	Cyclic process of speed control timer module.
timer_instance_t const *	p_speed_calc_timer_instance	Speed calculate timer module.
external_irq_instance_t const *	p_u_hall_irq_instance	U phase hall interrupt.
external_irq_instance_t const *	p_v_hall_irq_instance	V phase hall interrupt.

external_irq_instance_t const *	p_w_hall_irq_instance	W phase hall interrupt.

◆ motor_120_control_hall_instance_ctrl_t

struct motor_120_control_hall_instance_ctrl_t		
120 control hall instance control block		
Data Fields		
uint32_t	open	Used to determine if the channel is configured.
motor_120_control_status_t	active	Flag to set active/inactive the motor 120 control.
motor_120_control_run_mode_t	run_mode	Drive mode.
motor_120_control_timeout_error_flag_t	timeout_error_flag	Timeout error status.
motor_120_control_pattern_error_flag_t	pattern_error_flag	Hall pattern error status.
motor_120_control_rotation_direction_t	direction	Rotational direction (0:CW ,1:CCW)
float	f4_speed_calc_base	Base counts to calculate rotation speed.
float	f_rpm2rad	Translate value to radian/second to rpm.
float	f4_v_ref	Voltage reference (output of speed PI control)
float	f4_ref_speed_rad	Motor speed reference.
float	f4_ref_speed_rad_ctrl	Motor speed reference for speed PI control.
float	f4_speed_rad	Motor speed.
uint32_t	u4_cnt_speed_pi	Counter for period of speed PI control.
motor_120_control_wait_stop_flag_t	flag_wait_stop	Flag for waiting for motor stop.
uint32_t	u4_cnt_wait_stop	Counter for waiting motor stop.
motor_120_driver_phase_pattern_t	v_pattern	Voltage pattern.
motor_120_control_speed_ref_t	flag_speed_ref	Speed reference flag.
motor_120_control_voltage_ref_t	flag_voltage_ref	Voltage reference flag.
uint32_t	u4_cnt_timeout	Counter for timeout error.
uint32_t	u4_hall_timer_cnt	Value of timer counter.

uint32_t	u4_pre_hall_timer_cnt	Previous value of timer counter.
int32_t	s4_timer_cnt_ave	Counts for 360 degrees.
uint32_t	u4_timer_cnt_buf[MOTOR_120_CONTROL_HALL_TIMES]	Counts for 60 degrees.
uint32_t	u4_timer_cnt_num	Array element number before 360 degrees.
float	f4_pi_ctrl_err	PI control error.
float	f4_pi_ctrl_refi	PI control buffer of integral term.
uint32_t	u4_hall_intr_cnt	For start timing of speed calculation.
uint32_t	u4_adc_interrupt_cnt	Number of ADC interrupt processing.
motor_120_control_cfg_t const *	p_cfg	Pointer of configuration structure.
external_irq_callback_args_t	hall_interrupt_args	For call IRQ callbackSet function.
timer_callback_args_t	timer_args	For call timer callbackSet function.

Function Documentation

◆ RM_MOTOR_120_CONTROL_HALL_Open()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_Open (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_cfg_t const *const *p_cfg*)

Opens and configures the motor hall 120 detection module. Implements motor_120_control_api_t::open.

Example:

```
/* Initializes the module. */
    err = RM_MOTOR_120_CONTROL_HALL_Open(g_motor_120_control_hall0.p_ctrl,
g_motor_120_control_hall0.p_cfg);
```

Return values

FSP_SUCCESS	Motor 120 driver successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_HALL_Close()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_Close (motor_120_control_ctrl_t *const *p_ctrl*)

Disables specified motor hall 120 detection module. Implements motor_120_control_api_t::close.

Example:

```
/* Close the module. */
   (void) RM_MOTOR_120_CONTROL_HALL_Close(g_motor_120_control_hall0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_CONTROL_HALL_Run()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_Run (motor_120_control_ctrl_t *const *p_ctrl*)

Run motor (Start motor rotation). Implements motor_120_control_api_t::run.

Example:

```
/* Start motor rotation */
   (void) RM_MOTOR_120_CONTROL_HALL_Run(g_motor_120_control_hall0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_CONTROL_HALL_Stop()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_Stop (motor_120_control_ctrl_t *const *p_ctrl*)

Stop motor (Stop motor rotation). Implements motor_120_control_api_t::stop.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_120_CONTROL_HALL_Stop(g_motor_120_control_hall0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_CONTROL_HALL_Reset()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_Reset (motor_120_control_ctrl_t *const *p_ctrl*)

Reset variables of motor hall 120 detection module. Implements motor_120_control_api_t::reset.

Example:

```
/* Reset the process. */
   (void) RM_MOTOR_120_CONTROL_HALL_Reset(g_motor_120_control_hall0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_CONTROL_HALL_SpeedSet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_SpeedSet (motor_120_control_ctrl_t *const *p_ctrl*, float const *speed_rpm*)

Set speed[rpm]. Implements motor_120_control_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
    (void) RM_MOTOR_120_CONTROL_HALL_SpeedSet(g_motor_120_control_hall0.p_ctrl,
DEF_120_CONTROL_HALL_TEST_SPEED_REF);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_CONTROL_HALL_SpeedGet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_SpeedGet (motor_120_control_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm*)

Get speed. Implements motor_120_control_api_t::speedGet.

Example:

```
/* Get current motor speed */
    (void) RM_MOTOR_120_CONTROL_HALL_SpeedGet(g_motor_120_control_hall0.p_ctrl,
&smpl_speed);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_HALL_CurrentGet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_CurrentGet (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_driver_current_status_t *const *p_current_status*)

Get current. Implements motor_120_control_api_t::currentGet.

Example:

```
/* Get current */
    (void) RM_MOTOR_120_CONTROL_HALL_CurrentGet(g_motor_120_control_hall0.p_ctrl,
&smpl_current_status);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_HALL_WaitStopFlagGet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_WaitStopFlagGet (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_wait_stop_flag_t *const *p_flag*)

Get wait stop flag. Implements motor_120_control_api_t::waitStopFlagGet.

Example:

```
/* Get wait stop flag */
    (void)
RM_MOTOR_120_CONTROL_HALL_WaitStopFlagGet(g_motor_120_control_hall0.p_ctrl,
&smpl_wait_stop_flag);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_HALL_TimeoutErrorFlagGet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_TimeoutErrorFlagGet (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_timeout_error_flag_t *const *p_timeout_error_flag*)

Get timeout error flag. Implements motor_120_control_api_t::timeoutErrorFlagGet.

Example:

```
/* Get timeout error flag */
   (void)
RM_MOTOR_120_CONTROL_HALL_TimeoutErrorFlagGet(g_motor_120_control_hall0.p_ctrl,
&smpl_timeout_error_flag);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_HALL_PatternErrorFlagGet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_PatternErrorFlagGet (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_pattern_error_flag_t *const *p_pattern_error_flag*)

Get pattern error flag. Implements motor_120_control_api_t::patternErrorFlagGet.

Example:

```
/* Get pattern error flag */
   (void)
RM_MOTOR_120_CONTROL_HALL_PatternErrorFlagGet(g_motor_120_control_hall0.p_ctrl,
&smpl_pattern_error_flag);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_HALL_VoltageRefGet()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_VoltageRefGet (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_voltage_ref_t *const *p_voltage_ref*)

Get voltage ref. Implements motor_120_control_api_t::voltageRefGet.

Example:

```
/* Get voltage ref */
    (void) RM_MOTOR_120_CONTROL_HALL_VoltageRefGet(g_motor_120_control_hall0.p_ctrl,
&smpl_voltage_ref);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_HALL_ParameterUpdate()

fsp_err_t RM_MOTOR_120_CONTROL_HALL_ParameterUpdate (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_cfg_t const *const *p_cfg*)

Update the parameters of hall 120 detection module. Implements motor_120_control_api_t::parameterUpdate.

Example:

```
    (void)
RM_MOTOR_120_CONTROL_HALL_ParameterUpdate(g_motor_120_control_hall0.p_ctrl,
g_motor_120_control_hall0.p_cfg);
```

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

4.2.98 Motor 120 Control Sensorless (motor_120_control_sensorless)

Modules

Functions

fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_Open (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg) Opens and configures the motor sensorless 120 detection module. Implements motor_120_control_api_t::open. More...
fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_Close (motor_120_control_ctrl_t *const p_ctrl) Disables specified motor sensorless 120 detection module. Implements motor_120_control_api_t::close. More...
fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_Run (motor_120_control_ctrl_t *const p_ctrl) Run motor (Start motor rotation). Implements motor_120_control_api_t::run. More...
fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_Stop (motor_120_control_ctrl_t *const p_ctrl) Stop motor (Stop motor rotation). Implements motor_120_control_api_t::stop. More...
fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_Reset (motor_120_control_ctrl_t *const p_ctrl) Reset variables of motor sensorless 120 detection module. Implements motor_120_control_api_t::reset. More...
fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_SpeedSet (motor_120_control_ctrl_t *const p_ctrl, float const speed_rpm) Set speed[rpm]. Implements motor_120_control_api_t::speedSet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_SpeedGet (motor_120_control_ctrl_t *const p_ctrl, float *const p_speed_rpm) Get speed. Implements motor_120_control_api_t::speedGet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_CurrentGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status)

	Get current. Implements motor_120_control_api_t::currentGet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_WaitStopFlagGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_wait_stop_flag_t *const p_flag) Get wait stop flag. Implements motor_120_control_api_t::waitStopFlagGet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_TimeoutErrorFlagGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_timeout_error_flag_t *const p_timeout_error_flag) Get timeout error flag. Implements motor_120_control_api_t::timeoutErrorFlagGet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_PatternErrorFlagGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_pattern_error_flag_t *const p_pattern_error_flag) Get pattern error flag. Implements motor_120_control_api_t::patternErrorFlagGet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_VoltageRefGet (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_voltage_ref_t *const p_voltage_ref) Get voltage ref. Implements motor_120_control_api_t::voltageRefGet. More...
fsp_err_t	RM_MOTOR_120_CONTROL_SENSORLESS_ParameterUpdate (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg) Update the parameters of sensorless 120 detection module. Implements motor_120_control_api_t::parameterUpdate. More...

Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Interface.

Overview

The motor current is used to control the electric current of motor rotation in an appication. This module should be called cyclically after the A/D conversion of electric current of each phase in an application. This module calculates each phase voltage with input current reference, electric current and rotor angle.

Features

The motor 120 control sensorless module has below features.

- Calculate each phase(U/V/W) voltage.
- Decoupling control.
- Voltage error compensation.

Configuration

Build Time Configurations for rm_motor_120_control_sensorless

The following build time configurations are defined in fsp_cfg/rm_motor_120_control_sensorless_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > 120-degree conduction control sensorless (rm_motor_120_control_sensorless)

This module can be added to the Stacks tab via New Stack > Motor > 120-degree conduction control sensorless (rm_motor_120_control_sensorless).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_120_control_sensorless0	Module name.
General > Conduction type	• First 60 degree PWM • Complementary First 60 degree PWM	First 60 degree PWM	Select conduction type
General > Stop BEMF	Must be a valid non-negative value.	0.5	Value of stop motor BEMF (U+V+W)
General > Timeout counts (msec)	Must be a valid non-negative value.	2000	Undetected time
General > Maximum voltage for BOOT (V)	Must be a valid non-negative value.	8.0	Maximum output voltage for boot mode (V)
General > Maximum voltage (V)	Must be a valid non-negative value.	20.0	Maximum output voltage (V)
General > Minimum voltage (V)	Must be a valid non-negative value.	3.0	Minimum output voltage (V)
General > Carrier	Must be a valid non-	20.0	PWM carrier frequency

frequency (kHz)	negative value.		(kHz)
General > Adjusting angle	Manual Entry	0	Adjusting angle
General > Speed PI decimation	Must be a valid non-negative value.	1	Speed PI control decimation count
General > Free run timer frequency (MHz)	Must be a valid non-negative value.	120	Freerun timer frequency (MHz)
General > Speed LPF K	Must be a valid non-negative value.	1.0	Speed LPF parameter
General > Step of speed change	Must be a valid non-negative value.	0.2	Speed reference change step
General > Boot reference voltage (V)	Must be a valid non-negative value.	3.0	Voltage reference for boot mode
General > Voltage lamping time	Must be a valid non-negative value.	128	Voltage lamping time
General > Voltage constant adjust time	Must be a valid non-negative value.	64	Voltage constant adjust time value (msec)
General > Open loop start speed (rpm)	Manual Entry	150	Open loop start speed (rpm)
General > Open loop mode2 speed (rpm)	Manual Entry	185	to mode2 change speed (rpm)
General > Open loop mode3 speed (rpm)	Manual Entry	1000	to mode3 change speed (rpm)
General > Open loop start voltage (V)	Must be a valid non-negative value.	3.0	start reference voltage (V)
General > Open loop mode1 speed rate	Must be a valid non-negative value.	0.25	increase rate of reference speed (rpm/control period)
General > Open loop mode2 voltage rate	Must be a valid non-negative value.	0.00285	increase rate of reference voltage (v/control period)
General > Open loop mode2 speed rate	Must be a valid non-negative value.	0.71	increase rate of reference speed (rpm/control period)
General > Open loop mode3 voltage rate	Must be a valid non-negative value.	0.002	increase rate of reference voltage (v/control period)
General > Open loop maximum voltage (V)	Must be a valid non-negative value.	6.5	openloop maximum voltage (V)
General > PI control KP	Must be a valid non-negative value.	0.02	PI control gain of proportional term
General > PI control KI	Must be a valid non-negative value.	0.004	PI control gain of integral term

General > PI control limit	Must be a valid non-negative value.	24.0	PI control limit of integral term
Motor Parameter > Pole pairs	Must be a valid non-negative value.	2	Pole pairs
Motor Parameter > Resistance (ohm)	Must be a valid non-negative value.	6.447	Resistance
Motor Parameter > Inductance of d-axis (H)	Must be a valid non-negative value.	0.0045	Inductance of d-axis
Motor Parameter > Inductance of q-axis (H)	Must be a valid non-negative value.	0.0045	Inductance of q-axis
Motor Parameter > Permanent magnetic flux (Wb)	Must be a valid non-negative value.	0.02159	Permanent magnetic flux
Motor Parameter > Rotor inertia (kgm^2)	Must be a valid non-negative value.	1.8	Rotor inertia
Interrupts > Callback	Name must be a valid C symbol	NULL	callback function

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

- Set the period of current control with none-negative value.
- Set the reference voltage with none-negative value.

Examples

Basic Example

This is a basic example of minimal use of the motor 120 control sensorless in an application.

```
void motor_120_control_sensorless_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err =
```

```
RM_MOTOR_120_CONTROL_SENSORLESS_Open(g_motor_120_control_sensorless0.p_ctrl,
                                              g_motor_120_control_sensorless0.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Set speed reference before motor run */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_SpeedSet(g_motor_120_control_sensorless0.p_ctrl,

DEF_120_CONTROL_SENSORLESSHALL_TEST_OVSPD_LIM);
 /* Start motor rotation */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Run(g_motor_120_control_sensorless0.p_ctrl);
 /* Get current motor speed */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_SpeedGet(g_motor_120_control_sensorless0.p_ctrl,
&smpl_speed);
 /* Get current */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_CurrentGet(g_motor_120_control_sensorless0.p_ctrl,
&smpl_current_status);
 /* Get wait stop flag */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_WaitStopFlagGet
(g_motor_120_control_sensorless0.p_ctrl,
                                                          &smpl_wait_stop_flag);
 /* Get timeout error flag */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_TimeoutErrorFlagGet
(g_motor_120_control_sensorless0.p_ctrl,
                                                               &smpl_timeout_error_fl
ag);
 /* Get pattern error flag */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_PatternErrorFlagGet
(g_motor_120_control_sensorless0.p_ctrl,
```

```
                                                                &smpl_pattern_error_fl
ag);
 /* Get voltage ref */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_VoltageRefGet(g_motor_120_control_sensorless0.p_ctrl,
&smpl_voltage_ref);
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_ParameterUpdate
(g_motor_120_control_sensorless0.p_ctrl,

g_motor_120_control_sensorless0.p_cfg);
 /* Stop motor rotation */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Stop(g_motor_120_control_sensorless0.p_ctrl);
 /* Reset the process. */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Reset(g_motor_120_control_sensorless0.p_ctrl);
 /* Close the module. */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Close(g_motor_120_control_sensorless0.p_ctrl);
}
```

Data Structures

struct	motor_120_control_sensorless_extended_cfg_t
struct	motor_120_control_sensorless_instance_ctrl_t

Enumerations

enum	motor_120_control_sensorless_draw_in_position_t
enum	motor_120_control_sensorless_pattern_change_flag_t

Data Structure Documentation

◆ motor_120_control_sensorless_extended_cfg_t

struct motor_120_control_sensorless_extended_cfg_t
Extended configurations for motor 120 control sensorless

Data Fields		
float	f4_stop_bemf	Value of stop motor BEMF (U+V+W)
float	f4_max_boot_v	Max output voltage for boot mode (V)
float	f4_carrier_freq	Carrier wave frequency (kHz)
int32_t	s4_angle_shift_adjust	Adjusting angle.
float	f4_boot_ref_v	Voltage reference when zero speed (V)
uint32_t	u4_v_up_time	Voltage lamping time.
uint32_t	u4_v_const_time	Voltage constant adjust time value (ms)
int32_t	s4_ol_start_rpm	Start speed (rpm)
int32_t	s4_ol_mode1_change_rpm	To mode2 change speed (rpm)
int32_t	s4_ol_mode2_change_rpm	To mode3 change speed (rpm)
float	f4_ol_start_refv	Start reference voltage (V)
float	f4_ol_mode1_rate_rpm	Increase rate of reference speed (rpm/control period)
float	f4_ol_mode2_rate_refv	Increase rate of reference voltage (v/control period)
float	f4_ol_mode2_rate_rpm	Increase rate of reference speed (rpm/control period)
float	f4_ol_mode3_rate_refv	Increase rate of reference voltage (v/control period)
float	f4_ol_mode3_max_refv	Openloop max voltage (V)
motor_120_driver_instance_t const *	p_motor_120_driver_instance	Motor 120 driver access module.
timer_instance_t const *	p_speed_cyclic_timer_instance	Cyclic process of speed control timer module.
timer_instance_t const *	p_speed_calc_timer_instance	Speed calculate timer module.

◆ motor_120_control_sensorless_instance_ctrl_t

struct motor_120_control_sensorless_instance_ctrl_t

120 control sensorless instance control block

Data Fields		
uint32_t	open	Used to determine if the channel is configured.
motor_120_control_status_t	active	Flag to set active/inactive the motor 120 control.

motor_120_control_run_mode_t	run_mode	Drive mode.
motor_120_control_timeout_error_flag_t	timeout_error_flag	Timeout error status.
motor_120_control_pattern_error_flag_t	pattern_error_flag	Bemf pattern error status.
motor_120_control_rotation_direction_t	direction	Rotational direction (0:CW ,1:CCW)
float	f4_ol_pattern_set_calc	Counts to change timing of open loop pattern.
float	f4_speed_calc_base	Base counts to calculate rotation speed.
float	f_rpm2rad	Translate value to radian/second to rpm.
float	f4_v_ref	Voltage reference (output of speed PI control)
uint32_t	u4_pwm_duty	PWM duty.
float	f4_ref_speed_rad	Motor speed reference.
float	f4_ref_speed_rad_ctrl	Motor speed reference for speed PI control.
float	f4_speed_rad	Motor speed.
uint32_t	u4_cnt_speed_pi	Counter for period of speed PI control.
motor_120_control_wait_stop_flag_t	flag_wait_stop	Flag for waiting for motor stop.
float	f4_vn_ad	Neutral voltage.
uint32_t	u4_cnt_adj_v	Voltage lamping count adjustment.
motor_120_control_sensorless_draw_in_position_t	flag_draw_in	Status of draw in a initial position.
motor_120_driver_phase_pattern_t	v_pattern	Voltage pattern.
uint32_t	u4_v_pattern_num	Selecting pattern number for openloop drive.
uint32_t	u4_bemf_signal	Pattern of BEMF.
uint32_t	u4_pre_bemf_signal	Previous pattern of BEMF.
motor_120_control_sensorless_pattern_change_flag_t	flag_pattern_change	Pattern change flag.
motor_120_control_speed_ref_t	flag_speed_ref	Speed reference flag.
motor_120_control_voltage_ref_t	flag_voltage_ref	Voltage reference flag.

uint32_t	u4_ol_signal	Pattern of BEMF.
uint32_t	u4_ol_pattern_set	Openloop frequency.
uint32_t	u4_cnt_ol_pattern_set	Counter for openloop pattern change.
uint32_t	u4_cnt_timeout	Counter for timeout error.
uint32_t	u4_bemf_timer_cnt	Value of timer counter.
uint32_t	u4_pre_bemf_timer_cnt	Previous value of timer counter.
int32_t	s4_timer_cnt_ave	Counts for 360 degrees.
uint32_t	u4_timer_cnt_buf[MOTOR_120_CONTROL_SENSORLESS_TIMES]	Counts for 60 degrees.
uint32_t	u4_timer_cnt_num	Array element number before 360 degrees.
uint32_t	u4_cnt_carrier	Counter for carrier interrupt.
uint32_t	u4_pre_cnt_carrier	Previous carrier interrupt count.
uint32_t	u4_angle_shift_cnt	Shift degrees count.
float	f4_pi_ctrl_err	PI control error.
float	f4_pi_ctrl_refi	PI control buffer of integral term.
motor_120_control_cfg_t const *	p_cfg	Pointer of configuration structure.
timer_callback_args_t	timer_args	For call timer callbackSet function.

Enumeration Type Documentation

◆ motor_120_control_sensorless_draw_in_position_t

enum motor_120_control_sensorless_draw_in_position_t	
Draw in a initial position	
Enumerator	
MOTOR_120_CONTROL_SENSORLESS_DRAW_IN_POSITION_INIT	Inital parameter.
MOTOR_120_CONTROL_SENSORLESS_DRAW_IN_POSITION_1ST_TIME	Draw in a initial position of the 1st time.
MOTOR_120_CONTROL_SENSORLESS_DRAW_IN_POSITION_2ND_TIME	Draw in a initial position of the 2nd time.

◆ motor_120_control_sensorless_pattern_change_flag_t

enum motor_120_control_sensorless_pattern_change_flag_t

Pattern change

Enumerator	
MOTOR_120_CONTROL_SENSORLESS_PATTERN_CHANGE_FLAG_CLEAR	Inital parameter.
MOTOR_120_CONTROL_SENSORLESS_PATTERN_CHANGE_FLAG_SET	Voltage pattern change.

Function Documentation

◆ RM_MOTOR_120_CONTROL_SENSORLESS_Open()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_Open (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_cfg_t const *const *p_cfg*)

Opens and configures the motor sensorless 120 detection module. Implements motor_120_control_api_t::open.

Example:

```
/* Initializes the module. */
    err =
RM_MOTOR_120_CONTROL_SENSORLESS_Open(g_motor_120_control_sensorless0.p_ctrl,
                                     g_motor_120_control_sensorless0.p_cfg);
```

Return values

FSP_SUCCESS	Motor driver successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_SENSORLESS_Close()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_Close (motor_120_control_ctrl_t *const *p_ctrl*)

Disables specified motor sensorless 120 detection module. Implements motor_120_control_api_t::close.

Example:

```
/* Close the module. */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Close(g_motor_120_control_sensorless0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_CONTROL_SENSORLESS_Run()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_Run (motor_120_control_ctrl_t *const *p_ctrl*)

Run motor (Start motor rotation). Implements motor_120_control_api_t::run.

Example:

```
/* Start motor rotation */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Run(g_motor_120_control_sensorless0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_CONTROL_SENSORLESS_Stop()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_Stop (motor_120_control_ctrl_t *const *p_ctrl*)

Stop motor (Stop motor rotation). Implements motor_120_control_api_t::stop.

Example:

```
/* Stop motor rotation */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Stop(g_motor_120_control_sensorless0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_CONTROL_SENSORLESS_Reset()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_Reset (motor_120_control_ctrl_t *const *p_ctrl*)

Reset variables of motor sensorless 120 detection module. Implements motor_120_control_api_t::reset.

Example:

```
/* Reset the process. */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_Reset(g_motor_120_control_sensorless0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_CONTROL_SENSORLESS_SpeedSet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_SpeedSet (motor_120_control_ctrl_t *const *p_ctrl*, float const *speed_rpm*)

Set speed[rpm]. Implements motor_120_control_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_SpeedSet(g_motor_120_control_sensorless0.p_ctrl,

DEF_120_CONTROL_SENSORLESSHALL_TEST_OVSPD_LIM);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_CONTROL_SENSORLESS_SpeedGet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_SpeedGet (motor_120_control_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm*)

Get speed. Implements motor_120_control_api_t::speedGet.

Example:

```
/* Get current motor speed */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_SpeedGet(g_motor_120_control_sensorless0.p_ctrl,
&smpl_speed);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_SENSORLESS_CurrentGet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_CurrentGet (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_driver_current_status_t *const *p_current_status*)

Get current. Implements motor_120_control_api_t::currentGet.

Example:

```
 /* Get current */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_CurrentGet(g_motor_120_control_sensorless0.p_ctrl,
&smpl_current_status);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_SENSORLESS_WaitStopFlagGet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_WaitStopFlagGet (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_wait_stop_flag_t *const *p_flag*)

Get wait stop flag. Implements motor_120_control_api_t::waitStopFlagGet.

Example:

```
 /* Get wait stop flag */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_WaitStopFlagGet
(g_motor_120_control_sensorless0.p_ctrl,
                                                            &smpl_wait_stop_flag);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_SENSORLESS_TimeoutErrorFlagGet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_TimeoutErrorFlagGet (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_timeout_error_flag_t *const *p_timeout_error_flag*)

Get timeout error flag. Implements motor_120_control_api_t::timeoutErrorFlagGet.

Example:

```
 /* Get timeout error flag */
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_TimeoutErrorFlagGet
(g_motor_120_control_sensorless0.p_ctrl,
                                                                &smpl_timeout_error_fl
ag);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_SENSORLESS_PatternErrorFlagGet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_PatternErrorFlagGet (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_pattern_error_flag_t *const *p_pattern_error_flag*)

Get pattern error flag. Implements motor_120_control_api_t::patternErrorFlagGet.

Example:

```
/* Get pattern error flag */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_PatternErrorFlagGet
(g_motor_120_control_sensorless0.p_ctrl,
                                                               &smpl_pattern_error_fl
ag);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_SENSORLESS_VoltageRefGet()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_VoltageRefGet (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_voltage_ref_t *const *p_voltage_ref*)

Get voltage ref. Implements motor_120_control_api_t::voltageRefGet.

Example:

```
/* Get voltage ref */
   (void)
RM_MOTOR_120_CONTROL_SENSORLESS_VoltageRefGet(g_motor_120_control_sensorless0.p_ctrl,
&smpl_voltage_ref);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_CONTROL_SENSORLESS_ParameterUpdate()

fsp_err_t RM_MOTOR_120_CONTROL_SENSORLESS_ParameterUpdate (motor_120_control_ctrl_t *const *p_ctrl*, motor_120_control_cfg_t const *const *p_cfg*)

Update the parameters of sensorless 120 detection module. Implements motor_120_control_api_t::parameterUpdate.

Example:

```
    (void)
RM_MOTOR_120_CONTROL_SENSORLESS_ParameterUpdate
(g_motor_120_control_sensorless0.p_ctrl,

g_motor_120_control_sensorless0.p_cfg);
```

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

4.2.99 Motor 120 Degree (rm_motor_120_degree)

Modules

Functions

fsp_err_t	RM_MOTOR_120_DEGREE_Open (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg) Configure the motor in register start mode. Implements motor_api_t::open. More...
fsp_err_t	RM_MOTOR_120_DEGREE_Close (motor_ctrl_t *const p_ctrl) Disables specified motor control block. Implements motor_api_t::close. More...
fsp_err_t	RM_MOTOR_120_DEGREE_Run (motor_ctrl_t *const p_ctrl) Run motor (Start motor rotation). Implements motor_api_t::run. More...

fsp_err_t	RM_MOTOR_120_DEGREE_Stop (motor_ctrl_t *const p_ctrl) Stop motor (Stop motor rotation). Implements motor_api_t::stop. More...
fsp_err_t	RM_MOTOR_120_DEGREE_Reset (motor_ctrl_t *const p_ctrl) Reset motor control block. Implements motor_api_t::reset. More...
fsp_err_t	RM_MOTOR_120_DEGREE_ErrorSet (motor_ctrl_t *const p_ctrl, motor_error_t const error) Set error information. Implements motor_api_t::errorSet. More...
fsp_err_t	RM_MOTOR_120_DEGREE_SpeedSet (motor_ctrl_t *const p_ctrl, float const speed_rpm) Set speed reference[rpm]. Implements motor_api_t::speedSet. More...
fsp_err_t	RM_MOTOR_120_DEGREE_StatusGet (motor_ctrl_t *const p_ctrl, uint8_t *const p_status) Get current control status. Implements motor_api_t::statusGet. More...
fsp_err_t	RM_MOTOR_120_DEGREE_SpeedGet (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm) Get rotational speed. Implements motor_api_t::speedGet. More...
fsp_err_t	RM_MOTOR_120_DEGREE_WaitStopFlagGet (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag_wait_stop) Get wait stop flag. Implements motor_api_t::waitStopFlagGet. More...
fsp_err_t	RM_MOTOR_120_DEGREE_ErrorCheck (motor_ctrl_t *const p_ctrl, uint16_t *const p_error) Check the occurunce of error. Implements motor_api_t::errorCheck. More...
fsp_err_t	RM_MOTOR_120_DEGREE_PositionSet (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position) Set position reference. Implements motor_api_t::positionSet. More...

fsp_err_t	RM_MOTOR_120_DEGREE_AngleGet (motor_ctrl_t *const p_ctrl, float *const p_angle_rad) Set position reference. Implements motor_api_t::angleGet. More...

Detailed Description

Usual control of a SPM (Surface Permanent Magnet) motor on RA MCUs. This module implements the Motor 120 Degree (rm_motor_120_degree).

Overview

The motor 120 degree control is used to control a motor rotation in an appication. This module is implemented with using a SPM motor. User can start/stop motor rotation simply.

Features

The motor 120 degree module has below features.

- Start/Stop a motor rotation
- Error detection (over current, over speed, over voltage, low voltage)

Configuration

Build Time Configurations for rm_motor_120_degree

The following build time configurations are defined in fsp_cfg/rm_motor_120_degree_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > Motor 120 degree control (rm_motor_120_degree)

This module can be added to the Stacks tab via New Stack > Motor > Motor 120 degree control (rm_motor_120_degree).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_120_degree0	Module name.
General > Limit of over current (A)	Must be a valid non-negative value.	4.0	Limit of over current.(Detection threshold)
General > Limit of over voltage (V)	Must be a valid non-negative value.	28.0	Limit of over voltage.(Detection threshold)

General > Limit of over speed (rpm)	Must be a valid non-negative value.	3000.0	Limit of over speed.(Detection threshold)
General > Limit of low voltage (V)	Must be a valid non-negative value.	14.0	Limit of low voltage.(Detection threshold)
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function.

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple status transition process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

- Set the limit of electric current with non-negative value.
- Set the limit of input voltage with non-negative value.
- Set the limit of rotational speed with non-negative value.

Examples

Basic Example

This is a basic example of minimal use of the motor 120 degree in an application.

```
void motor_120_degree_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_120_DEGREE_Open(g_motor_120_degree0.p_ctrl,
g_motor_120_degree0.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Set speed reference before motor run */
    (void) RM_MOTOR_120_DEGREE_SpeedSet(g_motor_120_degree0.p_ctrl,
DEF_120_DEGREE_SPEED_REF);
 /* Start motor rotation */
    (void) RM_MOTOR_120_DEGREE_Run(g_motor_120_degree0.p_ctrl);
 /* Get current status */
```

```
    (void) RM_MOTOR_120_DEGREE_StatusGet(g_motor_120_degree0.p_ctrl, &smpl_status);
 /* Get current motor speed */
    (void) RM_MOTOR_120_DEGREE_SpeedGet(g_motor_120_degree0.p_ctrl, &smpl_speed);
 /* Get wait stop flag */
    (void) RM_MOTOR_120_DEGREE_WaitStopFlagGet(g_motor_120_degree0.p_ctrl,
&smpl_wait_stop_flag);
 /* Check error */
    (void) RM_MOTOR_120_DEGREE_ErrorCheck(g_motor_120_degree0.p_ctrl, &smpl_error);
 /* Stop motor rotation */
    (void) RM_MOTOR_120_DEGREE_Stop(g_motor_120_degree0.p_ctrl);
    (void) RM_MOTOR_120_DEGREE_ErrorSet(g_motor_120_degree0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
 /* Reset the process. */
    (void) RM_MOTOR_120_DEGREE_Reset(g_motor_120_degree0.p_ctrl);
 /* Close the module. */
    (void) RM_MOTOR_120_DEGREE_Close(g_motor_120_degree0.p_ctrl);
}
```

Data Structures

struct	motor_120_degree_statemachine_t
struct	motor_120_degree_extended_cfg_t

Enumerations

enum	motor_120_degree_ctrl_status_t
enum	motor_120_degree_ctrl_event_t

Data Structure Documentation

◆ motor_120_degree_statemachine_t

struct motor_120_degree_statemachine_t

Statemachine structure for motor 120 degree

Data Fields		
motor_120_degree_ctrl_status_t	status	The current system status.
motor_120_degree_ctrl_status_t	status_next	The next system status.
motor_120_degree_ctrl_event_t	current_event	The current event index.

uint16_t	u2_error_status	The error information.

◆ motor_120_degree_extended_cfg_t

struct motor_120_degree_extended_cfg_t		
Extended configurations for motor 120 degree		
Data Fields		
motor_120_control_instance_t const *	p_motor_120_control_instance	120 degree control Instance
float	f_overcurrent_limit	Over-current limit (A)
float	f_overvoltage_limit	Over-voltage limit (V)
float	f_overspeed_limit	Over-speed limit (rad/s)
float	f_lowvoltage_limit	Low-voltage limit (V)

Enumeration Type Documentation

◆ motor_120_degree_ctrl_status_t

enum motor_120_degree_ctrl_status_t	
Control state	
Enumerator	
MOTOR_120_DEGREE_CTRL_STATUS_STOP	Stop mode.
MOTOR_120_DEGREE_CTRL_STATUS_RUN	Run mode.
MOTOR_120_DEGREE_CTRL_STATUS_ERROR	Error mode.

◆ motor_120_degree_ctrl_event_t

enum motor_120_degree_ctrl_event_t	
Control event	
Enumerator	
MOTOR_120_DEGREE_CTRL_EVENT_STOP	Stop event.
MOTOR_120_DEGREE_CTRL_EVENT_RUN	Run event.
MOTOR_120_DEGREE_CTRL_EVENT_ERROR	Error event.
MOTOR_120_DEGREE_CTRL_EVENT_RESET	Reset event.

Function Documentation

◆ RM_MOTOR_120_DEGREE_Open()

fsp_err_t RM_MOTOR_120_DEGREE_Open (motor_ctrl_t *const *p_ctrl*, motor_cfg_t const *const *p_cfg*)

Configure the motor in register start mode. Implements motor_api_t::open.

Example:

```
/* Initializes the module. */
    err = RM_MOTOR_120_DEGREE_Open(g_motor_120_degree0.p_ctrl,
g_motor_120_degree0.p_cfg);
```

Return values

FSP_SUCCESS	Successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.

Note

This function should only be called once as motor configuration registers can only be written to once so subsequent calls will have no effect.

◆ RM_MOTOR_120_DEGREE_Close()

fsp_err_t RM_MOTOR_120_DEGREE_Close (motor_ctrl_t *const *p_ctrl*)

Disables specified motor control block. Implements motor_api_t::close.

Example:

```
/* Close the module. */
    (void) RM_MOTOR_120_DEGREE_Close(g_motor_120_degree0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_120_DEGREE_Run()

fsp_err_t RM_MOTOR_120_DEGREE_Run (motor_ctrl_t *const *p_ctrl*)

Run motor (Start motor rotation). Implements motor_api_t::run.

Example:

```
/* Start motor rotation */
    (void) RM_MOTOR_120_DEGREE_Run(g_motor_120_degree0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_120_DEGREE_Stop()

fsp_err_t RM_MOTOR_120_DEGREE_Stop (motor_ctrl_t *const *p_ctrl*)

Stop motor (Stop motor rotation). Implements motor_api_t::stop.

Example:

```
/* Stop motor rotation */
    (void) RM_MOTOR_120_DEGREE_Stop(g_motor_120_degree0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully stopped.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_120_DEGREE_Reset()

fsp_err_t RM_MOTOR_120_DEGREE_Reset (motor_ctrl_t *const *p_ctrl*)

Reset motor control block. Implements motor_api_t::reset.

Example:

```
/* Reset the process. */
   (void) RM_MOTOR_120_DEGREE_Reset(g_motor_120_degree0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_120_DEGREE_ErrorSet()

fsp_err_t RM_MOTOR_120_DEGREE_ErrorSet (motor_ctrl_t *const *p_ctrl*, motor_error_t const *error*)

Set error information. Implements motor_api_t::errorSet.

Example:

```
   (void) RM_MOTOR_120_DEGREE_ErrorSet(g_motor_120_degree0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
```

Return values

FSP_SUCCESS	Successfully set error infomation.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_120_DEGREE_SpeedSet()

fsp_err_t RM_MOTOR_120_DEGREE_SpeedSet (motor_ctrl_t *const *p_ctrl*, float const *speed_rpm*)

Set speed reference[rpm]. Implements motor_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
   (void) RM_MOTOR_120_DEGREE_SpeedSet(g_motor_120_degree0.p_ctrl,
DEF_120_DEGREE_SPEED_REF);
```

Return values

FSP_SUCCESS	Successfully set speed reference.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_120_DEGREE_StatusGet()

fsp_err_t RM_MOTOR_120_DEGREE_StatusGet (motor_ctrl_t *const *p_ctrl*, uint8_t *const *p_status*)

Get current control status. Implements motor_api_t::statusGet.

Example:

```
/* Get current status */
   (void) RM_MOTOR_120_DEGREE_StatusGet(g_motor_120_degree0.p_ctrl, &smpl_status);
```

Return values

FSP_SUCCESS	Successfully got current control status.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_POINTER	Data received pointer is invalid.

Note

◆ RM_MOTOR_120_DEGREE_SpeedGet()

fsp_err_t RM_MOTOR_120_DEGREE_SpeedGet (motor_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm*)

Get rotational speed. Implements motor_api_t::speedGet.

Example:

```
/* Get current motor speed */
    (void) RM_MOTOR_120_DEGREE_SpeedGet(g_motor_120_degree0.p_ctrl, &smpl_speed);
```

Return values

FSP_SUCCESS	Successfully got rotational speed.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_POINTER	Data received pointer is invalid.

Note

◆ RM_MOTOR_120_DEGREE_WaitStopFlagGet()

fsp_err_t RM_MOTOR_120_DEGREE_WaitStopFlagGet (motor_ctrl_t *const *p_ctrl*, motor_wait_stop_flag_t *const *p_flag_wait_stop*)

Get wait stop flag. Implements motor_api_t::waitStopFlagGet.

Example:

```
/* Get wait stop flag */
    (void) RM_MOTOR_120_DEGREE_WaitStopFlagGet(g_motor_120_degree0.p_ctrl,
&smpl_wait_stop_flag);
```

Return values

FSP_SUCCESS	Successfully got wait stop flag.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_POINTER	Data received pointer is invalid.

Note

◆ RM_MOTOR_120_DEGREE_ErrorCheck()

fsp_err_t RM_MOTOR_120_DEGREE_ErrorCheck (motor_ctrl_t *const *p_ctrl*, uint16_t *const *p_error*)

Check the occurunce of error. Implements motor_api_t::errorCheck.

Example:

```
/* Check error */
   (void) RM_MOTOR_120_DEGREE_ErrorCheck(g_motor_120_degree0.p_ctrl, &smpl_error);
```

Return values

FSP_SUCCESS	Successfully error checke process.
FSP_ERR_ASSERTION	p_ctrl is NULL.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_POINTER	Data received pointer is invalid.

Note

◆ RM_MOTOR_120_DEGREE_PositionSet()

fsp_err_t RM_MOTOR_120_DEGREE_PositionSet (motor_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position*)

Set position reference. Implements motor_api_t::positionSet.

Example:

Return values

FSP_ERR_UNSUPPORTED	Unsupported.

Note

◆ RM_MOTOR_120_DEGREE_AngleGet()

fsp_err_t RM_MOTOR_120_DEGREE_AngleGet (motor_ctrl_t *const *p_ctrl*, float *const *p_angle_rad*)

Set position reference. Implements motor_api_t::angleGet.

Example:

Return values

FSP_ERR_UNSUPPORTED	Unsupported.

Note

4.2.100 Motor 120 degree driver (rm_motor_120_driver)

Modules

Functions

fsp_err_t	RM_MOTOR_120_DRIVER_Open (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_cfg_t const *const p_cfg) Opens and configures the motor 120 driver module. Implements motor_120_driver_api_t::open. More...
fsp_err_t	RM_MOTOR_120_DRIVER_Close (motor_120_driver_ctrl_t *const p_ctrl) Disables specified motor 120 driver module. Implements motor_120_driver_api_t::close. More...
fsp_err_t	RM_MOTOR_120_DRIVER_Run (motor_120_driver_ctrl_t *const p_ctrl) Run motor (Start motor rotation). Implements motor_120_driver_api_t::run. More...
fsp_err_t	RM_MOTOR_120_DRIVER_Stop (motor_120_driver_ctrl_t *const p_ctrl) Stop motor (Stop motor rotation). Implements motor_120_driver_api_t::stop. More...
fsp_err_t	RM_MOTOR_120_DRIVER_Reset (motor_120_driver_ctrl_t *const p_ctrl) Reset variables of motor 120 driver module. Implements

	motor_120_driver_api_t::reset. More...
fsp_err_t	RM_MOTOR_120_DRIVER_PhaseVoltageSet (motor_120_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage) Set phase voltage data to calculate PWM duty. Implements motor_120_driver_api_t::phaseVoltageSet. More...
fsp_err_t	RM_MOTOR_120_DRIVER_PhasePatternSet (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_phase_pattern_t const pattern) Set phase voltage pattern. Implements motor_120_driver_api_t::phasePatternSet. More...
fsp_err_t	RM_MOTOR_120_DRIVER_CurrentGet (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status) Get calculated phase current, Vdc & Va_max data. Implements motor_120_driver_api_t::currentGet. More...
fsp_err_t	RM_MOTOR_120_DRIVER_CurrentOffsetCalc (motor_120_driver_ctrl_t *const p_ctrl) current offset detection. Implements motor_120_driver_api_t::currentOffsetCalc More...
fsp_err_t	RM_MOTOR_120_DRIVER_FlagCurrentOffsetGet (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_flag_offset_calc_t *const p_flag_offset) Get the flag of finish current offset detection. Implements motor_120_driver_api_t::flagCurrentOffsetGet. More...
fsp_err_t	RM_MOTOR_120_DRIVER_ParameterUpdate (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_cfg_t const *const p_cfg) Update the parameters of 120 driver module. Implements motor_120_driver_api_t::parameterUpdate. More...

Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Interface.

Overview

The motor 120 degree driver module is used to translate phase voltage to PWM duty and output PWM, and detect phase current, voltage and main line voltage. This module should be called cyclically at included A/D conversion finish interrupt.

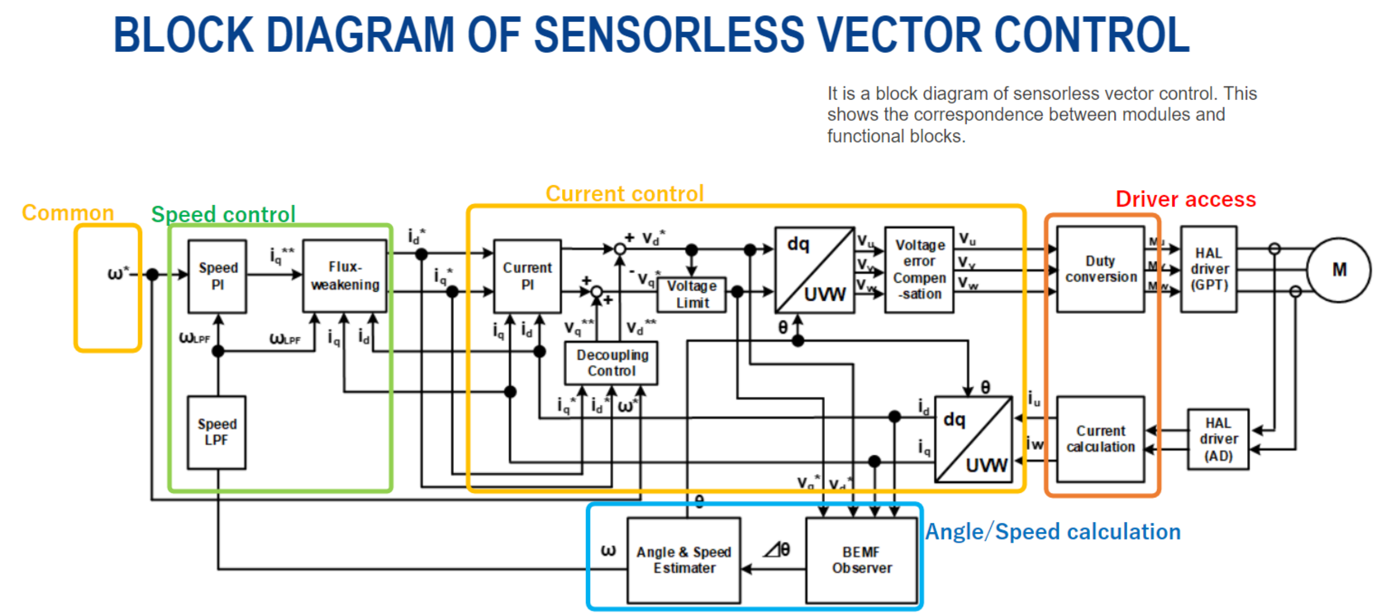


Figure 209: Image of Current Control Module(yellow block)

Features

The motor 120 degree driver module has below features.

- Calculate each phase(U/V/W) PWM duty according to reference and output PWM.
- Detect each phase current, phase voltage and main line voltage.
- Detect and correct A/D offset at phase current and voltage channel

Configuration

Build Time Configurations for rm_motor_120_driver

The following build time configurations are defined in fsp_cfg/rm_motor_120_driver_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > ADC and PWM modulation (rm_motor_120_driver)

This module can be added to the Stacks tab via New Stack > Motor > ADC and PWM modulation (rm_motor_120_driver).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_120_driver0	Module name.

General > 120 degree control type	• Sensorless • Hall	Sensorless	120 degree control type
General > PWM output port UP	Manual Entry	BSP_IO_PORT_06_PIN_01	PWM output port UP
General > PWM output port UN	Manual Entry	BSP_IO_PORT_06_PIN_00	PWM output port UN
General > PWM output port VP	Manual Entry	BSP_IO_PORT_01_PIN_13	PWM output port VP
General > PWM output port VN	Manual Entry	BSP_IO_PORT_01_PIN_14	PWM output port VN
General > PWM output port WP	Manual Entry	BSP_IO_PORT_01_PIN_11	PWM output port WP
General > PWM output port WN	Manual Entry	BSP_IO_PORT_01_PIN_12	PWM output port WN
General > PWM timer frequency (MHz)	Must be a valid non-negative value.	120	GPT PWM timer frequency
General > PWM carrier period (Microseconds)	Must be a valid non-negative value.	50	GPT PWM carrier period
General > Dead time (Raw counts)	Must be a valid non-negative value.	240	GPT PWM dead time
General > Current range (A)	Must be a valid non-negative value.	27.5	Current range to measure(Maximum input current)
General > Voltage range (V)	Must be a valid non-negative value.	111.0	Voltage range to measure(Maximum input main line voltage)
General > Resolution of A/D conversion	Manual Entry	0xFFF	Resolution of A/D conversion
General > Offset of A/D conversion for current	Manual Entry	0x745	Offset of A/D conversion for current
General > Conversion level of A/D conversion for voltage	Must be a valid non-negative value.	0.66	Conversion level of A/D conversion for voltage
General > Counts for current offset measurement	Must be a valid non-negative value.	500	How many times to measure current offset
General > Input voltage	Must be a valid non-negative value.	24.0	Input voltage
General > A/D conversion channel for U phase current	Manual Entry	ADC_CHANNEL_0	Specify the A/D channel for U phase current
General > A/D	Manual Entry	ADC_CHANNEL_2	Specify the A/D

conversion channel for W phase current			channel for W phase current
General > A/D conversion channel for main line voltage	Manual Entry	ADC_CHANNEL_5	Specify the A/D channel for main line voltage
General > A/D conversion channel for U phase voltage	Manual Entry	ADC_CHANNEL_18	Specify the A/D channel for U phase voltage
General > A/D conversion channel for V phase voltage	Manual Entry	ADC_CHANNEL_20	Specify the A/D channel for V phase voltage
General > A/D conversion channel for W phase voltage	Manual Entry	ADC_CHANNEL_6	Specify the A/D channel for W phase voltage
General > GTIOCA stop level	• Pin Level Low • Pin Level High	Pin Level High	Select the behavior of the output pin when the timer is stopped.
General > GTIOCB stop level	• Pin Level Low • Pin Level High	Pin Level High	Select the behavior of the output pin when the timer is stopped.
Modulation > Maximum duty	Must be a valid non-negative value.	0.9375	Maximum duty of PWM
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called at A/D conversion finish interrupt.

Clock Configuration

Set used clock with included GPT timer.

Pin Configuration

Depend on included GPT three phase module and ADC module.

Usage Notes

Limitations

Basically no limitation exists.

Examples

Basic Example

This is a basic example of minimal use of the Motor 120 degree driver in an application.

```
void motor_120_driver_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_120_DRIVER_Open(&g_motor_120_driver0.p_ctrl,
&g_motor_120_driver0.p_cfg);
 /* Start PWM output */
    err = RM_MOTOR_120_DRIVER_Run(&g_motor_120_driver0.p_ctrl);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at cyclic interrupt (e.g. included GPT PWM carrier
intterupt).
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Caclutarion of current offset */
       (void) RM_MOTOR_120_DRIVER_CurrentOffsetCalc(&g_motor_120_driver0.p_ctrl);
 /* Get electric current, main line voltage and maximum voltage component */
       (void) RM_MOTOR_120_DRIVER_CurrentGet(&g_motor_120_driver0.p_ctrl,
&g_current_status);
 /* Get the flag of A/D convderted current offset */
       (void) RM_MOTOR_120_DRIVER_FlagCurrentOffsetGet(&g_motor_120_driver0.p_ctrl,
&u1_get_flg_offset);
 // Perform current control process here
 /* Set phase voltage */
       (void) RM_MOTOR_120_DRIVER_PhaseVoltageSet(&g_motor_120_driver0.p_ctrl, 1.0F,
1.0F, 1.0F);
 /* Set phase pattern */
       (void) RM_MOTOR_120_DRIVER_PhasePatternSet(&g_motor_120_driver0.p_ctrl,
MOTOR_120_DRIVER_API_VP_ON_WN_PWM);
       (void) RM_MOTOR_120_DRIVER_ParameterUpdate(&g_motor_120_driver0.p_ctrl,
&g_motor_120_driver0.p_cfg);
    }
    (void) RM_MOTOR_120_DRIVER_Stop(&g_motor_120_driver0.p_ctrl);
```

```
    (void) RM_MOTOR_120_DRIVER_Reset(&g_motor_120_driver0.p_ctrl);
 //

    (void) RM_MOTOR_120_DRIVER_Close(&g_motor_120_driver0.p_ctrl);
}
```

Data Structures

struct	motor_120_driver_modulation_t
struct	motor_120_driver_extended_cfg_t

Enumerations

enum	motor_120_driver_status_t
enum	motor_120_driver_type_t

Data Structure Documentation

◆ motor_120_driver_modulation_t

struct motor_120_driver_modulation_t

Modulation parameter

Data Fields

float	f4_vdc	Main line voltage (Vdc) (V)
float	f4_max_duty	Maximum duty cycle.
float	f4_min_duty	Minimum duty cycle.
float	f4_neutral_duty	Duty cycle that represents 0 (V)

◆ motor_120_driver_extended_cfg_t

struct motor_120_driver_extended_cfg_t

Extended configurations for motor 120 driver

Data Fields

adc_instance_t const *	p_adc_instance	ADC module instance.
three_phase_instance_t const *	p_three_phase_instance	PWM output module instance (GPT three phase)
motor_120_driver_type_t	motor_120_type	120 degree control type
adc_channel_t	iu_ad_ch	A/D channel for U phase current.
adc_channel_t	iw_ad_ch	A/D channel for W phase current.

adc_channel_t	vdc_ad_ch	A/D channel for main line voltage.
adc_channel_t	vu_ad_ch	A/D channel for U phase voltage.
adc_channel_t	vv_ad_ch	A/D channel for V phase voltage.
adc_channel_t	vw_ad_ch	A/D channel for W phase voltage.
bsp_io_port_pin_t	port_up	PWM output port UP.
bsp_io_port_pin_t	port_un	PWM output port UN.
bsp_io_port_pin_t	port_vp	PWM output port VP.
bsp_io_port_pin_t	port_vn	PWM output port VN.
bsp_io_port_pin_t	port_wp	PWM output port WP.
bsp_io_port_pin_t	port_wn	PWM output port WN.
uint32_t	u4_pwm_timer_freq	PWM timer frequency (MHz)
uint32_t	u4_pwm_carrier_freq	PWM carrier frequency (kHz)
uint32_t	u4_deadtime	PWM deadtime (usec)
float	f_current_range	A/D current measure range (max current) (A)
float	f_vdc_range	A/D main line voltage measure range (max voltage) (V)
float	f_ad_resolution	A/D resolution.
float	f_ad_current_offset	A/D offset (Center value)
float	f_ad_voltage_conversion	A/D conversion level.
uint32_t	u4_offset_calc_count	Calculation counts for current offset.
motor_120_driver_modulation_t	mod_param	Modulation parameter.

Enumeration Type Documentation

◆ motor_120_driver_status_t

enum motor_120_driver_status_t

120 driver active flag

Enumerator	
MOTOR_120_DRIVER_STATUS_INACTIVE	120 driver status inactive
MOTOR_120_DRIVER_STATUS_ACTIVE	120 driver status active

◆ motor_120_driver_type_t

enum motor_120_driver_type_t

120 degree control type

Enumerator	
MOTOR_120_DRIVER_TYPE_SENSORLESS	120 degree sensorless control
MOTOR_120_DRIVER_TYPE_HALL	120 degree hall control

Function Documentation

◆ RM_MOTOR_120_DRIVER_Open()

fsp_err_t RM_MOTOR_120_DRIVER_Open (motor_120_driver_ctrl_t *const *p_ctrl*, motor_120_driver_cfg_t const *const *p_cfg*)

Opens and configures the motor 120 driver module. Implements motor_120_driver_api_t::open.

Example:

```
 /* Initializes the module. */
    err = RM_MOTOR_120_DRIVER_Open(&g_motor_120_driver0.p_ctrl,
&g_motor_120_driver0.p_cfg);
```

Return values

FSP_SUCCESS	Motor 120 driver successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_DRIVER_Close()

fsp_err_t RM_MOTOR_120_DRIVER_Close (motor_120_driver_ctrl_t *const *p_ctrl*)

Disables specified motor 120 driver module. Implements motor_120_driver_api_t::close.

Example:

```
    (void) RM_MOTOR_120_DRIVER_Close(&g_motor_120_driver0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_DRIVER_Run()

fsp_err_t RM_MOTOR_120_DRIVER_Run (motor_120_driver_ctrl_t *const *p_ctrl*)

Run motor (Start motor rotation). Implements motor_120_driver_api_t::run.

Example:

```
/* Start PWM output */
    err = RM_MOTOR_120_DRIVER_Run(&g_motor_120_driver0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_DRIVER_Stop()

fsp_err_t RM_MOTOR_120_DRIVER_Stop (motor_120_driver_ctrl_t *const *p_ctrl*)

Stop motor (Stop motor rotation). Implements motor_120_driver_api_t::stop.

Example:

```
(void) RM_MOTOR_120_DRIVER_Stop(&g_motor_120_driver0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_DRIVER_Reset()

fsp_err_t RM_MOTOR_120_DRIVER_Reset (motor_120_driver_ctrl_t *const *p_ctrl*)

Reset variables of motor 120 driver module. Implements motor_120_driver_api_t::reset.

Example:

```
(void) RM_MOTOR_120_DRIVER_Reset(&g_motor_120_driver0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_DRIVER_PhaseVoltageSet()

fsp_err_t RM_MOTOR_120_DRIVER_PhaseVoltageSet (motor_120_driver_ctrl_t *const *p_ctrl*, float const *u_voltage*, float const *v_voltage*, float const *w_voltage*)

Set phase voltage data to calculate PWM duty. Implements motor_120_driver_api_t::phaseVoltageSet.

Example:

```
/* Set phase voltage */
      (void) RM_MOTOR_120_DRIVER_PhaseVoltageSet(&g_motor_120_driver0.p_ctrl, 1.0F,
1.0F, 1.0F);
```

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_DRIVER_PhasePatternSet()

fsp_err_t RM_MOTOR_120_DRIVER_PhasePatternSet (motor_120_driver_ctrl_t *const *p_ctrl*, motor_120_driver_phase_pattern_t const *pattern*)

Set phase voltage pattern. Implements motor_120_driver_api_t::phasePatternSet.

Example:

```
/* Set phase pattern */
      (void) RM_MOTOR_120_DRIVER_PhasePatternSet(&g_motor_120_driver0.p_ctrl,
MOTOR_120_DRIVER_API_VP_ON_WN_PWM);
```

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_120_DRIVER_CurrentGet()

fsp_err_t RM_MOTOR_120_DRIVER_CurrentGet (motor_120_driver_ctrl_t *const *p_ctrl*, motor_120_driver_current_status_t *const *p_current_status*)

Get calculated phase current, Vdc & Va_max data. Implements motor_120_driver_api_t::currentGet .

Example:

```
/* Get electric current, main line voltage and maximum voltage component */
      (void) RM_MOTOR_120_DRIVER_CurrentGet(&g_motor_120_driver0.p_ctrl,
&g_current_status);
```

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_DRIVER_CurrentOffsetCalc()

fsp_err_t RM_MOTOR_120_DRIVER_CurrentOffsetCalc (motor_120_driver_ctrl_t *const *p_ctrl*)

current offset detection. Implements motor_120_driver_api_t::currentOffsetCalc

Example:

```
/* Caclutarion of current offset */
      (void) RM_MOTOR_120_DRIVER_CurrentOffsetCalc(&g_motor_120_driver0.p_ctrl);
```

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_DRIVER_FlagCurrentOffsetGet()

fsp_err_t RM_MOTOR_120_DRIVER_FlagCurrentOffsetGet (motor_120_driver_ctrl_t *const *p_ctrl*, motor_120_driver_flag_offset_calc_t *const *p_flag_offset*)

Get the flag of finish current offset detection. Implements motor_120_driver_api_t::flagCurrentOffsetGet.

Example:

```
/* Get the flag of A/D convderted current offset */
      (void) RM_MOTOR_120_DRIVER_FlagCurrentOffsetGet(&g_motor_120_driver0.p_ctrl,
&u1_get_flg_offset);
```

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_120_DRIVER_ParameterUpdate()

fsp_err_t RM_MOTOR_120_DRIVER_ParameterUpdate (motor_120_driver_ctrl_t *const *p_ctrl*, motor_120_driver_cfg_t const *const *p_cfg*)

Update the parameters of 120 driver module. Implements motor_120_driver_api_t::parameterUpdate.

Example:

```
      (void) RM_MOTOR_120_DRIVER_ParameterUpdate(&g_motor_120_driver0.p_ctrl,
&g_motor_120_driver0.p_cfg);
```

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

4.2.101 Motor Current (rm_motor_current)

Modules

Functions

fsp_err_t	RM_MOTOR_CURRENT_Open (motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg) Opens and configures the Motor Current Module. Implements motor_current_api_t::open. More...
fsp_err_t	RM_MOTOR_CURRENT_Close (motor_current_ctrl_t *const p_ctrl) Disables specified Motor Current Module. Implements motor_current_api_t::close. More...
fsp_err_t	RM_MOTOR_CURRENT_Reset (motor_current_ctrl_t *const p_ctrl) Reset variables of Motor Current Module. Implements motor_current_api_t::reset. More...
fsp_err_t	RM_MOTOR_CURRENT_Run (motor_current_ctrl_t *const p_ctrl) Run(Start) the Current Control. Implements motor_current_api_t::run. More...
fsp_err_t	RM_MOTOR_CURRENT_ParameterSet (motor_current_ctrl_t *const p_ctrl, motor_current_input_t const *const p_st_input) Set (Input) Parameter Data. Implements motor_current_api_t::parameterSet. More...
fsp_err_t	RM_MOTOR_CURRENT_CurrentReferenceSet (motor_current_ctrl_t *const p_ctrl, float const id_reference, float const iq_reference) Set Current Reference Data. Implements motor_current_api_t::currentReferenceSet. More...
fsp_err_t	RM_MOTOR_CURRENT_SpeedPhaseSet (motor_current_ctrl_t *const p_ctrl, float const speed, float const phase) Set Current Speed & rotor phase Data. Implements motor_current_api_t::speedPhaseSet. More...
fsp_err_t	RM_MOTOR_CURRENT_CurrentSet (motor_current_ctrl_t *const p_ctrl, motor_current_input_current_t const *const p_st_current, motor_current_input_voltage_t const *const p_st_voltage) Set d/q-axis Current & Voltage Data. Implements motor_current_api_t::currentSet. More...

fsp_err_t	RM_MOTOR_CURRENT_ParameterGet (motor_current_ctrl_t *const p_ctrl, motor_current_output_t *const p_st_output) Get Output Parameters. Implements motor_current_api_t::parameterGet. More...
fsp_err_t	RM_MOTOR_CURRENT_CurrentGet (motor_current_ctrl_t *const p_ctrl, float *const p_id, float *const p_iq) Get d/q-axis Current. Implements motor_current_api_t::currentGet. More...
fsp_err_t	RM_MOTOR_CURRENT_PhaseVoltageGet (motor_current_ctrl_t *const p_ctrl, motor_current_get_voltage_t *const p_voltage) Gets the set phase voltage. Implements motor_current_api_t::phaseVoltageGet. More...
fsp_err_t	RM_MOTOR_CURRENT_ParameterUpdate (motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg) Update the parameters of Current Control. Implements motor_current_api_t::parameterUpdate. More...
void	rm_motor_current_encoder_cyclic (motor_current_instance_t const *p_ctrl)

Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor current Interface.

Overview

The motor current is used to control the electric current of motor rotation in an appication. This module should be called cyclically after the A/D conversion of electric current of each phase in an application. This module calculates each phase voltage with input current reference, electric current and rotor angle.

Features

The Motor Current Module has below features.

- Calculate each phase(U/V/W) voltage.
- Decoupling Control.
- Voltage Error Compensation.

Configuration

Build Time Configurations for rm_motor_current

The following build time configurations are defined in fsp_cfg/rm_motor_current_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > Motor Current Controller (rm_motor_current)

This module can be added to the Stacks tab via New Stack > Motor > Motor Current Controller (rm_motor_current).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_current0	Module name.
General > Sensor type	• Sensorless • Encoder	Sensorless	Select sensor type
General > Shunt type	• 1 shunt • 2 shunt • 3 shunt	2 shunt	Select shunt type
General > Current control decimation	Must be a valid non-negative value.	0	Decimation of current control.
General > PWM carrier frequency (kHz)	Must be a valid non-negative value.	20.0F	PWM carrier frequency.
General > Input voltage (V)	Must be a valid non-negative value.	24.0F	Input voltage for limitation of current PI control.
General > Sample delay compensation	• Disable • Enable	Enable	Select enable/disable sample delay compensation.
General > Voltage error compensation	• Disable • Enable	Enable	Select enable/disable voltage error compensation.
General > Voltage error compensation table of voltage 1	Must be a valid non-negative value.	0.672F	Voltage error compensation table of voltage.
General > Voltage error compensation table of voltage 2	Must be a valid non-negative value.	0.945F	Voltage error compensation table of voltage.
General > Voltage error compensation table of voltage 3	Must be a valid non-negative value.	1.054F	Voltage error compensation table of voltage.
General > Voltage error compensation	Must be a valid non-negative value.	1.109F	Voltage error compensation table of

table of voltage 4			voltage.
General > Voltage error compensation table of voltage 5	Must be a valid non-negative value.	1.192F	Voltage error compensation table of voltage.
General > Voltage error compensation table of current 1	Must be a valid non-negative value.	0.013F	Voltage error compensation table of current.
General > Voltage error compensation table of current 2	Must be a valid non-negative value.	0.049F	Voltage error compensation table of current.
General > Voltage error compensation table of current 3	Must be a valid non-negative value.	0.080F	Voltage error compensation table of current.
General > Voltage error compensation table of current 4	Must be a valid non-negative value.	0.184F	Voltage error compensation table of current.
General > Voltage error compensation table of current 5	Must be a valid non-negative value.	0.751F	Voltage error compensation table of current.
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called at A/D conversion finish interrupt.
Design Parameter > Current PI loop omega (Hz)	Must be a valid non-negative value.	300.0F	Current PI loop omega
Design Parameter > Current PI loop zeta	Must be a valid non-negative value.	1.0F	Current PI loop zeta
Motor Parameter > Pole pairs	Must be a valid non-negative value.	2	Pole pairs
Motor Parameter > Resistance (ohm)	Must be a valid non-negative value.	8.5F	Resistance
Motor Parameter > Inductance of d-axis (H)	Must be a valid non-negative value.	0.0045F	Inductance of d-axis
Motor Parameter > Inductance of q-axis (H)	Must be a valid non-negative value.	0.0045F	Inductance of q-axis
Motor Parameter > Permanent magnetic flux (Wb)	Must be a valid non-negative value.	0.02159F	Permanent magnetic flux
Motor Parameter > Rotor inertia (kgm^2)	Must be a valid non-negative value.	0.0000028F	Rotor inertia

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

- Set the Period of Current Control with none-negative value.
- Set the Reference Voltage with none-negative value.

Examples

Basic Example

This is a basic example of minimal use of the Motor Current in an application.

```
void motor_current_basic_example (void)
{
 motor_current_input_current_t temp_input_current;
 motor_current_input_voltage_t temp_input_voltage;
 motor_current_get_voltage_t   temp_get_voltage;
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_CURRENT_Open(g_test_motor_current.p_ctrl,
g_test_motor_current.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at A/D conversion finish interrupt.
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Set current reference before get phase voltage */
       (void) RM_MOTOR_CURRENT_CurrentReferenceSet(g_test_motor_current.p_ctrl, 1.0F,
1.0F);
 /* Set speed and phase data before get phase voltage */
       (void) RM_MOTOR_CURRENT_SpeedPhaseSet(g_test_motor_current.p_ctrl, 104.72F,
```

```
1.0F);
       temp_input_current.iu    = 1.0F;
       temp_input_current.iv    = 1.0F;
       temp_input_current.iw    = 1.0F;
       temp_input_voltage.vdc    = 24.0F;
       temp_input_voltage.va_max = 24.0F;
 /* Set electric current and voltage before get phase voltage */
       (void) RM_MOTOR_CURRENT_CurrentSet(g_test_motor_current.p_ctrl,
temp_input_current, temp_input_voltage);
 /* Activate the process. */
       (void) RM_MOTOR_CURRENT_Run(g_test_motor_current.p_ctrl);
 /* Get d/q-axis current*/
       (void) RM_MOTOR_CURRENT_CurrentGet(g_test_motor_current.p_ctrl, &f_get_id,
&f_get_iq);
 /* Get the flag of PI control */
       (void) RM_MOTOR_CURRENT_PhaseVolageGet(g_test_motor_current.p_ctrl,
&temp_get_voltage);
 /* Get Output Parameter */
       (void) RM_MOTOR_CURRENT_ParameterGet(g_test_motor_current.p_ctrl,
&test_output);
       (void) RM_MOTOR_CURRENT_ParameterUpdate(g_test_motor_current.p_ctrl,
g_test_motor_current.p_cfg);
    }
 /* Reset the process. */
    (void) RM_MOTOR_CURRENT_Reset(g_test_motor_current.p_ctrl);
 /* Close the module. */
    (void) RM_MOTOR_CURRENT_Close(g_test_motor_current.p_ctrl);
}
```

Enumerations

enum	motor_current_shunt_type_t

Enumeration Type Documentation

◆ motor_current_shunt_type_t

enum motor_current_shunt_type_t

Selection of shunt type

Enumerator	
MOTOR_CURRENT_SHUNT_TYPE_1_SHUNT	Only use U phase current.
MOTOR_CURRENT_SHUNT_TYPE_2_SHUNT	Use U and W phase current.
MOTOR_CURRENT_SHUNT_TYPE_3_SHUNT	Use all phase current.

Function Documentation

◆ RM_MOTOR_CURRENT_Open()

fsp_err_t RM_MOTOR_CURRENT_Open (motor_current_ctrl_t *const *p_ctrl*, motor_current_cfg_t const *const *p_cfg*)

Opens and configures the Motor Current Module. Implements motor_current_api_t::open.

Return values

FSP_SUCCESS	Motor Current successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

◆ RM_MOTOR_CURRENT_Close()

fsp_err_t RM_MOTOR_CURRENT_Close (motor_current_ctrl_t *const *p_ctrl*)

Disables specified Motor Current Module. Implements motor_current_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_CURRENT_Reset()

fsp_err_t RM_MOTOR_CURRENT_Reset (motor_current_ctrl_t *const *p_ctrl*)

Reset variables of Motor Current Module. Implements motor_current_api_t::reset.

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_CURRENT_Run()

fsp_err_t RM_MOTOR_CURRENT_Run (motor_current_ctrl_t *const *p_ctrl*)

Run(Start) the Current Control. Implements motor_current_api_t::run.

Return values

FSP_SUCCESS	Successfully run.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_CURRENT_ParameterSet()

fsp_err_t RM_MOTOR_CURRENT_ParameterSet (motor_current_ctrl_t *const *p_ctrl*, motor_current_input_t const *const *p_st_input*)

Set (Input) Parameter Data. Implements motor_current_api_t::parameterSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input argument error.

◆ RM_MOTOR_CURRENT_CurrentReferenceSet()

fsp_err_t RM_MOTOR_CURRENT_CurrentReferenceSet (motor_current_ctrl_t *const *p_ctrl*, float const *id_reference*, float const *iq_reference*)

Set Current Reference Data. Implements motor_current_api_t::currentReferenceSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_CURRENT_SpeedPhaseSet()

fsp_err_t RM_MOTOR_CURRENT_SpeedPhaseSet (motor_current_ctrl_t *const *p_ctrl*, float const *speed*, float const *phase*)

Set Current Speed & rotor phase Data. Implements motor_current_api_t::speedPhaseSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_CURRENT_CurrentSet()

fsp_err_t RM_MOTOR_CURRENT_CurrentSet (motor_current_ctrl_t *const *p_ctrl*, motor_current_input_current_t const *const *p_st_current*, motor_current_input_voltage_t const *const *p_st_voltage*)

Set d/q-axis Current & Voltage Data. Implements motor_current_api_t::currentSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_CURRENT_ParameterGet()

fsp_err_t RM_MOTOR_CURRENT_ParameterGet (motor_current_ctrl_t *const *p_ctrl*, motor_current_output_t *const *p_st_output*)

Get Output Parameters. Implements motor_current_api_t::parameterGet.

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_CURRENT_CurrentGet()

fsp_err_t RM_MOTOR_CURRENT_CurrentGet (motor_current_ctrl_t *const *p_ctrl*, float *const *p_id*, float *const *p_iq*)

Get d/q-axis Current. Implements motor_current_api_t::currentGet.

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_CURRENT_PhaseVoltageGet()

fsp_err_t RM_MOTOR_CURRENT_PhaseVoltageGet (motor_current_ctrl_t *const *p_ctrl*, motor_current_get_voltage_t *const *p_voltage*)

Gets the set phase voltage. Implements motor_current_api_t::phaseVoltageGet.

Return values

FSP_SUCCESS	Successful data calculation.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_CURRENT_ParameterUpdate()

fsp_err_t RM_MOTOR_CURRENT_ParameterUpdate (motor_current_ctrl_t *const *p_ctrl*, motor_current_cfg_t const *const *p_cfg*)
Update the parameters of Current Control. Implements motor_current_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ rm_motor_current_encoder_cyclic()

void rm_motor_current_encoder_cyclic (motor_current_instance_t const * *p_ctrl*)
(end addtogroup MOTOR_CURRENT)

4.2.102 Motor Driver (rm_motor_driver)

Modules

Functions

fsp_err_t	RM_MOTOR_DRIVER_Open (motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg) Opens and configures the Motor Driver module. Implements motor_driver_api_t::open. More...
fsp_err_t	RM_MOTOR_DRIVER_Close (motor_driver_ctrl_t *const p_ctrl) Disables specified Motor Driver Module. Implements motor_driver_api_t::close. More...
fsp_err_t	RM_MOTOR_DRIVER_Reset (motor_driver_ctrl_t *const p_ctrl) Reset variables of Motor Driver Module. Implements motor_driver_api_t::reset. More...
fsp_err_t	RM_MOTOR_DRIVER_PhaseVoltageSet (motor_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage)

	Set Phase Voltage Data to calculate PWM duty. Implements motor_driver_api_t::phaseVoltageSet. More...
fsp_err_t	RM_MOTOR_DRIVER_CurrentGet (motor_driver_ctrl_t *const p_ctrl, motor_driver_current_get_t *const p_current_get) Get calculated phase Current, Vdc & Va_max data. Implements motor_driver_api_t::currentGet. More...
fsp_err_t	RM_MOTOR_DRIVER_FlagCurrentOffsetGet (motor_driver_ctrl_t *const p_ctrl, uint8_t *const p_flag_offset) Get the flag of finish current offset detection. Implements motor_driver_api_t::flagCurrentOffsetGet. More...
fsp_err_t	RM_MOTOR_DRIVER_CurrentOffsetRestart (motor_driver_ctrl_t *const p_ctrl) Restart the current offset detection. Implements motor_driver_api_t::currentOffsetRestart. More...
fsp_err_t	RM_MOTOR_DRIVER_ParameterUpdate (motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg) Update the parameters of Driver Module. Implements motor_driver_api_t::parameterUpdate. More...

Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor driver Interface.

Overview

The motor driver module is used to translate phase voltage to PWM duty and output PWM, and detect phase current and main line voltage. This module should be called cyclically at included A/D Conversion finish interrupt.

Features

The Motor Driver Module has below features.

- Calculate each phase(U/V/W) PWM duty according to reference and output PWM.
- Detect each phase current and main line voltage.
- Detect and correct A/D offset at phase current channel

Configuration

Build Time Configurations for rm_motor_driver

The following build time configurations are defined in fsp_cfg/rm_motor_driver_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
ADC_B Support	• Enabled • Disabled	Disabled	Select ADC_B module support.

Configurations for Motor > ADC and PWM Modulation (rm_motor_driver)

This module can be added to the Stacks tab via New Stack > Motor > ADC and PWM Modulation (rm_motor_driver).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_driver0	Module name.
General > Shunt type	• 1 shunt • 2 shunt • 3 shunt	2 shunt	Select shunt type
General > Modulation method	• SVPWM • SPWM	SVPWM	Select PWM modulation method
General > PWM Timer Frequency (MHz)	Must be a valid non-negative value.	120	GPT PWM timer frequency
General > PWM Carrier Period (Microseconds)	Must be a valid non-negative value.	50	GPT PWM carrier period
General > Dead Time (Raw Counts)	Must be a valid non-negative value.	240	GPT PWM dead time
General > Current Range (A)	Must be a valid non-negative value.	27.5F	Current range to measure(Maximum input current)
General > Voltage Range (V)	Must be a valid non-negative value.	111.0F	Voltage range to measure(Maximum input Main Line Voltage)
General > Counts for current offset measurement	Must be a valid non-negative value.	500	How many times to measure current offset
General > A/D conversion channel for U Phase current	Manual Entry	0	Specify the A/D channel for U phase current
General > A/D conversion channel for W Phase current	Manual Entry	2	Specify the A/D channel for W phase current

General > A/D conversion channel for Main Line Voltage	Manual Entry	5	Specify the A/D channel for main line voltage
General > A/D conversion channel for V Phase current	Manual Entry	1	Specify the A/D channel for V phase current
General > Adjustment value to current A/D	Must be a valid non-negative value.	20.0	Value to adjust 1shunt A/D double buffer
General > Minimum difference of PWM duty	Manual Entry	300	Minimum difference of PWM duty
General > Adjustment delay of A/D conversion	Manual Entry	240	Adjustment delay of A/D conversion
General > Input Voltage (V)	Must be a valid non-negative value.	24.0F	Input voltage
General > Resolution of A/D conversion	Manual Entry	0xFFF	Resolution of A/D conversion
General > Offset of A/D conversion for current	Manual Entry	0x745	Offset of A/D conversion for current
General > Conversion level of A/D conversion for voltage	Must be a valid non-negative value.	0.66F	Conversion level of A/D conversion for voltage
General > GTIOCA Stop Level	• Pin Level Low • Pin Level High	Pin Level High	Select the behavior of the output pin when the timer is stopped.
General > GTIOCB Stop Level	• Pin Level Low • Pin Level High	Pin Level High	Select the behavior of the output pin when the timer is stopped.
Modulation > Maximum Duty	Must be a valid non-negative value.	0.9375F	Maximum duty of PWM
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called at A/D conversion finish interrupt.

Clock Configuration

Set used clock with included GPT timer.

Pin Configuration

Depend on included GPT Three Phase Module and ADC Module.

Usage Notes

Limitations

Basically no limitation exists.

Examples

Basic Example

This is a basic example of minimal use of the Motor Driver in an application.

```
void motor_driver_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_DRIVER_Open(&g_motor_driver0.p_ctrl, &g_motor_driver0.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at cyclic interrupt (e.g. included GPT PWM Carrier
intterupt).
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Get electric current, main line voltage and maximum voltage component */
       (void) RM_MOTOR_DRIVER_CurrentGet(&g_motor_driver0.p_ctrl, &f_get_iu,
&f_get_iw, &f_get_vdc, &f_get_va_max);
 /* Get the flag of A/D convderted current offset */
       (void) RM_MOTOR_DRIVER_FlagCurrentOffsetGet(&g_motor_driver0.p_ctrl,
&u1_get_flg_offset);
 // Perform current control process here
 /* Set phase voltage */
       (void) RM_MOTOR_DRIVER_PhaseVoltageSet(&g_motor_driver0.p_ctrl, 1.0F, 1.0F,
1.0F);
       (void) RM_MOTOR_DRIVER_ParameterUpdate(&g_motor_driver0.p_ctrl,
&g_motor_driver0.p_cfg);
    }
    (void) RM_MOTOR_DRIVER_Reset(&g_motor_driver0.p_ctrl);
 //

    (void) RM_MOTOR_DRIVER_Close(&g_motor_driver0.p_ctrl);
```

```
}
```

Enumerations

enum	motor_driver_modulation_method_t

Enumeration Type Documentation

◆ motor_driver_modulation_method_t

enum motor_driver_modulation_method_t

Enumerator	
MOTOR_DRIVER_MODULATION_METHOD_SPWM	Sinusoidal pulse-width-modulation.
MOTOR_DRIVER_MODULATION_METHOD_SVPWM	Space vector pulse-width-modulation.

Function Documentation

◆ RM_MOTOR_DRIVER_Open()

fsp_err_t RM_MOTOR_DRIVER_Open (motor_driver_ctrl_t *const *p_ctrl*, motor_driver_cfg_t const *const *p_cfg*)

Opens and configures the Motor Driver module. Implements motor_driver_api_t::open.

Return values

FSP_SUCCESS	Motor Driver successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.

◆ RM_MOTOR_DRIVER_Close()

fsp_err_t RM_MOTOR_DRIVER_Close (motor_driver_ctrl_t *const *p_ctrl*)

Disables specified Motor Driver Module. Implements motor_driver_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_DRIVER_Reset()

fsp_err_t RM_MOTOR_DRIVER_Reset (motor_driver_ctrl_t *const *p_ctrl*)

Reset variables of Motor Driver Module. Implements motor_driver_api_t::reset.

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_DRIVER_PhaseVoltageSet()

fsp_err_t RM_MOTOR_DRIVER_PhaseVoltageSet (motor_driver_ctrl_t *const *p_ctrl*, float const *u_voltage*, float const *v_voltage*, float const *w_voltage*)

Set Phase Voltage Data to calculate PWM duty. Implements motor_driver_api_t::phaseVoltageSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_DRIVER_CurrentGet()

fsp_err_t RM_MOTOR_DRIVER_CurrentGet (motor_driver_ctrl_t *const *p_ctrl*, motor_driver_current_get_t *const *p_current_get*)

Get calculated phase Current, Vdc & Va_max data. Implements motor_driver_api_t::currentGet.

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_DRIVER_FlagCurrentOffsetGet()

fsp_err_t RM_MOTOR_DRIVER_FlagCurrentOffsetGet (motor_driver_ctrl_t *const *p_ctrl*, uint8_t *const *p_flag_offset*)

Get the flag of finish current offset detection. Implements motor_driver_api_t::flagCurrentOffsetGet .

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input parameter error.

◆ RM_MOTOR_DRIVER_CurrentOffsetRestart()

fsp_err_t RM_MOTOR_DRIVER_CurrentOffsetRestart (motor_driver_ctrl_t *const *p_ctrl*)

Restart the current offset detection. Implements motor_driver_api_t::currentOffsetRestart.

Return values

FSP_SUCCESS	Successfully restarted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_DRIVER_ParameterUpdate()

fsp_err_t RM_MOTOR_DRIVER_ParameterUpdate (motor_driver_ctrl_t *const *p_ctrl*, motor_driver_cfg_t const *const *p_cfg*)

Update the parameters of Driver Module. Implements motor_driver_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

4.2.103 Motor encoder vector control (rm_motor_encoder)

Modules

Functions

fsp_err_t	RM_MOTOR_ENCODER_Open (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg)
fsp_err_t	RM_MOTOR_ENCODER_Close (motor_ctrl_t *const p_ctrl) Disables specified Motor Encoder Control block. Implements motor_api_t::close. More...
fsp_err_t	RM_MOTOR_ENCODER_Run (motor_ctrl_t *const p_ctrl) Run Motor (Start motor rotation). Implements motor_api_t::run. More...
fsp_err_t	RM_MOTOR_ENCODER_Stop (motor_ctrl_t *const p_ctrl) Stop Motor (Stop motor rotation). Implements motor_api_t::stop. More...
fsp_err_t	RM_MOTOR_ENCODER_Reset (motor_ctrl_t *const p_ctrl) Reset Motor Encoder Control block. Implements motor_api_t::reset. More...
fsp_err_t	RM_MOTOR_ENCODER_ErrorSet (motor_ctrl_t *const p_ctrl, motor_error_t const error) Set error information. Implements motor_api_t::errorSet. More...
fsp_err_t	RM_MOTOR_ENCODER_SpeedSet (motor_ctrl_t *const p_ctrl, float const speed_rpm) Set speed reference[rpm]. Implements motor_api_t::speedSet. More...
fsp_err_t	RM_MOTOR_ENCODER_PositionSet (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position) Set position reference[degree]. Implements motor_api_t::positionSet. More...
fsp_err_t	RM_MOTOR_ENCODER_StatusGet (motor_ctrl_t *const p_ctrl, uint8_t *const p_status) Get current control status. Implements motor_api_t::statusGet. More...

fsp_err_t	RM_MOTOR_ENCODER_AngleGet (motor_ctrl_t *const p_ctrl, float *const p_angle_rad) Get current rotor angle. Implements motor_api_t::angleGet. More...
fsp_err_t	RM_MOTOR_ENCODER_SpeedGet (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm) Get rotational speed. Implements motor_api_t::speedGet. More...
fsp_err_t	RM_MOTOR_ENCODER_ErrorCheck (motor_ctrl_t *const p_ctrl, uint16_t *const p_error) Check the occurunce of Error. Implements motor_api_t::errorCheck. More...
fsp_err_t	RM_MOTOR_ENCODER_WaitStopFlagGet (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag) Get wait stop flag. Implements motor_api_t::waitStopFlagGet. More...

Detailed Description

Control a SPM motor on RA MCUs. This module implements the Motor encoder vector control (rm_motor_encoder).

Overview

The motor encoder vector control is used to control motor rotation in an application. This module is meant to be used with Surface Permanent Magnet (SPM) motors and allows applications to start or stop motor rotation easily.

Features

The motor encoder module has below features.

- Start/stop motor rotation
- Error detection (over current, over speed, over voltage, low voltage)

Configuration

Build Time Configurations for rm_motor_encoder

The following build time configurations are defined in fsp_cfg/rm_motor_encoder_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP)	Default (BSP)	If selected code for

	• Enabled • Disabled		parameter checking is included in the build.

Configurations for Motor > Motor Encoder Vector Control (rm_motor_encoder)

This module can be added to the Stacks tab via New Stack > Motor > Motor Encoder Vector Control (rm_motor_encoder).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_encoder0	Module name.
General > Limit of over current (A)	Must be a valid non-negative value.	2.0F	Limit of over current.(Detection threshold)
General > Limit of over voltage (V)	Must be a valid non-negative value.	28.0F	Limit of over voltage.(Detection threshold)
General > Limit of over speed (rpm)	Must be a valid non-negative value.	2100.0F	Limit of over speed.(Detection threshold)
General > Limit of low voltage (V)	Must be a valid non-negative value.	18.0F	Limit of low voltage.(Detection threshold)
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called at speed control cyclic interrupt.

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple status transition process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

Examples

Basic Example

This is a basic example of minimal use of the motor encoder module in an application.

```
void motor_encoder_basic_example (void)
```

```
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_ENCODER_Open(g_motor_encoder0.p_ctrl, g_motor_encoder0.p_cfg);
    handle_error(err);
 /* Set speed reference before motor run */
    (void) RM_MOTOR_ENCODER_SpeedSet(g_motor_encoder0.p_ctrl,
RM_MOTOR_ENCODER_TEST_OVER_SPEED_LIMIT);
 /* Set position reference before motor run */
    (void) RM_MOTOR_ENCODER_PositionSet(g_motor_encoder0.p_ctrl, &g_posref_sample1);
 /* Start motor rotation */
    (void) RM_MOTOR_ENCODER_Run(g_motor_encoder0.p_ctrl);
 /* Get current status */
    (void) RM_MOTOR_ENCODER_StatusGet(g_motor_encoder0.p_ctrl, &smpl_status);
 /* Get current rotor angle */
    (void) RM_MOTOR_ENCODER_AngleGet(g_motor_encoder0.p_ctrl, &smpl_angle);
 /* Get current motor speed */
    (void) RM_MOTOR_ENCODER_SpeedGet(g_motor_encoder0.p_ctrl, &smpl_speed);
 /* Check error */
    (void) RM_MOTOR_ENCODER_ErrorCheck(g_motor_encoder0.p_ctrl, &smpl_error);
 /* Stop motor rotation */
    (void) RM_MOTOR_ENCODER_Stop(g_motor_encoder0.p_ctrl);
 /* Stop motor rotation */
    (void) RM_MOTOR_ENCODER_ErrorSet(g_motor_encoder0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
 /* Reset Speed Control */
    (void) RM_MOTOR_ENCODER_Reset(g_motor_encoder0.p_ctrl);
 /* Close Speed Control */
    (void) RM_MOTOR_ENCODER_Close(g_motor_encoder0.p_ctrl);
}
```

Data Structures

struct	motor_encoder_callback_args_t

Enumerations

enum	motor_encoder_ctrl_t
enum	motor_encoder_ctrl_event_t
enum	motor_encoder_callback_event_t

Data Structure Documentation

◆ motor_encoder_callback_args_t

struct motor_encoder_callback_args_t		
Callback function parameter data		
Data Fields		
void const *	p_context	Placeholder for user data.
motor_encoder_callback_event_t	event	

Enumeration Type Documentation

◆ motor_encoder_ctrl_t

enum motor_encoder_ctrl_t	
Enumerator	
MOTOR_ENCODER_CTRL_STOP	Stop mode.
MOTOR_ENCODER_CTRL_RUN	Run mode.
MOTOR_ENCODER_CTRL_ERROR	Error mode.

◆ motor_encoder_ctrl_event_t

enum motor_encoder_ctrl_event_t	
Enumerator	
MOTOR_ENCODER_CTRL_EVENT_STOP	Stop event.
MOTOR_ENCODER_CTRL_EVENT_RUN	Run event.
MOTOR_ENCODER_CTRL_EVENT_ERROR	Error event.
MOTOR_ENCODER_CTRL_EVENT_RESET	Reset event.

◆ motor_encoder_callback_event_t

enum motor_encoder_callback_event_t	
Events that can trigger a callback function	
Enumerator	
MOTOR_ENCODER_CALLBACK_EVENT_SPEED_FORWARD	Event forward speed control.
MOTOR_ENCODER_CALLBACK_EVENT_SPEED_BACKWARD	Event backward speed control.
MOTOR_ENCODER_CALLBACK_EVENT_CURRENT_FORWARD	Event forward current control.
MOTOR_ENCODER_CALLBACK_EVENT_CURRENT_BACKWARD	Event backward current control.

Function Documentation

◆ RM_MOTOR_ENCODER_Open()

fsp_err_t RM_MOTOR_ENCODER_Open (motor_ctrl_t *const *p_ctrl*, motor_cfg_t const *const *p_cfg*)

Configure the MOTOR in register start mode. Implements motor_api_t::open.

This function should only be called once as MOTOR configuration registers can only be written to once so subsequent calls will have no effect.

Example:

```
/* Initializes the module. */
   err = RM_MOTOR_ENCODER_Open(g_motor_encoder0.p_ctrl, g_motor_encoder0.p_cfg);
```

Return values

FSP_SUCCESS	MOTOR successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

Note

◆ RM_MOTOR_ENCODER_Close()

fsp_err_t RM_MOTOR_ENCODER_Close (motor_ctrl_t *const *p_ctrl*)

Disables specified Motor Encoder Control block. Implements motor_api_t::close.

Example:

```
/* Close Speed Control */
   (void) RM_MOTOR_ENCODER_Close(g_motor_encoder0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_ENCODER_Run()

fsp_err_t RM_MOTOR_ENCODER_Run (motor_ctrl_t *const *p_ctrl*)

Run Motor (Start motor rotation). Implements motor_api_t::run.

Example:

```
/* Start motor rotation */
   (void) RM_MOTOR_ENCODER_Run(g_motor_encoder0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_ENCODER_Stop()

fsp_err_t RM_MOTOR_ENCODER_Stop (motor_ctrl_t *const *p_ctrl*)

Stop Motor (Stop motor rotation). Implements motor_api_t::stop.

Example:

```
/* Stop motor rotation */
   (void) RM_MOTOR_ENCODER_Stop(g_motor_encoder0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_ENCODER_Reset()

fsp_err_t RM_MOTOR_ENCODER_Reset (motor_ctrl_t *const *p_ctrl*)

Reset Motor Encoder Control block. Implements motor_api_t::reset.

Example:

```
/* Reset Speed Control */
   (void) RM_MOTOR_ENCODER_Reset(g_motor_encoder0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_ENCODER_ErrorSet()

fsp_err_t RM_MOTOR_ENCODER_ErrorSet (motor_ctrl_t *const *p_ctrl*, motor_error_t const *error*)

Set error information. Implements motor_api_t::errorSet.

Example:

```
/* Stop motor rotation */
    (void) RM_MOTOR_ENCODER_ErrorSet(g_motor_encoder0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_ENCODER_SpeedSet()

fsp_err_t RM_MOTOR_ENCODER_SpeedSet (motor_ctrl_t *const *p_ctrl*, float const *speed_rpm*)

Set speed reference[rpm]. Implements motor_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
    (void) RM_MOTOR_ENCODER_SpeedSet(g_motor_encoder0.p_ctrl,
RM_MOTOR_ENCODER_TEST_OVER_SPEED_LIMIT);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_ENCODER_PositionSet()

fsp_err_t RM_MOTOR_ENCODER_PositionSet (motor_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position*)

Set position reference[degree]. Implements motor_api_t::positionSet.

Example:

```
/* Set position reference before motor run */
    (void) RM_MOTOR_ENCODER_PositionSet(g_motor_encoder0.p_ctrl, &g_posref_sample1);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data set pointer is invalid..

Note

◆ RM_MOTOR_ENCODER_StatusGet()

fsp_err_t RM_MOTOR_ENCODER_StatusGet (motor_ctrl_t *const *p_ctrl*, uint8_t *const *p_status*)

Get current control status. Implements motor_api_t::statusGet.

Example:

```
/* Get current status */
    (void) RM_MOTOR_ENCODER_StatusGet(g_motor_encoder0.p_ctrl, &smpl_status);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_ENCODER_AngleGet()

fsp_err_t RM_MOTOR_ENCODER_AngleGet (motor_ctrl_t *const *p_ctrl*, float *const *p_angle_rad*)

Get current rotor angle. Implements motor_api_t::angleGet.

Example:

```
/* Get current rotor angle */
   (void) RM_MOTOR_ENCODER_AngleGet(g_motor_encoder0.p_ctrl, &smpl_angle);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_ENCODER_SpeedGet()

fsp_err_t RM_MOTOR_ENCODER_SpeedGet (motor_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm*)

Get rotational speed. Implements motor_api_t::speedGet.

Example:

```
/* Get current motor speed */
   (void) RM_MOTOR_ENCODER_SpeedGet(g_motor_encoder0.p_ctrl, &smpl_speed);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_ENCODER_ErrorCheck()

fsp_err_t RM_MOTOR_ENCODER_ErrorCheck (motor_ctrl_t *const *p_ctrl*, uint16_t *const *p_error*)

Check the occurunce of Error. Implements motor_api_t::errorCheck.

Example:

```
/* Check error */
    (void) RM_MOTOR_ENCODER_ErrorCheck(g_motor_encoder0.p_ctrl, &smpl_error);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_ENCODER_WaitStopFlagGet()

fsp_err_t RM_MOTOR_ENCODER_WaitStopFlagGet (motor_ctrl_t *const *p_ctrl*, motor_wait_stop_flag_t *const *p_flag*)

Get wait stop flag. Implements motor_api_t::waitStopFlagGet.

Example:

Return values

FSP_ERR_UNSUPPORTED	Unsupported.

Note

4.2.104 Motor Angle and Speed Estimation (rm_motor_estimate)

Modules

Functions

fsp_err_t	RM_MOTOR_ESTIMATE_Open (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg) Opens and configures the Angle Estimation module. Implements motor_angle_api_t::open. More...

fsp_err_t	RM_MOTOR_ESTIMATE_Close (motor_angle_ctrl_t *const p_ctrl) Disables specified Angle Estimation module. Implements motor_angle_api_t::close. More...
fsp_err_t	RM_MOTOR_ESTIMATE_Reset (motor_angle_ctrl_t *const p_ctrl) Reset variables of Angle Estimation module. Implements motor_angle_api_t::reset. More...
fsp_err_t	RM_MOTOR_ESTIMATE_CurrentSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage) Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet. More...
fsp_err_t	RM_MOTOR_ESTIMATE_SpeedSet (motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed) Set Speed Information. Implements motor_angle_api_t::speedSet. More...
fsp_err_t	RM_MOTOR_ESTIMATE_FlagPiCtrlSet (motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi) Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet. More...
fsp_err_t	RM_MOTOR_ESTIMATE_AngleSpeedGet (motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err) Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet. More...
fsp_err_t	RM_MOTOR_ESTIMATE_EstimatedComponentGet (motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq) Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet. More...
fsp_err_t	RM_MOTOR_ESTIMATE_ParameterUpdate (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg) Update the parameters of Angle&Speed Estimation. Implements motor_angle_api_t::parameterUpdate. More...

fsp_err_t	RM_MOTOR_ESTIMATE_InternalCalculate (motor_angle_ctrl_t *const p_ctrl) Calculate internal parameters. Implements motor_angle_api_t::internalCalculate. More...
fsp_err_t	RM_MOTOR_ESTIMATE_AngleAdjust (motor_angle_ctrl_t *const p_ctrl) Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust. More...
fsp_err_t	RM_MOTOR_ESTIMATE_EncoderCyclic (motor_angle_ctrl_t *const p_ctrl) Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic. More...
fsp_err_t	RM_MOTOR_ESTIMATE_InfoGet (motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info) Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet. More...

Detailed Description

Calculation proccess for the motor control on RA MCUs. This module implements the Motor angle Interface.

Overview

The motor angle and speed estimation module is used to calculate rotor angle and rotational speed in an application. This module should be called cyclically after the A/D conversion of electric current of each phase in an application.

Features

The Motor Angle and Speed Estimation Module has below features.

- Calculate rotor angle [radian].
- Calculate rotational speed [radian/second].

Configuration

Build Time Configurations for rm_motor_estimate

The following build time configurations are defined in fsp_cfg/rm_motor_estimate_cfg.h:

Configuration	Options	Default	Description

Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > Motor Angle (rm_motor_estimate)

This module can be added to the Stacks tab via New Stack > Motor > Motor Angle (rm_motor_estimate).

Configuration	Options	Default	Description
Motor Parameter > Pole pairs	Manual Entry	2	Pole pairs
Motor Parameter > Resistance[ohm]	Manual Entry	8.5F	Resistance
Motor Parameter > Inductance of d-axis[H]	Manual Entry	0.0045F	Inductance of d-axis
Motor Parameter > Inductance of q-axis[H]	Manual Entry	0.0045F	Inductance of q-axis
Motor Parameter > Permanent magnetic flux[Wb]	Manual Entry	0.02159F	Permanent magnetic flux
Motor Parameter > Rotor inertia[kgm^2]	Manual Entry	0.0000028F	Rotor inertia
Name	Name must be a valid C symbol	g_motor_angle0	Module name.
Openloop damping	• Disable • Enable	Enable	Openloop damping functionally enable or disable
Natural frequency of BEMF observer	Manual Entry	1000.0F	Natural frequency of BEMF observer
Damping ratio of BEMF observer	Manual Entry	1.0F	Damping ratio of BEMF observer
Natural frequency of PLL Speed estimate loop	Manual Entry	20.0F	Natural frequency of PLL Speed estimate loop
Damping ratio of PLL Speed estimate loop	Manual Entry	1.0F	Damping ratio of PLL Speed estimate loop
Control period	Manual Entry	0.00005F	Control period

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

Developers should be aware of the following limitations when using the Motor Angle and Speed Estimation:

Examples

Basic Example

This is a basic example of minimal use of the Motor Angle and Speed Estimation in an application.

```
void motor_estimate_basic_example (void)
{
 fsp_err_t             err = FSP_SUCCESS;
 motor_angle_current_t smpl_current:
 motor_angle_voltage_reference_t smpl_voltage;
 /* Initializes the module. */
    err = RM_MOTOR_ESTIMATE_Open(&g_mtr_angle0_ctrl, &g_mtr_angle_set0_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at A/D conversion finish interrupt.
  * This implementation is an example. */
 while (true)
    {
 /* Application work here. */
 /* Set PI Control Flag before get Angle/Speed and Estimated Component */
       (void) RM_MOTOR_ESTIMATE_FlagPiCtrlSet(&g_mtr_angle0_ctrl, 1U);
       smpl_current.id = 1.0F;
       smpl_current.iq = 1.0F;
       smpl_voltage.vd = 10.0F;
       smpl_voltage.vq = 10.0F;
 /* Set Current and Speed data before get Angle/Speed and Estimated Component */
       (void) RM_MOTOR_ESTIMATE_CurrentSet(&g_mtr_angle0_ctrl, smpl_current,
smpl_voltage);
 /* Set Internal Speed Reference & damping speed data before get Angle/Speed and
Estimated Component */
       (void) RM_MOTOR_ESTIMATE_SpeedSet(&g_mtr_angle0_ctrl, 104.27F, 10.0F);
```

```
 /* Get Angle/Speed data */
       (void) RM_MOTOR_ESTIMATE_AngleSpeedGet(&g_mtr_angle0_ctrl, &f_get_angle,
&f_get_speed, &f_get_phase_err);
 /* Get Estimated Component */
       (void) RM_MOTOR_ESTIMATE_EstimatedComponentGet(&g_mtr_angle0_ctrl, &f_get_ed,
&f_get_eq);
    }
}
```

Function Documentation

◆ RM_MOTOR_ESTIMATE_Open()

fsp_err_t RM_MOTOR_ESTIMATE_Open (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg*)

Opens and configures the Angle Estimation module. Implements motor_angle_api_t::open.

Return values

FSP_SUCCESS	MTR_ANGL_EST successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

◆ RM_MOTOR_ESTIMATE_Close()

fsp_err_t RM_MOTOR_ESTIMATE_Close (motor_angle_ctrl_t *const *p_ctrl*)

Disables specified Angle Estimation module. Implements motor_angle_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_Reset()

fsp_err_t RM_MOTOR_ESTIMATE_Reset (motor_angle_ctrl_t *const *p_ctrl*)

Reset variables of Angle Estimation module. Implements motor_angle_api_t::reset.

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_CurrentSet()

fsp_err_t RM_MOTOR_ESTIMATE_CurrentSet (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_current_t *const *p_st_current*, motor_angle_voltage_reference_t *const *p_st_voltage*)

Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet.

Return values

FSP_SUCCESS	Successfully set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_SpeedSet()

fsp_err_t RM_MOTOR_ESTIMATE_SpeedSet (motor_angle_ctrl_t *const *p_ctrl*, float const *speed_ctrl*, float const *damp_speed*)

Set Speed Information. Implements motor_angle_api_t::speedSet.

Return values

FSP_SUCCESS	Successfully set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_FlagPiCtrlSet()

fsp_err_t RM_MOTOR_ESTIMATE_FlagPiCtrlSet (motor_angle_ctrl_t *const *p_ctrl*, uint32_t const *flag_pi*)

Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet.

Return values

FSP_SUCCESS	Successfully set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_AngleSpeedGet()

fsp_err_t RM_MOTOR_ESTIMATE_AngleSpeedGet (motor_angle_ctrl_t *const *p_ctrl*, float *const *p_angle*, float *const *p_speed*, float *const *p_phase_err*)

Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet.

Return values

FSP_SUCCESS	Successful data get.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_EstimatedComponentGet()

fsp_err_t RM_MOTOR_ESTIMATE_EstimatedComponentGet (motor_angle_ctrl_t *const *p_ctrl*, float *const *p_ed*, float *const *p_eq*)

Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet.

Return values

FSP_SUCCESS	Successfully data gotten.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_ParameterUpdate()

fsp_err_t RM_MOTOR_ESTIMATE_ParameterUpdate (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg*)

Update the parameters of Angle&Speed Estimation. Implements motor_angle_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data is update.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_InternalCalculate()

fsp_err_t RM_MOTOR_ESTIMATE_InternalCalculate (motor_angle_ctrl_t *const *p_ctrl*)

Calculate internal parameters. Implements motor_angle_api_t::internalCalculate.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_AngleAdjust()

fsp_err_t RM_MOTOR_ESTIMATE_AngleAdjust (motor_angle_ctrl_t *const *p_ctrl*)

Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_EncoderCyclic()

fsp_err_t RM_MOTOR_ESTIMATE_EncoderCyclic (motor_angle_ctrl_t *const *p_ctrl*)

Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_ESTIMATE_InfoGet()

fsp_err_t RM_MOTOR_ESTIMATE_InfoGet (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_encoder_info_t *const *p_info*)

Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet.

Return values

FSP_ERR_UNSUPPORTED	Unsupported.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

4.2.105 Motor Position (rm_motor_position)

Modules

Functions

fsp_err_t	RM_MOTOR_POSITION_Open (motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg) Opens and configures the Motor Position Module. Implements motor_position_api_t::open. More...
fsp_err_t	RM_MOTOR_POSITION_Close (motor_position_ctrl_t *const p_ctrl) Disables specified Motor Position Module. Implements motor_position_api_t::close. More...
fsp_err_t	RM_MOTOR_POSITION_Reset (motor_position_ctrl_t *const p_ctrl)

	Reset the variables of Motor Position Module. Implements motor_position_api_t::reset. More...
fsp_err_t	RM_MOTOR_POSITION_PositionGet (motor_position_ctrl_t *const p_ctrl, int16_t *const p_position) Get Rotor Position Data [degree]. Implements motor_position_api_t::positionGet. More...
fsp_err_t	RM_MOTOR_POSITION_PositionSet (motor_position_ctrl_t *const p_ctrl, float const position_rad) Set Position Data from Encoder [radian]. Implements motor_position_api_t::positionSet. More...
fsp_err_t	RM_MOTOR_POSITION_PositionReferenceSet (motor_position_ctrl_t *const p_ctrl, int16_t const position_reference_deg) Set Position Reference Data [degree]. Implements motor_position_api_t::positionReferenceSet. More...
fsp_err_t	RM_MOTOR_POSITION_ControlModeSet (motor_position_ctrl_t *const p_ctrl, motor_position_ctrl_mode_t const mode) Set Position Control Mode. Implements motor_position_api_t::controlModeSet. More...
fsp_err_t	RM_MOTOR_POSITION_PositionControl (motor_position_ctrl_t *const p_ctrl) Calculates internal position reference.(Main process of Position Control) Implements motor_position_api_t::positionControl. More...
fsp_err_t	RM_MOTOR_POSITION_IpdSpeedPControl (motor_position_ctrl_t *const p_ctrl, float const ref_speed_rad, float const speed_rad, float *const p_iq_ref) Calculates the q-axis current reference by P control. Implements motor_position_api_t::ipdSpeedPControl. More...
fsp_err_t	RM_MOTOR_POSITION_SpeedReferencePControlGet (motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref) Get Speed Reference by P Control. Implements motor_position_api_t::speedReferencePControlGet. More...
fsp_err_t	RM_MOTOR_POSITION_SpeedReferenceIpdControlGet

	(motor_position_ctrl_t *const p_ctrl, float const max_speed_rad, float *const p_speed_ref) Get Speed Reference by IPD Control. Implements motor_position_api_t::speedReferenceIpdControlGet. More...
fsp_err_t	RM_MOTOR_POSITION_SpeedReferenceFeedforwardGet (motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref) Get Speed Reference by Feedforward. Implements motor_position_api_t::speedReferenceFeedforwardGet. More...
fsp_err_t	RM_MOTOR_POSITION_ParameterUpdate (motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg) Update the parameters of Position Control Calculation. Implements motor_position_api_t::parameterUpdate. More...

Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor position Interface.

Overview

The motor position is used to control the position of motor rotor in an appication. This module should be called cyclically in an application (e.g. in cyclic timer interrupt). This module calculates speed reference with inputted position reference and current rotational speed.

Features

The Motor Position Module has below features.

- Calculate speed reference.

Configuration

Build Time Configurations for rm_motor_position

The following build time configurations are defined in fsp_cfg/rm_motor_position_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > Motor Position Controller (rm_motor_position)

This module can be added to the Stacks tab via New Stack > Motor > Motor Position Controller (rm_motor_position).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_position0	Module name.
General > Position Dead Band	Manual Entry	1U	Ignored Area of Position Control.
General > Position Band Limit	Manual Entry	3U	Ignored Area of Position Control.
General > Speed Feedforward Ratio	Manual Entry	0.8F	Speed Feedforward Ratio.
General > Encoder Counts per one rotation	Manual Entry	1200.0F	Encoder Counts per one rotation.
General > Position Omega	Manual Entry	10.0F	Position Control Omega.
General > Period of Speed Control[sec]	Manual Entry	0.0005F	Period of Speed Control.
IPD > IPD LPF	• Disable • Enable	Disable	IPD LPF process enable or disable
IPD > Position Kp ratio	Manual Entry	0.3F	Position Kp ratio.
IPD > Position Feedforward ratio	Manual Entry	0.0F	Position Feedforward ratio.
IPD > Speed K ratio	Manual Entry	2.0F	Speed K ratio
IPD > Error Limit #1	Manual Entry	10.0F	Error Limitation #1
IPD > Error Limit #2	Manual Entry	0.2F	Error Limitation #2
IPD > LPF Omega	Manual Entry	500.0F	LPF Omega.
IPD > LPF Zeta	Manual Entry	1.0F	LPF Zeta.
Position Profiling > Interval Time	Manual Entry	400U	Interval Time.
Position Profiling > Accel Time	Manual Entry	0.3F	Accel Time.
Position Profiling > Maximum Accel Time	Manual Entry	8117.96F	Maximum Accelaration Time [sec]
Position Profiling > Accelaration Maximum Speed	Manual Entry	2000.0F	Accelaration Maximum Speed.
Position Profiling > Update Step of Timer	Manual Entry	0.0005F	Update Step of Timer.
Motor Parameter > Pole Pair	Manual Entry	7	Pole Pair
Motor Parameter >	Manual Entry	0.453F	Resistance

Resistance[ohm]			
Motor Parameter > Inductance of d-axis[H]	Manual Entry	0.0009447F	Inductance of d-axis
Motor Parameter > Inductance of q-axis[H]	Manual Entry	0.0009447F	Inductance of q-axis
Motor Parameter > Inertia[kgm^2]	Manual Entry	0.006198F	Inertia
Motor Parameter > Motor Inertia[kgm^2]	Manual Entry	0.00000962F	Motor Inertia

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

- Set the period of speed control with non-negative value.
- Set the limit of speed change step with non-negative value.
- Set the maximum speed with non-negative value.

Examples

Basic Example

This is a basic example of using the Motor Position module in an application.

```
void motor_position_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_POSITION_Open(g_motor_position0.p_ctrl, g_motor_position0.p_cfg);
    handle_error(err);
 /* Set working mode */
    RM_MOTOR_POSITION_ModeSet(g_motor_position0.p_ctrl,
MOTOR_POSITION_CTRL_MODE_STEP);
 /* Set position reference */
 RM_MOTOR_POSITION_PositionReferenceSet(g_motor_position0.p_ctrl, 180U);
 /* Basically run this module at cyclic interrupt (e.g. AGT timer).
```

```
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Perform Position Control Process */
 RM_MOTOR_POSITION_PositionControl(g_motor_position0.p_ctrl);
 /* Perform Speed IPD Control Process */
 RM_MOTOR_POSITION_IpdSpeedPControl(g_motor_position0.p_ctrl, 0.0F, 0.0F,
&temp_iq_ref);
 /* Get Position */
 RM_MOTOR_POSITION_PositionGet(g_motor_position0.p_ctrl, &temp_position);
 /* Update parameters */
 RM_MOTOR_POSITION_ParameterUpdate(g_motor_position0.p_ctrl,
&g_motor_position0.p_cfg);
    }
 /* Reset Speed Control */
 RM_MOTOR_POSITION_Reset(g_motor_position0.p_ctrl);
 /* Close Speed Control */
 RM_MOTOR_POSITION_Close(g_motor_position0.p_ctrl);
}
```

Enumerations

enum	motor_position_ipd_lpf_t

Enumeration Type Documentation

◆ motor_position_ipd_lpf_t

enum motor_position_ipd_lpf_t

Enumerator	
MOTOR_POSITION_IPD_LPF_DISABLE	ipd control is disabled
MOTOR_POSITION_IPD_LPF_ENABLE	ipd control is enabled

Function Documentation

◆ RM_MOTOR_POSITION_Open()

fsp_err_t RM_MOTOR_POSITION_Open (motor_position_ctrl_t *const *p_ctrl*, motor_position_cfg_t const *const *p_cfg*)

Opens and configures the Motor Position Module. Implements motor_position_api_t::open.

Return values

FSP_SUCCESS	Motor Position Module successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Set parameter is invalid.

◆ RM_MOTOR_POSITION_Close()

fsp_err_t RM_MOTOR_POSITION_Close (motor_position_ctrl_t *const *p_ctrl*)

Disables specified Motor Position Module. Implements motor_position_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_POSITION_Reset()

fsp_err_t RM_MOTOR_POSITION_Reset (motor_position_ctrl_t *const *p_ctrl*)

Reset the variables of Motor Position Module. Implements motor_position_api_t::reset.

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_POSITION_PositionGet()

fsp_err_t RM_MOTOR_POSITION_PositionGet (motor_position_ctrl_t *const *p_ctrl*, int16_t *const *p_position*)

Get Rotor Position Data [degree]. Implements motor_position_api_t::positionGet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_POSITION_PositionSet()

fsp_err_t RM_MOTOR_POSITION_PositionSet (motor_position_ctrl_t *const *p_ctrl*, float const *position_rad*)

Set Position Data from Encoder [radian]. Implements motor_position_api_t::positionSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_POSITION_PositionReferenceSet()

fsp_err_t RM_MOTOR_POSITION_PositionReferenceSet (motor_position_ctrl_t *const *p_ctrl*, int16_t const *position_reference_deg*)

Set Position Reference Data [degree]. Implements motor_position_api_t::positionReferenceSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_POSITION_ControlModeSet()

fsp_err_t RM_MOTOR_POSITION_ControlModeSet (motor_position_ctrl_t *const *p_ctrl*, motor_position_ctrl_mode_t const *mode*)

Set Position Control Mode. Implements motor_position_api_t::controlModeSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_POSITION_PositionControl()

fsp_err_t RM_MOTOR_POSITION_PositionControl (motor_position_ctrl_t *const *p_ctrl*)

Calculates internal position reference.(Main process of Position Control) Implements motor_position_api_t::positionControl.

Return values

FSP_SUCCESS	Successful data calculation.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_POSITION_IpdSpeedPControl()

fsp_err_t RM_MOTOR_POSITION_IpdSpeedPControl (motor_position_ctrl_t *const *p_ctrl*, float const *ref_speed_rad*, float const *speed_rad*, float *const *p_iq_ref*)

Calculates the q-axis current reference by P control. Implements motor_position_api_t::ipdSpeedPControl.

Return values

FSP_SUCCESS	Successful data calculation.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_POSITION_SpeedReferencePControlGet()

fsp_err_t RM_MOTOR_POSITION_SpeedReferencePControlGet (motor_position_ctrl_t *const *p_ctrl*, float *const *p_speed_ref*)

Get Speed Reference by P Control. Implements motor_position_api_t::speedReferencePControlGet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_POSITION_SpeedReferenceIpdControlGet()

fsp_err_t RM_MOTOR_POSITION_SpeedReferenceIpdControlGet (motor_position_ctrl_t *const *p_ctrl*, float const *max_speed_rad*, float *const *p_speed_ref*)

Get Speed Reference by IPD Control. Implements motor_position_api_t::speedReferenceIpdControlGet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_POSITION_SpeedReferenceFeedforwardGet()

fsp_err_t RM_MOTOR_POSITION_SpeedReferenceFeedforwardGet (motor_position_ctrl_t *const *p_ctrl*, float *const *p_speed_ref*)

Get Speed Reference by Feedforward. Implements motor_position_api_t::speedReferenceFeedforwardGet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_POSITION_ParameterUpdate()

fsp_err_t RM_MOTOR_POSITION_ParameterUpdate (motor_position_ctrl_t *const *p_ctrl*, motor_position_cfg_t const *const *p_cfg*)

Update the parameters of Position Control Calculation. Implements motor_position_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

4.2.106 Motor Angle and Speed Calculation with an Encoder (rm_motor_sense_encoder)

Modules

Functions

fsp_err_t	RM_MOTOR_SENSE_ENCODER_Open (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg) Opens and configures the Angle Encoder module. Implements motor_angle_api_t::open. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_Close (motor_angle_ctrl_t *const p_ctrl) Disables specified Angle Encoder module. Implements motor_angle_api_t::close. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_Reset (motor_angle_ctrl_t *const p_ctrl) Reset variables of Angle Encoder module. Implements motor_angle_api_t::reset. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_InternalCalculate (motor_angle_ctrl_t *const p_ctrl) Calculate internal parameters. Implements motor_angle_api_t::internalCalculate. More...

fsp_err_t	RM_MOTOR_SENSE_ENCODER_AngleSpeedGet (motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err) Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_AngleAdjust (motor_angle_ctrl_t *const p_ctrl) Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_EncoderCyclic (motor_angle_ctrl_t *const p_ctrl) Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_InfoGet (motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info) Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_ParameterUpdate (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg) Update the parameters of Angle&Speed calculation with an encoder. Implements motor_angle_api_t::parameterUpdate. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_CurrentSet (motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage) Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_SpeedSet (motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed) Set Speed Information. Implements motor_angle_api_t::speedSet. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_FlagPiCtrlSet (motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi) Set the flag of PI Control runs. Implements

	motor_angle_api_t::flagPiCtrlSet. More...
fsp_err_t	RM_MOTOR_SENSE_ENCODER_EstimatedComponentGet (motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq) Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet. More...

Detailed Description

Calculation proccess for the motor control on RA MCUs. This module implements the Motor angle Interface.

Overview

The motor angle and speed calculation with an encoder module is used to calculate rotor angle and rotational speed in an application. This module is designed to be used with the motor current module (rm_motor_current).

Features

The motor angle and speed calculation with an encoder module has the features listed below.

- Calculate rotor angle [radian].
- Calculate rotational speed [radian/second].

Configuration

Build Time Configurations for rm_motor_sense_encoder

The following build time configurations are defined in fsp_cfg/rm_motor_sense_encoder_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > Motor Angle (rm_motor_sense_encoder)

This module can be added to the Stacks tab via New Stack > Motor > Motor Angle (rm_motor_sense_encoder).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_sense_encoder0	Module name.
Motor Parameter > Pole pairs	Must be a valid non-negative value.	7	Pole pairs

Motor Parameter > Resistance (ohm)	Must be a valid non-negative value.	0.453F	Resistance
Motor Parameter > Inductance of d-axis (H)	Must be a valid non-negative value.	0.0009447F	Inductance of d-axis
Motor Parameter > Inductance of q-axis (H)	Must be a valid non-negative value.	0.0009447F	Inductance of q-axis
Motor Parameter > Permanent magnetic flux (Wb)	Must be a valid non-negative value.	0.006198F	Permanent magnetic flux
Motor Parameter > Rotor inertia (kgm^2)	Must be a valid non-negative value.	0.00000962F	Rotor inertia
Control Type	• Speed • Position	Position	Select control Type
Period of Current control (kHz)	Must be a valid non-negative value.	20.0F	Period of Current control
Period of Speed control (sec)	Must be a valid non-negative value.	0.0005F	Period of Speed control
PWM Carrier Frequency (kHz)	Must be a valid non-negative value.	20.0F	PWM Carrier Frequency
Decimation of Interrupt	Manual Entry	0U	Decimation of Interrupt
Counts per Rotation	Must be a valid non-negative value.	1200U	Encoder Counts per One Rotation
Counts for Angle Adjust	Must be a valid non-negative value.	512U	Counts for Angle Adjust (as working time)
Zero speed counts	Must be a valid non-negative value.	20000000U	Threshold counts to judge zero speed
Occupancy Time	Must be a valid non-negative value.	0.30F	Occupancy time of carrier interrup
Carrier Time	Must be a valid non-negative value.	0.000013F	Processing time of carrier interrupt
Process Time	Must be a valid non-negative value.	0.000001F	Processing time of encoder interrupt
Highspeed Change Margin (rpm)	Must be a valid non-negative value.	150U	Margin of toggle speed for high speed mode
LPF parameter for Highspeed Filter	Must be a valid non-negative value.	0.1F	Highspeed mode speed LPF parameter
Counts to change speed	Must be a valid non-negative value.	8U	Counts for mode change of position speed calculation

Clock Configuration

Pin Configuration

Usage Notes

Limitations

Developers should be aware of the following limitations when using the motor angle and speed calculation with an encoder: all configurations should be set as positive values.

Examples

Basic Example

This is a basic example of minimal use of the motor angle and speed calculation with an encoder in an application.

```
void motor_sense_encoder_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_SENSE_ENCODER_Open(&g_mtr_angle0_ctrl, &g_mtr_angle_set0_cfg);
    assert(FSP_SUCCESS == err);
 /* Basically run this module at A/D conversion finish interrupt.
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Initialize motor with the encoder */
       (void) RM_MOTOR_SENSE_ENCODER_AngleAdjust(&g_mtr_angle0_ctrl);
 /* Perform cyclic encoder process*/
       (void) RM_MOTOR_SENSE_ENCODER_EncoderCyclic(&g_mtr_angle0_ctrl);
 /* Calculate information with encoder signal input */
       (void) RM_MOTOR_SENSE_ENCODER_InternalCalculate(&g_mtr_angle0_ctrl);
 /* Get angle/speed data */
       (void) RM_MOTOR_SENSE_ENCODER_AngleSpeedGet(&g_mtr_angle0_ctrl, &f_get_angle,
&f_get_speed, &f_get_phase_err);
 /* Get calculated component */
       (void) RM_MOTOR_SENSE_ENCODER_InfoGet(&g_mtr_angle0_ctrl, &temp_info);
    }
 /* Reset the module */
```

```
    (void) RM_MOTOR_SENSE_ENCODER_Reset(&g_mtr_angle0_ctrl);
 /* Close the module */
    (void) RM_MOTOR_SENSE_ENCODER_Close(&g_mtr_angle0_ctrl);
}
```

Enumerations

enum	motor_sense_encoder_loop_t
enum	motor_sense_encoder_mode_t

Enumeration Type Documentation

◆ motor_sense_encoder_loop_t

enum motor_sense_encoder_loop_t

Enumerator	
MOTOR_SENSE_ENCODER_LOOP_SPEED	Speed control mode.
MOTOR_SENSE_ENCODER_LOOP_POSITION	Position control mode.

◆ motor_sense_encoder_mode_t

enum motor_sense_encoder_mode_t

Enumerator	
MOTOR_SENSE_ENCODER_MODE_INIT	Initialize mode (Start status)
MOTOR_SENSE_ENCODER_MODE_BOOT	Boot mode (Angle adjustment status)
MOTOR_SENSE_ENCODER_MODE_DRIVE	Drive mode (Normal work status)

Function Documentation

◆ RM_MOTOR_SENSE_ENCODER_Open()

fsp_err_t RM_MOTOR_SENSE_ENCODER_Open (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg*)

Opens and configures the Angle Encoder module. Implements motor_angle_api_t::open.

Return values

FSP_SUCCESS	Angle Encoder module successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

◆ RM_MOTOR_SENSE_ENCODER_Close()

fsp_err_t RM_MOTOR_SENSE_ENCODER_Close (motor_angle_ctrl_t *const *p_ctrl*)

Disables specified Angle Encoder module. Implements motor_angle_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_Reset()

fsp_err_t RM_MOTOR_SENSE_ENCODER_Reset (motor_angle_ctrl_t *const *p_ctrl*)

Reset variables of Angle Encoder module. Implements motor_angle_api_t::reset.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_InternalCalculate()

fsp_err_t RM_MOTOR_SENSE_ENCODER_InternalCalculate (motor_angle_ctrl_t *const *p_ctrl*)

Calculate internal parameters. Implements motor_angle_api_t::internalCalculate.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_AngleSpeedGet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_AngleSpeedGet (motor_angle_ctrl_t *const *p_ctrl*, float *const *p_angle*, float *const *p_speed*, float *const *p_phase_err*)

Gets the current rotor's angle and rotation speed. Implements motor_angle_api_t::angleSpeedGet.

Return values

FSP_SUCCESS	Successfully data calculated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_AngleAdjust()

fsp_err_t RM_MOTOR_SENSE_ENCODER_AngleAdjust (motor_angle_ctrl_t *const *p_ctrl*)

Angle Adjustment Process. Implements motor_angle_api_t::angleAdjust.

Return values

FSP_SUCCESS	Successfully data calculated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_EncoderCyclic()

fsp_err_t RM_MOTOR_SENSE_ENCODER_EncoderCyclic (motor_angle_ctrl_t *const *p_ctrl*)

Encoder Cyclic Process (Call in cyclic timer). Implements motor_angle_api_t::encoderCyclic.

Return values

FSP_SUCCESS	Successfully data calculated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_InfoGet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_InfoGet (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_encoder_info_t *const *p_info*)

Gets information of Encoder Angle Module. Implements motor_angle_api_t::infoGet.

Return values

FSP_SUCCESS	Successfully data calculated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_ParameterUpdate()

fsp_err_t RM_MOTOR_SENSE_ENCODER_ParameterUpdate (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_cfg_t const *const *p_cfg*)

Update the parameters of Angle&Speed calculation with an encoder. Implements motor_angle_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data is update.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SENSE_ENCODER_CurrentSet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_CurrentSet (motor_angle_ctrl_t *const *p_ctrl*, motor_angle_current_t *const *p_st_current*, motor_angle_voltage_reference_t *const *p_st_voltage*)

Set d/q-axis Current Data & Voltage Reference. Implements motor_angle_api_t::currentSet.

Return values

FSP_ERR_UNSUPPORTED	Motor sense encoder software currentSet is not supported.

◆ RM_MOTOR_SENSE_ENCODER_SpeedSet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_SpeedSet (motor_angle_ctrl_t *const *p_ctrl*, float const *speed_ctrl*, float const *damp_speed*)

Set Speed Information. Implements motor_angle_api_t::speedSet.

Return values

FSP_ERR_UNSUPPORTED	Motor sense encoder software speedSet is not supported.

◆ RM_MOTOR_SENSE_ENCODER_FlagPiCtrlSet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_FlagPiCtrlSet (motor_angle_ctrl_t *const *p_ctrl*, uint32_t const *flag_pi*)

Set the flag of PI Control runs. Implements motor_angle_api_t::flagPiCtrlSet.

Return values

FSP_ERR_UNSUPPORTED	Motor sense encoder software flagPiCtrlSet is not supported.

◆ RM_MOTOR_SENSE_ENCODER_EstimatedComponentGet()

fsp_err_t RM_MOTOR_SENSE_ENCODER_EstimatedComponentGet (motor_angle_ctrl_t *const *p_ctrl*, float *const *p_ed*, float *const *p_eq*)

Gets estimated d/q-axis component. Implements motor_angle_api_t::estimatedComponentGet.

Return values

FSP_ERR_UNSUPPORTED	Motor sense encoder software estimatedComponentGet is not supported.

4.2.107 Motor Sensorless Vector Control (rm_motor_sensorless)

Modules

Functions

fsp_err_t	RM_MOTOR_SENSORLESS_Open (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg)
fsp_err_t	RM_MOTOR_SENSORLESS_Close (motor_ctrl_t *const p_ctrl) Disables specified Motor Sensorless Control block. Implements motor_api_t::close. More...
fsp_err_t	RM_MOTOR_SENSORLESS_Run (motor_ctrl_t *const p_ctrl) Run Motor (Start motor rotation). Implements motor_api_t::run. More...
fsp_err_t	RM_MOTOR_SENSORLESS_Stop (motor_ctrl_t *const p_ctrl) Stop Motor (Stop motor rotation). Implements motor_api_t::stop. More...
fsp_err_t	RM_MOTOR_SENSORLESS_Reset (motor_ctrl_t *const p_ctrl) Reset Motor Sensorless Control block. Implements motor_api_t::reset . More...
fsp_err_t	RM_MOTOR_SENSORLESS_ErrorSet (motor_ctrl_t *const p_ctrl, motor_error_t const error) Set error information. Implements motor_api_t::errorSet. More...

fsp_err_t	RM_MOTOR_SENSORLESS_SpeedSet (motor_ctrl_t *const p_ctrl, float const speed_rpm) Set speed reference[rpm]. Implements motor_api_t::speedSet. More...
fsp_err_t	RM_MOTOR_SENSORLESS_StatusGet (motor_ctrl_t *const p_ctrl, uint8_t *const p_status) Get current control status. Implements motor_api_t::statusGet. More...
fsp_err_t	RM_MOTOR_SENSORLESS_AngleGet (motor_ctrl_t *const p_ctrl, float *const p_angle_rad) Get current rotor angle. Implements motor_api_t::angleGet. More...
fsp_err_t	RM_MOTOR_SENSORLESS_SpeedGet (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm) Get rotational speed. Implements motor_api_t::speedGet. More...
fsp_err_t	RM_MOTOR_SENSORLESS_ErrorCheck (motor_ctrl_t *const p_ctrl, uint16_t *const p_error) Check the occurunce of Error. Implements motor_api_t::errorCheck. More...
fsp_err_t	RM_MOTOR_SENSORLESS_PositionSet (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position) Set position reference. Implements motor_api_t::positionSet. More...
fsp_err_t	RM_MOTOR_SENSORLESS_WaitStopFlagGet (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag) Get wait stop flag. Implements motor_api_t::waitStopFlagGet. More...

Detailed Description

Usual control of a SPM motor on RA MCUs. This module implements the Motor Sensorless Vector Control (rm_motor_sensorless).

Overview

The motor sensorless vector control is used to control a motor rotation in an appication. This module is implemented with using SPM motor. User can start/stop motor rotation simply.

Features

The Motor Sensorless Module has below features.

- Start/Stop a motor rotation
- Error detection (over current, over speed, over voltage, low voltage)

Configuration

Build Time Configurations for rm_motor_sensorless

The following build time configurations are defined in fsp_cfg/rm_motor_sensorless_cfg.h:

Configuration	Options	Default	Description
Parameter checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Configurations for Motor > Motor Sensorless Vector Control (rm_motor_sensorless)

This module can be added to the Stacks tab via New Stack > Motor > Motor Sensorless Vector Control (rm_motor_sensorless).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_sensorless0	Module name.
General > Limit of over current (A)	Manual Entry	0.42F	Limit of over current.(Detection threshold)
General > Limit of over voltage (V)	Manual Entry	28.0F	Limit of over voltage.(Detection threshold)
General > Limit of over speed (rpm)	Manual Entry	3000.0F	Limit of over speed.(Detection threshold)
General > Limit of low voltage (V)	Manual Entry	14.0F	Limit of low voltage.(Detection threshold)
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called at speed control cyclic interrupt.

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple status transition process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

- Set the limit of electric current with non-negative value.
- Set the limit of input voltage with non-negative value.
- Set the limit of rotational speed with non-negative value.

Examples

Basic Example

This is a basic example of minimal use of the Motor Sensorless in an application.

```
void motor_sensorless_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_SENSORLESS_Open(g_motor_sensorless0.p_ctrl,
g_motor_sensorless0.p_cfg);
    assert(FSP_SUCCESS == err);
 /* Set speed reference before motor run */
    (void) RM_MOTOR_SENSORLESS_SpeedSet(g_motor_sensorless0.p_ctrl,
DEF_SENSORLESS_TEST_OVSPD_LIM);
 /* Start motor rotation */
    (void) RM_MOTOR_SENSORLESS_Run(g_motor_sensorless0.p_ctrl);
 /* Get current status */
    (void) RM_MOTOR_SENSORLESS_StatusGet(g_motor_sensorless0.p_ctrl, &smpl_status);
 /* Get current rotor angle */
    (void) RM_MOTOR_SENSORLESS_AngleGet(g_motor_sensorless0.p_ctrl, &smpl_angle);
 /* Get current motor speed */
    (void) RM_MOTOR_SENSORLESS_SpeedGet(g_motor_sensorless0.p_ctrl, &smpl_speed);
 /* Check error */
    (void) RM_MOTOR_SENSORLESS_ErrorCheck(g_motor_sensorless0.p_ctrl, &smpl_error);
 /* Stop motor rotation */
    (void) RM_MOTOR_SENSORLESS_Stop(g_motor_sensorless0.p_ctrl);
 /* Stop motor rotation */
```

```
    (void) RM_MOTOR_SENSORLESS_ErrorSet(g_motor_sensorless0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
 /* Reset Speed Control */
    (void) RM_MOTOR_SENSORLESS_Reset(g_motor_sensorless0.p_ctrl);
 /* Close Speed Control */
    (void) RM_MOTOR_SENSORLESS_Close(g_motor_sensorless0.p_ctrl);
}
```

Data Structures

struct	motor_sensorless_callback_args_t

Enumerations

enum	motor_sensorless_callback_event_t

Data Structure Documentation

◆ motor_sensorless_callback_args_t

struct motor_sensorless_callback_args_t

Callback function parameter data

Data Fields		
void const *	p_context	Placeholder for user data.
motor_sensorless_callback_event_t	event	

Enumeration Type Documentation

◆ motor_sensorless_callback_event_t

enum motor_sensorless_callback_event_t

Events that can trigger a callback function

Enumerator	
MOTOR_SENSORLESS_CALLBACK_EVENT_SPEED_FORWARD	Event forward Speed Control.
MOTOR_SENSORLESS_CALLBACK_EVENT_SPEED_BACKWARD	Event backward Speed Control.
MOTOR_SENSORLESS_CALLBACK_EVENT_CURRENT_FORWARD	Event forward Current Control.
MOTOR_SENSORLESS_CALLBACK_EVENT_CURRENT_BACKWARD	Event backward Current Control.

Function Documentation

◆ RM_MOTOR_SENSORLESS_Open()

fsp_err_t RM_MOTOR_SENSORLESS_Open (motor_ctrl_t *const *p_ctrl*, motor_cfg_t const *const *p_cfg*)

Configure the MOTOR in register start mode. Implements motor_api_t::open.

This function should only be called once as MOTOR configuration registers can only be written to once so subsequent calls will have no effect.

Example:

```
/* Initializes the module. */
    err = RM_MOTOR_SENSORLESS_Open(g_motor_sensorless0.p_ctrl,
g_motor_sensorless0.p_cfg);
```

Return values

FSP_SUCCESS	MOTOR successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

Note

◆ RM_MOTOR_SENSORLESS_Close()

fsp_err_t RM_MOTOR_SENSORLESS_Close (motor_ctrl_t *const *p_ctrl*)

Disables specified Motor Sensorless Control block. Implements motor_api_t::close.

Example:

```
/* Close Speed Control */
    (void) RM_MOTOR_SENSORLESS_Close(g_motor_sensorless0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_SENSORLESS_Run()

fsp_err_t RM_MOTOR_SENSORLESS_Run (motor_ctrl_t *const *p_ctrl*)

Run Motor (Start motor rotation). Implements motor_api_t::run.

Example:

```
/* Start motor rotation */
    (void) RM_MOTOR_SENSORLESS_Run(g_motor_sensorless0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_SENSORLESS_Stop()

fsp_err_t RM_MOTOR_SENSORLESS_Stop (motor_ctrl_t *const *p_ctrl*)

Stop Motor (Stop motor rotation). Implements motor_api_t::stop.

Example:

```
/* Stop motor rotation */
    (void) RM_MOTOR_SENSORLESS_Stop(g_motor_sensorless0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_SENSORLESS_Reset()

fsp_err_t RM_MOTOR_SENSORLESS_Reset (motor_ctrl_t *const *p_ctrl*)

Reset Motor Sensorless Control block. Implements motor_api_t::reset.

Example:

```
/* Reset Speed Control */
    (void) RM_MOTOR_SENSORLESS_Reset(g_motor_sensorless0.p_ctrl);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_SENSORLESS_ErrorSet()

fsp_err_t RM_MOTOR_SENSORLESS_ErrorSet (motor_ctrl_t *const *p_ctrl*, motor_error_t const *error*)

Set error information. Implements motor_api_t::errorSet.

Example:

```
/* Stop motor rotation */
    (void) RM_MOTOR_SENSORLESS_ErrorSet(g_motor_sensorless0.p_ctrl,
MOTOR_ERROR_OVER_CURRENT_HW);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_SENSORLESS_SpeedSet()

fsp_err_t RM_MOTOR_SENSORLESS_SpeedSet (motor_ctrl_t *const *p_ctrl*, float const *speed_rpm*)

Set speed reference[rpm]. Implements motor_api_t::speedSet.

Example:

```
/* Set speed reference before motor run */
    (void) RM_MOTOR_SENSORLESS_SpeedSet(g_motor_sensorless0.p_ctrl,
DEF_SENSORLESS_TEST_OVSPD_LIM);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

Note

◆ RM_MOTOR_SENSORLESS_StatusGet()

fsp_err_t RM_MOTOR_SENSORLESS_StatusGet (motor_ctrl_t *const *p_ctrl*, uint8_t *const *p_status*)

Get current control status. Implements motor_api_t::statusGet.

Example:

```
/* Get current status */
    (void) RM_MOTOR_SENSORLESS_StatusGet(g_motor_sensorless0.p_ctrl, &smpl_status);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_SENSORLESS_AngleGet()

fsp_err_t RM_MOTOR_SENSORLESS_AngleGet (motor_ctrl_t *const *p_ctrl*, float *const *p_angle_rad*)

Get current rotor angle. Implements motor_api_t::angleGet.

Example:

```
/* Get current rotor angle */
    (void) RM_MOTOR_SENSORLESS_AngleGet(g_motor_sensorless0.p_ctrl, &smpl_angle);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_SENSORLESS_SpeedGet()

fsp_err_t RM_MOTOR_SENSORLESS_SpeedGet (motor_ctrl_t *const *p_ctrl*, float *const *p_speed_rpm*)

Get rotational speed. Implements motor_api_t::speedGet.

Example:

```
/* Get current motor speed */
    (void) RM_MOTOR_SENSORLESS_SpeedGet(g_motor_sensorless0.p_ctrl, &smpl_speed);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_SENSORLESS_ErrorCheck()

fsp_err_t RM_MOTOR_SENSORLESS_ErrorCheck (motor_ctrl_t *const *p_ctrl*, uint16_t *const *p_error*)

Check the occurunce of Error. Implements motor_api_t::errorCheck.

Example:

```
/* Check error */
    (void) RM_MOTOR_SENSORLESS_ErrorCheck(g_motor_sensorless0.p_ctrl, &smpl_error);
```

Return values

FSP_SUCCESS	Successfully resetted.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Data received pointer is invalid..

Note

◆ RM_MOTOR_SENSORLESS_PositionSet()

fsp_err_t RM_MOTOR_SENSORLESS_PositionSet (motor_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position*)

Set position reference. Implements motor_api_t::positionSet.

Example:

Return values

FSP_ERR_UNSUPPORTED	Unsupported.

Note

◆ RM_MOTOR_SENSORLESS_WaitStopFlagGet()

fsp_err_t RM_MOTOR_SENSORLESS_WaitStopFlagGet (motor_ctrl_t *const *p_ctrl*, motor_wait_stop_flag_t *const *p_flag*)

Get wait stop flag. Implements motor_api_t::waitStopFlagGet.

Example:

Return values

FSP_ERR_UNSUPPORTED	Unsupported.

Note

4.2.108 Motor Speed (rm_motor_speed)

Modules

Functions

fsp_err_t	RM_MOTOR_SPEED_Open (motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg) Opens and configures the Motor Speed Module. Implements motor_speed_api_t::open. More...
fsp_err_t	RM_MOTOR_SPEED_Close (motor_speed_ctrl_t *const p_ctrl) Disables specified Motor Speed Module. Implements motor_speed_api_t::close. More...
fsp_err_t	RM_MOTOR_SPEED_Reset (motor_speed_ctrl_t *const p_ctrl) Reset the variables of Motor Speed Module. Implements motor_speed_api_t::reset. More...
fsp_err_t	RM_MOTOR_SPEED_Run (motor_speed_ctrl_t *const p_ctrl) Run(Start) the Motor Speed Control. Implements motor_speed_api_t::run. More...
fsp_err_t	RM_MOTOR_SPEED_SpeedReferenceSet (motor_speed_ctrl_t *const p_ctrl, float const speed_reference_rpm) Set Speed Reference Data. Implements motor_speed_api_t::speedReferenceSet. More...

fsp_err_t	RM_MOTOR_SPEED_PositionReferenceSet (motor_speed_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position_data) Set Position Reference Data. Implements motor_speed_api_t::positionReferenceSet. More...
fsp_err_t	RM_MOTOR_SPEED_ParameterSet (motor_speed_ctrl_t *const p_ctrl, motor_speed_input_t const *const p_st_input) Set Input parameters. Implements motor_speed_api_t::parameterSet. More...
fsp_err_t	RM_MOTOR_SPEED_SpeedControl (motor_speed_ctrl_t *const p_ctrl) Calculates the d/q-axis current reference.(Main process of Speed Control) Implements motor_speed_api_t::speedControl. More...
fsp_err_t	RM_MOTOR_SPEED_ParameterGet (motor_speed_ctrl_t *const p_ctrl, motor_speed_output_t *const p_st_output) Get Speed Control Parameters. Implements motor_speed_api_t::parameterGet. More...
fsp_err_t	RM_MOTOR_SPEED_ParameterUpdate (motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg) Update the parameters of Speed Control Calculation. Implements motor_speed_api_t::parameterUpdate. More...

Detailed Description

Calculation process for the motor control on RA MCUs. This module implements the Motor speed Interface.

Overview

The motor speed is used to control the speed of motor rotation in an appication. This module should be called cyclically in an application (e.g. in cyclic timer interrupt). This module caliculates d/q-axis current reference with input speed reference, current rotational speed, and d/q-axis current.

Features

The motor speed module has below features.

- Calculate d/q-axis electric current reference.
- Flux weakening process at high speed rotation.
- Open loop damping control when using sensorless type.
- Low pass filter of input rotational speed.

- Speed observer function when using encoder type.

Configuration

Build Time Configurations for rm_motor_speed

The following build time configurations are defined in fsp_cfg/rm_motor_speed_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Postion Support	• Enabled • Disabled	Disabled	Enable position algorithm support.

Configurations for Motor > Motor Speed Controller (rm_motor_speed)

This module can be added to the Stacks tab via New Stack > Motor > Motor Speed Controller (rm_motor_speed).

Configuration	Options	Default	Description
General > Name	Name must be a valid C symbol	g_motor_speed0	Module name.
General > Speed control period (sec)	Manual Entry	0.0005F	Period of speed control function.
General > Step of speed climbing (rpm)	Manual Entry	0.5F	Step of speed change at start of open-loop.
General > Maximum rotational speed (rpm)	Manual Entry	2650	Maximum rotational speed (Limit speed).
General > Speed LPF omega	Manual Entry	10.0F	Design parameter for speed LPF.
General > Speed at Id climbing (rpm)	Manual Entry	500	From this speed d-axis current is controlled climbing.
General > Limit of q-axis current (A)	Manual Entry	0.42F	Limit of q-axis current.
General > Step of speed feedback at open-loop	Manual Entry	0.20F	Step of speed feedback at open-loop.
General > Open-loop damping	• Disable • Enable	Enable	Select enable/disable Open-loop damping control.
General > Flux weakening	• Disable • Enable	Disable	Select enable/disable flux weakening control.
General > Torque compensation for	• Disable • Enable	Enable	Select enable/disable torque compensation

sensorless transition			for sensorless transition.
General > Speed observer	• Disable • Enable	Enable	Select enable/disable Speed observer process.
General > Control method	• PID • IPD	PID	Select the control method [PID or IPD].
General > Control type	• Sensoreless • Encoder	Sensoreless	
Open-Loop > Step of d-axis current climbing	Manual Entry	0.3F	Step of d-axis current climbing
Open-Loop > Step of d-axis current descending	Manual Entry	0.3F	Step of d-axis current descending
Open-Loop > Step of q-axis current descending ratio	Manual Entry	1.0F	Step of q-axis current descending ratio
Open-Loop > Reference of d-axis current	Manual Entry	0.3F	Reference of d-axis current
Open-Loop > Threshold of speed control descending	Manual Entry	600	When rotational speed reaches this speed, d-axis current is controlled descending.
Open-Loop > Threshold of speed control climbing	Manual Entry	500	Until rotational speed reaches this speed, d-axis current is controlled climbing.
Open-Loop > Period between open-loop to BEMF (sec)	Manual Entry	0.025F	Margin time between open-loop control changes to BEMF PI control.
Open-Loop > Phase error(degree) to decide sensor-less switch timing	Manual Entry	10	Phase error(degree) to decide sensor-less switch timing.
Design parameter > Speed PI loop omega	Manual Entry	5.0F	Speed PI loop omega
Design parameter > Speed PI loop zeta	Manual Entry	1.0F	Speed PI loop zeta
Design parameter > Estimated d-axis HPF omega	Manual Entry	2.5F	HPF cutoff frequency for ed (Hz)
Design parameter > Open-loop damping	Manual Entry	1.0F	Damping ratio of open-loop damping control

zeta			
Design parameter > Cutoff frequency of phase error LPF	Manual Entry	10.0F	Cutoff frequency of phase error LPF
Design parameter > Speed observer omega	Manual Entry	200.0F	Speed observer loop omega
Design parameter > Speed observer zeta	Manual Entry	1.0F	Speed observer loop zeta
Motor Parameter > Pole pairs	Manual Entry	2	Pole pairs
Motor Parameter > Resistance (ohm)	Manual Entry	8.5F	Resistance
Motor Parameter > Inductance of d-axis (H)	Manual Entry	0.0045F	Inductance of d-axis
Motor Parameter > Inductance of q-axis (H)	Manual Entry	0.0045F	Inductance of q-axis
Motor Parameter > Permanent magnetic flux (Wb)	Manual Entry	0.02159F	Permanent magnetic flux
Motor Parameter > Rotor inertia (kgm^2)	Manual Entry	0.0000028F	Rotor inertia
Interrupts > Callback	Name must be a valid C symbol	NULL	A user callback function. If this callback function is provided, it is called at timer interrupt.
Interrupts > Input data	Name must be a valid C symbol	NULL	Structure for Speed control Input. If you set this content, Speed Control function read these data automatically. (No need to use Set API.)
Interrupts > Output data	Name must be a valid C symbol	NULL	Structure for Speed control Output. If you set this content, Speed Control function write need data automatically. (No need to use Get API.)

Clock Configuration

This module doesn't depend on clock setting, because this module is a simple calculation process.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Limitations

- Set the period of speed control with none-negative value.
- Set the limit of speed change step with none-negative value.
- Set the maximum speed with none-negative value.

Examples

Basic Example

This is a basic example of minimal use of the motor speed in an application.

```
void motor_speed_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
 /* Initializes the module. */
    err = RM_MOTOR_SPEED_Open(g_motor_speed0.p_ctrl, g_motor_speed0.p_cfg);
    handle_error(err);
 /* Set speed reference before get current reference */
    (void) RM_MOTOR_SPEED_SpeedReferenceSet(g_motor_speed0.p_ctrl, 104.72F);
 /* Set position reference before get current reference
  * (Basically Exclusive to SpeedReferenceSet. This is only sample,) */
    (void) RM_MOTOR_SPEED_PositionReferenceSet(g_motor_speed0.p_ctrl,
&g_posref_sample);
 /* Basically run this module at cyclic interrupt (e.g. AGT timer).
  * This implementation is an example. */
 // while (true)
    {
 /* Application work here. */
 /* Set input parameter data before get current reference */
       (void) RM_MOTOR_SPEED_ParameterSet(g_motor_speed0.p_ctrl,
&g_test_speed_input);
 /* Activate Speed Process */
       (void) RM_MOTOR_SPEED_Run(g_motor_speed0.p_ctrl);
```

```
 /* Perform Speed Control Process */
       (void) RM_MOTOR_SPEED_SpeedControl(g_motor_speed0.p_ctrl);
 /* Get output parameters */
       (void) RM_MOTOR_SPEED_ParameterGet(g_motor_speed0.p_ctrl,
&g_test_speed_output);
 //

 /* Update parameters */
       (void) RM_MOTOR_SPEED_ParameterUpdate(g_motor_speed0.p_ctrl,
&g_motor_speed0.p_cfg);
    }
 /* Reset Speed Control */
    (void) RM_MOTOR_SPEED_Reset(g_motor_speed0.p_ctrl);
 /* Close Speed Control */
    (void) RM_MOTOR_SPEED_Close(g_motor_speed0.p_ctrl);
}
```

Enumerations

enum	motor_speed_control_type_t
enum	motor_speed_openloop_damping_t
enum	motor_speed_flux_weaken_t
enum	motor_speed_less_switch_t
enum	motor_speed_observer_switch_t
enum	motor_speed_ctrl_status_t

Enumeration Type Documentation

◆ motor_speed_control_type_t

enum motor_speed_control_type_t

Enumerator	
MOTOR_SPEED_CONTROL_TYPE_SENSORLESS	Sensorless type.
MOTOR_SPEED_CONTROL_TYPE_ENCODER	Encoder type.

◆ motor_speed_openloop_damping_t

enum motor_speed_openloop_damping_t	
Enumerator	
MOTOR_SPEED_OPENLOOP_DAMPING_DISABLE	Disable openloop damping.
MOTOR_SPEED_OPENLOOP_DAMPING_ENABLE	Enable openloop damping.

◆ motor_speed_flux_weaken_t

enum motor_speed_flux_weaken_t	
Enumerator	
MOTOR_SPEED_FLUX_WEAKEN_DISABLE	Disable flux-weakening control.
MOTOR_SPEED_FLUX_WEAKEN_ENABLE	Enable flux-weakening control.

◆ motor_speed_less_switch_t

enum motor_speed_less_switch_t	
Enumerator	
MOTOR_SPEED_LESS_SWITCH_DISABLE	Disable soft switching between open-loop mode and normal FOC mode.
MOTOR_SPEED_LESS_SWITCH_ENABLE	Enable soft switching between open-loop mode and normal FOC mode.

◆ motor_speed_observer_switch_t

enum motor_speed_observer_switch_t	
Enumerator	
MOTOR_SPEED_OBSERVER_SWITCH_DISABLE	Disable speed observer.
MOTOR_SPEED_OBSERVER_SWITCH_ENABLE	Enable speed observer.

◆ motor_speed_ctrl_status_t

enum motor_speed_ctrl_status_t

Enumerator	
MOTOR_SPEED_CTRL_STATUS_INIT	Speed control status is INIT.
MOTOR_SPEED_CTRL_STATUS_BOOT	Speed control status is BOOT.
MOTOR_SPEED_CTRL_STATUS_RUN	Speed control status is RUN.

Function Documentation

◆ RM_MOTOR_SPEED_Open()

fsp_err_t RM_MOTOR_SPEED_Open (motor_speed_ctrl_t *const *p_ctrl*, motor_speed_cfg_t const *const *p_cfg*)

Opens and configures the Motor Speed Module. Implements motor_speed_api_t::open.

Return values

FSP_SUCCESS	Motor Speed Module successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

◆ RM_MOTOR_SPEED_Close()

fsp_err_t RM_MOTOR_SPEED_Close (motor_speed_ctrl_t *const *p_ctrl*)

Disables specified Motor Speed Module. Implements motor_speed_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SPEED_Reset()

fsp_err_t RM_MOTOR_SPEED_Reset (motor_speed_ctrl_t *const *p_ctrl*)

Reset the variables of Motor Speed Module. Implements motor_speed_api_t::reset.

Return values

FSP_SUCCESS	Successfully reset.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SPEED_Run()

fsp_err_t RM_MOTOR_SPEED_Run (motor_speed_ctrl_t *const *p_ctrl*)

Run(Start) the Motor Speed Control. Implements motor_speed_api_t::run.

Return values

FSP_SUCCESS	Successfully start.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SPEED_SpeedReferenceSet()

fsp_err_t RM_MOTOR_SPEED_SpeedReferenceSet (motor_speed_ctrl_t *const *p_ctrl*, float const *speed_reference_rpm*)

Set Speed Reference Data. Implements motor_speed_api_t::speedReferenceSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SPEED_PositionReferenceSet()

fsp_err_t RM_MOTOR_SPEED_PositionReferenceSet (motor_speed_ctrl_t *const *p_ctrl*, motor_speed_position_data_t const *const *p_position_data*)

Set Position Reference Data. Implements motor_speed_api_t::positionReferenceSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Input structure pointer is NULL.

◆ RM_MOTOR_SPEED_ParameterSet()

fsp_err_t RM_MOTOR_SPEED_ParameterSet (motor_speed_ctrl_t *const *p_ctrl*, motor_speed_input_t const *const *p_st_input*)

Set Input parameters. Implements motor_speed_api_t::parameterSet.

Return values

FSP_SUCCESS	Successfully data is set.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_SPEED_SpeedControl()

fsp_err_t RM_MOTOR_SPEED_SpeedControl (motor_speed_ctrl_t *const *p_ctrl*)

Calculates the d/q-axis current reference.(Main process of Speed Control) Implements motor_speed_api_t::speedControl.

Return values

FSP_SUCCESS	Successful data calculation.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_MOTOR_SPEED_ParameterGet()

fsp_err_t RM_MOTOR_SPEED_ParameterGet (motor_speed_ctrl_t *const *p_ctrl*, motor_speed_output_t *const *p_st_output*)

Get Speed Control Parameters. Implements motor_speed_api_t::parameterGet.

Return values

FSP_SUCCESS	Successfully the flag is gotten.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Output pointer is NULL.

◆ RM_MOTOR_SPEED_ParameterUpdate()

fsp_err_t RM_MOTOR_SPEED_ParameterUpdate (motor_speed_ctrl_t *const *p_ctrl*, motor_speed_cfg_t const *const *p_cfg*)

Update the parameters of Speed Control Calculation. Implements motor_speed_api_t::parameterUpdate.

Return values

FSP_SUCCESS	Successfully data was updated.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

4.2.109 Azure RTOS NetX Secure Crypto Hardware Acceleration (rm_netx_secure_crypto)

Modules

Detailed Description

Hardware acceleration for the Netx Crypto implementation of the Microsoft Azure RTOS NetX Crypto API.

Overview

Please refer to the NetXDuo - NetX Crypto documentation for further details.

HW Overview

Crypto Peripheral version	Devices
SCE9	RA6M4, RA4M3, RA4M2, RA6M5
SCE7	RA6M3, RA6M2, RA6M1, RA6T1
SCE5	RA4W1, RA4M1
AES Engine	RA2A1, RA2E1, RA2E2, RA2L1

Note

NetX Crypto hardware acceleration is unsupported on 'SCE5' and 'AES Engine' crypto peripherals listed above.

Features

This module provides SCE9 hardware support for the following NetX Crypto operations

- SHA256 calculation
- SHA224 calculation
- MAC Operations
  - HMAC with SHA224
  - HMAC with SHA256
- AES
  - Keybits - 128, 192, 256
    - Encryption and Decryption.
    - Chaining Modes: CBC, CTR, GCM mode

AES Chaining Mode	HW Acceleration
CBC	Fully accelerated
CTR	Fully accelerated
GCM	Encrypt - Fully accelerated; Decrypt - Only GHASH and block cipher unit is HW accelerated
CCM	Only block cipher unit is HW accelerated

- Random number generation
- ECC
  - ECDSA: Supported Curves - SECP384R1, SECP256R1, SECP224R1.
  - ECDH: Supported Curves - SECP384R1, SECP256R1, SECP224R1.
- RSA
  - Signature Generation - RSA 2048 (Plain or Wrapped private) key. (This can be used for decryption)
  - Signature Verification - RSA 2048, RSA 3072 and RSA 4096 keys. (This can be used for encryption)
  - Supported encoding scheme: PKCS1V15 This module provides SCE7 hardware support for the following NetX Crypto operations
- AES
  - Keybits - 128, 256
    - Encryption and Decryption.
    - Chaining Modes: CBC, CTR, GCM mode

AES Chaining Mode	HW Acceleration

CBC	Fully accelerated
CTR	Fully accelerated
GCM	Only block cipher unit is HW accelerated
CCM	Only block cipher unit is HW accelerated

Configuration

Build Time Configurations for rm_netx_secure_crypto_sw_port

The following build time configurations are defined in fsp_cfg/middleware/rm_netx_secure_crypto_cfg.h:

Configuration	Options	Default	Description
Standalone Usage	• Use Standalone Crypto Only • Use with TLS	Use Standalone Crypto Only	Defines NX_CRYPTO_STANDALONE_ENABLE.
Maximum RSA Modulus size (bits)	• 1024 • 2048 • 3072 • 4096	4096	

Build Time Configurations for rm_netx_secure_crypto

The following build time configurations are defined in fsp_cfg/middleware/rm_netx_secure_crypto_cfg.h:

Configuration	Options	Default	Description
Hardware Acceleration > Hash > SHA256/224	MCU Specific Options		Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_SHA256_ALT.
Hardware Acceleration > Cipher > AES	MCU Specific Options		Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_AES_ALT
Hardware Acceleration > Public Key Cryptography (PKC) > ECC > ECC	MCU Specific Options		Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_ECC_ALT
Hardware Acceleration > Public Key Cryptography (PKC) > ECC > ECDSA Scratch Buffer Size (Bytes)	Value must be an integer	3016	Sets value of NX_CRYPTO_ECDSA_SCRATCH_BUFFER_SIZE
Hardware Acceleration > Public Key	Value must be an integer	2464	Sets value of NX_CRYPTO_ECDH_SCRATCH_BU

Cryptography (PKC) > ECC > ECDH Scratch Buffer Size (Bytes)			FFER_SIZE
Hardware Acceleration > Public Key Cryptography (PKC) > RSA > RSA	MCU Specific Options		Enables/Disables RSA HW support
Hardware Acceleration > Public Key Cryptography (PKC) > RSA > RSA 2048 (HW)	MCU Specific Options		Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_2048_ALT to allow HW support
Hardware Acceleration > Public Key Cryptography (PKC) > RSA > RSA 3072 Verify/Encryption (HW)	MCU Specific Options		Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_3072_ALT to allow HW support
Hardware Acceleration > Public Key Cryptography (PKC) > RSA > RSA 4096 Verify/Encryption (HW)	MCU Specific Options		Enables RSA NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_4096_ALT to allow HW support
Hardware Acceleration > Public Key Cryptography (PKC) > RSA > RSA Scratch Buffer Size (Bytes)	MCU Specific Options		Sets value of NX_CRYPTO_RSA_SCRATCH_BUFFER_SIZE
Hardware Acceleration > TRNG	• Enabled • Disabled	Enabled	Enables NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_TRNG_ALT.
Standalone Usage	• Use Standalone Crypto Only • Use with TLS	Use Standalone Crypto Only	Defines NX_CRYPTO_STANDALONE_ENABLE.

Random Number Generator Configuration

To enable hardware acceleration for the TRNG, the macro NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_TRNG_ALT must be defined in the configuration file. By default TRNG is enabled which can be disabled using the RA Configuration editor.

Once enabled 'rand' function will be mapped to HW TRNG; the 'srand' function is not supported, any calls to this function will have no effect. Functionality to re-seed the HW TRNG is not supported by the existing implementation.

If disabled, both 'rand' and 'srand' will be mapped to the C Standard Library. This would require setting up the heap as 'rand' implementation calls 'malloc'.

SHA256 Configuration

To enable hardware acceleration for the SHA256/224 calculation, the macro NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_SHA256_ALT must be defined in the configuration file. By default SHA256 is enabled which can be disabled using the RA Configuration editor.

AES Configuration

To enable hardware acceleration for the AES128/192/256 operation, the macro NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_AES_ALT must be defined in the configuration file. By default AES is enabled which can be disabled using the RA Configuration editor.

ECC Configuration

To enable hardware acceleration for the ECDSA and ECDH for curves SECP384R1, SECP256R1 and SECP224R1, the macro NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_ECC_ALT must be defined in the configuration file. By default ECC operations are enabled which can be disabled using the RA Configuration editor.

RSA Configuration

To enable hardware acceleration for the RSA Encrypt/Decrypt (or Sign/Verify) operation(s), the macro(s) below must be must be defined in the configuration file:

Configuration Macro	Feature / Operation
NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_2048_ALT	Signature Generation / Signature Verification (Encryption / Decryption)
NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_3072_ALT	Signature Verification (Encryption Only)
NETX_SECURE_CRYPTO_NX_CRYPTO_METHODS_RSA_4096_ALT	Signature Verification (Encryption Only)

By default RSA 2048 is enabled which can be disabled using the RA Configuration editor.

RSA software implementation is completely disabled when any of the above macros are enabled.

Usage Notes

Memory Alignment

Use 32bit aligned buffer pointers as arguments to APIs for best performance.

Hardware Initialization

_nx_crypto_initialize() must be invoked before using the NetX Crypto APIs to ensure that the SCE peripheral is initialized.

Memory Usage

Sufficient memory must be allocated to be used as 'crypto_metadata' for the chosen crypto operation(s). Refer Azure RTOS NetX Crypto API description for recommended 'crypto_metadata_size' based on selected crypto operations. Sufficient amount of memory must be allocated for the thread stack to support low level crypto operations when using this module in the standalone mode or through NetX Secure (TLS). A minimum stack of 0x1000 is required to use ECC and RSA. This is either the main stack in a bare metal application or the specific thread stack for an RTOS based application.

AES Usage

GCM mode

The first byte of the IV must indicate the length of the subsequent IV. For example if the IV is {0x00, 0x00, 0x00}, then the IV pointer passed to the _nx_crypto_method_aes_operation must store the IV as {0x03, 0x00, 0x00, 0x00}. Refer to the example code for actual usage.

CTR mode

For CTR mode the IV pointer must be as defined in Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP) under 'Figure 2. Counter Block Format'. The IV must be 8 bytes in length. The Nonce field in the reference above must be 4 bytes and should be passed to _nx_crypto_method_aes_operation through the key pointer stored after the actual AES key. For Example, if the AES 128-bit Plain Key is {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00} and the Nonce is {0x01, 0x07, 0xBD, 0xFD}, the key passed to the _nx_crypto_method_aes_operation during Encryption/Decryption should be set as {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x07, 0xBD, 0xFD}. This format would also be valid for Wrapped keys where the Nonce is appended at the end of the actual Wrapped key. Refer to the example code for actual usage.

The 'Block Counter' field in the above reference is fixed to {0x00, 0x00, 0x00, 0x01} at the beginning and increments internally after every subsequent AES block is processed. Test vectors that have the initial Block Counter not set to {0x00, 0x00, 0x00, 0x01} cannnot be used in this implementation.

ECC Usage

ECC operations include ECDH and ECDSA. As a part of ECDSA operation the input message can be hashed before signing or verification, or the message digest can be provided directly. ECC Scratch buffer size can be optionally reduced as supported ECC computations are now done by the HW. This is controlled by NX_CRYPTO_ECDSA_SCRATCH_BUFFER_SIZE and NX_CRYPTO_ECDH_SCRATCH_BUFFER_SIZE macros for ECDSA and ECDH respectively.

Operation	Key Format
ECDSA Signature	Plain private key; Wrapped private key
ECDSA Signature-Verify	Uncompressed public key
Key Generation using ECDSA operation API	Wrapped private key; Uncompressed public key
ECDH private key import	Plain private key; Wrapped private key (Allows for Uncompressed and Formatted public key)
ECDH public key export	Uncompressed public key
ECDH setup	Uncompressed public key
ECDH shared secret calculate	Uncompressed public key
Key Generation using ECDH operation API	Wrapped private key; Uncompressed public key

Note:

- Uncompressed public key is of the form (0x04 || Qx || Qy). Refer Section 2.2. Subject Public Key under RFC5480
- Formatted public key is of the form (Key Info (4 bytes) || Qx || Qy || Key Info (16 bytes)). This is the key which is used internally by SCE peripheral. (Qx and Qy are zero padded for

224 bit curves).

RSA Usage

Wrapped Key Usage

To use the NetX Crypto stack with wrapped private keys (for signature generation/decryption), a dummy pointer (non-NULL) should be passed to the 'key' parameter during _nx_crypto_method_rsa_init API call. However, the 'key_size_in_bits' parameter should be equal to the intended RSA modulus length in bits. The actual wrapped key must be passed as the 'key' parameter to the _nx_crypto_method_rsa_operation API call with its length in bits passed through the 'key_size_in_bits' parameter.

For PKCS1V15 operation 'key' and 'key_size_in_bits' parameters of _nx_crypto_method_pkcs1_v1_5_init are unused. These can be passed as NULL and 0 respectively. The actual wrapped key must be passed as the 'key' parameter to the _nx_crypto_method_pkcs1_v1_5_operation API call and the intended modulus length in bits must be passed through the 'key_size_in_bits' parameter.

Limitations

- Only little endian mode is supported.
- RSA CRT keys are not supported.
- ECJPAKE related operations are unsupported for NIST 224, 256 and 384 bit curves when HW ECC is eanbled.

Examples

Initialization Example

This example shows how to initialize the HW crypto engine. This step must be performed before any crypto algorithm is used.

```
/* Setup the platform; initialize the SCE and the TRNG */
    err = _nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
```

Hash Example

This is an example on calculating the SHA256 hash using the NetX Crypto API.

```
extern NX_CRYPTO_METHOD crypto_method_sha256;
const uint8_t NIST_SHA256ShortMsgLen200[] =
{
    0x2e, 0x7e, 0xa8, 0x4d, 0xa4, 0xbc, 0x4d, 0x7c, 0xfb, 0x46, 0x3e, 0x3f, 0x2c,
0x86, 0x47, 0x05,
    0x7a, 0xff, 0xf3, 0xfb, 0xec, 0xec, 0xa1, 0xd2, 00
```

```
};
const uint8_t NIST_SHA256ShortMsgLen200_expected[] =
{
    0x76, 0xe3, 0xac, 0xbc, 0x71, 0x88, 0x36, 0xf2, 0xdf, 0x8a, 0xd2, 0xd0, 0xd2,
0xd7, 0x6f, 0x0c,
    0xfa, 0x5f, 0xea, 0x09, 0x86, 0xbe, 0x91, 0x8f, 0x10, 0xbc, 0xee, 0x73, 0x0d,
0xf4, 0x41, 0xb9
};
void netx_secure_crypto_sha256_example (void)
{
 size_t   actual_hash_len = RM_NETX_SECURE_CRYPTO_EXAMPLE_SHA256_HASH_SIZE_BYTES;
    uint8_t actual_hash[RM_NETX_SECURE_CRYPTO_EXAMPLE_SHA256_HASH_SIZE_BYTES];
    uint8_t metadata[sizeof(NX_CRYPTO_SHA256)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_SHA256);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* Setup the platform; initialize the SCE and the TRNG */
    err = _nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto SHA256 init */
    err = _nx_crypto_method_sha256_init(&crypto_method_sha256, NX_CRYPTO_NULL, 0,
&handler, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto SHA256 operation - NX_CRYPTO_HASH_INITIALIZE */
    err = _nx_crypto_method_sha256_operation(NX_CRYPTO_HASH_INITIALIZE,
                                             handler,
                                             &crypto_method_sha256,
                                             NX_CRYPTO_NULL,
                                             0,
                                             NX_CRYPTO_NULL,
                                             0,
                                             NX_CRYPTO_NULL,
                                             NX_CRYPTO_NULL,
                                             0,
```

```
                                             metadata,
                                             metadata_size,
                                             NX_CRYPTO_NULL,
                                             NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Nx Crypto SHA256 operation - NX_CRYPTO_HASH_UPDATE,
 * call this multiple times if needed to hash multiple data batches */
   err =
      _nx_crypto_method_sha256_operation(NX_CRYPTO_HASH_UPDATE,
                                          handler,
                                          &crypto_method_sha256,
                                          NX_CRYPTO_NULL,
                                          0,
                                          (uint8_t *) NIST_SHA256ShortMsgLen200,
sizeof(NIST_SHA256ShortMsgLen200),
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL,
                                          0,
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Nx Crypto SHA256 operation - NX_CRYPTO_HASH_CALCULATE */
   err = _nx_crypto_method_sha256_operation(NX_CRYPTO_HASH_CALCULATE,
                                             handler,
                                             &crypto_method_sha256,
                                             NX_CRYPTO_NULL,
                                             0,
                                             NX_CRYPTO_NULL,
                                             0,
                                             NX_CRYPTO_NULL,
                                             (uint8_t *) actual_hash,
                                             actual_hash_len,
```

```
                                             metadata,
                                             metadata_size,
                                             NX_CRYPTO_NULL,
                                             NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Ensure generated SHA256 hash matches the expected digest */
    err = (uint32_t) memcmp(&actual_hash[0], &NIST_SHA256ShortMsgLen200_expected[0],
actual_hash_len);
    assert(0 == err);
}
```

AES Example

This is an example on using the NetX Crypto API to encrypt and decrypt multi-block data.

AES CBC Example

```
extern NX_CRYPTO_METHOD crypto_method_aes_cbc_256;
/* fe8901fecd3ccd2ec5fdc7c7a0b50519c245b42d611a5ef9e90268d59f3edf33 */
const uint8_t NIST_AES256_CBC_key[] =
{
    0xfe, 0x89, 0x01, 0xfe, 0xcd, 0x3c, 0xcd, 0x2e, 0xc5, 0xfd, 0xc7, 0xc7, 0xa0,
0xb5, 0x05, 0x19,
    0xc2, 0x45, 0xb4, 0x2d, 0x61, 0x1a, 0x5e, 0xf9, 0xe9, 0x02, 0x68, 0xd5, 0x9f,
0x3e, 0xdf, 0x33
};
/* 851e8764776e6796aab722dbb644ace8 */
const uint8_t NIST_AES256_CBC_iv[] =
{
    0xbd, 0x41, 0x6c, 0xb3, 0xb9, 0x89, 0x22, 0x28, 0xd8, 0xf1, 0xdf, 0x57, 0x56,
0x92, 0xe4, 0xd0
};
/* 6282b8c05c5c1530b97d4816ca434762 */
const uint8_t NIST_AES256_CBC_plaintext[] =
{
    0x8d, 0x3a, 0xa1, 0x96, 0xec, 0x3d, 0x7c, 0x9b, 0x5b, 0xb1, 0x22, 0xe7, 0xfe,
```

```
0x77, 0xfb, 0x12,
    0x95, 0xa6, 0xda, 0x75, 0xab, 0xe5, 0xd3, 0xa5, 0x10, 0x19, 0x4d, 0x3a, 0x8a,
0x41, 0x57, 0xd5,
    0xc8, 0x9d, 0x40, 0x61, 0x97, 0x16, 0x61, 0x98, 0x59, 0xda, 0x3e, 0xc9, 0xb2,
0x47, 0xce, 0xd9
};
/* 6acc04142e100a65f51b97adf5172c41 */
const uint8_t NIST_AES256_CBC_ciphertext[] =
{
    0x60, 0x8e, 0x82, 0xc7, 0xab, 0x04, 0x00, 0x7a, 0xdb, 0x22, 0xe3, 0x89, 0xa4,
0x47, 0x97, 0xfe,
    0xd7, 0xde, 0x09, 0x0c, 0x8c, 0x03, 0xca, 0x8a, 0x2c, 0x5a, 0xcd, 0x9e, 0x84,
0xdf, 0x37, 0xfb,
    0xc5, 0x8c, 0xe8, 0xed, 0xb2, 0x93, 0xe9, 0x8f, 0x02, 0xb6, 0x40, 0xd6, 0xd1,
0xd7, 0x24, 0x64
};
void netx_secure_crypto_aes256cbc_multipart_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_AES)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_AES);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* 3 AES Blocks */
    uint8_t generated_ciphertext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
    uint8_t generated_plaintext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = _nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
    err =
       _nx_crypto_method_aes_init(&crypto_method_aes_cbc_256, (uint8_t *)
NIST_AES256_CBC_key,
 sizeof(NIST_AES256_CBC_key) << 3U, &handler, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Encryption. */
```

```
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_ENCRYPT,
                                          handler,
                                          &crypto_method_aes_cbc_256,
                                          NULL,
                                          0,
                                          (uint8_t *) NIST_AES256_CBC_plaintext,
 sizeof(NIST_AES256_CBC_plaintext),
                                          (uint8_t *) NIST_AES256_CBC_iv,
                                          generated_ciphertext,
 sizeof(generated_ciphertext),
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated ciphertext matches the expected ciphertext */
    err = (uint32_t) memcmp(generated_ciphertext, NIST_AES256_CBC_ciphertext,
sizeof(generated_ciphertext));
    assert(0 == err);
 /* Decryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_DECRYPT,
                                          handler,
                                          &crypto_method_aes_cbc_256,
                                          NULL,
                                          0,
                                          (uint8_t *) NIST_AES256_CBC_ciphertext,
 sizeof(NIST_AES256_CBC_ciphertext),
                                          (uint8_t *) NIST_AES256_CBC_iv,
                                          generated_plaintext,
 sizeof(generated_plaintext),
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
```

```
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated plaintext matches the input plaintext */
    err = (uint32_t) memcmp(generated_plaintext, NIST_AES256_CBC_plaintext,
sizeof(generated_ciphertext));
    assert(0 == err);
}
```

AES GCM Example

```
extern NX_CRYPTO_METHOD crypto_method_aes_128_gcm_16;
/* 83F9D97D4AB759FDDCC3EF54A0E2A8EC */
static const uint8_t key_gcm_128[] =
{
    0x83, 0xF9, 0xD9, 0x7D, 0x4A, 0xB7, 0x59, 0xFD, 0xDC, 0xC3, 0xEF, 0x54, 0xA0,
0xE2, 0xA8, 0xEC
};
/* In case of IV the IV length must be the first byte followed by the actual IV.
 * In this example the IV length is 0x01 and the actual IV is 0xCF
 */
/* 01CF */
static const uint8_t iv_gcm_128[] =
{
    0x01, 0xCF
};
/* 77E6329CF9424F71C808DF9170BFD298 */
static const uint8_t plain_gcm_128[] =
{
    0x77, 0xE6, 0x32, 0x9C, 0xF9, 0x42, 0x4F, 0x71, 0xC8, 0x08, 0xDF, 0x91, 0x70,
0xBF, 0xD2, 0x98
};
/* 6DD49EAEB4103DAC8F97E3234946DD2D */
static const uint8_t aad_gcm_128[] =
{
    0x6D, 0xD4, 0x9E, 0xAE, 0xB4, 0x10, 0x3D, 0xAC, 0x8F, 0x97, 0xE3, 0x23, 0x49,
0x46, 0xDD, 0x2D
```

```
};
/* 50DE86A7A92A8A5EA33DB5696B96CD77AA181E84BC8B4BF5A68927C409D422CB */
static const uint8_t secret_gcm_128[] =
{
 /* Ciphertext */
    0x50, 0xDE, 0x86, 0xA7, 0xA9, 0x2A, 0x8A, 0x5E, 0xA3, 0x3D, 0xB5, 0x69, 0x6B,
0x96, 0xCD, 0x77,
 /* Tag */
    0xAA, 0x18, 0x1E, 0x84, 0xBC, 0x8B, 0x4B, 0xF5, 0xA6, 0x89, 0x27, 0xC4, 0x09,
0xD4, 0x22, 0xCB
};
void netx_secure_crypto_aes128gcm_multipart_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_AES)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_AES);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* 3 AES Blocks */
    uint8_t generated_ciphertext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
    uint8_t generated_plaintext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = _nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
    err =
       _nx_crypto_method_aes_init(&crypto_method_aes_128_gcm_16,
                                  (uint8_t *) key_gcm_128,
 sizeof(key_gcm_128) << 3U,
            &handler,
            metadata,
            metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Setup Additional Authentication Data */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_SET_ADDITIONAL_DATA,
                                          handler,
```

```
                                          &crypto_method_aes_128_gcm_16,
                                          NULL,
                                          0,
                                          (uint8_t *) aad_gcm_128,
sizeof(aad_gcm_128),
                                          NULL,
                                          NULL,
                                          0,
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Encryption. */
   err = _nx_crypto_method_aes_operation(NX_CRYPTO_ENCRYPT,
                                          handler,
                                          &crypto_method_aes_128_gcm_16,
                                          NULL,
                                          0,
                                          (uint8_t *) plain_gcm_128,
sizeof(plain_gcm_128),
                                          (uint8_t *) iv_gcm_128,
                                          generated_ciphertext,
sizeof(generated_ciphertext),
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* The 16 byte tag is appended to the generated ciphertext */
/* Verify generated tag matches the expected tag */
   err = (uint32_t) memcmp(&generated_ciphertext[sizeof(plain_gcm_128)],
&secret_gcm_128[sizeof(plain_gcm_128)], 16U);
   assert(0 == err);
```

```
 /* Verify generated ciphertext matches the expected ciphertext */
    err = (uint32_t) memcmp(generated_ciphertext, secret_gcm_128, sizeof
(secret_gcm_128));
    assert(0 == err);
 /* Setup Additional Authentication Data */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_SET_ADDITIONAL_DATA,
                                          handler,
                                          &crypto_method_aes_128_gcm_16,
                                          NULL,
                                          0,
                                          (uint8_t *) aad_gcm_128,
 sizeof(aad_gcm_128),
                                          NULL,
                                          NULL,
                                          0,
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Decryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_DECRYPT,
                                          handler,
                                          &crypto_method_aes_128_gcm_16,
                                          NULL,
                                          0,
                                          (uint8_t *) generated_ciphertext,
 sizeof(secret_gcm_128), /* ciphertext size + tag size */
                                          (uint8_t *) iv_gcm_128,
                                          generated_plaintext,
 sizeof(generated_plaintext),
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
```

```
                                          NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated plaintext matches the input plaintext */
    err = (uint32_t) memcmp(generated_plaintext, plain_gcm_128, sizeof
(plain_gcm_128));
    assert(0 == err);
}
```

AES CTR Example

```
NX_CRYPTO_METHOD crypto_method_aes_ctr_256 =
{
    NX_CRYPTO_ENCRYPTION_AES_CTR,            /* AES crypto
algorithm                   */
    NX_CRYPTO_AES_256_KEY_LEN_IN_BITS,       /* Key size in bits
*/
    NX_CRYPTO_AES_IV_LEN_IN_BITS,            /* IV size in
bits                        */
    0,                                       /* ICV size in bits, not
used             */
    (NX_CRYPTO_AES_BLOCK_SIZE_IN_BITS >> 3), /* Block size in bytes
*/
 sizeof(NX_CRYPTO_AES),                   /* Metadata size in bytes               */
    _nx_crypto_method_aes_init,             /* AES-CBC initialization
routine         */
    _nx_crypto_method_aes_cleanup,          /* AES-CBC cleanup routine
*/
    _nx_crypto_method_aes_ctr_operation      /* AES-CBC
operation                      */
};
/*Note: For CTR, the key_ctr is the conjunction of key and nonce. */
/* D0E78C4D0B30D33F5BF4A132B2F94A4A38963511A3904B117E35A37B5AAC8A193BF0D158 */
const uint8_t key_ctr_256[] =
{
 /* AES Key */
```

```
    0xD0, 0xE7, 0x8C, 0x4D, 0x0B, 0x30, 0xD3, 0x3F, 0x5B, 0xF4, 0xA1, 0x32, 0xB2,
0xF9, 0x4A, 0x4A,
    0x38, 0x96, 0x35, 0x11, 0xA3, 0x90, 0x4B, 0x11, 0x7E, 0x35, 0xA3, 0x7B, 0x5A,
0xAC, 0x8A, 0x19,
 /* Nonce */
    0x3B, 0xF0, 0xD1, 0x58,
};
/* A1A31704C8B7E16C */
const uint8_t iv_ctr_256[] =
{
    0xA1, 0xA3, 0x17, 0x04, 0xC8, 0xB7, 0xE1, 0x6C,
};
/* 981FA33222C5451017530155A4BF7F29 */
const uint8_t plain_ctr_256[] =
{
    0x98, 0x1F, 0xA3, 0x32, 0x22, 0xC5, 0x45, 0x10, 0x17, 0x53, 0x01, 0x55, 0xA4,
0xBF, 0x7F, 0x29,
};
/* 643B91B4E541B20AAAEAB77F2D328566 */
const uint8_t secret_ctr_256[] =
{
    0x64, 0x3B, 0x91, 0xB4, 0xE5, 0x41, 0xB2, 0x0A, 0xAA, 0xEA, 0xB7, 0x7F, 0x2D,
0x32, 0x85, 0x66,
};
void netx_secure_crypto_aes256ctr_multipart_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_AES)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_AES);
 void    * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* 3 AES Blocks */
    uint8_t generated_ciphertext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
    uint8_t generated_plaintext[3U * NX_CRYPTO_AES_BLOCK_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
```

```
    err = _nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
    err =
       _nx_crypto_method_aes_init(&crypto_method_aes_ctr_256,
                                   (uint8_t *) key_ctr_256,

crypto_method_aes_ctr_256.nx_crypto_key_size_in_bits,
              &handler,
              metadata,
              metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Encryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_ENCRYPT,
                                           handler,
                                           &crypto_method_aes_ctr_256,
                                     (uint8_t *) key_ctr_256,

crypto_method_aes_ctr_256.nx_crypto_key_size_in_bits,
                                           (uint8_t *) plain_ctr_256,
 sizeof(plain_ctr_256),
                                           (uint8_t *) iv_ctr_256,
                                           generated_ciphertext,
 sizeof(generated_ciphertext),
                                           metadata,
                                           metadata_size,
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated ciphertext matches the expected ciphertext */
    err = (uint32_t) memcmp(generated_ciphertext, secret_ctr_256, sizeof
(secret_ctr_256));
    assert(0 == err);
 /* Decryption. */
    err = _nx_crypto_method_aes_operation(NX_CRYPTO_DECRYPT,
```

```
                                          handler,
                                          &crypto_method_aes_ctr_256,
                                    (uint8_t *) key_ctr_256,

crypto_method_aes_ctr_256.nx_crypto_key_size_in_bits,
                                          (uint8_t *) secret_ctr_256,
 sizeof(secret_ctr_256),
                                          (uint8_t *) iv_ctr_256,
                                          generated_plaintext,
 sizeof(generated_plaintext),
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify generated plaintext matches the input plaintext */
    err = (uint32_t) memcmp(generated_plaintext, plain_ctr_256, sizeof
(plain_ctr_256));
    assert(0 == err);
}
```

ECDSA Example

This is an example on using the NetX Crypto API to sign and verify input message data. Based on the hash algorithm selected a digest is computed of the plain input message before sign/verify.

```
extern NX_CRYPTO_METHOD crypto_method_ecdsa;
extern NX_CRYPTO_METHOD crypto_method_ec_secp256;
extern NX_CRYPTO_METHOD crypto_method_sha256;
const uint8_t ECC_SECP256R1Keydata[] =
{
    0xf9, 0xa7, 0x68, 0x71, 0x24, 0x68, 0x9d, 0x32, 0x92, 0x6f, 0x1d, 0xfb, 0xbe,
0xf2, 0x61, 0x41, // NOLINT(readability-magic-numbers)
    0x07, 0x54, 0x0d, 0xb9, 0xa8, 0x8a, 0x8b, 0xc2, 0xd5, 0xe9, 0x38, 0x4b, 0xf9,
0xe5, 0x43, 0x5a // NOLINT(readability-magic-numbers)
```

```
};
const uint8_t ECC_SECP256R1PublicKeydata[] =
{
    0x04,
               /* ASN1 Constant */
    0x5b, 0xba, 0xd4, 0x2e, 0xb5, 0xc1, 0x07, 0xf2, 0x0e, 0x01, 0x95, 0x42, 0x6e,
0x90, 0xb8, 0x4e, // NOLINT(readability-magic-numbers)
    0xe9, 0x5a, 0xa1, 0xe8, 0x4c, 0x6c, 0xa5, 0x32, 0x3c, 0xf3, 0x09, 0xf5, 0xff,
0x8b, 0x3d, 0x26, // NOLINT(readability-magic-numbers)
    0xb6, 0x88, 0xc1, 0xdb, 0x02, 0xaf, 0x4d, 0xa5, 0x0e, 0x73, 0x61, 0x96, 0xb3,
0x59, 0x95, 0x6f, // NOLINT(readability-magic-numbers)
    0x5e, 0xc9, 0xa1, 0xf9, 0xb7, 0xb3, 0xb6, 0xdf, 0x54, 0x82, 0x79, 0xe3, 0xb6,
0x4e, 0xac, 0xb6 // NOLINT(readability-magic-numbers)
};
const uint8_t ECC_SECP256R1Message[] = "ASYMMETRIC_INPUT_FOR_SIGN......";
void netx_secure_crypto_ecdsa_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_ECDSA)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_ECDSA);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
    ULONG sig_length;
    NX_CRYPTO_EXTENDED_OUTPUT extended_output;
    uint8_t output[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = _nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Call the crypto initialization function. */
    err = _nx_crypto_method_ecdsa_init(&crypto_method_ecdsa, NX_CRYPTO_NULL, 0,
&handler, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Set hash method. */
    err = _nx_crypto_method_ecdsa_operation(NX_CRYPTO_HASH_METHOD_SET,
                                            handler,
```

```
                                          &crypto_method_ecdsa,
                                           NX_CRYPTO_NULL,
                                           0,
                                          (uint8_t *) &crypto_method_sha256,
sizeof(NX_CRYPTO_METHOD *),
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL,
                                           0,
                                           metadata,
                                           metadata_size,
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Set EC curve. */
   err =
      _nx_crypto_method_ecdsa_operation(NX_CRYPTO_EC_CURVE_SET,
                                         handler,
                                         &crypto_method_ecdsa,
                                         NX_CRYPTO_NULL,
                                         0,
                                         (uint8_t *) &crypto_method_ec_secp256,
sizeof(NX_CRYPTO_METHOD *),
                                         NX_CRYPTO_NULL,
                                         NX_CRYPTO_NULL,
                                         0,
                                         metadata,
                                         metadata_size,
                                         NX_CRYPTO_NULL,
                                         NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
   extended_output.nx_crypto_extended_output_data          = output;
   extended_output.nx_crypto_extended_output_length_in_byte = sizeof(output);
/* Sign the hash data using ECDSA. */
   err = _nx_crypto_method_ecdsa_operation(NX_CRYPTO_SIGNATURE_GENERATE,
```

```
                                            handler,
                                           &crypto_method_ec_secp256,
                                           (uint8_t *) ECC_SECP256R1Keydata,
sizeof(ECC_SECP256R1Keydata) << 3,
                                           (uint8_t *) ECC_SECP256R1Message,
sizeof(ECC_SECP256R1Message),
      NX_CRYPTO_NULL,
                                           (uint8_t *) &extended_output,
sizeof(extended_output),
      metadata,
      metadata_size,
      NX_CRYPTO_NULL,
      NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
   sig_length = extended_output.nx_crypto_extended_output_actual_size;
/* Verify the generated signature. */
   err = _nx_crypto_method_ecdsa_operation(NX_CRYPTO_SIGNATURE_VERIFY,
                                            handler,
                                           &crypto_method_ec_secp256,
                                           (uint8_t *) ECC_SECP256R1PublicKeydata,
sizeof(ECC_SECP256R1PublicKeydata) << 3,
                                           (uint8_t *) ECC_SECP256R1Message,
sizeof(ECC_SECP256R1Message),
      NX_CRYPTO_NULL,
      output,
      sig_length,
      metadata,
      metadata_size,
      NX_CRYPTO_NULL,
      NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
}
```

ECDH Example

This is an example on using the NetX Crypto API to generate a shared secret using ECDH. A shared secret is computed using known public key (from peer) and generated private key. Another shared secret is computed using the generated public key and known private key (imported to mimic peer). Both the shared secrets are checked to be the same.

```
extern NX_CRYPTO_METHOD crypto_method_ecdh;
/*Private key 59137e38152350b195c9718d39673d519838055ad908dd4757152fd8255c09bf */
const uint8_t ECC_SECP256R1Keydata_ecdh[] =
{
    0x59, 0x13, 0x7e, 0x38, 0x15, 0x23, 0x50, 0xb1, 0x95, 0xc9, 0x71, 0x8d, 0x39,
0x67, 0x3d, 0x51, // NOLINT(readability-magic-numbers)
    0x98, 0x38, 0x05, 0x5a, 0xd9, 0x08, 0xdd, 0x47, 0x57, 0x15, 0x2f, 0xd8, 0x25,
0x5c, 0x09, 0xbf, // NOLINT(readability-magic-numbers)
};
/*Public key 4, a8c5fdce8b62c5ada598f141adb3b26cf254c280b2857a63d2ad783a73115f6b,
806e1aafec4af80a0d786b3de45375b517a7e5b51ffb2c356537c9e6ef227d4a*/
const uint8_t ECC_SECP256R1PublicKeydata_ecdh[] =
{
    0x04,
    0xa8,0xc5, 0xfd, 0xce, 0x8b, 0x62, 0xc5, 0xad, 0xa5, 0x98, 0xf1, 0x41, 0xad,
0xb3, 0xb2, 0x6c, // NOLINT(readability-magic-numbers)
    0xf2,0x54, 0xc2, 0x80, 0xb2, 0x85, 0x7a, 0x63, 0xd2, 0xad, 0x78, 0x3a, 0x73,
0x11, 0x5f, 0x6b, // NOLINT(readability-magic-numbers)
    0x80,0x6e, 0x1a, 0xaf, 0xec, 0x4a, 0xf8, 0x0a, 0x0d, 0x78, 0x6b, 0x3d, 0xe4,
0x53, 0x75, 0xb5, // NOLINT(readability-magic-numbers)
    0x17,0xa7, 0xe5, 0xb5, 0x1f, 0xfb, 0x2c, 0x35, 0x65, 0x37, 0xc9, 0xe6, 0xef,
0x22, 0x7d, 0x4a, // NOLINT(readability-magic-numbers)
};
void netx_secure_crypto_ecdh_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_ECDH)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_ECDH);
    uint32_t err          = NX_CRYPTO_SUCCESS;
    uint8_t local_public_key[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
    uint32_t local_public_key_len = 0;
    uint8_t shared_secret[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
```

```
    uint32_t shared_secret_len = 0;
    uint8_t output[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
    NX_CRYPTO_EXTENDED_OUTPUT extended_output;
 /* Setup the platform; initialize the SCE and the TRNG */
    err = _nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Call the crypto initialization function. */
    err = _nx_crypto_method_ecdh_init(&crypto_method_ecdh, NX_CRYPTO_NULL, 0,
NX_CRYPTO_NULL, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Set EC curve. */
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_EC_CURVE_SET,
                                           NX_CRYPTO_NULL,
                                           &crypto_method_ecdh,
                                           NX_CRYPTO_NULL,
                                           0,
                                           (uint8_t *) &crypto_method_ec_secp256,
 sizeof(NX_CRYPTO_METHOD *),
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL,
                                           0,
                                           metadata,
                                           metadata_size,
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Generate local public key. This will generate a key pair.
  * The private wrapped key will be held by the ecdh context and the public key
(local_public_key)
  * will be returned for sharing with the peer.
  */
    extended_output.nx_crypto_extended_output_data          = local_public_key;
    extended_output.nx_crypto_extended_output_length_in_byte = sizeof
(local_public_key);
```

```
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_DH_SETUP,
                                           NX_CRYPTO_NULL,
                                           &crypto_method_ecdh,
                                           NX_CRYPTO_NULL,
                                           0,
                                           NX_CRYPTO_NULL,
                                           0,
                                           NX_CRYPTO_NULL,
                                           (uint8_t *) &extended_output,
 sizeof(extended_output),
                                           metadata,
                                           metadata_size,
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
    local_public_key_len = extended_output.nx_crypto_extended_output_actual_size;
 /* Calculate shared secret using the test (peer's) public key. */
    extended_output.nx_crypto_extended_output_data          = shared_secret;
    extended_output.nx_crypto_extended_output_length_in_byte = sizeof(shared_secret);
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_DH_CALCULATE,
                                           NX_CRYPTO_NULL,
                                           &crypto_method_ecdh,
                                           NX_CRYPTO_NULL,
                                           0,
                                           (uint8_t *)
ECC_SECP256R1PublicKeydata_ecdh,
 sizeof(ECC_SECP256R1PublicKeydata_ecdh),
                                           NX_CRYPTO_NULL,
                                           (uint8_t *) &extended_output,
 sizeof(extended_output),
                                           metadata,
                                           metadata_size,
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL);
```

```
    assert(NX_CRYPTO_SUCCESS == err);
    shared_secret_len = extended_output.nx_crypto_extended_output_actual_size;
    err = _nx_crypto_method_ecdh_cleanup(metadata);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Verify. The below operations will be carried out by the peer. */
 /* Call the crypto initialization function. */
    err = _nx_crypto_method_ecdh_init(&crypto_method_ecdh, NX_CRYPTO_NULL, 0,
NX_CRYPTO_NULL, metadata, metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Set EC curve. */
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_EC_CURVE_SET,
                                           NX_CRYPTO_NULL,
                                           &crypto_method_ecdh,
                                           NX_CRYPTO_NULL,
                                           0,
                                           (uint8_t *) &crypto_method_ec_secp256,
 sizeof(NX_CRYPTO_METHOD *),
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL,
                                           0,
                                           metadata,
                                           metadata_size,
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Import the test private key. The peer could generate its own key pair,
  * in this example a test private key is used for simplicity. */
    err =
       _nx_crypto_method_ecdh_operation(NX_CRYPTO_DH_KEY_PAIR_IMPORT,
         NX_CRYPTO_NULL,
                                        &crypto_method_ecdh,
                                        (uint8_t *) ECC_SECP256R1Keydata_ecdh,
                                        (NX_CRYPTO_KEY_SIZE)
(sizeof(ECC_SECP256R1Keydata_ecdh) << 3),
```

```
                                        (uint8_t *) ECC_SECP256R1PublicKeydata_ecdh,
 sizeof(ECC_SECP256R1PublicKeydata_ecdh),
         NX_CRYPTO_NULL,
         NX_CRYPTO_NULL,
                                        0,
              metadata,
         metadata_size,
         NX_CRYPTO_NULL,
         NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Calculate the shared secret using the local public key generated above and shared
with the peer. */
    extended_output.nx_crypto_extended_output_data          = output;
    extended_output.nx_crypto_extended_output_length_in_byte = sizeof(output);
    err = _nx_crypto_method_ecdh_operation(NX_CRYPTO_DH_CALCULATE,
                                           NX_CRYPTO_NULL,
                                           &crypto_method_ecdh,
                                           NX_CRYPTO_NULL,
                                           0,
                                           local_public_key,
                                           local_public_key_len,
                                           NX_CRYPTO_NULL,
                                           (uint8_t *) &extended_output,
 sizeof(extended_output),
                                           metadata,
                                           metadata_size,
                                           NX_CRYPTO_NULL,
                                           NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Validate the output. Both the parties must generate the same shared secret */
    err = (extended_output.nx_crypto_extended_output_actual_size !=
shared_secret_len);
    assert(NX_CRYPTO_SUCCESS == err);
    err = (uint32_t) memcmp(output, shared_secret,
```

```
extended_output.nx_crypto_extended_output_actual_size);
    assert(NX_CRYPTO_SUCCESS == err);
}
```

RSA Example

This is an example on using the NetX Crypto API to encrypt and decrypt input message data.

```
extern NX_CRYPTO_METHOD crypto_method_rsa;
/* 00010001 */
const uint8_t public_e[] =
{
    0x00, 0x01, 0x00, 0x01,
};
/* 13FF7429F8E851F1079CCFCE3B3CD8606ABA8607AD85CBB3057501EBD58811F3C04823171F192C048E
1E883AF8CF958810151D3874AEDC8EC4F88D2065C581569F1E200852DD40B6DFD1652659085A9DD1D3B86
9EA3617D904D209DE156A60BA5929D02F16430273D10720C2F28D2B95684DCAA6B9F6A508EA2CBBC11B9F
3F30D6201EA6CFFBBF1C44255CEC58EE70DBC872442BCCF115D8F743557B5DE5F42DDDA6CEAE7977793CC
9D90ADFE65E520F5520B615CF3B8C2DC82D7AC75EDB1297CF38AB23A37EED18D4DD45D9AD051B26401BE8
6E8C8E53F9585A702D02F1B5BD65F6739DFA6BFFE560CA130B6F1D4779C556C06D9CD29FB72D8851904F9
CDEE9 */
const uint8_t private_e_2048[] =
{
    0x13, 0xFF, 0x74, 0x29, 0xF8, 0xE8, 0x51, 0xF1, 0x07, 0x9C, 0xCF, 0xCE, 0x3B,
0x3C, 0xD8, 0x60,
    0x6A, 0xBA, 0x86, 0x07, 0xAD, 0x85, 0xCB, 0xB3, 0x05, 0x75, 0x01, 0xEB, 0xD5,
0x88, 0x11, 0xF3,
    0xC0, 0x48, 0x23, 0x17, 0x1F, 0x19, 0x2C, 0x04, 0x8E, 0x1E, 0x88, 0x3A, 0xF8,
0xCF, 0x95, 0x88,
    0x10, 0x15, 0x1D, 0x38, 0x74, 0xAE, 0xDC, 0x8E, 0xC4, 0xF8, 0x8D, 0x20, 0x65,
0xC5, 0x81, 0x56,
    0x9F, 0x1E, 0x20, 0x08, 0x52, 0xDD, 0x40, 0xB6, 0xDF, 0xD1, 0x65, 0x26, 0x59,
0x08, 0x5A, 0x9D,
    0xD1, 0xD3, 0xB8, 0x69, 0xEA, 0x36, 0x17, 0xD9, 0x04, 0xD2, 0x09, 0xDE, 0x15,
0x6A, 0x60, 0xBA,
```

```
    0x59, 0x29, 0xD0, 0x2F, 0x16, 0x43, 0x02, 0x73, 0xD1, 0x07, 0x20, 0xC2, 0xF2,
0x8D, 0x2B, 0x95,
    0x68, 0x4D, 0xCA, 0xA6, 0xB9, 0xF6, 0xA5, 0x08, 0xEA, 0x2C, 0xBB, 0xC1, 0x1B,
0x9F, 0x3F, 0x30,
    0xD6, 0x20, 0x1E, 0xA6, 0xCF, 0xFB, 0xBF, 0x1C, 0x44, 0x25, 0x5C, 0xEC, 0x58,
0xEE, 0x70, 0xDB,
    0xC8, 0x72, 0x44, 0x2B, 0xCC, 0xF1, 0x15, 0xD8, 0xF7, 0x43, 0x55, 0x7B, 0x5D,
0xE5, 0xF4, 0x2D,
    0xDD, 0xA6, 0xCE, 0xAE, 0x79, 0x77, 0x79, 0x3C, 0xC9, 0xD9, 0x0A, 0xDF, 0xE6,
0x5E, 0x52, 0x0F,
    0x55, 0x20, 0xB6, 0x15, 0xCF, 0x3B, 0x8C, 0x2D, 0xC8, 0x2D, 0x7A, 0xC7, 0x5E,
0xDB, 0x12, 0x97,
    0xCF, 0x38, 0xAB, 0x23, 0xA3, 0x7E, 0xED, 0x18, 0xD4, 0xDD, 0x45, 0xD9, 0xAD,
0x05, 0x1B, 0x26,
    0x40, 0x1B, 0xE8, 0x6E, 0x8C, 0x8E, 0x53, 0xF9, 0x58, 0x5A, 0x70, 0x2D, 0x02,
0xF1, 0xB5, 0xBD,
    0x65, 0xF6, 0x73, 0x9D, 0xFA, 0x6B, 0xFF, 0xE5, 0x60, 0xCA, 0x13, 0x0B, 0x6F,
0x1D, 0x47, 0x79,
    0xC5, 0x56, 0xC0, 0x6D, 0x9C, 0xD2, 0x9F, 0xB7, 0x2D, 0x88, 0x51, 0x90, 0x4F,
0x9C, 0xDE, 0xE9,
};
/* E0F5059966A8AEC4BF7CDAC8AE2430BDF61C54D09CAB9963CBF9A52AC641E384B6431D3B6A9D181151
9A2904E1170A44446C80E7638A4AF2720A7654AB740D8A151FDD216F3D6933422FD9AC14AEDE9CCD021EA
79E46925F4B18FD1AF2C0073CFC3A69AC71A2B3673D08136CDB01C379892601C7C857D68018DAE924CB8C
D29377A14C752B92BAFF14C3A49725AE2FEFAAD4686D8A7D9F94EB11BF81E05BD5D2586526FB129E73539
F9223D496B2ACA23CCACC34D5B18533BD0F5815A76F94F4F55D965FE61599B44BD8FBAD35F42B612A4C4F
2765B2097A5C0090EA8166D9C6DA1E03B6119736B794600491C48433132D0F15D5DE3BB4270DF6BC9012B
74931 */
const uint8_t m_2048[] =
{
    0xE0, 0xF5, 0x05, 0x99, 0x66, 0xA8, 0xAE, 0xC4, 0xBF, 0x7C, 0xDA, 0xC8, 0xAE,
0x24, 0x30, 0xBD,
    0xF6, 0x1C, 0x54, 0xD0, 0x9C, 0xAB, 0x99, 0x63, 0xCB, 0xF9, 0xA5, 0x2A, 0xC6,
0x41, 0xE3, 0x84,
```

```
    0xB6, 0x43, 0x1D, 0x3B, 0x6A, 0x9D, 0x18, 0x11, 0x51, 0x9A, 0x29, 0x04, 0xE1,
0x17, 0x0A, 0x44,
    0x44, 0x6C, 0x80, 0xE7, 0x63, 0x8A, 0x4A, 0xF2, 0x72, 0x0A, 0x76, 0x54, 0xAB,
0x74, 0x0D, 0x8A,
    0x15, 0x1F, 0xDD, 0x21, 0x6F, 0x3D, 0x69, 0x33, 0x42, 0x2F, 0xD9, 0xAC, 0x14,
0xAE, 0xDE, 0x9C,
    0xCD, 0x02, 0x1E, 0xA7, 0x9E, 0x46, 0x92, 0x5F, 0x4B, 0x18, 0xFD, 0x1A, 0xF2,
0xC0, 0x07, 0x3C,
    0xFC, 0x3A, 0x69, 0xAC, 0x71, 0xA2, 0xB3, 0x67, 0x3D, 0x08, 0x13, 0x6C, 0xDB,
0x01, 0xC3, 0x79,
    0x89, 0x26, 0x01, 0xC7, 0xC8, 0x57, 0xD6, 0x80, 0x18, 0xDA, 0xE9, 0x24, 0xCB,
0x8C, 0xD2, 0x93,
    0x77, 0xA1, 0x4C, 0x75, 0x2B, 0x92, 0xBA, 0xFF, 0x14, 0xC3, 0xA4, 0x97, 0x25,
0xAE, 0x2F, 0xEF,
    0xAA, 0xD4, 0x68, 0x6D, 0x8A, 0x7D, 0x9F, 0x94, 0xEB, 0x11, 0xBF, 0x81, 0xE0,
0x5B, 0xD5, 0xD2,
    0x58, 0x65, 0x26, 0xFB, 0x12, 0x9E, 0x73, 0x53, 0x9F, 0x92, 0x23, 0xD4, 0x96,
0xB2, 0xAC, 0xA2,
    0x3C, 0xCA, 0xCC, 0x34, 0xD5, 0xB1, 0x85, 0x33, 0xBD, 0x0F, 0x58, 0x15, 0xA7,
0x6F, 0x94, 0xF4,
    0xF5, 0x5D, 0x96, 0x5F, 0xE6, 0x15, 0x99, 0xB4, 0x4B, 0xD8, 0xFB, 0xAD, 0x35,
0xF4, 0x2B, 0x61,
    0x2A, 0x4C, 0x4F, 0x27, 0x65, 0xB2, 0x09, 0x7A, 0x5C, 0x00, 0x90, 0xEA, 0x81,
0x66, 0xD9, 0xC6,
    0xDA, 0x1E, 0x03, 0xB6, 0x11, 0x97, 0x36, 0xB7, 0x94, 0x60, 0x04, 0x91, 0xC4,
0x84, 0x33, 0x13,
    0x2D, 0x0F, 0x15, 0xD5, 0xDE, 0x3B, 0xB4, 0x27, 0x0D, 0xF6, 0xBC, 0x90, 0x12,
0xB7, 0x49, 0x31,
};
/* 551C2E268F7ED44D0E8B063F5B2B510CB809F53BD54E9956971E243B2363DA123C29AB4A009EDE1FCE
C54625971A4E3490F3EA398BF7386AAC34720E43FB0C795445B520AEE4D7694EE1474F60F77E1B5F09FE2
ED004333658D212122F040322D1564512A1540400F27E18049A762A5EDC9F072CA4F49F408252D42B31BC
35523373740E90DDDA6A8CE7865EEB7C694A662C74412406AB190FE0435DA2551F0C24A48939DDA58A023
9706D40B4977473689DC36CE5A4DF4EF892816CBDE2780D9389B7384674C93B1DDAF728F292B5671679FC
```

```
7175AC0A3B2197B809E7CF410417010F3B1316D10D82466C62F3A01667B70A714E0499400E255D4C39EA7
DE55C */
const uint8_t plain_2048[] =
{
    0x55, 0x1C, 0x2E, 0x26, 0x8F, 0x7E, 0xD4, 0x4D, 0x0E, 0x8B, 0x06, 0x3F, 0x5B,
0x2B, 0x51, 0x0C,
    0xB8, 0x09, 0xF5, 0x3B, 0xD5, 0x4E, 0x99, 0x56, 0x97, 0x1E, 0x24, 0x3B, 0x23,
0x63, 0xDA, 0x12,
    0x3C, 0x29, 0xAB, 0x4A, 0x00, 0x9E, 0xDE, 0x1F, 0xCE, 0xC5, 0x46, 0x25, 0x97,
0x1A, 0x4E, 0x34,
    0x90, 0xF3, 0xEA, 0x39, 0x8B, 0xF7, 0x38, 0x6A, 0xAC, 0x34, 0x72, 0x0E, 0x43,
0xFB, 0x0C, 0x79,
    0x54, 0x45, 0xB5, 0x20, 0xAE, 0xE4, 0xD7, 0x69, 0x4E, 0xE1, 0x47, 0x4F, 0x60,
0xF7, 0x7E, 0x1B,
    0x5F, 0x09, 0xFE, 0x2E, 0xD0, 0x04, 0x33, 0x36, 0x58, 0xD2, 0x12, 0x12, 0x2F,
0x04, 0x03, 0x22,
    0xD1, 0x56, 0x45, 0x12, 0xA1, 0x54, 0x04, 0x00, 0xF2, 0x7E, 0x18, 0x04, 0x9A,
0x76, 0x2A, 0x5E,
    0xDC, 0x9F, 0x07, 0x2C, 0xA4, 0xF4, 0x9F, 0x40, 0x82, 0x52, 0xD4, 0x2B, 0x31,
0xBC, 0x35, 0x52,
    0x33, 0x73, 0x74, 0x0E, 0x90, 0xDD, 0xDA, 0x6A, 0x8C, 0xE7, 0x86, 0x5E, 0xEB,
0x7C, 0x69, 0x4A,
    0x66, 0x2C, 0x74, 0x41, 0x24, 0x06, 0xAB, 0x19, 0x0F, 0xE0, 0x43, 0x5D, 0xA2,
0x55, 0x1F, 0x0C,
    0x24, 0xA4, 0x89, 0x39, 0xDD, 0xA5, 0x8A, 0x02, 0x39, 0x70, 0x6D, 0x40, 0xB4,
0x97, 0x74, 0x73,
    0x68, 0x9D, 0xC3, 0x6C, 0xE5, 0xA4, 0xDF, 0x4E, 0xF8, 0x92, 0x81, 0x6C, 0xBD,
0xE2, 0x78, 0x0D,
    0x93, 0x89, 0xB7, 0x38, 0x46, 0x74, 0xC9, 0x3B, 0x1D, 0xDA, 0xF7, 0x28, 0xF2,
0x92, 0xB5, 0x67,
    0x16, 0x79, 0xFC, 0x71, 0x75, 0xAC, 0x0A, 0x3B, 0x21, 0x97, 0xB8, 0x09, 0xE7,
0xCF, 0x41, 0x04,
    0x17, 0x01, 0x0F, 0x3B, 0x13, 0x16, 0xD1, 0x0D, 0x82, 0x46, 0x6C, 0x62, 0xF3,
0xA0, 0x16, 0x67,
```

```
    0xB7, 0x0A, 0x71, 0x4E, 0x04, 0x99, 0x40, 0x0E, 0x25, 0x5D, 0x4C, 0x39, 0xEA,
0x7D, 0xE5, 0x5C,
};
/* 10F904E071338569EC131401A7869F42F3BCAE252B5D3C8755FD24D47997A9CD4221D992B2871E0528
3B98841FC5C379C5D0E35B3938279B344299C3CF1566E0C994D0A9013AF64174F1379A4B5E4E9DE57491F
3078F6D10011EA55535D0763E538662C9996F4FCF8B64A768685AA417ADB6978743D3D1F513CF143DD6D3
83AD6357728A88928D39E27EA4D0B2AF92FC7F63875F9D6A70FAE7993C1FF04DF9A2F99216874BC123D4B
7DA7E7E8974CFC10ACF0C7BC8747526A8D16791F969082EA9B0C36D77B67C37B325682D74178E4234D52D
5635273301A6CC35E315AE74D659B1433576DAAE6780FA39E0550D971F2CB5817CAAFC24B5220E21C8CEE
E85DD */
const uint8_t secret_2048[] =
{
    0x10, 0xF9, 0x04, 0xE0, 0x71, 0x33, 0x85, 0x69, 0xEC, 0x13, 0x14, 0x01, 0xA7,
0x86, 0x9F, 0x42,
    0xF3, 0xBC, 0xAE, 0x25, 0x2B, 0x5D, 0x3C, 0x87, 0x55, 0xFD, 0x24, 0xD4, 0x79,
0x97, 0xA9, 0xCD,
    0x42, 0x21, 0xD9, 0x92, 0xB2, 0x87, 0x1E, 0x05, 0x28, 0x3B, 0x98, 0x84, 0x1F,
0xC5, 0xC3, 0x79,
    0xC5, 0xD0, 0xE3, 0x5B, 0x39, 0x38, 0x27, 0x9B, 0x34, 0x42, 0x99, 0xC3, 0xCF,
0x15, 0x66, 0xE0,
    0xC9, 0x94, 0xD0, 0xA9, 0x01, 0x3A, 0xF6, 0x41, 0x74, 0xF1, 0x37, 0x9A, 0x4B,
0x5E, 0x4E, 0x9D,
    0xE5, 0x74, 0x91, 0xF3, 0x07, 0x8F, 0x6D, 0x10, 0x01, 0x1E, 0xA5, 0x55, 0x35,
0xD0, 0x76, 0x3E,
    0x53, 0x86, 0x62, 0xC9, 0x99, 0x6F, 0x4F, 0xCF, 0x8B, 0x64, 0xA7, 0x68, 0x68,
0x5A, 0xA4, 0x17,
    0xAD, 0xB6, 0x97, 0x87, 0x43, 0xD3, 0xD1, 0xF5, 0x13, 0xCF, 0x14, 0x3D, 0xD6,
0xD3, 0x83, 0xAD,
    0x63, 0x57, 0x72, 0x8A, 0x88, 0x92, 0x8D, 0x39, 0xE2, 0x7E, 0xA4, 0xD0, 0xB2,
0xAF, 0x92, 0xFC,
    0x7F, 0x63, 0x87, 0x5F, 0x9D, 0x6A, 0x70, 0xFA, 0xE7, 0x99, 0x3C, 0x1F, 0xF0,
0x4D, 0xF9, 0xA2,
    0xF9, 0x92, 0x16, 0x87, 0x4B, 0xC1, 0x23, 0xD4, 0xB7, 0xDA, 0x7E, 0x7E, 0x89,
0x74, 0xCF, 0xC1,
```

```
    0x0A, 0xCF, 0x0C, 0x7B, 0xC8, 0x74, 0x75, 0x26, 0xA8, 0xD1, 0x67, 0x91, 0xF9,
0x69, 0x08, 0x2E,
    0xA9, 0xB0, 0xC3, 0x6D, 0x77, 0xB6, 0x7C, 0x37, 0xB3, 0x25, 0x68, 0x2D, 0x74,
0x17, 0x8E, 0x42,
    0x34, 0xD5, 0x2D, 0x56, 0x35, 0x27, 0x33, 0x01, 0xA6, 0xCC, 0x35, 0xE3, 0x15,
0xAE, 0x74, 0xD6,
    0x59, 0xB1, 0x43, 0x35, 0x76, 0xDA, 0xAE, 0x67, 0x80, 0xFA, 0x39, 0xE0, 0x55,
0x0D, 0x97, 0x1F,
    0x2C, 0xB5, 0x81, 0x7C, 0xAA, 0xFC, 0x24, 0xB5, 0x22, 0x0E, 0x21, 0xC8, 0xCE,
0xEE, 0x85, 0xDD,
};
void netx_secure_crypto_rsa_example (void)
{
    uint8_t metadata[sizeof(NX_CRYPTO_RSA)];
    uint32_t metadata_size = sizeof(NX_CRYPTO_RSA);
    uint32_t err          = NX_CRYPTO_SUCCESS;
 void   * handler       = NX_CRYPTO_NULL;
    uint8_t output[RM_NETX_SECURE_CRYPTO_EXAMPLE_OUTPUT_BUFFER_SIZE] = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
    err = _nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Encryption. */
    err =
       _nx_crypto_method_rsa_init(&crypto_method_rsa,
                                  (uint8_t *) m_2048,

RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(m_2048)),
              &handler,
              metadata,
              metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
    err = _nx_crypto_method_rsa_operation(NX_CRYPTO_ENCRYPT,
                                          handler,
                                          &crypto_method_rsa,
```

```
                                          (uint8_t *) public_e,



                     RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(public_e)),
                                          (uint8_t *) plain_2048,
 sizeof(m_2048),
                                          NX_CRYPTO_NULL,
         output,
 sizeof(m_2048),
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
    err = (uint32_t) memcmp(output, secret_2048, sizeof(m_2048));
    assert(0 == err);
    err = _nx_crypto_method_rsa_cleanup(metadata);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Decryption. */
    memset(output, 0, sizeof(output));
    err =
       _nx_crypto_method_rsa_init(&crypto_method_rsa,
                                  (uint8_t *) m_2048,

RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(m_2048)),
              &handler,
              metadata,
              metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
    err = _nx_crypto_method_rsa_operation(NX_CRYPTO_DECRYPT,
                                          handler,
                                          &crypto_method_rsa,
                                          (uint8_t *) private_e_2048,
```

```
                     RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(private_e_2048)),
                                          (uint8_t *) secret_2048,
 sizeof(m_2048),
                                          NX_CRYPTO_NULL,
         output,
 sizeof(m_2048),
                                          metadata,
                                          metadata_size,
                                          NX_CRYPTO_NULL,
                                          NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
    err = (uint32_t) memcmp(output, plain_2048, sizeof(m_2048));
    assert(0 == err);
}
```

RSA PKCS1V1.5 Example

This is an example on using the NetX Crypto API to sign and verify input message data. The plain input message is PKCS1V1.5 encoded before signature generation.

PKCS1V15 Example

HMAC SHA256 Example

This is an example on using the HMAC with SHA256 hash using the NetX Crypto API.

```
extern NX_CRYPTO_METHOD crypto_method_hmac_sha256;
/*
C4DA057B81EA740B697FFE1B6EB8591356BA6D5EA7F1B96E4F048030449ACD64E4BB271CB4DCF94937E6
*/
const uint8_t key_256[] =
{
    0xC4, 0xDA, 0x05, 0x7B, 0x81, 0xEA, 0x74, 0x0B, 0x69, 0x7F, 0xFE, 0x1B, 0x6E,
0xB8, 0x59, 0x13,
    0x56, 0xBA, 0x6D, 0x5E, 0xA7, 0xF1, 0xB9, 0x6E, 0x4F, 0x04, 0x80, 0x30, 0x44,
0x9A, 0xCD, 0x64,
    0xE4, 0xBB, 0x27, 0x1C, 0xB4, 0xDC, 0xF9, 0x49, 0x37, 0xE6,
```

```
};
/* BDACB6555D294D3AFFC245520116062D98F88D64276BDA593492AE71CFE16E46CABC287CB00DF21D96
066D5856C2224EEF609D4896302540078F3A0EE325F5337E */
const uint8_t plain_256[] =
{
    0xBD, 0xAC, 0xB6, 0x55, 0x5D, 0x29, 0x4D, 0x3A, 0xFF, 0xC2, 0x45, 0x52, 0x01,
0x16, 0x06, 0x2D,
    0x98, 0xF8, 0x8D, 0x64, 0x27, 0x6B, 0xDA, 0x59, 0x34, 0x92, 0xAE, 0x71, 0xCF,
0xE1, 0x6E, 0x46,
    0xCA, 0xBC, 0x28, 0x7C, 0xB0, 0x0D, 0xF2, 0x1D, 0x96, 0x06, 0x6D, 0x58, 0x56,
0xC2, 0x22, 0x4E,
    0xEF, 0x60, 0x9D, 0x48, 0x96, 0x30, 0x25, 0x40, 0x07, 0x8F, 0x3A, 0x0E, 0xE3,
0x25, 0xF5, 0x33,
    0x7E,
};
/* 940F986AC891C9000B72EF0CEC69AB66AF002E3A34EB8A3A5F94484E45C0396C */
const uint8_t secret_256[] =
{
    0x94, 0x0F, 0x98, 0x6A, 0xC8, 0x91, 0xC9, 0x00, 0x0B, 0x72, 0xEF, 0x0C, 0xEC,
0x69, 0xAB, 0x66,
    0xAF, 0x00, 0x2E, 0x3A, 0x34, 0xEB, 0x8A, 0x3A, 0x5F, 0x94, 0x48, 0x4E, 0x45,
0xC0, 0x39, 0x6C,
};
void netx_secure_crypto_hmac_sha256_example (void)
{
    uint8_t output[RM_NETX_SECURE_CRYPTO_EXAMPLE_SHA256_HASH_SIZE_BYTES] = {0};
    uint8_t metadata[sizeof(NX_CRYPTO_SHA256_HMAC)] = {0};
    uint32_t metadata_size = sizeof(NX_CRYPTO_SHA256_HMAC);
 void   * handler       = NX_CRYPTO_NULL;
    uint32_t err          = NX_CRYPTO_SUCCESS;
 /* Setup the platform; initialize the SCE and the TRNG */
    err = _nx_crypto_initialize();
    assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto HMAC-SHA256 init */
```

```
    err = _nx_crypto_method_hmac_sha256_init(&crypto_method_hmac_sha256,
                                             (UCHAR *) key_256,



                 RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(key_256)),
              &handler,
                                             metadata,
              metadata_size);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto HMAC-SHA256 operation - NX_CRYPTO_HASH_INITIALIZE */
    err =
       _nx_crypto_method_hmac_sha256_operation(NX_CRYPTO_HASH_INITIALIZE,
                                               handler,
                                                &crypto_method_hmac_sha256,
                                               (UCHAR *) key_256,

RM_NETX_SECURE_CRYPTO_BYTES_TO_BITS(sizeof(key_256)),
                                               NX_CRYPTO_NULL,
                                               0,
                                               NX_CRYPTO_NULL,
                                               NX_CRYPTO_NULL,
                                               0,
                                               metadata,
                                               metadata_size,
                                               NX_CRYPTO_NULL,
                                               NX_CRYPTO_NULL);
    assert(NX_CRYPTO_SUCCESS == err);
 /* Nx Crypto HMAC-SHA256 operation - NX_CRYPTO_HASH_UPDATE */
    err = _nx_crypto_method_hmac_sha256_operation(NX_CRYPTO_HASH_UPDATE,
                                               handler,
                                                  &crypto_method_hmac_sha256,
                                               NX_CRYPTO_NULL,
                                               0,
```

```
                                                (UCHAR *) plain_256,
sizeof(plain_256),
                                              NX_CRYPTO_NULL,
                                              NX_CRYPTO_NULL,
                                              0,
                                              metadata,
                                              metadata_size,
                                              NX_CRYPTO_NULL,
                                              NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Nx Crypto HMAC-SHA256 operation - NX_CRYPTO_HASH_CALCULATE */
   err = _nx_crypto_method_hmac_sha256_operation(NX_CRYPTO_HASH_CALCULATE,
                                              handler,
                                                 &crypto_method_hmac_sha256,
                                              NX_CRYPTO_NULL,
                                              0,
                                              NX_CRYPTO_NULL,
                                              0,
                                              NX_CRYPTO_NULL,
                                                (UCHAR *) output,
sizeof(output),
                                              metadata,
                                              metadata_size,
                                              NX_CRYPTO_NULL,
                                              NX_CRYPTO_NULL);
   assert(NX_CRYPTO_SUCCESS == err);
/* Ensure generated HMAC-SHA256 mac matches the expected mac */
   err = (uint32_t) memcmp(output, secret_256, sizeof(secret_256));
   assert(0 == err);
}
```

4.2.110 Azure RTOS NetX Duo Ether Driver (rm_netxduo_ether)

Modules

Overview

This module provides a NetX Duo driver that is implemented using the Ethernet Interface.

Please refer to the NetXDuo documentation for further details.

Features

- Packet Types Supported
  - ARP
  - IPv4
  - IPv6
- Link status callback
- Configurable IP MTU

Configuration

Configurations for Networking > NetX Duo Ethernet Driver (rm_netxduo_ether)

Configuration	Options	Default	Description
Name	Name must be a valid C symbol	g_netxduo_ether_0	Module name.
IP MTU	Value must be in the range [576, 1500] bytes.	1500	IP MTU

Usage Notes

Calculating the Packet Size for an IP instance

In order to ensure that there is enough space to store an entire Ethernet frame, the packet pool used for receiving packets must have a payload size that is 32 bytes larger than the configured ether_cfg_t::ether_buffer_size. The extra 32 bytes is needed in order to ensure that the allocated packets are properly aligned to 32 bytes.

ether_cfg_t::ether_buffer_size is calcualted from the IP MTU using the following formula:

ceil((rm_netxduo_ether_cfg_t::mtu + Ethernet Header (14) + Padding Bytes (2)) / 32) * 32

Examples

Basic Example

This is a basic example of minimal use of the NetX Duo Ether Driver in an application.

```
#define NETXDUO_EXAMPLE_IP_STACK_SIZE (2048U)
```

```
#define NETXDUO_EXAMPLE_ARP_CACHE_SIZE (2048U)
#define NETXDUO_EXAMPLE_PACKET_SIZE (1568U)
#define NETXDUO_EXAMPLE_PACKET_NUM (100U)
#define NETXDUO_EXAMPLE_PACKET_POOL_SIZE ((sizeof(NX_PACKET) +
NETXDUO_EXAMPLE_PACKET_SIZE) * \
  NETXDUO_EXAMPLE_PACKET_NUM)
static NX_IP g_ip;
static NX_PACKET_POOL g_packet_pool;
static uint8_t g_ip_stack_memory[NETXDUO_EXAMPLE_IP_STACK_SIZE]
BSP_ALIGN_VARIABLE(4);
static uint8_t g_packet_pool_memory[NETXDUO_EXAMPLE_PACKET_POOL_SIZE]
BSP_ALIGN_VARIABLE(4);
static uint8_t g_ip_arp_cache_memory[NETXDUO_EXAMPLE_ARP_CACHE_SIZE]
BSP_ALIGN_VARIABLE(4);
static void rm_netxduo_ether0 (NX_IP_DRIVER * driver_req_ptr)
{
 /* Pass the driver request and ethernet driver configuration to the NetX Duo Ether
Driver. */
    rm_netxduo_ether(driver_req_ptr, &g_netxduo_ether_instance);
}
void rm_netxduo_ether_example ()
{
    UINT status;
    nx_system_initialize();
 /* Create a packet pool for the IP instance. */
    status = nx_packet_pool_create(&g_packet_pool,
 "Packet Pool",
                                  NETXDUO_EXAMPLE_PACKET_SIZE,
                                  &g_packet_pool_memory[0],
                                  NETXDUO_EXAMPLE_PACKET_POOL_SIZE);
    assert(NX_SUCCESS == status);
 /* Create an IP instance using the rm_netxduo_ether driver and packet pool instance.
*/
    status = nx_ip_create(&g_ip,
```

```
 "IP Instance",
                         IP_ADDRESS(192, 168, 1, 2),
                         IP_ADDRESS(255, 255, 255, 0),
                         &g_packet_pool,
                         rm_netxduo_ether0,
                         &g_ip_stack_memory[0],
 sizeof(g_ip_stack_memory),
                         0);
    assert(NX_SUCCESS == status);
 /* Enable all modules that are required by the application. */
    status = nx_arp_enable(&g_ip, g_ip_arp_cache_memory, sizeof
(g_ip_arp_cache_memory));
    assert(NX_SUCCESS == status);
    status = nx_tcp_enable(&g_ip);
    assert(NX_SUCCESS == status);
    status = nx_icmp_enable(&g_ip);
    assert(NX_SUCCESS == status);
}
```

4.2.111 Azure RTOS NetX Duo WiFi Driver (rm_netxduo_wifi)

Modules

Overview

This module provides a NetX Duo driver that is implemented using the rm_wifi_onchip_silex driver.

Please refer to the NetXDuo documentation for further details.

Features

- Packet Types Supported
    - TCP/IPv4
- Configurable IP MTU

Configuration

Build Time Configurations for rm_netxduo_wifi

The following build time configurations are defined in fsp_cfg/middleware/rm_netxduo_wifi_cfg.h:

Configuration	Options	Default	Description
IP MTU (bytes)	Value must be in the range [576, 1500] bytes.	1500	IP MTU

Usage Notes

Connecting to a Wireless Network

NetX Duo does not support connecting to a wireless network directly and instead the driver functions should be called directly to establish a connection.

- rm_wifi_onchip_silex_open() should be called to initialize the WiFi module driver.
- rm_wifi_onchip_silex_scan() should be used to scan for access points.
- rm_wifi_onchip_silex_connect() should be used before opening a NetX IP instance in order to connect it to a network.
- rm_wifi_onchip_silex_network_info_get() should be used after an access point connection has been established to get DHCP acquired IP address, subnet mask, and gateway.

Unsupported NetX Duo Features

TCP client sockets are the sole feature supported by both the hardware TCP/IP feature in NetX Duo and the WiFi driver as well. Any protocol that does not utilize a TCP client socket will be unsupported.

- TCP server mode is not currently supported.
- DNS is not supported. The user can preform a DNS lookup by calling rm_wifi_onchip_silex_dns_query.
- ICMP is not supported. The user can perform a ping by calling rm_wifi_onchip_silex_ping.
- ARP is not supported. This should be handled directly by the WiFi module.
- UDP is not supported. The WiFi module driver does not support UDP sockets.
- SNTP is not supported. See the public APIs for communication with time servers Wifi Middleware (rm_wifi_onchip_silex).
- Static address assignment is not currently supported by the rm_wifi_onchip_silex driver. You must use DHCP.

Examples

Basic Example

This is a basic example of minimal use of the NetX Duo WiFi Driver in an application.

```
#define NETXDUO_EXAMPLE_IP_STACK_SIZE (2048U)

#define NETXDUO_EXAMPLE_ARP_CACHE_SIZE (2048U)

#define NETXDUO_EXAMPLE_PACKET_SIZE (1568U)

#define NETXDUO_EXAMPLE_PACKET_NUM (100U)
```

```
#define NETXDUO_EXAMPLE_PACKET_POOL_SIZE ((sizeof(NX_PACKET) +
NETXDUO_EXAMPLE_PACKET_SIZE) * \
  NETXDUO_EXAMPLE_PACKET_NUM)
#define NETXDUO_EXAMPLE_SSID "ssidName"
#define NETXDUO_EXAMPLE_PASSWORD "password"
static NX_IP g_ip0;
static NX_PACKET_POOL g_packet_pool0;
static uint8_t g_ip0_stack_memory[NETXDUO_EXAMPLE_IP_STACK_SIZE]
BSP_ALIGN_VARIABLE(4);
static uint8_t g_packet_pool0_pool_memory[NETXDUO_EXAMPLE_PACKET_POOL_SIZE]
BSP_ALIGN_VARIABLE(4);
extern wifi_onchip_silex_cfg_t g_wifi_onchip_silex_cfg;
void rm_netxduo_wifi_example ()
{
    UINT      status;
 fsp_err_t err;
    nx_system_initialize();
 /* Open WiFi module */
    err = rm_wifi_onchip_silex_open(&g_wifi_onchip_silex_cfg);
    assert(FSP_SUCCESS == err);
 /* Connect to desired AP */
    err = rm_wifi_onchip_silex_connect(NETXDUO_EXAMPLE_SSID, eWiFiSecurityWPA2,
NETXDUO_EXAMPLE_PASSWORD);
    assert(FSP_SUCCESS == err);
 /* Create a packet pool for the IP instance. */
    status = nx_packet_pool_create(&g_packet_pool0,
 "Packet Pool",
                                  NETXDUO_EXAMPLE_PACKET_SIZE,
                                  &g_packet_pool0_pool_memory[0],
                                  NETXDUO_EXAMPLE_PACKET_POOL_SIZE);
    assert(NX_SUCCESS == status);
 /* Create an IP instance using the rm_netxduo_wifi driver and packet pool instance.
*/
    status = nx_ip_create(&g_ip0,
```

```
 "IP Instance",
                        IP_ADDRESS(192, 168, 1, 2),
                        IP_ADDRESS(255, 255, 255, 0),
                        &g_packet_pool0,
                        rm_netxduo_wifi,
                        &g_ip0_stack_memory[0],
 sizeof(g_ip0_stack_memory),
                        0);
    assert(NX_SUCCESS == status);
 /* Enable all modules that are required by the application. */
    status = nx_tcp_enable(&g_ip0);
    assert(NX_SUCCESS == status);
}
```

TLS Example

This is a basic example of connecting a TLS socket.

```
#define NETXDUO_EXAMPLE_CRYPTO_METADATA_BUFFER_SIZE (18000U)
#define NETXDUO_EXAMPLE_TLS_PACKET_REASSEMBLY_BUFFER_SIZE (4000U)
#define NETXDUO_EXAMPLE_PORT (3005U)
#define NETXDUO_EXAMPLE_SERVER_PORT (9050U)
#define NETXDUO_EXAMPLE_IP IP_ADDRESS(1, 2, 3, 5)
NX_SECURE_TLS_CRYPTO       g_nx_crypto_tls_test_ciphers;
static UCHAR g_tls_crypto_metadata[NETXDUO_EXAMPLE_CRYPTO_METADATA_BUFFER_SIZE];
static UCHAR g_tls_packet_buffer[NETXDUO_EXAMPLE_TLS_PACKET_REASSEMBLY_BUFFER_SIZE];
static NX_SECURE_X509_CERT g_certificate;
extern const UCHAR g_trusted_ca_data; // User trusted certificates
extern USHORT g_trusted_ca_length;
void rm_netxduo_wifi_tls_example ()
{
    UINT                  status;
 fsp_err_t            err;
    NX_TCP_SOCKET       * p_socket;
    NX_SECURE_TLS_SESSION tls_session;
```

```
    nx_system_initialize();
 /* Initialize NetX Crypto */
    status = nx_crypto_initialize();
    assert(NX_SUCCESS == status);
 /* Open WiFi module */
    err = rm_wifi_onchip_silex_open(&g_wifi_onchip_silex_cfg);
    assert(FSP_SUCCESS == err);
 /* Connect to desired AP */
    err = rm_wifi_onchip_silex_connect(NETXDUO_EXAMPLE_SSID, eWiFiSecurityWPA2,
NETXDUO_EXAMPLE_PASSWORD);
    assert(FSP_SUCCESS == err);
 /* Create a packet pool for the IP instance. */
    status = nx_packet_pool_create(&g_packet_pool0,
 "Packet Pool",
                                   NETXDUO_EXAMPLE_PACKET_SIZE,
                                   &g_packet_pool0_pool_memory[0],
                                   NETXDUO_EXAMPLE_PACKET_POOL_SIZE);
    assert(NX_SUCCESS == status);
 /* Create an IP instance using the rm_netxduo_wifi driver and packet pool instance.
*/
    status = nx_ip_create(&g_ip0,
 "IP Instance",
                          IP_ADDRESS(192, 168, 1, 2),
                          IP_ADDRESS(255, 255, 255, 0),
                          &g_packet_pool0,
                          rm_netxduo_wifi,
                          &g_ip0_stack_memory[0],
 sizeof(g_ip0_stack_memory),
                          0);
    assert(NX_SUCCESS == status);
 /* Enable all modules that are required by the application. */
    status = nx_tcp_enable(&g_ip0);
    assert(NX_SUCCESS == status);
 /* Initialize the NetX Secure TLS system. */
```

```
    nx_secure_tls_initialize();
 /* Create a TCP socket to use for the TLS session. */
    status = nx_tcp_socket_create(&g_ip0,
                                  p_socket,
 "TLS Client Socket",
                                  NX_IP_NORMAL,
                                  NX_FRAGMENT_OKAY,
                                  NX_IP_TIME_TO_LIVE,
                                  1024 * 4,
                                  NX_NULL,
                                  NX_NULL);
    assert(NX_SUCCESS == status);
 /* Create a TLS session for our socket. This sets up the TLS session object for
  * later use */
    status =
       nx_secure_tls_session_create(&tls_session,
                                     &g_nx_crypto_tls_test_ciphers,
                                     g_tls_crypto_metadata,
                                     NETXDUO_EXAMPLE_CRYPTO_METADATA_BUFFER_SIZE);
    assert(NX_SUCCESS == status);
 /* Set the packet reassembly buffer for this TLS session. */
    status = nx_secure_tls_session_packet_buffer_set(&tls_session,
                                                    g_tls_packet_buffer,

NETXDUO_EXAMPLE_TLS_PACKET_REASSEMBLY_BUFFER_SIZE);
    assert(NX_SUCCESS == status);
 /* Initialize an X.509 certificate with the CA root certificate data. */
    status = nx_secure_x509_certificate_initialize(&g_certificate,
                                                   (UCHAR *) g_trusted_ca_data,
                                                   g_trusted_ca_length,
                                    NX_NULL,
                                                   0,
                                    NX_NULL,
                                                   0,
```

```
                                                    NX_SECURE_X509_KEY_TYPE_NONE);
    assert(NX_SUCCESS == status);
 /* Add the initialized certificate as a trusted root certificate. */
    status = nx_secure_tls_trusted_certificate_add(&tls_session, &g_certificate);
    assert(NX_SUCCESS == status);
 /* Bind the socket to a port. */
    status = nx_tcp_client_socket_bind(p_socket, NETXDUO_EXAMPLE_PORT,
NX_WAIT_FOREVER);
    assert(NX_SUCCESS == status);
 /* Connect TCP socket */
    status = nx_tcp_client_socket_connect(p_socket, NETXDUO_EXAMPLE_IP,
NETXDUO_EXAMPLE_SERVER_PORT, NX_WAIT_FOREVER);
    assert(NX_SUCCESS == status);
 /* Start the TLS Session using the connected TCP socket. This function will
  * ascertain from the TCP socket state that this is a TLS Client session. */
    status = nx_secure_tls_session_start(&tls_session, p_socket, NX_WAIT_FOREVER);
    assert(NX_SUCCESS == status);
}
```

4.2.112 Crypto Middleware (rm_psa_crypto)

Modules

Functions

fsp_err_t	RM_PSA_CRYPTO_TRNG_Read (uint8_t *const p_rngbuf, uint32_t num_req_bytes, uint32_t *p_num_gen_bytes) Reads requested length of random data from the TRNG. Generate nbytes of random bytes and store them in p_rngbuf buffer. More...
int	mbedtls_platform_setup (mbedtls_platform_context *ctx)
void	mbedtls_platform_teardown (mbedtls_platform_context *ctx)

Detailed Description

Hardware acceleration for the mbedCrypto implementation of the ARM PSA Crypto API.

Overview

Note

> ***The PSA Crypto module does not provide any interfaces to the user. This release uses the Mbed TLSversion 2.24.0 which conforms to the PSA Crypto API 1.0 specification. Consult the ARM documentation at https://armmbed.github.io/mbed-crypto/psa/#application-programming-interface for further information.*** *FSP 3.0 onward adopts a change by ARM where mbedCrypto has been integrated back to MbedTLS and the term mbedCrypto has been deprecated. The mbedCrypto term in FSP now refers to the crypto portion of the MbedTLS module.*

HW Overview

Crypto Peripheral version	Devices
SCE9	RA6M4, RA4M3, RA4M2, RA6M5
SCE7	RA6M3, RA6M2, RA6M1, RA6T1
SCE5	RA4W1, RA4M1
AES Engine	RA2A1, RA2E1, RA2E2, RA2L1

Features

The PSA_Crypto module provides hardware support for the following PSA Crypto operations

- SHA256 calculation
- SHA224 calculation
    - MAC Operations
- AES
    - Keybits - 128, 192, 256
    - Plain-Text Key Generation
    - Wrapped Key Generation
    - Encryption and Decryption with no padding and with PKCS7 padding.
    - CBC, CTR, CCM and GCM modes
    - MAC operations
    - Export and Import for Plaintext and Wrapped keys
- ECC
    - Curves:
        - SECP256R1
        - SECP256K1
        - Brainpool256R1
        - SECP384R1
        - Brainpool384R1
    - Plain-Text Key Generation (Unavailable on SCE9)
    - Wrapped Key Generation
    - Signing and Verification
    - Export and Import for Plaintext and Wrapped keys
    - ECDH Support
- RSA
    - Keybits - 2048. Verification only for 3072 and 4096 bits
    - Plain-Text Key Generation (Unavailable on SCE9)
    - Wrapped Key Generation
    - Signature Generation

    - Verification
    - Encryption and Decryption with PKCS1V15 and OAEP padding
    - Export and Import for Plaintext and Wrapped keys
- Random number generation
- Persistent Key Storage

Configuration

Build Time Configurations for mbedCrypto

The following build time configurations are defined in arm/mbedtls/config.h:

Configuration	Options	Default	Description
Hardware Acceleration > Key Format > AES	MCU Specific Options		Select AES key formats used
Hardware Acceleration > Key Format > ECC	MCU Specific Options		Select ECC key formats used
Hardware Acceleration > Key Format > RSA	MCU Specific Options		Select RSA key formats used
Hardware Acceleration > Hash > SHA256/224	MCU Specific Options		Defines MBEDTLS_SHA256_ALT and MBEDTLS_SHA256_PROCESS_ALT.
Hardware Acceleration > Cipher > AES	MCU Specific Options		Defines MBEDTLS_AES_ALT, MBEDTLS_AES_SETKEY_ENC_ALT, MBEDTLS_AES_SETKEY_DEC_ALT, MBEDTLS_AES_ENCRYPT_ALT and MBEDTLS_AES_DECRYPT_ALT
Hardware Acceleration > Public Key Cryptography (PKC) > ECC	MCU Specific Options		Defines MBEDTLS_ECP_ALT
Hardware Acceleration > Public Key Cryptography (PKC) > ECDSA	MCU Specific Options		Defines MBEDTLS_ECDSA_SIGN_ALT and MBEDTLS_ECDSA_VERIFY_ALT
Hardware Acceleration > Public Key Cryptography (PKC) > RSA	MCU Specific Options		Defines MBEDTLS_RSA_ALT.
Hardware Acceleration > Public Key Cryptography (PKC) > RSA 3072 Verify	MCU Specific Options		Enables RSA 3072 Verify.
Hardware Acceleration	MCU Specific Options		Enables RSA 4096

> Public Key Cryptography (PKC) > RSA 4096 Verify			Verify.
Hardware Acceleration > TRNG	Enabled	Enabled	Defines MBEDTLS_ENTROPY_HARDWARE_ALT.
Hardware Acceleration > Secure Crypto Engine Initialization	Enabled	Enabled	MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT
Platform > Alternate > MBEDTLS_PLATFORM_EXIT_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_EXIT_ALT
Platform > Alternate > MBEDTLS_PLATFORM_TIME_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_TIME_ALT
Platform > Alternate > MBEDTLS_PLATFORM_FPRINTF_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_FPRINTF_ALT
Platform > Alternate > MBEDTLS_PLATFORM_PRINTF_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_PRINTF_ALT
Platform > Alternate > MBEDTLS_PLATFORM_SNPRINTF_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_SNPRINTF_ALT
Platform > Alternate > MBEDTLS_PLATFORM_VSNPRINTF_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_VSNPRINTF_ALT
Platform > Alternate > MBEDTLS_PLATFORM_NV_SEED_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_NV_SEED_ALT
Platform > Alternate > MBEDTLS_PLATFORM_ZEROIZE_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_ZEROIZE_ALT
Platform > Alternate > MBEDTLS_PLATFORM_GMTIME_R_ALT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_GMTIME_R_ALT
Platform > MBEDTLS_HAVE_ASM	• Define • Undefine	Undefine	MBEDTLS_HAVE_ASM
Platform > MBEDTLS_NO_UDBL_DIVISION	• Define • Undefine	Undefine	MBEDTLS_NO_UDBL_DIVISION
Platform > MBEDTLS_NO_64BIT_MULTIPLICATION	• Define • Undefine	Undefine	MBEDTLS_NO_64BIT_MULTIPLICATION
Platform > MBEDTLS_HAVE_SSE2	• Define • Undefine	Undefine	MBEDTLS_HAVE_SSE2
Platform >	• Define	Undefine	MBEDTLS_HAVE_TIME

MBEDTLS_HAVE_TIME	• Undefine		
Platform > MBEDTLS_HAVE_TIME_DATE	• Define • Undefine	Undefine	MBEDTLS_HAVE_TIME_DATE
Platform > MBEDTLS_PLATFORM_MEMORY	• Define • Undefine	Define	MBEDTLS_PLATFORM_MEMORY
Platform > MBEDTLS_PLATFORM_NO_STD_FUNCTIONS	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_NO_STD_FUNCTIONS
Platform > MBEDTLS_TIMING_ALT	• Define • Undefine	Undefine	MBEDTLS_TIMING_ALT
Platform > MBEDTLS_NO_PLATFORM_ENTROPY	• Define • Undefine	Define	MBEDTLS_NO_PLATFORM_ENTROPY
Platform > MBEDTLS_ENTROPY_C	• Define • Undefine	Define	MBEDTLS_ENTROPY_C
Platform > MBEDTLS_PLATFORM_C	• Define • Undefine	Define	MBEDTLS_PLATFORM_C
Platform > MBEDTLS_PLATFORM_STD_CALLOC	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_CALLOC
Platform > MBEDTLS_PLATFORM_STD_CALLOC value	Manual Entry	calloc	MBEDTLS_PLATFORM_STD_CALLOC value
Platform > MBEDTLS_PLATFORM_STD_FREE	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_FREE
Platform > MBEDTLS_PLATFORM_STD_FREE value	Manual Entry	free	MBEDTLS_PLATFORM_STD_FREE value
Platform > MBEDTLS_PLATFORM_STD_EXIT	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_EXIT
Platform > MBEDTLS_PLATFORM_STD_EXIT value	Manual Entry	exit	MBEDTLS_PLATFORM_STD_EXIT value
Platform > MBEDTLS_PLATFORM_STD_TIME	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_TIME
Platform > MBEDTLS_PLATFORM_STD_TIME value	Manual Entry	time	MBEDTLS_PLATFORM_STD_TIME value
Platform > MBEDTLS_PLATFORM_STD_FPRINTF	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_FPRINTF
Platform > MBEDTLS_PLATFORM_STD_FPRINTF value	Manual Entry	fprintf	MBEDTLS_PLATFORM_STD_FPRINTF value
Platform > MBEDTLS_PLATFORM_STD_PRINTF	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_PRINTF

Platform > MBEDTLS_PLATFORM_STD_PRINTF value	Manual Entry	printf	MBEDTLS_PLATFORM_STD_PRINTF value
Platform > MBEDTLS_PLATFORM_STD_SNPRINTF	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_SNPRINTF
Platform > MBEDTLS_PLATFORM_STD_SNPRINTF value	Manual Entry	snprintf	MBEDTLS_PLATFORM_STD_SNPRINTF value
Platform > MBEDTLS_PLATFORM_STD_EXIT_SUCCESS	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_EXIT_SUCCESS
Platform > MBEDTLS_PLATFORM_STD_EXIT_SUCCESS value	Manual Entry	0	MBEDTLS_PLATFORM_STD_EXIT_SUCCESS value
Platform > MBEDTLS_PLATFORM_STD_EXIT_FAILURE	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_EXIT_FAILURE
Platform > MBEDTLS_PLATFORM_STD_EXIT_FAILURE value	Manual Entry	1	MBEDTLS_PLATFORM_STD_EXIT_FAILURE value
Platform > MBEDTLS_PLATFORM_STD_NV_SEED_READ	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_NV_SEED_READ
Platform > MBEDTLS_PLATFORM_STD_NV_SEED_READ value	Manual Entry	mbedtls_platform_std_nv_seed_read	MBEDTLS_PLATFORM_STD_NV_SEED_READ value
Platform > MBEDTLS_PLATFORM_STD_NV_SEED_WRITE	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_NV_SEED_WRITE
Platform > MBEDTLS_PLATFORM_STD_NV_SEED_WRITE value	Manual Entry	mbedtls_platform_std_nv_seed_write	MBEDTLS_PLATFORM_STD_NV_SEED_WRITE value
Platform > MBEDTLS_PLATFORM_STD_NV_SEED_FILE	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_STD_NV_SEED_FILE
Platform > MBEDTLS_PLATFORM_STD_NV_SEED_FILE value	Manual Entry		MBEDTLS_PLATFORM_STD_NV_SEED_FILE value
Platform > MBEDTLS_PLATFORM_CALLOC_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_CALLOC_MACRO
Platform > MBEDTLS_PLATFORM_CALLOC_MACRO value	Manual Entry	calloc	MBEDTLS_PLATFORM_CALLOC_MACRO value

Platform > MBEDTLS_PLATFORM_FREE_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_FREE_MACRO
Platform > MBEDTLS_PLATFORM_FREE_MACRO value	Manual Entry	free	MBEDTLS_PLATFORM_FREE_MACRO value
Platform > MBEDTLS_PLATFORM_EXIT_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_EXIT_MACRO
Platform > MBEDTLS_PLATFORM_EXIT_MACRO value	Manual Entry	exit	MBEDTLS_PLATFORM_EXIT_MACRO value
Platform > MBEDTLS_PLATFORM_TIME_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_TIME_MACRO
Platform > MBEDTLS_PLATFORM_TIME_MACRO value	Manual Entry	time	MBEDTLS_PLATFORM_TIME_MACRO value
Platform > MBEDTLS_PLATFORM_TIME_TYPE_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_TIME_TYPE_MACRO
Platform > MBEDTLS_PLATFORM_TIME_TYPE_MACRO value	Manual Entry	time_t	MBEDTLS_PLATFORM_TIME_TYPE_MACRO value
Platform > MBEDTLS_PLATFORM_FPRINTF_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_FPRINTF_MACRO
Platform > MBEDTLS_PLATFORM_FPRINTF_MACRO value	Manual Entry	fprintf	MBEDTLS_PLATFORM_FPRINTF_MACRO value
Platform > MBEDTLS_PLATFORM_PRINTF_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_PRINTF_MACRO
Platform > MBEDTLS_PLATFORM_PRINTF_MACRO value	Manual Entry	printf	MBEDTLS_PLATFORM_PRINTF_MACRO value
Platform > MBEDTLS_PLATFORM_SNPRINTF_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_SNPRINTF_MACRO
Platform > MBEDTLS_PLATFORM_SNPRINTF_MACRO value	Manual Entry	snprintf	MBEDTLS_PLATFORM_SNPRINTF_MACRO value
Platform > MBEDTLS_PLATFORM_VSNPRINTF_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_VSNPRINTF_MACRO
Platform > MBEDTLS_PLATFORM_VSNPRINTF_	Manual Entry	vsnprintf	MBEDTLS_PLATFORM_VSNPRINTF_MACRO

MACRO value			value
Platform > MBEDTLS_PLATFORM_NV_SEED_READ_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_NV_SEED_READ_MACRO
Platform > MBEDTLS_PLATFORM_NV_SEED_READ_MACRO value	Manual Entry	mbedtls_platform_std_nv_seed_read	MBEDTLS_PLATFORM_NV_SEED_READ_MACRO value
Platform > MBEDTLS_PARAM_FAILED	• Define • Undefine	Undefine	MBEDTLS_PARAM_FAILED
Platform > MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO	• Define • Undefine	Undefine	MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO
Platform > MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO value	Manual Entry	mbedtls_platform_std_nv_seed_write	MBEDTLS_PLATFORM_NV_SEED_WRITE_MACRO value
General > PSA_CRYPTO_SECURE	• Define • Undefine	Undefine	PSA_CRYPTO_SECURE
General > MBEDTLS_DEPRECATED_WARNING	• Define • Undefine	Undefine	MBEDTLS_DEPRECATED_WARNING
General > MBEDTLS_DEPRECATED_REMOVED	• Define • Undefine	Define	MBEDTLS_DEPRECATED_REMOVED
General > MBEDTLS_CHECK_PARAMS	• Define • Undefine	Define	MBEDTLS_CHECK_PARAMS
General > MBEDTLS_CHECK_PARAMS_ASSERT	• Define • Undefine	Undefine	MBEDTLS_CHECK_PARAMS_ASSERT
General > MBEDTLS_ERROR_STRERROR_DUMMY	• Define • Undefine	Define	MBEDTLS_ERROR_STRERROR_DUMMY
General > MBEDTLS_MEMORY_DEBUG	• Define • Undefine	Undefine	MBEDTLS_MEMORY_DEBUG
General > MBEDTLS_MEMORY_BACKTRACE	• Define • Undefine	Undefine	MBEDTLS_MEMORY_BACKTRACE
General > MBEDTLS_PSA_CRYPTO_SPM	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_SPM
General > MBEDTLS_SELF_TEST	• Define • Undefine	Undefine	MBEDTLS_SELF_TEST
General > MBEDTLS_THREADING_ALT	• Define • Undefine	Define	MBEDTLS_THREADING_ALT
General > MBEDTLS_THREADING_PTHREAD	• Define • Undefine	Undefine	MBEDTLS_THREADING_PTHREAD
General > MBEDTLS_USE_PSA_CRYPTO	Undefine	Undefine	MBEDTLS_USE_PSA_CRYPTO
General > MBEDTLS_V	• Define	Define	MBEDTLS_VERSION_FE

ERSION_FEATURES	• Undefine		ATURES
General > MBEDTLS_ERROR_C	• Define • Undefine	Define	MBEDTLS_ERROR_C
General > MBEDTLS_MEMORY_BUFFER_ALLOC_C	• Define • Undefine	Undefine	MBEDTLS_MEMORY_BUFFER_ALLOC_C
General > MBEDTLS_PSA_CRYPTO_C	• Define • Undefine	Define	MBEDTLS_PSA_CRYPTO_C
General > MBEDTLS_PSA_CRYPTO_SE_C	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_SE_C
General > MBEDTLS_THREADING_C	• Define • Undefine	Define	MBEDTLS_THREADING_C
General > MBEDTLS_TIMING_C	• Define • Undefine	Undefine	MBEDTLS_TIMING_C
General > MBEDTLS_VERSION_C	• Define • Undefine	Define	MBEDTLS_VERSION_C
General > MBEDTLS_MEMORY_ALIGN_MULTIPLE	• Define • Undefine	Undefine	MBEDTLS_MEMORY_ALIGN_MULTIPLE
General > MBEDTLS_MEMORY_ALIGN_MULTIPLE value	Manual Entry	4	MBEDTLS_MEMORY_ALIGN_MULTIPLE value
Cipher > Alternate > MBEDTLS_ARC4_ALT	• Define • Undefine	Undefine	MBEDTLS_ARC4_ALT
Cipher > Alternate > MBEDTLS_ARIA_ALT	• Define • Undefine	Undefine	MBEDTLS_ARIA_ALT
Cipher > Alternate > MBEDTLS_BLOWFISH_ALT	• Define • Undefine	Undefine	MBEDTLS_BLOWFISH_ALT
Cipher > Alternate > MBEDTLS_CAMELLIA_ALT	• Define • Undefine	Undefine	MBEDTLS_CAMELLIA_ALT
Cipher > Alternate > MBEDTLS_CCM_ALT	• Define • Undefine	Undefine	MBEDTLS_CCM_ALT
Cipher > Alternate > MBEDTLS_CHACHA20_ALT	• Define • Undefine	Undefine	MBEDTLS_CHACHA20_ALT
Cipher > Alternate > MBEDTLS_CHACHAPOLY_ALT	• Define • Undefine	Undefine	MBEDTLS_CHACHAPOLY_ALT
Cipher > Alternate > MBEDTLS_CMAC_ALT	• Define • Undefine	Undefine	MBEDTLS_CMAC_ALT
Cipher > Alternate > MBEDTLS_DES_ALT	• Define • Undefine	Undefine	MBEDTLS_DES_ALT

Cipher > Alternate > MBEDTLS_GCM_ALT	MCU Specific Options		MBEDTLS_GCM_ALT
Cipher > Alternate > MBEDTLS_NIST_KW_ALT	• Define • Undefine	Undefine	MBEDTLS_NIST_KW_ALT
Cipher > Alternate > MBEDTLS_XTEA_ALT	• Define • Undefine	Undefine	MBEDTLS_XTEA_ALT
Cipher > Alternate > MBEDTLS_DES_SETKEY_ALT	• Define • Undefine	Undefine	MBEDTLS_DES_SETKEY_ALT
Cipher > Alternate > MBEDTLS_DES_CRYPT_ECB_ALT	• Define • Undefine	Undefine	MBEDTLS_DES_CRYPT_ECB_ALT
Cipher > Alternate > MBEDTLS_DES3_CRYPT_ECB_ALT	• Define • Undefine	Undefine	MBEDTLS_DES3_CRYPT_ECB_ALT
Cipher > AES > MBEDTLS_AES_ROM_TABLES	• Define • Undefine	Undefine	MBEDTLS_AES_ROM_TABLES
Cipher > AES > MBEDTLS_AES_FEWER_TABLES	• Define • Undefine	Undefine	MBEDTLS_AES_FEWER_TABLES
Cipher > MBEDTLS_CAMELLIA_SMALL_MEMORY	• Define • Undefine	Undefine	MBEDTLS_CAMELLIA_SMALL_MEMORY
Cipher > MBEDTLS_CIPHER_MODE_CBC	• Define • Undefine	Define	MBEDTLS_CIPHER_MODE_CBC
Cipher > MBEDTLS_CIPHER_MODE_CFB	• Define • Undefine	Define	MBEDTLS_CIPHER_MODE_CFB
Cipher > MBEDTLS_CIPHER_MODE_CTR	• Define • Undefine	Define	MBEDTLS_CIPHER_MODE_CTR
Cipher > MBEDTLS_CIPHER_MODE_OFB	• Define • Undefine	Undefine	MBEDTLS_CIPHER_MODE_OFB
Cipher > MBEDTLS_CIPHER_MODE_XTS	• Define • Undefine	Undefine	MBEDTLS_CIPHER_MODE_XTS
Cipher > MBEDTLS_CIPHER_NULL_CIPHER	• Define • Undefine	Undefine	MBEDTLS_CIPHER_NULL_CIPHER
Cipher > MBEDTLS_CIPHER_PADDING_PKCS7	• Define • Undefine	Define	MBEDTLS_CIPHER_PADDING_PKCS7
Cipher > MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS	• Define • Undefine	Define	MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS
Cipher > MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN	• Define • Undefine	Define	MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN

Cipher > MBEDTLS_CIPHER_PADDING_ZEROS	• Define • Undefine	Define	MBEDTLS_CIPHER_PADDING_ZEROS
Cipher > MBEDTLS_AES_C	Define	Define	MBEDTLS_AES_C
Cipher > MBEDTLS_ARC4_C	• Define • Undefine	Undefine	MBEDTLS_ARC4_C
Cipher > MBEDTLS_BLOWFISH_C	• Define • Undefine	Undefine	MBEDTLS_BLOWFISH_C
Cipher > MBEDTLS_CAMELLIA_C	• Define • Undefine	Undefine	MBEDTLS_CAMELLIA_C
Cipher > MBEDTLS_ARIA_C	• Define • Undefine	Undefine	MBEDTLS_ARIA_C
Cipher > MBEDTLS_CCM_C	• Define • Undefine	Define	MBEDTLS_CCM_C
Cipher > MBEDTLS_CHACHA20_C	• Define • Undefine	Undefine	MBEDTLS_CHACHA20_C
Cipher > MBEDTLS_CHACHAPOLY_C	• Define • Undefine	Undefine	MBEDTLS_CHACHAPOLY_C
Cipher > MBEDTLS_CIPHER_C	• Define • Undefine	Define	MBEDTLS_CIPHER_C
Cipher > MBEDTLS_DES_C	• Define • Undefine	Undefine	MBEDTLS_DES_C
Cipher > MBEDTLS_GCM_C	• Define • Undefine	Define	MBEDTLS_GCM_C
Cipher > MBEDTLS_NIST_KW_C	• Define • Undefine	Undefine	MBEDTLS_NIST_KW_C
Cipher > MBEDTLS_XTEA_C	• Define • Undefine	Undefine	MBEDTLS_XTEA_C
Public Key Cryptography (PKC) > DHM > Alternate > MBEDTLS_DHM_ALT	• Define • Undefine	Undefine	MBEDTLS_DHM_ALT
Public Key Cryptography (PKC) > DHM > MBEDTLS_DHM_C	• Define • Undefine	Undefine	MBEDTLS_DHM_C
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECJPAKE_ALT	• Define • Undefine	Undefine	MBEDTLS_ECJPAKE_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBE	• Define • Undefine	Undefine	MBEDTLS_ECDSA_GENKEY_ALT

DTLS_ECDSA_GENKEY_ALT			
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_INTERNAL_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_INTERNAL_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_RANDOMIZE_JAC_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_RANDOMIZE_JAC_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_ADD_MIXED_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_ADD_MIXED_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_DOUBLE_JAC_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_DOUBLE_JAC_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_NORMALIZE_JAC_MANY_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_NORMALIZE_JAC_MANY_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_NORMALIZE_JAC_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_NORMALIZE_JAC_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_DOUBLE_ADD_MXZ_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_DOUBLE_ADD_MXZ_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_RANDOMIZE_MXZ_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_RANDOMIZE_MXZ_ALT
Public Key Cryptography (PKC) > ECC > Alternate > MBEDTLS_ECP_NORMALIZE_MXZ_ALT	• Define • Undefine	Undefine	MBEDTLS_ECP_NORMALIZE_MXZ_ALT
Public Key Cryptography (PKC) >	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP192R1_ENABLED

ECC > Curves > MBEDTLS_ECP_DP_SECP192R1_ENABLED			
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP224R1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP224R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP256R1_ENABLED	• Define • Undefine	Define	MBEDTLS_ECP_DP_SECP256R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP384R1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP384R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP521R1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP521R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP192K1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP192K1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP224K1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP224K1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_SECP256K1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_SECP256K1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_BP256R1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_BP256R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_BP384R1_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_BP384R1_ENABLED
Public Key	• Define	Undefine	MBEDTLS_ECP_DP_BP5

Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_BP512R1_ENABLED	• Undefine		12R1_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_CURVE25519_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_CURVE25519_ENABLED
Public Key Cryptography (PKC) > ECC > Curves > MBEDTLS_ECP_DP_CURVE448_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECP_DP_CURVE448_ENABLED
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_GEN_PUBLIC_ALT	• Define • Undefine	Undefine	MBEDTLS_ECDH_GEN_PUBLIC_ALT
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_COMPUTE_SHARED_ALT	• Define • Undefine	Undefine	MBEDTLS_ECDH_COMPUTE_SHARED_ALT
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_NIST_OPTIM	• Define • Undefine	Undefine	MBEDTLS_ECP_NIST_OPTIM
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_RESTARTABLE	• Define • Undefine	Undefine	MBEDTLS_ECP_RESTARTABLE
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_LEGACY_CONTEXT	• Define • Undefine	Undefine	MBEDTLS_ECDH_LEGACY_CONTEXT
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDSA_DETERMINISTIC	• Define • Undefine	Undefine	MBEDTLS_ECDSA_DETERMINISTIC
Public Key Cryptography (PKC) > ECC > MBEDTLS_PK_PARSE_EC_EXTENDED	• Define • Undefine	Undefine	MBEDTLS_PK_PARSE_EC_EXTENDED
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_C	• Define • Undefine	Undefine	MBEDTLS_ECDH_C
Public Key	• Define	Define	MBEDTLS_ECDSA_C

Cryptography (PKC) > ECC > MBEDTLS_ECDSA_C	• Undefine		
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_C	• Define • Undefine	Define	MBEDTLS_ECP_C
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECJPAKE_C	• Define • Undefine	Undefine	MBEDTLS_ECJPAKE_C
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_MAX_BITS	• Define • Undefine	Undefine	MBEDTLS_ECP_MAX_BITS
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_MAX_BITS value	Manual Entry	521	MBEDTLS_ECP_MAX_BITS value
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_WINDOW_SIZE	• Define • Undefine	Undefine	MBEDTLS_ECP_WINDOW_SIZE
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_WINDOW_SIZE value	Manual Entry	6	MBEDTLS_ECP_WINDOW_SIZE value
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_FIXED_POINT_OPTIM	• Define • Undefine	Undefine	MBEDTLS_ECP_FIXED_POINT_OPTIM
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECP_FIXED_POINT_OPTIM value	Manual Entry	1	MBEDTLS_ECP_FIXED_POINT_OPTIM value
Public Key Cryptography (PKC) > ECC > MBEDTLS_ECDH_VARIANT_EVEREST_ENABLED	• Define • Undefine	Undefine	MBEDTLS_ECDH_VARIANT_EVEREST_ENABLED
Public Key Cryptography (PKC) > RSA > MBEDTLS_PK_RSA_ALT_SUPPORT	• Define • Undefine	Undefine	MBEDTLS_PK_RSA_ALT_SUPPORT
Public Key Cryptography (PKC) > RSA > MBEDTLS_RSA_NO_CRT	• Define • Undefine	Define	MBEDTLS_RSA_NO_CRT

Public Key Cryptography (PKC) > RSA > MBEDTLS_RSA_C	• Define • Undefine	Define	MBEDTLS_RSA_C
Public Key Cryptography (PKC) > MBEDTLS_GENPRIME	• Define • Undefine	Define	MBEDTLS_GENPRIME
Public Key Cryptography (PKC) > MBEDTLS_PKCS1_V15	• Define • Undefine	Define	MBEDTLS_PKCS1_V15
Public Key Cryptography (PKC) > MBEDTLS_PKCS1_V21	• Define • Undefine	Define	MBEDTLS_PKCS1_V21
Public Key Cryptography (PKC) > MBEDTLS_ASN1_PARSE_C	• Define • Undefine	Define	MBEDTLS_ASN1_PARSE_C
Public Key Cryptography (PKC) > MBEDTLS_ASN1_WRITE_C	• Define • Undefine	Define	MBEDTLS_ASN1_WRITE_C
Public Key Cryptography (PKC) > MBEDTLS_BASE64_C	• Define • Undefine	Define	MBEDTLS_BASE64_C
Public Key Cryptography (PKC) > MBEDTLS_BIGNUM_C	• Define • Undefine	Define	MBEDTLS_BIGNUM_C
Public Key Cryptography (PKC) > MBEDTLS_OID_C	• Define • Undefine	Define	MBEDTLS_OID_C
Public Key Cryptography (PKC) > MBEDTLS_PEM_PARSE_C	• Define • Undefine	Define	MBEDTLS_PEM_PARSE_C
Public Key Cryptography (PKC) > MBEDTLS_PEM_WRITE_C	• Define • Undefine	Define	MBEDTLS_PEM_WRITE_C
Public Key Cryptography (PKC) > MBEDTLS_PK_C	• Define • Undefine	Define	MBEDTLS_PK_C
Public Key Cryptography (PKC) > MBEDTLS_PK_PARSE_C	• Define • Undefine	Define	MBEDTLS_PK_PARSE_C
Public Key Cryptography (PKC) > MBEDTLS_PK_WRITE_C	• Define • Undefine	Define	MBEDTLS_PK_WRITE_C

Public Key Cryptography (PKC) > MBEDTLS_PKCS5_C	• Define • Undefine	Define	MBEDTLS_PKCS5_C
Public Key Cryptography (PKC) > MBEDTLS_PKCS12_C	• Define • Undefine	Define	MBEDTLS_PKCS12_C
Public Key Cryptography (PKC) > MBEDTLS_MPI_WINDOW_SIZE	• Define • Undefine	Undefine	MBEDTLS_MPI_WINDOW_SIZE
Public Key Cryptography (PKC) > MBEDTLS_MPI_WINDOW_SIZE value	Manual Entry	6	MBEDTLS_MPI_WINDOW_SIZE value
Public Key Cryptography (PKC) > MBEDTLS_MPI_MAX_SIZE	• Define • Undefine	Undefine	MBEDTLS_MPI_MAX_SIZE
Public Key Cryptography (PKC) > MBEDTLS_MPI_MAX_SIZE value	Manual Entry	1024	MBEDTLS_MPI_MAX_SIZE value
Hash > Alternate > MBEDTLS_MD2_ALT	• Define • Undefine	Undefine	MBEDTLS_MD2_ALT
Hash > Alternate > MBEDTLS_MD4_ALT	• Define • Undefine	Undefine	MBEDTLS_MD4_ALT
Hash > Alternate > MBEDTLS_MD5_ALT	• Define • Undefine	Undefine	MBEDTLS_MD5_ALT
Hash > Alternate > MBEDTLS_RIPEMD160_ALT	• Define • Undefine	Undefine	MBEDTLS_RIPEMD160_ALT
Hash > Alternate > MBEDTLS_SHA1_ALT	• Define • Undefine	Undefine	MBEDTLS_SHA1_ALT
Hash > Alternate > MBEDTLS_SHA512_ALT	• Define • Undefine	Undefine	MBEDTLS_SHA512_ALT
Hash > Alternate > MBEDTLS_MD2_PROCESS_ALT	• Define • Undefine	Undefine	MBEDTLS_MD2_PROCESS_ALT
Hash > Alternate > MBEDTLS_MD4_PROCESS_ALT	• Define • Undefine	Undefine	MBEDTLS_MD4_PROCESS_ALT
Hash > Alternate > MBEDTLS_MD5_PROCESS_ALT	• Define • Undefine	Undefine	MBEDTLS_MD5_PROCESS_ALT
Hash > Alternate > MBEDTLS_RIPEMD160_PR	• Define • Undefine	Undefine	MBEDTLS_RIPEMD160_PROCESS_ALT

OCESS_ALT			
Hash > Alternate > MBEDTLS_SHA1_PROCESS_ALT	• Define • Undefine	Undefine	MBEDTLS_SHA1_PROCESS_ALT
Hash > Alternate > MBEDTLS_SHA512_PROCESS_ALT	• Define • Undefine	Undefine	MBEDTLS_SHA512_PROCESS_ALT
Hash > MBEDTLS_SHA256_SMALLER	• Define • Undefine	Undefine	MBEDTLS_SHA256_SMALLER
Hash > MBEDTLS_SHA512_SMALLER	• Define • Undefine	Undefine	MBEDTLS_SHA512_SMALLER
Hash > MBEDTLS_SHA512_NO_SHA384	• Define • Undefine	Undefine	MBEDTLS_SHA512_NO_SHA384
Hash > MBEDTLS_MD_C	• Define • Undefine	Define	MBEDTLS_MD_C
Hash > MBEDTLS_MD2_C	• Define • Undefine	Undefine	MBEDTLS_MD2_C
Hash > MBEDTLS_MD4_C	• Define • Undefine	Undefine	MBEDTLS_MD4_C
Hash > MBEDTLS_MD5_C	• Define • Undefine	Define	MBEDTLS_MD5_C
Hash > MBEDTLS_RIPEMD160_C	• Define • Undefine	Define	MBEDTLS_RIPEMD160_C
Hash > MBEDTLS_SHA1_C	• Define • Undefine	Define	MBEDTLS_SHA1_C
Hash > MBEDTLS_SHA256_C	• Define • Undefine	Define	MBEDTLS_SHA256_C
Hash > MBEDTLS_SHA512_C	• Define • Undefine	Undefine	MBEDTLS_SHA512_C
Message Authentication Code (MAC) > Alternate > MBEDTLS_POLY1305_ALT	• Define • Undefine	Undefine	MBEDTLS_POLY1305_ALT
Message Authentication Code (MAC) > MBEDTLS_CMAC_C	• Define • Undefine	Undefine	MBEDTLS_CMAC_C
Message Authentication Code (MAC) > MBEDTLS_HKDF_C	• Define • Undefine	Define	MBEDTLS_HKDF_C
Message Authentication Code	• Define • Undefine	Undefine	MBEDTLS_HMAC_DRBG_C

(MAC) > MBEDTLS_HMAC_DRBG_C			
Message Authentication Code (MAC) > MBEDTLS_POLY1305_C	• Define • Undefine	Undefine	MBEDTLS_POLY1305_C
RNG > MBEDTLS_TEST_NULL_ENTROPY	• Define • Undefine	Undefine	MBEDTLS_TEST_NULL_ENTROPY
RNG > MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES	• Define • Undefine	Undefine	MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES
RNG > MBEDTLS_ENTROPY_FORCE_SHA256	• Define • Undefine	Undefine	MBEDTLS_ENTROPY_FORCE_SHA256
RNG > MBEDTLS_ENTROPY_NV_SEED	• Define • Undefine	Undefine	MBEDTLS_ENTROPY_NV_SEED
RNG > MBEDTLS_PSA_INJECT_ENTROPY	• Define • Undefine	Undefine	MBEDTLS_PSA_INJECT_ENTROPY
RNG > MBEDTLS_CTR_DRBG_C	• Define • Undefine	Define	MBEDTLS_CTR_DRBG_C
RNG > MBEDTLS_CTR_DRBG_C_ALT	Define	Define	MBEDTLS_CTR_DRBG_C_ALT
RNG > MBEDTLS_HAVEGE_C	• Define • Undefine	Undefine	MBEDTLS_HAVEGE_C
RNG > MBEDTLS_CTR_DRBG_ENTROPY_LEN	• Define • Undefine	Undefine	RNG|MBEDTLS_CTR_DRBG_ENTROPY_LEN
RNG > MBEDTLS_CTR_DRBG_ENTROPY_LEN value	Manual Entry	48	RNG value|MBEDTLS_CTR_DRBG_ENTROPY_LEN
RNG > MBEDTLS_CTR_DRBG_RESEED_INTERVAL	• Define • Undefine	Undefine	RNG|MBEDTLS_CTR_DRBG_RESEED_INTERVAL
RNG > MBEDTLS_CTR_DRBG_RESEED_INTERVAL value	Manual Entry	10000	RNG value|MBEDTLS_CTR_DRBG_RESEED_INTERVAL
RNG > MBEDTLS_CTR_DRBG_MAX_INPUT	• Define • Undefine	Undefine	MBEDTLS_CTR_DRBG_MAX_INPUT
RNG > MBEDTLS_CTR_DRBG_MAX_INPUT value	Manual Entry	256	MBEDTLS_CTR_DRBG_MAX_INPUT value
RNG > MBEDTLS_CTR_DRBG_MAX_REQUEST	• Define • Undefine	Undefine	MBEDTLS_CTR_DRBG_MAX_REQUEST
RNG > MBEDTLS_CTR_DRBG_MAX_REQUEST value	Manual Entry	1024	MBEDTLS_CTR_DRBG_MAX_REQUEST value

RNG > MBEDTLS_CTR_DRBG_MAX_SEED_INPUT	• Define • Undefine	Undefine	MBEDTLS_CTR_DRBG_MAX_SEED_INPUT
RNG > MBEDTLS_CTR_DRBG_MAX_SEED_INPUT value	Manual Entry	384	MBEDTLS_CTR_DRBG_MAX_SEED_INPUT value
RNG > MBEDTLS_CTR_DRBG_USE_128_BIT_KEY	• Define • Undefine	Undefine	MBEDTLS_CTR_DRBG_USE_128_BIT_KEY
RNG > MBEDTLS_HMAC_DRBG_RESEED_INTERVAL	• Define • Undefine	Undefine	MBEDTLS_HMAC_DRBG_RESEED_INTERVAL
RNG > MBEDTLS_HMAC_DRBG_RESEED_INTERVAL value	Manual Entry	10000	MBEDTLS_HMAC_DRBG_RESEED_INTERVAL value
RNG > MBEDTLS_HMAC_DRBG_MAX_INPUT	• Define • Undefine	Undefine	MBEDTLS_HMAC_DRBG_MAX_INPUT
RNG > MBEDTLS_HMAC_DRBG_MAX_INPUT value	Manual Entry	256	MBEDTLS_HMAC_DRBG_MAX_INPUT value
RNG > MBEDTLS_HMAC_DRBG_MAX_REQUEST	• Define • Undefine	Undefine	MBEDTLS_HMAC_DRBG_MAX_REQUEST
RNG > MBEDTLS_HMAC_DRBG_MAX_REQUEST value	Manual Entry	1024	MBEDTLS_HMAC_DRBG_MAX_REQUEST value
RNG > MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT	• Define • Undefine	Undefine	MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT
RNG > MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT value	Manual Entry	384	MBEDTLS_HMAC_DRBG_MAX_SEED_INPUT value
RNG > MBEDTLS_ENTROPY_MAX_SOURCES	• Define • Undefine	Undefine	MBEDTLS_ENTROPY_MAX_SOURCES
RNG > MBEDTLS_ENTROPY_MAX_SOURCES value	Manual Entry	20	MBEDTLS_ENTROPY_MAX_SOURCES value
RNG > MBEDTLS_ENTROPY_MAX_GATHER	• Define • Undefine	Undefine	MBEDTLS_ENTROPY_MAX_GATHER
RNG > MBEDTLS_ENTROPY_MAX_GATHER value	Manual Entry	128	MBEDTLS_ENTROPY_MAX_GATHER value
RNG > MBEDTLS_ENTROPY_MIN_HARDWARE	• Define • Undefine	Undefine	MBEDTLS_ENTROPY_MIN_HARDWARE
RNG > MBEDTLS_ENTROPY_MIN_HARDWARE	Manual Entry	32	MBEDTLS_ENTROPY_MIN_HARDWARE value

value			
Storage > MBEDTLS_FS_IO	• Define • Undefine	Undefine	MBEDTLS_FS_IO
Storage > MBEDTLS_PSA_CRYPTO_KEY_ID_ENCODES_OWNER	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_KEY_ID_ENCODES_OWNER
Storage > MBEDTLS_PSA_CRYPTO_STORAGE_C	• Define • Undefine	Undefine	MBEDTLS_PSA_CRYPTO_STORAGE_C
Storage > MBEDTLS_PSA_ITS_FILE_C	• Define • Undefine	Undefine	MBEDTLS_PSA_ITS_FILE_C

SHA256 Configuration

To enable hardware acceleration for the SHA256/224 calculation, the macro MBEDTLS_SHA256_ALT and MBEDTLS_SHA256_PROCESS_ALT must be defined in the configuration file. By default SHA256 is enabled. SHA256 can be disabled, but SHA512 then needs to be enabled (software version) because the PSA implementation uses it for the entropy accumulator. This can be done using the RA Configuration editor.

AES Configuration

To enable hardware acceleration for the AES128/256 operation, the macro MBEDTLS_AES_SETKEY_ENC_ALT, MBEDTLS_AES_SETKEY_DEC_ALT, MBEDTLS_AES_ENCRYPT_ALT and MBEDTLS_AES_DECRYPT_ALT must be defined in the configuration file. By default AES is enabled. AES cannot be disabled because the PSA implementation requires it for the CTR_DRBG random number generator. This can be done using the RA Configuration editor.

ECC Configuration

To enable hardware acceleration for the ECC Key Generation operation, the macro MBEDTLS_ECP_ALT must be defined in the configuration file. For ECDSA, the macros MBEDTLS_ECDSA_SIGN_ALT and MBEDTLS_ECDSA_VERIFY_ALT must be defined. By default ECC, ECDSA and ECDHE are enabled. To disable ECC, undefine MBEDTLS_ECP_C, MBEDTLS_ECDSA_C and MBEDTLS_ECDH_C. This can be done using the RA Configuration editor.

RSA Configuration

To enable hardware acceleration for the RSA2048 operation, the macro MBEDTLS_RSA_ALT must be defined in the configuration file. By default RSA is enabled. To disable RSA, undefine MBEDTLS_RSA_C, MBEDTLS_PK_C, MBEDTLS_PK_PARSE_C, MBEDTLS_PK_WRITE_C. This can be done using the RA Configuration editor.

Wrapped Key Usage

To use the Secure Crypto Engine to generate and use wrapped keys, use PSA_KEY_TYPE_AES_WRAPPED or PSA_KEY_TYPE_ECC_KEY_PAIR_WRAPPED(curve) or PSA_KEY_TYPE_RSA_KEY_PAIR when setting the key type attribute. Setting the key's type attribute using this value will cause the SCE to use wrapped key mode for all operations related to that key. The user can use the export functionality to save the wrapped keys to user ROM and import it later for usage. This mode requires that Wrapped Key functionality for the algorithm is enabled in the project configuration.

Note

On the SCE9 devices, only the RSA public key can be exported. A file system must be used to store the internally generated private key.

Persistent Key Storage

Persistent key storage can be enabled by defining MBEDTLS_FS_IO, MBEDTLS_PSA_CRYPTO_STORAGE_C, and MBEDTLS_PSA_ITS_FILE_C. The key lifetime must also be specifed as PSA_KEY_LIFETIME_PERSISTENT. A lower level storage module must be added in the RA Configuration editor and initialized in the code before generating persistent keys. Persistent storage supports the use of plaintext and vendor keys. Refer to the lower level storage module documentation for information on how it should be initialized. To generate a persistent key the key must be assigned a unique id prior to calling generate using the psa_set_key_id api.

```
if (PSA_KEY_LIFETIME_IS_PERSISTENT(lifetime))
   {
/* Set the id to a positive integer. */
      psa_set_key_id(&attributes, (psa_key_id_t) 5);
   }
```

Platform Configuration

To run the mbedCrypto implementation of the PSA Crypto API on the MCU, the macro MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT must be defined in the configuration file. This enables code that will initialize the SCE. Parameter checking (General|MBEDTLS_CHECK_PARAMS) is enabled by default. To reduce code size, disable parameter checking.

Random Number Configuration

To run the mbedCrypto implementation of the PSA Crypto API on the MCU, the macro MBEDTLS_ENTROPY_HARDWARE_ALT must be defined in the configuration file. This enables using the TRNG as an entropy source. None of the other cryptographic operations (even in software only mode) will work without this feature.

Usage Notes

Hardware Initialization

mbedtls_platform_setup() must be invoked before using the PSA Crypto API to ensure that the SCE peripheral is initialized.

Memory Usage

In general, depending on the mbedCrypto features being used a heap size of 0x1000 to 0x5000 bytes is required. The total allocated heap should be the **sum** of the heap requirements of the individual algorithms:

Algorithm	Required Heap (bytes)
SHA256/224	None

AES	0x200
Hardware ECC	0x400
Software ECC	0x1800
RSA	0x1500

A minimum stack of 0x1000 is required where the module is used. This is either the main stack in a bare metal application or the task stack of the task used for crypto operations.

Limitations

- Only little endian mode is supported.

SCE9 Usage

The SCE9 is used in Compatibility Mode for mbedCrypto acceleration. The crypto capabilities in this mode on the SCE9 are different which results in the below usage limitations with mbedCrypto:

- The module includes **both** wrapped and plaintext keys code irrespective of whether the application requires it.
- Plaintext key generation is not supported for RSA and ECC; only wrapped keys can be generated.
- If ECDH is used, only wrapped key will be generated on SCE9 and will not return an error even if the user context is somehow set for plain key. This may be relevant only if the psa_key_agreement() function with plaintext key on SCE9 is attempted.

Note

For a detailed description of the different SCE9 operating modes, refer to Application Note R11AN0498.

Using PSA Crypto with TrustZone

Unlike FSP drivers, PSA Crypto cannot be configured as Non-secure callable in the RA Configurator for a secure project. The reason for this is that in order to achieve the security objective of controlling access to protected keys, both the PSA Crypto code as well as the keys must be placed in the secure region. Since the PSA Crypto API requires access to the keys directly during initialization and later via a key handle, allowing non-secure code to use the API by making it Non-secure callable will require the keys to be stored in non-secure memory.

This section will provide a short explanation of how to add PSA Crypto to a secure project and have it usable by the non-secure project without exposing the keys. In this example the secure project will contain an RSA private key and the non-secure project is expected to be able to perform sign and verify operations using that key.

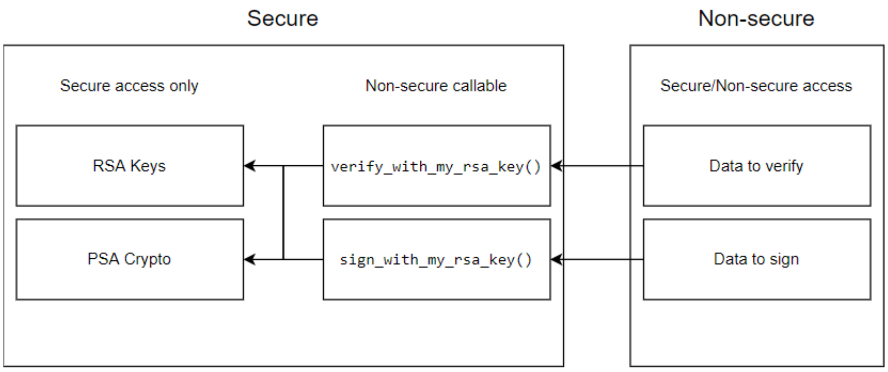


Figure 210: PSA Crypto Non-secure callable example

- Secure project
    - During secure project boot-up, psa_crypto_init() is called.
    - The RSA private key is programmed into secure flash either at the factory or by calling psa_generate_key() in persistent mode. Note that the data-flash area used by the LittleFS will have to be in the secure region if the key is generated as a persistent.
    - psa_import_key()/psa_open_key() are called with the resultant handle held in secure RAM.
    - The Non-secure callable section contains the following **user-defined** functions
    - verify_with_my_rsa_key(input_signature, input_hash, verification_result)
    - The implementation of this function in secure region will call psa_verify_hash() and return the result via verification_result.
    - sign_with_my_rsa_key(input_hash, output_signature)
    - The implementation of this function in secure region will call psa_sign_hash() and return the signature via output_signature.

- Non-secure project
    - Calls verify_with_my_rsa_key() to verify a signature. The implementation will use the public key that is present in the secure project.
    - Calls sign_with_my_rsa_key() to generate a signature. The implementation will use the private key that is present on the secure project.

  For more details on how to add user-code to the Non-secure callable region refer to the "Security Design with Arm TrustZone - IP Protection (R11AN0467EU0100)" Application Note.

Examples

Hash Example

This is an example on calculating the SHA256 hash using the PSA Crypto API.

```
const uint8_t NIST_SHA256ShortMsgLen200[] =
{
    0x2e, 0x7e, 0xa8, 0x4d, 0xa4, 0xbc, 0x4d, 0x7c, 0xfb, 0x46, 0x3e, 0x3f, 0x2c,
```

```
0x86, 0x47, 0x05,
    0x7a, 0xff, 0xf3, 0xfb, 0xec, 0xec, 0xa1, 0xd2, 00
};
const uint8_t NIST_SHA256ShortMsgLen200_expected[] =
{
    0x76, 0xe3, 0xac, 0xbc, 0x71, 0x88, 0x36, 0xf2, 0xdf, 0x8a, 0xd2, 0xd0, 0xd2,
0xd7, 0x6f, 0x0c,
    0xfa, 0x5f, 0xea, 0x09, 0x86, 0xbe, 0x91, 0x8f, 0x10, 0xbc, 0xee, 0x73, 0x0d,
0xf4, 0x41, 0xb9
};
void psa_crypto_sha256_example (void)
{
    psa_algorithm_t      alg               = PSA_ALG_SHA_256;
    psa_hash_operation_t operation         = {0};
 size_t               expected_hash_len = PSA_HASH_SIZE(alg);
    uint8_t             actual_hash[PSA_HASH_MAX_SIZE];
 size_t               actual_hash_len;
 mbedtls_platform_context ctx = {0};
 /* Setup the platform; initialize the SCE and the TRNG */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_hash_setup(&operation, alg))
    {
 /* Hash setup failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_hash_update(&operation, NIST_SHA256ShortMsgLen200,
sizeof(NIST_SHA256ShortMsgLen200)))
    {
 /* Hash calculation failed */
       debugger_break();
```

```
    }
 else if (PSA_SUCCESS != psa_hash_finish(&operation, &actual_hash[0], sizeof
(actual_hash), &actual_hash_len))
    {
 /* Reading calculated hash failed */
       debugger_break();
    }
 else if (0 != memcmp(&actual_hash[0], &NIST_SHA256ShortMsgLen200_expected[0],
actual_hash_len))
    {
 /* Hash compare of calculated value with expected value failed */
       debugger_break();
    }
 else if (0 != memcmp(&expected_hash_len, &actual_hash_len, sizeof
(expected_hash_len)))
    {
 /* Hash size compare of calculated value with expected value failed */
       debugger_break();
    }
 else
    {
 /* SHA256 calculation succeeded */
       debugger_break();
    }
 /* De-initialize the platform. This is currently a placeholder function which does
not do anything. */
 mbedtls_platform_teardown(&ctx);
}
```

AES Example

This is an example on using the PSA Crypto API to generate an AES256 key, encrypting and decrypting multi-block data and using PKCS7 padding.

```
static psa_status_t cipher_operation (psa_cipher_operation_t * operation,
```

```
 const uint8_t          * input,
 size_t                   input_size,
 size_t                   part_size,
                                    uint8_t                * output,
 size_t                   output_size,
 size_t                 * output_len)
{
    psa_status_t status;
 size_t       bytes_to_write = 0;
 size_t       bytes_written = 0;
 size_t       len            = 0;
    *output_len = 0;
 while (bytes_written != input_size)
    {
       bytes_to_write = (input_size - bytes_written > part_size ?
                         part_size :
                         input_size - bytes_written);
       status = psa_cipher_update(operation,
                                  input + bytes_written,
                                  bytes_to_write,
                                  output + *output_len,
                                  output_size - *output_len,
                                  &len);
 if (PSA_SUCCESS != status)
      {
 return status;
      }
       bytes_written += bytes_to_write;
       *output_len   += len;
    }
    status = psa_cipher_finish(operation, output + *output_len, output_size -
*output_len, &len);
 if (PSA_SUCCESS != status)
    {
```

```
 return status;
    }
    *output_len += len;
 return status;
}
void psa_crypto_aes256cbcmultipart_example (void)
{
 enum
    {
       block_size = PSA_BLOCK_CIPHER_BLOCK_SIZE(PSA_KEY_TYPE_AES),
       key_bits   = 256,
       input_size = 100,
       part_size = 10,
    };
 mbedtls_platform_context ctx         = {0};
 const psa_algorithm_t    alg         = PSA_ALG_CBC_PKCS7;
    psa_cipher_operation_t   operation_1 = PSA_CIPHER_OPERATION_INIT;
    psa_cipher_operation_t   operation_2 = PSA_CIPHER_OPERATION_INIT;
 size_t iv_len = 0;
    psa_key_handle_t    key_handle       = 0;
 size_t               encrypted_length = 0;
 size_t               decrypted_length = 0;
    uint8_t             iv[block_size]    = {0};
    uint8_t             input[input_size] = {0};
    uint8_t             encrypted_data[input_size + block_size] = {0};
    uint8_t             decrypted_data[input_size + block_size] = {0};
    psa_key_attributes_t attributes = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_lifetime_t   lifetime;
 /* Setup the platform; initialize the SCE */
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
 /* Platform initialization failed */
       debugger_break();
    }
```

```
 if (PSA_SUCCESS != psa_crypto_init())
    {
 /* PSA Crypto Initialization failed */
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_ENCRYPT |
PSA_KEY_USAGE_DECRYPT);
    psa_set_key_algorithm(&attributes, alg);
 /* To use wrapped keys instead of plaintext use PSA_KEY_TYPE_AES_WRAPPED. */
    psa_set_key_type(&attributes, PSA_KEY_TYPE_AES);
    psa_set_key_bits(&attributes, key_bits);
    lifetime = PSA_KEY_LIFETIME_VOLATILE;
 /* To use persistent keys:
  *......Use a lifetime value of PSA_KEY_LIFETIME_PERSISTENT
  * - The file system must be initialized prior to calling the generate/import key
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
    psa_set_key_lifetime(&attributes, lifetime);
 if (PSA_KEY_LIFETIME_IS_PERSISTENT(lifetime))
    {
 /* Set the id to a positive integer. */
       psa_set_key_id(&attributes, (psa_key_id_t) 5);
    }
 if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
    {
 /* Random number generation for input data failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
 /* Generating AES 256 key and allocating to key slot failed */
       debugger_break();
    }
```

```
 else if (PSA_SUCCESS != psa_cipher_encrypt_setup(&operation_1, key_handle, alg))
    {
 /* Initializing the encryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_generate_iv(&operation_1, iv, sizeof(iv),
&iv_len))
    {
 /* Generating the random IV failed */
       debugger_break();
    }
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_1, input, input_size, part_size,
encrypted_data, sizeof(encrypted_data),
                              &encrypted_length))
    {
 /* Encryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_1))
    {
 /* Terminating the encryption operation failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_decrypt_setup(&operation_2, key_handle, alg))
    {
 /* Initializing the decryption (with PKCS7 padding) operation handle failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_set_iv(&operation_2, iv, sizeof(iv)))
    {
 /* Setting the IV failed */
       debugger_break();
    }
```

```
 else if (PSA_SUCCESS !=
             cipher_operation(&operation_2, encrypted_data, encrypted_length,
part_size, decrypted_data,
 sizeof(decrypted_data), &decrypted_length))
    {
 /* Decryption failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_cipher_abort(&operation_2))
    {
 /* Terminating the decryption operation failed */
       debugger_break();
    }
 else if (0 != memcmp(input, decrypted_data, sizeof(input)))
    {
 /* Comparing the input data with decrypted data failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
 /* Destroying the key handle failed */
       debugger_break();
    }
 else
    {
 /* All the operations succeeded */
    }
 /* Close the SCE */
 mbedtls_platform_teardown(&ctx);
}
```

AES-CCM Example

This is an example on using the PSA Crypto API to generate an AES256 key, encrypting and decrypting multi-block data and using PKCS7 padding using AES-CCM.

```
 if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
    {
 /* Random plaintext input generation failed */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
 /* Key generation failed */
       debugger_break();
    }
 /* AES-CCM Encryption */
 else if (PSA_SUCCESS !=
             psa_aead_encrypt(key_handle, PSA_ALG_CCM, nonce, sizeof(nonce),
additional_data, sizeof(additional_data),
                              input, sizeof(input), encrypt, sizeof(encrypt),
&output_len))
    {
 /* AES-CCM Encryption failed */
       debugger_break();
    }
 /* AES-CCM Decryption */
 else if (PSA_SUCCESS !=
             psa_aead_decrypt(key_handle, PSA_ALG_CCM, nonce, sizeof(nonce),
additional_data, sizeof(additional_data),
                              encrypt, output_len, decrypt, sizeof(decrypt),
&output_len))
    {
 /* AES-CCM Decryption failed */
       debugger_break();
    }
 else if (0U != memcmp(input, decrypt, sizeof(input)))
    {
 /* The decrypted result did not match the plaintext input */
       debugger_break();
```

```
    }
else
    {
/* All operations were successful */
    }
```

CMAC Example

This is an example on using the PSA Crypto API to generate an AES256 key, followed by generation and verification of MAC for random data of known length.

```
if (PSA_SUCCESS != psa_generate_random(input, sizeof(input)))
    {
/* Random number generation failure */
        debugger_break();
    }
else if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
/* Key generation failure */
        debugger_break();
    }
/* Steps to generate the MAC */
else if (PSA_SUCCESS != psa_mac_sign_setup(&operation, key_handle, alg))
    {
/* MAC Sign setup failed */
        debugger_break();
    }
else if (PSA_SUCCESS != psa_mac_update(&operation, input, input_size))
    {
/* MAC update failed */
        debugger_break();
    }
else if (PSA_SUCCESS != psa_mac_sign_finish(&operation, AES_CMAC_mac, sizeof
(AES_CMAC_mac), &mac_ret))
    {
```

```
 /* MAC Sign operation failed */
       debugger_break();
    }
 else
    {
 /* All the operations succeeded for MAC generation */
    }
 /* Steps to verify the generated MAC */
 if (PSA_SUCCESS != psa_mac_verify_setup(&verify_operation, key_handle, alg))
    {
 /* MAC verification setup failure */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_mac_update(&verify_operation, input, input_size))
    {
 /* MAC update failure */
       debugger_break();
    }
 else if (PSA_SUCCESS != psa_mac_verify_finish(&verify_operation, AES_CMAC_mac,
mac_ret))
    {
 /* MAC verification failed */
       debugger_break();
    }
 else
    {
 /* All the operations succeeded for MAC verification */
    }
```

ECC Example

This is an example on using the PSA Crypto API to generate an ECC-P256R1 key, signing and verifying data after hashing it first using SHA256.

Note

> *Unlike RSA, ECDSA does not have any padding schemes. Thus the hash argument for the ECC sign operation MUST have a size larger than or equal to the curve size; i.e. for PSA_ECC_CURVE_SECP256R1 the payload size*

must be at least 256/8 bytes. nist.fips.186-4: " A hash function that provides a lower security strength than the security strength associated with the bit length of 'n' ordinarily should not be used, since this would reduce the security strength of the digital signature process to a level no greater than that provided by the hash function."

```
#define ECC_256_BIT_LENGTH 256
#define ECC_256_EXPORTED_SIZE 500
uint8_t exportedECC_SECP256R1Key[ECC_256_EXPORTED_SIZE];
size_t exportedECC_SECP256R1Keylength = 0;
void psa_ecc256R1_example (void)
{
/* This example generates an ECC-P256R1 keypair, performs signing and verification
operations.
 * It then exports the generated key into ASN1 DER format to a RAM array which can
then be programmed to flash.
 * It then re-imports that key, and performs signing and verification operations. */
 unsigned char            payload[] = "ASYMMETRIC_INPUT_FOR_SIGN......";
 unsigned char            signature1[PSA_SIGNATURE_MAX_SIZE] = {0};
 unsigned char            signature2[PSA_SIGNATURE_MAX_SIZE] = {0};
 size_t                   signature_length1 = 0;
 size_t                   signature_length2 = 0;
    psa_key_attributes_t    attributes       = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t    read_attributes   = PSA_KEY_ATTRIBUTES_INIT;
 mbedtls_platform_context ctx              = {0};
    psa_key_handle_t         ecc_key_handle    = {0};
    psa_hash_operation_t hash_operation = {0};
    uint8_t             payload_hash[PSA_HASH_MAX_SIZE];
 size_t               payload_hash_len;
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
       debugger_break();
    }
 /* Set key attributes */
```

```
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
PSA_KEY_USAGE_VERIFY_HASH | PSA_KEY_USAGE_EXPORT);
    psa_set_key_algorithm(&attributes, PSA_ALG_ECDSA(PSA_ALG_SHA_256));
 /* To use wrapped keys instead of plaintext:
  * - Use PSA_KEY_TYPE_ECC_KEY_PAIR_WRAPPED(PSA_ECC_FAMILY_SECP_R1).*/
    psa_set_key_type(&attributes, PSA_KEY_TYPE_ECC_KEY_PAIR(PSA_ECC_FAMILY_SECP_R1));
    psa_set_key_bits(&attributes, ECC_256_BIT_LENGTH);
 /* To use persistent keys instead of volatile:
  * - Use PSA_KEY_LIFETIME_PERSISTENT.
  * - The file system must be initialized prior to calling the generate/import key
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 /* Generate ECC P256R1 Key pair */
 if (PSA_SUCCESS != psa_generate_key(&attributes, &ecc_key_handle))
    {
       debugger_break();
    }
 /* Test the key information */
 if (PSA_SUCCESS != psa_get_key_attributes(ecc_key_handle, &read_attributes))
    {
       debugger_break();
    }
 /* Calculate the hash of the message */
 if (PSA_SUCCESS != psa_hash_setup(&hash_operation, PSA_ALG_SHA_256))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_hash_update(&hash_operation, payload, sizeof(payload)))
    {
       debugger_break();
    }
 if (PSA_SUCCESS !=
       psa_hash_finish(&hash_operation, &payload_hash[0], sizeof(payload_hash),
```

```
&payload_hash_len))
    {
       debugger_break();
    }
 /* Sign message using the private key
  * NOTE: The hash argument (payload_hash here) MUST have a size equal to the curve
size;
  * i.e. for SECP256R1 the payload size must be 256/8 bytes.
  * Similarly for SECP384R1 the payload size must be 384/8 bytes.
  * nist.fips.186-4: " A hash function that provides a lower security strength than
  * the security strength associated with the bit length of 'n' ordinarily should not
be used, since this
  * would reduce the security strength of the digital signature process to a level no
greater than that
  * provided by the hash function." */
 if (PSA_SUCCESS !=
       psa_sign_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature1,
 sizeof(signature1), &signature_length1))
    {
       debugger_break();
    }
 /* Verify the signature1 using the public key */
 if (PSA_SUCCESS !=
       psa_verify_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature1,
                        signature_length1))
    {
       debugger_break();
    }
 /* Export the key. The exported key can then be save to flash for later usage. */
 if (PSA_SUCCESS !=
       psa_export_key(ecc_key_handle, exportedECC_SECP256R1Key, sizeof
(exportedECC_SECP256R1Key),
```

```
                      &exportedECC_SECP256R1Keylength))
    {
       debugger_break();
    }
 /* Destroy the key and handle */
 if (PSA_SUCCESS != psa_destroy_key(ecc_key_handle))
    {
       debugger_break();
    }
 /* Import the previously exported key pair */
 if (PSA_SUCCESS !=
       psa_import_key(&attributes, exportedECC_SECP256R1Key,
exportedECC_SECP256R1Keylength, &ecc_key_handle))
    {
       debugger_break();
    }
 /* Sign message using the private key */
 if (PSA_SUCCESS !=
       psa_sign_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature2,
 sizeof(signature2), &signature_length2))
    {
       debugger_break();
    }
 /* Verify signature2 using the public key */
 if (PSA_SUCCESS !=
       psa_verify_hash(ecc_key_handle, PSA_ALG_ECDSA(PSA_ALG_SHA_256), payload_hash,
payload_hash_len, signature2,
                        signature_length2))
    {
       debugger_break();
    }
 /* Signatures cannot be compared since ECC signatures vary for the same data unless
Deterministic ECC is used which is not supported by the HW.
```

```
  * Only the verification operation can be used to validate signatures. */
}
```

RSA Example

This is an example on using the PSA Crypto API to generate an RSA2048 key, encrypting and decrypting multi-block data and using PKCS7 padding.

```
#define RSA_2048_BIT_LENGTH 2048
#define RSA_2048_EXPORTED_SIZE 1210
/* The RSA 2048 key pair export in der format is roughly as follows
 * RSA private keys:
 * RSAPrivateKey ::= SEQUENCE { -------------------------------------- 1 + 3
 * version Version, --------------------------------------- 1 + 1 + 1
 * modulus INTEGER, ---------------- n ------------------- 1 + 3 + 256 + 1
 * publicExponent INTEGER, ---------------- e ------------------- 1 + 4
 * privateExponent INTEGER, ---------------- d ------------------- 1 + 3 + 256 (276
for Wrapped)
 * prime1 INTEGER, ---------------- p ------------------- 1 + 3 + (256 / 2)
 * prime2 INTEGER, ---------------- q ------------------- 1 + 3 + (256 / 2)
 * exponent1 INTEGER, ---------------- d mod (p-1) --------- 1 + 2 + (256 / 2) (4 for
Wrapped)
 * exponent2 INTEGER, ---------------- d mod (q-1) --------- 1 + 2 + (256 / 2) (4 for
Wrapped)
 * coefficient INTEGER, ---------------- (inverse of q) mod p - 1 + 2 + (256 / 2) (4
for Wrapped)
 * otherPrimeInfos OtherPrimeInfos OPTIONAL ------------------------ 0 (not
supported)
 * }
 */
uint8_t exportedRSA2048Key[RSA_2048_EXPORTED_SIZE];
size_t exportedRSA2048Keylength = 0;
void psa_rsa2048_example (void)
{
/* This example generates an RSA2048 keypair, performs signing and verification
```

```
operations.
 * It then exports the generated key into ASN1 DER format to a RAM array which can
then be programmed to flash.
 * It then re-imports that key, and performs signing and verification operations. */
 mbedtls_platform_context ctx       = {0};
    psa_key_handle_t         key_handle = {0};
 unsigned char            payload[] = "ASYMMETRIC_INPUT_FOR_SIGN";
 unsigned char            signature1[PSA_SIGNATURE_MAX_SIZE] = {0};
 unsigned char            signature2[PSA_SIGNATURE_MAX_SIZE] = {0};
 size_t               signature_length1 = 0;
 size_t               signature_length2 = 0;
    psa_key_attributes_t attributes       = PSA_KEY_ATTRIBUTES_INIT;
    psa_key_attributes_t read_attributes   = PSA_KEY_ATTRIBUTES_INIT;
 if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
    {
       debugger_break();
    }
 if (PSA_SUCCESS != psa_crypto_init())
    {
       debugger_break();
    }
 /* Set key attributes */
    psa_set_key_usage_flags(&attributes, PSA_KEY_USAGE_SIGN_HASH |
PSA_KEY_USAGE_VERIFY_HASH | PSA_KEY_USAGE_EXPORT);
    psa_set_key_algorithm(&attributes, PSA_ALG_RSA_PKCS1V15_SIGN_RAW);
 /* To use wrapped keys instead of plaintext:
  * - Use PSA_KEY_TYPE_RSA_KEY_PAIR_WRAPPED. */
    psa_set_key_type(&attributes, PSA_KEY_TYPE_RSA_KEY_PAIR);
    psa_set_key_bits(&attributes, RSA_2048_BIT_LENGTH);
 /* To use persistent keys instead of volatile:
  * - Use PSA_KEY_LIFETIME_PERSISTENT.
  * - The file system must be initialized prior to calling the generate/import key
functions.
  * - Refer to the littlefs example to see how to format and mount the filesystem. */
```

```
    psa_set_key_lifetime(&attributes, PSA_KEY_LIFETIME_VOLATILE);
 /* Generate RSA 2048 Key pair */
 if (PSA_SUCCESS != psa_generate_key(&attributes, &key_handle))
    {
       debugger_break();
    }
 /* Test the key information */
 if (PSA_SUCCESS != psa_get_key_attributes(key_handle, &read_attributes))
    {
       debugger_break();
    }
 /* Sign message using the private key */
 if (PSA_SUCCESS !=
       psa_sign_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload, sizeof
(payload), signature1,
 sizeof(signature1), &signature_length1))
    {
       debugger_break();
    }
 /* Verify the signature1 using the public key */
 if (PSA_SUCCESS !=
       psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload,
sizeof(payload), signature1,
                        signature_length1))
    {
       debugger_break();
    }
 /* Export the key */
 if (PSA_SUCCESS !=
       psa_export_key(key_handle, exportedRSA2048Key, sizeof(exportedRSA2048Key),
&exportedRSA2048Keylength))
    {
       debugger_break();
    }
```

```
 /* Destroy the key and handle */
 if (PSA_SUCCESS != psa_destroy_key(key_handle))
    {
       debugger_break();
    }
 /* Import the previously exported key pair */
 if (PSA_SUCCESS != psa_import_key(&attributes, exportedRSA2048Key,
exportedRSA2048Keylength, &key_handle))
    {
       debugger_break();
    }
 /* Sign message using the private key */
 if (PSA_SUCCESS !=
       psa_sign_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload, sizeof
(payload), signature2,
 sizeof(signature2), &signature_length2))
    {
       debugger_break();
    }
 /* Verify signature2 using the public key */
 if (PSA_SUCCESS !=
       psa_verify_hash(key_handle, PSA_ALG_RSA_PKCS1V15_SIGN_RAW, payload,
sizeof(payload), signature2,
                        signature_length2))
    {
       debugger_break();
    }
 /* Compare signatures to verify that the same signature was generated */
 if (0 != memcmp(signature2, signature1, signature_length2))
    {
       debugger_break();
    }
    mbedtls_psa_crypto_free();
 mbedtls_platform_teardown(&ctx);
```

```
}
```

Function Documentation

◆ RM_PSA_CRYPTO_TRNG_Read()

fsp_err_t RM_PSA_CRYPTO_TRNG_Read (uint8_t *const *p_rngbuf*, uint32_t *num_req_bytes*, uint32_t * *p_num_gen_bytes*)

Reads requested length of random data from the TRNG. Generate nbytes of random bytes and store them in p_rngbuf buffer.

Return values

FSP_SUCCESS	Random number generation successful
FSP_ERR_ASSERTION	NULL input parameter(s).
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:

- s_generate_16byte_random_data

◆ mbedtls_platform_setup()

int mbedtls_platform_setup (mbedtls_platform_context * *ctx*)

This function initializes the SCE and the TRNG. It **must** be invoked before the crypto library can be used. This implementation is used if MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT is defined.

Example:

```
mbedtls_platform_context ctx = {0};
/* Setup the platform; initialize the SCE and the TRNG */
if (PSA_SUCCESS != mbedtls_platform_setup(&ctx))
```

Return values

0	Initialization was successful.
MBEDTLS_ERR_PLATFORM_HW_ACCEL_FAILED	SCE Initialization error.

◆ mbedtls_platform_teardown()

void mbedtls_platform_teardown (mbedtls_platform_context * *ctx*)

This implementation is used if MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT is defined. It is intended to de-initialize any items that were initialized in the mbedtls_platform_setup() function, but currently is only a placeholder function.

Example:

```
/* De-initialize the platform. This is currently a placeholder function which does
not do anything. */
mbedtls_platform_teardown(&ctx);
```

Return values

N/A	

4.2.113 Azure RTOS ThreadX Port (rm_threadx_port)

Modules

ThreadX port for RA MCUs.

Overview

Note

The ThreadX Port does not provide any interfaces to the user. Consult the ThreadX documentation at https://docs.microsoft.com/en-us/azure/rtos/threadx/ for further information.

Features

The RA ThreadX port supports the following features:

- Standard ThreadX configurations
- Hardware stack monitor

Configuration

Build Time Configurations for ThreadX

The following build time configurations are defined in fsp_cfg/azure/tx/tx_user.h:

Configuration	Options	Default	Description
General > Custom tx_user.h	Manual Entry		Add a path to your custom tx_user.h file. It

			can be used to override some or all of the configurations defined here, and to define additional configurations.
General > Error Checking	• Enabled • Disabled	Enabled	The ThreadX basic API error checking can be bypassed by compiling with the symbol TX_DISABLE_ERROR_CHECKING defined.
General > Max Priorities	Value must be a multiple of 32 or empty	32	Define the priority levels for ThreadX. Legal values range from 32 to 1024 and MUST be evenly divisible by 32.
General > Minimum Stack	Value must be greater than 0 or empty	200	Define the minimum stack for a ThreadX thread on this processor. If the size supplied during thread creation is less than this value, the thread create call will return an error.
General > Stack Filling	• Enabled • Disabled	Enabled	Determine is stack filling is enabled. By default, ThreadX stack filling is enabled, which places an 0xEF pattern in each byte of each thread's stack. This is used by debuggers with ThreadX-awareness and by the ThreadX run-time stack checking feature.
General > Preemption Threshold	• Enabled • Disabled	Disabled	Determine if preemption-threshold should be disabled. By default, preemption-threshold is disabled. If the application does not use preemption-threshold, it may be disabled to reduce code size and improve performance.
General > Notify	• Enabled	Disabled	Determine if the notify

Callbacks	• Disabled		callback option should be disabled. By default, notify callbacks are disabled. If the application does not use notify callbacks, they may be disabled to reduce code size and improve performance.
General > Inline Thread Resume Suspend	• Enabled • Disabled	Disabled	Determine if the tx_thread_resume and tx_thread_suspend services should have their internal code in-line. This results in a larger image, but improves the performance of the thread resume and suspend services.
General > Not Interruptable	• Enabled • Disabled	Disabled	Determine if the internal ThreadX code is non-interruptable. This results in smaller code size and less processing overhead, but increases the interrupt lockout time.
General > IAR Library Support	• Enabled • Disabled	Disabled	Enable IAR library support (IAR compiler only). When IAR Library Support is Enabled, enable the linker option --threaded_lib. In the IAR IDE, this can be enabled in Project > Options > General Options > Library Configuration > Enable thread support in library.
General > BSD Support	• Enabled • Disabled	Disabled	Defines TX_THREAD_EXTENSION_1 to bsd_err_no in order to support NXD BSD.
General > FileX Pointer	• Enabled • Disabled	Enabled	Determine if there is a FileX pointer in the thread control block. By default, the pointer is there for legacy/backwards

			compatibility. The pointer must also be there for applications using FileX. Disable this to save space in the thread control block.
Timer > Timer Ticks Per Second	Value must be greater than 0 or empty	100	Define the number of times the system timer runs per second. Default is 100 ticks per second, which results in a tick every 10ms.
Timer > Timer Thread Stack Size	Value must be greater than 0 or empty	1024	Define the system timer thread's default stack size and priority. These are only applicable if TX_TIMER_PROCESS_IN_ISR is disabled.
Timer > Timer Thread Priority	Value must be greater than 0 or empty	0	Define the system timer thread's default stack size and priority. These are only applicable if TX_TIMER_PROCESS_IN_ISR is disabled.
Timer > Timer Process In ISR	• Enabled • Disabled	Enabled	Determine if timer expirations (application timers, timeouts, and tx_thread_sleep calls should be processed within the a system timer thread or directly in the timer ISR. When disabled, the timer thread is used. When enabled, timer expiration processing is done directly from the timer ISR, thereby eliminating the timer thread control block, stack, and context switching to activate it.
Timer > Reactivate Inline	• Enabled • Disabled	Disabled	Determine if in-line timer reactivation should be used within the timer expiration processing. By default, this is disabled and a function call is used. When enabled,

			reactivating is performed in-line resulting in faster timer processing but slightly larger code size.
Timer > Timer	• Enabled • Disabled	Enabled	Determine if no timer processing is required. This option will help eliminate the timer processing when not needed.
Trace > Event Trace	• Enabled • Disabled	Disabled	Determine if the trace event logging code should be enabled. This causes slight increases in code size and overhead, but provides the ability to generate system trace information which is available for viewing in TraceX.
Trace > Trace Buffer Name	Name must be a valid C symbol	g_tx_trace_buffer	Name of trace buffer symbol, only used if Event Trace is enabled.
Trace > Memory section for Trace Buffer	Manual Entry	.bss	Specify the memory section where the Trace Buffer will be allocated, only used if Event Trace is enabled. To view TraceX data, export this buffer as raw binary data to a file (.trx extension recommended) and open it with Microsoft Azure RTOS TraceX.
Trace > Trace Buffer Size	Value must be greater than 0	65536	Trace buffer size in bytes, only used if Event Trace is enabled
Trace > Trace Buffer Number of Registries	Value must be greater than 0	30	Number of registries available to TraceX, only used if Event Trace is enabled
Performance > Block Pool Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers block pool performance information.
Performance > Byte Pool Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers byte

			pool performance information.
Performance > Event Flags Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers event flags performance information.
Performance > Mutex Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers mutex performance information.
Performance > Queue Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers queue performance information.
Performance > Semaphore Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers semaphore performance information.
Performance > Thread Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers thread performance information.
Performance > Timer Performance Info	• Enabled • Disabled	Disabled	When enabled, ThreadX gathers timer performance information.
RA > Hardware Thread Stack Monitoring	MCU Specific Options		Use RA Hardware Stack Monitors to monitor thread stacks for overflow. Not available on MCUs that support PSPLIM.
Interrupts > SysTick Interrupt Priority	MCU Specific Options		Select the Systick interrupt priority.
Interrupts > Maximum Interrupt Priority	MCU Specific Options		The maximum priority (lowest numerical value) an interrupt can have and use scheduler services. Interrupts with higher priority can interrupt most scheduler critical sections. Setting this to Priority 0 (highest) disables this feature. This feature is not available on MCUs that do not have the

BASEPRI register.

Clock Configuration

The ThreadX port uses the SysTick timer as the system clock. The timer rate is configured in the ThreadX component under General > Timer Ticks Per Second.

Pin Configuration

This module does not use I/O pins.

Usage Notes

Interrupt Priorities

When no threads are ready to run, the ThreadX port spins in the PendSV_Handler, which is fixed at the lowest interrupt priority. The MCU does not service any other interrupts of the lowest priority while no threads are ready to run.

To get around this limitation, the application can create an idle thread that is always ready to run. If the idle thread enters a lower power mode, make sure all interrupts that are required to resume the scheduler can wake the MCU in the configured power mode. If the application expects to wake after a certain number of ticks, the idle thread should not enter standby mode because the SysTick cannot wake the MCU from standby mode. See Low Power Modes (r_lpm) for more information regarding low power modes.

Warning
: Do not attempt to wake a thread from an interrupt with the lowest available interrupt priority unless the application has created an idle thread.

Hardware Stack Monitor

The hardware stack monitor generates an NMI if the PSP goes out of the memory area for the stack allocated for the current thread. A callback can be registered using R_BSP_GroupIrqWrite() to be called whenever a stack overflow or underflow of the PSP for a particular thread is detected.

Low Power Modes

The idle processing executes WFI() when no thread is ready to run. If the MCU is configured to enter software standby mode or deep software standby mode when the idle processing executes WFI(), the RA ThreadX port changes the low power mode to sleep mode so the idle processing can wake from SysTick. The low power mode settings are restored when the MCU wakes from sleep mode.

TrustZone Integration

When using an RTOS in a TrustZone project, ARM recommends keeping the RTOS in the non-secure project. Tasks may call non-secure callable functions if the thread has allocated a secure context (using tx_thread_secure_stack_allocate).

The secure context can be freed by deleting the thread or calling tx_thread_secure_stack_free.

Examples

Stack Monitor Example

This is an example of using the stack monitor in an application.

```
#if BSP_FEATURE_BSP_HAS_SP_MON
void stack_monitor_callback(bsp_grp_irq_t irq);
void stack_monitor_callback (bsp_grp_irq_t irq)
{
 FSP_PARAMETER_NOT_USED(irq);
 if (1U == R_MPU_SPMON->SP[0].CTL_b.ERROR)
    {
 /* Handle main stack monitor error here. */
    }
 if (1U == R_MPU_SPMON->SP[1].CTL_b.ERROR)
    {
 /* Handle process stack monitor error here. */
    }
}
void rm_threadx_port_stack_monitor_example (void)
{
 /* Register a callback to be called when the stack goes outside the allocated stack
area. */
 R_BSP_GroupIrqWrite(BSP_GRP_IRQ_MPU_STACK, stack_monitor_callback);
}
#else
/* Allocate stack space to return from UsageFault. */
uint32_t g_stack_overflow_exception_stack[8] BSP_ALIGN_VARIABLE(BSP_STACK_ALIGNMENT)
BSP_PLACE_IN_SECTION(
    BSP_SECTION_STACK);
/* MCUs that do not have an SPMON stack monitor use PSPLIM to detect stack overflows.
When a stack overflow error
 * occurs, the UsageFault_Handler fires if it has been enabled. */
void UsageFault_Handler (void)
{
 register uint32_t cfsr = SCB->CFSR;
 if (cfsr & SCB_CFSR_STKOF_Msk)
    {
```

```
 /* Update PSP and PSPLIM to point to an exception stack frame allocated for stack
overflows. */
 register uint32_t * p_exception_stack_frame = (uint32_t *)
(&g_stack_overflow_exception_stack);
       __set_PSP((uint32_t) p_exception_stack_frame);
       __set_PSPLIM((uint32_t) p_exception_stack_frame);
 /* Clear XPSR, only set T-bit. */
       p_exception_stack_frame[7] = 1U << 24;
 /* Set PC to stack overflow error while loop. When execution returns from the
UsageFault, it will go to the
  * stack_overflow_error_occurred function. It cannot return to the location where
the fault occurred because
  * the MCU does not save the exception stack frame to the stack when a stack
overflow error occurs. */
       p_exception_stack_frame[6] = (uint32_t) stack_overflow_error_occurred;
    }
 /* Clear flags. */
    SCB->CFSR = cfsr;
}
/* This function is called from UsageFault_Handler after a stack overflow occurs. */
void stack_overflow_error_occurred (void)
{
 /* When recovering from a stack overflow, move the thread to a while(1) loop. */
 while (1)
    {
 /* Do nothing. */
    }
}
void rm_threadx_port_stack_monitor_example (void)
{
 /* Enable usage fault. */
    SCB->SHCSR |= SCB_SHCSR_USGFAULTENA_Msk;
}
#endif
```

TrustZone Example

This is an example of calling tx_thread_secure_stack_allocate before calling any non-secure callable functions in a thread.

```
extern TX_THREAD * _tx_thread_current_ptr;
void rm_threadx_port_trustzone_thread_example (void)
{
 /* When ThreadX is used in a non-secure TrustZone application,
tx_thread_secure_stack_allocate must be called prior
  * to calling any non-secure callable function in a thread. The first parameter is a
pointer to the thread control block.
  * This function can be called when the thread is created or in the thread before an
non-secure callable function is
  * called. The second parameter is unused in the FSP implementation. */
    UINT status = tx_thread_secure_stack_allocate(_tx_thread_current_ptr, 0);
    assert(TX_SUCCESS == status);
    rm_threadx_port_nsc_function();
}
```

4.2.114 Intel TinyCrypt (rm_tinycrypt_port)

Modules

Functions

fsp_err_t	RM_TINCYRYPT_PORT_TRNG_Read (uint8_t *const p_rngbuf, uint32_t num_req_bytes) Reads requested length of random data from the TRNG. Generate num_req_bytes of random bytes and store them in p_rngbuf buffer. More...
fsp_err_t	RM_TINCYRYPT_PORT_Init (void)
int	default_CSPRNG (uint8_t *dest, unsigned int size) Implements the Cryptographically Secure Pseudo-Random Number Generator function required byt TinyCrypt. More...

Detailed Description

AES128 Hardware acceleration for TinyCrypt on the RA2 family.

Overview

Note

> ***The TinyCrypt port module does not provide any interfaces to the user. Consult the documentation at https://github.com/intel/tinycrypt/blob/master/documentation/tinycrypt.rst for further information.***

TinyCrypt is designed as a small footprint software crypto implementation to be used on resource constrained devices. The software only module is available in FSP on all RA devices. Hardware acceleration for AES-128 is provided only for the RA2 family. This release uses TinyCrypt v0.2.8.

Hardware Overview

Crypto Peripheral version	Devices
AES Engine	RA2A1, RA2E1, RA2L1

Features

For features supported by the software-only version, refer to the TinyCrypt documentation.

The TinyCrypt port module provides hardware support for the following operations

- AES
  - Keybits - 128
  - ECB, CBC, CTR, CCM and CMAC modes -TRNG

Configuration

Build Time Configurations for TinyCrypt_Acceleration

The following build time configurations are defined in fsp_cfg/rm_tinycrypt_port_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

AES Configuration

To enable hardware acceleration for the AES128 operation, choose TinyCrypt (H/W Accelerated) from the stack options . This feature is only supported on the RA2 family.

Usage Notes

Hardware Initialization

Invoke RM_TINCYRYPT_PORT_Init () to initialize the hardware before using Tinycrypt if either hardware acceleration or the TRNG is to be used.

Random Number Generation

There are two Pseudo-random Number Generators (PRNG) provided in TinyCrypt

- CTR-PRNG which uses AES128 internally in its implementation. Enabling AES128 hardware acceleration will improve the performance of this module.
- HMAC-PRNG which uses SHA256 internally in its implementation.

Both these implementations will only be able to provide a random pseudo-random number sequence if they are seeded with truly random data. The TRNG module that is present in hardware and available in rm_tinycrypt_port must be used to seed these modules. When using CTR-PRNG or HMAC-PRNG, use the RM_TINCYRYPT_PORT_TRNG_Read() function to obtain random data from the TRNG hardware and use that to seed the PRNG modules before invoking the pseudo-random number generation. If purely random data is sufficient for the application, then RM_TINCYRYPT_PORT_TRNG_Read() can be used directly instead. The hardware TRNG implements the CTR_DRBG spec.

Default CSPRNG

The TinyCrypt ECC implementation requires a platform specific implementation of the default_CSPRNG() function. This function has been implemented using the hardware TRNG in the port to support software ECC usage. When using TinyCrypt in S/W mode, it is necessary to implement default_CSPRNG() if using ECC signature generation (ECDSA) or key derivation (ECDH).

AES-128 Usage

The AES ECB mode implementation is provided in aes_encrypt.decrypt.c. All the other modes of AES operation including CBC, CCN, CMAC and CTR use the ECB mode for the block operation. On the RA2, the ECB mode has been hardware accelerated which improves performance of the other modes as well. Additionally the CBC and CTR modes are also accelerated.

To use the different AES modes, first initialize the hardware (on the RA2) and then use the functions defined in the header file of each AES mode. Note that TinyCrypt does not provide any type of padding or buffering so the data provided to these modes should be multiples of AES block size.

Memory Usage

TinyCrypt does not use dynamic allocation so there is no heap requirement.

Limitations

Usage with RA4 and RA6 devices

TinyCrypt (S/W Only) can be used on RA4 and RA6 devices. However, since ECC signature generation (ECDSA) and key derivation (ECDH) requires a random number source, that operation is currently not supported on these devices when using TinyCrypt (S/W Only). In order to support those operations the function default_CSPRNG() must be implemented in the user code.

TinyCrypt

- No padding is supported; the user is expected to provide adequately padded data depending on the algorithm used.
- AES Key generation is not supported.
- Key encoding/decoding is not supported.

Using TinyCrypt with TrustZone

Unlike FSP drivers, TinyCrypt cannot be configured as Non-secure callable in the RA Configurator for a secure project. The reason for this is that in order to achieve the security objective of controlling access to protected keys, both the crypto code as well as the keys must be placed in the secure region. Since the tinyCrypt API requires access to the keys directly during initialization and later via a key handle, allowing non-secure code to use the API by making it Non-secure callable will require the keys to be stored in non-secure memory.

This limitation is identical to that for PSA Crypto. Refer to the documentation of that module on how to create a crypto Non-Secure Callable layer to be used in such situations.

Examples

AES-CBC Example

This is an example on using TinyCrypt to encrypt and decrypt data using an AES-128 key in CBC mode.

```
#define TC_INPUT_PLAINTEXT_SIZE 64U
#define TF_AES_IV_SIZE TC_AES_BLOCK_SIZE
#define TC_OUTPUT_CIPHERTEXT_SIZE (TC_INPUT_PLAINTEXT_SIZE + TF_AES_IV_SIZE)
/*
 * NIST test vectors from SP 800-38a:
 *
 * Block #1
 * Plaintext 6bc1bee22e409f96e93d7e117393172a
 * Input Block 6bc0bce12a459991e134741a7f9e1925
 * Output Block 7649abac8119b246cee98e9b12e9197d
 * Ciphertext 7649abac8119b246cee98e9b12e9197d
 * Block #2
 * Plaintext ae2d8a571e03ac9c9eb76fac45af8e51
 * Input Block d86421fb9f1a1eda505ee1375746972c
 * Output Block 5086cb9b507219ee95db113a917678b2
 * Ciphertext 5086cb9b507219ee95db113a917678b2
 * Block #3
 * Plaintext 30c81c46a35ce411e5fbc1191a0a52ef
 * Input Block 604ed7ddf32efdff7020d0238b7c2a5d
 * Output Block 73bed6b8e3c1743b7116e69e22229516
 * Ciphertext 73bed6b8e3c1743b7116e69e22229516
 * Block #4
```

```
 * Plaintext f69f2445df4f9b17ad2b417be66c3710
 * Input Block 8521f2fd3c8eef2cdc3da7e5c44ea206
 * Output Block 3ff1caa1681fac09120eca307586e1a7
 * Ciphertext 3ff1caa1681fac09120eca307586e1a7
 */
const uint8_t cbc_key[TC_AES_KEY_SIZE] =
{
    0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7, 0x15, 0x88, 0x09,
0xcf, 0x4f, 0x3c
};
uint8_t cbc_iv[TC_AES_KEY_SIZE] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f
};
const uint8_t cbc_plaintext[64] =
{
    0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d, 0x7e, 0x11, 0x73,
0x93, 0x17, 0x2a,
    0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7, 0x6f, 0xac, 0x45,
0xaf, 0x8e, 0x51,
    0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb, 0xc1, 0x19, 0x1a,
0x0a, 0x52, 0xef,
    0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b, 0x41, 0x7b, 0xe6,
0x6c, 0x37, 0x10
};
uint8_t cbc_expected_ciphertext[TC_OUTPUT_CIPHERTEXT_SIZE] =
{
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, 0x0e, 0x0f, // NOLINT(readability-magic-numbers)
    0x76, 0x49, 0xab, 0xac, 0x81, 0x19, 0xb2, 0x46, 0xce, 0xe9, 0x8e, 0x9b, 0x12,
0xe9, 0x19, 0x7d, // NOLINT(readability-magic-numbers)
    0x50, 0x86, 0xcb, 0x9b, 0x50, 0x72, 0x19, 0xee, 0x95, 0xdb, 0x11, 0x3a, 0x91,
0x76, 0x78, 0xb2, // NOLINT(readability-magic-numbers)
```

```
    0x73, 0xbe, 0xd6, 0xb8, 0xe3, 0xc1, 0x74, 0x3b, 0x71, 0x16, 0xe6, 0x9e, 0x22,
0x22, 0x95, 0x16, // NOLINT(readability-magic-numbers)
    0x3f, 0xf1, 0xca, 0xa1, 0x68, 0x1f, 0xac, 0x09, 0x12, 0x0e, 0xca, 0x30, 0x75,
0x86, 0xe1, 0xa7 // NOLINT(readability-magic-numbers)
};
void tinycrypt_aes128cbc_example (void)
{
 struct tc_aes_key_sched_struct aes_keyschedule;
    uint8_t cbc_encrypted[TC_OUTPUT_CIPHERTEXT_SIZE] = {0U};
    uint8_t cbc_decrypted[TC_OUTPUT_CIPHERTEXT_SIZE] = {0U};
 if (TC_CRYPTO_SUCCESS != tc_aes128_set_encrypt_key(&aes_keyschedule, cbc_key))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_cbc_mode_encrypt(cbc_encrypted, sizeof(cbc_plaintext) +
TC_AES_BLOCK_SIZE, cbc_plaintext,
 sizeof(cbc_plaintext), cbc_iv, &aes_keyschedule))
    {
       debugger_break();
    }
 else if (0 != memcmp(&cbc_encrypted[0], &cbc_expected_ciphertext[0], sizeof
(cbc_encrypted)))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_cbc_mode_decrypt(cbc_decrypted, sizeof(cbc_encrypted),
&cbc_encrypted[TC_AES_BLOCK_SIZE],
 sizeof(cbc_encrypted), cbc_encrypted, &aes_keyschedule))
    {
       debugger_break();
    }
 else if (0 != memcmp(&cbc_plaintext[0], &cbc_decrypted[0], sizeof(cbc_plaintext)))
```

```
    {
       debugger_break();
    }
 else
    {
 /* Operation successful. */
 while (1)
      {
            ;
      }
    }
}
```

AES-CTR Example

This is an example on using TinyCrypt to encrypt and decrypt data using an AES-128 key in CTR mode.

```
#define TC_CTR_INPUT_PLAINTEXT_SIZE 64U
#define TF_AES_IV_SIZE TC_AES_BLOCK_SIZE
#define TC_CTR_OUTPUT_CIPHERTEXT_SIZE (TC_CTR_INPUT_PLAINTEXT_SIZE + TF_AES_IV_SIZE)
/*
 * NIST SP 800-38a CTR Test for encryption and decryption.
 */
const uint8_t ctr_key[TC_AES_KEY_SIZE] =
{
    0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7, 0x15, 0x88, 0x09,
0xcf, 0x4f, 0x3c
};
uint8_t ctr_iv[TC_AES_KEY_SIZE] =
{
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc,
0xfd, 0xfe, 0xff // NOLINT(readability-magic-numbers)
};
const uint8_t ctr_plaintext[64] =
```

```
{
    0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d, 0x7e, 0x11, 0x73,
0x93, 0x17, 0x2a,
    0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7, 0x6f, 0xac, 0x45,
0xaf, 0x8e, 0x51,
    0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb, 0xc1, 0x19, 0x1a,
0x0a, 0x52, 0xef,
    0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b, 0x41, 0x7b, 0xe6,
0x6c, 0x37, 0x10
};
const uint8_t ctr_expected_ciphertext[TC_CTR_OUTPUT_CIPHERTEXT_SIZE] =
{
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc,
0xfd, 0xfe, 0xff,
    0x87, 0x4d, 0x61, 0x91, 0xb6, 0x20, 0xe3, 0x26, 0x1b, 0xef, 0x68, 0x64, 0x99,
0x0d, 0xb6, 0xce,
    0x98, 0x06, 0xf6, 0x6b, 0x79, 0x70, 0xfd, 0xff, 0x86, 0x17, 0x18, 0x7b, 0xb9,
0xff, 0xfd, 0xff,
    0x5a, 0xe4, 0xdf, 0x3e, 0xdb, 0xd5, 0xd3, 0x5e, 0x5b, 0x4f, 0x09, 0x02, 0x0d,
0xb0, 0x3e, 0xab,
    0x1e, 0x03, 0x1d, 0xda, 0x2f, 0xbe, 0x03, 0xd1, 0x79, 0x21, 0x70, 0xa0, 0xf3,
0x00, 0x9c, 0xee
};
void tinycrypt_aes128ctr_example (void)
{
 struct tc_aes_key_sched_struct aes_keyschedule;
    uint8_t ctr_encrypted[TC_CTR_OUTPUT_CIPHERTEXT_SIZE] = {0U};
    uint8_t ctr_decrypted[TC_CTR_OUTPUT_CIPHERTEXT_SIZE] = {0U};
 if (0 != memcpy(ctr_encrypted, ctr_iv, sizeof(ctr_iv)))
    {
       debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS != tc_aes128_set_encrypt_key(&aes_keyschedule, ctr_key))
    {
```

```
        debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_ctr_mode(&ctr_encrypted[TC_AES_BLOCK_SIZE], sizeof(ctr_plaintext),
ctr_plaintext,
 sizeof(ctr_plaintext), ctr_iv, &aes_keyschedule))
    {
        debugger_break();
    }
 else if (0 != memcmp(&ctr_encrypted[0], &ctr_expected_ciphertext[0], sizeof
(ctr_encrypted)))
    {
        debugger_break();
    }
 else if (0 != memcpy(ctr_iv, ctr_encrypted, sizeof(ctr_iv)))
    {
        debugger_break();
    }
 else if (TC_CRYPTO_SUCCESS !=
             tc_ctr_mode(ctr_decrypted, sizeof(ctr_decrypted),
&ctr_encrypted[TC_AES_BLOCK_SIZE],
 sizeof(ctr_decrypted), ctr_iv, &aes_keyschedule))
    {
        debugger_break();
    }
 else if (0 != memcmp(&ctr_plaintext[0], &ctr_decrypted[0], sizeof(ctr_plaintext)))
    {
        debugger_break();
    }
 else
    {
 /* Operation successful. */
 while (1)
      {
```

```
            ;
      }
    }
}
```

CTR-PRNG Example

This is an example on using the CTR_PRNG module in TinyCrypt to obtain random data.

```
#define TC_ENTROPY_SIZE 64U
#define TC_CTRPRNG_OUTPUT_SIZE 32U
void tinycrypt_ctr_prng_example (void)
{
    TCCtrPrng_t cprng_ctx;
    uint8_t    seed[TC_ENTROPY_SIZE];
    uint8_t    ctr_prng_output_1[TC_CTRPRNG_OUTPUT_SIZE] = {0};
    uint8_t    ctr_prng_output_2[TC_CTRPRNG_OUTPUT_SIZE] = {0};
 /* Setup the platform; initialize the crypto engine. */
 if (0 != RM_TINCYRYPT_PORT_Init())
    {
       debugger_break();
    }
 /* Read random data from the TRNG to use as seed for the CTR_PRNG. */
 else if (FSP_SUCCESS != RM_TINCYRYPT_PORT_TRNG_Read(seed, sizeof(seed)))
    {
       debugger_break();
    }
 /* Initialize and seed the CTR_PRNG with the random data from the TRNG. */
 else if (TC_CRYPTO_SUCCESS != tc_ctr_prng_init(&cprng_ctx, seed, sizeof(seed), 0,
0))
    {
       debugger_break();
    }
 /* Read random data from the CTR_PRNG. */
 else if (TC_CRYPTO_SUCCESS !=
```

```
             tc_ctr_prng_generate(&cprng_ctx, 0, 0, ctr_prng_output_1, sizeof
(ctr_prng_output_1)))
    {
       debugger_break();
    }
 /* Check that the generated value is not 0. */
 else if (0 != memcmp(&ctr_prng_output_1[0], &ctr_prng_output_2[0], sizeof
(ctr_prng_output_1)))
    {
       debugger_break();
    }
 /* Read random data again from the TRNG. */
 else if (TC_CRYPTO_SUCCESS !=
             tc_ctr_prng_generate(&cprng_ctx, 0, 0, ctr_prng_output_1, sizeof
(ctr_prng_output_1)))
    {
       debugger_break();
    }
 /* Check that the generated value is different than the previous call. */
 else if (0 != memcmp(&ctr_prng_output_1[0], &ctr_prng_output_2[0], sizeof
(ctr_prng_output_1)))
    {
       debugger_break();
    }
 else
    {
 /* Operation successful. */
 while (1)
      {
            ;
      }
    }
}
```

Function Documentation

◆ RM_TINCYRYPT_PORT_TRNG_Read()

fsp_err_t RM_TINCYRYPT_PORT_TRNG_Read (uint8_t *const *p_rngbuf*, uint32_t *num_req_bytes*)

Reads requested length of random data from the TRNG. Generate num_req_bytes of random bytes and store them in p_rngbuf buffer.

Return values

FSP_SUCCESS	Random number generation successful
FSP_ERR_ASSERTION	NULL input parameter(s).
FSP_ERR_CRYPTO_UNKNOWN	An unknown error occurred.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- HW_SCE_RNG_Read

◆ RM_TINCYRYPT_PORT_Init()

fsp_err_t RM_TINCYRYPT_PORT_Init ( )

Initialize the SCE.

◆ default_CSPRNG()

int default_CSPRNG (uint8_t * *dest*, unsigned int *size*)

Implements the Cryptographically Secure Pseudo-Random Number Generator function required byt TinyCrypt.

Return values

TC_CRYPTO_SUCCESS	Random number generation successful
TC_CRYPTO_FAIL	Random number generation failed.

Returns

See Common Error Codes or functions called by this function for other possible return codes. This function calls:
- RM_TINCYRYPT_PORT_TRNG_Read

4.2.115 Capacitive Touch Middleware (rm_touch)

Modules

Functions

fsp_err_t	RM_TOUCH_Open (touch_ctrl_t *const p_ctrl, touch_cfg_t const *const p_cfg) Opens and configures the TOUCH Middle module. Implements touch_api_t::open. More...
fsp_err_t	RM_TOUCH_ScanStart (touch_ctrl_t *const p_ctrl) This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with RM_TOUCH_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements touch_api_t::scanStart. More...
fsp_err_t	RM_TOUCH_DataGet (touch_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position) Gets the 64-bit mask indicating which buttons are pressed. Also, this function gets the current position of where slider or wheel is being pressed. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements touch_api_t::dataGet. More...
fsp_err_t	RM_TOUCH_PadDataGet (touch_ctrl_t *const p_ctrl, uint16_t *p_pad_rx_coordinate, uint16_t *p_pad_tx_coordinate, uint8_t *p_pad_num_touch) This function gets the current position of pad is being pressed. Implements touch_api_t::padDataGet , g_touch_on_ctsu. More...
fsp_err_t	RM_TOUCH_ScanStop (touch_ctrl_t *const p_ctrl) Scan stop specified TOUCH control block. Implements touch_api_t::scanStop. More...
fsp_err_t	RM_TOUCH_CallbackSet (touch_ctrl_t *const p_api_ctrl, void(*p_callback)(touch_callback_args_t *), void const *const p_context, touch_callback_args_t *const p_callback_memory)
fsp_err_t	RM_TOUCH_Close (touch_ctrl_t *const p_ctrl) Disables specified TOUCH control block. Implements touch_api_t::close. More...
fsp_err_t	RM_TOUCH_SensitivityRatioGet (touch_ctrl_t *const p_ctrl, touch_sensitivity_info_t *p_touch_sensitivity_info) Get the touch sensitivity ratio. Implements

	touch_api_t::sensitivityRatioGet. More...
fsp_err_t	RM_TOUCH_ThresholdAdjust (touch_ctrl_t *const p_ctrl, touch_sensitivity_info_t *p_touch_sensitivity_info) Adjust the touch judgment threshold. Implements touch_api_t::thresholdAdjust. More...
fsp_err_t	RM_TOUCH_DriftControl (touch_ctrl_t *const p_ctrl, uint16_t input_drift_freq) Control drift correction. Implements touch_api_t::driftControl. More...

Detailed Description

This module supports the Capacitive Touch Sensing Unit (CTSU). It implements the Touch Middleware Interface.

Overview

The Touch Middleware uses the Capacitive Touch Sensing Unit (r_ctsu) API and provides application-level APIs for scanning touch buttons, sliders, and wheels. This module is configured via the QE for Capacitive Touch.

Features

- Supports touch buttons (Self and Mutual), sliders, and wheels
- Can retrieve the status of up to 64 buttons at once
- Software and external triggering
- Callback on scan end
- Collects and calculates usable scan results:
    - Slider position from 1 to 100 (percent)
    - Wheel position from 1 to 360 (degrees)
- Dynamic touch-judgment-threshold adjustment
- Calculate the XY coordinates of the pad(CTSU2)
- Optional (build time) support for real-time monitoring functionality through the QE tool over UART
- Optional (build time) support for tuning function through the QE Standalone Version tool over UART
- TrustZone Support

Configuration

Note

> ***This module is configured via the QE for Capacitive Touch. For information on how to use the QE tool, once the tool is installed click Help -> Help Contents in e2 studio and search for "QE".***

This module supports the QE monitor function. The monitor determines whether to use debugger or serial communications, determines the type of the information from QE and sends only the necessary information. This module supports the serial tuning function with the standalone version of QE. Generates a configuration configuration file by UART communication with QE.

Note

Multiple configurations can be defined within a single project allowing for different scan procedures or button layouts.

Build Time Configurations for rm_touch

The following build time configurations are defined in fsp_cfg/rm_touch_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Support for QE monitoring using UART	• Enabled • Disabled	Disabled	Enable SCI_UART support for QE monitoring.
Support for QE Tuning using UART	• Enabled • Disabled	Disabled	Enable SCI_UART support for QE Tuning.

Configurations for CapTouch > Touch (rm_touch)

This module can be added to the Stacks tab via New Stack > CapTouch > Touch (rm_touch). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description

Interrupt Configuration

Refer to the Capacitive Touch Sensing Unit (r_ctsu) section for details.

Clock Configuration

Refer to the Capacitive Touch Sensing Unit (r_ctsu) section for details.

Pin Configuration

Refer to the Capacitive Touch Sensing Unit (r_ctsu) section for details.

Usage Notes

Measurements and Data Processing

The module determines whether the button has been touched based on the change in capacitance and detects the position of the slider or wheel. This requires continued periodic measurements of capacitance. When developing your application, make sure to periodically call RM_TOUCH_ScanStart() and RM_TOUCH_DataGet(). For more details, refer to the sample application.

Button Touch Determination

Creating reference value and threshold

A touch button is not a mechanical button in which the ON/OF state is switched by hardware. The ON/OFF state is determined via software.
First, a reference value is created based on measurement results in the non-touch state. The initial reference value is the first measured value. The threshold is then determined with an arbitrary offset. If a measured value exceeds the threshold, the button is determined to be in the ON state, if it does not exceed the threshold, it is in the OFF state.
Processing for self-capacitance and mutual capacitance are basically the same. However, because the amount of capacitance decreases when a mutual capacitance button is touched, the user needs to set the threshold based on decreasing measured values to determine the ON/OFF state.
You can set the threshold for each button separately in the configuration settings (threshold in touch_button_cfg_t). The following functions are also included to deal with issues such as chattering suppression and changes in the external environment which affect actual touch recognition.

Positive Noise Filter/Negative Noise Filter

As a chattering countermeasure, you can confirm the ON/OFF state after a set number of consecutive ON or OFF determinations.
In the configuration settings (on_freq and off_freq in touch_cfg_t) set the number of consecutive ON or OFF states. You can do this for all buttons in the touch interface configuration. Be aware that, although this is an effective solution to improving chattering, the greater the number of consecutive states, the slower the response to actual touch.

Hysteresis

This is another chattering countermeasure. Offset the constant to the threshold after the state goes to ON, and prevent chattering by using hysteresis as the OFF-to-ON and ON-to-OFF threshold.
You can set the hysteresis value for each button in the configuration settings (hysteresis in touch_button_cfg_t). The larger the hysteresis, the more effective the countermeasure is in suppressing chattering. However, keep in mind that this will make it more difficult to return the state from ON-to-OFF of OFF-to-ON.

Drift Correction Process

As a countermeasure for changes in the external environment, the drift correction process refreshes the reference value.
After averaging the measured value in the OFF state over a set period, if the button is in the touch OFF state after a set period, the reference value is refreshed. The drift correction is only executed in the OFF state and is cleared when touch ON is determined.
Set the period in the configuration settings (drift_freq in touch_cfg_t). You can do this for all buttons in the touch interface configuration. This allows you to adjust the ability to determine the touch state despite changes in the external environment.

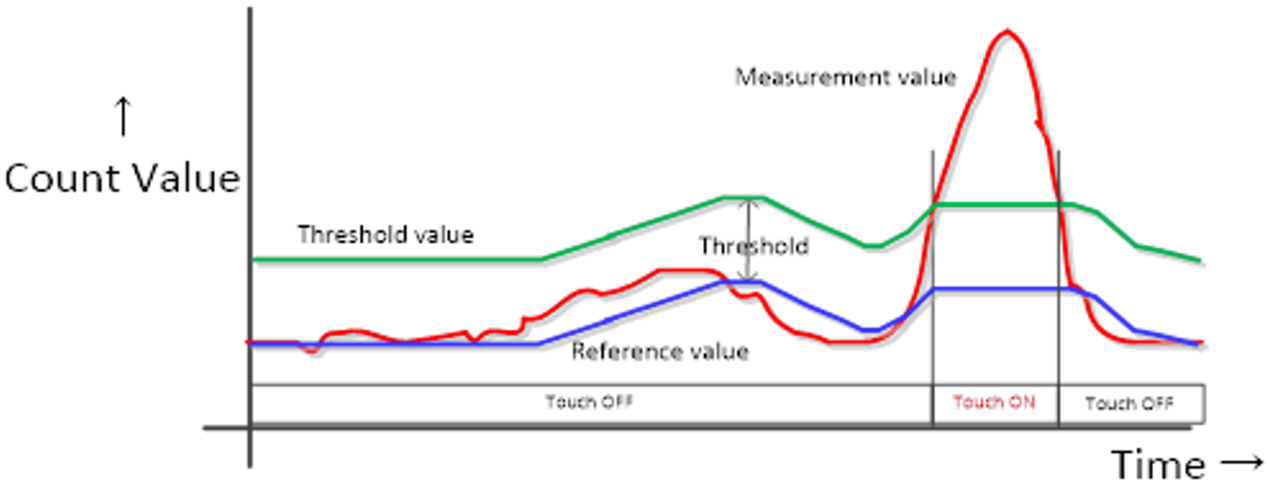


Figure 211: Button Touch Determination

Press and hold cancel

Strong noise or other sudden environment changes can disable the drift correction process, preventing return from the ON state. The press and hold cancel function implements the drift correction process and returns the button from the ON state by forcibly turning the state to OFF after a certain number of consecutive ON state periods.
Set the number of consecutive ON periods required for the press and hold cancel function to return the button to the OFF state in the configuration settings (cancel_freq in touch_cfg_t). You can do this for all buttons in the touch interface configuration.

Touch Position Detection of Slider/Wheel

Configure a slider with multiple terminals to be measured (TS) physically arranged in a straight line.
Configure a wheel with multiple terminals physically arranged in a circle.
The touch position is calculated from the measured values of the TS in the configuration. The calculation method for sliders and wheels is fundamentally the same.

1. Detect the maximum value (TS_MAX) among the terminals in the configuration.
2. Calculate the difference (d1, d2) between TS_MAX and the terminals on either side. (If the TS_MAX terminal is at one end of the slider, use the values of the two terminals to the right or left, accordingly.)
3. If the total of d1 and d2 exceeds the threshold, position calculation is initiated. If the total amount does not exceed the threshold, the position calculation process is ended.
4. With TS_MAX as the middle position, the ratio of d1 to d2 is used to calculate the position. The slider has a range of 1 to 100, and the while has a range of 1 to 360.

	Slider	Wheel
Electrode type	Self capacitance only	Self capacitance only
Number of electrodes	3-10	4+
Touch position output range	1-100	1-360
Default value (no touch)	0xFFFF	0xFFFF

Tuning the Touch Determination Adjustment

When QE tuning, a measurement is performed with a finger touching the button and the tuned parameters are output in the configuration file. The setting value of the threshold is 60% of the touch sensitivity between touch and non-touch state, and the setting value of the hysteresis coefficient is 5% of the threshold.
This module provides the functions for dynamic adjusting of these threshold and hysteresis coefficient.
They are two functions as below.

Adjusting the threshold and hysteresis coefficient to an arbitrary ratio.
Use RM_TOUCH_ThresholdAdjust().

When changing the touch determination threshold ratio from 60% QE set to 70% user specified, the touch determination thresholds are as below.
If you want to make this setting, set the member of the second argument as follows. It is also necessary to set the ratio of the amount of touch change and the hysteresis value.

*p_touch_sensitivity_ratio = 100,
old_threshold_ratio = 60,
new_threshold_ratio = 70,
new_hysteresis_ratio = 5

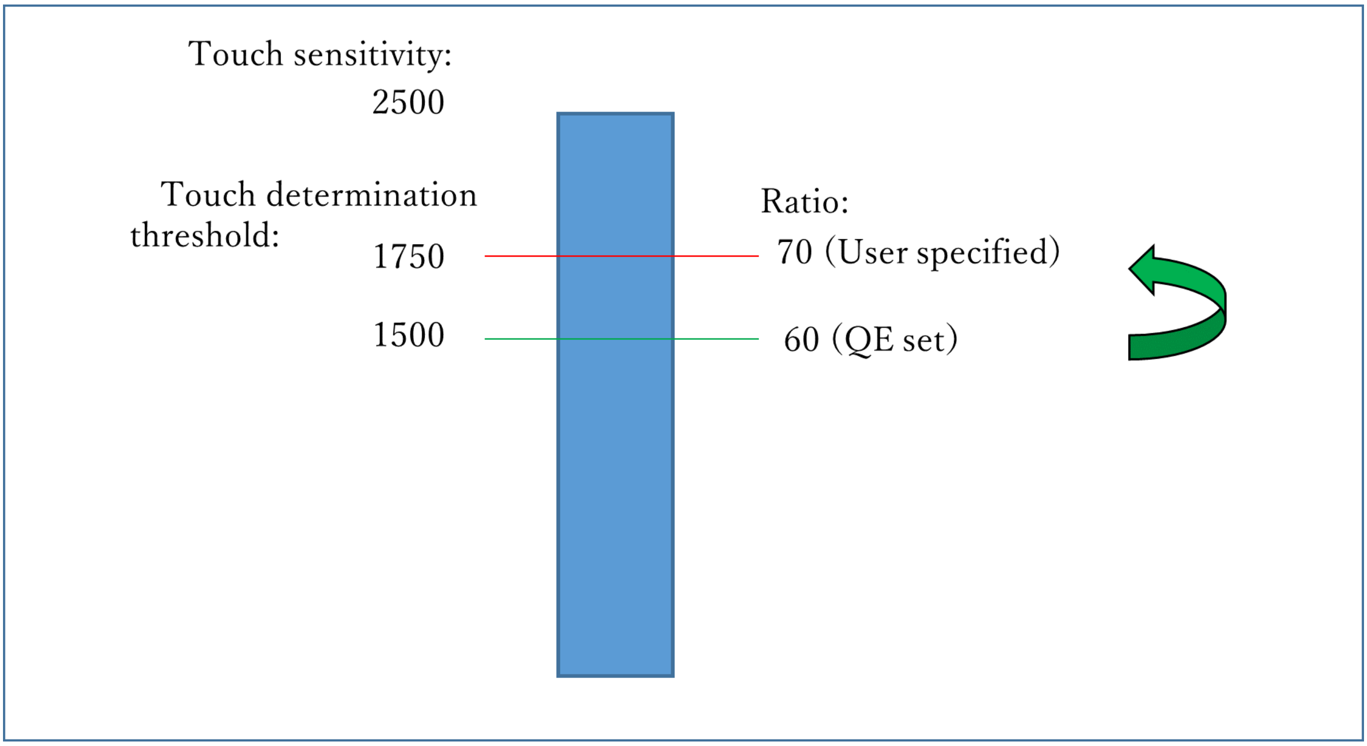


Figure 212: Example of changing the threshold ratio

Adjusting the threshold and hysteresis coefficient according to the current touch sensitivity.
Use RM_TOUCH_SensitivityRatioGet(), RM_TOUCH_ThresholdAdjust(), and RM_TOUCH_DriftControl().

When changing the kind of the overlay panel, the touch sensitivity differs from the one QE tuned. Wanting to use the software as it is without re-tuning. If you use a thicker overlay than that at QE tuning, the touch sensitivity decreases, and a touch may not be determined because of the same touch determination threshold. This function adjusts the touch determination threshold based on the ratio of the touch sensitivity after changing the overlay to the touch sensitivity at the QE tuning.

RM_TOUCH_SensitivityRatioGet() outputs the ratio of the current touch sensitivity assuming that the touch sensitivity at the QE setting is 100%.
The following figure shows the case where an overlay panel is thinner and the touch sensitivity increases.

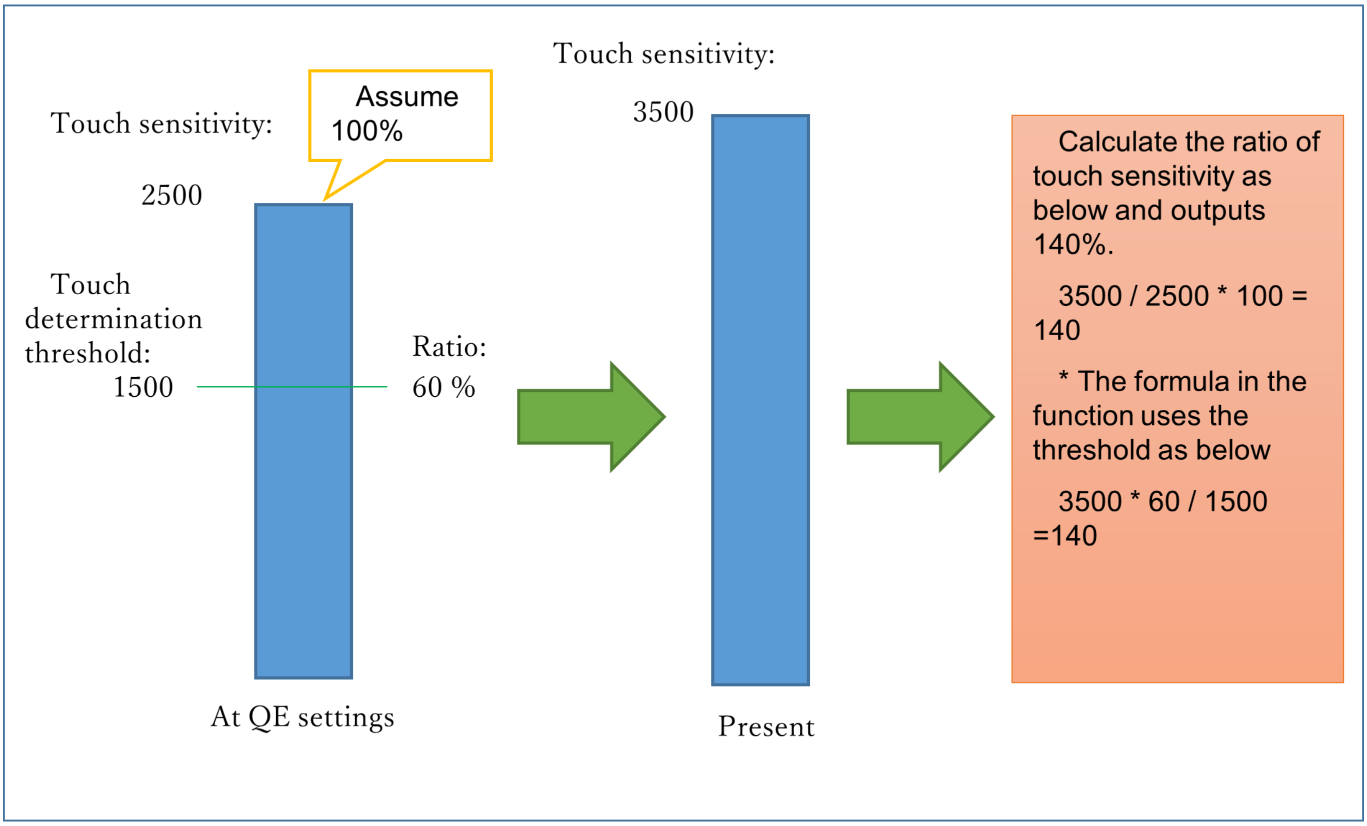


Figure 213: Example of increase touch sensitivity for thin overlay panels

Following figure shows the case where an overlay panel is thicker and the touch sensitivity decreases.

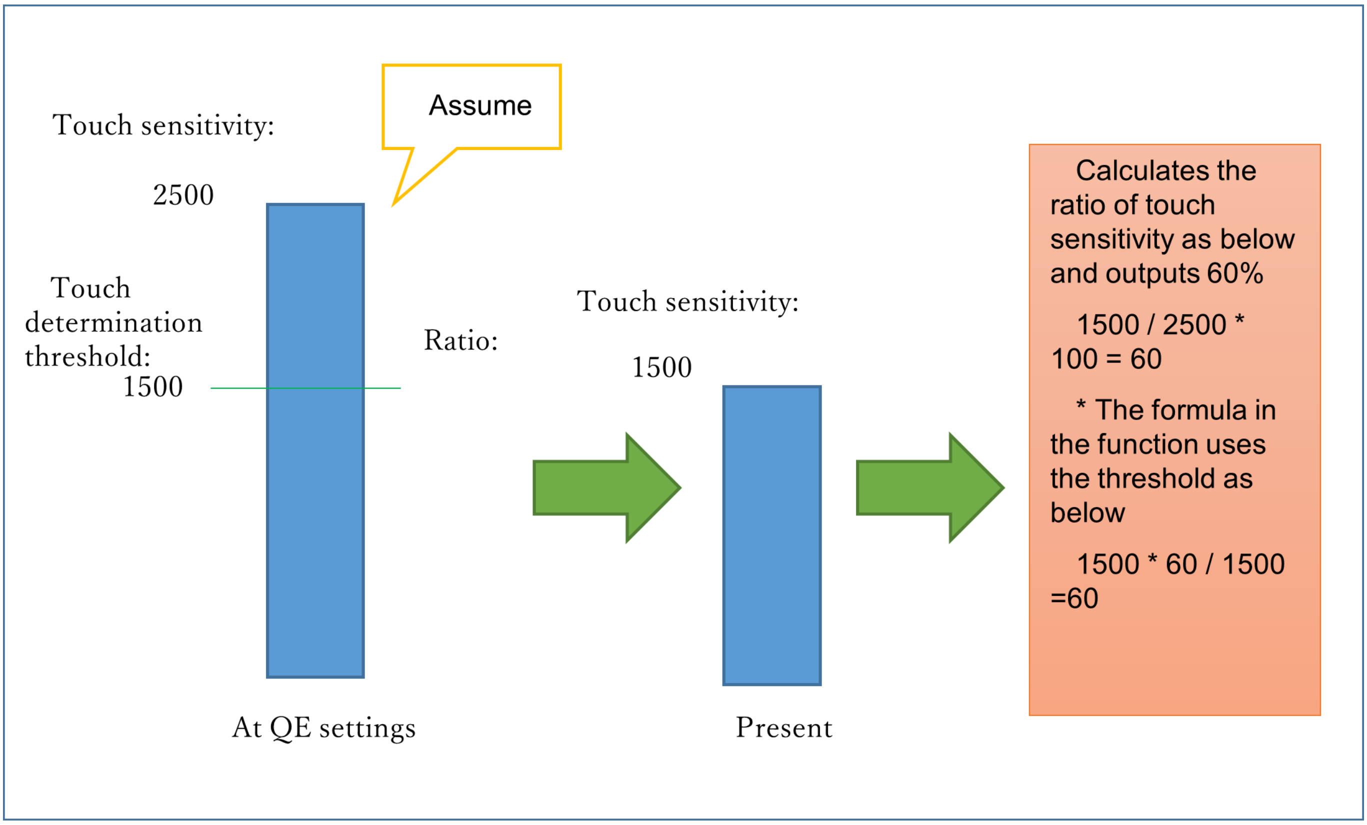


Figure 214: Example of decrease touch sensitivity for thicker overlay panels

RM_TOUCH_ThresholdAdjust() sets the new touch determination threshold and the hysteresis value by using the touch sensitivity ratio obtained with RM_TOUCH_SensitivityRatioGet() as arguments.

Example of calculation 1:
The touch sensitivity ratio is 140%, and the threshold set by QE is 1500.
Threshold = 140 * 1500 / 100 = 2100

*p_touch_sensitivity_ratio = 140,
old_threshold_ratio = 60,
new_threshold_ratio = 60,
new_hysteresis_ratio = 5

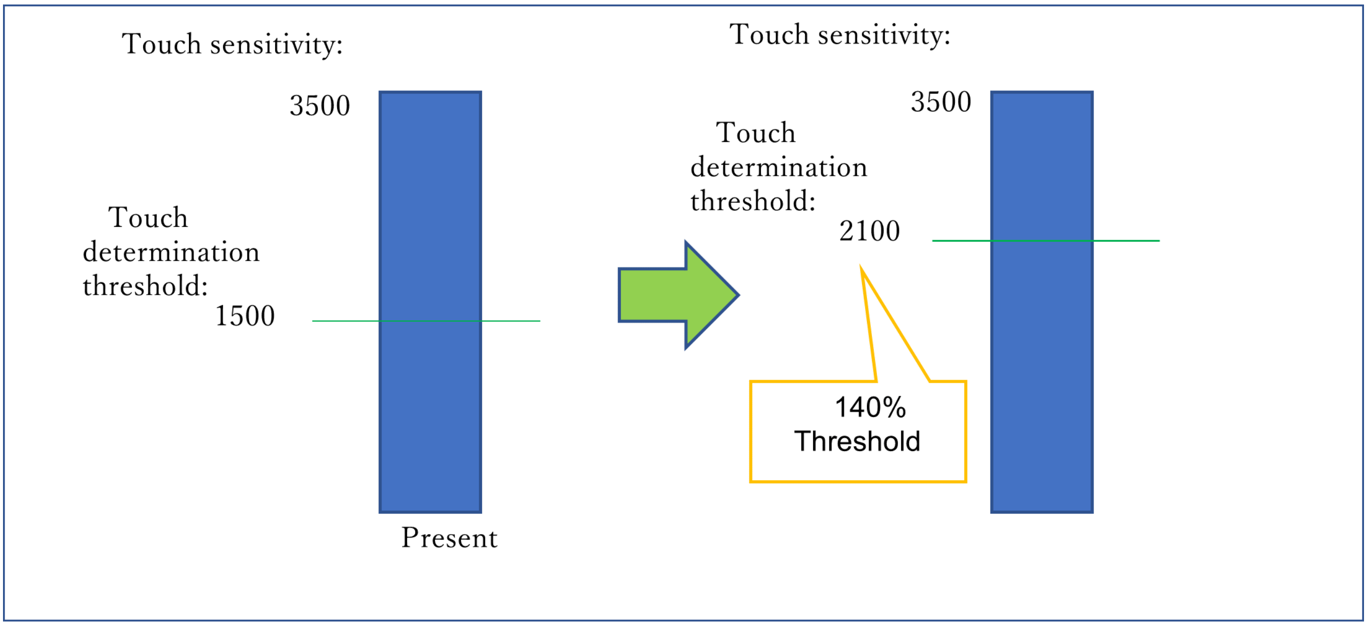


Figure 215: Example of calculation 1

Example of calculation 2:
The touch sensitivity ratio is 60%, and the threshold set by QE is 1500.
Threshold = 60 * 1500 / 100 = 900

*p_touch_sensitivity_ratio = 60,
old_threshold_ratio = 60,
new_threshold_ratio = 60,
new_hysteresis_ratio = 5

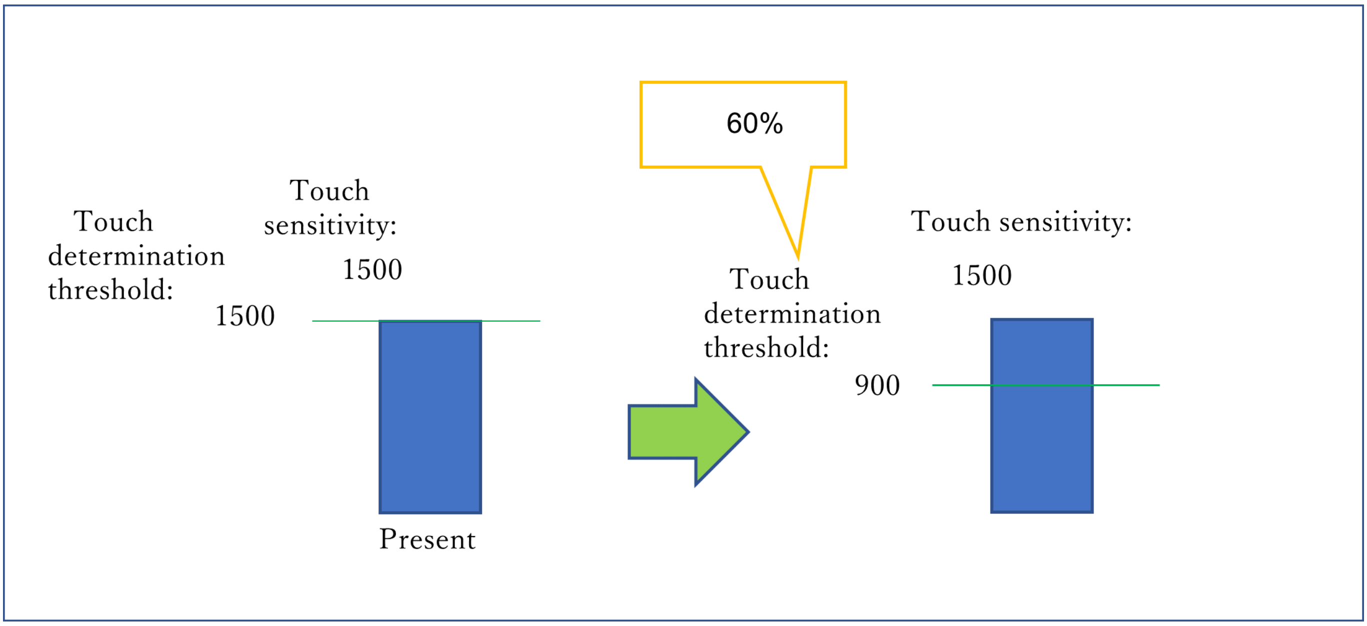


Figure 216: Example of calculation 2

RM_TOUCH_DriftContorol() set the second argument to 0 to stop the drift correction function.

When calculating the ratio of the touch change amount using RM_TOUCH_SensitivityRatioGet(), the touch change amount decreases due to the thick overlay, and the threshold value is not exceeded even if touched. Prevents the reference value from drifting.

Example of the application for adjustment using data flash without re-tuning or software rewriting

Enable UART communication to PC and 'tuning mode'. In tuning mode, the MCU transmits the ratio of the touch sensitivity in the touch state to the PC in real time. A user sends a command to decide the ratio while monitoring on the PC. The MCU stores the received ratio in the data flash. Make sure that the ratio stored in the data flash is read at the software activation, and the touch determination threshold is adjusted based on this stored value.

Pad

Configure a pad with multiple terminals physically arranged in cross.
The current position is Calculated from the measured values of the CTSU mutual scanning in the configuration.
Use RM_TOUCH_PadDataGet().
Pad is subject so some limitations:

	Pad
Electrode type	CFC mutual capacitance only
Number of electrodes	RX(TS-CFC)3+, TX(Any TS)3+
Touch position output range	rx_coodinate:(0 ~ rx_pixel), tx_coodinate:(0 ~ tx_pixsel)
Default value (no touch)	rx_coodinate:0xFFFF, tx_coodinate:0xFFFF
Pixel range	rx_pixel:(1 ~ 65535), tx_pixsel:(1 ~ 65535)

Pitch for each terminal can be set with QE. Pitch's default value is 64.
The relationship between pixel and pitch : Pixel = Pitch × number of TS - 1

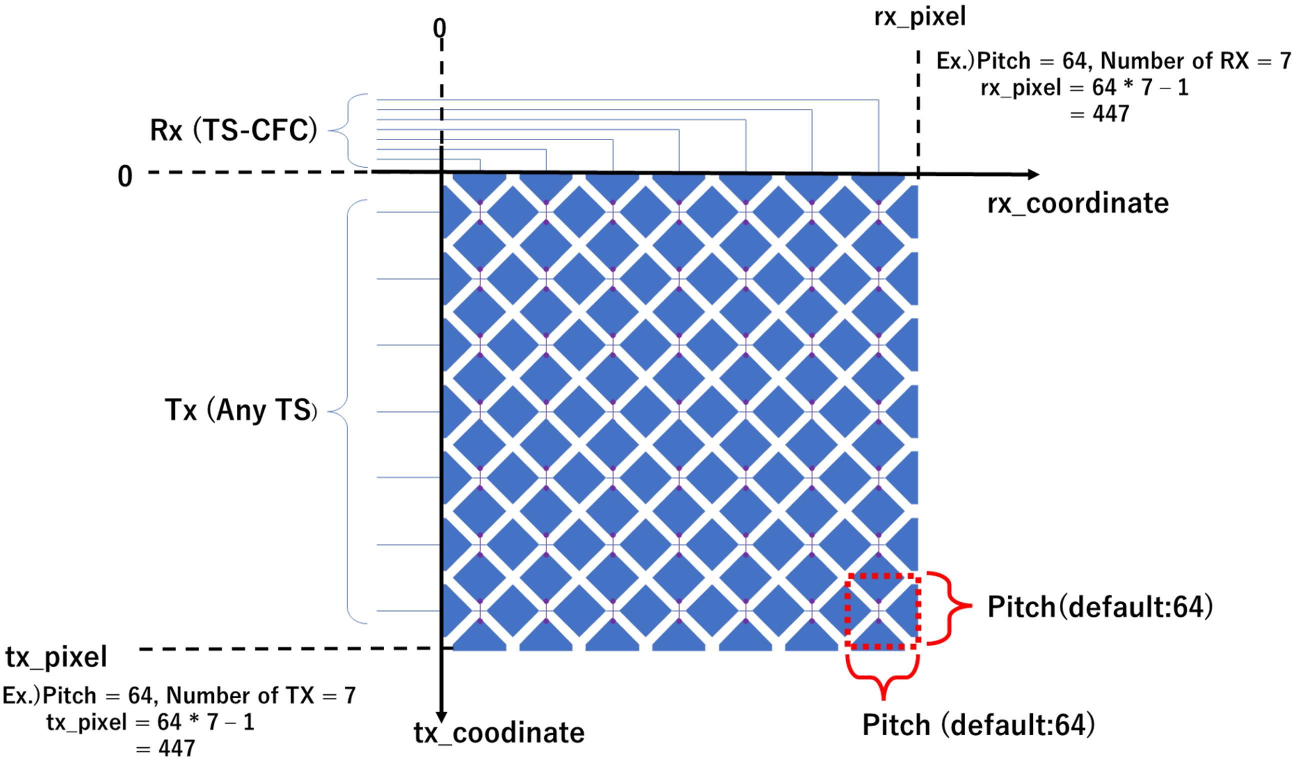


Figure 217: Example of Pad

TrustZone Support

In r_ctsu and rm_touch module, Non-Secure Callable Guard Functions are only generated from QE for Capacitive Touch. QE can be used for tuning in secure or flat project, but not in non-secure project. If you want to use in non-secure project, copy the output file from secure or flat project. Refer to QE Help for more information.

Examples

Basic Example

This is a basic example of minimal use of the TOUCH in an application.

```
void touch_basic_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_cfg);
 /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);
 while (true)
    {
 RM_TOUCH_ScanStart(&g_touch_ctrl);
```

```
 while (0 == g_flag)
      {
 /* Wait scan end callback */
       }
       g_flag = 0;
       err = RM_TOUCH_DataGet(&g_touch_ctrl, &button, slider, wheel);
 if (FSP_SUCCESS == err)
      {
 /* Application specific data processing. */
       }
    }
}
```

Multi Mode Example

This is a optional example of using both Self-capacitance and Mutual-capacitance. Refer to the Multi Mode Example in CTSU usage notes.

```
void touch_optional_example (void)
{
 fsp_err_t err = FSP_SUCCESS;
    err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_cfg);
    assert(FSP_SUCCESS == err);
    err = RM_TOUCH_Open(&g_touch_ctrl_mutual, &g_touch_cfg_mutual);
    assert(FSP_SUCCESS == err);
 while (true)
    {
 RM_TOUCH_ScanStart(&g_touch_ctrl);
 while (0 == g_flag)
      {
 /* Wait scan end callback */
       }
       g_flag = 0;
 RM_TOUCH_ScanStart(&g_touch_ctrl_mutual);
 while (0 == g_flag)
```

```
     {
/* Wait scan end callback */
      }
      g_flag = 0;
      err = RM_TOUCH_DataGet(&g_touch_ctrl, &button, slider, wheel);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
      }
      err = RM_TOUCH_DataGet(&g_touch_ctrl_mutual, &button, slider, wheel);
if (FSP_SUCCESS == err)
     {
/* Application specific data processing. */
      }
   }
}
```

Data Structures

struct	touch_button_info_t
struct	touch_slider_info_t
struct	touch_wheel_info_t
struct	touch_pad_info_t
struct	touch_instance_ctrl_t

Data Structure Documentation

◆ touch_button_info_t

struct touch_button_info_t

Information of button

Data Fields		
uint64_t	status	Touch result bitmap.
uint16_t *	p_threshold	Pointer to Threshold value array. g_touch_button_threshold[] is set by Open API.

uint16_t *	p_hysteresis	Pointer to Hysteresis value array. g_touch_button_hysteresis[] is set by Open API.
uint16_t *	p_reference	Pointer to Reference value array. g_touch_button_reference[] is set by Open API.
uint16_t *	p_on_count	Continuous touch counter. g_touch_button_on_count[] is set by Open API.
uint16_t *	p_off_count	Continuous non-touch counter. g_touch_button_off_count[] is set by Open API.
uint32_t *	p_drift_buf	Drift reference value. g_touch_button_drift_buf[] is set by Open API.
uint16_t *	p_drift_count	Drift counter. g_touch_button_drift_count[] is set by Open API.
uint8_t	on_freq	Copy from config by Open API.
uint8_t	off_freq	Copy from config by Open API.
uint16_t	drift_freq	Copy from config by Open API.
uint16_t	cancel_freq	Copy from config by Open API.

◆ touch_slider_info_t

struct touch_slider_info_t		
Information of slider		
Data Fields		
uint16_t *	p_position	Calculated Position data. g_touch_slider_position[] is set by Open API.
uint16_t *	p_threshold	Copy from config by Open API. g_touch_slider_threshold[] is set by Open API.

◆ touch_wheel_info_t

struct touch_wheel_info_t		
Information of wheel		
Data Fields		
uint16_t *	p_position	Calculated Position data. g_touch_wheel_position[] is set by Open API.

uint16_t *	p_threshold	Copy from config by Open API. g_touch_wheel_threshold[] is set by Open API.

◆ touch_pad_info_t

struct touch_pad_info_t		
Information of pad		
Data Fields		
uint16_t *	p_rx_coordinate	RX coordinate.
uint16_t *	p_tx_coordinate	TX coordinate.
uint16_t *	p_num_touch	number of touch
uint16_t *	p_threshold	Coordinate calculation threshold value.
uint16_t *	p_base_buf	ScanData Base Value Buffer.
uint16_t *	p_rx_pixel	X coordinate resolution.
uint16_t *	p_tx_pixel	Y coordinate resolution.
uint8_t *	p_max_touch	Maximum number of touch judgments used by the pad.
int32_t *	p_drift_buf	Drift reference value. g_touch_button_drift_buf[] is set by Open API.
uint16_t *	p_drift_count	Drift counter. g_touch_button_drift_count[] is set by Open API.
uint8_t	num_drift	Copy from config by Open API.

◆ touch_instance_ctrl_t

struct touch_instance_ctrl_t		
TOUCH private control block. DO NOT MODIFY. Initialization occurs when RM_TOUCH_Open() is called.		
Data Fields		
uint32_t	open	Whether or not driver is open.
touch_button_info_t	binfo	Information of button.
touch_slider_info_t	sinfo	Information of slider.
touch_wheel_info_t	winfo	Information of wheel.
bool	serial_tuning_enable	Flag of serial tuning status.
touch_pad_info_t	pinfo	Information of pad.
touch_cfg_t const *	p_touch_cfg	Pointer to initial configurations.
ctsu_instance_t const *	p_ctsu_instance	Pointer to CTSU instance.

Function Documentation

◆ RM_TOUCH_Open()

fsp_err_t RM_TOUCH_Open (touch_ctrl_t *const *p_ctrl*, touch_cfg_t const *const *p_cfg*)

Opens and configures the TOUCH Middle module. Implements touch_api_t::open.

Example:

```
err = RM_TOUCH_Open(&g_touch_ctrl, &g_touch_cfg);
```

Return values

FSP_SUCCESS	TOUCH successfully configured.
FSP_ERR_ASSERTION	Null pointer, or one or more configuration options is invalid.
FSP_ERR_ALREADY_OPEN	Module is already open. This module can only be opened once.
FSP_ERR_INVALID_ARGUMENT	Configuration parameter error.

◆ RM_TOUCH_ScanStart()

fsp_err_t RM_TOUCH_ScanStart (touch_ctrl_t *const *p_ctrl*)

This function should be called each time a periodic timer expires. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Before starting the next scan, first get the data with RM_TOUCH_DataGet(). If a different control block scan should be run, check the scan is complete before executing. Implements touch_api_t::scanStart.

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CTSU_SCANNING	Scanning this instance or other.
FSP_ERR_CTSU_NOT_GET_DATA	The previous data has not been retrieved by DataGet.

◆ RM_TOUCH_DataGet()

fsp_err_t RM_TOUCH_DataGet (touch_ctrl_t *const *p_ctrl*, uint64_t * *p_button_status*, uint16_t * *p_slider_position*, uint16_t * *p_wheel_position*)

Gets the 64-bit mask indicating which buttons are pressed. Also, this function gets the current position of where slider or wheel is being pressed. If initial offset tuning is enabled, The first several calls are used to tuning for the sensors. Implements touch_api_t::dataGet.

Return values

FSP_SUCCESS	Successfully data decoded.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CTSU_SCANNING	Scanning this instance.
FSP_ERR_CTSU_INCOMPLETE_TUNING	Incomplete initial offset tuning.

◆ RM_TOUCH_PadDataGet()

fsp_err_t RM_TOUCH_PadDataGet (touch_ctrl_t *const *p_ctrl*, uint16_t * *p_pad_rx_coordinate*, uint16_t * *p_pad_tx_coordinate*, uint8_t * *p_pad_num_touch*)

This function gets the current position of pad is being pressed. Implements touch_api_t::padDataGet , g_touch_on_ctsu.

Return values

FSP_SUCCESS	Successfully data decoded.
FSP_ERR_ASSERTION	Null pointer.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CTSU_SCANNING	Scanning this instance.

◆ RM_TOUCH_ScanStop()

fsp_err_t RM_TOUCH_ScanStop (touch_ctrl_t *const *p_ctrl*)

Scan stop specified TOUCH control block. Implements touch_api_t::scanStop.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_TOUCH_CallbackSet()

fsp_err_t RM_TOUCH_CallbackSet (touch_ctrl_t *const *p_api_ctrl*, void(*)(touch_callback_args_t *) *p_callback*, void const *const *p_context*, touch_callback_args_t *const *p_callback_memory*)

Updates the user callback and has option of providing memory for callback structure. Implements touch_api_t::callbackSet

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.

◆ RM_TOUCH_Close()

fsp_err_t RM_TOUCH_Close (touch_ctrl_t *const *p_ctrl*)

Disables specified TOUCH control block. Implements touch_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_TOUCH_SensitivityRatioGet()

fsp_err_t RM_TOUCH_SensitivityRatioGet (touch_ctrl_t *const *p_ctrl*, touch_sensitivity_info_t * *p_touch_sensitivity_info*)

Get the touch sensitivity ratio. Implements touch_api_t::sensitivityRatioGet.

Return values

FSP_SUCCESS	Successfully touch sensitivity ratio got.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_CTSU_SCANNING	Scanning this instance.
FSP_ERR_CTSU_INCOMPLETE_TUNING	Incomplete initial offset tuning.

◆ RM_TOUCH_ThresholdAdjust()

fsp_err_t RM_TOUCH_ThresholdAdjust (touch_ctrl_t *const *p_ctrl*, touch_sensitivity_info_t * *p_touch_sensitivity_info*)

Adjust the touch judgment threshold. Implements touch_api_t::thresholdAdjust.

Return values

FSP_SUCCESS	Successfully touch judgment threshold was adjusted.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_TOUCH_DriftControl()

fsp_err_t RM_TOUCH_DriftControl (touch_ctrl_t *const *p_ctrl*, uint16_t *input_drift_freq*)

Control drift correction. Implements touch_api_t::driftControl.

Return values

FSP_SUCCESS	Successfully drift correction was controlled.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

4.2.116 USBX Porting Layer (rm_usbx_port)

Modules

Functions

Refer to USB (r_usb_basic) for the common API (r_usb_basic) to be called from the application.

Overview

This USB driver works by combining USBX and r_usb_basic module.

How to Configuration

- Using a class other than HMSC. The following describes how to configure USBX using PCDC as an example.

1.Select [New Stack]->[Azure RTOS]->[USBX]->[PCDC]

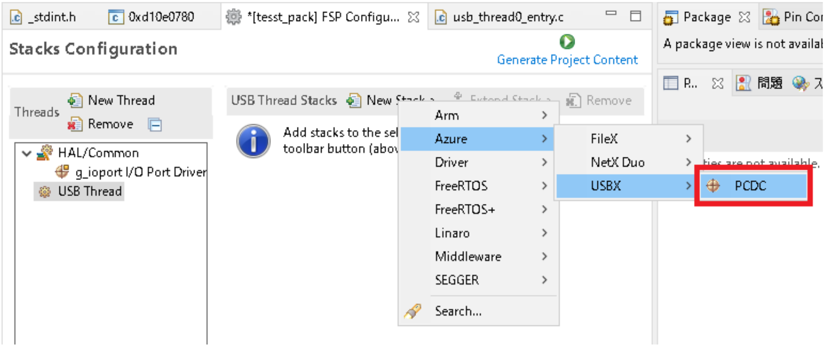


Figure 218: Select USB Device Class

2.The following is displayed when selecting USBX PCDC.

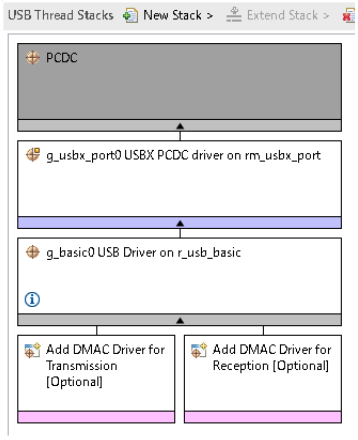


Figure 219: USBX PCDC Stack

3.Select the USB pipe to use.

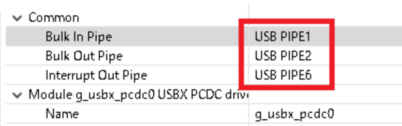


Figure 220: Select using USB Pipe

- Using HMSC. Since HMSC is used in a different way from other USBX modules, the usage is described below.
  1.Select [New Stack]->[Azure RTOS]->[FileX]->[FileX on USBX]

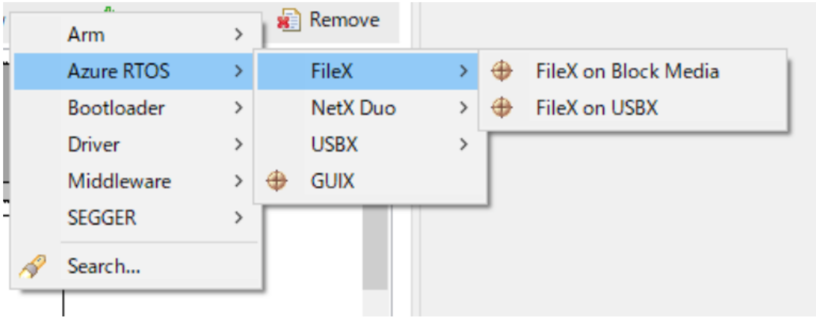


Figure 221: Select USB Device Class

2.The following is displayed when selecting Filex on USBX.

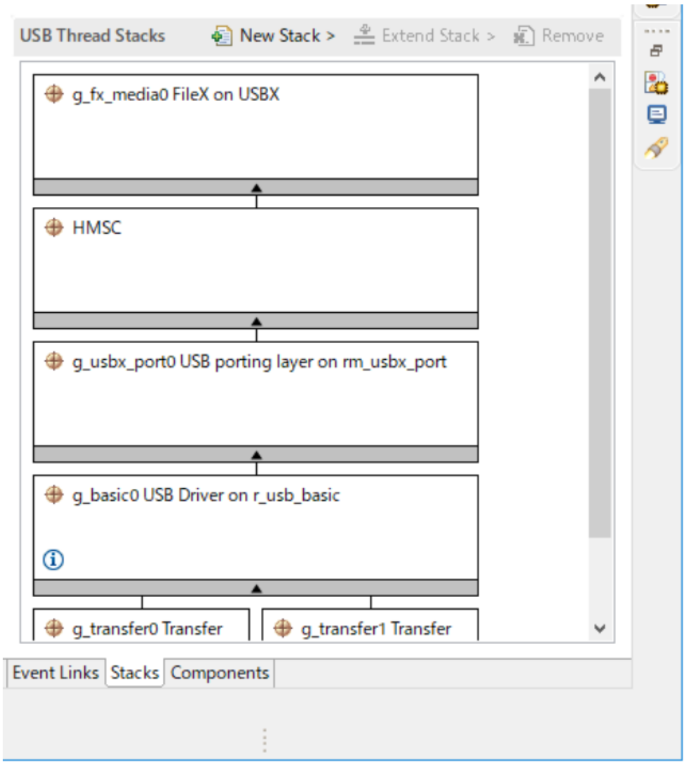


Figure 222: FileX on USBX Stack

Note

- The following are notes on using the ux_host_class_cdc_acm_read function or the ux_device_class_cdc_acm_read function.

1. Please specify a multiple of MaxPacketSize to the 3rd argument(requested_length). since If the value of the 3rd argument is not multiple of MaxPacketSize, all data sent by USB Host or USB Peripheral may not be received correctly.
2. Please specify start address of thr area allocated a size larger than 3rd argument(requested_length) to the 2nd argument(data_pointer or buffer).

- HMSC uses FileX. For more information on FileX, please refer to the following URL.

https://docs.microsoft.com/en-us/azure/rtos/filex/

Limitations

1.Please call the initialization function in the application program. Please be sure to call R_USB_Open function after calling the following functions.

- Peripheral
  - PCDC / PHID / PAUD
    - ux_system_initialize
    - ux_device_stack_initialize
    - ux_device_stack_class_register
  - PMSC
    - ux_system_initialize
    - ux_device_stack_initialize
- Host
  - HCDC / HHID
    - ux_system_initialize
    - ux_host_stack_initialize
  - HMSC
    - ux_system_initialize
    - ux_host_stack_initialize
    - fx_system_initialize

2.Set the value for 1000 Ticks per second in the "Timer Ticks Per Second" item.

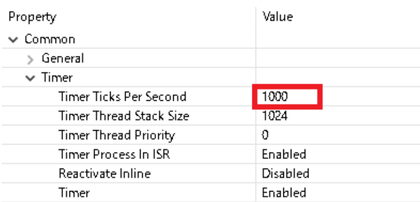


Figure 223: Specify value of Timer Ticks Per Second

3.Set the Thread priority to a value of 21.

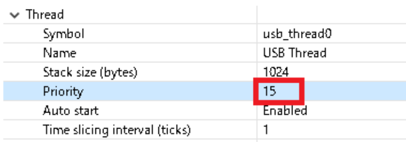


Figure 224: Specify value of Thread priority

4.When using the following functions, call the following functions in the following order after calling the R_USB_Close function. NOTE : Do not call R_USB_Close function in Peripheral Audio Class.

- Peripheral
  - PCDC / PHID
    - _ux_device_stack_class_unregister
    - _ux_device_stack_uninitialize
  - PMSC

- _ux_device_stack_uninitialize
- Host
  - _ux_host_stack_uninitialize

5.The USBX HID class does not support OUT transfer.

6.When using USBX HHID, please use Wired device.

7.This module does not support the MCU(RA4M1 and RA2A1) since the RAM size is small.

Descriptor

Templates for USBX descriptors can be found in ra/fsp/src/rm_usbx_port folder. Also, please be sure to use your vendor ID.
Change the descriptor.c.template file for each class as follows if High-speed mode is used.

- rm_usbx_pcdc_descriptor.c.template
  1. Comment on lines 108 and 243.
  2. Delete the "//" on lines 109 and 242.
- rm_usbx_pmsc_descriptor.c.template
  1. Comment on lines 78 and 153.
  2. Delete the "//" on lines 79 and 152.
- rm_usbx_paud_descriptor.c.template
  1. Comment on lines 167 and 485.
  2. Delete the "//" on lines 168 and 486.

There are two types of templates for PHID descriptor.
Keyboard templates should be referred to rm_usbx_phid_descriptor_keyboard.c.template.
Mouse templates should be referred to rm_usbx_phid_descriptor_mouse.c.template.

Examples

USBX PCDC Example

PCDC loopback example is as follows.

```
#define VALUE_105 (105)
#define VALUE_2 (2)
#define VALUE_103 (103)
#define VALUE_93 (93)
/******************************************************************************
 * Function Name : ux_cdc_device0_instance_activate
 * Description : Get instance
 * Arguments : void * cdc_instance : Pointer to the area store the instance pointer
 * Return value : none
```

```
 ******************************************************************************/
static void ux_cdc_device0_instance_activate (void * cdc_instance)
{
 /* Save the CDC instance. */
    g_cdc = (UX_SLAVE_CLASS_CDC_ACM *) cdc_instance;
}
/******************************************************************************
 * End of function ux_cdc_device0_instance_activate
 ******************************************************************************/
/******************************************************************************
 * Function Name : ux_cdc_device0_instance_deactivate
 * Description : Clear instance
 * Arguments : void * cdc_instance : Pointer to area store the instance pointer
 * Return value : none
 ******************************************************************************/
static void ux_cdc_device0_instance_deactivate (void * cdc_instance)
{
 FSP_PARAMETER_NOT_USED(cdc_instance);
    g_cdc = UX_NULL;
}
/******************************************************************************
 * End of function ux_cdc_device0_instance_deactivate
 ******************************************************************************/
/******************************************************************************
 * Function Name : apl_status_change_cb
 * Description : USB callback function for USB status change
 * Arguments : ULONG status : USB status
 * Return value : UX_SUCCESS
 ******************************************************************************/
UINT apl_status_change_cb (ULONG status)
{
 switch (status)
    {
 case UX_DEVICE_ATTACHED:
```

```
            g_attach = USB_YES;
 break;
 case UX_DEVICE_REMOVED:
            g_attach = USB_NO;
 break;
 default:
 break;
    }
 return UX_SUCCESS;
}
/*******************************************************************************
 * End of function apl_status_change_cb
 *******************************************************************************/
/*******************************************************************************
 * Function Name : usbx_pcdc_sample
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 *******************************************************************************/
void usbx_pcdc_sample (void)
{
 fsp_err_t err;
    uint32_t ret;
    uint32_t size;
    _ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    _ux_device_stack_initialize(g_device_framework_hi_speed,
                               VALUE_103,
                               g_device_framework_full_speed,
                               VALUE_93,
                               g_string_framework,
                               VALUE_105,
                               g_language_id_framework,
                               VALUE_2,
                               apl_status_change_cb);
```

```
    g_ux_device_class_cdc_acm0_parameter.ux_slave_class_cdc_acm_instance_activate   =
ux_cdc_device0_instance_activate;
    g_ux_device_class_cdc_acm0_parameter.ux_slave_class_cdc_acm_instance_deactivate =
       ux_cdc_device0_instance_deactivate;
    _ux_device_stack_class_register(_ux_system_slave_class_cdc_acm_name,
_ux_device_class_cdc_acm_entry, 1, 0x00,
                                    (void *) &g_ux_device_class_cdc_acm0_parameter);
    err = g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS == err)
    {
 while (1)
      {
 if (USB_YES == g_attach)
      {
 while (g_cdc == UX_NULL)
      {
                   ;
      }
                ret = _ux_device_class_cdc_acm_read(g_cdc, g_buf, DATA_LEN,
&g_actual_length);
 if (UX_SUCCESS == ret)
      {
                   size = g_actual_length;
                   _ux_device_class_cdc_acm_write(g_cdc, g_buf, size,
&g_actual_length);
      }
      }
      }
    }
}
/*******************************************************************************
 * End of function usbx_pcdc_sample
 *******************************************************************************/
```

USBX HCDC Example

The main functions of the HCDC loopback example are as follows:

1. Virtual UART control settings are configured by transmitting the class request SET_LINE_CODING to the CDC device.
2. Sends receive (Bulk In transfer) requests to a CDC peripheral device and receives data.
3. Loops received data back to the peripheral by means of Bulk Out transfers.

The main loop performs loopback processing in which data received from a CDC peripheral device is transmitted unaltered back to the peripheral.

```
 #define VALUE_100 (100)
UINT ux_host_usr_event_notification (ULONG event, UX_HOST_CLASS * host_class, VOID *
instance)
{
 if (_ux_utility_memory_compare(_ux_system_host_class_cdc_acm_name, host_class,

_ux_utility_string_length_get(_ux_system_host_class_cdc_acm_name)) ==
       UX_SUCCESS)
    {
 if (event == UX_FSP_DEVICE_INSERTION) /* Check if there is a device insertion. */
      {
            p_cdc_acm = (UX_HOST_CLASS_CDC_ACM *) instance;
 if
(p_cdc_acm->ux_host_class_cdc_acm_interface->ux_interface_descriptor.bInterfaceClass
!=
                UX_HOST_CLASS_CDC_DATA_CLASS)
      {
 /* It seems the DATA class is on the second interface. Or we hope ! */
                p_cdc_acm = p_cdc_acm->ux_host_class_cdc_acm_next_instance;
 /* Check again this interface, if this is not the data interface, we give up. */
 if
(p_cdc_acm->ux_host_class_cdc_acm_interface->ux_interface_descriptor.bInterfaceClass
!=
                  UX_HOST_CLASS_CDC_DATA_CLASS)
      {
 /* We did not find a proper data interface. */
                   p_cdc_acm = UX_NULL;
```

```
      }
      }
 if (p_cdc_acm != UX_NULL)
      {
                tx_event_flags_set(&g_cdcacm_activate_event_flags0, CDCACM_FLAG,
TX_OR);
      }
      }
 else if (event == UX_FSP_DEVICE_REMOVAL) /* Check if there is a device removal. */
      {
            tx_event_flags_set(&g_cdcacm_activate_event_flags0, ~CDCACM_FLAG,
TX_AND);
            p_cdc_acm = UX_NULL;
      }
 else
      {
      }
    }
 return UX_SUCCESS;
}
void buffer_clear (uint8_t * p)
{
    uint16_t counter;
 for (counter = 0; counter < DATA_LEN; counter++)
    {
       *p = 0U;
    }
}
/*******************************************************************************
 * Function Name : usbx_hcdc_sample
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 *******************************************************************************/
```

```
/* CDCACM Host Thread entry function */
void usbx_hcdc_sample (void)
{
    uint32_t status;
    ULONG    actual_flags;
    uint16_t counter = 0;
 for (counter = 0; counter < DATA_LEN; counter++)
    {
       g_write_buf[counter] = (uint8_t) counter;
    }
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_host_stack_initialize(ux_host_usr_event_notification);
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
       tx_event_flags_get(&g_cdcacm_activate_event_flags0, CDCACM_FLAG, TX_OR,
&actual_flags, TX_WAIT_FOREVER);
 if (p_cdc_acm != UX_NULL)
      {
 if (0 == g_is_communicate)
      {
                tx_thread_sleep(VALUE_100);
                g_is_communicate = 1;
      }
            status = ux_host_class_cdc_acm_write(p_cdc_acm, g_write_buf, DATA_LEN,
&g_write_actual_length);
 if ((UX_SUCCESS == status) && (DATA_LEN == g_write_actual_length))
      {
                g_read_actual_length = 0;
                buffer_clear(g_read_buf);
                status = ux_host_class_cdc_acm_read(p_cdc_acm, g_read_buf, DATA_LEN,
&g_read_actual_length);
 if ((UX_SUCCESS == status) && (DATA_LEN == g_read_actual_length))
      {
```

```
 for (counter = 0; counter < DATA_LEN; counter++)
      {
 if ((uint8_t) counter != g_read_buf[counter])
      {
 while (1)
      {
                     ;
      }
      }
      }
      }
      }
      }
    }
}
```

USBX PMSC Example

PMSC storage example is as follows.

```
const rm_block_media_cfg_t g_rm_block_media0_cfg =
{.p_extend = NULL, .p_callback = NULL, .p_context = NULL, };
rm_block_media_instance_t g_rm_block_media0 =
{.p_api = &g_rm_block_media_on_user_media, .p_ctrl = NULL, .p_cfg =
&g_rm_block_media0_cfg, };
/*******************************************************************************
 * Function Name : usbx_pmsc_sample
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 *******************************************************************************/
void usbx_pmsc_sample (void)
{
 fsp_err_t err;
    UINT      ret;
```

```
    ULONG    size;
    _ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    _ux_device_stack_initialize(
       g_device_framework_hi_speed,
       60,
       g_device_framework_full_speed,
       50,
       g_string_framework,
       93,
       g_language_id_framework,
       2,
       UX_NULL
       );
    err = g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS == err)
    {
 while (1)
      {
            ;
      }
    }
}
```

USBX HMSC Example

HMSC storage example is as follows. See usbx_hmsc_sample for the basic operation of HMSC. Also, please refer to usbx_hmsc_sample_format for the format of the USB memory.

```
#define UX_STORAGE_BUFFER_SIZE (64 * 1024)
#define EVENT_USB_PLUG_IN (1UL << 0)
#define EVENT_USB_PLUG_OUT (1UL << 1)
#define MEMPOOL_SIZE (63488)
#define DATA_LEN (2048)
#define VALUE_32 (32)
#define VALUE_100 (100)
```

```
#define VALUE_200 (200)
#define VALUE_256 (256)
#define VALUE_1024 (1024)
#define DEVICE_INSERTION (0x01U)
#define DEVICE_REMOVAL (0x02U)
static uint16_t g_read_buf[UX_STORAGE_BUFFER_SIZE];
static uint16_t g_write_buf[UX_STORAGE_BUFFER_SIZE];
static FX_FILE g_file;
static UCHAR g_fx_media0_media_memory[UX_STORAGE_BUFFER_SIZE];
static uint8_t g_ux_pool_memory[MEMPOOL_SIZE];
static FX_MEDIA * g_p_media = UX_NULL;
TX_EVENT_FLAGS_GROUP g_usb_plug_events;
UINT usb_host_plug_event_notification(ULONG usb_event, UX_HOST_CLASS * host_class,
VOID * instance);
UINT ux_system_host_storage_fx_media_get(UX_HOST_CLASS_STORAGE       * instance,
                                        UX_HOST_CLASS_STORAGE_MEDIA **
p_storage_media,
                                        FX_MEDIA                   ** p_fx_media);
static UINT apl_change_function (ULONG event, UX_HOST_CLASS * host_class, VOID *
instance)
{
    UINT                         status = UX_SUCCESS;
    UX_HOST_CLASS               * class;
    UX_HOST_CLASS_STORAGE       * storage;
    UX_HOST_CLASS_STORAGE_MEDIA * storage_media;
    (void) instance;
 /* Check the class container if it is for a USBX Host Mass Storage class. */
 if (UX_SUCCESS ==
       _ux_utility_memory_compare(_ux_system_host_class_storage_name,
                                  host_class,

_ux_utility_string_length_get(_ux_system_host_class_storage_name)))
    {
 /* Check if there is a device insertion. */
```

```
 if (DEVICE_INSERTION == event)
     {
           status = ux_host_stack_class_get(_ux_system_host_class_storage_name,
&class);
 if (UX_SUCCESS != status)
     {
 return status;
     }
           status = ux_host_stack_class_instance_get(class, 0, (void **) &storage);
 if (UX_SUCCESS != status)
     {
 return status;
     }
 if (UX_HOST_CLASS_INSTANCE_LIVE != storage->ux_host_class_storage_state)
     {
 return UX_ERROR;
     }
           storage_media = class->ux_host_class_media;
           g_p_media    = &storage_media->ux_host_class_storage_media;
           tx_event_flags_set(&g_usb_plug_events, EVENT_USB_PLUG_IN, TX_OR);
     }
 else if (DEVICE_REMOVAL == event) /* Check if there is a device removal. */
     {
           g_p_media = UX_NULL;
           tx_event_flags_set(&g_usb_plug_events, EVENT_USB_PLUG_OUT, TX_OR);
     }
 else
     {
     }
   }
 return status;
}
/*******************************************************************************
 * Function Name : usbx_hmsc_sample
```

```
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 ******************************************************************************/
void usbx_hmsc_sample (void)
{
    ULONG      actual_length = 0;
    ULONG      actual_flags;
    UINT       tx_return;
    UINT       fx_return;
    uint16_t   data_count = 0;
    FX_MEDIA * p_media    = UX_NULL;
    CHAR       volume[VALUE_32];
    fx_system_initialize();
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_host_stack_initialize(apl_change_function);
    tx_event_flags_create(&g_usb_plug_events, (CHAR *) "USB Plug Event Flags");
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 while (1)
    {
 // Wait until device inserted.
       tx_return = tx_event_flags_get(&g_usb_plug_events,
                                        EVENT_USB_PLUG_IN,
                                        TX_OR_CLEAR,
                                        &actual_flags,
                                        TX_WAIT_FOREVER);
 if (TX_SUCCESS != tx_return)
      {
            tx_thread_sleep(TX_WAIT_FOREVER);
      }
 // Get the pointer to FileX Media Control Block for a USB flash device
       p_media = g_p_media;
 // Retrieve the volume name of the opened media from the Data sector
       fx_return = fx_media_volume_get(p_media, volume, FX_DIRECTORY_SECTOR);
```

```
 if (FX_SUCCESS == fx_return)
      {
 // Set the default directory in the opened media, arbitrary name called "firstdir"
            fx_directory_default_set(p_media, "firstdir");
 // Suspend this thread for 200 time-ticks
            tx_thread_sleep(VALUE_100);
 // Try to open the file, 'counter.txt'.
            fx_return = fx_file_open(p_media, &g_file, "counter.txt",
(FX_OPEN_FOR_READ | FX_OPEN_FOR_WRITE));
 if (FX_SUCCESS != fx_return)
      {
 // The 'counter.txt' file is not found, so create a new file
                fx_return = fx_file_create(p_media, "counter.txt");
 if (FX_SUCCESS != fx_return)
      {
 break;
      }
 // Open that file
              fx_return = fx_file_open(p_media, &g_file, "counter.txt",
(FX_OPEN_FOR_READ | FX_OPEN_FOR_WRITE));
 if (FX_SUCCESS != fx_return)
      {
 break;
      }
      }
 // Already open a file, then read the file in blocks
 // Set a specified byte offset for reading
            fx_return = fx_file_seek(&g_file, 0);
 if (FX_SUCCESS == fx_return)
      {
                fx_return = fx_file_read(&g_file, g_read_buf, DATA_LEN,
&actual_length);
 if ((FX_SUCCESS == fx_return) || (FX_END_OF_FILE == fx_return))
      {
```

```
 if (data_count == VALUE_1024)
      {
 // empty file
                        data_count = 0;
      }
 for (uint16_t data_max_count = data_count; data_count < (data_max_count +
VALUE_256); data_count++)
      {
                        g_write_buf[data_count] = data_count;
      }
 // Set the specified byte offset for writing
              fx_return = fx_file_seek(&g_file, 0);
 if (FX_SUCCESS == fx_return)
      {
 // Write the file in blocks
                        fx_return = fx_file_write(&g_file, g_write_buf, DATA_LEN);
 if (FX_SUCCESS == fx_return)
      {
      }
 else
      {
              tx_thread_sleep(TX_WAIT_FOREVER);
      }
      }
      }
      }
 else
      {
              tx_thread_sleep(TX_WAIT_FOREVER);
      }
 // Close already opened file
            fx_return = fx_file_close(&g_file);
 if (FX_SUCCESS != fx_return)
      {
```

```
              tx_thread_sleep(TX_WAIT_FOREVER);
      }
            tx_thread_sleep(VALUE_200);
      }
 else
      {
            tx_thread_sleep(TX_WAIT_FOREVER);
      }
 /* flush the media */
       fx_return = fx_media_flush(p_media);
 if (FX_SUCCESS != fx_return)
      {
            tx_thread_sleep(TX_WAIT_FOREVER);
      }
 /* close the media */
       fx_return = fx_media_close(p_media);
 if (FX_SUCCESS != fx_return)
      {
            tx_thread_sleep(TX_WAIT_FOREVER);
      }
 // Wait for unplugging the USB
       tx_event_flags_get(&g_usb_plug_events, EVENT_USB_PLUG_OUT, TX_OR_CLEAR,
&actual_flags, TX_WAIT_FOREVER);
    } // while(1)
}
void usbx_hmsc_sample_format (void)
{
    ULONG      actual_flags;
    UINT       tx_return;
    UINT       status = UX_SUCCESS;
    FX_MEDIA * p_media = UX_NULL;
    fx_system_initialize();
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_host_stack_initialize(apl_change_function);
```

```
    tx_event_flags_create(&g_usb_plug_events, (CHAR *) "USB Plug Event Flags");
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 // Wait until device inserted.
    tx_return = tx_event_flags_get(&g_usb_plug_events,
                                   EVENT_USB_PLUG_IN,
                                   TX_OR_CLEAR,
                                   &actual_flags,
                                   TX_WAIT_FOREVER);
 if (TX_SUCCESS != tx_return)
    {
       tx_thread_sleep(TX_WAIT_FOREVER);
    }
 // Get the pointer to FileX Media Control Block for a USB flash device
    p_media = g_p_media;
    memset(g_fx_media0_media_memory, 0x00, sizeof(g_fx_media0_media_memory));
    status = fx_media_format(p_media,                                 // Pointer to
FileX media control block.
                             p_media->fx_media_driver_entry,          // Driver entry
                             p_media->fx_media_driver_info,            // Pointer to
Block Media Driver
                             g_fx_media0_media_memory,                // Media buffer
pointer
                             p_media->fx_media_memory_size,            // Media buffer
size
 "sample",                                // Volume Name
                             p_media->fx_media_number_of_FATs,         // Number of
FATs
                             p_media->fx_media_root_directory_entries, // Directory
Entries
                             p_media->fx_media_hidden_sectors,         // Hidden
sectors
                             (ULONG) p_media->fx_media_total_sectors, // Total sectors
                             p_media->fx_media_bytes_per_sector,       // Sector size
                             p_media->fx_media_sectors_per_cluster,    // Sectors per
```

```
cluster
                            p_media->fx_media_heads,                  // Heads (disk
media)
                            p_media->fx_media_sectors_per_track);
 if ((uint8_t) FX_SUCCESS != status)
    {
       __BKPT(0);
    }
}
```

USBX PHID Example

PHID keyboard example is as follows.

```
/******************************************************************************
 * Function Name : usbx_phid_keyboard_sample
 * Description : Application task (loopback processing)
 * Arguments : none
 * Return value : none
 ******************************************************************************/
void usbx_phid_keyboard_sample (void)
{
    UX_SLAVE_CLASS_HID_EVENT hid_event;
    UCHAR    key;
 fsp_err_t err;
    _ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    _ux_device_stack_initialize(NULL,
                               0,
                               g_device_framework_full_speed,
                               VALUE_52,
                               g_string_framework,
                               VALUE_53,
                               g_language_id_framework,
                               VALUE_2,
                               apl_status_change_cb);
```

```
    g_ux_device_class_hid_parameter.ux_slave_class_hid_instance_activate         =
ux_hid_instance_activate;
    g_ux_device_class_hid_parameter.ux_slave_class_hid_instance_deactivate       =
ux_hid_instance_deactivate;
    g_ux_device_class_hid_parameter.ux_device_class_hid_parameter_report_address =
g_apl_report;
    g_ux_device_class_hid_parameter.ux_device_class_hid_parameter_report_length =
REPORT_DESCRIPTOR_LENGTH;
    g_ux_device_class_hid_parameter.ux_device_class_hid_parameter_callback       =
apl_hid_set_report_cb;
    g_ux_device_class_hid_parameter.ux_device_class_hid_parameter_report_id      = 0;
    ux_device_stack_class_register(_ux_system_slave_class_hid_name,
                                  _ux_device_class_hid_entry,
                                  1,
                                  0x00,
                                  (void *) &g_ux_device_class_hid_parameter);
 /* Set the first key to 'a' which is 04. */
    key = 0x04;
 /* reset the HID event structure. */
    ux_utility_memory_set(&hid_event, 0, sizeof(UX_SLAVE_CLASS_HID_EVENT));
    err = g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS == err)
    {
 while (1)
      {
 if (USB_NO == g_suspend)
      {
 while (UX_NULL == g_hid)
      {
 /* Then wait. */
                tx_thread_sleep(10);
      }
 /* 5sec wait */
              usb_cpu_delay_xms((uint16_t) VALUE_5000);
```

```
/* Then insert a key into the keyboard event. Length is fixed to 8. */
               hid_event.ux_device_class_hid_event_length = 8;
/* First byte is a modifier byte. */
               hid_event.ux_device_class_hid_event_buffer[0] = 0;
/* Second byte is reserved. */
               hid_event.ux_device_class_hid_event_buffer[1] = 0;
/* The 6 next bytes are keys. We only have one key here. */
               hid_event.ux_device_class_hid_event_buffer[2] = key;
/* Set the keyboard event. */
               ux_device_class_hid_event_set(g_hid, &hid_event);
/* Next event has the key depressed. */
               hid_event.ux_device_class_hid_event_buffer[2] = 0;
/* Length is fixed to 8. */
               hid_event.ux_device_class_hid_event_length = 8;
/* Set the keyboard event. */
               ux_device_class_hid_event_set(g_hid, &hid_event);
/* Are we at the end of alphabet ? */
if (key != (0x04 + 26))
     {
/* Next key. */
                  key++;
     }
else
     {
/* Start over again. */
     key = 0x04;
     }
               apl_status_change_cb(UX_DEVICE_SUSPENDED);
     }
else
     {
if (USB_NO == g_remote_wakeup)
     {
                  tx_thread_sleep(VALUE_10000);
```

```
 /* Remote wakeup processing */
                  g_remote_wakeup = USB_YES;
                  ux_device_stack_host_wakeup();
                  apl_status_change_cb(UX_DEVICE_RESUMED);
      }
      }
      }
    }
}
```

USBX HHID Example

HHID user interface example is as follows.

```
void keyboard_update_task (ULONG thread_input)
{
    ULONG usbx_return_value;
 /* keyboard button masks, set by ux_host_class_hid_keyboard_buttons_get call */
    ULONG keyboard_key = 0;
 /* keyboard state masks, set by ux_host_class_hid_keyboard_buttons_get call */
    ULONG keyboard_state = 0;
 FSP_PARAMETER_NOT_USED(thread_input);
 while (1)
    {
       usbx_return_value = ux_host_class_hid_keyboard_key_get(
            (UX_HOST_CLASS_HID_KEYBOARD *)
hid_keyboard_client->ux_host_class_hid_client_local_instance,
            &keyboard_key,
            &keyboard_state);
 if ((usbx_return_value == UX_SUCCESS) || (usbx_return_value == UX_NO_KEY_PRESS))
      {
            hid_devices_info.device_connected = KEYBOARD_DEVICE;
            hid_devices_info.key             = keyboard_key;
            hid_devices_info.keyboard_status = keyboard_state;
 /* Clear the states for next read */
```

```
            keyboard_key   = 0;
            keyboard_state = 0;
 /* copy the keyboard states to queue */
            tx_queue_send(&device_parameters, &hid_devices_info, TX_NO_WAIT);
      }
       tx_thread_sleep(10);
    }
}
void mouse_update_task (ULONG thread_input)
{
 /* mouse button masks, set by ux_host_class_hid_mouse_buttons_get call */
    ULONG mouse_buttons;
 /* X co-ordinate displacement of mouse */
    SLONG mouse_x_position = 0;
 /* Y co-ordinate displacement of mouse */
    SLONG mouse_y_position = 0;
 /* variable to hold USBX return values */
    ULONG usbx_return_value;
 FSP_PARAMETER_NOT_USED(thread_input);
 while (1)
    {
       usbx_return_value = ux_host_class_hid_mouse_buttons_get(
            (UX_HOST_CLASS_HID_MOUSE *)
hid_mouse_client->ux_host_class_hid_client_local_instance,
            &mouse_buttons);
 if (usbx_return_value == UX_SUCCESS)
      {
            usbx_return_value = ux_host_class_hid_mouse_position_get(
              (UX_HOST_CLASS_HID_MOUSE *)
hid_mouse_client->ux_host_class_hid_client_local_instance,
                &mouse_x_position,
                &mouse_y_position);
      }
 if (usbx_return_value == UX_SUCCESS)
```

```
      {
            hid_devices_info.device_connected = MOUSE_DEVICE;
            hid_devices_info.key               = mouse_buttons;
            hid_devices_info.mouse_direction_X = mouse_x_position;
            hid_devices_info.mouse_direction_Y = mouse_y_position;
            tx_queue_send(&device_parameters, &hid_devices_info, TX_NO_WAIT);
      }
       tx_thread_sleep(10);
    }
}
void usbx_hhid_sample (void)
{
    uint8_t i;
    ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    ux_host_stack_initialize(ux_system_host_hid_change_function);
    g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
    tx_thread_create(&keyboard_update,
                     (CHAR *) "keyboard_update_task",
                     keyboard_update_task,
                     (ULONG) NULL,
                     &keyboard_update_stack,
                     2048,
                     22,
         2,
                     1,
                     TX_AUTO_START);
    tx_thread_create(&mouse_update,
                     (CHAR *) "mouse_update_task",
                     mouse_update_task,
                     (ULONG) NULL,
                     &mouse_update_stack,
                     1024,
                     22,
         2,
```

```
                     1,
                     TX_AUTO_START);
 while (1)
    {
 for (i = 0; i < UX_HOST_CLASS_HID_MAX_CLIENTS; i++)
      {
 /* Check whether the instance registered? through USB HID device insertion callback?
*/
 if (NULL != hid_class_keyboard_instance[i])
      {
                UX_HOST_CLASS_HID_CLIENT * hid_client =
hid_class_keyboard_instance[i]->ux_host_class_hid_client;
                hid_keyboard_client = hid_client;
         tx_thread_sleep(10);
      }
 /* Check whether the instance registered? through USB HID device insertion callback?
*/
 if (NULL != hid_class_mouse_instance[i])
      {
                UX_HOST_CLASS_HID_CLIENT * hid_client =
hid_class_mouse_instance[i]->ux_host_class_hid_client;
                hid_mouse_client = hid_client;
         tx_thread_sleep(10);
      }
 /* If multiple similar type devices are connected, allow them one by one share data
with application */
         tx_thread_sleep(10);
      }
    }
}
```

USBX PAUD Example

PAUD example is as follows.

```
#define VALUE_275 (275)
#define VALUE_226 (226)
#define VALUE_93 (93)
#define VALUE_2 (2)
#define STACK_SIZE (1024U)
#define NUM_OF_FRAME (8U)
#define USB_MAX_PACKET_SIZE_IN (200U)
#define USB_MAX_PACKET_SZIE_OUT (192U)
#define USB_APL_ON (1U)
#define USB_APL_OFF (0U)
/*** Please enable the following macro when supporting High-speed. ***/
// #define APL_AUDIO_20
/*********************************************************************/
static uint8_t g_read_buf[NUM_OF_FRAME][USB_MAX_PACKET_SZIE_OUT];
static uint8_t g_write_buf[USB_MAX_PACKET_SIZE_IN];
static UX_DEVICE_CLASS_AUDIO * volatile g_p_audio = UX_NULL;
static volatile uint8_t g_read_wp = 0U;
static volatile uint32_t g_read_alternate_setting = USB_APL_OFF;
static volatile uint32_t g_write_alternate_setting = USB_APL_OFF;
static volatile uint8_t g_apl_usb_status = USB_APL_DETACH;
#ifdef APL_AUDIO_20
static UX_DEVICE_CLASS_AUDIO20_CONTROL_GROUP g_audio20_control_group;
static UX_DEVICE_CLASS_AUDIO20_CONTROL g_audio20_control[2];
#else  /* APL_AUDIO_20 */
static UX_DEVICE_CLASS_AUDIO10_CONTROL_GROUP g_audio_control_group;
static UX_DEVICE_CLASS_AUDIO10_CONTROL g_audio_control[2];
#endif  /* APL_AUDIO_20 */
/******************************************************************************
 * Function Name : apl_status_change_cb
 * Description : USB callback function for USB status change
 * Arguments : ULONG status : USB status
 * Return value : UX_SUCCESS
 ******************************************************************************/
static UINT apl_status_change_cb (ULONG status)
```

```
{
 switch (status)
    {
 case UX_DEVICE_ATTACHED:
     {
           g_apl_usb_status = USB_APL_DEFAULT;
 break;
     }
 case UX_DEVICE_CONFIGURED:
     {
           g_apl_usb_status = USB_APL_CONFIGURED;
 break;
     }
 case UX_DEVICE_REMOVED:
     {
           g_apl_usb_status = USB_APL_DETACH;
           g_read_wp                = 0U;
           g_read_alternate_setting = USB_APL_OFF;
           g_write_alternate_setting = USB_APL_OFF;
 break;
     }
 case UX_DEVICE_SUSPENDED:
     {
 if (USB_APL_CONFIGURED == g_apl_usb_status)
     {
               g_apl_usb_status = USB_APL_SUSPEND;
     }
 break;
     }
 case UX_DEVICE_RESUMED:
     {
 if (USB_APL_SUSPEND == g_apl_usb_status)
     {
             g_apl_usb_status = USB_APL_CONFIGURED;
```

```
      }
 break;
      }
 default:
      {
 break;
      }
    }
 return UX_SUCCESS;
}
/******************************************************************************
 * End of function apl_status_change_cb
 ******************************************************************************/
/******************************************************************************
 * Function Name : apl_audio_read_instance_activate
 * Description : Get instance
 * Arguments : void * p_instance : Pointer to the area store the instance pointer
 * Return value : none
 ******************************************************************************/
static void apl_audio_instance_activate (void * p_instance)
{
 /* Save the CDC instance. */
    g_p_audio = (UX_DEVICE_CLASS_AUDIO *) p_instance;
}
/******************************************************************************
 * End of function apl_audio_read_instance_activate
 ******************************************************************************/
/******************************************************************************
 * Function Name : apl_audio_read_instance_deactivate
 * Description : Clear instance
 * Arguments : void * p_instance : Pointer to area store the instance pointer
 * Return value : none
 ******************************************************************************/
static void apl_audio_instance_deactivate (void * p_instance)
```

```
{
 FSP_PARAMETER_NOT_USED(p_instance);
    g_p_audio = UX_NULL;
}
/*******************************************************************************
 * End of function apl_audio_read_instance_deactivate
 *******************************************************************************/
#ifdef APL_AUDIO_20
/*******************************************************************************
 * Function Name : apl_audio20_request_process
 * Description : Audio20 Control Request Processing
 * Arguments : UX_DEVICE_CLASS_AUDIO * : Pointer to Audio instance
 * : UX_SLAVE_TRANSFER * : Pointer to UX_SLAVE_TRANSFER structure
 * Return value : UX_SUCCESS
 *******************************************************************************/
static UINT apl_audio20_request_process (UX_DEVICE_CLASS_AUDIO * p_audio,
UX_SLAVE_TRANSFER * p_transfer)
{
    UINT    ux_err;
    uint8_t i;
    uint8_t number;
    ux_err = ux_device_class_audio20_control_process(p_audio, p_transfer,
&g_audio20_control_group);
 if (UX_SUCCESS == ux_err)
    {
 /* Request handled, check changes */
       number = (uint8_t)
g_audio20_control_group.ux_device_class_audio20_control_group_controls_nb;
       for (i = 0; i < number; i++)
      {
 switch (g_audio20_control[i].ux_device_class_audio20_control_changed)
      {
 case UX_DEVICE_CLASS_AUDIO20_CONTROL_MUTE_CHANGED:
      {
```

```
                   g_control_mute[i] =
g_audio20_control[i].ux_device_class_audio20_control_mute[0];
 break;
      }
 case UX_DEVICE_CLASS_AUDIO20_CONTROL_VOLUME_CHANGED:
      {
                   g_control_volume[i] =
g_audio20_control[i].ux_device_class_audio20_control_volume[0];
 break;
      }
 default:
      {
 break;
      }
      }
      }
    }
 return UX_SUCCESS;
}
/*******************************************************************************
 * End of function apl_audio20_request_process
 *******************************************************************************/
#else  /* APL_AUDIO_20 */
/*******************************************************************************
 * Function Name : apl_audio10_request_process
 * Description : Audio10 Control Request Processing
 * Arguments : UX_DEVICE_CLASS_AUDIO * : Pointer to Audio instance
 * : UX_SLAVE_TRANSFER * : Pointer to UX_SLAVE_TRANSFER structure
 * Return value : UX_SUCCESS
 *******************************************************************************/
static UINT apl_audio10_request_process (UX_DEVICE_CLASS_AUDIO * p_audio,
UX_SLAVE_TRANSFER * p_transfer)
{
    UINT    ux_err;
```

```
    uint8_t i;
    uint8_t number;
    ux_err = ux_device_class_audio10_control_process(p_audio, p_transfer,
&g_audio_control_group);
 if (UX_SUCCESS == ux_err)
    {
 /* Request handled, check changes */
       number = (uint8_t)
g_audio_control_group.ux_device_class_audio10_control_group_controls_nb;
       for (i = 0; i < number; i++)
      {
 switch (g_audio_control[i].ux_device_class_audio10_control_changed)
      {
 case UX_DEVICE_CLASS_AUDIO10_CONTROL_MUTE_CHANGED:
      {
                   g_control_mute[i] =
g_audio_control[i].ux_device_class_audio10_control_mute[0];
 break;
      }
 case UX_DEVICE_CLASS_AUDIO10_CONTROL_VOLUME_CHANGED:
      {
                   g_control_volume[i] =
g_audio_control[i].ux_device_class_audio10_control_volume[0];;
 break;
      }
 default:
      {
 break;
      }
      }
      }
    }
 return UX_SUCCESS;
}
```

```
/******************************************************************************
 * End of function apl_audio10_request_process
 ******************************************************************************/
#endif  /* APL_AUDIO_20 */
/******************************************************************************
 * Function Name : apl_audio_read_change
 * Description : Callback function called when switching alternate setting value of
OUT transfer
 * Arguments : UX_DEVICE_CLASS_AUDIO_STREAM * : Pointer to
UX_DEVICE_CLASS_AUDIO_STREAM structure
 * : ULONG : Alternate Setting Value
 * Return value : UX_SUCCESS
 ******************************************************************************/
static void apl_audio_read_change (UX_DEVICE_CLASS_AUDIO_STREAM * p_stream, ULONG
alternate_setting)
{
    UINT ux_err;
 if (USB_APL_ON == alternate_setting)
    {
       ux_device_class_audio_reception_start(p_stream);
    }
 else
    {
 if (USB_APL_ON == g_read_alternate_setting)
      {
 /* Alternate Setting 1 --> 0 */
            g_read_wp = 0U;
      }
    }
    g_read_alternate_setting = alternate_setting;
}
/******************************************************************************
 * End of function apl_audio_read_change
 ******************************************************************************/
```

```
/*******************************************************************************
 * Function Name : apl_audio_read_done
 * Description : Callback function called when completing of OUT transfer reception
 * Arguments : UX_DEVICE_CLASS_AUDIO_STREAM * : Pointer to
UX_DEVICE_CLASS_AUDIO_STREAM structure
 * : ULONG : Actual Length
 * Return value : UX_SUCCESS
 *******************************************************************************/
static void apl_audio_read_done (UX_DEVICE_CLASS_AUDIO_STREAM * p_stream, ULONG
actual_length)
{
    UINT    ux_err;
    UCHAR * p_buffer;
    ULONG   length;
    UINT    i;
 FSP_PARAMETER_NOT_USED(actual_length);
 if (USB_APL_ON == g_read_alternate_setting)
    {
       ux_err = ux_device_class_audio_read_frame_get(p_stream, &p_buffer, &length);
 if (UX_SUCCESS == ux_err)
      {
 for (i = 0; i < length; i++)
      {
                g_read_buf[g_read_wp][i] = *(p_buffer + i);
      }
            g_read_wp++;
            g_read_wp %= NUM_OF_FRAME;
            ux_device_class_audio_read_frame_free(p_stream);
      }
    }
}
/*******************************************************************************
 * End of function apl_audio_read_done
 *******************************************************************************/
```

```
/******************************************************************************
 * Function Name : apl_audio_write_change
 * Description : Callback function called when switching alternate setting value of
IN transfer
 * Arguments : UX_DEVICE_CLASS_AUDIO_STREAM * : Pointer to
UX_DEVICE_CLASS_AUDIO_STREAM structure
 * : ULONG : Alternate Setting Value
 * Return value : UX_SUCCESS
 ******************************************************************************/
static void apl_audio_write_change (UX_DEVICE_CLASS_AUDIO_STREAM * p_stream, ULONG
alternate_setting)
{
    UINT ux_err;
 if (USB_APL_ON == alternate_setting)
    {
       ux_err = ux_device_class_audio_frame_write(p_stream, g_write_buf,
USB_MAX_PACKET_SIZE_IN);
 if (UX_SUCCESS == ux_err)
      {
            ux_device_class_audio_transmission_start(p_stream);
      }
    }
    g_write_alternate_setting = alternate_setting;
}
/******************************************************************************
 * End of function apl_audio_write_change
 ******************************************************************************/
/******************************************************************************
 * Function Name : apl_audio_write_done
 * Description : Callback function called when completing of IN transfer transmission
 * Arguments : UX_DEVICE_CLASS_AUDIO_STREAM * : Pointer to
UX_DEVICE_CLASS_AUDIO_STREAM structure
 * : ULONG : Actual Length
 * Return value : None
```

```
 ******************************************************************************/
static void apl_audio_write_done (UX_DEVICE_CLASS_AUDIO_STREAM * p_stream, ULONG
actual_length)
{
 FSP_PARAMETER_NOT_USED(actual_length);
 if (USB_APL_ON == g_write_alternate_setting)
    {
       ux_device_class_audio_frame_write(p_stream, g_write_buf,
USB_MAX_PACKET_SIZE_IN);
    }
}
/******************************************************************************
 * End of function apl_audio_write_done
 ******************************************************************************/
/******************************************************************************
 * Function Name : usbx_paudio_apl_init
 * Description : Initialization processing
 * Arguments : None
 * Return value : None
 ******************************************************************************/
void usbx_paudio_apl_init (void)
{
 fsp_err_t err;
    UINT      ux_err;
    uint16_t i = 0;
    UX_DEVICE_CLASS_AUDIO_STREAM_PARAMETER audio_stream_parameter[2];
    UX_DEVICE_CLASS_AUDIO_PARAMETER       audio_parameter;
    _ux_system_initialize((CHAR *) g_ux_pool_memory, MEMPOOL_SIZE, UX_NULL, 0);
    _ux_device_stack_initialize(g_device_framework_hi_speed,
                               VALUE_275,
                               g_device_framework_full_speed,
                               VALUE_226,
                               g_string_framework,
                               VALUE_93,
```

```
                               g_language_id_framework,
                               VALUE_2,
                               apl_status_change_cb);
 /* Read Initialization */
    audio_stream_parameter[0].ux_device_class_audio_stream_parameter_callbacks.ux_dev
ice_class_audio_stream_change =
       apl_audio_read_change;
    audio_stream_parameter[0].ux_device_class_audio_stream_parameter_callbacks.ux_dev
ice_class_audio_stream_frame_done =
       apl_audio_read_done;

audio_stream_parameter[0].ux_device_class_audio_stream_parameter_thread_stack_size
= STACK_SIZE;
    audio_stream_parameter[0].ux_device_class_audio_stream_parameter_max_frame_buffer
_nb   = NUM_OF_FRAME;
    audio_stream_parameter[0].ux_device_class_audio_stream_parameter_max_frame_buffer
_size = USB_MAX_PACKET_SZIE_OUT;
    audio_stream_parameter[0].ux_device_class_audio_stream_parameter_thread_entry
      =
       ux_device_class_audio_read_thread_entry;
 /* Write Initialization */
    audio_stream_parameter[1].ux_device_class_audio_stream_parameter_callbacks.ux_dev
ice_class_audio_stream_change =
       apl_audio_write_change;
    audio_stream_parameter[1].ux_device_class_audio_stream_parameter_callbacks.ux_dev
ice_class_audio_stream_frame_done =
       apl_audio_write_done;

audio_stream_parameter[1].ux_device_class_audio_stream_parameter_thread_stack_size
= STACK_SIZE;
    audio_stream_parameter[1].ux_device_class_audio_stream_parameter_max_frame_buffer
_nb   = NUM_OF_FRAME;
    audio_stream_parameter[1].ux_device_class_audio_stream_parameter_max_frame_buffer
_size = USB_MAX_PACKET_SIZE_IN;
```

```
    audio_stream_parameter[1].ux_device_class_audio_stream_parameter_thread_entry
      =
       ux_device_class_audio_write_thread_entry;
    audio_parameter.ux_device_class_audio_parameter_callbacks.ux_slave_class_audio_in
stance_activate =
       apl_audio_instance_activate;
    audio_parameter.ux_device_class_audio_parameter_callbacks.ux_slave_class_audio_in
stance_deactivate =
       apl_audio_instance_deactivate;
    audio_parameter.ux_device_class_audio_parameter_callbacks.ux_device_class_audio_c
ontrol_process
#ifdef APL_AUDIO_20
       = apl_audio20_request_process;
#else
       = apl_audio10_request_process;
#endif
    audio_parameter.ux_device_class_audio_parameter_streams_nb = 2;
    audio_parameter.ux_device_class_audio_parameter_streams    =
&audio_stream_parameter[0];
    ux_err =
       ux_device_stack_class_register(_ux_system_slave_class_audio_name,
                                        _ux_device_class_audio_entry,
                                     1,
                                     0x00,
                                        (void *) &audio_parameter);
 if (UX_SUCCESS != ux_err)
    {
       USB_APL_AUDIO_ERROR();
    }
#ifdef APL_AUDIO_20
    g_audio20_control[0].ux_device_class_audio20_control_cs_id             = 0x10;
    g_audio20_control[0].ux_device_class_audio20_control_sampling_frequency = 48000;
    g_audio20_control[0].ux_device_class_audio20_control_fu_id             = 5;
    g_audio20_control[0].ux_device_class_audio20_control_mute[0]            = 0;
```

```
    g_audio20_control[0].ux_device_class_audio20_control_volume_min[0]      = 0;
    g_audio20_control[0].ux_device_class_audio20_control_volume_max[0]      = 100;
    g_audio20_control[0].ux_device_class_audio20_control_volume[0]          = 50;
    g_audio20_control[1].ux_device_class_audio20_control_cs_id             = 0x10;
    g_audio20_control[1].ux_device_class_audio20_control_sampling_frequency = 48000;
    g_audio20_control[1].ux_device_class_audio20_control_fu_id             = 8;
    g_audio20_control[1].ux_device_class_audio20_control_mute[0]            = 0;
    g_audio20_control[1].ux_device_class_audio20_control_volume_min[0]      = 0;
    g_audio20_control[1].ux_device_class_audio20_control_volume_max[0]      = 200;
    g_audio20_control[1].ux_device_class_audio20_control_volume[0]          = 100;
    g_audio20_control_group.ux_device_class_audio20_control_group_controls_nb = 2;
    g_audio20_control_group.ux_device_class_audio20_control_group_controls    =
&g_audio20_control[0];
#else  /* APL_AUDIO_20 */
    g_audio_control[0].ux_device_class_audio10_control_fu_id         = 5;
    g_audio_control[0].ux_device_class_audio10_control_mute[0]       = 0;
    g_audio_control[0].ux_device_class_audio10_control_volume[0]    = 0;
    g_audio_control[0].ux_device_class_audio10_control_volume_min[0] = 0;
    g_audio_control[0].ux_device_class_audio10_control_volume_max[0] = 0x80;
    g_audio_control[0].ux_device_class_audio10_control_volume_res[0] = 0x40;
    g_audio_control[1].ux_device_class_audio10_control_fu_id                = 8;
    g_audio_control[1].ux_device_class_audio10_control_mute[0]             = 0x10;
    g_audio_control[1].ux_device_class_audio10_control_volume[0]            = 0x00;
    g_audio_control[1].ux_device_class_audio10_control_volume_min[0]       = 0;
    g_audio_control[1].ux_device_class_audio10_control_volume_max[0]       = 0xF0;
    g_audio_control[1].ux_device_class_audio10_control_volume_res[0]       = 0x80;
    g_audio_control_group.ux_device_class_audio10_control_group_controls_nb = 2;
    g_audio_control_group.ux_device_class_audio10_control_group_controls    =
&g_audio_control[0];
#endif  /* APL_AUDIO_20 */
 for (i = 0; i < USB_MAX_PACKET_SIZE_IN; i++)
    {
       g_write_buf[i] = (UCHAR) (i & 0xFF);
    }
```

```
    err = g_usb_on_usb.open(&g_basic0_ctrl, &g_basic0_cfg);
 if (FSP_SUCCESS != err)
    {
       USB_APL_AUDIO_ERROR();
    }
}
/******************************************************************************
 * End of function usbx_paudio_apl_init
 ******************************************************************************/
/******************************************************************************
 * Function Name : usbx_paudio_apl
 * Description : Application task for USB Audio
 * Arguments : none
 * Return value : none
 ******************************************************************************/
void usbx_paud_sample (void)
{
    usbx_paudio_apl_init();
 while (1)
    {
 switch (g_apl_usb_status)
      {
 case USB_APL_CONFIGURED:
      {
 /* Application Processing */
 break;
      }
 case USB_APL_DETACH:
      {
 break;
      }
 case USB_APL_SUSPEND:
      {
 break;
```

```
    }
default:
    {
break;
    }
    }
  }
}
```

4.2.117 Virtual EEPROM (rm_vee_flash)

Modules

Functions

fsp_err_t	RM_VEE_FLASH_Open (rm_vee_ctrl_t *const p_api_ctrl, rm_vee_cfg_t const *const p_cfg)
fsp_err_t	RM_VEE_FLASH_RecordWrite (rm_vee_ctrl_t *const p_api_ctrl, uint32_t const rec_id, uint8_t const *const p_rec_data, uint32_t const num_bytes)
fsp_err_t	RM_VEE_FLASH_RecordPtrGet (rm_vee_ctrl_t *const p_api_ctrl, uint32_t const rec_id, uint8_t **const pp_rec_data, uint32_t *const p_num_bytes)
fsp_err_t	RM_VEE_FLASH_RefDataWrite (rm_vee_ctrl_t *const p_api_ctrl, uint8_t const *const p_ref_data)
fsp_err_t	RM_VEE_FLASH_RefDataPtrGet (rm_vee_ctrl_t *const p_api_ctrl, uint8_t **const pp_ref_data)
fsp_err_t	RM_VEE_FLASH_StatusGet (rm_vee_ctrl_t *const p_api_ctrl, rm_vee_status_t *const p_status)
fsp_err_t	RM_VEE_FLASH_Refresh (rm_vee_ctrl_t *const p_api_ctrl)
fsp_err_t	RM_VEE_FLASH_Format (rm_vee_ctrl_t *const p_api_ctrl, uint8_t const *const p_ref_data)
fsp_err_t	RM_VEE_FLASH_CallbackSet (rm_vee_ctrl_t *const p_api_ctrl, void(*p_callback)(rm_vee_callback_args_t *), void const *const p_context, rm_vee_callback_args_t *const p_callback_memory)
fsp_err_t	RM_VEE_FLASH_Close (rm_vee_ctrl_t *const p_api_ctrl)

Detailed Description

Virtual EEPROM on RA MCUs. This module implements the Virtual EEPROM Interface.

Overview

This VEE module emulates basic EEPROM capabilities. Support is provided for reading and writing both common records and reference data (originally programmed during product assembly or test). A count of the number of segments erased throughout the lifetime of the application is maintained and can be accessed at any time. Wear leveling is handled automatically by the driver.

Features

- Writing and reading user defined records of any length to data flash.
- Wear leveling is handled automatically.
- Reference data such as calibration data programmed at assembly or test time is preserved.
- Reference data can be updated at run time.
- Fault resilient design.

Data Flash Segmentation

Wear leveling is handled by changing the location in the data flash where a record is stored every time that it is updated. This change in physical location of the record is transparent to the user. Any time an update for a specific record ID is written, it is written to the next unused location in data flash and its location is stored in RAM for quick look-up later. When required, only the most recent version of these records is automatically copied to the next blank segment in data flash. The data flash area is divided into a number of equal-size segments. There is only one segment active at a time. A segment contains two areas- the record area (which is the vast majority of the segment) and the reference data area which contains optional data typically programmed during assembly or final test. Records and updated reference data are written to this segment until one of the two areas becomes full. The record area must be able to hold at least one of every record ID possible and still have space left over for record updates.

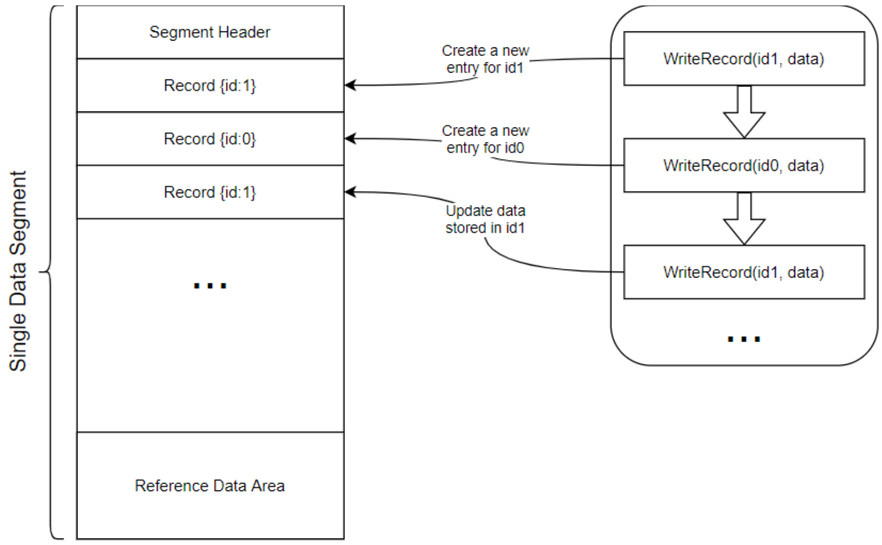


Figure 225: Segment Data Format

When a segment does not have sufficient space for additional records or updated reference data, a Refresh occurs. This process copies the most recent record for each ID as well as the latest version of reference data (if any) to the next segment. The very first time VEE runs on an MCU, it marks the last segment as active whether there is reference data configured or not. The end of VEE data flash area is used to provide an easily identified physical flash address that can be used while programming reference data without requiring Virtual EEPROM middleware.

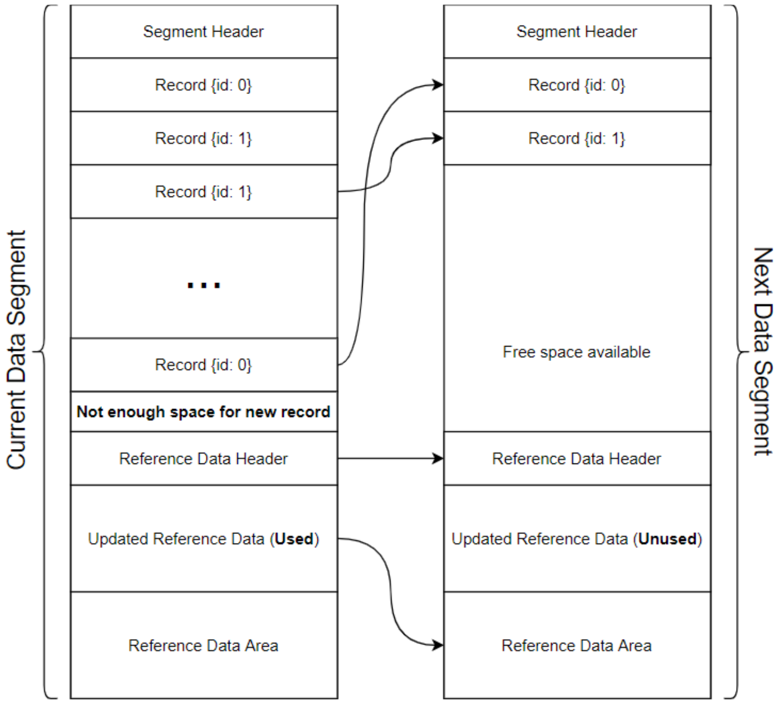


Figure 226: Refresh Operation

Record Format

Each record begins with a header that contains the record size, followed by the data, and the trailer. The trailer contains a validation code which is used for internal purposes only and is not a 16-bit CRC or ECC value. If that level of error checking is desired, the user should include that in the record data passed to the driver. Padding is added between the end of user data and the trailer to ensure the trailer is aligned properly.

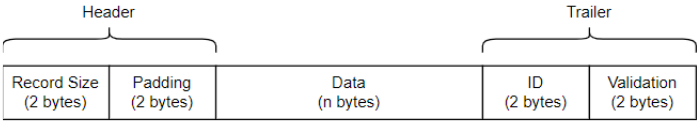


Figure 227: Record Format

Reference Data Area

VEE can be configured for the presence of reference data. The original programmed reference data must be located at the end of the VEE data flash area. An area of equal size is reserved below this in case updated reference data becomes available later. Below that is a header which indicates whether the update area has been written to.

Padding (2 bytes)	Validation (2 bytes)	Update Area (n bytes)	Reference Data (n bytes)

Figure 228: Reference Data Area Format

Just as with records, the validation code is used for internal purposes only and is not a 16-bit CRC or ECC value. If that level of error checking is desired, the user should include that in the updated reference data passed to the driver.

Fault Tolerance

The Virtual EEPROM has a fault tolerant design. If for any reason an operation fails before it is completed the next time the module is opened a refresh will occur. Any corrupted data will be discarded.

Configuration

Build Time Configurations for rm_vee_flash

The following build time configurations are defined in fsp_cfg/rm_vee_flash_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Reference Data Support	• Enabled • Disabled	Disabled	Support writing reference data to the end of the segment.
Refresh Buffer Size	Value must be an integer greater than 0 and a multiple of 4 bytes.	32	The size of the internal buffer used to copying data from one flash segment to another during a refresh operation. This is required because data flash to data flash transfers are not supported by the hardware.

Configurations for Storage > Virtual EEPROM on Flash (rm_vee_flash)

This module can be added to the Stacks tab via New Stack > Storage > Virtual EEPROM on Flash (rm_vee_flash). Non-secure callable guard functions can be generated for this module by right clicking the module in the RA Configuration tool and checking the "Non-secure Callable" box.

Configuration	Options	Default	Description
Name	Name must be a valid	g_vee0	Module name.

	C symbol		
Record Max ID	Value must be an integer.	16	Set this value to the highest record ID in use.
Number of Segments	Value must be an integer.	2	Set value to number of segments desired in data flash (minimum 2). The fewer the segments, the fewer refreshes occur, but the longer refreshes take to complete (erase time).
Start Address	Manual Entry	BSP_FEATURE_FLASH_DATA_FLASH_START	Start address of the flash area used by Virtual EEPROM.
Total Size	Manual Entry	BSP_DATA_FLASH_SIZE_BYTES	The total size (In bytes) of the flash area used by Virtual EEPROM.
Reference Data Size	Value must be an integer.	0	The size of the reference area (In bytes) used by Virtual EEPROM.
Callback	Name must be a valid C symbol	vee_callback	A user callback function can be provided. If this callback function is provided, it will be called from the flash interrupt service routine (ISR).

Clock Configuration

There is no clock configuration for the RM_VEE_FLASH module.

Pin Configuration

This module does not use I/O pins.

Usage Notes

A refresh buffer is required to copy data between segments. Data flash cannot be simultaneously read from and written to. Data will be temporarily copied into RAM during refresh operations.

Examples

Basic Example

This is a basic example of minimal use of the RM_VEE_FLASH module in an application.

```
volatile bool callback_called = false;
/* Record ID to use for storing pressure data. */
#define ID_PRESSURE (0U)
/* Example data structure. */
typedef struct st_pressure
{
    uint32_t timestamp;
    uint16_t low;
    uint16_t average;
    uint16_t high;
} pressure_t;
void rm_vee_example ()
{
 /* Open the Virtual EEPROM Module. */
 fsp_err_t err = RM_VEE_FLASH_Open(&g_vee_ctrl, &g_vee_cfg);
 if (FSP_SUCCESS != err)
    {
       error_handler();
    }
 /* Read pressure data from external sensor. */
    pressure_t pressure_data;
    get_pressure_data(&pressure_data);
 /* Write the pressure data to a Virtual EEPROM Record. */
    err = RM_VEE_FLASH_RecordWrite(&g_vee_ctrl, ID_PRESSURE, (uint8_t *)
&pressure_data, sizeof(pressure_t));
 if (FSP_SUCCESS != err)
    {
       error_handler();
    }
 /* Wait for the Virtual EEPROM callback to indicate it finished writing data. */
 while (false == callback_called)
    {
       ;
    }
```

```
 /* Get a pointer to the record that is stored in data flash. */
    uint32_t    length;
    pressure_t * p_pressure_data;
    err = RM_VEE_FLASH_RecordPtrGet(&g_vee_ctrl, ID_PRESSURE, (uint8_t **)
&p_pressure_data, &length);
 if (FSP_SUCCESS != err)
    {
       error_handler();
    }
 /* Close the Virtual EEPROM Module. */
    err = RM_VEE_FLASH_Close(&g_vee_ctrl);
 if (FSP_SUCCESS != err)
    {
       error_handler();
    }
}
void rm_vee_tests_callback (rm_vee_callback_args_t * p_args)
{
    callback_called = true;
 FSP_PARAMETER_NOT_USED(p_args);
}
```

Data Structures

struct	rm_vee_flash_cfg_t
struct	rm_vee_flash_instance_ctrl_t

Data Structure Documentation

◆ rm_vee_flash_cfg_t

struct rm_vee_flash_cfg_t		
User configuration structure, used in open function		
Data Fields		
flash_instance_t const *	p_flash	Pointer to a flash instance.

◆ rm_vee_flash_instance_ctrl_t

struct rm_vee_flash_instance_ctrl_t

Instance control block. This is private to the FSP and should not be used or modified by the application.

Function Documentation

◆ RM_VEE_FLASH_Open()

fsp_err_t RM_VEE_FLASH_Open (rm_vee_ctrl_t *const *p_api_ctrl*, rm_vee_cfg_t const *const *p_cfg*)

Open the RM_VEE_FLASH driver module

Implements rm_vee_api_t::open

Initializes the driver's internal structures and opens the Flash driver. The Flash driver must be closed prior to opening VEE. The error code FSP_SUCCESS_RECOVERY indicates that VEE detected corrupted data; most likely due to a power loss during a data flash write or erase. In these cases, an automatic internal Refresh is performed and the partially written data is lost.

Return values

FSP_SUCCESS	Successful. FSP_SUCCESS_RECOVERY changed to FSP_SUCCESS
FSP_ERR_ASSERTION	An input parameter is NULL.
FSP_ERR_ALREADY_OPEN	This function has already been called.
FSP_ERR_PE_FAILURE	This error indicates that a flash programming, erase, or blankcheck operation has failed in hardware.
FSP_ERR_TIMEOUT	Interrupts disabled outside of VEE
FSP_ERR_NOT_INITIALIZED	Corruption found. A refresh is required.
FSP_ERR_INVALID_ARGUMENT	The supplied configuration is invalid.

◆ RM_VEE_FLASH_RecordWrite()

fsp_err_t RM_VEE_FLASH_RecordWrite (rm_vee_ctrl_t *const *p_api_ctrl*, uint32_t const *rec_id*, uint8_t const *const *p_rec_data*, uint32_t const *num_bytes*)

Writes a record to data flash.

Implements rm_vee_api_t::recordWrite

This function writes num_bytes of data pointed to by p_rec_data to data flash. This function returns immediately after starting the flash write. BE SURE NOT TO MODIFY the data buffer contents until after the write completes. This includes exiting the calling function when the data buffer is a local variable (stack may be used by another function and corrupt the data buffer contents).

Return values

FSP_SUCCESS	Write started successfully.
FSP_ERR_NOT_OPEN	The module has not been opened.
FSP_ERR_ASSERTION	An input parameter is NULL.
FSP_ERR_INVALID_ARGUMENT	An argument contains an illegal value.
FSP_ERR_INVALID_MODE	The operation cannot be started in the current mode.
FSP_ERR_IN_USE	Last API call still executing.
FSP_ERR_PE_FAILURE	This error indicates that a flash programming, erase, or blankcheck operation has failed in hardware.
FSP_ERR_TIMEOUT	Flash write timed out (Should not be possible when flash bgo is used).
FSP_ERR_NOT_INITIALIZED	Corruption found. A refresh is required.

◆ RM_VEE_FLASH_RecordPtrGet()

fsp_err_t RM_VEE_FLASH_RecordPtrGet (rm_vee_ctrl_t *const *p_api_ctrl*, uint32_t const *rec_id*, uint8_t **const *pp_rec_data*, uint32_t *const *p_num_bytes*)

Gets a pointer to the most recent record data.

Implements rm_vee_api_t::recordPtrGet

This function sets the argument pointer to the most recent version of the record data in flash. Flash cannot be accessed for reading and writing at the same time. Therefore, reading the data at p_ref_data must be completed prior to initiating any type of Flash write.

Return values

FSP_SUCCESS	Successful.
FSP_ERR_NOT_OPEN	The module has not been opened.
FSP_ERR_IN_USE	Last API call still executing.
FSP_ERR_ASSERTION	p_ref_data is NULL.
FSP_ERR_INVALID_ARGUMENT	Record data not configured.
FSP_ERR_NOT_FOUND	The record associated with the requested ID could not be found.

◆ RM_VEE_FLASH_RefDataWrite()

fsp_err_t RM_VEE_FLASH_RefDataWrite (rm_vee_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_ref_data*)

Writes new Reference data to the reference update area.

Implements rm_vee_api_t::refDataWrite

This function writes VEE_CFG_REF_DATA_SIZE bytes pointed to by p_ref_data to data flash. This function returns immediately after starting the flash write. BE SURE NOT TO MODIFY the data buffer contents until after the write completes.

Return values

FSP_SUCCESS	Write started successfully.
FSP_ERR_NOT_OPEN	The module has not been opened.
FSP_ERR_IN_USE	Last API call still executing.
FSP_ERR_ASSERTION	An input parameter is NULL.
FSP_ERR_INVALID_MODE	The operation cannot be started in the current mode.
FSP_ERR_PE_FAILURE	This error indicates that a flash programming, erase, or blankcheck operation has failed in hardware.
FSP_ERR_TIMEOUT	Flash write timed out (Should not be possible when flash bgo is used).
FSP_ERR_UNSUPPORTED	Reference data is not supported in the current configuration.
FSP_ERR_NOT_INITIALIZED	Corruption found. A refresh is required.

◆ RM_VEE_FLASH_RefDataPtrGet()

fsp_err_t RM_VEE_FLASH_RefDataPtrGet (rm_vee_ctrl_t *const *p_api_ctrl*, uint8_t **const *pp_ref_data*)

Gets a pointer to the most recent reference data.

Implements rm_vee_api_t::recordPtrGet

This function sets the argument pointer to the most recent version of the reference data in flash. Flash cannot be accessed for reading and writing at the same time.

Return values

FSP_SUCCESS	Successful.
FSP_ERR_NOT_OPEN	The module has not been opened.
FSP_ERR_IN_USE	Last API call still executing.
FSP_ERR_ASSERTION	An input parameter is NULL.
FSP_ERR_UNSUPPORTED	Reference data is not supported in the current configuration.
FSP_ERR_NOT_FOUND	No reference data was found.

◆ RM_VEE_FLASH_StatusGet()

fsp_err_t RM_VEE_FLASH_StatusGet (rm_vee_ctrl_t *const *p_api_ctrl*, rm_vee_status_t *const *p_status*)

Get the current status of the driver.

Implements rm_vee_api_t::statusGet

This command is typically used to verify that the last Write or Refresh command has completed before attempting to perform another API call.

Return values

FSP_SUCCESS	Successful.
FSP_ERR_NOT_OPEN	The module has not been opened.
FSP_ERR_ASSERTION	An input parameter is NULL.

◆ RM_VEE_FLASH_Refresh()

fsp_err_t RM_VEE_FLASH_Refresh (rm_vee_ctrl_t *const *p_api_ctrl*)

Manually start a refresh operation

Implements rm_vee_api_t::refresh

This function is used to start a segment Refresh at any time. The Refresh process by default occurs automatically when no more record or reference data space is available and a Write is requested. However, the app may desire to force a refresh when it knows it is running low on space and large amounts of data are about to be recorded.

Return values

FSP_SUCCESS	Successful.
FSP_ERR_NOT_OPEN	The module has not been opened.
FSP_ERR_ASSERTION	An input parameter is NULL.
FSP_ERR_IN_USE	Last API call still executing.
FSP_ERR_PE_FAILURE	This error indicates that a flash programming, erase, or blankcheck operation has failed in hardware.
FSP_ERR_INVALID_MODE	The operation cannot be started in the current mode.
FSP_ERR_TIMEOUT	Flash write timed out (Should not be possible when flash bgo is used).
FSP_ERR_NOT_INITIALIZED	Corruption found. A refresh is required.

◆ RM_VEE_FLASH_Format()

fsp_err_t RM_VEE_FLASH_Format (rm_vee_ctrl_t *const *p_api_ctrl*, uint8_t const *const *p_ref_data*)

Start a manual format operation.

Implements rm_vee_api_t::format

Return values

FSP_SUCCESS	Successful.
FSP_ERR_NOT_OPEN	The module has not been opened.
FSP_ERR_IN_USE	Last API call still executing.
FSP_ERR_PE_FAILURE	This error indicates that a flash programming, erase, or blankcheck operation has failed
FSP_ERR_ASSERTION	An input parameter is NULL.
FSP_ERR_TIMEOUT	Flash write timed out (Should not be possible when flash bgo is used).
FSP_ERR_NOT_INITIALIZED	Corruption found. A refresh is required.

◆ RM_VEE_FLASH_CallbackSet()

fsp_err_t RM_VEE_FLASH_CallbackSet (rm_vee_ctrl_t *const *p_api_ctrl*, void(*)(rm_vee_callback_args_t *) *p_callback*, void const *const *p_context*, rm_vee_callback_args_t *const *p_callback_memory*)

Updates the user callback with the option to provide memory for the callback argument structure.

Implements rm_vee_api_t::callbackSet.

Return values

FSP_SUCCESS	Callback updated successfully.
FSP_ERR_ASSERTION	A required pointer is NULL.
FSP_ERR_NOT_OPEN	The control block has not been opened.
FSP_ERR_NO_CALLBACK_MEMORY	p_callback is non-secure and p_callback_memory is either secure or NULL.

◆ RM_VEE_FLASH_Close()

fsp_err_t RM_VEE_FLASH_Close (rm_vee_ctrl_t *const *p_api_ctrl*)

Closes the Flash driver and VEE driver.

Implements rm_vee_api_t::close

Return values

FSP_SUCCESS	Successful.
FSP_ERR_NOT_OPEN	The module has not been opened.
FSP_ERR_ASSERTION	An input parameter is NULL.

4.2.118 AWS Device Provisioning

Modules

AWS Device Provisioning example software.

Overview

Terminology

The terminology defined below will be used in the following sections.

Term	Description
Service Provider	Entity that provides the cloud infrastructure and associated services, for example, AWS/Azure.
Device Manufacturer	Entity that provides the MCU, for example, Renesas.
OEM	Entity that uses the MCU to create a product.
Customer	End user of OEM product.

Device ID

For systems that intend to use Public Key Certificate (PKC), the Device ID is in the form of a key pair (RSA or ECC). A PKC comprises of a **public key**, metadata, and finally a signature over all that. This signature is generated by the entity that issues the certificate and is known as a CA (Certificate Authority). The most common format for a public certificate is the X.509 format which is typically PEM (base 64) encoded such that the certificate is human-readable. It can also be DER encoded which is binary encoding and thus not human readable. The **public key** portion of the Device ID is used for the Device Certificate.

Provisioning

Device Provisioning refers to the process by which a service provider links a certificate to a Device ID and thus a device. Depending on the provisioning model, an existing certificate from the device may be used or a new one will be issued at this stage. Provisioning (also referred to as Registration) occurs with respect to a particular service provider, for example, AWS or Azure. It is necessary that the certificate is issued by the service provider or a CA known to those providers. When a device is provisioned with AWS for example, the AWS IoT service associates the Device ID (and thus the device) with a specific certificate. The certificate will be programmed into the device and for all future transactions with AWS, the certificate will be used as the means of identifying the device. The public and private key are also stored on the MCU.

Provisioning Models

Provisioning services vary between service providers. There are essentially three general provisioning models.

1. Provisioning happens on the production line. This requires the provisioning Infrastructure to be present on the production line. This is the most secure model, but is expensive.
2. Devices are programmed with a shared credential that is linked into the code at build time and the provisioning occurs when a customer uses the device for the first time. The shared credential and a unique device serial number are used to uniquely identify the device during the provisioning process. So long as the product only has the shared credential, it will only operate with limited (as defined by certificate policy) functionality .Once the provisioning is done, then the device will be fully functional. This is the most common use case for consumer products where no sensitive information is being transmitted. AWS provides an example of this model.
3. Devices have no identity programmed in the factory; provisioning occurs through some other device like a smartphone which is already trusted by the service provider.

In all these cases, the Device Identity

1. Is unique to the device
2. Must have restricted access within the device
3. Can be used to issue more than one certificate and the certificates themselves have to be updatable in the field.

AWS uses the PKCS11 API to erase, store and retrieve certificates. These PKCS11 functions (Write, Read and Erase) are separated out into a Physical Abstraction Layer (PAL) which the OEM/Device Manufacturer is expected to implement for the type of memory that they intend to use. The internal rm_aws_pkcs11_pal module implements these requirements on RA MCU data flash.

AWS Provisioning Example

AWS provides an **example** implementation to support device provisioning. This implementation uses the PKCS11 API to store device credentials into the PKCS11 defined memory. The implementation (aws_dev_mode_key_provisioning.c) exposes two functions:

1. vDevModeKeyProvisioning()
2. vAlternateKeyProvisioning()

Both of these functions require that the device credentials be provided in PEM format. Using either of these example functions as is in production is not recommended; but vAlternateKeyProvisioning() provides more flexibility because of the ability to provide credentials as arguments.

Credentials can be created as follows:

- Create your own CA and use that to generate the device certificate. This CA will have to be registered with the service provider with which the product will be used, for example Register your CA with AWS.
- Use AWS to generate the device certificate.

Examples

Basic Example

This is a basic example of provisioning a device using the AWS demo implementation.

```
#define keyCLIENT_CERTIFICATE_PEM \
  "-----BEGIN CERTIFICATE-----\n" \
  "MIIDETCCAfkCFHwd2yn8zn5qB2ChYUT9Mvbi9Xp1MA0GCSqGSIb3DQEBCwUAMEUx\n" \
  "CzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRl\n" \
  "cm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcNMTkwOTExMjEyMjU0WhcNMjAwOTEwMjEy\n" \
  "MjU0WjBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UE\n" \
  "CgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOC\n" \
  "AQ8AMIIBCgKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABMA0GCSqGSIb3DQEBCwUA\n" \
  "A4IBAQCdqq59ubdRY9EiV3bleKXeqG7+8HgBHdm0X9dgq10nD37p00YLyuZLE9NM\n" \
  "066G/VcflGrx/Nzw+/UuI7/UuBbBS/3ppHRnsZqBIl8nnr/ULrFQy8z3vKtL1q3C\n" \
  "DxabjPONlPO2keJeTTA71N/RCEMwJoa8i0XKXGdu/hQo6x4n+Gq73fEiGCl99xsc\n" \
  "4tIO4yPS4lv+uXBzEUzoEy0CLIkiDesnT5lLeCyPmUNoU89HU95IusZT7kygCHHd\n" \
  "72am1ic3X8PKc268KT3ilr3VMhK67C+iIIkfrM5AiU+oOIRrIHSC/p0RigJg3rXA\n" \
  "GBIRHvt+OYF9fDeG7U4QDJNCfGW+\n" \
  "-----END CERTIFICATE-----"
#define keyCLIENT_PRIVATE_KEY_PEM \
  "-----BEGIN RSA PRIVATE KEY-----\n" \
  "MIIEowIBAAKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
```

```
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABAoIBAQCGR2hC/ZVJhqIM\n" \
  "c2uuJZKpElpIIBBPOObZwwS3IYR4UUjzVgMn7UbbmxflLXD8lzfZU4YVp0vTH5lC\n" \
  "07qvYuXpHqtnj+GEok837VYCtUY9AuHeDM/2paV3awNV15E1PFG1Jd3pqnH7tJw6\n" \
  "VBZBDiGNNt1agN/UnoSlMfvpU0r8VGPXCBNxe3JY5QyBJPI1wF4LcxRI+eYmr7Ja\n" \
  "/cjn97DZotgz4B7gUNu8XIEkUOTwPabZINY1zcLWiXTMA+8qTniPVk653h14Xqt4\n" \
  "4o4D4YCTpwJcmxSV1m21/6+uyuXr9SIKAE+Ys2cYLA46x+rwLaW5fUoQ5hHa0Ytb\n" \
  "RYJ4SrtBAoGBANWtwlE69N0hq5xDPckSbNGubIeG8P4mBhGkJxIqYoqugGLMDiGX\n" \
  "4bltrjr2TPWaxTo3pPavLJiBMIsENA5KU+c/r0jLkxgEp9MIVJrtNgkCiDQqogBG\n" \
  "j4IJL2iQwXoLCqk2tx/dh9Mww+7SETE7EPNrv4UrYaGN5AEvpf5W+NHPAoGBAMQ6\n" \
  "wVa0Mx1PlA4enY2rfE3WXP8bzjleSOwR75JXqG2WbPC0/cszwbyPWOEqRpBZfvD/\n" \
  "QFkKx06xp1C09XwiQanr2gDucYXHeEKg/9iuJV1UkMQp95ojlhtSXdRZV7/l4pmN\n" \
  "fpB2vcAptX/4gY4tDrWMO08JNnRjE7duC+rmmk1hAoGAS4L0QLCNB/h2JOq+Uuhn\n" \
  "/FGfmOVfFPFrA6D3DbxcxpWUWVWzSLvb0SOphryzxbfEKyau7V5KbDp7ZSU/IC20\n" \
  "KOygjSEkAkDi7fjrrTRW/Cgg6g6G4YIOBO4qCtHdDbwJMHNdk6096qw5EZS67qLp\n" \
  "Apz5OZ5zChySjri/+HnTxJECgYBysGSP6IJ3fytplTtAshnU5JU2BWpi3ViBoXoE\n" \
  "bndilajWhvJO8dEqBB5OfAcCF0y6TnWtlT8oH21LHnjcNKlsRw0Dvllbd1oylybx\n" \
  "3da41dRG0sCEtoflMB7nHdDLt/DZDnoKtVvyFG6gfP47utn+Ahgn+Zp6K+46J3eP\n" \
  "s3g8AQKBgE/PJiaF8pbBXaZOuwRRA9GOMSbDIF6+jBYTYp4L9wk4+LZArKtyI+4k\n" \
  "Md2DUvHwMC+ddOtKqjYnLm+V5cSbvu7aPvBZtwxghzTUDcf7EvnA3V/bQBh3R0z7\n" \
  "pVsxTyGRmBSeLdbUWACUbX9LXdpudarPAJ59daWmP3mBEVmWdzUw\n" \
  "-----END RSA PRIVATE KEY-----"
void device_provisioning_example (void)
{
 /* Initialize IOT FreeRTOS Libraries */
    SYSTEM_Init();
 /* Initialize the crypto hardware acceleration. */
 mbedtls_platform_setup(NULL);
    ProvisioningParams_t params;
 /* Provision device with provided credentials. The provided credentials are written
to data flash.
  * In production, the credentials can be provided over a comms channel instead of
being linked into the image.
  * The same example provisioning function, vAlternateKeyProvisioning, can be used in
```

```
that case. */
    params.pucClientPrivateKey       = (uint8_t *) keyCLIENT_PRIVATE_KEY_PEM;
    params.pucClientCertificate      = (uint8_t *) keyCLIENT_CERTIFICATE_PEM;
    params.ulClientPrivateKeyLength = 1 + strlen((const char *)
params.pucClientPrivateKey);
    params.ulClientCertificateLength = 1 + strlen((const char *)
params.pucClientCertificate);
    params.pucJITPCertificate       = NULL;
    params.ulJITPCertificateLength   = 0;
    vAlternateKeyProvisioning(&params);
}
```

Limitations

The provisioning code is an example provided by AWS. It must be modified to meet product requirements.

4.2.119 AWS HTTPS

Modules

This module provides the AWS HTTPS integration documentation.

Overview

The AWS HTTPS core library can be used to send HTTP and HTTPS requests. The documentation for the library can be found at the following link: coreHTTP.

Features

- Secure and Non-secure HTTP requests
- Mutually authenticated connections

Configuration

Memory Usage

The AWS HTTPS library relies heavily on dynamic memory allocation for thread/task creation as well as other uses. It is recommended to tweak the thread stack configuration values based on usage. Noteable values are:

AWS IoT Common

- IoT Thread Default Stack Size
- IoT Network Receive Task Stack Size

FreeRTOS Thread

- General|Minimal Stack Size

FreeRTOS Plus TCP

- Stack size in words (not bytes)

Usage Notes

Limitations

- MbedTLS must be initialized and key provisioning must be done before starting a secure connection. Refer to AWS Secure Sockets.

Examples

HTTPS GET request

```
/* Certificate copied from https://www.amazontrust.com/repository/AmazonRootCA1.pem
*/
static const char g_server_certificate[] = "-----BEGIN CERTIFICATE-----\n" \
 "MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF\n" \
 "ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6\n" \
 "b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL\n" \
 "MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv\n" \
 "b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj\n" \
 "ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM\n" \
 "9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw\n" \
 "IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6\n" \
 "VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L\n" \
 "93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm\n" \
 "jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC\n" \
 "AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA\n" \
 "A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI\n" \
 "U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs\n" \
 "N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv\n" \
 "o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU\n" \
```

```
 "5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy\n" \
 "rqXRfboQnoZsG4q5WTP468SQvvG5\n"                                     \
 "-----END CERTIFICATE-----";
/* Default settings to use if DHCP fails. */
const uint8_t g_default_ip_address[4] = {192, 168, 0, 100};
const uint8_t g_default_subnet_mask[4] = {255, 255, 255, 0};
const uint8_t g_default_gateway[4] = {192, 168, 0, 1};
const uint8_t g_default_dns[4] = {8, 8, 8, 8};
void https_example_entry (void * pvParameters)
{
 FSP_PARAMETER_NOT_USED(pvParameters);
 /* Initialize IOT FreeRTOS Libraries */
    SYSTEM_Init();
 /* Initialize the crypto hardware acceleration. */
 mbedtls_platform_setup(NULL);
 /* Initialize the logging task. */
    assert(pdPASS == xLoggingTaskInitialize(256, 1, 10));
 /* In order to use the PKCS11 PAL, littlefs must be configured. */
 fsp_err_t fsp_err_status = RM_LITTLEFS_FLASH_Open(g_rm_littlefs0.p_ctrl,
g_rm_littlefs0.p_cfg);
    assert(FSP_SUCCESS == fsp_err_status);
 /* Reformat littlefs to ensure that data flash is in a known state. */
    assert(0 == lfs_format(&g_rm_littlefs0_lfs, &g_rm_littlefs0_lfs_cfg));
 /* Mount littlefs. */
    assert(0 == lfs_mount(&g_rm_littlefs0_lfs, &g_rm_littlefs0_lfs_cfg));
 /*
  * Write the keys into data flash using the PKCS11 PAL so that they can be used
during TLS setup
  * Note that in an application this will only be done when provisioning a device
with a private key.
  * Once a device has been provisioned, the keys will persist in data flash.
  */
    ProvisioningParams_t params;
    params.pucClientPrivateKey       = (uint8_t *) g_client_private_key;
```

```
    params.pucClientCertificate      = (uint8_t *) g_client_certificate;
    params.ulClientPrivateKeyLength = sizeof(g_client_private_key);
    params.ulClientCertificateLength = sizeof(g_client_certificate);
    params.pucJITPCertificate       = NULL;
    params.ulJITPCertificateLength   = 0;
    uint32_t err = (uint32_t) vAlternateKeyProvisioning(&params);
    assert(0 == err);
/* Start up the network stack. */
    FreeRTOS_IPInit(g_default_ip_address,
                   g_default_subnet_mask,
                   g_default_gateway,
                   g_default_dns,
                   g_ether0.p_cfg->p_mac_address);
while (pdFALSE == FreeRTOS_IsNetworkUp())
    {
        vTaskDelay(10);
    }
    ServerInfo_t            xServerInfo    = {0};
    SocketsConfig_t         xSocketsConfig = {0};
    TransportSocketStatus_t xNetworkStatus = TRANSPORT_SOCKET_STATUS_SUCCESS;
/* Initializer server information. */
    xServerInfo.pHostName      = "postman-echo.com";
    xServerInfo.hostNameLength = strlen(xServerInfo.pHostName);
    xServerInfo.port           = HTTPS_EXAMPLE_TLS_PORT;
/* Configure credentials for TLS authenticated session. */
    xSocketsConfig.enableTls         = true;
    xSocketsConfig.pAlpnProtos       = NULL;
    xSocketsConfig.maxFragmentLength = 0;
    xSocketsConfig.disableSni       = false;
    xSocketsConfig.pRootCa          = g_server_certificate;
    xSocketsConfig.rootCaSize       = sizeof(g_server_certificate);
    xSocketsConfig.sendTimeoutMs    = HTTPS_EXAMPLE_TIMEOUT;
    xSocketsConfig.recvTimeoutMs    = HTTPS_EXAMPLE_TIMEOUT;
    NetworkContext_t xNetworkContext = {0};
```

```
 /* Attempt to create a authenticated TLS connection. */
    xNetworkStatus = SecureSocketsTransport_Connect(&xNetworkContext, &xServerInfo,
&xSocketsConfig);
    assert(TRANSPORT_SOCKET_STATUS_SUCCESS == xNetworkStatus);
    TransportInterface_t xTransportInterface;
 /* Define the transport interface. */
    xTransportInterface.pNetworkContext = &xNetworkContext;
    xTransportInterface.send            = SecureSocketsTransport_Send;
    xTransportInterface.recv            = SecureSocketsTransport_Recv;
    HTTPRequestInfo_t    xRequestInfo    = {0};
    HTTPRequestHeaders_t xRequestHeaders = {0};
 /* Configure a GET request. */
    xRequestInfo.pHost    = "postman-echo.com";
    xRequestInfo.hostLen   = strlen(xRequestInfo.pHost);
    xRequestInfo.pMethod   = HTTP_METHOD_GET;
    xRequestInfo.methodLen = strlen(HTTP_METHOD_GET);
    xRequestInfo.pPath    = "/get?arg1=val1&arg2=val2";
    xRequestInfo.pathLen   = strlen(xRequestInfo.pPath);
    xRequestInfo.reqFlags = HTTP_REQUEST_KEEP_ALIVE_FLAG;
 /* Set the buffer used for storing request headers. */
 static uint8_t ucUserBuffer[HTTPS_EXAMPLE_USER_BUFFER_SIZE];
    xRequestHeaders.pBuffer   = ucUserBuffer;
    xRequestHeaders.bufferLen = sizeof(ucUserBuffer);
 /* Initialize the request. */
    HTTPStatus_t xHTTPStatus = HTTPClient_InitializeRequestHeaders(&xRequestHeaders,
&xRequestInfo);
    assert(HTTPSuccess == xHTTPStatus);
 /* Reuse the user buffer for storing the response headers. */
    HTTPResponse_t xResponse = {0};
    xResponse.pBuffer   = ucUserBuffer;
    xResponse.bufferLen = sizeof(ucUserBuffer);
 /* Send the request. */
    xHTTPStatus = HTTPClient_Send(&xTransportInterface, &xRequestHeaders, (uint8_t *)
NULL, 0, &xResponse, 0);
```

```
    assert(HTTPSuccess == xHTTPStatus);
    xNetworkStatus = SecureSocketsTransport_Disconnect(&xNetworkContext);
    assert(TRANSPORT_SOCKET_STATUS_SUCCESS == xNetworkStatus);
 /* The HTTPS request has completed. The result is stored in xResponse. */
}
```

4.2.120 AWS MQTT

Modules

This module provides the AWS MQTT integration documentation.

Overview

The AWS MQTT library can connect to either AWS or a third party MQTT broker such as Mosquitto. The documentation for the library can be found at the following link: coreMQTT.

Features

- MQTT connections over TLS to an AWS IoT Endpoint or Mosquitto server
- Unsecure MQTT connections to Mosquitto servers. This is not recommended for production and should only be done to a local server for testiing.

Configuration

Memory Usage

The AWS MQTT library relies heavily on dynamic memory allocation for thread/task creation as well as other uses. Depending on the configuration it may be required to provde as much as 110k heap. To decrease this it is recommended to tweak the thread stack configuration values based on usage. Noteable values are:

AWS IoT Common

- IoT Thread Default Stack Size
- IoT Network Receive Task Stack Size

FreeRTOS Thread

- General|Minimal Stack Size

FreeRTOS Plus TCP

- Stack size in words (not bytes)

Usage Notes

The AWS MQTT library utilizes a system taskpool to queue up messages. This system task pool must be created before calling into the MQTT library. iot_init.c has been provided for easy initialization of this taskpool via IotSdk_Init().

The AWS MQTT Demo has been provided to easily demonstrate MQTT functionality. An example of initializiing the system taskpool and running the MQTT demo has been provided below.

Limitations

- aws_clientcredential.h and aws_clientcredential_keys.h need to be added manually.
- The IoT Thread must have a higher priorty than the Network Receive Thread.
- MbedTLS must be initialized and key provisioning must be done before starting a secure connection. Refer to AWS Secure Sockets.

Examples

Non-secure connection to a Mosquitto server

```
#define IOT_LOG_STACK_SIZE (256)
const uint8_t g_ip_address[4] = {169, 254, 57, 49};
const uint8_t g_net_mask[4] = {255, 255, 0, 0};
const uint8_t g_gateway_address[4] = {169, 254, 57, 49};
const uint8_t g_dns_address[4] = {8, 8, 8, 8};
const uint8_t g_mac_address[6] = {0x66, 0x66, 0x66, 0x66, 0x66, 0x66};
void mqtt_non_secure_example ()
{
 /* Initialize IOT FreeRTOS Libraries */
    SYSTEM_Init();
 /* Initialize the crypto hardware acceleration. */
 mbedtls_platform_setup(NULL);
    xLoggingTaskInitialize(IOT_LOG_STACK_SIZE, 1, 10);
 /* Start up the network stack. */
    FreeRTOS_IPInit(g_ip_address, g_net_mask, g_gateway_address, g_dns_address,
g_mac_address);
 while (pdFALSE == FreeRTOS_IsNetworkUp())
    {
       vTaskDelay(1);
    }
 /* AWS MQTT APIs can now be called. Please refer to the documentation linked above.
*/
```

```
 /* If using the MQTT Wrapper IotSdk_Init() must be called prior to using MQTT. */
}
```

Secure connection to a Mosquitto server

Note

MbedTLS must be initialized and key provisioning must be done before starting a secure connection. Refer to AWS Secure Sockets.

```
#define keyCLIENT_CERTIFICATE_PEM \
  "-----BEGIN CERTIFICATE-----\n" \
  "example_certificate_formatting\n" \
  "-----END CERTIFICATE-----"
#define keyCLIENT_PRIVATE_KEY_PEM \
  "-----BEGIN RSA PRIVATE KEY-----\n" \
  "example_certificate_formatting\n" \
  "-----END RSA PRIVATE KEY-----"
void mqtt_secure_example ()
{
 /* Initialize IOT FreeRTOS Libraries */
    SYSTEM_Init();
 /* Initialize the crypto hardware acceleration. */
 mbedtls_platform_setup(NULL);
    xLoggingTaskInitialize(IOT_LOG_STACK_SIZE, 1, 10);
    ProvisioningParams_t params;
 /* Write the keys into a secure region in data flash. */
    params.pucClientPrivateKey       = (uint8_t *) keyCLIENT_PRIVATE_KEY_PEM;
    params.pucClientCertificate      = (uint8_t *) keyCLIENT_CERTIFICATE_PEM;
    params.ulClientPrivateKeyLength = 1 + strlen((const char *)
params.pucClientPrivateKey);
    params.ulClientCertificateLength = 1 + strlen((const char *)
params.pucClientCertificate);
    params.pucJITPCertificate       = NULL;
    params.ulJITPCertificateLength   = 0;
    vAlternateKeyProvisioning(&params);
 /* Start up the network stack. */
```

```
    FreeRTOS_IPInit(g_ip_address, g_net_mask, g_gateway_address, g_dns_address,
g_mac_address);
 while (pdFALSE == FreeRTOS_IsNetworkUp())
    {
       vTaskDelay(1);
    }
 /* AWS MQTT APIs can now be called. Please refer to the documentation linked above.
*/
 /* If using the MQTT Wrapper IotSdk_Init() must be called prior to using MQTT. */
}
```

4.2.121 Wifi Middleware (rm_wifi_onchip_silex)

Modules

Functions

fsp_err_t	RM_WIFI_ONCHIP_SILEX_EpochTimeGet (time_t *p_utc_time)
fsp_err_t	RM_WIFI_ONCHIP_SILEX_LocalTimeGet (uint8_t *p_local_time, uint32_t size_string)
fsp_err_t	RM_WIFI_ONCHIP_SILEX_SntpEnableSet (wifi_onchip_silex_sntp_enable_t enable)
fsp_err_t	RM_WIFI_ONCHIP_SILEX_SntpServerIpAddressSet (uint8_t *p_ip_address)
fsp_err_t	RM_WIFI_ONCHIP_SILEX_SntpTimeZoneSet (int32_t hours, uint32_t minutes, wifi_onchip_silex_sntp_daylight_savings_enable_t daylightSavingsEnable)

Detailed Description

Wifi and Socket implementation using the Silex SX-ULPGN WiFi module on RA MCUs.

Overview

This Middleware module supplies an implementation for the FreeRTOS Secure Sockets and WiFi interfaces using the Silex SX-ULPGN module.

You can find specifics about the WiFi and Secure Socket interface APIs supported by this module at these web sites: Secure Sockets API or Wifi API.

The SX-ULPGN is a low-power, compact IEEE 802.11b/g/n 2.4GHz 1x1 Wireless LAN module equipped with the Qualcomm® QCA4010 Wireless SOC. The module comes readily equipped with radio certification for Japan, North America and Europe. More information about this module can be found at the Silex Web Site

Features

The WiFi Onchip Silex Middleware driver supplies these features:

- Supports connect/disconnect to a b/g/n (2.4GHz) WiFi Access Point using Open, WPA, and WPA2 security. Encryption types can be either TKIP, or CCMP(AES).
- Supports retrieval of the module device MAC address.
- Once connected you can acquire the assigned module device IP.
- Supports a WiFi network scan capability to get a list of local Access Points.
- Supports a Ping function to test network connectivity.
- Supports a DNS Query call to retrieve the IPv4 address of a supplied URL.
- Supports a BSD style Secure Socket interface.
- Drive supports 1 or 2 UARTs for interfacing with the SX-ULPGN module. The second UART is considered optional.

Configuration

Build Time Configurations for rm_wifi_onchip_silex

The following build time configurations are defined in fsp_cfg/rm_wifi_onchip_silex_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.
Number of supported socket instances	Refer to the RA Configuration tool for available options.	1	Enable number of socket instances
Size of RX buffer for socket	Manual Entry	4096	
Size of TX buffer for CMD Port	Manual Entry	1500	
Size of RX buffer for CMD Port	Manual Entry	3000	
Semaphore maximum timeout	Manual Entry	10000	
Number of retries for AT commands	Manual Entry	10	
Module Reset Port	Refer to the RA Configuration tool for available options.	06	Specify the module reset pin port for the MCU.
Module Reset Pin	Refer to the RA Configuration tool for	03	Specify the module reset pin for the MCU.

	available options.		
Enable SNTP Client	• Enabled • Disabled	Disabled	Should the SNTP client of the module be enabled

Configurations for Networking > WiFi Onchip Silex Driver using r_sci_uart (rm_wifi_onchip_silex)

Configuration	Options	Default	Description
SNTP server IPv4 address	Must be a valid IPv4 address	0.0.0.0	
STNP Timezone Offset from UTC Hours	Must be between 12 and -12 hours	0	Value in hours from 12 to -12
STNP Timezone Offset from UTC Minutes	Must be between 0 and 59 minutes	0	Value in minutes from 0 to 59
Use Daylight Savings Time	• Enabled • Disabled	Disabled	Specify if daytime savings should be used for local time calulation

Note: It is suggested that when using the Silex Module that DTC and FIFO are enabled in the UART configuration to facilitate a more reliable data transfer between module and MCU.

Note: If you wish to use flow control then you must enable flow control in the RA Configuration editor. This can be found in the UART setting. It is advantageous to use flow control all the time since it allows the hardware to gate the flow of data across the serial bus. Without hardware flow control for faster data rate you will most likely see an overflow condition between MCU and the module device.

Note: Higher baud rates are supported in the RA Configuration editor and should be changed in the first UART configuration. There is no need to change the second UART baud rate since it is only used as an AT command channel.

Note: It is a good idea to also enable the FIFO in the UART configuration settings if you plan to use higher baud rates.

Interrupt Configuration

Refer to Serial Communications Interface (SCI) UART (r_sci_uart). R_SCI_UART_Open() is called by Wifi Middleware (rm_wifi_onchip_silex).

Clock Configuration

Refer to Serial Communications Interface (SCI) UART (r_sci_uart).

Pin Configuration

Refer to Serial Communications Interface (SCI) UART (r_sci_uart). R_SCI_UART_Open() is called by Wifi Middleware (rm_wifi_onchip_silex)

Usage Notes

Limitations

- WiFi AP connections do not currently support WEP security.
- When operating with a single UART only single socket connections are possible. To support multiple sockets two UART channels must be connected to the module. When using the Renesas-provided SX-ULPGN PMOD board the second UART channel is on pins 9 and 10 of the PMOD header.
- Network connection parameters SSID and Passphrase for the Access Point can not contain any commas. This is a current limitation of the Silex module firmware. The rm_wifi_onchip_silex_connect() function will return an error if a comma is detected.
- When operating with a single UART and there is an active socket connection you cannot call WIFI_Scan(), WIFI_Ping(), SOCKETS_GetHostByName(), WIFI_GetMAC(), or WIFI_GetIPInfo(). Calling one of these function will return an error code in this situation. These commands are blocked in the one UART case during an active socket connection because they could cause data loss. To avoid this limitation please configure the hardware to use both UARTs.
- The Silex WiFi modules SNTP support requires all configuration changes to made when the WiFi is disconnected from an Access Point. This is a limitation of the Silex module firmware. If changes to the default SNTP settings are required then the application will have to close the current AP connection, make the necessary SNTP changes, and then re-establish the original connection.

Examples

Basic Example

This is a basic example of minimal use of WiFi Middleware in an application.

```
void wifi_onchip_basic_example (void)
{
    WIFIReturnCode_t wifi_err;
 /* Setup Access Point connection parameters */
    WIFINetworkParams_t net_params =
    {
       .ucChannel                 = 0,
       .xPassword.xWPA.cPassphrase = "password",
       .ucSSID                    = "access_point_ssid",
       .xPassword.xWPA.ucLength = 8,
       .ucSSIDLength            = 17,
       .xSecurity               = eWiFiSecurityWPA2,
    };
    SocketsSockaddr_t addr            = {0};
    int32_t          number_bytes_rx = 0;
    int32_t          number_bytes_tx = 0;
    memset(scan_data, 0, sizeof(WIFIScanResult_t) * MAX_WIFI_SCAN_RESULTS);
```

```
    memset(g_socket_recv_buffer, 0, sizeof(uint8_t) * SX_WIFI_SOCKET_RX_BUFFER_SIZE);
 /* Open connection to the Wifi Module */
    wifi_err = WIFI_On();
    assert(eWiFiSuccess == wifi_err);
 /* Connect to the Access Point */
    wifi_err = WIFI_ConnectAP(&net_params);
    assert(eWiFiSuccess == wifi_err);
 /* Get address assigned by AP */
    WIFIIPConfiguration_t ipInfo;
    wifi_err = WIFI_GetIPInfo(&ipInfo);
    assert(eWiFiSuccess == wifi_err);
 /* Ping an address accessible on the network */
    uint8_t       ip_address[4] = {216, 58, 194, 174}; // NOLINT
 const uint16_t ping_count    = 3;
 const uint32_t intervalMS    = 100;
    wifi_err = WIFI_Ping(&ip_address[0], ping_count, intervalMS);
    assert(eWiFiSuccess == wifi_err);
 /* Scan the local Wifi network for other APs */
    wifi_err = WIFI_Scan(&scan_data[0], MAX_WIFI_SCAN_RESULTS);
    assert(eWiFiSuccess == wifi_err);
 /* Do a DNS Query for IP address of server */
    addr.ulAddress = SOCKETS_GetHostByName("www.renesas.com");
    addr.usPort    = SOCKETS_htons(80);
 /* Initialize the Socket Interface */
    BaseType_t sock_err = SOCKETS_Init();
    assert(pdPASS == sock_err);
 /* Create a socket instance */
    Socket_t socket1 = SOCKETS_Socket(SOCKETS_AF_INET, SOCKETS_SOCK_STREAM,
SOCKETS_IPPROTO_TCP);
    assert(NULL != socket1);
 /* Connect to an server using address */
    sock_err = SOCKETS_Connect(socket1, &addr, sizeof(SocketsSockaddr_t));
    assert(pdPASS == sock_err);
 /* Send a HTTP Get call to server */
```

```
    number_bytes_tx = SOCKETS_Send(socket1, HTTP_GET_string, strlen(HTTP_GET_string),
0);
    assert(number_bytes_tx > 0);
 /* Receive the HTTP GET call reply */
    number_bytes_rx = SOCKETS_Recv(socket1, g_socket_recv_buffer,
SX_WIFI_SOCKET_RX_BUFFER_SIZE, 0);
    assert(number_bytes_rx > 0);
 /* Close the socket connection */
    SOCKETS_Close(socket1);
 /* Shutdown the WIFI and Socket Interfaces */
    WIFI_Off();
}
```

SNTP example

An example of using Simple Network Time Protocol (SNTP) on WiFi in an application.

```
#define RM_WIFI_ONCHIP_SILEX_TEMP_BUFFER_SIZE (64)
/*
 * Example of the use of SNTP with Wifi. Example gets the epoch time and local
 * system time strings. It is also demonstrated how the user will need to disconnect
 * from the access point to make changes to the SNTP configuration during runtime.
 *
 * Function assumes that the SNTP has been enabled and configured with proper
 * SNTP server address. For brevity error checking has not been implemented.
 *
 */
void wifi_onchip_sntp_example (void)
{
 /* Setup Access Point connection parameters */
    WIFINetworkParams_t net_params =
    {
       .ucSSID                     = "access_point_ssid",
       .ucSSIDLength               = 17,
       .xPassword.xWPA.cPassphrase = "password",
```

```
        .xPassword.xWPA.ucLength    = 8,
        .ucChannel                  = 0,
        .xSecurity                  = eWiFiSecurityWPA2
    };
    uint8_t local_time[RM_WIFI_ONCHIP_SILEX_TEMP_BUFFER_SIZE];
    time_t current_sys_time = 0;
 // SNTP IP address
    uint8_t ip_address_sntp_server_valid[4] = {216, 239, 35, 0}; // NOLINT : Static
IP address
    memset(local_time, 0, sizeof(local_time));
 /* Open connection to the Wifi Module */
    WIFI_On();
 /* Connect to the Access Point */
    WIFI_ConnectAP(&net_params);
 /* Get the Epoch time in seconds since Jan 1, 1970 UTC */
 RM_WIFI_ONCHIP_SILEX_EpochTimeGet(&current_sys_time);
 /* Get the local time string */
 RM_WIFI_ONCHIP_SILEX_LocalTimeGet((uint8_t *) local_time, sizeof(local_time));
 /* Disconnect from the access point to make changes to the SNTP configuration */
    WIFI_Disconnect();
 /* Change the IP address of the server */
 RM_WIFI_ONCHIP_SILEX_SntpServerIpAddressSet((uint8_t *)
ip_address_sntp_server_valid);
 /* Change the timezone to PST with daylight saving enabled */
 RM_WIFI_ONCHIP_SILEX_SntpTimeZoneSet(-7, 0,
WIFI_ONCHIP_SILEX_SNTP_DAYLIGHT_SAVINGS_ENABLE);
 /* Connect back to the access point */
    WIFI_ConnectAP(&net_params);
 /* Get the Epoch time in seconds since Jan 1, 1970 UTC */
 RM_WIFI_ONCHIP_SILEX_EpochTimeGet(&current_sys_time);
 /* Get the local time string in format [DayOfWeek Month DayOfMonth Year
Hour:Minute:Second] */
 RM_WIFI_ONCHIP_SILEX_LocalTimeGet((uint8_t *) local_time, sizeof(local_time));
 /* Disconnect from the Access Point and shutdown the WIFI module*/
```

```
    WIFI_Disconnect();
    WIFI_Off();
}
```

Multi-Socket Example

Data Structures

struct	wifi_onchip_silex_cfg_t
struct	ulpgn_socket_t
struct	wifi_onchip_silex_instance_ctrl_t

Enumerations

enum	sx_ulpgn_socket_status_t
enum	sx_ulpgn_socket_rw
enum	wifi_onchip_silex_sntp_enable_t
enum	wifi_onchip_silex_sntp_daylight_savings_enable_t

Data Structure Documentation

◆ wifi_onchip_silex_cfg_t

struct wifi_onchip_silex_cfg_t

User configuration structure, used in open function

Data Fields		
const uint32_t	num_uarts	Number of UART interfaces to use.
const uint32_t	num_sockets	Number of sockets to initialize.
const bsp_io_port_pin_t	reset_pin	Reset pin used for module.
const uart_instance_t *	uart_instances[WIFI_ONCHIP_SILEX_CFG_MAX_NUMBER_UART_PORTS]	SCI UART instances.
const wifi_onchip_silex_sntp_enable_t	sntp_enabled	Enable/Disable the SNTP Client.
const uint8_t *	sntp_server_ip	The SNTP server IP address string.
const int32_t	sntp_timezone_offset_from_utc_hours	Timezone offset from UTC in (+/-) hours.

const uint32_t	sntp_timezone_offset_from_utc_minutes	Timezone offset from UTC in minutes.
const wifi_onchip_silex_sntp_daylight_savings_enable_t	sntp_timezone_use_daylight_savings	Enable/Disable use of daylight saving time.
void const *	p_context	User defined context passed into callback function.
void const *	p_extend	Pointer to extended configuration by instance of interface.

◆ ulpgn_socket_t

struct ulpgn_socket_t		
Silex ULPGN Wifi internal socket instance structure		
Data Fields		
StreamBufferHandle_t	socket_byteq_hdl	Socket stream buffer handle.
StaticStreamBuffer_t	socket_byteq_struct	Structure to hold stream buffer info.
uint8_t	socket_recv_buff[WIFI_ONCHIP_SILEX_CFG_MAX_SOCKET_RX_SIZE]	Socket receive buffer used by byte queue.
uint32_t	socket_status	Current socket status.
uint32_t	socket_recv_error_count	Socket receive error count.
uint32_t	socket_create_flag	Flag to determine in socket has been created.
uint32_t	socket_read_write_flag	flag to determine if read and/or write channels are active.

◆ wifi_onchip_silex_instance_ctrl_t

struct wifi_onchip_silex_instance_ctrl_t		
WIFI_ONCHIP_SILEX private control block. DO NOT MODIFY.		
Data Fields		
uint32_t	open	Flag to indicate if wifi instance has been initialized.
wifi_onchip_silex_cfg_t const *	p_wifi_onchip_silex_cfg	Pointer to initial configurations.
bsp_io_port_pin_t	reset_pin	Wifi module reset pin.
uint32_t	num_uarts	number of UARTS currently used for communication with module
uint32_t	tx_data_size	Size of the data to send.
uint32_t	num_creatable_sockets	Number of simultaneous

		sockets supported.
uint32_t	curr_cmd_port	Current UART instance index for AT commands.
uint32_t	curr_data_port	Current UART instance index for data.
uint8_t	cmd_rx_queue_buf[WIFI_ONCHIP_SILEX_CFG_CMD_RX_BUF_SIZE]	Command port receive buffer used by byte queue // FreeRTOS.
StreamBufferHandle_t	socket_byteq_hdl	Socket stream buffer handle.
StaticStreamBuffer_t	socket_byteq_struct	Structure to hold stream buffer info.
volatile uint32_t	curr_socket_index	Currently active socket instance.
uint8_t	cmd_tx_buff[WIFI_ONCHIP_SILEX_CFG_CMD_TX_BUF_SIZE]	Command send buffer.
uint8_t	cmd_rx_buff[WIFI_ONCHIP_SILEX_CFG_CMD_RX_BUF_SIZE]	Command receive buffer.
uint32_t	at_cmd_mode	Current command mode.
uint8_t	curr_ipaddr[4]	Current IP address of module.
uint8_t	curr_subnetmask[4]	Current Subnet Mask of module.
uint8_t	curr_gateway[4]	Current GAteway of module.
SemaphoreHandle_t	tx_sem	Transmit binary semaphore handle.
SemaphoreHandle_t	rx_sem	Receive binary semaphore handle.
uint8_t	last_data[WIFI_ONCHIP_SILEX_RETURN_TEXT_LENGTH]	Tailing buffer used for command parser.
uart_instance_t *	uart_instance_objects[WIFI_ONCHIP_SILEX_CFG_MAX_NUMBER_UART_PORTS]	UART instance objects.
SemaphoreHandle_t	uart_tei_sem[WIFI_ONCHIP_SILEX_CFG_MAX_NUMBER_UART_PORTS]	UART transmission end binary semaphore.
ulpgn_socket_t	sockets[WIFI_ONCHIP_SILEX_CFG_NUM_CREATEABLE_SOCKETS]	Internal socket instances.

Enumeration Type Documentation

◆ sx_ulpgn_socket_status_t

enum sx_ulpgn_socket_status_t
Silex ULPGN Wifi socket status types

◆ sx_ulpgn_socket_rw

enum sx_ulpgn_socket_rw
Silex socket shutdown channels

◆ wifi_onchip_silex_sntp_enable_t

enum wifi_onchip_silex_sntp_enable_t
Silex WiFi module enable/disable for SNTP

◆ wifi_onchip_silex_sntp_daylight_savings_enable_t

enum wifi_onchip_silex_sntp_daylight_savings_enable_t
Silex WiFi module enable/disable for SNTP

Function Documentation

◆ RM_WIFI_ONCHIP_SILEX_EpochTimeGet()

fsp_err_t RM_WIFI_ONCHIP_SILEX_EpochTimeGet (time_t * *p_utc_time*)

Get the current system time as the number of seconds since epoch 1970-01-01 00:00:00 UTC

This will retrieve time info from an NTP server at the address entered via an during configuration. If the server isn't set or the client isn't enabled, then it will return an error. The date/time is retrieved as the number of seconds since 00:00:00 UTC January 1, 1970

Parameters

[out]	p_utc_time	Returns the epoch time in seconds.

Return values

FSP_SUCCESS	Successfully retrieved the system time from module.
FSP_ERR_ASSERTION	The parameter utc_time or p_instance_ctrl is NULL.
FSP_ERR_WIFI_FAILED	Error occurred with command to Wifi module.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_WIFI_ONCHIP_SILEX_LocalTimeGet()

fsp_err_t RM_WIFI_ONCHIP_SILEX_LocalTimeGet (uint8_t * *p_local_time*, uint32_t *size_string*)

Get the current local time based on current timezone in a string format

Get the current local time based on current timezone in a string . Exp: Wed Oct 15 1975 07:06:00

Parameters

[out]	p_local_time	Returns local time in string format.
[in]	size_string	Size of p_local_time string buffer.The size of this string needs to be at least 25 bytes

Return values

FSP_SUCCESS	Successfully returned the local time string.
FSP_ERR_ASSERTION	The parameter local_time or p_instance_ctrl is NULL.
FSP_ERR_WIFI_FAILED	Error occurred with command to Wifi module.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_SIZE	String size value passed in exceeds maximum.

◆ RM_WIFI_ONCHIP_SILEX_SntpEnableSet()

fsp_err_t RM_WIFI_ONCHIP_SILEX_SntpEnableSet (wifi_onchip_silex_sntp_enable_t *enable*)

Enable or Disable the SNTP Client Service

Set the SNTP Client to Enable or Disable

Parameters

[in]	enable	Can be set to enable/disable for SNTP support.

Return values

FSP_SUCCESS	Successfully set the value.
FSP_ERR_WIFI_FAILED	Error occurred with command to Wifi module.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_WIFI_ONCHIP_SILEX_SntpServerIpAddressSet()

fsp_err_t RM_WIFI_ONCHIP_SILEX_SntpServerIpAddressSet (uint8_t * *p_ip_address*)

Update the SNTP Server IP Address

Set the SNTP Client Server IP Address

Parameters

[in]	p_ip_address	Pointer to IP address of SNTP server in byte array format.

Return values

FSP_SUCCESS	Successfully set the value.
FSP_ERR_ASSERTION	The parameter p_ip_address or p_instance_ctrl is NULL.
FSP_ERR_WIFI_FAILED	Error occurred with command to Wifi module.
FSP_ERR_NOT_OPEN	Module is not open.

◆ RM_WIFI_ONCHIP_SILEX_SntpTimeZoneSet()

fsp_err_t RM_WIFI_ONCHIP_SILEX_SntpTimeZoneSet (int32_t *hours*, uint32_t *minutes*, wifi_onchip_silex_sntp_daylight_savings_enable_t *daylightSavingsEnable*)

Update the SNTP Timezone

Set the SNTP Client Timezone

Parameters

[in]	hours	Number of hours (+/-) used for timezone offset from GMT.
[in]	minutes	Number of minutes used for timezone offset from GMT.
[in]	daylightSavingsEnable	Enable/Disable daylight saving in the timezone calculation.

Return values

FSP_SUCCESS	Successfully set the value.
FSP_ERR_WIFI_FAILED	Error occurred with command to Wifi module.
FSP_ERR_NOT_OPEN	Module is not open.
FSP_ERR_INVALID_ARGUMENT	Parameter passed into function was invalid.

4.2.122 AWS Secure Sockets

Modules

This module provides the AWS Secure Sockets implementation.

Overview

Features

Information about the features provided by the AWS Secure Sockets Library is available in the FreeRTOS Libraries User Guide.

The FSP implementation supports using Secure Sockets with either Ethernet or WiFi. These stacks can be added in FSP via the RA Configuration editor under FreeRTOS | Secure Sockets.

Dependencies

The Secure Sockets library has two dependencies:

1. A TCP/IP implementation
2. A TLS implementation

For TCP/IP, AWS have provided the FreeRTOS TCP/IP implementation. For TLS, AWS have chosen mbedTLS, but use PKCS11 for storage and invoking the crypto portion of mbedTLS. For more information about AWS Secure Sockets, refer to the AWS documentation. An example of Secure Sockets usage is on the same page.

mbedTLS

mbedTLS is ARM's implementation of the TLS and SSL protocols as well as the cryptographic primitives required by those implementations. mbedTLS is also solely used for its cryptographic features even if the TLS/SSL portions are not used. With PSA, ARM have created a separate API for cryptography. Starting with mbedTLS3, crypto implementation has been moved out to a new module called mbedCrypto (PSA Crypto API) and a build time configuration can direct the mbedTLS3 implementation to use either the old mbedtls cryptography functions or use the new PSA Crypto API. Since the current version of mbedCrypto (PSA Crypto API) implements both the old mbedtls crypto API as well as the new PSA Crypto API, either option is functional for now.

CipherSuites

During the TLS connection setup stage, the client has to indicate to the server the type of cryptographic operations that it supports. This is referred to as the ciphersuite. The entire list of ciphersuites supported by mbedTLS can be found in mbedtls/ssl_ciphersuites.h.

Configuration

In FSP, Secure Sockets can be added as a new stack via FreeRTOS | Secure Sockets | Secure Sockets on WiFi or Secure Sockets on FreeRTOS Plus TCP. All required dependant modules, except heap, are automatically added. To complete the configuration,

- Add a heap instance and use the same one for all dependencies.
- Resolve the module configuration requirements.

Usage Notes

For detailed documentation on Secure Sockets consult the AWS documentation.

Examples

Basic Example

This is a basic example of using the Secure Sockets API with Ethernet. The message "hello, world!" is sent to a remote socket.

```
#define SECURE_SOCKETS_EXAMPLE_BUFFER_SIZE (64)
static const char SERVER_CERTIFICATE_PEM[] =
 "-----BEGIN CERTIFICATE-----\n"
 "MIIDazCCAlOgAwIBAgIURabL79ayIywQv0y8SPnbZ1FYDRIwDQYJKoZIhvcNAQEL\n"
```

```
  "BQAwRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAfBgNVBAoM\n"
  "GEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDAeFw0xOTA5MTEyMTIyMjZaFw0yMDA5\n"
  "MTAyMTIyMjZaMEUxCzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEw\n"
  "HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwggEiMA0GCSqGSIb3DQEB\n"
  "AQUAA4IBDwAwggEKAoIBAQDSA3h+5sT58FHgnovnQzsVHQ0H/3TsnEKwVzyBwTQl\n"
  "s4PbG6VXCWyyJWjdJ4XMH1oU8gAlxauFbwOO98Aquei4K3Pi/ynKNBeX4VJcLyE5\n"
  "Azq7nRIIwt4+OoZ5kV7v8JIoLY5i+Ktn3zq1t0y1ZmK6Uk/rRPonb+Kx7wQPx7jq\n"
  "ZIZGda+CgF6ZedidPcABuggqD1y3U2gLiRPoBhe9nN2hG60rRp7vhbWMF0pzTDXu\n"
  "BKF7XSTbhYz3pl6NeOCLh5E3t8x908Ui5W1zDN3iOysrcwQFtCiGTvzNtxSfli1+\n"
  "PugIt9Q2vlYmuz5qI+juxHftJSXO86M5SV7exqUOXP9RAgMBAAGjUzBRMB0GA1Ud\n"
  "DgQWBBQG8VNJEJUjpTKMjmrOY3XApNp5lDAfBgNVHSMEGDAWgBQG8VNJEJUjpTKM\n"
  "jmrOY3XApNp5lDAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IBAQAt\n"
  "CabfjsYUnG8tt3/GDdhjsuG+SfeQe11S73pZi3+L616bPH5MNUv+LkgR/1AFEqt5\n"
  "WadKVTgzW5Ork1t7CfkYwrOHbyhyaaDPzERjMCfCcl8lQluBy6vE/lEb0hWq6XlO\n"
  "f6+8i+VKxWkSIXs2ZQqqYSOTTzAjHSsiiuE5WsC00ErvCvnC7uD6+3Y7W1uQRkFZ\n"
  "uSd9AN1ixPvAFi69FF/ymlJv6vII5GXOVDrIwdr50bMNuezMEx6qMNDADRH8iEaL\n"
  "JaSgfklczGiI1i7MPD4JTtsXOgKwxcBDAa0zQDVA5uBGEIOhva3m5X70N4iO7W0V\n"
  "eEhZekKeg3Fl3t/CXi8l\n"
  "-----END CERTIFICATE-----";
#define keyCLIENT_CERTIFICATE_PEM \
   "-----BEGIN CERTIFICATE-----\n" \
   "MIIDETCCAfkCFHwd2yn8zn5qB2ChYUT9Mvbi9Xp1MA0GCSqGSIb3DQEBCwUAMEUx\n" \
   "CzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRl\n" \
   "cm5ldCBXaWRnaXRzIFB0eSBMdGQwHhcNMTkwOTExMjEyMjU0WhcNMjAwOTEwMjEy\n" \
   "MjU0WjBFMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UE\n" \
   "CgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOC\n" \
   "AQ8AMIIBCgKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
   "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
   "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
   "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
   "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
   "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABMA0GCSqGSIb3DQEBCwUA\n" \
   "A4IBAQCdqq59ubdRY9EiV3bleKXeqG7+8HgBHdm0X9dgq10nD37p00YLyuZLE9NM\n" \
   "066G/VcflGrx/Nzw+/UuI7/UuBbBS/3ppHRnsZqBIl8nnr/ULrFQy8z3vKtL1q3C\n" \
```

```
  "DxabjPONlPO2keJeTTA71N/RCEMwJoa8i0XKXGdu/hQo6x4n+Gq73fEiGCl99xsc\n" \
  "4tIO4yPS4lv+uXBzEUzoEy0CLIkiDesnT5lLeCyPmUNoU89HU95IusZT7kygCHHd\n" \
  "72am1ic3X8PKc268KT3ilr3VMhK67C+iIIkfrM5AiU+oOIRrIHSC/p0RigJg3rXA\n" \
  "GBIRHvt+OYF9fDeG7U4QDJNCfGW+\n" \
  "-----END CERTIFICATE-----"
#define keyCLIENT_PRIVATE_KEY_PEM \
  "-----BEGIN RSA PRIVATE KEY-----\n" \
  "MIIEowIBAAKCAQEAo8oThJXSMDo41oL7HTpC4TX8NalBvnkFw30Av67dl/oZDjVA\n" \
  "iXPnZkhVppLnj++0/Oed0M7UwNUO2nurQt6yTYrvW7E8ZPjAlC7ueJcGYZhOaVv2\n" \
  "bhSmigjFQru2lw5odSuYy5+22CCgxft58nrRCo5Bk+GwWgZmcrxe/BzutRHQ7X4x\n" \
  "dYJhyhBOi2R1Kt8XsbuWilfgfkVhhkVklFeKqiypdQM6cnPWo/G4DyW34jOXzzEM\n" \
  "FLWvQOQLCKUZOgjJBnFdbx8oOOwMkYCChbV7gqPE6cw0Zy26CvlLQiINyonLPbNT\n" \
  "c64sS/ZBGPZFOPJmb4tG2nipYgZ1hO/r++jCbwIDAQABAoIBAQCGR2hC/ZVJhqIM\n" \
  "c2uuJZKpElpIIBBPOObZwwS3IYR4UUjzVgMn7UbbmxflLXD8lzfZU4YVp0vTH5lC\n" \
  "07qvYuXpHqtnj+GEok837VYCtUY9AuHeDM/2paV3awNV15E1PFG1Jd3pqnH7tJw6\n" \
  "VBZBDiGNNt1agN/UnoSlMfvpU0r8VGPXCBNxe3JY5QyBJPI1wF4LcxRI+eYmr7Ja\n" \
  "/cjn97DZotgz4B7gUNu8XIEkUOTwPabZINY1zcLWiXTMA+8qTniPVk653h14Xqt4\n" \
  "4o4D4YCTpwJcmxSV1m21/6+uyuXr9SIKAE+Ys2cYLA46x+rwLaW5fUoQ5hHa0Ytb\n" \
  "RYJ4SrtBAoGBANWtwlE69N0hq5xDPckSbNGubIeG8P4mBhGkJxIqYoqugGLMDiGX\n" \
  "4bltrjr2TPWaxTo3pPavLJiBMIsENA5KU+c/r0jLkxgEp9MIVJrtNgkCiDQqogBG\n" \
  "j4IJL2iQwXoLCqk2tx/dh9Mww+7SETE7EPNrv4UrYaGN5AEvpf5W+NHPAoGBAMQ6\n" \
  "wVa0Mx1PlA4enY2rfE3WXP8bzjleSOwR75JXqG2WbPC0/cszwbyPWOEqRpBZfvD/\n" \
  "QFkKx06xp1C09XwiQanr2gDucYXHeEKg/9iuJV1UkMQp95ojlhtSXdRZV7/l4pmN\n" \
  "fpB2vcAptX/4gY4tDrWMO08JNnRjE7duC+rmmk1hAoGAS4L0QLCNB/h2JOq+Uuhn\n" \
  "/FGfmOVfFPFrA6D3DbxcxpWUWVWzSLvb0SOphryzxbfEKyau7V5KbDp7ZSU/IC20\n" \
  "KOygjSEkAkDi7fjrrTRW/Cgg6g6G4YIOBO4qCtHdDbwJMHNdk6096qw5EZS67qLp\n" \
  "Apz5OZ5zChySjri/+HnTxJECgYBysGSP6IJ3fytplTtAshnU5JU2BWpi3ViBoXoE\n" \
  "bndilajWhvJO8dEqBB5OfAcCF0y6TnWtlT8oH21LHnjcNKlsRw0Dvllbd1oylybx\n" \
  "3da41dRG0sCEtoflMB7nHdDLt/DZDnoKtVvyFG6gfP47utn+Ahgn+Zp6K+46J3eP\n" \
  "s3g8AQKBgE/PJiaF8pbBXaZOuwRRA9GOMSbDIF6+jBYTYp4L9wk4+LZArKtyI+4k\n" \
  "Md2DUvHwMC+ddOtKqjYnLm+V5cSbvu7aPvBZtwxghzTUDcf7EvnA3V/bQBh3R0z7\n" \
  "pVsxTyGRmBSeLdbUWACUbX9LXdpudarPAJ59daWmP3mBEVmWdzUw\n" \
  "-----END RSA PRIVATE KEY-----"
const uint8_t g_ip_address[4] = {169, 254, 57, 49};
```

```
const uint8_t g_net_mask[4] = {255, 255, 0, 0};
const uint8_t g_gateway_address[4] = {169, 254, 57, 49};
const uint8_t g_dns_address[4] = {8, 8, 8, 8};
const uint8_t g_mac_address[6] = {0x66, 0x66, 0x66, 0x66, 0x66, 0x66};
static uint8_t g_buffer[SECURE_SOCKETS_EXAMPLE_BUFFER_SIZE];
/*************************************************************************************
***********************************
 * Refer to the following link for detailed API information:
 * https://docs.aws.amazon.com/freertos/latest/lib-
ref/html2/secure_sockets/secure_sockets_function_primary.html
 ************************************************************************************
**********************************/
void secure_sockets_ethernet_example (void)
{
 /* Initialize IOT FreeRTOS Libraries */
    SYSTEM_Init();
 /* Initialize the crypto hardware acceleration. */
 mbedtls_platform_setup(NULL);
    xLoggingTaskInitialize(256, 1, 10); // NOLINT(readability-magic-numbers)
    ProvisioningParams_t params;
 /* Write the keys into a secure region in data flash. */
    params.pucClientPrivateKey       = (uint8_t *) keyCLIENT_PRIVATE_KEY_PEM;
    params.pucClientCertificate      = (uint8_t *) keyCLIENT_CERTIFICATE_PEM;
    params.ulClientPrivateKeyLength = 1 + strlen((const char *)
params.pucClientPrivateKey);
    params.ulClientCertificateLength = 1 + strlen((const char *)
params.pucClientCertificate);
    params.pucJITPCertificate       = NULL;
    params.ulJITPCertificateLength   = 0;
    vAlternateKeyProvisioning(&params);
 /* Start up the network stack. */
    FreeRTOS_IPInit(g_ip_address, g_net_mask, g_gateway_address, g_dns_address,
g_mac_address);
 while (pdFALSE == FreeRTOS_IsNetworkUp())
```

```
    {
       vTaskDelay(1);
    }
    Socket_t socket = SOCKETS_Socket(SOCKETS_AF_INET, SOCKETS_SOCK_STREAM,
SOCKETS_IPPROTO_TCP);
 if (SOCKETS_INVALID_SOCKET == socket)
    {
 /* Could not create socket. */
       __BKPT(0);
    }
 /* Enable TLS and configure the server certificate. */
    SOCKETS_SetSockOpt(socket, 0, SOCKETS_SO_REQUIRE_TLS, NULL, (size_t) 0);
    SOCKETS_SetSockOpt(socket, 0, SOCKETS_SO_TRUSTED_SERVER_CERTIFICATE,
SERVER_CERTIFICATE_PEM,
 sizeof(SERVER_CERTIFICATE_PEM));
 /* Connect to a remote server */
    SocketsSockaddr_t server_addr;
    server_addr.usPort    = SOCKETS_htons(9001);
    server_addr.ulAddress = SOCKETS_inet_addr_quick(192, 168, 0, 3);
 if (0 != SOCKETS_Connect(socket, &server_addr, sizeof(server_addr)))
    {
 /* Could not connect to server. */
       __BKPT(0);
    }
 /* Send a message and check that the correct number of bytes were transferred */
 const char msg[] = "hello, world!\n";
 if (sizeof(msg) != SOCKETS_Send(socket, msg, sizeof(msg), 0))
    {
 /* Failed to send data. */
       __BKPT(0);
    }
 if (0 != SOCKETS_Shutdown(socket, SOCKETS_SHUT_RDWR))
    {
       __BKPT(0);
```

```
    }
 /* Follow socket shutdown example:
  * https://freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_TCP/API/close.html
  */
 while (0 <= SOCKETS_Recv(socket, g_buffer, sizeof(g_buffer), 0))
    {
       vTaskDelay(10);
    }
    SOCKETS_Close(socket);
}
const char * pcApplicationHostnameHook (void)
{
 /* Assign the name "FreeRTOS" to this network node. This function will
  * be called during the DHCP: the machine will be registered with an IP
  * address plus this name. */
 return "FreeRTOS";
}
void vApplicationIPNetworkEventHook (eIPCallbackEvent_t eNetworkEvent)
{
 FSP_PARAMETER_NOT_USED(eNetworkEvent);
}
```

4.2.123 ZMOD4XXX Sensor Middleware (rm_zmod4xxx)

Modules

Functions

fsp_err_t	RM_ZMOD4XXX_Open (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_cfg_t const *const p_cfg) This function should be called when start a measurement and when measurement data is stale data. Sends the slave address to the zmod4xxx and start a measurement. Implements rm_zmod4xxx_api_t::open. More...
fsp_err_t	RM_ZMOD4XXX_MeasurementStart (rm_zmod4xxx_ctrl_t *const p_api_ctrl)

	This function should be called when start a measurement. Implements rm_zmod4xxx_api_t::measurementStart. More...
fsp_err_t	RM_ZMOD4XXX_MeasurementStop (rm_zmod4xxx_ctrl_t *const p_api_ctrl) This function should be called when stop a measurement. Implements rm_zmod4xxx_api_t::measurementStop. More...
fsp_err_t	RM_ZMOD4XXX_StatusCheck (rm_zmod4xxx_ctrl_t *const p_api_ctrl) This function should be called when polling is used. It reads the status of sensor. Implements rm_zmod4xxx_api_t::statusCheck. More...
fsp_err_t	RM_ZMOD4XXX_Read (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data) This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::read. More...
fsp_err_t	RM_ZMOD4XXX_TemperatureAndHumiditySet (rm_zmod4xxx_ctrl_t *const p_api_ctrl, float temperature, float humidity) This function is valid only for OAQ_2nd_Gen. This function should be called before DataCalculate. Humidity and temperature measurements are needed for ambient compensation. Implements rm_zmod4xxx_api_t::temperatureAndHumiditySet. More...
fsp_err_t	RM_ZMOD4XXX_Iaq1stGenDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_1st_data_t *const p_zmod4xxx_data) This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::iaq1stGenDataCalculate. More...
fsp_err_t	RM_ZMOD4XXX_Iaq2ndGenDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_2nd_data_t *const p_zmod4xxx_data) This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::iaq2ndGenDataCalculate. More...
fsp_err_t	RM_ZMOD4XXX_OdorDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data,

	rm_zmod4xxx_odor_data_t *const p_zmod4xxx_data) This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::odorDataCalculate. More...
fsp_err_t	RM_ZMOD4XXX_SulfurOdorDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_sulfur_odor_data_t *const p_zmod4xxx_data) This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::sulfurOdorDataCalculate. More...
fsp_err_t	RM_ZMOD4XXX_Oaq1stGenDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_oaq_1st_data_t *const p_zmod4xxx_data) This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::oaq1stGenDataCalculate. More...
fsp_err_t	RM_ZMOD4XXX_Oaq2ndGenDataCalculate (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_oaq_2nd_data_t *const p_zmod4xxx_data) This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::oaq2ndGenDataCalculate. More...
fsp_err_t	RM_ZMOD4XXX_Close (rm_zmod4xxx_ctrl_t *const p_api_ctrl) This function should be called when close the sensor. Implements rm_zmod4xxx_api_t::close. More...

Detailed Description

Middleware to implement the ZMOD4XXX sensor interface. This module implements the ZMOD4XXX Middleware Interface.

Overview

This module provides an API for configuring and controlling the ZMOD4XXX sensor. Supported ZMOD4XXX sensors are below.

- ZMOD4410
- ZMOD4510

I2C communication with the ZMOD4XXX sensor is realized by connecting with the rm_comms_i2c module.

Features

The ZMOD4XXX sensor interface implementation has the following key features:

- Initialize the sensor for measurement
- Start a measurement at any time
- Read status register for wait until the measurement is done. This will also be signaled by an interrupt
- Get the ADC data from the sensor
- Input the ADC data and acquire the air quality values by calculation in the library.

Configuration

RM_ZMOD4XXX_CFG_OPERATION_MODE is set according to the library selected.

Build Time Configurations for rm_zmod4xxx

The following build time configurations are defined in fsp_cfg/rm_zmod4xxx_cfg.h:

Configuration	Options	Default	Description
Parameter Checking	• Default (BSP) • Enabled • Disabled	Default (BSP)	If selected code for parameter checking is included in the build.

Build Time Configurations for rm_zmod4410_iaq_1st

The following build time configurations are defined in fsp_cfg/rm_zmod4xxx_lib_cfg.h:

Configuration	Options	Default	Description
[DEPRECATED] Power mode (This will be removed in FSP v4.0.0)	• Continuous • Low Power	Continuous	Select power mode

Configurations for Sensor > ZMOD4XXX Gas Sensor (rm_zmod4xxx)

This module can be added to the Stacks tab via New Stack > Sensor > ZMOD4XXX Gas Sensor (rm_zmod4xxx).

Configuration	Options	Default	Description
Name	Manual Entry	g_zmod4xxx_sensor0	Module name.
Comms I2C Callback	Name must be a valid C symbol	zmod4xxx_comms_i2c_callback	A user COMMS I2C callback function can be provided.
IRQ Callback	Name must be a valid C symbol	zmod4xxx_irq_callback	A user IRQ callback function can be provided.

Configurations for Sensor > ZMOD4410 IAQ 1st Generation (rm_zmod4xxx)

Configuration	Options	Default	Description
Operation Mode	• Continuous • Low Power	Continuous	Select the operation mode.

Pin Configuration

This module use SDA and SCL pins of I2C Master and SCI I2C.

Usage Notes

ZMOD4410 datasheet is here.
The ZMOD4410 has five modes of operation.
The ZMOD4410 will respond to TVOC immediately upon start-up; however, a conditioning period of 48 hours followed by a sensor module restart in an ambient environment is recommended to improve stability and obtain maximum performance.
Best results are achieved with continuous operation because the module algorithm can learn about the environment over time.

Mode	Method	Description
Operation Mode 1	IAQ 1st Generation Continuous	Measurement of UBA levels for IAQ and eCO2, provides continuous data
Operation Mode 2	IAQ 1st Generation Low Power	Measurement of UBA levels for IAQ and eCO2, fixed sampling interval of 6 seconds
Operation Mode 3	IAQ 2nd Generation	Using AI for improved ppm TVOC, IAQ and eCO2 functionality (recommended for new designs)
Operation Mode 4	Odor	Control signal based on Air Quality Changes
Operation Mode 5	Sulfur-based Odor Discrimination	The odors in “sulfur” (sulfur based) and “acceptable” (organic based) and shows an intensity level of the smell

ZMOD4510 datasheet is here.
The ZMOD4510 has two modes of operation.
The ZMOD4510 in OAQ 1st Gen operation will respond to typical outdoor gases after a warm-up time of 60 min, consisting of 20 min for stabilization and 40 min for baseline finding.
For OAQ 2nd Gen operation a response to ozone will be seen after a warmup time of 30 min.
In all operation modes a conditioning period of 48 hours followed by a sensor module restart in an ambient environment is recommended to improve stability and obtain maximum performance.

Mode	Method	Description
Operation Mode 1	OAQ 1st Generation	Measurement of Air Quality
Operation Mode 2	OAQ 2nd Generation	Selective Ozone featuring Ultra-Low Power

A library corresponding to each of these modes is required. By setting in RA Configuration, the library will be generated in the ra/fsp/lib/rm_zmod4xxx folder of your project.

If an RTOS is used, blocking and bus lock is available.

- If blocking of an I2C bus is required, it is necessary to create a semaphore for blocking.
- If bus lock is required, it is necessary to create a mutex for bus lock. Bus lock is only available when a semaphore for blocking is used.

Bus Initialization

The ZMOD4XXX interface expects a bus instance to be opened before opening any ZMOD4XXX device. The interface will handle switching between devices on the bus but will not open or close the bus instance. The user should open the bus with the appropriate I2C Master Interface open call.

Initialization

Initialize with RM_ZMOD4XXX_Open(). One channel of timer is required to measure the waiting time at initialization.

From measurement start to data acquisition

After normal completion, start the measurement with RM_ZMOD4XXX_MeasurementStart(). An endless loop continuously checks the status of the ZMOD4XXX sensor and reads its data. The raw data is subsequently processed, and the air quality values are calculated.

If IRQ is enabled

1. Call RM_ZMOD4XXX_MeasurementStart().
2. Wait until RM_ZMOD4XXX_EVENT_MEASUREMENT_COMPLETE is received via IRQ callback.
3. Call RM_ZMOD4XXX_Read(). This function will read the ADC data.
4. Wait until RM_ZMOD4XXX_EVENT_SUCCESS is received.
5. Call the DataCalculate API according to the mode.

If IRQ is disabled

1. Call RM_ZMOD4XXX_MeasurementStart().
2. Wait until RM_ZMOD4XXX_EVENT_SUCCESS is received.
3. Call RM_ZMOD4XXX_StatusCheck(). This function will execute a status check over I2C.
4. If RM_ZMOD4XXX_EVENT_MEASUREMENT_NOT_COMPLETE is received in callback, user should wait some time and then call RM_ZMOD4XXX_StatusCheck() again.
5. Wait until RM_ZMOD4XXX_EVENT_MEASUREMENT_COMPLETE is received.
6. Call RM_ZMOD4XXX_Read() and read the ADC data.
7. Wait until RM_ZMOD4XXX_EVENT_SUCCESS is received.
8. Call the DataCalculate API according to the mode.

Examples

Basic Example

These are basic examples of minimal use of ZMOD4XXX sensor implementation in an application.

IAQ 1st Gen. Continuous mode

```
void rm_zmod4xxx_iaq_1st_gen_continuous_basic_example (void)
{
 fsp_err_t                  err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_iaq_1st_data_t zmod4410_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                            p_extend->p_blocking_semaphore->p_semaphore_name,
                            (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
```

```
                        p_extend->p_bus_recursive_mutex->p_mutex_name,
                        TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
#if ZMOD4XXX_IRQ_ENABLE
    g_zmod4xxx_irq_callback_flag = 0;
#endif
    g_zmod4xxx_i2c_callback_flag = 0;
    err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
    handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
    {
    }
    g_zmod4xxx_i2c_callback_flag = 0;
 while (1)
    {
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
```

```
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
       err = RM_ZMOD4XXX_Iaq1stGenDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4410_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
    }
}
```

IAQ 1st Gen. Low Power mode

```
void rm_zmod4xxx_iaq_1st_gen_low_power_basic_example (void)
{
 fsp_err_t                  err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_iaq_1st_data_t zmod4410_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                            p_extend->p_blocking_semaphore->p_semaphore_name,
                            (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
```

```
                        p_extend->p_bus_recursive_mutex->p_mutex_name,
                        TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
```

```
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
       err = RM_ZMOD4XXX_Iaq1stGenDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4410_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
 /* Delay required time. See Table 3 in the ZMOD4410 Programming Manual. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_5475_MS, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

IAQ 2nd Gen.

```
void rm_zmod4xxx_iaq_2nd_gen_basic_example (void)
{
 fsp_err_t                  err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_iaq_2nd_data_t zmod4410_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                            p_extend->p_blocking_semaphore->p_semaphore_name,
                            (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
```

```
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                        p_extend->p_bus_recursive_mutex->p_mutex_name,
                        TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
```

```
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
       err = RM_ZMOD4XXX_Iaq2ndGenDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4410_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
 /* Delay required time. See Table 4 in the ZMOD4410 Programming Manual. */
```

```
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_1990_MS, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

Odor

```
void rm_zmod4xxx_odor_basic_example (void)
{
 fsp_err_t               err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t raw_data;
 rm_zmod4xxx_odor_data_t zmod4410_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                            p_extend->p_blocking_semaphore->p_semaphore_name,
                            (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
```

```
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                        p_extend->p_bus_recursive_mutex->p_mutex_name,
                        TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
#if ZMOD4XXX_IRQ_ENABLE
    g_zmod4xxx_irq_callback_flag = 0;
#endif
    g_zmod4xxx_i2c_callback_flag = 0;
    err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
    handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
    {
    }
    g_zmod4xxx_i2c_callback_flag = 0;
 while (1)
    {
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
```

```
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
       err = RM_ZMOD4XXX_OdorDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4410_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
```

```
      handle_error(err);
      }
    }
}
```

Sulfur Odor

```
void rm_zmod4xxx_sulfur_odor_basic_example (void)
{
 fsp_err_t                     err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t         raw_data;
 rm_zmod4xxx_sulfur_odor_data_t zmod4410_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                            p_extend->p_blocking_semaphore->p_semaphore_name,
                            (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
```

```
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                        p_extend->p_bus_recursive_mutex->p_mutex_name,
                        TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
```

```
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
       err = RM_ZMOD4XXX_SulfurOdorDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4410_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4410_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else
```

```
      {
      handle_error(err);
      }
 /* Delay required time. See Table 6 in the ZMOD4410 Programming Manual. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_1990_MS, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

OAQ 1st Gen.

```
void rm_zmod4xxx_oaq_1st_gen_basic_example (void)
{
 fsp_err_t                  err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_oaq_1st_data_t zmod4510_data;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                            p_extend->p_blocking_semaphore->p_semaphore_name,
                            (ULONG) 0);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
```

```
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                        p_extend->p_bus_recursive_mutex->p_mutex_name,
                        TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
```

```
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
       err = RM_ZMOD4XXX_Oaq1stGenDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4510_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4510_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
```

```
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
    }
}
```

OAQ 2nd Gen.

```
void rm_zmod4xxx_oaq_2nd_gen_basic_example (void)
{
 fsp_err_t                  err = FSP_SUCCESS;
 rm_zmod4xxx_raw_data_t    raw_data;
 rm_zmod4xxx_oaq_2nd_data_t zmod4510_data;
 float temperature = ZMOD4XXX_DEFAULT_TEMPERATURE_20F;
 float humidity    = ZMOD4XXX_DEFAULT_HUMIDITY_50F;
 /* Open the I2C bus if it is not already open. */
    rm_comms_i2c_bus_extended_cfg_t * p_extend =
       (rm_comms_i2c_bus_extended_cfg_t *)
g_zmod4xxx_cfg.p_comms_instance->p_cfg->p_extend;
 i2c_master_instance_t * p_driver_instance = (i2c_master_instance_t *)
p_extend->p_driver_instance;
    p_driver_instance->p_api->open(p_driver_instance->p_ctrl,
p_driver_instance->p_cfg);
#if BSP_CFG_RTOS
 /* Create a semaphore for blocking if a semaphore is not NULL */
 if (NULL != p_extend->p_blocking_semaphore)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_semaphore_create(p_extend->p_blocking_semaphore->p_semaphore_handle,
                           p_extend->p_blocking_semaphore->p_semaphore_name,
                           (ULONG) 0);
```

```
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_blocking_semaphore->p_semaphore_handle) =
            xSemaphoreCreateCountingStatic((UBaseType_t) 1,
                                           (UBaseType_t) 0,

p_extend->p_blocking_semaphore->p_semaphore_memory);
 #endif
    }
 /* Create a recursive mutex for bus lock if a recursive mutex is not NULL */
 if (NULL != p_extend->p_bus_recursive_mutex)
    {
 #if BSP_CFG_RTOS == 1 // AzureOS
       tx_mutex_create(p_extend->p_bus_recursive_mutex->p_mutex_handle,
                        p_extend->p_bus_recursive_mutex->p_mutex_name,
                        TX_INHERIT);
 #elif BSP_CFG_RTOS == 2 // FreeRTOS
       *(p_extend->p_bus_recursive_mutex->p_mutex_handle) =
            xSemaphoreCreateRecursiveMutexStatic(p_extend->p_bus_recursive_mutex->p_m
utex_memory);
 #endif
    }
#endif
    err = RM_ZMOD4XXX_Open(&g_zmod4xxx_ctrl, &g_zmod4xxx_cfg);
 /* Handle any errors. This function should be defined by the user. */
    handle_error(err);
 while (1)
    {
#if ZMOD4XXX_IRQ_ENABLE
      g_zmod4xxx_irq_callback_flag = 0;
#endif
      g_zmod4xxx_i2c_callback_flag = 0;
      err = RM_ZMOD4XXX_MeasurementStart(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
```

```
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 do
      {
#if ZMOD4XXX_IRQ_ENABLE
 while (0U == g_zmod4xxx_irq_callback_flag)
      {
      }
      g_zmod4xxx_irq_callback_flag = 0;
#else
            err = RM_ZMOD4XXX_StatusCheck(&g_zmod4xxx_ctrl);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
#endif
            err = RM_ZMOD4XXX_Read(&g_zmod4xxx_ctrl, &raw_data);
 if (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED)
      {
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_50_MS, BSP_DELAY_UNITS_MILLISECONDS);
      }
      } while (err == FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED);
      handle_error(err);
 while (0U == g_zmod4xxx_i2c_callback_flag)
      {
      }
      g_zmod4xxx_i2c_callback_flag = 0;
 /* Set the current temperature and humidity */
       err = RM_ZMOD4XXX_TemperatureAndHumiditySet(&g_zmod4xxx_ctrl,
                                                 temperature,
                                                 humidity);
      handle_error(err);
```

```
       err = RM_ZMOD4XXX_Oaq2ndGenDataCalculate(&g_zmod4xxx_ctrl, &raw_data,
&zmod4510_data);
 if (FSP_SUCCESS == err)
      {
 /* Describe the process by referring to zmod4510_data */
      }
 else if (FSP_ERR_SENSOR_IN_STABILIZATION == err)
      {
 /* Gas data is invalid. */
      }
 else
      {
      handle_error(err);
      }
 /* Delay required time. See Table 4 in the ZMOD4510 Programming Manual. */
 R_BSP_SoftwareDelay(ZMOD4XXX_WAIT_1990_MS, BSP_DELAY_UNITS_MILLISECONDS);
    }
}
```

Data Structures

struct	rm_zmod4xxx_init_process_params_t
struct	rm_zmod4xxx_instance_ctrl_t

Enumerations

enum	rm_zmod4xxx_lib_type_t

Data Structure Documentation

◆ rm_zmod4xxx_init_process_params_t

struct rm_zmod4xxx_init_process_params_t

ZMOD4XXX initialization process block

Data Fields		
volatile uint32_t	delay_ms	Delay milliseconds.
volatile bool	communication_finished	Communication flag for blocking.
volatile bool	measurement_finished	IRQ flag.

volatile rm_zmod4xxx_event_t	event	Callback event.

◆ rm_zmod4xxx_instance_ctrl_t

struct rm_zmod4xxx_instance_ctrl_t	
ZMOD4XXX control block	
Data Fields	
uint32_t	open
	Open flag.
uint8_t	buf [RM_ZMOD4XXX_MAX_I2C_BUF_SIZE]
	Buffer for I2C communications.
uint8_t	register_address
	Register address to access.
rm_zmod4xxx_status_params_t	status
	Status parameter.
volatile rm_zmod4xxx_event_t	event
	Callback event.
rm_zmod4xxx_init_process_params_t	init_process_params
	For the initialization process.
rm_zmod4xxx_cfg_t const *	p_cfg
	Pointer of configuration block.
rm_comms_instance_t const *	p_comms_i2c_instance

	Pointer of I2C Communications Middleware instance structure.
rm_zmod4xxx_lib_extended_cfg_t *	p_zmod4xxx_lib
	Pointer of ZMOD4XXX Lib extended configuration.
void const *	p_irq_instance
	Pointer to IRQ instance.
void const *	p_context
	Pointer to the user-provided context.
void(*	p_comms_callback)(rm_zmod4xxx_callback_args_t *p_args)
	I2C Communications callback.
void(*	p_irq_callback)(rm_zmod4xxx_callback_args_t *p_args)
	IRQ callback.

Enumeration Type Documentation

◆ rm_zmod4xxx_lib_type_t

enum rm_zmod4xxx_lib_type_t
ZMOD4XXX Library type

Function Documentation

◆ RM_ZMOD4XXX_Open()

fsp_err_t RM_ZMOD4XXX_Open (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_cfg_t const *const *p_cfg*)

This function should be called when start a measurement and when measurement data is stale data. Sends the slave address to the zmod4xxx and start a measurement. Implements rm_zmod4xxx_api_t::open.

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_ALREADY_OPEN	Module is already open.
FSP_ERR_UNSUPPORTED	Unsupport product ID.
FSP_ERR_TIMEOUT	communication is timeout.
FSP_ERR_ABORTED	communication is aborted.

◆ RM_ZMOD4XXX_MeasurementStart()

fsp_err_t RM_ZMOD4XXX_MeasurementStart (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*)

This function should be called when start a measurement. Implements rm_zmod4xxx_api_t::measurementStart.

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not opened configured.
FSP_ERR_TIMEOUT	communication is timeout.
FSP_ERR_ABORTED	communication is aborted.

◆ RM_ZMOD4XXX_MeasurementStop()

fsp_err_t RM_ZMOD4XXX_MeasurementStop (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*)

This function should be called when stop a measurement. Implements rm_zmod4xxx_api_t::measurementStop.

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not opened configured.
FSP_ERR_TIMEOUT	communication is timeout.
FSP_ERR_ABORTED	communication is aborted.

◆ RM_ZMOD4XXX_StatusCheck()

fsp_err_t RM_ZMOD4XXX_StatusCheck (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*)

This function should be called when polling is used. It reads the status of sensor. Implements rm_zmod4xxx_api_t::statusCheck.

Return values

FSP_SUCCESS	Successfully started.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not opened configured.
FSP_ERR_TIMEOUT	communication is timeout.
FSP_ERR_ABORTED	communication is aborted.

◆ RM_ZMOD4XXX_Read()

fsp_err_t RM_ZMOD4XXX_Read (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*)

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::read.

Return values

FSP_SUCCESS	Successfully results are read.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not opened configured.
FSP_ERR_TIMEOUT	Communication is timeout.
FSP_ERR_ABORTED	Communication is aborted.
FSP_ERR_SENSOR_MEASUREMENT_NOT_FINISHED	Measurement is not finished.

◆ RM_ZMOD4XXX_TemperatureAndHumiditySet()

fsp_err_t RM_ZMOD4XXX_TemperatureAndHumiditySet (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, float *temperature*, float *humidity*)

This function is valid only for OAQ_2nd_Gen. This function should be called before DataCalculate. Humidity and temperature measurements are needed for ambient compensation. Implements rm_zmod4xxx_api_t::temperatureAndHumiditySet.

Return values

FSP_SUCCESS	Successfully results are read.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not opened configured.

◆ RM_ZMOD4XXX_Iaq1stGenDataCalculate()

fsp_err_t RM_ZMOD4XXX_Iaq1stGenDataCalculate (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_iaq_1st_data_t *const *p_zmod4xxx_data*)

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::iaq1stGenDataCalculate .

Return values

FSP_SUCCESS	Successfully results are read.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not opened configured.

◆ RM_ZMOD4XXX_Iaq2ndGenDataCalculate()

fsp_err_t RM_ZMOD4XXX_Iaq2ndGenDataCalculate (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_iaq_2nd_data_t *const *p_zmod4xxx_data*)

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::iaq2ndGenDataCalculate.

Return values

FSP_SUCCESS	Successfully results are read.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not opened configured.

◆ RM_ZMOD4XXX_OdorDataCalculate()

fsp_err_t RM_ZMOD4XXX_OdorDataCalculate (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_odor_data_t *const *p_zmod4xxx_data*)

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::odorDataCalculate.

Return values

FSP_SUCCESS	Successfully results are read.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not opened configured.

◆ RM_ZMOD4XXX_SulfurOdorDataCalculate()

fsp_err_t RM_ZMOD4XXX_SulfurOdorDataCalculate (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_sulfur_odor_data_t *const *p_zmod4xxx_data*)

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::sulfurOdorDataCalculate.

Return values

FSP_SUCCESS	Successfully results are read.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not opened configured.

◆ RM_ZMOD4XXX_Oaq1stGenDataCalculate()

fsp_err_t RM_ZMOD4XXX_Oaq1stGenDataCalculate (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_oaq_1st_data_t *const *p_zmod4xxx_data*)

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::oaq1stGenDataCalculate.

Return values

FSP_SUCCESS	Successfully results are read.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not opened configured.

◆ RM_ZMOD4XXX_Oaq2ndGenDataCalculate()

fsp_err_t RM_ZMOD4XXX_Oaq2ndGenDataCalculate (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*, rm_zmod4xxx_raw_data_t *const *p_raw_data*, rm_zmod4xxx_oaq_2nd_data_t *const *p_zmod4xxx_data*)

This function should be called when measurement finishes. To check measurement status either polling or busy/interrupt pin can be used. Implements rm_zmod4xxx_api_t::oaq2ndGenDataCalculate.

Return values

FSP_SUCCESS	Successfully results are read.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not opened configured.

◆ RM_ZMOD4XXX_Close()

fsp_err_t RM_ZMOD4XXX_Close (rm_zmod4xxx_ctrl_t *const *p_api_ctrl*)

This function should be called when close the sensor. Implements rm_zmod4xxx_api_t::close.

Return values

FSP_SUCCESS	Successfully closed.
FSP_ERR_ASSERTION	Null pointer passed as a parameter.
FSP_ERR_NOT_OPEN	Module is not open.

4.3 Interfaces

Detailed Description

The FSP interfaces provide APIs for common functionality. They can be implemented by one or more modules. Modules can use other modules as dependencies using this interface layer.

Modules

	ADC Interface Interface for A/D Converters.
	BLE Interface Interface for Bluetooth Low Energy functions.
	CAC Interface Interface for clock frequency accuracy measurements.
	CAN Interface Interface for CAN peripheral.
	CEC Interface Interface for CEC peripheral.
	CGC Interface Interface for clock generation.
	Comparator Interface Interface for comparators.
	CRC Interface Interface for cyclic redundancy checking.
	CTSU Interface Interface for Capacitive Touch Sensing Unit (CTSU) functions.

	DAC Interface Interface for D/A converters.
	Display Interface Interface for LCD panel displays.
	DOC Interface Interface for the Data Operation Circuit.
	ELC Interface Interface for the Event Link Controller.
	Ethernet Interface Interface for Ethernet functions.
	Ethernet PHY Interface Interface for Ethernet PHY functions.
	External IRQ Interface Interface for detecting external interrupts.
	Flash Interface Interface for the Flash Memory.
	I2C Master Interface Interface for I2C master communication.
	I2C Slave Interface Interface for I2C slave communication.
	I2S Interface Interface for I2S audio communication.

	I3C Interface Interface for I3C.
	I/O Port Interface Interface for accessing I/O ports and configuring I/O functionality.
	JPEG Codec Interface Interface for JPEG functions.
	Key Matrix Interface Interface for key matrix functions.
	Low Power Modes Interface Interface for accessing low power modes.
	Low Voltage Detection Interface Interface for Low Voltage Detection.
	OPAMP Interface Interface for Operational Amplifiers.
	PDC Interface Interface for PDC functions.
	POEG Interface Interface for the Port Output Enable for GPT.
	PTP Interface Interface for PTP functions.
	RTC Interface Interface for accessing the Realtime Clock.
	SD/MMC Interface

	Interface for accessing SD, eMMC, and SDIO devices.
	SLCDC Interface Interface for Segment LCD controllers.
	SPI Interface Interface for SPI communications.
	SPI Flash Interface Interface for accessing external SPI flash devices.
	Three-Phase Interface Interface for three-phase timer functions.
	Timer Interface Interface for timer functions.
	Transfer Interface Interface for data transfer functions.
	UART Interface Interface for UART communications.
	USB Interface Interface for USB functions.
	USB HCDC Interface Interface for USB HCDC functions.
	USB HHID Interface Interface for USB HHID functions.
	USB HMSC Interface Interface for USB HMSC functions.

	USB PCDC Interface Interface for USB PCDC functions.
	USB PHID Interface Interface for USB PHID functions.
	USB PMSC Interface Interface for USB PMSC functions.
	WDT Interface Interface for watch dog timer functions.
	ADPCM Decoder Interface Interface for ADPCM decoder.
	AUDIO PLAYBACK Interface Interface for the Audio Playback.
	BLE ABS Interface Interface for Bluetooth Low Energy Abstraction functions.
	Block Media Interface Interface for block media memory access.
	Communicatons Middleware Interface Interface for Communications Middleware functions.
	FileX Block Media Port Interface Interface for FileX Block Media port.
	FreeRTOS+FAT Port Interface Interface for FreeRTOS+FAT port.

	FSXXXX Middleware Interface Interface for FSXXXX Middleware functions.
	HS300X Middleware Interface Interface for HS300X Middleware functions.
	LittleFS Interface Interface for LittleFS access.
	Interface Interface for motor 120 control functions.
	Interface Interface for motor driver functions.
	Motor angle Interface Interface for motor angle and speed calculation functions.
	Motor Interface Interface for Motor functions.
	Motor current Interface Interface for motor current functions.
	Motor driver Interface Interface for motor driver functions.
	Motor position Interface Interface for motor position functions.
	Motor speed Interface Interface for motor speed functions.
	Touch Middleware Interface

	Interface for Touch Middleware functions.
	Virtual EEPROM Interface Interface for Virtual EEPROM access.
	ZMOD4XXX Middleware Interface Interface for ZMOD4XXX Middleware functions.
	SCE Interface Interface for Secure Crypto Engine (SCE) functions.

4.3.1 ADC Interface

Interfaces

Detailed Description

Interface for A/D Converters.

Summary

The ADC interface provides standard ADC functionality including one-shot mode (single scan), continuous scan and group scan. It also allows configuration of hardware and software triggers for starting scans. After each conversion an interrupt can be triggered, and if a callback function is provided, the call back is invoked with the appropriate event information.

Implemented by: Analog to Digital Converter (r_adc)

Data Structures

struct	adc_status_t
struct	adc_callback_args_t
struct	adc_info_t
struct	adc_cfg_t
struct	adc_api_t
struct	adc_instance_t

Typedefs

typedef void	adc_ctrl_t

Enumerations

enum	adc_mode_t
enum	adc_resolution_t
enum	adc_alignment_t
enum	adc_trigger_t
enum	adc_event_t
enum	adc_channel_t
enum	adc_group_id_t
enum	adc_group_mask_t
enum	adc_state_t

Data Structure Documentation

◆ adc_status_t

struct adc_status_t

ADC status.

Data Fields		
adc_state_t	state	Current state.

◆ adc_callback_args_t

struct adc_callback_args_t

ADC callback arguments definitions

Data Fields		
uint16_t	unit	ADC device in use.
adc_event_t	event	ADC callback event.
void const *	p_context	Placeholder for user data.
adc_channel_t	channel	Channel of conversion result. Only valid for r_adc ADC_EVENT _CONVERSION_COMPLETE.
uint64_t	channel_mask	Channel mask for conversion result. Only valid for r_adc_b.

adc_group_mask_t	group_mask	Group Mask.

◆ adc_info_t

struct adc_info_t		
ADC Information Structure for Transfer Interface		
Data Fields		
__I void *	p_address	The address to start reading the data from.
uint32_t	length	The total number of transfers to read.
transfer_size_t	transfer_size	The size of each transfer.
elc_peripheral_t	elc_peripheral	Name of the peripheral in the ELC list.
elc_event_t	elc_event	Name of the ELC event for the peripheral.
uint32_t	calibration_data	Temperature sensor calibration data (0xFFFFFFFF if unsupported) for reference voltage.
int16_t	slope_microvolts	Temperature sensor slope in microvolts/degrees C.
bool	calibration_ongoing	Calibration is in progress.

◆ adc_cfg_t

struct adc_cfg_t	
ADC general configuration	
Data Fields	
uint16_t	unit
	ADC unit to be used.
adc_mode_t	mode
	ADC operation mode.
adc_resolution_t	resolution
	ADC resolution.

adc_alignment_t	alignment
	Specify left or right alignment; ignored if addition used.
adc_trigger_t	trigger
	Default and Group A trigger source.
IRQn_Type	scan_end_irq
	Scan end IRQ number.
IRQn_Type	scan_end_b_irq
	Scan end group B IRQ number.
uint8_t	scan_end_ipl
	Scan end interrupt priority.
uint8_t	scan_end_b_ipl
	Scan end group B interrupt priority.
void(*	p_callback)(adc_callback_args_t *p_args)
	Callback function; set to NULL for none.
void const *	p_context
	Placeholder for user data. Passed to the user callback in adc_callback_args_t.
void const *	p_extend
	Extension parameter for hardware specific settings.

◆ **adc_api_t**

struct adc_api_t	
ADC functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)
fsp_err_t(*	scanCfg)(adc_ctrl_t *const p_ctrl, void const *const p_extend)
fsp_err_t(*	scanStart)(adc_ctrl_t *const p_ctrl)
fsp_err_t(*	scanGroupStart)(adc_ctrl_t *p_ctrl, adc_group_mask_t group_mask)
fsp_err_t(*	scanStop)(adc_ctrl_t *const p_ctrl)
fsp_err_t(*	scanStatusGet)(adc_ctrl_t *const p_ctrl, adc_status_t *p_status)
fsp_err_t(*	read)(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data)
fsp_err_t(*	read32)(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data)
fsp_err_t(*	calibrate)(adc_ctrl_t *const p_ctrl, void const *p_extend)
fsp_err_t(*	offsetSet)(adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset)
fsp_err_t(*	callbackSet)(adc_ctrl_t *const p_api_ctrl, void(*p_callback)(adc_callback_args_t *), void const *const p_context, adc_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(adc_ctrl_t *const p_ctrl)
fsp_err_t(*	infoGet)(adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)

Field Documentation

◆ open

fsp_err_t(* adc_api_t::open) (adc_ctrl_t *const p_ctrl, adc_cfg_t const *const p_cfg)

Initialize ADC Unit; apply power, set the operational mode, trigger sources, interrupt priority, and configurations common to all channels and sensors.

Implemented as

- R_ADC_Open()
- R_ADC_B_Open()
- R_SDADC_Open()

Precondition

Configure peripheral clocks, ADC pins and IRQs prior to calling this function.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_cfg	Pointer to configuration structure

◆ scanCfg

fsp_err_t(* adc_api_t::scanCfg) (adc_ctrl_t *const p_ctrl, void const *const p_extend)

Configure the scan including the channels, groups, and scan triggers to be used for the unit that was initialized in the open call. Some configurations are not supported for all implementations. See implementation for details.

Implemented as

- R_ADC_ScanCfg()
- R_ADC_B_ScanCfg()
- R_SDADC_ScanCfg()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_extend	See implementation for details

◆ scanStart

fsp_err_t(* adc_api_t::scanStart) (adc_ctrl_t *const p_ctrl)

Start the scan (in case of a software trigger), or enable the hardware trigger.

Implemented as

- R_ADC_ScanStart()
- R_SDADC_ScanStart()

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ scanGroupStart

fsp_err_t(* adc_api_t::scanGroupStart) (adc_ctrl_t *p_ctrl, adc_group_mask_t group_mask)

Start the scan group (in case of a software trigger), or enable the hardware trigger.

Implemented as

- R_ADC_B_ScanGroupStart()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	group_mask	Mask of groups to start

◆ scanStop

fsp_err_t(* adc_api_t::scanStop) (adc_ctrl_t *const p_ctrl)

Stop the ADC scan (in case of a software trigger), or disable the hardware trigger.

Implemented as

- R_ADC_ScanStop()
- R_SDADC_ScanStop()

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ scanStatusGet

fsp_err_t(* adc_api_t::scanStatusGet) (adc_ctrl_t *const p_ctrl, adc_status_t *p_status)

Check scan status.

Implemented as

- R_ADC_StatusGet()
- R_ADC_B_StatusGet()
- R_SDADC_StatusGet()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[out]	p_status	Pointer to store current status in

◆ read

fsp_err_t(* adc_api_t::read) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint16_t *const p_data)

Read ADC conversion result.

Implemented as

- R_ADC_Read()
- R_ADC_B_Read()
- R_SDADC_Read()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	reg_id	ADC channel to read (see enumeration adc_channel_t)
[in]	p_data	Pointer to variable to load value into.

◆ read32

fsp_err_t(* adc_api_t::read32) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, uint32_t *const p_data)

Read ADC conversion result into a 32-bit word.

Implemented as

- R_ADC_Read32()
- R_ADC_B_Read32()
- R_SDADC_Read32()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	reg_id	ADC channel to read (see enumeration adc_channel_t)
[in]	p_data	Pointer to variable to load value into.

◆ calibrate

fsp_err_t(* adc_api_t::calibrate) (adc_ctrl_t *const p_ctrl, void const *p_extend)

Calibrate ADC or associated PGA (programmable gain amplifier). The driver may require implementation specific arguments to the p_extend input. Not supported for all implementations. See implementation for details.

Implemented as

- R_ADC_Calibrate()
- R_ADC_B_Calibrate()
- R_SDADC_Calibrate()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_extend	Pointer to implementation specific arguments

◆ offsetSet

fsp_err_t(* adc_api_t::offsetSet) (adc_ctrl_t *const p_ctrl, adc_channel_t const reg_id, int32_t const offset)

Set offset for input PGA configured for differential input. Not supported for all implementations. See implementation for details.

Implemented as

- R_SDADC_OffsetSet()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	reg_id	ADC channel to read (see enumeration adc_channel_t)
[in]	offset	See implementation for details.

◆ callbackSet

fsp_err_t(* adc_api_t::callbackSet) (adc_ctrl_t *const p_api_ctrl, void(*p_callback)(adc_callback_args_t *), void const *const p_context, adc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_ADC_CallbackSet()
- R_ADC_B_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the ADC control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* adc_api_t::close) (adc_ctrl_t *const p_ctrl)
Close the specified ADC unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.

Implemented as

- R_ADC_Close()
- R_ADC_B_Close()
- R_SDADC_Close()

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ infoGet

fsp_err_t(* adc_api_t::infoGet) (adc_ctrl_t *const p_ctrl, adc_info_t *const p_adc_info)
Return the ADC data register address of the first (lowest number) channel and the total number of bytes to be read in order for the DTC/DMAC to read the conversion results of all configured channels. Return the temperature sensor calibration and slope data.

Implemented as

- R_ADC_InfoGet()
- R_ADC_B_InfoGet()
- R_SDADC_InfoGet()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[out]	p_adc_info	Pointer to ADC information structure

◆ adc_instance_t

struct adc_instance_t
This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
adc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
adc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
void const *	p_channel_cfg	Pointer to the channel configuration structure for this instance.
adc_api_t const *	p_api	Pointer to the API structure for

		this instance.

Typedef Documentation

◆ adc_ctrl_t

typedef void adc_ctrl_t
ADC control block. Allocate using driver instance control structure from driver instance header file.

Enumeration Type Documentation

◆ adc_mode_t

enum adc_mode_t

ADC operation mode definitions

Enumerator	
ADC_MODE_SINGLE_SCAN	Single scan - one or more channels.
ADC_MODE_GROUP_SCAN	Two trigger sources to trigger scan for two groups which contain one or more channels.
ADC_MODE_CONTINUOUS_SCAN	Continuous scan - one or more channels.

◆ adc_resolution_t

enum adc_resolution_t

ADC data resolution definitions

Enumerator	
ADC_RESOLUTION_12_BIT	12 bit resolution
ADC_RESOLUTION_10_BIT	10 bit resolution
ADC_RESOLUTION_8_BIT	8 bit resolution
ADC_RESOLUTION_14_BIT	14 bit resolution
ADC_RESOLUTION_16_BIT	16 bit resolution
ADC_RESOLUTION_24_BIT	24 bit resolution

◆ adc_alignment_t

enum adc_alignment_t	
ADC data alignment definitions	
Enumerator	
ADC_ALIGNMENT_RIGHT	Data alignment right.
ADC_ALIGNMENT_LEFT	Data alignment left.

◆ adc_trigger_t

enum adc_trigger_t	
ADC trigger mode definitions	
Enumerator	
ADC_TRIGGER_SOFTWARE	Software trigger; not for group modes.
ADC_TRIGGER_SYNC_ELC	Synchronous trigger via ELC.
ADC_TRIGGER_ASYNC_EXTERNAL	External asynchronous trigger; not for group modes.

◆ adc_event_t

enum adc_event_t	
ADC callback event definitions	
Enumerator	
ADC_EVENT_SCAN_COMPLETE	Normal/Group A scan complete.
ADC_EVENT_SCAN_COMPLETE_GROUP_B	Group B scan complete.
ADC_EVENT_CALIBRATION_COMPLETE	Calibration complete.
ADC_EVENT_CONVERSION_COMPLETE	Conversion complete.
ADC_EVENT_CALIBRATION_REQUEST	Calibration requested.
ADC_EVENT_CONVERSION_ERROR	Scan error.
ADC_EVENT_OVERFLOW	Overflow occurred.
ADC_EVENT_LIMIT_CLIP	Limiter clipping occurred.
ADC_EVENT_FIFO_READ_REQUEST	FIFO read requested.
ADC_EVENT_FIFO_OVERFLOW	FIFO overflow occurred.
ADC_EVENT_WINDOW_COMPARE_A	Window A comparison condition met.
ADC_EVENT_WINDOW_COMPARE_B	Window B comparison condition met.

◆ adc_channel_t

enum adc_channel_t	
ADC channels	
Enumerator	
ADC_CHANNEL_0	ADC channel 0.
ADC_CHANNEL_1	ADC channel 1.
ADC_CHANNEL_2	ADC channel 2.
ADC_CHANNEL_3	ADC channel 3.
ADC_CHANNEL_4	ADC channel 4.

ADC_CHANNEL_5	ADC channel 5.
ADC_CHANNEL_6	ADC channel 6.
ADC_CHANNEL_7	ADC channel 7.
ADC_CHANNEL_8	ADC channel 8.
ADC_CHANNEL_9	ADC channel 9.
ADC_CHANNEL_10	ADC channel 10.
ADC_CHANNEL_11	ADC channel 11.
ADC_CHANNEL_12	ADC channel 12.
ADC_CHANNEL_13	ADC channel 13.
ADC_CHANNEL_14	ADC channel 14.
ADC_CHANNEL_15	ADC channel 15.
ADC_CHANNEL_16	ADC channel 16.
ADC_CHANNEL_17	ADC channel 17.
ADC_CHANNEL_18	ADC channel 18.
ADC_CHANNEL_19	ADC channel 19.
ADC_CHANNEL_20	ADC channel 20.
ADC_CHANNEL_21	ADC channel 21.
ADC_CHANNEL_22	ADC channel 22.
ADC_CHANNEL_23	ADC channel 23.
ADC_CHANNEL_24	ADC channel 24.
ADC_CHANNEL_25	ADC channel 25.
ADC_CHANNEL_26	ADC channel 26.
ADC_CHANNEL_27	ADC channel 27.
ADC_CHANNEL_28	ADC channel 28.

ADC_CHANNEL_DUPLEX_A	Data duplexing register A.
ADC_CHANNEL_DUPLEX_B	Data duplexing register B.
ADC_CHANNEL_DUPLEX	Data duplexing register.
ADC_CHANNEL_TEMPERATURE	Temperature sensor output.
ADC_CHANNEL_VOLT	Internal reference voltage.
ADC_CHANNEL_0	ADC channel 0.
ADC_CHANNEL_1	ADC channel 1.
ADC_CHANNEL_2	ADC channel 2.
ADC_CHANNEL_3	ADC channel 3.
ADC_CHANNEL_4	ADC channel 4.
ADC_CHANNEL_5	ADC channel 5.
ADC_CHANNEL_6	ADC channel 6.
ADC_CHANNEL_7	ADC channel 7.
ADC_CHANNEL_8	ADC channel 8.
ADC_CHANNEL_9	ADC channel 9.
ADC_CHANNEL_10	ADC channel 10.
ADC_CHANNEL_11	ADC channel 11.
ADC_CHANNEL_12	ADC channel 12.
ADC_CHANNEL_13	ADC channel 13.
ADC_CHANNEL_14	ADC channel 14.
ADC_CHANNEL_15	ADC channel 15.
ADC_CHANNEL_16	ADC channel 16.
ADC_CHANNEL_17	ADC channel 17.
ADC_CHANNEL_18	ADC channel 18.

ADC_CHANNEL_19	ADC channel 19.
ADC_CHANNEL_20	ADC channel 20.
ADC_CHANNEL_21	ADC channel 21.
ADC_CHANNEL_22	ADC channel 22.
ADC_CHANNEL_23	ADC channel 23.
ADC_CHANNEL_24	ADC channel 24.
ADC_CHANNEL_25	ADC channel 25.
ADC_CHANNEL_26	ADC channel 26.
ADC_CHANNEL_27	ADC channel 27.
ADC_CHANNEL_28	ADC channel 28.
ADC_CHANNEL_SELF_DIAGNOSIS	Self-Diagnosis channel.
ADC_CHANNEL_TEMPERATURE	Temperature sensor output.
ADC_CHANNEL_VOLT	Internal reference voltage.
ADC_CHANNEL_DA0	D/A Converter Channel 0.
ADC_CHANNEL_DA1	D/A Converter Channel 1.
ADC_CHANNEL_DA2	D/A Converter Channel 2.
ADC_CHANNEL_DA3	D/A Converter Channel 3.

◆ adc_group_id_t

enum adc_group_id_t	
Enumerator	
ADC_GROUP_ID_0	Group ID 0.
ADC_GROUP_ID_1	Group ID 1.
ADC_GROUP_ID_2	Group ID 2.
ADC_GROUP_ID_3	Group ID 3.
ADC_GROUP_ID_4	Group ID 4.
ADC_GROUP_ID_5	Group ID 5.
ADC_GROUP_ID_6	Group ID 6.
ADC_GROUP_ID_7	Group ID 7.
ADC_GROUP_ID_8	Group ID 8.

◆ adc_group_mask_t

enum adc_group_mask_t	
Enumerator	
ADC_GROUP_MASK_NONE	Group Mask Unknown or None.
ADC_GROUP_MASK_0	Group Mask 0.
ADC_GROUP_MASK_1	Group Mask 1.
ADC_GROUP_MASK_2	Group Mask 2.
ADC_GROUP_MASK_3	Group Mask 3.
ADC_GROUP_MASK_4	Group Mask 4.
ADC_GROUP_MASK_5	Group Mask 5.
ADC_GROUP_MASK_6	Group Mask 6.
ADC_GROUP_MASK_7	Group Mask 7.
ADC_GROUP_MASK_8	Group Mask 8.
ADC_GROUP_MASK_ALL	All Groups.

◆ adc_state_t

enum adc_state_t	
ADC states.	
Enumerator	
ADC_STATE_IDLE	ADC is idle.
ADC_STATE_SCAN_IN_PROGRESS	ADC scan in progress.

4.3.2 BLE Interface

Interfaces

Functions

ble_status_t	R_BLE_Open (void) Open the BLE protocol stack. More...
ble_status_t	R_BLE_Close (void) Close the BLE protocol stack. More...
ble_status_t	R_BLE_Execute (void) Execute the BLE task. More...
uint32_t	R_BLE_IsTaskFree (void) Check the BLE task queue is free or not. More...
ble_status_t	R_BLE_SetEvent (ble_event_cb_t cb) Set event. More...
uint32_t	R_BLE_GetVersion (void) Get the BLE FIT module version. More...
uint32_t	R_BLE_GetLibType (void) Get the type of BLE protocol stack library. More...

Detailed Description

Interface for Bluetooth Low Energy functions.

Summary

The BLE interface for the Bluetooth Low Energy (BLE) peripheral provides Bluetooth Low Energy functionality.

The Bluetooth Low Energy interface can be implemented by:

- Bluetooth Low Energy Library of Extended Configuration (r_ble)
- Bluetooth Low Energy Library of Balance Configuration (r_ble)
- Bluetooth Low Energy Library of Compact Configuration (r_ble)

Modules

	GAP

	GATT_COMMON
	GATT_SERVER
	GATT_CLIENT
	L2CAP
	VS

Macros

#define	BLE_VERSION_MAJOR
#define	BLE_VERSION_MINOR
#define	BLE_LIB_EXTENDED
#define	BLE_LIB_BALANCE
#define	BLE_LIB_COMPACT

Typedefs

typedef void(*	ble_event_cb_t) (void) ble_event_cb_t is the callback function type for R_BLE_SetEvent(). More...

Macro Definition Documentation

◆ BLE_VERSION_MAJOR

```
#define BLE_VERSION_MAJOR
```

BLE Module Major Version.

◆ BLE_VERSION_MINOR

```
#define BLE_VERSION_MINOR
```

BLE Module Minor Version.

◆ BLE_LIB_EXTENDED

```
#define BLE_LIB_EXTENDED
```

BLE Protocol Stack Library Extended type.

◆ BLE_LIB_BALANCE

#define BLE_LIB_BALANCE
BLE Protocol Stack Library Balance type.

◆ BLE_LIB_COMPACT

#define BLE_LIB_COMPACT
BLE Protocol Stack Library Compacy type.

Typedef Documentation

◆ ble_event_cb_t

ble_event_cb_t

ble_event_cb_t is the callback function type for R_BLE_SetEvent().

Parameters

[in]	void	

Returns
none

Function Documentation

◆ R_BLE_Open()

ble_status_t R_BLE_Open (void)

Open the BLE protocol stack.

This function should be called once before using the BLE protocol stack.

Return values

BLE_SUCCESS(0x0000)	Success

◆ R_BLE_Close()

ble_status_t R_BLE_Close (void)

Close the BLE protocol stack.

This function should be called once to close the BLE protocol stack.

Return values

BLE_SUCCESS(0x0000)	Success

◆ R_BLE_Execute()

ble_status_t R_BLE_Execute (void)

Execute the BLE task.

This handles all the task queued in the BLE protocol stack internal task queue and return. This function should be called repeatedly in the main loop.

Return values

BLE_SUCCESS(0x0000)	Success

◆ R_BLE_IsTaskFree()

uint32_t R_BLE_IsTaskFree (void)

Check the BLE task queue is free or not.

This function returns the BLE task queue free status. When this function returns 0x0, call R_BLE_Execute() to execute the BLE task.

Return values

0x0	BLE task queue is not free
0x1	BLE task queue is free

◆ R_BLE_SetEvent()

ble_status_t R_BLE_SetEvent (ble_event_cb_t *cb*)

Set event.

This function add an event in the BLE protocol stack internal queue. The event is handled in R_BLE_Execute just like Bluetooth event. This function is intended to be called in hardware interrupt context. Even if calling this function with the same cb before the cb is invoked, only one event is registered. The maximum number of the events can be registered at a time is eight.

Parameters

cb	The callback for the event.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_ALREADY_IN_PROGRESS(0x000A)	The event already registered with the callback.
BLE_ERR_CONTEXT_FULL(0x000B)	No free slot for the event.

◆ R_BLE_GetVersion()

uint32_t R_BLE_GetVersion (void)

Get the BLE FIT module version.

This function returns the BLE FIT module version.
The major version(BLE_VERSION_MAJOR) is contained in the two most significant bytes, and the minor version(BLE_VERSION_MINOR) occupies the remaining two bytes.

Return values

BLE_VERSION_MAJOR | BLE_VERSION_MINOR	

◆ R_BLE_GetLibType()

uint32_t R_BLE_GetLibType (void)

Get the type of BLE protocol stack library.

This function returns the type of BLE protocol stack library.

Return values

BLE_LIB_EXTENDED(0x00)	Extended
BLE_LIB_BALANCE(0x01)	Balance
BLE_LIB_COMPACT(0x02)	Compact

4.3.2.1 GAP

Interfaces » BLE Interface

Functions

ble_status_t	R_BLE_GAP_Init (ble_gap_app_cb_t gap_cb) Initialize the Host Stack. More...
ble_status_t	R_BLE_GAP_Terminate (void) Terminate the Host Stack. More...
ble_status_t	R_BLE_GAP_UpdConn (uint16_t conn_hdl, uint8_t mode, uint16_t accept, st_ble_gap_conn_param_t *p_conn_updt_param) Update the connection parameters. More...
ble_status_t	R_BLE_GAP_SetDataLen (uint16_t conn_hdl, uint16_t tx_octets, uint16_t tx_time) Update the packet size and the packet transmit time. More...
ble_status_t	R_BLE_GAP_Disconnect (uint16_t conn_hdl, uint8_t reason) Disconnect the link. More...
ble_status_t	R_BLE_GAP_SetPhy (uint16_t conn_hdl, st_ble_gap_set_phy_param_t *p_phy_param) Set the phy for connection. More...

ble_status_t	R_BLE_GAP_SetDefPhy (st_ble_gap_set_def_phy_param_t *p_def_phy_param) Set the default phy which allows remote device to change. More...
ble_status_t	R_BLE_GAP_SetPrivMode (st_ble_dev_addr_t *p_addr, uint8_t *p_privacy_mode, uint8_t device_num) Set the privacy mode. More...
ble_status_t	R_BLE_GAP_ConfWhiteList (uint8_t op_code, st_ble_dev_addr_t *p_addr, uint8_t device_num) Set White List. More...
ble_status_t	R_BLE_GAP_GetVerInfo (void) Get the version number of the Controller and the host stack. More...
ble_status_t	R_BLE_GAP_ReadPhy (uint16_t conn_hdl) Get the phy settings. More...
ble_status_t	R_BLE_GAP_ConfRslvList (uint8_t op_code, st_ble_dev_addr_t *p_addr, st_ble_gap_rslv_list_key_set_t *p_peer_irk, uint8_t device_num) Set Resolving List. More...
ble_status_t	R_BLE_GAP_EnableRpa (uint8_t enable) Enable/Disable address resolution and generation of a resolvable private address. More...
ble_status_t	R_BLE_GAP_SetRpaTo (uint16_t rpa_timeout) Set the update time of resolvable private address. More...
ble_status_t	R_BLE_GAP_ReadRpa (st_ble_dev_addr_t *p_addr) Get the resolvable private address of local device. More...
ble_status_t	R_BLE_GAP_ReadRssi (uint16_t conn_hdl) Get RSSI. More...

ble_status_t	R_BLE_GAP_ReadChMap (uint16_t conn_hdl) Get the Channel Map. More...
ble_status_t	R_BLE_GAP_SetRandAddr (uint8_t *p_random_addr) Set a random address. More...
ble_status_t	R_BLE_GAP_SetAdvParam (st_ble_gap_adv_param_t *p_adv_param) Set advertising parameters. More...
ble_status_t	R_BLE_GAP_SetAdvSresData (st_ble_gap_adv_data_t *p_adv_srsp_data) Set advertising data/scan response data/periodic advertising data. More...
ble_status_t	R_BLE_GAP_StartAdv (uint8_t adv_hdl, uint16_t duration, uint8_t max_extd_adv_evts) Start advertising. More...
ble_status_t	R_BLE_GAP_StopAdv (uint8_t adv_hdl) Stop advertising. More...
ble_status_t	R_BLE_GAP_SetPerdAdvParam (st_ble_gap_perd_adv_param_t *p_perd_adv_param) Set periodic advertising parameters. More...
ble_status_t	R_BLE_GAP_StartPerdAdv (uint8_t adv_hdl) Start periodic advertising. More...
ble_status_t	R_BLE_GAP_StopPerdAdv (uint8_t adv_hdl) Stop periodic advertising. More...
ble_status_t	R_BLE_GAP_GetRemainAdvBufSize (uint16_t *p_remain_adv_data_size, uint16_t *p_remain_perd_adv_data_size) Get buffer size for advertising data/scan response data/periodic advertising data in the Controller. More...

ble_status_t	R_BLE_GAP_RemoveAdvSet (uint8_t op_code, uint8_t adv_hdl) Delete advertising set. More...
ble_status_t	R_BLE_GAP_CreateConn (st_ble_gap_create_conn_param_t *p_param) Request for a link establishment. More...
ble_status_t	R_BLE_GAP_CancelCreateConn (void) Cancel the request for a link establishment. More...
ble_status_t	R_BLE_GAP_SetChMap (uint8_t *p_channel_map) Set the Channel Map. More...
ble_status_t	R_BLE_GAP_StartScan (st_ble_gap_scan_param_t *p_scan_param, st_ble_gap_scan_on_t *p_scan_enable) Set scan parameter and start scan. More...
ble_status_t	R_BLE_GAP_StopScan (void) Stop scan. More...
ble_status_t	R_BLE_GAP_CreateSync (st_ble_dev_addr_t *p_addr, uint8_t adv_sid, uint16_t skip, uint16_t sync_to) Request for a periodic sync establishment. More...
ble_status_t	R_BLE_GAP_CancelCreateSync (void) Cancel the request for a periodic sync establishment. More...
ble_status_t	R_BLE_GAP_TerminateSync (uint16_t sync_hdl) Terminate the periodic sync. More...
ble_status_t	R_BLE_GAP_ConfPerdAdvList (uint8_t op_code, st_ble_dev_addr_t *p_addr, uint8_t *p_adv_sid_set, uint8_t device_num) Set Periodic Advertiser List. More...
ble_status_t	R_BLE_GAP_AuthorizeDev (uint16_t conn_hdl, uint8_t author_flag)

	Authorize a remote device. More...
ble_status_t	R_BLE_GAP_GetRemDevInfo (uint16_t conn_hdl) Get the information about remote device. More...
ble_status_t	R_BLE_GAP_SetPairingParams (st_ble_gap_pairing_param_t *p_pair_param) Set the parameters using pairing. More...
ble_status_t	R_BLE_GAP_SetLocIdInfo (st_ble_dev_addr_t *p_lc_id_addr, uint8_t *p_lc_irk) Set the IRK and the identity address distributed to a remote device. More...
ble_status_t	R_BLE_GAP_SetLocCsrk (uint8_t *p_local_csrk) Set the CSRK distributed to a remote device. More...
ble_status_t	R_BLE_GAP_StartPairing (uint16_t conn_hdl) Start pairing. More...
ble_status_t	R_BLE_GAP_ReplyPairing (uint16_t conn_hdl, uint8_t response) Reply the pairing request from a remote device. More...
ble_status_t	R_BLE_GAP_StartEnc (uint16_t conn_hdl) Encryption the link. More...
ble_status_t	R_BLE_GAP_ReplyPasskeyEntry (uint16_t conn_hdl, uint32_t passkey, uint8_t response) Reply the passkey entry request. More...
ble_status_t	R_BLE_GAP_ReplyNumComp (uint16_t conn_hdl, uint8_t response) Reply the numeric comparison request. More...
ble_status_t	R_BLE_GAP_NotifyKeyPress (uint16_t conn_hdl, uint8_t key_press) Notify the input key type which a remote device inputs in the passkey entry. More...

ble_status_t	R_BLE_GAP_GetDevSecInfo (uint16_t conn_hdl, st_ble_gap_auth_info_t *p_sec_info) Get the security information about the remote device. More...
ble_status_t	R_BLE_GAP_ReplyExKeyInfoReq (uint16_t conn_hdl) Distribute the keys of local device. More...
ble_status_t	R_BLE_GAP_SetRemOobData (st_ble_dev_addr_t *p_addr, uint8_t oob_data_flag, st_ble_gap_oob_data_t *p_oob) Set the oob data from a remote device. More...
ble_status_t	R_BLE_GAP_CreateScOobData (void) Create data for oob in secure connection. More...
ble_status_t	R_BLE_GAP_SetBondInfo (st_ble_gap_bond_info_t *p_bond_info, uint8_t device_num, uint8_t *p_set_num) Set the bonding information stored in non-volatile memory to the host stack. More...
void	R_BLE_GAP_DeleteBondInfo (int32_t local, int32_t remote, st_ble_dev_addr_t *p_addr, ble_gap_del_bond_cb_t gap_del_bond_cb) This function deletes the bonding information in Host Stack. When a function for deleting the bonding information stored in non-volatile area is registered by the gap_del_bond_cb parameter, it is deleted as well as the bonding information in Host Stack. More...
ble_status_t	R_BLE_GAP_ReplyLtkReq (uint16_t conn_hdl, uint16_t ediv, uint8_t *p_peer_rand, uint8_t response) Reply the LTK request from a remote device. More...

Detailed Description

(end addtogroup BLE_API)

Data Structures

struct	st_ble_evt_data_t st_ble_evt_data_t is the type of the data notified in a GAP Event. More...

struct	st_ble_dev_addr_t st_ble_dev_addr_t is the type of bluetooth device address(BD_ADDR). More...
struct	st_ble_gap_ext_adv_param_t Advertising parameters. More...
struct	st_ble_gap_adv_data_t Advertising data/scan response data/periodic advertising data. More...
struct	st_ble_gap_perd_adv_param_t Periodic advertising parameter. More...
struct	st_ble_gap_scan_phy_param_t Scan parameters per scan PHY. More...
struct	st_ble_gap_ext_scan_param_t Scan parameters. More...
struct	st_ble_gap_scan_on_t Parameters configured when scanning starts. More...
struct	st_ble_gap_conn_param_t Connection parameters included in connection interval, slave latency, supervision timeout, ce length. More...
struct	st_ble_gap_conn_phy_param_t Connection parameters per PHY. More...
struct	st_ble_gap_create_conn_param_t Connection parameters used in R_BLE_GAP_CreateConn(). More...
struct	st_ble_gap_rslv_list_key_set_t

	IRK of a remote device and IRK type of local device used in R_BLE_GAP_ConfRslvList(). More...
struct	st_ble_gap_set_phy_param_t PHY configuration parameters used in R_BLE_GAP_SetPhy(). More...
struct	st_ble_gap_set_def_phy_param_t PHY preferences which allows a remote device to set used in R_BLE_GAP_SetDefPhy(). More...
struct	st_ble_gap_auth_info_t Pairing parameters required from a remote device or information about keys distributed from a remote device. More...
struct	st_ble_gap_key_dist_t Keys distributed from a remote device. More...
struct	st_ble_gap_key_ex_param_t This structure includes the distributed keys and negotiated LTK size. More...
struct	st_ble_gap_pairing_param_t Pairing parameters used in R_BLE_GAP_SetPairingParams(). More...
struct	st_ble_gap_oob_data_t Oob data received from the remote device. This is used in R_BLE_GAP_SetRemOobData(). More...
struct	st_ble_gap_ver_num_t Version number of host stack. More...
struct	st_ble_gap_loc_ver_info_t Version number of Controller. More...
struct	st_ble_gap_loc_dev_info_evt_t Version information of local device. More...

struct	st_ble_gap_hw_err_evt_t Hardware error that is notified from Controller. More...
struct	st_ble_gap_cmd_err_evt_t HCI Command error. More...
struct	st_ble_gap_adv_rept_t Advertising Report. More...
struct	st_ble_gap_ext_adv_rept_t Extended Advertising Report. More...
struct	st_ble_gap_perd_adv_rept_t Periodic Advertising Report. More...
struct	st_ble_gap_adv_rept_evt_t Advertising report. More...
union	st_ble_gap_adv_rept_evt_t.param Advertising Report. More...
struct	st_ble_gap_adv_set_evt_t Advertising handle. More...
struct	st_ble_gap_adv_off_evt_t Information about the advertising set which stops advertising. More...
struct	st_ble_gap_adv_data_evt_t This structure notifies that advertising data has been set to Controller by R_BLE_GAP_SetAdvSresData(). More...
struct	st_ble_gap_rem_adv_set_evt_t This structure notifies that an advertising set has been removed.

	More...
struct	st_ble_gap_conn_evt_t This structure notifies that a link has been established. More...
struct	st_ble_gap_disconn_evt_t This structure notifies that a link has been disconnected. More...
struct	st_ble_gap_rd_ch_map_evt_t This structure notifies that Channel Map has been retrieved by R_BLE_GAP_ReadChMap(). More...
struct	st_ble_gap_rd_rssi_evt_t This structure notifies that RSSI has been retrieved by R_BLE_GAP_ReadRssi(). More...
struct	st_ble_gap_dev_info_evt_t This structure notifies that information about remote device has been retrieved by R_BLE_GAP_GetRemDevInfo(). More...
struct	st_ble_gap_conn_upd_evt_t This structure notifies that connection parameters has been updated. More...
struct	st_ble_gap_conn_upd_req_evt_t This structure notifies that a request for connection parameters update has been received. More...
struct	st_ble_gap_conn_hdl_evt_t This structure notifies that a GAP Event that includes only connection handle has occurred. More...
struct	st_ble_gap_data_len_chg_evt_t This structure notifies that the packet data length has been updated. More...

struct	st_ble_gap_rd_rpa_evt_t This structure notifies that the local resolvable private address has been retrieved by R_BLE_GAP_ReadRpa(). More...
struct	st_ble_gap_phy_upd_evt_t This structure notifies that PHY for a connection has been updated. More...
struct	st_ble_gap_phy_rd_evt_t This structure notifies that the PHY settings has been retrieved by R_BLE_GAP_ReadPhy(). More...
struct	st_ble_gap_scan_req_recv_evt_t This structure notifies that a Scan Request packet has been received from a Scanner. More...
struct	st_ble_gap_sync_est_evt_t This structure notifies that a Periodic sync has been established. More...
struct	st_ble_gap_sync_hdl_evt_t This structure notifies that a GAP Event that includes only sync handle has occurred. More...
struct	st_ble_gap_white_list_conf_evt_t This structure notifies that White List has been configured. More...
struct	st_ble_gap_rslv_list_conf_evt_t This structure notifies that Resolving List has been configured. More...
struct	st_ble_gap_perd_list_conf_evt_t This structure notifies that Periodic Advertiser List has been configured. More...
struct	st_ble_gap_set_priv_mode_evt_t This structure notifies that Privacy Mode has been configured.

	More...
struct	st_ble_gap_pairing_req_evt_t This structure notifies that a pairing request from a remote device has been received. More...
struct	st_ble_gap_passkey_display_evt_t This structure notifies that a request for Passkey display in pairing has been received. More...
struct	st_ble_gap_num_comp_evt_t This structure notifies that a request for Numeric Comparison in pairing has been received. More...
struct	st_ble_gap_key_press_ntf_evt_t This structure notifies that the remote device has input a key in Passkey Entry. More...
struct	st_ble_gap_pairing_info_evt_t This structure notifies that the pairing has completed. More...
struct	st_ble_gap_enc_chg_evt_t This structure notifies that the encryption status of a link has been changed. More...
struct	st_ble_gap_peer_key_info_evt_t This structure notifies that the remote device has distributed the keys. More...
struct	st_ble_gap_ltk_req_evt_t This structure notifies that a LTK request from a remote device has been received. More...
struct	st_ble_gap_ltk_rsp_evt_t This structure notifies that local device has replied to the LTK request from the remote device. More...

struct	st_ble_gap_sc_oob_data_evt_t This structure notifies that OOB data for Secure Connections has been generated by R_BLE_GAP_CreateScOobData(). More...
struct	st_ble_gap_bond_info_t Bonding information used in R_BLE_GAP_SetBondInfo(). More...

Macros

#define	BLE_BD_ADDR_LEN
#define	BLE_MASTER
#define	BLE_SLAVE
#define	BLE_GAP_ADDR_PUBLIC
#define	BLE_GAP_ADDR_RAND
#define	BLE_GAP_ADDR_RPA_ID_PUBLIC Resolvable Private Address. More...
#define	BLE_GAP_ADDR_RPA_ID_RANDOM Resolvable Private Address. More...
#define	BLE_GAP_AD_FLAGS_LE_LIM_DISC_MODE LE Limited Discoverable Mode flag used in AD type.
#define	BLE_GAP_AD_FLAGS_LE_GEN_DISC_MODE LE General Discoverable Mode flag used in AD type.
#define	BLE_GAP_AD_FLAGS_BR_EDR_NOT_SUPPORTED BR/EDR Not Supported flag used in AD type.
#define	BLE_GAP_ADV_DATA_MODE Advertising data.
#define	BLE_GAP_SCAN_RSP_DATA_MODE

	Scan response data.
#define	BLE_GAP_PERD_ADV_DATA_MODE Periodic advertising data.
#define	BLE_GAP_ADV_CH_37 Use 37 CH.
#define	BLE_GAP_ADV_CH_38 Use 38 CH.
#define	BLE_GAP_ADV_CH_39 Use 39 CH.
#define	BLE_GAP_ADV_CH_ALL Use 37 - 39 CH.
#define	BLE_GAP_SCAN_PASSIVE Passive Scan.
#define	BLE_GAP_SCAN_ACTIVE Active Scan.
#define	BLE_GAP_SCAN_INTV_MIN Active Scan.
#define	BLE_GAP_SCAN_FILT_DUPLIC_DISABLE Duplicate filter disabled.
#define	BLE_GAP_SCAN_FILT_DUPLIC_ENABLE Duplicate filter enabled.
#define	BLE_GAP_SCAN_FILT_DUPLIC_ENABLE_FOR_PERIOD Duplicate filtering enabled, reset for each scan period.

#define	BLE_GAP_SCAN_ALLOW_ADV_ALL Accept all advertising and scan response PDUs except directed advertising PDUs not addressed to local device.
#define	BLE_GAP_SCAN_ALLOW_ADV_WLST Accept only advertising and scan response PDUs from remote devices whose address is registered in the White List. Directed advertising PDUs which are not addressed to local device is ignored.
#define	BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED Accept all advertising and scan response PDUs except directed advertising PDUs whose the target address is identity address but doesn't address local device. However directed advertising PDUs whose the target address is the local resolvable private address are accepted.
#define	BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST Accept all advertising and scan response PDUs. The following are excluded. More...
#define	BLE_GAP_INIT_FILT_USE_ADDR White List is not used.
#define	BLE_GAP_INIT_FILT_USE_WLST White List is used.
#define	BLE_GAP_DATA_0_CLEAR Clear the advertising data/scan response data/periodic advertising data in the advertising set.
#define	BLE_GAP_DATA_0_DID_UPD Update Advertising DID without changing advertising data.
#define	BLE_GAP_NET_PRIV_MODE Network Privacy Mode.
#define	BLE_GAP_DEV_PRIV_MODE

	Device Privacy Mode.
#define	BLE_GAP_REM_FEATURE_SIZE The length of the features supported by a remote device.
#define	BLE_GAP_NOT_AUTHORIZED Not authorize the remote device.
#define	BLE_GAP_AUTHORIZED Authorize the remote device.
#define	BLE_GAP_RMV_ADV_SET_REM_OP Delete an advertising set.
#define	BLE_GAP_RMV_ADV_SET_CLR_OP Delete all the advertising sets.
#define	BLE_GAP_SC_PROC_GEN General Discovery Procedure.
#define	BLE_GAP_SC_PROC_LIM Limited Discovery Procedure.
#define	BLE_GAP_SC_PROC_OBS Observation Procedure.
#define	BLE_GAP_LIST_ADD_DEV Add the device to the list.
#define	BLE_GAP_LIST_REM_DEV Delete the device from the list.
#define	BLE_GAP_LIST_CLR Clear the list.

#define	BLE_GAP_WHITE_LIST_MAX_ENTRY The maximum entry number of White List.
#define	BLE_GAP_RSLV_LIST_MAX_ENTRY The maximum entry number of Resolving List.
#define	BLE_GAP_PERD_LIST_MAX_ENTRY The maximum entry number of Periodic Advertiser List.
#define	BLE_GAP_RPA_DISABLED Disable RPA generation/resolution.
#define	BLE_GAP_RPA_ENABLED Enable RPA generation/resolution.
#define	BLE_GAP_RL_LOC_KEY_ALL_ZERO All-zero IRK.
#define	BLE_GAP_RL_LOC_KEY_REGISTERED The IRK registered by R_BLE_GAP_SetLocIdInfo().
#define	BLE_MAX_NO_OF_ADV_SETS_SUPPORTED The maximum number of advertising set for the Abstraction API.
#define	BLE_GAP_LEGACY_PROP_ADV_IND Connectable and scannable undirected Legacy Advertising Packet.
#define	BLE_GAP_LEGACY_PROP_ADV_DIRECT_IND Connectable directed (low duty cycle) Legacy Advertising Packet.
#define	BLE_GAP_LEGACY_PROP_ADV_HDC_DIRECT_IND Connectable directed (high duty cycle) Legacy Advertising Packet.

#define BLE_GAP_LEGACY_PROP_ADV_SCAN_IND

Scannable undirected Legacy Advertising Packet.

#define BLE_GAP_LEGACY_PROP_ADV_NONCONN_IND

Non-connectable and non-scannable undirected Legacy Advertising Packet.

#define BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_UNDIRECT

Connectable and non-scannable undirected Extended Advertising Packet.

#define BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_DIRECT

Connectable and non-scannable directed (low duty cycle) Extended Advertising Packet.

#define BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_HDC_DIRECT

Connectable and non-scannable directed (high duty cycle) Extended Advertising Packet.

#define BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_UNDIRECT

Non-connectable and scannable undirected Extended Advertising Packet.

#define BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_DIRECT

Non-connectable and scannable directed (low duty cycle) Extended Advertising Packet.

#define BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_HDC_DIRECT

Non-connectable and scannable directed (high duty cycle) Extended Advertising Packet.

#define BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_UNDIRECT

Non-connectable and non-scannable undirected Extended Advertising Packet.

#define BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_DIRECT

Non-connectable and non-scannable directed (low duty cycle)

Extended Advertising Packet.

#define BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_HDC_DIRECT

Non-connectable and non-scannable directed (high duty cycle) Extended Advertising Packet.

#define BLE_GAP_EXT_PROP_ADV_ANONYMOUS

Omit the advertiser address from Extended Advertising Packet.

#define BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER

Indicate that the advertising data includes TX Power.

#define BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_ANY

Process scan and connection requests from all devices.

#define BLE_GAP_ADV_ALLOW_SCAN_WLST_CONN_ANY

Process connection requests from all devices and scan requests from only devices that are in the White List.

#define BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_WLST

Process scan requests from all devices and connection requests from only devices that are in the White List.

#define BLE_GAP_ADV_ALLOW_SCAN_WLST_CONN_WLST

Process scan and connection requests from only devices in the White List.

#define BLE_GAP_ADV_PHY_1M

Use 1M PHY.

#define BLE_GAP_ADV_PHY_2M

Use 2M PHY.

#define BLE_GAP_ADV_PHY_CD

Use Coded PHY.

#define BLE_GAP_SCAN_REQ_NTF_DISABLE

Disable Scan Request Notification.

#define BLE_GAP_SCAN_REQ_NTF_ENABLE

Enable Scan Request Notification.

#define BLE_GAP_PERD_PROP_TX_POWER

Indicate that periodic advertising data includes Tx Power.

#define BLE_GAP_INVALID_ADV_HDL

Invalid advertising handle.

#define BLE_GAP_SET_PHYS_HOST_PREF_1M

Use 1M PHY.

#define BLE_GAP_SET_PHYS_HOST_PREF_2M

Use 2M PHY.

#define BLE_GAP_SET_PHYS_HOST_PREF_CD

Use Coded PHY.

#define BLE_GAP_SET_PHYS_OP_HOST_NO_PREF

No preferred coding.

#define BLE_GAP_SET_PHYS_OP_HOST_PREF_S_2

Use S=2 coding.

#define BLE_GAP_SET_PHYS_OP_HOST_PREF_S_8

Use S=8 coding.

#define BLE_GAP_CONN_UPD_MODE_REQ

Request for updating the connection parameters.

#define BLE_GAP_CONN_UPD_MODE_RSP

Reply a connection parameter update request.

#define BLE_GAP_CONN_UPD_ACCEPT

Accept the update request.

#define BLE_GAP_CONN_UPD_REJECT

Reject the update request.

#define BLE_GAP_CH_MAP_SIZE

The size of channel map.

#define BLE_GAP_INVALID_CONN_HDL

Invalid Connection handle.

#define BLE_GAP_NOT_USE_CONN_HDL

This macro indicates that connection handle is not used.

#define BLE_GAP_INIT_CONN_HDL

Initial Connection handle.

#define BLE_GAP_PAIRING_ACCEPT

Accept a request regarding pairing.

#define BLE_GAP_PAIRING_REJECT

Reject a request regarding pairing.

#define BLE_GAP_LTK_REQ_ACCEPT

Reply for the LTK request.

#define BLE_GAP_LTK_REQ_DENY

Reject the LTK request.

#define BLE_GAP_LESC_PASSKEY_ENTRY_STARTED

Notify that passkey entry started.

#define BLE_GAP_LESC_PASSKEY_DIGIT_ENTERED

Notify that passkey digit entered.

#define BLE_GAP_LESC_PASSKEY_DIGIT_ERASED

Notify that passkey digit erased.

#define BLE_GAP_LESC_PASSKEY_CLEARED

Notify that passkey cleared.

#define BLE_GAP_LESC_PASSKEY_ENTRY_COMPLETED

Notify that passkey entry completed.

#define BLE_GAP_SEC_MITM_BEST_EFFORT

MITM Protection not required.

#define BLE_GAP_SEC_MITM_STRICT

MITM Protection required.

#define BLE_GAP_KEY_DIST_ENCKEY

LTK.

#define BLE_GAP_KEY_DIST_IDKEY

IRK and Identity Address.

#define BLE_GAP_KEY_DIST_SIGNKEY

CSRK.

#define BLE_GAP_ID_ADDR_SIZE

The size of identity address.

#define BLE_GAP_IRK_SIZE

The size of IRK.

#define	BLE_GAP_CSRK_SIZE The size of CSRK.
#define	BLE_GAP_LTK_SIZE The size of LTK.
#define	BLE_GAP_EDIV_SIZE The size of EDIV.
#define	BLE_GAP_RAND_64_BIT_SIZE The size of Rand.
#define	BLE_GAP_UNAUTH_PAIRING Unauthenticated pairing.
#define	BLE_GAP_AUTH_PAIRING Authenticated pairing.
#define	BLE_GAP_LEGACY_PAIRING Legacy pairing.
#define	BLE_GAP_LESC_PAIRING Secure Connections.
#define	BLE_GAP_BONDING_NONE The device doesn't support Bonding.
#define	BLE_GAP_BONDING The device supports Bonding.
#define	BLE_GAP_IOCAP_DISPLAY_ONLY Display Only iocapability. More...

#define	BLE_GAP_IOCAP_DISPLAY_YESNO
	Display Yes/No iocapability. More...
#define	BLE_GAP_IOCAP_KEYBOARD_ONLY
	Keyboard Only iocapability. More...
#define	BLE_GAP_IOCAP_NOINPUT_NOOUTPUT
	No Input No Output iocapability. More...
#define	BLE_GAP_IOCAP_KEYBOARD_DISPLAY
	Keyboard Display iocapability. More...
#define	BLE_GAP_OOB_DATA_NOT_PRESENT
	Reply that No OOB data has been received when pairing.
#define	BLE_GAP_OOB_DATA_PRESENT
	Reply that the OOB data has been received when pairing.
#define	BLE_GAP_SC_BEST_EFFORT
	Accept Legacy pairing and Secure Connections.
#define	BLE_GAP_SC_STRICT
	Accept only Secure Connections.
#define	BLE_GAP_SC_KEY_PRESS_NTF_NOT_SPRT
	Not support for Key Press Notification.
#define	BLE_GAP_SC_KEY_PRESS_NTF_SPRT
	Support for Key Press Notification.
#define	BLE_GAP_LEGACY_OOB_SIZE
	The size of Temporary Key for OOB in legacy pairing.
#define	BLE_GAP_OOB_CONFIRM_VAL_SIZE

	The size of Confirmation Value for OOB in Secure Connections.
#define	BLE_GAP_OOB_RANDOM_VAL_SIZE The size of Rand for OOB in Secure Connections.
#define	BLE_GAP_SEC_DEL_LOC_NONE Delete no local keys.
#define	BLE_GAP_SEC_DEL_LOC_IRK Delete local IRK.
#define	BLE_GAP_SEC_DEL_LOC_CSRK Delete local CSRK.
#define	BLE_GAP_SEC_DEL_LOC_ALL Delete all local keys.
#define	BLE_GAP_SEC_DEL_REM_NONE Delete no remote device keys.
#define	BLE_GAP_SEC_DEL_REM_SA Delete a key specified by the p_addr parameter.
#define	BLE_GAP_SEC_DEL_REM_NOT_CONN Delete keys of not connected remote devices.
#define	BLE_GAP_SEC_DEL_REM_ALL Delete all remote device keys.

Typedefs

typedef void(*	ble_gap_app_cb_t) (uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t *p_event_data) ble_gap_app_cb_t is the GAP Event callback function type. More...

typedef void(*	ble_gap_del_bond_cb_t) (st_ble_dev_addr_t *p_addr) ble_gap_del_bond_cb_t is the type of the callback function for delete bonding information stored in non-volatile area. This type is used in R_BLE_GAP_DeleteBondInfo(). More...
typedef st_ble_gap_ext_adv_param_t	st_ble_gap_adv_param_t Advertising parameters. More...
typedef st_ble_gap_ext_scan_param_t	st_ble_gap_scan_param_t Scan parameters. More...

Enumerations

enum	e_ble_gap_evt_t GAP Event Identifier. More...

Data Structure Documentation

◆ st_ble_evt_data_t

struct st_ble_evt_data_t

st_ble_evt_data_t is the type of the data notified in a GAP Event.

Data Fields		
uint16_t	param_len	The size of GAP Event parameters.
void *	p_param	GAP Event parameters. This parameter differs in each GAP Event.

◆ st_ble_dev_addr_t

struct st_ble_dev_addr_t

st_ble_dev_addr_t is the type of bluetooth device address(BD_ADDR).

Note

The BD address setting format is little endian.
If the address is "AA:BB:CC:DD:EE:FF", set the byte array in the order {0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA}.

Data Fields		
uint8_t	addr[BLE_BD_ADDR_LEN]	BD_ADDR.

uint8_t	type	Bluetooth address type. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Public Address.</td></tr><tr><td>BLE_GAP_ADDR_RAND(0x01)</td><td>Random Address.</td></tr></table>

◆ **st_ble_gap_ext_adv_param_t**

struct st_ble_gap_ext_adv_param_t		
Advertising parameters.		
Data Fields		
uint8_t	adv_hdl	Advertising handle identifying the advertising set to be set the advertising parameters. Valid range is 0x00 - 0x03. In the first advertising parameters setting, the advertising set specified by adv_hdl is generated. The Advertising Set ID(Advertising SID) of the advertising set is same as adv_hdl.
uint16_t	adv_prop_type	Advertising packet type. Legacy advertising PDU type, or bitwise or of Extended advertising PDU type and Extended advertising option. <table><tr><th>category</th><th>macro</th><th>description</th></tr><tr><td>Legacy Advertising PDU type</td><td>BLE_GAP_LEGACY_PROP_ADV_IND(0x0013)</td><td>Connectable and scannable undirected Legacy Advertising Packet</td></tr><tr><td></td><td>BLE_GAP_LEGACY_PROP_ADV_DIRECT_IND(</td><td>Connectable directed (low duty</td></tr></table>

			0x0015)	cycle) Legacy Advertising Packet
			BLE_GAP_LEGACY_PROP_ADV_HDC_DIRECT_IND(0x001D)	Connectable directed (high duty cycle) Legacy Advertising Packet
			BLE_GAP_LEGACY_PROP_ADV_SCAN_IND(0x0012)	Scannable undirected Legacy Advertising Packet
			BLE_GAP_LEGACY_PROP_ADV_NONCONN_IND(0x0010)	Non-connectable and non-scannable undirected Legacy Advertising Packet
		Extended Advertising PDU type	BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_UNDIRECT(0x0001)	Connectable and non-scannable undirected Extended Advertising Packet
			BLE_GAP_EXT_PROP_ADV_CONN_NOSCAN_DIRECT(0x0005)	Connectable and non-scannable directed (low duty cycle) Extended Advertising Packet
			BLE_GAP_EXT_PR	Connectable and

			OP_ADV_CONN_NOSCAN_HDC_DIRECT(0x000D)	non-scannable directed (high duty cycle) Extended Advertising Packet
			BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_UNDIRECT(0x0002)	Non-connectable and scannable undirected Extended Advertising Packet
			BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_DIRECT(0x0006)	Non-connectable and scannable directed (low duty cycle) Extended Advertising Packet
			BLE_GAP_EXT_PROP_ADV_NOCONN_SCAN_HDC_DIRECT(0x000E)	Non-connectable and scannable directed (high duty cycle) Extended Advertising Packet
			BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_UNDIRECT(0x0000)	Non-connectable and non-scannable undirected Extended Advertising

				Packet
			BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_DIRECT(0x0004)	Non-connectable and non-scannable directed (low duty cycle) Extended Advertising Packet
			BLE_GAP_EXT_PROP_ADV_NOCONN_NOSCAN_HDC_DIRECT(0x000C)	Non-connectable and non-scannable directed (high duty cycle) Extended Advertising Packet
		Extended Advertising Option	BLE_GAP_EXT_PROP_ADV_ANONYMOUS(0x0020)	Omit the advertiser address from Extended Advertising Packet.
			BLE_GAP_EXT_PROP_ADV_INCLUDE_TX_POWER(0x0040)	Indicate that the advertising data includes TX Power.
uint32_t	adv_intv_min	Minimum advertising interval. Time(ms) = adv_intv_min * 0.625. Valid range is 0x00000020 - 0x00FFFFFF.		
uint32_t	adv_intv_max	Maximum Advertising interval.		

<table>
<tr><td></td><td></td><td>Time(ms) = adv_intv_max * 0.625.
Valid range is 0x00000020 - 0x00FFFFFF.</td></tr>
<tr><td>uint8_t</td><td>adv_ch_map</td><td>The adv_ch_map is channels used in advertising with primary advertising channels.

It is a bitwise OR of the following values.
<table>
<tr><th>macro</th><th>description</th></tr>
<tr><td>BLE_GAP_ADV_CH_37(0x01)</td><td>Use 37 CH.</td></tr>
<tr><td>BLE_GAP_ADV_CH_38(0x02)</td><td>Use 38 CH.</td></tr>
<tr><td>BLE_GAP_ADV_CH_39(0x04)</td><td>Use 39 CH.</td></tr>
<tr><td>BLE_GAP_ADV_CH_ALL(0x07)</td><td>Use 37 - 39 CH.</td></tr>
</table></td></tr>
<tr><td>uint8_t</td><td>o_addr_type</td><td>Own BD Address Type.
<table>
<tr><th>macro</th><th>description</th></tr>
<tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Public Address</td></tr>
<tr><td>BLE_GAP_ADDR_RAND(0x01)</td><td>Random Address</td></tr>
<tr><td>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)</td><td>Resolvable Private Address.
If the IRK of local device has not been registered in Resolving List, public address is used.</td></tr>
<tr><td>BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)</td><td>Resolvable Private Address.
If the IRK of local device has not been registered in Resolving List, the random</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td></td><td>address specified by the o_addr field is used.</td></tr>
<tr><td>uint8_t</td><td>o_addr[BLE_BD_ADDR_LEN]</td><td>Random address set to the advertising set, when the o_addr_type field is BLE_GAP_ADDR_RAND.

When the o_addr_type field is other than BLE_GAP_ADDR_RAND, this field is ignored.

Note
The BD address setting format is little endian.
If the address is "AA:BB:CC:DD:EE:FF", set the byte array in the order {0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA}.</td></tr>
<tr><td>uint8_t</td><td>p_addr_type</td><td>Peer address type.

When the Advertising PDU type is other than directed or the o_addr_type is BLE_GAP_ADDR_PUBLIC or BLE_GAP_ADDR_RAND,this field is ignored.

<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Public Address</td></tr><tr><td>BLE_GAP_ADDR_RAND(0x01)</td><td>Random Address</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>p_addr[BLE_BD_ADDR_LEN]</td><td>Peer address.

When the Advertising PDU type is other than directed or the o_addr_type is BLE_GAP_ADDR_PUBLIC or BLE_GAP_ADDR_RAND,this field is ignored.

Note
The BD address setting format is little endian.
If the address is</td></tr>
</table>

<table>
<tr><td></td><td></td><td>"AA:BB:CC:DD:EE:FF", set the byte array in the order {0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA}.</td></tr>
<tr><td>uint8_t</td><td>filter_policy</td><td>Advertising Filter Policy.
<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_ANY(0x00)</td><td>Process scan and connection requests from all devices.</td></tr><tr><td>BLE_GAP_ADV_ALLOW_SCAN_WLST_CONN_ANY(0x01)</td><td>Process connection requests from all devices and scan requests from only devices that are in the White List.</td></tr><tr><td>BLE_GAP_ADV_ALLOW_SCAN_ANY_CONN_WLST(0x02)</td><td>Process scan requests from all devices and connection requests from only devices that are in the White List.</td></tr><tr><td>BLE_GAP_ADV_ALLOW_SCAN_WLST_CONN_WLST(0x03)</td><td>Process scan and connection requests from only devices in the White List.</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>adv_phy</td><td>Primary ADV PHY.

In this parameter, only 1M PHY and Coded PHY can be specified, and 2M PHY cannot be specified.
<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADV_PHY_1M(0x01)</td><td>Use 1M PHY as Primary Advertising PHY.
When the adv_prop_typ</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td></td><td><table><tr><td></td><td>e field is Legacy Advertising PDU type, this field shall be set to BLE_GAP_ADV_PHY_1M.</td></tr><tr><td>BLE_GAP_ADV_PHY_CD(0x03)</td><td>Use Coded PHY(S=8) as Primary Advertising PHY. Coding scheme is configured by R_BLE_VS_SetCodingScheme().</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>sec_adv_max_skip</td><td>Secondary ADV Max Skip.

Valid range is 0x00 - 0xFF.
When this field is 0x00, AUX_ADV_IND is sent before the next advertising event.
When the adv_prop_type field is Legacy Advertising PDU, this field is ignored.</td></tr>
<tr><td>uint8_t</td><td>sec_adv_phy</td><td>Secondary ADV Phy.

When the adv_prop_type is Legacy Advertising PDU, this field is ignored.
<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADV_PHY_1M(0x01)</td><td>Use 1M PHY as Secondary Advertising PHY.</td></tr><tr><td>BLE_GAP_ADV_PHY_2M(0x02)</td><td>Use 2M PHY as Secondary Advertising PHY.</td></tr><tr><td>BLE_GAP_ADV_PHY_CD(0x03)</td><td>Use Coded PHY(S=8) as Secondary Advertising PHY.</td></tr></table>Coding scheme is configured by R_BLE_VS_SetCodingScheme().</td></tr>
<tr><td>uint8_t</td><td>scan_req_ntf_flag</td><td>Scan Request Notifications Flag.</td></tr>
</table>

		When the adv_prop_type field is non-scannable Advertising PDU, this field is ignored. **macro** / **description** BLE_GAP_SCAN_REQ_NTF_DISABLE(0x00): Disable Scan Request Notification. BLE_GAP_SCAN_REQ_NTF_ENABLE(0x01): Enable Scan Request Notification. When a Scan Request Packet from Scanner has been received, the BLE_GAP_EVENT_SCAN_REQ_RECV event is notified.

◆ st_ble_gap_adv_data_t

struct st_ble_gap_adv_data_t		
Advertising data/scan response data/periodic advertising data.		
Data Fields		
uint8_t	adv_hdl	Advertising handle identifying the advertising set to be set advertising data/scan response/periodic advertising data. Valid range is 0x00 - 0x03.
uint8_t	data_type	Data type. **macro** / **description** BLE_GAP_ADV_DATA_MODE(0x00): Advertising data. BLE_GAP_SCAN_RSP_DATA_MODE(0x01): Scan response data. BLE_GAP_PERD_ADV_DATA_MODE(0x02): Periodic advertising data.
uint16_t	data_length	The length of advertising

<table>
<tr><td></td><td></td><td>data/scan response data/periodic advertising data (in bytes).

In case of Legacy Advertising PDU, the length is 0 - 31 bytes.
In case of Extended Advertising PDU, the length is 0 - 1650 bytes.
Note that the length of the advertising data/scan response data in the BLE_MAX_NO_OF_ADV_SETS_SUPPORTED number of the advertising sets may not exceed the buffer size(4250 bytes) in Controller.

In case of periodic advertising data, the length is 0 - 1650 bytes.
Note that the length of the periodic advertising data in the BLE_MAX_NO_OF_ADV_SETS_SUPPORTED number of the advertising sets may not exceed the buffer size(4306 bytes) in Controller.

When this field is 0, the operations specified by the zero_length_flag is executed.</td></tr>
<tr><td>uint8_t *</td><td>p_data</td><td>Advertising data/scan response data/periodic advertising data.

When the data_length field is 0, this field is ignored.</td></tr>
<tr><td>uint8_t</td><td>zero_length_flag</td><td>Operation when the data_length field is 0.

If the data_length is other than 0, this field is ignored.

<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_DATA_0_CLEAR(0x01)</td><td>Clear the advertising data/scan response data/periodic advertising data in the advertising set.</td></tr></table></td></tr>
</table>

		BLE_GAP_DATA_0_DID_UPD(0x02)	Update Advertising DID without changing advertising data. If the data_type field is BLE_GAP_ADV_DATA_MODE, this value is allowed.

◆ st_ble_gap_perd_adv_param_t

<table>
<tr><th colspan="3">struct st_ble_gap_perd_adv_param_t</th></tr>
<tr><td colspan="3">Periodic advertising parameter.</td></tr>
<tr><td colspan="3">Data Fields</td></tr>
<tr><td>uint8_t</td><td>adv_hdl</td><td>Advertising handle identifying the advertising set to be set periodic advertising parameter.

Valid range is 0x00 - 0x03.</td></tr>
<tr><td>uint16_t</td><td>prop_type</td><td>Periodic ADV Properties.

The prop_type field is set to the following values.
If the type of the periodic advertising data cannot be applied from the following, set 0x0000.

<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_PERD_PROP_TX_POWER(0x0040)</td><td>Indicate that periodic advertising data includes Tx Power.</td></tr></table></td></tr>
<tr><td>uint16_t</td><td>perd_intv_min</td><td>Minimum Periodic Advertising Interval.

Time(ms) = perd_intv_min * 1.25.
Valid range is 0x0006 - 0xFFFF.</td></tr>
<tr><td>uint16_t</td><td>perd_intv_max</td><td>Maximum Periodic Advertising Interval.

Time(ms) = perd_intv_max * 1.25.
Valid range is 0x0006 - 0xFFFF.</td></tr>
</table>

◆ st_ble_gap_scan_phy_param_t

struct st_ble_gap_scan_phy_param_t		
Scan parameters per scan PHY. In case of start scanning with both 1M PHY and Coded PHY, adjust scan windows and scan intervals according to the following. p_phy_param_1M->scan_window / p_phy_param_1M->scan_intv + p_phy_param_coded->scan_window / p_phy_param_coded->scan_intv <= 1		
Data Fields		
uint8_t	scan_type	Scan type. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_SCAN_PASSIVE(0x00)</td><td>Passive Scan.</td></tr><tr><td>BLE_GAP_SCAN_ACTIVE(0x01)</td><td>Active Scan.</td></tr></table>
uint16_t	scan_intv	Scan interval. interval(ms) = scan_intv * 0.625. Valid range is 0x0000 and 0x0004 - 0xFFFF.
uint16_t	scan_window	Scan window. window(ms) = scan_window * 0.625. Valid range is 0x0000 and 0x0004 - 0xFFFF.

◆ st_ble_gap_ext_scan_param_t

struct st_ble_gap_ext_scan_param_t		
Scan parameters.		
Data Fields		
uint8_t	o_addr_type	Own BD Address Type. In case of passive scan, this field is ignored. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Public Address</td></tr><tr><td>BLE_GAP_ADDR_RAND(0x01)</td><td>Random Address</td></tr></table>

		BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)	Resolvable Private Address. If the IRK of local device has not been registered in Resolving List, public address is used.
		BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)	Resolvable Private Address. If the IRK of local device has not been registered in Resolving List, the random address set by R_BLE_GAP_SetRandAddr() is used.
uint8_t	filter_policy	Scan Filter Policy.	
		macro	description
		BLE_GAP_SCAN_ALLOW_ADV_ALL(0x00)	Accept all advertising and scan response PDUs except directed advertising PDUs not addressed to local device.
		BLE_GAP_SCAN_ALLOW_ADV_WLST(0x01)	Accept only advertising and scan response PDUs from remote devices whose address is registered in the White List. Directed advertising PDUs which are not addressed to

			local device is ignored.
		BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED(0x02)	Accept all advertising and scan response PDUs except directed advertising PDUs whose the target address is identity address but doesn't address local device. However directed advertising PDUs whose the target address is the local resolvable private address are accepted.
		BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST(0x03)	Accept all advertising and scan response PDUs. The following are excluded. • Advertising and scan response PDUs where the advertiser's identity address is not in the White

		List. • Directed advertising PDUs whose the target address is identity address but doesn't address local device. However directed advertising PDUs whose the target address is the local resolvable private address are accepted.
st_ble_gap_scan_phy_param_t *	p_phy_param_1M	Scan parameters 1M PHY. When this field is NULL, Controller doesn't set the scan parameters for 1M PHY.
st_ble_gap_scan_phy_param_t *	p_phy_param_coded	Scan parameters Coded PHY. When this field is NULL, Controller doesn't set the scan parameters for Coded PHY.

◆ **st_ble_gap_scan_on_t**

struct st_ble_gap_scan_on_t

Parameters configured when scanning starts.

Data Fields

Type	Field	Description
uint8_t	proc_type	Procedure type. macro / description: BLE_GAP_SC_PROC_OBS(0x00) — Observation Procedure. Notify all advertising PDUs. BLE_GAP_SC_PROC_LIM(0x01) — Limited Discovery Procedure. Notify advertising PDUs from only devices in the limited discoverable mode. BLE_GAP_SC_PROC_GEN(0x02) — General Discovery Procedure. Notify advertising PDUs from devices in the limited discoverable mode and the general discoverable mode.
uint8_t	filter_dups	Filter duplicates. macro / description: BLE_GAP_SCAN_FILT_DUPLIC_DISABLE(0x00) — Duplicate filter disabled. BLE_GAP_SCAN_FILT_DUPLIC_ENABLE(0x01) — Duplicate filter enabled. BLE_GAP_SCAN_FILT_DUPLIC_ENABLE_FOR_PERIOD(0x02) — Duplicate filtering enabled, reset for each scan period

uint16_t	duration	Scan duration. Time(ms) = duration * 10. Valid range is 0x0000 - 0xFFFF. If this field is set to 0x0000, scanning is continued until R_BLE_GAP_StopScan() is called. When the period field is zero and the time specified the duration field expires, BLE_GAP_EVENT_SCAN_TO event notifies the application layer that scanning stops.
uint16_t	period	Scan period. Time(s) = N * 1.28. Valid range is 0x0000 - 0xFFFF. If the duration field is set to 0x0000, this field is ignored.

◆ **st_ble_gap_conn_param_t**

struct st_ble_gap_conn_param_t

Connection parameters included in connection interval, slave latency, supervision timeout, ce length.

This structure is used in R_BLE_GAP_CreateConn() and R_BLE_GAP_UpdConn().

Set the fields in this structure to match the following condition.

Supervision_timeout(ms) >= (1 + conn_latency) * conn_intv_max_Time(ms)

conn_intv_max_Time(ms) = conn_intv_max * 1.25 Supervision_timeout(ms) = sup_to * 10

Data Fields		
uint16_t	conn_intv_min	Minimum connection interval. Time(ms) = conn_intv_min * 1.25. Valid range is 0x0006 - 0x0C80.
uint16_t	conn_intv_max	Maximum connection interval. Time(ms) = conn_intv_max * 1.25. Valid range is 0x0006 - 0x0C80.
uint16_t	conn_latency	Slave latency. Valid range is 0x0000 - 0x01F3.
uint16_t	sup_to	Supervision timeout.

		Time(ms) = sup_to * 10. Valid range is 0x000A - 0x0C80.
uint16_t	min_ce_length	Minimum CE Length. Valid range is 0x0000 - 0xFFFF.
uint16_t	max_ce_length	Maximum CE Length. Valid range is 0x0000 - 0xFFFF.

◆ **st_ble_gap_conn_phy_param_t**

struct st_ble_gap_conn_phy_param_t		
Connection parameters per PHY.		
Data Fields		
uint16_t	scan_intv	Scan interval. Time(ms) = scan_intv * 0.625. Valid range is 0x0004 - 0xFFFF.
uint16_t	scan_window	Scan window. Time(ms) = scan_window * 0.625. Valid range is 0x0004 - 0xFFFF.
st_ble_gap_conn_param_t *	p_conn_param	Connection interval, slave latency, supervision timeout, and CE length.

◆ **st_ble_gap_create_conn_param_t**

struct st_ble_gap_create_conn_param_t		
Connection parameters used in R_BLE_GAP_CreateConn().		
Data Fields		
uint8_t	init_filter_policy	This field specifies whether the White List is used or not, when connecting with a remote device. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_INIT_FILT_USE_ADDR(0x00)</td><td>White List is not used. The remote device to be connected is specified by the *remote_bd_addr* field and the</td></tr></table>

<table>
<tr><td></td><td></td><td><table><tr><td></td><td>remote_bd_addr_type field is used.</td></tr><tr><td>BLE_GAP_INIT_FILT_USE_WLST(0x01)</td><td>White List is used.
The remote device registered in White List is connected with local device.
The remote_bd_addr field and the remote_bd_addr_type field are ignored.</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>remote_bd_addr[BLE_BD_ADDR_LEN]</td><td>Address of the device to be connected.

Note
The BD address setting format is little endian.
If the address is "AA:BB:CC:DD:EE:FF", set the byte array in the order {0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA}.</td></tr>
<tr><td>uint8_t</td><td>remote_bd_addr_type</td><td>Address type of the device to be connected.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Public Address or Public Identity Address</td></tr><tr><td>BLE_GAP_ADDR_RAND(0x01)</td><td>Random Address or Random (Static) Identity Address</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>own_addr_type</td><td>Address type which local device uses in creating a link with the remote device.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x0</td><td>Public Address</td></tr></table></td></tr>
</table>

		0) BLE_GAP_ADDR_RAND(0x01) Random Address BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02) Resolvable Private Address. If the IRK of local device has not been registered in Resolving List, public address is used. BLE_GAP_ADDR_RPA_ID_RANDOM(0x03) Resolvable Private Address. If the IRK of local device has not been registered in Resolving List, the random address set by R_BLE_GAP_SetRandAddr().
st_ble_gap_conn_phy_param_t *	p_conn_param_1M	Connection parameters for 1M PHY. If this field is set to NULL, 1M PHY is not used in connecting.
st_ble_gap_conn_phy_param_t *	p_conn_param_2M	Connection parameters for 2M PHY. If this field is set to NULL, 2M PHY is not used in connecting.
st_ble_gap_conn_phy_param_t *	p_conn_param_coded	Connection parameters for Coded PHY. If this field is set to NULL, Coded PHY is not used in connecting.

◆ st_ble_gap_rslv_list_key_set_t

struct st_ble_gap_rslv_list_key_set_t
IRK of a remote device and IRK type of local device used in R_BLE_GAP_ConfRslvList().
Data Fields

uint8_t	remote_irk[BLE_GAP_IRK_SIZE]	IRK of a remote device to be registered in the Resolving List.
uint8_t	local_irk_type	IRK type of the local device to be registered in the Resolving List. macro: BLE_GAP_RL_LOC_KEY_ALL_ZERO(0x00) — description: All-zero IRK. macro: BLE_GAP_RL_LOC_KEY_REGISTERED(0x01) — description: The IRK registered by R_BLE_GAP_SetLocIdInfo().

◆ st_ble_gap_set_phy_param_t

struct st_ble_gap_set_phy_param_t		
PHY configuration parameters used in R_BLE_GAP_SetPhy().		
Data Fields		
uint8_t	tx_phys	Transmitter PHY preference. The tx_phys field is set to a bitwise OR of the following values. All other values are ignored. macro: BLE_GAP_SET_PHYS_HOST_PREF_1M(0x01) — description: Use 1M PHY for Transmitter PHY. macro: BLE_GAP_SET_PHYS_HOST_PREF_2M(0x02) — description: Use 2M PHY for Transmitter PHY. macro: BLE_GAP_SET_PHYS_HOST_PREF_CD(0x04) — description: Use Coded PHY for Transmitter PHY.
uint8_t	rx_phys	Receiver PHY preference. The rx_phys field is set to a bitwise OR of the following values. All other values are ignored. macro — description

		<table><tr><td>BLE_GAP_SET_PHYS_HOST_PREF_1M(0x01)</td><td>Use 1M PHY for Receiver PHY.</td></tr><tr><td>BLE_GAP_SET_PHYS_HOST_PREF_2M(0x02)</td><td>Use 2M PHY for Receiver PHY.</td></tr><tr><td>BLE_GAP_SET_PHYS_HOST_PREF_CD(0x04)</td><td>Use Coded PHY for Receiver PHY.</td></tr></table>
uint16_t	phy_options	Coding scheme used in Coded PHY. Select one of the following. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_SET_PHYS_OP_HOST_NO_PREF(0x00)</td><td>No preferred coding.</td></tr><tr><td>BLE_GAP_SET_PHYS_OP_HOST_PREF_S_2(0x01)</td><td>Use S=2 coding.</td></tr><tr><td>BLE_GAP_SET_PHYS_OP_HOST_PREF_S_8(0x02)</td><td>Use S=8 coding.</td></tr></table>

◆ st_ble_gap_set_def_phy_param_t

struct st_ble_gap_set_def_phy_param_t		
PHY preferences which allows a remote device to set used in R_BLE_GAP_SetDefPhy().		
Data Fields		
uint8_t	tx_phys	Transmitter PHY preferences which a remote device may change. The tx_phys field is set to a bitwise OR of the following values. All other values are ignored. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_SET_PHYS_HOST_</td><td>Allow a remote device</td></tr></table>

<table>
<tr><td></td><td></td><td><table>
<tr><td>PREF_1M(0x01)</td><td>to set 1M PHY for transmitter PHY.</td></tr>
<tr><td>BLE_GAP_SET_PHYS_HOST_PREF_2M(0x02)</td><td>Allow a remote device to set 2M PHY for transmitter PHY.</td></tr>
<tr><td>BLE_GAP_SET_PHYS_HOST_PREF_CD(0x04)</td><td>Allow a remote device to set Coded PHY for transmitter PHY.</td></tr>
</table></td></tr>
<tr><td>uint8_t</td><td>rx_phys</td><td>Receiver PHY preferences which a remote device may change.

The rx_phys field is set to a bitwise OR of the following values. All other values are ignored.
<table>
<tr><th>macro</th><th>description</th></tr>
<tr><td>BLE_GAP_SET_PHYS_HOST_PREF_1M(0x01)</td><td>Allow a remote device to set 1M PHY for receiver PHY.</td></tr>
<tr><td>BLE_GAP_SET_PHYS_HOST_PREF_2M(0x02)</td><td>Allow a remote device to set 2M PHY for receiver PHY.</td></tr>
<tr><td>BLE_GAP_SET_PHYS_HOST_PREF_CD(0x04)</td><td>Allow a remote device to set Coded PHY for receiver PHY.</td></tr>
</table></td></tr>
</table>

◆ st_ble_gap_auth_info_t

<table>
<tr><th colspan="3">struct st_ble_gap_auth_info_t</th></tr>
<tr><td colspan="3">Pairing parameters required from a remote device or information about keys distributed from a remote device.</td></tr>
<tr><td colspan="3">Data Fields</td></tr>
<tr><td>uint8_t</td><td>security</td><td>Security level.</td></tr>
</table>

		<table><tr><th>value</th><th>description</th></tr><tr><td>0x01</td><td>The remote device requests Unauthenticated pairing.</td></tr><tr><td>0x02</td><td>The remote device requests Authenticated pairing.</td></tr></table>
uint8_t	pair_mode	Pairing mode.<table><tr><th>value</th><th>description</th></tr><tr><td>0x01</td><td>The remote device requests Legacy pairing.</td></tr><tr><td>0x02</td><td>The remote device requests Secure Connections.</td></tr></table>
uint8_t	bonding	Bonding policy.<table><tr><th>value</th><th>description</th></tr><tr><td>0x00</td><td>The remote device does not store the Bonding information.</td></tr><tr><td>0x01</td><td>The remote device stores the Bonding information.</td></tr></table>
uint8_t	ekey_size	Encryption key size.

◆ st_ble_gap_key_dist_t

struct st_ble_gap_key_dist_t		
Keys distributed from a remote device.		
Data Fields		
uint8_t	enc_info[BLE_GAP_LTK_SIZE]	LTK.
uint8_t	mid_info[BLE_GAP_EDIV_SIZE +BLE_GAP_RAND_64_BIT_SIZE]	Ediv and rand. The first two bytes is ediv, the remaining bytes are rand.

uint8_t	id_info[BLE_GAP_IRK_SIZE]	IRK.
uint8_t	id_addr_info[BLE_GAP_ID_ADDR_SIZE]	Identity address. The first byte is address type. The remaining bytes are device address.
uint8_t	sign_info[BLE_GAP_CSRK_SIZE]	CSRK.

◆ st_ble_gap_key_ex_param_t

struct st_ble_gap_key_ex_param_t		
This structure includes the distributed keys and negotiated LTK size.		
Data Fields		
st_ble_gap_key_dist_t *	p_keys_info	Key information.
uint8_t	keys	Type of the distributed keys. This field is a bitwise OR of the following values. Bit Number / description: 0 — LTK and Master Identification. LTK is distributed in Secure Connections, even if the bit is 1. 1 — IRK and Identity Address Information. 2 — CSRK
uint8_t	ekey_size	The negotiated LTK size.

◆ st_ble_gap_pairing_param_t

struct st_ble_gap_pairing_param_t		
Pairing parameters used in R_BLE_GAP_SetPairingParams().		
Data Fields		
uint8_t	iocap	IO capabilities of local device. Select one of the following. macro / description: BLE_GAP_IOC — Output

		AP_DISPLAY_ONLY(0x00)	function : Local device has the ability to display a 6 digit decimal number. Input function : None
		BLE_GAP_IOCAP_DISPLAY_YESNO(0x01)	Output function : Output function : Local device has the ability to display a 6 digit decimal number. Input function : Local device has the ability to indicate 'yes' or 'no'
		BLE_GAP_IOCAP_KEYBOARD_ONLY(0x02)	Output function : None Input function : Local device has the ability to input the number '0' - '9'.
		BLE_GAP_IOCAP_NOINPUT_NOOUTPUT(0x03)	Output function : None Input function : None
		BLE_GAP_IOCAP_KEYBOARD_DISPLAY(0x04)	Output function : Output function : Local device has the ability to display a 6 digit decimal number. Input function : Local device has the ability

		to input the number '0' - '9'.
uint8_t	mitm	MITM protection policy. Select one of the following. macro | description BLE_GAP_SEC_MITM_BEST_EFFORT(0x00) | MITM Protection not required. BLE_GAP_SEC_MITM_STRICT (0x01) | MITM Protection required.
uint8_t	bonding	Bonding policy. macro | description BLE_GAP_BONDING_NONE(0x00) | Local device doesn't stores Bonding information. BLE_GAP_BONDING (0x01) | Local device stores Bonding information.
uint8_t	max_key_size	Maximum LTK size(in bytes). Valid range is 7 - 16. This field shall be set to a value not less than the min_key_size field.
uint8_t	min_key_size	Minimum LTK size(in bytes). Valid range is 7 - 16. This field shall be set to a value not more than the max_key_size field.
uint8_t	loc_key_dist	Type of keys to be distributed from local device. The loc_key_dist field is set to a bitwise OR of the following values. macro | description BLE_GAP_KEY_DIST_ENCKEY | LTK

		(0x01) BLE_GAP_KEY_DIST_IDKEY(0x02) — IRK and Identity Address. BLE_GAP_KEY_DIST_SIGNKEY(0x04) — CSRK
uint8_t	rem_key_dist	Type of keys which local device requests a remote device to distribute. The rem_key_dist field is set to a bitwise OR of the following values. **macro** — **description** BLE_GAP_KEY_DIST_ENCKEY(0x01) — LTK. In case of Secure Connections, LTK is notified even if this bit is not set. BLE_GAP_KEY_DIST_IDKEY(0x02) — IRK and Identity Address. BLE_GAP_KEY_DIST_SIGNKEY(0x04) — CSRK
uint8_t	key_notf	Support for Key Press Notification in Passkey Entry. **macro** — **description** BLE_GAP_SC_KEY_PRESS_NTF_NOT_SPRT(0x00) — Not support for Key Press Notification. BLE_GAP_SC_KEY_PRESS_NTF_SPRT(0x01) — Support for Key Press Notification.
uint8_t	sec_conn_only	Determine whether to accept only Secure Connections or not. **macro** — **description** BLE_GAP_SC_BEST_EFFORT(0x00) — Accept Legacy pairing and Secure

			Connections.
		BLE_GAP_SC_STRICT(0x01)	Accept only Secure Connections.

◆ **st_ble_gap_oob_data_t**

struct st_ble_gap_oob_data_t		
Oob data received from the remote device. This is used in R_BLE_GAP_SetRemOobData().		
Data Fields		
uint8_t	legacy_oob[BLE_GAP_LEGACY_OOB_SIZE]	OOB data used in Legacy Pairing.
uint8_t	sc_cnf_val[BLE_GAP_OOB_CONFIRM_VAL_SIZE]	OOB confirmation value used in Secure Connections.
uint8_t	sc_rand[BLE_GAP_OOB_RANDOM_VAL_SIZE]	OOB rand used in Secure Connections.

◆ **st_ble_gap_ver_num_t**

struct st_ble_gap_ver_num_t		
Version number of host stack.		
Data Fields		
uint8_t	major	Major version number.
uint8_t	minor	Minor version number.
uint8_t	subminor	Subminor version number.

◆ **st_ble_gap_loc_ver_info_t**

struct st_ble_gap_loc_ver_info_t		
Version number of Controller. Refer Bluetooth SIG Assigned Number(https://www.bluetooth.com/specifications/assigned-numbers).		
Data Fields		
uint8_t	hci_ver	Bluetooth HCI version.
uint16_t	hci_rev	Bluetooth HCI revision.
uint8_t	lmp_ver	Link Layer revision.
uint16_t	mnf_name	Manufacturer ID.
uint16_t	lmp_sub_ver	Link Layer subversion.

◆ **st_ble_gap_loc_dev_info_evt_t**

struct st_ble_gap_loc_dev_info_evt_t

Version information of local device.		
Data Fields		
st_ble_dev_addr_t	l_dev_addr	Bluetooth Device Address.
st_ble_gap_ver_num_t	l_ver_num	Version number of host stack in local device.
st_ble_gap_loc_ver_info_t	l_bt_info	Version number of Controller in local device.

◆ st_ble_gap_hw_err_evt_t

struct st_ble_gap_hw_err_evt_t		
Hardware error that is notified from Controller.		
Data Fields		
uint8_t	hw_code	The hw_code field indicates the cause of the hardware error.

◆ st_ble_gap_cmd_err_evt_t

struct st_ble_gap_cmd_err_evt_t		
HCI Command error.		
Data Fields		
uint16_t	op_code	The opcode of HCI Command which caused the error.
uint32_t	module_id	Module ID which caused the error.

◆ st_ble_gap_adv_rept_t

struct st_ble_gap_adv_rept_t		
Advertising Report.		
Data Fields		
uint8_t	num	The number of Advertising Reports received.
uint8_t	adv_type	Type of Advertising Packet. <table><tr><th>valuer</th><th>description</th></tr><tr><td>0x00</td><td>Connectable and scannable undirected advertising(ADV_IND).</td></tr><tr><td>0x01</td><td>Connectable directed advertising(ADV_DIRECT_IND).</td></tr></table>

		0x02 Scannable undirected advertising(ADV_SCAN_IND). 0x03 Non-connectable undirected advertising(ADV_NONCONN_IND). 0x04 Scan response(SCAN_RSP).
uint8_t	addr_type	Address type of the advertiser. value / description 0x00 Public Address. 0x01 Random Address. 0x02 Public Identity Address which could be resolved in Controller. 0x03 Random Identity Address which could be resolved in Controller.
uint8_t *	p_addr	Address of the advertiser. *Note* *The BD address setting format is little endian.*
uint8_t	len	Length of Advertising data(in bytes). Valid range is 0 - 31.
int8_t	rssi	RSSI(in dBm). Valid range is -127 <= tx_pwr <= 20 and 127. If the tx_pwr is 127, it means that RSSI could not be retrieved.
uint8_t *	p_data	Advertising data/Scan Response Data.

◆ st_ble_gap_ext_adv_rept_t

struct st_ble_gap_ext_adv_rept_t		
Extended Advertising Report.		
Data Fields		
uint8_t	num	The number of Advertising Reports received.
uint16_t	adv_type	Type of Advertising Packet. <table><tr><th>Bit Number</th><th>description</th></tr><tr><td>0</td><td>Connectable advertising.</td></tr><tr><td>1</td><td>Scannable advertising.</td></tr><tr><td>2</td><td>Directed advertising.</td></tr><tr><td>3</td><td>Scan response.</td></tr><tr><td>4</td><td>Legacy advertising PDU.</td></tr><tr><td>5-6</td><td>The status of Advertising Data/Scan Response Data. Data Status: 00b = Complete 01b = Incomplete, more data come 10b = Incomplete, data truncated, no more to come</td></tr><tr><td>All other bits</td><td>Reserved for future use</td></tr></table>
uint8_t	addr_type	Address type of the advertiser. <table><tr><th>value</th><th>description</th></tr><tr><td>0x00</td><td>Public Address.</td></tr><tr><td>0x01</td><td>Random</td></tr></table>

		Address. 0x02 — Public Identity Address which could be resolved in Controller. 0x03 — Random Identity Address which could be resolved in Controller. 0xFF — Anonymous advertisement.
uint8_t *	p_addr	Address of the advertiser. *Note* *The BD address setting format is little endian.*
uint8_t	adv_phy	The primary PHY configuration of the advertiser. The primary PHY configuration of the advertiser. value — description 0x01 — 1M PHY 0x03 — Coded PHY
uint8_t	sec_adv_phy	The secondary PHY configuration of the advertiser. value — description 0x00 — Nothing has been received with Secondary Advertising Channel. 0x01 — The Secondary Advertising PHY configuration was 1M PHY. 0x02 — The Secondary Advertising

		PHY configuration was 2M PHY. 0x03 — The Secondary Advertising PHY configuration was Coded PHY.
uint8_t	adv_sid	Advertising SID included in the received Advertising Report. Valid range is 0 <= adv_sid <= 0x0F and 0xFF. If the adv_sid is 0xFF, there is no field which includes SID.
int8_t	tx_pwr	TX power(in dBm). Valid range is -127 <= tx_pwr <= 20 and 127. If the tx_pwr is 127, it means that Tx power could not be retrieved.
int8_t	rssi	RSSI(in dBm). Valid range is -127 <= tx_pwr <= 20 and 127. If the tx_pwr is 127, it means that RSSI could not be retrieved.
uint16_t	perd_adv_intv	Periodic Advertising interval. If the perd_adv_intv is 0x0000, it means that this advertising is not periodic advertising. If the perd_adv_intv is 0x0006 - 0xFFFF, it means that this field is the Periodic Advertising interval. Periodic Advertising interval = per_adv_intr * 1.25ms.
uint8_t	dir_addr_type	The address type of Direct Advertising PDU. value — description 0x00 — Public Address. 0x01 — Random Address.

		0x02	Public Identity Address which could be resolved in Controller.
		0x03	Random Identity Address which could be resolved in Controller.
		0xFE	Resolvable Privacy Address which could not be resolved in Controller.
uint8_t *	p_dir_addr	Address of Direct Advertising PDU. *Note* *The BD address setting format is little endian.*	
uint8_t	len	Length of Advertising data(in bytes). Valid range is 0 - 229.	
uint8_t *	p_data	Advertising data/Scan Response Data.	

◆ st_ble_gap_perd_adv_rept_t

struct st_ble_gap_perd_adv_rept_t		
Periodic Advertising Report.		
Data Fields		
uint16_t	sync_hdl	Sync handle. Valid range is 0x0000 - 0x0EFF.
int8_t	tx_pwr	TX power(in dBm). Valid range is -127 <= tx_pwr <= 20 and 127. If tx_pwr is 127, it means that Tx power could not be retrieved.
int8_t	rssi	RSSI(in dBm). Valid range is -127 <= rssi <=

		20 and 127. If rssi is 127, it means that RSSI could not be retrieved.
uint8_t	rfu	Reserved for future use.
uint8_t	data_status	Reserved for future use. value / description 0x00: Data Complete. 0x01: Data incomplete, more data to come. 0x02: Data incomplete, data truncated, no more to come.
uint8_t	len	Length of Periodic Advertising data(in bytes). Valid range is 0 - 247.
uint8_t *	p_data	Periodic Advertising data.

◆ st_ble_gap_adv_rept_evt_t

struct st_ble_gap_adv_rept_evt_t		
Advertising report.		
Data Fields		
uint8_t	adv_rpt_type	Data type. value / description 0x00: Advertising Report. 0x01: Extended Advertising Report. 0x02: Periodic Advertising Report. If the BLE Protocol Stack library type is "extended", the adv_rpt_type field in a Legacy Advertising Report event is 0x01.

union st_ble_gap_adv_rept_evt_t	param	Advertising Report.

◆ st_ble_gap_adv_rept_evt_t.param

union st_ble_gap_adv_rept_evt_t.param		
Advertising Report.		
Data Fields		
st_ble_gap_adv_rept_t *	p_adv_rpt	Advertising Report.
st_ble_gap_ext_adv_rept_t *	p_ext_adv_rpt	Extended Advertising Report.
st_ble_gap_perd_adv_rept_t *	p_per_adv_rpt	Periodic Advertising Report.

◆ st_ble_gap_adv_set_evt_t

struct st_ble_gap_adv_set_evt_t		
Advertising handle.		
Data Fields		
uint8_t	adv_hdl	Advertising handle specifying the advertising set configured advertising parameters.

◆ st_ble_gap_adv_off_evt_t

struct st_ble_gap_adv_off_evt_t		
Information about the advertising set which stops advertising.		
Data Fields		
uint8_t	adv_hdl	Advertising handle identifying the advertising set which has stopped advertising. Valid range is 0x00 - 0x03.
uint8_t	reason	The reason for stopping advertising. <table><tr><th>value</th><th>description</th></tr><tr><td>0x01</td><td>Advertising has been stopped by R_BLE_GAP_StopAdv().</td></tr><tr><td>0x02</td><td>Because the duration specified by R_BLE_GAP_StartAdv() was expired, advertising</td></tr></table>

		has terminated. 0x03: Because the max_extd_adv_evts parameter specified by R_BLE_GAP_StartAdv() was reached, advertising has terminated. 0x04: Because the connection was established with the remote device, advertising has terminated.
uint16_t	conn_hdl	Connection handle. If the reason field is 0x04, this field indicates connection handle identifying the remote device connected with local device. If other reasons, ignore this field.
uint8_t	num_comp_ext_adv_evts	The number of the advertising event that has been received until advertising has terminated. If max_extd_adv_evts by R_BLE_GAP_StartAdv() is not 0, this parameter is valid.

◆ st_ble_gap_adv_data_evt_t

struct st_ble_gap_adv_data_evt_t		
This structure notifies that advertising data has been set to Controller by R_BLE_GAP_SetAdvSresData().		
Data Fields		
uint8_t	adv_hdl	Advertising handle identifying the advertising set to which advertising data/scan response data/periodic advertising data is set.

uint8_t	data_type	Type of the data set to the advertising set. value / description BLE_GAP_ADV_DATA_MODE(0x00) — Advertising data BLE_GAP_SCAN_RSP_DATA_MODE(0x01) — Scan response data BLE_GAP_PERD_ADV_DATA_MODE(0x02) — Periodic advertising data

◆ st_ble_gap_rem_adv_set_evt_t

struct st_ble_gap_rem_adv_set_evt_t		
This structure notifies that an advertising set has been removed.		
Data Fields		
uint8_t	remove_op	This field indicates that the advertising set has been removed or cleared. value / description 0x01 — The advertising set has been removed. 0x02 — The advertising set has been cleared.
uint8_t	adv_hdl	Advertising handle identifying the advertising set which has been removed. If the advertising set has been cleared, this field is ignored.

◆ st_ble_gap_conn_evt_t

struct st_ble_gap_conn_evt_t		
This structure notifies that a link has been established.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the created link.
uint8_t	role	The role of the link.

		<table><tr><th>value</th><th>description</th></tr><tr><td>0x00</td><td>Master</td></tr><tr><td>0x01</td><td>Slave</td></tr></table>
uint8_t	remote_addr_type	Address type of the remote device. <table><tr><th>value</th><th>description</th></tr><tr><td>0x00</td><td>Public Address</td></tr><tr><td>0x01</td><td>Random Address</td></tr><tr><td>0x02</td><td>Public Identity Address. It indicates that the Controller could resolve the resolvable private address of the remote device.</td></tr><tr><td>0x03</td><td>Random Identity Address. It indicates that the Controller could resolve the resolvable private address of the remote device.</td></tr></table>
uint8_t	remote_addr[BLE_BD_ADDR_LEN]	Address of the remote device. *Note* *The BD address setting format is little endian.*
uint8_t	local_rpa[BLE_BD_ADDR_LEN]	Resolvable private address that local device used in connection procedure. The local device address used in creating the link when the address type was set to BLE_GAP_ADDR_RPA_ID_PUBLIC or BLE_GAP_ADDR_RPA_ID_RANDOM by

		R_BLE_GAP_SetAdvParam() or R_BLE_GAP_CreateConn(). If the address type was set to other than BLE_GAP_ADDR_RPA_ID_PUBLIC and BLE_GAP_ADDR_RPA_ID_RANDOM, this field is set to all-zero. *Note* *The BD address setting format is little endian.*
uint8_t	remote_rpa[BLE_BD_ADDR_LEN]	Resolvable private address that the remote device used in connection procedure. This field indicates the remote resolvable private address when remote_addr_type is 0x02 or 0x03. If remote_addr_type is other than 0x02 and 0x03, this field is set to all-zero. *Note* *The BD address setting format is little endian.*
uint16_t	conn_intv	Connection interval. Valid range is 0x0006 - 0x0C80. Time(ms) = conn_intv * 1.25.
uint16_t	conn_latency	Slave latency. Valid range is 0x0000 - 0x01F3.
uint16_t	sup_to	Supervision timeout. Valid range is 0x000A - 0x0C80.Time(ms) = sup_to * 10.
uint8_t	clk_acc	Master_Clock_Accuracy. value / description: 0x00 500ppm 0x01 250ppm 0x02 150ppm 0x03 100ppm 0x04 75ppm 0x05 50ppm

		0x06	30ppm
		0x07	20ppm

◆ st_ble_gap_disconn_evt_t

struct st_ble_gap_disconn_evt_t		
This structure notifies that a link has been disconnected.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the link disconnected.
uint8_t	reason	The reason for disconnection. Refer Core Specification Vol.2 Part D ,"2 Error Code Descriptions".

◆ st_ble_gap_rd_ch_map_evt_t

struct st_ble_gap_rd_ch_map_evt_t		
This structure notifies that Channel Map has been retrieved by R_BLE_GAP_ReadChMap().		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the link whose Channel Map was retrieved.
uint8_t	ch_map[BLE_GAP_CH_MAP_SIZE]	Channel Map.

◆ st_ble_gap_rd_rssi_evt_t

struct st_ble_gap_rd_rssi_evt_t		
This structure notifies that RSSI has been retrieved by R_BLE_GAP_ReadRssi().		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the link whose RSSI was retrieved.
int8_t	rssi	RSSI(in dBm). Valid range is -127 < rssi < 20 and 127. If this field is 127, it indicates that RSSI could not be retrieved.

◆ st_ble_gap_dev_info_evt_t

struct st_ble_gap_dev_info_evt_t

This structure notifies that information about remote device has been retrieved by R_BLE_GAP_GetRemDevInfo().

Data Fields		
uint16_t	conn_hdl	Connection handle identifying the remote device whose information has been retrieved.
uint8_t	get_status	Information about the remote device. This field is a bitwise OR of the following values. Bit Number — description bit0 — Address bit1 — Version, company_id, subversion bit2 — Feature All other bits — Reserved for future use
st_ble_dev_addr_t	addr	Address of the remote device.
uint8_t	version	The version of Link Layer of the remote device. Refer to Bluetooth SIG Assigned Number (https://www.bluetooth.com/specifications/assigned-numbers) regarding defined number.
uint16_t	company_id	The manufacturer ID of the remote device. Refer to Bluetooth SIG Assigned Number (https://www.bluetooth.com/specifications/assigned-numbers) regarding defined number.
uint16_t	subversion	The subversion of Link Layer.
uint8_t	features[BLE_GAP_REM_FEATURE_SIZE]	LE feature supported in the remote device. Refer to Core Spec Vol 6, Part B 4.6 FEATURE SUPPORT.

◆ st_ble_gap_conn_upd_evt_t

```
struct st_ble_gap_conn_upd_evt_t
```

This structure notifies that connection parameters has been updated.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the connection whose parameters has been updated.
uint16_t	conn_intv	Updated Connection Interval. Valid range is 0x0006 - 0x0C80. Time(ms) = conn_intv * 1.25.
uint16_t	conn_latency	Updated slave latency. Valid range is 0x0000 - 0x01F3.
uint16_t	sup_to	Updated supervision timeout. Valid range is 0x000A - 0x0C80. Time(ms) = sup_to * 10.

◆ st_ble_gap_conn_upd_req_evt_t

struct st_ble_gap_conn_upd_req_evt_t		
This structure notifies that a request for connection parameters update has been received.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the link that was requested to update connection parameters.
uint16_t	conn_intv_min	Minimum connection interval. Valid range is 0x0006 - 0x0C80. Time(ms) = conn_intv_min * 1.25.
uint16_t	conn_intv_max	Maximum connection interval. Valid range is 0x0006 - 0x0C80. Time(ms) = conn_intv_max * 1.25.
uint16_t	conn_latency	Slave latency. Valid range is 0x0000 - 0x01F3.
uint16_t	sup_to	Supervision timeout. Valid range is 0x000A - 0x0C80. Time(ms) = sup_to * 10

◆ st_ble_gap_conn_hdl_evt_t

struct st_ble_gap_conn_hdl_evt_t

This structure notifies that a GAP Event that includes only connection handle has occurred.		
Data Fields		
uint16_t	conn_hdl	Connection handle.

◆ st_ble_gap_data_len_chg_evt_t

struct st_ble_gap_data_len_chg_evt_t		
This structure notifies that the packet data length has been updated.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the link that updated Data Length.
uint16_t	tx_octets	Updated transmission packet size(in bytes). Valid range is 0x001B - 0x00FB.
uint16_t	tx_time	Updated transmission time(us). Valid range is 0x0148 - 0x4290.
uint16_t	rx_octets	Updated receive packet size(in bytes). Valid range is 0x001B - 0x00FB.
uint16_t	rx_time	Updated receive time(us). Valid range is 0x0148 - 0x4290.

◆ st_ble_gap_rd_rpa_evt_t

struct st_ble_gap_rd_rpa_evt_t		
This structure notifies that the local resolvable private address has been retrieved by R_BLE_GAP_ReadRpa().		
Data Fields		
st_ble_dev_addr_t	addr	The resolvable private address of local device.

◆ st_ble_gap_phy_upd_evt_t

struct st_ble_gap_phy_upd_evt_t		
This structure notifies that PHY for a connection has been updated.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying

		the link that has been updated.
uint8_t	tx_phy	Transmitter PHY. <table><tr><th>value</th><th>description</th></tr><tr><td>0x01</td><td>The transmitter PHY has been updated to 1M PHY.</td></tr><tr><td>0x02</td><td>The transmitter PHY has been updated to 2M PHY.</td></tr><tr><td>0x03</td><td>The transmitter PHY has been updated to Coded PHY.</td></tr></table>
uint8_t	rx_phy	Receiver PHY. <table><tr><th>value</th><th>description</th></tr><tr><td>0x01</td><td>The receiver PHY has been updated to 1M PHY.</td></tr><tr><td>0x02</td><td>The receiver PHY has been updated to 2M PHY.</td></tr><tr><td>0x03</td><td>The receiver PHY has been updated to Coded PHY.</td></tr></table>

◆ st_ble_gap_phy_rd_evt_t

struct st_ble_gap_phy_rd_evt_t		
This structure notifies that the PHY settings has been retrieved by R_BLE_GAP_ReadPhy().		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the link that has been retrieved the PHY settings.
uint8_t	tx_phy	Transmitter PHY. <table><tr><th>value</th><th>description</th></tr><tr><td>0x01</td><td>The transmitter</td></tr></table>

		PHY has been updated to 1M PHY. 0x02 — The transmitter PHY has been updated to 2M PHY. 0x03 — The transmitter PHY has been updated to Coded PHY.
uint8_t	rx_phy	Receiver PHY. value — description 0x01 — The receiver PHY has been updated to 1M PHY. 0x02 — The receiver PHY has been updated to 2M PHY. 0x03 — The receiver PHY has been updated to Coded PHY.

◆ st_ble_gap_scan_req_recv_evt_t

struct st_ble_gap_scan_req_recv_evt_t		
This structure notifies that a Scan Request packet has been received from a Scanner.		
Data Fields		
uint8_t	adv_hdl	Advertising handle identifying the advertising set that has received the Scan Request.
uint8_t	scanner_addr_type	Address type of the Scanner. value — description 0x00 — Public Address. 0x01 — Random Address. 0x02 — Public Identity Address which could be resolved in

		Controller. 0x03 Random Identity Address which could be resolved in Controller.
uint8_t	scanner_addr[BLE_BD_ADDR_LEN]	Address of the Scanner. *Note* *The BD address setting format is little endian.*

◆ st_ble_gap_sync_est_evt_t

struct st_ble_gap_sync_est_evt_t		
This structure notifies that a Periodic sync has been established.		
Data Fields		
uint16_t	sync_hdl	Sync handle identifying the Periodic Sync that has been established.
uint8_t	adv_sid	Advertising SID identifying the advertising set that has established the Periodic Sync.
uint8_t	adv_addr_type	Address type of the advertiser. value / description 0x00 Public Address. 0x01 Random Address. 0x02 Public Identity Address which could be resolved in Controller. 0x03 Random Identity Address which could be resolved in Controller.
uint8_t *	p_adv_addr	Address of the advertiser. *Note* *The BD address setting format is little endian.*

uint8_t	adv_phy	Advertising PHY. <table><tr><th>value</th><th>description</th></tr><tr><td>0x01</td><td>Advertiser PHY is 1M PHY.</td></tr><tr><td>0x02</td><td>Advertiser PHY is 2M PHY.</td></tr><tr><td>0x03</td><td>Advertiser PHY is Coded PHY.</td></tr></table>
uint16_t	perd_adv_intv	Periodic Advertising Interval. Valid range is 0x0006 - 0xFFFF. Time(ms) = perd_adv_intv * 1.25.
uint8_t	adv_clk_acc	Advertiser Clock Accuracy. <table><tr><th>value</th><th>description</th></tr><tr><td>0x00</td><td>500ppm</td></tr><tr><td>0x01</td><td>250ppm</td></tr><tr><td>0x02</td><td>150ppm</td></tr><tr><td>0x03</td><td>100ppm</td></tr><tr><td>0x04</td><td>75ppm</td></tr><tr><td>0x05</td><td>50ppm</td></tr><tr><td>0x06</td><td>30ppm</td></tr><tr><td>0x07</td><td>20ppm</td></tr></table>

◆ st_ble_gap_sync_hdl_evt_t

struct st_ble_gap_sync_hdl_evt_t		
This structure notifies that a GAP Event that includes only sync handle has occurred.		
Data Fields		
uint16_t	sync_hdl	Sync handle.

◆ st_ble_gap_white_list_conf_evt_t

struct st_ble_gap_white_list_conf_evt_t		
This structure notifies that White List has been configured.		
Data Fields		
uint8_t	op_code	The operation for White List. <table><tr><th>value</th><th>description</th></tr></table>

		0x01	A device was added to White List.
		0x02	A device was deleted from White List.
		0x03	White List was cleared.
uint8_t	num	The number or devices which have been added to or deleted from White List.	

◆ **st_ble_gap_rslv_list_conf_evt_t**

struct st_ble_gap_rslv_list_conf_evt_t		
This structure notifies that Resolving List has been configured.		
Data Fields		
uint8_t	op_code	The operation for Resolving List. value / description 0x01 — A device was added to Resolving List. 0x02 — A device was deleted from Resolving List. 0x03 — Resolving List was cleared.
uint8_t	num	The number or devices which have been added to or deleted from Resolving List.

◆ **st_ble_gap_perd_list_conf_evt_t**

struct st_ble_gap_perd_list_conf_evt_t		
This structure notifies that Periodic Advertiser List has been configured.		
Data Fields		
uint8_t	op_code	The operation for Periodic Advertiser List. value / description 0x01 — A device was added to Periodic Advertiser List.

		0x02 A device was deleted from Periodic Advertiser List. 0x03 Periodic Advertiser List was cleared.
uint8_t	num	The number or devices which have been added to or deleted from Periodic Advertiser List.

◆ st_ble_gap_set_priv_mode_evt_t

struct st_ble_gap_set_priv_mode_evt_t		
This structure notifies that Privacy Mode has been configured.		
Data Fields		
uint8_t	num	The number or devices which have been set privacy mode.

◆ st_ble_gap_pairing_req_evt_t

struct st_ble_gap_pairing_req_evt_t		
This structure notifies that a pairing request from a remote device has been received.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the remote device that sent the pairing request.
st_ble_dev_addr_t	bd_addr	The address of the remote device.
st_ble_gap_auth_info_t	auth_info	The Pairing parameters of the remote device.

◆ st_ble_gap_passkey_display_evt_t

struct st_ble_gap_passkey_display_evt_t		
This structure notifies that a request for Passkey display in pairing has been received.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the remote device that requested Passkey display.
uint32_t	passkey	Passkey. This field is a 6 digit decimal number(000000-999999).

◆ st_ble_gap_num_comp_evt_t

struct st_ble_gap_num_comp_evt_t		
This structure notifies that a request for Numeric Comparison in pairing has been received.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the remote device that requested Numeric Comparison.
uint32_t	numeric	The number to be confirmed in Numeric Comparison. This field is a 6 digit decimal number(000000-999999).

◆ st_ble_gap_key_press_ntf_evt_t

struct st_ble_gap_key_press_ntf_evt_t		
This structure notifies that the remote device has input a key in Passkey Entry.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the remote device that input a key.
uint8_t	key_type	Type of the key that the remote device input. value / description 0x00: Passkey entry started. 0x01: Passkey digit entered. 0x02: Passkey digit erased. 0x03: Passkey cleared. 0x04: Passkey entry completed.

◆ st_ble_gap_pairing_info_evt_t

struct st_ble_gap_pairing_info_evt_t		
This structure notifies that the pairing has completed.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the remote device that the pairing has been done with.
st_ble_dev_addr_t	bd_addr	Address of the remote device.

st_ble_gap_auth_info_t	auth_info	Key information exchanged in pairing. If local device supports bonding, store the information in non-volatile memory in order to set it to host stack after power re-supply.

◆ st_ble_gap_enc_chg_evt_t

struct st_ble_gap_enc_chg_evt_t		
This structure notifies that the encryption status of a link has been changed.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the link that has been changed.
uint8_t	enc_status	Encryption Status. value / description 0x00: Encryption OFF. 0x01: Encryption ON. 0x03: Encryption updated by Encryption Key Refresh Completed.

◆ st_ble_gap_peer_key_info_evt_t

struct st_ble_gap_peer_key_info_evt_t		
This structure notifies that the remote device has distributed the keys.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the remote device that has distributed the keys.
st_ble_dev_addr_t	bd_addr	Address of the remote device.
st_ble_gap_key_ex_param_t	key_ex_param	Distributed keys. If local device supports bonding, store the keys in non-volatile memory and at power re-supply set to the host stack by R_BLE_GAP_SetBondInfo().

◆ st_ble_gap_ltk_req_evt_t

struct st_ble_gap_ltk_req_evt_t		
This structure notifies that a LTK request from a remote device has been received.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the remote device which requests for the LTK.
uint16_t	ediv	Ediv.
uint8_t *	p_peer_rand	Rand.

◆ **st_ble_gap_ltk_rsp_evt_t**

struct st_ble_gap_ltk_rsp_evt_t		
This structure notifies that local device has replied to the LTK request from the remote device.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the remote device to be sent the response to the LTK request.
uint8_t	response	The response to the LTK request. value / description: 0x00 — Local device replied with the stored LTK. 0x01 — Local device rejected the LTK request, because the LTK was not found.

◆ **st_ble_gap_sc_oob_data_evt_t**

struct st_ble_gap_sc_oob_data_evt_t		
This structure notifies that OOB data for Secure Connections has been generated by R_BLE_GAP_CreateScOobData().		
Data Fields		
uint8_t *	p_sc_oob_conf	Confirmation value(16 bytes) of OOB Data.
uint8_t *	p_sc_oob_rand	Rand(16bytes) of OOB Data.

◆ **st_ble_gap_bond_info_t**

struct st_ble_gap_bond_info_t

Bonding information used in R_BLE_GAP_SetBondInfo().		
Data Fields		
st_ble_dev_addr_t *	p_addr	Address of the device which exchanged the keys.
st_ble_gap_auth_info_t *	p_auth_info	Information about the keys.
st_ble_gap_key_ex_param_t *	p_keys	Keys distributed from the remote device in paring.

Macro Definition Documentation

◆ BLE_BD_ADDR_LEN

```
#define BLE_BD_ADDR_LEN
```

Bluetooth Device Address Size

◆ BLE_MASTER

```
#define BLE_MASTER
```

Master Role.

◆ BLE_SLAVE

```
#define BLE_SLAVE
```

Slave Role.

◆ BLE_GAP_ADDR_PUBLIC

```
#define BLE_GAP_ADDR_PUBLIC
```

Public Address.

◆ BLE_GAP_ADDR_RAND

```
#define BLE_GAP_ADDR_RAND
```

Random Address.

◆ BLE_GAP_ADDR_RPA_ID_PUBLIC

```
#define BLE_GAP_ADDR_RPA_ID_PUBLIC
```

Resolvable Private Address.

If the IRK of local device has not been registered in Resolving List, public address is used.

◆ BLE_GAP_ADDR_RPA_ID_RANDOM

```
#define BLE_GAP_ADDR_RPA_ID_RANDOM
```

Resolvable Private Address.

If the IRK of local device has not been registered in Resolving List, random address is used.

◆ BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST

```
#define BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST
```

Accept all advertising and scan response PDUs.
The following are excluded.

- Advertising and scan response PDUs where the advertiser's identity address is not in the White List.
- Directed advertising PDUs whose the target address is identity address but doesn't address local device. However directed advertising PDUs whose the target address is the local resolvable private address are accepted.

◆ BLE_GAP_IOCAP_DISPLAY_ONLY

```
#define BLE_GAP_IOCAP_DISPLAY_ONLY
```

Display Only iocapability.

Output function : Local device has the ability to display a 6 digit decimal number.
Input function : None

◆ BLE_GAP_IOCAP_DISPLAY_YESNO

```
#define BLE_GAP_IOCAP_DISPLAY_YESNO
```

Display Yes/No iocapability.

Output function : Output function : Local device has the ability to display a 6 digit decimal number.
Input function : Local device has the ability to indicate 'yes' or 'no'

◆ BLE_GAP_IOCAP_KEYBOARD_ONLY

```
#define BLE_GAP_IOCAP_KEYBOARD_ONLY
```

Keyboard Only iocapability.

Output function : None
Input function : Local device has the ability to input the number '0' - '9'.

◆ BLE_GAP_IOCAP_NOINPUT_NOOUTPUT

```
#define BLE_GAP_IOCAP_NOINPUT_NOOUTPUT
```

No Input No Output iocapability.

Output function : None
Input function : None

◆ BLE_GAP_IOCAP_KEYBOARD_DISPLAY

```
#define BLE_GAP_IOCAP_KEYBOARD_DISPLAY
```

Keyboard Display iocapability.

Output function : Output function : Local device has the ability to display a 6 digit decimal number.
Input function : Local device has the ability to input the number '0' - '9'.

Typedef Documentation

◆ ble_gap_app_cb_t

```
ble_gap_app_cb_t
```

ble_gap_app_cb_t is the GAP Event callback function type.

Parameters

[in]	event_type	The type of GAP Event.
[in]	event_result	The result of API call which generates the GAP Event.
[in]	p_event_data	Data notified in the GAP Event.

Returns
none

◆ ble_gap_del_bond_cb_t

ble_gap_del_bond_cb_t
ble_gap_del_bond_cb_t is the type of the callback function for delete bonding information stored in non-volatile area. This type is used in R_BLE_GAP_DeleteBondInfo(). **Parameters** [in] p_addr The parameter returns the address of the remote device whose keys are deleted by R_BLE_GAP_DeleteBondInfo() . If R_BLE_GAP_DeleteBondInfo() deletes the keys of all remote devices, the parameter returns NULL. **Returns** none

◆ st_ble_gap_adv_param_t

typedef st_ble_gap_ext_adv_param_t st_ble_gap_adv_param_t
Advertising parameters. **See also** st_ble_gap_ext_adv_param_t

◆ st_ble_gap_scan_param_t

typedef st_ble_gap_ext_scan_param_t st_ble_gap_scan_param_t
Scan parameters. **See also** st_ble_gap_ext_scan_param_t

Enumeration Type Documentation

◆ e_ble_gap_evt_t

enum e_ble_gap_evt_t	
GAP Event Identifier.	
Enumerator	
BLE_GAP_EVENT_INVALID	Invalid GAP Event. **Event Code: 0x1001** **Event Data:** none
BLE_GAP_EVENT_STACK_ON	Host Stack has been initialized. When initializing host stack by R_BLE_GAP_Init() has been completed, BLE_GAP_EVENT_STACK_ON event is notified. **Event Code: 0x1002** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** none
BLE_GAP_EVENT_STACK_OFF	Host Stack has been terminated. When terminating host stack by R_BLE_GAP_Terminate() has been completed, BLE_GAP_EVENT_STACK_OFF event is notified. **Event Code: 0x1003** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_STATE(0x0008) — When function was called, host stack has not yet been initialized. **Event Data:** none
BLE_GAP_EVENT_LOC_VER_INFO	Version information of local device.

	When version information of local device has been retrieved by R_BLE_GAP_GetVerInfo(), BLE_GAP_EVENT_LOC_VER_INFO event is notified. **Event Code: 0x1004** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_loc_dev_info_evt_t
BLE_GAP_EVENT_HW_ERR	Hardware Error. When hardware error has been received from Controller, BLE_GAP_EVENT_HW_ERR event is notified. **Event Code: 0x1005** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_hw_err_evt_t
BLE_GAP_EVENT_CMD_ERR	Command Status Error. When the error of HCI Command has occurred after a R_BLE GAP API call, BLE_GAP_EVENT_CMD_ERR event is notified. **Event Code: 0x1101** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_cmd_err_evt_t
BLE_GAP_EVENT_ADV_REPT_IND	Advertising Report. When advertising PDUs has been received after scanning was started by R_BLE_GAP_StartScan().

	Event Code: 0x1102 **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** st_ble_gap_adv_rept_evt_t
BLE_GAP_EVENT_ADV_PARAM_SET_COMP	Advertising parameters have been set. Advertising parameters have been configured by R_BLE_GAP_SetAdvParam(). **Event Code: 0x1103** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — The advertising type that doesn't support advertising data/scan response data was specified to the advertising set which has already set advertising data/scan response data. BLE_ERR_INVALID_OPERATION(0x0009) — The reason for this error is as follows. • Advertising parameters were configured to the advertising set in advertising. • The sec_adv_phy field in adv_paran was not

	specified when Periodic Advertising was started. **Event Data:** st_ble_gap_adv_set_evt_t
BLE_GAP_EVENT_ADV_DATA_UPD_COMP	Advertising data has been set. This event notifies that Advertising Data/Scan Response Data/Periodic Advertising Data has been set to the advertising set by R_BLE_GAP_SetAdvSresData(). **Event Code: 0x1104** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_OPERATION(0x0009) — The reason for this error is as follows. • The advertising set that doesn't support advertising data/scan response data was set to the data. • The advertising set that supports legacy advertising was set to advertising data/scan response data larger than 31 bytes.

	• The advertising set that has advertising data/scan response data greater than or equal to 252 bytes was set the data in advertising. • The advertising set that has periodic advertising data greater than or equal to 253 bytes was set the data in advertising. BLE_ERR_MEM_ALLOC_FAILED(0x000C) — Length exceeded the length that the advertising set could be set. BLE_ERR_INVALID_HDL(0x000E) — The advertising set specified by R_BLE_GAP_SetAdvSresData() has not been created. **Event Data:** st_ble_gap_adv_data_evt_t
BLE_GAP_EVENT_ADV_ON	Advertising has started. When advertising has been started by R_BLE_GAP_StartAdv(), this event is notified to the application layer. **Event Code: 0x1105**

result:

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows. • The advertising data length set to the advertising set for connectable extended advertising was invalid. • If o_addr_type field in adv_param used in R_BLE_GAP_SetAdvParam() is 0x03, the address which is set in o_addr field of adv_param has not been registered in Resolving List.
BLE_ERR_INVALID_OPERATION(0x0009)	Setting of advertising data/scan response data has not been completed.
BLE_ERR_INVALID_HDL(0x000E)	The advertising set specified by R_BLE_GAP_StartAdv() has not been created.

	BLE_ERR_LIMIT_EXCEEDED(0x0010) When the maximum connections are established, a new connectable advertising tried starting. **Event Data:** st_ble_gap_adv_set_evt_t
BLE_GAP_EVENT_ADV_OFF	Advertising has stopped. This event notifies the application layer that advertising has stopped. **Event Code: 0x1106** **result:** BLE_SUCCESS(0x0000) Success BLE_ERR_INVALID_HDL(0x000E) The advertising set specified by R_BLE_GAP_StopAdv() has not been created. **Event Data:** st_ble_gap_adv_off_evt_t
BLE_GAP_EVENT_PERD_ADV_PARAM_SET_COMP	Periodic advertising parameters have been set. This event notifies the application layer that Periodic Advertising Parameters has been configured by R_BLE_GAP_SetPerdAdvParam(). **Event Code: 0x1107** **result:** BLE_SUCCESS(0x0000) Success BLE_ERR_INVALID_ARG(0x0003) The advertising set was the setting for anonymous advertising. BLE_ERR_INVALID_OPERATION(0x0009) The advertising set was configured to the parameters in

	periodic advertising. BLE_ERR_INVALID_HDL(0x000E) — The advertising set specified by R_BLE_GAP_SetPerdAdvParam() has not been created. **Event Data:** st_ble_gap_adv_set_evt_t
BLE_GAP_EVENT_PERD_ADV_ON	Periodic advertising has started. When Periodic Advertising has been started by R_BLE_GAP_StartPerdAdv(), this event is notified to the application layer. **Event Code: 0x1108** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_OPERATION(0x0009) — The periodic advertising data set in the advertising set has not been completed. BLE_ERR_INVALID_HDL(0x000E) — The advertising set specified by R_BLE_GAP_StartPerdAdv() has not been created. **Event Data:** st_ble_gap_adv_set_evt_t
BLE_GAP_EVENT_PERD_ADV_OFF	Periodic advertising has stopped. When Periodic Advertising has terminated by R_BLE_GAP_StopPerdAdv(), this event is notified to the application layer. **Event Code: 0x1109** **result:** BLE_SUCCESS(0x0000) — Success

	BLE_ERR_INVALID_HDL(0x000E) — The advertising set specified by R_BLE_GAP_StopPerdAdv() has not been created. **Event Data:** st_ble_gap_adv_set_evt_t
BLE_GAP_EVENT_ADV_SET_REMOVE_COMP	Advertising set has been deleted. When the advertising set has been removed by R_BLE_GAP_RemoveAdvSet(), this event is notified to the application layer. **Event Code: 0x110A** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_OPERATION(0x0009) — When the advertising set was in advertising, R_BLE_GAP_RemoveAdvSet() was called. BLE_ERR_INVALID_HDL(0x000E) — The advertising set specified by R_BLE_GAP_RemoveAdvSet() has not been created. **Event Data:** st_ble_gap_rem_adv_set_evt_t
BLE_GAP_EVENT_SCAN_ON	Scanning has started. When scanning has started by R_BLE_GAP_StartScan(), this event is notified to the application layer. **Event Code: 0x110B** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — The reason for this error is as follows:

	- Scan interval or scan window was invalid. - When filter_dup field in scan_enable was BLE_GAP_SCAN_FILT_DUPLIC_ENABLE_FOR_PERIOD(0x02), period field in scan_enable was 0. - duration field in scan_enable was larger than period in scan_enable. BLE_ERR_INVALID_OPERATION(0x0009) — In scanning, R_BLE_GAP_StartScan() was called. **Event Data:** none
BLE_GAP_EVENT_SCAN_OFF	Scanning has stopped. When scanning has been stopped by R_BLE_GAP_StopScan(), this event is notified to the application layer. **Event Code: 0x110C** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** none
BLE_GAP_EVENT_SCAN_TO	Scanning has stopped, because duration specified by API expired.

	When the scan duration specified by R_BLE_GAP_StartScan() has expired, this event notifies scanning has stopped. **Event Code: 0x110D** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** none
BLE_GAP_EVENT_CREATE_CONN_COMP	Connection Request has been sent to Controller. This event notifies a request for a connection has been sent to Controller. **Event Code: 0x110E** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — The reason for this error is as follows: • Scan interval or scan windows specified by R_BLE_GAP_CreateConn() is invalid. • Although the own_addr_type field in p_param was set to 0x03, random address had not been registered in Resolving

	List. BLE_ERR_INVALID_OPERATION(0x0009) — R_BLE_GAP_CreateConn() was called while creating a link by previous R_BLE_GAP_CreateConn() call . BLE_ERR_LIMIT_EXCEEDED(0x0010) — When the maximum connections are established, R_BLE_GAP_CreateConn() was called. **Event Data:** none
BLE_GAP_EVENT_CONN_IND	Link has been established. This event notifies a link has been established. **Event Code: 0x110F** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_HDL(0x000E) — The request for a connection has been cancelled by R_BLE_GAP_CancelCreateConn(). **Event Data:** st_ble_gap_conn_evt_t
BLE_GAP_EVENT_DISCONN_IND	Link has been disconnected. This event notifies a link has been disconnected. **Event Code: 0x1110** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** st_ble_gap_disconn_evt_t

BLE_GAP_EVENT_CONN_CANCEL_COMP	Connection Cancel Request has been sent to Controller. This event notifies the request for a connection has been cancelled by R_BLE_GAP_CancelCreateConn(). **Event Code: 0x1111** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_OPERATION(0x0009) — When a request for a connection has not been sent to Controller, R_BLE_GAP_CancelCreateConn() was called. **Event Data:** none
BLE_GAP_EVENT_WHITE_LIST_CONF_COMP	The White List has been configured. When White List has been configured, this event is notified to the application layer. **Event Code: 0x1112** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_STATE(0x0008) — The add or delete operation was called, before the previous clear operation has been completed. BLE_ERR_INVALID_OPERATION(0x0009) — While doing advertising or scanning or creating a link with the White List, R_BLE_GAP_ConfWhiteList() was called. BLE_ERR_MEM_ALLOC_FAILED(0x000 — White List has already registered

	C) the maximum number of devices. **Event Data:** st_ble_gap_white_list_conf_evt_t
BLE_GAP_EVENT_RAND_ADDR_SET_COMP	Random address has been set to Controller. This event notifies Controller has been set the random address by R_BLE_GAP_SetRandAddr(). **Event Code: 0x1113** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_OPERATION(0x0009) — When local device was in legacy advertising, R_BLE_GAP_SetRandAddr() was called. **Event Data:** none
BLE_GAP_EVENT_CH_MAP_RD_COMP	Channel Map has been retrieved. This event notifies Channel Map has been retrieved by R_BLE_GAP_ReadChMap(). **Event Code: 0x1114** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_HDL(0x000E) — The remote device specified by R_BLE_GAP_ReadChMap() was not found. **Event Data:** st_ble_gap_rd_ch_map_evt_t
BLE_GAP_EVENT_CH_MAP_SET_COMP	Channel Map has set. This event notifies Channel Map has been configured by R_BLE_GAP_SetChMap().

<table>
<tr><td></td><td>Event Code: 0x1115
result:
<table><tr><td>BLE_SUCCESS(0x0000)</td><td>Success</td></tr><tr><td>BLE_ERR_INVALID_ARG(0x0003)</td><td>The channel map specified by R_BLE_GAP_SetChMap() was all-zero.</td></tr></table>Event Data:
none</td></tr>
<tr><td>BLE_GAP_EVENT_RSSI_RD_COMP</td><td>RSSI has been retrieved.
This event notifies RSSI has been retrieved by R_BLE_GAP_ReadRssi().
Event Code: 0x1116
result:
<table><tr><td>BLE_SUCCESS(0x0000)</td><td>Success</td></tr><tr><td>BLE_ERR_INVALID_HDL(0x000E)</td><td>The remote device specified by R_BLE_GAP_ReadRssi() was not found.</td></tr></table>Event Data:
st_ble_gap_rd_rssi_evt_t</td></tr>
<tr><td>BLE_GAP_EVENT_GET_REM_DEV_INFO</td><td>Information about the remote device has been retrieved.
This event notifies information about the remote device has been retrieved by R_BLE_GAP_GetRemDevInfo().
Event Code: 0x1117
result:
<table><tr><td>BLE_SUCCESS(0x0000)</td><td>Success</td></tr></table>Event Data:
st_ble_gap_dev_info_evt_t</td></tr>
</table>

BLE_GAP_EVENT_CONN_PARAM_UPD_COMP	Connection parameters has been configured. This event notifies the connection parameters has been updated. **Event Code: 0x1118** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_DATA(0x0002) — Local device rejected the request for updating connection parameters. BLE_ERR_INVALID_ARG(0x0003) — The remote device rejected the connection parameters suggested from local device. BLE_ERR_UNSUPPORTED(0x0007) — The remote device doesn't support connection parameters update feature. **Event Data:** st_ble_gap_conn_upd_evt_t
BLE_GAP_EVENT_CONN_PARAM_UPD_REQ	Local device has received the request for configuration of connection parameters. This event notifies the request for connection parameters update has been received. **Event Code: 0x1119** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** st_ble_gap_conn_upd_req_evt_t
BLE_GAP_EVENT_AUTH_PL_TO_EXPIRED	Authenticated Payload Timeout. This event notifies Authenticated Payload

	Timeout has occurred. **Event Code: 0x111A** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** st_ble_gap_conn_hdl_evt_t
BLE_GAP_EVENT_SET_DATA_LEN_COMP	The request for update transmission packet size and transmission time have been sent to Controller. This event notifies a request for updating packet data length and transmission timer has been sent to Controller. **Event Code: 0x111B** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — The tx_octets or tx_time parameter specified by R_BLE_GAP_SetDataLen() is invalid. BLE_ERR_UNSUPPORTED(0x0007) — The remote device does not support updating packet data length and transmission time. BLE_ERR_INVALID_HDL(0x000E) — When R_BLE_GAP_SetDataLen() was called, the connection was not established. **Event Data:** st_ble_gap_conn_hdl_evt_t
BLE_GAP_EVENT_DATA_LEN_CHG	Transmission packet size and transmission time have been changed. This event notifies packet data length and transmission time have been updated.

	Event Code: 0x111C **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** st_ble_gap_data_len_chg_evt_t
BLE_GAP_EVENT_RSLV_LIST_CONF_COMP	The Resolving List has been configured. When Resolving List has been configured by R_BLE_GAP_ConfRslvList(), this event is notified to the application layer. **Event Code: 0x111D** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_STATE(0x0008) — The add or delete operation was called, before the previous clear operation has been completed. BLE_ERR_INVALID_OPERATION(0x0009) — While doing advertising or scanning or creating a link with resolvable private address, R_BLE_GAP_ConfRslvList() was called. BLE_ERR_MEM_ALLOC_FAILED(0x000C) — Resolving List has already registered the maximum number of devices. BLE_ERR_INVALID_HDL(0x000E) — The specified Identity Address was not found in Resolving List. **Event Data:** st_ble_gap_rslv_list_conf_evt_t
BLE_GAP_EVENT_RPA_EN_COMP	Resolvable private address function has been

	enabled or disabled. When Resolvable Private Address function in Controller has been enabled by R_BLE_GAP_EnableRpa(), this event is notified to the application layer. **Event Code: 0x111E** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_OPERATION(0x0009) — While advertising, scanning, or establishing a link with resolvable private address, R_BLE_GAP_EnableRpa() was called. **Event Data:** none
BLE_GAP_EVENT_SET_RPA_TO_COMP	The update time of resolvable private address has been changed. When Resolvable Private Address Timeout in Controller has been updated by R_BLE_GAP_SetRpaTo(), this event is notified to the application layer. **Event Code: 0x111F** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — The rpa_timeout parameter specified by R_BLE_GAP_SetRpaTo() is out of range. **Event Data:** none
BLE_GAP_EVENT_RD_RPA_COMP	The resolvable private address of local device has been retrieved. When the resolvable private address of local

	device has been retrieved by R_BLE_GAP_ReadRpa(), this event is notified to the application layer. **Event Code: 0x1120** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_HDL(0x000E) — The identity address specified by R_BLE_GAP_ReadRpa() was not registered in Resolving List. **Event Data:** st_ble_gap_rd_rpa_evt_t
BLE_GAP_EVENT_PHY_UPD	PHY for connection has been changed. This event notifies the application layer that PHY for a connection has been updated. **Event Code: 0x1121** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** st_ble_gap_phy_upd_evt_t
BLE_GAP_EVENT_PHY_SET_COMP	The request for updating PHY for connection has been sent to Controller. When Controller has received a request for updating PHY for a connection by R_BLE_GAP_SetPhy(), this event is notified to the application layer. **Event Code: 0x1122** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_HDL(0x000E) — The remote device specified by

	R_BLE_GAP_SetPhy() was not found. **Event Data:** st_ble_gap_conn_hdl_evt_t
BLE_GAP_EVENT_DEF_PHY_SET_COMP	The request for setting default PHY has been sent to Controller. When the PHY preferences which a remote device may change has been configured by R_BLE_GAP_SetDefPhy(), this event is notified to the application layer. **Event Code: 0x1123** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** none
BLE_GAP_EVENT_PHY_RD_COMP	PHY configuration has been retrieved. When the PHY settings has been retrieved by R_BLE_GAP_ReadPhy(), this event is notified to the application layer. **Event Code: 0x1124** **result:** BLE_SUCCESS(0x0000) Success BLE_ERR_INVALID_HDL(0x000E) The link specified by R_BLE_GAP_ReadPhy() was not found. **Event Data:** st_ble_gap_phy_rd_evt_t
BLE_GAP_EVENT_SCAN_REQ_RECV	Scan Request has been received. This event notifies the application layer that a Scan Request packet has been received from a Scanner.

	Event Code: 0x1125 **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** st_ble_gap_scan_req_recv_evt_t
BLE_GAP_EVENT_CREATE_SYNC_COMP	The request for establishing a periodic sync has been sent to Controller. This event notifies the application layer that Controller has received a request for a Periodic Sync establishment. **Event Code: 0x1126** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_OPERATION(0x0009) — When R_BLE_GAP_CreateSync() was called, this event for previous the API call has not been received. BLE_ERR_ALREADY_IN_PROGRESS(0x000A) — The advertising set specified by R_BLE_GAP_CreateSync() has already established a periodic sync. **Event Data:** none
BLE_GAP_EVENT_SYNC_EST	The periodic advertising sync has been established. This event notifies the application layer that a Periodic sync has been established. **Event Code: 0x1127** **result:** BLE_SUCCESS(0x0000) — Success

	BLE_ERR_NOT_YET_READY(0x0012) — The request for a Periodic Sync establishment was cancelled by R_BLE_GAP_CancelCreateSync(). **Event Data:** st_ble_gap_sync_est_evt_t
BLE_GAP_EVENT_SYNC_TERM	The periodic advertising sync has been terminated. This event notifies the application layer that the Periodic Sync has been terminated by R_BLE_GAP_TerminateSync(). **Event Code: 0x1128** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_OPERATION(0x0009) — While establishing a Periodic Sync by R_BLE_GAP_CreateSync(), R_BLE_GAP_TerminateSync() was called. BLE_ERR_INVALID_HDL(0x000E) — The sync handle specified by R_BLE_GAP_TerminateSync() was not found. **Event Data:** st_ble_gap_sync_hdl_evt_t
BLE_GAP_EVENT_SYNC_LOST	The periodic advertising sync has been lost. This event notifies the application layer that the Periodic Sync has been lost. **Event Code: 0x1129** **result:** BLE_SUCCESS(0x0000) — Success

	Event Data: st_ble_gap_sync_hdl_evt_t
BLE_GAP_EVENT_SYNC_CREATE_CANCEL_COMP	The request for cancel of establishing a periodic advertising sync has been sent to Controller. This event notifies the request for a Periodic Sync establishment has been cancelled by R_BLE_GAP_CancelCreateSync(). **Event Code: 0x112A** **result:** BLE_SUCCESS(0x0000): Success BLE_ERR_INVALID_OPERATION(0x0009): When R_BLE_GAP_CancelCreateSync() was called, a request for a Periodic Sync establishment by R_BLE_GAP_CreateSync() has not been sent to Controller. **Event Data:** none
BLE_GAP_EVENT_PERD_LIST_CONF_COMP	The Periodic Advertiser list has been configured. When Periodic Advertiser List has been configured by R_BLE_GAP_ConfPerdAdvList(), this event is notified to the application layer. **Event Code: 0x112B** **result:** BLE_SUCCESS(0x0000): Success BLE_ERR_INVALID_ARG(0x0003): The advertiser has already been registered in Periodic Advertiser List. BLE_ERR_INVALID_STATE(0x0008): The add or delete operation was

	called, before the previous clear operation has been completed. BLE_ERR_INVALID_OPERATION(0x0009) — When establishing a periodic sync by R_BLE_GAP_CreateSync(), R_BLE_GAP_ConfPerdAdvList() was called. BLE_ERR_MEM_ALLOC_FAILED(0x000C) — Periodic Advertiser List has already registered the maximum number of devices. BLE_ERR_INVALID_HDL(0x000E) — The device specified by R_BLE_GAP_ConfPerdAdvList() was not found. **Event Data:** st_ble_gap_perd_list_conf_evt_t
BLE_GAP_EVENT_PRIV_MODE_SET_COMP	Privacy Mode has been configured. This event notifies the application layer that the Privacy Mode has been configured by R_BLE_GAP_SetPrivMode(). **Event Code: 0x112B** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — Address type or privacy mode is out of range. BLE_ERR_INVALID_OPERATION(0x0009) — While advertising, scanning, or establishing a link with resolvable private address, R_BLE_GAP_SetPrivMode() was called. BLE_ERR_INVALID_HDL(0x000E) — The address specified by

	R_BLE_GAP_SetPrivMode() has not been registered in Resolving List. **Event Data:** none
BLE_GAP_EVENT_PAIRING_REQ	The pairing request from a remote device has been received. This event notifies the application layer that a pairing request from a remote device has been received. **Event Code: 0x1401** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_pairing_info_evt_t
BLE_GAP_EVENT_PASSKEY_ENTRY_REQ	The request for input passkey has been received. This event notifies that a request for Passkey input in pairing has been received. **Event Code: 0x1402** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_conn_hdl_evt_t
BLE_GAP_EVENT_PASSKEY_DISPLAY_REQ	The request for displaying a passkey has been received. This event notifies that a request for Passkey display in pairing has been received. **Event Code: 0x1403** **result:** BLE_SUCCESS(0x0 Success

	000) **Event Data:** st_ble_gap_passkey_display_evt_t
BLE_GAP_EVENT_NUM_COMP_REQ	The request for confirmation with Numeric Comparison has received. This event notifies that a request for Numeric Comparison in pairing has been received. **Event Code: 0x1404** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_num_comp_evt_t
BLE_GAP_EVENT_KEY_PRESS_NTF	Key Notification from a remote device has been received. This event notifies the application layer that the remote device has input a key in Passkey Entry. **Event Code: 0x1405** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_key_press_ntf_evt_t
BLE_GAP_EVENT_PAIRING_COMP	Pairing has been completed. This event notifies the application layer that the pairing has completed. **Event Code: 0x1406** **result:** BLE_SUCCESS(0x0000) Success BLE_ERR_SMP_LE_PASSKEY_ENTRY_F PassKey Entry is failed.

AIL(0x2001)	
BLE_ERR_SMP_LE_OOB_DATA_NOT_AVAILABLE(0x2002)	OOB Data is not available.
BLE_ERR_SMP_LE_AUTH_REQ_NOT_MET(0x2003)	The requested pairing can not be performed because of IO Capability.
BLE_ERR_SMP_LE_CONFIRM_VAL_NOT_MATCH(0x2004)	Confirmation value does not match.
BLE_ERR_SMP_LE_PAIRING_NOT_SPRT(0x2005)	Pairing is not supported.
BLE_ERR_SMP_LE_INSUFFICIENT_ENC_KEY_SIZE(0x2006)	Encryption Key Size is insufficient.
BLE_ERR_SMP_LE_CMD_NOT_SPRT(0x2007)	The pairing command received is not supported.
BLE_ERR_SMP_LE_UNSPECIFIED_REASON(0x2008)	Pairing failed with an unspecified reason.
BLE_ERR_SMP_LE_REPEATED_ATTEMPTS(0x2009)	The number of repetition exceeded the upper limit.
BLE_ERR_SMP_LE_INVALID_PARAM(0x200A)	Invalid parameter is set.
BLE_ERR_SMP_LE_DHKEY_CHECK_FAIL(0x200B)	DHKey Check error.
BLE_ERR_SMP_LE_NUM_COMP_FAIL(0x200C)	Numeric Comparison failure.
BLE_ERR_SMP_LE_DISCONNECTED(0x200F)	Disconnection in pairing.
BLE_ERR_SMP_LE_TO(0x2011)	Failure due to timeout.
BLE_ERR_SMP_LE_LOC_KEY_MISSING(0x2014)	Pairing/Encryption failure because local device lost the LTK.

	Event Data: st_ble_gap_pairing_info_evt_t
BLE_GAP_EVENT_ENC_CHG	Key Notification from a remote device has been received. This event notifies the application layer that the encryption status of a link has been changed. **Event Code: 0x1407** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_enc_chg_evt_t
BLE_GAP_EVENT_PEER_KEY_INFO	Keys has been received from a remote device. This event notifies the application layer that the remote device has distributed the keys. **Event Code: 0x1408** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_peer_key_info_evt_t
BLE_GAP_EVENT_EX_KEY_REQ	The request for key distribution has been received. When local device has been received a request for key distribution to remote device, this event is notified to the application layer. **Event Code: 0x1409** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_conn_hdl_evt_t

BLE_GAP_EVENT_LTK_REQ	LTK has been request from a remote device. When local device has been received a LTK request from a remote device, this event is notified to the application layer. **Event Code: 0x140A** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_ltk_req_evt_t
BLE_GAP_EVENT_LTK_RSP_COMP	LTK reply has been sent to Controller. When local device has replied to the LTK request from the remote device, this event is notified to the application layer. **Event Code: 0x140B** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_ltk_rsp_evt_t
BLE_GAP_EVENT_SC_OOB_CREATE_COMP	The authentication data to be used in Secure Connections OOB has been created. This event notifies OOB data for Secure Connections has been generated by R_BLE_GAP_CreateScOobData(). **Event Code: 0x140C** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gap_sc_oob_data_evt_t

Function Documentation

◆ R_BLE_GAP_Init()

ble_status_t R_BLE_GAP_Init (ble_gap_app_cb_t *gap_cb*)

Initialize the Host Stack.

Host stack is initialized with this function. Before using All the R_BLE APIs, it's necessary to call this function. A callback function is registered with this function. In order to receive the GAP event, it's necessary to register a callback function. The result of this API call is notified in BLE_GAP_EVENT_STACK_ON event.

Parameters

[in]	gap_cb	A callback function registered with this function.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	gap_cb is specified as NULL.
BLE_ERR_INVALID_STATE(0x0008)	The reason for this error is as follows: • Host Stack was already initialized. • The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_Terminate()

ble_status_t R_BLE_GAP_Terminate (void)

Terminate the Host Stack.

Host stack is terminated with this function. In order to reset all the Bluetooth functions, it's necessary to call this function. The result of this API call is notified in BLE_GAP_EVENT_STACK_OFF event.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	Host stack hasn't been initialized.

◆ R_BLE_GAP_UpdConn()

ble_status_t R_BLE_GAP_UpdConn (uint16_t *conn_hdl*, uint8_t *mode*, uint16_t *accept*, st_ble_gap_conn_param_t * *p_conn_updt_param*)

Update the connection parameters.

This function updates the connection parameters or replies a request for updating connection parameters notified by BLE_GAP_EVENT_CONN_PARAM_UPD_REQ event. When the connection parameters has been updated, BLE_GAP_EVENT_CONN_PARAM_UPD_COMP event is notified to the application layer.

Parameters

<table>
<tr><td>[in]</td><td>conn_hdl</td><td>Connection handle identifying the link to be updated.</td></tr>
<tr><td>[in]</td><td>mode</td><td>Connection parameter update request or response.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_CONN_UPD_MODE_REQ (0x01)</td><td>Request for updating the connection parameters.</td></tr><tr><td>BLE_GAP_CONN_UPD_MODE_RSP (0x02)</td><td>Reply a connection parameter update request.</td></tr></table></td></tr>
<tr><td>[in]</td><td>accept</td><td>When mode is BLE_GAP_CONN_UPD_MODE_RSP, accept or reject the connection parameters update request. If mode is BLE_GAP_CONN_UPD_MODE_REQ, accept is ignored.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_CONN_UPD_ACCEPT (0x0000)</td><td>Accept the update request.</td></tr><tr><td>BLE_GAP_CONN_UPD_REJECT (0x0001)</td><td>Reject the update request.</td></tr></table></td></tr>
<tr><td>[in]</td><td>p_conn_updt_param</td><td>Connection parameters to be updated. When mode is BLE_GAP_CONN_UPD_MODE_RSP and accept is</td></tr>
</table>

		BLE_GAP_CONN_UPD_REJECT, p_conn_updt_param is ignored.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	When accept is BLE_GAP_CONN_UPD_ACCEPT, p_conn_updt_param is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The following is out of range. • mode • accept • conn_intv_min field in p_conn_updt_param • conn_intv_max field in p_conn_updt_param • conn_latency in p_conn_updt_param • sup_to in p_conn_updt_param • conn_hdl
BLE_ERR_INVALID_STATE(0x0008)	Not connected with the remote device.
BLE_ERR_CONTEXT_FULL(0x000B)	Sending a L2CAP command, an error occurred.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.

◆ R_BLE_GAP_SetDataLen()

ble_status_t R_BLE_GAP_SetDataLen (uint16_t *conn_hdl*, uint16_t *tx_octets*, uint16_t *tx_time*)

Update the packet size and the packet transmit time.

This function requests for changing the maximum transmission packet size and the maximum packet transmission time. When Controller has received the request from host stack, BLE_GAP_EVENT_SET_DATA_LEN_COMP event is notified to the application layer. When the transmission packet size or the transmission time has been changed, BLE_GAP_EVENT_DATA_LEN_CHG event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the link whose the transmission packet size or the transmission time to be changed.
[in]	tx_octets	Maximum transmission packet size. Valid range is 0x001B - 0x00FB.
[in]	tx_time	Maximum transmission time(us). Valid range is 0x0148 - 0x4290.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_Disconnect()

ble_status_t R_BLE_GAP_Disconnect (uint16_t *conn_hdl*, uint8_t *reason*)

Disconnect the link.

This function disconnects a link. When the link has disconnected, BLE_GAP_EVENT_DISCONN_IND event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the link to be disconnected.
[in]	reason	The reason for disconnection. Usually, set 0x13 which indicates that a user disconnects the link. If setting other than 0x13, refer the error code described in Core Specification Vol.2 Part D ,"2 Error Code Descriptions".

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	conn_hdl is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.

◆ R_BLE_GAP_SetPhy()

ble_status_t R_BLE_GAP_SetPhy (uint16_t *conn_hdl*, st_ble_gap_set_phy_param_t * *p_phy_param*)

Set the phy for connection.

This function sets the PHY preferences for the connection. The result of this API call is notified in BLE_GAP_EVENT_PHY_SET_COMP event. When the PHY has been updated, BLE_GAP_EVENT_PHY_UPD event is notified to the application layer.

After PHY update, the PHY accept configuration of local device is the same as the values in BLE_GAP_EVENT_PHY_UPD event.
For example, after calling R_BLE_GAP_SetPhy(), if tx_phy, rx_phy by BLE_GAP_EVENT_PHY_UPD event are updated to 2M PHY, the PHY accept configuration is 2M PHY only.
Therefore after receiving BLE_GAP_EVENT_PHY_UPD event, if local device wants to accept the other PHY configuration, it needs to call R_BLE_GAP_SetPhy() with the desired PHY accept configuration.

Because the maximum transmission packet size or the maximum transmission time might be updated by PHY update, if the same packet size or transmission time as the previous one is desired, change the maximum transmission packet size or the maximum transmission time by R_BLE_GAP_SetDataLen().

Parameters

[in]	conn_hdl	Connection handle identifying the link whose PHY to be updated.
[in]	p_phy_param	PHY preferences.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_phy_param is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	conn_hdl or option field in p_phy_param is out of range.
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_SetDefPhy()

ble_status_t R_BLE_GAP_SetDefPhy (st_ble_gap_set_def_phy_param_t * *p_def_phy_param*)

Set the default phy which allows remote device to change.

This function sets the PHY preferences which a remote device may change. The result of this API call is notified in BLE_GAP_EVENT_DEF_PHY_SET_COMP event.

Parameters

[in]	p_def_phy_param	The PHY preference which a remote device may change.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_def_phy_param is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	tx_phys or tx_phys field in p_def_phy_param is out of range.
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_SetPrivMode()

ble_status_t R_BLE_GAP_SetPrivMode (st_ble_dev_addr_t * *p_addr*, uint8_t * *p_privacy_mode*, uint8_t *device_num*)

Set the privacy mode.

This function sets privacy mode for the remote device registered in Resolving List. By default, Network Privacy Mode is set.
The result of this API call is notified in BLE_GAP_EVENT_PRIV_MODE_SET_COMP event.

Parameters

[in]	p_addr	An array of identity address of the remote device to set privacy mode. The number of elements is specified by device_num.
[in]	p_privacy_mode	An array of privacy mode to set to remote device. The number of elements is specified by device_num. The following value is set as

		the privacy mode. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_NET_PRIV_MODE (0x00)</td><td>Network Privacy Mode.</td></tr><tr><td>BLE_GAP_DEV_PRIV_MODE (0x01)</td><td>Device Privacy Mode.</td></tr></table>
[in]	device_num	The number of devices to set privacy mode. Valid range is 1-BLE_GAP_RSLV_LIST_MAX_ENTRY.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_addr or p_privacy_mode is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The following parameter is out of range. • The address type in p_addr. • The privacy mode specified by p_privacy_mode. • device_num
BLE_ERR_INVALID_STATE(0x0008)	The reason for this error is as follows: • While configuring privacy mode, this function was called. • The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_ConfWhiteList()

ble_status_t R_BLE_GAP_ConfWhiteList (uint8_t *op_code*, st_ble_dev_addr_t * *p_addr*, uint8_t *device_num*)

Set White List.

This function supports the following operations regarding White List.

- Add the device to White List.
- Delete the device from White List.

- Clear White List.

  The total number of White List entries is defined as BLE_GAP_WHITE_LIST_MAX_ENTRY. The result of this API call is notified in BLE_GAP_EVENT_WHITE_LIST_CONF_COMP event.

Parameters

[in]	op_code	The operation for White List. macro / description: BLE_GAP_LIST_ADD_DEV(0x01): Add the device to the list. BLE_GAP_LIST_REM_DEV(0x02): Delete the device from the list. BLE_GAP_LIST_CLR(0x03): Clear the list.
[in]	p_addr	An array of device address to add / delete to the list. The number of elements is specified by device_num. If op_code is BLE_GAP_LIST_CLR, p_addr is ignored.
[in]	device_num	The number of devices add / delete to the list. Valid range is 1-BLE_GAP_WHITE_LIST_MAX_ENTRY. If op_code is BLE_GAP_LIST_CLR, device_num is ignored.

Return values

BLE_SUCCESS(0x0000)	Success

BLE_ERR_INVALID_PTR(0x0001)	When op_code is BLE_GAP_LIST_ADD_DEV or BLE_GAP_LIST_REM_DEV, p_addr is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	op_code or address type field in p_addr is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The reason for this error is as follows: • While operating White List, this function was called. • The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for operating the White List.

◆ R_BLE_GAP_GetVerInfo()

ble_status_t R_BLE_GAP_GetVerInfo (void)

Get the version number of the Controller and the host stack.

This function retrieves the version information of local device. The result of this API call is notified in BLE_GAP_EVENT_LOC_VER_INFO event.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_ReadPhy()

ble_status_t R_BLE_GAP_ReadPhy (uint16_t *conn_hdl*)

Get the phy settings.

This function gets the PHY settings for the connection. The result of this API call is notified in BLE_GAP_EVENT_PHY_RD_COMP event.

Parameters

[in]	conn_hdl	Connection handle identifying the link whose PHY settings to be retrieved.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	conn_hdl is out of range.
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_ConfRslvList()

ble_status_t R_BLE_GAP_ConfRslvList (uint8_t *op_code*, st_ble_dev_addr_t * *p_addr*, st_ble_gap_rslv_list_key_set_t * *p_peer_irk*, uint8_t *device_num*)

Set Resolving List.

This function supports the following operations regarding Resolving List.

- Add the device to Resolving List.
- Delete the device from Resolving List.

- Clear Resolving List.

  In order to generate a resolvable private address, a local IRK needs to be registered by R_BLE_GAP_SetLocIdInfo(). If communicating with the identity address, register all-zero IRK as local IRK. In order to resolve resolvable private address of the remote device, the IRK distributed from the remote device needs to be added to Resolving List. The total number of Resolving List entries is defined as BLE_GAP_RESOLV_LIST_MAX_ENTRY. The result of this API call is notified in BLE_GAP_EVENT_RSLV_LIST_CONF_COMP event.

Parameters

[in]	op_code	The operation for Resolving List. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_LIST_ADD_DEV(0x01)</td><td>Add the device to the list.</td></tr><tr><td>BLE_GAP_LIST_REM_DEV(0x02)</td><td>Delete the device from the list.</td></tr><tr><td>BLE_GAP_LIST_CLR(0x03)</td><td>Clear the list.</td></tr></table>
[in]	p_addr	An array of Identity Addresses to add / delete to the list. The number of elements is specified by device_num. If op_code is BLE_GAP_LIST_CLR, p_addr is ignored.
[in]	p_peer_irk	The remote IRK and the type of local IRK added to Resolving List. If op_code is other than BLE_GAP_LIST_ADD_DEV, p_peer_irk is ignored. The number of elements is

		specified by device_num.
[in]	device_num	The number of devices add / delete to the list. Valid range is 1-BLE_GAP_RSLV_LIST_MAX_ENTRY. If op_code is BLE_GAP_LIST_CLR, device_num is ignored.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The reason for this error is as follows: • When added to or deleted from the list, p_addr is specified as NULL. • When added to the list, p_peer_irk is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • op_code is out of range. • When op_code is BLE_GAP_LIST_ADD_DEV or BLE_GAP_LIST_REM_DEV, device_num is out of range. • When op_code is BLE_GAP_LIST_ADD_DEV or BLE_GAP_LIST_REM_DEV, address type field in p_addr is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The reason for this error is as follows: • While operating Resolving List, this function was called. • The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for operating the Resolving List.
BLE_ERR_INVALID_HDL(0x000E)	The specified Identity Address was not found in Resolving List.

◆ R_BLE_GAP_EnableRpa()

ble_status_t R_BLE_GAP_EnableRpa (uint8_t *enable*)

Enable/Disable address resolution and generation of a resolvable private address.

This function enables or disables RPA functionality. The RPA functionality includes the following.

- Generation of local resolvable private address
- Resolution of remote resolvable private address

In order to do advertising, scanning or creating a link with local resolvable private address, the RPA functionality needs to be enabled. After enabling the RPA functionality and the identity address of remote device and the IRKs of local/remote device is registered, local device can generate own resolvable private address in the time interval set by R_BLE_GAP_SetRpaTo(), and can resolve a resolvable private address of a remote device. It is recommended that the RPA functionality is called immediately after the initialization by R_BLE_GAP_Init(). The result of this API call is notified in BLE_GAP_EVENT_RPA_EN_COMP event.

Parameters

[in]	enable	Enable or disable address resolution function. macro / description: BLE_GAP_RPA_DISABLED(0x00): Disable RPA generation/resolution. BLE_GAP_RPA_ENABLED(0x01): Enable RPA generation/resolution.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	enable is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_SetRpaTo()

ble_status_t R_BLE_GAP_SetRpaTo (uint16_t *rpa_timeout*)

Set the update time of resolvable private address.

This function sets the time interval to update the resolvable private address. The result of this API call is notified in BLE_GAP_EVENT_SET_RPA_TO_COMP event.

Parameters

[in]	rpa_timeout	Time interval to update resolvable private address in seconds. Valid range is 0x003C - 0xA1B8. Default is 900s.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_ReadRpa()

ble_status_t R_BLE_GAP_ReadRpa (st_ble_dev_addr_t * *p_addr*)

Get the resolvable private address of local device.

This function retrieves the local resolvable private address. Before getting the address, enable the resolvable private address function by R_BLE_GAP_EnableRpa(). The result of this API call is notified in BLE_GAP_EVENT_RD_RPA_COMP event.

Parameters

[in]	p_addr	Identity address registered in Resolving List.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_addr is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	Address type in p_addr is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The reason for this error is as follows. • When retrieving the local resolvable private address, this function was called. • The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_ReadRssi()

ble_status_t R_BLE_GAP_ReadRssi (uint16_t *conn_hdl*)

Get RSSI.

This function retrieves RSSI. The result of this API call is notified in BLE_GAP_EVENT_RSSI_RD_COMP event.

Parameters

[in]	conn_hdl	Connection handle identifying the link whose RSSI to be retrieved.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	conn_hdl is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_ReadChMap()

ble_status_t R_BLE_GAP_ReadChMap (uint16_t *conn_hdl*)

Get the Channel Map.

This function retrieves the channel map. The result of this API call is notified in BLE_GAP_EVENT_CH_MAP_RD_COMP event.

Parameters

[in]	conn_hdl	Connection handle identifying the link whose channel map to be retrieved.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	conn_hdl is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_SetRandAddr()

ble_status_t R_BLE_GAP_SetRandAddr (uint8_t * *p_random_addr*)

Set a random address.

This function sets static address or non-resolvable private address to Controller. Refer to Core Specification Vol 6, PartB, "1.3.2 Random Device Address" regarding the format of the random address. Resolvable private address cannot set by this API. The result of this API call is notified in BLE_GAP_EVENT_RAND_ADDR_SET_COMP event.

Parameters

[in]	p_random_addr	Static address or non-resolvable private address. The BD address setting format is little endian.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_random_addr is specified as NULL.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_SetAdvParam()

ble_status_t R_BLE_GAP_SetAdvParam (st_ble_gap_adv_param_t * *p_adv_param*)

Set advertising parameters.

This function sets advertising parameters. It's possible to do advertising where the advertising parameters are different every each advertising set. The number of advertising set in the Controller is defined as BLE_MAX_NO_OF_ADV_SETS_SUPPORTED. Each advertising set is identified with advertising handle (0x00-0x03). Create an advertising set with this function before start advertising, setting periodic advertising parameters, start periodic advertising, setting advertising data/scan response data/periodic advertising data. The result of this API call is notified in BLE_GAP_EVENT_ADV_PARAM_SET_COMP event.

Parameters

[in]	p_adv_param	Advertising parameters.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_adv_param is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The below p_adv_param field value is out of range. • adv_handle • adv_intv_min/adv_intv_max • adv_ch_map • o_addr_type • p_addr_type • adv_phy • sec_adv_phy • scan_req_ntf_flag
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_SetAdvSresData()

ble_status_t R_BLE_GAP_SetAdvSresData (st_ble_gap_adv_data_t * *p_adv_srsp_data*)

Set advertising data/scan response data/periodic advertising data.

This function sets advertising data/scan response data/periodic advertising data to the advertising set. It is necessary to create an advertising set by R_BLE_GAP_SetAdvParam(), before calling this function. Set advertising data/scan response data/periodic advertising data, after allocating the memory for the data. The following shall be applied regarding the adv_prop_type field and the data_type field in st_ble_gap_adv_param_t parameter specified in R_BLE_GAP_SetAdvParam().

The following shall be applied regarding the adv_prop_type field and the data_type field in st_ble_gap_adv_param_t parameter specified in R_BLE_GAP_SetAdvParam().

- When adv_prop_type is Legacy Advertising PDU type,
  - it's possible to set advertising data/scan response data up to 31 bytes.
  - advertising data/scan response data can be updated by this function in advertising.
- When adv_prop_type is Extended Advertising PDU type,
  - it's possible to set at most 1650 bytes of data as advertising data/scan response data per 1 advertising set.
  - the total buffer size in Controller for advertising data/scan response data is 4250 bytes. Therefore please note that more than 4250 bytes of advertising data/scan response data can not be set to all the advertising sets. Please refer to Figure 1.1 and Figure 1.2 about examples of setting advertising data/scan response data.
  - it's possible to update advertising data/scan response data in advertising, if the data_length field in st_ble_gap_adv_data_t parameter is up to 251 bytes.

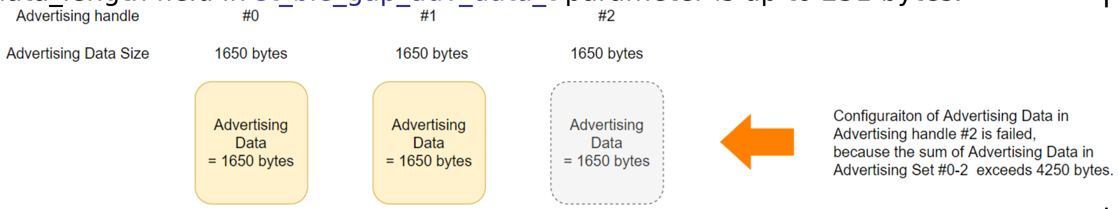


Figure 229: Figure 1.1

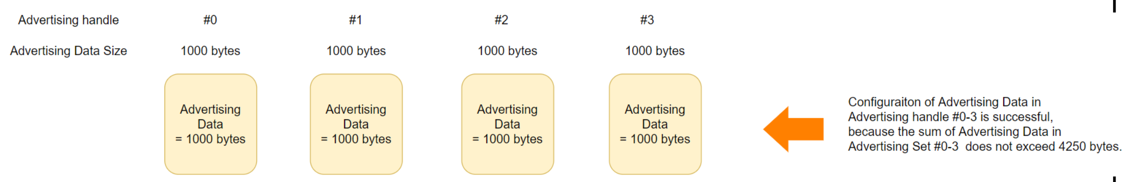


Figure 230: Figure 1.2

- When periodic advertising data is set,
  - At most 1650 bytes of data can be set to 1 advertising set.
  - The total buffer size in Controller for periodic advertising data is 4306 bytes. Therefore please note that more than 4306 bytes of periodic advertising data can not be set to all the advertising sets.
  - it's possible to update periodic advertising data in advertising, if the data_length field in st_ble_gap_adv_data_t parameter is up to 252 bytes.

The result of this API call is notified in BLE_GAP_EVENT_ADV_DATA_UPD_COMP event.

Parameters

[in]	p_adv_srsp_data	Advertising data/scan response data/periodic advertising data.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The reason for this error is as follows: • p_adv_srsp_data is specified as NULL. • data_length field in p_adv_srsp_data parameter is not 0 and p_data field is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The following field in p_adv_srsp_data parameter is out of range. • adv_hdl • data_type • data_length • zero_length_flag
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_StartAdv()

ble_status_t R_BLE_GAP_StartAdv (uint8_t *adv_hdl*, uint16_t *duration*, uint8_t *max_extd_adv_evts*)

Start advertising.

This function starts advertising. Create the advertising set specified with adv_hdl by R_BLE_GAP_SetAdvParam(), before calling this function. The result of this API call is notified in BLE_GAP_EVENT_ADV_ON event.

Parameters

[in]	adv_hdl	The advertising handle pointing to the advertising set which starts advertising. The valid range is 0x00 - 0x03.
[in]	duration	The duration for which the advertising set identified by adv_hdl is enabled. Time = duration * 10ms. When the duration expires, BLE_GAP_EVENT_ADV_OFF event notifies that advertising is stopped. The valid range is 0x0000 - 0xFFFF. The duration parameter is ignored when the value is set to 0x0000.
[in]	max_extd_adv_evts	The maximum number of advertising events that be sent during advertising. When all the advertising events(max_extd_adv_evts) have been sent, BLE_GAP_EVENT_ADV_OFF event notifies that advertising is stopped. The max_extd_adv_evts parameter is ignored when the value is set to 0x00.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	adv_hdl is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_StopAdv()

ble_status_t R_BLE_GAP_StopAdv (uint8_t *adv_hdl*)

Stop advertising.

This function stops advertising. The result of this API call is notified in BLE_GAP_EVENT_ADV_OFF event.

Parameters

[in]	adv_hdl	The advertising handle pointing to the advertising set which stops advertising. The valid range is 0x00 - 0x03.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	adv_hdl is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_SetPerdAdvParam()

ble_status_t R_BLE_GAP_SetPerdAdvParam (st_ble_gap_perd_adv_param_t * *p_perd_adv_param*)

Set periodic advertising parameters.

This function sets periodic advertising parameters. Create the advertising set which supports Non-Connectable, Non-Scannable advertising by R_BLE_GAP_SetAdvParam() before setting periodic advertising parameters. The result of this API call is notified in BLE_GAP_EVENT_PERD_ADV_PARAM_SET_COMP event.

Parameters

[in]	p_perd_adv_param	Periodic advertising parameters.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_perd_adv_param is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The following field in the p_perd_adv_param parameter is out of range. • adv_hdl • perd_intv_min or perd_intv_max • prop_type is neither 0x0000 nor 0x0040(BLE_GAP_PERD_PROP_TX_POWER)
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_StartPerdAdv()

ble_status_t R_BLE_GAP_StartPerdAdv (uint8_t *adv_hdl*)

Start periodic advertising.

This function starts periodic advertising. Set periodic advertising parameters to the advertising set, before starting periodic advertising. The result of this API call is notified in BLE_GAP_EVENT_PERD_ADV_ON event.

Parameters

[in]	adv_hdl	Advertising handle identifying the advertising set which starts periodic advertising.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	adv_hdl is out of range.
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_StopPerdAdv()

ble_status_t R_BLE_GAP_StopPerdAdv (uint8_t *adv_hdl*)

Stop periodic advertising.

This function stops periodic advertising. If the return value of this API is BLE_SUCCESS, the result is notified in BLE_GAP_EVENT_PERD_ADV_OFF event.

Parameters

[in]	adv_hdl	Specify the handle of Advertising Set to stop Periodic Advertising.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	adv_hdl is out of range.
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_GetRemainAdvBufSize()

ble_status_t R_BLE_GAP_GetRemainAdvBufSize (uint16_t * *p_remain_adv_data_size*, uint16_t * *p_remain_perd_adv_data_size*)

Get buffer size for advertising data/scan response data/periodic advertising data in the Controller.

This function gets the total size of advertising data/scan response data/periodic advertising data which can be currently set to Controller(all of the advertising sets). The application layer gets the data sizes via the parameters. By this API function call, no events occur.

Parameters

[out]	p_remain_adv_data_size	The free buffer size of Controller to which advertising data/scan response data can be currently set.
[out]	p_remain_perd_adv_data_size	The free buffer size of Controller to which periodic advertising data can be currently set.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_remain_adv_data_size or p_remain_perd_adv_data_size is specified as NULL.

◆ R_BLE_GAP_RemoveAdvSet()

ble_status_t R_BLE_GAP_RemoveAdvSet (uint8_t *op_code*, uint8_t *adv_hdl*)

Delete advertising set.

This function deletes an advertising set or deletes all the advertising sets. The result of this API call is notified in BLE_GAP_EVENT_ADV_SET_REMOVE_COMP event.

Parameters

[in]	op_code	The operation for delete or clear. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_RMV_ADV_SET_REM_OP(0x01)</td><td>Delete an advertising set.</td></tr><tr><td>BLE_GAP_RMV_ADV_SET_CLR_OP(0x02)</td><td>Delete all the advertising sets.</td></tr></table>
[in]	adv_hdl	Advertising handle identifying the advertising set deleted. If op_code is BLE_GAP_RMV_ADV_SET_CLR_OP, adv_hdl is ignored.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • op_code is out of range. • When op_code is BLE_GAP_RMV_ADV_SET_REM_OP(0x01), adv_hdl is out of range.
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_CreateConn()

ble_status_t R_BLE_GAP_CreateConn (st_ble_gap_create_conn_param_t * *p_param*)

Request for a link establishment.

This function sends a connection request to a remote device to create a link. When Controller has received a request for establishment of a link from host stack, BLE_GAP_EVENT_CREATE_CONN_COMP event is notified to the application layer. When the link is established, BLE_GAP_EVENT_CONN_IND event is notified to the application layer.

Parameters

[in]	p_param	Connection parameters.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The reason for this error is as follows: • p_param is specified as NULL. • p_conn_param_1M field and p_conn_param_2M and p_conn_param_coded field in p_param are specified as NULL. • When creating a link with 1M PHY, p_conn_param in p_conn_param_1M field in p_param is specified as NULL. • When creating a link with 2M PHY, p_conn_param in p_conn_param_2M field in p_param is specified as NULL. • When creating a link with coded MPHY, p_conn_param in p_conn_param_coded field in p_param is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • init_filter_policy in p_param is out of range. • remote_bd_addr_type field or own_addr_type address field in p_param is out of range.
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_CancelCreateConn()

ble_status_t R_BLE_GAP_CancelCreateConn (void)

Cancel the request for a link establishment.

This function cancels a request for establishing a link. When Controller has received the cancel request from host stack, BLE_GAP_EVENT_CONN_CANCEL_COMP event is notified to the application layer. When the cancel procedure has completed, BLE_GAP_EVENT_CONN_IND event is notified to the application layer.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_SetChMap()

ble_status_t R_BLE_GAP_SetChMap (uint8_t * *p_channel_map*)

Set the Channel Map.

This function sets the channel map. The result of this API call is notified in BLE_GAP_EVENT_CH_MAP_SET_COMP event.

Parameters

[in]	p_channel_map	Channel map.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_channel_map is specified as NULL.
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_StartScan()

ble_status_t R_BLE_GAP_StartScan (st_ble_gap_scan_param_t * *p_scan_param*, st_ble_gap_scan_on_t * *p_scan_enable*)

Set scan parameter and start scan.

This function starts scanning. When scanning for the first time, set the p_scan_param. Setting scan parameters can be omitted by specifying p_scan_param as NULL after next time. The result of this API call is notified in BLE_GAP_EVENT_SCAN_ON event. Advertising report is notified in BLE_GAP_EVENT_ADV_REPT_IND event. Figure 1.3 shows the relationship between scan period, scan duration, scan interval and scan window.

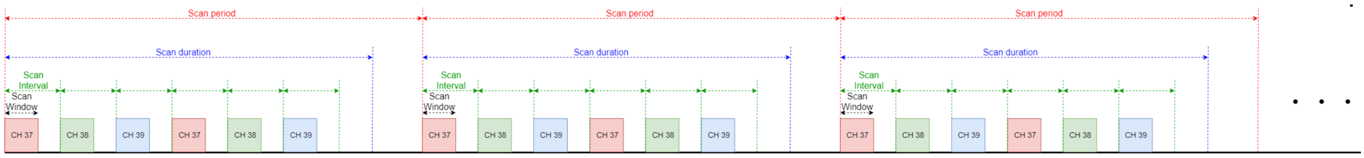


Figure 231: Figure 1.3

When scan duration is non-zero, scan period is zero and scan duration expires, BLE_GAP_EVENT_SCAN_TO event is notified to the application layer.

Parameters

[in]	p_scan_param	Scan parameter. When p_scan_param is specified as NULL, host stack doesn't set scan parameters and start scanning with the previous parameters.
[in]	p_scan_enable	Scan period, scan duration, duplicate filter and procedure type.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The reason for this error is as follows: • p_scan_enable is specified as NULL. • p_phy_param_1M field and p_phy_param_coded field in p_scan_param are specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • proc_type field in p_scan_enable is out of range. • filter_dups in p_scan_enable is out of range. • o_addr_type in p_scan_param is out of range. • filter_policy in p_scan_param is out of range. • scan_type of p_scan_param's p_phy_param_1M or p_phy_param_coded is out of range.
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_StopScan()

ble_status_t R_BLE_GAP_StopScan (void)

Stop scan.

This function stops scanning. The result of this API call is notified in BLE_GAP_EVENT_SCAN_OFF event.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_CreateSync()

ble_status_t R_BLE_GAP_CreateSync (st_ble_dev_addr_t * *p_addr*, uint8_t *adv_sid*, uint16_t *skip*, uint16_t *sync_to*)

Request for a periodic sync establishment.

This function sends a request for establishment of a periodic sync to a advertiser. In order to create a periodic sync, scan needs to be starting by R_BLE_GAP_StartScan(). When Controller has received the request from host stack, BLE_GAP_EVENT_CREATE_SYNC_COMP event is notified to the application layer. When the periodic sync is established, BLE_GAP_EVENT_SYNC_EST event is notified to the application layer.

Parameters

[in]	p_addr	The address of periodic advertiser.When p_addr is specified as NULL, local device creates a periodic sync with the advertiser registered in Periodic Advertiser List.
[in]	adv_sid	Advertising SID. When p_addr is specified as NULL, adv_sid is ignored. Valid range is 0x00 - 0x0F.
[in]	skip	The number of consecutive periodic advertising packets that local device may skip after receiving a periodic advertising packet. Valid range is 0x0000 - 0x01F3.

[in]	sync_to	The maximum permitted time between successful receives.When sync_to expires, the periodic sync is lost. Time(ms) = sync_to * 10. Valid range is 0x000A - 0x4000.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_addr is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The following parameter is out of range. • address type in p_addr • adv_sid • skip • sync_to
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_CancelCreateSync()

```
ble_status_t R_BLE_GAP_CancelCreateSync ( void  )
```

Cancel the request for a periodic sync establishment.

This function cancels a request for establishing a periodic sync. The result of this API call is notified in BLE_GAP_EVENT_SYNC_CREATE_CANCEL_COMP event.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_TerminateSync()

ble_status_t R_BLE_GAP_TerminateSync (uint16_t *sync_hdl*)

Terminate the periodic sync.

This function terminates a periodic sync. The result of this API call is notified in BLE_GAP_EVENT_SYNC_TERM event.

Parameters

[in]	sync_hdl	Sync handle identifying the periodic sync to be terminated.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	sync_hdl is out of range.
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_ConfPerdAdvList()

ble_status_t R_BLE_GAP_ConfPerdAdvList (uint8_t *op_code*, st_ble_dev_addr_t * *p_addr*, uint8_t * *p_adv_sid_set*, uint8_t *device_num*)

Set Periodic Advertiser List.

This function supports the following operations regarding Periodic Advertiser List.

- Add the device to Periodic Advertiser List.
- Delete the device from Periodic Advertiser List.
- Clear Periodic Advertiser List.

  The total number of Periodic Advertiser List entries is defined as BLE_GAP_PERD_LIST_MAX_ENTRY. The result of this API call is notified in BLE_GAP_EVENT_PERD_LIST_CONF_COMP event.

Parameters

[in]	op_code	The operation for Periodic Advertiser List. macro: description BLE_GAP_LIST_ADD_DEV(0x01): Add the device to the list. BLE_GAP_LIST_REM_DEV(0x02): Delete the device from the list. BLE_GAP_LIST_CLR(0x03): Clear the list.
[in]	p_addr	An array of device address to add / delete to the list. The number of elements is specified by device_num. If op_code is BLE_GAP_LIST_CLR, p_addr is ignored.
[in]	p_adv_sid_set	An array of SID of the advertiser to add / delete to the list. The number of elements is specified by device_num. If op_code is BLE_GAP_LIST_CLR, p_adv_sid_set is ignored.
[in]	device_num	The number of devices add / delete to the list.

			Valid range is 1-BLE_GAP_PERD_LIST_MAX_ENTRY. If op_code is BLE_GAP_LIST_CLR, device_num is ignored.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	When op_code is BLE_GAP_LIST_ADD_DEV or BLE_GAP_LIST_REM_DEV, p_addr or p_adv_sid_set is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	op_code or address type field in p_addr or p_adv_sid_set or device_num is out of range.
BLE_ERR_UNSUPPORTED(0x0007)	Not supported.
BLE_ERR_INVALID_STATE(0x0008)	The reason for this error is as follows: • While operating Periodic Advertiser List, this function was called. • The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for operating periodic advertiser.

◆ R_BLE_GAP_AuthorizeDev()

ble_status_t R_BLE_GAP_AuthorizeDev (uint16_t *conn_hdl*, uint8_t *author_flag*)

Authorize a remote device.

User authorizes a remote device by this function. This function is used when a remote device accesses a GATT Characteristic in local device which requests user authorization. The result of this API call is returned by a return value.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device to be authorized or not by user.
[in]	author_flag	Authorize or not the remote device. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_NOT_AUTHORIZED(0x00)</td><td>Not authorize the remote device.</td></tr><tr><td>BLE_GAP_AUTHORIZED(0x01)</td><td>Authorize the remote device.</td></tr></table>

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	author_flag is out of range.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.

◆ R_BLE_GAP_GetRemDevInfo()

ble_status_t R_BLE_GAP_GetRemDevInfo (uint16_t *conn_hdl*)

Get the information about remote device.

This function retrieves information about the remote device. The information includes BD_ADDR, the version number and LE features. The result of this API call is notified in BLE_GAP_EVENT_GET_REM_DEV_INFO event.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device whose information to be retrieved.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_SetPairingParams()

ble_status_t R_BLE_GAP_SetPairingParams (st_ble_gap_pairing_param_t * *p_pair_param*)

Set the parameters using pairing.

This function sets the parameters used in pairing. The parameters set by this API are sent to the remote device when pairing occurred. The result of this API call is returned by a return value.

Parameters

[in]	p_pair_param	Pairing parameters.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The following field in p_pair_param is out of range. • iocap • max_key_size • mitm • bonding • key_notf • sec_conn_only

◆ R_BLE_GAP_SetLocIdInfo()

ble_status_t R_BLE_GAP_SetLocIdInfo (st_ble_dev_addr_t * *p_lc_id_addr*, uint8_t * *p_lc_irk*)

Set the IRK and the identity address distributed to a remote device.

This function registers local IRK and identity address of local device in host stack. The IRK and the identity address are distributed to a remote device in pairing. The result of this API call is returned by a return value.

Parameters

[in]	p_lc_id_addr	Identity address to be registered in host stack.
[in]	p_lc_irk	IRK to be registered in host stack.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_lc_id_addr or p_lc_irk is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	Address type field in p_lc_id_addr is out of range.

◆ R_BLE_GAP_SetLocCsrk()

ble_status_t R_BLE_GAP_SetLocCsrk (uint8_t * *p_local_csrk*)

Set the CSRK distributed to a remote device.

This function registers local CSRK in host stack. The CSRK is distributed to a remote device in pairing. The result of this API call is returned by a return value.

Parameters

[in]	p_local_csrk	CSRK to be registered in host stack.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_local_csrk is specified as NULL.

◆ R_BLE_GAP_StartPairing()

ble_status_t R_BLE_GAP_StartPairing (uint16_t *conn_hdl*)

Start pairing.

This function starts pairing with a remote device. The result of this API call is returned by a return value. The result of pairing is notified in BLE_GAP_EVENT_PAIRING_COMP event.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device which local device starts pairing with.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	While generating OOB data, this function was called.
BLE_ERR_CONTEXT_FULL(0x000B)	While pairing, this function was called.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.

◆ R_BLE_GAP_ReplyPairing()

ble_status_t R_BLE_GAP_ReplyPairing (uint16_t *conn_hdl*, uint8_t *response*)

Reply the pairing request from a remote device.

This function replies to the pairing request from the remote device. The pairing request from the remote device is notified in BLE_GAP_EVENT_PAIRING_REQ event. The result of this API call is returned by a return value. The result of pairing is notified in BLE_GAP_EVENT_PAIRING_COMP event.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device which local device starts pairing with.
[in]	response	Accept or reject the pairing request from the remote device. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_PAIRING_ACCEPT(0x00)</td><td>Accept the pairing request.</td></tr><tr><td>BLE_GAP_PAIRING_REJECT(0x01)</td><td>Reject the pairing request.</td></tr></table>

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	response is out of range.
BLE_ERR_INVALID_STATE(0x0008)	While generating OOB data, this function was called.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.
BLE_ERR_NOT_YET_READY(0x0012)	When this function was called, host stack has not yet received BLE_GAP_EVENT_PAIRING_REQ event.

◆ R_BLE_GAP_StartEnc()

ble_status_t R_BLE_GAP_StartEnc (uint16_t *conn_hdl*)

Encryption the link.

This function starts encryption of the link. In case of master device, the local device requests for the encryption to a remote device. In case of slave device, the local device sends a Security Request to a remote device. After receiving the Security Request, the remote device requests for the encryption to the local device. The result of the encryption is returned in BLE_GAP_EVENT_ENC_CHG event.

Parameters

[in]	conn_hdl	Connection handle identifying the link which is encrypted.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The reason for this error is as follows: • Pairing has not been completed. • The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.

◆ R_BLE_GAP_ReplyPasskeyEntry()

ble_status_t R_BLE_GAP_ReplyPasskeyEntry (uint16_t *conn_hdl*, uint32_t *passkey*, uint8_t *response*)

Reply the passkey entry request.

When BLE_GAP_EVENT_PASSKEY_ENTRY_REQ event is notified, the response to passkey entry is sent by this function. The result of this API call is returned by a return value.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device which the reply to passkey entry is sent.
[in]	passkey	Passkey. The valid range is 000000 - 999999 in decimal.
[in]	response	Active or negative reply to passkey entry. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_PAIRING_ACCEPT(0x00)</td><td>Accept the passkey entry pairing.</td></tr><tr><td>BLE_GAP_PAIRING_REJECT(0x01)</td><td>Reject the passkey entry pairing.</td></tr></table>

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	passkey or response is out of range.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.
BLE_ERR_NOT_YET_READY(0x0012)	When this function was called, pairing has not yet started.

◆ R_BLE_GAP_ReplyNumComp()

ble_status_t R_BLE_GAP_ReplyNumComp (uint16_t *conn_hdl*, uint8_t *response*)

Reply the numeric comparison request.

When BLE_GAP_EVENT_NUM_COMP_REQ event is notified, the response to Numeric Comparison is sent by this function. The result of this API call is returned by a return value.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device which the reply to Numeric Comparison is sent.
[in]	response	Active or negative reply in Numeric Comparison. macro / description: BLE_GAP_PAIRING_ACCEPT(0x00): The number displayed in the local is the same as the one of the remote. BLE_GAP_PAIRING_REJECT(0x01): The number displayed in the local is differs from the one of the remote.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	response is out of range.
BLE_ERR_INVALID_STATE(0x0008)	When this function was called, host stack has not yet received BLE_GAP_EVENT_NUM_COMP_REQ event.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.
BLE_ERR_NOT_YET_READY(0x0012)	When this function was called, pairing has not yet started.

◆ R_BLE_GAP_NotifyKeyPress()

ble_status_t R_BLE_GAP_NotifyKeyPress (uint16_t *conn_hdl*, uint8_t *key_press*)

Notify the input key type which a remote device inputs in the passkey entry.

This function notifies the input key type to the remote device in passkey entry. The result is returned from this API.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device to which the key notification is sent.
[in]	key_press	Input key type. macro / description: BLE_GAP_LESC_PASSKEY_ENTRY_STARTED(0x00) — Notify that passkey entry started. BLE_GAP_LESC_PASSKEY_DIGIT_ENTERED(0x01) — Notify that passkey digit entered. BLE_GAP_LESC_PASSKEY_DIGIT_ERASED(0x02) — Notify that passkey digit erased. BLE_GAP_LESC_PASSKEY_CLEARED(0x03) — Notify that passkey cleared. BLE_GAP_LESC_PASSKEY_ENTRY_COMPLETED(0x04) — Notify that passkey entry completed.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	key_press parameter is out of range.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.
BLE_ERR_NOT_YET_READY(0x0012)	When this function was called, pairing has not yet started.

◆ R_BLE_GAP_GetDevSecInfo()

ble_status_t R_BLE_GAP_GetDevSecInfo (uint16_t *conn_hdl*, st_ble_gap_auth_info_t * *p_sec_info*)

Get the security information about the remote device.

This function gets the parameters which has been negotiated with the remote device in pairing. The parameters can be retrieved after pairing. The result is returned by p_sec_info.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device whose bonding information is retrieved.
[in]	p_sec_info	Return the security information which has been negotiated in pairing.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_sec_info is specified as NULL.
BLE_ERR_INVALID_STATE(0x0008)	The remote device bonding information has not been set to host stack.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.

◆ R_BLE_GAP_ReplyExKeyInfoReq()

ble_status_t R_BLE_GAP_ReplyExKeyInfoReq (uint16_t *conn_hdl*)

Distribute the keys of local device.

When key exchange request is notified by BLE_GAP_EVENT_EX_KEY_REQ event at pairing, keys of the local device are distributed. The result is returned from this API.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device to which the key is distributed.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.
BLE_ERR_NOT_YET_READY(0x0012)	When this function was called, pairing has not yet started.

◆ R_BLE_GAP_SetRemOobData()

ble_status_t R_BLE_GAP_SetRemOobData (st_ble_dev_addr_t * *p_addr*, uint8_t *oob_data_flag*, st_ble_gap_oob_data_t * *p_oob*)

Set the oob data from a remote device.

This function registers the OOB data received from a remote device. When oob_data_flag indicates that the OOB data has been received, the setting regarding OOB data is reflected in pairing. In order to do OOB pairing, set the OOB data received from the remote device before pairing. The result is returned from this API.

Parameters

[in]	p_addr	The remote device address.
[in]	oob_data_flag	This parameter indicates whether the local device has received the OOB data from the remote device or not. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_OOB_DATA_NOT_PRESENT(0x00)</td><td>Reply that No OOB data has been received when pairing.</td></tr><tr><td>BLE_GAP_OOB_DATA_PRESENT(0x01)</td><td>Reply that the OOB data has been received when pairing.</td></tr></table>
[in]	p_oob	The OOB data received from the remote device.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The reason for this error is as follows. • p_addr is specified as NULL. • oob_data_flag is BLE_GAP_OOB_DATA_PRESENT and p_oob is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	oob_data_flag is out of range.
BLE_ERR_CONTEXT_FULL(0x000B)	There is no room to register the OOB data received from a remote device.

◆ R_BLE_GAP_CreateScOobData()

ble_status_t R_BLE_GAP_CreateScOobData (void)

Create data for oob in secure connection.

This function generates the OOB data distributed to a remote device in Secure Connections. The result of this API call is notified in BLE_GAP_EVENT_SC_OOB_CREATE_COMP event.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The reason for this error is as follows: • This function was called in pairing. • The task for host stack is not running.
BLE_ERR_ALREADY_IN_PROGRESS(0x000A)	This function was called in creating OOB data.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.

◆ R_BLE_GAP_SetBondInfo()

ble_status_t R_BLE_GAP_SetBondInfo (st_ble_gap_bond_info_t * *p_bond_info*, uint8_t *device_num*, uint8_t * *p_set_num*)

Set the bonding information stored in non-volatile memory to the host stack.

Set the bonding information of the remote device in the host stack. After power re-supply, when the remote device bonding information stored in non-volatile memory is set to host stack, this function is used. Host stack can be set the number specified by the device_num parameter of bonding information.

Parameters

[in]	p_bond_info	An array of bonding information. The number of elements is specified by device_num.
[in]	device_num	The number of the devices of which host stack registers bonding information.
[in]	p_set_num	The number of the devices whose bonding information was registered in host stack.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_bond_info or p_set_num is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	device_num is out of range.
BLE_ERR_CONTEXT_FULL(0x000B)	Host stack already has the maximum number of bonding information.

◆ R_BLE_GAP_DeleteBondInfo()

void R_BLE_GAP_DeleteBondInfo (int32_t *local*, int32_t *remote*, st_ble_dev_addr_t * *p_addr*, ble_gap_del_bond_cb_t *gap_del_bond_cb*)

This function deletes the bonding information in Host Stack.
When a function for deleting the bonding information stored in non-volatile area is registered by the gap_del_bond_cb parameter, it is deleted as well as the bonding information in Host Stack.

Parameters

[in]	local	The type of the local bonding information to be deleted. macro / description: BLE_GAP_SEC_DEL_LOC_NONE(0x00) — Delete no local keys. BLE_GAP_SEC_DEL_LOC_IRK(0x01) — Delete local IRK and identity address. BLE_GAP_SEC_DEL_LOC_CSRK(0x02) — Delete local CSRK. BLE_GAP_SEC_DEL_LOC_ALL(0x03) — Delete all local keys.
[in]	remote	The type of the remote bonding information to be deleted. macro / description: BLE_GAP_SEC_DEL_REM_NONE(0x00) — Delete no remote device keys. BLE_GAP_SEC_DEL_REM_SA(0x01) — Delete the keys specified by the p_addr parameter. BLE_GAP_SEC_DEL_REM_NOT_CONN(0x02) — Delete keys of not connected remote devices. BLE_GAP_SEC_DEL_REM_ALL(0x03) — Delete all remote device keys.

[in]	p_addr	p_addr is specified as the address of the remote device whose keys are deleted when the rem_info parameter is set to BLE_GAP_SEC_DEL_REM_SA(0x01).
[in]	gap_del_bond_cb	This parameter is a callback function which deletes the bonding information stored in non-volatile area. After deleting the bonding information stored in Host Stack, the callback function is called. If no bonding information is stored in non-volatile area, specify the parameter as NULL.

Return values

none	

◆ R_BLE_GAP_ReplyLtkReq()

ble_status_t R_BLE_GAP_ReplyLtkReq (uint16_t *conn_hdl*, uint16_t *ediv*, uint8_t * *p_peer_rand*, uint8_t *response*)

Reply the LTK request from a remote device.

This function replies to the LTK request in BLE_GAP_EVENT_LTK_REQ event from a remote device. The result of the LTK reply is returned in BLE_GAP_EVENT_LTK_RSP_COMP event. When the link encryption has completed, BLE_GAP_EVENT_ENC_CHG event is notified.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device which sent the LTK request.
[in]	ediv	Ediv notified in BLE_GAP_EVENT_LTK_REQ event.
[in]	p_peer_rand	Rand notified in BLE_GAP_EVENT_LTK_REQ event.
[in]	response	Response to the LTK request. If

		"BLE_GAP_LTK_REQ_ACCEPT" is specified, when no LTK has been exchanged in pairing, reject the LTK request. macro / description BLE_GAP_LTK_REQ_ACCEPT(0x00): Reply for the LTK request. BLE_GAP_LTK_REQ_DENY(0x01): Reject the LTK request.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	p_peer_rand is specified as NULL in case of legacy pairing.
BLE_ERR_INVALID_ARG(0x0003)	response is out of range.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.

4.3.2.2 GATT_COMMON

Interfaces » BLE Interface

Functions

ble_status_t	R_BLE_GATT_GetMtu (uint16_t conn_hdl, uint16_t *p_mtu)
	This function gets the current MTU used in GATT communication. More...

Detailed Description

Function Documentation

◆ R_BLE_GATT_GetMtu()

ble_status_t R_BLE_GATT_GetMtu (uint16_t *conn_hdl*, uint16_t * *p_mtu*)

This function gets the current MTU used in GATT communication.

Both GATT server and GATT Client can use this function.
The result of this API call is returned by a return value.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server or the GATT Client.
[in]	p_mtu	The Current MTU. Before MTU exchange, this parameter is 23 bytes. After MTU exchange, this parameter is the negotiated MTU.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The mtu parameter is NULL.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server or the GATT Client specified by conn_hdl was not found.

4.3.2.3 GATT_SERVER

Interfaces » BLE Interface

Functions

ble_status_t	R_BLE_GATTS_Init (uint8_t cb_num) This function initializes the GATT Server and registers the number of the callbacks for GATT Server event. More...
ble_status_t	R_BLE_GATTS_SetDbInst (st_ble_gatts_db_cfg_t *p_db_inst) This function sets GATT Database to host stack. More...
ble_status_t	R_BLE_GATTS_RegisterCb (ble_gatts_app_cb_t cb, uint8_t priority) This function registers a callback for GATT Server event. More...
ble_status_t	R_BLE_GATTS_DeregisterCb (ble_gatts_app_cb_t cb)

	This function deregisters the callback function for GATT Server event. More...
ble_status_t	R_BLE_GATTS_Notification (uint16_t conn_hdl, st_ble_gatt_hdl_value_pair_t *p_ntf_data) This function sends a notification of an attribute's value. More...
ble_status_t	R_BLE_GATTS_Indication (uint16_t conn_hdl, st_ble_gatt_hdl_value_pair_t *p_ind_data) This function sends a indication of an attribute's value. More...
ble_status_t	R_BLE_GATTS_GetAttr (uint16_t conn_hdl, uint16_t attr_hdl, st_ble_gatt_value_t *p_value) This function gets a attribute value from the GATT Database. More...
ble_status_t	R_BLE_GATTS_SetAttr (uint16_t conn_hdl, uint16_t attr_hdl, st_ble_gatt_value_t *p_value) This function sets an attribute value to the GATT Database. More...
ble_status_t	R_BLE_GATTS_SendErrRsp (uint16_t error_code) This function sends an error response to a remote device. More...
ble_status_t	R_BLE_GATTS_RspExMtu (uint16_t conn_hdl, uint16_t mtu) This function replies to a MTU Exchange Request from a remote device. More...
ble_status_t	R_BLE_GATTS_SetPrepareQueue (st_ble_gatt_pre_queue_t *p_pre_queues, uint8_t queue_num) Register prepare queue and buffer in Host Stack. More...

Detailed Description

Data Structures

struct	st_ble_gatt_value_t Attribute Value. More...

struct	st_ble_gatt_hdl_value_pair_t Attribute handle and attribute Value. More...
struct	st_ble_gatt_queue_att_val_t Queued writes Attribute Value. More...
struct	st_ble_gatt_queue_pair_t Queued writes Attribute Value. More...
struct	st_ble_gatt_queue_elm_t Prepare Write Queue element for long chracteristic. More...
struct	st_ble_gatt_pre_queue_t Prepare Write Queue for long chracteristic. More...
struct	st_ble_gatts_db_params_t Attribute value to be set to or retrieved from the GATT Database and the access type from the GATT Client. More...
struct	st_ble_gatts_db_conn_hdl_t Information about the service or the characteristic that the attribute belongs to. More...
struct	st_ble_gatts_db_access_evt_t This structure notifies that the GATT Database has been accessed from a GATT Client. More...
struct	st_ble_gatts_conn_evt_t This structure notifies that the link with the GATT Client has been established. More...
struct	st_ble_gatts_disconn_evt_t This structure notifies that the link with the GATT Client has been disconnected. More...
struct	st_ble_gatts_ex_mtu_req_evt_t

	This structure notifies that a MTU Exchange Request PDU has been received from a GATT Client. More...
struct	st_ble_gatts_cfm_evt_t This structure notifies that a Confirmation PDU has been received from a GATT Client. More...
struct	st_ble_gatts_read_by_type_rsp_evt_t This structure notifies that a Read By Type Response PDU has been sent from GATT Server. More...
struct	st_ble_gatts_read_rsp_evt_t This structure notifies that a Read Response PDU has been sent from GATT Server. More...
struct	st_ble_gatts_read_blob_rsp_evt_t This structure notifies that a Read Blob Response PDU has been sent from GATT Server. More...
struct	st_ble_gatts_read_multi_rsp_evt_t This structure notifies that a Read Multiple Response PDU has been sent from GATT Server. More...
struct	st_ble_gatts_write_rsp_evt_t This structure notifies that a Write Response PDU has been sent from GATT Server. More...
struct	st_ble_gatts_prepare_write_rsp_evt_t This structure notifies that a Prepare Write Response PDU has been sent from GATT Server. More...
struct	st_ble_gatts_exe_write_rsp_evt_t This structure notifies that a Execute Write Response PDU has been sent from GATT Server. More...
struct	st_ble_gatts_db_uuid_cfg_t A structure that defines the information on the position where UUIDs

	are used. More...
struct	st_ble_gatts_db_attr_cfg_t A structure that defines the detailed information of the attributes. More...
struct	st_ble_gatts_db_attr_list_t The number of attributes are stored. More...
struct	st_ble_gatts_db_char_cfg_t A structure that defines the detailed information of the characteristics. More...
struct	st_ble_gatts_db_serv_cfg_t A structure that defines the detailed information of the characteristics. More...
struct	st_ble_gatts_db_cfg_t This is the structure of GATT Database that is specified in R_BLE_GATTS_SetDbInst(). More...
struct	st_ble_gatts_evt_data_t st_ble_gatts_evt_data_t is the type of the data notified in a GATT Server Event. More...

Macros

#define	BLE_GATT_DEFAULT_MTU GATT Default MTU.
#define	BLE_GATT_16_BIT_UUID_FORMAT GATT Identification for 16-bit UUID Format.
#define	BLE_GATT_128_BIT_UUID_FORMAT GATT Identification for 128-bit UUID Format.
#define	BLE_GATT_16_BIT_UUID_SIZE

GATT 16-bit UUID Size.

#define BLE_GATT_128_BIT_UUID_SIZE

GATT 128-bit UUID Size.

#define BLE_GATT_INVALID_ATTR_HDL_VAL

GATT Invalid Attribute Handle Value.

#define BLE_GATT_ATTR_HDL_START_RANGE

GATT Attribute Handle Start Range.

#define BLE_GATT_ATTR_HDL_END_RANGE

GATT Attribute Handle End Range.

#define BLE_GATTS_CLI_CNFG_NOTIFICATION

GATT Client Configuration values. Enable Notification.

#define BLE_GATTS_CLI_CNFG_INDICATION

GATT Client Configuration values. Enable Indication.

#define BLE_GATTS_CLI_CNFG_DEFAULT

GATT Client Configuration values. Default value or disable notification/indication.

#define BLE_GATTS_SER_CNFG_BROADCAST

GATT Server Configuration values. Enable broadcast.

#define BLE_GATTS_SER_CNFG_DEFAULT

GATT Server Configuration values. Default value.

#define BLE_GATTS_MAX_CB

GATT Server Callback Number.

#define BLE_GATTS_OP_CHAR_VALUE_READ_REQ

	Characteristic Value Local Read Operation.
#define	BLE_GATTS_OP_CHAR_VALUE_WRITE_REQ Characteristic Value Local Write Operation.
#define	BLE_GATTS_OP_CHAR_VALUE_WRITE_WITHOUT_REQ Characteristic Value Local Write Without Response Operation.
#define	BLE_GATTS_OP_CHAR_CLI_CNFG_READ_REQ Characteristic Client Configuration Local Read Operation.
#define	BLE_GATTS_OP_CHAR_CLI_CNFG_WRITE_REQ Characteristic Client Configuration Local Write Operation.
#define	BLE_GATTS_OP_CHAR_SER_CNFG_READ_REQ Characteristic Server Configuration Local Read Operation.
#define	BLE_GATTS_OP_CHAR_SER_CNFG_WRITE_REQ Characteristic Server Configuration Local Write Operation.
#define	BLE_GATTS_OP_CHAR_PEER_READ_REQ Characteristic Value Peer Read Operation.
#define	BLE_GATTS_OP_CHAR_PEER_WRITE_REQ Characteristic Value Peer Write Operation.
#define	BLE_GATTS_OP_CHAR_PEER_WRITE_CMD Characteristic Value Peer Write Command.
#define	BLE_GATTS_OP_CHAR_PEER_CLI_CNFG_READ_REQ Characteristic Client Configuration Peer Read Operation.
#define	BLE_GATTS_OP_CHAR_PEER_CLI_CNFG_WRITE_REQ Characteristic Client Configuration Peer Write Operation.

#define	BLE_GATTS_OP_CHAR_PEER_SER_CNFG_READ_REQ Characteristic Server Configuration Peer Read Operation.
#define	BLE_GATTS_OP_CHAR_PEER_SER_CNFG_WRITE_REQ Characteristic Server Configuration Peer Write Operation.
#define	BLE_GATTS_OP_CHAR_PEER_USR_DESC_READ_REQ Characteristic User Description Peer Read Operation.
#define	BLE_GATTS_OP_CHAR_PEER_USR_DESC_WRITE_REQ Characteristic User Description Peer Write Operation.
#define	BLE_GATTS_OP_CHAR_PEER_HLD_DESC_READ_REQ Characteristic Higher Layer Defined Descriptor Peer Read Operation.
#define	BLE_GATTS_OP_CHAR_PEER_HLD_DESC_WRITE_REQ Characteristic Higher Layer Defined Descriptor Peer Write Operation.
#define	BLE_GATTS_OP_CHAR_REQ_AUTHOR Operation Required Authorization.
#define	BLE_GATT_DB_READ Allow clients to read.
#define	BLE_GATT_DB_WRITE Allow clients to write.
#define	BLE_GATT_DB_WRITE_WITHOUT_RSP Allow clients to write without response.
#define	BLE_GATT_DB_READ_WRITE Allow clients to access of all.

#define	BLE_GATT_DB_NO_AUXILIARY_PROPERTY No auxiliary properties.
#define	BLE_GATT_DB_FIXED_LENGTH_PROPERTY Fixed length attribute value.
#define	BLE_GATT_DB_AUTHORIZATION_PROPERTY Attributes requiring authorization.
#define	BLE_GATT_DB_ATTR_DISABLED The attribute is disabled. If this value is set, the attribute cannot be found and accessed by a GATT Client.
#define	BLE_GATT_DB_128_BIT_UUID_FORMAT Attribute with 128 bit UUID.
#define	BLE_GATT_DB_PEER_SPECIFIC_VAL_PROPERTY Attribute managed by each GATT Client.
#define	BLE_GATT_DB_CONST_ATTR_VAL_PROPERTY Fixed attribute value.
#define	BLE_GATT_DB_SER_SECURITY_UNAUTH Unauthenticated pairing(Security Mode1 Security Level 2, Security Mode 2 Security Level 1). Unauthenticated pairing is required to access the service.
#define	BLE_GATT_DB_SER_SECURITY_AUTH Authenticated pairing(Security Mode1 Security Level 3, Security Mode 2 Security Level 2). Authenticated pairing is required to access the service.
#define	BLE_GATT_DB_SER_SECURITY_SECONN Authenticated LE secure connections that generates 16bytes LTK(Security Mode1 Security Level 4). Authenticated LE secure connections pairing that generates 16bytes LTK is required to access the service. If this bit is set, bit24-27 are ignored.

#define	BLE_GATT_DB_SER_SECURITY_ENC
	Encryption. Encryption by the LTK exchanged in pairing is required to access.
#define	BLE_GATT_DB_SER_NO_SECURITY_PROPERTY
	No Security(Security Mode1 Security Level 1).
#define	BLE_GATT_DB_SER_ENC_KEY_SIZE_DONT_CARE
	7-byte or larger encryption key.
#define	BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_7
	7-byte encryption key.
#define	BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_8
	8-byte encryption key.
#define	BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_9
	9-byte encryption key.
#define	BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_10
	10-byte encryption key.
#define	BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_11
	11-byte encryption key.
#define	BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_12
	12-byte encryption key.
#define	BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_13
	13-byte encryption key.
#define	BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_14
	14-byte encryption key.

#define	BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_15 15-byte encryption key.
#define	BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_16 16-byte encryption key.

Typedefs

typedef void(*	ble_gatts_app_cb_t) (uint16_t event_type, ble_status_t event_result, st_ble_gatts_evt_data_t *p_event_data) ble_gatts_app_cb_t is the GATT Server Event callback function type. More...

Enumerations

enum	e_r_ble_gatts_evt_t GATT Server Event Identifier. More...

Data Structure Documentation

◆ st_ble_gatt_value_t

struct st_ble_gatt_value_t

Attribute Value.

Data Fields		
uint16_t	value_len	Length of the attribute value.
uint8_t *	p_value	Attribute Value.

◆ st_ble_gatt_hdl_value_pair_t

struct st_ble_gatt_hdl_value_pair_t

Attribute handle and attribute Value.

Data Fields		
uint16_t	attr_hdl	Attribute Handle.
st_ble_gatt_value_t	value	Attribute Value.

◆ st_ble_gatt_queue_att_val_t

struct st_ble_gatt_queue_att_val_t

Queued writes Attribute Value.

Data Fields

uint8_t *	p_value	Attribute Value for Queued Write .
uint16_t	value_len	Length of the attribute value.
uint16_t	padding	padding.

◆ st_ble_gatt_queue_pair_t

struct st_ble_gatt_queue_pair_t		
Queued writes Attribute Value.		
Data Fields		
st_ble_gatt_queue_att_val_t	queue_value	Attribute Value for Queued Write.
uint16_t	attr_hdl	Attribute Handle.

◆ st_ble_gatt_queue_elm_t

struct st_ble_gatt_queue_elm_t		
Prepare Write Queue element for long chracteristic.		
Data Fields		
st_ble_gatt_queue_pair_t	queue_value_pair	Part of Long Characteristic Value and Characteristic Value Handle.
uint16_t	offset	Offset that indicates the location to be written.

◆ st_ble_gatt_pre_queue_t

struct st_ble_gatt_pre_queue_t		
Prepare Write Queue for long chracteristic.		
Data Fields		
uint8_t *	p_buf_start	Buffer start address for Write Long Characteristic Request.
st_ble_gatt_queue_elm_t *	p_queue	Prepare Write Queue for Long Characteristic Value.
uint16_t	buffer_len	Buffer length.
uint16_t	conn_hdl	Connection Handle.
uint16_t	buf_offset	Current buffer offset.
uint8_t	queue_size	Number of elements in the prepare write queue.
uint8_t	queue_idx	Index of Prepare Write Queue.

◆ st_ble_gatts_db_params_t

struct st_ble_gatts_db_params_t

Attribute value to be set to or retrieved from the GATT Database and the access type from the GATT Client.		
Data Fields		
st_ble_gatt_value_t	value	Attribute value to be set to or retrieved from the GATT Database. Note that the address of the value field in the value field is invalid in case of read access.
uint16_t	attr_hdl	Attribute handle identifying the attribute to be set or retrieved.
uint8_t	db_op	Type of the access to GATT Database from the GATT Client. See also access_type_to_gatt_database

◆ st_ble_gatts_db_conn_hdl_t

struct st_ble_gatts_db_conn_hdl_t		
Information about the service or the characteristic that the attribute belongs to.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the GATT Client that accesses to the GATT DataBase.
uint8_t	service_id	ID of the service that the attribute belongs to.
uint8_t	char_id	ID of the Characteristic that the attribute belongs to.

◆ st_ble_gatts_db_access_evt_t

struct st_ble_gatts_db_access_evt_t		
This structure notifies that the GATT Database has been accessed from a GATT Client.		
Data Fields		
st_ble_gatts_db_conn_hdl_t *	p_handle	Information about the service or the characteristic that the attribute belongs to.
st_ble_gatts_db_params_t *	p_params	Attribute value to be set to or retrieved from the GATT Database and the access type from the GATT Client.

◆ st_ble_gatts_conn_evt_t

struct st_ble_gatts_conn_evt_t

This structure notifies that the link with the GATT Client has been established.		
Data Fields		
st_ble_dev_addr_t *	p_addr	Address of the GATT Client.

◆ st_ble_gatts_disconn_evt_t

struct st_ble_gatts_disconn_evt_t		
This structure notifies that the link with the GATT Client has been disconnected.		
Data Fields		
st_ble_dev_addr_t *	p_addr	Address of the GATT Client.

◆ st_ble_gatts_ex_mtu_req_evt_t

struct st_ble_gatts_ex_mtu_req_evt_t		
This structure notifies that a MTU Exchange Request PDU has been received from a GATT Client.		
Data Fields		
uint16_t	mtu	Maximum receive MTU size by GATT Client.

◆ st_ble_gatts_cfm_evt_t

struct st_ble_gatts_cfm_evt_t		
This structure notifies that a Confirmation PDU has been received from a GATT Client.		
Data Fields		
uint16_t	attr_hdl	Attribute handle identifying the Characteristic sent by the Indication PDU.

◆ st_ble_gatts_read_by_type_rsp_evt_t

struct st_ble_gatts_read_by_type_rsp_evt_t		
This structure notifies that a Read By Type Response PDU has been sent from GATT Server.		
Data Fields		
uint16_t	attr_hdl	Attribute handle identifying the Characteristic read by the Read By Type Request PDU.

◆ st_ble_gatts_read_rsp_evt_t

struct st_ble_gatts_read_rsp_evt_t		
This structure notifies that a Read Response PDU has been sent from GATT Server.		
Data Fields		
uint16_t	attr_hdl	Attribute handle identifying the Characteristic read by the Read Request PDU.

◆ st_ble_gatts_read_blob_rsp_evt_t

struct st_ble_gatts_read_blob_rsp_evt_t		
This structure notifies that a Read Blob Response PDU has been sent from GATT Server.		
Data Fields		
uint16_t	attr_hdl	Attribute handle identifying the Characteristic read by the Read Blob Request PDU.

◆ st_ble_gatts_read_multi_rsp_evt_t

struct st_ble_gatts_read_multi_rsp_evt_t		
This structure notifies that a Read Multiple Response PDU has been sent from GATT Server.		
Data Fields		
uint8_t	count	The number of attribute read by the Read Multiple Request PDU.
uint16_t *	p_attr_hdl_list	The list of attribute read by the Read Multiple Request PDU.

◆ st_ble_gatts_write_rsp_evt_t

struct st_ble_gatts_write_rsp_evt_t		
This structure notifies that a Write Response PDU has been sent from GATT Server.		
Data Fields		
uint16_t	attr_hdl	Attribute handle identifying the Characteristic written by the Write Request PDU.

◆ st_ble_gatts_prepare_write_rsp_evt_t

struct st_ble_gatts_prepare_write_rsp_evt_t		
This structure notifies that a Prepare Write Response PDU has been sent from GATT Server.		
Data Fields		
uint16_t	attr_hdl	Attribute handle identifying the Characteristic written by the Prepare Write Request PDU.
uint16_t	length	The length of written bytes by the Prepare Write Request PDU.
uint16_t	offset	The offset of the first octet to be written.

◆ st_ble_gatts_exe_write_rsp_evt_t

struct st_ble_gatts_exe_write_rsp_evt_t
This structure notifies that a Execute Write Response PDU has been sent from GATT Server.
Data Fields

uint8_t	exe_flag	The flag that indicates whether execution or cancellation. value / description 0x00 — Cancellation. 0x01 — Execution.

◆ st_ble_gatts_db_uuid_cfg_t

struct st_ble_gatts_db_uuid_cfg_t		
A structure that defines the information on the position where UUIDs are used.		
Data Fields		
uint16_t	offset	The position of the defined UUID is specified by offset value in uuid_table of st_ble_gatts_db_cfg_t.
uint16_t	first	The attribute handle that indicates the first position in st_ble_gatts_db_attr_cfg_t for the defined UUID is specified.
uint16_t	last	The attribute handle that indicates the last position in st_ble_gatts_db_attr_cfg_t for the defined UUID is specified.

◆ st_ble_gatts_db_attr_cfg_t

struct st_ble_gatts_db_attr_cfg_t		
A structure that defines the detailed information of the attributes.		
Data Fields		
uint8_t	desc_prop	The properties of attribute are specified. Set the following properties by a bitwise OR. macro / description BLE_GATT_DB_READ(0x01) — Allow clients to read. BLE_GATT_DB_WRITE(0x02) — Allow clients to write. BLE_GATT_DB_WRITE_WITHOUT_RSP(0x04) — Allow clients to write. BLE_GATT_DB — Allow clients

<table>
<tr><td></td><td></td><td><table><tr><td>_READ_WRITE (0x07)</td><td>to access of all.</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>aux_prop</td><td>The auxiliary properties of attribute are specified.

Set the following properties by a bitwise OR.
<table>
<tr><th>macro</th><th>description</th></tr>
<tr><td>BLE_GATT_DB_NO_AUXILIARY_PROPERTY(0x00)</td><td>No auxiliary properties. It is invalid when used with other properties at the same time.</td></tr>
<tr><td>BLE_GATT_DB_FIXED_LENGTH_PROPERTY(0x01)</td><td>Fixed length attribute value.</td></tr>
<tr><td>BLE_GATT_DB_AUTHORIZATION_PROPERTY(0x02)</td><td>Attributes requiring authorization.</td></tr>
<tr><td>BLE_GATT_DB_ATTR_DISABLED(0x10)</td><td>The attribute is disabled. If this value is set, the attribute cannot be found and accessed by a GATT Client. It is invalid when used with other properties at the same time.</td></tr>
<tr><td>BLE_GATT_DB_128_BIT_UUID_FORMAT(0x20)</td><td>Attribute with 128 bit UUID. If this macro is not set, the attribute value is 16-bits UUID.</td></tr>
<tr><td>BLE_GATT_DB_PEER_SPECIFIC_VAL_PROPE</td><td>Attribute managed by each GATT</td></tr>
</table></td></tr>
</table>

		RTY(0x40) Client. BLE_GATT_DB_CONST_ATTR_VAL_PROPERTY(0x80) Fixed attribute value. Writing from Client and setting from Server are prohibited.
uint16_t	length	The length of the attribute value is specified.
uint16_t	next	The position of the next attribute with the same UUID as the defined attribute is specified by an attribute handle.
uint16_t	uuid_offset	The storage area of attribute value. UUID of the defined attribute is set by specifying the position of the UUID registered in uuid_table of st_ble_gatts_db_cfg_t with the array offset value.
uint8_t *	p_data_offset	Storage area of attribute value. The address in the array registered in No.1-No.4 is specified to set the attribute value storage area of the defined attribute.

◆ st_ble_gatts_db_attr_list_t

struct st_ble_gatts_db_attr_list_t		
The number of attributes are stored.		
Data Fields		
uint8_t	count	The number of the services or the characteristics.

◆ st_ble_gatts_db_char_cfg_t

struct st_ble_gatts_db_char_cfg_t		
A structure that defines the detailed information of the characteristics.		
Data Fields		
st_ble_gatts_db_attr_list_t	list	The total number of attributes in the defined characteristic is

		specified.
uint16_t	start_hdl	The first attribute handle of the characteristic is specified.
uint8_t	service_id	The index of service to which the characteristic belongs is specified.

◆ st_ble_gatts_db_serv_cfg_t

<table>
<tr><td colspan="3">struct st_ble_gatts_db_serv_cfg_t</td></tr>
<tr><td colspan="3">A structure that defines the detailed information of the characteristics.</td></tr>
<tr><td colspan="3">Data Fields</td></tr>
<tr><td>st_ble_gatts_db_attr_list_t</td><td>list</td><td>The total number of service declarations in the defined service is specified.</td></tr>
<tr><td>uint32_t</td><td>desc</td><td>The properties of the defined service are specified.

Set the security level, the security mode and the key size with a bitwise OR. The bit0-bit3 are specified as the security level. Select one of the following.

<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GATT_DB_SER_SECURITY_UNAUTH(0x00000001)</td><td>Unauthenticated pairing(Security Mode1 Security Level 2, Security Mode 2 Security Level 1) Unauthenticated pairing is required to access the service.</td></tr><tr><td>BLE_GATT_DB_SER_SECURITY_AUTH(0x00000002)</td><td>Authenticated pairing(Security Mode1 Security Level 3, Security Mode 2 Security Level 2) Authenticated pairing is required to</td></tr></table></td></tr>
</table>

		access the service. BLE_GATT_DB_SER_SECURITY_SECONN(0x00000004) — Authenticated LE secure connections that generates 16bytes LTK(Security Mode1 Security Level 4) Authenticated LE secure connections pairing that generates 16bytes LTK is required to access the service. If this bit is set, bit24-27 are ignored. The bit4 is specified as the security mode. macro: BLE_GATT_DB_SER_SECURITY_ENC(0x00000010) — description: Encryption Encryption by the LTK exchanged in pairing is required to access. If the security requirement of the service is not needed, specify the bit0-bit4 to BLE_GATT_DB_SER_NO_SECURITY_PROPERTY(0x00000000) .(Security Mode1 Security Level 1) The bit24-bit27 are specified as the key size required by the defined service. Select one of the following. macro: BLE_GATT_DB_SER_ENCRYP — description: 7-byte encryption

		BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_7(0x01000000)	key.
		BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_8(0x02000000)	8-byte encryption key.
		BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_9(0x03000000)	9-byte encryption key.
		BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_10(0x04000000)	10-byte encryption key.
		BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_11(0x05000000)	11-byte encryption key.
		BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_12(0x06000000)	12-byte encryption key.
		BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_13(0x07000000)	13-byte encryption key.
		BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_14(0x08000000)	14-byte encryption key.
		BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_15(0x09000000)	15-byte encryption key.
		BLE_GATT_DB_SER_ENCRYPT_KEY_SIZE_16(0x0A000000)	16-byte encryption key.
		BLE_GATT_DB_SER_ENC_KEY_SIZE_DONT_CARE(0x0000	7-byte or larger encryption key.

		0000) Other bits are reserved.
uint16_t	start_hdl	The start attribute handle of the defined service is specified.
uint16_t	end_hdl	The end attribute handle of the defined service is specified.
uint8_t	char_start_idx	The start index of the characteristic that belongs to the defined service is specified.
uint8_t	char_end_idx	The end index of the characteristic that belongs to the defined service is specified.

◆ st_ble_gatts_db_cfg_t

struct st_ble_gatts_db_cfg_t		
This is the structure of GATT Database that is specified in R_BLE_GATTS_SetDbInst().		
Data Fields		
const uint8_t *	p_uuid_table	The array to register the UUID to be used.
uint8_t *	p_attr_val_table	The array to register variable attribute values.
const uint8_t *	p_const_attr_val_table	The array to register fixed attribute values.
uint8_t *	p_rem_spec_val_table	The array to manage the attribute values handled for each GATT client.
const uint8_t *	p_const_rem_spec_val_table	The array to register the default of the attribute value handled by each GATT client.
const st_ble_gatts_db_uuid_cfg_t *	p_uuid_cfg	The array to register information on the position where UUIDs are used.
const st_ble_gatts_db_attr_cfg_t *	p_attr_cfg	The array to register the detailed information of attributes.
const st_ble_gatts_db_char_cfg_t *	p_char_cfg	The array to register the detailed information of characteristics.
const st_ble_gatts_db_serv_cfg_t *	p_serv_cfg	The array to register the detailed information of services.
uint8_t	serv_cnt	The number of services included in the GATT Database.

uint8_t	char_cnt	The number of characteristics included in the GATT Database.
uint8_t	uuid_type_cnt	The number of UUIDs included in the GATT Database.
uint8_t	peer_spec_val_cnt	The total size of attribute value that needs to be managed for each GATT client.

◆ **st_ble_gatts_evt_data_t**

struct st_ble_gatts_evt_data_t		
st_ble_gatts_evt_data_t is the type of the data notified in a GATT Server Event.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the GATT Client.
uint16_t	param_len	The size of GATT Server Event parameters.
void *	p_param	GATT Server Event parameters. This parameter differs in each GATT Server Event.

Typedef Documentation

◆ **ble_gatts_app_cb_t**

ble_gatts_app_cb_t

ble_gatts_app_cb_t is the GATT Server Event callback function type.

Parameters

[in]	event_type	The type of GATT Server Event.
[in]	event_result	The result of GATT Server Event
[in]	p_event_data	Data notified by GATT Server Event.

Returns
none

Enumeration Type Documentation

◆ **e_r_ble_gatts_evt_t**

enum e_r_ble_gatts_evt_t	
GATT Server Event Identifier.	
Enumerator	
BLE_GATTS_EVENT_EX_MTU_REQ	MTU Exchange Request has been received. This event notifies the application layer that a MTU Exchange Request PDU has been received from a GATT Client. Need to reply to the request by R_BLE_GATTS_RspExMtu(). **Event Code: 0x3002** **Event Data:** st_ble_gatts_ex_mtu_req_evt_tBLE_GATTS_EVENT_EX_MTU_REQ
BLE_GATTS_EVENT_READ_BY_TYPE_RSP_COMP	Read By Type Response has been sent. This event notifies the application layer that a Read By Type Response PDU has been sent from GATT Server to the GATT Client. **Event Code: 0x3009** **Event Data:** st_ble_gatts_read_by_type_rsp_evt_tBLE_GATTS_EVENT_READ_BY_TYPE_RSP_COMP
BLE_GATTS_EVENT_READ_RSP_COMP	Read Response has been sent. This event notifies the application layer that a Read Response PDU has been sent from GATT Server to the GATT Client. **Event Code: 0x300B** **Event Data:** st_ble_gatts_read_rsp_evt_tBLE_GATTS_EVENT_READ_RSP_COMP
BLE_GATTS_EVENT_READ_BLOB_RSP_COMP	Read Blob Response has been sent. This event notifies the application layer that a Read Blob Response PDU has been sent from GATT Server to the GATT Client. **Event Code: 0x300D** **Event Data:**

	st_ble_gatts_read_blob_rsp_evt_tBLE_GATTS_EVENT_READ_BLOB_RSP_COMP
BLE_GATTS_EVENT_READ_MULTI_RSP_COMP	Read Multiple Response has been sent. This event notifies the application layer that a Read Multiple Response PDU has been sent from GATT Server to the GATT Client. **Event Code: 0x300F** **Event Data:** st_ble_gatts_read_multi_rsp_evt_tBLE_GATTS_EVENT_READ_MULTI_RSP_COMP
BLE_GATTS_EVENT_WRITE_RSP_COMP	Write Response has been sent. This event notifies the application layer that a Write Response PDU has been sent from GATT Server to the GATT Client. **Event Code: 0x3013** **Event Data:** st_ble_gatts_write_rsp_evt_tBLE_GATTS_EVENT_WRITE_RSP_COMP
BLE_GATTS_EVENT_PREPARE_WRITE_RSP_COMP	Prepare Write Response has been sent. This event notifies the application layer that a Prepare Write Response PDU has been sent from GATT Server to the GATT Client. **Event Code: 0x3017** **Event Data:** st_ble_gatts_prepare_write_rsp_evt_tBLE_GATTS_EVENT_PREPARE_WRITE_RSP_COMP
BLE_GATTS_EVENT_EXE_WRITE_RSP_COMP	Execute Write Response has been sent. This event notifies the application layer that a Execute Write Response PDU has been sent from GATT Server to the GATT Client. **Event Code: 0x3019** **Event Data:** st_ble_gatts_exe_write_rsp_evt_tBLE_GATTS_EVENT_EXE_WRITE_RSP_COMP
BLE_GATTS_EVENT_HDL_VAL_CNF	Confirmation has been received. This event notifies the application layer that a

	Confirmation PDU has been received from a GATT Client. **Event Code: 0x301E** **Event Data:** st_ble_gatts_cfm_evt_tBLE_GATTS_EVENT_HDL_VAL_CNF
BLE_GATTS_EVENT_DB_ACCESS_IND	The GATT Database has been accessed from a GATT Client. This event notifies the application layer that the GATT Database has been accessed from a GATT Client. **Event Code: 0x3040** **Event Data:** st_ble_gatts_db_access_evt_tBLE_GATTS_EVENT_DB_ACCESS_IND
BLE_GATTS_EVENT_CONN_IND	A connection has been established. This event notifies the application layer that the link with the GATT Client has been established. **Event Code: 0x3081** **Event Data:** st_ble_gatts_conn_evt_tBLE_GATTS_EVENT_CONN_IND
BLE_GATTS_EVENT_DISCONN_IND	A connection has been disconnected. This event notifies the application layer that the link with the GATT Client has been disconnected. **Event Code: 0x3082** **Event Data:** st_ble_gatts_disconn_evt_tBLE_GATTS_EVENT_DISCONN_IND
BLE_GATTS_EVENT_INVALID	Invalid GATT Server Event. **Event Code: 0x30FF** **Event Data:** noneBLE_GATTS_EVENT_INVALID

Function Documentation

◆ R_BLE_GATTS_Init()

ble_status_t R_BLE_GATTS_Init (uint8_t *cb_num*)

This function initializes the GATT Server and registers the number of the callbacks for GATT Server event.

Specify the cb_num parameter to a value between 1 and BLE_GATTS_MAX_CB.
R_BLE_GATTS_RegisterCb() registers the callback.
The result of this API call is returned by a return value.

Parameters

[in]	cb_num	The number of callbacks to be registered.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The cb_num parameter is out of range.

◆ R_BLE_GATTS_SetDbInst()

ble_status_t R_BLE_GATTS_SetDbInst (st_ble_gatts_db_cfg_t * *p_db_inst*)

This function sets GATT Database to host stack.

The result of this API call is returned by a return value.

Parameters

[in]	p_db_inst	GATT Database to be set.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The reason for this error is as follows. • The db_inst parameter is specified as NULL. • The array in the db_inst is specified as NULL.

◆ R_BLE_GATTS_RegisterCb()

ble_status_t R_BLE_GATTS_RegisterCb (ble_gatts_app_cb_t *cb*, uint8_t *priority*)

This function registers a callback for GATT Server event.

The number of the callback that may be registered by this function is the value specified by R_BLE_GATTS_Init().
The result of this API call is returned by a return value.

Parameters

[in]	cb	Callback function for GATT Server event.
[in]	priority	The priority of the callback function. Valid range is 1 <= priority <= BLE_GATTS_MAX_CB. A lower priority number means a higher priority level.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The cb parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The priority parameter is out of range.
BLE_ERR_CONTEXT_FULL(0x000B)	Host stack has already registered the maximum number of callbacks.

◆ R_BLE_GATTS_DeregisterCb()

ble_status_t R_BLE_GATTS_DeregisterCb (ble_gatts_app_cb_t *cb*)

This function deregisters the callback function for GATT Server event.

The result of this API call is returned by a return value.

Parameters

[in]	cb	The callback function to be deregistered.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The cb parameter is specified as NULL.
BLE_ERR_NOT_FOUND(0x000D)	The callback has not been registered.

◆ R_BLE_GATTS_Notification()

ble_status_t R_BLE_GATTS_Notification (uint16_t *conn_hdl*, st_ble_gatt_hdl_value_pair_t * *p_ntf_data*)

This function sends a notification of an attribute's value.

The maximum length of the attribute value that can be sent with notification is MTU-3.
The result of this API call is returned by a return value.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device to be sent the notification.
[in]	p_ntf_data	The attribute value to send.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_ntf_data parameter or the value field in the value field in the p_ntf_data parameter is NULL.
BLE_ERR_INVALID_ARG(0x0003)	The value_len field in the value field in the p_ntf_data parameter is 0 or the attr_hdl field in the p_ntf_data parameters is 0.
BLE_ERR_INVALID_OPERATION(0x0009)	This function was called while processing other request.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl was not found.

◆ R_BLE_GATTS_Indication()

ble_status_t R_BLE_GATTS_Indication (uint16_t *conn_hdl*, st_ble_gatt_hdl_value_pair_t * *p_ind_data*)

This function sends a indication of an attribute's value.

The maximum length of the attribute value that can be sent with indication is MTU-3.
The result of this API call is returned by a return value.
The remote device that receives a indication sends a confirmation.
BLE_GATTS_EVENT_HDL_VAL_CNF event notifies the application layer that the confirmation has been received.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device to be sent the indication.
[in]	p_ind_data	The attribute value to send.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_ind_data parameter or the value field in the value field in the p_ind_data parameter is NULL.
BLE_ERR_INVALID_ARG(0x0003)	The value_len field in the value field in the p_ind_data parameter is 0 or the attr_hdl field in the p_ind_data parameters is 0.
BLE_ERR_INVALID_OPERATION(0x0009)	This function was called while processing other request.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl was not found.

◆ R_BLE_GATTS_GetAttr()

ble_status_t R_BLE_GATTS_GetAttr (uint16_t *conn_hdl*, uint16_t *attr_hdl*, st_ble_gatt_value_t * *p_value*)

This function gets a attribute value from the GATT Database.

The result of this API call is returned by a return value.

Parameters

[in]	conn_hdl	If the attribute value that has information about the remote device is retrieved, specify the remote device with the conn_hdl parameter. When information about the remote device is not required, set the conn_hdl parameter to BLE_GAP_INVALID_CONN_HDL.
[in]	attr_hdl	The attribute handle of the attribute value to be retrieved.
[out]	p_value	The attribute value to be retrieved.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_value parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The attr_hdl parameter is 0 or larger than the last attribute handle of GATT Database.
BLE_ERR_INVALID_STATE(0x0008)	The attribute is not in a state to be read.
BLE_ERR_INVALID_OPERATION(0x0009)	The attribute cannot be read.
BLE_ERR_NOT_FOUND(0x000D)	The attribute specified by the attr_hdl parameter is not belonging to any services or characteristics.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by the conn_hdl parameter was not found.

◆ R_BLE_GATTS_SetAttr()

ble_status_t R_BLE_GATTS_SetAttr (uint16_t *conn_hdl*, uint16_t *attr_hdl*, st_ble_gatt_value_t * *p_value*)

This function sets an attribute value to the GATT Database.

The result of this API call is returned by a return value.

Parameters

[in]	conn_hdl	If the attribute value that has information about the remote device is retrieved, specify the remote device with the conn_hdl parameter. When information about the remote device is not required, set the conn_hdl parameter to BLE_GAP_INVALID_CONN_HDL.
[in]	attr_hdl	The attribute handle of the attribute value to be set.
[in]	p_value	The attribute value to be set.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_value parameter is specified as NULL.
BLE_ERR_INVALID_DATA(0x0002)	The write size is larger than the length of the attribute value.
BLE_ERR_INVALID_ARG(0x0003)	The attr_hdl parameter is 0 or larger than the last attribute handle of GATT Database.
BLE_ERR_INVALID_STATE(0x0008)	The attribute is not in a state to be written.
BLE_ERR_INVALID_OPERATION(0x0009)	The attribute cannot be written.
BLE_ERR_NOT_FOUND(0x000D)	The attribute specified by the attr_hdl parameter is not belonging to any services or characteristics.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by the conn_hdl parameter was not found.

◆ R_BLE_GATTS_SendErrRsp()

ble_status_t R_BLE_GATTS_SendErrRsp (uint16_t *error_code*)

This function sends an error response to a remote device.

The result is returned from the API.
The error code specified in the callback is notified as Error Response to the remote device.
The result of this API call is returned by a return value.

Parameters

[in]	error_code	The error codes to be notified the client. It is a bitwise OR of GATT Error Group ID : 0x3000 and the following error codes defined in Core Spec and Core Spec Supplement.

Error Code	description
BLE_ERR_GATT_INVALID_HANDLE(0x3001)	Invalid attribute handle
BLE_ERR_GATT_READ_NOT_PERMITTED(0x3002)	The attribute cannot be read.
BLE_ERR_GATT_WRITE_NOT_PERMITTED(0x3003)	The attribute cannot be written.
BLE_ERR_GATT_INVALID_PDU(0x3004)	Invalid PDU.
BLE_ERR_GATT_INSUFFICIENT_AUTHENTICATION(0x3005)	The authentication to access the attribute is insufficient.
BLE_ERR_GATT_REQUEST_NOT_SUPPORTED(0x3006)	The request is not supported.
BLE_ERR_GATT_INVALID_OFFSET(0x3007)	The specified offset is larger than the length of the attribute value.
BLE_ERR_GATT_INSUFFICIENT_AUTHO	Authorization is required to access

				RIZATION(0x3008)	the attribute.
				BLE_ERR_GATT_PREPARE_WRITE_QUEUE_FULL(0x3009)	The Write Queue in the GATT Server is full.
				BLE_ERR_GATT_ATTRIBUTE_NOT_FOUND(0x300A)	The specified attribute is not found.
				BLE_ERR_GATT_ATTRIBUTE_NOT_LONG(0x300B)	The attribute cannot be read by Read Blob Request.
				BLE_ERR_GATT_INSUFFICIENT_ENC_KEY_SIZE(0x300C)	The Encryption Key Size is insufficient.
				BLE_ERR_GATT_INVALID_ATTRIBUTE_LEN(0x300D)	The length of the specified attribute is invalid.
				BLE_ERR_GATT_UNLIKELY_ERROR(0x300E)	Because an error has occurred, the process cannot be advanced.
				BLE_ERR_GATT_INSUFFICIENT_ENCRYPTION(0x300F)	Encryption is required to access the attribute.
				BLE_ERR_GATT_UNSUPPORTED_GROUP_TYPE(0x3010)	The type of the specified attribute is not supported.
				BLE_ERR_GATT_INSUFFICIENT_RESOURCES(0x3011)	The resource to complete the request is insufficient.
				0x3080 -	Application

		0x309F	Error. The upper layer defines the error codes.
		0x30E0 - 0x30FF	The error code defined in Common Profile and Service Error Core Specification Supplement(CSS). CSS ver.7 defines the error codes from 0x30FC to 0x30FF.
		BLE_ERR_GATT_WRITE_REQ_REJECTED(0x30FC)	The Write Request has not been completed due to the reason other than Permission.
		BLE_ERR_GATT_CCCD_IMPROPERLY_CFG(0x30FD)	The CCCD is set to be invalid.
		BLE_ERR_GATT_PROC_ALREADY_IN_PROGRESS(0x30FE)	The request is now in progress.
		BLE_ERR_GATT_OUT_OF_RANGE(0x30FF)	The attribute value is out of range.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The Group ID of the error_code parameter is not 0x3000, or it is 0x3000.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other error response,this function was called.

◆ R_BLE_GATTS_RspExMtu()

ble_status_t R_BLE_GATTS_RspExMtu (uint16_t *conn_hdl*, uint16_t *mtu*)

This function replies to a MTU Exchange Request from a remote device.

BLE_GATTS_EVENT_EX_MTU_REQ event notifies the application layer that a MTU Exchange Request has been received. Therefore when the callback has received the event, call this function.
The new MTU is the minimum of the mtu parameter specified by this function and the mtu field in BLE_GATTS_EVENT_EX_MTU_REQ event.
Default MTU size is 23 bytes.
The result of this API call is returned by a return value.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device to be sent MTU Exchange Response.
[in]	mtu	The maximum size(in bytes) of the GATT PDU that GATT Server can receive. Valid range is 23 <= mtu <= 247.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The mtu parameter is out of range.
BLE_ERR_INVALID_OPERATION(0x0009)	This function was called while processing other request.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl was not found.

◆ R_BLE_GATTS_SetPrepareQueue()

ble_status_t R_BLE_GATTS_SetPrepareQueue (st_ble_gatt_pre_queue_t * *p_pre_queues*, uint8_t *queue_num*)

Register prepare queue and buffer in Host Stack.

This function registers the prepare queue and buffer for long chracteristic write and reliable writes. The result of this API call is returned by a return value.

Parameters

[in]	p_pre_queues	The prepare write queues to be registered.
[in]	queue_num	The number of prepare write queues to be registered.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_pre_queue parameter is specified as NULL.

4.3.2.4 GATT_CLIENT

Interfaces » BLE Interface

Functions

ble_status_t	R_BLE_GATTC_Init (uint8_t cb_num) This function initializes the GATT Client and registers the number of the callbacks for GATT Client event. More...
ble_status_t	R_BLE_GATTC_RegisterCb (ble_gattc_app_cb_t cb, uint8_t priority) This function registers a callback function for GATT Client event. More...
ble_status_t	R_BLE_GATTC_DeregisterCb (ble_gattc_app_cb_t cb) This function deregisters the callback function for GATT Client event. More...
ble_status_t	R_BLE_GATTC_ReqExMtu (uint16_t conn_hdl, uint16_t mtu) This function sends a MTU Exchange Request PDU to a GATT Server

	in order to change the current MTU. More...
ble_status_t	R_BLE_GATTC_DiscAllPrimServ (uint16_t conn_hdl) This function discovers all Primary Services in a GATT Server. More...
ble_status_t	R_BLE_GATTC_DiscPrimServ (uint16_t conn_hdl, uint8_t *p_uuid, uint8_t uuid_type) This function discovers Primary Service specified by p_uuid in a GATT Server. More...
ble_status_t	R_BLE_GATTC_DiscAllSecondServ (uint16_t conn_hdl) This function discovers all Secondary Services in a GATT Server. More...
ble_status_t	R_BLE_GATTC_DiscIncServ (uint16_t conn_hdl, st_ble_gatt_hdl_range_t *p_range) This function discovers Included Services within the specified attribute handle range in a GATT Server. More...
ble_status_t	R_BLE_GATTC_DiscAllChar (uint16_t conn_hdl, st_ble_gatt_hdl_range_t *p_range) This function discovers Characteristic within the specified attribute handle range in a GATT Server. More...
ble_status_t	R_BLE_GATTC_DiscCharByUuid (uint16_t conn_hdl, uint8_t *p_uuid, uint8_t uuid_type, st_ble_gatt_hdl_range_t *p_range) This function discovers Characteristic specified by uuid within the specified attribute handle range in a GATT Server. More...
ble_status_t	R_BLE_GATTC_DiscAllCharDesc (uint16_t conn_hdl, st_ble_gatt_hdl_range_t *p_range) This function discovers Characteristic Descriptor within the specified attribute handle range in a GATT Server. More...
ble_status_t	R_BLE_GATTC_ReadChar (uint16_t conn_hdl, uint16_t value_hdl) This function reads a Characteristic/Characteristic Descriptor in a GATT Server. More...

ble_status_t	R_BLE_GATTC_ReadCharUsingUuid (uint16_t conn_hdl, uint8_t *p_uuid, uint8_t uuid_type, st_ble_gatt_hdl_range_t *p_range) This function reads a Characteristic in a GATT Server using a specified UUID. More...
ble_status_t	R_BLE_GATTC_ReadLongChar (uint16_t conn_hdl, uint16_t value_hdl, uint16_t offset) This function reads a Long Characteristic in a GATT Server. More...
ble_status_t	R_BLE_GATTC_ReadMultiChar (uint16_t conn_hdl, st_ble_gattc_rd_multi_req_param_t *p_list) This function reads multiple Characteristics in a GATT Server. More...
ble_status_t	R_BLE_GATTC_WriteCharWithoutRsp (uint16_t conn_hdl, st_ble_gatt_hdl_value_pair_t *p_write_data) This function writes a Characteristic in a GATT Server without response. More...
ble_status_t	R_BLE_GATTC_SignedWriteChar (uint16_t conn_hdl, st_ble_gatt_hdl_value_pair_t *p_write_data) This function writes Signed Data to a Characteristic in a GATT Server without response. More...
ble_status_t	R_BLE_GATTC_WriteChar (uint16_t conn_hdl, st_ble_gatt_hdl_value_pair_t *p_write_data) This function writes a Characteristic in a GATT Server. More...
ble_status_t	R_BLE_GATTC_WriteLongChar (uint16_t conn_hdl, st_ble_gatt_hdl_value_pair_t *p_write_data, uint16_t offset) This function writes a Long Characteristic in a GATT Server. More...
ble_status_t	R_BLE_GATTC_ReliableWrites (uint16_t conn_hdl, st_ble_gattc_reliable_writes_char_pair_t *p_char_pair, uint8_t pair_num, uint8_t auto_flag) This function performs the Reliable Writes procedure described in GATT Specification. More...
ble_status_t	R_BLE_GATTC_ExecWrite (uint16_t conn_hdl, uint8_t exe_flag) If the auto execute of Reliable Writes is not specified by

	R_BLE_GATTC_ReliableWrites(), this function is used to execute a write to Characteristic. More...

Detailed Description

Data Structures

struct	st_ble_gatt_hdl_range_t Attribute handle range. More...
struct	st_ble_gattc_reliable_writes_char_pair_t This is used in R_BLE_GATTC_ReliableWrites() to specify the pair of Characteristic Value and Characteristic Value Handle. More...
struct	st_ble_gattc_conn_evt_t This structure notifies that the link with the GATT Server has been established. More...
struct	st_ble_gattc_disconn_evt_t This structure notifies that the link with the GATT Server has been disconnected. More...
struct	st_ble_gattc_ex_mtu_rsp_evt_t This structure notifies that a MTU Exchange Response PDU has been received from a GATT Server. More...
struct	st_ble_gattc_serv_16_evt_t This structure notifies that a 16-bit UUID Service has been discovered. More...
struct	st_ble_gattc_serv_128_evt_t This structure notifies that a 128-bit UUID Service has been discovered. More...
struct	st_ble_gattc_inc_serv_16_evt_t This structure notifies that a 16-bit UUID Included Service has been discovered. More...

struct	st_ble_gattc_inc_serv_128_evt_t This structure notifies that a 128-bit UUID Included Service has been discovered. More...
struct	st_ble_gattc_char_16_evt_t This structure notifies that a 16-bit UUID Characteristic has been discovered. More...
struct	st_ble_gattc_char_128_evt_t This structure notifies that a 128-bit UUID Characteristic has been discovered. More...
struct	st_ble_gattc_char_desc_16_evt_t This structure notifies that a 16-bit UUID Characteristic Descriptor has been discovered. More...
struct	st_ble_gattc_char_desc_128_evt_t This structure notifies that a 128-bit UUID Characteristic Descriptor has been discovered. More...
struct	st_ble_gattc_err_rsp_evt_t This structure notifies that a Error Response PDU has been received from a GATT Server. More...
struct	st_ble_gattc_ntf_evt_t This structure notifies that a Notification PDU has been received from a GATT Server. More...
struct	st_ble_gattc_ind_evt_t This structure notifies that a Indication PDU has been received from a GATT Server. More...
struct	st_ble_gattc_rd_char_evt_t This structure notifies that read response to R_BLE_GATTC_ReadChar() or R_BLE_GATTC_ReadCharUsingUuid() has been received from a GATT Server. More...

struct	st_ble_gattc_wr_char_evt_t This structure notifies that write response to R_BLE_GATTC_WriteChar() has been received from a GATT Server. More...
struct	st_ble_gattc_rd_multi_char_evt_t This structure notifies that read response to R_BLE_GATTC_ReadMultiChar() has been received from a GATT Server. More...
struct	st_ble_gattc_char_part_wr_evt_t This structure notifies that write response to R_BLE_GATTC_WriteLongChar() or R_BLE_GATTC_ReliableWrites() has been received from a GATT Server. More...
struct	st_ble_gattc_reliable_writes_comp_evt_t This structure notifies that a response to R_BLE_GATTC_ExecWrite() has been received from a GATT Server. More...
struct	st_ble_gattc_rd_multi_req_param_t This is used in R_BLE_GATTC_ReadMultiChar() to specify multiple Characteristics to be read. More...
struct	st_ble_gattc_evt_data_t st_ble_gattc_evt_data_t is the type of the data notified in a GATT Client Event. More...
struct	st_ble_gatt_value_t Attribute Value. More...
struct	st_ble_gatt_hdl_value_pair_t Attribute handle and attribute Value. More...

Macros

#define	BLE_GATTC_EXECUTE_WRITE_CANCEL_FLAG
#define	BLE_GATTC_EXECUTE_WRITE_EXEC_FLAG
#define	BLE_GATTC_MAX_CB

	GATT Client Callback Number.
#define	BLE_GATTC_EXEC_AUTO Auto execution.
#define	BLE_GATTC_EXEC_NOT_AUTO Not auto execution.
#define	BLE_GATTC_RELIABLE_WRITES_MAX_CHAR_PAIR Length of the Queue used with Prepare Write procedure to write a characteristic whose size is larger than MTU.

Typedefs

typedef void(*	ble_gattc_app_cb_t) (uint16_t event_type, ble_status_t event_result, st_ble_gattc_evt_data_t *p_event_data) ble_gattc_app_cb_t is the GATT Client Event callback function type. More...

Enumerations

enum	e_r_ble_gattc_evt_t GATT Client Event Identifier. More...

Data Structure Documentation

◆ st_ble_gatt_hdl_range_t

struct st_ble_gatt_hdl_range_t		
Attribute handle range.		
Data Fields		
uint16_t	start_hdl	Start Attribute Handle.
uint16_t	end_hdl	End Attribute Handle.

◆ st_ble_gattc_reliable_writes_char_pair_t

struct st_ble_gattc_reliable_writes_char_pair_t		
This is used in R_BLE_GATTC_ReliableWrites() to specify the pair of Characteristic Value and Characteristic Value Handle.		
Data Fields		

st_ble_gatt_hdl_value_pair_t	write_data	Pair of Characteristic Value and Characteristic Value Handle.
uint16_t	offset	Offset that indicates the location to be written. Normally, set 0 to this parameter. If this parameter sets to a value other than 0,Adjust the offset parameter and the length of the value to be written not to exceed the length of the Characteristic.

◆ st_ble_gattc_conn_evt_t

struct st_ble_gattc_conn_evt_t		
This structure notifies that the link with the GATT Server has been established.		
Data Fields		
st_ble_dev_addr_t *	p_addr	Address of the GATT Server.

◆ st_ble_gattc_disconn_evt_t

struct st_ble_gattc_disconn_evt_t		
This structure notifies that the link with the GATT Server has been disconnected.		
Data Fields		
st_ble_dev_addr_t *	p_addr	Address of the GATT Server.

◆ st_ble_gattc_ex_mtu_rsp_evt_t

struct st_ble_gattc_ex_mtu_rsp_evt_t		
This structure notifies that a MTU Exchange Response PDU has been received from a GATT Server.		
Data Fields		
uint16_t	mtu	MTU size(in bytes) that GATT Server can receive.

◆ st_ble_gattc_serv_16_evt_t

struct st_ble_gattc_serv_16_evt_t		
This structure notifies that a 16-bit UUID Service has been discovered.		
Data Fields		
st_ble_gatt_hdl_range_t	range	Attribute handle range of the 16-bit UUID service.
uint16_t	uuid_16	Service UUID.

◆ st_ble_gattc_serv_128_evt_t

struct st_ble_gattc_serv_128_evt_t

This structure notifies that a 128-bit UUID Service has been discovered.		
Data Fields		
st_ble_gatt_hdl_range_t	range	Attribute handle range of the 128-bit UUID service.
uint8_t	uuid_128[BLE_GATT_128_BIT_UUID_SIZE]	Service UUID.

◆ st_ble_gattc_inc_serv_16_evt_t

struct st_ble_gattc_inc_serv_16_evt_t		
This structure notifies that a 16-bit UUID Included Service has been discovered.		
Data Fields		
uint16_t	decl_hdl	Service Declaration handle of the 16-bit UUID Included Service.
st_ble_gattc_serv_16_evt_t	service	The contents of the Included Service.

◆ st_ble_gattc_inc_serv_128_evt_t

struct st_ble_gattc_inc_serv_128_evt_t		
This structure notifies that a 128-bit UUID Included Service has been discovered.		
Data Fields		
uint16_t	decl_hdl	Service Declaration handle of the 128-bit UUID Included Service.
st_ble_gattc_serv_128_evt_t	service	The contents of the Included Service.

◆ st_ble_gattc_char_16_evt_t

struct st_ble_gattc_char_16_evt_t		
This structure notifies that a 16-bit UUID Characteristic has been discovered.		
Data Fields		
uint16_t	decl_hdl	Attribute handle of Characteristic Declaration.
uint8_t	cproperty	Characteristic Properties. It is a bitwise OR of the following values. Refer to Core Spec [Vol.3] Generic Attribute Profile(GATT) "3.3.1.1 Characteristic Properties" regarding the details of the Characteristic Properties.

		value / description: 0x01 Broadcast property 0x02 Read property 0x04 Write Without Response property 0x08 Write property 0x10 Notify property 0x20 Indicate property 0x40 Authenticated Signed Writes property 0x80 Extended Properties property
uint16_t	value_hdl	Value Handle of the Characteristic.
uint16_t	uuid_16	Characteristic UUID.

◆ st_ble_gattc_char_128_evt_t

struct st_ble_gattc_char_128_evt_t		
This structure notifies that a 128-bit UUID Characteristic has been discovered.		
Data Fields		
uint16_t	decl_hdl	Attribute Handle of Characteristic Declaration.
uint8_t	cproperty	Characteristic Properties. It is a bitwise OR of the following values. Refer to Core Spec [Vol.3] Generic Attribute Profile(GATT) "3.3.1.1 Characteristic Properties" regarding the details of the Characteristic Properties. value / description: 0x01 Broadcast property

		0x02 Read property 0x04 Write Without Response property 0x08 Write property 0x10 Notify property 0x20 Indicate property 0x40 Authenticated Signed Writes property 0x80 Extended Properties property
uint16_t	value_hdl	Value Handle of the Characteristic.
uint8_t	uuid_128[BLE_GATT_128_BIT_UUID_SIZE]	Characteristic UUID.

◆ st_ble_gattc_char_desc_16_evt_t

struct st_ble_gattc_char_desc_16_evt_t		
This structure notifies that a 16-bit UUID Characteristic Descriptor has been discovered.		
Data Fields		
uint16_t	desc_hdl	Attribute Handle of Characteristic Descriptor.
uint16_t	uuid_16	Characteristic Descriptor UUID.

◆ st_ble_gattc_char_desc_128_evt_t

struct st_ble_gattc_char_desc_128_evt_t		
This structure notifies that a 128-bit UUID Characteristic Descriptor has been discovered.		
Data Fields		
uint16_t	desc_hdl	Attribute Handle of Characteristic Descriptor.
uint8_t	uuid_128[BLE_GATT_128_BIT_UUID_SIZE]	Characteristic Descriptor UUID.

◆ st_ble_gattc_err_rsp_evt_t

struct st_ble_gattc_err_rsp_evt_t
This structure notifies that a Error Response PDU has been received from a GATT Server.

<table>
<tr><th colspan="3">Data Fields</th></tr>
<tr><td>uint8_t</td><td>op_code</td><td>The op code of the ATT Request that causes the Error Response.
<table><tr><th>op_code</th></tr><tr><td>Exchange MTU Request(0x02)</td></tr><tr><td>Find Information Request(0x04)</td></tr><tr><td>Find By Type Value Request(0x06)</td></tr><tr><td>Read By Type Request(0x08)</td></tr><tr><td>Read Request(0x0A)</td></tr><tr><td>Read Blob Request(0x0C)</td></tr><tr><td>Read Multiple Request(0x0E)</td></tr><tr><td>Read by Group Type Request(0x10)</td></tr><tr><td>Write Request(0x12)</td></tr><tr><td>Prepare Write Request(0x16)</td></tr><tr><td>Execute Write Request(0x18)</td></tr></table></td></tr>
<tr><td>uint16_t</td><td>attr_hdl</td><td>Attribute handle that is target for the request.</td></tr>
<tr><td>uint16_t</td><td>rsp_code</td><td>The error codes notified from the GATT Server.

It is a bitwise OR of GATT Error Group ID : 0x3000 and the following error codes defined in Core Spec and Core Spec Supplement.
<table><tr><th>Error Code</th><th>description</th></tr><tr><td>BLE_ERR_GATT_INVALID_HANDLE(0x3001)</td><td>Invalid attribute handle</td></tr><tr><td>BLE_ERR_GATT_READ_NOT_PERMITTED(0x3002)</td><td>The attribute cannot be read.</td></tr><tr><td>BLE_ERR_GATT_WRITE_NOT_PERMITTED(0x3003)</td><td>The attribute cannot be written.</td></tr><tr><td>BLE_ERR_GAT</td><td>Invalid PDU.</td></tr></table></td></tr>
</table>

		T_INVALID_PDU(0x3004)	
		BLE_ERR_GATT_INSUFFICIENT_AUTHENTICATION(0x3005)	The authentication to access the attribute is insufficient.
		BLE_ERR_GATT_REQUEST_NOT_SUPPORTED(0x3006)	The request is not supported.
		BLE_ERR_GATT_INVALID_OFFSET(0x3007)	The specified offset is larger than the length of the attribute value.
		BLE_ERR_GATT_INSUFFICIENT_AUTHORIZATION(0x3008)	Authorization is required to access the attribute.
		BLE_ERR_GATT_PREPARE_WRITE_QUEUE_FULL(0x3009)	The Write Queue in the GATT Server is full.
		BLE_ERR_GATT_ATTRIBUTE_NOT_FOUND(0x300A)	The specified attribute is not found.
		BLE_ERR_GATT_ATTRIBUTE_NOT_LONG(0x300B)	The attribute cannot be read by Read Blob Request.
		BLE_ERR_GATT_INSUFFICIENT_ENC_KEY_SIZE(0x300C)	The Encryption Key Size is insufficient.
		BLE_ERR_GATT_INVALID_ATTRIBUTE_LEN(0x300D)	The length of the specified attribute is invalid.
		BLE_ERR_GATT_UNLIKELY_ERROR(0x300E)	Because an error has occurred, the process cannot be advanced.

		BLE_ERR_GATT_INSUFFICIENT_ENCRYPTION(0x300F)	Encryption is required to access the attribute.
		BLE_ERR_GATT_UNSUPPORTED_GROUP_TYPE(0x3010)	The type of the specified attribute is not supported.
		BLE_ERR_GATT_INSUFFICIENT_RESOURCES(0x3011)	The resource to complete the request is insufficient.
		0x3080 - 0x309F	Application Error. The upper layer defines the error codes.
		0x30E0 - 0x30FF	The error code defined in Common Profile and Service Error Core Specification Supplement(CSS). CSS ver.7 defines the error codes from 0x30FC to 0x30FF.
		BLE_ERR_GATT_WRITE_REQ_REJECTED(0x30FC)	The Write Request has not been completed due to the reason other than Permission.
		BLE_ERR_GATT_CCCD_IMPROPERLY_CFG(0x30FD)	The CCCD is set to be invalid.
		BLE_ERR_GATT_PROC_ALREADY_IN_PROGRESS(0x30FE)	The request is now in progress.
		BLE_ERR_GATT_OUT_OF_RA	The attribute value is out of

		NGE(0x30FF) range.

◆ **st_ble_gattc_ntf_evt_t**

struct st_ble_gattc_ntf_evt_t		
This structure notifies that a Notification PDU has been received from a GATT Server.		
Data Fields		
st_ble_gatt_hdl_value_pair_t	data	Characteristic that causes the Notification.

◆ **st_ble_gattc_ind_evt_t**

struct st_ble_gattc_ind_evt_t		
This structure notifies that a Indication PDU has been received from a GATT Server.		
Data Fields		
st_ble_gatt_hdl_value_pair_t	data	Characteristic that causes the Indication.

◆ **st_ble_gattc_rd_char_evt_t**

struct st_ble_gattc_rd_char_evt_t		
This structure notifies that read response to R_BLE_GATTC_ReadChar() or R_BLE_GATTC_ReadCharUsingUuid() has been received from a GATT Server.		
Data Fields		
st_ble_gatt_hdl_value_pair_t	read_data	The contents of the Characteristic that has been read.

◆ **st_ble_gattc_wr_char_evt_t**

struct st_ble_gattc_wr_char_evt_t		
This structure notifies that write response to R_BLE_GATTC_WriteChar() has been received from a GATT Server.		
Data Fields		
uint16_t	value_hdl	Value Handle of the Characteristic/Characteristic Descriptor that has been written.

◆ **st_ble_gattc_rd_multi_char_evt_t**

struct st_ble_gattc_rd_multi_char_evt_t		
This structure notifies that read response to R_BLE_GATTC_ReadMultiChar() has been received from a GATT Server.		
Data Fields		
uint16_t	value_hdl_num	The number of Value Handles of the Characteristics that has

		been read.
st_ble_gatt_value_t	multi_char_val	The contents of multiple Characteristics that have been read.

◆ st_ble_gattc_char_part_wr_evt_t

struct st_ble_gattc_char_part_wr_evt_t

This structure notifies that write response to R_BLE_GATTC_WriteLongChar() or R_BLE_GATTC_ReliableWrites() has been received from a GATT Server.

Data Fields		
st_ble_gatt_hdl_value_pair_t	write_data	The data to be written to the Characteristic/Long Characteristic/Long Characteristic Descriptor.
uint16_t	offset	Offset that indicates the location to be written.

◆ st_ble_gattc_reliable_writes_comp_evt_t

struct st_ble_gattc_reliable_writes_comp_evt_t

This structure notifies that a response to R_BLE_GATTC_ExecWrite() has been received from a GATT Server.

Data Fields		
uint8_t	exe_flag	This field indicates the command of the Execute Write that has been done. value: description 0x00: Cancel the write. 0x01: Execute the write.

◆ st_ble_gattc_rd_multi_req_param_t

struct st_ble_gattc_rd_multi_req_param_t

This is used in R_BLE_GATTC_ReadMultiChar() to specify multiple Characteristics to be read.

Data Fields		
uint16_t *	p_hdl_list	List of Value Handles that point the Characteristics to be read.
uint16_t	list_count	The number of Value Handles included in the hdl_list parameter.

◆ st_ble_gattc_evt_data_t

struct st_ble_gattc_evt_data_t

st_ble_gattc_evt_data_t is the type of the data notified in a GATT Client Event.		
Data Fields		
uint16_t	conn_hdl	Connection handle identifying the GATT Server.
uint16_t	param_len	The size of GATT Client Event parameters.
void *	p_param	GATT Client Event parameters. This parameter differs in each GATT Client Event.

◆ st_ble_gatt_value_t

struct st_ble_gatt_value_t		
Attribute Value.		
Data Fields		
uint16_t	value_len	Length of the attribute value.
uint8_t *	p_value	Attribute Value.

◆ st_ble_gatt_hdl_value_pair_t

struct st_ble_gatt_hdl_value_pair_t		
Attribute handle and attribute Value.		
Data Fields		
uint16_t	attr_hdl	Attribute Handle.
st_ble_gatt_value_t	value	Attribute Value.

Macro Definition Documentation

◆ BLE_GATTC_EXECUTE_WRITE_CANCEL_FLAG

#define BLE_GATTC_EXECUTE_WRITE_CANCEL_FLAG
GATT Execute Write Cancel Flag.

◆ BLE_GATTC_EXECUTE_WRITE_EXEC_FLAG

#define BLE_GATTC_EXECUTE_WRITE_EXEC_FLAG
GATT Execute Write Execute Flag.

Typedef Documentation

◆ ble_gattc_app_cb_t

ble_gattc_app_cb_t
ble_gattc_app_cb_t is the GATT Client Event callback function type.

Parameters

[in]	event_type	The type of GATT Client Event.
[in]	event_result	The result of GATT Client Event
[in]	p_event_data	Data notified by GATT Client Event.

Returns
none

Enumeration Type Documentation

◆ e_r_ble_gattc_evt_t

enum e_r_ble_gattc_evt_t
GATT Client Event Identifier.

Enumerator	
BLE_GATTC_EVENT_ERROR_RSP	This event notifies the application layer that a problem has occurred in the GATT Server while processing a request from GATT Client. When GATT Client has received a Error Response PDU from a GATT Server, BLE_GATTC_EVENT_ERROR_RSP event is notified the application layer. **Event Code: 0x4001** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gattc_err_rsp_evt_tBLE_GATTC_EVENT_ERROR_RSP
BLE_GATTC_EVENT_EX_MTU_RSP	This event notifies the application layer that a MTU Exchange Response PDU has been received from a GATT Server.

	Event Code: 0x4003 **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_RSP_TIMEOUT(0x0011) — 30 seconds or more have passed without receiving a Exchange MTU Response since GATT Client sent a Exchange MTU Request PDU to the GATT Server. **Event Data:** st_ble_gattc_ex_mtu_rsp_evt_tBLE_GATTC_EVENT_EX_MTU_RSP
BLE_GATTC_EVENT_CHAR_READ_BY_UUID_RSP	When the read of Characteristic specified by UUID has been completed, this event is notified to the application layer. **Event Code: 0x4009** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_RSP_TIMEOUT(0x0011) — 30 seconds or more have passed without receiving a Exchange MTU Response since GATT Client sent a Exchange MTU Request PDU to the GATT Server. **Event Data:** st_ble_gattc_rd_char_evt_tBLE_GATTC_EVENT_CHAR_READ_BY_UUID_RSP
BLE_GATTC_EVENT_CHAR_READ_RSP	When the read of Characteristic/Characteristic Descriptor has been completed, this event is notified to the application layer. **Event Code: 0x400B** **result:**

	BLE_SUCCESS(0x0000) Success BLE_ERR_RSP_TIMEOUT(0x0011) 30 seconds or more have passed without receiving a read response since GATT Client sent a request for read by R_BLE_GATTC_ReadCharUsingUuid() to the GATT Server. **Event Data:** st_ble_gattc_rd_char_evt_tBLE_GATTC_EVENT_CHAR_READ_RSP
BLE_GATTC_EVENT_CHAR_PART_READ_RSP	After calling R_BLE_GATTC_ReadLongChar(), this event notifies the application layer that the partial contents of Long Characteristic/Long Characteristic Descriptor has been received from the GATT Server. **Event Code: 0x400D** **result:** BLE_SUCCESS(0x0000) Success BLE_ERR_RSP_TIMEOUT(0x0011) 30 seconds or more have passed without receiving a read response since GATT Client sent a request for read by R_BLE_GATTC_ReadLongChar() to the GATT Server. **Event Data:** st_ble_gattc_rd_char_evt_tBLE_GATTC_EVENT_CHAR_PART_READ_RSP
BLE_GATTC_EVENT_MULTI_CHAR_READ_RSP	This event notifies the application layer that the read of multiple Characteristics has been completed. **Event Code: 0x400F**

	result: BLE_SUCCESS(0x0000) — Success BLE_ERR_RSP_TIMEOUT(0x0011) — 30 seconds or more have passed without receiving a read response since GATT Client sent a request for read by R_BLE_GATTC_ReadMultiChar() to the GATT Server. **Event Data:** st_ble_gattc_rd_multi_char_evt_tBLE_GATTC_EVENT_MULTI_CHAR_READ_RSP
BLE_GATTC_EVENT_CHAR_WRITE_RSP	This event notifies the application layer that the write of Characteristic/Characteristic Descriptor has been completed. **Event Code: 0x4013** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_RSP_TIMEOUT(0x0011) — 30 seconds or more have passed without receiving a write response since GATT Client sent a request for write by R_BLE_GATTC_WriteChar() to the GATT Server. **Event Data:** st_ble_gattc_wr_char_evt_tBLE_GATTC_EVENT_CHAR_WRITE_RSP
BLE_GATTC_EVENT_CHAR_PART_WRITE_RSP	This event notifies the application layer of the one of the following. • A segmentation to be written to Long Characteristic/Long Characteristic Descriptor has been sent to the GATT

	Server. • The data written to one Characteristic by Reliable Writes has been sent to the GATT Server. **Event Code: 0x4017** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_RSP_TIMEOUT(0x0011) — 30 seconds or more have passed without receiving a response since GATT Client sent a request for segmentation write by R_BLE_GATTC_WriteLongChar(), or 1 Characteristic write by R_BLE_GATTC_ReliableWrites() to the GATT Server. **Event Data:** st_ble_gattc_char_part_wr_evt_tBLE_GATTC_EVENT_CHAR_PART_WRITE_RSP
BLE_GATTC_EVENT_HDL_VAL_NTF	This event notifies the application layer that a Notification has been received from a GATT Server. **Event Code: 0x401B** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** st_ble_gattc_ntf_evt_tBLE_GATTC_EVENT_HDL_VAL_NTF
BLE_GATTC_EVENT_HDL_VAL_IND	This event notifies the application layer that a Indication has been received from a GATT Server. When the GATT Client has received a Indication, host stack automatically sends a

	Confirmation to the GATT Server. **Event Code: 0x401D** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_MEM_ALLOC_FAILED(0x000C) — Insufficient resource is needed to generate the confirmation packet. **Event Data:** st_ble_gattc_ind_evt_tBLE_GATTC_EVENT_HDL_VAL_IND
BLE_GATTC_EVENT_CONN_IND	This event notifies the application layer that the link with the GATT Server has been established. **Event Code: 0x4081** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** st_ble_gattc_conn_evt_tBLE_GATTC_EVENT_CONN_IND
BLE_GATTC_EVENT_DISCONN_IND	This event notifies the application layer that the link with the GATT Server has been disconnected. **Event Code: 0x4082** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** st_ble_gattc_disconn_evt_tBLE_GATTC_EVENT_DISCONN_IND
BLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND	This event notifies the application layer that 16-bit UUID Primary Service has been

	discovered. **Event Code: 0x40E0** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gattc_serv_16_evt_tBLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND
BLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND	This event notifies the application layer that 128-bit UUID Primary Service has been discovered. **Event Code: 0x40E1** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gattc_serv_128_evt_tBLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND
BLE_GATTC_EVENT_ALL_PRIM_SERV_DISC_COMP	When the Primary Service discovery by R_BLE_GATTC_DiscAllPrimServ() has been completed, this event is notified to the application layer. **Event Code: 0x40E2** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** noneBLE_GATTC_EVENT_ALL_PRIM_SERV_DISC_COMP
BLE_GATTC_EVENT_PRIM_SERV_DISC_COMP	When the Primary Service discovery by R_BLE_GATTC_DiscPrimServ() has been completed, this event is notified to the application layer. **Event Code: 0x40E3**

	result: BLE_SUCCESS(0x0000) Success **Event Data:** noneBLE_GATTC_EVENT_PRIM_SERV_DISC_COMP
BLE_GATTC_EVENT_SECOND_SERV_16_DISC_IND	This event notifies the application layer that 16-bit UUID Secondary Service has been discovered. **Event Code: 0x40E4** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gattc_serv_16_evt_tBLE_GATTC_EVENT_SECOND_SERV_16_DISC_IND
BLE_GATTC_EVENT_SECOND_SERV_128_DISC_IND	This event notifies the application layer that 128-bit UUID Secondary Service has been discovered. **Event Code: 0x40E5** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gattc_serv_128_evt_tBLE_GATTC_EVENT_SECOND_SERV_128_DISC_IND
BLE_GATTC_EVENT_ALL_SECOND_SERV_DISC_COMP	When the Primary Service discovery by R_BLE_GATTC_DiscAllSecondServ() has been completed, this event is notified to the application layer. **Event Code: 0x40E6** **result:** BLE_SUCCESS(0x0000) Success

	Event Data: noneBLE_GATTC_EVENT_ALL_SECOND_SERV_DISC_COMP
BLE_GATTC_EVENT_INC_SERV_16_DISC_IND	This event notifies the application layer that Included Service that includes 16-bit UUID Service has been discovered. **Event Code: 0x40E7** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gattc_inc_serv_16_evt_tBLE_GATTC_EVENT_INC_SERV_16_DISC_IND
BLE_GATTC_EVENT_INC_SERV_128_DISC_IND	This event notifies the application layer that Included Service that includes 128-bit UUID Service has been discovered. **Event Code: 0x40E8** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gattc_inc_serv_128_evt_tBLE_GATTC_EVENT_INC_SERV_128_DISC_IND
BLE_GATTC_EVENT_INC_SERV_DISC_COMP	When the Included Service discovery by R_BLE_GATTC_DiscIncServ() has been completed, this event is notified to the application layer. **Event Code: 0x40E9** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** noneBLE_GATTC_EVENT_INC_SERV_DISC_COMP

BLE_GATTC_EVENT_CHAR_16_DISC_IND	This event notifies the application layer that 16-bit UUID Characteristic has been discovered. **Event Code: 0x40EA** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gattc_char_16_evt_tBLE_GATTC_EVENT_CHAR_16_DISC_IND
BLE_GATTC_EVENT_CHAR_128_DISC_IND	This event notifies the application layer that 128-bit UUID Characteristic has been discovered. **Event Code: 0x40EB** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gattc_char_128_evt_tBLE_GATTC_EVENT_CHAR_128_DISC_IND
BLE_GATTC_EVENT_ALL_CHAR_DISC_COMP	When the Characteristic discovery by R_BLE_GATTC_DiscAllChar() has been completed, this event is notified to the application layer. **Event Code: 0x40EC** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** noneBLE_GATTC_EVENT_ALL_CHAR_DISC_COMP
BLE_GATTC_EVENT_CHAR_DISC_COMP	When the Characteristic discovery by R_BLE_GATTC_DiscCharByUuid() has been completed, this event is notified to the application layer.

	Event Code: 0x40ED **result:** BLE_SUCCESS(0x0000) Success **Event Data:** noneBLE_GATTC_EVENT_CHAR_DISC_COMP
BLE_GATTC_EVENT_CHAR_DESC_16_DISC_IND	This event notifies the application layer that 16-bit UUID Characteristic Descriptor has been discovered. **Event Code: 0x40EE** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gattc_char_desc_16_evt_tBLE_GATTC_EVENT_CHAR_DESC_16_DISC_IND
BLE_GATTC_EVENT_CHAR_DESC_128_DISC_IND	This event notifies the application layer that 128-bit UUID Characteristic Descriptor has been discovered. **Event Code: 0x40EF** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_gattc_char_desc_128_evt_tBLE_GATTC_EVENT_CHAR_DESC_128_DISC_IND
BLE_GATTC_EVENT_ALL_CHAR_DESC_DISC_COMP	When the Characteristic Descriptor discovery by R_BLE_GATTC_DiscAllCharDesc() has been completed, this event is notified to the application layer. **Event Code: 0x40F0** **result:** BLE_SUCCESS(0x0 Success

	000) **Event Data:** noneBLE_GATTC_EVENT_ALL_CHAR_DESC_DISC_COMP
BLE_GATTC_EVENT_LONG_CHAR_READ_COMP	After calling R_BLE_GATTC_ReadLongChar(), this event notifies the application layer that all of the contents of the Characteristic/Long Characteristic Descriptor has been received from the GATT Server. **Event Code: 0x40F1** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** noneBLE_GATTC_EVENT_LONG_CHAR_READ_COMP
BLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP	This event notifies that the application layer that the write of Long Characteristic/Long Characteristic Descriptor has been completed. **Event Code: 0x40F2** **result:** BLE_SUCCESS(0x0000) Success BLE_ERR_RSP_TIMEOUT(0x0011) 30 seconds or more have passed without receiving a response since GATT Client sent a request for write by R_BLE_GATTC_WriteLongChar() to the GATT Server. **Event Data:** noneBLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP
BLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP	This event notifies that the application layer that the GATT Server has received the data to

	be written to the Characteristics. **Event Code: 0x40F3** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** noneBLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP
BLE_GATTC_EVENT_RELIABLE_WRITES_COMP	This event notifies the application layer that the Reliable Writes has been completed. **Event Code: 0x40F4** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_RSP_TIMEOUT(0x0011) — 30 seconds or more have passed without receiving a response since GATT Client sent a request for execute write by R_BLE_GATTC_ReliableWrites() or R_BLE_GATTC_ExecWrite() to the GATT Server. **Event Data:** st_ble_gattc_reliable_writes_comp_evt_tBLE_GATTC_EVENT_RELIABLE_WRITES_COMP
BLE_GATTC_EVENT_INVALID	Invalid GATT Client Event. **Event Code: 0x40FF** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** noneBLE_GATTC_EVENT_INVALID

Function Documentation

◆ R_BLE_GATTC_Init()

ble_status_t R_BLE_GATTC_Init (uint8_t *cb_num*)

This function initializes the GATT Client and registers the number of the callbacks for GATT Client event.

Specify the cb_num parameter to a value between 1 and BLE_GATTC_MAX_CB.
R_BLE_GATTC_RegisterCb() registers the callback.
The result of this API call is returned by a return value.

Parameters

[in]	cb_num	The number of callbacks to be registered.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The cb_num parameter is out of range.

◆ R_BLE_GATTC_RegisterCb()

ble_status_t R_BLE_GATTC_RegisterCb (ble_gattc_app_cb_t *cb*, uint8_t *priority*)

This function registers a callback function for GATT Client event.

The number of the callback that may be registered by this function is the value specified by R_BLE_GATTC_Init().
The result of this API call is returned by a return value.

Parameters

[in]	cb	Callback function for GATT Client event.
[in]	priority	The priority of the callback function. Valid range is 1 <= priority <= BLE_GATTC_MAX_CB. A lower priority number means a higher priority level.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The cb parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The priority parameter is out of range.
BLE_ERR_CONTEXT_FULL(0x000B)	Host stack has already registered the maximum number of callbacks.

◆ R_BLE_GATTC_DeregisterCb()

ble_status_t R_BLE_GATTC_DeregisterCb (ble_gattc_app_cb_t *cb*)

This function deregisters the callback function for GATT Client event.

The result of this API call is returned by a return value.

Parameters

[in]	cb	The callback function to be deregistered.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The cb parameter is specified as NULL.
BLE_ERR_NOT_FOUND(0x000D)	The callback has not been registered.

◆ R_BLE_GATTC_ReqExMtu()

ble_status_t R_BLE_GATTC_ReqExMtu (uint16_t *conn_hdl*, uint16_t *mtu*)

This function sends a MTU Exchange Request PDU to a GATT Server in order to change the current MTU.

MTU Exchange Response is notified by BLE_GATTC_EVENT_EX_MTU_RSP event.
The new MTU is the minimum value of the mtu parameter specified by this function and the mtu field in BLE_GATTC_EVENT_EX_MTU_RSP event. Default MTU size is 23 bytes.
The result of this API call is returned by a return value.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be sent.
[in]	mtu	The maximum size(in bytes) of the GATT PDU that GATT Client can receive. Valid range is 23 <= mtu <= 247.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The mtu parameter is out of range.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_DiscAllPrimServ()

ble_status_t R_BLE_GATTC_DiscAllPrimServ (uint16_t *conn_hdl*)

This function discovers all Primary Services in a GATT Server.

When 16-bit UUID Primary Service has been discovered,
BLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND event is notified to the application layer.
When 128-bit UUID Primary Service has been discovered,
BLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND event is notified to the application layer.
When the Primary Service discovery has been completed,
BLE_GATTC_EVENT_ALL_PRIM_SERV_DISC_COMP event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be discovered.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_OPERATION(0x0009)	This function was called while processing other request.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_DiscPrimServ()

ble_status_t R_BLE_GATTC_DiscPrimServ (uint16_t *conn_hdl*, uint8_t * *p_uuid*, uint8_t *uuid_type*)

This function discovers Primary Service specified by p_uuid in a GATT Server.

When Primary Service whose uuid is the same as the specified uuid has been discovered, BLE_GATTC_EVENT_PRIM_SERV_16_DISC_IND event or BLE_GATTC_EVENT_PRIM_SERV_128_DISC_IND event is notified to the application layer. When the Primary Service discovery has been completed, BLE_GATTC_EVENT_PRIM_SERV_DISC_COMP event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be discovered.
[in]	p_uuid	UUID of Primary Service to be discovered.
[in]	uuid_type	UUID type(16-bit or 128-bit). macro: BLE_GATT_16_BIT_UUID_FORMAT(0x01) — description: 16-bit UUID macro: BLE_GATT_128_BIT_UUID_FORMAT(0x02) — description: 128-bit UUID

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_uuid parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The uuid_type parameter is out of range.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_DiscAllSecondServ()

ble_status_t R_BLE_GATTC_DiscAllSecondServ (uint16_t *conn_hdl*)

This function discovers all Secondary Services in a GATT Server.

When a 16-bit UUID Secondary Service has been discovered, BLE_GATTC_EVENT_SECOND_SERV_16_DISC_IND event is notified to the application layer. When a 128-bit UUID Secondary Service has been discovered, BLE_GATTC_EVENT_SECOND_SERV_128_DISC_IND event is notified to the application layer. When the Secondary Service discovery has been completed, BLE_GATTC_EVENT_ALL_SECOND_SERV_DISC_COMP event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be discovered.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_DiscIncServ()

ble_status_t R_BLE_GATTC_DiscIncServ (uint16_t *conn_hdl*, st_ble_gatt_hdl_range_t * *p_range*)

This function discovers Included Services within the specified attribute handle range in a GATT Server.

When Included Service that includes 16-bit UUID Service has been discovered, BLE_GATTC_EVENT_INC_SERV_16_DISC_IND event is notified to the application layer. When Included Service that includes 128-bit UUID Service has been discovered, BLE_GATTC_EVENT_INC_SERV_128_DISC_IND event is notified to the application layer. When the Included Service discovery has been completed, BLE_GATTC_EVENT_INC_SERV_DISC_COMP event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be discovered.
[in]	p_range	Retrieval range of Included Service.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_range parameter is specified as NULL.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_DiscAllChar()

ble_status_t R_BLE_GATTC_DiscAllChar (uint16_t *conn_hdl*, st_ble_gatt_hdl_range_t * *p_range*)

This function discovers Characteristic within the specified attribute handle range in a GATT Server.

When 16-bit UUID Characteristic has been discovered, BLE_GATTC_EVENT_CHAR_16_DISC_IND event is notified to the application layer.
When 128-bit UUID Characteristic has been discovered, BLE_GATTC_EVENT_CHAR_128_DISC_IND event is notified to the application layer.
When the Characteristic discovery has been completed, BLE_GATTC_EVENT_ALL_CHAR_DISC_COMP event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be discovered.
[in]	p_range	Retrieval range of Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_range parameter is specified as NULL.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_DiscCharByUuid()

ble_status_t R_BLE_GATTC_DiscCharByUuid (uint16_t *conn_hdl*, uint8_t * *p_uuid*, uint8_t *uuid_type*, st_ble_gatt_hdl_range_t * *p_range*)

This function discovers Characteristic specified by uuid within the specified attribute handle range in a GATT Server.

When 16-bit UUID Characteristic has been discovered, BLE_GATTC_EVENT_CHAR_16_DISC_IND event is notified to the application layer.
When 128-bit UUID Characteristic has been discovered, BLE_GATTC_EVENT_CHAR_128_DISC_IND event is notified to the application layer.
When the Characteristic discovery has been completed, BLE_GATTC_EVENT_CHAR_DISC_COMP event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be discovered.
[in]	p_uuid	UUID of Characteristic to be discovered.
[in]	uuid_type	UUID type of Characteristic to be discovered. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GATT_16_BIT_UUID_FORMAT(0x01)</td><td>The p_uuid parameter is 16-bit UUID.</td></tr><tr><td>BLE_GATT_128_BIT_UUID_FORMAT(0x02)</td><td>The p_uuid parameter is 128-bit UUID.</td></tr></table>
[in]	p_range	Retrieval range of Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_uuid parameter or the p_range parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The uuid_type parameter is out of range.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_DiscAllCharDesc()

ble_status_t R_BLE_GATTC_DiscAllCharDesc (uint16_t *conn_hdl*, st_ble_gatt_hdl_range_t * *p_range*)

This function discovers Characteristic Descriptor within the specified attribute handle range in a GATT Server.

When 16-bit UUID Characteristic Descriptor has been discovered, BLE_GATTC_EVENT_CHAR_DESC_16_DISC_IND event is notified to the application layer. When 128-bit UUID Characteristic Descriptor has been discovered, BLE_GATTC_EVENT_CHAR_DESC_128_DISC_IND event is notified to the application layer. When the Characteristic Descriptor discovery has been completed, BLE_GATTC_EVENT_ALL_CHAR_DESC_DISC_COMP event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be discovered.
[in]	p_range	Retrieval range of Characteristic Descriptor.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_range parameter is specified as NULL.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_ReadChar()

ble_status_t R_BLE_GATTC_ReadChar (uint16_t *conn_hdl*, uint16_t *value_hdl*)

This function reads a Characteristic/Characteristic Descriptor in a GATT Server.

The result of the read is notified in BLE_GATTC_EVENT_CHAR_READ_RSP event.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be read.
[in]	value_hdl	Value handle of the Characteristic/Characteristic Descriptor to be read.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	0 is specified in the value_hdl parameter.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_ReadCharUsingUuid()

ble_status_t R_BLE_GATTC_ReadCharUsingUuid (uint16_t *conn_hdl*, uint8_t * *p_uuid*, uint8_t *uuid_type*, st_ble_gatt_hdl_range_t * *p_range*)

This function reads a Characteristic in a GATT Server using a specified UUID.

The result of the read is notified in BLE_GATTC_EVENT_CHAR_READ_BY_UUID_RSP event.

Parameters

[in]	conn_hdl	Connection handle that identifies Characteristic to be read to GATT Server.
[in]	p_uuid	UUID of the Characteristic to be read.
[in]	uuid_type	UUID type of the Characteristic to be read. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GATT_16_BIT_UUID_FORMAT(0x01)</td><td>The p_uuid parameter is 16-bit UUID.</td></tr><tr><td>BLE_GATT_128_BIT_UUID_FORMAT(0x02)</td><td>The p_uuid parameter is 128-bit UUID.</td></tr></table>
[in]	p_range	Retrieval range of Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_uuid parameter or the p_range parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The uuid_type parameter is out of range.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_ReadLongChar()

ble_status_t R_BLE_GATTC_ReadLongChar (uint16_t *conn_hdl*, uint16_t *value_hdl*, uint16_t *offset*)

This function reads a Long Characteristic in a GATT Server.

The contents of the Long Characteristic that has been read is notified every MTU-1 bytes to the application layer by BLE_GATTC_EVENT_CHAR_READ_RSP event.
When all of the contents has been received in GATT Client, BLE_GATTC_EVENT_LONG_CHAR_READ_COMP event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be read.
[in]	value_hdl	Value handle of the Long Characteristic to be read.
[in]	offset	Offset that indicates the location to be read. Normally, set 0 to this parameter.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	0 is specified in the value_hdl parameter.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_ReadMultiChar()

ble_status_t R_BLE_GATTC_ReadMultiChar (uint16_t *conn_hdl*, st_ble_gattc_rd_multi_req_param_t * *p_list*)

This function reads multiple Characteristics in a GATT Server.

The contents of the multiple Characteristics that has been read is notified to the application layer by BLE_GATTC_EVENT_MULTI_CHAR_READ_RSP event.

Parameters

[in]	conn_hdl	Connection handle that identifies Characteristic to be read to GATT Server.
[in]	p_list	List of Value Handles that point the Characteristics to be read.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_list parameter or the p_hdl_list field in the p_list parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	0 is specified in the list_count field in the p_list parameter.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_WriteCharWithoutRsp()

ble_status_t R_BLE_GATTC_WriteCharWithoutRsp (uint16_t *conn_hdl*, st_ble_gatt_hdl_value_pair_t * *p_write_data*)

This function writes a Characteristic in a GATT Server without response.

The result is returned from the API.

Parameters

[in]	conn_hdl	Connection handle that identifies Characteristic to be read to GATT Server.
[in]	p_write_data	Value to be written to the Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_write_data parameter or the p_value field in the value field in the p_write_data parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • 0 is specified in the value_len field in the p_value field in the p_write_data parameter. • 0 is specified in the attr_hdl field in the p_write_data parameter.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_SignedWriteChar()

ble_status_t R_BLE_GATTC_SignedWriteChar (uint16_t *conn_hdl*, st_ble_gatt_hdl_value_pair_t * *p_write_data*)

This function writes Signed Data to a Characteristic in a GATT Server without response.

The result of this API call is returned by a return value.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be written.
[in]	p_write_data	Signed Data to be written to the Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_write_data parameter or the p_value field in the value field in the p_write_data parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • 0 is specified in the value_len field in the value field in the p_write_data parameter. • 0 is specified in the attr_hdl field in the p_write_data parameter.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function or Signed Data.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_WriteChar()

ble_status_t R_BLE_GATTC_WriteChar (uint16_t *conn_hdl*, st_ble_gatt_hdl_value_pair_t * *p_write_data*)

This function writes a Characteristic in a GATT Server.

The result of the write is notified in BLE_GATTC_EVENT_CHAR_WRITE_RSP event.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be written.
[in]	p_write_data	Value to be written to the Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_write_data parameter or the p_value field in the value field in the p_write_data parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • 0 is specified in the value_len field in the value field in the p_write_data parameter. • 0 is specified in the attr_hdl field in the p_write_data parameter.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_WriteLongChar()

ble_status_t R_BLE_GATTC_WriteLongChar (uint16_t *conn_hdl*, st_ble_gatt_hdl_value_pair_t * *p_write_data*, uint16_t *offset*)

This function writes a Long Characteristic in a GATT Server.

The result of a write that has been done every segmentation is notified to the application layer in BLE_GATTC_EVENT_CHAR_PART_WRITE_RSP event.
The maximum writable size to a Long Characteristic with this function is 512 bytes.
When all of the contents has been written to the Long Characteristic, BLE_GATTC_EVENT_LONG_CHAR_WRITE_COMP event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be written.
[in]	p_write_data	Value to be written to the Long Characteristic.
[in]	offset	Offset that indicates the location to be written. Normally, set 0 to this parameter. If this parameter sets to a value other than 0, adjust the offset parameter and the length of the value to be written not to exceed the length of the Long Characteristic.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_write_data parameter or the p_value field in the value field in the p_write_data parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: • The value_len field in the value field in the p_write_data parameter is 0. • The sum of the value_len field in the value field in the p_write_data parameter and the offset parameter larger than 512. • The attr_hdl field in the p_write_data parameter is 0.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_ReliableWrites()

ble_status_t R_BLE_GATTC_ReliableWrites (uint16_t *conn_hdl*, st_ble_gattc_reliable_writes_char_pair_t * *p_char_pair*, uint8_t *pair_num*, uint8_t *auto_flag*)

This function performs the Reliable Writes procedure described in GATT Specification.

When the data written to the Characteristic has been transmitted, BLE_GATTC_EVENT_CHAR_PART_WRITE_RSP event is notified to the application layer.
If the data included in the event is different from the data that GATT Client has sent, host stack automatically cancels the Reliable Writes.
After all of the contents has been sent to the GATT Server, if the auto_flag parameter has been set to BLE_GATTC_EXEC_AUTO, the GATT Server automatically writes the data to the Characteristic.
If the auto_flag parameter has been set to BLE_GATTC_EXEC_NOT_AUTO, BLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP event notifies the application layer in GATT Client that all of the contents has been sent to the GATT Server. Then GATT Client requests for writing the data to the Characteristic to the GATT Server with R_BLE_GATTC_ExecWrite().
When the write has been done, BLE_GATTC_EVENT_RELIABLE_WRITES_COMP event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the GATT Server to be written.
[in]	p_char_pair	Pair of Characteristic Value and Characteristic Value Handle identifying the Characteristic to be written by Reliable Writes.
[in]	pair_num	The number of the pairs specified by the p_char_pair parameter. Valid range is 0 < pair_num <= BLE_GATTC_RELIABLE_WRITES_MAX_CHAR_PAIR.
[in]	auto_flag	The flag that indicates whether auto execution or not. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GATTC_EXEC_AUTO(0x01)</td><td>Auto execution.</td></tr><tr><td>BLE_GATTC_EXEC_NOT_AUTO (0x02)</td><td>Not auto execution.</td></tr></table>

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The reason for this error is as follows:

	- The p_char_pair parameter is specified as NULL. - The p_value field in the value field in the write_data field in the p_char_pair parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The reason for this error is as follows: - The pair_num parameter or the auto_flag parameter is out of range. - The value_len field in the value field in the write_data field in the p_char_pair parameter is 0.
BLE_ERR_INVALID_OPERATION(0x0009)	While processing other request, this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function or to store the temporary write data.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

◆ R_BLE_GATTC_ExecWrite()

ble_status_t R_BLE_GATTC_ExecWrite (uint16_t *conn_hdl*, uint8_t *exe_flag*)

If the auto execute of Reliable Writes is not specified by R_BLE_GATTC_ReliableWrites(), this function is used to execute a write to Characteristic.

When all of the contents has been sent to the GATT Server,
BLE_GATTC_EVENT_RELIABLE_WRITES_TX_COMP event notifies the application layer.
After this event has been received, execute the write by this function.
The result of the write is notified by BLE_GATTC_EVENT_RELIABLE_WRITES_COMP event.

Parameters

[in]	conn_hdl	Connection handle identifying the target GATT Server.
[in]	exe_flag	The flag that indicates whether execution or cancellation. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GATTC_EXECUTE_WRITE_CANCEL_FLAG(0x00)</td><td>Execute the write.</td></tr><tr><td>BLE_GATTC_EXECUTE_WRITE_EXEC_FLAG(0x01)</td><td>Cancel the write.</td></tr></table>

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The exe_flag parameter is out of range.
BLE_ERR_INVALID_OPERATION(0x0009)	The reason for this error is as follows: • GATT Client has not requested for Reliable Writes by R_BLE_GATTC_ReliableWrites(). • Although auto execution has been specified by R_BLE_GATTC_ReliableWrites(), this function was called.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The GATT Server specified by conn_hdl was not found.

4.3.2.5 L2CAP

Interfaces » BLE Interface

Functions

ble_status_t	R_BLE_L2CAP_RegisterCfPsm (ble_l2cap_cf_app_cb_t cb, uint16_t psm, uint16_t lwm) This function registers PSM that uses L2CAP CBFC Channel and a callback for L2CAP event. More...
ble_status_t	R_BLE_L2CAP_DeregisterCfPsm (uint16_t psm) This function stops the use of the L2CAP CBFC Channel specified by the psm parameter and deregisters the callback function for L2CAP event. More...
ble_status_t	R_BLE_L2CAP_ReqCfConn (uint16_t conn_hdl, st_ble_l2cap_conn_req_param_t *p_conn_req_param) This function sends a connection request for L2CAP CBFC Channel. More...
ble_status_t	R_BLE_L2CAP_RspCfConn (st_ble_l2cap_conn_rsp_param_t *p_conn_rsp_param) This function replies to the connection request for L2CAP CBFC Channel from the remote device. More...
ble_status_t	R_BLE_L2CAP_DisconnectCf (uint16_t lcid) This function sends a disconnection request for L2CAP CBFC Channel. More...
ble_status_t	R_BLE_L2CAP_SendCfCredit (uint16_t lcid, uint16_t credit) This function sends credit to a remote device. More...
ble_status_t	R_BLE_L2CAP_SendCfData (uint16_t conn_hdl, uint16_t lcid, uint16_t data_len, uint8_t *p_sdu) This function sends the data to a remote device via L2CAP CBFC Channel. More...

Detailed Description

Data Structures

struct	st_ble_l2cap_conn_req_param_t L2CAP CBFC Channel connection request parameters. More...
struct	st_ble_l2cap_conn_rsp_param_t L2CAP CBFC Channel connection response parameters. More...
struct	st_ble_l2cap_cf_conn_evt_t L2CAP CBFC Channel connection parameters. More...
struct	st_ble_l2cap_cf_data_evt_t Sent/Received Data parameters. More...
struct	st_ble_l2cap_cf_credit_evt_t Credit parameters of local or remote device. More...
struct	st_ble_l2cap_cf_disconn_evt_t Disconnection parameters. More...
struct	st_ble_l2cap_rej_evt_t Command Reject parameters. More...
struct	st_ble_l2cap_cf_evt_data_t st_ble_l2cap_cf_evt_data_t is the type of the data notified in a L2CAP Event. More...

Macros

#define	BLE_L2CAP_MAX_CBFC_PSM The maximum number of callbacks that host stack can register.
#define	BLE_L2CAP_CF_RSP_SUCCESS Notify the remote device that the connection can be established.

#define	BLE_L2CAP_CF_RSP_RFSD_INSF_AUTH Notify the remote device that the connection can not be established because of insufficient authentication.
#define	BLE_L2CAP_CF_RSP_RFSD_INSF_AUTRZ Notify the remote device that the connection can not be established because of insufficient Authorization.
#define	BLE_L2CAP_CF_RSP_RFSD_INSF_ENC_KEY Notify the remote device that the connection can not be established because of Encryption Key Size.
#define	BLE_L2CAP_CF_RSP_RFSD_INSF_ENC Notify the remote device that the connection can not be established because of Encryption.
#define	BLE_L2CAP_CF_RSP_RFSD_UNAC_PARAM Notify the remote device that the connection can not be established because the parameters is unacceptable to local device.

Typedefs

typedef void(*	ble_l2cap_cf_app_cb_t) (uint16_t event_type, ble_status_t event_result, st_ble_l2cap_cf_evt_data_t *p_event_data) ble_l2cap_cf_app_cb_t is the L2CAP Event callback function type. More...

Enumerations

enum	e_r_ble_l2cap_cf_evt_t L2CAP Event Identifier. More...

Data Structure Documentation

◆ st_ble_l2cap_conn_req_param_t

struct st_ble_l2cap_conn_req_param_t		
L2CAP CBFC Channel connection request parameters.		
Data Fields		
uint16_t	local_psm	Identifier indicating the

		protocol/profile that uses L2CAP CBFC Channel on local device.
uint16_t	remote_psm	Identifier indicating the protocol/profile that uses L2CAP CBFC Channel on remote device.
uint16_t	mtu	MTU size(byte) receivable on L2CAP CBFC Channel.
uint16_t	mps	MPS size(byte) receivable on L2CAP CBFC Channel.
uint16_t	credit	The number of LE-Frame that local device can receive.

◆ st_ble_l2cap_conn_rsp_param_t

struct st_ble_l2cap_conn_rsp_param_t		
L2CAP CBFC Channel connection response parameters.		
Data Fields		
uint16_t	lcid	CID identifying the L2CAP CBFC Channel on local device. The valid range is 0x40-0x40 + BLE_L2CAP_MAX_CBFC_PSM - 1.
uint16_t	response	The response to the connection request. Select one of the following. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_L2CAP_CF_RSP_SUCCESS(0x0000)</td><td>Notify the remote device that the connection can be established.</td></tr><tr><td>BLE_L2CAP_CF_RSP_RFSD_INSF_AUTH(0x0005)</td><td>Notify the remote device that the connection can not be established because of insufficient authentication.</td></tr><tr><td>BLE_L2CAP_CF_RSP_RFSD_INSF_AUTRZ(0x0006)</td><td>Notify the remote device that the connection can not be established</td></tr></table>

			because of insufficient Authorization.
		BLE_L2CAP_CF_RSP_RFSD_INSF_ENC_KEY (0x0007)	Notify the remote device that the connection can not be established because of Encryption Key Size.
		BLE_L2CAP_CF_RSP_RFSD_INSF_ENC(0x0008)	Notify the remote device that the connection can not be established because of Encryption.
		BLE_L2CAP_CF_RSP_RFSD_UNAC_PARAM(0x000B)	Notify the remote device that the connection can not be established because the parameters is unacceptable to local device.
uint16_t	mtu	MTU(byte) of packet that L2CAP CBFC Channel on local device can receive.	
uint16_t	mps	MPS(byte) of packet that L2CAP CBFC Channel on local device can receive.	
uint16_t	credit	The number of LE-Frame that L2CAP CBFC Channel on local device can receive.	

◆ st_ble_l2cap_cf_conn_evt_t

struct st_ble_l2cap_cf_conn_evt_t		
L2CAP CBFC Channel connection parameters.		
Data Fields		
uint16_t	cid	CID identifying the L2CAP CBFC Channel.
uint16_t	psm	PSM allocated by the cid field.

uint16_t	mtu	MTU of local/remote device.
uint16_t	mps	MPS of local/remote device.
uint16_t	credit	Credit of local/remote device.

◆ st_ble_l2cap_cf_data_evt_t

struct st_ble_l2cap_cf_data_evt_t		
Sent/Received Data parameters.		
Data Fields		
uint16_t	cid	CID identifying the L2CAP CBFC Channel that has sent or received the data .
uint16_t	psm	PSM allocated by the cid field.
uint16_t	data_len	Data length.
uint8_t *	p_data	Sent/Received data.

◆ st_ble_l2cap_cf_credit_evt_t

struct st_ble_l2cap_cf_credit_evt_t		
Credit parameters of local or remote device.		
Data Fields		
uint16_t	cid	CID identifying the L2CAP CBFC Channel.
uint16_t	psm	PSM allocated by the cid field.
uint16_t	credit	Current credit of local/remote device.

◆ st_ble_l2cap_cf_disconn_evt_t

struct st_ble_l2cap_cf_disconn_evt_t		
Disconnection parameters.		
Data Fields		
uint16_t	cid	CID identifying the L2CAP CBFC Channel that has been disconnected.

◆ st_ble_l2cap_rej_evt_t

struct st_ble_l2cap_rej_evt_t		
Command Reject parameters.		
Data Fields		
uint16_t	reason	The reason that the remote device has sent Command Reject.

uint16_t	data_1	Optional information about the reason that the remote device has sent Command Reject.
uint16_t	data_2	Optional information about the reason that the remote device has sent Command Reject.

◆ st_ble_l2cap_cf_evt_data_t

struct st_ble_l2cap_cf_evt_data_t

st_ble_l2cap_cf_evt_data_t is the type of the data notified in a L2CAP Event.

Data Fields		
uint16_t	conn_hdl	Connection handle identifying the remote device.
uint16_t	param_len	The size of L2CAP Event parameters.
void *	p_param	L2CAP Event parameters. This parameter differs in each L2CAP Event.

Typedef Documentation

◆ ble_l2cap_cf_app_cb_t

ble_l2cap_cf_app_cb_t

ble_l2cap_cf_app_cb_t is the L2CAP Event callback function type.

Parameters

[in]	event_type	The type of L2CAP Event.
[in]	event_result	The result of L2CAP Event
[in]	p_event_data	Data notified by L2CAP Event.

Returns
none

Enumeration Type Documentation

◆ e_r_ble_l2cap_cf_evt_t

enum e_r_ble_l2cap_cf_evt_t

L2CAP Event Identifier.

Enumerator	
BLE_L2CAP_EVENT_CF_CONN_CNF	After the connection request for L2CAP CBFC Channel has been sent with R_BLE_L2CAP_ReqCfConn(), when the L2CAP CBFC Channel connection response has been received, BLE_L2CAP_EVENT_CF_CONN_CNF event occurs. **Event Code: 0x5001** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_RSP_TIMEOUT(0x0011) — L2CAP Command timeout. BLE_ERR_L2CAP_PSM_NOT_SUPPORTED(0x4002) — PSM specified by R_BLE_L2CAP_ReqCfConn() is not supported. BLE_ERR_L2CAP_NO_RESOURCE(0x4004) — No resource for connection. BLE_ERR_L2CAP_INSUF_AUTHEN(0x4005) — Insufficient authentication. BLE_ERR_L2CAP_INSUF_AUTHOR(0x4006) — Insufficient authorization. BLE_ERR_L2CAP_INSUF_ENC_KEY_SIZE(0x4007) — Insufficient encryption key size. BLE_ERR_L2CAP_REFUSE_INSUF_ENC(0x4008) — Insufficient encryption. BLE_ERR_L2CAP_REFUSE_INVALID_SCID(0x4009) — Invalid Source CID. BLE_ERR_L2CAP_REFUSE_SCID_ALREADY_ALLOC(0x400A) — Source CID already allocated.

	<table><tr><td>BLE_ERR_L2CAP_REFUSE_UNACCEPTABLE_PARAM(0x400B)</td><td>Unacceptable parameters.</td></tr></table> **Event Data:** st_ble_l2cap_cf_conn_evt_t
BLE_L2CAP_EVENT_CF_CONN_IND	When a connection request for L2CPA CBFC Channel has been received from a remote device, BLE_L2CAP_EVENT_CF_CONN_IND event occurs. **Event Code: 0x5002** **result:** <table><tr><td>BLE_SUCCESS(0x0000)</td><td>Success</td></tr><tr><td>BLE_ERR_NOT_FOUND(0x000D)</td><td>CF connection request has not been received or lcid not found.</td></tr><tr><td>BLE_ERR_L2CAP_PSM_NOT_SUPPORTED(0x4002)</td><td>PSM specified by R_BLE_L2CAP_ReqCfConn() is not supported.</td></tr></table> **Event Data:** st_ble_l2cap_cf_conn_evt_t
BLE_L2CAP_EVENT_CF_DISCONN_CNF	After local device has sent a disconnection request for L2CAP CBFC Channel by R_BLE_L2CAP_DisconnectCf(), when the local device has received the response, BLE_L2CAP_EVENT_CF_DISCONN_CNF event occurs. **Event Code: 0x5003** **result:** <table><tr><td>BLE_SUCCESS(0x0000)</td><td>Success</td></tr></table> **Event Data:** st_ble_l2cap_cf_disconn_evt_t
BLE_L2CAP_EVENT_CF_DISCONN_IND	When local device has received a disconnection request for L2CAP CBFC Channel

	from the remote device, BLE_L2CAP_EVENT_CF_DISCONN_IND event occurs. Host stack automatically replies the to the disconnection request. **Event Code: 0x5004** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_l2cap_cf_disconn_evt_t
BLE_L2CAP_EVENT_CF_RX_DATA_IND	When local device has received data on L2CAP CBFC Channel, BLE_L2CAP_EVENT_CF_RX_DATA_IND event occurs. **Event Code: 0x5005** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_l2cap_cf_data_evt_t
BLE_L2CAP_EVENT_CF_LOW_RX_CRD_IND	When the credit of the L2CAP CBFC Channel has reached the Low Water Mark, BLE_L2CAP_EVENT_CF_LOW_RX_CRD_IND event occurs. **Event Code: 0x5006** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_l2cap_cf_credit_evt_t
BLE_L2CAP_EVENT_CF_TX_CRD_IND	When local device has received credit from a remote device, BLE_L2CAP_EVENT_CF_TX_CRD_IND event occurs.

	Event Code: 0x5007 **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_l2cap_cf_credit_evt_t
BLE_L2CAP_EVENT_CF_TX_DATA_CNF	When the data transmission has been completed from host stack to Controller, BLE_L2CAP_EVENT_CF_TX_DATA_CNF event occurs. **Event Code: 0x5008** **result:** BLE_SUCCESS(0x0000) Success BLE_ERR_DISCONNECTED(0x000F) While transmitting data, L2CAP CBFC Channel has been disconnected. **Event Data:** st_ble_l2cap_cf_data_evt_t
BLE_L2CAP_EVENT_CMD_REJ	When local device has received Command Reject PDU, BLE_L2CAP_EVENT_CMD_REJ event occurs. **Event Code: 0x5009** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_l2cap_rej_evt_t

Function Documentation

◆ R_BLE_L2CAP_RegisterCfPsm()

ble_status_t R_BLE_L2CAP_RegisterCfPsm (ble_l2cap_cf_app_cb_t *cb*, uint16_t *psm*, uint16_t *lwm*)

This function registers PSM that uses L2CAP CBFC Channel and a callback for L2CAP event.

Only one callback is available per PSM. Configure in each PSM the Low Water Mark of the LE-Frames that the local device can receive.
When the number of the credit reaches the Low Water Mark, BLE_L2CAP_EVENT_CF_LOW_RX_CRD_IND event is notified to the application layer.
The number of PSM is defined as BLE_L2CAP_MAX_CBFC_PSM.
The result of this API call is returned by a return value.

Parameters

[in]	cb	Callback function for L2CAP event.
[in]	psm	Identifier indicating the protocol/profile that uses L2CAP CBFC Channel.

type	range	description
Fixed, SIG assigned	0x0001 - 0x007F	PSM defined by SIG. For more information on PSM, refer Bluetooth SIG Assigned Number (https://www.bluetooth.com/specifications/assigned-numbers).
Dynamic	0x0080 - 0x00FF	Statically allocated PSM by custom protocol or dynamicall

		y allocated PSM by GATT Service.
[in]	lwm	Low Water Mark that indicates the LE-Frame numbers that the local device can receive.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The cb parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The psm parameter is out of range.
BLE_ERR_CONTEXT_FULL(0x000B)	More than BLE_L2CAP_MAX_CBFC_PSM+1 PSMs, callbacks has been registered.

◆ R_BLE_L2CAP_DeregisterCfPsm()

ble_status_t R_BLE_L2CAP_DeregisterCfPsm (uint16_t *psm*)

This function stops the use of the L2CAP CBFC Channel specified by the psm parameter and deregisters the callback function for L2CAP event.

The result of this API call is returned by a return value.

Parameters

[in]	psm	PSM that is to be stopped to use the L2CAP CBFC Channel. Set the PSM registered by R_BLE_L2CAP_RegisterCfPsm().

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_NOT_FOUND(0x000D)	The callback function allocated by the psm parameter is not found.

◆ R_BLE_L2CAP_ReqCfConn()

ble_status_t R_BLE_L2CAP_ReqCfConn (uint16_t *conn_hdl*, st_ble_l2cap_conn_req_param_t * *p_conn_req_param*)

This function sends a connection request for L2CAP CBFC Channel.

The connection response is notified by BLE_L2CAP_EVENT_CF_CONN_CNF event.
The result of this API call is returned by a return value.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device that the connection request is sent to.
[in]	p_conn_req_param	Connection request parameters.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_conn_req_param parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The mtu parameter or the mps parameter is out of range.
BLE_ERR_INVALID_STATE(0x0008)	CF Channel connection has not been established.
BLE_ERR_CONTEXT_FULL(0x000B)	New CF Channel can not be registered or other L2CAP Command is processing.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	Insufficient memory is needed to generate this function.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by conn_hdl is not found.
BLE_ERR_NOT_YET_READY(0x0012)	The psm parameter is not registered.

◆ R_BLE_L2CAP_RspCfConn()

ble_status_t R_BLE_L2CAP_RspCfConn (st_ble_l2cap_conn_rsp_param_t * *p_conn_rsp_param*)

This function replies to the connection request for L2CAP CBFC Channel from the remote device.

The connection request is notified by BLE_L2CAP_EVENT_CF_CONN_IND event. The result of this API call is returned by a return value.

Parameters

[in]	p_conn_rsp_param	Connection response parameters.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_conn_rsp_param parameter is specified as NULL.
BLE_ERR_NOT_FOUND(0x000D)	A connection request for L2CAP CBFC Channel has not been received, or CID specified by the lcid field in the p_conn_rsp_param parameter is not found.

◆ R_BLE_L2CAP_DisconnectCf()

ble_status_t R_BLE_L2CAP_DisconnectCf (uint16_t *lcid*)

This function sends a disconnection request for L2CAP CBFC Channel.

When L2CAP CBFC Channel has been disconnected, BLE_L2CAP_EVENT_CF_DISCONN_CNF event is notified to the application layer.

Parameters

[in]	lcid	CID identifying the L2CAP CBFC Channel that has been disconnected. The valid range is 0x40 - (0x40 + BLE_L2CAP_MAX_CBFC_PSM - 1).

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_OPERATION(0x0009)	CF Channel connection has not been established.
BLE_ERR_CONTEXT_FULL(0x000B)	This function was called while processing other L2CAP command.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for L2CAP Command.
BLE_ERR_NOT_FOUND(0x000D)	CID specified the lcid parameter is not found.

◆ R_BLE_L2CAP_SendCfCredit()

ble_status_t R_BLE_L2CAP_SendCfCredit (uint16_t *lcid*, uint16_t *credit*)

This function sends credit to a remote device.

In L2CAP CBFC communication, if credit is 0, the remote device stops data transmission.
Therefore when processing the received data has been completed and local device affords to receive data, the remote device is notified of the number of LE-Frame that local device can receive by this function and local device can continue to receive data from the remote device.
The result of this API call is returned by a return value.

Parameters

[in]	lcid	CID identifying the L2CAP CBFC Channel on local device that sends credit.
[in]	credit	Credit to be sent to the remote device.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The credit parameter is set to 0.
BLE_ERR_CONTEXT_FULL(0x000B)	This function was called while processing other L2CAP command.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for L2CAP Command.

◆ R_BLE_L2CAP_SendCfData()

ble_status_t R_BLE_L2CAP_SendCfData (uint16_t *conn_hdl*, uint16_t *lcid*, uint16_t *data_len*, uint8_t * *p_sdu*)

This function sends the data to a remote device via L2CAP CBFC Channel.

When the data transmission to Controller has been completed, BLE_L2CAP_EVENT_CF_TX_DATA_CNF event is notified to the application layer.

Parameters

[in]	conn_hdl	Connection handle identifying the remote device to be sent the data.
[in]	lcid	CID identifying the L2CAP CBFC Channel on local device used in the data transmission.
[in]	data_len	Length of the data.
[in]	p_sdu	Service Data Unit. Input the data length specified by the data_len parameter to the first 2 bytes (Little Endian).

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_data parameter is specified as NULL.
BLE_ERR_INVALID_ARG(0x0003)	The length parameter is out of range.
BLE_ERR_INVALID_STATE(0x0008)	CF Channel connection has not been established or the data whose length exceeds the MTU has been sent.
BLE_ERR_ALREADY_IN_PROGRESS(0x000A)	Data transmission has been already started.
BLE_ERR_CONTEXT_FULL(0x000B)	L2CAP task queue is full.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for L2CAP Command.
BLE_ERR_NOT_FOUND(0x000D)	CID specified the lcid parameter is not found.
BLE_ERR_INVALID_HDL(0x000E)	The remote device specified by the conn_hdl parameter is not found.

4.3.2.6 VS

Interfaces » BLE Interface

Functions

ble_status_t	R_BLE_VS_Init (ble_vs_app_cb_t vs_cb) This function initializes Vendor Specific API and registers a callback function for Vendor Specific Event. More...
ble_status_t	R_BLE_VS_StartTxTest (st_ble_vs_tx_test_param_t *p_tx_test_param) This function starts extended Transmitter Test. More...
ble_status_t	R_BLE_VS_StartRxTest (st_ble_vs_rx_test_param_t *p_rx_test_param) This function starts extended Receiver Test. More...
ble_status_t	R_BLE_VS_EndTest (void) This function terminates the extended transmitter or receiver test. More...
ble_status_t	R_BLE_VS_SetTxPower (uint16_t conn_hdl, uint8_t tx_power) This function configures transmit power. More...
ble_status_t	R_BLE_VS_GetTxPower (uint16_t conn_hdl) This function gets transmit power. More...
ble_status_t	R_BLE_VS_SetCodingScheme (uint8_t coding_scheme) This function configure default Coding scheme(S=8 or S=2) that is used in the case of selecting Coded PHY in Primary advertising PHY or Secondary advertising PHY advertising or request for link establishment. More...
ble_status_t	R_BLE_VS_SetRfControl (st_ble_vs_set_rf_ctrl_param_t *p_rf_ctrl) This function performs power control on RF. More...
ble_status_t	R_BLE_VS_SetBdAddr (uint8_t area, st_ble_dev_addr_t *p_addr) This function sets public/random address of local device to the area specified by the parameter. More...

ble_status_t	R_BLE_VS_GetBdAddr (uint8_t area, uint8_t addr_type) This function gets currently configured public/random address. More...
ble_status_t	R_BLE_VS_GetRand (uint8_t rand_size) This function generates 4-16 bytes of random number used in creating keys. More...
ble_status_t	R_BLE_VS_StartTxFlowEvtNtf (void) This function starts the notification(BLE_VS_EVENT_TX_FLOW_STATE_CHG event) of the state transition of TxFlow. More...
ble_status_t	R_BLE_VS_StopTxFlowEvtNtf (void) This function stops the notification(BLE_VS_EVENT_TX_FLOW_STATE_CHG event) of the state transition of TxFlow. More...
ble_status_t	R_BLE_VS_GetTxBufferNum (uint32_t *p_buffer_num) This function retrieves the number of the available transmission packet buffers. More...
ble_status_t	R_BLE_VS_SetTxLimit (uint32_t tx_queue_lwm, uint32_t tx_queue_hwm) This function sets the threshold for notifying the application layer of the TxFlow state. More...
ble_status_t	R_BLE_VS_SetScanChMap (uint16_t ch_map) This function sets the scan channel map. More...
ble_status_t	R_BLE_VS_GetScanChMap (void) This function gets currently scan channel map. More...

Detailed Description

Data Structures

struct	st_ble_vs_tx_test_param_t

	This is the extended transmitter test parameters used in R_BLE_VS_StartTxTest(). More...
struct	st_ble_vs_rx_test_param_t This is the extended receiver test parameters used in R_BLE_VS_StartRxTest(). More...
struct	st_ble_vs_set_rf_ctrl_param_t This is the RF parameters used in R_BLE_VS_SetRfControl(). More...
struct	st_ble_vs_test_end_evt_t This structure notifies that the extended test has been terminated. More...
struct	st_ble_vs_set_tx_pwr_comp_evt_t This structure notifies that tx power has been set. More...
struct	st_ble_vs_get_tx_pwr_comp_evt_t This structure notifies that tx power has been retrieved. More...
struct	st_ble_vs_set_rf_ctrl_comp_evt_t This structure notifies that RF has been configured. More...
struct	st_ble_vs_get_bd_addr_comp_evt_t This structure notifies that BD_ADDR has been retrieved. More...
struct	st_ble_vs_get_rand_comp_evt_t This structure notifies that random number has been generated. More...
struct	st_ble_vs_tx_flow_chg_evt_t This structure notifies that the state transition of TxFlow has been changed. More...
struct	st_ble_vs_evt_data_t st_ble_vs_evt_data_t is the type of the data notified in a Vendor

	Specific Event. More...
struct	st_ble_vs_get_scan_ch_map_comp_evt_t This structure notifies that current scan channel map. More...

Macros

#define	BLE_VS_TX_POWER_HIGH High power level.
#define	BLE_VS_TX_POWER_MID Middle power level.
#define	BLE_VS_TX_POWER_LOW Low power level.
#define	BLE_VS_ADDR_AREA_REG Address in register is written or read.
#define	BLE_VS_ADDR_AREA_DATA_FLASH Address in DataFlash is written or read.
#define	BLE_VS_EH_TX_PL_PRBS9 PRBS9 sequence '11111111100000111101..'.
#define	BLE_VS_EH_TX_PL_11110000 Repeated '11110000'.
#define	BLE_VS_EH_TX_PL_10101010 Repeated '10101010'.
#define	BLE_VS_EH_TX_PL_PRBS15 PRBS15 sequence.
#define	BLE_VS_EH_TX_PL_11111111

	Repeated '11111111'.
#define	BLE_VS_EH_TX_PL_00000000
	Repeated '00000000'.
#define	BLE_VS_EH_TX_PL_00001111
	Repeated '00001111'.
#define	BLE_VS_EH_TX_PL_01010101
	Repeated '01010101'.
#define	BLE_VS_EH_TEST_PHY_1M
	1M PHY used in Transmitter/Receiver test.
#define	BLE_VS_EH_TEST_PHY_2M
	2M PHY used in Transmitter/Receiver test.
#define	BLE_VS_EH_TEST_PHY_CODED
	Coded PHY used in Receiver test.
#define	BLE_VS_EH_TEST_PHY_CODED_S_8
	Coded PHY(S=8) used in Transmitter test.
#define	BLE_VS_EH_TEST_PHY_CODED_S_2
	Coded PHY(S=2) used in Transmitter test.
#define	BLE_VS_RF_OFF
	RF power off.
#define	BLE_VS_RF_ON
	RF power on.
#define	BLE_VS_RF_INIT_PARAM_NOT_CHG
	The parameters are not changed in RF power on.

#define	BLE_VS_RF_INIT_PARAM_CHG The parameters are changed in RF power on.
#define	BLE_VS_CS_PRIM_ADV_S_8 Coding scheme for Primary Advertising PHY(S=8).
#define	BLE_VS_CS_PRIM_ADV_S_2 Coding scheme for Primary Advertising PHY(S=2).
#define	BLE_VS_CS_SECOND_ADV_S_8 Coding scheme for Secondary Advertising PHY(S=8).
#define	BLE_VS_CS_SECOND_ADV_S_2 Coding scheme for Secondary Advertising PHY(S=2).
#define	BLE_VS_CS_CONN_S_8 Coding scheme for request for link establishment(S=8).
#define	BLE_VS_CS_CONN_S_2 Coding scheme for request for link establishment(S=2).
#define	BLE_VS_TX_FLOW_CTL_ON It means that the number of buffer has reached the High Water Mark from flow off state.
#define	BLE_VS_TX_FLOW_CTL_OFF It means that the number of buffer has reached the Low Water Mark from flow on state.

Typedefs

typedef void(*	ble_vs_app_cb_t) (uint16_t event_type, ble_status_t event_result, st_ble_vs_evt_data_t *p_event_data) ble_vs_app_cb_t is the Vendor Specific Event callback function type. More...

Enumerations

enum	e_r_ble_vs_evt_t
	Vendor Specific Event Identifier. More...

Data Structure Documentation

◆ st_ble_vs_tx_test_param_t

struct st_ble_vs_tx_test_param_t

This is the extended transmitter test parameters used in R_BLE_VS_StartTxTest().

Data Fields		
uint8_t	ch	Channel used in Tx test.
uint8_t	test_data_len	Length(in bytes) of the packet used in Tx Test.
uint8_t	packet_payload	Packet Payload.
uint8_t	phy	Transmitter PHY used in test.
uint8_t	tx_power	Tx Power Level used in DTM Tx Test.
uint8_t	option	Option.
uint16_t	num_of_packet	The number of packet to be sent.

◆ st_ble_vs_rx_test_param_t

struct st_ble_vs_rx_test_param_t

This is the extended receiver test parameters used in R_BLE_VS_StartRxTest().

Data Fields		
uint8_t	ch	Channel used in Rx test.
uint8_t	phy	Receiver PHY used in the test.

◆ st_ble_vs_set_rf_ctrl_param_t

struct st_ble_vs_set_rf_ctrl_param_t

This is the RF parameters used in R_BLE_VS_SetRfControl().

Data Fields		
uint8_t	power	RF power on/off.
uint8_t	option	This field indicates whether the parameters change in RF power on.
uint8_t	clval	RF rapid clock frequency adjust value(OSC internal CL adjust).

uint8_t	slow_clock	RF slow clock configurations.
uint8_t	tx_power	Set tx power in power on.
uint8_t	rf_option	Set RF option.

◆ st_ble_vs_test_end_evt_t

struct st_ble_vs_test_end_evt_t		
This structure notifies that the extended test has been terminated.		
Data Fields		
uint16_t	num_of_packet	The number of packet successfully received in the receiver test.
uint16_t	num_of_crc_err_packet	The number of CRC error packets in the receiver test.
int8_t	ave_rssi	Average RSSI(dBm) in the receiver test.
int8_t	max_rssi	Maximum RSSI(dBm) in the receiver test.
int8_t	min_rssi	Minimum RSSI(dBm) in the receiver test.

◆ st_ble_vs_set_tx_pwr_comp_evt_t

struct st_ble_vs_set_tx_pwr_comp_evt_t		
This structure notifies that tx power has been set.		
Data Fields		
uint16_t	conn_hdl	Connection handle that identifying the link whose tx power has been set.
int8_t	curr_tx_pwr	Tx power that has been set(dBm).

◆ st_ble_vs_get_tx_pwr_comp_evt_t

struct st_ble_vs_get_tx_pwr_comp_evt_t		
This structure notifies that tx power has been retrieved.		
Data Fields		
uint16_t	conn_hdl	Connection handle that identifying the link whose tx power has been retrieved.
int8_t	curr_tx_pwr	Current tx power(dBm).
int8_t	max_tx_pwr	Maximum tx power(dBm).

◆ st_ble_vs_set_rf_ctrl_comp_evt_t

struct st_ble_vs_set_rf_ctrl_comp_evt_t		
This structure notifies that RF has been configured.		
Data Fields		
uint8_t	ctrl	The result of RF power control.

◆ st_ble_vs_get_bd_addr_comp_evt_t

struct st_ble_vs_get_bd_addr_comp_evt_t		
This structure notifies that BD_ADDR has been retrieved.		
Data Fields		
uint8_t	area	The area that public/random address has been retrieved. value / description BLE_VS_ADDR_AREA_REG(0x00): Register. BLE_VS_ADDR_AREA_DATA_FLASH(0x01): Data Flash.
st_ble_dev_addr_t	addr	The address that has been retrieved.

◆ st_ble_vs_get_rand_comp_evt_t

struct st_ble_vs_get_rand_comp_evt_t		
This structure notifies that random number has been generated.		
Data Fields		
uint8_t	rand_size	Length of random number.
uint8_t *	p_rand	Random number.

◆ st_ble_vs_tx_flow_chg_evt_t

struct st_ble_vs_tx_flow_chg_evt_t		
This structure notifies that the state transition of TxFlow has been changed.		
Data Fields		
uint8_t	state	The state of the flow control. value / description BLE_VS_TX_FLOW_CTL_ON(0x00): The number of buffer has reached the High Water Mark from flow off state.

		BLE_VS_TX_FLOW_CTL_OFF(0x01)	The number of buffer has reached the Low Water Mark from flow on state.
uint32_t	buffer_num	The number of the current transmission buffers.	

◆ st_ble_vs_evt_data_t

struct st_ble_vs_evt_data_t		
st_ble_vs_evt_data_t is the type of the data notified in a Vendor Specific Event.		
Data Fields		
uint16_t	param_len	The size of Vendor Specific Event parameters.
void *	p_param	Vendor Specific Event parameters. This parameter differs in each Vendor Specific Event.

◆ st_ble_vs_get_scan_ch_map_comp_evt_t

struct st_ble_vs_get_scan_ch_map_comp_evt_t		
This structure notifies that current scan channel map.		
Data Fields		
uint8_t	ch_map	The result of current scan channel map.

Typedef Documentation

◆ ble_vs_app_cb_t

ble_vs_app_cb_t

ble_vs_app_cb_t is the Vendor Specific Event callback function type.

Parameters

[in]	event_type	The type of Vendor Specific Event.
[in]	event_result	The result of API call which generates the Vendor Specific Event.
[in]	p_event_data	Data notified in the Vendor Specific Event.

Returns

none

Enumeration Type Documentation

◆ e_r_ble_vs_evt_t

enum e_r_ble_vs_evt_t

Vendor Specific Event Identifier.

Enumerator	
BLE_VS_EVENT_SET_TX_POWER	This event notifies that the tx power has been set by R_BLE_VS_SetTxPower(). **Event Code: 0x8001** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — The tx_power parameter specified by R_BLE_VS_SetTxPower() is out of range. BLE_ERR_INVALID_HDL(0x000E) — The link identified with the conn_hdl specified by R_BLE_VS_SetTxPower() is not found. **Event Data:** st_ble_vs_set_tx_pwr_comp_evt_t
BLE_VS_EVENT_GET_TX_POWER	This event notifies that the tx power has been retrieved by R_BLE_VS_GetTxPower(). **Event Code: 0x8002** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_HDL(0x000E) — The link identified with the conn_hdl specified by R_BLE_VS_GetTxPower() is not found. **Event Data:**

	st_ble_vs_get_tx_pwr_comp_evt_t
BLE_VS_EVENT_TX_TEST_START	This event notifies that the extended transmitter test has been started by R_BLE_VS_StartTxTest(). **Event Code: 0x8003** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — The parameter specified by R_BLE_VS_StartTxTest() is out of range. **Event Data:** none
BLE_VS_EVENT_TX_TEST_TERM	This event notifies that the number specified by R_BLE_VS_StartTxTest() of packets has been sent. **Event Code: 0x8004** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** none
BLE_VS_EVENT_RX_TEST_START	This event notifies that the extended receiver test has been started by R_BLE_VS_StartRxTest(). **Event Code: 0x8005** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — The parameter specified by R_BLE_VS_StartRxTest() is out of range.

	Event Data: none
BLE_VS_EVENT_TEST_END	This event notifies that the extended test has been terminated by R_BLE_VS_EndTest(). **Event Code: 0x8006** **result:** BLE_SUCCESS(0x0000) — Success **Event Data:** st_ble_vs_test_end_evt_t
BLE_VS_EVENT_SET_CODING_SCHEME_COMP	This event notifies that the coding scheme has been configured by R_BLE_VS_SetCodingScheme(). **Event Code: 0x8007** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — The coding_scheme parameter specified by R_BLE_VS_SetCodingScheme() is out of range. **Event Data:** none
BLE_VS_EVENT_RF_CONTROL_COMP	This event notifies that the RF has been configured by R_BLE_VS_SetRfControl(). **Event Code: 0x8008** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — The parameter specified by R_BLE_VS_SetRfControl() is out of

	range. BLE_ERR_INVALID_OPERATION(0x0009) — During the power on or the power off, the same power state is specified by R_BLE_VS_SetRfControl(). **Event Data:** st_ble_vs_set_rf_ctrl_comp_evt_t
BLE_VS_EVENT_SET_ADDR_COMP	This event notifies that public/random address has been set by R_BLE_VS_SetBdAddr(). **Event Code: 0x8009** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — The area parameter or the type field in the p_addr parameter specified by R_BLE_VS_SetBdAddr() is out of range. **Event Data:** none
BLE_VS_EVENT_GET_ADDR_COMP	This event notifies that public/random address has been retrieved by R_BLE_VS_GetBdAddr(). **Event Code: 0x800A** **result:** BLE_SUCCESS(0x0000) — Success BLE_ERR_INVALID_ARG(0x0003) — The area parameter or the type field in the p_addr parameter specified by R_BLE_VS_GetBdAddr() is out of range.

	Event Data: st_ble_vs_get_bd_addr_comp_evt_t
BLE_VS_EVENT_GET_RAND	This event notifies the application layer that random number has been generated by R_BLE_VS_GetRand(). **Event Code: 0x800B** **result:** BLE_SUCCESS(0x0000) Success BLE_ERR_INVALID_ARG(0x0003) The rand_size parameter specified by R_BLE_VS_GetRand() is out of range. **Event Data:** st_ble_vs_get_rand_comp_evt_t
BLE_VS_EVENT_TX_FLOW_STATE_CHG	This event notifies the application layer of the state transition of TxFlow. **Event Code: 0x800C** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_vs_tx_flow_chg_evt_t
BLE_VS_EVENT_FAIL_DETECT	This event notifies a failure occurs in RF. After receiving the event, reset MCU or RF. **Event Code: 0x800D** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** None
BLE_VS_EVENT_SET_SCAN_CH_MAP	This event notifies that scan channel map has

	been set by R_BLE_VS_SetScanChMap(). **Event Code: 0x800E** **result:** BLE_SUCCESS(0x0000) Success BLE_ERR_INVALID_ARG(0x0003) The ch_map parameter specified by R_BLE_VS_SetScanChMap() is out of range. **Event Data:** none
BLE_VS_EVENT_GET_SCAN_CH_MAP	This event notifies that scan channel map has been retrieved by R_BLE_VS_GetScanChMap(). **Event Code: 0x800F** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** st_ble_vs_get_scan_ch_map_comp_evt_t
BLE_VS_EVENT_INVALID	Invalid VS Event. **Event Code: 0x80FF** **result:** BLE_SUCCESS(0x0000) Success **Event Data:** none

Function Documentation

◆ R_BLE_VS_Init()

ble_status_t R_BLE_VS_Init (ble_vs_app_cb_t *vs_cb*)

This function initializes Vendor Specific API and registers a callback function for Vendor Specific Event.

The result of this API call is returned by a return value.

Parameters

[in]	vs_cb	Callback function to be registered.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The vs_cb parameter is specified as NULL.
BLE_ERR_CONTEXT_FULL(0x000B)	Callback function has already been registered.

◆ R_BLE_VS_StartTxTest()

ble_status_t R_BLE_VS_StartTxTest (st_ble_vs_tx_test_param_t * *p_tx_test_param*)

This function starts extended Transmitter Test.

The following extended transmitter test functions of DTM Tx are supported by this function.

- Tx Power
- Tx Modulation Enable/Modulation Disable
- Tx packet transmission/continuous transmission
- Tx packets count

The result of this API call is notified in BLE_VS_EVENT_TX_TEST_START event.
If the num_of_packet field in the p_tx_test_param parameter is other than 0x0000, BLE_VS_EVENT_TX_TEST_TERM event notifies the application layer that the number of packet has been sent.
If R_BLE_VS_EndTest() is called before the specified number of packets completions, BLE_VS_EVENT_TX_TEST_TERM event is not notified to the application layer.

The condition that phy field in the p_tx_test_param parameter is BLE_VS_EH_TEST_PHY_CODED_S_8(0x03) and option field is modulation(bit0:0) & continuous transmission(bit1:1) is not supported.

Parameters

[in]	p_tx_test_param	Tx Test parameters.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_tx_test_param parameter is specified as NULL.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

◆ R_BLE_VS_StartRxTest()

ble_status_t R_BLE_VS_StartRxTest (st_ble_vs_rx_test_param_t * *p_rx_test_param*)

This function starts extended Receiver Test.

The result of this API call is notified in BLE_VS_EVENT_RX_TEST_START event. The following extended receiver test functions of DTM Rx are supported by this function.

- Calculating the maximum, the minimum and the average of RSSI in the receiver test.
- The number of CRC error packets in the receiver test.

The transmitter is configured to one of the following, the receiver can't receive the packets by this function.

- Tx Non-Modulation Enable
- Tx continuous transmission
  After R_BLE_VS_EndTest() has been called, the receiver test result value are notified in BLE_VS_EVENT_TEST_END event.

Parameters

[in]	p_rx_test_param	The extended receiver test parameters.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_rx_test_param parameter is specified as NULL.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

◆ R_BLE_VS_EndTest()

ble_status_t R_BLE_VS_EndTest (void)

This function terminates the extended transmitter or receiver test.

The result of this API call is notified in BLE_VS_EVENT_TEST_END event. In case of extended receiver test, this event notifies the application layer of the result of the extended receiver test.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

◆ R_BLE_VS_SetTxPower()

ble_status_t R_BLE_VS_SetTxPower (uint16_t *conn_hdl*, uint8_t *tx_power*)

This function configures transmit power.

This function configures the following transmit power.

- The transmit power used in sending advertising PDU, scan request PDU, connection request PDU (in not connected state)
- The transmit power used in sending PDU in connected state. When configuring the transmit power used in not connected state, set the conn_hdl parameter to BLE_GAP_INIT_CONN_HDL(0xFFFF).
  When the transmit power used in connected state is configured, set the conn_hdl parameter to the connection handle of the link.
  Select one of the following transmit power levels.
- High
- Middle
- Low

Max transmit power of "High" is dependent on the configuration of the firmware. The result of this API call is notified in BLE_VS_EVENT_SET_TX_POWER event.

Parameters

[in]	conn_hdl	Connection handle identifying the link whose transmit power to be configured. If non connected state, set BLE_GAP_INIT_CONN_HDL(0xFFFF).
[in]	tx_power	Transmission power. Select one of the following. • BLE_VS_TX_POWER_HIGH(0x00) • BLE_VS_TX_POWER_MID(0x01) • BLE_VS_TX_POWER_LOW(0x02)

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

◆ R_BLE_VS_GetTxPower()

ble_status_t R_BLE_VS_GetTxPower (uint16_t *conn_hdl*)

This function gets transmit power.

This function gets the following transmit power.

- The transmit power used in sending advertising PDU, scan request PDU, connection request PDU (in not connected state)
- The transmit power used in sending PDU in connected state. When getting the transmit power used in not connected state, set the conn_hdl parameter to BLE_GAP_INIT_CONN_HDL(0xFFFF).
  When the transmit power used in connected state is retrieved, set the conn_hdl parameter to the connection handle of the link. The result of this API call is notified in BLE_VS_EVENT_GET_TX_POWER event.

Parameters

[in]	conn_hdl	Connection handle identifying the link whose transmit power to be retrieved. If non connected state, set BLE_GAP_INIT_CONN_HDL(0xFFFF).

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

◆ R_BLE_VS_SetCodingScheme()

ble_status_t R_BLE_VS_SetCodingScheme (uint8_t *coding_scheme*)

This function configure default Coding scheme(S=8 or S=2) that is used in the case of selecting Coded PHY in Primary advertising PHY or Secondary advertising PHY advertising or request for link establishment.

After setting the default Coding scheme by this function, configure the advertising parameters by R_BLE_GAP_SetAdvParam() or send a request for link establishment.
The result of this API call is notified in BLE_VS_EVENT_SET_CODING_SCHEME_COMP event.

Parameters

[in]	coding_scheme	Coding scheme for Primary advertising PHY, Secondary advertising PHY, request for link establishment.The coding_scheme field is set to a bitwise OR of the following values.

bit	description
bit0	Coding scheme for Primary Advertising PHY(0:S=8/1:S=2).
bit1	Coding scheme for Secondary Advertising PHY(0:S=8/1:S=2).
bit2	Coding scheme for request for link establishment(0:S=8/1:S=2).
All other bits	Reserved for future use.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

◆ R_BLE_VS_SetRfControl()

ble_status_t R_BLE_VS_SetRfControl (st_ble_vs_set_rf_ctrl_param_t * *p_rf_ctrl*)

This function performs power control on RF.

If BLE communication is not used for a long time, RF reduces the power consumption by moving to the RF Power-Down Mode.
When RF power on, RF initialization processing is executed.
After RF power off by this function, API functions other than this are not available until RF power on again.
The result of this API call is notified in BLE_VS_EVENT_RF_CONTROL_COMP event. After RF power on again with this function, call R_BLE_GAP_Terminate(), R_BLE_GAP_Init() in order to restart the host stack.

Parameters

[in]	p_rf_ctrl	RF parameters.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_rf_ctrl parameter is specified as NULL.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

◆ R_BLE_VS_SetBdAddr()

ble_status_t R_BLE_VS_SetBdAddr (uint8_t *area*, st_ble_dev_addr_t * *p_addr*)

This function sets public/random address of local device to the area specified by the parameter.

If the address is written in non-volatile area, the address is used as default address on the next MCU reset.
For more information on the random address, refer to Core Specification Vol 6, PartB, "1.3.2 Random Device Address".
The result of this API call is notified in BLE_VS_EVENT_SET_ADDR_COMP event.

Parameters

[in]	area	The area that the address is to be written in. Select one of the following. macro / description: BLE_VS_ADDR_AREA_REG(0x00): Address writing to non-volatile area is not performed. Only the address in register is written. BLE_VS_ADDR_AREA_DATA_FLASH(0x01): Address wiring to DataFlash area is performed.
[in]	p_addr	The address to be set to the area. Set BLE_GAP_ADDR_PUBLIC(0x00) or BLE_GAP_ADDR_RAND(0x01) to the type field in the p_addr parameter.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_addr parameter is specified as NULL.
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

◆ R_BLE_VS_GetBdAddr()

ble_status_t R_BLE_VS_GetBdAddr (uint8_t *area*, uint8_t *addr_type*)

This function gets currently configured public/random address.

The area parameter specifies the place where this function retrieves public/random address. The result of this API call is notified in BLE_VS_EVENT_GET_ADDR_COMP event.

Parameters

[in]	area	The area that the address is to be retrieved. Select one of the following. macro / description: BLE_VS_ADDR_AREA_REG(0x00) — Retrieve the address in register. BLE_VS_ADDR_AREA_DATA_FLASH(0x01) — Retrieve the address in DataFlash area.
[in]	addr_type	The address type that is type of the address to be retrieved. macro / description: BLE_GAP_ADDR_PUBLIC(0x00) — Public address. BLE_GAP_ADDR_RAND(0x01) — Random address.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

◆ R_BLE_VS_GetRand()

ble_status_t R_BLE_VS_GetRand (uint8_t *rand_size*)

This function generates 4-16 bytes of random number used in creating keys.

The result of this API call is notified in BLE_VS_EVENT_GET_RAND event.

Parameters

[in]	rand_size	Length of the random number (byte). The valid range is 4<=rand_size<=16.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_STATE(0x0008)	The task for host stack is not running.
BLE_ERR_MEM_ALLOC_FAILED(0x000C)	There are no memories for Vendor Specific Command.

◆ R_BLE_VS_StartTxFlowEvtNtf()

ble_status_t R_BLE_VS_StartTxFlowEvtNtf (void)

This function starts the notification(BLE_VS_EVENT_TX_FLOW_STATE_CHG event) of the state transition of TxFlow.

If the number of the available transmission packet buffers is the following, BLE_VS_EVENT_TX_FLOW_STATE_CHG event notifies the application layer of the state of the TxFlow.

- The number of the available transmission packet buffers is less than Low Water Mark.
- The number of the available transmission packet buffers is more than High Water Mark. The result of this API call is returned by a return value.

Return values

BLE_SUCCESS(0x0000)	Success

◆ R_BLE_VS_StopTxFlowEvtNtf()

ble_status_t R_BLE_VS_StopTxFlowEvtNtf (void)

This function stops the notification(BLE_VS_EVENT_TX_FLOW_STATE_CHG event) of the state transition of TxFlow.

The result of this API call is returned by a return value.

Return values

BLE_SUCCESS(0x0000)	Success

◆ R_BLE_VS_GetTxBufferNum()

ble_status_t R_BLE_VS_GetTxBufferNum (uint32_t * *p_buffer_num*)

This function retrieves the number of the available transmission packet buffers.

The maximum number of the available buffers is 10.
The result of this API call is returned by a return value.

Parameters

[out]	p_buffer_num	The number of the available transmission packet buffers.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_PTR(0x0001)	The p_buffer_num parameter is specified as NULL.

◆ R_BLE_VS_SetTxLimit()

ble_status_t R_BLE_VS_SetTxLimit (uint32_t *tx_queue_lwm*, uint32_t *tx_queue_hwm*)

This function sets the threshold for notifying the application layer of the TxFlow state.

Call this function before the notification(BLE_VS_EVENT_TX_FLOW_STATE_CHG event) has been started by R_BLE_VS_StartTxFlowEvtNtf().
The result is returned from this API.
Vendor Specific API supports the flow control function(TxFlow) for the transmission on L2CAP fixed channel in Basic Mode such as GATT.
Host stack has 10 transmission packet buffers for the transmission.
When the number of the available transmission packet buffers has been less than Low Water Mark, the state of TxFlow transmits into the TxFlow OFF state from the TxFlow ON state that is the initial state and host stack notifies the application layer of timing to stop packet transmission.
When host stack has sent the transmission packets to Controller and the number of the available transmission packet buffers has been more than High Water Mark, the state of TxFlow transmits into the TxFlow ON state from the TxFlow OFF state and host stack notifies the application layer of

timing to restart packet transmission.
It is possible to perform flow control on a fixed channel by using the event notification.

Parameters

[in]	tx_queue_lwm	Low Water Mark. Set 0-9 less than tx_queue_hwm to the parameter. When the number of the available transmission packet buffers has been less than the value specified by the tx_queue_lwm parameter, host stack notifies the application layer of the timing to stop packet transmission.
[in]	tx_queue_hwm	High Water Mark. Set 1-10 more than tx_queue_lwm to the parameter. When the number of the available transmission packet buffers has been more than the value specified by the tx_queue_hwm parameter, host stack notifies the application layer of the timing to restart packet transmission.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The tx_queue_lwm parameter or the tx_queue_hwm parameter is out of range.

◆ R_BLE_VS_SetScanChMap()

ble_status_t R_BLE_VS_SetScanChMap (uint16_t *ch_map*)

This function sets the scan channel map.

Set specify the scan channel for use.
At least one channel must be enabled.

Note

Calling this API while Scan is already running will not change the channel map.

Parameters

[in]	ch_map	Specify the channel map for use. **bit** / **description** bit0: Enable channel 37 for use (0:disable, 1:enable) bit1: Enable channel 38 for use (0:disable, 1:enable) bit2: Enable channel 39 for use (0:disable, 1:enable) All other bits: Reserved for future use.

Return values

BLE_SUCCESS(0x0000)	Success
BLE_ERR_INVALID_ARG(0x0003)	The ch_map parameter is out of range.

◆ R_BLE_VS_GetScanChMap()

ble_status_t R_BLE_VS_GetScanChMap (void)

This function gets currently scan channel map.

The result of this API call is notified in BLE_VS_EVENT_GET_SCAN_CH_MAP event.

Return values

BLE_SUCCESS(0x0000)	Success

4.3.3 CAC Interface

Interfaces

Detailed Description

Interface for clock frequency accuracy measurements.

Summary

The interface for the clock frequency accuracy measurement circuit (CAC) peripheral is used to check a system clock frequency with a reference clock signal by counting the number of pulses of the clock to be measured.

Implemented by: Clock Frequency Accuracy Measurement Circuit (r_cac)

Data Structures

struct	cac_ref_clock_config_t
struct	cac_meas_clock_config_t
struct	cac_callback_args_t
struct	cac_cfg_t
struct	cac_api_t
struct	cac_instance_t

Typedefs

typedef void	cac_ctrl_t

Enumerations

enum	cac_event_t
enum	cac_clock_type_t
enum	cac_clock_source_t
enum	cac_ref_divider_t
enum	cac_ref_digfilter_t
enum	cac_ref_edge_t
enum	cac_meas_divider_t

Data Structure Documentation

◆ cac_ref_clock_config_t

struct cac_ref_clock_config_t

Structure defining the settings that apply to reference clock configuration.

Data Fields

cac_ref_divider_t	divider	Divider specification for the Reference clock.
cac_clock_source_t	clock	Clock source for the Reference clock.
cac_ref_digfilter_t	digfilter	Digital filter selection for the CACREF ext clock.
cac_ref_edge_t	edge	Edge detection for the Reference clock.

◆ cac_meas_clock_config_t

struct cac_meas_clock_config_t

Structure defining the settings that apply to measurement clock configuration.

Data Fields

cac_meas_divider_t	divider	Divider specification for the Measurement clock.
cac_clock_source_t	clock	Clock source for the Measurement clock.

◆ cac_callback_args_t

struct cac_callback_args_t

Callback function parameter data

Data Fields

cac_event_t	event	The event can be used to identify what caused the callback.
void const *	p_context	Value provided in configuration structure.

◆ cac_cfg_t

struct cac_cfg_t	
CAC Configuration	
Data Fields	
cac_ref_clock_config_t	cac_ref_clock
	Reference clock specific settings.
cac_meas_clock_config_t	cac_meas_clock
	Measurement clock specific settings.
uint16_t	cac_upper_limit
	The upper limit counter threshold.
uint16_t	cac_lower_limit
	The lower limit counter threshold.
IRQn_Type	mendi_irq
	Measurement End IRQ number.
IRQn_Type	ovfi_irq
	Measurement Overflow IRQ number.
IRQn_Type	ferri_irq
	Frequency Error IRQ number.

uint8_t	mendi_ipl
	Measurement end interrupt priority.
uint8_t	ovfi_ipl
	Overflow interrupt priority.
uint8_t	ferri_ipl
	Frequency error interrupt priority.
void(*	p_callback)(cac_callback_args_t *p_args)
	Callback provided when a CAC interrupt ISR occurs.
void const *	p_context
	Passed to user callback in cac_callback_args_t.
void const *	p_extend
	CAC hardware dependent configuration */.

◆ cac_api_t

struct cac_api_t	
CAC functions implemented at the HAL layer API	
Data Fields	
fsp_err_t(*	open)(cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)
fsp_err_t(*	startMeasurement)(cac_ctrl_t *const p_ctrl)
fsp_err_t(*	stopMeasurement)(cac_ctrl_t *const p_ctrl)
fsp_err_t(*	read)(cac_ctrl_t *const p_ctrl, uint16_t *const p_counter)

fsp_err_t(*	callbackSet)(cac_ctrl_t *const p_api_ctrl, void(*p_callback)(cac_callback_args_t *), void const *const p_context, cac_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(cac_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* cac_api_t::open) (cac_ctrl_t *const p_ctrl, cac_cfg_t const *const p_cfg)

Open function for CAC device.

Parameters

[out]	p_ctrl	Pointer to CAC device control. Must be declared by user.
[in]	cac_cfg_t	Pointer to CAC configuration structure.

◆ startMeasurement

fsp_err_t(* cac_api_t::startMeasurement) (cac_ctrl_t *const p_ctrl)

Begin a measurement for the CAC peripheral.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ stopMeasurement

fsp_err_t(* cac_api_t::stopMeasurement) (cac_ctrl_t *const p_ctrl)

End a measurement for the CAC peripheral.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ read

fsp_err_t(* cac_api_t::read) (cac_ctrl_t *const p_ctrl, uint16_t *const p_counter)

Read function for CAC peripheral.

Parameters

[in]	p_ctrl	Control for the CAC device context.
[in]	p_counter	Pointer to variable in which to store the current CACNTBR register contents.

◆ callbackSet

fsp_err_t(* cac_api_t::callbackSet) (cac_ctrl_t *const p_api_ctrl, void(*p_callback)(cac_callback_args_t *), void const *const p_context, cac_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_CAC_CallbackSet()

Parameters

[in]	p_ctrl	Control block set in cac_api_t::open call
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* cac_api_t::close) (cac_ctrl_t *const p_ctrl)

Close function for CAC device.

Parameters

[in]	p_ctrl	Pointer to CAC device control.

◆ cac_instance_t

struct cac_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
cac_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
cac_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
cac_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ cac_ctrl_t

typedef void cac_ctrl_t
CAC control block. Allocate an instance specific control block to pass into the CAC API calls. **Implemented as** ◦ cac_instance_ctrl_t

Enumeration Type Documentation

◆ cac_event_t

enum cac_event_t	
Event types returned by the ISR callback when used in CAC interrupt mode	
Enumerator	
CAC_EVENT_FREQUENCY_ERROR	Frequency error.
CAC_EVENT_MEASUREMENT_COMPLETE	Measurement complete.
CAC_EVENT_COUNTER_OVERFLOW	Counter overflow.

◆ cac_clock_type_t

enum cac_clock_type_t	
Enumeration of the two possible clocks.	
Enumerator	
CAC_CLOCK_MEASURED	Measurement clock.
CAC_CLOCK_REFERENCE	Reference clock.

◆ cac_clock_source_t

enum cac_clock_source_t	
Enumeration of the possible clock sources for both the reference and measurement clocks.	
Enumerator	
CAC_CLOCK_SOURCE_MAIN_OSC	Main clock oscillator.
CAC_CLOCK_SOURCE_SUBCLOCK	Sub-clock.
CAC_CLOCK_SOURCE_HOCO	HOCO (High speed on chip oscillator)
CAC_CLOCK_SOURCE_MOCO	MOCO (Middle speed on chip oscillator)
CAC_CLOCK_SOURCE_LOCO	LOCO (Low speed on chip oscillator)
CAC_CLOCK_SOURCE_PCLKB	PCLKB (Peripheral Clock B)
CAC_CLOCK_SOURCE_IWDT	IWDT-dedicated on-chip oscillator.
CAC_CLOCK_SOURCE_EXTERNAL	Externally supplied measurement clock on CACREF pin.

◆ cac_ref_divider_t

enum cac_ref_divider_t	
Enumeration of available dividers for the reference clock.	
Enumerator	
CAC_REF_DIV_32	Reference clock divided by 32.
CAC_REF_DIV_128	Reference clock divided by 128.
CAC_REF_DIV_1024	Reference clock divided by 1024.
CAC_REF_DIV_8192	Reference clock divided by 8192.

◆ cac_ref_digfilter_t

enum cac_ref_digfilter_t	
Enumeration of available digital filter settings for an external reference clock.	
Enumerator	
CAC_REF_DIGITAL_FILTER_OFF	No digital filter on the CACREF pin for reference clock.
CAC_REF_DIGITAL_FILTER_1	Sampling clock for digital filter = measuring frequency.
CAC_REF_DIGITAL_FILTER_4	Sampling clock for digital filter = measuring frequency/4.
CAC_REF_DIGITAL_FILTER_16	Sampling clock for digital filter = measuring frequency/16.

◆ cac_ref_edge_t

enum cac_ref_edge_t	
Enumeration of available edge detect settings for the reference clock.	
Enumerator	
CAC_REF_EDGE_RISE	Rising edge detect for the Reference clock.
CAC_REF_EDGE_FALL	Falling edge detect for the Reference clock.
CAC_REF_EDGE_BOTH	Both Rising and Falling edges detect for the Reference clock.

◆ cac_meas_divider_t

enum cac_meas_divider_t	
Enumeration of available dividers for the measurement clock	
Enumerator	
CAC_MEAS_DIV_1	Measurement clock divided by 1.
CAC_MEAS_DIV_4	Measurement clock divided by 4.
CAC_MEAS_DIV_8	Measurement clock divided by 8.
CAC_MEAS_DIV_32	Measurement clock divided by 32.

4.3.4 CAN Interface

Interfaces

Detailed Description

Interface for CAN peripheral.

Summary

The CAN interface provides common APIs for CAN HAL drivers. CAN interface supports following features.

- Full-duplex CAN communication
- Generic CAN parameter setting

- Interrupt driven transmit/receive processing
- Callback function support with returning event code
- Hardware resource locking during a transaction

Implemented by:

- Controller Area Network (r_can)
- Controller Area Network - Flexible Data (r_canfd)

Data Structures

struct	can_info_t
struct	can_bit_timing_cfg_t
struct	can_frame_t
struct	can_callback_args_t
struct	can_cfg_t
struct	can_api_t
struct	can_instance_t

Typedefs

typedef void	can_ctrl_t

Enumerations

enum	can_event_t
enum	can_operation_mode_t
enum	can_test_mode_t
enum	can_id_mode_t
enum	can_frame_type_t

Data Structure Documentation

◆ can_info_t

struct can_info_t

CAN status info

Data Fields		
uint32_t	status	Useful information from the CAN status register.

uint32_t	rx_mb_status	RX Message Buffer New Data flags.
uint32_t	rx_fifo_status	RX FIFO Empty flags.
uint8_t	error_count_transmit	Transmit error count.
uint8_t	error_count_receive	Receive error count.
uint32_t	error_code	Error code, cleared after reading.

◆ can_bit_timing_cfg_t

struct can_bit_timing_cfg_t		
CAN bit rate configuration.		
Data Fields		
uint32_t	baud_rate_prescaler	Baud rate prescaler. Valid values: 1 - 1024.
uint32_t	time_segment_1	Time segment 1 control.
uint32_t	time_segment_2	Time segment 2 control.
uint32_t	synchronization_jump_width	Synchronization jump width.

◆ can_frame_t

struct can_frame_t		
CAN data Frame		
Data Fields		
uint32_t	id	CAN ID.
can_id_mode_t	id_mode	Standard or Extended ID (IDE).
can_frame_type_t	type	Frame type (RTR).
uint8_t	data_length_code	CAN Data Length Code (DLC).
uint32_t	options	Implementation-specific options.
uint8_t	data[CAN_DATA_BUFFER_LENGTH]	CAN data.

◆ can_callback_args_t

struct can_callback_args_t		
CAN callback parameter definition		
Data Fields		
uint32_t	channel	Device channel number.
can_event_t	event	Event code.
uint32_t	error	Error code.

union can_callback_args_t	__unnamed__	
can_frame_t *	p_frame	
void const *	p_context	Context provided to user during callback.
can_frame_t	frame	Received frame data.

◆ **can_cfg_t**

struct can_cfg_t	
CAN Configuration	
Data Fields	
uint32_t	channel
	CAN channel.
can_bit_timing_cfg_t *	p_bit_timing
	CAN bit timing.
void(*	p_callback)(can_callback_args_t *p_args)
	Pointer to callback function.
void const *	p_context
	User defined callback context.
void const *	p_extend
	CAN hardware dependent configuration.
uint8_t	ipl
	Error/Transmit/Receive interrupt priority.
IRQn_Type	error_irq
	Error IRQ number.

IRQn_Type	rx_irq
	Receive IRQ number.
IRQn_Type	tx_irq
	Transmit IRQ number.

◆ can_api_t

struct can_api_t
Shared Interface definition for CAN

Data Fields

fsp_err_t(*	open)(can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)
fsp_err_t(*	write)(can_ctrl_t *const p_ctrl, uint32_t buffer_number, can_frame_t *const p_frame)
fsp_err_t(*	read)(can_ctrl_t *const p_ctrl, uint32_t buffer_number, can_frame_t *const p_frame)
fsp_err_t(*	close)(can_ctrl_t *const p_ctrl)
fsp_err_t(*	modeTransition)(can_ctrl_t *const p_api_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode)
fsp_err_t(*	infoGet)(can_ctrl_t *const p_ctrl, can_info_t *const p_info)
fsp_err_t(*	callbackSet)(can_ctrl_t *const p_api_ctrl, void(*p_callback)(can_callback_args_t *), void const *const p_context, can_callback_args_t *const p_callback_memory)

Field Documentation

◆ **open**

fsp_err_t(* can_api_t::open) (can_ctrl_t *const p_ctrl, can_cfg_t const *const p_cfg)

Open function for CAN device

Implemented as

- R_CAN_Open()
- R_CANFD_Open()

Parameters

[in,out]	p_ctrl	Pointer to the CAN control block. Must be declared by user. Value set here.
[in]	can_cfg_t	Pointer to CAN configuration structure. All elements of this structure must be set by user.

◆ **write**

fsp_err_t(* can_api_t::write) (can_ctrl_t *const p_ctrl, uint32_t buffer_number, can_frame_t *const p_frame)

Write function for CAN device

Implemented as

- R_CAN_Write()
- R_CANFD_Write()

Parameters

[in]	p_ctrl	Pointer to the CAN control block.
[in]	buffer	Buffer number (mailbox or message buffer) to write to.
[in]	p_frame	Pointer for frame of CAN ID, DLC, data and frame type to write.

◆ read

fsp_err_t(* can_api_t::read) (can_ctrl_t *const p_ctrl, uint32_t buffer_number, can_frame_t *const p_frame)

Read function for CAN device

Implemented as

- R_CANFD_Read()

Parameters

[in]	p_ctrl	Pointer to the CAN control block.
[in]	buffer	Message buffer (number) to read from.
[in]	p_frame	Pointer to store the CAN ID, DLC, data and frame type.

◆ close

fsp_err_t(* can_api_t::close) (can_ctrl_t *const p_ctrl)

Close function for CAN device

Implemented as

- R_CAN_Close()

Parameters

[in]	p_ctrl	Pointer to the CAN control block.

◆ modeTransition

fsp_err_t(* can_api_t::modeTransition) (can_ctrl_t *const p_api_ctrl, can_operation_mode_t operation_mode, can_test_mode_t test_mode)

Mode Transition function for CAN device

Implemented as

- R_CAN_ModeTransition()
- R_CANFD_ModeTransition()

Parameters

[in]	p_ctrl	Pointer to the CAN control block.
[in]	operation_mode	Destination CAN operation state.
[in]	test_mode	Destination CAN test state.

◆ infoGet

fsp_err_t(* can_api_t::infoGet) (can_ctrl_t *const p_ctrl, can_info_t *const p_info)

Get CAN channel info.

Implemented as

- R_CAN_InfoGet()
- R_CANFD_InfoGet()

Parameters

[in]	p_ctrl	Handle for channel (pointer to channel control block)
[out]	p_info	Memory address to return channel specific data to.

◆ callbackSet

fsp_err_t(* can_api_t::callbackSet) (can_ctrl_t *const p_api_ctrl, void(*p_callback)(can_callback_args_t *), void const *const p_context, can_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_CAN_CallbackSet()
- R_CANFD_CallbackSet()

Parameters

[in]	p_ctrl	Control block set in can_api_t::open call.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ can_instance_t

struct can_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
can_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.

can_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
can_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ can_ctrl_t

typedef void can_ctrl_t

CAN control block. Allocate an instance specific control block to pass into the CAN API calls.

Implemented as

- can_instance_ctrl_t
- canfd_instance_ctrl_t

Enumeration Type Documentation

◆ can_event_t

enum can_event_t	
CAN event codes	
Enumerator	
CAN_EVENT_ERR_WARNING	Error Warning event.
CAN_EVENT_ERR_PASSIVE	Error Passive event.
CAN_EVENT_ERR_BUS_OFF	Bus Off event.
CAN_EVENT_BUS_RECOVERY	Bus Off Recovery event.
CAN_EVENT_MAILBOX_MESSAGE_LOST	Mailbox has been overrun.
CAN_EVENT_ERR_BUS_LOCK	Bus lock detected (32 consecutive dominant bits).
CAN_EVENT_ERR_CHANNEL	Channel error has occurred.
CAN_EVENT_TX_ABORTED	Transmit abort event.
CAN_EVENT_RX_COMPLETE	Receive complete event.
CAN_EVENT_TX_COMPLETE	Transmit complete event.
CAN_EVENT_ERR_GLOBAL	Global error has occurred.
CAN_EVENT_TX_FIFO_EMPTY	Transmit FIFO is empty.

◆ can_operation_mode_t

enum can_operation_mode_t	
CAN Operation modes	
Enumerator	
CAN_OPERATION_MODE_NORMAL	CAN Normal Operation Mode.
CAN_OPERATION_MODE_RESET	CAN Reset Operation Mode.
CAN_OPERATION_MODE_HALT	CAN Halt Operation Mode.
CAN_OPERATION_MODE_SLEEP	CAN Sleep Operation Mode.

◆ can_test_mode_t

enum can_test_mode_t	
CAN Test modes	
Enumerator	
CAN_TEST_MODE_DISABLED	CAN Test Mode Disabled.
CAN_TEST_MODE_LISTEN	CAN Test Listen Mode.
CAN_TEST_MODE_LOOPBACK_EXTERNAL	CAN Test External Loopback Mode.
CAN_TEST_MODE_LOOPBACK_INTERNAL	CAN Test Internal Loopback Mode.
CAN_TEST_MODE_INTERNAL_BUS	CANFD Internal CAN Bus Communication Test Mode.

◆ can_id_mode_t

enum can_id_mode_t	
CAN ID modes	
Enumerator	
CAN_ID_MODE_STANDARD	Standard IDs of 11 bits used.
CAN_ID_MODE_EXTENDED	Extended IDs of 29 bits used.

◆ can_frame_type_t

enum can_frame_type_t	
CAN frame types	
Enumerator	
CAN_FRAME_TYPE_DATA	Data frame.
CAN_FRAME_TYPE_REMOTE	Remote frame.

4.3.5 CEC Interface

Interfaces

Detailed Description

Interface for CEC peripheral.

Summary

The CEC interface provides common APIs for CEC HAL drivers and supports the following features:

- Opening and closing the CEC module.
- Allocation for full range of local address settings (TV, Recording Device, Playback Device, etc.)
- Supports a user-callback function (required), invoked when transmit, receive, or error interrupts are received.

Implemented by:

- Consumer Electronics Control (r_cec)

Data Structures

union	cec_message_t
struct	cec_callback_args_t
struct	cec_cfg_t
struct	cec_api_t
struct	cec_instance_t

Typedefs

typedef void	cec_ctrl_t

Enumerations

enum	cec_addr_t
enum	cec_clock_source_t
enum	cec_state_t
enum	cec_error_t
enum	cec_event_t

Data Structure Documentation

◆ cec_message_t

union cec_message_t

CEC message		
Data Fields		
struct cec_message_t	__unnamed__	
uint8_t	raw_data[CEC_DATA_BUFFER_LENGTH+2 *sizeof(uint8_t)]	Contiguous raw data.

◆ **cec_callback_args_t**

struct cec_callback_args_t		
CEC callback parameter definition		
Data Fields		
cec_event_t	event	Event code.
void const *	p_context	Context provided to user during callback.
bool	addr_match	Local addresss matches message destination.
uint8_t	data_byte	Received data byte (INTDA)
cec_status_t	status	CEC Module status data.
cec_error_t	errors	Error code bitfield.

◆ **cec_cfg_t**

struct cec_cfg_t	
CEC Configuration	
Data Fields	
cec_timing_t const *	bit_timing_cfg
	CEC Bit Timing Configuration.
uint16_t	rx_data_sample_time
	Receive Data Sample Time Setting.
uint16_t	rx_data_bit_reference_width
	Receive Data Bit Reference Width.
void(*	p_callback)(cec_callback_args_t *p_args)
	Pointer to callback function.

void const *	p_context
	User defined callback context.
uint8_t	ipl
	Error/Data/Message interrupt priority level.
IRQn_Type	error_irq
	Error IRQ number.
IRQn_Type	data_irq
	Data IRQ number.
IRQn_Type	msg_irq
	Communication Complete IRQ number.
void *	p_extend
	Pointer to extended configuration structure.

◆ cec_api_t

struct cec_api_t
Shared Interface definition for CEC

Data Fields

fsp_err_t(*	open)(cec_ctrl_t *const p_ctrl, cec_cfg_t const *const p_cfg)
fsp_err_t(*	mediaInit)(cec_ctrl_t *const p_ctrl, cec_addr_t local_address)
fsp_err_t(*	write)(cec_ctrl_t *const p_ctrl, cec_message_t const *const p_message, uint32_t message_size)

fsp_err_t(*	close)(cec_ctrl_t *const p_ctrl)
fsp_err_t(*	statusGet)(cec_ctrl_t *const p_ctrl, cec_status_t *const p_status)
fsp_err_t(*	callbackSet)(cec_ctrl_t *const p_ctrl, void(*p_callback)(cec_callback_args_t *), void const *const p_context, cec_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* cec_api_t::open) (cec_ctrl_t *const p_ctrl, cec_cfg_t const *const p_cfg)

Open function for CEC device

Implemented as

- R_CEC_Open()

Parameters

[in,out]	p_ctrl	Pointer to the CEC control block. Must be declared by user. Value set here.
[in]	p_cfg	Pointer to CEC configuration structure. All elements of this structure must be set by user.

◆ mediaInit

fsp_err_t(* cec_api_t::mediaInit) (cec_ctrl_t *const p_ctrl, cec_addr_t local_address)

Initializes the CEC device. May be called any time after the CEC module has been opened. This API blocks until the device initialization procedure is complete.

Implemented as

- R_CEC_MediaInit()

Parameters

[in]	p_ctrl	Pointer to CEC instance control block.
[out]	local_address	Desired Logical address for local device.

◆ write

fsp_err_t(* cec_api_t::write) (cec_ctrl_t *const p_ctrl, cec_message_t const *const p_message, uint32_t message_size)

Write function for CEC device

Implemented as

- R_CEC_Write()

Parameters

[in]	p_ctrl	Pointer to CEC instance control block
[in]	p_message	Message data
[in]	message_size	Total size of entire message

◆ close

fsp_err_t(* cec_api_t::close) (cec_ctrl_t *const p_ctrl)

Close function for CEC device

Implemented as

- R_CEC_Close()

Parameters

[in]	p_ctrl	Pointer to CEC instance control block
[out]	p_message	Message data

◆ statusGet

fsp_err_t(* cec_api_t::statusGet) (cec_ctrl_t *const p_ctrl, cec_status_t *const p_status)

Get CEC channel info.

Implemented as

- R_CEC_StatusGet()

Parameters

[in]	p_ctrl	Pointer to CEC instance control block
[out]	p_status	Memory address to return channel specific data to.

◆ **callbackSet**

fsp_err_t(* cec_api_t::callbackSet) (cec_ctrl_t *const p_ctrl, void(*p_callback)(cec_callback_args_t *), void const *const p_context, cec_callback_args_t *const p_callback_memory)

Specify callback function, optional context pointer and working memory pointer.

Implemented as

- R_CEC_CallbackSet()

Parameters

[in]	p_ctrl	Control block set in cec_api_t::open call.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_callback_memory	Pointer to volatile memory where callback structure cec be allocated. Callback arguments allocated here are only valid during the callback.

◆ **cec_instance_t**

struct cec_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
cec_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
cec_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
cec_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ **cec_ctrl_t**

typedef void cec_ctrl_t

CEC control block. Allocate an instance specific control block to pass into the CEC API calls.

Implemented as

- cec_instance_ctrl_t

Enumeration Type Documentation

◆ cec_addr_t

enum cec_addr_t	
CEC Addresses	
Enumerator	
CEC_ADDR_TV	CEC Address for TV.
CEC_ADDR_RECORDING_DEVICE_1	CEC Address for Recording Device 1.
CEC_ADDR_RECORDING_DEVICE_2	CEC Address for Recording Devide 2.
CEC_ADDR_TUNER_1	CEC Address for Tuner 1.
CEC_ADDR_PLAYBACK_DEVICE_1	CEC Address for Playback Device 1.
CEC_ADDR_AUDIO_SYSTEM	CEC Address for Audio System.
CEC_ADDR_TUNER_2	CEC Address for Tuner 2.
CEC_ADDR_TUNER_3	CEC Address for Tuner 3.
CEC_ADDR_PLAYBACK_DEVICE_2	CEC Address for Playback Device 2.
CEC_ADDR_RECORDING_DEVICE_3	CEC Address for Recording Device 3.
CEC_ADDR_TUNER_4	CEC Address for Tuner 4.
CEC_ADDR_PLAYBACK_DEVICE_3	CEC Address for Playback Device 3.
CEC_ADDR_SPECIFIC_USE	CEC Address for Specific Use.
CEC_ADDR_UNREGISTERED	CEC Address for Unregistered Devices.
CEC_ADDR_BROADCAST	CEC Broadcast message.

◆ cec_clock_source_t

enum cec_clock_source_t	
CEC Source Clock	
Enumerator	
CEC_CLOCK_SOURCE_PCLKB_DIV_32	PCLKB / 32 is the source of the CEC Clock.
CEC_CLOCK_SOURCE_PCLKB_DIV_64	PCLKB / 64 is the source of the CEC Clock.
CEC_CLOCK_SOURCE_PCLKB_DIV_128	PCLKB / 128 is the source of the CEC Clock.
CEC_CLOCK_SOURCE_PCLKB_DIV_256	PCLKB / 256 is the source of the CEC Clock.
CEC_CLOCK_SOURCE_PCLKB_DIV_512	PCLKB / 512 is the source of the CEC Clock.
CEC_CLOCK_SOURCE_PCLKB_DIV_1024	PCLKB / 1024 is the source of the CEC Clock.
CEC_CLOCK_SOURCE_CECCLK	CECCLK is the source of the CEC Clock.
CEC_CLOCK_SOURCE_CECCLK_DIV_256	CECCLK / 256 is the source of the CEC Clock.

◆ cec_state_t

enum cec_state_t	
CEC State	
Enumerator	
CEC_STATE_UNINIT	Module requires initialization.
CEC_STATE_READY	Module ready for operation.
CEC_STATE_TX_ACTIVE	Transmit in progress, either direct or broadcast.
CEC_STATE_RX_ACTIVE	Receive in progress, either direct or broadcast.
CEC_STATE_BUSY	CEC Signal Free Time has not yet elapsed.

◆ cec_error_t

enum cec_error_t	
CEC Error Code	
Enumerator	
CEC_ERROR_NONE	No errors currently active.
CEC_ERROR_OERR	Overrun error.
CEC_ERROR_UERR	Unterrun Error.
CEC_ERROR_ACKERR	ACK Error.
CEC_ERROR_TERR	Timing Error.
CEC_ERROR_TXERR	Transmission Error.
CEC_ERROR_AERR	Bus arbitration Loss.
CEC_ERROR_BLERR	Bus lock error.
CEC_ERROR_ADDR	Address allocation error.

◆ cec_event_t

enum cec_event_t	
CEC event codes	
Enumerator	
CEC_EVENT_RX_DATA	Receive Data byte event.
CEC_EVENT_RX_COMPLETE	Receive complete event.
CEC_EVENT_TX_COMPLETE	Transmit complete event.
CEC_EVENT_READY	CEC Address allocated and module is now ready.
CEC_EVENT_ERR	Error has occurred.

4.3.6 CGC Interface

Interfaces

Detailed Description

Interface for clock generation.

Summary

The CGC interface provides the ability to configure and use all of the CGC module's capabilities. Among the capabilities is the selection of several clock sources to use as the system clock source. Additionally, the system clocks can be divided down to provide a wide range of frequencies for various system and peripheral needs.

Clock stability can be checked and clocks may also be stopped to save power when not needed. The API has a function to return the frequency of the system and system peripheral clocks at run time. There is also a feature to detect when the main oscillator has stopped, with the option of calling a user provided callback function.

The CGC interface is implemented by:

- Clock Generation Circuit (r_cgc)

Data Structures

struct	cgc_callback_args_t
struct	cgc_pll_cfg_t
union	cgc_divider_cfg_t
struct	cgc_cfg_t
struct	cgc_clocks_cfg_t
struct	cgc_api_t
struct	cgc_instance_t

Typedefs

typedef void	cgc_ctrl_t

Enumerations

enum	cgc_event_t
enum	cgc_clock_t
enum	cgc_pll_div_t
enum	cgc_pll_mul_t

enum	cgc_sys_clock_div_t
enum	cgc_usb_clock_div_t
enum	cgc_clock_change_t

Data Structure Documentation

◆ cgc_callback_args_t

struct cgc_callback_args_t		
Callback function parameter data		
Data Fields		
cgc_event_t	event	The event can be used to identify what caused the callback.
void const *	p_context	Placeholder for user data.

◆ cgc_pll_cfg_t

struct cgc_pll_cfg_t		
Clock configuration structure - Used as an input parameter to the cgc_api_t::clockStart function for the PLL clock.		
Data Fields		
cgc_clock_t	source_clock	PLL source clock (main oscillator or HOCO)
cgc_pll_div_t	divider	PLL divider.
cgc_pll_mul_t	multiplier	PLL multiplier.

◆ cgc_divider_cfg_t

union cgc_divider_cfg_t		
Clock configuration structure - Used as an input parameter to the cgc_api_t::systemClockSet and cgc_api_t::systemClockGet functions.		
Data Fields		
uint32_t	sckdivcr_w	(@ 0x4001E020) System clock Division control register
struct cgc_divider_cfg_t	__unnamed__	

◆ cgc_cfg_t

struct cgc_cfg_t
Configuration options.

◆ cgc_clocks_cfg_t

struct cgc_clocks_cfg_t		
Clock configuration		
Data Fields		
cgc_clock_t	system_clock	System clock source enumeration.
cgc_pll_cfg_t	pll_cfg	PLL configuration structure.
cgc_pll_cfg_t	pll2_cfg	PLL2 configuration structure.
cgc_divider_cfg_t	divider_cfg	Clock dividers structure.
cgc_clock_change_t	loco_state	State of LOCO.
cgc_clock_change_t	moco_state	State of MOCO.
cgc_clock_change_t	hoco_state	State of HOCO.
cgc_clock_change_t	mainosc_state	State of Main oscillator.
cgc_clock_change_t	pll_state	State of PLL.
cgc_clock_change_t	pll2_state	State of PLL2.

◆ cgc_api_t

struct cgc_api_t	
CGC functions implemented at the HAL layer follow this API.	
Data Fields	
fsp_err_t(*	open)(cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg)
fsp_err_t(*	clocksCfg)(cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg)
fsp_err_t(*	clockStart)(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg)
fsp_err_t(*	clockStop)(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)
fsp_err_t(*	clockCheck)(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)
fsp_err_t(*	systemClockSet)(cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg)
fsp_err_t(*	systemClockGet)(cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg)

fsp_err_t(*	oscStopDetectEnable)(cgc_ctrl_t *const p_ctrl)
fsp_err_t(*	oscStopDetectDisable)(cgc_ctrl_t *const p_ctrl)
fsp_err_t(*	oscStopStatusClear)(cgc_ctrl_t *const p_ctrl)
fsp_err_t(*	callbackSet)(cgc_ctrl_t *const p_api_ctrl, void(*p_callback)(cgc_callback_args_t *), void const *const p_context, cgc_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(cgc_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* cgc_api_t::open) (cgc_ctrl_t *const p_ctrl, cgc_cfg_t const *const p_cfg)

Initial configuration

Implemented as

- R_CGC_Open()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	p_cfg	Pointer to configuration

◆ clocksCfg

fsp_err_t(* cgc_api_t::clocksCfg) (cgc_ctrl_t *const p_ctrl, cgc_clocks_cfg_t const *const p_clock_cfg)

Configure all system clocks.

Implemented as

- R_CGC_ClocksCfg()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	p_clock_cfg	Pointer to desired configuration of system clocks

◆ clockStart

fsp_err_t(* cgc_api_t::clockStart) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_pll_cfg_t const *const p_pll_cfg)

Start a clock.

Implemented as

- R_CGC_ClockStart()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	clock_source	Clock source to start
[in]	p_pll_cfg	Pointer to PLL configuration, can be NULL if clock_source is not CGC_CLOCK_PLL or CGC_CLOCK_PLL2

◆ clockStop

fsp_err_t(* cgc_api_t::clockStop) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)

Stop a clock.

Implemented as

- R_CGC_ClockStop()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	clock_source	The clock source to stop

◆ clockCheck

fsp_err_t(* cgc_api_t::clockCheck) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source)

Check the stability of the selected clock.

Implemented as

- R_CGC_ClockCheck()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	clock_source	Which clock source to check for stability

◆ systemClockSet

fsp_err_t(* cgc_api_t::systemClockSet) (cgc_ctrl_t *const p_ctrl, cgc_clock_t clock_source, cgc_divider_cfg_t const *const p_divider_cfg)

Set the system clock.

Implemented as

- R_CGC_SystemClockSet()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	clock_source	Clock source to set as system clock
[in]	p_divider_cfg	Pointer to the clock divider configuration

◆ systemClockGet

fsp_err_t(* cgc_api_t::systemClockGet) (cgc_ctrl_t *const p_ctrl, cgc_clock_t *const p_clock_source, cgc_divider_cfg_t *const p_divider_cfg)

Get the system clock information.

Implemented as

- R_CGC_SystemClockGet()

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_clock_source	Returns the current system clock
[out]	p_divider_cfg	Returns the current system clock dividers

◆ oscStopDetectEnable

fsp_err_t(* cgc_api_t::oscStopDetectEnable) (cgc_ctrl_t *const p_ctrl)

Enable and optionally register a callback for Main Oscillator stop detection.

Implemented as

- R_CGC_OscStopDetectEnable()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	p_callback	Callback function that will be called by the NMI interrupt when an oscillation stop is detected. If the second argument is "false", then this argument can be NULL.
[in]	enable	Enable/disable Oscillation Stop Detection

◆ oscStopDetectDisable

fsp_err_t(* cgc_api_t::oscStopDetectDisable) (cgc_ctrl_t *const p_ctrl)

Disable Main Oscillator stop detection.

Implemented as

- R_CGC_OscStopDetectDisable()

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ oscStopStatusClear

fsp_err_t(* cgc_api_t::oscStopStatusClear) (cgc_ctrl_t *const p_ctrl)

Clear the oscillator stop detection flag.

Implemented as

- R_CGC_OscStopStatusClear()

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ callbackSet

fsp_err_t(* cgc_api_t::callbackSet) (cgc_ctrl_t *const p_api_ctrl, void(*p_callback)(cgc_callback_args_t *), void const *const p_context, cgc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_CGC_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the CGC control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* cgc_api_t::close) (cgc_ctrl_t *const p_ctrl)
Close the CGC driver.

Implemented as

- R_CGC_Close()

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ cgc_instance_t

struct cgc_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
cgc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
cgc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
cgc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ cgc_ctrl_t

typedef void cgc_ctrl_t
CGC control block. Allocate an instance specific control block to pass into the CGC API calls.

Implemented as

- cgc_instance_ctrl_t

Enumeration Type Documentation

◆ cgc_event_t

enum cgc_event_t	
Events that can trigger a callback function	
Enumerator	
CGC_EVENT_OSC_STOP_DETECT	Oscillator stop detection has caused the event.

◆ cgc_clock_t

enum cgc_clock_t	
System clock source identifiers - The source of ICLK, BCLK, FCLK, PCLKS A-D and UCLK prior to the system clock divider	
Enumerator	
CGC_CLOCK_HOCO	The high speed on chip oscillator.
CGC_CLOCK_MOCO	The middle speed on chip oscillator.
CGC_CLOCK_LOCO	The low speed on chip oscillator.
CGC_CLOCK_MAIN_OSC	The main oscillator.
CGC_CLOCK_SUBCLOCK	The subclock oscillator.
CGC_CLOCK_PLL	The PLL oscillator.
CGC_CLOCK_PLL2	The PLL2 oscillator.

◆ cgc_pll_div_t

enum cgc_pll_div_t	
PLL divider values	
Enumerator	
CGC_PLL_DIV_1	PLL divider of 1.
CGC_PLL_DIV_2	PLL divider of 2.
CGC_PLL_DIV_3	PLL divider of 3 (S7, S5 only)
CGC_PLL_DIV_4	PLL divider of 4 (S3 only)

◆ cgc_pll_mul_t

enum cgc_pll_mul_t	
PLL multiplier values	
Enumerator	
CGC_PLL_MUL_8_0	PLL multiplier of 8.0.
CGC_PLL_MUL_9_0	PLL multiplier of 9.0.
CGC_PLL_MUL_10_0	PLL multiplier of 10.0.
CGC_PLL_MUL_10_5	PLL multiplier of 10.5.
CGC_PLL_MUL_11_0	PLL multiplier of 11.0.
CGC_PLL_MUL_11_5	PLL multiplier of 11.5.
CGC_PLL_MUL_12_0	PLL multiplier of 12.0.
CGC_PLL_MUL_12_5	PLL multiplier of 12.5.
CGC_PLL_MUL_13_0	PLL multiplier of 13.0.
CGC_PLL_MUL_13_5	PLL multiplier of 13.5.
CGC_PLL_MUL_14_0	PLL multiplier of 14.0.
CGC_PLL_MUL_14_5	PLL multiplier of 14.5.
CGC_PLL_MUL_15_0	PLL multiplier of 15.0.
CGC_PLL_MUL_15_5	PLL multiplier of 15.5.
CGC_PLL_MUL_16_0	PLL multiplier of 16.0.
CGC_PLL_MUL_16_5	PLL multiplier of 16.5.
CGC_PLL_MUL_17_0	PLL multiplier of 17.0.
CGC_PLL_MUL_17_5	PLL multiplier of 17.5.
CGC_PLL_MUL_18_0	PLL multiplier of 18.0.
CGC_PLL_MUL_18_5	PLL multiplier of 18.5.
CGC_PLL_MUL_19_0	PLL multiplier of 19.0.

CGC_PLL_MUL_19_5	PLL multiplier of 19.5.
CGC_PLL_MUL_20_0	PLL multiplier of 20.0.
CGC_PLL_MUL_20_5	PLL multiplier of 20.5.
CGC_PLL_MUL_21_0	PLL multiplier of 21.0.
CGC_PLL_MUL_21_5	PLL multiplier of 21.5.
CGC_PLL_MUL_22_0	PLL multiplier of 22.0.
CGC_PLL_MUL_22_5	PLL multiplier of 22.5.
CGC_PLL_MUL_23_0	PLL multiplier of 23.0.
CGC_PLL_MUL_23_5	PLL multiplier of 23.5.
CGC_PLL_MUL_24_0	PLL multiplier of 24.0.
CGC_PLL_MUL_24_5	PLL multiplier of 24.5.
CGC_PLL_MUL_25_0	PLL multiplier of 25.0.
CGC_PLL_MUL_25_5	PLL multiplier of 25.5.
CGC_PLL_MUL_26_0	PLL multiplier of 26.0.
CGC_PLL_MUL_26_5	PLL multiplier of 26.5.
CGC_PLL_MUL_27_0	PLL multiplier of 27.0.
CGC_PLL_MUL_27_5	PLL multiplier of 27.5.
CGC_PLL_MUL_28_0	PLL multiplier of 28.0.
CGC_PLL_MUL_28_5	PLL multiplier of 28.5.
CGC_PLL_MUL_29_0	PLL multiplier of 29.0.
CGC_PLL_MUL_29_5	PLL multiplier of 29.5.
CGC_PLL_MUL_30_0	PLL multiplier of 30.0.
CGC_PLL_MUL_31_0	PLL multiplier of 31.0.

◆ cgc_sys_clock_div_t

enum cgc_sys_clock_div_t	
System clock divider vlues - The individually selectable divider of each of the system clocks, ICLK, BCLK, FCLK, PCLKS A-D.	
Enumerator	
CGC_SYS_CLOCK_DIV_1	System clock divided by 1.
CGC_SYS_CLOCK_DIV_2	System clock divided by 2.
CGC_SYS_CLOCK_DIV_4	System clock divided by 4.
CGC_SYS_CLOCK_DIV_8	System clock divided by 8.
CGC_SYS_CLOCK_DIV_16	System clock divided by 16.
CGC_SYS_CLOCK_DIV_32	System clock divided by 32.
CGC_SYS_CLOCK_DIV_64	System clock divided by 64.

◆ cgc_usb_clock_div_t

enum cgc_usb_clock_div_t	
USB clock divider values	
Enumerator	
CGC_USB_CLOCK_DIV_3	Divide USB source clock by 3.
CGC_USB_CLOCK_DIV_4	Divide USB source clock by 4.
CGC_USB_CLOCK_DIV_5	Divide USB source clock by 5.

◆ cgc_clock_change_t

enum cgc_clock_change_t	
Clock options	
Enumerator	
CGC_CLOCK_CHANGE_START	Start the clock.
CGC_CLOCK_CHANGE_STOP	Stop the clock.
CGC_CLOCK_CHANGE_NONE	No change to the clock.

4.3.7 Comparator Interface

Interfaces

Detailed Description

Interface for comparators.

Summary

The comparator interface provides standard comparator functionality, including generating an event when the comparator result changes.

Implemented by:

- High-Speed Analog Comparator (r_acmphs)
- Low-Power Analog Comparator (r_acmplp)

Data Structures

struct	comparator_info_t
struct	comparator_status_t
struct	comparator_callback_args_t
struct	comparator_cfg_t
struct	comparator_api_t
struct	comparator_instance_t

Typedefs

typedef void	comparator_ctrl_t

Enumerations

enum	comparator_mode_t
enum	comparator_trigger_t
enum	comparator_polarity_invert_t
enum	comparator_pin_output_t
enum	comparator_filter_t
enum	comparator_state_t

Data Structure Documentation

◆ comparator_info_t

struct comparator_info_t

Comparator information.

Data Fields		
uint32_t	min_stabilization_wait_us	Minimum stabilization wait time in microseconds.

◆ comparator_status_t

struct comparator_status_t

Comparator status.

Data Fields		
comparator_state_t	state	Current comparator state.

◆ comparator_callback_args_t

struct comparator_callback_args_t

Callback function parameter data

Data Fields		
void const *	p_context	Placeholder for user data. Set in comparator_api_t::open function in comparator_cfg_t.
uint32_t	channel	The physical hardware channel that caused the interrupt.

◆ comparator_cfg_t

struct comparator_cfg_t

User configuration structure, used in open function

Data Fields

uint8_t	channel
	Hardware channel used.
comparator_mode_t	mode
	Normal or window mode.
comparator_trigger_t	trigger
	Trigger setting.
comparator_filter_t	filter
	Digital filter clock divisor setting.
comparator_polarity_invert_t	invert
	Whether to invert output.
comparator_pin_output_t	pin_output
	Whether to include output on output pin.
uint8_t	vref_select
	Internal Vref Select.
uint8_t	ipl
	Interrupt priority.
IRQn_Type	irq
	NVIC interrupt number.

void(*	p_callback)(comparator_callback_args_t *p_args)
void const *	p_context
void const *	p_extend
	Comparator hardware dependent configuration.

Field Documentation

◆ p_callback

void(* comparator_cfg_t::p_callback) (comparator_callback_args_t *p_args)

Callback called when comparator event occurs.

◆ p_context

void const* comparator_cfg_t::p_context

Placeholder for user data. Passed to the user callback in comparator_callback_args_t.

◆ comparator_api_t

struct comparator_api_t

Comparator functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg)
fsp_err_t(*	outputEnable)(comparator_ctrl_t *const p_ctrl)
fsp_err_t(*	infoGet)(comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info)
fsp_err_t(*	statusGet)(comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status)
fsp_err_t(*	close)(comparator_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* comparator_api_t::open) (comparator_ctrl_t *const p_ctrl, comparator_cfg_t const *const p_cfg)

Initialize the comparator.

Implemented as

- R_ACMPHS_Open()
- R_ACMPLP_Open()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	p_cfg	Pointer to configuration

◆ outputEnable

fsp_err_t(* comparator_api_t::outputEnable) (comparator_ctrl_t *const p_ctrl)

Start the comparator.

Implemented as

- R_ACMPHS_OutputEnable()
- R_ACMPLP_OutputEnable()

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ infoGet

fsp_err_t(* comparator_api_t::infoGet) (comparator_ctrl_t *const p_ctrl, comparator_info_t *const p_info)

Provide information such as the recommended minimum stabilization wait time.

Implemented as

- R_ACMPHS_InfoGet()
- R_ACMPLP_InfoGet()

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_info	Comparator information stored here

◆ **statusGet**

fsp_err_t(* comparator_api_t::statusGet) (comparator_ctrl_t *const p_ctrl, comparator_status_t *const p_status)

Provide current comparator status.

Implemented as

- R_ACMPHS_StatusGet()
- R_ACMPLP_StatusGet()

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_status	Status stored here

◆ **close**

fsp_err_t(* comparator_api_t::close) (comparator_ctrl_t *const p_ctrl)

Stop the comparator.

Implemented as

- R_ACMPHS_Close()
- R_ACMPLP_Close()

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ **comparator_instance_t**

struct comparator_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
comparator_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
comparator_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
comparator_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ comparator_ctrl_t

typedef void comparator_ctrl_t
Includes board and MCU related header files. Comparator control block. Allocate an instance specific control block to pass into the comparator API calls. **Implemented as** ◦ acmphs_instance_ctrl_t ◦ acmplp_instance_ctrl_t

Enumeration Type Documentation

◆ comparator_mode_t

enum comparator_mode_t

Select whether to invert the polarity of the comparator output.

Enumerator	
COMPARATOR_MODE_NORMAL	Normal mode.
COMPARATOR_MODE_WINDOW	Window mode, not supported by all implementations.

◆ comparator_trigger_t

enum comparator_trigger_t

Trigger type: rising edge, falling edge, both edges, low level.

Enumerator	
COMPARATOR_TRIGGER_RISING	Rising edge trigger.
COMPARATOR_TRIGGER_FALLING	Falling edge trigger.
COMPARATOR_TRIGGER_BOTH_EDGE	Both edges trigger.

◆ comparator_polarity_invert_t

enum comparator_polarity_invert_t	
Select whether to invert the polarity of the comparator output.	
Enumerator	
COMPARATOR_POLARITY_INVERT_OFF	Do not invert polarity.
COMPARATOR_POLARITY_INVERT_ON	Invert polarity.

◆ comparator_pin_output_t

enum comparator_pin_output_t	
Select whether to include the comparator output on the output pin.	
Enumerator	
COMPARATOR_PIN_OUTPUT_OFF	Do not include comparator output on output pin.
COMPARATOR_PIN_OUTPUT_ON	Include comparator output on output pin.

◆ comparator_filter_t

enum comparator_filter_t	
Comparator digital filtering sample clock divisor settings.	
Enumerator	
COMPARATOR_FILTER_OFF	Disable debounce filter.
COMPARATOR_FILTER_1	Filter using PCLK divided by 1, not supported by all implementations.
COMPARATOR_FILTER_8	Filter using PCLK divided by 8.
COMPARATOR_FILTER_16	Filter using PCLK divided by 16, not supported by all implementations.
COMPARATOR_FILTER_32	Filter using PCLK divided by 32.

◆ comparator_state_t

enum comparator_state_t	
Current comparator state.	
Enumerator	
COMPARATOR_STATE_OUTPUT_LOW	VCMP < VREF if polarity is not inverted, VCMP > VREF if inverted.
COMPARATOR_STATE_OUTPUT_HIGH	VCMP > VREF if polarity is not inverted, VCMP < VREF if inverted.
COMPARATOR_STATE_OUTPUT_DISABLED	comparator_api_t::outputEnable() has not been called

4.3.8 CRC Interface

Interfaces

Detailed Description

Interface for cyclic redundancy checking.

Summary

The CRC (Cyclic Redundancy Check) calculator generates CRC codes using five different polynomials including 8 bit, 16 bit, and 32 bit variations. Calculation can be performed by sending data to the block using the CPU or by snooping on read or write activity on one of 10 SCI channels.

Implemented by:

- Cyclic Redundancy Check (CRC) Calculator (r_crc)

Data Structures

struct	crc_input_t
struct	crc_cfg_t
struct	crc_api_t
struct	crc_instance_t

Typedefs

typedef void	crc_ctrl_t

Enumerations

enum	crc_polynomial_t
enum	crc_bit_order_t
enum	crc_snoop_direction_t
enum	crc_snoop_address_t

Data Structure Documentation

◆ crc_input_t

struct crc_input_t
Structure for CRC inputs

◆ crc_cfg_t

struct crc_cfg_t

User configuration structure, used in open function

Data Fields		
crc_polynomial_t	polynomial	CRC Generating Polynomial Switching (GPS)
crc_bit_order_t	bit_order	CRC Calculation Switching (LMS)
crc_snoop_address_t	snoop_address	Register Snoop Address (CRCSA)
void const *	p_extend	CRC Hardware Dependent Configuration.

◆ crc_api_t

struct crc_api_t

CRC driver structure. General CRC functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)
fsp_err_t(*	close)(crc_ctrl_t *const p_ctrl)
fsp_err_t(*	crcResultGet)(crc_ctrl_t *const p_ctrl, uint32_t *crc_result)
fsp_err_t(*	snoopEnable)(crc_ctrl_t *const p_ctrl, uint32_t crc_seed)

fsp_err_t(*	snoopDisable)(crc_ctrl_t *const p_ctrl)
fsp_err_t(*	calculate)(crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *p_crc_result)

Field Documentation

◆ open

fsp_err_t(* crc_api_t::open) (crc_ctrl_t *const p_ctrl, crc_cfg_t const *const p_cfg)

Open the CRC driver module.

Implemented as

- R_CRC_Open()

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[in]	p_cfg	Pointer to a configuration structure.

◆ close

fsp_err_t(* crc_api_t::close) (crc_ctrl_t *const p_ctrl)

Close the CRC module driver

Implemented as

- R_CRC_Close()

Parameters

[in]	p_ctrl	Pointer to crc device handle

Return values

FSP_SUCCESS	Configuration was successful.

◆ crcResultGet

fsp_err_t(* crc_api_t::crcResultGet) (crc_ctrl_t *const p_ctrl, uint32_t *crc_result)

Return the current calculated value.

Implemented as

- R_CRC_CalculatedValueGet()

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[out]	crc_result	The calculated value from the last CRC calculation.

◆ snoopEnable

fsp_err_t(* crc_api_t::snoopEnable) (crc_ctrl_t *const p_ctrl, uint32_t crc_seed)

Configure and Enable snooping.

Implemented as

- R_CRC_SnoopEnable()

Parameters

[in]	p_ctrl	Pointer to CRC device handle.
[in]	crc_seed	CRC seed.

◆ snoopDisable

fsp_err_t(* crc_api_t::snoopDisable) (crc_ctrl_t *const p_ctrl)

Disable snooping.

Implemented as

- R_CRC_SnoopDisable()

Parameters

[in]	p_ctrl	Pointer to CRC device handle.

◆ calculate

fsp_err_t(* crc_api_t::calculate) (crc_ctrl_t *const p_ctrl, crc_input_t *const p_crc_input, uint32_t *p_crc_result)

Perform a CRC calculation on a block of data.

Implemented as

- R_CRC_Calculate()

Parameters

[in]	p_ctrl	Pointer to crc device handle.
[in]	p_crc_input	A pointer to structure for CRC inputs
[out]	crc_result	The calculated value of the CRC calculation.

◆ crc_instance_t

struct crc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
crc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
crc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
crc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ crc_ctrl_t

typedef void crc_ctrl_t

CRC control block. Allocate an instance specific control block to pass into the CRC API calls.

Implemented as

- crc_instance_ctrl_t

Enumeration Type Documentation

◆ crc_polynomial_t

enum crc_polynomial_t	
CRC Generating Polynomial Switching (GPS).	
Enumerator	
CRC_POLYNOMIAL_CRC_8	8-bit CRC-8 (X^8 + X^2 + X + 1)
CRC_POLYNOMIAL_CRC_16	16-bit CRC-16 (X^16 + X^15 + X^2 + 1)
CRC_POLYNOMIAL_CRC_CCITT	16-bit CRC-CCITT (X^16 + X^12 + X^5 + 1)
CRC_POLYNOMIAL_CRC_32	32-bit CRC-32 (X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1)
CRC_POLYNOMIAL_CRC_32C	32-bit CRC-32C (X^32 + X^28 + X^27 + X^26 + X^25 + X^23 + X^22 + X^20 + X^19 + X^18 + X^14 + X^13 + X^11 + X^10 + X^9 + X^8 + X^6 + 1)

◆ crc_bit_order_t

enum crc_bit_order_t	
CRC Calculation Switching (LMS)	
Enumerator	
CRC_BIT_ORDER_LMS_LSB	Generates CRC for LSB first communication.
CRC_BIT_ORDER_LMS_MSB	Generates CRC for MSB first communication.

◆ crc_snoop_direction_t

enum crc_snoop_direction_t	
Snoop-On-Write/Read Switch (CRCSWR)	
Enumerator	
CRC_SNOOP_DIRECTION_RECEIVE	Snoop-on-read.
CRC_SNOOP_DIRECTION_TRANSMIT	Snoop-on-write.

◆ crc_snoop_address_t

enum crc_snoop_address_t

Snoop SCI register Address (lower 14 bits)

Enumerator	
CRC_SNOOP_ADDRESS_NONE	Snoop mode disabled.
CRC_SNOOP_ADDRESS_SCI0_TDR	Snoop SCI0 transmit data register.
CRC_SNOOP_ADDRESS_SCI1_TDR	Snoop SCI1 transmit data register.
CRC_SNOOP_ADDRESS_SCI2_TDR	Snoop SCI2 transmit data register.
CRC_SNOOP_ADDRESS_SCI3_TDR	Snoop SCI3 transmit data register.
CRC_SNOOP_ADDRESS_SCI4_TDR	Snoop SCI4 transmit data register.
CRC_SNOOP_ADDRESS_SCI5_TDR	Snoop SCI5 transmit data register.
CRC_SNOOP_ADDRESS_SCI6_TDR	Snoop SCI6 transmit data register.
CRC_SNOOP_ADDRESS_SCI7_TDR	Snoop SCI7 transmit data register.
CRC_SNOOP_ADDRESS_SCI8_TDR	Snoop SCI8 transmit data register.
CRC_SNOOP_ADDRESS_SCI9_TDR	Snoop SCI9 transmit data register.
CRC_SNOOP_ADDRESS_SCI0_FTDRL	Snoop SCI0 transmit FIFO data register.
CRC_SNOOP_ADDRESS_SCI1_FTDRL	Snoop SCI1 transmit FIFO data register.
CRC_SNOOP_ADDRESS_SCI2_FTDRL	Snoop SCI2 transmit FIFO data register.
CRC_SNOOP_ADDRESS_SCI3_FTDRL	Snoop SCI3 transmit FIFO data register.
CRC_SNOOP_ADDRESS_SCI4_FTDRL	Snoop SCI4 transmit FIFO data register.
CRC_SNOOP_ADDRESS_SCI5_FTDRL	Snoop SCI5 transmit FIFO data register.
CRC_SNOOP_ADDRESS_SCI6_FTDRL	Snoop SCI6 transmit FIFO data register.
CRC_SNOOP_ADDRESS_SCI7_FTDRL	Snoop SCI7 transmit FIFO data register.
CRC_SNOOP_ADDRESS_SCI8_FTDRL	Snoop SCI8 transmit FIFO data register.
CRC_SNOOP_ADDRESS_SCI9_FTDRL	Snoop SCI9 transmit FIFO data register.

CRC_SNOOP_ADDRESS_SCI0_RDR	Snoop SCI0 receive data register.
CRC_SNOOP_ADDRESS_SCI1_RDR	Snoop SCI1 receive data register.
CRC_SNOOP_ADDRESS_SCI2_RDR	Snoop SCI2 receive data register.
CRC_SNOOP_ADDRESS_SCI3_RDR	Snoop SCI3 receive data register.
CRC_SNOOP_ADDRESS_SCI4_RDR	Snoop SCI4 receive data register.
CRC_SNOOP_ADDRESS_SCI5_RDR	Snoop SCI5 receive data register.
CRC_SNOOP_ADDRESS_SCI6_RDR	Snoop SCI6 receive data register.
CRC_SNOOP_ADDRESS_SCI7_RDR	Snoop SCI7 receive data register.
CRC_SNOOP_ADDRESS_SCI8_RDR	Snoop SCI8 receive data register.
CRC_SNOOP_ADDRESS_SCI9_RDR	Snoop SCI9 receive data register.
CRC_SNOOP_ADDRESS_SCI0_FRDRL	Snoop SCI0 receive FIFO data register.
CRC_SNOOP_ADDRESS_SCI1_FRDRL	Snoop SCI1 receive FIFO data register.
CRC_SNOOP_ADDRESS_SCI2_FRDRL	Snoop SCI2 receive FIFO data register.
CRC_SNOOP_ADDRESS_SCI3_FRDRL	Snoop SCI3 receive FIFO data register.
CRC_SNOOP_ADDRESS_SCI4_FRDRL	Snoop SCI4 receive FIFO data register.
CRC_SNOOP_ADDRESS_SCI5_FRDRL	Snoop SCI5 receive FIFO data register.
CRC_SNOOP_ADDRESS_SCI6_FRDRL	Snoop SCI6 receive FIFO data register.
CRC_SNOOP_ADDRESS_SCI7_FRDRL	Snoop SCI7 receive FIFO data register.
CRC_SNOOP_ADDRESS_SCI8_FRDRL	Snoop SCI8 receive FIFO data register.
CRC_SNOOP_ADDRESS_SCI9_FRDRL	Snoop SCI9 receive FIFO data register.

4.3.9 CTSU Interface

Interfaces

Detailed Description

Interface for Capacitive Touch Sensing Unit (CTSU) functions.

Summary

The CTSU interface provides CTSU functionality.

The CTSU interface can be implemented by:

- Capacitive Touch Sensing Unit (r_ctsu)

Data Structures

struct	ctsu_callback_args_t
struct	ctsu_element_cfg_t
struct	ctsu_cfg_t
struct	ctsu_api_t
struct	ctsu_instance_t

Typedefs

typedef void	ctsu_ctrl_t

Enumerations

enum	ctsu_event_t
enum	ctsu_cap_t
enum	ctsu_txvsel_t
enum	ctsu_txvsel2_t
enum	ctsu_atune1_t
enum	ctsu_atune12_t
enum	ctsu_md_t
enum	ctsu_posel_t
enum	ctsu_ssdiv_t
enum	ctsu_specific_data_type_t

Data Structure Documentation

◆ ctsu_callback_args_t

struct ctsu_callback_args_t		
Callback function parameter data		
Data Fields		
ctsu_event_t	event	The event can be used to identify what caused the callback.
void const *	p_context	Placeholder for user data. Set in ctsu_api_t::open function in ctsu_cfg_t.

◆ ctsu_element_cfg_t

struct ctsu_element_cfg_t		
CTSU Configuration parameters. Element Configuration		
Data Fields		
ctsu_ssdiv_t	ssdiv	CTSU Spectrum Diffusion Frequency Division Setting (CTSU Only)
uint16_t	so	CTSU Sensor Offset Adjustment.
uint8_t	snum	CTSU Measurement Count Setting.
uint8_t	sdpa	CTSU Base Clock Setting.

◆ ctsu_cfg_t

struct ctsu_cfg_t	
User configuration structure, used in open function	
Data Fields	
ctsu_cap_t	cap
	CTSU Scan Start Trigger Select.
ctsu_txvsel_t	txvsel
	CTSU Transmission Power Supply Select.
ctsu_txvsel2_t	txvsel2
	CTSU Transmission Power Supply Select 2 (CTSU2 Only)

ctsu_atune1_t	atune1
	CTSU Power Supply Capacity Adjustment (CTSU Only)
ctsu_atune12_t	atune12
	CTSU Power Supply Capacity Adjustment (CTSU2 Only)
ctsu_md_t	md
	CTSU Measurement Mode Select.
ctsu_posel_t	posel
	CTSU Non-Measured Channel Output Select (CTSU2 Only)
uint8_t	ctsuchac0
	TS00-TS07 enable mask.
uint8_t	ctsuchac1
	TS08-TS15 enable mask.
uint8_t	ctsuchac2
	TS16-TS23 enable mask.
uint8_t	ctsuchac3
	TS24-TS31 enable mask.
uint8_t	ctsuchac4
	TS32-TS39 enable mask.

uint8_t	ctsuchtrc0
	TS00-TS07 mutual-tx mask.
uint8_t	ctsuchtrc1
	TS08-TS15 mutual-tx mask.
uint8_t	ctsuchtrc2
	TS16-TS23 mutual-tx mask.
uint8_t	ctsuchtrc3
	TS24-TS31 mutual-tx mask.
uint8_t	ctsuchtrc4
	TS32-TS39 mutual-tx mask.
ctsu_element_cfg_t const *	p_elements
	Pointer to elements configuration array.
uint8_t	num_rx
	Number of receive terminals.
uint8_t	num_tx
	Number of transmit terminals.
uint16_t	num_moving_average
	Number of moving average for measurement data.
bool	tunning_enable

	Initial offset tuning flag.
void(*	p_callback)(ctsu_callback_args_t *p_args)
	Callback provided when CTSUFN ISR occurs.
transfer_instance_t const *	p_transfer_tx
	DTC instance for transmit at CTSUWR. Set to NULL if unused.
transfer_instance_t const *	p_transfer_rx
	DTC instance for receive at CTSURD. Set to NULL if unused.
adc_instance_t const *	p_adc_instance
	ADC instance for temperature correction.
IRQn_Type	write_irq
	CTSU_CTSUWR interrupt vector.
IRQn_Type	read_irq
	CTSU_CTSURD interrupt vector.
IRQn_Type	end_irq
	CTSU_CTSUFN interrupt vector.
void const *	p_context
	User defined context passed into callback function.
void const *	p_extend
	Pointer to extended configuration by instance of interface.

uint16_t	tuning_self_target_value
	Target self value for initial offset tuning.
uint16_t	tuning_mutual_target_value
	Target mutual value for initial offset tuning.

◆ **ctsu_api_t**

struct ctsu_api_t

Functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg)
fsp_err_t(*	scanStart)(ctsu_ctrl_t *const p_ctrl)
fsp_err_t(*	dataGet)(ctsu_ctrl_t *const p_ctrl, uint16_t *p_data)
fsp_err_t(*	scanStop)(ctsu_ctrl_t *const p_ctrl)
fsp_err_t(*	diagnosis)(ctsu_ctrl_t *const p_ctrl)
fsp_err_t(*	callbackSet)(ctsu_ctrl_t *const p_api_ctrl, void(*p_callback)(ctsu_callback_args_t *), void const *const p_context, ctsu_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(ctsu_ctrl_t *const p_ctrl)
fsp_err_t(*	specificDataGet)(ctsu_ctrl_t *const p_ctrl, uint16_t *p_specific_data, ctsu_specific_data_type_t specific_data_type)
fsp_err_t(*	dataInsert)(ctsu_ctrl_t *const p_ctrl, uint16_t *p_insert_data)

Field Documentation

◆ open

fsp_err_t(* ctsu_api_t::open) (ctsu_ctrl_t *const p_ctrl, ctsu_cfg_t const *const p_cfg)

Open driver.

Implemented as

- R_CTSU_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ scanStart

fsp_err_t(* ctsu_api_t::scanStart) (ctsu_ctrl_t *const p_ctrl)

Scan start.

Implemented as

- R_CTSU_ScanStart()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ dataGet

fsp_err_t(* ctsu_api_t::dataGet) (ctsu_ctrl_t *const p_ctrl, uint16_t *p_data)

Data get.

Implemented as

- R_CTSU_DataGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_data	Pointer to get data array.

◆ scanStop

fsp_err_t(* ctsu_api_t::scanStop) (ctsu_ctrl_t *const p_ctrl)

ScanStop.

Implemented as

- R_CTSU_ScanStop()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ diagnosis

fsp_err_t(* ctsu_api_t::diagnosis) (ctsu_ctrl_t *const p_ctrl)

Diagnosis.

Implemented as

- R_CTSU_Diagnosis()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ callbackSet

fsp_err_t(* ctsu_api_t::callbackSet) (ctsu_ctrl_t *const p_api_ctrl, void(*p_callback)(ctsu_callback_args_t *), void const *const p_context, ctsu_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_CTSU_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the CTSU control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* ctsu_api_t::close) (ctsu_ctrl_t *const p_ctrl)

Close driver.

Implemented as

- R_CTSU_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ specificDataGet

fsp_err_t(* ctsu_api_t::specificDataGet) (ctsu_ctrl_t *const p_ctrl, uint16_t *p_specific_data, ctsu_specific_data_type_t specific_data_type)

Specific Data get.

Implemented as

- R_CTSU_SpecificDataGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_specific_data	Pointer to get specific data array.
[in]	specific_data_type	Specific data type

◆ dataInsert

fsp_err_t(* ctsu_api_t::dataInsert) (ctsu_ctrl_t *const p_ctrl, uint16_t *p_insert_data)

Data Insert.

Implemented as

- R_CTSU_DataInsert()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_insert_data	Pointer to insert data.

◆ ctsu_instance_t

struct ctsu_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
ctsu_ctrl_t *	p_ctrl	Pointer to the control structure

		for this instance.
ctsu_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ctsu_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ ctsu_ctrl_t

typedef void ctsu_ctrl_t
CTSU Control block. Allocate an instance specific control block to pass into the API calls. **Implemented as** ◦ ctsu_instance_ctrl_t

Enumeration Type Documentation

◆ ctsu_event_t

enum ctsu_event_t

CTSU Events for callback function

Enumerator	
CTSU_EVENT_SCAN_COMPLETE	Normal end.
CTSU_EVENT_OVERFLOW	Sensor counter overflow (CTSUST.CTSUSOVF set)
CTSU_EVENT_ICOMP	Abnormal TSCAP voltage (CTSUERRS.CTSUICOMP set)
CTSU_EVENT_ICOMP1	Abnormal sensor current (CTSUSR.ICOMP1 set)

◆ ctsu_cap_t

enum ctsu_cap_t

CTSU Scan Start Trigger Select

Enumerator	
CTSU_CAP_SOFTWARE	Scan start by software trigger.
CTSU_CAP_EXTERNAL	Scan start by external trigger.

◆ ctsu_txvsel_t

enum ctsu_txvsel_t	
CTSU Transmission Power Supply Select	
Enumerator	
CTSU_TXVSEL_VCC	VCC selected.
CTSU_TXVSEL_INTERNAL_POWER	Internal logic power supply selected.

◆ ctsu_txvsel2_t

enum ctsu_txvsel2_t	
CTSU Transmission Power Supply Select 2 (CTSU2 Only)	
Enumerator	
CTSU_TXVSEL_MODE	Follow TXVSEL setting.
CTSU_TXVSEL_VCC_PRIVATE	VCC private selected.

◆ ctsu_atune1_t

enum ctsu_atune1_t	
CTSU Power Supply Capacity Adjustment (CTSU Only)	
Enumerator	
CTSU_ATUNE1_NORMAL	Normal output (40uA)
CTSU_ATUNE1_HIGH	High-current output (80uA)

◆ ctsu_atune12_t

enum ctsu_atune12_t	
CTSU Power Supply Capacity Adjustment (CTSU2 Only)	
Enumerator	
CTSU_ATUNE12_80UA	High-current output (80uA)
CTSU_ATUNE12_40UA	Normal output (40uA)
CTSU_ATUNE12_20UA	Low-current output (20uA)
CTSU_ATUNE12_160UA	Very high-current output (160uA)

◆ ctsu_md_t

enum ctsu_md_t	
CTSU Measurement Mode Select	
Enumerator	
CTSU_MODE_SELF_MULTI_SCAN	Self-capacitance multi scan mode.
CTSU_MODE_MUTUAL_FULL_SCAN	Mutual capacitance full scan mode.
CTSU_MODE_MUTUAL_CFC_SCAN	Mutual capacitance cfc scan mode (CTSU2 Only)
CTSU_MODE_CURRENT_SCAN	Current scan mode (CTSU2 Only)
CTSU_MODE_CORRECTION_SCAN	Correction scan mode (CTSU2 Only)
CTSU_MODE_DIAGNOSIS_SCAN	Diagnosis scan mode.

◆ ctsu_posel_t

enum ctsu_posel_t	
CTSU Non-Measured Channel Output Select (CTSU2 Only)	
Enumerator	
CTSU_POSEL_LOW_GPIO	Output low through GPIO.
CTSU_POSEL_HI_Z	Hi-Z.
CTSU_POSEL_LOW	Setting prohibited.
CTSU_POSEL_SAME_PULSE	Same phase pulse output as transmission channel through the power setting by the TXVSEL[1:0] bits.

◆ ctsu_ssdiv_t

enum ctsu_ssdiv_t	
CTSU Spectrum Diffusion Frequency Division Setting (CTSU Only)	
Enumerator	
CTSU_SSDIV_4000	4.00 <= Base clock frequency (MHz)
CTSU_SSDIV_2000	2.00 <= Base clock frequency (MHz) < 4.00
CTSU_SSDIV_1330	1.33 <= Base clock frequency (MHz) < 2.00
CTSU_SSDIV_1000	1.00 <= Base clock frequency (MHz) < 1.33
CTSU_SSDIV_0800	0.80 <= Base clock frequency (MHz) < 1.00
CTSU_SSDIV_0670	0.67 <= Base clock frequency (MHz) < 0.80
CTSU_SSDIV_0570	0.57 <= Base clock frequency (MHz) < 0.67
CTSU_SSDIV_0500	0.50 <= Base clock frequency (MHz) < 0.57
CTSU_SSDIV_0440	0.44 <= Base clock frequency (MHz) < 0.50
CTSU_SSDIV_0400	0.40 <= Base clock frequency (MHz) < 0.44
CTSU_SSDIV_0360	0.36 <= Base clock frequency (MHz) < 0.40
CTSU_SSDIV_0330	0.33 <= Base clock frequency (MHz) < 0.36
CTSU_SSDIV_0310	0.31 <= Base clock frequency (MHz) < 0.33
CTSU_SSDIV_0290	0.29 <= Base clock frequency (MHz) < 0.31
CTSU_SSDIV_0270	0.27 <= Base clock frequency (MHz) < 0.29
CTSU_SSDIV_0000	0.00 <= Base clock frequency (MHz) < 0.27

◆ ctsu_specific_data_type_t

enum ctsu_specific_data_type_t
CTSU select data type for slect data get

4.3.10 DAC Interface

Interfaces

Detailed Description

Interface for D/A converters.

Summary

The DAC interface provides standard Digital/Analog Converter functionality. A DAC application writes digital sample data to the device and generates analog output on the DAC output pin.

Implemented by:

- Digital to Analog Converter (r_dac)
- Digital to Analog Converter (r_dac8)

Data Structures

struct	dac_info_t
struct	dac_cfg_t
struct	dac_api_t
struct	dac_instance_t

Typedefs

typedef void	dac_ctrl_t

Enumerations

enum	dac_data_format_t

Data Structure Documentation

◆ dac_info_t

struct dac_info_t		
DAC information structure to store various information for a DAC		
Data Fields		
uint8_t	bit_width	Resolution of the DAC.

◆ dac_cfg_t

struct dac_cfg_t

DAC Open API configuration parameter

Data Fields		
uint8_t	channel	ID associated with this DAC channel.
bool	ad_da_synchronized	AD/DA synchronization.
void const *	p_extend	

◆ dac_api_t

struct dac_api_t
DAC driver structure. General DAC functions implemented at the HAL layer follow this API.

Data Fields

fsp_err_t(*	open)(dac_ctrl_t *const p_ctrl, dac_cfg_t const *const p_cfg)
fsp_err_t(*	close)(dac_ctrl_t *const p_ctrl)
fsp_err_t(*	write)(dac_ctrl_t *const p_ctrl, uint16_t value)
fsp_err_t(*	start)(dac_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(dac_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* dac_api_t::open) (dac_ctrl_t *const p_ctrl, dac_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_DAC_Open()
- R_DAC8_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ close

fsp_err_t(* dac_api_t::close) (dac_ctrl_t *const p_ctrl)
Close the D/A Converter.

Implemented as

- R_DAC_Close()
- R_DAC8_Close()

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.

◆ write

fsp_err_t(* dac_api_t::write) (dac_ctrl_t *const p_ctrl, uint16_t value)
Write sample value to the D/A Converter.

Implemented as

- R_DAC_Write()
- R_DAC8_Write()

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.
[in]	value	Sample value to be written to the D/A Converter.

◆ start

fsp_err_t(* dac_api_t::start) (dac_ctrl_t *const p_ctrl)
Start the D/A Converter if it has not been started yet.

Implemented as

- R_DAC_Start()
- R_DAC8_Start()

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.

◆ **stop**

fsp_err_t(* dac_api_t::stop) (dac_ctrl_t *const p_ctrl)

Stop the D/A Converter if the converter is running.

Implemented as

- R_DAC_Stop()
- R_DAC8_Stop()

Parameters

[in]	p_ctrl	Control block set in dac_api_t::open call for this timer.

◆ **dac_instance_t**

struct dac_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
dac_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
dac_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
dac_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ **dac_ctrl_t**

typedef void dac_ctrl_t

DAC control block. Allocate an instance specific control block to pass into the DAC API calls.

Implemented as

- dac_instance_ctrl_t
- dac8_instance_ctrl_t

Enumeration Type Documentation

◆ dac_data_format_t

enum dac_data_format_t	
DAC Open API data format settings.	
Enumerator	
DAC_DATA_FORMAT_FLUSH_RIGHT	LSB of data is flush to the right leaving the top 4 bits unused.
DAC_DATA_FORMAT_FLUSH_LEFT	MSB of data is flush to the left leaving the bottom 4 bits unused.

4.3.11 Display Interface

Interfaces

Detailed Description

Interface for LCD panel displays.

Summary

The display interface provides standard display functionality:

- Signal timing configuration for LCD panels with RGB interface.
- Dot clock source selection (internal or external) and frequency divider.
- Blending of multiple graphics layers on the background screen.
- Color correction (brightness/configuration/gamma correction).
- Interrupts and callback function.

Implemented by: Graphics LCD Controller (r_glcdc)

Data Structures

struct	display_timing_t
struct	display_color_t
struct	display_coordinate_t
struct	display_brightness_t
struct	display_contrast_t
struct	display_correction_t

struct	gamma_correction_t
struct	display_gamma_correction_t
struct	display_clut_t
struct	display_input_cfg_t
struct	display_output_cfg_t
struct	display_layer_t
struct	display_callback_args_t
struct	display_cfg_t
struct	display_runtime_cfg_t
struct	display_clut_cfg_t
struct	display_status_t
struct	display_api_t
struct	display_instance_t

Typedefs

typedef void	display_ctrl_t

Enumerations

enum	display_frame_layer_t
enum	display_state_t
enum	display_event_t
enum	display_in_format_t
enum	display_out_format_t
enum	display_endian_t
enum	display_color_order_t
enum	display_signal_polarity_t
enum	display_sync_edge_t

enum	display_fade_control_t
enum	display_fade_status_t

Data Structure Documentation

◆ display_timing_t

struct display_timing_t		
Display signal timing setting		
Data Fields		
uint16_t	total_cyc	Total cycles in one line or total lines in one frame.
uint16_t	display_cyc	Active video cycles or lines.
uint16_t	back_porch	Back porch cycles or lines.
uint16_t	sync_width	Sync signal asserting width.
display_signal_polarity_t	sync_polarity	Sync signal polarity.

◆ display_color_t

struct display_color_t
RGB Color setting

◆ display_coordinate_t

struct display_coordinate_t		
Contrast (gain) correction setting		
Data Fields		
int16_t	x	Coordinate X, this allows to set signed value.
int16_t	y	Coordinate Y, this allows to set signed value.

◆ display_brightness_t

struct display_brightness_t		
Brightness (DC) correction setting		
Data Fields		
bool	enable	Brightness Correction On/Off.
uint16_t	r	Brightness (DC) adjustment for R channel.
uint16_t	g	Brightness (DC) adjustment for G channel.

uint16_t	b	Brightness (DC) adjustment for B channel.

◆ display_contrast_t

struct display_contrast_t		
Contrast (gain) correction setting		
Data Fields		
bool	enable	Contrast Correction On/Off.
uint8_t	r	Contrast (gain) adjustment for R channel.
uint8_t	g	Contrast (gain) adjustment for G channel.
uint8_t	b	Contrast (gain) adjustment for B channel.

◆ display_correction_t

struct display_correction_t		
Color correction setting		
Data Fields		
display_brightness_t	brightness	Brightness.
display_contrast_t	contrast	Contrast.

◆ gamma_correction_t

struct gamma_correction_t		
Gamma correction setting for each color		
Data Fields		
bool	enable	Gamma Correction On/Off.
uint16_t *	gain	Gain adjustment.
uint16_t *	threshold	Start threshold.

◆ display_gamma_correction_t

struct display_gamma_correction_t		
Gamma correction setting		
Data Fields		
gamma_correction_t	r	Gamma correction for R channel.
gamma_correction_t	g	Gamma correction for G channel.
gamma_correction_t	b	Gamma correction for B

		channel.

◆ display_clut_t

struct display_clut_t		
CLUT setting		
Data Fields		
uint32_t	color_num	The number of colors in CLUT.
const uint32_t *	p_clut	Address of the area storing the CLUT data (in ARGB8888 format)

◆ display_input_cfg_t

struct display_input_cfg_t		
Graphics plane input configuration structure		
Data Fields		
uint32_t *	p_base	Base address to the frame buffer.
uint16_t	hsize	Horizontal pixel size in a line.
uint16_t	vsize	Vertical pixel size in a frame.
uint32_t	hstride	Memory stride (bytes) in a line.
display_in_format_t	format	Input format setting.
bool	line_descending_enable	Line descending enable.
bool	lines_repeat_enable	Line repeat enable.
uint16_t	lines_repeat_times	Expected number of line repeating.

◆ display_output_cfg_t

struct display_output_cfg_t		
Display output configuration structure		
Data Fields		
display_timing_t	htiming	Horizontal display cycle setting.
display_timing_t	vtiming	Vertical display cycle setting.
display_out_format_t	format	Output format setting.
display_endian_t	endian	Bit order of output data.
display_color_order_t	color_order	Color order in pixel.
display_signal_polarity_t	data_enable_polarity	Data Enable signal polarity.
display_sync_edge_t	sync_edge	Signal sync edge selection.
display_color_t	bg_color	Background color.

display_brightness_t	brightness	Brightness setting.
display_contrast_t	contrast	Contrast setting.
display_gamma_correction_t *	p_gamma_correction	Pointer to gamma correction setting.
bool	dithering_on	Dithering on/off.

◆ display_layer_t

struct display_layer_t		
Graphics layer blend setup parameter structure		
Data Fields		
display_coordinate_t	coordinate	Blending location (starting point of image)
display_color_t	bg_color	Color outside region.
display_fade_control_t	fade_control	Layer fade-in/out control on/off.
uint8_t	fade_speed	Layer fade-in/out frame rate.

◆ display_callback_args_t

struct display_callback_args_t		
Display callback parameter definition		
Data Fields		
display_event_t	event	Event code.
void const *	p_context	Context provided to user during callback.

◆ display_cfg_t

struct display_cfg_t	
Display main configuration structure	
Data Fields	
display_input_cfg_t	input [2]
	Graphics input frame setting. More...
display_output_cfg_t	output
	Graphics output frame setting.
display_layer_t	layer [2]
	Graphics layer blend setting.

uint8_t	line_detect_ipl
	Line detect interrupt priority.
uint8_t	underflow_1_ipl
	Underflow 1 interrupt priority.
uint8_t	underflow_2_ipl
	Underflow 2 interrupt priority.
IRQn_Type	line_detect_irq
	Line detect interrupt vector.
IRQn_Type	underflow_1_irq
	Underflow 1 interrupt vector.
IRQn_Type	underflow_2_irq
	Underflow 2 interrupt vector.
void(*	p_callback)(display_callback_args_t *p_args)
	Pointer to callback function. More...
void const *	p_context
	User defined context passed into callback function.
void const *	p_extend
	Display hardware dependent configuration. More...

Field Documentation

◆ input

display_input_cfg_t display_cfg_t::input[2]

Graphics input frame setting.

Generic configuration for display devices

◆ p_callback

void(* display_cfg_t::p_callback) (display_callback_args_t *p_args)

Pointer to callback function.

Configuration for display event processing

◆ p_extend

void const* display_cfg_t::p_extend

Display hardware dependent configuration.

Pointer to display peripheral specific configuration

◆ display_runtime_cfg_t

struct display_runtime_cfg_t

Display main configuration structure

Data Fields		
display_input_cfg_t	input	Graphics input frame setting. Generic configuration for display devices
display_layer_t	layer	Graphics layer alpha blending setting.

◆ display_clut_cfg_t

struct display_clut_cfg_t

Display CLUT configuration structure

Data Fields		
uint32_t *	p_base	Pointer to CLUT source data.
uint16_t	start	Beginning of CLUT entry to be updated.
uint16_t	size	Size of CLUT entry to be updated.

◆ display_status_t

struct display_status_t

Display Status		
Data Fields		
display_state_t	state	Status of GLCDC module.
display_fade_status_t	fade_status[DISPLAY_FRAME_LAYER_2+1]	Status of fade-in/fade-out status.

◆ display_api_t

struct display_api_t	
Shared Interface definition for display peripheral	
Data Fields	
fsp_err_t(*	open)(display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg)
fsp_err_t(*	close)(display_ctrl_t *const p_ctrl)
fsp_err_t(*	start)(display_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(display_ctrl_t *const p_ctrl)
fsp_err_t(*	layerChange)(display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)
fsp_err_t(*	bufferChange)(display_ctrl_t const *const p_ctrl, uint8_t *const framebuffer, display_frame_layer_t frame)
fsp_err_t(*	correction)(display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param)
fsp_err_t(*	clut)(display_ctrl_t const *const p_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t layer)
fsp_err_t(*	clutEdit)(display_ctrl_t const *const p_ctrl, display_frame_layer_t layer, uint8_t index, uint32_t color)
fsp_err_t(*	statusGet)(display_ctrl_t const *const p_ctrl, display_status_t *const p_status)

Field Documentation

◆ open

fsp_err_t(* display_api_t::open) (display_ctrl_t *const p_ctrl, display_cfg_t const *const p_cfg)

Open display device.

Implemented as

- R_GLCDC_Open()

Parameters

[in,out]	p_ctrl	Pointer to display interface control block. Must be declared by user. Value set here.
[in]	p_cfg	Pointer to display configuration structure. All elements of this structure must be set by user.

◆ close

fsp_err_t(* display_api_t::close) (display_ctrl_t *const p_ctrl)

Close display device.

Implemented as

- R_GLCDC_Close()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ start

fsp_err_t(* display_api_t::start) (display_ctrl_t *const p_ctrl)

Display start.

Implemented as

- R_GLCDC_Start()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ stop

fsp_err_t(* display_api_t::stop) (display_ctrl_t *const p_ctrl)

Display stop.

Implemented as

- R_GLCDC_Stop()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ layerChange

fsp_err_t(* display_api_t::layerChange) (display_ctrl_t const *const p_ctrl, display_runtime_cfg_t const *const p_cfg, display_frame_layer_t frame)

Change layer parameters at runtime.

Implemented as

- R_GLCDC_LayerChange()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	p_cfg	Pointer to run-time layer configuration structure.
[in]	frame	Number of graphic frames.

◆ bufferChange

fsp_err_t(* display_api_t::bufferChange) (display_ctrl_t const *const p_ctrl, uint8_t *const framebuffer, display_frame_layer_t frame)

Change layer framebuffer pointer.

Implemented as

- R_GLCDC_BufferChange()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	framebuffer	Pointer to desired framebuffer.
[in]	frame	Number of graphic frames.

◆ correction

fsp_err_t(* display_api_t::correction) (display_ctrl_t const *const p_ctrl, display_correction_t const *const p_param)

Color correction.

Implemented as

- R_GLCDC_ColorCorrection()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	param	Pointer to color correction configuration structure.

◆ clut

fsp_err_t(* display_api_t::clut) (display_ctrl_t const *const p_ctrl, display_clut_cfg_t const *const p_clut_cfg, display_frame_layer_t layer)

Set CLUT for display device.

Implemented as

- R_GLCDC_ClutUpdate()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	p_clut_cfg	Pointer to CLUT configuration structure.
[in]	layer	Layer number corresponding to the CLUT.

◆ clutEdit

fsp_err_t(* display_api_t::clutEdit) (display_ctrl_t const *const p_ctrl, display_frame_layer_t layer, uint8_t index, uint32_t color)

Set CLUT element for display device.

Implemented as

- R_GLCDC_ClutEdit()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	layer	Layer number corresponding to the CLUT.
[in]	index	CLUT element index.
[in]	color	Desired CLUT index color.

◆ statusGet

fsp_err_t(* display_api_t::statusGet) (display_ctrl_t const *const p_ctrl, display_status_t *const p_status)

Get status for display device.

Implemented as

- R_GLCDC_StatusGet()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	status	Pointer to display interface status structure.

◆ display_instance_t

struct display_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
display_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
display_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
display_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ display_ctrl_t

typedef void display_ctrl_t
Display control block. Allocate an instance specific control block to pass into the display API calls. **Implemented as** ◦ glcdc_instance_ctrl_tDisplay control block

Enumeration Type Documentation

◆ display_frame_layer_t

enum display_frame_layer_t

Display frame number

Enumerator	
DISPLAY_FRAME_LAYER_1	Frame layer 1.
DISPLAY_FRAME_LAYER_2	Frame layer 2.

◆ display_state_t

enum display_state_t

Display interface operation state

Enumerator	
DISPLAY_STATE_CLOSED	Display closed.
DISPLAY_STATE_OPENED	Display opened.
DISPLAY_STATE_DISPLAYING	Displaying.

◆ display_event_t

enum display_event_t	
Display event codes	
Enumerator	
DISPLAY_EVENT_GR1_UNDERFLOW	Graphics frame1 underflow occurs.
DISPLAY_EVENT_GR2_UNDERFLOW	Graphics frame2 underflow occurs.
DISPLAY_EVENT_LINE_DETECTION	Designated line is processed.

◆ display_in_format_t

enum display_in_format_t	
Input format setting	
Enumerator	
DISPLAY_IN_FORMAT_32BITS_ARGB8888	ARGB8888, 32 bits.
DISPLAY_IN_FORMAT_32BITS_RGB888	RGB888, 32 bits.
DISPLAY_IN_FORMAT_16BITS_RGB565	RGB565, 16 bits.
DISPLAY_IN_FORMAT_16BITS_ARGB1555	ARGB1555, 16 bits.
DISPLAY_IN_FORMAT_16BITS_ARGB4444	ARGB4444, 16 bits.
DISPLAY_IN_FORMAT_CLUT8	CLUT8.
DISPLAY_IN_FORMAT_CLUT4	CLUT4.
DISPLAY_IN_FORMAT_CLUT1	CLUT1.

◆ display_out_format_t

enum display_out_format_t	
Output format setting	
Enumerator	
DISPLAY_OUT_FORMAT_24BITS_RGB888	RGB888, 24 bits.
DISPLAY_OUT_FORMAT_18BITS_RGB666	RGB666, 18 bits.
DISPLAY_OUT_FORMAT_16BITS_RGB565	RGB565, 16 bits.
DISPLAY_OUT_FORMAT_8BITS_SERIAL	SERIAL, 8 bits.

◆ display_endian_t

enum display_endian_t	
Data endian select	
Enumerator	
DISPLAY_ENDIAN_LITTLE	Little-endian.
DISPLAY_ENDIAN_BIG	Big-endian.

◆ display_color_order_t

enum display_color_order_t	
RGB color order select	
Enumerator	
DISPLAY_COLOR_ORDER_RGB	Color order RGB.
DISPLAY_COLOR_ORDER_BGR	Color order BGR.

◆ display_signal_polarity_t

enum display_signal_polarity_t	
Polarity of a signal select	
Enumerator	
DISPLAY_SIGNAL_POLARITY_LOACTIVE	Low active signal.
DISPLAY_SIGNAL_POLARITY_HIACTIVE	High active signal.

◆ display_sync_edge_t

enum display_sync_edge_t	
Signal synchronization edge select	
Enumerator	
DISPLAY_SIGNAL_SYNC_EDGE_RISING	Signal is synchronized to rising edge.
DISPLAY_SIGNAL_SYNC_EDGE_FALLING	Signal is synchronized to falling edge.

◆ display_fade_control_t

enum display_fade_control_t	
Fading control	
Enumerator	
DISPLAY_FADE_CONTROL_NONE	Applying no fading control.
DISPLAY_FADE_CONTROL_FADEIN	Applying fade-in control.
DISPLAY_FADE_CONTROL_FADEOUT	Applying fade-out control.

◆ display_fade_status_t

enum display_fade_status_t	
Fading status	
Enumerator	
DISPLAY_FADE_STATUS_NOT_UNDERWAY	Fade-in/fade-out is not in progress.
DISPLAY_FADE_STATUS_FADING_UNDERWAY	Fade-in or fade-out is in progress.
DISPLAY_FADE_STATUS_PENDING	Fade-in/fade-out is configured but not yet started.

4.3.12 DOC Interface

Interfaces

Detailed Description

Interface for the Data Operation Circuit.

Defines the API and data structures for the DOC implementation of the Data Operation Circuit (DOC) interface.

Summary

This module implements the DOC_API using the Data Operation Circuit (DOC).

Implemented by: Data Operation Circuit (r_doc)

Data Structures

struct	doc_status_t
struct	doc_callback_args_t
struct	doc_cfg_t
struct	doc_api_t
struct	doc_instance_t

Typedefs

typedef void	doc_ctrl_t

Enumerations

enum	doc_event_t
enum	doc_bit_width_t

Data Structure Documentation

◆ doc_status_t

struct doc_status_t
DOC status

◆ doc_callback_args_t

struct doc_callback_args_t		
Callback function parameter data.		
Data Fields		
void const *	p_context	Set in doc_api_t::open function in doc_cfg_t. Placeholder for user data.

◆ doc_cfg_t

struct doc_cfg_t	
User configuration structure, used in the open function.	
Data Fields	
doc_event_t	event
	Select enumerated value from doc_event_t.
doc_bit_width_t	bit_width
	The bit width of operations.
uint32_t	doc_data
uint32_t	doc_data_extra
uint8_t	ipl
	DOC interrupt priority.

IRQn_Type	irq
	NVIC interrupt number assigned to this instance.
void(*	p_callback)(doc_callback_args_t *p_args)
void const *	p_context

Field Documentation

◆ **doc_data**

uint32_t doc_cfg_t::doc_data

Initial/Reference data for addition, subtraction, and comparison operations.

- In Addition and Subtraction mode, this value sets the initial value of the operations.
- In Comparison match, mismatch, lower, and upper modes, this value is compared with data that is written.
- In Comparison inside window and outside window modes, this value is used as the lower bound for comparisons.

◆ **doc_data_extra**

uint32_t doc_cfg_t::doc_data_extra

Additional reference data for use in Window Comparison operations.

- In Comparison inside window and outside window modes, this value is used as the upper bound for comparisons.

◆ **p_callback**

void(* doc_cfg_t::p_callback) (doc_callback_args_t *p_args)

Callback provided when a DOC ISR occurs.

◆ **p_context**

void const* doc_cfg_t::p_context

Placeholder for user data. Passed to the user callback in doc_callback_args_t.

◆ **doc_api_t**

struct doc_api_t

Data Operation Circuit (DOC) API structure. DOC functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)
fsp_err_t(*	close)(doc_ctrl_t *const p_ctrl)
fsp_err_t(*	statusGet)(doc_ctrl_t *const p_ctrl, doc_status_t *p_status)
fsp_err_t(*	read)(doc_ctrl_t *const p_ctrl, uint32_t *p_result)
fsp_err_t(*	write)(doc_ctrl_t *const p_ctrl, uint32_t data)
fsp_err_t(*	callbackSet)(doc_ctrl_t *const p_api_ctrl, void(*p_callback)(doc_callback_args_t *), void const *const p_context, doc_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* doc_api_t::open) (doc_ctrl_t *const p_ctrl, doc_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_DOC_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ close

fsp_err_t(* doc_api_t::close) (doc_ctrl_t *const p_ctrl)

Allow the driver to be reconfigured. Will reduce power consumption.

Implemented as

- R_DOC_Close()

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.

◆ statusGet

fsp_err_t(* doc_api_t::statusGet) (doc_ctrl_t *const p_ctrl, doc_status_t *p_status)

DEPRECATED - Gets the result of addition/subtraction operations and stores it in the provided pointer p_status.

Implemented as

- R_DOC_StatusGet()

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.
[out]	p_data	Provides the 16 bit result of the addition/subtraction operation at the user defined location.

◆ read

fsp_err_t(* doc_api_t::read) (doc_ctrl_t *const p_ctrl, uint32_t *p_result)

Gets the result of addition/subtraction operations and stores it in the provided pointer p_result.

Implemented as

- R_DOC_Read()

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.
[in]	p_result	The result of the DOC operation.

◆ **write**

fsp_err_t(* doc_api_t::write) (doc_ctrl_t *const p_ctrl, uint32_t data)

Write to the DODIR register.

Implemented as

- R_DOC_Write()

Parameters

[in]	p_ctrl	Control block set in doc_api_t::open call.
[in]	data	data to be written to DOC DODIR register.

◆ **callbackSet**

fsp_err_t(* doc_api_t::callbackSet) (doc_ctrl_t *const p_api_ctrl, void(*p_callback)(doc_callback_args_t *), void const *const p_context, doc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_DOC_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the DOC control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ **doc_instance_t**

struct doc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
doc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
doc_cfg_t const *	p_cfg	Pointer to the configuration

		structure for this instance.
doc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ doc_ctrl_t

typedef void doc_ctrl_t

DOC control block. Allocate an instance specific control block to pass into the DOC API calls.

Implemented as

- doc_instance_ctrl_t

Enumeration Type Documentation

◆ doc_event_t

enum doc_event_t

Event that can trigger a callback function.

Enumerator	
DOC_EVENT_COMPARISON_MISMATCH	The data is not equal to the reference data setting.
DOC_EVENT_ADDITION	Addition resulted in a value greater than the max for the configured bit width.
DOC_EVENT_SUBTRACTION	Subtraction resulted in a value less than 0.
DOC_EVENT_COMPARISON_MATCH	The data is equal to the reference data settting.
DOC_EVENT_COMPARISON_LOWER	The data is less than the reference data setting.
DOC_EVENT_COMPARISON_UPPER	The data is greater than the reference data setting.
DOC_EVENT_COMPARISON_INSIDE_WINDOW	The data is between the two reference data settings.
DOC_EVENT_COMPARISON_OUTSIDE_WINDOW	The data is outside the two reference data setttings.

◆ doc_bit_width_t

enum doc_bit_width_t	
The bit width used during operations.	
Enumerator	
DOC_BIT_WIDTH_16	Operations are 16-bit.
DOC_BIT_WIDTH_32	Operations are 32-bit.

4.3.13 ELC Interface

Interfaces

Detailed Description

Interface for the Event Link Controller.

Data Structures

struct	elc_cfg_t
struct	elc_api_t
struct	elc_instance_t

Typedefs

typedef void	elc_ctrl_t

Enumerations

enum	elc_peripheral_t
enum	elc_software_event_t

Data Structure Documentation

◆ elc_cfg_t

struct elc_cfg_t		
Main configuration structure for the Event Link Controller		
Data Fields		
elc_event_t const	link[ELC_PERIPHERAL_NUM]	Event link register (ELSR)

		settings.

◆ **elc_api_t**

struct elc_api_t	
ELC driver structure. General ELC functions implemented at the HAL layer follow this API.	
Data Fields	
fsp_err_t(*	open)(elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg)
fsp_err_t(*	close)(elc_ctrl_t *const p_ctrl)
fsp_err_t(*	softwareEventGenerate)(elc_ctrl_t *const p_ctrl, elc_software_event_t event_num)
fsp_err_t(*	linkSet)(elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal)
fsp_err_t(*	linkBreak)(elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral)
fsp_err_t(*	enable)(elc_ctrl_t *const p_ctrl)
fsp_err_t(*	disable)(elc_ctrl_t *const p_ctrl)

Field Documentation

◆ **open**

fsp_err_t(* elc_api_t::open) (elc_ctrl_t *const p_ctrl, elc_cfg_t const *const p_cfg)

Initialize all links in the Event Link Controller.

Implemented as

- R_ELC_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* elc_api_t::close) (elc_ctrl_t *const p_ctrl)

Disable all links in the Event Link Controller and close the API.

Implemented as

- R_ELC_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ softwareEventGenerate

fsp_err_t(* elc_api_t::softwareEventGenerate) (elc_ctrl_t *const p_ctrl, elc_software_event_t event_num)

Generate a software event in the Event Link Controller.

Implemented as

- R_ELC_SoftwareEventGenerate()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	eventNum	Software event number to be generated.

◆ linkSet

fsp_err_t(* elc_api_t::linkSet) (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral, elc_event_t signal)

Create a single event link.

Implemented as

- R_ELC_LinkSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	peripheral	The peripheral block that will receive the event signal.
[in]	signal	The event signal.

◆ linkBreak

fsp_err_t(* elc_api_t::linkBreak) (elc_ctrl_t *const p_ctrl, elc_peripheral_t peripheral)

Break an event link.

Implemented as

- R_ELC_LinkBreak()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	peripheral	The peripheral that should no longer be linked.

◆ enable

fsp_err_t(* elc_api_t::enable) (elc_ctrl_t *const p_ctrl)

Enable the operation of the Event Link Controller.

Implemented as

- R_ELC_Enable()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ disable

fsp_err_t(* elc_api_t::disable) (elc_ctrl_t *const p_ctrl)

Disable the operation of the Event Link Controller.

Implemented as

- R_ELC_Disable()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ elc_instance_t

struct elc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
elc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
elc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.

elc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ **elc_ctrl_t**

typedef void elc_ctrl_t
ELC control block. Allocate an instance specific control block to pass into the ELC API calls. **Implemented as** ◦ elc_instance_ctrl_t

Enumeration Type Documentation

◆ **elc_peripheral_t**

enum elc_peripheral_t
Possible peripherals to be linked to event signals (not all available on all MCUs)

◆ **elc_software_event_t**

enum elc_software_event_t
Software event number

Enumerator	
ELC_SOFTWARE_EVENT_0	Software event 0.
ELC_SOFTWARE_EVENT_1	Software event 1.

4.3.14 Ethernet Interface

Interfaces

Detailed Description

Interface for Ethernet functions.

Summary

The Ethernet interface provides Ethernet functionality. The Ethernet interface supports the following features:

- Transmit/receive processing (Blocking and Non-Blocking)
- Callback function with returned event code
- Magic packet detection mode support
- Auto negotiation support
- Flow control support
- Multicast filtering support

Implemented by:

- Ethernet (r_ether)

Data Structures

struct	ether_instance_descriptor_t
struct	ether_callback_args_t
struct	ether_cfg_t
struct	ether_api_t
struct	ether_instance_t

Typedefs

typedef void	ether_ctrl_t

Enumerations

enum	ether_wake_on_lan_t
enum	ether_flow_control_t
enum	ether_multicast_t
enum	ether_promiscuous_t
enum	ether_zerocopy_t
enum	ether_event_t

Data Structure Documentation

◆ ether_instance_descriptor_t

struct ether_instance_descriptor_t
EDMAC descriptor as defined in the hardware manual. Structure must be packed at 1 byte.

◆ ether_callback_args_t

struct ether_callback_args_t		
Callback function parameter data		
Data Fields		
uint32_t	channel	Device channel number.
ether_event_t	event	Event code.
uint32_t	status_ecsr	ETHERC status register for interrupt handler.
uint32_t	status_eesr	ETHERC/EDMAC status register for interrupt handler.
void const *	p_context	Placeholder for user data. Set in ether_api_t::open function in ether_cfg_t.

◆ ether_cfg_t

struct ether_cfg_t	
Configuration parameters.	
Data Fields	
uint8_t	channel
	Channel.
ether_zerocopy_t	zerocopy
	Zero copy enable or disable in Read/Write function.
ether_multicast_t	multicast
	Multicast enable or disable.
ether_promiscuous_t	promiscuous
	Promiscuous mode enable or disable.
ether_flow_control_t	flow_control
	Flow control functionally enable or disable.
ether_padding_t	padding

	Padding length inserted into the received Ethernet frame.
uint32_t	padding_offset
	Offset of the padding inserted into the received Ethernet frame.
uint32_t	broadcast_filter
	Limit of the number of broadcast frames received continuously.
uint8_t *	p_mac_address
	Pointer of MAC address.
ether_instance_descriptor_t *	p_rx_descriptors
	Receive descriptor buffer pool.
ether_instance_descriptor_t *	p_tx_descriptors
	Transmit descriptor buffer pool.
uint8_t	num_tx_descriptors
	Number of transmission descriptor.
uint8_t	num_rx_descriptors
	Number of receive descriptor.
uint8_t **	pp_ether_buffers
	Transmit and receive buffer.
uint32_t	ether_buffer_size

	Size of transmit and receive buffer.
IRQn_Type	irq
	NVIC interrupt number.
uint32_t	interrupt_priority
	NVIC interrupt priority.
void(*	p_callback)(ether_callback_args_t *p_args)
	Callback provided when an ISR occurs.
ether_phy_instance_t const *	p_ether_phy_instance
	Pointer to ETHER_PHY instance.
void const *	p_context
	Placeholder for user data. More...
void const *	p_extend
	Placeholder for user extension.

Field Documentation

◆ p_context

void const* ether_cfg_t::p_context

Placeholder for user data.

Placeholder for user data. Passed to the user callback in ether_callback_args_t.

◆ ether_api_t

struct ether_api_t

Functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(ether_ctrl_t *const p_api_ctrl, ether_cfg_t const *const p_cfg)
fsp_err_t(*	close)(ether_ctrl_t *const p_api_ctrl)
fsp_err_t(*	read)(ether_ctrl_t *const p_api_ctrl, void *const p_buffer, uint32_t *const length_bytes)
fsp_err_t(*	bufferRelease)(ether_ctrl_t *const p_api_ctrl)
fsp_err_t(*	rxBufferUpdate)(ether_ctrl_t *const p_api_ctrl, void *const p_buffer)
fsp_err_t(*	write)(ether_ctrl_t *const p_api_ctrl, void *const p_buffer, uint32_t const frame_length)
fsp_err_t(*	linkProcess)(ether_ctrl_t *const p_api_ctrl)
fsp_err_t(*	wakeOnLANEnable)(ether_ctrl_t *const p_api_ctrl)
fsp_err_t(*	txStatusGet)(ether_ctrl_t *const p_api_ctrl, void *const p_buffer_address)

Field Documentation

◆ open

fsp_err_t(* ether_api_t::open) (ether_ctrl_t *const p_api_ctrl, ether_cfg_t const *const p_cfg)

Open driver.

Implemented as

- R_ETHER_Open()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ close

fsp_err_t(* ether_api_t::close) (ether_ctrl_t *const p_api_ctrl)

Close driver.

Implemented as

- R_ETHER_Close()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ read

fsp_err_t(* ether_api_t::read) (ether_ctrl_t *const p_api_ctrl, void *const p_buffer, uint32_t *const length_bytes)

Read packet if data is available.

Implemented as

- R_ETHER_Read()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_buffer	Pointer to where to store read data.
[in]	length_bytes	Number of bytes in buffer

◆ bufferRelease

fsp_err_t(* ether_api_t::bufferRelease) (ether_ctrl_t *const p_api_ctrl)

Release rx buffer from buffer pool process in zero-copy read operation.

Implemented as

- R_ETHER_BufferRelease()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ rxBufferUpdate

fsp_err_t(* ether_api_t::rxBufferUpdate) (ether_ctrl_t *const p_api_ctrl, void *const p_buffer)

Update the buffer pointer in the current receive descriptor.

Implemented as

- R_ETHER_RxBufferUpdate()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_buffer	New address to write into the rx buffer descriptor.

◆ write

fsp_err_t(* ether_api_t::write) (ether_ctrl_t *const p_api_ctrl, void *const p_buffer, uint32_t const frame_length)

Write packet.

Implemented as

- R_ETHER_Write()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_buffer	Pointer to data to write.
[in]	frame_length	Send ethernet frame size (without 4 bytes of CRC data size).

◆ linkProcess

fsp_err_t(* ether_api_t::linkProcess) (ether_ctrl_t *const p_api_ctrl)

Process link.

Implemented as

- R_ETHER_LinkProcess()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ wakeOnLANEnable

fsp_err_t(* ether_api_t::wakeOnLANEnable) (ether_ctrl_t *const p_api_ctrl)

Enable magic packet detection.

Implemented as

- R_ETHER_WakeOnLANEnable()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ txStatusGet

fsp_err_t(* ether_api_t::txStatusGet) (ether_ctrl_t *const p_api_ctrl, void *const p_buffer_address)

Get the address of the most recently sent buffer.

Implemented as

- R_ETHER_TxStatusGet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[out]	p_buffer_address	Pointer to the address of the most recently sent buffer.

◆ ether_instance_t

struct ether_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
ether_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
ether_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ether_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ ether_ctrl_t

typedef void ether_ctrl_t
Control block. Allocate an instance specific control block to pass into the API calls. **Implemented as** ◦ ether_instance_ctrl_t

Enumeration Type Documentation

◆ ether_wake_on_lan_t

enum ether_wake_on_lan_t

Wake on LAN

Enumerator	
ETHER_WAKE_ON_LAN_DISABLE	Disable Wake on LAN.
ETHER_WAKE_ON_LAN_ENABLE	Enable Wake on LAN.

◆ ether_flow_control_t

enum ether_flow_control_t

Flow control functionality

Enumerator	
ETHER_FLOW_CONTROL_DISABLE	Disable flow control functionality.
ETHER_FLOW_CONTROL_ENABLE	Enable flow control functionality with pause frames.

◆ ether_multicast_t

enum ether_multicast_t

Multicast Filter

Enumerator	
ETHER_MULTICAST_DISABLE	Disable reception of multicast frames.
ETHER_MULTICAST_ENABLE	Enable reception of multicast frames.

◆ ether_promiscuous_t

enum ether_promiscuous_t	
Promiscuous Mode	
Enumerator	
ETHER_PROMISCUOUS_DISABLE	Only receive packets with current MAC address, multicast, and broadcast.
ETHER_PROMISCUOUS_ENABLE	Receive all packets.

◆ ether_zerocopy_t

enum ether_zerocopy_t	
Zero copy	
Enumerator	
ETHER_ZEROCOPY_DISABLE	Disable zero copy in Read/Write function.
ETHER_ZEROCOPY_ENABLE	Enable zero copy in Read/Write function.

◆ ether_event_t

enum ether_event_t	
Event code of callback function	
Enumerator	
ETHER_EVENT_WAKEON_LAN	Magic packet detection event.
ETHER_EVENT_LINK_ON	Link up detection event.
ETHER_EVENT_LINK_OFF	Link down detection event.
ETHER_EVENT_INTERRUPT	Interrupt event.

4.3.15 Ethernet PHY Interface

Interfaces

Detailed Description

Interface for Ethernet PHY functions.

Summary

The Ethernet PHY module (r_ether_phy) provides an API for standard Ethernet PHY communications applications that use the ETHERC peripheral.

The Ethernet PHY interface supports the following features:

- Auto negotiation support
- Flow control support
- Link status check support

Implemented by:

- Ethernet PHY (r_ether_phy)

Data Structures

struct	ether_phy_cfg_t
struct	ether_phy_api_t
struct	ether_phy_instance_t

Typedefs

typedef void	ether_phy_ctrl_t

Enumerations

enum	ether_phy_flow_control_t
enum	ether_phy_link_speed_t
enum	ether_phy_mii_type_t

Data Structure Documentation

◆ ether_phy_cfg_t

struct ether_phy_cfg_t		
Configuration parameters.		
Data Fields		
uint8_t	channel	Channel.
uint8_t	phy_lsi_address	Address of PHY-LSI.
uint32_t	phy_reset_wait_time	Wait time for PHY-LSI reboot.
int32_t	mii_bit_access_wait_time	Wait time for MII/RMII access.

ether_phy_flow_control_t	flow_control	Flow control functionally enable or disable.
ether_phy_mii_type_t	mii_type	Interface type is MII or RMII.
void const *	p_context	Placeholder for user data. Passed to the user callback in ether_phy_callback_args_t.
void const *	p_extend	Placeholder for user extension.

◆ ether_phy_api_t

struct ether_phy_api_t

Functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(ether_phy_ctrl_t *const p_api_ctrl, ether_phy_cfg_t const *const p_cfg)
fsp_err_t(*	close)(ether_phy_ctrl_t *const p_api_ctrl)
fsp_err_t(*	startAutoNegotiate)(ether_phy_ctrl_t *const p_api_ctrl)
fsp_err_t(*	linkPartnerAbilityGet)(ether_phy_ctrl_t *const p_api_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause)
fsp_err_t(*	linkStatusGet)(ether_phy_ctrl_t *const p_api_ctrl)

Field Documentation

◆ open

fsp_err_t(* ether_phy_api_t::open) (ether_phy_ctrl_t *const p_api_ctrl, ether_phy_cfg_t const *const p_cfg)

Open driver.

Implemented as

- R_ETHER_PHY_Open()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ close

fsp_err_t(* ether_phy_api_t::close) (ether_phy_ctrl_t *const p_api_ctrl)

Close driver.

Implemented as

- R_ETHER_PHY_Close()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ startAutoNegotiate

fsp_err_t(* ether_phy_api_t::startAutoNegotiate) (ether_phy_ctrl_t *const p_api_ctrl)

Start auto negotiation.

Implemented as

- R_ETHER_PHY_StartAutoNegotiate()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ linkPartnerAbilityGet

fsp_err_t(* ether_phy_api_t::linkPartnerAbilityGet) (ether_phy_ctrl_t *const p_api_ctrl, uint32_t *const p_line_speed_duplex, uint32_t *const p_local_pause, uint32_t *const p_partner_pause)

Get the partner ability.

Implemented as

- R_ETHER_PHY_LinkPartnerAbilityGet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[out]	p_line_speed_duplex	Pointer to the location of both the line speed and the duplex.
[out]	p_local_pause	Pointer to the location to store the local pause bits.
[out]	p_partner_pause	Pointer to the location to store the partner pause bits.

◆ linkStatusGet

fsp_err_t(* ether_phy_api_t::linkStatusGet) (ether_phy_ctrl_t *const p_api_ctrl)

Get Link status from PHY-LSI interface.

Implemented as

- R_ETHER_PHY_LinkStatusGet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ ether_phy_instance_t

struct ether_phy_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
ether_phy_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
ether_phy_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ether_phy_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ ether_phy_ctrl_t

typedef void ether_phy_ctrl_t

Control block. Allocate an instance specific control block to pass into the API calls.

Implemented as

- ether_phy_instance_ctrl_t

Enumeration Type Documentation

◆ ether_phy_flow_control_t

enum ether_phy_flow_control_t	
Flow control functionality	
Enumerator	
ETHER_PHY_FLOW_CONTROL_DISABLE	Disable flow control functionality.
ETHER_PHY_FLOW_CONTROL_ENABLE	Enable flow control functionality with pause frames.

◆ ether_phy_link_speed_t

enum ether_phy_link_speed_t	
Link speed	
Enumerator	
ETHER_PHY_LINK_SPEED_NO_LINK	Link is not established.
ETHER_PHY_LINK_SPEED_10H	Link status is 10Mbit/s and half duplex.
ETHER_PHY_LINK_SPEED_10F	Link status is 10Mbit/s and full duplex.
ETHER_PHY_LINK_SPEED_100H	Link status is 100Mbit/s and half duplex.
ETHER_PHY_LINK_SPEED_100F	Link status is 100Mbit/s and full duplex.

◆ ether_phy_mii_type_t

enum ether_phy_mii_type_t	
Media-independent interface	
Enumerator	
ETHER_PHY_MII_TYPE_MII	MII.
ETHER_PHY_MII_TYPE_RMII	RMII.

4.3.16 External IRQ Interface

Interfaces

Detailed Description

Interface for detecting external interrupts.

Summary

The External IRQ Interface is for configuring interrupts to fire when a trigger condition is detected on an external IRQ pin.

The External IRQ Interface can be implemented by:

- Interrupt Controller Unit (r_icu)

Data Structures

struct	external_irq_callback_args_t
struct	external_irq_cfg_t
struct	external_irq_api_t
struct	external_irq_instance_t

Typedefs

typedef void	external_irq_ctrl_t

Enumerations

enum	external_irq_trigger_t
enum	external_irq_pclk_div_t

Data Structure Documentation

◆ external_irq_callback_args_t

struct external_irq_callback_args_t		
Callback function parameter data		
Data Fields		
void const *	p_context	Placeholder for user data. Set in external_irq_api_t::open function in external_irq_cfg_t.
uint32_t	channel	The physical hardware channel that caused the interrupt.

◆ external_irq_cfg_t

struct external_irq_cfg_t
User configuration structure, used in open function

Data Fields	
uint8_t	channel
	Hardware channel used.
uint8_t	ipl
	Interrupt priority.
IRQn_Type	irq
	NVIC interrupt number assigned to this instance.
external_irq_trigger_t	trigger
	Trigger setting.
external_irq_pclk_div_t	pclk_div
	Digital filter clock divisor setting.
bool	filter_enable
	Digital filter enable/disable setting.
void(*	p_callback)(external_irq_callback_args_t *p_args)
void const *	p_context
void const *	p_extend
	External IRQ hardware dependent configuration.

Field Documentation

◆ **p_callback**

void(* external_irq_cfg_t::p_callback) (external_irq_callback_args_t *p_args)
Callback provided external input trigger occurs.

◆ **p_context**

void const* external_irq_cfg_t::p_context
Placeholder for user data. Passed to the user callback in external_irq_callback_args_t.

◆ **external_irq_api_t**

struct external_irq_api_t
External interrupt driver structure. External interrupt functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)
fsp_err_t(*	enable)(external_irq_ctrl_t *const p_ctrl)
fsp_err_t(*	disable)(external_irq_ctrl_t *const p_ctrl)
fsp_err_t(*	callbackSet)(external_irq_ctrl_t *const p_api_ctrl, void(*p_callback)(external_irq_callback_args_t *), void const *const p_context, external_irq_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(external_irq_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* external_irq_api_t::open) (external_irq_ctrl_t *const p_ctrl, external_irq_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_ICU_ExternalIrqOpen()

Parameters

[out]	p_ctrl	Pointer to control block. Must be declared by user. Value set here.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ enable

fsp_err_t(* external_irq_api_t::enable) (external_irq_ctrl_t *const p_ctrl)

Enable callback when an external trigger condition occurs.

Implemented as

- R_ICU_ExternalIrqEnable()

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ disable

fsp_err_t(* external_irq_api_t::disable) (external_irq_ctrl_t *const p_ctrl)

Disable callback when external trigger condition occurs.

Implemented as

- R_ICU_ExternalIrqDisable()

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ callbackSet

fsp_err_t(* external_irq_api_t::callbackSet) (external_irq_ctrl_t *const p_api_ctrl, void(*p_callback)(external_irq_callback_args_t *), void const *const p_context, external_irq_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_ICU_ExternalIrqCallbackSet()

Parameters

[in]	p_ctrl	Pointer to the Extneral IRQ control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* external_irq_api_t::close) (external_irq_ctrl_t *const p_ctrl)

Allow driver to be reconfigured. May reduce power consumption.

Implemented as

- R_ICU_ExternalIrqClose()

Parameters

[in]	p_ctrl	Control block set in Open call for this external interrupt.

◆ external_irq_instance_t

struct external_irq_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
external_irq_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
external_irq_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.

external_irq_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ external_irq_ctrl_t

typedef void external_irq_ctrl_t

External IRQ control block. Allocate an instance specific control block to pass into the external IRQ API calls.

Implemented as

- icu_instance_ctrl_t

Enumeration Type Documentation

◆ external_irq_trigger_t

enum external_irq_trigger_t

Condition that will trigger an interrupt when detected.

Enumerator	
EXTERNAL_IRQ_TRIG_FALLING	Falling edge trigger.
EXTERNAL_IRQ_TRIG_RISING	Rising edge trigger.
EXTERNAL_IRQ_TRIG_BOTH_EDGE	Both edges trigger.
EXTERNAL_IRQ_TRIG_LEVEL_LOW	Low level trigger.

◆ external_irq_pclk_div_t

enum external_irq_pclk_div_t

External IRQ input pin digital filtering sample clock divisor settings. The digital filter rejects trigger conditions that are shorter than 3 periods of the filter clock.

Enumerator	
EXTERNAL_IRQ_PCLK_DIV_BY_1	Filter using PCLK divided by 1.
EXTERNAL_IRQ_PCLK_DIV_BY_8	Filter using PCLK divided by 8.
EXTERNAL_IRQ_PCLK_DIV_BY_32	Filter using PCLK divided by 32.
EXTERNAL_IRQ_PCLK_DIV_BY_64	Filter using PCLK divided by 64.

4.3.17 Flash Interface

Interfaces

Detailed Description

Interface for the Flash Memory.

Summary

The Flash interface provides the ability to read, write, erase, and blank check the code flash and data flash regions.

The Flash interface is implemented by:

- Low-Power Flash Driver (r_flash_lp)

Data Structures

struct	flash_block_info_t
struct	flash_regions_t
struct	flash_info_t
struct	flash_callback_args_t
struct	flash_cfg_t
struct	flash_api_t
struct	flash_instance_t

Typedefs

typedef void	flash_ctrl_t

Enumerations

enum	flash_result_t
enum	flash_startup_area_swap_t
enum	flash_event_t
enum	flash_id_code_mode_t
enum	flash_status_t

Data Structure Documentation

◆ flash_block_info_t

struct flash_block_info_t		
Flash block details stored in factory flash.		
Data Fields		
uint32_t	block_section_st_addr	Starting address for this block section (blocks of this size)
uint32_t	block_section_end_addr	Ending address for this block section (blocks of this size)
uint32_t	block_size	Flash erase block size.
uint32_t	block_size_write	Flash write block size.

◆ flash_regions_t

struct flash_regions_t		
Flash block details		
Data Fields		
uint32_t	num_regions	Length of block info array.
flash_block_info_t const *	p_block_array	Block info array base address.

◆ flash_info_t

struct flash_info_t		
Information about the flash blocks		
Data Fields		
flash_regions_t	code_flash	Information about the code flash regions.
flash_regions_t	data_flash	Information about the code flash regions.

◆ flash_callback_args_t

struct flash_callback_args_t		
Callback function parameter data		
Data Fields		
flash_event_t	event	Event can be used to identify what caused the callback (flash ready or error).
void const *	p_context	Placeholder for user data. Set in flash_api_t::open function in::flash_cfg_t.

◆ flash_cfg_t

struct flash_cfg_t	
FLASH Configuration	
Data Fields	
bool	data_flash_bgo
	True if BGO (Background Operation) is enabled for Data Flash.
void(*	p_callback)(flash_callback_args_t *p_args)
	Callback provided when a Flash interrupt ISR occurs.
void const *	p_extend
	FLASH hardware dependent configuration.
void const *	p_context
	Placeholder for user data. Passed to user callback in flash_callback_args_t.
uint8_t	ipl
	Flash ready interrupt priority.
IRQn_Type	irq
	Flash ready interrupt number.
uint8_t	err_ipl
	Flash error interrupt priority (unused in r_flash_lp)
IRQn_Type	err_irq
	Flash error interrupt number (unused in r_flash_lp)

◆ flash_api_t

struct flash_api_t	
Shared Interface definition for FLASH	
Data Fields	
fsp_err_t(*	open)(flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)
fsp_err_t(*	write)(flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)
fsp_err_t(*	erase)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)
fsp_err_t(*	blankCheck)(flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)
fsp_err_t(*	infoGet)(flash_ctrl_t *const p_ctrl, flash_info_t *const p_info)
fsp_err_t(*	close)(flash_ctrl_t *const p_ctrl)
fsp_err_t(*	statusGet)(flash_ctrl_t *const p_ctrl, flash_status_t *const p_status)
fsp_err_t(*	accessWindowSet)(flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)
fsp_err_t(*	accessWindowClear)(flash_ctrl_t *const p_ctrl)
fsp_err_t(*	idCodeSet)(flash_ctrl_t *const p_ctrl, uint8_t const *const p_id_bytes, flash_id_code_mode_t mode)
fsp_err_t(*	reset)(flash_ctrl_t *const p_ctrl)
fsp_err_t(*	updateFlashClockFreq)(flash_ctrl_t *const p_ctrl)
fsp_err_t(*	startupAreaSelect)(flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)

fsp_err_t(*	bankSwap)(flash_ctrl_t *const p_ctrl)
fsp_err_t(*	callbackSet)(flash_ctrl_t *const p_api_ctrl, void(*p_callback)(flash_callback_args_t *), void const *const p_context, flash_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* flash_api_t::open) (flash_ctrl_t *const p_ctrl, flash_cfg_t const *const p_cfg)

Open FLASH device.

Implemented as

- R_FLASH_LP_Open()
- R_FLASH_HP_Open()

Parameters

[out]	p_ctrl	Pointer to FLASH device control. Must be declared by user. Value set here.
[in]	flash_cfg_t	Pointer to FLASH configuration structure. All elements of this structure must be set by the user.

◆ write

fsp_err_t(* flash_api_t::write) (flash_ctrl_t *const p_ctrl, uint32_t const src_address, uint32_t const flash_address, uint32_t const num_bytes)

Write FLASH device.

Implemented as

- R_FLASH_LP_Write()
- R_FLASH_HP_Write()

Parameters

[in]	p_ctrl	Control for the FLASH device context.
[in]	src_address	Address of the buffer containing the data to write to Flash.
[in]	flash_address	Code Flash or Data Flash address to write. The address must be on a programming line boundary.
[in]	num_bytes	The number of bytes to write. This number must be a multiple of the programming size. For Code Flash this is FLASH_MIN_PGM_SIZE_CF. For Data Flash this is FLASH_MIN_PGM_SIZE_DF.

Warning
Specifying a number that is not a multiple of the programming size will result in SF_FLASH_ERR_BYTES being returned and no data written.

◆ erase

fsp_err_t(* flash_api_t::erase) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_blocks)

Erase FLASH device.

Implemented as

- R_FLASH_LP_Erase()
- R_FLASH_HP_Erase()

Parameters

[in]	p_ctrl	Control for the FLASH device.
[in]	address	The block containing this address is the first block erased.
[in]	num_blocks	Specifies the number of blocks to be erased, the starting block determined by the block_erase_address.

◆ blankCheck

fsp_err_t(* flash_api_t::blankCheck) (flash_ctrl_t *const p_ctrl, uint32_t const address, uint32_t const num_bytes, flash_result_t *const p_blank_check_result)

Blank check FLASH device.

Implemented as

- R_FLASH_LP_BlankCheck()
- R_FLASH_HP_BlankCheck()

Parameters

[in]	p_ctrl	Control for the FLASH device context.
[in]	address	The starting address of the Flash area to blank check.
[in]	num_bytes	Specifies the number of bytes that need to be checked. See the specific handler for details.
[out]	p_blank_check_result	Pointer that will be populated by the API with the results of the blank check operation in non-BGO (blocking) mode. In this case the blank check operation completes here and the result is returned. In Data Flash BGO mode the blank check operation is only started here and the result obtained later when the supplied callback routine is called. In this case FLASH_RESULT_BGO_ACTIVE will be returned in p_blank_check_result.

◆ infoGet

fsp_err_t(* flash_api_t::infoGet) (flash_ctrl_t *const p_ctrl, flash_info_t *const p_info)

Close FLASH device.

Implemented as

- R_FLASH_LP_InfoGet()
- R_FLASH_HP_InfoGet()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[out]	p_info	Pointer to FLASH info structure.

◆ close

fsp_err_t(* flash_api_t::close) (flash_ctrl_t *const p_ctrl)

Close FLASH device.

Implemented as

- R_FLASH_LP_Close()
- R_FLASH_HP_Close()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ statusGet

fsp_err_t(* flash_api_t::statusGet) (flash_ctrl_t *const p_ctrl, flash_status_t *const p_status)

Get Status for FLASH device.

Implemented as

- R_FLASH_LP_StatusGet()
- R_FLASH_HP_StatusGet()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[out]	p_ctrl	Pointer to the current flash status.

◆ accessWindowSet

fsp_err_t(* flash_api_t::accessWindowSet) (flash_ctrl_t *const p_ctrl, uint32_t const start_addr, uint32_t const end_addr)

Set Access Window for FLASH device.

Implemented as

- R_FLASH_LP_AccessWindowSet()
- R_FLASH_HP_AccessWindowSet()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	start_addr	Determines the Starting block for the Code Flash access window.
[in]	end_addr	Determines the Ending block for the Code Flash access window. This address will not be within the access window.

◆ accessWindowClear

fsp_err_t(* flash_api_t::accessWindowClear) (flash_ctrl_t *const p_ctrl)

Clear any existing Code Flash access window for FLASH device.

Implemented as

- R_FLASH_LP_AccessWindowClear()
- R_FLASH_HP_AccessWindowClear()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	start_addr	Determines the Starting block for the Code Flash access window.
[in]	end_addr	Determines the Ending block for the Code Flash access window.

◆ idCodeSet

fsp_err_t(* flash_api_t::idCodeSet) (flash_ctrl_t *const p_ctrl, uint8_t const *const p_id_bytes, flash_id_code_mode_t mode)

Set ID Code for FLASH device. Setting the ID code can restrict access to the device. The ID code will be required to connect to the device. Bits 126 and 127 are set based on the mode.

For example, uint8_t id_bytes[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0x00}; with mode FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT will result in an ID code of 00112233445566778899aabbccddeec0

With mode FLASH_ID_CODE_MODE_LOCKED, it will result in an ID code of 00112233445566778899aabbccddee80

Implemented as

- R_FLASH_LP_IdCodeSet()
- R_FLASH_HP_IdCodeSet()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	p_id_bytes	Ponter to the ID Code to be written.
[in]	mode	Mode used for checking the ID code.

◆ reset

fsp_err_t(* flash_api_t::reset) (flash_ctrl_t *const p_ctrl)

Reset function for FLASH device.

Implemented as

- R_FLASH_LP_Reset()
- R_FLASH_HP_Reset()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ updateFlashClockFreq

fsp_err_t(* flash_api_t::updateFlashClockFreq) (flash_ctrl_t *const p_ctrl)

Update Flash clock frequency (FCLK) and recalculate timeout values

Implemented as

- R_FLASH_LP_UpdateFlashClockFreq()
- R_FLASH_HP_UpdateFlashClockFreq()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ startupAreaSelect

fsp_err_t(* flash_api_t::startupAreaSelect) (flash_ctrl_t *const p_ctrl, flash_startup_area_swap_t swap_type, bool is_temporary)

Select which block - Default (Block 0) or Alternate (Block 1) is used as the start-up area block.

Implemented as

- R_FLASH_LP_StartUpAreaSelect()
- R_FLASH_HP_StartUpAreaSelect()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.
[in]	swap_type	FLASH_STARTUP_AREA_BLOCK0, FLASH_STARTUP_AREA_BLOCK1 or FLASH_STARTUP_AREA_BTFLG.
[in]	is_temporary	True or false. See table below.

swap_type	is_temporary	Operation
FLASH_STARTUP_AREA_BLOCK0	false	On next reset Startup area will be Block 0.
FLASH_STARTUP_AREA_BLOCK1	true	Startup area is immediately, but temporarily switched to Block 1.
FLASH_STARTUP_AREA_BTFLG	true	Startup area is immediately, but temporarily switched to the Block determined by the Configuration BTFLG.

◆ bankSwap

fsp_err_t(* flash_api_t::bankSwap) (flash_ctrl_t *const p_ctrl)

Swap the bank used as the startup area. Only valid in dual bank mode.

Implemented as

- R_FLASH_HP_BankSwap()

Parameters

[in]	p_ctrl	Pointer to FLASH device control.

◆ callbackSet

fsp_err_t(* flash_api_t::callbackSet) (flash_ctrl_t *const p_api_ctrl, void(*p_callback)(flash_callback_args_t *), void const *const p_context, flash_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_FLASH_HP_CallbackSet()

Parameters

[in]	p_ctrl	Control block set in flash_api_t::open call for this timer.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ flash_instance_t

struct flash_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
flash_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
flash_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.

flash_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ flash_ctrl_t

typedef void flash_ctrl_t
Flash control block. Allocate an instance specific control block to pass into the flash API calls. **Implemented as** ◦ flash_lp_instance_ctrl_t ◦ flash_hp_instance_ctrl_t

Enumeration Type Documentation

◆ flash_result_t

enum flash_result_t

Result type for certain operations

Enumerator	
FLASH_RESULT_BLANK	Return status for Blank Check Function.
FLASH_RESULT_NOT_BLANK	Return status for Blank Check Function.
FLASH_RESULT_BGO_ACTIVE	Flash is configured for BGO mode. Result is returned in callback.

◆ flash_startup_area_swap_t

enum flash_startup_area_swap_t

Parameter for specifying the startup area swap being requested by startupAreaSelect()

Enumerator	
FLASH_STARTUP_AREA_BTFLG	Startup area will be set based on the value of the BTFLG.
FLASH_STARTUP_AREA_BLOCK0	Startup area will be set to Block 0.
FLASH_STARTUP_AREA_BLOCK1	Startup area will be set to Block 1.

◆ flash_event_t

enum flash_event_t	
Event types returned by the ISR callback when used in Data Flash BGO mode	
Enumerator	
FLASH_EVENT_ERASE_COMPLETE	Erase operation successfully completed.
FLASH_EVENT_WRITE_COMPLETE	Write operation successfully completed.
FLASH_EVENT_BLANK	Blank check operation successfully completed. Specified area is blank.
FLASH_EVENT_NOT_BLANK	Blank check operation successfully completed. Specified area is NOT blank.
FLASH_EVENT_ERR_DF_ACCESS	Data Flash operation failed. Can occur when writing an unerased section.
FLASH_EVENT_ERR_CF_ACCESS	Code Flash operation failed. Can occur when writing an unerased section.
FLASH_EVENT_ERR_CMD_LOCKED	Operation failed, FCU is in Locked state (often result of an illegal command)
FLASH_EVENT_ERR_FAILURE	Erase or Program Operation failed.
FLASH_EVENT_ERR_ONE_BIT	A 1-bit error has been corrected when reading the flash memory area by the sequencer.

◆ flash_id_code_mode_t

enum flash_id_code_mode_t	
ID Code Modes for writing to ID code registers	
Enumerator	
FLASH_ID_CODE_MODE_UNLOCKED	ID code is ignored.
FLASH_ID_CODE_MODE_LOCKED_WITH_ALL_ERASE_SUPPORT	ID code is checked. All erase is available.
FLASH_ID_CODE_MODE_LOCKED	ID code is checked.

◆ flash_status_t

enum flash_status_t	
Flash status	
Enumerator	
FLASH_STATUS_IDLE	The flash is idle.
FLASH_STATUS_BUSY	The flash is currently processing a command.

4.3.18 I2C Master Interface

Interfaces

Detailed Description

Interface for I2C master communication.

Summary

The I2C master interface provides a common API for I2C HAL drivers. The I2C master interface supports:

- Interrupt driven transmit/receive processing
- Callback function support which can return an event code

Implemented by:

- I2C Master on IIC (r_iic_master)
- Serial Communications Interface (SCI) I2C (r_sci_i2c)
- I2C Master on IIC/I3C (r_iic_b_master)
- Serial Communications Interface (SCI) I2C (r_sci_b_i2c)

Data Structures

struct	i2c_master_callback_args_t
struct	i2c_master_status_t
struct	i2c_master_cfg_t
struct	i2c_master_api_t
struct	i2c_master_instance_t

Typedefs

typedef void	i2c_master_ctrl_t

Enumerations

enum	i2c_master_rate_t
enum	i2c_master_addr_mode_t
enum	i2c_master_event_t

Data Structure Documentation

◆ i2c_master_callback_args_t

struct i2c_master_callback_args_t

I2C callback parameter definition

Data Fields		
void const *	p_context	Pointer to user-provided context.
i2c_master_event_t	event	Event code.

◆ i2c_master_status_t

struct i2c_master_status_t

I2C status indicators

Data Fields		
bool	open	True if driver is open.

◆ i2c_master_cfg_t

struct i2c_master_cfg_t

I2C configuration block

Data Fields

uint8_t	channel
	Identifier recognizable by implementation. More...
i2c_master_rate_t	rate
	Device's maximum clock rate from enum i2c_rate_t.
uint32_t	slave

	The address of the slave device.
i2c_master_addr_mode_t	addr_mode
	Indicates how slave fields should be interpreted.
uint8_t	ipl
	Interrupt priority level. Same for RXI, TXI, TEI and ERI.
IRQn_Type	rxi_irq
	Receive IRQ number.
IRQn_Type	txi_irq
	Transmit IRQ number.
IRQn_Type	tei_irq
	Transmit end IRQ number.
IRQn_Type	eri_irq
	Error IRQ number.
transfer_instance_t const *	p_transfer_tx
	DTC instance for I2C transmit.Set to NULL if unused. More...
transfer_instance_t const *	p_transfer_rx
	DTC instance for I2C receive. Set to NULL if unused.
void(*	p_callback)(i2c_master_callback_args_t *p_args)
	Pointer to callback function. More...

void const *	p_context
	Pointer to the user-provided context.
void const *	p_extend
	Any configuration data needed by the hardware. More...

Field Documentation

◆ channel

uint8_t i2c_master_cfg_t::channel

Identifier recognizable by implementation.

Generic configuration

◆ p_transfer_tx

transfer_instance_t const* i2c_master_cfg_t::p_transfer_tx

DTC instance for I2C transmit.Set to NULL if unused.

DTC support

◆ p_callback

void(* i2c_master_cfg_t::p_callback) (i2c_master_callback_args_t *p_args)

Pointer to callback function.

Parameters to control software behavior

◆ p_extend

void const* i2c_master_cfg_t::p_extend

Any configuration data needed by the hardware.

Implementation-specific configuration

◆ i2c_master_api_t

struct i2c_master_api_t

Interface definition for I2C access as master

Data Fields

fsp_err_t(*	open)(i2c_master_ctrl_t *const p_ctrl, i2c_master_cfg_t const *const p_cfg)

fsp_err_t(*	read)(i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)
fsp_err_t(*	write)(i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)
fsp_err_t(*	abort)(i2c_master_ctrl_t *const p_ctrl)
fsp_err_t(*	slaveAddressSet)(i2c_master_ctrl_t *const p_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode)
fsp_err_t(*	callbackSet)(i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void const *const p_context, i2c_master_callback_args_t *const p_callback_memory)
fsp_err_t(*	statusGet)(i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status)
fsp_err_t(*	close)(i2c_master_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* i2c_master_api_t::open) (i2c_master_ctrl_t *const p_ctrl, i2c_master_cfg_t const *const p_cfg)

Opens the I2C Master driver and initializes the hardware.

Implemented as

- R_IIC_MASTER_Open()
- R_SCI_I2C_Open()
- R_IIC_B_MASTER_Open()
- R_SCI_B_I2C_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements are set here.
[in]	p_cfg	Pointer to configuration structure.

◆ read

fsp_err_t(* i2c_master_api_t::read) (i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes, bool const restart)

Performs a read operation on an I2C Master device.

Implemented as

- R_SCI_I2C_Read()
- R_IIC_MASTER_Read()
- R_IIC_B_MASTER_Read()
- R_SCI_B_I2C_Read()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.
[in]	p_dest	Pointer to the location to store read data.
[in]	bytes	Number of bytes to read.
[in]	restart	Specify if the restart condition should be issued after reading.

◆ write

fsp_err_t(* i2c_master_api_t::write) (i2c_master_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes, bool const restart)

Performs a write operation on an I2C Master device.

Implemented as

- R_IIC_MASTER_Write()
- R_SCI_I2C_Write()
- R_IIC_B_MASTER_Write()
- R_SCI_B_I2C_Write()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.
[in]	p_src	Pointer to the location to get write data from.
[in]	bytes	Number of bytes to write.
[in]	restart	Specify if the restart condition should be issued after writing.

◆ abort

fsp_err_t(* i2c_master_api_t::abort) (i2c_master_ctrl_t *const p_ctrl)

Performs a reset of the peripheral.

Implemented as

- R_IIC_MASTER_Abort()
- R_SCI_I2C_Abort()
- R_IIC_B_MASTER_Abort()
- R_SCI_B_I2C_Abort()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.

◆ slaveAddressSet

fsp_err_t(* i2c_master_api_t::slaveAddressSet) (i2c_master_ctrl_t *const p_ctrl, uint32_t const slave, i2c_master_addr_mode_t const addr_mode)

Sets address of the slave device without reconfiguring the bus.

Implemented as

- R_IIC_MASTER_SlaveAddressSet()
- R_SCI_I2C_SlaveAddressSet()
- R_IIC_B_MASTER_SlaveAddressSet()
- R_SCI_B_I2C_SlaveAddressSet()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.
[in]	slave_address	Address of the slave device.
[in]	address_mode	Addressing mode.

◆ callbackSet

fsp_err_t(* i2c_master_api_t::callbackSet) (i2c_master_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_master_callback_args_t *), void const *const p_context, i2c_master_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_IIC_MASTER_CallbackSet()
- R_SCI_I2C_CallbackSet()
- R_IIC_B_MASTER_CallbackSet()
- R_SCI_B_I2C_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the IIC Master control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ statusGet

fsp_err_t(* i2c_master_api_t::statusGet) (i2c_master_ctrl_t *const p_api_ctrl, i2c_master_status_t *p_status)

Gets the status of the configured I2C device.

Implemented as

- R_IIC_MASTER_StatusGet()
- R_SCI_I2C_StatusGet()
- R_IIC_B_MASTER_StatusGet()
- R_SCI_B_I2C_StatusGet()

Parameters

[in]	p_ctrl	Pointer to the IIC Master control block.
[out]	p_status	Pointer to store current status.

◆ close

fsp_err_t(* i2c_master_api_t::close) (i2c_master_ctrl_t *const p_ctrl)

Closes the driver and releases the I2C Master device.

Implemented as

- R_IIC_MASTER_Close()
- R_SCI_I2C_Close()
- R_IIC_B_MASTER_Close()
- R_SCI_B_I2C_Close()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_api_master_t::open call.

◆ i2c_master_instance_t

struct i2c_master_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
i2c_master_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
i2c_master_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
i2c_master_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ i2c_master_ctrl_t

typedef void i2c_master_ctrl_t

I2C control block. Allocate an instance specific control block to pass into the I2C API calls.

Implemented as

- iic_master_instance_ctrl_t

Enumeration Type Documentation

◆ i2c_master_rate_t

enum i2c_master_rate_t	
Communication speed options	
Enumerator	
I2C_MASTER_RATE_STANDARD	100 kHz
I2C_MASTER_RATE_FAST	400 kHz
I2C_MASTER_RATE_FASTPLUS	1 MHz

◆ i2c_master_addr_mode_t

enum i2c_master_addr_mode_t	
Addressing mode options	
Enumerator	
I2C_MASTER_ADDR_MODE_7BIT	Use 7-bit addressing mode.
I2C_MASTER_ADDR_MODE_10BIT	Use 10-bit addressing mode.

◆ i2c_master_event_t

enum i2c_master_event_t	
Callback events	
Enumerator	
I2C_MASTER_EVENT_ABORTED	A transfer was aborted.
I2C_MASTER_EVENT_RX_COMPLETE	A receive operation was completed successfully.
I2C_MASTER_EVENT_TX_COMPLETE	A transmit operation was completed successfully.

4.3.19 I2C Slave Interface

Interfaces

Detailed Description

Interface for I2C slave communication.

Summary

The I2C slave interface provides a common API for I2C HAL drivers. The I2C slave interface supports:

- Interrupt driven transmit/receive processing
- Callback function support which returns a event codes

Implemented by:

- I2C Slave on IIC (r_iic_slave)
- I2C Slave on IIC/I3C (r_iic_b_slave)

Data Structures

struct	i2c_slave_callback_args_t
struct	i2c_slave_cfg_t
struct	i2c_slave_api_t
struct	i2c_slave_instance_t

Typedefs

typedef void	i2c_slave_ctrl_t

Enumerations

enum	i2c_slave_rate_t
enum	i2c_slave_addr_mode_t
enum	i2c_slave_event_t

Data Structure Documentation

◆ i2c_slave_callback_args_t

struct i2c_slave_callback_args_t		
I2C callback parameter definition		
Data Fields		
void const *	p_context	Pointer to user-provided context.
uint32_t	bytes	Number of received/transmitted bytes in buffer.
i2c_slave_event_t	event	Event code.

◆ i2c_slave_cfg_t

struct i2c_slave_cfg_t	
I2C configuration block	
Data Fields	
uint8_t	channel
	Identifier recognizable by implementation. More...
i2c_slave_rate_t	rate
	Device's maximum clock rate from enum i2c_rate_t.
uint16_t	slave
	The address of the slave device.
i2c_slave_addr_mode_t	addr_mode
	Indicates how slave fields should be interpreted.
bool	general_call_enable
	Allow a General call from master.
IRQn_Type	rxi_irq
	Receive IRQ number.
IRQn_Type	txi_irq
	Transmit IRQ number.
IRQn_Type	tei_irq
	Transmit end IRQ number.
IRQn_Type	eri_irq

	Error IRQ number.
uint8_t	ipl
	Interrupt priority level for RXI, TXI and TER interrupts.
uint8_t	eri_ipl
	Interrupt priority level for ERI interrupt.
bool	clock_stretching_enable
	Low Hold SCL during reception for the period between the 9th and the 1st clock cycle.
void(*	p_callback)(i2c_slave_callback_args_t *p_args)
	Pointer to callback function. More...
void const *	p_context
	Pointer to the user-provided context.
void const *	p_extend
	Any configuration data needed by the hardware. More...

Field Documentation

◆ channel

uint8_t i2c_slave_cfg_t::channel

Identifier recognizable by implementation.

Generic configuration

◆ p_callback

void(* i2c_slave_cfg_t::p_callback) (i2c_slave_callback_args_t *p_args)
Pointer to callback function. Parameters to control software behavior

◆ p_extend

void const* i2c_slave_cfg_t::p_extend
Any configuration data needed by the hardware. Implementation-specific configuration

◆ i2c_slave_api_t

struct i2c_slave_api_t
Interface definition for I2C access as slave

Data Fields

fsp_err_t(*	open)(i2c_slave_ctrl_t *const p_ctrl, i2c_slave_cfg_t const *const p_cfg)
fsp_err_t(*	read)(i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t(*	write)(i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)
fsp_err_t(*	callbackSet)(i2c_slave_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_slave_callback_args_t *), void const *const p_context, i2c_slave_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(i2c_slave_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* i2c_slave_api_t::open) (i2c_slave_ctrl_t *const p_ctrl, i2c_slave_cfg_t const *const p_cfg)

Opens the I2C Slave driver and initializes the hardware.

Implemented as

- R_IIC_SLAVE_Open()
- R_IIC_B_SLAVE_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements are set here.
[in]	p_cfg	Pointer to configuration structure.

◆ read

fsp_err_t(* i2c_slave_api_t::read) (i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Performs a read operation on an I2C Slave device.

Implemented as

- R_IIC_SLAVE_Read()
- R_IIC_B_SLAVE_Read()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_slave_api_t::open call.
[in]	p_dest	Pointer to the location to store read data.
[in]	bytes	Number of bytes to read.

◆ write

fsp_err_t(* i2c_slave_api_t::write) (i2c_slave_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)

Performs a write operation on an I2C Slave device.

Implemented as

- R_IIC_SLAVE_Write()
- R_IIC_B_SLAVE_Write()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_slave_api_t::open call.
[in]	p_src	Pointer to the location to get write data from.
[in]	bytes	Number of bytes to write.

◆ callbackSet

fsp_err_t(* i2c_slave_api_t::callbackSet) (i2c_slave_ctrl_t *const p_api_ctrl, void(*p_callback)(i2c_slave_callback_args_t *), void const *const p_context, i2c_slave_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_IIC_SLAVE_CallbackSet()
- R_IIC_B_SLAVE_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the IIC Slave control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* i2c_slave_api_t::close) (i2c_slave_ctrl_t *const p_ctrl)

Closes the driver and releases the I2C Slave device.

Implemented as

- R_IIC_SLAVE_Close()
- R_IIC_B_SLAVE_Close()

Parameters

[in]	p_ctrl	Pointer to control block set in i2c_slave_api_t::open call.

◆ i2c_slave_instance_t

struct i2c_slave_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
i2c_slave_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
i2c_slave_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
i2c_slave_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ i2c_slave_ctrl_t

typedef void i2c_slave_ctrl_t

I2C control block. Allocate an instance specific control block to pass into the I2C API calls.

Implemented as

- iic_slave_instance_ctrl_t

Enumeration Type Documentation

◆ i2c_slave_rate_t

enum i2c_slave_rate_t	
Communication speed options	
Enumerator	
I2C_SLAVE_RATE_STANDARD	100 kHz
I2C_SLAVE_RATE_FAST	400 kHz
I2C_SLAVE_RATE_FASTPLUS	1 MHz

◆ i2c_slave_addr_mode_t

enum i2c_slave_addr_mode_t	
Addressing mode options	
Enumerator	
I2C_SLAVE_ADDR_MODE_7BIT	Use 7-bit addressing mode.
I2C_SLAVE_ADDR_MODE_10BIT	Use 10-bit addressing mode.

◆ i2c_slave_event_t

enum i2c_slave_event_t	
Callback events	
Enumerator	
I2C_SLAVE_EVENT_ABORTED	A transfer was aborted.
I2C_SLAVE_EVENT_RX_COMPLETE	A receive operation was completed successfully.
I2C_SLAVE_EVENT_TX_COMPLETE	A transmit operation was completed successfully.
I2C_SLAVE_EVENT_RX_REQUEST	A read operation expected from slave. Detected a write from master.
I2C_SLAVE_EVENT_TX_REQUEST	A write operation expected from slave. Detected a read from master.
I2C_SLAVE_EVENT_RX_MORE_REQUEST	A read operation expected from slave. Master sends out more data than configured to be read in slave.
I2C_SLAVE_EVENT_TX_MORE_REQUEST	A write operation expected from slave. Master requests more data than configured to be written by slave.
I2C_SLAVE_EVENT_GENERAL_CALL	General Call address received from Master. Detected a write from master.

4.3.20 I2S Interface

Interfaces

Detailed Description

Interface for I2S audio communication.

Summary

The I2S (Inter-IC Sound) interface provides APIs and definitions for I2S audio communication.

Known Implementations

Serial Sound Interface (r_ssi)

Data Structures

struct	i2s_callback_args_t
struct	i2s_status_t
struct	i2s_cfg_t
struct	i2s_api_t
struct	i2s_instance_t

Typedefs

typedef void	i2s_ctrl_t

Enumerations

enum	i2s_pcm_width_t
enum	i2s_word_length_t
enum	i2s_event_t
enum	i2s_mode_t
enum	i2s_mute_t
enum	i2s_ws_continue_t
enum	i2s_state_t

Data Structure Documentation

◆ i2s_callback_args_t

struct i2s_callback_args_t

Callback function parameter data

Data Fields		
void const *	p_context	Placeholder for user data. Set in i2s_api_t::open function in i2s_cfg_t.
i2s_event_t	event	The event can be used to identify what caused the callback (overflow or error).

◆ i2s_status_t

struct i2s_status_t		
I2S status.		
Data Fields		
i2s_state_t	state	Current I2S state.

◆ i2s_cfg_t

struct i2s_cfg_t	
User configuration structure, used in open function	
Data Fields	
uint32_t	channel
i2s_pcm_width_t	pcm_width
	Audio PCM data width.
i2s_word_length_t	word_length
	Audio word length, bits must be >= i2s_cfg_t::pcm_width bits.
i2s_ws_continue_t	ws_continue
	Whether to continue WS transmission during idle state.
i2s_mode_t	operating_mode
	Master or slave mode.
transfer_instance_t const *	p_transfer_tx
transfer_instance_t const *	p_transfer_rx
void(*	p_callback)(i2s_callback_args_t *p_args)
void const *	p_context
void const *	p_extend

	Extension parameter for hardware specific settings.
uint8_t	rxi_ipl
	Receive interrupt priority.
uint8_t	txi_ipl
	Transmit interrupt priority.
uint8_t	idle_err_ipl
	Idle/Error interrupt priority.
IRQn_Type	txi_irq
	Transmit IRQ number.
IRQn_Type	rxi_irq
	Receive IRQ number.
IRQn_Type	int_irq
	Idle/Error IRQ number.

Field Documentation

◆ channel

uint32_t i2s_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

◆ p_transfer_tx

transfer_instance_t const* i2s_cfg_t::p_transfer_tx

To use DTC during write, link a DTC instance here. Set to NULL if unused.

◆ p_transfer_rx

transfer_instance_t const* i2s_cfg_t::p_transfer_rx
To use DTC during read, link a DTC instance here. Set to NULL if unused.

◆ p_callback

void(* i2s_cfg_t::p_callback) (i2s_callback_args_t *p_args)
Callback provided when an I2S ISR occurs. Set to NULL for no CPU interrupt.

◆ p_context

void const* i2s_cfg_t::p_context
Placeholder for user data. Passed to the user callback in i2s_callback_args_t.

◆ i2s_api_t

struct i2s_api_t	
I2S functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)
fsp_err_t(*	stop)(i2s_ctrl_t *const p_ctrl)
fsp_err_t(*	mute)(i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)
fsp_err_t(*	write)(i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes)
fsp_err_t(*	read)(i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)
fsp_err_t(*	writeRead)(i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes)
fsp_err_t(*	statusGet)(i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status)
fsp_err_t(*	close)(i2s_ctrl_t *const p_ctrl)
fsp_err_t(*	callbackSet)(i2s_ctrl_t *const p_api_ctrl, void(*p_callback)(i2s_callback_args_t *), void const *const p_context,

	i2s_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* i2s_api_t::open) (i2s_ctrl_t *const p_ctrl, i2s_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_SSI_Open()

Precondition

Peripheral clocks and any required output pins should be configured prior to calling this function.

Note

To reconfigure after calling this function, call i2s_api_t::close first.

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ stop

fsp_err_t(* i2s_api_t::stop) (i2s_ctrl_t *const p_ctrl)

Stop communication. Communication is stopped when callback is called with I2S_EVENT_IDLE.

Implemented as

- R_SSI_Stop()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.

◆ mute

fsp_err_t(* i2s_api_t::mute) (i2s_ctrl_t *const p_ctrl, i2s_mute_t const mute_enable)

Enable or disable mute.

Implemented as

- R_SSI_Mute()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	mute_enable	Whether to enable or disable mute.

◆ write

fsp_err_t(* i2s_api_t::write) (i2s_ctrl_t *const p_ctrl, void const *const p_src, uint32_t const bytes)

Write I2S data. All transmit data is queued when callback is called with I2S_EVENT_TX_EMPTY. Transmission is complete when callback is called with I2S_EVENT_IDLE.

Implemented as

- R_SSI_Write()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	p_src	Buffer of PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, padding 0s will be added to transmission to make it a multiple of 8.

◆ read

fsp_err_t(* i2s_api_t::read) (i2s_ctrl_t *const p_ctrl, void *const p_dest, uint32_t const bytes)

Read I2S data. Reception is complete when callback is called with I2S_EVENT_RX_EMPTY.

Implemented as

- R_SSI_Read()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	p_dest	Buffer to store PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffer. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, receive will stop at the multiple of 8 below requested bytes.

◆ writeRead

fsp_err_t(* i2s_api_t::writeRead) (i2s_ctrl_t *const p_ctrl, void const *const p_src, void *const p_dest, uint32_t const bytes)

Simultaneously write and read I2S data. Transmission and reception are complete when callback is called with I2S_EVENT_IDLE.

Implemented as

- R_SSI_WriteRead()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[in]	p_src	Buffer of PCM samples. Must be 4 byte aligned.
[in]	p_dest	Buffer to store PCM samples. Must be 4 byte aligned.
[in]	bytes	Number of bytes in the buffers. Recommended requesting a multiple of 8 bytes. If not a multiple of 8, padding 0s will be added to transmission to make it a multiple of 8, and receive will stop at the multiple of 8 below requested bytes.

◆ statusGet

fsp_err_t(* i2s_api_t::statusGet) (i2s_ctrl_t *const p_ctrl, i2s_status_t *const p_status)

Get current status and store it in provided pointer p_status.

Implemented as

- R_SSI_StatusGet()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.
[out]	p_status	Current status of the driver.

◆ close

fsp_err_t(* i2s_api_t::close) (i2s_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

Implemented as

- R_SSI_Close()

Parameters

[in]	p_ctrl	Control block set in i2s_api_t::open call for this instance.

◆ callbackSet

fsp_err_t(* i2s_api_t::callbackSet) (i2s_ctrl_t *const p_api_ctrl, void(*p_callback)(i2s_callback_args_t *), void const *const p_context, i2s_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_SSI_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the I2S control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ i2s_instance_t

struct i2s_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
i2s_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
i2s_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
i2s_api_t const *	p_api	Pointer to the API structure for

		this instance.

Typedef Documentation

◆ i2s_ctrl_t

typedef void i2s_ctrl_t
I2S control block. Allocate an instance specific control block to pass into the I2S API calls. **Implemented as** ◦ ssi_instance_ctrl_t

Enumeration Type Documentation

◆ i2s_pcm_width_t

enum i2s_pcm_width_t
Audio PCM width

Enumerator	
I2S_PCM_WIDTH_8_BITS	Using 8-bit PCM.
I2S_PCM_WIDTH_16_BITS	Using 16-bit PCM.
I2S_PCM_WIDTH_18_BITS	Using 18-bit PCM.
I2S_PCM_WIDTH_20_BITS	Using 20-bit PCM.
I2S_PCM_WIDTH_22_BITS	Using 22-bit PCM.
I2S_PCM_WIDTH_24_BITS	Using 24-bit PCM.
I2S_PCM_WIDTH_32_BITS	Using 24-bit PCM.

◆ i2s_word_length_t

enum i2s_word_length_t	
Audio system word length.	
Enumerator	
I2S_WORD_LENGTH_8_BITS	Using 8-bit system word length.
I2S_WORD_LENGTH_16_BITS	Using 16-bit system word length.
I2S_WORD_LENGTH_24_BITS	Using 24-bit system word length.
I2S_WORD_LENGTH_32_BITS	Using 32-bit system word length.
I2S_WORD_LENGTH_48_BITS	Using 48-bit system word length.
I2S_WORD_LENGTH_64_BITS	Using 64-bit system word length.
I2S_WORD_LENGTH_128_BITS	Using 128-bit system word length.
I2S_WORD_LENGTH_256_BITS	Using 256-bit system word length.

◆ i2s_event_t

enum i2s_event_t	
Events that can trigger a callback function	
Enumerator	
I2S_EVENT_IDLE	Communication is idle.
I2S_EVENT_TX_EMPTY	Transmit buffer is below FIFO trigger level.
I2S_EVENT_RX_FULL	Receive buffer is above FIFO trigger level.

◆ i2s_mode_t

enum i2s_mode_t	
I2S communication mode	
Enumerator	
I2S_MODE_SLAVE	Slave mode.
I2S_MODE_MASTER	Master mode.

◆ i2s_mute_t

enum i2s_mute_t	
Mute audio samples.	
Enumerator	
I2S_MUTE_OFF	Disable mute.
I2S_MUTE_ON	Enable mute.

◆ i2s_ws_continue_t

enum i2s_ws_continue_t	
Whether to continue WS (word select line) transmission during idle state.	
Enumerator	
I2S_WS_CONTINUE_ON	Enable WS continue mode.
I2S_WS_CONTINUE_OFF	Disable WS continue mode.

◆ i2s_state_t

enum i2s_state_t	
Possible status values returned by i2s_api_t::statusGet.	
Enumerator	
I2S_STATE_IN_USE	I2S is in use.
I2S_STATE_STOPPED	I2S is stopped.

4.3.21 I3C Interface

Interfaces

Detailed Description

Interface for I3C.

Summary

The I3C interface provides APIs and definitions for I3C communication.

Known Implementations

I3C Bus Interface (r_i3c)

Data Structures

struct	i3c_device_status_t
struct	i3c_slave_info_t
struct	i3c_device_table_cfg_t
struct	i3c_device_cfg_t
struct	i3c_command_descriptor_t
struct	i3c_callback_args_t
struct	i3c_cfg_t
struct	i3c_api_t
struct	i3c_instance_t

Enumerations

enum	i3c_common_command_code_t
enum	i3c_event_t
enum	i3c_device_type_t
enum	i3c_device_protocol_t
enum	i3c_address_assignment_mode_t

enum	i3c_ibi_type_t

Data Structure Documentation

◆ i3c_device_status_t

struct i3c_device_status_t

The current status of the slave device (See GETSTATUS in the MIPI I3C Specification v1.0).

Data Fields		
uint8_t	pending_interrupt	Contains the interrupt number of any pending interrupt, or 0 if no interrupts are pending.
uint8_t	vendor_status	Reserved for vendor-specific meaning.

◆ i3c_slave_info_t

struct i3c_slave_info_t

Device characteristics that define the I3C capabilities of a slave.

Data Fields		
uint8_t	pid[6]	Provisional ID.
union i3c_slave_info_t	__unnamed__	
uint8_t	dcr	Device Characteristics Register.

◆ i3c_device_table_cfg_t

struct i3c_device_table_cfg_t

Structure for configuring an entry in the device table when the driver is in master mode (See i3c_api_t::masterDeviceTableSet).

Data Fields		
uint8_t	static_address	I3C Static address / I2C address for this device.
uint8_t	dynamic_address	Dynamic address for the device. This address will be assigned during Dynamic Address Assignment.
i3c_device_protocol_t	device_protocol	The protocol used to communicate with this device (I3C / I2C Legacy).
bool	ibi_accept	Accept or reject IBI requests from this device.
bool	master_request_accept	Accept mastership requests from this device.
bool	ibi_payload	IBI requests from this device

		have a data payload. Note: When the device is configured using ENTDAA, the ibi_payload will automatically be updated based on the value of BCR.

◆ **i3c_device_cfg_t**

struct i3c_device_cfg_t		
Structure for configuring a slave address when the driver is in slave mode (See i3c_api_t::deviceCfgSet).		
Data Fields		
uint8_t	static_address	I3C Static address / I2C address for this device.
uint8_t	dynamic_address	Dynamic address for this device. Note that the dynamic address will automatically be updated when ENTDAA is completed.
i3c_slave_info_t	slave_info	PID, BCR, and DCR registers for the device (Slave mode only).

◆ **i3c_command_descriptor_t**

struct i3c_command_descriptor_t		
Descriptor for completing CCC transfers.		
Data Fields		
uint8_t	command_code	Common Command Code for the transfer.
uint8_t *	p_buffer	Buffer for reading or writing data.
uint32_t	length	Length of the data portion of the command.
bool	restart	If true, issue a repeated-start after the transfer is completed.
bool	rnw	Set to true if the command type is Direct Get.

◆ **i3c_callback_args_t**

struct i3c_callback_args_t		
Arguments that are passed to the user callback when an event occurs.		
Data Fields		
i3c_event_t	event	The type of event that has

		occurred.
uint32_t	event_status	Status flags associated with the event.
uint32_t	transfer_size	Number of bytes transferred.
i3c_slave_info_t const *	p_slave_info	A pointer to the Characteristics Registers read during ENTDAA.
uint8_t	dynamic_address	The dynamic address that was assigned to the slave during ENTDAA.
i3c_ibi_type_t	ibi_type	The type of IBI that has been received.
uint8_t	ibi_address	The address of the device that sent the IBI.
uint8_t	command_code	The command code of the received command.
void const *	p_context	User defined context.

◆ **i3c_cfg_t**

struct i3c_cfg_t

User configuration structure, used in open function

Data Fields

uint32_t	channel
i3c_device_type_t	device_type
void(*	p_callback)(i3c_callback_args_t const *const p_args)
void const *	p_context
	Pointer to the user-provided context.
void const *	p_extend

Field Documentation

◆ **channel**

uint32_t i3c_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

◆ **device_type**

i3c_device_type_t i3c_cfg_t::device_type
The type of device.

◆ **p_callback**

void(* i3c_cfg_t::p_callback) (i3c_callback_args_t const *const p_args)
Pointer to the user callback.

◆ **p_extend**

void const* i3c_cfg_t::p_extend
Pointer to extended configuration.

◆ **i3c_api_t**

struct i3c_api_t
I3C functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(i3c_ctrl_t *const p_ctrl, i3c_cfg_t const *const p_cfg)
fsp_err_t(*	enable)(i3c_ctrl_t *const p_ctrl)
fsp_err_t(*	deviceCfgSet)(i3c_ctrl_t *const p_ctrl, i3c_device_cfg_t const *const p_device_cfg)
fsp_err_t(*	masterDeviceTableSet)(i3c_ctrl_t *const p_ctrl, uint32_t device_index, i3c_device_table_cfg_t const *const p_device_table_cfg)
fsp_err_t(*	deviceSelect)(i3c_ctrl_t *const p_ctrl, uint32_t device_index, uint32_t bitrate_mode)
fsp_err_t(*	dynamicAddressAssignmentStart)(i3c_ctrl_t *const p_ctrl, i3c_address_assignment_mode_t address_assignment_mode, uint32_t starting_device_index, uint32_t device_count)
fsp_err_t(*	slaveStatusSet)(i3c_ctrl_t *const p_ctrl, i3c_device_status_t device_status)
fsp_err_t(*	commandSend)(i3c_ctrl_t *const p_ctrl, i3c_command_descriptor_t

	*p_command_descriptor)
fsp_err_t(*	write)(i3c_ctrl_t *const p_ctrl, uint8_t const *const p_data, uint32_t length, bool restart)
fsp_err_t(*	read)(i3c_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t length, bool restart)
fsp_err_t(*	ibiWrite)(i3c_ctrl_t *const p_ctrl, i3c_ibi_type_t ibi_type, uint8_t const *const p_data, uint32_t length)
fsp_err_t(*	ibiRead)(i3c_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t length)
fsp_err_t(*	close)(i3c_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* i3c_api_t::open) (i3c_ctrl_t *const p_ctrl, i3c_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_I3C_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ enable

fsp_err_t(* i3c_api_t::enable) (i3c_ctrl_t *const p_ctrl)

Enable the I3C device.

Implemented as

- R_I3C_Enable()

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.

◆ deviceCfgSet

fsp_err_t(* i3c_api_t::deviceCfgSet) (i3c_ctrl_t *const p_ctrl, i3c_device_cfg_t const *const p_device_cfg)

Set the configuration of this device.

Implemented as

- R_I3C_DeviceCfgSet()

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	p_device_cfg	Pointer to device configuration.

◆ masterDeviceTableSet

fsp_err_t(* i3c_api_t::masterDeviceTableSet) (i3c_ctrl_t *const p_ctrl, uint32_t device_index, i3c_device_table_cfg_t const *const p_device_table_cfg)

Set the configuration for the device at the given index in the device table. The configuration will be used by transfers when it is selected by deviceSelect.

Note: This function is not used in slave mode.

Implemented as

- R_I3C_MasterDeviceTableSet()

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	device_index	Index into the device table.
[in]	p_device_table_cfg	Pointer to the table settings for the entry in the master device table.

◆ deviceSelect

fsp_err_t(* i3c_api_t::deviceSelect) (i3c_ctrl_t *const p_ctrl, uint32_t device_index, uint32_t bitrate_mode)

In master mode, select the device for the next transfer.

Note: This function is not used in slave mode.

Implemented as

- R_I3C_DeviceSelect()

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	device_index	Index into the device table.
[in]	bitrate_setting	The bitrate settings for the selected device.

◆ dynamicAddressAssignmentStart

fsp_err_t(* i3c_api_t::dynamicAddressAssignmentStart) (i3c_ctrl_t *const p_ctrl, i3c_address_assignment_mode_t address_assignment_mode, uint32_t starting_device_index, uint32_t device_count)

Start Dynamic Address Assignment by sending either the ENTDAA or SETDASA command See i3c_address_assignment_mode_t for more information.

Note: This function is not used in slave mode.

Implemented as

- R_I3C_DynamicAddressAssignmentStart()

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	address_assignment_mode	The command to use for Dynamic Address Assignment.
[in]	starting_device_index	The device index that will be used to assign the first device during Dynamic Address Assignment.
[in]	device_count	The number of devices to assign (Only used with I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA).

◆ slaveStatusSet

fsp_err_t(* i3c_api_t::slaveStatusSet) (i3c_ctrl_t *const p_ctrl, i3c_device_status_t device_status)

Set the status returned to the master in response to a GETSTATUS command.

Note: This function is not used in master mode.

Implemented as

- R_I3C_SlaveStatusSet()

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	device_status	New status settings for responding to the GETSTATUS command code.

◆ commandSend

fsp_err_t(* i3c_api_t::commandSend) (i3c_ctrl_t *const p_ctrl, i3c_command_descriptor_t *p_command_descriptor)

Send a broadcast or directed command to slave devices on the bus.

Note: This function is not used in slave mode.

Implemented as

- R_I3C_CommandSend()

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	p_command_descriptor	A descriptor for executing the command.

◆ write

fsp_err_t(* i3c_api_t::write) (i3c_ctrl_t *const p_ctrl, uint8_t const *const p_data, uint32_t length, bool restart)

In master mode: Start a write transfer. When the transfer is completed send a stop condition or a repeated-start. In slave mode: Set the write buffer and configure the number of bytes that will be transferred before the the transfer is ended by the slave via the 'T' bit or by the master issueing a stop condition.

Implemented as

- R_I3C_Write()

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	p_data	Pointer to a buffer to write.
[in]	length	Number of bytes to transfer.
[in]	restart	If true, issue a repeated-start after the transfer is completed (Master only).

◆ read

fsp_err_t(* i3c_api_t::read) (i3c_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t length, bool restart)

In master mode: Start a read transfer. When the transfer is completed, send a stop condition or a repeated-start. In slave mode: Set the read buffer for storing data read during the transfer. When the buffer is full, the application will receive a callback requesting a new read buffer. If no buffer is provided by the application, the driver will discard any remaining bytes read during the transfer.

Implemented as

- R_I3C_Read()

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	p_data	Pointer to a buffer to store the bytes read during the transfer.
[in]	length	Number of bytes to transfer.
[in]	restart	If true, issue a repeated-start after the transfer is completed (Master only).

◆ ibiWrite

fsp_err_t(* i3c_api_t::ibiWrite) (i3c_ctrl_t *const p_ctrl, i3c_ibi_type_t ibi_type, uint8_t const *const p_data, uint32_t length)

Initiate an IBI write operation.

Note: This function is not used in master mode.

Implemented as

- R_I3C_IbiWrite()

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	p_data	Pointer to a buffer to start the bytes read during the transfer.
[in]	length	Number of bytes to transfer.

◆ ibiRead

fsp_err_t(* i3c_api_t::ibiRead) (i3c_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t length)

Set the read buffer for storing received IBI data (This function is not used in slave mode).

Implemented as

- R_I3C_IbiRead()

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.
[in]	p_data	Pointer to a buffer to store the bytes read during the transfer.
[in]	length	Number of bytes to transfer.

◆ close

fsp_err_t(* i3c_api_t::close) (i3c_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

Implemented as

- R_I3C_Close()

Parameters

[in]	p_ctrl	Control block set in i3c_api_t::open call for this instance.

◆ i3c_instance_t

struct i3c_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
i3c_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
i3c_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
i3c_api_t const *	p_api	Pointer to the API structure for this instance.

Enumeration Type Documentation

◆ i3c_common_command_code_t

enum i3c_common_command_code_t	
Common Command Codes defined by MIPI I3C Specification v1.1.	
Enumerator	
I3C_CCC_BROADCAST_ENEC	Enable Slave initiated events.
I3C_CCC_BROADCAST_DISEC	Disable Slave initiated events.
I3C_CCC_BROADCAST_ENTAS0	Enter Activity State 0.
I3C_CCC_BROADCAST_ENTAS1	Enter Activity State 1.
I3C_CCC_BROADCAST_ENTAS2	Enter Activity State 2.
I3C_CCC_BROADCAST_ENTAS3	Enter Activity State 3.
I3C_CCC_BROADCAST_RSTDAA	Reset Dynamic Address Assignment.
I3C_CCC_BROADCAST_ENTDAA	Enter Dynamic Address Assignment.
I3C_CCC_BROADCAST_DEFSVLS	Define List of Slaves.
I3C_CCC_BROADCAST_SETMWL	Set Max Write Length.
I3C_CCC_BROADCAST_SETMRL	Set Max Read Length.
I3C_CCC_BROADCAST_ENTTM	Enter Test Mode.
I3C_CCC_BROADCAST_ENTHDR0	Enter HDR Mode 0.
I3C_CCC_BROADCAST_ENTHDR1	Enter HDR Mode 1.
I3C_CCC_BROADCAST_ENTHDR2	Enter HDR Mode 2.
I3C_CCC_BROADCAST_ENTHDR3	Enter HDR Mode 3.
I3C_CCC_BROADCAST_ENTHDR4	Enter HDR Mode 4 (Reserved for future definition).
I3C_CCC_BROADCAST_ENTHDR5	Enter HDR Mode 5 (Reserved for future definition).
I3C_CCC_BROADCAST_ENTHDR6	Enter HDR Mode 6 (Reserved for future definition).
I3C_CCC_BROADCAST_ENTHDR7	Enter HDR Mode 7 (Reserved for future

	definition).
I3C_CCC_BROADCAST_SETXTIME	Set Exchange Timing Info.
I3C_CCC_BROADCAST_SETAASA	Set All Addresses to Static Address.
I3C_CCC_DIRECT_ENEC	Enable Slave initiated events.
I3C_CCC_DIRECT_DISEC	Disable Slave initiated events.
I3C_CCC_DIRECT_ENTAS0	Enter Activity State 0.
I3C_CCC_DIRECT_ENTAS1	Enter Activity State 1.
I3C_CCC_DIRECT_ENTAS2	Enter Activity State 2.
I3C_CCC_DIRECT_ENTAS3	Enter Activity State 3.
I3C_CCC_DIRECT_RSTDAA	Reset Dynamic Address Assignment.
I3C_CCC_DIRECT_SETDASA	Set Dynamic Address from Static Address.
I3C_CCC_DIRECT_SETNEWDA	Set New Dynamic Address.
I3C_CCC_DIRECT_SETMWL	Set Max Write Length.
I3C_CCC_DIRECT_SETMRL	Set Max Read Length.
I3C_CCC_DIRECT_GETMWL	Get Max Write Length.
I3C_CCC_DIRECT_GETMRL	Get Max Read Length.
I3C_CCC_DIRECT_GETPID	Get Provisional ID.
I3C_CCC_DIRECT_GETBCR	Get Bus Characteristic Register.
I3C_CCC_DIRECT_GETDCR	Get Device Characteristic Register.
I3C_CCC_DIRECT_GETSTATUS	Get Device Status.
I3C_CCC_DIRECT_GETACCMST	Get Accept Mastership.
I3C_CCC_DIRECT_GETMXDS	Get Max Data Speed.
I3C_CCC_DIRECT_SETXTIME	Set Exchange Timing Information.
I3C_CCC_DIRECT_GETXTIME	Get Exchange Timing Information.

◆ i3c_event_t

enum i3c_event_t

I3C Events that result in a callback.

Enumerator	
I3C_EVENT_ENTDAA_ADDRESS_PHASE	Events that only occur in Master mode. A Slave device has finished writing its PID, BCR, and DCR. This information is provided in i3c_callback_args_t::p_slave_info.
I3C_EVENT_IBI_READ_COMPLETE	An IBI has successfully been read.
I3C_EVENT_IBI_READ_BUFFER_FULL	There is no more space in the IBI read buffer. The application may provide another buffer by calling i3c_api_t::ibiRead.
I3C_EVENT_READ_BUFFER_FULL	Events that only occur in Slave mode. There is no more space in the read buffer. The application may provide another buffer by calling i3c_api_t::read.
I3C_EVENT_IBI_WRITE_COMPLETE	A IBI was written successfully.
I3C_EVENT_HDR_EXIT_PATTERN_DETECTED	The HDR exit pattern was detected on the bus.
I3C_EVENT_ADDRESS_ASSIGNMENT_COMPLETE	Dynamic Address Assignment has completed. Events that are common to Master and Slave mode.
I3C_EVENT_COMMAND_COMPLETE	A command was completed.
I3C_EVENT_WRITE_COMPLETE	A write transfer has completed.
I3C_EVENT_READ_COMPLETE	A read transfer has completed.
I3C_EVENT_TIMEOUT_DETECTED	SCL is stuck at the logic high or logic low level during a transfer.
I3C_EVENT_INTERNAL_ERROR	An internal error occurred.

◆ i3c_device_type_t

enum i3c_device_type_t	
The type of device.	
Enumerator	
I3C_DEVICE_TYPE_MAIN_MASTER	The main master starts in master mode and is responsible for configuring the bus.
I3C_DEVICE_TYPE_SLAVE	A slave device listens to the bus for relevant I3C Commands (CCCs) sent by the current master, and responds accordingly. Slave devices may also initiate In-band interrupts and Hot-Join requests.

◆ i3c_device_protocol_t

enum i3c_device_protocol_t	
Identifies the protocol for transferring data with the device on the bus.	
Enumerator	
I3C_DEVICE_PROTOCOL_I2C	Transfers will use legacy I2C protocal with open-drain output at a reduced baudrate.
I3C_DEVICE_PROTOCOL_I3C	Transfers will use I3C SDR mode.

◆ i3c_address_assignment_mode_t

enum i3c_address_assignment_mode_t	
Address Assignment Mode.	
Enumerator	
I3C_ADDRESS_ASSIGNMENT_MODE_ENTDAA	Send the ENTDAA command to enter Dynamic Address Assignment mode and assign dynamic addresses in order, starting with the starting device index. The procedure is completed after the specified number of devices have been configured. The callback will be called after the PID, DCR, and BCR registers have been read for each device.

◆ **i3c_ibi_type_t**

enum i3c_ibi_type_t	
The type of In-Band Interrupt.	
Enumerator	
I3C_IBI_TYPE_INTERRUPT	Application specific In-Band Interrupt for notifying the master when an event occurs.
I3C_IBI_TYPE_HOT_JOIN	Request the master to perform the Dynamic Address Assignment process.
I3C_IBI_TYPE_MASTERSHIP_REQUEST	Request the master to give up control of the bus.

4.3.22 I/O Port Interface

Interfaces

Detailed Description

Interface for accessing I/O ports and configuring I/O functionality.

Summary

The IOPort shared interface provides the ability to access the IOPorts of a device at both bit and port level. Port and pin direction can be changed.

IOPORT Interface description: I/O Ports (r_ioport)

Data Structures

struct	ioport_pin_cfg_t
struct	ioport_cfg_t
struct	ioport_api_t
struct	ioport_instance_t

Typedefs

typedef uint16_t	ioport_size_t IO port size on this device. More...

typedef void	ioport_ctrl_t

Enumerations

enum	ioport_peripheral_t
enum	ioport_ethernet_channel_t
enum	ioport_ethernet_mode_t
enum	ioport_cfg_options_t
enum	ioport_pwpr_t

Data Structure Documentation

◆ ioport_pin_cfg_t

struct ioport_pin_cfg_t		
Pin identifier and pin PFS pin configuration value		
Data Fields		
uint32_t	pin_cfg	Pin PFS configuration - Use ioport_cfg_options_t parameters to configure.
bsp_io_port_pin_t	pin	Pin identifier.

◆ ioport_cfg_t

struct ioport_cfg_t		
Multiple pin configuration data for loading into PFS registers by R_IOPORT_Init()		
Data Fields		
uint16_t	number_of_pins	Number of pins for which there is configuration data.
ioport_pin_cfg_t const *	p_pin_cfg_data	Pin configuration data.

◆ ioport_api_t

struct ioport_api_t	
IOPort driver structure. IOPort functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)
fsp_err_t(*	close)(ioport_ctrl_t *const p_ctrl)

fsp_err_t(*	pinsCfg)(ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)
fsp_err_t(*	pinCfg)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg)
fsp_err_t(*	pinEventInputRead)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event)
fsp_err_t(*	pinEventOutputWrite)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value)
fsp_err_t(*	pinRead)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value)
fsp_err_t(*	pinWrite)(ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level)
fsp_err_t(*	portDirectionSet)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask)
fsp_err_t(*	portEventInputRead)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_event_data)
fsp_err_t(*	portEventOutputWrite)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value)
fsp_err_t(*	portRead)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value)
fsp_err_t(*	portWrite)(ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask)

Field Documentation

◆ **open**

fsp_err_t(* ioport_api_t::open) (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)

Initialize internal driver data and initial pin configurations. Called during startup. Do not call this API during runtime. Use ioport_api_t::pinsCfg for runtime reconfiguration of multiple pins.

Implemented as

- R_IOPORT_Open()

Parameters

[in]	p_cfg	Pointer to pin configuration data array.

◆ **close**

fsp_err_t(* ioport_api_t::close) (ioport_ctrl_t *const p_ctrl)

Close the API.

Implemented as

- R_IOPORT_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ **pinsCfg**

fsp_err_t(* ioport_api_t::pinsCfg) (ioport_ctrl_t *const p_ctrl, const ioport_cfg_t *p_cfg)

Configure multiple pins.

Implemented as

- R_IOPORT_PinsCfg()

Parameters

[in]	p_cfg	Pointer to pin configuration data array.

◆ pinCfg

fsp_err_t(* ioport_api_t::pinCfg) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, uint32_t cfg)

Configure settings for an individual pin.

Implemented as

- R_IOPORT_PinCfg()

Parameters

[in]	pin	Pin to be read.
[in]	cfg	Configuration options for the pin.

◆ pinEventInputRead

fsp_err_t(* ioport_api_t::pinEventInputRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_event)

Read the event input data of the specified pin and return the level.

Implemented as

- R_IOPORT_PinEventInputRead()

Parameters

[in]	pin	Pin to be read.
[in]	p_pin_event	Pointer to return the event data.

◆ pinEventOutputWrite

fsp_err_t(* ioport_api_t::pinEventOutputWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t pin_value)

Write pin event data.

Implemented as

- R_IOPORT_PinEventOutputWrite()

Parameters

[in]	pin	Pin event data is to be written to.
[in]	pin_value	Level to be written to pin output event.

◆ pinRead

fsp_err_t(* ioport_api_t::pinRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t *p_pin_value)

Read level of a pin.

Implemented as

- R_IOPORT_PinRead()

Parameters

[in]	pin	Pin to be read.
[in]	p_pin_value	Pointer to return the pin level.

◆ pinWrite

fsp_err_t(* ioport_api_t::pinWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_pin_t pin, bsp_io_level_t level)

Write specified level to a pin.

Implemented as

- R_IOPORT_PinWrite()

Parameters

[in]	pin	Pin to be written to.
[in]	level	State to be written to the pin.

◆ portDirectionSet

fsp_err_t(* ioport_api_t::portDirectionSet) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t direction_values, ioport_size_t mask)

Set the direction of one or more pins on a port.

Implemented as

- R_IOPORT_PortDirectionSet()

Parameters

[in]	port	Port being configured.
[in]	direction_values	Value controlling direction of pins on port (1 - output, 0 - input).
[in]	mask	Mask controlling which pins on the port are to be configured.

◆ portEventInputRead

fsp_err_t(* ioport_api_t::portEventInputRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_event_data)

Read captured event data for a port.

Implemented as

- R_IOPORT_PortEventInputRead()

Parameters

[in]	port	Port to be read.
[in]	p_event_data	Pointer to return the event data.

◆ portEventOutputWrite

fsp_err_t(* ioport_api_t::portEventOutputWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t event_data, ioport_size_t mask_value)

Write event output data for a port.

Implemented as

- R_IOPORT_PortEventOutputWrite()

Parameters

[in]	port	Port event data will be written to.
[in]	event_data	Data to be written as event data to specified port.
[in]	mask_value	Each bit set to 1 in the mask corresponds to that bit's value in event data. being written to port.

◆ portRead

fsp_err_t(* ioport_api_t::portRead) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t *p_port_value)

Read states of pins on the specified port.

Implemented as

- R_IOPORT_PortRead()

Parameters

[in]	port	Port to be read.
[in]	p_port_value	Pointer to return the port value.

◆ portWrite

fsp_err_t(* ioport_api_t::portWrite) (ioport_ctrl_t *const p_ctrl, bsp_io_port_t port, ioport_size_t value, ioport_size_t mask)

Write to multiple pins on a port.

Implemented as

- R_IOPORT_PortWrite()

Parameters

[in]	port	Port to be written to.
[in]	value	Value to be written to the port.
[in]	mask	Mask controlling which pins on the port are written to.

◆ ioport_instance_t

struct ioport_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
ioport_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
ioport_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ioport_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ ioport_size_t

typedef uint16_t ioport_size_t
IO port size on this device. IO port type used with ports

◆ ioport_ctrl_t

typedef void ioport_ctrl_t
IOPORT control block. Allocate an instance specific control block to pass into the IOPORT API calls. **Implemented as** ◦ ioport_instance_ctrl_t

Enumeration Type Documentation

◆ ioport_peripheral_t

enum ioport_peripheral_t

Superset of all peripheral functions.

Enumerator	
IOPORT_PERIPHERAL_IO	Pin will functions as an IO pin
IOPORT_PERIPHERAL_DEBUG	Pin will function as a DEBUG pin
IOPORT_PERIPHERAL_AGT	Pin will function as an AGT peripheral pin
IOPORT_PERIPHERAL_GPT0	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_GPT1	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_SCI0_2_4_6_8	Pin will function as an SCI peripheral pin
IOPORT_PERIPHERAL_SCI1_3_5_7_9	Pin will function as an SCI peripheral pin
IOPORT_PERIPHERAL_SPI	Pin will function as a SPI peripheral pin
IOPORT_PERIPHERAL_IIC	Pin will function as a IIC peripheral pin
IOPORT_PERIPHERAL_KEY	Pin will function as a KEY peripheral pin
IOPORT_PERIPHERAL_CLKOUT_COMP_RTC	Pin will function as a clock/comparator/RTC peripheral pin
IOPORT_PERIPHERAL_CAC_AD	Pin will function as a CAC/ADC peripheral pin
IOPORT_PERIPHERAL_BUS	Pin will function as a BUS peripheral pin
IOPORT_PERIPHERAL_CTSU	Pin will function as a CTSU peripheral pin
IOPORT_PERIPHERAL_ACMPHS	Pin will function as a CMPHS peripheral pin
IOPORT_PERIPHERAL_LCDC	Pin will function as a segment LCD peripheral pin
IOPORT_PERIPHERAL_DE_SCI1_3_5_7_9	Pin will function as an SCI peripheral DEn pin
IOPORT_PERIPHERAL_DALI	Pin will function as a DALI peripheral pin
IOPORT_PERIPHERAL_DE_SCI0_2_4_6_8	Pin will function as an SCI DEn peripheral pin
IOPORT_PERIPHERAL_CAN	Pin will function as a CAN peripheral pin

IOPORT_PERIPHERAL_QSPI	Pin will function as a QSPI peripheral pin
IOPORT_PERIPHERAL_SSI	Pin will function as an SSI peripheral pin
IOPORT_PERIPHERAL_USB_FS	Pin will function as a USB full speed peripheral pin
IOPORT_PERIPHERAL_USB_HS	Pin will function as a USB high speed peripheral pin
IOPORT_PERIPHERAL_GPT2	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_SDHI_MMC	Pin will function as an SD/MMC peripheral pin
IOPORT_PERIPHERAL_GPT3	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_ETHER_MII	Pin will function as an Ethernet MMI peripheral pin
IOPORT_PERIPHERAL_GPT4	Pin will function as a GPT peripheral pin
IOPORT_PERIPHERAL_ETHER_RMII	Pin will function as an Ethernet RMMI peripheral pin
IOPORT_PERIPHERAL_PDC	Pin will function as a PDC peripheral pin
IOPORT_PERIPHERAL_LCD_GRAPHICS	Pin will function as a graphics LCD peripheral pin
IOPORT_PERIPHERAL_CAC	Pin will function as a CAC peripheral pin
IOPORT_PERIPHERAL_TRACE	Pin will function as a debug trace peripheral pin
IOPORT_PERIPHERAL_OSPI	Pin will function as a OSPI peripheral pin
IOPORT_PERIPHERAL_CEC	Pin will function as a CEC peripheral pin
IOPORT_PERIPHERAL_PGAOUT0	Pin will function as a PGAOUT peripheral pin
IOPORT_PERIPHERAL_PGAOUT1	Pin will function as a PGAOUT peripheral pin

◆ ioport_ethernet_channel_t

enum ioport_ethernet_channel_t	
Enumerator	
IOPORT_ETHERNET_CHANNEL_0	Used to select Ethernet channel 0.
IOPORT_ETHERNET_CHANNEL_1	Used to select Ethernet channel 1.
IOPORT_ETHERNET_CHANNEL_END	Marks end of enum - used by parameter checking.

◆ ioport_ethernet_mode_t

enum ioport_ethernet_mode_t	
Enumerator	
IOPORT_ETHERNET_MODE_RMII	Ethernet PHY mode set to MII.
IOPORT_ETHERNET_MODE_MII	Ethernet PHY mode set to RMII.
IOPORT_ETHERNET_MODE_END	Marks end of enum - used by parameter checking.

◆ ioport_cfg_options_t

enum ioport_cfg_options_t	
Options to configure pin functions	
Enumerator	
IOPORT_CFG_PORT_DIRECTION_INPUT	Sets the pin direction to input (default)
IOPORT_CFG_PORT_DIRECTION_OUTPUT	Sets the pin direction to output.
IOPORT_CFG_PORT_OUTPUT_LOW	Sets the pin level to low.
IOPORT_CFG_PORT_OUTPUT_HIGH	Sets the pin level to high.
IOPORT_CFG_PULLUP_ENABLE	Enables the pin's internal pull-up.
IOPORT_CFG_PIM_TTL	Enables the pin's input mode.
IOPORT_CFG_NMOS_ENABLE	Enables the pin's NMOS open-drain output.
IOPORT_CFG_PMOS_ENABLE	Enables the pin's PMOS open-drain ouput.
IOPORT_CFG_DRIVE_MID	Sets pin drive output to medium.
IOPORT_CFG_DRIVE_HS_HIGH	Sets pin drive output to high along with supporting high speed.
IOPORT_CFG_DRIVE_MID_IIC	Sets pin to drive output needed for IIC on a 20mA port.
IOPORT_CFG_DRIVE_HIGH	Sets pin drive output to high.
IOPORT_CFG_EVENT_RISING_EDGE	Sets pin event trigger to rising edge.
IOPORT_CFG_EVENT_FALLING_EDGE	Sets pin event trigger to falling edge.
IOPORT_CFG_EVENT_BOTH_EDGES	Sets pin event trigger to both edges.
IOPORT_CFG_IRQ_ENABLE	Sets pin as an IRQ pin.
IOPORT_CFG_ANALOG_ENABLE	Enables pin to operate as an analog pin.
IOPORT_CFG_PERIPHERAL_PIN	Enables pin to operate as a peripheral pin.

◆ ioport_pwpr_t

enum ioport_pwpr_t	
Enumerator	
IOPORT_PFS_WRITE_DISABLE	Disable PFS write access.
IOPORT_PFS_WRITE_ENABLE	Enable PFS write access.

4.3.23 JPEG Codec Interface

Interfaces

Detailed Description

Interface for JPEG functions.

Data Structures

struct	jpeg_encode_image_size_t
struct	jpeg_callback_args_t
struct	jpeg_cfg_t
struct	jpeg_api_t
struct	jpeg_instance_t

Typedefs

typedef void	jpeg_ctrl_t

Enumerations

enum	jpeg_color_space_t
enum	jpeg_data_order_t
enum	jpeg_status_t
enum	jpeg_decode_pixel_format_t
enum	jpeg_decode_subsample_t

Data Structure Documentation

◆ jpeg_encode_image_size_t

struct jpeg_encode_image_size_t		
Image parameter structure		
Data Fields		
uint16_t	horizontal_stride_pixels	Horizontal stride.
uint16_t	horizontal_resolution	Horizontal Resolution in pixels.
uint16_t	vertical_resolution	Vertical Resolution in pixels.

◆ jpeg_callback_args_t

struct jpeg_callback_args_t		
Callback status structure		
Data Fields		
jpeg_status_t	status	JPEG status.
uint32_t	image_size	JPEG image size.
void const *	p_context	Pointer to user-provided context.

◆ jpeg_cfg_t

struct jpeg_cfg_t	
User configuration structure, used in open function.	
Data Fields	
IRQn_Type	jedi_irq
	Data transfer interrupt IRQ number.
IRQn_Type	jdti_irq
	Decompression interrupt IRQ number.
uint8_t	jdti_ipl
	Data transfer interrupt priority.
uint8_t	jedi_ipl
	Decompression interrupt priority.

jpeg_mode_t	default_mode
	Mode to use at startup.
jpeg_data_order_t	decode_input_data_order
	Input data stream byte order.
jpeg_data_order_t	decode_output_data_order
	Output data stream byte order.
jpeg_decode_pixel_format_t	pixel_format
	Pixel format.
uint8_t	alpha_value
	Alpha value to be applied to decoded pixel data. Only valid for ARGB8888 format.
void(*	p_decode_callback)(jpeg_callback_args_t *p_args)
	User-supplied callback functions.
void const *	p_decode_context
	Placeholder for user data. Passed to user callback in jpeg_callback_args_t.
jpeg_data_order_t	encode_input_data_order
	Input data stream byte order.
jpeg_data_order_t	encode_output_data_order
	Output data stream byte order.

uint16_t	dri_marker
	DRI Marker setting (0 = No DRI or RST marker)
uint16_t	horizontal_resolution
	Horizontal resolution of input image.
uint16_t	vertical_resolution
	Vertical resolution of input image.
uint16_t	horizontal_stride_pixels
	Horizontal stride of input image.
uint8_t const *	p_quant_luma_table
	Luma quantization table.
uint8_t const *	p_quant_chroma_table
	Chroma quantization table.
uint8_t const *	p_huffman_luma_ac_table
	Huffman AC table for luma.
uint8_t const *	p_huffman_luma_dc_table
	Huffman DC table for luma.
uint8_t const *	p_huffman_chroma_ac_table
	Huffman AC table for chroma.

uint8_t const *	p_huffman_chroma_dc_table
	Huffman DC table for chroma.
void(*	p_encode_callback)(jpeg_callback_args_t *p_args)
	User-supplied callback functions.
void const *	p_encode_context
	Placeholder for user data. Passed to user callback in jpeg_callback_args_t.

◆ jpeg_api_t

struct jpeg_api_t	
JPEG functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(jpeg_ctrl_t *const p_ctrl, jpeg_cfg_t const *const p_cfg)
fsp_err_t(*	inputBufferSet)(jpeg_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
fsp_err_t(*	outputBufferSet)(jpeg_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)
fsp_err_t(*	statusGet)(jpeg_ctrl_t *const p_ctrl, jpeg_status_t *const p_status)
fsp_err_t(*	close)(jpeg_ctrl_t *const p_ctrl)
fsp_err_t(*	horizontalStrideSet)(jpeg_ctrl_t *const p_ctrl, uint32_t horizontal_stride)
fsp_err_t(*	pixelFormatGet)(jpeg_ctrl_t *const p_ctrl, jpeg_color_space_t *const p_color_space)
fsp_err_t(*	imageSubsampleSet)(jpeg_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample,

	jpeg_decode_subsample_t vertical_subsample)
fsp_err_t(*	linesDecodedGet)(jpeg_ctrl_t *const p_ctrl, uint32_t *const p_lines)
fsp_err_t(*	imageSizeGet)(jpeg_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)
fsp_err_t(*	imageSizeSet)(jpeg_ctrl_t *const p_ctrl, jpeg_encode_image_size_t *p_image_size)
fsp_err_t(*	modeSet)(jpeg_ctrl_t *const p_ctrl, jpeg_mode_t mode)

Field Documentation

◆ open

fsp_err_t(* jpeg_api_t::open) (jpeg_ctrl_t *const p_ctrl, jpeg_cfg_t const *const p_cfg)

Initial configuration

Implemented as

- R_JPEG_Open()

Precondition

none

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ inputBufferSet

fsp_err_t(* jpeg_api_t::inputBufferSet) (jpeg_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

Assign input data buffer to JPEG codec.

Implemented as

- R_JPEG_InputBufferSet()

Precondition

the JPEG codec module must have been opened properly.

Note

The buffer starting address must be 8-byte aligned.

Parameters

[in]	p_ctrl	Control block set in jpeg_api_t::open call.
[in]	p_buffer	Pointer to the input buffer space
[in]	buffer_size	Size of the input buffer

◆ outputBufferSet

fsp_err_t(* jpeg_api_t::outputBufferSet) (jpeg_ctrl_t *const p_ctrl, void *p_buffer, uint32_t buffer_size)

Assign output buffer to JPEG codec for storing output data.

Implemented as

- R_JPEG_OutputBufferSet()

Precondition

The JPEG codec module must have been opened properly.

Note

The buffer starting address must be 8-byte aligned. For the decoding process, the HLD driver automatically computes the number of lines of the image to decoded so the output data fits into the given space. If the supplied output buffer is not able to hold the entire frame, the application should call the Output Full Callback function so it can be notified when additional buffer space is needed.

Parameters

[in]	p_ctrl	Control block set in jpeg_api_t::open call.
[in]	p_buffer	Pointer to the output buffer space
[in]	buffer_size	Size of the output buffer

◆ statusGet

fsp_err_t(* jpeg_api_t::statusGet) (jpeg_ctrl_t *const p_ctrl, jpeg_status_t *const p_status)

Retrieve current status of the JPEG codec module.

Implemented as

- R_JPEG_StatusGet()

Precondition

the JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_api_t::open call.
[out]	p_status	JPEG module status

◆ close

fsp_err_t(* jpeg_api_t::close) (jpeg_ctrl_t *const p_ctrl)

Cancel an outstanding operation.

Implemented as

- R_JPEG_Close()

Precondition

the JPEG codec module must have been opened properly.

Note

If the encoding or the decoding operation is finished without errors, the HLD driver automatically closes the device. In this case, application does not need to explicitly close the JPEG device.

Parameters

[in]	p_ctrl	Control block set in jpeg_api_t::open call.

◆ horizontalStrideSet

fsp_err_t(* jpeg_api_t::horizontalStrideSet) (jpeg_ctrl_t *const p_ctrl, uint32_t horizontal_stride)

Configure the horizontal stride value.

Implemented as

- R_JPEG_DecodeHorizontalStrideSet()

Precondition
The JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_api_t::open call.
[in]	horizontal_stride	Horizontal stride value to be used for the decoded image data.
[in]	buffer_size	Size of the output buffer

◆ pixelFormatGet

fsp_err_t(* jpeg_api_t::pixelFormatGet) (jpeg_ctrl_t *const p_ctrl, jpeg_color_space_t *const p_color_space)

Get the input pixel format.

Implemented as

- R_JPEG_DecodePixelFormatGet()

Precondition
the JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_api_t::open call.
[out]	p_color_space	JPEG input format.

◆ imageSubsampleSet

fsp_err_t(* jpeg_api_t::imageSubsampleSet) (jpeg_ctrl_t *const p_ctrl, jpeg_decode_subsample_t horizontal_subsample, jpeg_decode_subsample_t vertical_subsample)

Configure the horizontal and vertical subsample settings.

Implemented as

- R_JPEG_DecodeImageSubsampleSet()

Precondition
The JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_api_t::open call.
[in]	horizontal_subsample	Horizontal subsample value
[in]	vertical_subsample	Vertical subsample value

◆ linesDecodedGet

fsp_err_t(* jpeg_api_t::linesDecodedGet) (jpeg_ctrl_t *const p_ctrl, uint32_t *const p_lines)

Return the number of lines decoded into the output buffer.

Implemented as

- R_JPEG_DecodeLinesDecodedGet()

Precondition
the JPEG codec module must have been opened properly.

Parameters

[in]	p_ctrl	Control block set in jpeg_api_t::open call.
[out]	p_lines	Number of lines decoded

◆ imageSizeGet

fsp_err_t(* jpeg_api_t::imageSizeGet) (jpeg_ctrl_t *const p_ctrl, uint16_t *p_horizontal_size, uint16_t *p_vertical_size)

Retrieve image size during decoding operation.

Implemented as

- R_JPEG_DecodeImageSizeGet()

Precondition

the JPEG codec module must have been opened properly.

Note

If the encoding or the decoding operation is finished without errors, the HLD driver automatically closes the device. In this case, application does not need to explicitly close the JPEG device.

Parameters

[in]	p_ctrl	Control block set in jpeg_api_t::open call.
[out]	p_horizontal_size	Image horizontal size, in number of pixels.
[out]	p_vertical_size	Image vertical size, in number of pixels.

◆ imageSizeSet

fsp_err_t(* jpeg_api_t::imageSizeSet) (jpeg_ctrl_t *const p_ctrl, jpeg_encode_image_size_t *p_image_size)

Set image parameters to JPEG Codec

Implemented as

- R_JPEG_EncodeImageSizeSet()

Precondition

The JPEG codec module must have been opened properly.

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_image_size	Pointer to the RAW image parameters

◆ modeSet

fsp_err_t(* jpeg_api_t::modeSet) (jpeg_ctrl_t *const p_ctrl, jpeg_mode_t mode)

Switch between encode and decode mode or vice-versa.

Implemented as

- R_JPEG_ModeSet()

Precondition
The JPEG codec module must have been opened properly. The JPEG Codec can only perform one operation at a time and requires different configuration for encode and decode. This function facilitates easy switching between the two modes in case both are needed in an application.

Parameters

[in]	p_ctrl	Control block set in jpeg_api_t::open call.
[in]	mode	Mode to switch to

◆ jpeg_instance_t

struct jpeg_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
jpeg_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
jpeg_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
jpeg_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ jpeg_ctrl_t

typedef void jpeg_ctrl_t

JPEG decode control block. Allocate an instance specific control block to pass into the JPEG decode API calls.

Implemented as

- jpeg_instance_ctrl_t

Enumeration Type Documentation

◆ jpeg_color_space_t

enum jpeg_color_space_t	
Configuration for this module Image color space definitions	
Enumerator	
JPEG_COLOR_SPACE_YCBCR444	Color Space YCbCr 444.
JPEG_COLOR_SPACE_YCBCR422	Color Space YCbCr 422.
JPEG_COLOR_SPACE_YCBCR420	Color Space YCbCr 420.
JPEG_COLOR_SPACE_YCBCR411	Color Space YCbCr 411.

◆ jpeg_data_order_t

enum jpeg_data_order_t	
Multi-byte Data Format	
Enumerator	
JPEG_DATA_ORDER_NORMAL	(1)(2)(3)(4)(5)(6)(7)(8) Normal byte order
JPEG_DATA_ORDER_BYTE_SWAP	(2)(1)(4)(3)(6)(5)(8)(7) Byte Swap
JPEG_DATA_ORDER_WORD_SWAP	(3)(4)(1)(2)(7)(8)(5)(6) Word Swap
JPEG_DATA_ORDER_WORD_BYTE_SWAP	(4)(3)(2)(1)(8)(7)(6)(5) Word-Byte Swap
JPEG_DATA_ORDER_LONGWORD_SWAP	(5)(6)(7)(8)(1)(2)(3)(4) Longword Swap
JPEG_DATA_ORDER_LONGWORD_BYTE_SWAP	(6)(5)(8)(7)(2)(1)(4)(3) Longword Byte Swap
JPEG_DATA_ORDER_LONGWORD_WORD_SWAP	(7)(8)(5)(6)(3)(4)(1)(2) Longword Word Swap
JPEG_DATA_ORDER_LONGWORD_WORD_BYTE_SWAP	(8)(7)(6)(5)(4)(3)(2)(1) Longword Word Byte Swap

◆ jpeg_status_t

enum jpeg_status_t	
JPEG HLD driver internal status information. The driver can simultaneously be in more than any one status at the same time. Parse the status bit-fields using the definitions in this enum to determine driver status	
Enumerator	
JPEG_STATUS_NONE	JPEG codec module is not initialized.
JPEG_STATUS_IDLE	JPEG Codec module is open but not running.
JPEG_STATUS_RUNNING	JPEG Codec is running.
JPEG_STATUS_HEADER_PROCESSING	JPEG Codec module is reading the JPEG header information.
JPEG_STATUS_INPUT_PAUSE	JPEG Codec paused waiting for more input data.
JPEG_STATUS_OUTPUT_PAUSE	JPEG Codec paused after it decoded the number of lines specified by user.
JPEG_STATUS_IMAGE_SIZE_READY	JPEG decoding operation obtained image size, and paused.
JPEG_STATUS_ERROR	JPEG Codec module encountered an error.
JPEG_STATUS_OPERATION_COMPLETE	JPEG Codec has completed the operation.

◆ jpeg_decode_pixel_format_t

enum jpeg_decode_pixel_format_t	
Pixel Data Format	
Enumerator	
JPEG_DECODE_PIXEL_FORMAT_ARGB8888	Pixel Data ARGB8888 format.
JPEG_DECODE_PIXEL_FORMAT_RGB565	Pixel Data RGB565 format.

◆ jpeg_decode_subsample_t

enum jpeg_decode_subsample_t	
Data type for horizontal and vertical subsample settings. This setting applies only to the decoding operation.	
Enumerator	
JPEG_DECODE_OUTPUT_NO_SUBSAMPLE	No subsample. The image is decoded with no reduction in size.
JPEG_DECODE_OUTPUT_SUBSAMPLE_HALF	The output image size is reduced by half.
JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_QUARTER	The output image size is reduced to one-quarter.
JPEG_DECODE_OUTPUT_SUBSAMPLE_ONE_EIGHTH	The output image size is reduced to one-eighth.

4.3.24 Key Matrix Interface

Interfaces

Detailed Description

Interface for key matrix functions.

Summary

The KEYMATRIX interface provides standard Key Matrix functionality including event generation on a rising or falling edge for one or more channels at the same time. The generated event indicates all channels that are active in that instant via a bit mask. This allows the interface to be used with a matrix configuration or a one-to-one hardware implementation that is triggered on either a rising or a falling edge.

Implemented by:

- Key Interrupt (r_kint)

Data Structures

struct	keymatrix_callback_args_t
struct	keymatrix_cfg_t
struct	keymatrix_api_t

struct	keymatrix_instance_t

Typedefs

typedef void	keymatrix_ctrl_t

Enumerations

enum	keymatrix_trigger_t

Data Structure Documentation

◆ keymatrix_callback_args_t

struct keymatrix_callback_args_t

Callback function parameter data

Data Fields		
void const *	p_context	Holder for user data. Set in keymatrix_api_t::open function in keymatrix_cfg_t.
uint32_t	channel_mask	Bit vector representing the physical hardware channel(s) that caused the interrupt.

◆ keymatrix_cfg_t

struct keymatrix_cfg_t

User configuration structure, used in open function

Data Fields

uint32_t	channel_mask
	Key Input channel(s). Bit mask of channels to open.
keymatrix_trigger_t	trigger
	Key Input trigger setting.
uint8_t	ipl
	Interrupt priority level.
IRQn_Type	irq
	NVIC IRQ number.

void(*	p_callback)(keymatrix_callback_args_t *p_args)
	Callback for key interrupt ISR.
void const *	p_context
	Holder for user data. Passed to callback in keymatrix_user_cb_data_t.
void const *	p_extend
	Extension parameter for hardware specific settings.

◆ **keymatrix_api_t**

struct keymatrix_api_t

Key Matrix driver structure. Key Matrix functions implemented at the HAL layer will use this API.

Data Fields

fsp_err_t(*	open)(keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg)
fsp_err_t(*	enable)(keymatrix_ctrl_t *const p_ctrl)
fsp_err_t(*	disable)(keymatrix_ctrl_t *const p_ctrl)
fsp_err_t(*	close)(keymatrix_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* keymatrix_api_t::open) (keymatrix_ctrl_t *const p_ctrl, keymatrix_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_KINT_Open()

Parameters

[out]	p_ctrl	Pointer to control block. Must be declared by user. Value set in this function.
[in]	p_cfg	Pointer to configuration structure. All elements of the structure must be set by user.

◆ enable

fsp_err_t(* keymatrix_api_t::enable) (keymatrix_ctrl_t *const p_ctrl)

Enable Key interrupt

Implemented as

- R_KINT_Enable()

Parameters

[in]	p_ctrl	Control block pointer set in Open call for this Key interrupt.

◆ disable

fsp_err_t(* keymatrix_api_t::disable) (keymatrix_ctrl_t *const p_ctrl)

Disable Key interrupt.

Implemented as

- R_KINT_Disable()

Parameters

[in]	p_ctrl	Control block pointer set in Open call for this Key interrupt.

◆ close

fsp_err_t(* keymatrix_api_t::close) (keymatrix_ctrl_t *const p_ctrl)

Allow driver to be reconfigured. May reduce power consumption.

Implemented as

- R_KINT_Close()

Parameters

[in]	p_ctrl	Control block pointer set in Open call for this Key interrupt.

◆ keymatrix_instance_t

struct keymatrix_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
keymatrix_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
keymatrix_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
keymatrix_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ keymatrix_ctrl_t

typedef void keymatrix_ctrl_t

Key matrix control block. Allocate an instance specific control block to pass into the key matrix API calls.

Implemented as

- kint_instance_ctrl_t

Enumeration Type Documentation

◆ keymatrix_trigger_t

enum keymatrix_trigger_t	
Trigger type: rising edge, falling edge	
Enumerator	
KEYMATRIX_TRIG_FALLING	Falling edge trigger.
KEYMATRIX_TRIG_RISING	Rising edge trigger.

4.3.25 Low Power Modes Interface

Interfaces

Detailed Description

Interface for accessing low power modes.

Summary

This section defines the API for the LPM (Low Power Mode) Driver. The LPM Driver provides functions for controlling power consumption by configuring and transitioning to a low power mode. The LPM driver supports configuration of MCU low power modes using the LPM hardware peripheral. The LPM driver supports low power modes deep standby, standby, sleep, and snooze.

Note

Not all low power modes are available on all MCUs.

The LPM interface is implemented by:

- Low Power Modes (r_lpm)

Data Structures

struct	lpm_cfg_t
struct	lpm_api_t
struct	lpm_instance_t

Typedefs

typedef void	lpm_ctrl_t

Enumerations

enum	lpm_mode_t

enum	lpm_snooze_request_t
enum	lpm_snooze_end_t
enum	lpm_snooze_cancel_t
enum	lpm_snooze_dtc_t
enum	lpm_standby_wake_source_t
enum	lpm_io_port_t
enum	lpm_power_supply_t
enum	lpm_deep_standby_cancel_edge_t
enum	lpm_deep_standby_cancel_source_t
enum	lpm_output_port_enable_t

Data Structure Documentation

◆ lpm_cfg_t

struct lpm_cfg_t		
User configuration structure, used in open function		
Data Fields		
lpm_mode_t	low_power_mode	Low Power Mode
lpm_standby_wake_source_bits_t	standby_wake_sources	Bitwise list of sources to wake from standby
lpm_snooze_request_t	snooze_request_source	Snooze request source
lpm_snooze_end_bits_t	snooze_end_sources	Bitwise list of snooze end sources
lpm_snooze_cancel_t	snooze_cancel_sources	List of snooze cancel sources
lpm_snooze_dtc_t	dtc_state_in_snooze	State of DTC in snooze mode, enabled or disabled
void const *	p_extend	Placeholder for extension.

◆ lpm_api_t

struct lpm_api_t	
LPM driver structure. General LPM functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)

fsp_err_t(*	close)(lpm_ctrl_t *const p_api_ctrl)
fsp_err_t(*	lowPowerReconfigure)(lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)
fsp_err_t(*	lowPowerModeEnter)(lpm_ctrl_t *const p_api_ctrl)
fsp_err_t(*	ioKeepClear)(lpm_ctrl_t *const p_api_ctrl)

Field Documentation

◆ open

fsp_err_t(* lpm_api_t::open) (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)

Initialization function

Implemented as

- R_LPM_Open()

◆ close

fsp_err_t(* lpm_api_t::close) (lpm_ctrl_t *const p_api_ctrl)

Initialization function

Implemented as

- R_LPM_Close()

◆ lowPowerReconfigure

fsp_err_t(* lpm_api_t::lowPowerReconfigure) (lpm_ctrl_t *const p_api_ctrl, lpm_cfg_t const *const p_cfg)

Configure a low power mode.

Implemented as

- R_LPM_LowPowerReconfigure()

Parameters

[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ **lowPowerModeEnter**

fsp_err_t(* lpm_api_t::lowPowerModeEnter) (lpm_ctrl_t *const p_api_ctrl)
Enter low power mode (sleep/standby/deep standby) using WFI macro. Function will return after waking from low power mode. **Implemented as** ◦ R_LPM_LowPowerModeEnter()

◆ **ioKeepClear**

fsp_err_t(* lpm_api_t::ioKeepClear) (lpm_ctrl_t *const p_api_ctrl)
Clear the IOKEEP bit after deep software standby. ◦ **Implemented as** • R_LPM_IoKeepClear()

◆ **lpm_instance_t**

struct lpm_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
lpm_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
lpm_cfg_t const *const	p_cfg	Pointer to the configuration structure for this instance.
lpm_api_t const *const	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ **lpm_ctrl_t**

typedef void lpm_ctrl_t
LPM control block. Allocate an instance specific control block to pass into the LPM API calls. **Implemented as** ◦ lpm_instance_ctrl_t

Enumeration Type Documentation

◆ lpm_mode_t

enum lpm_mode_t	
Low power modes	
Enumerator	
LPM_MODE_SLEEP	Sleep mode.
LPM_MODE_STANDBY	Software Standby mode.
LPM_MODE_STANDBY_SNOOZE	Software Standby mode with Snooze mode enabled.
LPM_MODE_DEEP	Deep Software Standby mode.

◆ lpm_snooze_request_t

enum lpm_snooze_request_t	
Snooze request sources	
Enumerator	
LPM_SNOOZE_REQUEST_RXD0_FALLING	Enable RXD0 falling edge snooze request.
LPM_SNOOZE_REQUEST_IRQ0	Enable IRQ0 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ1	Enable IRQ1 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ2	Enable IRQ2 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ3	Enable IRQ3 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ4	Enable IRQ4 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ5	Enable IRQ5 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ6	Enable IRQ6 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ7	Enable IRQ7 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ8	Enable IRQ8 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ9	Enable IRQ9 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ10	Enable IRQ10 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ11	Enable IRQ11 pin snooze request.

LPM_SNOOZE_REQUEST_IRQ12	Enable IRQ12 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ13	Enable IRQ13 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ14	Enable IRQ14 pin snooze request.
LPM_SNOOZE_REQUEST_IRQ15	Enable IRQ15 pin snooze request.
LPM_SNOOZE_REQUEST_KEY	Enable KR snooze request.
LPM_SNOOZE_REQUEST_ACMPHS0	Enable High-speed analog comparator 0 snooze request.
LPM_SNOOZE_REQUEST_RTC_ALARM	Enable RTC alarm snooze request.
LPM_SNOOZE_REQUEST_RTC_PERIOD	Enable RTC period snooze request.
LPM_SNOOZE_REQUEST_AGT1_UNDERFLOW	Enable AGT1 underflow snooze request.
LPM_SNOOZE_REQUEST_AGT1_COMPARE_A	Enable AGT1 compare match A snooze request.
LPM_SNOOZE_REQUEST_AGT1_COMPARE_B	Enable AGT1 compare match B snooze request.
LPM_SNOOZE_REQUEST_AGT3_UNDERFLOW	Enable AGT3 underflow snooze request.
LPM_SNOOZE_REQUEST_AGT3_COMPARE_A	Enable AGT3 compare match A snooze request.
LPM_SNOOZE_REQUEST_AGT3_COMPARE_B	Enable AGT3 compare match B snooze request.

◆ lpm_snooze_end_t

enum lpm_snooze_end_t	
Snooze end control	
Enumerator	
LPM_SNOOZE_END_STANDBY_WAKE_SOURCES	Transition from Snooze to Normal mode directly.
LPM_SNOOZE_END_AGT1_UNDERFLOW	AGT1 underflow.
LPM_SNOOZE_END_DTC_TRANS_COMPLETE	Last DTC transmission completion.
LPM_SNOOZE_END_DTC_TRANS_COMPLETE_NEGATED	Not Last DTC transmission completion.
LPM_SNOOZE_END_ADC0_COMPARE_MATCH	ADC Channel 0 compare match.
LPM_SNOOZE_END_ADC0_COMPARE_MISMATCH	ADC Channel 0 compare mismatch.
LPM_SNOOZE_END_ADC1_COMPARE_MATCH	ADC 1 compare match.
LPM_SNOOZE_END_ADC1_COMPARE_MISMATCH	ADC 1 compare mismatch.
LPM_SNOOZE_END_SCI0_ADDRESS_MATCH	SCI0 address mismatch.
LPM_SNOOZE_END_AGT3_UNDERFLOW	AGT3 underflow.

◆ lpm_snooze_cancel_t

enum lpm_snooze_cancel_t	
Snooze cancel control	
Enumerator	
LPM_SNOOZE_CANCEL_SOURCE_NONE	No snooze cancel source.
LPM_SNOOZE_CANCEL_SOURCE_ADC0_WCMPM	ADC Channel 0 window compare match.
LPM_SNOOZE_CANCEL_SOURCE_ADC0_WCMPUM	ADC Channel 0 window compare mismatch.
LPM_SNOOZE_CANCEL_SOURCE_DTC_COMPLETE	DTC transfer completion.
LPM_SNOOZE_CANCEL_SOURCE_DOC_DOPCI	Data operation circuit interrupt.

◆ lpm_snooze_dtc_t

enum lpm_snooze_dtc_t	
DTC Enable in Snooze Mode	
Enumerator	
LPM_SNOOZE_DTC_DISABLE	Disable DTC operation.
LPM_SNOOZE_DTC_ENABLE	Enable DTC operation.

◆ lpm_standby_wake_source_t

enum lpm_standby_wake_source_t	
Wake from standby mode sources, does not apply to sleep or deep standby modes	
Enumerator	
LPM_STANDBY_WAKE_SOURCE_IRQ0	IRQ0.
LPM_STANDBY_WAKE_SOURCE_IRQ1	IRQ1.
LPM_STANDBY_WAKE_SOURCE_IRQ2	IRQ2.
LPM_STANDBY_WAKE_SOURCE_IRQ3	IRQ3.
LPM_STANDBY_WAKE_SOURCE_IRQ4	IRQ4.
LPM_STANDBY_WAKE_SOURCE_IRQ5	IRQ5.
LPM_STANDBY_WAKE_SOURCE_IRQ6	IRQ6.
LPM_STANDBY_WAKE_SOURCE_IRQ7	IRQ7.
LPM_STANDBY_WAKE_SOURCE_IRQ8	IRQ8.
LPM_STANDBY_WAKE_SOURCE_IRQ9	IRQ9.
LPM_STANDBY_WAKE_SOURCE_IRQ10	IRQ10.
LPM_STANDBY_WAKE_SOURCE_IRQ11	IRQ11.
LPM_STANDBY_WAKE_SOURCE_IRQ12	IRQ12.
LPM_STANDBY_WAKE_SOURCE_IRQ13	IRQ13.
LPM_STANDBY_WAKE_SOURCE_IRQ14	IRQ14.

LPM_STANDBY_WAKE_SOURCE_IRQ15	IRQ15.
LPM_STANDBY_WAKE_SOURCE_IWDT	Independent watchdog interrupt.
LPM_STANDBY_WAKE_SOURCE_KEY	Key interrupt.
LPM_STANDBY_WAKE_SOURCE_LVD1	Low Voltage Detection 1 interrupt.
LPM_STANDBY_WAKE_SOURCE_LVD2	Low Voltage Detection 2 interrupt.
LPM_STANDBY_WAKE_SOURCE_VBATT	VBATT Monitor interrupt.
LPM_STANDBY_WAKE_SOURCE_ACMPHS0	Analog Comparator High-speed 0 interrupt.
LPM_STANDBY_WAKE_SOURCE_ACMPLP0	Analog Comparator Low-speed 0 interrupt.
LPM_STANDBY_WAKE_SOURCE_RTCALM	RTC Alarm interrupt.
LPM_STANDBY_WAKE_SOURCE_RTCPRD	RTC Period interrupt.
LPM_STANDBY_WAKE_SOURCE_USBHS	USB High-speed interrupt.
LPM_STANDBY_WAKE_SOURCE_USBFS	USB Full-speed interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT1UD	AGT1 underflow interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT1CA	AGT1 compare match A interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT1CB	AGT1 compare match B interrupt.
LPM_STANDBY_WAKE_SOURCE_IIC0	I2C 0 interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT3UD	AGT3 underflow interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT3CA	AGT3 compare match A interrupt.
LPM_STANDBY_WAKE_SOURCE_AGT3CB	AGT3 compare match B interrupt.

◆ lpm_io_port_t

enum lpm_io_port_t	
I/O port state after Deep Software Standby mode	
Enumerator	
LPM_IO_PORT_RESET	When the Deep Software Standby mode is canceled, the I/O ports are in the reset state
LPM_IO_PORT_NO_CHANGE	When the Deep Software Standby mode is canceled, the I/O ports are in the same state as in the Deep Software Standby mode

◆ lpm_power_supply_t

enum lpm_power_supply_t	
Power supply control	
Enumerator	
LPM_POWER_SUPPLY_DEEPCUT0	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is supplied in deep software standby mode
LPM_POWER_SUPPLY_DEEPCUT1	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode
LPM_POWER_SUPPLY_DEEPCUT3	Power to the standby RAM, Low-speed on-chip oscillator, AGTn, and USBFS/HS resume detecting unit is not supplied in deep software standby mode. In addition, LVD is disabled and the low power function in a poweron reset circuit is enabled

◆ lpm_deep_standby_cancel_edge_t

enum lpm_deep_standby_cancel_edge_t	
Deep Standby Interrupt Edge	
Enumerator	
LPM_DEEP_STANDBY_CANCEL_SOURCE_EDGE_NONE	No options for a deep standby cancel source.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_RIS	

ING	IRQ0-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0_FALLING	IRQ0-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_RISING	IRQ1-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1_FALLING	IRQ1-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_RISING	IRQ2-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2_FALLING	IRQ2-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_RISING	IRQ3-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3_FALLING	IRQ3-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_RISING	IRQ4-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4_FALLING	IRQ4-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_RISING	IRQ5-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5_FALLING	IRQ5-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_RISING	IRQ6-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6_FALLING	IRQ6-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_RISING	IRQ7-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7_FALLING	IRQ7-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_RISING	IRQ8-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8_FALLING	IRQ8-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_RISING	IRQ9-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9_FALLING	IRQ9-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_RISING	IRQ10-DS Pin Rising Edge.

LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10_FALLING	IRQ10-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_RISING	IRQ11-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11_FALLING	IRQ11-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_RISING	IRQ12-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12_FALLING	IRQ12-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_RISING	IRQ13-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13_FALLING	IRQ13-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_RISING	IRQ14-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14_FALLING	IRQ14-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_RISING	IRQ14-DS Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ15_FALLING	IRQ14-DS Pin Falling Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1_RISING	LVD1 Rising Slope.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1_FALLING	LVD1 Falling Slope.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2_RISING	LVD2 Rising Slope.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2_FALLING	LVD2 Falling Slope.
LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI_RISING	NMI Pin Rising Edge.
LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI_FALLING	NMI Pin Falling Edge.

◆ lpm_deep_standby_cancel_source_t

enum lpm_deep_standby_cancel_source_t	
Deep Standby cancel sources	
Enumerator	
LPM_DEEP_STANDBY_CANCEL_SOURCE_RESET_ONLY	Cancel deep standby only by reset.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ0	IRQ0.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ1	IRQ1.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ2	IRQ2.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ3	IRQ3.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ4	IRQ4.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ5	IRQ5.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ6	IRQ6.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ7	IRQ7.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ8	IRQ8.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ9	IRQ9.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ10	IRQ10.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ11	IRQ11.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ12	IRQ12.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ13	IRQ13.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ14	IRQ14.
LPM_DEEP_STANDBY_CANCEL_SOURCE_IRQ15	IRQ15.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD1	LVD1.
LPM_DEEP_STANDBY_CANCEL_SOURCE_LVD2	LVD2.
LPM_DEEP_STANDBY_CANCEL_SOURCE_RTC_INTERVAL	RTC Interval Interrupt.

LPM_DEEP_STANDBY_CANCEL_SOURCE_RTC_ALARM	RTC Alarm Interrupt.
LPM_DEEP_STANDBY_CANCEL_SOURCE_NMI	NMI.
LPM_DEEP_STANDBY_CANCEL_SOURCE_USBFS	USBFS Suspend/Resume.
LPM_DEEP_STANDBY_CANCEL_SOURCE_USBHS	USBHS Suspend/Resume.
LPM_DEEP_STANDBY_CANCEL_SOURCE_AGT1	AGT1 Underflow.

◆ **lpm_output_port_enable_t**

enum lpm_output_port_enable_t

Output port enable

Enumerator	
LPM_OUTPUT_PORT_ENABLE_HIGH_IMPEDANCE	0: In Software Standby Mode or Deep Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins are set to the high-impedance state. In Snooze, the status of the address bus and bus control signals are same as before entering Software Standby Mode.
LPM_OUTPUT_PORT_ENABLE_RETAIN	1: In Software Standby Mode, the address output pins, data output pins, and other bus control signal output pins retain the output state.

4.3.26 Low Voltage Detection Interface

Interfaces

Detailed Description

Interface for Low Voltage Detection.

Summary

The LVD driver provides functions for configuring the LVD voltage monitors and detectors.

Implemented by:

- Low Voltage Detection (r_lvd)

Data Structures

struct	lvd_status_t
struct	lvd_callback_args_t
struct	lvd_cfg_t
struct	lvd_api_t
struct	lvd_instance_t

Typedefs

typedef void	lvd_ctrl_t

Enumerations

enum	lvd_threshold_t
enum	lvd_response_t
enum	lvd_voltage_slope_t
enum	lvd_sample_clock_t
enum	lvd_negation_delay_t
enum	lvd_threshold_crossing_t
enum	lvd_current_state_t

Data Structure Documentation

◆ lvd_status_t

struct lvd_status_t		
Current state of a voltage monitor.		
Data Fields		
lvd_threshold_crossing_t	crossing_detected	Threshold crossing detection (latched)
lvd_current_state_t	current_state	Instantaneous status of monitored voltage (above or below threshold)

◆ lvd_callback_args_t

struct lvd_callback_args_t
LVD callback parameter definition

Data Fields		
uint32_t	monitor_number	Monitor number.
lvd_current_state_t	current_state	Current state of the voltage monitor.
void const *	p_context	Placeholder for user data.

◆ lvd_cfg_t

struct lvd_cfg_t
LVD configuration structure

Data Fields

uint32_t	monitor_number
lvd_threshold_t	voltage_threshold
lvd_response_t	detection_response
lvd_voltage_slope_t	voltage_slope
lvd_negation_delay_t	negation_delay
lvd_sample_clock_t	sample_clock_divisor
IRQn_Type	irq
uint8_t	monitor_ipl
void(*	p_callback)(lvd_callback_args_t *p_args)
void const *	p_context
void const *	p_extend

Field Documentation

◆ **monitor_number**

uint32_t lvd_cfg_t::monitor_number

Monitor number, 1, 2, ...

◆ **voltage_threshold**

lvd_threshold_t lvd_cfg_t::voltage_threshold

Threshold for out of range voltage detection

◆ **detection_response**

lvd_response_t lvd_cfg_t::detection_response

Response on detecting a threshold crossing

◆ **voltage_slope**

lvd_voltage_slope_t lvd_cfg_t::voltage_slope

Direction of voltage crossing that will trigger a detection (Rising Edge, Falling Edge, Both).

◆ **negation_delay**

lvd_negation_delay_t lvd_cfg_t::negation_delay

Negation of LVD signal follows reset or voltage in range

◆ **sample_clock_divisor**

lvd_sample_clock_t lvd_cfg_t::sample_clock_divisor

Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering

◆ **irq**

IRQn_Type lvd_cfg_t::irq

Interrupt number.

◆ **monitor_ipl**

uint8_t lvd_cfg_t::monitor_ipl

Interrupt priority level.

◆ **p_callback**

void(* lvd_cfg_t::p_callback) (lvd_callback_args_t *p_args)

User function to be called from interrupt

◆ **p_context**

void const* lvd_cfg_t::p_context

Placeholder for user data. Passed to the user callback in

◆ p_extend

void const* lvd_cfg_t::p_extend
Extension parameter for hardware specific settings

◆ lvd_api_t

struct lvd_api_t
LVD driver API structure. LVD driver functions implemented at the HAL layer will adhere to this API.

Data Fields

fsp_err_t(*	open)(lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)
fsp_err_t(*	statusGet)(lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)
fsp_err_t(*	statusClear)(lvd_ctrl_t *const p_ctrl)
fsp_err_t(*	callbackSet)(lvd_ctrl_t *const p_api_ctrl, void(*p_callback)(lvd_callback_args_t *), void const *const p_context, lvd_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(lvd_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* lvd_api_t::open) (lvd_ctrl_t *const p_ctrl, lvd_cfg_t const *const p_cfg)

Initializes a low voltage detection driver according to the passed-in configuration structure.

Implemented as

- R_LVD_Open()

Parameters

[in]	p_ctrl	Pointer to control structure for the driver instance
[in]	p_cfg	Pointer to the configuration structure for the driver instance

◆ statusGet

fsp_err_t(* lvd_api_t::statusGet) (lvd_ctrl_t *const p_ctrl, lvd_status_t *p_lvd_status)

Get the current state of the monitor, (threshold crossing detected, voltage currently above or below threshold). Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE.

Implemented as

- R_LVD_StatusGet()

Parameters

[in]	p_ctrl	Pointer to the control structure for the driver instance
[in,out]	p_lvd_status	Pointer to a lvd_status_t structure

◆ statusClear

fsp_err_t(* lvd_api_t::statusClear) (lvd_ctrl_t *const p_ctrl)

Clears the latched status of the monitor. Must be used if the peripheral was initialized with lvd_response_t set to LVD_RESPONSE_NONE.

Implemented as

- R_LVD_StatusClear()

Parameters

[in]	p_ctrl	Pointer to the control structure for the driver instance

◆ callbackSet

fsp_err_t(* lvd_api_t::callbackSet) (lvd_ctrl_t *const p_api_ctrl, void(*p_callback)(lvd_callback_args_t *), void const *const p_context, lvd_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_LVD_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the LVD control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* lvd_api_t::close) (lvd_ctrl_t *const p_ctrl)

Disables the LVD peripheral. Closes the driver instance.

Implemented as

- R_LVD_Close()

Parameters

[in]	p_ctrl	Pointer to the control structure for the driver instance

◆ lvd_instance_t

struct lvd_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
lvd_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
lvd_cfg_t const *	p_cfg	Pointer to the configuration structure for this interface instance.

lvd_api_t const *	p_api	Pointer to the API structure for this interface instance.

Typedef Documentation

◆ lvd_ctrl_t

typedef void lvd_ctrl_t
LVD control block. Allocate an instance specific control block to pass into the LVD API calls. **Implemented as** ◦ lvd_instance_ctrl_t

Enumeration Type Documentation

◆ lvd_threshold_t

enum lvd_threshold_t

Register definitions, common services, and error codes. Voltage detection level The thresholds supported by each MCU are in the MCU User's Manual as well as in the r_lvd module description on the stack tab of the RA project.

Enumerator	
LVD_THRESHOLD_MONITOR_1_LEVEL_4_29V	4.29V
LVD_THRESHOLD_MONITOR_1_LEVEL_4_14V	4.14V
LVD_THRESHOLD_MONITOR_1_LEVEL_4_02V	4.02V
LVD_THRESHOLD_MONITOR_1_LEVEL_3_84V	3.84V
LVD_THRESHOLD_MONITOR_1_LEVEL_3_10V	3.10V
LVD_THRESHOLD_MONITOR_1_LEVEL_3_00V	3.00V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_90V	2.90V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_79V	2.79V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_68V	2.68V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_58V	2.58V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_48V	2.48V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_20V	2.20V

LVD_THRESHOLD_MONITOR_1_LEVEL_1_96V	1.96V
LVD_THRESHOLD_MONITOR_1_LEVEL_1_86V	1.86V
LVD_THRESHOLD_MONITOR_1_LEVEL_1_75V	1.75V
LVD_THRESHOLD_MONITOR_1_LEVEL_1_65V	1.65V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_99V	2.99V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_92V	2.92V
LVD_THRESHOLD_MONITOR_1_LEVEL_2_85V	2.85V
LVD_THRESHOLD_MONITOR_2_LEVEL_4_29V	4.29V
LVD_THRESHOLD_MONITOR_2_LEVEL_4_14V	4.14V
LVD_THRESHOLD_MONITOR_2_LEVEL_4_02V	4.02V
LVD_THRESHOLD_MONITOR_2_LEVEL_3_84V	3.84V
LVD_THRESHOLD_MONITOR_2_LEVEL_2_99V	2.99V
LVD_THRESHOLD_MONITOR_2_LEVEL_2_92V	2.92V
LVD_THRESHOLD_MONITOR_2_LEVEL_2_85V	2.85V

◆ lvd_response_t

enum lvd_response_t

Response types for handling threshold crossing event.

Enumerator	
LVD_RESPONSE_NMI	Non-maskable interrupt.
LVD_RESPONSE_INTERRUPT	Maskable interrupt.
LVD_RESPONSE_RESET	Reset.
LVD_RESPONSE_NONE	No response, status must be requested via statusGet function.

◆ lvd_voltage_slope_t

enum lvd_voltage_slope_t	
The direction from which Vcc must cross the threshold to trigger a detection (rising, falling, or both).	
Enumerator	
LVD_VOLTAGE_SLOPE_RISING	When VCC >= Vdet2 (rise) is detected.
LVD_VOLTAGE_SLOPE_FALLING	When VCC < Vdet2 (drop) is detected.
LVD_VOLTAGE_SLOPE_BOTH	When drop and rise are detected.

◆ lvd_sample_clock_t

enum lvd_sample_clock_t	
Sample clock divider, use LVD_SAMPLE_CLOCK_DISABLED to disable digital filtering	
Enumerator	
LVD_SAMPLE_CLOCK_LOCO_DIV_2	Digital filter sample clock is LOCO divided by 2.
LVD_SAMPLE_CLOCK_LOCO_DIV_4	Digital filter sample clock is LOCO divided by 4.
LVD_SAMPLE_CLOCK_LOCO_DIV_8	Digital filter sample clock is LOCO divided by 8.
LVD_SAMPLE_CLOCK_LOCO_DIV_16	Digital filter sample clock is LOCO divided by 16.
LVD_SAMPLE_CLOCK_DISABLED	Digital filter is disabled.

◆ lvd_negation_delay_t

enum lvd_negation_delay_t	
Negation delay of LVD reset signal follows reset or voltage in range	
Enumerator	
LVD_NEGATION_DELAY_FROM_VOLTAGE	Negation follows a stabilization time (tLVDn) after VCC > Vdet1 is detected. If a transition to software standby or deep software standby is to be made, the only possible value for the RN bit is LVD_NEGATION_DELAY_FROM_VOLTAGE
LVD_NEGATION_DELAY_FROM_RESET	Negation follows a stabilization time (tLVDn) after assertion of the LVDn reset. If a transition to software standby or deep software standby is to be made, the only possible value for the RN bit is LVD_NEGATION_DELAY_FROM_VOLTAGE

◆ lvd_threshold_crossing_t

enum lvd_threshold_crossing_t	
Threshold crossing detection (latched)	
Enumerator	
LVD_THRESHOLD_CROSSING_NOT_DETECTED	Threshold crossing has not been detected.
LVD_THRESHOLD_CROSSING_DETECTED	Threshold crossing has been detected.

◆ lvd_current_state_t

enum lvd_current_state_t	
Instantaneous status of VCC (above or below threshold)	
Enumerator	
LVD_CURRENT_STATE_BELOW_THRESHOLD	VCC < threshold.
LVD_CURRENT_STATE_ABOVE_THRESHOLD	VCC >= threshold or monitor is disabled.

4.3.27 OPAMP Interface

Interfaces

Detailed Description

Interface for Operational Amplifiers.

Summary

The OPAMP interface provides standard operational amplifier functionality, including starting and stopping the amplifier.

Implemented by: Operational Amplifier (r_opamp)

Data Structures

struct	opamp_trim_args_t
struct	opamp_info_t
struct	opamp_status_t
struct	opamp_cfg_t
struct	opamp_api_t
struct	opamp_instance_t

Typedefs

typedef void	opamp_ctrl_t

Enumerations

enum	opamp_trim_cmd_t
enum	opamp_trim_input_t

Data Structure Documentation

◆ opamp_trim_args_t

struct opamp_trim_args_t		
OPAMP trim arguments.		
Data Fields		
uint8_t	channel	Channel.
opamp_trim_input_t	input	Which input of the channel above.

◆ opamp_info_t

struct opamp_info_t		
OPAMP information.		
Data Fields		
uint32_t	min_stabilization_wait_us	Minimum stabilization wait time in microseconds.

◆ opamp_status_t

struct opamp_status_t		
OPAMP status.		
Data Fields		
uint32_t	operating_channel_mask	Bitmask of channels currently operating.

◆ opamp_cfg_t

struct opamp_cfg_t		
OPAMP general configuration.		
Data Fields		
void const *	p_extend	Extension parameter for hardware specific settings.

◆ opamp_api_t

struct opamp_api_t	
OPAMP functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(opamp_ctrl_t *const p_ctrl, opamp_cfg_t const *const p_cfg)
fsp_err_t(*	start)(opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)
fsp_err_t(*	stop)(opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)
fsp_err_t(*	trim)(opamp_ctrl_t *const p_ctrl, opamp_trim_cmd_t const cmd, opamp_trim_args_t const *const p_args)
fsp_err_t(*	infoGet)(opamp_ctrl_t *const p_ctrl, opamp_info_t *const p_info)
fsp_err_t(*	statusGet)(opamp_ctrl_t *const p_ctrl, opamp_status_t *const p_status)

fsp_err_t(*	close)(opamp_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* opamp_api_t::open) (opamp_ctrl_t *const p_ctrl, opamp_cfg_t const *const p_cfg)

Initialize the operational amplifier.

Implemented as

- R_OPAMP_Open()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	p_cfg	Pointer to configuration

◆ start

fsp_err_t(* opamp_api_t::start) (opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)

Start the op-amp(s).

Implemented as

- R_OPAMP_Start()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	channel_mask	Bitmask of channels to start

◆ stop

fsp_err_t(* opamp_api_t::stop) (opamp_ctrl_t *const p_ctrl, uint32_t const channel_mask)

Stop the op-amp(s).

Implemented as

- R_OPAMP_Stop()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	channel_mask	Bitmask of channels to stop

◆ trim

fsp_err_t(* opamp_api_t::trim) (opamp_ctrl_t *const p_ctrl, opamp_trim_cmd_t const cmd, opamp_trim_args_t const *const p_args)

Trim the op-amp(s). Not supported on all MCUs. See implementation for procedure details.

Implemented as

- R_OPAMP_Trim()

Parameters

[in]	p_ctrl	Pointer to instance control block
[in]	cmd	Trim command
[in]	p_args	Pointer to arguments for the command

◆ infoGet

fsp_err_t(* opamp_api_t::infoGet) (opamp_ctrl_t *const p_ctrl, opamp_info_t *const p_info)

Provide information such as the recommended minimum stabilization wait time.

Implemented as

- R_OPAMP_InfoGet()

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_info	OPAMP information stored here

◆ statusGet

fsp_err_t(* opamp_api_t::statusGet) (opamp_ctrl_t *const p_ctrl, opamp_status_t *const p_status)

Provide status of each op-amp channel.

Implemented as

- R_OPAMP_StatusGet()

Parameters

[in]	p_ctrl	Pointer to instance control block
[out]	p_status	Status stored here

◆ close

fsp_err_t(* opamp_api_t::close) (opamp_ctrl_t *const p_ctrl)
Close the specified OPAMP unit by ending any scan in progress, disabling interrupts, and removing power to the specified A/D unit.

Implemented as

- R_OPAMP_Close()

Parameters

[in]	p_ctrl	Pointer to instance control block

◆ opamp_instance_t

struct opamp_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
opamp_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
opamp_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
opamp_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ opamp_ctrl_t

typedef void opamp_ctrl_t
OPAMP control block. Allocate using driver instance control structure from driver instance header file.

Enumeration Type Documentation

◆ opamp_trim_cmd_t

enum opamp_trim_cmd_t	
Includes board and MCU related header files. Trim command.	
Enumerator	
OPAMP_TRIM_CMD_START	Initialize trim state machine.
OPAMP_TRIM_CMD_NEXT_STEP	Move to next step in state machine.
OPAMP_TRIM_CMD_CLEAR_BIT	Clear trim bit.

◆ opamp_trim_input_t

enum opamp_trim_input_t	
Trim input.	
Enumerator	
OPAMP_TRIM_INPUT_PCH	Trim non-inverting (+) input.
OPAMP_TRIM_INPUT_NCH	Trim inverting (-) input.

4.3.28 PDC Interface

Interfaces

Detailed Description

Interface for PDC functions.

Summary

The PDC interface provides the functionality for capturing an image from an image sensor/camera. When a capture is complete a transfer complete interrupt is triggered.

Implemented by:

- Parallel Data Capture (r_pdc)

Data Structures

struct	pdc_callback_args_t
struct	pdc_cfg_t

struct	pdc_api_t
struct	pdc_instance_t

Typedefs

typedef void	pdc_ctrl_t

Enumerations

enum	pdc_clock_division_t
enum	pdc_endian_t
enum	pdc_hsync_polarity_t
enum	pdc_vsync_polarity_t
enum	pdc_event_t

Data Structure Documentation

◆ pdc_callback_args_t

struct pdc_callback_args_t

Callback function parameter data

Data Fields

pdc_event_t	event	Event causing the callback.
uint8_t *	p_buffer	Pointer to buffer containing the captured data.
void const *	p_context	Placeholder for user data. Set in pdc_api_t::open function in pdc_cfg_t.

◆ pdc_cfg_t

struct pdc_cfg_t

PDC configuration parameters.

Data Fields

uint16_t	x_capture_start_pixel
	Horizontal position to start capture.
uint16_t	x_capture_pixels
	Number of horizontal pixels to capture.

uint16_t	y_capture_start_pixel
	Vertical position to start capture.
uint16_t	y_capture_pixels
	Number of vertical lines/pixels to capture.
pdc_clock_division_t	clock_division
	Clock divider.
pdc_endian_t	endian
	Endian of capture data.
pdc_hsync_polarity_t	hsync_polarity
	Polarity of HSYNC input.
pdc_vsync_polarity_t	vsync_polarity
	Polarity of VSYNC input.
uint8_t *	p_buffer
	Pointer to buffer to write image into.
uint8_t	bytes_per_pixel
	Number of bytes per pixel.
uint8_t	pdc_ipl
	PDC interrupt priority.

uint8_t	transfer_req_ipl
	Transfer interrupt priority.
IRQn_Type	pdc_irq
	PDC IRQ number.
IRQn_Type	transfer_req_irq
	Transfer request IRQ number.
transfer_instance_t const *	p_lower_lvl_transfer
	Pointer to the transfer instance the PDC should use.
void(*	p_callback)(pdc_callback_args_t *p_args)
	Callback provided when a PDC transfer ISR occurs.
void const *	p_context
	User defined context passed to callback function.
void const *	p_extend
	Placeholder for user data.

◆ pdc_api_t

struct pdc_api_t	
PDC functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(pdc_ctrl_t *const p_ctrl, pdc_cfg_t const *const p_cfg)
fsp_err_t(*	close)(pdc_ctrl_t *const p_ctrl)

fsp_err_t(*	captureStart)(pdc_ctrl_t *const p_ctrl, uint8_t *const p_buffer)

Field Documentation

◆ open

fsp_err_t(* pdc_api_t::open) (pdc_ctrl_t *const p_ctrl, pdc_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_PDC_Open()

Note

To reconfigure after calling this function, call pdc_api_t::close first.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ close

fsp_err_t(* pdc_api_t::close) (pdc_ctrl_t *const p_ctrl)

Closes the driver and allows reconfiguration. May reduce power consumption.

Implemented as

- R_PDC_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ captureStart

fsp_err_t(* pdc_api_t::captureStart) (pdc_ctrl_t *const p_ctrl, uint8_t *const p_buffer)

Start a capture.

Implemented as

- R_PDC_CaptureStart()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_buffer	Pointer to store captured image data.

◆ pdc_instance_t

struct pdc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
pdc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
pdc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
pdc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ pdc_ctrl_t

typedef void pdc_ctrl_t

PDC control block. Allocate an instance specific control block to pass into the PDC API calls.

Implemented as

- pdc_instance_ctrl_t

Enumeration Type Documentation

◆ pdc_clock_division_t

enum pdc_clock_division_t

Clock divider applied to PDC clock to provide PCKO output frequency

Enumerator	
PDC_CLOCK_DIVISION_2	CLK / 2.
PDC_CLOCK_DIVISION_4	CLK / 4.
PDC_CLOCK_DIVISION_6	CLK / 6.
PDC_CLOCK_DIVISION_8	CLK / 8.
PDC_CLOCK_DIVISION_10	CLK / 10.
PDC_CLOCK_DIVISION_12	CLK / 12.
PDC_CLOCK_DIVISION_14	CLK / 14.
PDC_CLOCK_DIVISION_16	CLK / 16.

◆ pdc_endian_t

enum pdc_endian_t	
Endian of captured data	
Enumerator	
PDC_ENDIAN_LITTLE	Data is in little endian format.
PDC_ENDIAN_BIG	Data is in big endian format.

◆ pdc_hsync_polarity_t

enum pdc_hsync_polarity_t	
Polarity of input HSYNC signal	
Enumerator	
PDC_HSYNC_POLARITY_HIGH	HSYNC signal is active high.
PDC_HSYNC_POLARITY_LOW	HSYNC signal is active low.

◆ pdc_vsync_polarity_t

enum pdc_vsync_polarity_t	
Polarity of input VSYNC signal	
Enumerator	
PDC_VSYNC_POLARITY_HIGH	VSYNC signal is active high.
PDC_VSYNC_POLARITY_LOW	VSYNC signal is active low.

◆ pdc_event_t

enum pdc_event_t	
PDC events	
Enumerator	
PDC_EVENT_TRANSFER_COMPLETE	Complete frame transferred by DMAC/DTC.
PDC_EVENT_RX_DATA_READY	Receive data ready interrupt.
PDC_EVENT_FRAME_END	Frame end interrupt.
PDC_EVENT_ERR_OVERRUN	Overrun interrupt.
PDC_EVENT_ERR_UNDERRUN	Underrun interrupt.
PDC_EVENT_ERR_V_SET	Vertical line setting error interrupt.
PDC_EVENT_ERR_H_SET	Horizontal byte number setting error interrupt.

4.3.29 POEG Interface

Interfaces

Detailed Description

Interface for the Port Output Enable for GPT.

Defines the API and data structures for the Port Output Enable for GPT (POEG) interface.

Summary

The POEG disables GPT output pins based on configurable events.

Implemented by: Port Output Enable for GPT (r_poeg)

Data Structures

struct	poeg_status_t
struct	poeg_callback_args_t
struct	poeg_cfg_t

struct	poeg_api_t
struct	poeg_instance_t

Typedefs

typedef void	poeg_ctrl_t

Enumerations

enum	poeg_state_t
enum	poeg_trigger_t
enum	poeg_gtetrg_polarity_t
enum	poeg_gtetrg_noise_filter_t

Data Structure Documentation

◆ poeg_status_t

struct poeg_status_t

POEG status

Data Fields		
poeg_state_t	state	Current state of POEG.

◆ poeg_callback_args_t

struct poeg_callback_args_t

Callback function parameter data.

Data Fields		
void const *	p_context	Placeholder for user data, set in poeg_cfg_t.

◆ poeg_cfg_t

struct poeg_cfg_t

User configuration structure, used in the open function.

Data Fields

poeg_trigger_t	trigger
	Select one or more triggers for the POEG.
poeg_gtetrg_polarity_t	polarity
	Select the polarity for the GTETRG pin.

poeg_gtetrg_noise_filter_t	noise_filter
	Configure the GTETRG noise filter.
void(*	p_callback)(poeg_callback_args_t *p_args)
void const *	p_context
uint32_t	channel
	Channel 0 corresponds to GTETRGA, 1 to GTETRGB, etc.
IRQn_Type	irq
	NVIC interrupt number assigned to this instance.
uint8_t	ipl
	POEG interrupt priority.

Field Documentation

◆ p_callback

void(* poeg_cfg_t::p_callback) (poeg_callback_args_t *p_args)

Callback called when a POEG interrupt occurs.

◆ p_context

void const* poeg_cfg_t::p_context

Placeholder for user data. Passed to the user callback in poeg_callback_args_t.

◆ poeg_api_t

struct poeg_api_t

Port Output Enable for GPT (POEG) API structure. POEG functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(poeg_ctrl_t *const p_ctrl, poeg_cfg_t const *const p_cfg)

fsp_err_t(*	statusGet)(poeg_ctrl_t *const p_ctrl, poeg_status_t *p_status)
fsp_err_t(*	callbackSet)(poeg_ctrl_t *const p_api_ctrl, void(*p_callback)(poeg_callback_args_t *), void const *const p_context, poeg_callback_args_t *const p_callback_memory)
fsp_err_t(*	outputDisable)(poeg_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(poeg_ctrl_t *const p_ctrl)
fsp_err_t(*	close)(poeg_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* poeg_api_t::open) (poeg_ctrl_t *const p_ctrl, poeg_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_POEG_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ statusGet

fsp_err_t(* poeg_api_t::statusGet) (poeg_ctrl_t *const p_ctrl, poeg_status_t *p_status)

Gets the current driver state.

Implemented as

- R_POEG_StatusGet()

Parameters

[in]	p_ctrl	Control block set in poeg_api_t::open call.
[out]	p_status	Provides the current state of the POEG.

◆ callbackSet

fsp_err_t(* poeg_api_t::callbackSet) (poeg_ctrl_t *const p_api_ctrl, void(*p_callback)(poeg_callback_args_t *), void const *const p_context, poeg_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_POEG_CallbackSet()

Parameters

[in]	p_ctrl	Control block set in poeg_api_t::open call for this timer.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ **outputDisable**

fsp_err_t(* poeg_api_t::outputDisable) (poeg_ctrl_t *const p_ctrl)

Disables GPT output pins by software request.

Implemented as

- R_POEG_OutputDisable()

Parameters

[in]	p_ctrl	Control block set in poeg_api_t::open call.

◆ **reset**

fsp_err_t(* poeg_api_t::reset) (poeg_ctrl_t *const p_ctrl)

Attempts to clear status flags to reenable GPT output pins. Confirm all status flags are cleared after calling this function by calling poeg_api_t::statusGet().

Implemented as

- R_POEG_Reset()

Parameters

[in]	p_ctrl	Control block set in poeg_api_t::open call.

◆ **close**

fsp_err_t(* poeg_api_t::close) (poeg_ctrl_t *const p_ctrl)

Disables POEG interrupt.

Implemented as

- R_POEG_Close()

Parameters

[in]	p_ctrl	Control block set in poeg_api_t::open call.

◆ **poeg_instance_t**

struct poeg_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
poeg_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
poeg_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.

poeg_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ poeg_ctrl_t

typedef void poeg_ctrl_t
DOC control block. Allocate an instance specific control block to pass into the DOC API calls. **Implemented as** ◦ poeg_instance_ctrl_t

Enumeration Type Documentation

◆ poeg_state_t

enum poeg_state_t

POEG states.

Enumerator	
POEG_STATE_NO_DISABLE_REQUEST	GPT output is not disabled by POEG.
POEG_STATE_PIN_DISABLE_REQUEST	GPT output disabled due to GTETRG pin level.
POEG_STATE_GPT_OR_COMPARATOR_DISABLE_REQUEST	GPT output disabled due to high speed analog comparator or GPT.
POEG_STATE_OSCILLATION_STOP_DISABLE_REQUEST	GPT output disabled due to main oscillator stop.
POEG_STATE_SOFTWARE_STOP_DISABLE_REQUEST	GPT output disabled due to poeg_api_t::outputDisable()
POEG_STATE_PIN_DISABLE_REQUEST_ACTIVE	GPT output disable request active from the GTETRG pin. If a filter is used, this flag represents the state of the filtered input.

◆ poeg_trigger_t

enum poeg_trigger_t	
Triggers that will disable GPT output pins.	
Enumerator	
POEG_TRIGGER_SOFTWARE	Software disable is always supported with POEG. Select this option if no other triggers are used.
POEG_TRIGGER_PIN	Disable GPT output based on GTETRG input level.
POEG_TRIGGER_GPT_OUTPUT_LEVEL	Disable GPT output based on GPT output pin levels.
POEG_TRIGGER_OSCILLATION_STOP	Disable GPT output based on main oscillator stop.
POEG_TRIGGER_ACMPHS0	Disable GPT output based on ACMPHS0 comparator result.
POEG_TRIGGER_ACMPHS1	Disable GPT output based on ACMPHS1 comparator result.
POEG_TRIGGER_ACMPHS2	Disable GPT output based on ACMPHS2 comparator result.
POEG_TRIGGER_ACMPHS3	Disable GPT output based on ACMPHS3 comparator result.
POEG_TRIGGER_ACMPHS4	Disable GPT output based on ACMPHS4 comparator result.
POEG_TRIGGER_ACMPHS5	Disable GPT output based on ACMPHS5 comparator result.

◆ poeg_gtetrg_polarity_t

enum poeg_gtetrg_polarity_t	
GTETRG polarity.	
Enumerator	
POEG_GTETRG_POLARITY_ACTIVE_HIGH	Disable GPT output based when GTETRG input level is high.
POEG_GTETRG_POLARITY_ACTIVE_LOW	Disable GPT output based when GTETRG input level is low.

◆ poeg_gtetrg_noise_filter_t

enum poeg_gtetrg_noise_filter_t	
GTETRG noise filter. For the input signal to pass through the noise filter, the active level set in poeg_gtetrg_polarity_t must be read 3 consecutive times at the sampling clock selected.	
Enumerator	
POEG_GTETRG_NOISE_FILTER_DISABLED	No noise filter applied to GTETRG input.
POEG_GTETRG_NOISE_FILTER_PCLKB_DIV_1	Apply noise filter with sample clock PCLKB.
POEG_GTETRG_NOISE_FILTER_PCLKB_DIV_8	Apply noise filter with sample clock PCLKB/8.
POEG_GTETRG_NOISE_FILTER_PCLKB_DIV_32	Apply noise filter with sample clock PCLKB/32.
POEG_GTETRG_NOISE_FILTER_PCLKB_DIV_128	Apply noise filter with sample clock PCLKB/128.

4.3.30 PTP Interface

Interfaces

Detailed Description

Interface for PTP functions.

Summary

The PTP interface provides the functionality for using PTP.

Implemented by:

- Precision Time Protocol (r_ptp)

Data Structures

struct	ptp_clock_properties_t
struct	ptp_time_t
struct	ptp_message_flags_t
struct	ptp_message_header_t
struct	ptp_message_sync_t
struct	ptp_message_pdelay_req_t
struct	ptp_message_pdelay_resp_t
struct	ptp_message_announce_t
struct	ptp_message_signaling_t
struct	ptp_message_management_t
struct	ptp_message_t
struct	ptp_callback_args_t
struct	ptp_pulse_timer_common_cfg_t
struct	ptp_pulse_timer_cfg_t
struct	ptp_sync_state_cfg_t
struct	ptp_synfp_cfg_t
struct	ptp_synfp_cfg_t.ether
struct	ptp_synfp_cfg_t.ipv4
struct	ptp_stca_cfg_t
struct	ptp_cfg_t
struct	ptp_api_t
struct	ptp_instance_t

Typedefs

typedef enum PTP_PACKED e_ptp_ctrl_field	ptp_ctrl_field_t
typedef ptp_message_sync_t	ptp_message_delay_req_t
typedef ptp_message_sync_t	ptp_message_follow_up_t
typedef ptp_message_pdelay_resp_t	ptp_message_delay_resp_t
typedef ptp_message_delay_resp_t	ptp_message_pdelay_resp_follow_up_t

Enumerations

enum	ptp_message_type_t
enum	ptp_port_state_t
enum	ptp_clock_delay_mechanism_t
enum	ptp_frame_format_t
enum	ptp_frame_filter_mode_t
enum	ptp_stca_clock_freq_t
enum	ptp_stca_clock_sel_t
enum	ptp_message_interval_t
enum	ptp_clock_correction_mode_t
enum	ptp_event_t
enum	ptp_ethernet_phy_interface_t

Variables

enum PTP_PACKED	e_ptp_ctrl_field

Data Structure Documentation

◆ ptp_clock_properties_t

struct ptp_clock_properties_t

Clock properties used in the best master clock algorithm (BMCA) in order to determine the grandmaster clock.

In master mode, these properties will be advertised in announce messages.

Note: The final property used in BMCA is the clock ID. This is usually configured at runtime because it is often based on the hardware address.

Data Fields		
uint8_t	priority1	Priority1 value used in best master calculation.
uint8_t	cclass	Class value.
uint8_t	accuracy	Accuracy of the clock.
uint16_t	variance	Variance of the clock.
uint8_t	priority2	Priority2 value used as secondary priority in best master calculation.

◆ ptp_time_t

struct ptp_time_t		
Structure for storing time with nanosecond precision .		
Data Fields		
uint16_t	seconds_upper	Upper 16 bits of the seconds.
uint32_t	seconds_lower	Lower 32 bits of the seconds.
uint32_t	nanoseconds	Nanoseconds.

◆ ptp_message_flags_t

struct ptp_message_flags_t
Flags field in PTP message header.

◆ ptp_message_header_t

struct ptp_message_header_t		
Commom PTP Message Header.		
Data Fields		
uint8_t	message_type: 4	The message type.
uint8_t	sdoid_major: 4	Standard Organization ID Major.
uint8_t	version: 4	PTP Version.
uint8_t	minor_version: 4	PTP Minor Version.
uint16_t	message_length	The total message length (Including the header).
uint8_t	domain	The clock domain.

uint8_t	sdoid_minor: 8	Standard Organization ID minor.
ptp_message_flags_t	flags	Flags set in the message.
uint64_t	correction_field	Correction Field that is updated when a message passes through a transparent clock.
uint32_t	reserved	
uint8_t	clock_id[8]	Clock ID that the message was sent from.
uint16_t	source_port_id	Port ID that the message was sent from.
uint16_t	sequence_id	Sequence ID of the message.
ptp_ctrl_field_t	control_field	Control field (Message specifc).
uint8_t	log_message_interval	Logbase2 of the message period.

◆ ptp_message_sync_t

struct ptp_message_sync_t		
Sync Message Type (0x00).		
Data Fields		
ptp_time_t	origin_timestamp	Timestamp when the message was transmitted.

◆ ptp_message_pdelay_req_t

struct ptp_message_pdelay_req_t		
PDelay_req Message Type (0x02).		
Data Fields		
ptp_time_t	origin_timestamp	Timestamp when the message was transmitted.
uint8_t	reserved[10]	

◆ ptp_message_pdelay_resp_t

struct ptp_message_pdelay_resp_t		
PDelay_resp Message Type (0x03).		
Data Fields		
ptp_time_t	origin_timestamp	Timestamp when the message was transmitted.
uint8_t	source_port_identity[10]	Clock ID + sourcePortId.

◆ ptp_message_announce_t

struct ptp_message_announce_t		
Announce Message Type (0x0B).		
Data Fields		
ptp_time_t	origin_timestamp	Timestamp when the message was transmitted.
uint16_t	current_utc_offset	Offset from UTC in seconds.
uint8_t	reserved	
ptp_clock_properties_t	clock_properties	Clock properties used in Best Master Clock Algorithm.
uint8_t	clock_id[8]	Clock ID that the message was sent from.
uint16_t	steps_removed	The number of boundary clocks between the clock and the grand master clock.
uint8_t	time_source	The source of time (Eg. INTERNAL_OSC).

◆ ptp_message_signaling_t

struct ptp_message_signaling_t		
Signaling Message Type (0x0C).		
Data Fields		
uint8_t	target_clock_id[8]	ID of the target PTP instance.
uint16_t	target_port_id	Port of the target PTP instance.

◆ ptp_message_management_t

struct ptp_message_management_t		
Management Message Type (0x0D).		
Data Fields		
uint8_t	target_clock_id[8]	ID of the target PTP instance.
uint16_t	target_port_id	Port of the target PTP instance.
uint8_t	starting_boundary_hops	The starting number of times the message is retransmitted by boundary clocks.
uint8_t	boundary_hops	The remaining number of retransmissions.
uint8_t	action	The action that will be taken on reception of the message.
uint8_t	reserved	

◆ ptp_message_t

struct ptp_message_t		
Complete PTP Message.		
Data Fields		
ptp_message_header_t	header	Header of the message.
union ptp_message_t	__unnamed__	

◆ **ptp_callback_args_t**

struct ptp_callback_args_t		
Arguments passed to p_ptp_callback.		
Data Fields		
ptp_event_t	event	Event that caused the callback.
ptp_message_t const *	p_message	The message received (PTP message fields will be little endian).
uint8_t const *	p_tlv_data	Start of TLV data (TLV data will be big endian).
uint16_t	tlv_data_size	Total bytes of TLV data.
uint32_t	pulse_timer_channel	Channel of the pulse timer that caused ptp_event_t::PTP_EVENT_PULSE_TIMER_MINT_RISING_EDGE.
void const *	p_context	Context value set in the configuration.

◆ **ptp_pulse_timer_common_cfg_t**

struct ptp_pulse_timer_common_cfg_t		
Structure for configuring the IPLS IRQ settings that are common to all pulse timer channels.		
Data Fields		
ptp_enable_t	ipls_rising_irq	Enable the IPLS IRQ when a rising edge is detected.
ptp_enable_t	ipls_falling_irq	Enable the IPLS IRQ when a falling edge is detected.
ptp_enable_t	ipls_rising_irq_auto_clear	Auto disable the rising edge IRQ after the first rising edge is detected.
ptp_enable_t	ipls_falling_irq_auto_clear	Auto disable the falling edge IRQ after the first falling edge is detected.

◆ **ptp_pulse_timer_cfg_t**

struct ptp_pulse_timer_cfg_t

Structure for configuring a pulse timer channel.		
Data Fields		
ptp_time_t	start_time	The exact time when the timer will start.
uint32_t	period	The period of the timer in nanoseconds.
uint32_t	pulse	The pulse width of the timer in nanoseconds.
ptp_enable_t	mint_rising_irq	Enable MINT rising edge IRQ.
ptp_enable_t	ipls_rising_event	Enable IPLS rising edge ELC event.
ptp_enable_t	ipls_falling_event	Enable IPLS falling edge ELC event.
ptp_enable_t	ipls_rising_event_auto_clear	Enable IPLS rising edge ELC event.
ptp_enable_t	ipls_falling_event_auto_clear	Enable IPLS falling edge ELC event.
ptp_enable_t	ipls_irq_source	Enable using this channel as a source for the IPLS IRQ.

◆ ptp_sync_state_cfg_t

struct ptp_sync_state_cfg_t		
Configuration settings for determining when the PTP clock is synchronized.		
Data Fields		
uint64_t	threshold	The maximum clock offset required to transition between synchronization states.
uint8_t	count	The number of times the clock must be above the threshold in order to transition between synchronization states.

◆ ptp_synfp_cfg_t

struct ptp_synfp_cfg_t		
Configuration settings for the SYNFP.		
Data Fields		
ptp_ethernet_phy_interface_t	ethernet_phy_interface	The type of interface used to communicate with the PHY.
ptp_frame_format_t	frame_format	Frame format used to transport PTP messages.
ptp_frame_filter_mode_t	frame_filter	Frame filter mode.

uint8_t	clock_domain	Clock domain that the clock operates in.
ptp_enable_t	clock_domain_filter	Filter out messages from other clock domains.
ptp_message_interval_t	announce_interval	Interval for transmitting announce messages.
ptp_message_interval_t	sync_interval	Interval for transmitting sync messages.
ptp_message_interval_t	delay_req_interval	Interval for transmitting delay_req messages.
uint32_t	message_timeout	Timeout in milliseconds for receiving PTP messages.
ptp_clock_properties_t	clock_properties	Clock properties used in annonce messages.
uint8_t	timesource	TimeSource field used in announce messages.
uint8_t *	p_multicast_addr_filter	Filter for multicast packets.
struct ptp_synfp_cfg_t	ether	Valid if frame_format is set to Ethernet II or IEEE 802.3.
struct ptp_synfp_cfg_t	ipv4	Valid if frame_format is set to IPV4_UDP.

◆ ptp_synfp_cfg_t.ether

struct ptp_synfp_cfg_t.ether

Valid if frame_format is set to Ethernet II or IEEE 802.3.

Data Fields

uint8_t *	p_primary_mac_addr	The MAC address to send primary messages.
uint8_t *	p_pdelay_mac_addr	The MAC address to send p2p messages.

◆ ptp_synfp_cfg_t.ipv4

struct ptp_synfp_cfg_t.ipv4

Valid if frame_format is set to IPV4_UDP.

Data Fields

uint32_t	primary_ip_addr	The IP address to send primary messages.
uint32_t	pdelay_ip_addr	The IP address to send pdelay messages.
uint8_t	event_tos	Type of service for event messages.

uint8_t	general_tos	Type of service for general messages.
uint8_t	primary_ttl	Time to live for primary messages.
uint8_t	pdelay_ttl	Time to live for pdelay messages.
uint16_t	event_udp_port	The port to send event messages.
uint16_t	general_udp_port	The port to send general messages.

◆ ptp_stca_cfg_t

struct ptp_stca_cfg_t

Configuration settings for the STCA.

Data Fields		
ptp_stca_clock_freq_t	clock_freq	Select the clock frequency of the STCA.
ptp_stca_clock_sel_t	clock_sel	Select the input clock to the STCA.
ptp_clock_correction_mode_t	clock_correction_mode	Select the clock correction mode.
uint8_t	gradient_worst10_interval	Select the interval for the gradient worst10 acquisition.
ptp_sync_state_cfg_t	sync_threshold	Configure the synchronization threshold.
ptp_sync_state_cfg_t	sync_loss_threshold	Configure the SYnchronization lost threshold.

◆ ptp_cfg_t

struct ptp_cfg_t

User configuration structure, used in open function

Data Fields

ptp_synfp_cfg_t	synfp
	Configuration settings for the SYNFP.
ptp_stca_cfg_t	stca
	Configuration settings for the STCA.

edmac_instance_t *	p_edmac_instance
	Pointer to PTP edmac instance.
uint16_t	buffer_size
	The maximum Ethernet packet size that can be transmitted or received.
uint8_t **	p_rx_buffers
	Pointer to list of buffers used to receive PTP packets.
uint8_t **	p_tx_buffers
	Pointer to list of buffers used to transmit PTP packets.
IRQn_Type	mint_irq
	Interrupt number for PTP event IRQ.
IRQn_Type	ipls_irq
	Interrupt number for PTP timer IRQ.
uint8_t	mint_ipl
	Interrupt priority of the PTP event IRQ.
uint8_t	ipls_ipl
	Interrupt priority of the PTP timer IRQ.
void(*	p_callback)(ptp_callback_args_t *p_args)

Field Documentation

◆ **p_callback**

void(* ptp_cfg_t::p_callback) (ptp_callback_args_t *p_args)
Callback for handling received PTP events.

◆ **ptp_api_t**

struct ptp_api_t
Timer API structure. General timer functions implemented at the HAL layer follow this API.

Data Fields

fsp_err_t(*	open)(ptp_ctrl_t *const p_ctrl, ptp_cfg_t const *const p_cfg)
fsp_err_t(*	macAddrSet)(ptp_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr)
fsp_err_t(*	ipAddrSet)(ptp_ctrl_t *const p_ctrl, uint32_t ip_addr)
fsp_err_t(*	localClockIdSet)(ptp_ctrl_t *const p_ctrl, uint8_t const *const p_clock_id)
fsp_err_t(*	masterClockIdSet)(ptp_ctrl_t *const p_ctrl, uint8_t const *const p_clock_id, uint16_t port_id)
fsp_err_t(*	messageFlagsSet)(ptp_ctrl_t *const p_ctrl, ptp_message_type_t message_type, ptp_message_flags_t flags)
fsp_err_t(*	currentUtcOffsetSet)(ptp_ctrl_t *const p_ctrl, uint16_t offset)
fsp_err_t(*	portStateSet)(ptp_ctrl_t *const p_ctrl, uint32_t state)
fsp_err_t(*	messageSend)(ptp_ctrl_t *const p_ctrl, ptp_message_t const *const p_message, uint8_t const *const p_tlv_data, uint16_t tlv_data_size)
fsp_err_t(*	localClockValueSet)(ptp_ctrl_t *const p_ctrl, ptp_time_t const *const p_time)
fsp_err_t(*	localClockValueGet)(ptp_ctrl_t *const p_ctrl, ptp_time_t *const p_time)

fsp_err_t(*	pulseTimerCommonConfig)(ptp_ctrl_t *const p_ctrl, ptp_pulse_timer_common_cfg_t *p_timer_cfg)
fsp_err_t(*	pulseTimerEnable)(ptp_ctrl_t *const p_ctrl, uint32_t channel, ptp_pulse_timer_cfg_t *const p_timer_cfg)
fsp_err_t(*	pulseTimerDisable)(ptp_ctrl_t *const p_ctrl, uint32_t channel)
fsp_err_t(*	close)(ptp_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* ptp_api_t::open) (ptp_ctrl_t *const p_ctrl, ptp_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_PTP_Open()

Note

To reconfigure after calling this function, call ptp_api_t::close first.

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ macAddrSet

fsp_err_t(* ptp_api_t::macAddrSet) (ptp_ctrl_t *const p_ctrl, uint8_t const *const p_mac_addr)

Set the MAC address for the PTP.

Implemented as

- R_PTP_MacAddrSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_hw_addr	Pointer to the 6 byte MAC address.

◆ ipAddrSet

fsp_err_t(* ptp_api_t::ipAddrSet) (ptp_ctrl_t *const p_ctrl, uint32_t ip_addr)

Set the IP address for the PTP.

Implemented as

- R_PTP_IpAddrSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	ip_addr	32 bit IPv4 address of the PTP.

◆ localClockIdSet

fsp_err_t(* ptp_api_t::localClockIdSet) (ptp_ctrl_t *const p_ctrl, uint8_t const *const p_clock_id)

Set the local clock ID (Usually based off of the PTP MAC address).

Implemented as

- R_PTP_LocalClockIdSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_clock_id	Pointer to 8 byte clock ID.

◆ masterClockIdSet

fsp_err_t(* ptp_api_t::masterClockIdSet) (ptp_ctrl_t *const p_ctrl, uint8_t const *const p_clock_id, uint16_t port_id)

Set the master clock ID (Usually obtained from previously received announce message).

Implemented as

- R_PTP_MasterClockIdSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_clock_id	Pointer to 8 byte clock ID.
[in]	port_id	The port on the master clock.

◆ messageFlagsSet

fsp_err_t(* ptp_api_t::messageFlagsSet) (ptp_ctrl_t *const p_ctrl, ptp_message_type_t message_type, ptp_message_flags_t flags)

Set the flags field for the given message type.

Implemented as

- R_PTP_MessageFlagsSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	message_type	The message type.
[in]	flags	Flags to set.

◆ currentUtcOffsetSet

fsp_err_t(* ptp_api_t::currentUtcOffsetSet) (ptp_ctrl_t *const p_ctrl, uint16_t offset)

Sets the offsetFromMaster field in announce messages.

Implemented as

- R_PTP_CurrentUtcOffsetSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	offset	New currentUtcOffset value.

◆ portStateSet

fsp_err_t(* ptp_api_t::portStateSet) (ptp_ctrl_t *const p_ctrl, uint32_t state)

Transition to a new clock state.

Implemented as

- R_PTP_PortStateSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	state	The state to transition into.

◆ messageSend

fsp_err_t(* ptp_api_t::messageSend) (ptp_ctrl_t *const p_ctrl, ptp_message_t const *const p_message, uint8_t const *const p_tlv_data, uint16_t tlv_data_size)

Send a PTP message. Appropriate fields in the PTP message will be endian swapped. The application must ensure that the TLV data is in big endian format.

Implemented as

- R_PTP_MessageSend()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_message	Pointer to a PTP message.
[in]	p_tlv_data	Pointer to TLV data that is appended to the end of the PTP message.
[in]	tlv_data_size	Size of the TLV data in bytes.

◆ localClockValueSet

fsp_err_t(* ptp_api_t::localClockValueSet) (ptp_ctrl_t *const p_ctrl, ptp_time_t const *const p_time)

Set the local clock value.

Implemented as

- R_PTP_LocalClockValueSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_time	Pointer to the new time setting.

◆ localClockValueGet

fsp_err_t(* ptp_api_t::localClockValueGet) (ptp_ctrl_t *const p_ctrl, ptp_time_t *const p_time)

Get the local clock value.

Implemented as

- R_PTP_LocalClockValueGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_time	Pointer to store the current time setting.

◆ pulseTimerCommonConfig

fsp_err_t(* ptp_api_t::pulseTimerCommonConfig) (ptp_ctrl_t *const p_ctrl, ptp_pulse_timer_common_cfg_t *p_timer_cfg)

Configuration that is common to all of the pulse timers.

Implemented as

- R_PTP_PulseTimerCommonConfig()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_timer_cfg	Pointer to the pulse timer common configuration.

◆ pulseTimerEnable

fsp_err_t(* ptp_api_t::pulseTimerEnable) (ptp_ctrl_t *const p_ctrl, uint32_t channel, ptp_pulse_timer_cfg_t *const p_timer_cfg)

Setup a pulse timer.

Implemented as

- R_PTP_PulseTimerEnable()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	channel	The pulse timer channel to setup.
[in]	p_timer_cfg	Pointer to the pulse timer configuration.

◆ pulseTimerDisable

fsp_err_t(* ptp_api_t::pulseTimerDisable) (ptp_ctrl_t *const p_ctrl, uint32_t channel)

Stop a pulse timer.

Implemented as

- R_PTP_PulseTimerDisable()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	channel	The pulse timer channel to stop.

◆ close

fsp_err_t(* ptp_api_t::close) (ptp_ctrl_t *const p_ctrl)
Stop PTP operation.

Implemented as

- R_PTP_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ ptp_instance_t

struct ptp_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
ptp_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
ptp_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
ptp_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ ptp_ctrl_field_t

typedef enum PTP_PACKED e_ptp_ctrl_field ptp_ctrl_field_t

The control field for PTP message header.

◆ ptp_message_delay_req_t

typedef ptp_message_sync_t ptp_message_delay_req_t

Delay_req Message Type (0x01).

◆ ptp_message_follow_up_t

typedef ptp_message_sync_t ptp_message_follow_up_t

Follow_up Message Type (0x08).

◆ ptp_message_delay_resp_t

typedef ptp_message_pdelay_resp_t ptp_message_delay_resp_t
Delay_resp Message Type (0x09).

◆ ptp_message_pdelay_resp_follow_up_t

typedef ptp_message_delay_resp_t ptp_message_pdelay_resp_follow_up_t
PDelay_resp_follow_up Message Type (0x0A).

Enumeration Type Documentation

◆ ptp_message_type_t

enum ptp_message_type_t

Standard PTP message types.

Enumerator	
PTP_MESSAGE_TYPE_SYNC	Sync Message Type.
PTP_MESSAGE_TYPE_DELAY_REQ	Delay_req Message Type.
PTP_MESSAGE_TYPE_PDELAY_REQ	PDelay_req Message Type.
PTP_MESSAGE_TYPE_PDELAY_RESP	PDelay_resp Message Type.
PTP_MESSAGE_TYPE_FOLLOW_UP	Follow_up Message Type.
PTP_MESSAGE_TYPE_DELAY_RESP	Delay_resp Message Type.
PTP_MESSAGE_TYPE_PDELAY_RESP_FOLLOW_UP	PDelay_resp_follow_up Message Type.
PTP_MESSAGE_TYPE_ANNOUNCE	Announce Message Type.
PTP_MESSAGE_TYPE_SIGNALING	Signaling Message Type.
PTP_MESSAGE_TYPE_MANAGEMENT	Management Message Type.

◆ ptp_port_state_t

enum ptp_port_state_t	
Possible states that the PTP instance can be in.	
Enumerator	
PTP_PORT_STATE_GENERATE_ANNOUNCE	Generate Announce Messages.
PTP_PORT_STATE_GENERATE_SYNC	Generate Sync Messages.
PTP_PORT_STATE_GENERATE_DELAY_REQ	Generate Delay_req Messages.
PTP_PORT_STATE_GENERATE_PDELAY_REQ	Generate PDelay_req Messages.
PTP_PORT_STATE_RECEIVE_ANNOUNCE	Receive Announce Messages from ptp_cfg_t::p_callback.
PTP_PORT_STATE_RECEIVE_SYNC	Receive Sync Messages from ptp_cfg_t::p_callback.
PTP_PORT_STATE_RECEIVE_FOLLOW_UP	Receive Follow_up Messages from ptp_cfg_t::p_callback.
PTP_PORT_STATE_RECEIVE_DELAY_REQ	Receive Delay_req Messages from ptp_cfg_t::p_callback.
PTP_PORT_STATE_RECEIVE_DELAY_RESP	Receive Delay_resp Messages from ptp_cfg_t::p_callback.
PTP_PORT_STATE_RECEIVE_PDELAY_REQ	Receive PDelay_req Messages from ptp_cfg_t::p_callback.
PTP_PORT_STATE_RECEIVE_PDELAY_RESP	Receive PDelay_resp Messages from ptp_cfg_t::p_callback.
PTP_PORT_STATE_RECEIVE_PDELAY_RESP_FOLLOW_UP	Receive PDelay_resp_follow_up Messages from ptp_cfg_t::p_callback.
PTP_PORT_STATE_RECEIVE_MANAGEMENT	Receive Management Messages from ptp_cfg_t::p_callback.
PTP_PORT_STATE_RECEIVE_SIGNALING	Receive Signaling Messages from ptp_cfg_t::p_callback.
PTP_PORT_STATE_PROCESS_SYNC	Enable Sync Message processing.
PTP_PORT_STATE_PROCESS_FOLLOW_UP	Enable Follow_up Message processing.

PTP_PORT_STATE_PROCESS_DELAY_REQ	Enable Delay_req Message processing.
PTP_PORT_STATE_PROCESS_DELAY_RESP	Enable Delay_resp Message processing.
PTP_PORT_STATE_PROCESS_PDELAY_REQ	Enable PDelay_req Message processing.
PTP_PORT_STATE_PROCESS_PDELAY_RESP	Enable PDelay_resp Message processing.
PTP_PORT_STATE_PROCESS_PDELAY_RESP_FOLLOW_UP	Enable PDelay_resp_follow_up Message processing.
PTP_PORT_STATE_PASSIVE	Configure the PTP instance to only receive Announce, Management, and Signaling Messages.
PTP_PORT_STATE_E2E_MASTER	Configure the PTP instance to operate as a E2E Master.
PTP_PORT_STATE_E2E_SLAVE	Configure the PTP instance to operate as a E2E Slave.
PTP_PORT_STATE_P2P_MASTER	Configure the PTP instance to operate as a P2P Master.
PTP_PORT_STATE_P2P_SLAVE	Configure the PTP instance to operate as a P2P Slave.
PTP_PORT_STATE_RECEIVE_ALL	Configure the PTP instance to receive all messages.
PTP_PORT_STATE_DISABLE	Disable all PTP message generation, processing, and reception.

◆ ptp_clock_delay_mechanism_t

enum ptp_clock_delay_mechanism_t

The mechanism used for delay messages.

Enumerator	
PTP_CLOCK_DELAY_MECHANISM_E2E	End to end delay mechanism.
PTP_CLOCK_DELAY_MECHANISM_P2P	Peer to peer delay mechanism.

◆ ptp_frame_format_t

enum ptp_frame_format_t	
Frame formats that PTP messages can be encapsulated in.	
Enumerator	
PTP_FRAME_FORMAT_ETHERII	Send PTP messages using Ethernet II frames.
PTP_FRAME_FORMAT_IEEE802_3	Send PTP messages using IEEE802_3 frames.
PTP_FRAME_FORMAT_ETHERII_IPV4_UDP	Send PTP messages using Ethernet II frames with an IP and UDP header.
PTP_FRAME_FORMAT_IEEE802_3_IPV4_UDP	Send PTP messages using IEEE802.3 frames with an IP and UDP header.

◆ ptp_frame_filter_mode_t

enum ptp_frame_filter_mode_t	
Filter PTP messages based on destination MAC address. Messages that pass the filter will be transferred to the ETHERC EDMAC.	
Enumerator	
PTP_FRAME_FILTER_MODE_EXT_PROMISCUOUS_MODE	Receive all packets.
PTP_FRAME_FILTER_MODE_UNICAST_MULTICAST	Receive all unicast packets destined for the PTP and all multicast packets.
PTP_FRAME_FILTER_MODE_UNICAST_MULTICAST_FILTERED	Receive Unicast packets destined for the PTP and filter configured multicast packets.
PTP_FRAME_FILTER_MODE_UNICAST	Receive unicast packets destined for the PTP.

◆ ptp_stca_clock_freq_t

enum ptp_stca_clock_freq_t	
STCA input clock frequency.	
Enumerator	
PTP_STCA_CLOCK_FREQ_20MHZ	20 Mhz Input Clock
PTP_STCA_CLOCK_FREQ_25MHZ	25 Mhz Input Clock
PTP_STCA_CLOCK_FREQ_50MHZ	50 Mhz Input Clock
PTP_STCA_CLOCK_FREQ_100MHZ	100 Mhz Input Clock

◆ ptp_stca_clock_sel_t

enum ptp_stca_clock_sel_t	
STCA input clock selection.	
Enumerator	
PTP_STCA_CLOCK_SEL_PCLKA_DIV_1	PCLKA.
PTP_STCA_CLOCK_SEL_PCLKA_DIV_2	PCLKA Divided by 2.
PTP_STCA_CLOCK_SEL_PCLKA_DIV_3	PCLKA Divided by 3.
PTP_STCA_CLOCK_SEL_PCLKA_DIV_4	PCLKA Divided by 4.
PTP_STCA_CLOCK_SEL_PCLKA_DIV_5	PCLKA Divided by 5.
PTP_STCA_CLOCK_SEL_PCLKA_DIV_6	PCLKA Divided by 6.
PTP_STCA_CLOCK_SEL_REF50CK0	50-MHz Reference Signal for timing in RMII mode (STCA clock frequency is 25 Mhz when REF50CK0 is used).

◆ ptp_message_interval_t

enum ptp_message_interval_t	
Message interval for transmitting PTP messages.	
Enumerator	
PTP_MESSAGE_INTERVAL_1_128	1 / 128 seconds
PTP_MESSAGE_INTERVAL_1_64	1 / 64 seconds
PTP_MESSAGE_INTERVAL_1_32	1 / 32 seconds
PTP_MESSAGE_INTERVAL_1_16	1 / 16 seconds
PTP_MESSAGE_INTERVAL_1_8	1 / 8 seconds
PTP_MESSAGE_INTERVAL_1_4	1 / 4 seconds
PTP_MESSAGE_INTERVAL_1_2	1 / 2 seconds
PTP_MESSAGE_INTERVAL_1	1 seconds
PTP_MESSAGE_INTERVAL_2	2 seconds
PTP_MESSAGE_INTERVAL_4	4 seconds
PTP_MESSAGE_INTERVAL_8	8 seconds
PTP_MESSAGE_INTERVAL_16	16 seconds
PTP_MESSAGE_INTERVAL_32	32 seconds
PTP_MESSAGE_INTERVAL_64	64 seconds

◆ ptp_clock_correction_mode_t

enum ptp_clock_correction_mode_t	
Clock correction mode.	
Enumerator	
PTP_CLOCK_CORRECTION_MODE1	Correct the local clock using the current offsetFromMaster value.
PTP_CLOCK_CORRECTION_MODE2	Correct the local clock using the calculated clock gradient.

◆ ptp_event_t

enum ptp_event_t	
PTP events provided by ptp_cfg_t::p_callback.	
Enumerator	
PTP_EVENT_SYNC_ACQUIRED	The local clock is synchronized to the master clock.
PTP_EVENT_SYNC_LOST	The local clock is not synchronized to the master clcok.
PTP_EVENT_SYNC_MESSAGE_TIMEOUT	A sync message has not been received for the configured time.
PTP_EVENT_WORST10_ACQUIRED	Gradient worst10 values has been calcualted.
PTP_EVENT_OFFSET_FROM_MASTER_UPDATED	The offset from the master clock has been updated.
PTP_EVENT_LOG_MESSAGE_INT_CHANGED	The message interval was changed.
PTP_EVENT_MEAN_PATH_DELAY_UPDATED	The mean path delay has been updated.
PTP_EVENT_DELAY_RESP_TIMEOUT	A delay_resp has not been received for the configured time.
PTP_EVENT_LOG_MESSAGE_INT_OUT_OF_RANGE	The updated message interval is out of range.
PTP_EVENT_DELAY_REQ_FIFO_OVERFLOW	The FIFO buffer for storing information from received Delay_Req messages holds 32 or more entries.
PTP_EVENT_LOOP_RECEPTION_DETECTED	A packet with the same sourcePortIdendity as the local clock was received.
PTP_EVENT_CTRL_INFO_ABNORMALITY	A malformed frame was received (EDMAC, ETHERC, and EPTPC must be reset).
PTP_EVENT_DELAY_RESP_PROCESSING_HALTED	Processing of delay_resp messages has been halted.
PTP_EVENT_MESSAGE_GENERATION_HALTED	Generation of messages has been halted.
PTP_EVENT_MESSAGE_RECEIVED	A PTP message was received from the EDMAC.
PTP_EVENT_MESSAGE_TRANSMIT_COMPLETE	A PTP message has been transmitted.

PTP_EVENT_PULSE_TIMER_MINT_RISING_EDGE	A rising edge occurred on a pulse timer channel.
PTP_EVENT_PULSE_TIMER_IPLS_COMMON	A rising or falling edge occurred on any pulse timer channel.

◆ ptp_ethernet_phy_interface_t

enum ptp_ethernet_phy_interface_t

The Ethernet PHY interface type.

Enumerator	
PTP_ETHERNET_PHY_INTERFACE_MII	Media-independant interface.
PTP_ETHERNET_PHY_INTERFACE_RMII	Reduced media-independant interface.

Variable Documentation

◆ e_ptp_ctrl_field

enum PTP_PACKED e_ptp_ctrl_field

The control field for PTP message header.

4.3.31 RTC Interface

Interfaces

Detailed Description

Interface for accessing the Realtime Clock.

Summary

The RTC Interface is for configuring Real Time Clock (RTC) functionality including alarm, periodic notiification and error adjustment.

The Real Time Clock Interface can be implemented by:

- Realtime Clock (r_rtc)

Data Structures

struct	rtc_callback_args_t

struct	rtc_error_adjustment_cfg_t
struct	rtc_alarm_time_t
struct	rtc_info_t
struct	rtc_cfg_t
struct	rtc_api_t
struct	rtc_instance_t

Typedefs

typedef struct tm	rtc_time_t
typedef void	rtc_ctrl_t

Enumerations

enum	rtc_event_t
enum	rtc_clock_source_t
enum	rtc_status_t
enum	rtc_error_adjustment_t
enum	rtc_error_adjustment_mode_t
enum	rtc_error_adjustment_period_t
enum	rtc_periodic_irq_select_t

Data Structure Documentation

◆ rtc_callback_args_t

struct rtc_callback_args_t

Callback function parameter data

Data Fields

rtc_event_t	event	The event can be used to identify what caused the callback (compare match or error).
void const *	p_context	Placeholder for user data.

◆ rtc_error_adjustment_cfg_t

struct rtc_error_adjustment_cfg_t		
Time error adjustment value configuration		
Data Fields		
rtc_error_adjustment_mode_t	adjustment_mode	Automatic Adjustment Enable/Disable.
rtc_error_adjustment_period_t	adjustment_period	Error Adjustment period.
rtc_error_adjustment_t	adjustment_type	Time error adjustment setting.
uint32_t	adjustment_value	Value of the prescaler for error adjustment.

◆ rtc_alarm_time_t

struct rtc_alarm_time_t		
Alarm time setting structure		
Data Fields		
rtc_time_t	time	Time structure.
bool	sec_match	Enable the alarm based on a match of the seconds field.
bool	min_match	Enable the alarm based on a match of the minutes field.
bool	hour_match	Enable the alarm based on a match of the hours field.
bool	mday_match	Enable the alarm based on a match of the days field.
bool	mon_match	Enable the alarm based on a match of the months field.
bool	year_match	Enable the alarm based on a match of the years field.
bool	dayofweek_match	Enable the alarm based on a match of the dayofweek field.

◆ rtc_info_t

struct rtc_info_t		
RTC Information Structure for information returned by infoGet()		
Data Fields		
rtc_clock_source_t	clock_source	Clock source for the RTC block.
rtc_status_t	status	RTC run status.

◆ rtc_cfg_t

struct rtc_cfg_t
User configuration structure, used in open function

Data Fields	
rtc_clock_source_t	clock_source
	Clock source for the RTC block.
uint32_t	freq_compare_value_loco
	The frequency comparison value for LOCO.
rtc_error_adjustment_cfg_t const *const	p_err_cfg
	Pointer to Error Adjustment configuration.
uint8_t	alarm_ipl
	Alarm interrupt priority.
IRQn_Type	alarm_irq
	Alarm interrupt vector.
uint8_t	periodic_ipl
	Periodic interrupt priority.
IRQn_Type	periodic_irq
	Periodic interrupt vector.
uint8_t	carry_ipl
	Carry interrupt priority.
IRQn_Type	carry_irq
	Carry interrupt vector.

void(*	p_callback)(rtc_callback_args_t *p_args)
	Called from the ISR.
void const *	p_context
	User defined context passed into callback function.
void const *	p_extend
	RTC hardware dependant configuration.

◆ **rtc_api_t**

struct rtc_api_t

RTC driver structure. General RTC functions implemented at the HAL layer follow this API.

Data Fields

fsp_err_t(*	open)(rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)
fsp_err_t(*	close)(rtc_ctrl_t *const p_ctrl)
fsp_err_t(*	calendarTimeSet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)
fsp_err_t(*	calendarTimeGet)(rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)
fsp_err_t(*	calendarAlarmSet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)
fsp_err_t(*	calendarAlarmGet)(rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)
fsp_err_t(*	periodicIrqRateSet)(rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate)
fsp_err_t(*	errorAdjustmentSet)(rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg)

fsp_err_t(*	callbackSet)(rtc_ctrl_t *const p_ctrl, void(*p_callback)(rtc_callback_args_t *), void const *const p_context, rtc_callback_args_t *const p_callback_memory)
fsp_err_t(*	infoGet)(rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info)

Field Documentation

◆ open

fsp_err_t(* rtc_api_t::open) (rtc_ctrl_t *const p_ctrl, rtc_cfg_t const *const p_cfg)

Open the RTC driver.

Implemented as

- R_RTC_Open()

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_cfg	Pointer to the configuration structure

◆ close

fsp_err_t(* rtc_api_t::close) (rtc_ctrl_t *const p_ctrl)

Close the RTC driver.

Implemented as

- R_RTC_Close()

Parameters

[in]	p_ctrl	Pointer to RTC device handle.

◆ calendarTimeSet

fsp_err_t(* rtc_api_t::calendarTimeSet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)

Set the calendar time and start the calender counter

Implemented as

- R_RTC_CalendarTimeSet()

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_time	Pointer to a time structure that contains the time to set
[in]	clock_start	Flag that starts the clock right after it is set

◆ calendarTimeGet

fsp_err_t(* rtc_api_t::calendarTimeGet) (rtc_ctrl_t *const p_ctrl, rtc_time_t *const p_time)

Get the calendar time.

Implemented as

- R_RTC_CalendarTimeGet()

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[out]	p_time	Pointer to a time structure that contains the time to get

◆ calendarAlarmSet

fsp_err_t(* rtc_api_t::calendarAlarmSet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)

Set the calendar alarm time and enable the alarm interrupt.

Implemented as

- R_RTC_CalendarAlarmSet()

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	p_alarm	Pointer to an alarm structure that contains the alarm time to set
[in]	irq_enable_flag	Enable the ALARM irq if set

◆ calendarAlarmGet

fsp_err_t(* rtc_api_t::calendarAlarmGet) (rtc_ctrl_t *const p_ctrl, rtc_alarm_time_t *const p_alarm)

Get the calendar alarm time.

Implemented as

- R_RTC_CalendarAlarmGet()

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[out]	p_alarm	Pointer to an alarm structure to fill up with the alarm time

◆ periodicIrqRateSet

fsp_err_t(* rtc_api_t::periodicIrqRateSet) (rtc_ctrl_t *const p_ctrl, rtc_periodic_irq_select_t const rate)

Set the periodic irq rate

Implemented as

- R_RTC_PeriodicIrqRateSet()

Parameters

[in]	p_ctrl	Pointer to RTC device handle
[in]	rate	Rate of periodic interrupts

◆ errorAdjustmentSet

fsp_err_t(* rtc_api_t::errorAdjustmentSet) (rtc_ctrl_t *const p_ctrl, rtc_error_adjustment_cfg_t const *const err_adj_cfg)

Set time error adjustment.

Implemented as

- R_RTC_ErrorAdjustmentSet()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	err_adj_cfg	Pointer to the Error Adjustment Config

◆ callbackSet

fsp_err_t(* rtc_api_t::callbackSet) (rtc_ctrl_t *const p_ctrl, void(*p_callback)(rtc_callback_args_t *), void const *const p_context, rtc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_RTC_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the RTC control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated

◆ infoGet

fsp_err_t(* rtc_api_t::infoGet) (rtc_ctrl_t *const p_ctrl, rtc_info_t *const p_rtc_info)

Return the currently configure clock source for the RTC

Implemented as

- R_RTC_InfoGet()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[out]	p_rtc_info	Pointer to RTC information structure

◆ rtc_instance_t

struct rtc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
rtc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
rtc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
rtc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ rtc_time_t

typedef struct tm rtc_time_t
Date and time structure defined in C standard library <time.h>

◆ rtc_ctrl_t

typedef void rtc_ctrl_t
RTC control block. Allocate an instance specific control block to pass into the RTC API calls. **Implemented as** ◦ rtc_instance_ctrl_t

Enumeration Type Documentation

◆ rtc_event_t

enum rtc_event_t

Events that can trigger a callback function

Enumerator	
RTC_EVENT_ALARM_IRQ	Real Time Clock ALARM IRQ.
RTC_EVENT_PERIODIC_IRQ	Real Time Clock PERIODIC IRQ.

◆ rtc_clock_source_t

enum rtc_clock_source_t

Clock source for the RTC block

Enumerator	
RTC_CLOCK_SOURCE_SUBCLK	Sub-clock oscillator.
RTC_CLOCK_SOURCE_LOCO	Low power On Chip Oscillator.

◆ rtc_status_t

enum rtc_status_t	
RTC run state	
Enumerator	
RTC_STATUS_STOPPED	RTC counter is stopped.
RTC_STATUS_RUNNING	RTC counter is running.

◆ rtc_error_adjustment_t

enum rtc_error_adjustment_t	
Time error adjustment settings	
Enumerator	
RTC_ERROR_ADJUSTMENT_NONE	Adjustment is not performed.
RTC_ERROR_ADJUSTMENT_ADD_PRESCALER	Adjustment is performed by the addition to the prescaler.
RTC_ERROR_ADJUSTMENT_SUBTRACT_PRESCALER	Adjustment is performed by the subtraction from the prescaler.

◆ rtc_error_adjustment_mode_t

enum rtc_error_adjustment_mode_t	
Time error adjustment mode settings	
Enumerator	
RTC_ERROR_ADJUSTMENT_MODE_MANUAL	Adjustment mode is set to manual.
RTC_ERROR_ADJUSTMENT_MODE_AUTOMATIC	Adjustment mode is set to automatic.

◆ rtc_error_adjustment_period_t

enum rtc_error_adjustment_period_t	
Time error adjustment period settings	
Enumerator	
RTC_ERROR_ADJUSTMENT_PERIOD_1_MINUTE	Adjustment period is set to every one minute.
RTC_ERROR_ADJUSTMENT_PERIOD_10_SECOND	Adjustment period is set to every ten second.
RTC_ERROR_ADJUSTMENT_PERIOD_NONE	Adjustment period not supported in manual mode.

◆ rtc_periodic_irq_select_t

enum rtc_periodic_irq_select_t	
Periodic Interrupt select	
Enumerator	
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_256_SECOND	A periodic irq is generated every 1/256 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_128_SECOND	A periodic irq is generated every 1/128 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_64_SECOND	A periodic irq is generated every 1/64 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_32_SECOND	A periodic irq is generated every 1/32 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_16_SECOND	A periodic irq is generated every 1/16 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_8_SECOND	A periodic irq is generated every 1/8 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_4_SECOND	A periodic irq is generated every 1/4 second.
RTC_PERIODIC_IRQ_SELECT_1_DIV_BY_2_SECOND	A periodic irq is generated every 1/2 second.
RTC_PERIODIC_IRQ_SELECT_1_SECOND	A periodic irq is generated every 1 second.
RTC_PERIODIC_IRQ_SELECT_2_SECOND	A periodic irq is generated every 2 seconds.

4.3.32 SD/MMC Interface

Interfaces

Detailed Description

Interface for accessing SD, eMMC, and SDIO devices.

Summary

The r_sdhi interface provides standard SD and eMMC media functionality. This interface also supports SDIO.

The SD/MMC interface is implemented by:

- SD/MMC Host Interface (r_sdhi)

Data Structures

struct	sdmmc_status_t
struct	sdmmc_device_t
struct	sdmmc_callback_args_t
struct	sdmmc_cfg_t
struct	sdmmc_api_t
struct	sdmmc_instance_t

Typedefs

typedef void	sdmmc_ctrl_t

Enumerations

enum	sdmmc_card_type_t
enum	sdmmc_bus_width_t
enum	sdmmc_io_transfer_mode_t
enum	sdmmc_io_address_mode_t
enum	sdmmc_io_write_mode_t
enum	sdmmc_event_t
enum	sdmmc_card_detect_t

enum	sdmmc_write_protect_t
enum	sdmmc_r1_state_t

Data Structure Documentation

◆ sdmmc_status_t

struct sdmmc_status_t

Current status.

Data Fields

bool	initialized	False if card was removed (only applies if MCU supports card detection and SDnCD pin is connected), true otherwise. If ready is false, call sdmmc_api_t::mediaInit to reinitialize it
bool	transfer_in_progress	true = Card is busy
bool	card_inserted	Card detect status, true if card detect is not used.

◆ sdmmc_device_t

struct sdmmc_device_t

Information obtained from the media device.

Data Fields

sdmmc_card_type_t	card_type	SD, eMMC, or SDIO.
bool	write_protected	true = Card is write protected
uint32_t	clock_rate	Current clock rate.
uint32_t	sector_count	Sector count.
uint32_t	sector_size_bytes	Sector size.
uint32_t	erase_sector_count	Minimum erasable unit (in 512 byte sectors)

◆ sdmmc_callback_args_t

struct sdmmc_callback_args_t

Callback function parameter data

Data Fields

sdmmc_event_t	event	The event can be used to identify what caused the callback.

sdmmc_response_t	response	Response from card, only valid if SDMMC_EVENT_RESPONSE is set in event.
void const *	p_context	Placeholder for user data.

◆ sdmmc_cfg_t

struct sdmmc_cfg_t	
SD/MMC Configuration	
Data Fields	
uint8_t	channel
	Channel of SD/MMC host interface.
sdmmc_bus_width_t	bus_width
	Device bus width is 1, 4 or 8 bits wide.
transfer_instance_t const *	p_lower_lvl_transfer
	Transfer instance used to move data with DMA or DTC.
void(*	p_callback)(sdmmc_callback_args_t *p_args)
	Pointer to callback function.
void const *	p_context
	User defined context passed into callback function.
void const *	p_extend
	SD/MMC hardware dependent configuration.
uint32_t	block_size
sdmmc_card_detect_t	card_detect

sdmmc_write_protect_t	write_protect
IRQn_Type	access_irq
	Access IRQ number.
IRQn_Type	sdio_irq
	SDIO IRQ number.
IRQn_Type	card_irq
	Card IRQ number.
IRQn_Type	dma_req_irq
	DMA request IRQ number.
uint8_t	access_ipl
	Access interrupt priority.
uint8_t	sdio_ipl
	SDIO interrupt priority.
uint8_t	card_ipl
	Card interrupt priority.
uint8_t	dma_req_ipl
	DMA request interrupt priority.

Field Documentation

◆ block_size

uint32_t sdmmc_cfg_t::block_size

Block size in bytes. Block size must be 512 bytes for SD cards and eMMC devices. Block size can be 1-512 bytes for SDIO.

◆ card_detect

sdmmc_card_detect_t sdmmc_cfg_t::card_detect

Whether or not card detection is used.

◆ write_protect

sdmmc_write_protect_t sdmmc_cfg_t::write_protect

Select whether or not to use the write protect pin. Select Not Used if the MCU or device does not have a write protect pin.

◆ sdmmc_api_t

struct sdmmc_api_t

SD/MMC functions implemented at the HAL layer API.

Data Fields

fsp_err_t(*	open)(sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)
fsp_err_t(*	mediaInit)(sdmmc_ctrl_t *const p_ctrl, sdmmc_device_t *const p_device)
fsp_err_t(*	read)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t(*	write)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t(*	readIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)
fsp_err_t(*	writeIo)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)
fsp_err_t(*	readIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode,

	sdmmc_io_address_mode_t address_mode)
fsp_err_t(*	writeIoExt)(sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)
fsp_err_t(*	ioIntEnable)(sdmmc_ctrl_t *const p_ctrl, bool enable)
fsp_err_t(*	statusGet)(sdmmc_ctrl_t *const p_ctrl, sdmmc_status_t *const p_status)
fsp_err_t(*	erase)(sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)
fsp_err_t(*	callbackSet)(sdmmc_ctrl_t *const p_api_ctrl, void(*p_callback)(sdmmc_callback_args_t *), void const *const p_context, sdmmc_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(sdmmc_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* sdmmc_api_t::open) (sdmmc_ctrl_t *const p_ctrl, sdmmc_cfg_t const *const p_cfg)

Open the SD/MMC driver.

Implemented as

- R_SDHI_Open()

Parameters

[in]	p_ctrl	Pointer to SD/MMC instance control block.
[in]	p_cfg	Pointer to SD/MMC instance configuration structure.

◆ mediaInit

fsp_err_t(* sdmmc_api_t::mediaInit) (sdmmc_ctrl_t *const p_ctrl, sdmmc_device_t *const p_device)

Initializes an SD/MMC device. If the device is a card, the card must be plugged in prior to calling this API. This API blocks until the device initialization procedure is complete.

Implemented as

- R_SDHI_MediaInit()

Parameters

[in]	p_ctrl	Pointer to SD/MMC instance control block.
[out]	p_device	Pointer to store device information.

◆ read

fsp_err_t(* sdmmc_api_t::read) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const start_sector, uint32_t const sector_count)

Read data from an SD/MMC channel. This API is not supported for SDIO devices.

Implemented as

- R_SDHI_Read()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_dest	Pointer to data buffer to read data to.
[in]	start_sector	First sector address to read.
[in]	sector_count	Number of sectors to read. All sectors must be in the range of sdmmc_device_t::sector_count.

◆ write

fsp_err_t(* sdmmc_api_t::write) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const start_sector, uint32_t const sector_count)

Write data to SD/MMC channel. This API is not supported for SDIO devices.

Implemented as

- R_SDHI_Write()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	p_source	Pointer to data buffer to write data from.
[in]	start_sector	First sector address to write to.
[in]	sector_count	Number of sectors to write. All sectors must be in the range of sdmmc_device_t::sector_count.

◆ readIo

fsp_err_t(* sdmmc_api_t::readIo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address)

Read one byte of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented as

- R_SDHI_ReadIo()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_data	Pointer to location to store data byte.
[in]	function	SDIO Function Number.
[in]	address	SDIO register address.

◆ writeIo

fsp_err_t(* sdmmc_api_t::writeIo) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_data, uint32_t const function, uint32_t const address, sdmmc_io_write_mode_t const read_after_write)

Write one byte of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented as

- R_SDHI_WriteIo()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in,out]	p_data	Pointer to data byte to write. Read data is also provided here if read_after_write is true.
[in]	function	SDIO Function Number.
[in]	address	SDIO register address.
[in]	read_after_write	Whether or not to read back the same register after writing

◆ readIoExt

fsp_err_t(* sdmmc_api_t::readIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const function, uint32_t const address, uint32_t *const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)

Read multiple bytes or blocks of I/O data from an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented as

- R_SDHI_ReadIoExt()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_dest	Pointer to data buffer to read data to.
[in]	function	SDIO Function Number.
[in]	address	SDIO register address.
[in]	count	Number of bytes or blocks to read, maximum 512 bytes or 511 blocks.
[in]	transfer_mode	Byte or block mode
[in]	address_mode	Fixed or incrementing address mode

◆ writeIoExt

fsp_err_t(* sdmmc_api_t::writeIoExt) (sdmmc_ctrl_t *const p_ctrl, uint8_t const *const p_source, uint32_t const function, uint32_t const address, uint32_t const count, sdmmc_io_transfer_mode_t transfer_mode, sdmmc_io_address_mode_t address_mode)

Write multiple bytes or blocks of I/O data to an SDIO device. This API is not supported for SD or eMMC memory devices.

Implemented as

- R_SDHI_WriteIoExt()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	p_source	Pointer to data buffer to write data from.
[in]	function_number	SDIO Function Number.
[in]	address	SDIO register address.
[in]	count	Number of bytes or blocks to write, maximum 512 bytes or 511 blocks.
[in]	transfer_mode	Byte or block mode
[in]	address_mode	Fixed or incrementing address mode

◆ ioIntEnable

fsp_err_t(* sdmmc_api_t::ioIntEnable) (sdmmc_ctrl_t *const p_ctrl, bool enable)

Enables SDIO interrupt for SD/MMC instance. This API is not supported for SD or eMMC memory devices.

Implemented as

- R_SDHI_IoIntEnable

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	enable	Interrupt enable = true, interrupt disable = false.

◆ statusGet

fsp_err_t(* sdmmc_api_t::statusGet) (sdmmc_ctrl_t *const p_ctrl, sdmmc_status_t *const p_status)

Get SD/MMC device status.

Implemented as

- R_SDHI_StatusGet()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[out]	p_status	Pointer to current driver status.

◆ erase

fsp_err_t(* sdmmc_api_t::erase) (sdmmc_ctrl_t *const p_ctrl, uint32_t const start_sector, uint32_t const sector_count)

Erase SD/MMC sectors. The sector size for erase is fixed at 512 bytes. This API is not supported for SDIO devices.

Implemented as

- R_SDHI_Erase

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.
[in]	start_sector	First sector to erase. Must be a multiple of sdmmc_device_t::erase_sector_count.
[in]	sector_count	Number of sectors to erase. Must be a multiple of sdmmc_device_t::erase_sector_count. All sectors must be in the range of sdmmc_device_t::sector_count.

◆ callbackSet

fsp_err_t(* sdmmc_api_t::callbackSet) (sdmmc_ctrl_t *const p_api_ctrl, void(*p_callback)(sdmmc_callback_args_t *), void const *const p_context, sdmmc_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_SDHI_CallbackSet()

Parameters

[in]	p_ctrl	Control block set in sdmmc_api_t::open call.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* sdmmc_api_t::close) (sdmmc_ctrl_t *const p_ctrl)

Close open SD/MMC device.

Implemented as

- R_SDHI_Close()

Parameters

[in]	p_ctrl	Pointer to an open SD/MMC instance control block.

◆ sdmmc_instance_t

struct sdmmc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
sdmmc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
sdmmc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
sdmmc_api_t const *	p_api	Pointer to the API structure for

		this instance.

Typedef Documentation

◆ sdmmc_ctrl_t

typedef void sdmmc_ctrl_t
SD/MMC control block. Allocate an instance specific control block to pass into the SD/MMC API calls. **Implemented as** ◦ sdmmc_instance_ctrl_t

Enumeration Type Documentation

◆ sdmmc_card_type_t

enum sdmmc_card_type_t

SD/MMC media uses SD protocol or MMC protocol.

Enumerator	
SDMMC_CARD_TYPE_MMC	The media is an eMMC device.
SDMMC_CARD_TYPE_SD	The media is an SD card.
SDMMC_CARD_TYPE_SDIO	The media is an SDIO card.

◆ sdmmc_bus_width_t

enum sdmmc_bus_width_t

SD/MMC data bus is 1, 4 or 8 bits wide.

Enumerator	
SDMMC_BUS_WIDTH_1_BIT	Data bus is 1 bit wide.
SDMMC_BUS_WIDTH_4_BITS	Data bus is 4 bits wide.
SDMMC_BUS_WIDTH_8_BITS	Data bus is 8 bits wide.

◆ sdmmc_io_transfer_mode_t

enum sdmmc_io_transfer_mode_t	
SDIO transfer mode, configurable in SDIO read/write extended commands.	
Enumerator	
SDMMC_IO_MODE_TRANSFER_BYTE	SDIO byte transfer mode.
SDMMC_IO_MODE_TRANSFER_BLOCK	SDIO block transfer mode.

◆ sdmmc_io_address_mode_t

enum sdmmc_io_address_mode_t	
SDIO address mode, configurable in SDIO read/write extended commands.	
Enumerator	
SDMMC_IO_ADDRESS_MODE_FIXED	Write all data to the same address.
SDMMC_IO_ADDRESS_MODE_INCREMENT	Increment destination address after each write.

◆ sdmmc_io_write_mode_t

enum sdmmc_io_write_mode_t	
Controls the RAW (read after write) flag of CMD52. Used to read back the status after writing a control register.	
Enumerator	
SDMMC_IO_WRITE_MODE_NO_READ	Write only (do not read back)
SDMMC_IO_WRITE_READ_AFTER_WRITE	Read back the register after write.

◆ sdmmc_event_t

enum sdmmc_event_t	
Events that can trigger a callback function	
Enumerator	
SDMMC_EVENT_CARD_REMOVED	Card removed event.
SDMMC_EVENT_CARD_INSERTED	Card inserted event.
SDMMC_EVENT_RESPONSE	Response event.
SDMMC_EVENT_SDIO	IO event.
SDMMC_EVENT_TRANSFER_COMPLETE	Read or write complete.
SDMMC_EVENT_TRANSFER_ERROR	Read or write failed.
SDMMC_EVENT_ERASE_COMPLETE	Erase completed.
SDMMC_EVENT_ERASE_BUSY	Erase timeout, poll sdmmc_api_t::statusGet.

◆ sdmmc_card_detect_t

enum sdmmc_card_detect_t	
Card detection configuration options.	
Enumerator	
SDMMC_CARD_DETECT_NONE	Card detection unused.
SDMMC_CARD_DETECT_CD	Card detection using the CD pin.

◆ sdmmc_write_protect_t

enum sdmmc_write_protect_t	
Write protection configuration options.	
Enumerator	
SDMMC_WRITE_PROTECT_NONE	Write protection unused.
SDMMC_WRITE_PROTECT_WP	Write protection using WP pin.

◆ sdmmc_r1_state_t

enum sdmmc_r1_state_t	
Card state when receiving the prior command.	
Enumerator	
SDMMC_R1_STATE_IDLE	Idle State.
SDMMC_R1_STATE_READY	Ready State.
SDMMC_R1_STATE_IDENT	Identification State.
SDMMC_R1_STATE_STBY	Stand-by State.
SDMMC_R1_STATE_TRAN	Transfer State.
SDMMC_R1_STATE_DATA	Sending-data State.
SDMMC_R1_STATE_RCV	Receive-data State.
SDMMC_R1_STATE_PRG	Programming State.
SDMMC_R1_STATE_DIS	Disconnect State (between programming and stand-by)
SDMMC_R1_STATE_IO	This is an I/O card and memory states do not apply.

4.3.33 SLCDC Interface

Interfaces

Detailed Description

Interface for Segment LCD controllers.

Data Structures

struct	slcdc_cfg_t
struct	slcdc_api_t
struct	slcdc_instance_t

Typedefs

typedef void	slcdc_ctrl_t

Enumerations

enum	slcdc_bias_method_t
enum	slcdc_time_slice_t
enum	slcdc_waveform_t
enum	slcdc_drive_volt_gen_t
enum	slcdc_display_area_control_blink_t
enum	slcdc_display_area_t
enum	slcdc_contrast_t
enum	slcdc_display_on_off_t
enum	slcdc_display_enable_disable_t
enum	slcdc_display_clock_t
enum	slcdc_clk_div_t

Data Structure Documentation

◆ slcdc_cfg_t

struct slcdc_cfg_t

SLCDC configuration block

Data Fields		
slcdc_display_clock_t	slcdc_clock	LCD clock source (LCDSCKSEL)
slcdc_clk_div_t	slcdc_clock_setting	LCD clock setting (LCDC0)
slcdc_bias_method_t	bias_method	LCD display bias method select (LBAS bit)
slcdc_time_slice_t	time_slice	Time slice of LCD display select (LDTY bit)
slcdc_waveform_t	waveform	LCD display waveform select (LWAVE bit)
slcdc_drive_volt_gen_t	drive_volt_gen	LCD Drive Voltage Generator Select (MDSET bit)
slcdc_contrast_t	contrast	LCD Boost Level (contrast setting)

◆ slcdc_api_t

struct slcdc_api_t
SLCDC functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg)
fsp_err_t(*	write)(slcdc_ctrl_t *const p_ctrl, uint8_t const start_segment, uint8_t const *p_data, uint8_t const segment_count)
fsp_err_t(*	modify)(slcdc_ctrl_t *const p_ctrl, uint8_t const segment, uint8_t const data_mask, uint8_t const data)
fsp_err_t(*	start)(slcdc_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(slcdc_ctrl_t *const p_ctrl)
fsp_err_t(*	setContrast)(slcdc_ctrl_t *const p_ctrl, slcdc_contrast_t const contrast)
fsp_err_t(*	setDisplayArea)(slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area)
fsp_err_t(*	close)(slcdc_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* slcdc_api_t::open) (slcdc_ctrl_t *const p_ctrl, slcdc_cfg_t const *const p_cfg)

Open SLCDC.

Implemented as

- R_SLCDC_Open()

Parameters

[in,out]	p_ctrl	Pointer to display interface control block. Must be declared by user.
[in]	p_cfg	Pointer to display configuration structure. All elements of this structure must be set by the user.

◆ write

fsp_err_t(* slcdc_api_t::write) (slcdc_ctrl_t *const p_ctrl, uint8_t const start_segment, uint8_t const *p_data, uint8_t const segment_count)

Write data to the SLCDC segment data array. Specifies the initial display data. Except when using 8-time slice mode, store values in the lower 4 bits when writing to the A-pattern area and in the upper 4 bits when writing to the B-pattern area.

Implemented as

- R_SLCDC_Write()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	start_segment	Specify the start segment number to be written.
[in]	p_data	Pointer to the display data to be written to the specified segments.
[in]	segment_count	Number of segments to be written.

◆ modify

fsp_err_t(* slcdc_api_t::modify) (slcdc_ctrl_t *const p_ctrl, uint8_t const segment, uint8_t const data_mask, uint8_t const data)

Rewrite data in the SLCDC segment data array. Rewrites the LCD display data in 1-bit units. If a bit is not specified for rewriting, the value stored in the bit is held as it is.

Implemented as

- R_SLCDC_Modify()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	segment	The segment to be written.
[in]	data_mask	Mask the data being displayed. Set 0 to the bit to be rewritten and set 1 to the other bits. Multiple bits can be rewritten.
[in]	data	Specify display data to rewrite to the specified segment.

◆ start

fsp_err_t(* slcdc_api_t::start) (slcdc_ctrl_t *const p_ctrl)

Enable display signal output. Displays the segment data on the LCD.

Implemented as

- R_SLCDC_Start()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ stop

fsp_err_t(* slcdc_api_t::stop) (slcdc_ctrl_t *const p_ctrl)

Disable display signal output. Stops displaying data on the LCD.

Implemented as

- R_SLCDC_Stop()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ **setContrast**

fsp_err_t(* slcdc_api_t::setContrast) (slcdc_ctrl_t *const p_ctrl, slcdc_contrast_t const contrast)

Set the display contrast. This function can be used only when the internal voltage boosting method is used for drive voltage generation.

Implemented as

- R_SLCDC_SetContrast()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ **setDisplayArea**

fsp_err_t(* slcdc_api_t::setDisplayArea) (slcdc_ctrl_t *const p_ctrl, slcdc_display_area_t const display_area)

Set LCD display area. This function sets a specified display area, A-pattern or B-pattern. This function can be used to 'blink' the display between A-pattern and B-pattern area data.

When using blinking, the RTC is required to operate before this function is executed. To configure the RTC, follow the steps below. 1) Open RTC 2) Set Periodic IRQ 3) Start RTC counter 4) Enable IRQ, RTC_EVENT_PERIODIC_IRQ Refer to the User's Manual for the detailed procedure.

Implemented as

- R_SLCDC_SetDisplayArea()

Parameters

[in]	p_ctrl	Pointer to display interface control block.
[in]	display_area	Display area to be used, A-pattern or B-pattern area.

◆ **close**

fsp_err_t(* slcdc_api_t::close) (slcdc_ctrl_t *const p_ctrl)

Close SLCDC.

Implemented as

- R_SLCDC_Close()

Parameters

[in]	p_ctrl	Pointer to display interface control block.

◆ **slcdc_instance_t**

struct slcdc_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
slcdc_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
slcdc_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
slcdc_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ slcdc_ctrl_t

typedef void slcdc_ctrl_t

SLCDC control block. Allocate an instance specific control block to pass into the SLCDC API calls.

Implemented as

- slcdc_instance_ctrl_tSLCDC control block

Enumeration Type Documentation

◆ slcdc_bias_method_t

enum slcdc_bias_method_t

LCD display bias method.

Enumerator	
SLCDC_BIAS_2	1/2 bias method
SLCDC_BIAS_3	1/3 bias method
SLCDC_BIAS_4	1/4 bias method

◆ slcdc_time_slice_t

enum slcdc_time_slice_t	
Time slice of LCD display.	
Enumerator	
SLCDC_STATIC	Static.
SLCDC_SLICE_2	2-time slice
SLCDC_SLICE_3	3-time slice
SLCDC_SLICE_4	4-time slice
SLCDC_SLICE_8	8-time slice

◆ slcdc_waveform_t

enum slcdc_waveform_t	
LCD display waveform select.	
Enumerator	
SLCDC_WAVE_A	Waveform A.
SLCDC_WAVE_B	Waveform B.

◆ slcdc_drive_volt_gen_t

enum slcdc_drive_volt_gen_t	
LCD Drive Voltage Generator Select.	
Enumerator	
SLCDC_VOLT_EXTERNAL	External resistance division method.
SLCDC_VOLT_INTERNAL	Internal voltage boosting method.
SLCDC_VOLT_CAPACITOR	Capacitor split method.

◆ slcdc_display_area_control_blink_t

enum slcdc_display_area_control_blink_t	
Display Data Area Control	
Enumerator	
SLCDC_NOT_BLINKING	Display either A-pattern or B-pattern data.
SLCDC_BLINKING	Alternately display A-pattern and B-pattern data.

◆ slcdc_display_area_t

enum slcdc_display_area_t	
Display Area data	
Enumerator	
SLCDC_DISP_A	Display A-pattern data.
SLCDC_DISP_B	Display B-pattern data.
SLCDC_DISP_BLINK	Blink between A- and B-pattern.

◆ slcdc_contrast_t

enum slcdc_contrast_t	
LCD Boost Level (contrast) settings	
Enumerator	
SLCDC_CONTRAST_0	Contrast level 0.
SLCDC_CONTRAST_1	Contrast level 1.
SLCDC_CONTRAST_2	Contrast level 2.
SLCDC_CONTRAST_3	Contrast level 3.
SLCDC_CONTRAST_4	Contrast level 4.
SLCDC_CONTRAST_5	Contrast level 5.
SLCDC_CONTRAST_6	Contrast level 6.
SLCDC_CONTRAST_7	Contrast level 7.
SLCDC_CONTRAST_8	Contrast level 8.
SLCDC_CONTRAST_9	Contrast level 9.
SLCDC_CONTRAST_10	Contrast level 10.
SLCDC_CONTRAST_11	Contrast level 11.
SLCDC_CONTRAST_12	Contrast level 12.
SLCDC_CONTRAST_13	Contrast level 13.
SLCDC_CONTRAST_14	Contrast level 14.
SLCDC_CONTRAST_15	Contrast level 15.

◆ slcdc_display_on_off_t

enum slcdc_display_on_off_t	
LCD Display Enable/Disable	
Enumerator	
SLCDC_DISP_OFF	Display off.
SLCDC_DISP_ON	Display on.

◆ slcdc_display_enable_disable_t

enum slcdc_display_enable_disable_t	
LCD Display output enable	
Enumerator	
SLCDC_DISP_DISABLE	Output ground level to segment/common pins.
SLCDC_DISP_ENABLE	Output enable.

◆ slcdc_display_clock_t

enum slcdc_display_clock_t	
LCD Display clock selection	
Enumerator	
SLCDC_CLOCK_LOCO	Display clock source LOCO.
SLCDC_CLOCK_SOSC	Display clock source SOSC.
SLCDC_CLOCK_MOSC	Display clock source MOSC.
SLCDC_CLOCK_HOCO	Display clock source HOCO.

◆ slcdc_clk_div_t

enum slcdc_clk_div_t	
LCD clock settings	
Enumerator	
SLCDC_CLK_DIVISOR_LOCO_4	LOCO Clock/4.

SLCDC_CLK_DIVISOR_LOCO_8	LOCO Clock/8.
SLCDC_CLK_DIVISOR_LOCO_16	LOCO Clock/16.
SLCDC_CLK_DIVISOR_LOCO_32	LOCO Clock/32.
SLCDC_CLK_DIVISOR_LOCO_64	LOCO Clock/64.
SLCDC_CLK_DIVISOR_LOCO_128	LOCO Clock/128.
SLCDC_CLK_DIVISOR_LOCO_256	LOCO Clock/256.
SLCDC_CLK_DIVISOR_LOCO_512	LOCO Clock/512.
SLCDC_CLK_DIVISOR_LOCO_1024	LOCO Clock/1024.
SLCDC_CLK_DIVISOR_HOCO_256	HOCO Clock/256.
SLCDC_CLK_DIVISOR_HOCO_512	HOCO Clock/512.
SLCDC_CLK_DIVISOR_HOCO_1024	HOCO Clock/1024.
SLCDC_CLK_DIVISOR_HOCO_2048	HOCO Clock/2048.
SLCDC_CLK_DIVISOR_HOCO_4096	HOCO Clock/4096.
SLCDC_CLK_DIVISOR_HOCO_8192	HOCO Clock/8192.
SLCDC_CLK_DIVISOR_HOCO_16384	HOCO Clock/16384.
SLCDC_CLK_DIVISOR_HOCO_32768	HOCO Clock/32768.
SLCDC_CLK_DIVISOR_HOCO_65536	HOCO Clock/65536.
SLCDC_CLK_DIVISOR_HOCO_131072	HOCO Clock/131072.
SLCDC_CLK_DIVISOR_HOCO_262144	HOCO Clock/262144.
SLCDC_CLK_DIVISOR_HOCO_524288	HOCO Clock/524288.

4.3.34 SPI Interface

Interfaces

Detailed Description

Interface for SPI communications.

Summary

Provides a common interface for communication using the SPI Protocol.

Implemented by:

- Serial Peripheral Interface (r_spi)
- Serial Communications Interface (SCI) SPI (r_sci_spi)

Data Structures

struct	spi_callback_args_t
struct	spi_write_read_guard_args_t
struct	spi_cfg_t
struct	spi_api_t
struct	spi_instance_t

Typedefs

typedef void	spi_ctrl_t

Enumerations

enum	spi_bit_width_t
enum	spi_mode_t
enum	spi_clk_phase_t
enum	spi_clk_polarity_t
enum	spi_mode_fault_t
enum	spi_bit_order_t
enum	spi_event_t

Data Structure Documentation

◆ spi_callback_args_t

struct spi_callback_args_t
Common callback parameter definition

Data Fields		
uint32_t	channel	Device channel number.
spi_event_t	event	Event code.
void const *	p_context	Context provided to user during callback.

◆ spi_write_read_guard_args_t

struct spi_write_read_guard_args_t
Non-secure arguments for write-read guard function

◆ spi_cfg_t

struct spi_cfg_t	
SPI interface configuration	
Data Fields	
uint8_t	channel
	Channel number to be used.
IRQn_Type	rxi_irq
	Receive Buffer Full IRQ number.
IRQn_Type	txi_irq
	Transmit Buffer Empty IRQ number.
IRQn_Type	tei_irq
	Transfer Complete IRQ number.
IRQn_Type	eri_irq
	Error IRQ number.
uint8_t	rxi_ipl
	Receive Interrupt priority.

uint8_t	txi_ipl
	Transmit Interrupt priority.
uint8_t	tei_ipl
	Transfer Complete Interrupt priority.
uint8_t	eri_ipl
	Error Interrupt priority.
spi_mode_t	operating_mode
	Select master or slave operating mode.
spi_clk_phase_t	clk_phase
	Data sampling on odd or even clock edge.
spi_clk_polarity_t	clk_polarity
	Clock level when idle.
spi_mode_fault_t	mode_fault
	Mode fault error (master/slave conflict) flag.
spi_bit_order_t	bit_order
	Select to transmit MSB/LSB first.
transfer_instance_t const *	p_transfer_tx
	To use SPI DTC/DMA write transfer, link a DTC/DMA instance here. Set to NULL if unused.

transfer_instance_t const *	p_transfer_rx
	To use SPI DTC/DMA read transfer, link a DTC/DMA instance here. Set to NULL if unused.
void(*	p_callback)(spi_callback_args_t *p_args)
	Pointer to user callback function.
void const *	p_context
	User defined context passed to callback function.
void const *	p_extend
	Extended SPI hardware dependent configuration.

◆ **spi_api_t**

struct spi_api_t

Shared Interface definition for SPI

Data Fields

fsp_err_t(*	open)(spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)
fsp_err_t(*	read)(spi_ctrl_t *const p_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t(*	write)(spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t(*	writeRead)(spi_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)
fsp_err_t(*	callbackSet)(spi_ctrl_t *const p_api_ctrl, void(*p_callback)(spi_callback_args_t *), void const *const p_context, spi_callback_args_t *const p_callback_memory)

fsp_err_t(*	close)(spi_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* spi_api_t::open) (spi_ctrl_t *p_ctrl, spi_cfg_t const *const p_cfg)

Initialize a channel for SPI communication mode.

Implemented as

- R_SPI_Open()
- R_SPI_B_Open()
- R_SCI_SPI_Open()

Parameters

[in,out]	p_ctrl	Pointer to user-provided storage for the control block.
[in]	p_cfg	Pointer to SPI configuration structure.

◆ read

fsp_err_t(* spi_api_t::read) (spi_ctrl_t *const p_ctrl, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)

Receive data from a SPI device.

Implemented as

- R_SPI_Read()
- R_SPI_B_Read()
- R_SCI_SPI_Read()

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[in]	length	Number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Data bit width to be transferred.
[out]	p_dest	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count.

◆ write

fsp_err_t(* spi_api_t::write) (spi_ctrl_t *const p_ctrl, void const *p_src, uint32_t const length, spi_bit_width_t const bit_width)

Transmit data to a SPI device.

Implemented as

- R_SPI_Write()
- R_SPI_B_Write()
- R_SCI_SPI_Write()

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[in]	p_src	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
[in]	length	Number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Data bit width to be transferred.

◆ writeRead

fsp_err_t(* spi_api_t::writeRead) (spi_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t const length, spi_bit_width_t const bit_width)

Simultaneously transmit data to a SPI device while receiving data from a SPI device (full duplex).

Implemented as

- R_SPI_WriteRead()
- R_SPI_B_WriteRead()
- R_SCI_SPI_WriteRead()

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.
[in]	p_src	Pointer to a source data buffer from which data will be transmitted to a SPI device. The argument must not be NULL.
[out]	p_dest	Pointer to destination buffer into which data will be copied that is received from a SPI device. It is the responsibility of the caller to ensure that adequate space is available to hold the requested data count. The argument must not be NULL.
[in]	length	Number of units of data to be transferred (unit size specified by the bit_width).
[in]	bit_width	Data bit width to be transferred.

◆ callbackSet

fsp_err_t(* spi_api_t::callbackSet) (spi_ctrl_t *const p_api_ctrl, void(*p_callback)(spi_callback_args_t *), void const *const p_context, spi_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_SPI_CallbackSet()
- R_SPI_B_CallbackSet()
- R_SCI_SPI_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the SPI control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* spi_api_t::close) (spi_ctrl_t *const p_ctrl)

Remove power to the SPI channel designated by the handle and disable the associated interrupts.

Implemented as

- R_SPI_Close()
- R_SPI_B_Close()
- R_SCI_SPI_Close()

Parameters

[in]	p_ctrl	Pointer to the control block for the channel.

◆ spi_instance_t

struct spi_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
spi_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
spi_cfg_t const *	p_cfg	Pointer to the configuration

		structure for this instance.
spi_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ spi_ctrl_t

typedef void spi_ctrl_t

SPI control block. Allocate an instance specific control block to pass into the SPI API calls.

Implemented as

- spi_instance_ctrl_t
- spi_b_instance_ctrl_t
- sci_spi_instance_ctrl_t

Enumeration Type Documentation

◆ spi_bit_width_t

enum spi_bit_width_t

Data bit width

Enumerator	
SPI_BIT_WIDTH_4_BITS	Data bit width is 4 bits (byte)
SPI_BIT_WIDTH_5_BITS	Data bit width is 5 bits (byte)
SPI_BIT_WIDTH_6_BITS	Data bit width is 6 bits (byte)
SPI_BIT_WIDTH_7_BITS	Data bit width is 7 bits (byte)
SPI_BIT_WIDTH_8_BITS	Data bit width is 8 bits (byte)
SPI_BIT_WIDTH_9_BITS	Data bit width is 9 bits (word)
SPI_BIT_WIDTH_10_BITS	Data bit width is 10 bits (word)
SPI_BIT_WIDTH_11_BITS	Data bit width is 11 bits (word)
SPI_BIT_WIDTH_12_BITS	Data bit width is 12 bits (word)
SPI_BIT_WIDTH_13_BITS	Data bit width is 13 bits (word)
SPI_BIT_WIDTH_14_BITS	Data bit width is 14 bits (word)

SPI_BIT_WIDTH_15_BITS	Data bit width is 15 bits (word)
SPI_BIT_WIDTH_16_BITS	Data bit width is 16 bits (word)
SPI_BIT_WIDTH_17_BITS	Data bit width is 17 bits (word)
SPI_BIT_WIDTH_18_BITS	Data bit width is 18 bits (word)
SPI_BIT_WIDTH_19_BITS	Data bit width is 19 bits (word)
SPI_BIT_WIDTH_20_BITS	Data bit width is 20 bits (longword)
SPI_BIT_WIDTH_21_BITS	Data bit width is 21 bits (word)
SPI_BIT_WIDTH_22_BITS	Data bit width is 22 bits (word)
SPI_BIT_WIDTH_23_BITS	Data bit width is 23 bits (longword)
SPI_BIT_WIDTH_24_BITS	Data bit width is 24 bits (longword)
SPI_BIT_WIDTH_25_BITS	Data bit width is 25 bits (longword)
SPI_BIT_WIDTH_26_BITS	Data bit width is 26 bits (word)
SPI_BIT_WIDTH_27_BITS	Data bit width is 27 bits (word)
SPI_BIT_WIDTH_28_BITS	Data bit width is 28 bits (word)
SPI_BIT_WIDTH_29_BITS	Data bit width is 29 bits (word)
SPI_BIT_WIDTH_30_BITS	Data bit width is 30 bits (longword)
SPI_BIT_WIDTH_31_BITS	Data bit width is 31 bits (longword)
SPI_BIT_WIDTH_32_BITS	Data bit width is 32 bits (longword)

◆ spi_mode_t

enum spi_mode_t

Master or slave operating mode

Enumerator	
SPI_MODE_MASTER	Channel operates as SPI master.
SPI_MODE_SLAVE	Channel operates as SPI slave.

◆ spi_clk_phase_t

enum spi_clk_phase_t	
Clock phase	
Enumerator	
SPI_CLK_PHASE_EDGE_ODD	0: Data sampling on odd edge, data variation on even edge
SPI_CLK_PHASE_EDGE_EVEN	1: Data variation on odd edge, data sampling on even edge

◆ spi_clk_polarity_t

enum spi_clk_polarity_t	
Clock polarity	
Enumerator	
SPI_CLK_POLARITY_LOW	0: Clock polarity is low when idle
SPI_CLK_POLARITY_HIGH	1: Clock polarity is high when idle

◆ spi_mode_fault_t

enum spi_mode_fault_t	
Mode fault error flag. This error occurs when the device is setup as a master, but the SSLA line does not seem to be controlled by the master. This usually happens when the connecting device is also acting as master. A similar situation can also happen when configured as a slave.	
Enumerator	
SPI_MODE_FAULT_ERROR_ENABLE	Mode fault error flag on.
SPI_MODE_FAULT_ERROR_DISABLE	Mode fault error flag off.

◆ spi_bit_order_t

enum spi_bit_order_t	
Bit order	
Enumerator	
SPI_BIT_ORDER_MSB_FIRST	Send MSB first in transmission.
SPI_BIT_ORDER_LSB_FIRST	Send LSB first in transmission.

◆ spi_event_t

enum spi_event_t	
SPI events	
Enumerator	
SPI_EVENT_TRANSFER_COMPLETE	The data transfer was completed.
SPI_EVENT_TRANSFER_ABORTED	The data transfer was aborted.
SPI_EVENT_ERR_MODE_FAULT	Mode fault error.
SPI_EVENT_ERR_READ_OVERFLOW	Read overflow error.
SPI_EVENT_ERR_PARITY	Parity error.
SPI_EVENT_ERR_OVERRUN	Overrun error.
SPI_EVENT_ERR_FRAMING	Framing error.
SPI_EVENT_ERR_MODE_UNDERRUN	Underrun error.

4.3.35 SPI Flash Interface

Interfaces

Detailed Description

Interface for accessing external SPI flash devices.

Summary

The SPI flash API provides an interface that configures, writes, and erases sectors in SPI flash devices.

Implemented by:

- Octa Serial Peripheral Interface for Flash and RAM devices (r_ospi)
- Quad Serial Peripheral Interface Flash (r_qspi)

Data Structures

struct	spi_flash_erase_command_t
struct	spi_flash_direct_transfer_t
struct	spi_flash_cfg_t
struct	spi_flash_status_t
struct	spi_flash_api_t
struct	spi_flash_instance_t

Typedefs

typedef void	spi_flash_ctrl_t

Enumerations

enum	spi_flash_read_mode_t
enum	spi_flash_protocol_t
enum	spi_flash_address_bytes_t
enum	spi_flash_data_lines_t
enum	spi_flash_dummy_clocks_t
enum	spi_flash_direct_transfer_dir_t

Data Structure Documentation

◆ spi_flash_erase_command_t

struct spi_flash_erase_command_t		
Structure to define an erase command and associated erase size.		
Data Fields		
uint16_t	command	Erase command.
uint32_t	size	Size of erase for associated

		command, set to SPI_FLASH_ERASE_SIZE_CHIP_ERASE for chip erase.

◆ spi_flash_direct_transfer_t

struct spi_flash_direct_transfer_t		
Structure to define a direct transfer.		
Data Fields		
uint32_t	address	Starting address.
uint32_t	data	Data.
uint16_t	command	Transfer command.
uint8_t	dummy_cycles	Number of dummy cycles.
uint8_t	command_length	Command length.
uint8_t	address_length	Address lengrh.
uint8_t	data_length	Data length.

◆ spi_flash_cfg_t

struct spi_flash_cfg_t		
User configuration structure used by the open function		
Data Fields		
spi_flash_protocol_t	spi_protocol	Initial SPI protocol. SPI protocol can be changed in spi_flash_api_t::spiProtocolSet.
spi_flash_read_mode_t	read_mode	Read mode.
spi_flash_address_bytes_t	address_bytes	Number of bytes used to represent the address.
spi_flash_dummy_clocks_t	dummy_clocks	Number of dummy clocks to use for fast read operations.
spi_flash_data_lines_t	page_program_address_lines	Number of lines used to send address for page program command. This should either be 1 or match the number of lines used in the selected read mode.
uint8_t	write_status_bit	Which bit determines write status.
uint8_t	write_enable_bit	Which bit determines write status.
uint32_t	page_size_bytes	Page size in bytes (maximum number of bytes for page program). Used to specify single continuous write size

		(bytes) in case of OSPI RAM.
uint8_t	page_program_command	Page program command.
uint8_t	write_enable_command	Command to enable write or erase, typically 0x06.
uint8_t	status_command	Command to read the write status.
uint8_t	read_command	Read command - OSPI SPI mode only.
uint8_t	xip_enter_command	Command to enter XIP mode.
uint8_t	xip_exit_command	Command to exit XIP mode.
uint8_t	erase_command_list_length	Length of erase command list.
spi_flash_erase_command_t const *	p_erase_command_list	List of all erase commands and associated sizes.
void const *	p_extend	Pointer to implementation specific extended configurations.

◆ **spi_flash_status_t**

struct spi_flash_status_t		
Status.		
Data Fields		
bool	write_in_progress	Whether or not a write is in progress. This is determined by reading the spi_flash_cfg_t::write_status_bit from the spi_flash_cfg_t::status_command.

◆ **spi_flash_api_t**

struct spi_flash_api_t	
SPI flash implementations follow this API.	
Data Fields	
fsp_err_t(*	open)(spi_flash_ctrl_t *p_ctrl, spi_flash_cfg_t const *const p_cfg)
fsp_err_t(*	directWrite)(spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write)
fsp_err_t(*	directRead)(spi_flash_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

fsp_err_t(*	directTransfer)(spi_flash_ctrl_t *p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction)
fsp_err_t(*	spiProtocolSet)(spi_flash_ctrl_t *p_ctrl, spi_flash_protocol_t spi_protocol)
fsp_err_t(*	write)(spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count)
fsp_err_t(*	erase)(spi_flash_ctrl_t *p_ctrl, uint8_t *const p_device_address, uint32_t byte_count)
fsp_err_t(*	statusGet)(spi_flash_ctrl_t *p_ctrl, spi_flash_status_t *const p_status)
fsp_err_t(*	xipEnter)(spi_flash_ctrl_t *p_ctrl)
fsp_err_t(*	xipExit)(spi_flash_ctrl_t *p_ctrl)
fsp_err_t(*	bankSet)(spi_flash_ctrl_t *p_ctrl, uint32_t bank)
fsp_err_t(*	autoCalibrate)(spi_flash_ctrl_t *p_ctrl)
fsp_err_t(*	close)(spi_flash_ctrl_t *p_ctrl)

Field Documentation

◆ open

fsp_err_t(* spi_flash_api_t::open) (spi_flash_ctrl_t *p_ctrl, spi_flash_cfg_t const *const p_cfg)

Open the SPI flash driver module.

Implemented as

- R_OSPI_Open()
- R_QSPI_Open()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_cfg	Pointer to a configuration structure

◆ directWrite

fsp_err_t(* spi_flash_api_t::directWrite) (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint32_t const bytes, bool const read_after_write)

Write raw data to the SPI flash.

Implemented as

- R_OSPI_DirectWrite()
- R_QSPI_DirectWrite()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_src	Pointer to raw data to write, must include any required command/address
[in]	bytes	Number of bytes to write
[in]	read_after_write	If true, the slave select remains asserted and the peripheral does not return to direct communications mode. If false, the slave select is deasserted and memory mapped access is possible after this function returns if the device is not busy.

◆ directRead

fsp_err_t(* spi_flash_api_t::directRead) (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Read raw data from the SPI flash. Must follow a call to spi_flash_api_t::directWrite.

Implemented as

- R_OSPI_DirectRead()
- R_QSPI_DirectRead()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[out]	p_dest	Pointer to read raw data into
[in]	bytes	Number of bytes to read

◆ directTransfer

fsp_err_t(* spi_flash_api_t::directTransfer) (spi_flash_ctrl_t *p_ctrl, spi_flash_direct_transfer_t *const p_transfer, spi_flash_direct_transfer_dir_t direction)

Direct Read/Write raw data to the SPI flash.

Implemented as

- R_OSPI_DirectTransfer()
- R_QSPI_DirectTransfer()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_data	Pointer to command, address and data values and lengths
[in]	direction	Direct Read/Write

◆ spiProtocolSet

fsp_err_t(* spi_flash_api_t::spiProtocolSet) (spi_flash_ctrl_t *p_ctrl, spi_flash_protocol_t spi_protocol)

Change the SPI protocol in the driver. The application must change the SPI protocol on the device.

Implemented as

- R_OSPI_SpiProtocolSet()
- R_QSPI_SpiProtocolSet()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	spi_protocol	Desired SPI protocol

◆ write

fsp_err_t(* spi_flash_api_t::write) (spi_flash_ctrl_t *p_ctrl, uint8_t const *const p_src, uint8_t *const p_dest, uint32_t byte_count)

Program a page of data to the flash.

Implemented as

- R_OSPI_Write()
- R_QSPI_Write()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_src	The memory address of the data to write to the flash device
[in]	p_dest	The location in the flash device address space to write the data to
[in]	byte_count	The number of bytes to write

◆ erase

fsp_err_t(* spi_flash_api_t::erase) (spi_flash_ctrl_t *p_ctrl, uint8_t *const p_device_address, uint32_t byte_count)

Erase a certain number of bytes of the flash.

Implemented as

- R_OSPI_Erase()
- R_QSPI_Erase()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	p_device_address	The location in the flash device address space to start the erase from
[in]	byte_count	The number of bytes to erase. Set to SPI_FLASH_ERASE_SIZE_CHIP_ERASE to erase entire chip.

◆ statusGet

fsp_err_t(* spi_flash_api_t::statusGet) (spi_flash_ctrl_t *p_ctrl, spi_flash_status_t *const p_status)

Get the write or erase status of the flash.

Implemented as

- R_OSPI_StatusGet()
- R_QSPI_StatusGet()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[out]	p_status	Current status of the SPI flash device stored here.

◆ xipEnter

fsp_err_t(* spi_flash_api_t::xipEnter) (spi_flash_ctrl_t *p_ctrl)

Enter XIP mode.

Implemented as

- R_OSPI_XipEnter()
- R_QSPI_XipEnter()

Parameters

[in]	p_ctrl	Pointer to a driver handle

◆ xipExit

fsp_err_t(* spi_flash_api_t::xipExit) (spi_flash_ctrl_t *p_ctrl)

Exit XIP mode.

Implemented as

- R_OSPI_XipExit()
- R_QSPI_XipExit()

Parameters

[in]	p_ctrl	Pointer to a driver handle

◆ bankSet

fsp_err_t(* spi_flash_api_t::bankSet) (spi_flash_ctrl_t *p_ctrl, uint32_t bank)

Select the bank to access. See implementation for details.

Implemented as

- R_OSPI_BankSet()
- R_QSPI_BankSet()

Parameters

[in]	p_ctrl	Pointer to a driver handle
[in]	bank	The bank number

◆ autoCalibrate

fsp_err_t(* spi_flash_api_t::autoCalibrate) (spi_flash_ctrl_t *p_ctrl)

AutoCalibrate the SPI flash driver module. Expected to be used when auto-calibrating OSPI RAM device.

Implemented as

- R_OSPI_AutoCalibrate()
- R_QSPI_AutoCalibrate()

Parameters

[in]	p_ctrl	Pointer to a driver handle

◆ close

fsp_err_t(* spi_flash_api_t::close) (spi_flash_ctrl_t *p_ctrl)

Close the SPI flash driver module.

Implemented as

- R_OSPI_Close()
- R_QSPI_Close()

Parameters

[in]	p_ctrl	Pointer to a driver handle

◆ spi_flash_instance_t

struct spi_flash_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
spi_flash_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.

spi_flash_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
spi_flash_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ spi_flash_ctrl_t

typedef void spi_flash_ctrl_t

SPI flash control block. Allocate an instance specific control block to pass into the SPI flash API calls.

Implemented as

- qspi_instance_ctrl_t
- ospi_instance_ctrl_t

Enumeration Type Documentation

◆ spi_flash_read_mode_t

enum spi_flash_read_mode_t

Read mode.

Enumerator	
SPI_FLASH_READ_MODE_STANDARD	Standard Read Mode (no dummy cycles)
SPI_FLASH_READ_MODE_FAST_READ	Fast Read Mode (dummy cycles between address and data)
SPI_FLASH_READ_MODE_FAST_READ_DUAL_OUTPUT	Fast Read Dual Output Mode (data on 2 lines)
SPI_FLASH_READ_MODE_FAST_READ_DUAL_IO	Fast Read Dual I/O Mode (address and data on 2 lines)
SPI_FLASH_READ_MODE_FAST_READ_QUAD_OUTPUT	Fast Read Quad Output Mode (data on 4 lines)
SPI_FLASH_READ_MODE_FAST_READ_QUAD_IO	Fast Read Quad I/O Mode (address and data on 4 lines)

◆ spi_flash_protocol_t

enum spi_flash_protocol_t	
SPI protocol.	
Enumerator	
SPI_FLASH_PROTOCOL_EXTENDED_SPI	Extended SPI mode (commands on 1 line)
SPI_FLASH_PROTOCOL_QPI	QPI mode (commands on 4 lines). Note that the application must ensure the device is in QPI mode.
SPI_FLASH_PROTOCOL_SOPI	SOPI mode (command and data on 8 lines). Note that the application must ensure the device is in SOPI mode.
SPI_FLASH_PROTOCOL_DOPI	DOPI mode (command and data on 8 lines, dual data rate). Note that the application must ensure the device is in DOPI mode.

◆ spi_flash_address_bytes_t

enum spi_flash_address_bytes_t	
Number of bytes in the address.	
Enumerator	
SPI_FLASH_ADDRESS_BYTES_3	3 address bytes
SPI_FLASH_ADDRESS_BYTES_4	4 address bytes with standard commands. If this option is selected, the application must issue the EN4B command using spi_flash_api_t::directWrite() if required by the device.
SPI_FLASH_ADDRESS_BYTES_4_4BYTE_READ_CODE	4 address bytes using standard 4-byte command set.

◆ spi_flash_data_lines_t

enum spi_flash_data_lines_t	
Number of data lines used.	
Enumerator	
SPI_FLASH_DATA_LINES_1	1 data line
SPI_FLASH_DATA_LINES_2	2 data lines
SPI_FLASH_DATA_LINES_4	4 data lines

◆ spi_flash_dummy_clocks_t

enum spi_flash_dummy_clocks_t	
Number of dummy cycles for fast read operations.	
Enumerator	
SPI_FLASH_DUMMY_CLOCKS_DEFAULT	Default is 6 clocks for Fast Read Quad I/O, 4 clocks for Fast Read Dual I/O, and 8 clocks for other fast read instructions including Fast Read Quad Output, Fast Read Dual Output, and Fast Read.
SPI_FLASH_DUMMY_CLOCKS_3	3 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_4	4 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_5	5 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_6	6 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_7	7 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_8	8 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_9	9 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_10	10 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_11	11 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_12	12 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_13	13 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_14	14 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_15	15 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_16	16 dummy clocks
SPI_FLASH_DUMMY_CLOCKS_17	17 dummy clocks

◆ spi_flash_direct_transfer_dir_t

enum spi_flash_direct_transfer_dir_t
Direct Read and Write direction

4.3.36 Three-Phase Interface

Interfaces

Detailed Description

Interface for three-phase timer functions.

Summary

The Three-Phase interface provides functionality for synchronous start/stop/reset control of three timer channels for use in 3-phase motor control applications.

Implemented by:

- General PWM Timer Three-Phase Motor Control Driver (r_gpt_three_phase)

Data Structures

struct	three_phase_duty_cycle_t
struct	three_phase_cfg_t
struct	three_phase_api_t
struct	three_phase_instance_t

Typedefs

typedef void	three_phase_ctrl_t

Enumerations

enum	three_phase_channel_t
enum	three_phase_buffer_mode_t

Data Structure Documentation

◆ three_phase_duty_cycle_t

struct three_phase_duty_cycle_t

Struct for passing duty cycle values to three_phase_api_t::dutyCycleSet		
Data Fields		
uint32_t	duty[3]	Duty cycle. Note: When the GPT instances are configured in TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM_MODE3, this value sets the duty cycle count that is transfered to GTCCRA/B at the trough.
uint32_t	duty_buffer[3]	Double-buffer for duty cycle values. Note: When the GPT instances are configured in TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM_MODE3, this value sets the duty cycle count that is transfered to GTCCRA/B at the crest.

◆ three_phase_cfg_t

struct three_phase_cfg_t		
User configuration structure, used in open function		
Data Fields		
three_phase_buffer_mode_t	buffer_mode	Single or double-buffer mode.
timer_instance_t const *	p_timer_instance[3]	Pointer to the timer instance structs.
three_phase_channel_t	callback_ch	Channel to enable callback when using three_phase_api_t::callbackSet.
uint32_t	channel_mask	Bitmask of timer channels used by this module.
void const *	p_context	Placeholder for user data. Passed to the user callback in timer_callback_args_t.
void const *	p_extend	Extension parameter for hardware specific settings.

◆ three_phase_api_t

struct three_phase_api_t	
Three-Phase API structure.	
Data Fields	
fsp_err_t(*	open)(three_phase_ctrl_t *const p_ctrl, three_phase_cfg_t const *const p_cfg)

fsp_err_t(*	start)(three_phase_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(three_phase_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(three_phase_ctrl_t *const p_ctrl)
fsp_err_t(*	dutyCycleSet)(three_phase_ctrl_t *const p_ctrl, three_phase_duty_cycle_t *const p_duty_cycle)
fsp_err_t(*	callbackSet)(three_phase_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void const *const p_context, timer_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(three_phase_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* three_phase_api_t::open) (three_phase_ctrl_t *const p_ctrl, three_phase_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_GPT_THREE_PHASE_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ **start**

fsp_err_t(* three_phase_api_t::start) (three_phase_ctrl_t *const p_ctrl)

Start all three timers synchronously.

Implemented as

- R_GPT_THREE_PHASE_Start()

Parameters

[in]	p_ctrl	Control block set in three_phase_api_t::open call for this timer.

◆ **stop**

fsp_err_t(* three_phase_api_t::stop) (three_phase_ctrl_t *const p_ctrl)

Stop all three timers synchronously.

Implemented as

- R_GPT_THREE_PHASE_Stop()

Parameters

[in]	p_ctrl	Control block set in three_phase_api_t::open call for this timer.

◆ **reset**

fsp_err_t(* three_phase_api_t::reset) (three_phase_ctrl_t *const p_ctrl)

Reset all three timers synchronously.

Implemented as

- R_GPT_THREE_PHASE_Reset()

Parameters

[in]	p_ctrl	Control block set in three_phase_api_t::open call for this timer.

◆ dutyCycleSet

fsp_err_t(* three_phase_api_t::dutyCycleSet) (three_phase_ctrl_t *const p_ctrl, three_phase_duty_cycle_t *const p_duty_cycle)

Sets the duty cycle match values. If the timer is counting, the updated duty cycle is reflected after the next timer expiration.

Implemented as

- R_GPT_THREE_PHASE_DutyCycleSet()

Parameters

[in]	p_ctrl	Control block set in three_phase_api_t::open call for this timer.
[in]	p_duty_cycle	Duty cycle values for all three timer channels.

◆ callbackSet

fsp_err_t(* three_phase_api_t::callbackSet) (three_phase_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void const *const p_context, timer_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_GPT_THREE_PHASE_CallbackSet()

Parameters

[in]	p_ctrl	Control block set in three_phase_api_t::open call.
[in]	p_callback	Callback function to register with GPT U-channel
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* three_phase_api_t::close) (three_phase_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

Implemented as

- R_GPT_THREE_PHASE_Close()

Parameters

[in]	p_ctrl	Control block set in three_phase_api_t::open call for this timer.

◆ three_phase_instance_t

struct three_phase_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
three_phase_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
three_phase_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
three_phase_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ three_phase_ctrl_t

typedef void three_phase_ctrl_t

Three-Phase control block. Allocate an instance specific control block to pass into the timer API calls.

Implemented as

- gpt_three_phase_instance_ctrl_t

Enumeration Type Documentation

◆ three_phase_channel_t

enum three_phase_channel_t	
Timer channel indices	
Enumerator	
THREE_PHASE_CHANNEL_U	U-channel index.
THREE_PHASE_CHANNEL_V	V-channel index.
THREE_PHASE_CHANNEL_W	W-channel index.

◆ three_phase_buffer_mode_t

enum three_phase_buffer_mode_t	
Buffering mode	
Enumerator	
THREE_PHASE_BUFFER_MODE_SINGLE	Single-buffer mode.
THREE_PHASE_BUFFER_MODE_DOUBLE	Double-buffer mode.

4.3.37 Timer Interface

Interfaces

Detailed Description

Interface for timer functions.

Summary

The general timer interface provides standard timer functionality including periodic mode, one-shot mode, PWM output, and free-running timer mode. After each timer cycle (overflow or underflow), an interrupt can be triggered.

If an instance supports output compare mode, it is provided in the extension configuration timer_on_<instance>_cfg_t defined in r_<instance>.h.

Implemented by:

- General PWM Timer (r_gpt)
- Asynchronous General Purpose Timer (r_agt)

Data Structures

struct	timer_callback_args_t
struct	timer_info_t
struct	timer_status_t
struct	timer_cfg_t
struct	timer_api_t
struct	timer_instance_t

Typedefs

typedef void	timer_ctrl_t

Enumerations

enum	timer_event_t
enum	timer_variant_t
enum	timer_state_t
enum	timer_mode_t
enum	timer_direction_t
enum	timer_source_div_t

Data Structure Documentation

◆ timer_callback_args_t

struct timer_callback_args_t

Callback function parameter data

Data Fields		
void const *	p_context	Placeholder for user data. Set in timer_api_t::open function in timer_cfg_t.
timer_event_t	event	The event can be used to identify what caused the callback.
uint32_t	capture	Most recent capture, only valid if event is TIMER_EVENT_CAPTURE_A or TIMER_EVENT_CAPTURE_B.

◆ timer_info_t

struct timer_info_t		
Timer information structure to store various information for a timer resource		
Data Fields		
timer_direction_t	count_direction	Clock counting direction of the timer.
uint32_t	clock_frequency	Clock frequency of the timer counter.
uint32_t	period_counts	Period in raw timer counts. *Note* *For triangle wave PWM modes, the full period is double this value.*

◆ timer_status_t

struct timer_status_t		
Current timer status.		
Data Fields		
uint32_t	counter	Current counter value.
timer_state_t	state	Current timer state (running or stopped)

◆ timer_cfg_t

struct timer_cfg_t	
User configuration structure, used in open function	
Data Fields	
timer_mode_t	mode
	Select enumerated value from timer_mode_t.
uint32_t	period_counts
	Period in raw timer counts.
timer_source_div_t	source_div
	Source clock divider.
uint32_t	duty_cycle_counts

	Duty cycle in counts.
uint8_t	channel
uint8_t	cycle_end_ipl
	Cycle end interrupt priority.
IRQn_Type	cycle_end_irq
	Cycle end interrupt.
void(*	p_callback)(timer_callback_args_t *p_args)
void const *	p_context
void const *	p_extend
	Extension parameter for hardware specific settings.

Field Documentation

◆ **channel**

uint8_t timer_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

◆ **p_callback**

void(* timer_cfg_t::p_callback) (timer_callback_args_t *p_args)

Callback provided when a timer ISR occurs. Set to NULL for no CPU interrupt.

◆ **p_context**

void const* timer_cfg_t::p_context

Placeholder for user data. Passed to the user callback in timer_callback_args_t.

◆ **timer_api_t**

struct timer_api_t

Timer API structure. General timer functions implemented at the HAL layer follow this API.

Data Fields

fsp_err_t(*	open)(timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)
fsp_err_t(*	start)(timer_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(timer_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(timer_ctrl_t *const p_ctrl)
fsp_err_t(*	enable)(timer_ctrl_t *const p_ctrl)
fsp_err_t(*	disable)(timer_ctrl_t *const p_ctrl)
fsp_err_t(*	periodSet)(timer_ctrl_t *const p_ctrl, uint32_t const period)
fsp_err_t(*	dutyCycleSet)(timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)
fsp_err_t(*	infoGet)(timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)
fsp_err_t(*	statusGet)(timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)
fsp_err_t(*	callbackSet)(timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void const *const p_context, timer_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(timer_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* timer_api_t::open) (timer_ctrl_t *const p_ctrl, timer_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_GPT_Open()
- R_AGT_Open()

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ start

fsp_err_t(* timer_api_t::start) (timer_ctrl_t *const p_ctrl)

Start the counter.

Implemented as

- R_GPT_Start()
- R_AGT_Start()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ stop

fsp_err_t(* timer_api_t::stop) (timer_ctrl_t *const p_ctrl)

Stop the counter.

Implemented as

- R_GPT_Stop()
- R_AGT_Stop()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ reset

fsp_err_t(* timer_api_t::reset) (timer_ctrl_t *const p_ctrl)

Reset the counter to the initial value.

Implemented as

- R_GPT_Reset()
- R_AGT_Reset()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ enable

fsp_err_t(* timer_api_t::enable) (timer_ctrl_t *const p_ctrl)

Enables input capture.

Implemented as

- R_GPT_Enable()
- R_AGT_Enable()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ disable

fsp_err_t(* timer_api_t::disable) (timer_ctrl_t *const p_ctrl)

Disables input capture.

Implemented as

- R_GPT_Disable()
- R_AGT_Disable()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ periodSet

fsp_err_t(* timer_api_t::periodSet) (timer_ctrl_t *const p_ctrl, uint32_t const period)

Set the time until the timer expires. See implementation for details of period update timing.

Implemented as

- R_GPT_PeriodSet()
- R_AGT_PeriodSet()

Note

Timer expiration may or may not generate a CPU interrupt based on how the timer is configured in timer_api_t::open.

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[in]	p_period	Time until timer should expire.

◆ dutyCycleSet

fsp_err_t(* timer_api_t::dutyCycleSet) (timer_ctrl_t *const p_ctrl, uint32_t const duty_cycle_counts, uint32_t const pin)

Sets the number of counts for the pin level to be high. If the timer is counting, the updated duty cycle is reflected after the next timer expiration.

Implemented as

- R_GPT_DutyCycleSet()
- R_AGT_DutyCycleSet()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[in]	duty_cycle_counts	Time until duty cycle should expire.
[in]	pin	Which output pin to update. See implementation for details.

◆ infoGet

fsp_err_t(* timer_api_t::infoGet) (timer_ctrl_t *const p_ctrl, timer_info_t *const p_info)

Stores timer information in p_info.

Implemented as

- R_GPT_InfoGet()
- R_AGT_InfoGet()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[out]	p_info	Collection of information for this timer.

◆ statusGet

fsp_err_t(* timer_api_t::statusGet) (timer_ctrl_t *const p_ctrl, timer_status_t *const p_status)

Get the current counter value and timer state and store it in p_status.

Implemented as

- R_GPT_StatusGet()
- R_AGT_StatusGet()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[out]	p_status	Current status of this timer.

◆ callbackSet

fsp_err_t(* timer_api_t::callbackSet) (timer_ctrl_t *const p_api_ctrl, void(*p_callback)(timer_callback_args_t *), void const *const p_context, timer_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_GPT_CallbackSet()
- R_AGT_CallbackSet()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* timer_api_t::close) (timer_ctrl_t *const p_ctrl)

Allows driver to be reconfigured and may reduce power consumption.

Implemented as

- R_GPT_Close()
- R_AGT_Close()

Parameters

[in]	p_ctrl	Control block set in timer_api_t::open call for this timer.

◆ timer_instance_t

struct timer_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
timer_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.

timer_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
timer_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ timer_ctrl_t

typedef void timer_ctrl_t

Timer control block. Allocate an instance specific control block to pass into the timer API calls.

Implemented as

- gpt_instance_ctrl_t
- agt_instance_ctrl_t

Enumeration Type Documentation

◆ timer_event_t

enum timer_event_t

Events that can trigger a callback function

Enumerator	
TIMER_EVENT_CYCLE_END	Requested timer delay has expired or timer has wrapped around.
TIMER_EVENT_CREST	Timer crest event (counter is at a maximum, triangle-wave PWM only)
TIMER_EVENT_CAPTURE_A	A capture has occurred on signal A.
TIMER_EVENT_CAPTURE_B	A capture has occurred on signal B.
TIMER_EVENT_TROUGH	Timer trough event (counter is 0, triangle-wave PWM only.

◆ timer_variant_t

enum timer_variant_t	
Timer variant types.	
Enumerator	
TIMER_VARIANT_32_BIT	32-bit timer
TIMER_VARIANT_16_BIT	16-bit timer

◆ timer_state_t

enum timer_state_t	
Possible status values returned by timer_api_t::statusGet.	
Enumerator	
TIMER_STATE_STOPPED	Timer is stopped.
TIMER_STATE_COUNTING	Timer is running.

◆ timer_mode_t

enum timer_mode_t	
Timer operational modes	
Enumerator	
TIMER_MODE_PERIODIC	Timer restarts after period elapses.
TIMER_MODE_ONE_SHOT	Timer stops after period elapses.
TIMER_MODE_PWM	Timer generates saw-wave PWM output.
TIMER_MODE_TRIANGLE_WAVE_SYMMETRIC_PWM	Timer generates symmetric triangle-wave PWM output.
TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM	Timer generates asymmetric triangle-wave PWM output.
TIMER_MODE_TRIANGLE_WAVE_ASYMMETRIC_PWM_MODE3	Timer generates Asymmetric Triangle-wave PWM output. In PWM mode 3, the duty cycle does not need to be updated at each tough/crest interrupt. Instead, the trough and crest duty cycle values can be set once and only need to be updated when the application needs to change the duty cycle.

◆ timer_direction_t

enum timer_direction_t	
Direction of timer count	
Enumerator	
TIMER_DIRECTION_DOWN	Timer count goes up.
TIMER_DIRECTION_UP	Timer count goes down.

◆ timer_source_div_t

enum timer_source_div_t	
PCLK divisors	
Enumerator	
TIMER_SOURCE_DIV_1	Timer clock source divided by 1.
TIMER_SOURCE_DIV_2	Timer clock source divided by 2.
TIMER_SOURCE_DIV_4	Timer clock source divided by 4.
TIMER_SOURCE_DIV_8	Timer clock source divided by 8.
TIMER_SOURCE_DIV_16	Timer clock source divided by 16.
TIMER_SOURCE_DIV_32	Timer clock source divided by 32.
TIMER_SOURCE_DIV_64	Timer clock source divided by 64.
TIMER_SOURCE_DIV_128	Timer clock source divided by 128.
TIMER_SOURCE_DIV_256	Timer clock source divided by 256.
TIMER_SOURCE_DIV_512	Timer clock source divided by 512.
TIMER_SOURCE_DIV_1024	Timer clock source divided by 1024.

4.3.38 Transfer Interface

Interfaces

Detailed Description

Interface for data transfer functions.

Summary

The transfer interface supports background data transfer (no CPU intervention).

Implemented by:

- Data Transfer Controller (r_dtc)
- Direct Memory Access Controller (r_dmac)

Data Structures

struct	transfer_properties_t
struct	transfer_info_t
struct	transfer_cfg_t
struct	transfer_api_t
struct	transfer_instance_t

Typedefs

typedef void	transfer_ctrl_t

Enumerations

enum	transfer_mode_t
enum	transfer_size_t
enum	transfer_addr_mode_t
enum	transfer_repeat_area_t
enum	transfer_chain_mode_t
enum	transfer_irq_t
enum	transfer_start_mode_t

Data Structure Documentation

◆ transfer_properties_t

struct transfer_properties_t

Driver specific information.

Data Fields		
uint32_t	block_count_max	Maximum number of blocks.
uint32_t	block_count_remaining	Number of blocks remaining.
uint32_t	transfer_length_max	Maximum number of transfers.
uint32_t	transfer_length_remaining	Number of transfers remaining.

◆ transfer_info_t

struct transfer_info_t

This structure specifies the properties of the transfer.

Warning

When using DTC, this structure corresponds to the descriptor block registers required by the DTC. The following components may be modified by the driver: p_src, p_dest, num_blocks, and length.
When using DTC, do NOT reuse this structure to configure multiple transfers. Each transfer must have a unique transfer_info_t.
When using DTC, this structure must not be allocated in a temporary location. Any instance of this structure must remain in scope until the transfer it is used for is closed.

Note

When using DTC, consider placing instances of this structure in a protected section of memory.

Data Fields		
union transfer_info_t	__unnamed__	
void const *volatile	p_src	Source pointer.
void *volatile	p_dest	Destination pointer.
volatile uint16_t	num_blocks	Number of blocks to transfer when using TRANSFER_MODE_BLOCK (both DTC an DMAC) or TRANSFER_MODE_REPEAT (DMAC only) or TRANSFER_MODE_REPEAT_BLOCK (DMAC only), unused in other modes.
volatile uint16_t	length	Length of each transfer. Range limited for TRANSFER_MODE_BLOCK, TRANSFER_MODE_REPEAT, and TRANSFER_MODE_REPEAT_BLOCK see HAL driver for details.

◆ transfer_cfg_t

struct transfer_cfg_t

Driver configuration set in transfer_api_t::open. All elements except p_extend are required and must be initialized.

Data Fields		
transfer_info_t *	p_info	Pointer to transfer configuration options. If using chain transfer (DTC only), this can be a pointer to an array of chained transfers that will be completed in order.
void const *	p_extend	Extension parameter for hardware specific settings.

◆ transfer_api_t

struct transfer_api_t

Transfer functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg)
fsp_err_t(*	reconfigure)(transfer_ctrl_t *const p_ctrl, transfer_info_t *p_info)
fsp_err_t(*	reset)(transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers)
fsp_err_t(*	enable)(transfer_ctrl_t *const p_ctrl)
fsp_err_t(*	disable)(transfer_ctrl_t *const p_ctrl)
fsp_err_t(*	softwareStart)(transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)
fsp_err_t(*	softwareStop)(transfer_ctrl_t *const p_ctrl)
fsp_err_t(*	infoGet)(transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_properties)
fsp_err_t(*	close)(transfer_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* transfer_api_t::open) (transfer_ctrl_t *const p_ctrl, transfer_cfg_t const *const p_cfg)

Initial configuration.

Implemented as

- R_DTC_Open()
- R_DMAC_Open()

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ reconfigure

fsp_err_t(* transfer_api_t::reconfigure) (transfer_ctrl_t *const p_ctrl, transfer_info_t *p_info)

Reconfigure the transfer. Enable the transfer if p_info is valid.

Implemented as

- R_DTC_Reconfigure()
- R_DMAC_Reconfigure()

Parameters

[in,out]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_info	Pointer to a new transfer info structure.

◆ reset

fsp_err_t(* transfer_api_t::reset) (transfer_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint16_t const num_transfers)

Reset source address pointer, destination address pointer, and/or length, keeping all other settings the same. Enable the transfer if p_src, p_dest, and length are valid.

Implemented as

- R_DTC_Reset()
- R_DMAC_Reset()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[in]	p_src	Pointer to source. Set to NULL if source pointer should not change.
[in]	p_dest	Pointer to destination. Set to NULL if destination pointer should not change.
[in]	num_transfers	Transfer length in normal mode or number of blocks in block mode. In DMAC only, resets number of repeats (initially stored in transfer_info_t::num_blocks) in repeat mode. Not used in repeat mode for DTC.

◆ enable

fsp_err_t(* transfer_api_t::enable) (transfer_ctrl_t *const p_ctrl)

Enable transfer. Transfers occur after the activation source event (or when transfer_api_t::softwareStart is called if ELC_EVENT_ELC_NONE is chosen as activation source).

Implemented as

- R_DTC_Enable()
- R_DMAC_Enable()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ disable

fsp_err_t(* transfer_api_t::disable) (transfer_ctrl_t *const p_ctrl)

Disable transfer. Transfers do not occur after the activation source event (or when transfer_api_t::softwareStart is called if ELC_EVENT_ELC_NONE is chosen as the DMAC activation source).

Note

If a transfer is in progress, it will be completed. Subsequent transfer requests do not cause a transfer.

Implemented as

- R_DTC_Disable()
- R_DMAC_Disable()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ softwareStart

fsp_err_t(* transfer_api_t::softwareStart) (transfer_ctrl_t *const p_ctrl, transfer_start_mode_t mode)

Start transfer in software.

Warning

Only works if ELC_EVENT_ELC_NONE is chosen as the DMAC activation source.

Note

Not supported for DTC.

Implemented as

- R_DMAC_SoftwareStart()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[in]	mode	Select mode from transfer_start_mode_t.

◆ softwareStop

fsp_err_t(* transfer_api_t::softwareStop) (transfer_ctrl_t *const p_ctrl)

Stop transfer in software. The transfer will stop after completion of the current transfer.

Note

Not supported for DTC.
Only applies for transfers started with TRANSFER_START_MODE_REPEAT.

Warning

Only works if ELC_EVENT_ELC_NONE is chosen as the DMAC activation source.

Implemented as

- R_DMAC_SoftwareStop()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ infoGet

fsp_err_t(* transfer_api_t::infoGet) (transfer_ctrl_t *const p_ctrl, transfer_properties_t *const p_properties)

Provides information about this transfer.

Implemented as

- R_DTC_InfoGet()
- R_DMAC_InfoGet()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.
[out]	p_properties	Driver specific information.

◆ close

fsp_err_t(* transfer_api_t::close) (transfer_ctrl_t *const p_ctrl)

Releases hardware lock. This allows a transfer to be reconfigured using transfer_api_t::open.

Implemented as

- R_DTC_Close()
- R_DMAC_Close()

Parameters

[in]	p_ctrl	Control block set in transfer_api_t::open call for this transfer.

◆ transfer_instance_t

struct transfer_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
transfer_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
transfer_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
transfer_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ transfer_ctrl_t

typedef void transfer_ctrl_t

Transfer control block. Allocate an instance specific control block to pass into the transfer API calls.

Implemented as

- dtc_instance_ctrl_t
- dmac_instance_ctrl_t

Enumeration Type Documentation

◆ transfer_mode_t

enum transfer_mode_t	
Transfer mode describes what will happen when a transfer request occurs.	
Enumerator	
TRANSFER_MODE_NORMAL	In normal mode, each transfer request causes a transfer of transfer_size_t from the source pointer to the destination pointer. The transfer length is decremented and the source and address pointers are updated according to transfer_addr_mode_t. After the transfer length reaches 0, transfer requests will not cause any further transfers.
TRANSFER_MODE_REPEAT	Repeat mode is like normal mode, except that when the transfer length reaches 0, the pointer to the repeat area and the transfer length will be reset to their initial values. If DMAC is used, the transfer repeats only transfer_info_t::num_blocks times. After the transfer repeats transfer_info_t::num_blocks times, transfer requests will not cause any further transfers. If DTC is used, the transfer repeats continuously (no limit to the number of repeat transfers).
TRANSFER_MODE_BLOCK	In block mode, each transfer request causes transfer_info_t::length transfers of transfer_size_t. After each individual transfer, the source and destination pointers are updated according to transfer_addr_mode_t. After the block transfer is complete, transfer_info_t::num_blocks is decremented. After the transfer_info_t::num_blocks reaches 0, transfer requests will not cause any further transfers.
TRANSFER_MODE_REPEAT_BLOCK	In addition to block mode features, repeat-block mode supports a ring buffer of blocks and offsets within a block (to split blocks into arrays of their first data, second data, etc.)

◆ transfer_size_t

enum transfer_size_t	
Transfer size specifies the size of each individual transfer. Total transfer length = transfer_size_t * transfer_length_t	
Enumerator	
TRANSFER_SIZE_1_BYTE	Each transfer transfers a 8-bit value.
TRANSFER_SIZE_2_BYTE	Each transfer transfers a 16-bit value.
TRANSFER_SIZE_4_BYTE	Each transfer transfers a 32-bit value.

◆ transfer_addr_mode_t

enum transfer_addr_mode_t	
Address mode specifies whether to modify (increment or decrement) pointer after each transfer.	
Enumerator	
TRANSFER_ADDR_MODE_FIXED	Address pointer remains fixed after each transfer.
TRANSFER_ADDR_MODE_OFFSET	Offset is added to the address pointer after each transfer.
TRANSFER_ADDR_MODE_INCREMENTED	Address pointer is incremented by associated transfer_size_t after each transfer.
TRANSFER_ADDR_MODE_DECREMENTED	Address pointer is decremented by associated transfer_size_t after each transfer.

◆ transfer_repeat_area_t

enum transfer_repeat_area_t

Repeat area options (source or destination). In TRANSFER_MODE_REPEAT, the selected pointer returns to its original value after transfer_info_t::length transfers. In TRANSFER_MODE_BLOCK and TRANSFER_MODE_REPEAT_BLOCK, the selected pointer returns to its original value after each transfer.

Enumerator	
TRANSFER_REPEAT_AREA_DESTINATION	Destination area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK or TRANSFER_MODE_REPEAT_BLOCK.
TRANSFER_REPEAT_AREA_SOURCE	Source area repeated in TRANSFER_MODE_REPEAT or TRANSFER_MODE_BLOCK or TRANSFER_MODE_REPEAT_BLOCK.

◆ transfer_chain_mode_t

enum transfer_chain_mode_t

Chain transfer mode options.

Note

Only applies for DTC.

Enumerator	
TRANSFER_CHAIN_MODE_DISABLED	Chain mode not used.
TRANSFER_CHAIN_MODE_EACH	Switch to next transfer after a single transfer from this transfer_info_t.
TRANSFER_CHAIN_MODE_END	Complete the entire transfer defined in this transfer_info_t before chaining to next transfer.

◆ transfer_irq_t

enum transfer_irq_t	
Interrupt options.	
Enumerator	
TRANSFER_IRQ_END	Interrupt occurs only after last transfer. If this transfer is chained to a subsequent transfer, the interrupt will occur only after subsequent chained transfer(s) are complete. Warning DTC triggers the interrupt of the activation source. Choosing TRANSFER_IRQ_END with DTC will prevent activation source interrupts until the transfer is complete.
TRANSFER_IRQ_EACH	Interrupt occurs after each transfer. *Note* *Not available in all HAL drivers. See HAL driver for details.*

◆ transfer_start_mode_t

enum transfer_start_mode_t	
Select whether to start single or repeated transfer with software start.	
Enumerator	
TRANSFER_START_MODE_SINGLE	Software start triggers single transfer.
TRANSFER_START_MODE_REPEAT	Software start transfer continues until transfer is complete.

4.3.39 UART Interface

Interfaces

Detailed Description

Interface for UART communications.

Summary

The UART interface provides common APIs for UART HAL drivers. The UART interface supports the following features:

- Full-duplex UART communication
- Interrupt driven transmit/receive processing
- Callback function with returned event code
- Runtime baud-rate change
- Hardware resource locking during a transaction
- CTS/RTS hardware flow control support (with an associated IOPORT pin)

Implemented by:

- Serial Communications Interface (SCI) UART (r_sci_uart)
- Serial Communications Interface (SCI) UART (r_sci_b_uart)

Data Structures

struct	uart_info_t
struct	uart_callback_args_t
struct	uart_cfg_t
struct	uart_api_t
struct	uart_instance_t

Typedefs

typedef void	uart_ctrl_t

Enumerations

enum	uart_event_t
enum	uart_data_bits_t
enum	uart_parity_t
enum	uart_stop_bits_t
enum	uart_dir_t

Data Structure Documentation

◆ uart_info_t

struct uart_info_t
UART driver specific information

Data Fields		
uint32_t	write_bytes_max	Maximum bytes that can be written at this time. Only applies if uart_cfg_t::p_transfer_tx is not NULL.
uint32_t	read_bytes_max	Maximum bytes that are available to read at one time. Only applies if uart_cfg_t::p_transfer_rx is not NULL.

◆ uart_callback_args_t

struct uart_callback_args_t

UART Callback parameter definition

Data Fields		
uint32_t	channel	Device channel number.
uart_event_t	event	Event code.
uint32_t	data	Contains the next character received for the events UART_EVENT_RX_CHAR, UART_EVENT_ERR_PARITY, UART_EVENT_ERR_FRAMING, or UART_EVENT_ERR_OVERFLOW. Otherwise unused.
void const *	p_context	Context provided to user during callback.

◆ uart_cfg_t

struct uart_cfg_t

UART Configuration

Data Fields

uint8_t	channel
	Select a channel corresponding to the channel number of the hardware.
uart_data_bits_t	data_bits
	Data bit length (8 or 7 or 9)
uart_parity_t	parity

	Parity type (none or odd or even)
uart_stop_bits_t	stop_bits
	Stop bit length (1 or 2)
uint8_t	rxi_ipl
	Receive interrupt priority.
IRQn_Type	rxi_irq
	Receive interrupt IRQ number.
uint8_t	txi_ipl
	Transmit interrupt priority.
IRQn_Type	txi_irq
	Transmit interrupt IRQ number.
uint8_t	tei_ipl
	Transmit end interrupt priority.
IRQn_Type	tei_irq
	Transmit end interrupt IRQ number.
uint8_t	eri_ipl
	Error interrupt priority.
IRQn_Type	eri_irq
	Error interrupt IRQ number.

transfer_instance_t const *	p_transfer_rx
transfer_instance_t const *	p_transfer_tx
void(*	p_callback)(uart_callback_args_t *p_args)
	Pointer to callback function.
void const *	p_context
	User defined context passed into callback function.
void const *	p_extend
	UART hardware dependent configuration.

Field Documentation

◆ p_transfer_rx

transfer_instance_t const* uart_cfg_t::p_transfer_rx

Optional transfer instance used to receive multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the read API is limited to one byte at a time.

◆ p_transfer_tx

transfer_instance_t const* uart_cfg_t::p_transfer_tx

Optional transfer instance used to send multiple bytes without interrupts. Set to NULL if unused. If NULL, the number of bytes allowed in the write APIs is limited to one byte at a time.

◆ uart_api_t

struct uart_api_t

Shared Interface definition for UART

Data Fields

fsp_err_t(*	open)(uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)
fsp_err_t(*	read)(uart_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

fsp_err_t(*	write)(uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)
fsp_err_t(*	baudSet)(uart_ctrl_t *const p_ctrl, void const *const p_baudrate_info)
fsp_err_t(*	infoGet)(uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)
fsp_err_t(*	communicationAbort)(uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort)
fsp_err_t(*	callbackSet)(uart_ctrl_t *const p_api_ctrl, void(*p_callback)(uart_callback_args_t *), void const *const p_context, uart_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(uart_ctrl_t *const p_ctrl)
fsp_err_t(*	readStop)(uart_ctrl_t *const p_ctrl, uint32_t *remaining_bytes)

Field Documentation

◆ open

fsp_err_t(* uart_api_t::open) (uart_ctrl_t *const p_ctrl, uart_cfg_t const *const p_cfg)

Open UART device.

Implemented as

- R_SCI_UART_Open()
- R_SCI_B_UART_Open()

Parameters

[in,out]	p_ctrl	Pointer to the UART control block. Must be declared by user. Value set here.
[in]	uart_cfg_t	Pointer to UART configuration structure. All elements of this structure must be set by user.

◆ **read**

fsp_err_t(* uart_api_t::read) (uart_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Read from UART device. The read buffer is used until the read is complete. When a transfer is complete, the callback is called with event UART_EVENT_RX_COMPLETE. Bytes received outside an active transfer are received in the callback function with event UART_EVENT_RX_CHAR. The maximum transfer size is reported by infoGet().

Implemented as

- R_SCI_UART_Read()
- R_SCI_B_UART_Read()

Parameters

[in]	p_ctrl	Pointer to the UART control block for the channel.
[in]	p_dest	Destination address to read data from.
[in]	bytes	Read data length.

◆ **write**

fsp_err_t(* uart_api_t::write) (uart_ctrl_t *const p_ctrl, uint8_t const *const p_src, uint32_t const bytes)

Write to UART device. The write buffer is used until write is complete. Do not overwrite write buffer contents until the write is finished. When the write is complete (all bytes are fully transmitted on the wire), the callback called with event UART_EVENT_TX_COMPLETE. The maximum transfer size is reported by infoGet().

Implemented as

- R_SCI_UART_Write()
- R_SCI_B_UART_Write()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	p_src	Source address to write data to.
[in]	bytes	Write data length.

◆ baudSet

fsp_err_t(* uart_api_t::baudSet) (uart_ctrl_t *const p_ctrl, void const *const p_baudrate_info)

Change baud rate.

Warning

Calling this API aborts any in-progress transmission and disables reception until the new baud settings have been applied.

Implemented as

- R_SCI_UART_BaudSet()
- R_SCI_B_UART_BaudSet()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	p_baudrate_info	Pointer to module specific information for configuring baud rate.

◆ infoGet

fsp_err_t(* uart_api_t::infoGet) (uart_ctrl_t *const p_ctrl, uart_info_t *const p_info)

Get the driver specific information.

Implemented as

- R_SCI_UART_InfoGet()
- R_SCI_B_UART_InfoGet()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	baudrate	Baud rate in bps.

◆ communicationAbort

fsp_err_t(* uart_api_t::communicationAbort) (uart_ctrl_t *const p_ctrl, uart_dir_t communication_to_abort)

Abort ongoing transfer.

Implemented as

- R_SCI_UART_Abort()
- R_SCI_B_UART_Abort()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	communication_to_abort	Type of abort request.

◆ callbackSet

fsp_err_t(* uart_api_t::callbackSet) (uart_ctrl_t *const p_api_ctrl, void(*p_callback)(uart_callback_args_t *), void const *const p_context, uart_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_SCI_Uart_CallbackSet()
- R_SCI_B_Uart_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* uart_api_t::close) (uart_ctrl_t *const p_ctrl)

Close UART device.

Implemented as

- R_SCI_UART_Close()
- R_SCI_B_UART_Close()

Parameters

[in]	p_ctrl	Pointer to the UART control block.

◆ readStop

fsp_err_t(* uart_api_t::readStop) (uart_ctrl_t *const p_ctrl, uint32_t *remaining_bytes)

Stop ongoing read and return the number of bytes remaining in the read.

Implemented as

- R_SCI_UART_ReadStop()
- R_SCI_B_UART_ReadStop()

Parameters

[in]	p_ctrl	Pointer to the UART control block.
[in,out]	remaining_bytes	Pointer to location to store remaining bytes for read.

◆ uart_instance_t

struct uart_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
uart_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
uart_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
uart_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ uart_ctrl_t

typedef void uart_ctrl_t
UART control block. Allocate an instance specific control block to pass into the UART API calls. **Implemented as** - sci_uart_instance_ctrl_t

Enumeration Type Documentation

◆ uart_event_t

enum uart_event_t

UART Event codes

Enumerator	
UART_EVENT_RX_COMPLETE	Receive complete event.
UART_EVENT_TX_COMPLETE	Transmit complete event.
UART_EVENT_RX_CHAR	Character received.
UART_EVENT_ERR_PARITY	Parity error event.
UART_EVENT_ERR_FRAMING	Mode fault error event.
UART_EVENT_ERR_OVERFLOW	FIFO Overflow error event.
UART_EVENT_BREAK_DETECT	Break detect error event.
UART_EVENT_TX_DATA_EMPTY	Last byte is transmitting, ready for more data.

◆ uart_data_bits_t

enum uart_data_bits_t

UART Data bit length definition

Enumerator	
UART_DATA_BITS_9	Data bits 9-bit.
UART_DATA_BITS_8	Data bits 8-bit.
UART_DATA_BITS_7	Data bits 7-bit.

◆ uart_parity_t

enum uart_parity_t	
UART Parity definition	
Enumerator	
UART_PARITY_OFF	No parity.
UART_PARITY_EVEN	Even parity.
UART_PARITY_ODD	Odd parity.

◆ uart_stop_bits_t

enum uart_stop_bits_t	
UART Stop bits definition	
Enumerator	
UART_STOP_BITS_1	Stop bit 1-bit.
UART_STOP_BITS_2	Stop bits 2-bit.

◆ uart_dir_t

enum uart_dir_t	
UART transaction definition	
Enumerator	
UART_DIR_RX_TX	Both RX and TX.
UART_DIR_RX	Only RX.
UART_DIR_TX	Only TX.

4.3.40 USB Interface

Interfaces

Detailed Description

Interface for USB functions.

Summary

The USB interface provides USB functionality.

The USB interface can be implemented by:

- USB (r_usb_basic)

Data Structures

struct	usb_cfg_t
struct	usb_api_t
struct	usb_instance_t

Macros

#define	USB_BREQUEST b15-8
#define	USB_GET_STATUS USB Standard request Get Status.
#define	USB_CLEAR_FEATURE USB Standard request Clear Feature.
#define	USB_REQRESERVED USB Standard request Reqreserved.
#define	USB_SET_FEATURE USB Standard request Set Feature.
#define	USB_REQRESERVED1 USB Standard request Reqreserved1.
#define	USB_SET_ADDRESS USB Standard request Set Address.

#define USB_GET_DESCRIPTOR

USB Standard request Get Descriptor.

#define USB_SET_DESCRIPTOR

USB Standard request Set Descriptor.

#define USB_GET_CONFIGURATION

USB Standard request Get Configuration.

#define USB_SET_CONFIGURATION

USB Standard request Set Configuration.

#define USB_GET_INTERFACE

USB Standard request Get Interface.

#define USB_SET_INTERFACE

USB Standard request Set Interface.

#define USB_SYNCH_FRAME

USB Standard request Synch Frame.

#define USB_HOST_TO_DEV

From host to device.

#define USB_DEV_TO_HOST

From device to host.

#define USB_STANDARD

Standard Request.

#define USB_CLASS

Class Request.

#define USB_VENDOR

	Vendor Request.
#define	USB_DEVICE Device.
#define	USB_INTERFACE Interface.
#define	USB_ENDPOINT End Point.
#define	USB_OTHER Other.
#define	USB_NULL NULL pointer.
#define	USB_IP0 USB0 module.
#define	USB_IP1 USB1 module.
#define	USB_PIPE0 Pipe Number0.
#define	USB_PIPE1 Pipe Number1.
#define	USB_PIPE2 Pipe Number2.
#define	USB_PIPE3 Pipe Number3.

#define	USB_PIPE4 Pipe Number4.
#define	USB_PIPE5 Pipe Number5.
#define	USB_PIPE6 Pipe Number6.
#define	USB_PIPE7 Pipe Number7.
#define	USB_PIPE8 Pipe Number8.
#define	USB_PIPE9 Pipe Number9.
#define	USB_EP0 End Point Number0.
#define	USB_EP1 End Point Number1.
#define	USB_EP2 End Point Number2.
#define	USB_EP3 End Point Number3.
#define	USB_EP4 End Point Number4.

#define USB_EP5

End Point Number5.

#define USB_EP6

End Point Number6.

#define USB_EP7

End Point Number7.

#define USB_EP8

End Point Number8.

#define USB_EP9

End Point Number9.

#define USB_EP10

End Point Number10.

#define USB_EP11

End Point Number11.

#define USB_EP12

End Point Number12.

#define USB_EP13

End Point Number13.

#define USB_EP14

End Point Number14.

#define USB_EP15

End Point Number15.

#define USB_EP_DIR

b7: Endpoint Direction

#define USB_EP_DIR_IN

b7: Endpoint Direction In

#define USB_EP_DIR_OUT

b7: Endpoint Direction Out

#define USB_DT_DEVICE

Device Descriptor.

#define USB_DT_CONFIGURATION

Configuration Descriptor.

#define USB_DT_STRING

String Descriptor.

#define USB_DT_INTERFACE

Interface Descriptor.

#define USB_DT_ENDPOINT

Endpoint Descriptor.

#define USB_DT_DEVICE_QUALIFIER

Device Qualifier Descriptor.

#define USB_DT_OTHER_SPEED_CONF

Other Speed Configuration Descriptor.

#define USB_DT_INTERFACE_POWER

Interface Power Descriptor.

#define USB_DT_OTGDESCRIPTOR

OTG Descriptor.

#define	USB_DT_HUBDESCRIPTOR HUB descriptor.
#define	USB_IFCLS_NOT Un corresponding Class.
#define	USB_IFCLS_AUD Audio Class.
#define	USB_IFCLS_CDC CDC Class.
#define	USB_IFCLS_CDCC CDC-Control Class.
#define	USB_IFCLS_HID HID Class.
#define	USB_IFCLS_PHY Physical Class.
#define	USB_IFCLS_IMG Image Class.
#define	USB_IFCLS_PRN Printer Class.
#define	USB_IFCLS_MAS Mass Storage Class.
#define	USB_IFCLS_HUB HUB Class.

#define USB_IFCLS_CDCD

CDC-Data Class.

#define USB_IFCLS_CHIP

Chip/Smart Card Class.

#define USB_IFCLS_CNT

Content-Security Class.

#define USB_IFCLS_VID

Video Class.

#define USB_IFCLS_DIAG

Diagnostic Device.

#define USB_IFCLS_WIRE

Wireless Controller.

#define USB_IFCLS_APL

Application-Specific.

#define USB_IFCLS_VEN

Vendor-Specific Class.

#define USB_EP_IN

In Endpoint.

#define USB_EP_OUT

Out Endpoint.

#define USB_EP_ISO

Isochronous Transfer.

#define USB_EP_BULK

	Bulk Transfer.
#define	USB_EP_INT Interrupt Transfer.
#define	USB_CF_RESERVED Reserved(set to 1)
#define	USB_CF_SELFP Self Powered.
#define	USB_CF_BUSP Bus Powered.
#define	USB_CF_RWUPON Remote Wake up ON.
#define	USB_CF_RWUPOFF Remote Wake up OFF.
#define	USB_DD_BLENGTH Device Descriptor Length.
#define	USB_CD_BLENGTH Configuration Descriptor Length.
#define	USB_ID_BLENGTH Interface Descriptor Length.
#define	USB_ED_BLENGTH Endpoint Descriptor Length.

Typedefs

typedef void	usb_ctrl_t

Enumerations

enum	usb_speed_t
enum	usb_setup_status_t
enum	usb_status_t
enum	usb_class_t
enum	usb_bcport_t
enum	usb_onoff_t
enum	usb_transfer_t
enum	usb_transfer_type_t
enum	usb_mode_t
enum	usb_compliancetest_status_t

Data Structure Documentation

◆ usb_cfg_t

struct usb_cfg_t

USB configuration.

Data Fields		
usb_mode_t	usb_mode	USB_MODE_HOST/USB_MODE_PERI.
usb_speed_t	usb_speed	USB speed (USB_HS/USB_FS/USB_LS)
uint8_t	module_number	USB module number (USB_IP0/USB_IP1)
usb_class_t	type	USB device class etc.
usb_descriptor_t *	p_usb_reg	Pointer to the usb_decriptor_t structure area.
usb_compliance_cb_t *	usb_complience_cb	
IRQn_Type	irq	USBI dedicated interrupt number storage variable.
IRQn_Type	irq_r	USBR dedicated interrupt number storage variable.
IRQn_Type	irq_d0	FS D0FIFO dedicated interrupt

		number storage variable.
IRQn_Type	irq_d1	FS D1FIFO dedicated interrupt number storage variable.
IRQn_Type	hsirq	USBIR dedicated interrupt number storage variable.
IRQn_Type	hsirq_d0	HS D0FIFO dedicated interrupt number storage variable.
IRQn_Type	hsirq_d1	HS D1FIFO dedicated interrupt number storage variable.
uint8_t	ipl	Variable to store the interrupt priority of USBI.
uint8_t	ipl_r	Variable to store the interrupt priority of USBR.
uint8_t	ipl_d0	Variable to store the interrupt priority of FS D0FIFO.
uint8_t	ipl_d1	Variable to store the interrupt priority of FS D1FIFO.
uint8_t	hsipl	Variable to store the interrupt priority of USBIR.
uint8_t	hsipl_d0	Variable to store the interrupt priority of HS D0FIFO.
uint8_t	hsipl_d1	Variable to store the interrupt priority of HS D1FIFO.
usb_callback_t *	p_usb_apl_callback	Application Callback.
void const *	p_context	Other Context.
const transfer_instance_t *	p_transfer_tx	Send context.
const transfer_instance_t *	p_transfer_rx	Receive context.

◆ usb_api_t

struct usb_api_t	
Functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(usb_ctrl_t *const p_api_ctrl, usb_cfg_t const *const p_cfg)
fsp_err_t(*	close)(usb_ctrl_t *const p_api_ctrl)
fsp_err_t(*	read)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t destination)

fsp_err_t(*	write)(usb_ctrl_t *const p_api_ctrl, uint8_t const *const p_buf, uint32_t size, uint8_t destination)
fsp_err_t(*	stop)(usb_ctrl_t *const p_api_ctrl, usb_transfer_t direction, uint8_t destination)
fsp_err_t(*	suspend)(usb_ctrl_t *const p_api_ctrl)
fsp_err_t(*	resume)(usb_ctrl_t *const p_api_ctrl)
fsp_err_t(*	vbusSet)(usb_ctrl_t *const p_api_ctrl, uint16_t state)
fsp_err_t(*	infoGet)(usb_ctrl_t *const p_api_ctrl, usb_info_t *p_info, uint8_t destination)
fsp_err_t(*	pipeRead)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)
fsp_err_t(*	pipeWrite)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)
fsp_err_t(*	pipeStop)(usb_ctrl_t *const p_api_ctrl, uint8_t pipe_number)
fsp_err_t(*	usedPipesGet)(usb_ctrl_t *const p_api_ctrl, uint16_t *p_pipe, uint8_t destination)
fsp_err_t(*	pipeInfoGet)(usb_ctrl_t *const p_api_ctrl, usb_pipe_t *p_info, uint8_t pipe_number)
fsp_err_t(*	eventGet)(usb_ctrl_t *const p_api_ctrl, usb_status_t *event)
fsp_err_t(*	callback)(usb_callback_t *p_callback)
fsp_err_t(*	pullUp)(usb_ctrl_t *const p_api_ctrl, uint8_t state)
fsp_err_t(*	hostControlTransfer)(usb_ctrl_t *const p_api_ctrl, usb_setup_t *p_setup, uint8_t *p_buf, uint8_t device_address)

fsp_err_t(*	periControlDataGet)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size)
fsp_err_t(*	periControlDataSet)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size)
fsp_err_t(*	periControlStatusSet)(usb_ctrl_t *const p_api_ctrl, usb_setup_status_t status)
fsp_err_t(*	remoteWakeup)(usb_ctrl_t *const p_api_ctrl)
fsp_err_t(*	moduleNumberGet)(usb_ctrl_t *const p_api_ctrl, uint8_t *module_number)
fsp_err_t(*	classTypeGet)(usb_ctrl_t *const p_api_ctrl, usb_class_t *class_type)
fsp_err_t(*	deviceAddressGet)(usb_ctrl_t *const p_api_ctrl, uint8_t *device_address)
fsp_err_t(*	pipeNumberGet)(usb_ctrl_t *const p_api_ctrl, uint8_t *pipe_number)
fsp_err_t(*	deviceStateGet)(usb_ctrl_t *const p_api_ctrl, uint16_t *state)
fsp_err_t(*	dataSizeGet)(usb_ctrl_t *const p_api_ctrl, uint32_t *data_size)
fsp_err_t(*	setupGet)(usb_ctrl_t *const p_api_ctrl, usb_setup_t *setup)

Field Documentation

◆ open

fsp_err_t(* usb_api_t::open) (usb_ctrl_t *const p_api_ctrl, usb_cfg_t const *const p_cfg)

Start the USB module

Implemented as

- R_USB_Open()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* usb_api_t::close) (usb_ctrl_t *const p_api_ctrl)

Stop the USB module

Implemented as

- R_USB_Close()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ read

fsp_err_t(* usb_api_t::read) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t destination)

Request USB data read

Implemented as

- R_USB_Read()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores read data.
[in]	size	Read request size.
[in]	destination	In Host mode, it represents the device address, and in Peripheral mode, it represents the device class.

◆ write

fsp_err_t(* usb_api_t::write) (usb_ctrl_t *const p_api_ctrl, uint8_t const *const p_buf, uint32_t size, uint8_t destination)

Request USB data write

Implemented as

- R_USB_Write()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores write data.
[in]	size	Read request size.
[in]	destination	In Host mode, it represents the device address, and in Peripheral mode, it represents the device class.

◆ stop

fsp_err_t(* usb_api_t::stop) (usb_ctrl_t *const p_api_ctrl, usb_transfer_t direction, uint8_t destination)

Stop USB data read/write processing

Implemented as

- R_USB_Stop()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	direction	Receive (USB_TRANSFER_READ) or send (USB_TRANSFER_WRITE).
[in]	destination	In Host mode, it represents the device address, and in Peripheral mode, it represents the device class.

◆ suspend

fsp_err_t(* usb_api_t::suspend) (usb_ctrl_t *const p_api_ctrl)

Request suspend

Implemented as

- R_USB_Suspend()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ resume

fsp_err_t(* usb_api_t::resume) (usb_ctrl_t *const p_api_ctrl)

Request resume

Implemented as

- R_USB_Resume()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ vbusSet

fsp_err_t(* usb_api_t::vbusSet) (usb_ctrl_t *const p_api_ctrl, uint16_t state)

Sets VBUS supply start/stop.

Implemented as

- R_USB_VbusSet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	state	VBUS supply start/stop specification

◆ infoGet

fsp_err_t(* usb_api_t::infoGet) (usb_ctrl_t *const p_api_ctrl, usb_info_t *p_info, uint8_t destination)

Get information on USB device.

Implemented as

- R_USB_InfoGet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_info	Pointer to usb_info_t structure area.
[in]	destination	Device address for Host.

◆ pipeRead

fsp_err_t(* usb_api_t::pipeRead) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)

Request data read from specified pipe

Implemented as

- R_USB_PipeRead()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores read data.
[in]	size	Read request size.
[in]	pipe_number	Pipe Number.

◆ pipeWrite

fsp_err_t(* usb_api_t::pipeWrite) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size, uint8_t pipe_number)

Request data write to specified pipe

Implemented as

- R_USB_PipeWrite()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_buf	Pointer to area that stores write data.
[in]	size	Read request size.
[in]	pipe_number	Pipe Number.

◆ pipeStop

fsp_err_t(* usb_api_t::pipeStop) (usb_ctrl_t *const p_api_ctrl, uint8_t pipe_number)

Stop USB data read/write processing to specified pipe

Implemented as

- R_USB_PipeStop()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	pipe_number	Pipe Number.

◆ usedPipesGet

fsp_err_t(* usb_api_t::usedPipesGet) (usb_ctrl_t *const p_api_ctrl, uint16_t *p_pipe, uint8_t destination)

Get pipe number

Implemented as

- R_USB_UsedPipesGet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_pipe	Pointer to area that stores the selected pipe number (bit map information).
[in]	destination	Device address for Host.

◆ pipeInfoGet

fsp_err_t(* usb_api_t::pipeInfoGet) (usb_ctrl_t *const p_api_ctrl, usb_pipe_t *p_info, uint8_t pipe_number)

Get pipe information

Implemented as

- R_USB_PipeInfoGet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_info	Pointer to usb_pipe_t structure area.
[in]	pipe_number	Pipe Number.

◆ eventGet

fsp_err_t(* usb_api_t::eventGet) (usb_ctrl_t *const p_api_ctrl, usb_status_t *event)

Return USB-related completed events (OS less only)

Implemented as

- R_USB_EventGet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[out]	event	Pointer to event.

◆ callback

fsp_err_t(* usb_api_t::callback) (usb_callback_t *p_callback)

Register a callback function to be called upon completion of a USB related event. (RTOS only)

Implemented as

- R_USB_Callback()

Parameters

[in]	p_callback	Pointer to Callback function.

◆ pullUp

fsp_err_t(* usb_api_t::pullUp) (usb_ctrl_t *const p_api_ctrl, uint8_t state)

Pull-up enable/disable setting of D+/D- line.

Implemented as

- R_USB_PullUp()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	state	Pull-up enable/disable setting.

◆ hostControlTransfer

fsp_err_t(* usb_api_t::hostControlTransfer) (usb_ctrl_t *const p_api_ctrl, usb_setup_t *p_setup, uint8_t *p_buf, uint8_t device_address)

Performs settings and transmission processing when transmitting a setup packet.

Implemented as

- R_USB_HostControlTransfer()

Parameters

[in]	p_api_ctrl	USB control structure.
[in]	p_setup	Setup packet information.
[in]	p_buf	Transfer area information.
[in]	device_address	Device address information.

◆ periControlDataGet

fsp_err_t(* usb_api_t::periControlDataGet) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size)

Receives data sent by control transfer.

Implemented as

- R_USB_PeriControlDataGet()

Parameters

[in]	p_api_ctrl	USB control structure.
[in]	p_buf	Data reception area information.
[in]	size	Data reception size information.

◆ periControlDataSet

fsp_err_t(* usb_api_t::periControlDataSet) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_buf, uint32_t size)

Performs transfer processing for control transfer.

Implemented as

- R_USB_PeriControlDataSet()

Parameters

[in]	p_api_ctrl	USB control structure.
[in]	p_buf	Area information for data transfer.
[in]	size	Transfer size information.

◆ periControlStatusSet

fsp_err_t(* usb_api_t::periControlStatusSet) (usb_ctrl_t *const p_api_ctrl, usb_setup_status_t status)

Set the response to the setup packet.

Implemented as

- R_USB_PeriControlStatusSet()

Parameters

[in]	p_api_ctrl	USB control structure.
[in]	status	USB port startup information.

◆ remoteWakeup

fsp_err_t(* usb_api_t::remoteWakeup) (usb_ctrl_t *const p_api_ctrl)

Sends a remote wake-up signal to the connected Host.

Implemented as

- R_USB_RemoteWakeup()

Parameters

[in]	p_api_ctrl	USB control structure.

◆ moduleNumberGet

fsp_err_t(* usb_api_t::moduleNumberGet) (usb_ctrl_t *const p_api_ctrl, uint8_t *module_number)

This API gets the module number.

Implemented as

- R_USB_ModuleNumberGet()

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	module_number	Module number to get.

◆ classTypeGet

fsp_err_t(* usb_api_t::classTypeGet) (usb_ctrl_t *const p_api_ctrl, usb_class_t *class_type)

This API gets the module number.

Implemented as

- R_USB_ClassTypeGet()

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	class_type	Class type to get.

◆ deviceAddressGet

fsp_err_t(* usb_api_t::deviceAddressGet) (usb_ctrl_t *const p_api_ctrl, uint8_t *device_address)

This API gets the device address.

Implemented as

- R_USB_DeviceAddressGet()

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	device_address	Device address to get.

◆ pipeNumberGet

fsp_err_t(* usb_api_t::pipeNumberGet) (usb_ctrl_t *const p_api_ctrl, uint8_t *pipe_number)

This API gets the pipe number.

Implemented as

- R_USB_PipeNumberGet()

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	pipe_number	Pipe number to get.

◆ deviceStateGet

fsp_err_t(* usb_api_t::deviceStateGet) (usb_ctrl_t *const p_api_ctrl, uint16_t *state)

This API gets the state of the device.

Implemented as

- R_USB_DeviceStateGet()

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	state	Device state to get.

◆ dataSizeGet

fsp_err_t(* usb_api_t::dataSizeGet) (usb_ctrl_t *const p_api_ctrl, uint32_t *data_size)

This API gets the data size.

Implemented as

- R_USB_DataSizeGet()

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	data_size	Data size to get.

◆ setupGet

fsp_err_t(* usb_api_t::setupGet) (usb_ctrl_t *const p_api_ctrl, usb_setup_t *setup)

This API gets the setup type.

Implemented as

- R_USB_SetupGet()

Parameters

[in]	p_api_ctrl	USB control structure.
[out]	setup	Setup type to get.

◆ usb_instance_t

struct usb_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
usb_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
usb_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
usb_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ usb_ctrl_t

typedef void usb_ctrl_t

USB control block. Allocate an instance specific control block to pass into the USB API calls.

Implemented as

- usb_instance_ctrl_t

Enumeration Type Documentation

◆ usb_speed_t

enum usb_speed_t	
USB speed type	
Enumerator	
USB_SPEED_LS	Low speed operation.
USB_SPEED_FS	Full speed operation.
USB_SPEED_HS	Hi speed operation.

◆ usb_setup_status_t

enum usb_setup_status_t	
USB request result	
Enumerator	
USB_SETUP_STATUS_ACK	ACK response.
USB_SETUP_STATUS_STALL	STALL response.

◆ usb_status_t

enum usb_status_t	
USB driver status	
Enumerator	
USB_STATUS_POWERED	Powered State.
USB_STATUS_DEFAULT	Default State.
USB_STATUS_ADDRESS	Address State.
USB_STATUS_CONFIGURED	Configured State.
USB_STATUS_SUSPEND	Suspend State.
USB_STATUS_RESUME	Resume State.
USB_STATUS_DETACH	Detach State.
USB_STATUS_REQUEST	Request State.
USB_STATUS_REQUEST_COMPLETE	Request Complete State.
USB_STATUS_READ_COMPLETE	Read Complete State.
USB_STATUS_WRITE_COMPLETE	Write Complete State.
USB_STATUS_BC	battery Charge State
USB_STATUS_OVERCURRENT	Over Current state.
USB_STATUS_NOT_SUPPORT	Device Not Support.
USB_STATUS_NONE	None Status.
USB_STATUS_MSC_CMD_COMPLETE	MSC_CMD Complete.

◆ usb_class_t

enum usb_class_t	
USB class type	
Enumerator	
USB_CLASS_PCDC	PCDC Class.
USB_CLASS_PCDCC	PCDCC Class.
USB_CLASS_PCDC2	PCDC2 Class.
USB_CLASS_PCDCC2	PCDCC2 Class.
USB_CLASS_PHID	PHID Class.
USB_CLASS_PAUD	PAUD Class.
USB_CLASS_PVND	PVND Class.
USB_CLASS_HCDC	HCDC Class.
USB_CLASS_HCDCC	HCDCC Class.
USB_CLASS_HHID	HHID Class.
USB_CLASS_HVND	HVND Class.
USB_CLASS_HMSC	HMSC Class.
USB_CLASS_PMSC	PMSC Class.
USB_CLASS_REQUEST	USB Class Request.
USB_CLASS_END	USB Class End Code.

◆ usb_bcport_t

enum usb_bcport_t	
USB battery charging type	
Enumerator	
USB_BCPORT_SDP	SDP port settings.
USB_BCPORT_CDP	CDP port settings.
USB_BCPORT_DCP	DCP port settings.

◆ usb_onoff_t

enum usb_onoff_t	
USB status	
Enumerator	
USB_OFF	USB Off State.
USB_ON	USB On State.

◆ usb_transfer_t

enum usb_transfer_t	
USB read/write type	
Enumerator	
USB_TRANSFER_READ	Data Receive communication.
USB_TRANSFER_WRITE	Data transmission communication.

◆ usb_transfer_type_t

enum usb_transfer_type_t	
USB transfer type	
Enumerator	
USB_TRANSFER_TYPE_BULK	Bulk communication.
USB_TRANSFER_TYPE_INT	Interrupt communication.
USB_TRANSFER_TYPE_ISO	Isochronous communication.

◆ usb_mode_t

enum usb_mode_t	
Enumerator	
USB_MODE_HOST	Host mode.
USB_MODE_PERI	Peripheral mode.

◆ usb_compliancetest_status_t

enum usb_compliancetest_status_t	
Enumerator	
USB_COMPLIANCETEST_ATTACH	Device Attach Detection.
USB_COMPLIANCETEST_DETACH	Device Detach Detection.
USB_COMPLIANCETEST_TPL	TPL device connect.
USB_COMPLIANCETEST_NOTTPL	Not TPL device connect.
USB_COMPLIANCETEST_HUB	USB Hub connect.
USB_COMPLIANCETEST_OVRC	Over current.
USB_COMPLIANCETEST_NORES	Response Time out for Control Read Transfer.
USB_COMPLIANCETEST_SETUP_ERR	Setup Transaction Error.

4.3.41 USB HCDC Interface

Interfaces

Detailed Description

Interface for USB HCDC functions.

Summary

The USB HCDC interface provides USB HCDC functionality.

The USB HCDC interface can be implemented by:

- USB Host Communications Device Class Driver (r_usb_hcdc)

Data Structures

struct	usb_hcdc_encapsulated_t
struct	usb_hcdc_abstractstate_t
struct	usb_hcdc_countrysetting_t
union	usb_hcdc_commfeature_t
struct	usb_hcdc_linecoding_t
struct	usb_hcdc_controllinestate_t
struct	usb_hcdc_serialstate_t
struct	usb_hcdc_breakduration_t

Enumerations

enum	usb_hcdc_data_bit_t
enum	usb_hcdc_stop_bit_t
enum	usb_hcdc_parity_bit_t
enum	usb_hcdc_line_speed_t
enum	usb_hcdc_feature_selector_t

Data Structure Documentation

◆ usb_hcdc_encapsulated_t

struct usb_hcdc_encapsulated_t		
Encapsulated data		
Data Fields		
uint8_t *	p_data	Protocol dependent data.
uint16_t	wlength	Data length in bytes.

◆ usb_hcdc_abstractstate_t

struct usb_hcdc_abstractstate_t		
Abstract Control Model (ACM) settings bitmap		
Data Fields		
uint16_t	bis: 1	Idle enable.
uint16_t	bdms: 1	Data multiplexing enable.
uint16_t	rsv: 14	Reserved.

◆ usb_hcdc_countrysetting_t

struct usb_hcdc_countrysetting_t		
Country code data		
Data Fields		
uint16_t	country_code	Country code.

◆ usb_hcdc_commfeature_t

union usb_hcdc_commfeature_t		
Feature setting data		
Data Fields		
usb_hcdc_abstractstate_t	abstract_state	ACM settings bitmap.
usb_hcdc_countrysetting_t	country_setting	Country code.

◆ usb_hcdc_linecoding_t

struct usb_hcdc_linecoding_t		
Virtual UART configuration (line coding)		
Data Fields		
usb_hcdc_line_speed_t	dwdte_rate	Data terminal rate in bits per second.
usb_hcdc_stop_bit_t	bchar_format	Stop bits.
usb_hcdc_parity_bit_t	bparity_type	Parity.
usb_hcdc_data_bit_t	bdata_bits	Data bits.

uint8_t	rsv	Reserved.

◆ usb_hcdc_controllinestate_t

struct usb_hcdc_controllinestate_t		
Virtual UART control signal bitmap		
Data Fields		
uint16_t	bdtr: 1	DTR.
uint16_t	brts: 1	RTS.
uint16_t	rsv: 14	Reserved.

◆ usb_hcdc_serialstate_t

struct usb_hcdc_serialstate_t		
Virtual UART state bitmap		
Data Fields		
uint16_t	brx_carrier: 1	DCD signal.
uint16_t	btx_carrier: 1	DSR signal.
uint16_t	bbreak: 1	Break detection status.
uint16_t	bring_signal: 1	Ring signal.
uint16_t	bframing: 1	Framing error.
uint16_t	bparity: 1	Parity error.
uint16_t	bover_run: 1	Over Run error.
uint16_t	rsv: 9	Reserved.

◆ usb_hcdc_breakduration_t

struct usb_hcdc_breakduration_t		
Break duration data		
Data Fields		
uint16_t	wtime_ms	Duration of Break.

Enumeration Type Documentation

◆ usb_hcdc_data_bit_t

enum usb_hcdc_data_bit_t	
Virtual UART data length	
Enumerator	
USB_HCDC_DATA_BIT_7	7 bits
USB_HCDC_DATA_BIT_8	8 bits

◆ usb_hcdc_stop_bit_t

enum usb_hcdc_stop_bit_t	
Virtual UART stop bit length	
Enumerator	
USB_HCDC_STOP_BIT_1	1 bit
USB_HCDC_STOP_BIT_15	1.5 bits
USB_HCDC_STOP_BIT_2	2 bits

◆ usb_hcdc_parity_bit_t

enum usb_hcdc_parity_bit_t	
Virtual UART parity bit setting	
Enumerator	
USB_HCDC_PARITY_BIT_NONE	No parity bit.
USB_HCDC_PARITY_BIT_ODD	Odd parity.
USB_HCDC_PARITY_BIT_EVEN	Even parity.

◆ usb_hcdc_line_speed_t

enum usb_hcdc_line_speed_t
Virtual UART bitrate

◆ usb_hcdc_feature_selector_t

enum usb_hcdc_feature_selector_t
Feature Selector

4.3.42 USB HHID Interface

Interfaces

Detailed Description

Interface for USB HHID functions.

Summary

The USB HHID interface provides USB HHID functionality.

The USB HHID interface can be implemented by:

- USB Host Human Interface Device Class Driver (r_usb_hhid)

Data Structures

struct	usb_hhid_api_t
struct	usb_hhid_instance_t

Macros

#define	USB_HID_OTHER Other.
#define	USB_HID_KEYBOARD Keyboard.
#define	USB_HID_MOUSE Mouse.
#define	USB_HID_IN In Transfer.

#define	USB_HID_OUT
	Out Transfer.

Data Structure Documentation

◆ usb_hhid_api_t

struct usb_hhid_api_t

USB HHID functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	typeGet)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_type, uint8_t device_address)
fsp_err_t(*	maxPacketSizeGet)(usb_ctrl_t *const p_api_ctrl, uint16_t *p_size, uint8_t direction, uint8_t device_address)

Field Documentation

◆ typeGet

fsp_err_t(* usb_hhid_api_t::typeGet) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_type, uint8_t device_address)

Get HID protocol.(USB Mouse/USB Keyboard/Other Type.)

Implemented as

- R_USB_HHID_TypeGet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_type	Pointer to store HID protocol value.
[in]	device_address	Device Address.

◆ **maxPacketSizeGet**

fsp_err_t(* usb_hhid_api_t::maxPacketSizeGet) (usb_ctrl_t *const p_api_ctrl, uint16_t *p_size, uint8_t direction, uint8_t device_address)

Obtains max packet size for the connected HID device. The max packet size is set to the area. Set the direction (USB_HID_IN/USB_HID_OUT).

Implemented as

- R_USB_HHID_MaxPacketSizeGet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_size	Pointer to the area to store the max package size.
[in]	direction	Transfer direction.
[in]	device_address	Device Address.

◆ **usb_hhid_instance_t**

struct usb_hhid_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
usb_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
usb_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
usb_hhid_api_t const *	p_api	Pointer to the API structure for this instance.

4.3.43 USB HMSC Interface

Interfaces

Detailed Description

Interface for USB HMSC functions.

Summary

The USB HMSC interface provides USB HMSC functionality.

The USB HMSC interface can be implemented by:

- USB Host Mass Storage Class Driver (r_usb_hmsc)

Data Structures

struct	usb_hmsc_api_t

Enumerations

enum	usb_atapi_t
enum	usb_csw_result_t

Data Structure Documentation

◆ usb_hmsc_api_t

struct usb_hmsc_api_t

USB HMSC functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	storageCommand)(usb_ctrl_t *const p_api_ctrl, uint8_t *buf, uint8_t command, uint8_t destination)
fsp_err_t(*	driveNumberGet)(usb_ctrl_t *const p_api_ctrl, uint8_t *p_drive, uint8_t destination)
fsp_err_t(*	storageReadSector)(uint16_t drive_number, uint8_t *const buff, uint32_t sector_number, uint16_t sector_count)
fsp_err_t(*	storageWriteSector)(uint16_t drive_number, uint8_t const *const buff, uint32_t sector_number, uint16_t sector_count)
fsp_err_t(*	semaphoreGet)(void)
fsp_err_t(*	semaphoreRelease)(void)

Field Documentation

◆ storageCommand

fsp_err_t(* usb_hmsc_api_t::storageCommand) (usb_ctrl_t *const p_api_ctrl, uint8_t *buf, uint8_t command, uint8_t destination)

Processing for MassStorage(ATAPI) command.

Implemented as

- R_USB_HMSC_StorageCommand()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	*buf	Pointer to the buffer area to store the transfer data.
[in]	command	ATAPI command.
[in]	destination	Represents a device address.

◆ driveNumberGet

fsp_err_t(* usb_hmsc_api_t::driveNumberGet) (usb_ctrl_t *const p_api_ctrl, uint8_t *p_drive, uint8_t destination)

Get number of Storage drive.

Implemented as

- R_USB_HMSC_DriveNumberGet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[out]	p_drive	Store address for Drive No.
[in]	destination	Represents a device address.

◆ storageReadSector

fsp_err_t(* usb_hmsc_api_t::storageReadSector) (uint16_t drive_number, uint8_t *const buff, uint32_t sector_number, uint16_t sector_count)

Read sector information.

Implemented as

- R_USB_HMSC_StorageReadSector()

Parameters

[in]	drive_number	Drive number.
[out]	*buff	Pointer to the buffer area to store the transfer data.
[in]	sector_number	The sector number to start with.
[in]	sector_count	Transmit with the sector size of the number of times.

◆ storageWriteSector

fsp_err_t(* usb_hmsc_api_t::storageWriteSector) (uint16_t drive_number, uint8_t const *const buff, uint32_t sector_number, uint16_t sector_count)

Write sector information.

Implemented as

- R_USB_HMSC_StorageWriteSector()

Parameters

[in]	drive_number	Drive number.
[in]	*buff	Pointer to the buffer area to store the transfer data.
[in]	sector_number	The sector number to start with.
[in]	sector_count	Transmit with the sector size of the number of times.

◆ semaphoreGet

fsp_err_t(* usb_hmsc_api_t::semaphoreGet) (void)

Get Semaphore.

Implemented as

- R_USB_HMSC_SemaphoreGet()

◆ semaphoreRelease

fsp_err_t(* usb_hmsc_api_t::semaphoreRelease) (void)

Release Semaphore.

Implemented as

- R_USB_HMSC_SemaphoreRelease()

Enumeration Type Documentation

◆ usb_atapi_t

enum usb_atapi_t	
ATAPI commands	
Enumerator	
USB_ATAPI_TEST_UNIT_READY	Test Unit Ready.
USB_ATAPI_REQUEST_SENSE	Request Sense.
USB_ATAPI_FORMAT_UNIT	Format Unit.
USB_ATAPI_INQUIRY	Inquiry.
USB_ATAPI_MODE_SELECT6	Mode Select6.
USB_ATAPI_MODE_SENSE6	Mode Sense6.
USB_ATAPI_START_STOP_UNIT	Start Stop Unit.
USB_ATAPI_PREVENT_ALLOW	Prevent Allow.
USB_ATAPI_READ_FORMAT_CAPACITY	Read Format Capacity.
USB_ATAPI_READ_CAPACITY	Read Capacity.
USB_ATAPI_READ10	Read10.
USB_ATAPI_WRITE10	Write10.
USB_ATAPI_SEEK	Seek.
USB_ATAPI_WRITE_AND_VERIFY	Write and Verify.
USB_ATAPI_VERIFY10	Verify10.
USB_ATAPI_MODE_SELECT10	Mode Select10.
USB_ATAPI_MODE_SENSE10	Mode Sense10.

◆ usb_csw_result_t

enum usb_csw_result_t	
Command Status Wrapper (CSW)	
Enumerator	
USB_CSW_RESULT_SUCCESS	CSW was successful.
USB_CSW_RESULT_FAIL	CSW failed.
USB_CSW_RESULT_PHASE	CSW has phase error.

4.3.44 USB PCDC Interface

Interfaces

Detailed Description

Interface for USB PCDC functions.

Summary

The USB PCDC interface provides USB PCDC functionality.

The USB PCDC interface can be implemented by:

- USB Peripheral Communications Device Class (r_usb_pcdc)

Data Structures

struct	usb_serial_state_bitmap_t
union	usb_sci_serialstate_t
struct	usb_pcdc_linecoding_t
struct	usb_pcdc_ctrllinestate_t

Macros

#define	USB_PCDC_SET_LINE_CODING Set Line Coding.
#define	USB_PCDC_GET_LINE_CODING

	Get Line Coding.
#define	USB_PCDC_SET_CONTROL_LINE_STATE Control Line State.
#define	USB_PCDC_SERIAL_STATE Serial State Code.
#define	USB_PCDC_SETUP_TBL_BSIZE Setup packet table size (uint16_t * 5)

Data Structure Documentation

◆ usb_serial_state_bitmap_t

struct usb_serial_state_bitmap_t		
Virtual UART signal state		
Data Fields		
uint16_t	b_rx_carrier: 1	DCD signal.
uint16_t	b_tx_carrier: 1	DSR signal.
uint16_t	b_break: 1	Break signal.
uint16_t	b_ring_signal: 1	Ring signal.
uint16_t	b_framing: 1	Framing error.
uint16_t	b_parity: 1	Parity error.
uint16_t	b_over_run: 1	Overrun error.
uint16_t	rsv: 9	Reserved.

◆ usb_sci_serialstate_t

union usb_sci_serialstate_t		
Class Notification Serial State		
Data Fields		
uint32_t	word	Word Access.
usb_serial_state_bitmap_t	bit	Bit Access.

◆ usb_pcdc_linecoding_t

struct usb_pcdc_linecoding_t

Virtual UART communication settings		
Data Fields		
uint32_t	dw_dte_rate	Bitrate.
uint8_t	b_char_format	Stop bits.
uint8_t	b_parity_type	Parity.
uint8_t	b_data_bits	Data bits.
uint8_t	rsv	Reserved.

◆ **usb_pcdc_ctrllinestate_t**

struct usb_pcdc_ctrllinestate_t		
Virtual UART control line state		
Data Fields		
uint16_t	bdtr: 1	DTR.
uint16_t	brts: 1	RTS.
uint16_t	rsv: 14	Reserved.

4.3.45 USB PHID Interface

Interfaces

Detailed Description

Interface for USB PHID functions.

Summary

The USB interface provides USB functionality.

The USB PHID interface can be implemented by:

- USB Peripheral Human Interface Device Class (r_usb_phid)

4.3.46 USB PMSC Interface

Interfaces

Detailed Description

Interface for USB PMSC functions.

Summary

The USB PMSC interface provides USB PMSC functionality.

The USB PMSC interface can be implemented by:

- USB Peripheral Mass Storage Class (r_usb_pmsc)

Macros

#define	USB_MASS_STORAGE_RESET Mass storage reset request code.
#define	USB_GET_MAX_LUN Get max logical unit number request code.

4.3.47 WDT Interface

Interfaces

Detailed Description

Interface for watch dog timer functions.

Summary

The WDT interface for the Watchdog Timer (WDT) peripheral provides watchdog functionality including resetting the device or generating an interrupt.

The watchdog timer interface can be implemented by:

- Watchdog Timer (r_wdt)
- Independent Watchdog Timer (r_iwdt)

Data Structures

struct	wdt_callback_args_t
struct	wdt_timeout_values_t
struct	wdt_cfg_t
struct	wdt_api_t

struct	wdt_instance_t

Typedefs

typedef void	wdt_ctrl_t

Enumerations

enum	wdt_timeout_t
enum	wdt_clock_division_t
enum	wdt_window_start_t
enum	wdt_window_end_t
enum	wdt_reset_control_t
enum	wdt_stop_control_t
enum	wdt_status_t

Data Structure Documentation

◆ wdt_callback_args_t

struct wdt_callback_args_t

Callback function parameter data

Data Fields		
void const *	p_context	Placeholder for user data. Set in wdt_api_t::open function in wdt_cfg_t.

◆ wdt_timeout_values_t

struct wdt_timeout_values_t

WDT timeout data. Used to return frequency of WDT clock and timeout period

Data Fields		
uint32_t	clock_frequency_hz	Frequency of watchdog clock after divider.
uint32_t	timeout_clocks	Timeout period in units of watchdog clock ticks.

◆ wdt_cfg_t

struct wdt_cfg_t

WDT configuration parameters.

Data Fields	
wdt_timeout_t	timeout
	Timeout period.
wdt_clock_division_t	clock_division
	Clock divider.
wdt_window_start_t	window_start
	Refresh permitted window start position.
wdt_window_end_t	window_end
	Refresh permitted window end position.
wdt_reset_control_t	reset_control
	Select NMI or reset generated on underflow.
wdt_stop_control_t	stop_control
	Select whether counter operates in sleep mode.
void(*	p_callback)(wdt_callback_args_t *p_args)
	Callback provided when a WDT NMI ISR occurs.
void const *	p_context
void const *	p_extend
	Placeholder for user extension.

Field Documentation

◆ p_context

void const* wdt_cfg_t::p_context
Placeholder for user data. Passed to the user callback in wdt_callback_args_t.

◆ wdt_api_t

struct wdt_api_t	
WDT functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)
fsp_err_t(*	refresh)(wdt_ctrl_t *const p_ctrl)
fsp_err_t(*	statusGet)(wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)
fsp_err_t(*	statusClear)(wdt_ctrl_t *const p_ctrl, const wdt_status_t status)
fsp_err_t(*	counterGet)(wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)
fsp_err_t(*	timeoutGet)(wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)
fsp_err_t(*	callbackSet)(wdt_ctrl_t *const p_api_ctrl, void(*p_callback)(wdt_callback_args_t *), void const *const p_context, wdt_callback_args_t *const p_callback_memory)

Field Documentation

◆ open

fsp_err_t(* wdt_api_t::open) (wdt_ctrl_t *const p_ctrl, wdt_cfg_t const *const p_cfg)

Initialize the WDT in register start mode. In auto-start mode with NMI output it registers the NMI callback.

Implemented as

- R_WDT_Open()
- R_IWDT_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ refresh

fsp_err_t(* wdt_api_t::refresh) (wdt_ctrl_t *const p_ctrl)

Refresh the watchdog timer.

Implemented as

- R_WDT_Refresh()
- R_IWDT_Refresh()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ statusGet

fsp_err_t(* wdt_api_t::statusGet) (wdt_ctrl_t *const p_ctrl, wdt_status_t *const p_status)

Read the status of the WDT.

Implemented as

- R_WDT_StatusGet()
- R_IWDT_StatusGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_status	Pointer to variable to return status information through.

◆ statusClear

fsp_err_t(* wdt_api_t::statusClear) (wdt_ctrl_t *const p_ctrl, const wdt_status_t status)

Clear the status flags of the WDT.

Implemented as

- R_WDT_StatusClear()
- R_IWDT_StatusClear()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	status	Status condition(s) to clear.

◆ counterGet

fsp_err_t(* wdt_api_t::counterGet) (wdt_ctrl_t *const p_ctrl, uint32_t *const p_count)

Read the current WDT counter value.

Implemented as

- R_WDT_CounterGet()
- R_IWDT_CounterGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_count	Pointer to variable to return current WDT counter value.

◆ timeoutGet

fsp_err_t(* wdt_api_t::timeoutGet) (wdt_ctrl_t *const p_ctrl, wdt_timeout_values_t *const p_timeout)

Read the watchdog timeout values.

Implemented as

- R_WDT_TimeoutGet()
- R_IWDT_TimeoutGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_timeout	Pointer to structure to return timeout values.

◆ callbackSet

fsp_err_t(* wdt_api_t::callbackSet) (wdt_ctrl_t *const p_api_ctrl, void(*p_callback)(wdt_callback_args_t *), void const *const p_context, wdt_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- R_WDT_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the WDT control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ wdt_instance_t

struct wdt_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
wdt_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
wdt_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
wdt_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ wdt_ctrl_t

typedef void wdt_ctrl_t

WDT control block. Allocate an instance specific control block to pass into the WDT API calls.

Implemented as

- wdt_instance_ctrl_t
- iwdt_instance_ctrl_t

Enumeration Type Documentation

◆ wdt_timeout_t

enum wdt_timeout_t

WDT time-out periods.

Enumerator	
WDT_TIMEOUT_128	128 clock cycles
WDT_TIMEOUT_512	512 clock cycles
WDT_TIMEOUT_1024	1024 clock cycles
WDT_TIMEOUT_2048	2048 clock cycles
WDT_TIMEOUT_4096	4096 clock cycles
WDT_TIMEOUT_8192	8192 clock cycles
WDT_TIMEOUT_16384	16384 clock cycles

◆ wdt_clock_division_t

enum wdt_clock_division_t	
WDT clock division ratio.	
Enumerator	
WDT_CLOCK_DIVISION_1	CLK/1.
WDT_CLOCK_DIVISION_4	CLK/4.
WDT_CLOCK_DIVISION_16	CLK/16.
WDT_CLOCK_DIVISION_32	CLK/32.
WDT_CLOCK_DIVISION_64	CLK/64.
WDT_CLOCK_DIVISION_128	CLK/128.
WDT_CLOCK_DIVISION_256	CLK/256.
WDT_CLOCK_DIVISION_512	CLK/512.
WDT_CLOCK_DIVISION_2048	CLK/2048.
WDT_CLOCK_DIVISION_8192	CLK/8192.

◆ wdt_window_start_t

enum wdt_window_start_t	
WDT refresh permitted period window start position.	
Enumerator	
WDT_WINDOW_START_25	Start position = 25%.
WDT_WINDOW_START_50	Start position = 50%.
WDT_WINDOW_START_75	Start position = 75%.
WDT_WINDOW_START_100	Start position = 100%.

◆ wdt_window_end_t

enum wdt_window_end_t	
WDT refresh permitted period window end position.	
Enumerator	
WDT_WINDOW_END_75	End position = 75%.
WDT_WINDOW_END_50	End position = 50%.
WDT_WINDOW_END_25	End position = 25%.
WDT_WINDOW_END_0	End position = 0%.

◆ wdt_reset_control_t

enum wdt_reset_control_t	
WDT Counter underflow and refresh error control.	
Enumerator	
WDT_RESET_CONTROL_NMI	NMI request when counter underflows.
WDT_RESET_CONTROL_RESET	Reset request when counter underflows.

◆ wdt_stop_control_t

enum wdt_stop_control_t	
WDT Counter operation in sleep mode.	
Enumerator	
WDT_STOP_CONTROL_DISABLE	Count will not stop when device enters sleep mode.
WDT_STOP_CONTROL_ENABLE	Count will automatically stop when device enters sleep mode.

◆ wdt_status_t

enum wdt_status_t	
WDT status	
Enumerator	
WDT_STATUS_NO_ERROR	No status flags set.
WDT_STATUS_UNDERFLOW_ERROR	Underflow flag set.
WDT_STATUS_REFRESH_ERROR	Refresh error flag set. Refresh outside of permitted window.
WDT_STATUS_UNDERFLOW_AND_REFRESH_ERROR	Underflow and refresh error flags set.

4.3.48 ADPCM Decoder Interface

Interfaces

Detailed Description

Interface for ADPCM decoder.

Summary

The ADPCM decoder interface provides functionality to decode the 4bit ADPCM data to 16bit PCM output.

Implemented by: ADPCM Decoder (rm_adpcm_decoder)

Data Structures

struct	adpcm_decoder_cfg_t
struct	adpcm_decoder_api_t
struct	adpcm_decoder_instance_t

Typedefs

typedef void	adpcm_decoder_ctrl_t

Data Structure Documentation

◆ adpcm_decoder_cfg_t

struct adpcm_decoder_cfg_t
Audio Decoder general configuration

◆ adpcm_decoder_api_t

struct adpcm_decoder_api_t
Audio Decoder interface API.

Data Fields

fsp_err_t(*	open)(adpcm_decoder_ctrl_t *const p_ctrl, adpcm_decoder_cfg_t const *const p_cfg)
fsp_err_t(*	decode)(adpcm_decoder_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t src_len_bytes)
fsp_err_t(*	reset)(adpcm_decoder_ctrl_t *const p_ctrl)
fsp_err_t(*	close)(adpcm_decoder_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* adpcm_decoder_api_t::open) (adpcm_decoder_ctrl_t *const p_ctrl, adpcm_decoder_cfg_t const *const p_cfg)

Initialize Audio Decoder device.

Implemented as

- RM_ADPCM_DECODER_Open()

Note

To reconfigure after calling this function, call adpcm_decoder_api_t::close first.

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_cfg	Pointer to configuration structure

◆ decode

fsp_err_t(* adpcm_decoder_api_t::decode) (adpcm_decoder_ctrl_t *const p_ctrl, void const *p_src, void *p_dest, uint32_t src_len_bytes)

Decodes the compressed data and stores it in output buffer.

Implemented as

- RM_ADPCM_DECODER_Decode()

Parameters

[in]	p_ctrl	Pointer to control handle structure
[in]	p_src	Pointer to a source data buffer from which data will be picked up for decode operation. The argument must not be NULL.
[out]	p_dest	Pointer to the location to store the decoded data.
[in]	p_dest	Number of bytes to be decoded.

◆ reset

fsp_err_t(* adpcm_decoder_api_t::reset) (adpcm_decoder_ctrl_t *const p_ctrl)

Resets the ADPCM driver.

Implemented as

- RM_ADPCM_DECODER_Reset()

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ close

fsp_err_t(* adpcm_decoder_api_t::close) (adpcm_decoder_ctrl_t *const p_ctrl)

Close the specified Audio decoder modules.

Implemented as

- RM_ADPCM_DECODER_Close()

Parameters

[in]	p_ctrl	Pointer to control handle structure

◆ adpcm_decoder_instance_t

struct adpcm_decoder_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
adpcm_decoder_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
adpcm_decoder_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
adpcm_decoder_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ adpcm_decoder_ctrl_t

typedef void adpcm_decoder_ctrl_t
Audio Decoder control block. Allocate an instance specific control block to pass into the Audio Decoder API calls. **Implemented as** ◦ adpcm_decoder_instance_ctrl_t

4.3.49 AUDIO PLAYBACK Interface

Interfaces

Detailed Description

Interface for the Audio Playback.

Defines the API and data structures for the Audio Playback implementation.

Summary

This module provides common interface for Audio Playback.

Implemented by: Audio Playback with PWM (rm_audio_playback_pwm)

Data Structures

struct	audio_playback_callback_args_t
struct	audio_playback_cfg_t

struct	audio_playback_api_t
struct	audio_playback_instance_t

Typedefs

typedef void	audio_playback_ctrl_t

Enumerations

enum	audio_playback_event_t

Data Structure Documentation

◆ audio_playback_callback_args_t

struct audio_playback_callback_args_t

Callback function parameter data

Data Fields		
void *	p_context	Placeholder for user data.
audio_playback_event_t	event	Event that triggered the callback.

◆ audio_playback_cfg_t

struct audio_playback_cfg_t

Audio Playback configuration parameters.

Data Fields

void const *	p_extend
	Hardware dependent configuration.
void(*	p_callback)(audio_playback_callback_args_t *p_args)
void *	p_context

Field Documentation

◆ p_callback

void(* audio_playback_cfg_t::p_callback) (audio_playback_callback_args_t *p_args)

Callback called when play is complete.

◆ **p_context**

void* audio_playback_cfg_t::p_context
Placeholder for user data. Passed to the user callback in audio_playback_callback_args_t.

◆ **audio_playback_api_t**

struct audio_playback_api_t

Audio Playback functions implemented by the Audio Playback drivers will follow this API.

Data Fields

fsp_err_t(*	open)(audio_playback_ctrl_t *const p_ctrl, audio_playback_cfg_t const *const p_cfg)
fsp_err_t(*	start)(audio_playback_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(audio_playback_ctrl_t *const p_ctrl)
fsp_err_t(*	play)(audio_playback_ctrl_t *const p_ctrl, void const *const p_buffer, uint32_t length)
fsp_err_t(*	close)(audio_playback_ctrl_t *const p_ctrl)

Field Documentation

◆ **open**

fsp_err_t(* audio_playback_api_t::open) (audio_playback_ctrl_t *const p_ctrl, audio_playback_cfg_t const *const p_cfg)

Open a audio playback module.

Implemented as

- RM_AUDIO_PLAYBACK_PWM_Open()

Parameters

[in]	p_ctrl	Pointer to memory allocated for control block.
[in]	p_cfg	Pointer to the hardware configurations.

◆ start

fsp_err_t(* audio_playback_api_t::start) (audio_playback_ctrl_t *const p_ctrl)

Start audio playback hardware.

Implemented as

- RM_AUDIO_PLAYBACK_PWM_Start()

Parameters

[in]	p_ctrl	Pointer to control block.

◆ stop

fsp_err_t(* audio_playback_api_t::stop) (audio_playback_ctrl_t *const p_ctrl)

Stop audio playback hardware.

Implemented as

- RM_AUDIO_PLAYBACK_PWM_Stop()

Parameters

[in]	p_ctrl	Pointer to control block.

◆ play

fsp_err_t(* audio_playback_api_t::play) (audio_playback_ctrl_t *const p_ctrl, void const *const p_buffer, uint32_t length)

Play audio buffer.

Implemented as

- RM_AUDIO_PLAYBACK_PWM_Play()

Parameters

[in]	p_ctrl	Pointer to control block.
[in]	p_buffer	Pointer to buffer with PCM samples to play. Data must be scaled for audio playback hardware.
[in]	length	Length of data in p_buffer.

◆ close

fsp_err_t(* audio_playback_api_t::close) (audio_playback_ctrl_t *const p_ctrl)

Close the audio driver.

Implemented as

- RM_AUDIO_PLAYBACK_PWM_Close()

Parameters

[in]	p_ctrl	Pointer to control block initialized in audio_playback_api_t::open.

◆ audio_playback_instance_t

struct audio_playback_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
audio_playback_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
audio_playback_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
audio_playback_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ audio_playback_ctrl_t

typedef void audio_playback_ctrl_t

Audio Playback control block. Allocate an instance specific control block to pass into the AUDIO_PLAYBACK API calls.

Implemented as

- audio_playback_pwm_instance_ctrl_t

Enumeration Type Documentation

◆ audio_playback_event_t

enum audio_playback_event_t	
Callback event types.	
Enumerator	
AUDIO_PLAYBACK_EVENT_PLAYBACK_COMPLETE	Audio playback complete event.

4.3.50 BLE ABS Interface

Interfaces

Detailed Description

Interface for Bluetooth Low Energy Abstraction functions.

Summary

The BLE ABS interface for the Bluetooth Low Energy Abstraction (BLE ABS) peripheral provides Bluetooth Low Energy Abstraction functionality.

The Bluetooth Low Energy Abstraction interface can be implemented by:

- Bluetooth Low Energy Abstraction (rm_ble_abs)

Data Structures

struct	ble_device_address_t
struct	ble_gap_connection_parameter_t
struct	ble_gap_connection_phy_parameter_t
struct	ble_gap_scan_phy_parameter_t
struct	ble_gap_scan_on_t
struct	ble_abs_callback_args_t
struct	ble_abs_pairing_parameter_t
struct	ble_abs_gatt_server_callback_set_t
struct	ble_abs_gatt_client_callback_set_t

struct	ble_abs_legacy_advertising_parameter_t
struct	ble_abs_extend_advertising_parameter_t
struct	ble_abs_non_connectable_advertising_parameter_t
struct	ble_abs_periodic_advertising_parameter_t
struct	ble_abs_scan_phy_parameter_t
struct	ble_abs_scan_parameter_t
struct	ble_abs_connection_phy_parameter_t
struct	ble_abs_connection_parameter_t
struct	ble_abs_cfg_t
struct	ble_abs_api_t
struct	ble_abs_instance_t

Macros

#define	BLE_ABS_ADVERTISING_PHY_LEGACY Non-Connectable Legacy Advertising phy setting.

Typedefs

typedef void(*	ble_gap_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t *p_event_data)
typedef void(*	ble_vendor_specific_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_vs_evt_data_t *p_event_data)
typedef void(*	ble_gatt_server_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_gatts_evt_data_t *p_event_data)
typedef void(*	ble_gatt_client_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_gattc_evt_data_t *p_event_data)
typedef void(*	ble_abs_delete_bond_application_callback_t) (st_ble_dev_addr_t *p_addr)
typedef void	ble_abs_ctrl_t

Enumerations

enum	ble_abs_advertising_filter_t

enum	ble_abs_local_bond_information_t
enum	ble_abs_remote_bond_information_t
enum	ble_abs_delete_non_volatile_area_t

Data Structure Documentation

◆ ble_device_address_t

struct ble_device_address_t		
st_ble_device_address is the type of bluetooth device address(BD_ADDR).		
Data Fields		
uint8_t	addr[BLE_BD_ADDR_LEN]	bluetooth device address.
uint8_t	type	the type of bluetooth device address.

◆ ble_gap_connection_parameter_t

struct ble_gap_connection_parameter_t		
ble_gap_connection_parameter_t is Connection parameters included in connection interval, slave latency, supervision timeout, ce length.		
Data Fields		
uint16_t	conn_intv_min	Minimum connection interval.
uint16_t	conn_intv_max	Maximum connection interval.
uint16_t	conn_latency	Slave latency.
uint16_t	sup_to	Supervision timeout.
uint16_t	min_ce_length	Minimum CE Length.
uint16_t	max_ce_length	Maximum CE Length.

◆ ble_gap_connection_phy_parameter_t

struct ble_gap_connection_phy_parameter_t		
ble_gap_connection_phy_parameter_t is Connection parameters per PHY.		
Data Fields		
uint16_t	scan_intv	Scan interval.
uint16_t	scan_window	Scan window.
ble_gap_connection_parameter_t *	p_conn_param	Connection interval, slave latency, supervision timeout, and CE length.

◆ ble_gap_scan_phy_parameter_t

struct ble_gap_scan_phy_parameter_t

Scan parameters per scan PHY.		
Data Fields		
uint8_t	scan_type	Scan type.
uint16_t	scan_intv	Scan interval.
uint16_t	scan_window	Scan window.

◆ ble_gap_scan_on_t

struct ble_gap_scan_on_t		
Parameters configured when scanning starts.		
Data Fields		
uint8_t	proc_type	Procedure type.
uint8_t	filter_dups	Filter duplicates.
uint16_t	duration	Scan duration.
uint16_t	period	Scan period.

◆ ble_abs_callback_args_t

struct ble_abs_callback_args_t		
Callback function parameter data		
Data Fields		
uint32_t	channel	Select a channel corresponding to the channel number of the hardware.
ble_event_cb_t	ble_abs_event	The event can be used to identify what caused the callback.
void const *	p_context	Placeholder for user data. Set in ble_abs_api_t::open function in ble_abs_cfg_t.

◆ ble_abs_pairing_parameter_t

struct ble_abs_pairing_parameter_t		
st_ble_abs_pairing_parameter_t includes the pairing parameters.		
Data Fields		
uint8_t	io_capabilitie_local_device	IO capabilities of local device.
uint8_t	mitm_protection_policy	MITM protection policy.
uint8_t	secure_connection_only	Determine whether to accept only Secure Connections or not.
uint8_t	local_key_distribute	Type of keys to be distributed from local device.

uint8_t	remote_key_distribute	Type of keys which local device requests a remote device to distribute.
uint8_t	maximum_key_size	Maximum LTK size.
uint8_t	padding[2]	padding

◆ ble_abs_gatt_server_callback_set_t

struct ble_abs_gatt_server_callback_set_t		
GATT Server callback function and the priority.		
Data Fields		
ble_gatt_server_application_callback_t	gatt_server_callback_function	GATT Server callback function.
uint8_t	gatt_server_callback_priority	The priority number of GATT Server callback function.

◆ ble_abs_gatt_client_callback_set_t

struct ble_abs_gatt_client_callback_set_t		
GATT Client callback function and the priority.		
Data Fields		
ble_gatt_client_application_callback_t	gatt_client_callback_function	GATT Client callback function.
uint8_t	gatt_client_callback_priority	The priority number of GATT Client callback function.

◆ ble_abs_legacy_advertising_parameter_t

struct ble_abs_legacy_advertising_parameter_t		
st_ble_abs_legacy_advertising_parameter_t is the parameters for legacy advertising.		
Data Fields		
ble_device_address_t *	p_peer_address	The remote device address. If the p_peer_address parameter is not NULL, Direct Connectable Advertising is performed to the remote address. If the p_peer_address parameter is NULL, Undirect Connectable Advertising is performed according to the advertising filter policy specified by the filter parameter.
uint8_t *	p_advertising_data	Advertising Data. If the p_advertising_data is specified as NULL, Advertising

		Data is not included in the advertising PDU.
uint8_t *	p_scan_response_data	Scan Response Data. If the p_scan_response_data is specified as NULL, Scan Response Data is not included in the advertising PDU.
uint32_t	fast_advertising_interval	Advertising with the fast_advertising_interval parameter continues for the period specified by the fast_period parameter. Time(ms) = fast_advertising_interval * 0.625. If the fast_period parameter is 0, this parameter is ignored. Valid range is 0x00000020 - 0x00FFFFFF.
uint32_t	slow_advertising_interval	After the elapse of the fast_period, advertising with the slow_advertising_interval parameter continues for the period specified by the slow_advertising_interval parameter. Time(ms) = slow_advertising_interval * 0.625. If the slow_advertising_interval parameter is 0, this parameter is ignored. Valid range is 0x00000020 - 0x00FFFFFF.
uint16_t	fast_advertising_period	The period which advertising with the fast_advertising_interval parameter continues for. Time = duration * 10ms. After the elapse of the fast_advertising_period, BLE_GAP_EVENT_ADV_OFF event notifies that the advertising has stopped. Valid range is 0x0000 - 0xFFFF. If the fast_advertising_period parameter is 0x0000, advertising with the fast_advertising_interval parameter is not performed.
uint16_t	slow_advertising_period	The period which advertising with the

		slow_advertising_interval parameter continues for. Time = duration * 10ms. After the elapse of the slow_advertising_period, BLE_GAP_EVENT_ADV_OFF event notifies that the advertising has stopped. Valid range is 0x0000 - 0xFFFF. If the slow_advertising_period parameter is 0x0000, the advertising continues.
uint16_t	advertising_data_length	Advertising data length(byte). Valid range is 0-31. If the advertising_data_length is 0, Advertising Data is not included in the advertising PDU.
uint16_t	scan_response_data_length	Scan response data length (in bytes). Scan Response Data(byte). Valid range is 0-31. If the scan_response_data_length is 0, Scan Response Data is not included in the advertising PDU.
uint8_t	advertising_channel_map	The channel map used for the advertising packet transmission. It is a bitwise OR of the following values. macro: description BLE_GAP_ADV_CH_37(0x01): Use 37 CH. BLE_GAP_ADV_CH_38(0x02): Use 38 CH. BLE_GAP_ADV_CH_39(0x04): Use 38 CH. BLE_GAP_ADV_CH_ALL(0x07): Use 37 - 39 CH.
uint8_t	advertising_filter_policy	Advertising filter policy. If the p_peer_address parameter is NULL, the advertising is performed according to the advertising filter policy. If the p_peer_address parameter is not NULL, this parameter is ignored.

		<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY(0x00)</td><td>Process scan and connection requests from all devices.</td></tr><tr><td>BLE_ABS_ADVERTISING_FILTER_ALLOW_WHITE_LIST(0x01)</td><td>Process scan and connection requests from only devices in the White List.</td></tr></table>
uint8_t	own_bluetooth_address_type	Own Bluetooth address type. Select one of the following.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Public Address</td></tr><tr><td>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)</td><td>Resolvable Private Address. If the IRK of local device has not been registered in Resolving List, public address is used.</td></tr></table>
uint8_t	own_bluetooth_address[6]	Own Bluetooth address.
uint8_t	padding[3]	padding

◆ ble_abs_extend_advertising_parameter_t

struct ble_abs_extend_advertising_parameter_t		
st_ble_abs_extend_advertising_parameter_t is the parameters for extended advertising.		
Data Fields		
ble_device_address_t *	p_peer_address	The remote device address. If the p_addr parameter is not NULL, Direct Connectable Advertising is performed to the remote address. If the p_addr parameter is NULL, Undirect Connectable Advertising is performed according to the advertising filter policy specified by the filter parameter.

uint8_t *	p_advertising_data	Advertising data. If p_adv_data is specified as NULL, advertising data is not set.
uint32_t	fast_advertising_interval	Advertising with the fast_advertising_interval parameter continues for the period specified by the fast_advertising_period parameter. Time(ms) = fast_advertising_interval * 0.625. If the fast_advertising_period parameter is 0, this parameter is ignored. Valid range is 0x00000020 - 0x00FFFFFF.
uint32_t	slow_advertising_interval	After the elapse of the fast_advertising_period, advertising with the slow_advertising_interval parameter continues for the period specified by the slow_advertising_period parameter. Time(ms) = fast_advertising_interval * 0.625. If the fast_advertising_period parameter is 0, this parameter is ignored. Valid range is 0x00000020 - 0x00FFFFFF.
uint16_t	fast_advertising_period	The period which advertising with the fast_advertising_interval parameter continues for. Time = duration * 10ms. After the elapse of the fast_advertising_period, BLE_GAP_EVENT_ADV_OFF event notifies that the advertising has stopped. Valid range is 0x0000 - 0xFFFF. If the fast_advertising_period parameter is 0x0000, the fast_advertising_interval parameter is ignored.
uint16_t	slow_advertising_period	The period which advertising with the slow_advertising_interval

		parameter continues for. Time = duration * 10ms. After the elapse of the slow_advertising_period, BLE_GAP_EVENT_ADV_OFF event notifies that the advertising has stopped. Valid range is 0x0000 - 0xFFFF. If the slow_advertising_period parameter is 0x0000, the advertising continues.
uint16_t	advertising_data_length	Advertising data length (in bytes). Valid range is 0-229. If the adv_data_length is 0, Advertising Data is not included in the advertising PDU.
uint8_t	advertising_channel_map	The channel map used for the advertising packet transmission. It is a bitwise OR of the following values. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADV_CH_37(0x01)</td><td>Use 37 CH.</td></tr><tr><td>BLE_GAP_ADV_CH_38(0x02)</td><td>Use 38 CH.</td></tr><tr><td>BLE_GAP_ADV_CH_39(0x04)</td><td>Use 38 CH.</td></tr><tr><td>BLE_GAP_ADV_CH_ALL(0x07)</td><td>Use 37 - 39 CH.</td></tr></table>
uint8_t	advertising_filter_policy	Advertising filter policy. If the p_peer_address parameter is NULL, the advertising is performed according to the advertising filter policy. If the p_peer_address parameter is not NULL, this parameter is ignored. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY(0x00)</td><td>Process scan and connection requests from all devices.</td></tr><tr><td>BLE_ABS_ADV</td><td>Process scan</td></tr></table>

<table>
<tr><td></td><td></td><td><table><tr><td>ERTISING_FILTER_ALLOW_WHITE_LIST(0x01)</td><td>and connection requests from only devices in the White List.</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>own_bluetooth_address_type</td><td>Own Bluetooth address type.
Select one of the following.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Public Address</td></tr><tr><td>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)</td><td>Resolvable Private Address.
If the IRK of local device has not been registered in Resolving List, public address is used.</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>own_bluetooth_address[6]</td><td>Own Bluetooth address.</td></tr>
<tr><td>uint8_t</td><td>primary_advertising_phy</td><td>Primary advertising PHY.
In this parameter, only 1M PHY and Coded PHY can be specified, and 2M PHY cannot be specified.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADV_PHY_1M(0x01)</td><td>Use 1M PHY as Primary Advertising PHY.
When the adv_prop_type field is Legacy Advertising PDU type, this field shall be set to BLE_GAP_ADV_PHY_1M.</td></tr><tr><td>BLE_GAP_ADV_PHY_CD(0x03)</td><td>Use Coded PHY as Primary Advertising PHY.
Coding</td></tr></table></td></tr>
</table>

		scheme is configured by R_BLE_VS_SetCodingScheme().
uint8_t	secondary_advertising_phy	Secondary advertising Phy. Select one of the following. macro: BLE_GAP_ADV_PHY_1M(0x01) — description: Use 1M PHY as Secondary Advertising PHY. macro: BLE_GAP_ADV_PHY_2M(0x02) — description: Use 2M PHY as Secondary Advertising PHY. macro: BLE_GAP_ADV_PHY_CD(0x03) — description: Use Coded PHY(S=8) as Secondary Advertising PHY. Coding scheme is configured by R_BLE_VS_SetCodingScheme().
uint8_t	padding[3]	padding

◆ ble_abs_non_connectable_advertising_parameter_t

struct ble_abs_non_connectable_advertising_parameter_t		
st_ble_abs_non_connectable_advertising_parameter_t is the parameters for non-connectable advertising.		
Data Fields		
ble_device_address_t *	p_peer_address	The remote device address. If the p_peer_address parameter is not NULL, Direct Connectable Advertising is performed to the remote address. If the p_peer_address parameter is NULL, Undirect Connectable Advertising is performed according to the advertising filter policy specified by the filter parameter.
uint8_t *	p_advertising_data	Advertising data. If p_adv_data

		is specified as NULL, advertising data is not set.
uint32_t	advertising_interval	Advertising with the advertising_interval parameter continues for the period specified by the duration parameter. Time(ms) = advertising_interval * 0.625. If the duration parameter is 0x0000, the advertising with the advertising_interval parameter continue. Valid range is 0x00000020 - 0x00FFFFFF.
uint16_t	advertising_duration	The period which advertising with the advertising_interval parameter continues for. Time = advertising_duration * 10ms. After the elapse of the advertising_duration, BLE_GAP_EVENT_ADV_OFF event notifies that the advertising has stopped. Valid range is 0x0000 - 0xFFFF. If the advertising_duration parameter is 0x0000, the advertising continues.
uint16_t	advertising_data_length	Advertising data length (in bytes). If the primary_advertising_phy parameter is BLE_ABS_ADVERTISING_PHY_LEGACY(0x00), the valid range is 0-31. If the primary_advertising_phy parameter is the other values, the valid range is 0-1650. If the advertising_data_length parameter is 0, Advertising Data is not included in the advertising PDU.
uint8_t	advertising_channel_map	The channel map used for the advertising packet transmission. It is a bitwise OR of the following values. macro: description BLE_GAP_ADV_CH_37(0x01): Use 37 CH.

<table>
<tr><td></td><td></td><td><table><tr><td>BLE_GAP_ADV_CH_38(0x02)</td><td>Use 38 CH.</td></tr><tr><td>BLE_GAP_ADV_CH_39(0x04)</td><td>Use 38 CH.</td></tr><tr><td>BLE_GAP_ADV_CH_ALL(0x07)</td><td>Use 37 - 39 CH.</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>own_bluetooth_address_type</td><td>Own Bluetooth address type.
Select one of the following.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Public Address</td></tr><tr><td>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)</td><td>Resolvable Private Address.
If the IRK of local device has not been registered in Resolving List, public address is used.</td></tr></table></td></tr>
<tr><td>uint8_t</td><td>own_bluetooth_address[6]</td><td>Own Bluetooth address.</td></tr>
<tr><td>uint8_t</td><td>primary_advertising_phy</td><td>Primary advertising PHY.
In this parameter, only 1M PHY and Coded PHY can be specified, and 2M PHY cannot be specified.<table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_ABS_ADVERTISING_PHY_LEGACY(0x00)</td><td>Use 1M PHY as Primary Advertising PHY for Non-Connectable Legacy Advertising.
If Periodic Advertising is performed, this value shall not set to the adv_phy parameter.</td></tr><tr><td>BLE_GAP_ADV_PHY_1M(0x01)</td><td>Use 1M PHY as Primary Advertising</td></tr></table></td></tr>
</table>

		PHY. When the adv_prop_type field is Legacy Advertising PDU type, this field shall be set to BLE_GAP_ADV_PHY_1M. BLE_GAP_ADV_PHY_CD(0x03) — Use Coded PHY as Primary Advertising PHY. Coding scheme is configured by R_BLE_VS_SetCodingScheme().
uint8_t	secondary_advertising_phy	Secondary advertising Phy. Select one of the following. **macro** — **description** BLE_GAP_ADV_PHY_1M(0x01) — Use 1M PHY as Secondary Advertising PHY. BLE_GAP_ADV_PHY_2M(0x02) — Use 2M PHY as Secondary Advertising PHY. BLE_GAP_ADV_PHY_CD(0x03) — Use Coded PHY(S=8) as Secondary Advertising PHY. Coding scheme is configured by R_BLE_VS_SetCodingScheme().
uint8_t	padding[2]	padding

◆ ble_abs_periodic_advertising_parameter_t

struct ble_abs_periodic_advertising_parameter_t
st_ble_abs_periodic_advertising_parameter_t is the parameters for periodic advertising.

Data Fields		
ble_abs_non_connectable_advertising_parameter_t	advertising_parameter	Advertising parameters.
uint8_t *	p_periodic_advertising_data	Periodic advertising data. If p_perd_adv_data is specified as NULL, periodic advertising data is not set.
uint16_t	periodic_advertising_interval	Periodic advertising interval. Time(ms) = periodic_advertising_interval * 1.25. Valid range is 0x0006 - 0xFFFF.
uint16_t	periodic_advertising_data_length	Periodic advertising data length (in bytes). Valid range is 0 - 1650. If the periodic_advertising_data_length is 0, Periodic Advertising Data is not included in the advertising PDU.

◆ **ble_abs_scan_phy_parameter_t**

struct ble_abs_scan_phy_parameter_t

st_ble_abs_scan_phy_parameter_t is the phy parameters for scan.

Data Fields		
uint16_t	fast_scan_interval	Fast scan interval. Interval(ms) = fast_scan_interval * 0.625. Valid range is 0x0004 - 0xFFFF.
uint16_t	slow_scan_interval	Slow Scan interval. Slow Scan interval(ms) = slow_scan_interval * 0.625. Valid range is 0x0004 - 0xFFFF.
uint16_t	fast_scan_window	Fast Scan window. Fast Scan window(ms) = fast_scan_window * 0.625. Valid range is 0x0004 - 0xFFFF.
uint16_t	slow_scan_window	Slow Scan window. Slow Scan window(ms) = slow_scan_window * 0.625. Valid range is 0x0004 - 0xFFFF.
uint8_t	scan_type	Scan type. <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_SCAN_PASSIVE(0x00)</td><td>Passive Scan.</td></tr></table>

		BLE_GAP_SCAN_ACTIVE(0x01)	Active Scan.
uint8_t	padding[3]	padding.	

◆ **ble_abs_scan_parameter_t**

struct ble_abs_scan_parameter_t		
st_ble_abs_scan_parameter_t is the parameters for scan.		
Data Fields		
ble_abs_scan_phy_parameter_t *	p_phy_parameter_1M	Scan parameters for receiving the advertising packets in 1M PHY. In case of not receiving the advertising packets in 1M PHY, this field is specified as NULL. p_phy_parameter_1M or p_phy_parameter_coded field shall be set to scan parameters.
ble_abs_scan_phy_parameter_t *	p_phy_parameter_coded	Scan parameters for receiving the advertising packets in Coded PHY. In case of not receiving the advertising packets in Coded PHY, this field is specified as NULL. p_phy_parameter_1M or p_phy_parameter_coded field shall be set to scan parameters.
uint8_t *	p_filter_data	Data for Advertising Data filtering. The p_filter_data parameter is used for the advertising data in single advertising report. The advertising data composed of multiple advertising reports is not filtered by this parameter. If the p_filter_data parameter is specified as NULL, the filtering is not done.
uint16_t	fast_scan_period	The period which scan with the fast scan interval/fast scan window continues for. Time(ms) = fast_scan_period * 10. Valid range is 0x0000 - 0xFFFF. If the fast_scan_period parameter is 0x0000, scan with the fast scan interval/fast scan

		window is not performed. After the elapse of the fast_scan_period, BLE_GAP_EVENT_SCAN_TO event notifies that the scan has stopped.
uint16_t	slow_scan_period	The period which scan with the slow scan interval/slow scan window continues for. Time = slow_scan_period * 10ms. Valid range is 0x0000 - 0xFFFF. If the slow_scan_period parameter is 0x0000, the scan continues. After the elapse of the slow_scan_period, BLE_GAP_EVENT_SCAN_TO event notifies that the scan has stopped.
uint16_t	filter_data_length	The length of the data specified by the p_filter_data parameter. Valid range is 0x0000-0x0010. If the filter_data_length parameter is 0, the filtering is not done.
uint8_t	device_scan_filter_policy	Scan Filter Policy. Select one of the following. . • Address type setting (Field [7:4]) <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Use Public Address.</td></tr><tr><td>BLE_GAP_ADDR_RAND(0x01)</td><td>Use Random Address.</td></tr><tr><td>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)</td><td>If the IRK of local device has been registered in Resolving list, use RPA. If not, use</td></tr></table>

			Public Address.
			BLE_GAP_ADDR_RPA_ID_RANDOM(0x03) — If the IRK of local device has been registered in Resolving list, use RPA. If not, use Random Address.
			• White list setting (Field [3:0])

macro	description
BLE_GAP_SCAN_ALLOW_ADV_ALL(0x00)	Accept all advertising and scan response PDUs except directed advertising PDUs not addressed to local device.
BLE_GAP_SCAN_ALLOW_ADV_WLST(0x01)	Accept only advertising and scan response PDUs from remote devices whose address is registered in the White List. Directed advertising PDUs which are not

				addressed to local device is ignored.
			BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED(0x02)	Accept all advertising and scan response PDUs except directed advertising PDUs whose the target address is identity address but doesn't address local device. However directed advertising PDUs whose the target address is the local resolvable private address are accepted.
			BLE_GAP_SCAN_ALLOW_ADV_EXCEPT_DIRECTED_WLST(0x03)	Accept all advertising and scan response PDUs. The following are excluded. • Advertising an

			d scan response PDUs where the advertiser's identity address is not in the White List. • Directed advertising PDUs whose the target add

			ress is identity address but doesn't address local device. However directed advertising PDUs whose the target address is the local resolvabl

<table>
<tr><td></td><td></td><td>e private address are accepted.</td></tr>
<tr><td>uint8_t</td><td>filter_duplicate</td><td>Filter duplicates.
Maximum number of filtered devices is 8.
The 9th and subsequent devices are not filtered by this parameter.
<table>
<tr><th>macro</th><th>description</th></tr>
<tr><td>BLE_GAP_SCAN_FILT_DUPLIC_DISABLE(0x00)</td><td>Duplicate filter disabled.</td></tr>
<tr><td>BLE_GAP_SCAN_FILT_DUPLIC_ENABLE(0x01)</td><td>Duplicate filter enabled.</td></tr>
<tr><td>BLE_GAP_SCAN_FILT_DUPLIC_ENABLE_FOR_PERIOD(0x02))</td><td>Duplicate filtering enabled, reset for each scan period.</td></tr>
</table></td></tr>
<tr><td>uint8_t</td><td>filter_ad_type</td><td>The AD type of the data specified by the p_filter_data parameter.
The AD type identifier values are defined in Bluetooth SIG Assigned Number
(https://www.bluetooth.com/specifications/assigned-numbers).</td></tr>
<tr><td>uint8_t</td><td>padding[3]</td><td>Padding.</td></tr>
</table>

◆ ble_abs_connection_phy_parameter_t

struct ble_abs_connection_phy_parameter_t
st_ble_abs_connection_phy_parameter_t is the phy parameters for create connection.

Data Fields		
uint16_t	connection_interval	Connection interval. Time(ms) = connection_interval * 1.25. Valid range is 0x0006 - 0x0C80.
uint16_t	connection_slave_latency	Slave latency. Valid range is 0x0000 - 0x01F3.
uint16_t	supervision_timeout	Supervision timeout. Time(ms) = supervision_timeout * 10. Valid range is 0x000A - 0x0C80.
uint8_t	padding[2]	Padding.

◆ ble_abs_connection_parameter_t

struct ble_abs_connection_parameter_t
st_ble_abs_connection_parameter_t is the parameters for create connection.

Data Fields		
ble_abs_connection_phy_parameter_t *	p_connection_phy_parameter_1M	Connection interval, slave latency, supervision timeout for 1M PHY. The p_connection_phy_parameter_1M is specified as NULL, a connection request is not sent with 1M PHY.
ble_abs_connection_phy_parameter_t *	p_connection_phy_parameter_2M	Connection interval, slave latency, supervision timeout for 2M PHY. The p_connection_phy_parameter_2M is specified as NULL, a connection request is not sent with 2M PHY.
ble_abs_connection_phy_parameter_t *	p_connection_phy_parameter_coded	Connection interval, slave latency, supervision timeout for Coded PHY. The p_connection_phy_parameter_coded is specified as NULL, a connection request is not sent with Coded PHY.
ble_device_address_t *	p_device_address	Address of the device to be connected. If the filter field is BLE_GAP_INIT_FILT_USE_WLST(0x01), this parameter is ignored and please fill p_device_address.addr with 0x00.
uint8_t	filter_parameter	The filter field specifies whether

		the White List is used or not, when connecting with a remote device. . • Address type setting (Field [7:4]) <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_ADDR_PUBLIC(0x00)</td><td>Use Public Address.</td></tr><tr><td>BLE_GAP_ADDR_RAND(0x01)</td><td>Use Random Address.</td></tr><tr><td>BLE_GAP_ADDR_RPA_ID_PUBLIC(0x02)</td><td>If the IRK of local device has been registered in Resolving list, use RPA. If not, use Public Address.</td></tr><tr><td>BLE_GAP_ADDR_RPA_ID_RANDOM(0x03)</td><td>If the IRK of local device has been registered in Resolving list, use RPA. If not, use Random Address.</td></tr></table>• White list setting (Field [3:0]) <table><tr><th>macro</th><th>description</th></tr><tr><td>BLE_GAP_INIT_FILT_USE_ADDR(0x00)</td><td>White List is not used. The remote device to</td></tr></table>

			be connected is specified by the p_addr field is used.
		BLE_GAP_INIT_FILT_USE_WLST (0x01)	White List is used. The remote device registered in White List is connected with local device. The p_addr field is ignored.
uint8_t	connection_timeout	The time(sec) to cancel the create connection request. Valid range is 0 <= connection_timeout <= 10. If the connection_timeout field is 0, the create connection request is not canceled. .	
uint8_t	padding[2]	Padding.	

◆ **ble_abs_cfg_t**

struct ble_abs_cfg_t	
BLE ABS configuration parameters.	
Data Fields	
uint32_t	channel
	Select a channel corresponding to the channel number of the hardware. More...
ble_gap_application_callback_t	gap_callback
	GAP callback function.

ble_vendor_specific_application_callback_t	vendor_specific_callback
	Vendor Specific callback function.
ble_abs_gatt_server_callback_set_t *	p_gatt_server_callback_list
	GATT Server callback set.
uint8_t	gatt_server_callback_list_number
	The number of GATT Server callback functions.
ble_abs_gatt_client_callback_set_t *	p_gatt_client_callback_list
	GATT Client callback set.
uint8_t	gatt_client_callback_list_number
	The number of GATT Client callback functions.
ble_abs_pairing_parameter_t *	p_pairing_parameter
	Pairing parameters.
flash_instance_t const *	p_flash_instance
	Pointer to flash instance.
timer_instance_t const *	p_timer_instance
	Pointer to timer instance.
void(*	p_callback)(ble_abs_callback_args_t *p_args)

	Callback provided when a BLE ISR occurs.
void const *	p_context
	Placeholder for user data. Passed to the user callback in ble_abs_callback_args_t.
void const *	p_extend
	Placeholder for user extension.

Field Documentation

◆ channel

uint32_t ble_abs_cfg_t::channel

Select a channel corresponding to the channel number of the hardware.

the parameters for initialization.

◆ ble_abs_api_t

struct ble_abs_api_t

BLE ABS functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(ble_abs_ctrl_t *const p_ctrl, ble_abs_cfg_t const *const p_cfg)
fsp_err_t(*	close)(ble_abs_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(ble_abs_ctrl_t *const p_ctrl, ble_event_cb_t init_callback)
fsp_err_t(*	startLegacyAdvertising)(ble_abs_ctrl_t *const p_ctrl, ble_abs_legacy_advertising_parameter_t const *const p_advertising_parameter)
fsp_err_t(*	startExtendedAdvertising)(ble_abs_ctrl_t *const p_ctrl, ble_abs_extend_advertising_parameter_t const *const p_advertising_parameter)
fsp_err_t(*	startNonConnectableAdvertising)(ble_abs_ctrl_t *const p_ctrl,

	ble_abs_non_connectable_advertising_parameter_t const *const p_advertising_parameter)
fsp_err_t(*	startPeriodicAdvertising)(ble_abs_ctrl_t *const p_ctrl, ble_abs_periodic_advertising_parameter_t const *const p_advertising_parameter)
fsp_err_t(*	startScanning)(ble_abs_ctrl_t *const p_ctrl, ble_abs_scan_parameter_t const *const p_scan_parameter)
fsp_err_t(*	createConnection)(ble_abs_ctrl_t *const p_ctrl, ble_abs_connection_parameter_t const *const p_connection_parameter)
fsp_err_t(*	setLocalPrivacy)(ble_abs_ctrl_t *const p_ctrl, uint8_t const *const p_lc_irk, uint8_t privacy_mode)
fsp_err_t(*	startAuthentication)(ble_abs_ctrl_t *const p_ctrl, uint16_t connection_handle)
fsp_err_t(*	deleteBondInformation)(ble_abs_ctrl_t *const p_ctrl, ble_abs_bond_information_parameter_t const *const p_bond_information_parameter)
fsp_err_t(*	importKeyInformation)(ble_abs_ctrl_t *const p_ctrl, ble_device_address_t *p_local_identity_address, uint8_t *p_local_irk, uint8_t *p_local_csrk)
fsp_err_t(*	exportKeyInformation)(ble_abs_ctrl_t *const p_ctrl, ble_device_address_t *p_local_identity_address, uint8_t *p_local_irk, uint8_t *p_local_csrk)

Field Documentation

◆ open

fsp_err_t(* ble_abs_api_t::open) (ble_abs_ctrl_t *const p_ctrl, ble_abs_cfg_t const *const p_cfg)

Initialize the BLE ABS in register start mode.

Implemented as

- RM_BLE_ABS_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ close

fsp_err_t(* ble_abs_api_t::close) (ble_abs_ctrl_t *const p_ctrl)

Close the BLE ABS.

Implemented as

- RM_BLE_ABS_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* ble_abs_api_t::reset) (ble_abs_ctrl_t *const p_ctrl, ble_event_cb_t init_callback)

Close the BLE ABS.

Implemented as

- RM_BLE_ABS_Reset()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	init_callback	callback function to initialize Host Stack.

◆ startLegacyAdvertising

fsp_err_t(* ble_abs_api_t::startLegacyAdvertising) (ble_abs_ctrl_t *const p_ctrl, ble_abs_legacy_advertising_parameter_t const *const p_advertising_parameter)

Start Legacy Connectable Advertising.

Implemented as

- RM_BLE_ABS_StartLegacyAdvertising()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_advertising_parameter	Pointer to Advertising parameters for Legacy Advertising.

◆ startExtendedAdvertising

fsp_err_t(* ble_abs_api_t::startExtendedAdvertising) (ble_abs_ctrl_t *const p_ctrl, ble_abs_extend_advertising_parameter_t const *const p_advertising_parameter)

Start Extended Connectable Advertising.

Implemented as

- RM_BLE_ABS_StartExtendedAdvertising()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_advertising_parameter	Pointer to Advertising parameters for extend Advertising.

◆ startNonConnectableAdvertising

fsp_err_t(* ble_abs_api_t::startNonConnectableAdvertising) (ble_abs_ctrl_t *const p_ctrl, ble_abs_non_connectable_advertising_parameter_t const *const p_advertising_parameter)

Start Non-Connectable Advertising.

Implemented as

- RM_BLE_ABS_StartNonConnectableAdvertising()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_advertising_parameter	Pointer to Advertising parameters for non-connectable Advertising.

◆ startPeriodicAdvertising

fsp_err_t(* ble_abs_api_t::startPeriodicAdvertising) (ble_abs_ctrl_t *const p_ctrl, ble_abs_periodic_advertising_parameter_t const *const p_advertising_parameter)

Start Periodic Advertising.

Implemented as

- RM_BLE_ABS_StartPeriodicAdvertising()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_advertising_parameter	Pointer to Advertising parameters for periodic Advertising.

◆ startScanning

fsp_err_t(* ble_abs_api_t::startScanning) (ble_abs_ctrl_t *const p_ctrl, ble_abs_scan_parameter_t const *const p_scan_parameter)

Start scanning.

Implemented as

- RM_BLE_ABS_StartScanning()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_scan_parameter	Pointer to scan parameter.

◆ createConnection

fsp_err_t(* ble_abs_api_t::createConnection) (ble_abs_ctrl_t *const p_ctrl, ble_abs_connection_parameter_t const *const p_connection_parameter)

Request create connection.

Implemented as

- RM_BLE_ABS_CreateConnection()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_connection_parameter	Pointer to connection parameter.

◆ setLocalPrivacy

fsp_err_t(* ble_abs_api_t::setLocalPrivacy) (ble_abs_ctrl_t *const p_ctrl, uint8_t const *const p_lc_irk, uint8_t privacy_mode)

Configure local device privacy.

Implemented as

- RM_BLE_ABS_SetLocalPrivacy()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_lc_irk	Pointer to IRK to be registered in the resolving list.
[in]	privacy_mode	privacy_mode privacy mode.

◆ startAuthentication

fsp_err_t(* ble_abs_api_t::startAuthentication) (ble_abs_ctrl_t *const p_ctrl, uint16_t connection_handle)

Start pairing or encryption.

Implemented as

- RM_BLE_ABS_StartAuthentication()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	connection_handle	Connection handle identifying the remote device.

◆ deleteBondInformation

fsp_err_t(* ble_abs_api_t::deleteBondInformation) (ble_abs_ctrl_t *const p_ctrl, ble_abs_bond_information_parameter_t const *const p_bond_information_parameter)

Delete bond information.

Implemented as

- RM_BLE_ABS_DeleteBondInformation()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_bond_information_parameter	Pointer to bond information parameter.

◆ importKeyInformation

fsp_err_t(* ble_abs_api_t::importKeyInformation) (ble_abs_ctrl_t *const p_ctrl, ble_device_address_t *p_local_identity_address, uint8_t *p_local_irk, uint8_t *p_local_csrk)

Import local identity address, keys information to local storage.

Implemented as

- RM_BLE_ABS_ImportKeyInformation()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_local_identity_address	Pointer to local identiry address.
[in]	uint8_t	p_local_irk Pointer to local IRK
[in]	uint8_t	p_local_csrk Pointer to local CSRK

◆ exportKeyInformation

fsp_err_t(* ble_abs_api_t::exportKeyInformation) (ble_abs_ctrl_t *const p_ctrl, ble_device_address_t *p_local_identity_address, uint8_t *p_local_irk, uint8_t *p_local_csrk)

Export local identity address, keys information from local storage.

Implemented as

- RM_BLE_ABS_ExportKeyInformation()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_local_identity_address	Pointer to local identiry address.
[out]	uint8_t	p_local_irk Pointer to local IRK
[out]	uint8_t	p_local_csrk Pointer to local CSRK

◆ ble_abs_instance_t

struct ble_abs_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
ble_abs_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
ble_abs_cfg_t const *	p_cfg	Pointer to the configuration

		structure for this instance.
ble_abs_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ ble_gap_application_callback_t

typedef void(* ble_gap_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_evt_data_t *p_event_data)
ble_gap_application_callback_t is the GAP Event callback function type.

◆ ble_vendor_specific_application_callback_t

typedef void(* ble_vendor_specific_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_vs_evt_data_t *p_event_data)
ble_vendor_specific_application_callback_t is the Vendor Specific Event callback function type.

◆ ble_gatt_server_application_callback_t

typedef void(* ble_gatt_server_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_gatts_evt_data_t *p_event_data)
ble_gatt_server_application_callback_t is the GATT Server Event callback function type.

◆ ble_gatt_client_application_callback_t

typedef void(* ble_gatt_client_application_callback_t) (uint16_t event_type, ble_status_t event_result, st_ble_gattc_evt_data_t *p_event_data)
ble_gatt_client_application_callback_t is the GATT Server Event callback function type.

◆ ble_abs_delete_bond_application_callback_t

typedef void(* ble_abs_delete_bond_application_callback_t) (st_ble_dev_addr_t *p_addr)
ble_abs_delete_bond_application_callback_t is the delete bond information Event callback function type.

◆ ble_abs_ctrl_t

typedef void ble_abs_ctrl_t
BLE ABS control block. Allocate an instance specific control block to pass into the BLE ABS API calls. **Implemented as** ◦ ble_abs_instance_ctrl_t

Enumeration Type Documentation

◆ ble_abs_advertising_filter_t

enum ble_abs_advertising_filter_t	
Advertising Filter Policy	
Enumerator	
BLE_ABS_ADVERTISING_FILTER_ALLOW_ANY	Receive a connect request from all devices.
BLE_ABS_ADVERTISING_FILTER_ALLOW_WHITE_LIST	Receive a connect request from only the devices registered in White List.

◆ ble_abs_local_bond_information_t

enum ble_abs_local_bond_information_t	
Local keys delete policy	
Enumerator	
BLE_ABS_LOCAL_BOND_INFORMATION_NONE	Delete no local keys.
BLE_ABS_LOCAL_BOND_INFORMATION_ALL	Delete all local keys.

◆ ble_abs_remote_bond_information_t

enum ble_abs_remote_bond_information_t	
Remote keys delete policy	
Enumerator	
BLE_ABS_REMOTE_BOND_INFORMATION_NONE	Delete no remote device keys.
BLE_ABS_REMOTE_BOND_INFORMATION_SPECIFIED	Delete the keys specified by the device address.
BLE_ABS_REMOTE_BOND_INFORMATION_ALL	Delete all remote device keys.

◆ ble_abs_delete_non_volatile_area_t

enum ble_abs_delete_non_volatile_area_t	
Deletion policy for non-volatile memory	
Enumerator	
BLE_ABS_DELETE_NON_VOLATILE_AREA_DISABLE	Delete no keys stored in storage.
BLE_ABS_DELETE_NON_VOLATILE_AREA_ENABLE	Delete the keys stored in storage.

4.3.51 Block Media Interface

Interfaces

Detailed Description

Interface for block media memory access.

Summary

The block media interface supports reading, writing, and erasing media devices. All functions are non-blocking if possible. The callback is used to determine when an operation completes.

Implemented by:

- SD/MMC Block Media Implementation (rm_block_media_sdmmc)
- SPI Block Media Implementation (rm_block_media_spi)
- USB HMSC Block Media Implementation (rm_block_media_usb)

Data Structures

struct	rm_block_media_info_t
struct	rm_block_media_callback_args_t
struct	rm_block_media_cfg_t
struct	rm_block_media_status_t
struct	rm_block_media_api_t
struct	rm_block_media_instance_t

Typedefs

typedef void	rm_block_media_ctrl_t

Enumerations

enum	rm_block_media_event_t

Data Structure Documentation

◆ rm_block_media_info_t

struct rm_block_media_info_t

Block media device information supported by the instance

Data Fields		
uint32_t	sector_size_bytes	Sector size in bytes.
uint32_t	num_sectors	Total number of sectors.
bool	reentrant	True if connected block media driver is reentrant.
bool	write_protected	True if connected block media device is write protected.

◆ rm_block_media_callback_args_t

struct rm_block_media_callback_args_t

Callback function parameter data

Data Fields		
rm_block_media_event_t	event	The event can be used to identify what caused the callback.
void const *	p_context	Placeholder for user data.

◆ rm_block_media_cfg_t

struct rm_block_media_cfg_t	
User configuration structure, used in open function	
Data Fields	
void(*	p_callback)(rm_block_media_callback_args_t *p_args)
	DEPRECATED - Block size, must be a power of 2 multiple of sector_size_bytes. More...
void const *	p_context
	User defined context passed into callback function.
void const *	p_extend
	Extension parameter for hardware specific settings.

Field Documentation

◆ p_callback

void(* rm_block_media_cfg_t::p_callback) (rm_block_media_callback_args_t *p_args)
DEPRECATED - Block size, must be a power of 2 multiple of sector_size_bytes. Pointer to callback function

◆ rm_block_media_status_t

struct rm_block_media_status_t		
Current status		
Data Fields		
bool	initialized	False if rm_block_media_api_t::mediaInit has not been called since media was inserted, true otherwise.
bool	busy	True if media is busy with a previous write/erase operation.
bool	media_inserted	Media insertion status, true if media is not removable.

◆ rm_block_media_api_t

struct rm_block_media_api_t

Block media interface API.

Data Fields

fsp_err_t(*	open)(rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)
fsp_err_t(*	mediaInit)(rm_block_media_ctrl_t *const p_ctrl)
fsp_err_t(*	read)(rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t(*	write)(rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t(*	erase)(rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks)
fsp_err_t(*	callbackSet)(rm_block_media_ctrl_t *const p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void const *const p_context, rm_block_media_callback_args_t *const p_callback_memory)
fsp_err_t(*	statusGet)(rm_block_media_ctrl_t *const p_ctrl, rm_block_media_status_t *const p_status)
fsp_err_t(*	infoGet)(rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info)
fsp_err_t(*	close)(rm_block_media_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* rm_block_media_api_t::open) (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_cfg_t const *const p_cfg)

Initialize block media device. rm_block_media_api_t::mediaInit must be called to complete the initialization procedure.

Implemented as

- RM_BLOCK_MEDIA_SDMMC_Open
- RM_BLOCK_MEDIA_SPI_Open
- RM_BLOCK_MEDIA_USB_Open

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ mediaInit

fsp_err_t(* rm_block_media_api_t::mediaInit) (rm_block_media_ctrl_t *const p_ctrl)

Initializes a media device. If the device is removable, it must be plugged in prior to calling this API. This function blocks until media initialization is complete.

Implemented as

- RM_BLOCK_MEDIA_SDMMC_MediaInit
- RM_BLOCK_MEDIA_SPI_MediaInit
- RM_BLOCK_MEDIA_USB_MediaInit

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.

◆ read

fsp_err_t(* rm_block_media_api_t::read) (rm_block_media_ctrl_t *const p_ctrl, uint8_t *const p_dest_address, uint32_t const block_address, uint32_t const num_blocks)

Reads blocks of data from the specified memory device address to the location specified by the caller.

Implemented as

- RM_BLOCK_MEDIA_SDMMC_Read
- RM_BLOCK_MEDIA_SPI_Read
- RM_BLOCK_MEDIA_USB_Read

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.
[out]	p_dest_address	Destination to read the data into.
[in]	block_address	Block address to read the data from.
[in]	num_blocks	Number of blocks of data to read.

◆ write

fsp_err_t(* rm_block_media_api_t::write) (rm_block_media_ctrl_t *const p_ctrl, uint8_t const *const p_src_address, uint32_t const block_address, uint32_t const num_blocks)

Writes blocks of data to the specified device memory address.

Implemented as

- RM_BLOCK_MEDIA_SDMMC_Write
- RM_BLOCK_MEDIA_SPI_Write
- RM_BLOCK_MEDIA_USB_Write

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.
[in]	p_src_address	Address to read the data to be written.
[in]	block_address	Block address to write the data to.
[in]	num_blocks	Number of blocks of data to write.

◆ erase

fsp_err_t(* rm_block_media_api_t::erase) (rm_block_media_ctrl_t *const p_ctrl, uint32_t const block_address, uint32_t const num_blocks)

Erases blocks of data from the memory device.

Implemented as

- RM_BLOCK_MEDIA_SDMMC_Erase
- RM_BLOCK_MEDIA_SPI_Erase
- RM_BLOCK_MEDIA_USB_Erase

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.
[in]	block_address	Block address to start the erase process at.
[in]	num_blocks	Number of blocks of data to erase.

◆ callbackSet

fsp_err_t(* rm_block_media_api_t::callbackSet) (rm_block_media_ctrl_t *const p_ctrl, void(*p_callback)(rm_block_media_callback_args_t *), void const *const p_context, rm_block_media_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- RM_BLOCK_MEDIA_SDMMC_CallbackSet()

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ statusGet

fsp_err_t(* rm_block_media_api_t::statusGet) (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_status_t *const p_status)

Get status of connected device.

Implemented as

- RM_BLOCK_MEDIA_SDMMC_StatusGet
- RM_BLOCK_MEDIA_SPI_StatusGet
- RM_BLOCK_MEDIA_USB_StatusGet

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.
[out]	p_status	Pointer to store current status.

◆ infoGet

fsp_err_t(* rm_block_media_api_t::infoGet) (rm_block_media_ctrl_t *const p_ctrl, rm_block_media_info_t *const p_info)

Returns information about the block media device.

Implemented as

- RM_BLOCK_MEDIA_SDMMC_InfoGet
- RM_BLOCK_MEDIA_SPI_InfoGet
- RM_BLOCK_MEDIA_USB_InfoGet

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.
[out]	p_info	Pointer to information structure. All elements of this structure will be set by the function.

◆ close

fsp_err_t(* rm_block_media_api_t::close) (rm_block_media_ctrl_t *const p_ctrl)

Closes the module.

Implemented as

- RM_BLOCK_MEDIA_SDMMC_Close
- RM_BLOCK_MEDIA_SPI_Close
- RM_BLOCK_MEDIA_USB_Close

Parameters

[in]	p_ctrl	Control block set in rm_block_media_api_t::open call.

◆ rm_block_media_instance_t

struct rm_block_media_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
rm_block_media_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
rm_block_media_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
rm_block_media_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ rm_block_media_ctrl_t

typedef void rm_block_media_ctrl_t

Block media API control block. Allocate an instance specific control block to pass into the block media API calls.

Implemented as

- rm_block_media_sdmmc_instance_ctrl_t
- rm_block_media_spi_instance_ctrl_t
- rm_block_media_usb_instance_ctrl_t

Enumeration Type Documentation

◆ **rm_block_media_event_t**

enum rm_block_media_event_t	
Events that can trigger a callback function	
Enumerator	
RM_BLOCK_MEDIA_EVENT_MEDIA_REMOVED	Media removed event.
RM_BLOCK_MEDIA_EVENT_MEDIA_INSERTED	Media inserted event.
RM_BLOCK_MEDIA_EVENT_OPERATION_COMPLETE	Read, write, or erase completed.
RM_BLOCK_MEDIA_EVENT_ERROR	Error on media operation.
RM_BLOCK_MEDIA_EVENT_POLL_STATUS	Poll rm_block_media_api_t::statusGet for write/erase completion.
RM_BLOCK_MEDIA_EVENT_MEDIA_SUSPEND	Media suspended event.
RM_BLOCK_MEDIA_EVENT_MEDIA_RESUME	Media resumed event.
RM_BLOCK_MEDIA_EVENT_WAIT	Indication to user that they should wait for an interrupt on a pending operation.
RM_BLOCK_MEDIA_EVENT_WAIT_END	Indication to user that interrupt has been received and waiting can end.

4.3.52 Communicatons Middleware Interface

Interfaces

Detailed Description

Interface for Communications Middleware functions.

Summary

The Communications interface provides multiple communications functionality.

The Communications interface can be implemented by:

- I2C Communicatons Middleware (rm_comms_i2c)

Data Structures

struct	rm_comms_write_read_params_t
struct	rm_comms_callback_args_t
struct	rm_comms_cfg_t
struct	rm_comms_api_t
struct	rm_comms_instance_t

Typedefs

typedef void	rm_comms_ctrl_t

Enumerations

enum	rm_comms_event_t

Data Structure Documentation

◆ rm_comms_write_read_params_t

struct rm_comms_write_read_params_t
Struct to pack params for writeRead

◆ rm_comms_callback_args_t

struct rm_comms_callback_args_t
Communications middleware callback parameter definition

◆ rm_comms_cfg_t

struct rm_comms_cfg_t	
Communications middleware configuration block	
Data Fields	
uint32_t	semaphore_timeout
	timeout for callback.
void(*	p_callback)(rm_comms_callback_args_t *p_args)
	Pointer to callback function, mostly used if using non-blocking functionality.
void const *	p_lower_level_cfg

	Pointer to lower level driver configuration structure.
void const *	p_extend
	Pointer to extended configuration by instance of interface.
void const *	p_context
	Pointer to the user-provided context.

◆ **rm_comms_api_t**

struct rm_comms_api_t	
COMM APIs	
Data Fields	
fsp_err_t(*	open)(rm_comms_ctrl_t *const p_ctrl, rm_comms_cfg_t const *const p_cfg)
fsp_err_t(*	close)(rm_comms_ctrl_t *const p_ctrl)
fsp_err_t(*	read)(rm_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)
fsp_err_t(*	write)(rm_comms_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)
fsp_err_t(*	writeRead)(rm_comms_ctrl_t *const p_ctrl, rm_comms_write_read_params_t write_read_params)

Field Documentation

◆ **open**

fsp_err_t(* rm_comms_api_t::open) (rm_comms_ctrl_t *const p_ctrl, rm_comms_cfg_t const *const p_cfg)

Open driver.

Implemented as

- RM_COMMS_I2C_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ **close**

fsp_err_t(* rm_comms_api_t::close) (rm_comms_ctrl_t *const p_ctrl)

Close driver.

Implemented as

- RM_COMMS_I2C_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ **read**

fsp_err_t(* rm_comms_api_t::read) (rm_comms_ctrl_t *const p_ctrl, uint8_t *const p_dest, uint32_t const bytes)

Read data.

Implemented as

- RM_COMMS_I2C_Read()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_dest	Pointer to the location to store read data.
[in]	bytes	Number of bytes to read.

◆ write

fsp_err_t(* rm_comms_api_t::write) (rm_comms_ctrl_t *const p_ctrl, uint8_t *const p_src, uint32_t const bytes)

Write data.

Implemented as

- RM_COMMS_I2C_Write()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_src	Pointer to the location to get write data from.
[in]	bytes	Number of bytes to write.

◆ writeRead

fsp_err_t(* rm_comms_api_t::writeRead) (rm_comms_ctrl_t *const p_ctrl, rm_comms_write_read_params_t write_read_params)

Write bytes over comms followed by a read, will have a struct for params.

Implemented as

- RM_COMMS_I2C_WriteRead()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	write_read_params	Parameters structure.

◆ rm_comms_instance_t

struct rm_comms_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Typedef Documentation

◆ rm_comms_ctrl_t

typedef void rm_comms_ctrl_t

Communications control block. Allocate an instance specific control block to pass into the Communications API calls.

Implemented as

- rm_comms_i2c_instance_ctrl_t

Enumeration Type Documentation

◆ rm_comms_event_t

enum rm_comms_event_t
Event in the callback function

4.3.53 FileX Block Media Port Interface

Interfaces

Detailed Description

Interface for FileX Block Media port.

Summary

The FileX block media port provides notifications for insertion and removal of removable media and provides initialization functions required by FileX.

The FileX Block media interface can be implemented by: Azure RTOS FileX Block Media I/O Driver (rm_filex_block_media)

Data Structures

struct	rm_filex_block_media_callback_args_t
struct	rm_filex_block_media_cfg_t
struct	rm_filex_block_media_api_t
struct	rm_filex_block_media_instance_t

Typedefs

typedef void	rm_filex_block_media_ctrl_t

Enumerations

enum	rm_filex_block_media_partition_t

Data Structure Documentation

◆ rm_filex_block_media_callback_args_t

struct rm_filex_block_media_callback_args_t

Callback function parameter data		
Data Fields		
rm_block_media_event_t	event	The event can be used to identify what caused the callback.
void const *	p_context	Placeholder for user data.

◆ **rm_filex_block_media_cfg_t**

struct rm_filex_block_media_cfg_t	
Block media configuration structure	
Data Fields	
rm_block_media_instance_t *	p_lower_lvl_block_media
	Lower level block media pointer.
rm_filex_block_media_partition_t	partition
	Partition to use for partitioned media.
void(*	p_callback)(rm_filex_block_media_callback_args_t *p_args)
	Pointer to callback function.

◆ **rm_filex_block_media_api_t**

struct rm_filex_block_media_api_t	
FileX block media functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(rm_filex_block_media_ctrl_t *const p_ctrl, rm_filex_block_media_cfg_t const *const p_cfg)
fsp_err_t(*	close)(rm_filex_block_media_ctrl_t *const p_ctrl)

Field Documentation

◆ **open**

fsp_err_t(* rm_filex_block_media_api_t::open) (rm_filex_block_media_ctrl_t *const p_ctrl, rm_filex_block_media_cfg_t const *const p_cfg)

Open media device.

Implemented as

- RM_FILEX_BLOCK_MEDIA_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ **close**

fsp_err_t(* rm_filex_block_media_api_t::close) (rm_filex_block_media_ctrl_t *const p_ctrl)

Close media device.

Implemented as

- RM_FILEX_BLOCK_MEDIA_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ **rm_filex_block_media_instance_t**

struct rm_filex_block_media_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

rm_filex_block_media_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
rm_filex_block_media_cfg_t const *const	p_cfg	Pointer to the configuration structure for this instance.
rm_filex_block_media_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ **rm_filex_block_media_ctrl_t**

typedef void rm_filex_block_media_ctrl_t

Block media control structure

Enumeration Type Documentation

◆ rm_filex_block_media_partition_t

enum rm_filex_block_media_partition_t	
Partitions that can be selected to use FileX with	
Enumerator	
RM_FILEX_BLOCK_MEDIA_PARTITION0	Partition 0 (0x01BE) in Master Boot Record Partition Table.
RM_FILEX_BLOCK_MEDIA_PARTITION1	Partition 1 (0x01CE) in Master Boot Record Partition Table.
RM_FILEX_BLOCK_MEDIA_PARTITION2	Partition 2 (0x01DE) in Master Boot Record Partition Table.
RM_FILEX_BLOCK_MEDIA_PARTITION3	Partition 3 (0x01EE) in Master Boot Record Partition Table.

4.3.54 FreeRTOS+FAT Port Interface

Interfaces

Detailed Description

Interface for FreeRTOS+FAT port.

Summary

The FreeRTOS+FAT port provides notifications for insertion and removal of removable media and provides initialization functions required by FreeRTOS+FAT.

The FreeRTOS+FAT interface can be implemented by: FreeRTOS+FAT Port (rm_freertos_plus_fat)

Data Structures

struct	rm_freertos_plus_fat_callback_args_t
struct	rm_freertos_plus_fat_device_t
struct	rm_freertos_plus_fat_api_t
struct	rm_freertos_plus_fat_instance_t

Enumerations

enum	rm_freertos_plus_fat_event_t
enum	rm_freertos_plus_fat_type_t

Data Structure Documentation

◆ rm_freertos_plus_fat_callback_args_t

struct rm_freertos_plus_fat_callback_args_t

Callback function parameter data

Data Fields		
rm_freertos_plus_fat_event_t	event	The event can be used to identify what caused the callback.
void const *	p_context	Placeholder for user data.

◆ rm_freertos_plus_fat_device_t

struct rm_freertos_plus_fat_device_t

Information obtained from the media device.

Data Fields		
uint32_t	sector_count	Sector count.
uint32_t	sector_size_bytes	Sector size in bytes.

◆ rm_freertos_plus_fat_api_t

struct rm_freertos_plus_fat_api_t

FreeRTOS plus Fat functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_cfg_t const *const p_cfg)
fsp_err_t(*	mediaInit)(rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_device_t *const p_device)
fsp_err_t(*	diskInit)(rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_disk_cfg_t const *const p_disk_cfg, FF_Disk_t *const p_disk)
fsp_err_t(*	diskDeinit)(rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk)
fsp_err_t(*	infoGet)(rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const

	p_disk, rm_freertos_plus_fat_info_t *const p_info)
fsp_err_t(*	close)(rm_freertos_plus_fat_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* rm_freertos_plus_fat_api_t::open) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_cfg_t const *const p_cfg)

Open media device.

Implemented as

- RM_FREERTOS_PLUS_FAT_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ mediaInit

fsp_err_t(* rm_freertos_plus_fat_api_t::mediaInit) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_device_t *const p_device)

Initializes a media device. If the device is removable, it must be plugged in prior to calling this API. This function blocks until media initialization is complete.

Implemented as

- RM_FREERTOS_PLUS_FAT_MediaInit

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_device	Pointer to store device information.

◆ diskInit

fsp_err_t(* rm_freertos_plus_fat_api_t::diskInit) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, rm_freertos_plus_fat_disk_cfg_t const *const p_disk_cfg, FF_Disk_t *const p_disk)

Initializes a FreeRTOS+FAT FF_Disk_t structure.

Implemented as

- RM_FREERTOS_PLUS_FAT_DiskInit

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_disk_cfg	Pointer to disk configurations
[out]	p_disk	Pointer to store FreeRTOS+FAT disk structure.

◆ diskDeinit

fsp_err_t(* rm_freertos_plus_fat_api_t::diskDeinit) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk)

Deinitializes a FreeRTOS+FAT FF_Disk_t structure.

Implemented as

- RM_FREERTOS_PLUS_FAT_DiskDeinit

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_disk_cfg	Pointer to disk configurations
[out]	p_disk	Pointer to store FreeRTOS+FAT disk structure.

◆ **infoGet**

fsp_err_t(* rm_freertos_plus_fat_api_t::infoGet) (rm_freertos_plus_fat_ctrl_t *const p_ctrl, FF_Disk_t *const p_disk, rm_freertos_plus_fat_info_t *const p_info)

Returns information about the media device.

Implemented as

- RM_FREERTOS_PLUS_FAT_InfoGet

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_info	Pointer to information structure. All elements of this structure will be set by the function.

◆ **close**

fsp_err_t(* rm_freertos_plus_fat_api_t::close) (rm_freertos_plus_fat_ctrl_t *const p_ctrl)

Close media device.

Implemented as

- RM_FREERTOS_PLUS_FAT_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ **rm_freertos_plus_fat_instance_t**

struct rm_freertos_plus_fat_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
rm_freertos_plus_fat_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
rm_freertos_plus_fat_cfg_t const *const	p_cfg	Pointer to the configuration structure for this instance.
rm_freertos_plus_fat_api_t const *	p_api	Pointer to the API structure for this instance.

Enumeration Type Documentation

◆ rm_freertos_plus_fat_event_t

enum rm_freertos_plus_fat_event_t	
Events that can trigger a callback function	
Enumerator	
RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_REMOVED	Media removed event.
RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_INSERTED	Media inserted event.
RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_SUSPEND	Media suspended event.
RM_FREERTOS_PLUS_FAT_EVENT_MEDIA_RESUME	Media resumed event.

◆ rm_freertos_plus_fat_type_t

enum rm_freertos_plus_fat_type_t	
Enumerator	
RM_FREERTOS_PLUS_FAT_TYPE_FAT32	FAT32 disk.
RM_FREERTOS_PLUS_FAT_TYPE_FAT16	FAT16 disk.
RM_FREERTOS_PLUS_FAT_TYPE_FAT12	FAT12 disk.

4.3.55 FSXXXX Middleware Interface

Interfaces

Detailed Description

Interface for FSXXXX Middleware functions.

Summary

The FSXXXX interface provides FSXXXX functionality.

The FSXXXX interface can be implemented by:

- FS2012 Sensor Middleware (rm_fs2012)

Data Structures

struct	rm_fsxxxx_callback_args_t
struct	rm_fsxxxx_raw_data_t
struct	rm_fsxxxx_sensor_data_t
struct	rm_fsxxxx_data_t
struct	rm_fsxxxx_cfg_t
struct	rm_fsxxxx_api_t
struct	rm_fsxxxx_instance_t

Typedefs

typedef void	rm_fsxxxx_ctrl_t

Enumerations

enum	rm_fsxxxx_event_t

Data Structure Documentation

◆ rm_fsxxxx_callback_args_t

struct rm_fsxxxx_callback_args_t
FSXXXX callback parameter definition

◆ rm_fsxxxx_raw_data_t

struct rm_fsxxxx_raw_data_t
FSXXXX raw data

◆ rm_fsxxxx_sensor_data_t

struct rm_fsxxxx_sensor_data_t

FSXXXX sensor data block

Data Fields		
int16_t	integer_part	
int16_t	decimal_part	To two decimal places.

◆ rm_fsxxxx_data_t

struct rm_fsxxxx_data_t
FSXXXX data block

◆ rm_fsxxxx_cfg_t

struct rm_fsxxxx_cfg_t	
FSXXXX Configuration	
Data Fields	
rm_comms_instance_t const *	p_instance
	Pointer to Communications Middleware instance.
void const *	p_context
	Pointer to the user-provided context.
void const *	p_extend
	Pointer to extended configuration by instance of interface.
void(*	p_callback)(rm_fsxxxx_callback_args_t *p_args)
	Pointer to callback function.

◆ rm_fsxxxx_api_t

struct rm_fsxxxx_api_t	
FSXXXX APIs	
Data Fields	
fsp_err_t(*	open)(rm_fsxxxx_ctrl_t *const p_ctrl, rm_fsxxxx_cfg_t const *const p_cfg)
fsp_err_t(*	read)(rm_fsxxxx_ctrl_t *const p_ctrl, rm_fsxxxx_raw_data_t *const p_raw_data)
fsp_err_t(*	dataCalculate)(rm_fsxxxx_ctrl_t *const p_ctrl, rm_fsxxxx_raw_data_t *const p_raw_data, rm_fsxxxx_data_t *const p_fsxxxx_data)
fsp_err_t(*	close)(rm_fsxxxx_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* rm_fsxxxx_api_t::open) (rm_fsxxxx_ctrl_t *const p_ctrl, rm_fsxxxx_cfg_t const *const p_cfg)

Open sensor.

Implemented as

- RM_FS2012_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ read

fsp_err_t(* rm_fsxxxx_api_t::read) (rm_fsxxxx_ctrl_t *const p_ctrl, rm_fsxxxx_raw_data_t *const p_raw_data)

Read ADC data from FSXXXX.

Implemented as

- RM_FS2012_Read()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_raw_data	Pointer to raw data structure.

◆ dataCalculate

fsp_err_t(* rm_fsxxxx_api_t::dataCalculate) (rm_fsxxxx_ctrl_t *const p_ctrl, rm_fsxxxx_raw_data_t *const p_raw_data, rm_fsxxxx_data_t *const p_fsxxxx_data)

Calculate flow values from ADC data.

Implemented as

- RM_FS2012_DataCalculate()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_raw_data	Pointer to raw data.
[in]	p_fsxxxx_data	Pointer to FSXXXX data structure.

◆ close

fsp_err_t(* rm_fsxxxx_api_t::close) (rm_fsxxxx_ctrl_t *const p_ctrl)
Close FSXXXX. **Implemented as** ◦ RM_FS2012_Close() **Parameters** [in] p_ctrl — Pointer to control structure.

◆ rm_fsxxxx_instance_t

struct rm_fsxxxx_instance_t

FSXXXX instance

Data Fields		
rm_fsxxxx_ctrl_t *	p_ctrl	Pointer to the control structure for this instance
rm_fsxxxx_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance
rm_fsxxxx_api_t const *	p_api	Pointer to the API structure for this instance

Typedef Documentation

◆ rm_fsxxxx_ctrl_t

typedef void rm_fsxxxx_ctrl_t
FSXXXX control block. Allocate an instance specific control block to pass into the FSXXXX API calls. **Implemented as** ◦ rm_fsxxxx_instance_ctrl_t

Enumeration Type Documentation

◆ rm_fsxxxx_event_t

enum rm_fsxxxx_event_t
Event in the callback function

4.3.56 HS300X Middleware Interface

Interfaces

Detailed Description

Interface for HS300X Middleware functions.

Summary

The HS300X interface provides HS300X functionality.

The HS300X interface can be implemented by:

- HS300X Sensor Middleware (rm_hs300x)

Data Structures

struct	rm_hs300x_callback_args_t
struct	rm_hs300x_raw_data_t
struct	rm_hs300x_sensor_data_t
struct	rm_hs300x_data_t
struct	rm_hs300x_cfg_t
struct	rm_hs300x_api_t
struct	rm_hs300x_instance_t

Typedefs

typedef void	rm_hs300x_ctrl_t

Enumerations

enum	rm_hs300x_event_t
enum	rm_hs300x_data_type_t
enum	rm_hs300x_resolution_t

Data Structure Documentation

◆ rm_hs300x_callback_args_t

struct rm_hs300x_callback_args_t
HS300X callback parameter definition

◆ rm_hs300x_raw_data_t

struct rm_hs300x_raw_data_t		
HS300X raw data		
Data Fields		
uint8_t	humidity[2]	Upper 2 bits of 0th element are data status.
uint8_t	temperature[2]	Lower 2 bits of 1st element are mask.

◆ rm_hs300x_sensor_data_t

struct rm_hs300x_sensor_data_t		
HS300X sensor data block		
Data Fields		
int16_t	integer_part	
int16_t	decimal_part	To two decimal places.

◆ rm_hs300x_data_t

struct rm_hs300x_data_t
HS300X data block

◆ rm_hs300x_cfg_t

struct rm_hs300x_cfg_t	
HS300X Configuration	
Data Fields	
rm_comms_instance_t const *	p_instance
	Pointer to Communications Middleware instance.
void const *	p_context
	Pointer to the user-provided context.
void const *	p_extend
	Pointer to extended configuration by instance of interface.

void(*	p_callback)(rm_hs300x_callback_args_t *p_args)
	Pointer to callback function.

◆ rm_hs300x_api_t

struct rm_hs300x_api_t	
HS300X APIs	
Data Fields	
fsp_err_t(*	open)(rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_cfg_t const *const p_cfg)
fsp_err_t(*	measurementStart)(rm_hs300x_ctrl_t *const p_ctrl)
fsp_err_t(*	read)(rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_raw_data_t *const p_raw_data)
fsp_err_t(*	dataCalculate)(rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_raw_data_t *const p_raw_data, rm_hs300x_data_t *const p_hs300x_data)
fsp_err_t(*	programmingModeEnter)(rm_hs300x_ctrl_t *const p_ctrl)
fsp_err_t(*	resolutionChange)(rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_data_type_t const data_type, rm_hs300x_resolution_t const resolution)
fsp_err_t(*	sensorIdGet)(rm_hs300x_ctrl_t *const p_ctrl, uint32_t *const p_sensor_id)
fsp_err_t(*	programmingModeExit)(rm_hs300x_ctrl_t *const p_ctrl)
fsp_err_t(*	close)(rm_hs300x_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* rm_hs300x_api_t::open) (rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_cfg_t const *const p_cfg)

Open sensor.

Implemented as

- RM_HS300X_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ measurementStart

fsp_err_t(* rm_hs300x_api_t::measurementStart) (rm_hs300x_ctrl_t *const p_ctrl)

Start a measurement.

Implemented as

- RM_HS300X_MeasurementStart()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ read

fsp_err_t(* rm_hs300x_api_t::read) (rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_raw_data_t *const p_raw_data)

Read ADC data from HS300X.

Implemented as

- RM_HS300X_Read()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_raw_data	Pointer to raw data structure.

◆ dataCalculate

fsp_err_t(* rm_hs300x_api_t::dataCalculate) (rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_raw_data_t *const p_raw_data, rm_hs300x_data_t *const p_hs300x_data)

Calculate humidity and temperature values from ADC data.

Implemented as

- RM_HS300X_DataCalculate()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_raw_data	Pointer to raw data.
[in]	p_hs300x_data	Pointer to HS300X data structure.

◆ programmingModeEnter

fsp_err_t(* rm_hs300x_api_t::programmingModeEnter) (rm_hs300x_ctrl_t *const p_ctrl)

Enter the programming mode.

Implemented as

- RM_HS300X_ProgrammingModeEnter()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ resolutionChange

fsp_err_t(* rm_hs300x_api_t::resolutionChange) (rm_hs300x_ctrl_t *const p_ctrl, rm_hs300x_data_type_t const data_type, rm_hs300x_resolution_t const resolution)

Change the sensor resolution.

Implemented as

- RM_HS300X_ResolutionChange()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	data_type	Data type of HS300X.
[in]	resolution	Resolution type of HS300X.

◆ sensorIdGet

fsp_err_t(* rm_hs300x_api_t::sensorIdGet) (rm_hs300x_ctrl_t *const p_ctrl, uint32_t *const p_sensor_id)

Get the sensor ID.

Implemented as

- RM_HS300X_SensorIdGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_sensor_id	Pointer to sensor ID of HS300X.

◆ programmingModeExit

fsp_err_t(* rm_hs300x_api_t::programmingModeExit) (rm_hs300x_ctrl_t *const p_ctrl)

Exit the programming mode.

Implemented as

- RM_HS300X_ProgrammingModeExit()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ close

fsp_err_t(* rm_hs300x_api_t::close) (rm_hs300x_ctrl_t *const p_ctrl)

Close HS300X.

Implemented as

- RM_HS300X_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ rm_hs300x_instance_t

struct rm_hs300x_instance_t

HS300X instance

Data Fields		
rm_hs300x_ctrl_t *	p_ctrl	Pointer to the control structure for this instance
rm_hs300x_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance

rm_hs300x_api_t const *	p_api	Pointer to the API structure for this instance

Typedef Documentation

◆ rm_hs300x_ctrl_t

typedef void rm_hs300x_ctrl_t
HS300X control block. Allocate an instance specific control block to pass into the HS300X API calls. **Implemented as** ◦ rm_hs300x_instance_ctrl_t

Enumeration Type Documentation

◆ rm_hs300x_event_t

enum rm_hs300x_event_t
Event in the callback function

◆ rm_hs300x_data_type_t

enum rm_hs300x_data_type_t
Data type of HS300X

◆ rm_hs300x_resolution_t

enum rm_hs300x_resolution_t
Resolution type of HS300X

4.3.57 LittleFS Interface

Interfaces

Detailed Description

Interface for LittleFS access.

Summary

The LittleFS Port configures a fail-safe filesystem designed for microcontrollers on top of a lower level storage device.

Implemented by: LittleFS Flash Port (rm_littlefs_flash)

Data Structures

struct	rm_littlefs_cfg_t
struct	rm_littlefs_api_t
struct	rm_littlefs_instance_t

Typedefs

typedef void	rm_littlefs_ctrl_t

Data Structure Documentation

◆ rm_littlefs_cfg_t

struct rm_littlefs_cfg_t		
User configuration structure, used in open function		
Data Fields		
struct lfs_config const *	p_lfs_cfg	Pointer LittleFS configuration structure.
void const *	p_extend	Pointer to hardware dependent configuration.

◆ rm_littlefs_api_t

struct rm_littlefs_api_t	
LittleFS Port interface API.	
Data Fields	
fsp_err_t(*	open)(rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg)
fsp_err_t(*	close)(rm_littlefs_ctrl_t *const p_ctrl)

Field Documentation

◆ **open**

fsp_err_t(* rm_littlefs_api_t::open) (rm_littlefs_ctrl_t *const p_ctrl, rm_littlefs_cfg_t const *const p_cfg)
Initialize The lower level storage device.

Implemented as

- RM_LITTLEFS_FLASH_Open

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ **close**

fsp_err_t(* rm_littlefs_api_t::close) (rm_littlefs_ctrl_t *const p_ctrl)
Closes the module and lower level storage device.

Implemented as

- RM_LITTLEFS_FLASH_Close

Parameters

[in]	p_ctrl	Control block set in rm_littlefs_api_t::open call.

◆ **rm_littlefs_instance_t**

struct rm_littlefs_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
rm_littlefs_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
rm_littlefs_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
rm_littlefs_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ rm_littlefs_ctrl_t

typedef void rm_littlefs_ctrl_t
LittleFS Port API control block. Allocate an instance specific control block to pass into the LittleFS Port API calls. **Implemented as** ◦ rm_littlefs_flash_instance_ctrl_t

4.3.58 Interface

Interfaces

Detailed Description

Interface for motor 120 control functions.

Summary

The motor 120 control interface for speed calculation and setting, fixed cycle processing

The motor 120 control interface can be implemented by:

- Motor 120 Control Sensorless (motor_120_control_sensorless)
- Motor 120 Control Hall (motor_120_control_hall)

Data Structures

struct	motor_120_control_callback_args_t
struct	motor_120_control_motor_parameter_t
struct	motor_120_control_cfg_t
struct	motor_120_control_api_t
struct	motor_120_control_instance_t

Typedefs

typedef void	motor_120_control_ctrl_t

Enumerations

enum	motor_120_control_event_t
enum	motor_120_conduction_type_t

enum	motor_120_control_status_t
enum	motor_120_control_run_mode_t
enum	motor_120_control_rotation_direction_t
enum	motor_120_control_wait_stop_flag_t
enum	motor_120_control_timeout_error_flag_t
enum	motor_120_control_pattern_error_flag_t
enum	motor_120_control_speed_ref_t
enum	motor_120_control_voltage_ref_t

Data Structure Documentation

◆ motor_120_control_callback_args_t

struct motor_120_control_callback_args_t

Callback function parameter data

Data Fields		
motor_120_control_event_t	event	Event trigger.
void const *	p_context	Placeholder for user data.

◆ motor_120_control_motor_parameter_t

struct motor_120_control_motor_parameter_t

Motor parameter for motor 120 control

Data Fields		
uint32_t	u4_motor_pp	Pole pairs.
float	f4_motor_r	Resistance (ohm)
float	f4_motor_ld	Inductance for d-axis (H)
float	f4_motor_lq	Inductance for q-axis (H)
float	f4_motor_m	Magnet flux (Wb)
float	f4_motor_j	Rotor inertia (kgm^2)

◆ motor_120_control_cfg_t

struct motor_120_control_cfg_t

Configuration parameters.

Data Fields

motor_120_conduction_type_t	conduction_type
	0:First 60 degree PWM, 1:Complementary first 60 degree PWM
uint32_t	u4_timeout_cnt
	Undetected time.
float	f4_max_drive_v
	Max output voltage (V)
float	f4_min_drive_v
	Min output voltage (V)
uint32_t	u4_speed_pi_decimation
	Speed PI control decimation counter.
uint32_t	u4_free_run_timer_freq
	Speed calc free run timer frequency (MHz)
float	f4_speed_lpf_k
	Speed LPF parameter.
float	f4_limit_speed_change
	Speed ref change limit.
float	f4_pi_ctrl_kp
	PI control error.

float	f4_pi_ctrl_ki
	PI control buffer of integral term.
float	f4_pi_ctrl_ilimit
	PI control limit of integral term.
motor_120_control_motor_parameter_t	motor_param
	Motor parameter.
void(*	p_callback)(motor_120_control_callback_args_t *p_args)
	Callback function.
void const *	p_context
	Placeholder for user data.
void const *	p_extend
	Extended configurations.

◆ motor_120_control_api_t

struct motor_120_control_api_t	
Functions implemented at the HAL layer will follow these APIs.	
Data Fields	
fsp_err_t(*	open)(motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_120_control_ctrl_t *const p_ctrl)
fsp_err_t(*	run)(motor_120_control_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(motor_120_control_ctrl_t *const p_ctrl)

fsp_err_t(*	reset)(motor_120_control_ctrl_t *const p_ctrl)
fsp_err_t(*	speedSet)(motor_120_control_ctrl_t *const p_ctrl, float const speed_rpm)
fsp_err_t(*	speedGet)(motor_120_control_ctrl_t *const p_ctrl, float *const p_speed_rpm)
fsp_err_t(*	currentGet)(motor_120_control_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status)
fsp_err_t(*	waitStopFlagGet)(motor_120_control_ctrl_t *const p_ctrl, motor_120_control_wait_stop_flag_t *const p_flag)
fsp_err_t(*	timeoutErrorFlagGet)(motor_120_control_ctrl_t *const p_ctrl, motor_120_control_timeout_error_flag_t *const p_timeout_error_flag)
fsp_err_t(*	patternErrorFlagGet)(motor_120_control_ctrl_t *const p_ctrl, motor_120_control_pattern_error_flag_t *const p_pattern_error_flag)
fsp_err_t(*	voltageRefGet)(motor_120_control_ctrl_t *const p_ctrl, motor_120_control_voltage_ref_t *const p_voltage_ref)
fsp_err_t(*	parameterUpdate)(motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_120_control_api_t::open) (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg)

Initialize the motor 120 control module.

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_Open()
- RM_MOTOR_120_CONTROL_HALL_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_120_control_api_t::close) (motor_120_control_ctrl_t *const p_ctrl)

Close the motor 120 control module

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_Close()
- RM_MOTOR_120_CONTROL_HALL_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ run

fsp_err_t(* motor_120_control_api_t::run) (motor_120_control_ctrl_t *const p_ctrl)

Run the motor 120 control module

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_Run()
- RM_MOTOR_120_CONTROL_HALL_Run()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ stop

fsp_err_t(* motor_120_control_api_t::stop) (motor_120_control_ctrl_t *const p_ctrl)

Stop the motor 120 control module

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_Stop()
- RM_MOTOR_120_CONTROL_HALL_Stop()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_120_control_api_t::reset) (motor_120_control_ctrl_t *const p_ctrl)

Reset variables of the motor 120 control module

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_Reset()
- RM_MOTOR_120_CONTROL_HALL_Reset()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ speedSet

fsp_err_t(* motor_120_control_api_t::speedSet) (motor_120_control_ctrl_t *const p_ctrl, float const speed_rpm)

Set speed[rpm]

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_SpeedSet()
- RM_MOTOR_120_CONTROL_HALL_SpeedSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	speed_rpm	Pointer to get speed data[rpm]

◆ speedGet

fsp_err_t(* motor_120_control_api_t::speedGet) (motor_120_control_ctrl_t *const p_ctrl, float *const p_speed_rpm)

Get speed.

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_SpeedGet()
- RM_MOTOR_120_CONTROL_HALL_SpeedGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_speed_rpm	Pointer to get speed data[rpm]

◆ currentGet

fsp_err_t(* motor_120_control_api_t::currentGet) (motor_120_control_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status)

Get phase current, Vdc and Va_max data.

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_CurrentGet()
- RM_MOTOR_120_CONTROL_HALL_CurrentGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_current_status	Pointer to get data structure.

◆ waitStopFlagGet

fsp_err_t(* motor_120_control_api_t::waitStopFlagGet) (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_wait_stop_flag_t *const p_flag)

Get wait stop flag.

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_WaitStopFlagGet()
- RM_MOTOR_120_CONTROL_HALL_WaitStopFlagGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_flag	Pointer to wait stop flag

◆ timeoutErrorFlagGet

fsp_err_t(* motor_120_control_api_t::timeoutErrorFlagGet) (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_timeout_error_flag_t *const p_timeout_error_flag)

Get timerout error flag.

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_TimeoutErrorFlagGet()
- RM_MOTOR_120_CONTROL_HALL_TimeoutErrorFlagGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_timeout_error_flag	Pointer to timeout error flag

◆ patternErrorFlagGet

fsp_err_t(* motor_120_control_api_t::patternErrorFlagGet) (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_pattern_error_flag_t *const p_pattern_error_flag)

Get pattern error flag.

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_PatternErrorFlagGet()
- RM_MOTOR_120_CONTROL_HALL_PatternErrorFlagGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_pattern_error_flag	Pointer to pattern error flag

◆ voltageRefGet

fsp_err_t(* motor_120_control_api_t::voltageRefGet) (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_voltage_ref_t *const p_voltage_ref)

Get voltage ref.

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_VoltageRefGet()
- RM_MOTOR_120_CONTROL_HALL_VoltageRefGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_voltage_ref	Pointer to flag voltage ref

◆ parameterUpdate

fsp_err_t(* motor_120_control_api_t::parameterUpdate) (motor_120_control_ctrl_t *const p_ctrl, motor_120_control_cfg_t const *const p_cfg)

Update configuration parameters for the calculation in the motor 120 control module

Implemented as

- RM_MOTOR_120_CONTROL_SENSORLESS_ParameterUpdate()
- RM_MOTOR_120_CONTROL_HALL_ParameterUpdate()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_120_control_instance_t

struct motor_120_control_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_120_control_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_120_control_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_120_control_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_120_control_ctrl_t

typedef void motor_120_control_ctrl_t

Control block. Allocate an instance specific control block to pass into the API calls.

Implemented as

- motor_120_control_sensorless_instance_ctrl_t
- motor_120_control_hall_instance_ctrl_t

Enumeration Type Documentation

◆ motor_120_control_event_t

enum motor_120_control_event_t	
Events that can trigger a callback function	
Enumerator	
MOTOR_120_CONTROL_EVENT_ADC_FORWARD	Event before motor 120 driver process.
MOTOR_120_CONTROL_EVENT_ADC_BACKWARD	Event after motor 120 driver process.
MOTOR_120_CONTROL_EVENT_CYCLE_FORWARD	Before cyclic process of speed control.
MOTOR_120_CONTROL_EVENT_CYCLE_BACKWARD	After cyclic process of speed control.

◆ motor_120_conduction_type_t

enum motor_120_conduction_type_t	
Enumerator	
MOTOR_120_CONDUCTION_TYPE_FIRST60	First 60 degree PWM.
MOTOR_120_CONDUCTION_TYPE_COMPLEMENTARY	Complementary first 60 degree PWM.

◆ motor_120_control_status_t

enum motor_120_control_status_t	
120 control status	
Enumerator	
MOTOR_120_CONTROL_STATUS_INACTIVE	120 control status inactive
MOTOR_120_CONTROL_STATUS_ACTIVE	120 control status active

◆ motor_120_control_run_mode_t

enum motor_120_control_run_mode_t	
Run mode	
Enumerator	
MOTOR_120_CONTROL_RUN_MODE_INIT	Run mode init.
MOTOR_120_CONTROL_RUN_MODE_BOOT	Run mode boot.
MOTOR_120_CONTROL_RUN_MODE_DRIVE	Run mode drive.

◆ motor_120_control_rotation_direction_t

enum motor_120_control_rotation_direction_t	
Rotation direction	
Enumerator	
MOTOR_120_CONTROL_ROTATION_DIRECTION_CW	Clockwise.
MOTOR_120_CONTROL_ROTATION_DIRECTION_CCW	Counter clockwise.
MOTOR_120_CONTROL_ROTATION_DIRECTION_MAX	Max value.

◆ motor_120_control_wait_stop_flag_t

enum motor_120_control_wait_stop_flag_t	
Flag for waiting for motor stop	
Enumerator	
MOTOR_120_CONTROL_WAIT_STOP_FLAG_CLEAR	Wait stop flag clear.
MOTOR_120_CONTROL_WAIT_STOP_FLAG_SET	Wait stop flag set.

◆ motor_120_control_timeout_error_flag_t

enum motor_120_control_timeout_error_flag_t	
Flag for timeout error status	
Enumerator	
MOTOR_120_CONTROL_TIMEOUT_ERROR_FLAG_CLEAR	Timeout error flag clear.
MOTOR_120_CONTROL_TIMEOUT_ERROR_FLAG_SET	Timeout error flag set.

◆ motor_120_control_pattern_error_flag_t

enum motor_120_control_pattern_error_flag_t	
Flag for pattern error status	
Enumerator	
MOTOR_120_CONTROL_PATTERN_ERROR_FLAG_CLEAR	Pattern error flag clear.
MOTOR_120_CONTROL_PATTERN_ERROR_FLAG_SET	Pattern error flag set.

◆ motor_120_control_speed_ref_t

enum motor_120_control_speed_ref_t	
Speed reference status	
Enumerator	
MOTOR_120_CONTROL_SPEED_REF_ZERO_CONST	Speed reference zero const.
MOTOR_120_CONTROL_SPEED_REF_OPENLOOP_1	Speed reference openloop 1.
MOTOR_120_CONTROL_SPEED_REF_OPENLOOP_2	Speed reference openloop 2.
MOTOR_120_CONTROL_SPEED_REF_OPENLOOP_3	Speed reference openloop 3.
MOTOR_120_CONTROL_SPEED_REF_CHANGE	Speed reference change.

◆ motor_120_control_voltage_ref_t

enum motor_120_control_voltage_ref_t	
Voltage reference status	
Enumerator	
MOTOR_120_CONTROL_VOLTAGE_REF_ZERO_CONST	Voltage reference zero const.
MOTOR_120_CONTROL_VOLTAGE_REF_UP	Voltage reference up.
MOTOR_120_CONTROL_VOLTAGE_REF_CONST	Voltage reference const.
MOTOR_120_CONTROL_VOLTAGE_REF_OPENLOOP	Voltage reference opneloop.
MOTOR_120_CONTROL_VOLTAGE_REF_PI_OUTPUT	Voltage reference pi output.

4.3.59 Interface

Interfaces

Detailed Description

Interface for motor driver functions.

Summary

The MOTOR_120_DRIVER interface for setting the PWM modulation duty

The motor current control interface can be implemented by:

- Motor 120 degree driver (rm_motor_120_driver)

Data Structures

struct	motor_120_driver_callback_args_t
struct	motor_120_driver_current_status_t
struct	motor_120_driver_cfg_t
struct	motor_120_driver_api_t
struct	motor_120_driver_instance_t

Typedefs

typedef void	motor_120_driver_ctrl_t

Enumerations

enum	motor_120_driver_event_t
enum	motor_120_driver_flag_offset_calc_t
enum	motor_120_driver_phase_pattern_t

Data Structure Documentation

◆ motor_120_driver_callback_args_t

struct motor_120_driver_callback_args_t

Callback function parameter data

Data Fields		
motor_120_driver_event_t	event	Event trigger.
void const *	p_context	Placeholder for user data.

◆ motor_120_driver_current_status_t

struct motor_120_driver_current_status_t

Current data get structure

Data Fields		
float	iu	U phase current (A)
float	iv	V phase current (A)
float	iw	W phase current (A)
float	vdc	Main line voltage (V)
float	vu	U phase voltage (V)
float	vv	V phase voltage (V)
float	vw	W phase voltage (V)

◆ motor_120_driver_cfg_t

struct motor_120_driver_cfg_t

Configuration parameters.

Data Fields

void(*	p_callback)(motor_120_driver_callback_args_t *p_args)
	Callback function.

void const *	p_context
	Placeholder for user data.
void const *	p_extend
	Placeholder for user extension.

◆ **motor_120_driver_api_t**

struct motor_120_driver_api_t	
Functions implemented at the HAL layer will follow these APIs.	
Data Fields	
fsp_err_t(*	open)(motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_120_driver_ctrl_t *const p_ctrl)
fsp_err_t(*	run)(motor_120_driver_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(motor_120_driver_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_120_driver_ctrl_t *const p_ctrl)
fsp_err_t(*	phaseVoltageSet)(motor_120_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage)
fsp_err_t(*	phasePatternSet)(motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_phase_pattern_t const pattern)
fsp_err_t(*	currentGet)(motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status)
fsp_err_t(*	currentOffsetCalc)(motor_120_driver_ctrl_t *const p_ctrl)
fsp_err_t(*	flagCurrentOffsetGet)(motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_flag_offset_calc_t *const p_flag_offset)

fsp_err_t(*	parameterUpdate)(motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_cfg_t const *const p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_120_driver_api_t::open) (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_cfg_t const *const p_cfg)

Initialize the motor 120 driver module.

Implemented as

- RM_MOTOR_120_DRIVER_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_120_driver_api_t::close) (motor_120_driver_ctrl_t *const p_ctrl)

Close the motor 120 driver module

Implemented as

- RM_MOTOR_120_DRIVER_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ run

fsp_err_t(* motor_120_driver_api_t::run) (motor_120_driver_ctrl_t *const p_ctrl)

Run the motor 120 driver module

Implemented as

- RM_MOTOR_120_DRIVER_Run()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ stop

fsp_err_t(* motor_120_driver_api_t::stop) (motor_120_driver_ctrl_t *const p_ctrl)

Stop the motor 120 driver module

Implemented as

- RM_MOTOR_120_DRIVER_Stop()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_120_driver_api_t::reset) (motor_120_driver_ctrl_t *const p_ctrl)

Reset variables of the motor 120 driver module

Implemented as

- RM_MOTOR_120_DRIVER_Reset()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ phaseVoltageSet

fsp_err_t(* motor_120_driver_api_t::phaseVoltageSet) (motor_120_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage)

Set (Input) phase voltage data into the motor 120 driver module

Implemented as

- RM_MOTOR_120_DRIVER_PhaseVoltageSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	u_voltage	U phase voltage [V]
[in]	v_voltage	V phase voltage [V]
[in]	w_voltage	W phase voltage [V]

◆ phasePatternSet

fsp_err_t(* motor_120_driver_api_t::phasePatternSet) (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_phase_pattern_t const pattern)

Set phase voltage pattern the motor 120 driver module

Implemented as

- RM_MOTOR_120_DRIVER_PhasePatternSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	pattern	Voltage pattern

◆ currentGet

fsp_err_t(* motor_120_driver_api_t::currentGet) (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_current_status_t *const p_current_status)

Get phase current, Vdc and Va_max data from the motor 120 driver module

Implemented as

- RM_MOTOR_120_DRIVER_CurrentGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_current_status	Pointer to get data structure.

◆ currentOffsetCalc

fsp_err_t(* motor_120_driver_api_t::currentOffsetCalc) (motor_120_driver_ctrl_t *const p_ctrl)

current offset detection from the motor 120 driver module

Implemented as

- RM_MOTOR_120_DRIVER_CurrentOffsetCalc()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ flagCurrentOffsetGet

fsp_err_t(* motor_120_driver_api_t::flagCurrentOffsetGet) (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_flag_offset_calc_t *const p_flag_offset)

Get the flag of finish current offset detection from the motor 120 driver module

Implemented as

- RM_MOTOR_120_DRIVER_FlagCurrentOffsetGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_flag_offset	Flag of finish current offset detection

◆ parameterUpdate

fsp_err_t(* motor_120_driver_api_t::parameterUpdate) (motor_120_driver_ctrl_t *const p_ctrl, motor_120_driver_cfg_t const *const p_cfg)

Update configuration parameters for the calculation in the motor 120 driver module

Implemented as

- RM_MOTOR_120_DRIVER_ParameterUpdate()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_120_driver_instance_t

struct motor_120_driver_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_120_driver_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_120_driver_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_120_driver_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_120_driver_ctrl_t

typedef void motor_120_driver_ctrl_t
Control block. Allocate an instance specific control block to pass into the API calls. **Implemented as** ◦ motor_120_driver_instance_ctrl_t

Enumeration Type Documentation

◆ motor_120_driver_event_t

enum motor_120_driver_event_t

Events that can trigger a callback function

Enumerator	
MOTOR_120_DRIVER_EVENT_FORWARD	Event before motor 120 driver process (before current control timing)
MOTOR_120_DRIVER_EVENT_120_CONTROL	Event 120 detection.
MOTOR_120_DRIVER_EVENT_BACKWARD	Event after motor 120 driver process (after PWM duty setting)

◆ motor_120_driver_flag_offset_calc_t

enum motor_120_driver_flag_offset_calc_t

The flag represents that the offset measurement is finished

Enumerator	
MOTOR_120_DRIVER_FLAG_OFFSET_CALC_CLEAR	Offset calculation not finished.
MOTOR_120_DRIVER_FLAG_OFFSET_CALC_OFF_FINISH	Off voltage offset calculation finished.
MOTOR_120_DRIVER_FLAG_OFFSET_CALC_ALL_FINISH	All offset calculation finished.

◆ motor_120_driver_phase_pattern_t

enum motor_120_driver_phase_pattern_t	
Phase voltage pattern	
Enumerator	
MOTOR_120_DRIVER_PHASE_PATTERN_ERROR	Phase voltage pattern error.
MOTOR_120_DRIVER_PHASE_PATTERN_UP_PWM_VN_ON	Up(PWM) to Vn(on)
MOTOR_120_DRIVER_PHASE_PATTERN_UP_PWM_WN_ON	Up(PWM) to Wn(on)
MOTOR_120_DRIVER_PHASE_PATTERN_VP_PWM_UN_ON	Vp(PWM) to Un(on)
MOTOR_120_DRIVER_PHASE_PATTERN_VP_PWM_WN_ON	Vp(PWM) to Wn(on)
MOTOR_120_DRIVER_PHASE_PATTERN_WP_PWM_UN_ON	Wp(PWM) to Un(on)
MOTOR_120_DRIVER_PHASE_PATTERN_WP_PWM_VN_ON	Wp(PWM) to Vn(on)
MOTOR_120_DRIVER_PHASE_PATTERN_UP_ON_VN_PWM	Up(on) to Vn(PWM)
MOTOR_120_DRIVER_PHASE_PATTERN_UP_ON_WN_PWM	Up(on) to Wn(PWM)
MOTOR_120_DRIVER_PHASE_PATTERN_VP_ON_UN_PWM	Vp(on) to Un(PWM)
MOTOR_120_DRIVER_PHASE_PATTERN_VP_ON_WN_PWM	Vp(on) to Wn(PWM)
MOTOR_120_DRIVER_PHASE_PATTERN_WP_ON_UN_PWM	Wp(on) to Un(PWM)
MOTOR_120_DRIVER_PHASE_PATTERN_WP_ON_VN_PWM	Wp(on) to Vn(PWM)
MOTOR_120_DRIVER_PHASE_PATTERN_U_PWM_VN_ON	U(PWM) to Vn(on)
MOTOR_120_DRIVER_PHASE_PATTERN_U_PWM_WN_ON	U(PWM) to Wn(on)
MOTOR_120_DRIVER_PHASE_PATTERN_V_PWM_UN_ON	V(PWM) to Un(on)
MOTOR_120_DRIVER_PHASE_PATTERN_V_PWM_WN_ON	V(PWM) to Wn(on)
MOTOR_120_DRIVER_PHASE_PATTERN_W_PWM_UN_ON	W(PWM) to Un(on)

MOTOR_120_DRIVER_PHASE_PATTERN_W_PWM_VN_ON	W(PWM) to Vn(on)
MOTOR_120_DRIVER_PHASE_PATTERN_UP_ON_V_PWM	Up(on) to V(PWM)
MOTOR_120_DRIVER_PHASE_PATTERN_UP_ON_W_PWM	Up(on) to W(PWM)
MOTOR_120_DRIVER_PHASE_PATTERN_VP_ON_U_PWM	Vp(on) to U(PWM)
MOTOR_120_DRIVER_PHASE_PATTERN_VP_ON_W_PWM	Vp(on) to W(PWM)
MOTOR_120_DRIVER_PHASE_PATTERN_WP_ON_U_PWM	Wp(on) to U(PWM)
MOTOR_120_DRIVER_PHASE_PATTERN_WP_ON_V_PWM	Wp(on) to V(PWM)

4.3.60 Motor angle Interface

Interfaces

Detailed Description

Interface for motor angle and speed calculation functions.

Summary

The Motor angle interface calculates the rotor angle and rotational speed from other data.

The motor angle interface can be implemented by:

- Motor Angle and Speed Estimation (rm_motor_estimate)
- Motor Angle and Speed Calculation with an Encoder (rm_motor_sense_encoder)

Data Structures

struct	motor_angle_cfg_t
struct	motor_angle_current_t
struct	motor_angle_voltage_reference_t
struct	motor_angle_encoder_info_t
struct	motor_angle_api_t

struct	motor_angle_instance_t

Typedefs

typedef void	motor_angle_ctrl_t

Enumerations

enum	motor_sense_encoder_angle_adjust_t

Data Structure Documentation

◆ motor_angle_cfg_t

struct motor_angle_cfg_t

Configuration parameters.

◆ motor_angle_current_t

struct motor_angle_current_t

Interface structure

Data Fields		
float	id	d-axis current
float	iq	q-axis current

◆ motor_angle_voltage_reference_t

struct motor_angle_voltage_reference_t

Motor angle voltage reference

Data Fields		
float	vd	d-axis voltage reference
float	vq	q-axis voltage reference

◆ motor_angle_encoder_info_t

struct motor_angle_encoder_info_t

Motor angle encoder adjustment info

Data Fields		
motor_sense_encoder_angle_adjust_t	e_adjust_status	Encoder Adjustment Status.
uint8_t	u1_adjust_count_full	Adjustment count became full.

◆ motor_angle_api_t

struct motor_angle_api_t

Functions implemented as application interface will follow these APIs.

Data Fields	
fsp_err_t(*	open)(motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_angle_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_angle_ctrl_t *const p_ctrl)
fsp_err_t(*	currentSet)(motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage)
fsp_err_t(*	speedSet)(motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed)
fsp_err_t(*	flagPiCtrlSet)(motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi)
fsp_err_t(*	internalCalculate)(motor_angle_ctrl_t *const p_ctrl)
fsp_err_t(*	angleSpeedGet)(motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err)
fsp_err_t(*	angleAdjust)(motor_angle_ctrl_t *const p_ctrl)
fsp_err_t(*	encoderCyclic)(motor_angle_ctrl_t *const p_ctrl)
fsp_err_t(*	estimatedComponentGet)(motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq)
fsp_err_t(*	infoGet)(motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info)
fsp_err_t(*	parameterUpdate)(motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_angle_api_t::open) (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *const p_cfg)

Initialize the Motor_Angle.

Implemented as

- RM_MOTOR_ESTIMATE_Open()
- RM_MOTOR_SENSE_ENCODER_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_angle_api_t::close) (motor_angle_ctrl_t *const p_ctrl)

Close (Finish) the Motor_Angle.

Implemented as

- RM_MOTOR_ESTIMATE_Close()
- RM_MOTOR_SENSE_ENCODER_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_angle_api_t::reset) (motor_angle_ctrl_t *const p_ctrl)

Reset the Motor_Angle.

Implemented as

- RM_MOTOR_ESTIMATE_Reset()
- RM_MOTOR_SENSE_ENCODER_Reset()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ currentSet

fsp_err_t(* motor_angle_api_t::currentSet) (motor_angle_ctrl_t *const p_ctrl, motor_angle_current_t *const p_st_current, motor_angle_voltage_reference_t *const p_st_voltage)

Set (Input) Current & Voltage Reference data into the Motor_Angle.

Implemented as

- RM_MOTOR_ESTIMATE_CurrentSet()
- RM_MOTOR_SENSE_ENCODER_CurrentSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_st_current	Pointer to current structure
[in]	p_st_voltage	Pointer to voltage Reference structure

◆ speedSet

fsp_err_t(* motor_angle_api_t::speedSet) (motor_angle_ctrl_t *const p_ctrl, float const speed_ctrl, float const damp_speed)

Set (Input) Speed Information into the Motor_Angle.

Implemented as

- RM_MOTOR_ESTIMATE_SpeedSet()
- RM_MOTOR_SENSE_ENCODER_SpeedSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	speed_ctrl	Control reference of rotational speed [rad/s]
[in]	damp_speed	Damping rotational speed [rad/s]

◆ flagPiCtrlSet

fsp_err_t(* motor_angle_api_t::flagPiCtrlSet) (motor_angle_ctrl_t *const p_ctrl, uint32_t const flag_pi)

Set the flag of PI Control runs.

Implemented as

- RM_MOTOR_ESTIMATE_FlagPiCtrlSet()
- RM_MOTOR_SENSE_ENCODER_FlagPiCtrlSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	flag_pi	The flag of PI control runs

◆ internalCalculate

fsp_err_t(* motor_angle_api_t::internalCalculate) (motor_angle_ctrl_t *const p_ctrl)

Calculate internal parameters of encoder process.

Implemented as

- RM_MOTOR_ESTIMATE_InternalCalculate()
- RM_MOTOR_SENSE_ENCODER_InternalCalculate()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ angleSpeedGet

fsp_err_t(* motor_angle_api_t::angleSpeedGet) (motor_angle_ctrl_t *const p_ctrl, float *const p_angle, float *const p_speed, float *const p_phase_err)

Get rotor angle and rotational speed from the Motor_Angle.

Implemented as

- RM_MOTOR_ESTIMATE_AngleSpeedGet()
- RM_MOTOR_SENSE_ENCODER_AngleSpeedGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_angl	Memory address to get rotor angle data
[out]	p_speed	Memory address to get rotational speed data
[out]	p_phase_err	Memory address to get phase(angle) error data

◆ angleAdjust

fsp_err_t(* motor_angle_api_t::angleAdjust) (motor_angle_ctrl_t *const p_ctrl)

Angle Adjustment Process.

Implemented as

- RM_MOTOR_ESTIMATE_AngleAdjust()
- RM_MOTOR_SENSE_ENCODER_AngleAdjust()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ encoderCyclic

fsp_err_t(* motor_angle_api_t::encoderCyclic) (motor_angle_ctrl_t *const p_ctrl)

Encoder Cyclic Process.

Implemented as

- RM_MOTOR_ESTIMATE_EncoderCyclic()
- RM_MOTOR_SENSE_ENCODER_EncoderCyclic()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ estimatedComponentGet

fsp_err_t(* motor_angle_api_t::estimatedComponentGet) (motor_angle_ctrl_t *const p_ctrl, float *const p_ed, float *const p_eq)

Get estimated d/q-axis component from the Motor_Angle.

Implemented as

- RM_MOTOR_ESTIMATE_EstimatedComponentGet()
- RM_MOTOR_SENSE_ENCODER_EstimatedComponentGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_ed	Memory address to get estimated d-axis component
[out]	p_eq	Memory address to get estimated q-axis component

◆ infoGet

fsp_err_t(* motor_angle_api_t::infoGet) (motor_angle_ctrl_t *const p_ctrl, motor_angle_encoder_info_t *const p_info)

Get Encoder Calculate Information.

Implemented as

- RM_MOTOR_ESTIMATE_InfoGet()
- RM_MOTOR_SENSE_ENCODER_InfoGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_info	Memory address to get encoder calculate information

◆ parameterUpdate

fsp_err_t(* motor_angle_api_t::parameterUpdate) (motor_angle_ctrl_t *const p_ctrl, motor_angle_cfg_t const *p_cfg)

Update Parameters for the calculation in the Motor_Angle.

Implemented as

- RM_MOTOR_ESTIMATE_ParameterUpdate()
- RM_MOTOR_SENSE_ENCODER_ParameterUpdate()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_angle_instance_t

struct motor_angle_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_angle_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_angle_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_angle_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_angle_ctrl_t

typedef void motor_angle_ctrl_t
Motor Angle Control block. Allocate an instance specific control block to pass into the API calls. **Implemented as** ◦ motor_angle_ctrl_t

Enumeration Type Documentation

◆ motor_sense_encoder_angle_adjust_t

enum motor_sense_encoder_angle_adjust_t

Enumerator	
MOTOR_SENSE_ENCODER_ANGLE_ADJUST_90_DEGREE	Roter Angle Adjustment to pull in 90degree.
MOTOR_SENSE_ENCODER_ANGLE_ADJUST_0_DEGREE	Roter Angle Adjustment to pull in 0degree.
MOTOR_SENSE_ENCODER_ANGLE_ADJUST_FINISH	Roter Angle Adjustment Finish.

4.3.61 Motor Interface

Interfaces

Detailed Description

Interface for Motor functions.

Summary

The Motor interface provides Motor functionality.

Implemented by:

- Motor Sensorless Vector Control (rm_motor_sensorless)
- Motor encoder vector control (rm_motor_encoder)
- Motor 120 Degree (rm_motor_120_degree)

Data Structures

struct	motor_callback_args_t

struct	motor_cfg_t
struct	motor_api_t
struct	motor_instance_t

Typedefs

typedef void	motor_ctrl_t

Enumerations

enum	motor_error_t
enum	motor_callback_event_t
enum	motor_wait_stop_flag_t

Data Structure Documentation

◆ motor_callback_args_t

struct motor_callback_args_t		
callback function parameter data		
Data Fields		
void const *	p_context	Placeholder for user data.
motor_callback_event_t	event	

◆ motor_cfg_t

struct motor_cfg_t	
Configuration parameters.	
Data Fields	
motor_speed_instance_t const *	p_motor_speed_instance
	Speed Instance.
motor_current_instance_t const *	p_motor_current_instance
	Current Instance.
void(*	p_callback)(motor_callback_args_t *p_args)

void const *	p_extend
	Placeholder for user extension.

Field Documentation

◆ p_callback

void(* motor_cfg_t::p_callback) (motor_callback_args_t *p_args)
Placeholder for user data. Passed to the user callback in motor_callback_args_t.

◆ motor_api_t

struct motor_api_t

Functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_ctrl_t *const p_ctrl)
fsp_err_t(*	run)(motor_ctrl_t *const p_ctrl)
fsp_err_t(*	stop)(motor_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_ctrl_t *const p_ctrl)
fsp_err_t(*	errorSet)(motor_ctrl_t *const p_ctrl, motor_error_t const error)
fsp_err_t(*	speedSet)(motor_ctrl_t *const p_ctrl, float const speed_rpm)
fsp_err_t(*	positionSet)(motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position)
fsp_err_t(*	statusGet)(motor_ctrl_t *const p_ctrl, uint8_t *const p_status)
fsp_err_t(*	angleGet)(motor_ctrl_t *const p_ctrl, float *const p_angle_rad)

fsp_err_t(*	speedGet)(motor_ctrl_t *const p_ctrl, float *const p_speed_rpm)
fsp_err_t(*	waitStopFlagGet)(motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag)
fsp_err_t(*	errorCheck)(motor_ctrl_t *const p_ctrl, uint16_t *const p_error)

Field Documentation

◆ open

fsp_err_t(* motor_api_t::open) (motor_ctrl_t *const p_ctrl, motor_cfg_t const *const p_cfg)

Open driver.

Implemented as

- RM_MOTOR_SENSORLESS_Open()
- RM_MOTOR_ENCODER_Open()
- RM_MOTOR_120_DEGREE_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_api_t::close) (motor_ctrl_t *const p_ctrl)

Close driver.

Implemented as

- RM_MOTOR_SENSORLESS_Close()
- RM_MOTOR_ENCODER_Close()
- RM_MOTOR_120_DEGREE_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ run

fsp_err_t(* motor_api_t::run) (motor_ctrl_t *const p_ctrl)

Run the motor. (Start the motor rotation.)

Implemented as

- RM_MOTOR_SENSORLESS_Run()
- RM_MOTOR_ENCODER_Run()
- RM_MOTOR_120_DEGREE_Run()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ stop

fsp_err_t(* motor_api_t::stop) (motor_ctrl_t *const p_ctrl)

Stop the motor. (Stop the motor rotation.)

Implemented as

- RM_MOTOR_SENSORLESS_Stop()
- RM_MOTOR_ENCODER_Stop()
- RM_MOTOR_120_DEGREE_Stop()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_api_t::reset) (motor_ctrl_t *const p_ctrl)

Reset the motor control. (Recover from the error status.)

Implemented as

- RM_MOTOR_SENSORLESS_Reset()
- RM_MOTOR_ENCODER_Reset()
- RM_MOTOR_120_DEGREE_Reset()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ errorSet

fsp_err_t(* motor_api_t::errorSet) (motor_ctrl_t *const p_ctrl, motor_error_t const error)

Set Error Information.

Implemented as

- RM_MOTOR_SENSORLESS_ErrorSet()
- RM_MOTOR_ENCODER_ErrorSet()
- RM_MOTOR_120_DEGREE_ErrorSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	error	Happend error code

◆ speedSet

fsp_err_t(* motor_api_t::speedSet) (motor_ctrl_t *const p_ctrl, float const speed_rpm)

Set rotation speed.

Implemented as

- RM_MOTOR_SENSORLESS_SpeedSet()
- RM_MOTOR_ENCODER_SpeedSet()
- RM_MOTOR_120_DEGREE_SpeedSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	speed_rpm	Required rotation speed [rpm]

◆ positionSet

fsp_err_t(* motor_api_t::positionSet) (motor_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position)

Set reference position.

Implemented as

- RM_MOTOR_SENSORLESS_PositionSet()
- RM_MOTOR_ENCODER_PositionSet()
- RM_MOTOR_120_DEGREE_PositionSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_position	Pointer to set required data

◆ statusGet

fsp_err_t(* motor_api_t::statusGet) (motor_ctrl_t *const p_ctrl, uint8_t *const p_status)

Get the motor control status.

Implemented as

- RM_MOTOR_SENSORLESS_StatusGet()
- RM_MOTOR_ENCODER_StatusGet()
- RM_MOTOR_120_DEGREE_StatusGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_status	Pointer to get the motor control status

◆ angleGet

fsp_err_t(* motor_api_t::angleGet) (motor_ctrl_t *const p_ctrl, float *const p_angle_rad)

Get the rotor angle.

Implemented as

- RM_MOTOR_SENSORLESS_AngleGet()
- RM_MOTOR_ENCODER_AngleGet()
- RM_MOTOR_120_DEGREE_AngleGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_angle_rad	Pointer to get the rotor angle [rad]

◆ speedGet

fsp_err_t(* motor_api_t::speedGet) (motor_ctrl_t *const p_ctrl, float *const p_speed_rpm)

Get the rotation speed.

Implemented as

- RM_MOTOR_SENSORLESS_SpeedGet()
- RM_MOTOR_ENCODER_SpeedGet()
- RM_MOTOR_120_DEGREE_SpeedGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_speed_rpm	Pointer to get the rotation speed [rpm]

◆ waitStopFlagGet

fsp_err_t(* motor_api_t::waitStopFlagGet) (motor_ctrl_t *const p_ctrl, motor_wait_stop_flag_t *const p_flag)

Get wait stop flag.

Implemented as

- RM_MOTOR_SENSORLESS_WaitStopFlagGet()
- RM_MOTOR_ENCODER_WaitStopFlagGet()
- RM_MOTOR_120_DEGREE_WaitStopFlagGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_flag	Pointer to wait stop flag

◆ errorCheck

fsp_err_t(* motor_api_t::errorCheck) (motor_ctrl_t *const p_ctrl, uint16_t *const p_error)

Check the error occurrence

Implemented as

- RM_MOTOR_SENSORLESS_ErrorCheck()
- RM_MOTOR_ENCODER_ErrorCheck()
- RM_MOTOR_120_DEGREE_ErrorCheck()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_error	Pointer to get occured error

◆ motor_instance_t

struct motor_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_ctrl_t

typedef void motor_ctrl_t
Motor Control block. Allocate an instance specific control block to pass into the API calls. **Implemented as** ◦ motor_instance_ctrl_t

Enumeration Type Documentation

◆ motor_error_t

enum motor_error_t
Error information

◆ motor_callback_event_t

enum motor_callback_event_t

Events that can trigger a callback function

Enumerator	
MOTOR_CALLBACK_EVENT_SPEED_FORWARD	Event forward speed control.
MOTOR_CALLBACK_EVENT_SPEED_BACKWARD	Event backward speed control.
MOTOR_CALLBACK_EVENT_CURRENT_FORWARD	Event forward current control.
MOTOR_CALLBACK_EVENT_CURRENT_BACKWARD	Event backward current control.
MOTOR_CALLBACK_EVENT_ADC_FORWARD	Event before motor 120 driver process.
MOTOR_CALLBACK_EVENT_ADC_BACKWARD	Event after motor 120 driver process.
MOTOR_CALLBACK_EVENT_CYCLE_FORWARD	Before cyclic process of speed control.
MOTOR_CALLBACK_EVENT_CYCLE_BACKWARD	After cyclic process of speed control.

◆ motor_wait_stop_flag_t

enum motor_wait_stop_flag_t	
Flag for waiting for motor stop	
Enumerator	
MOTOR_WAIT_STOP_FLAG_CLEAR	Wait stop flag clear.
MOTOR_WAIT_STOP_FLAG_SET	Wait stop flag set.

4.3.62 Motor current Interface

Interfaces

Detailed Description

Interface for motor current functions.

Summary

The Motor current interface for getting the PWM modulation duty from electric current and speed

The motor current control interface can be implemented by:

- Motor Current (rm_motor_current)

Data Structures

struct	motor_current_output_t
struct	motor_current_input_current_t
struct	motor_current_input_voltage_t
struct	motor_current_get_voltage_t
struct	motor_current_cfg_t
struct	motor_current_api_t
struct	motor_current_instance_t

Typedefs

typedef void	motor_current_ctrl_t

Enumerations

enum	motor_current_event_t

Data Structure Documentation

◆ motor_current_output_t

struct motor_current_output_t		
Structure of interface to speed control Output parameters		
Data Fields		
float	f_id	D-axis current [A].
float	f_iq	Q-axis current [A].
float	f_vamax	
float	f_speed_rad	Speed value [rad/s].
float	f_speed_rpm	Speed value [rpm].
float	f_rotor_angle	Motor rotor angle [rad].
float	f_position_rad	Motor rotor position [rad].
float	f_ed	Estimated d-axis component[V] of flux due to the permanent magnet.
float	f_eq	Estimated q-axis component[V] of flux due to the permanent magnet.
float	f_phase_err_rad	Phase error [rad].
uint8_t	u1_flag_get_iref	Flag to set d/q-axis current reference.
uint8_t	u1_adjust_status	Angle adjustment satatus.
uint8_t	u1_adjust_count_full	Angle adjustment count full.

◆ motor_current_input_current_t

struct motor_current_input_current_t		
Three-phase input current		
Data Fields		
float	iu	U phase current[A].
float	iv	V phase current[A].
float	iw	W phase current[A].

◆ motor_current_input_voltage_t

struct motor_current_input_voltage_t

Input voltage		
Data Fields		
float	vdc	Main line voltage[V].
float	va_max	Maximum magnitude of voltage vector[V].

◆ motor_current_get_voltage_t

struct motor_current_get_voltage_t		
Struct to get motor current		
Data Fields		
float	u_voltage	U phase voltage[V].
float	v_voltage	V phase voltage[V].
float	w_voltage	W phase voltage[V].
float	vd_reference	d-axis voltage reference
float	vq_reference	q-axis voltage reference

◆ motor_current_cfg_t

struct motor_current_cfg_t
Configuration parameters.

◆ motor_current_api_t

struct motor_current_api_t	
Functions implemented at the Motor Current Module will follow these APIs.	
Data Fields	
fsp_err_t(*	open)(motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_current_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_current_ctrl_t *const p_ctrl)
fsp_err_t(*	run)(motor_current_ctrl_t *const p_ctrl)
fsp_err_t(*	parameterSet)(motor_current_ctrl_t *const p_ctrl, motor_current_input_t const *const p_st_input)

fsp_err_t(*	currentReferenceSet)(motor_current_ctrl_t *const p_ctrl, float const id_reference, float const iq_reference)
fsp_err_t(*	speedPhaseSet)(motor_current_ctrl_t *const p_ctrl, float const speed_rad, float const phase_rad)
fsp_err_t(*	currentSet)(motor_current_ctrl_t *const p_ctrl, motor_current_input_current_t const *const p_st_current, motor_current_input_voltage_t const *const p_st_voltage)
fsp_err_t(*	parameterGet)(motor_current_ctrl_t *const p_ctrl, motor_current_output_t *const p_st_output)
fsp_err_t(*	currentGet)(motor_current_ctrl_t *const p_ctrl, float *const p_id, float *const p_iq)
fsp_err_t(*	phaseVoltageGet)(motor_current_ctrl_t *const p_ctrl, motor_current_get_voltage_t *const p_voltage)
fsp_err_t(*	parameterUpdate)(motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_current_api_t::open) (motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg)

Initialize the motor current module.

Implemented as

- RM_MOTOR_CURRENT_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_current_api_t::close) (motor_current_ctrl_t *const p_ctrl)

Close (Finish) the motor current module.

Implemented as

- RM_MOTOR_CURRENT_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_current_api_t::reset) (motor_current_ctrl_t *const p_ctrl)

Reset variables for the motor current module.

Implemented as

- RM_MOTOR_CURRENT_Reset()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ run

fsp_err_t(* motor_current_api_t::run) (motor_current_ctrl_t *const p_ctrl)

Activate the motor current control.

Implemented as

- RM_MOTOR_CURRENT_Run()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ parameterSet

fsp_err_t(* motor_current_api_t::parameterSet) (motor_current_ctrl_t *const p_ctrl, motor_current_input_t const *const p_st_input)

Set (Input) parameters into the motor current module.

Implemented as

- RM_MOTOR_CURRENT_ParameterSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_st_input	Pointer to input data structure(speed control output data)

◆ currentReferenceSet

fsp_err_t(* motor_current_api_t::currentReferenceSet) (motor_current_ctrl_t *const p_ctrl, float const id_reference, float const iq_reference)

Set (Input) Current reference into the motor current module.

Implemented as

- RM_MOTOR_CURRENT_CurrentReferenceSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	id_reference	D-axis current reference [A]
[in]	iq_reference	Q-axis current reference [A]

◆ speedPhaseSet

fsp_err_t(* motor_current_api_t::speedPhaseSet) (motor_current_ctrl_t *const p_ctrl, float const speed_rad, float const phase_rad)

Set (Input) Speed & Phase data into the motor current module.

Implemented as

- RM_MOTOR_CURRENT_SpeedPhaseSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	speed_rad	Rotational speed [rad/s]
[in]	phase_rad	Rotor phase [rad]

◆ currentSet

fsp_err_t(* motor_current_api_t::currentSet) (motor_current_ctrl_t *const p_ctrl, motor_current_input_current_t const *const p_st_current, motor_current_input_voltage_t const *const p_st_voltage)

Set (Input) Current data into the motor current module.

Implemented as

- RM_MOTOR_CURRENT_CurrentSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_st_current	Pointer to input current structure
[in]	p_st_voltage	Pointer to input voltage structure

◆ parameterGet

fsp_err_t(* motor_current_api_t::parameterGet) (motor_current_ctrl_t *const p_ctrl, motor_current_output_t *const p_st_output)

Get (output) parameters from the motor current module

Implemented as

- RM_MOTOR_CURRENT_ParameterGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_st_output	Pointer to output data structure(speed control input data)

◆ currentGet

fsp_err_t(* motor_current_api_t::currentGet) (motor_current_ctrl_t *const p_ctrl, float *const p_id, float *const p_iq)

Get d/q-axis current

Implemented as

- RM_MOTOR_CURRENT_CurrentGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_id	Pointer to get d-axis current [A]
[out]	p_iq	Pointer to get q-axis current [A]

◆ phaseVoltageGet

fsp_err_t(* motor_current_api_t::phaseVoltageGet) (motor_current_ctrl_t *const p_ctrl, motor_current_get_voltage_t *const p_voltage)

Get phase output voltage

Implemented as

- RM_MOTOR_CURRENT_PhaseVoltageGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_voltage	Pointer to get voltages

◆ parameterUpdate

fsp_err_t(* motor_current_api_t::parameterUpdate) (motor_current_ctrl_t *const p_ctrl, motor_current_cfg_t const *const p_cfg)

Update parameters for the calculation in the motor current control.

Implemented as

- RM_MOTOR_CURRENT_ParameterUpdate()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_current_instance_t

struct motor_current_instance_t		
This structure encompasses everything that is needed to use an instance of this interface.		
Data Fields		
motor_current_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_current_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_current_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_current_ctrl_t

typedef void motor_current_ctrl_t

Control block. Allocate an instance specific control block to pass into the API calls.

Implemented as

- motor_current_ctrl_t

Enumeration Type Documentation

◆ motor_current_event_t

enum motor_current_event_t	
Events that can trigger a callback function	
Enumerator	
MOTOR_CURRENT_EVENT_FORWARD	Event forward current control.
MOTOR_CURRENT_EVENT_DATA_SET	Event set speed control output data.
MOTOR_CURRENT_EVENT_BACKWARD	Event backward current control.

4.3.63 Motor driver Interface

Interfaces

Detailed Description

Interface for motor driver functions.

Summary

The Motor driver interface for setting the PWM modulation duty

The motor current control interface can be implemented by:

- Motor Driver (rm_motor_driver)

Data Structures

struct	motor_driver_callback_args_t
struct	motor_driver_current_get_t
struct	motor_driver_cfg_t
struct	motor_driver_api_t
struct	motor_driver_instance_t

Typedefs

typedef void	motor_driver_ctrl_t

Enumerations

enum	motor_driver_event_t
enum	motor_driver_shunt_type_t

Data Structure Documentation

◆ motor_driver_callback_args_t

struct motor_driver_callback_args_t

Callback function parameter data

Data Fields		
motor_driver_event_t	event	Event trigger.
void const *	p_context	Placeholder for user data.

◆ motor_driver_current_get_t

struct motor_driver_current_get_t

Current Data Get Structure

Data Fields		
float	iu	U phase current [A].
float	iv	V phase current [A].

float	iw	W phase current [A].
float	vdc	Main Line Voltage [V].
float	va_max	maximum magnitude of voltage vector

◆ motor_driver_cfg_t

struct motor_driver_cfg_t	
Configuration parameters.	
Data Fields	
adc_channel_t	iu_ad_ch
	A/D Channel for U Phase Current.
adc_channel_t	iv_ad_ch
	A/D Channel for V Phase Current.
adc_channel_t	iw_ad_ch
	A/D Channel for W Phase Current.
adc_channel_t	vdc_ad_ch
	A/D Channel for Main Line Voltage.
motor_driver_shunt_type_t	shunt
	Selection of shunt type.
void const *	p_context
	Placeholder for user data.

◆ motor_driver_api_t

struct motor_driver_api_t
Functions implemented at the HAL layer will follow these APIs.
Data Fields

fsp_err_t(*	open)(motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_driver_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_driver_ctrl_t *const p_ctrl)
fsp_err_t(*	phaseVoltageSet)(motor_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage)
fsp_err_t(*	currentGet)(motor_driver_ctrl_t *const p_ctrl, motor_driver_current_get_t *const p_current_get)
fsp_err_t(*	flagCurrentOffsetGet)(motor_driver_ctrl_t *const p_ctrl, uint8_t *const p_flag_offset)
fsp_err_t(*	currentOffsetRestart)(motor_driver_ctrl_t *const p_ctrl)
fsp_err_t(*	parameterUpdate)(motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_driver_api_t::open) (motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg)

Initialize the Motor Driver Module.

Implemented as

- RM_MOTOR_DRIVER_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_driver_api_t::close) (motor_driver_ctrl_t *const p_ctrl)

Close the Motor Driver Module

Implemented as

- RM_MOTOR_DRIVER_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_driver_api_t::reset) (motor_driver_ctrl_t *const p_ctrl)

Reset variables of the Motor Driver Module

Implemented as

- RM_MOTOR_DRIVER_Reset()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ phaseVoltageSet

fsp_err_t(* motor_driver_api_t::phaseVoltageSet) (motor_driver_ctrl_t *const p_ctrl, float const u_voltage, float const v_voltage, float const w_voltage)

Set (Input) Phase Voltage data into the Motor Driver Module

Implemented as

- RM_MOTOR_DRIVER_PhaseVoltageSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	u_voltage	U phase voltage [V]
[in]	v_voltage	V phase voltage [V]
[in]	w_voltage	W phase voltage [V]

◆ currentGet

fsp_err_t(* motor_driver_api_t::currentGet) (motor_driver_ctrl_t *const p_ctrl, motor_driver_current_get_t *const p_current_get)

Get Phase current, Vdc and Va_max data from the Motor Driver Module

Implemented as

- RM_MOTOR_DRIVER_CurrentGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_current_get	Pointer to get data structure.

◆ flagCurrentOffsetGet

fsp_err_t(* motor_driver_api_t::flagCurrentOffsetGet) (motor_driver_ctrl_t *const p_ctrl, uint8_t *const p_flag_offset)

Get the flag of finish current offset detection from the Motor Driver Module

Implemented as

- RM_MOTOR_DRIVER_FlagCurrentOffsetGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_flag_offset	Flag of finish current offset detection

◆ currentOffsetRestart

fsp_err_t(* motor_driver_api_t::currentOffsetRestart) (motor_driver_ctrl_t *const p_ctrl)

Restart current offset detection

Implemented as

- RM_MOTOR_DRIVER_CurrentOffsetRestart()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ parameterUpdate

fsp_err_t(* motor_driver_api_t::parameterUpdate) (motor_driver_ctrl_t *const p_ctrl, motor_driver_cfg_t const *const p_cfg)
Update Configuration Parameters for the calculation in the Motor Driver Module

Implemented as

- RM_MOTOR_DRIVER_ParameterUpdate()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_driver_instance_t

struct motor_driver_instance_t
This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_driver_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_driver_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_driver_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_driver_ctrl_t

typedef void motor_driver_ctrl_t
Control block. Allocate an instance specific control block to pass into the API calls.

Implemented as

- motor_driver_ctrl_t

Enumeration Type Documentation

◆ motor_driver_event_t

enum motor_driver_event_t	
Events that can trigger a callback function	
Enumerator	
MOTOR_DRIVER_EVENT_FORWARD	Event before Motor Driver Process (before Current Control timing)
MOTOR_DRIVER_EVENT_CURRENT	Event Current Control timing.
MOTOR_DRIVER_EVENT_BACKWARD	Event after Motor Driver Process (after PWM duty setting)

◆ motor_driver_shunt_type_t

enum motor_driver_shunt_type_t	
Selection of shunt type	
Enumerator	
MOTOR_DRIVER_SHUNT_TYPE_1_SHUNT	Only use U phase current.
MOTOR_DRIVER_SHUNT_TYPE_2_SHUNT	Use U and W phase current.
MOTOR_DRIVER_SHUNT_TYPE_3_SHUNT	Use all phase current.

4.3.64 Motor position Interface

Interfaces

Detailed Description

Interface for motor position functions.

Summary

The Motor position interface for getting the speed references from Encoder Sensor

The motor position interface can be implemented by:

- Motor Position (rm_motor_position)

Data Structures

struct	motor_position_cfg_t
struct	motor_position_api_t
struct	motor_position_instance_t

Typedefs

typedef void	motor_position_ctrl_t

Enumerations

enum	motor_position_ctrl_mode_t

Data Structure Documentation

◆ motor_position_cfg_t

struct motor_position_cfg_t

Configuration parameters.

Data Fields

void const *	p_context	Placeholder for user data.
void const *	p_extend	

◆ motor_position_api_t

struct motor_position_api_t

Functions implemented at the HAL layer will follow these APIs.

Data Fields

fsp_err_t(*	open)(motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_position_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_position_ctrl_t *const p_ctrl)
fsp_err_t(*	positionGet)(motor_position_ctrl_t *const p_ctrl, int16_t *const p_position)
fsp_err_t(*	positionSet)(motor_position_ctrl_t *const p_ctrl, float const position_rad)
fsp_err_t(*	positionReferenceSet)(motor_position_ctrl_t *const p_ctrl, int16_t const position_reference_deg)

fsp_err_t(*	controlModeSet)(motor_position_ctrl_t *const p_ctrl, motor_position_ctrl_mode_t const mode)
fsp_err_t(*	positionControl)(motor_position_ctrl_t *const p_ctrl)
fsp_err_t(*	ipdSpeedPControl)(motor_position_ctrl_t *const p_ctrl, float const ref_speed_rad, float const speed_rad, float *const p_iq_ref)
fsp_err_t(*	speedReferencePControlGet)(motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref)
fsp_err_t(*	speedReferenceIpdControlGet)(motor_position_ctrl_t *const p_ctrl, float const max_speed_rad, float *const p_speed_ref)
fsp_err_t(*	speedReferenceFeedforwardGet)(motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref)
fsp_err_t(*	parameterUpdate)(motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_position_api_t::open) (motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg)

Initialize the Motor Position Module.

Implemented as

- RM_MOTOR_POSITION_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_position_api_t::close) (motor_position_ctrl_t *const p_ctrl)

Close (Finish) the Motor Position Module.

Implemented as

- RM_MOTOR_POSITION_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_position_api_t::reset) (motor_position_ctrl_t *const p_ctrl)

Reset(Stop) the Motor Position Module.

Implemented as

- RM_MOTOR_POSITION_Reset()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ positionGet

fsp_err_t(* motor_position_api_t::positionGet) (motor_position_ctrl_t *const p_ctrl, int16_t *const p_position)

Get Position data.

Implemented as

- RM_MOTOR_POSITION_PositionGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_position	Pointer to get position data

◆ positionSet

fsp_err_t(* motor_position_api_t::positionSet) (motor_position_ctrl_t *const p_ctrl, float const position_rad)

Set Position data from Encoder.

Implemented as

- RM_MOTOR_POSITION_PositionSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	position_rad	Position data [radian]

◆ positionReferenceSet

fsp_err_t(* motor_position_api_t::positionReferenceSet) (motor_position_ctrl_t *const p_ctrl, int16_t const position_reference_deg)

Set (Input) Position reference into the Motor Position Module.

Implemented as

- RM_MOTOR_POSITION_PositionReferenceSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	position_refernce_deg	Position reference [degree]

◆ controlModeSet

fsp_err_t(* motor_position_api_t::controlModeSet) (motor_position_ctrl_t *const p_ctrl, motor_position_ctrl_mode_t const mode)

Set (Input) Position Control Mode.

Implemented as

- RM_MOTOR_POSITION_ControlModeSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	mode	Position Control Mode

◆ positionControl

fsp_err_t(* motor_position_api_t::positionControl) (motor_position_ctrl_t *const p_ctrl)

Calculate internal position reference

Implemented as

- RM_MOTOR_POSITION_PositionControl()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ ipdSpeedPControl

fsp_err_t(* motor_position_api_t::ipdSpeedPControl) (motor_position_ctrl_t *const p_ctrl, float const ref_speed_rad, float const speed_rad, float *const p_iq_ref)

Calculate iq reference

Implemented as

- RM_MOTOR_POSITION_IpdSpeedPControl()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	ref_speed_rad	Speed Reference [rad/sec]
[in]	speed_rad	Current Speed [rad/sec]
[out]	p_iq_ref	Pointer to get iq reference

◆ speedReferencePControlGet

fsp_err_t(* motor_position_api_t::speedReferencePControlGet) (motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref)

Get Speed Reference by P Control

Implemented as

- RM_MOTOR_POSITION_SpeedReferencePControlGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_speed_ref	Pointer to get speed reference

◆ speedReferenceIpdControlGet

fsp_err_t(* motor_position_api_t::speedReferenceIpdControlGet) (motor_position_ctrl_t *const p_ctrl, float const max_speed_rad, float *const p_speed_ref)

Get Speed Reference by IPD Control

Implemented as

- RM_MOTOR_POSITION_SpeedReferenceIpdControlGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_speed_ref	Pointer to get speed reference

◆ speedReferenceFeedforwardGet

fsp_err_t(* motor_position_api_t::speedReferenceFeedforwardGet) (motor_position_ctrl_t *const p_ctrl, float *const p_speed_ref)

Get Speed Reference by Speed Feedforward

Implemented as

- RM_MOTOR_POSITION_SpeedReferenceFeedforwardGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_speed_ref	Pointer to get speed reference

◆ parameterUpdate

fsp_err_t(* motor_position_api_t::parameterUpdate) (motor_position_ctrl_t *const p_ctrl, motor_position_cfg_t const *const p_cfg)

Update Parameters for the calculation in the Motor Position Module.

Implemented as

- RM_MOTOR_POSITION_ParameterUpdate()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_position_instance_t

struct motor_position_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_position_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_position_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_position_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ **motor_position_ctrl_t**

typedef void motor_position_ctrl_t

Control block. Allocate an instance specific control block to pass into the API calls.

Implemented as

- motor_position_ctrl_t

Enumeration Type Documentation

◆ **motor_position_ctrl_mode_t**

enum motor_position_ctrl_mode_t

Position Control Mode

4.3.65 Motor speed Interface

Interfaces

Detailed Description

Interface for motor speed functions.

Summary

The Motor speed interface for getting the current references from electric current and rotational speed

The motor speed interface can be implemented by:

- Motor Speed (rm_motor_speed)

Data Structures

struct	motor_speed_callback_args_t
struct	motor_speed_position_data_t
struct	motor_speed_cfg_t
struct	motor_speed_api_t
struct	motor_speed_instance_t

Typedefs

typedef void	motor_speed_ctrl_t

Enumerations

enum	motor_speed_event_t
enum	motor_speed_loop_mode_t
enum	motor_speed_step_t

Data Structure Documentation

◆ motor_speed_callback_args_t

struct motor_speed_callback_args_t

Callback function parameter data

Data Fields		
void const *	p_context	Placeholder for user data.
motor_speed_event_t	event	

◆ motor_speed_position_data_t

struct motor_speed_position_data_t

Motor speed and position structure

Data Fields		
motor_speed_step_t	e_step_mode	Select step mode.
motor_speed_loop_mode_t	e_loop_mode	Select control mode.
int16_t	position_reference_degree	Position reference [degree].

◆ motor_speed_cfg_t

struct motor_speed_cfg_t

Configuration parameters.

Data Fields	
motor_speed_input_t *	st_input
	Input data structure for automatic set.
motor_speed_output_t *	st_output
	Output data structure for automatic receive.
motor_position_instance_t const *	p_position_instance
	Position module instance.
void const *	p_context
	Placeholder for user data.

◆ **motor_speed_api_t**

struct motor_speed_api_t	
Functions implemented at the HAL layer will follow these APIs.	
Data Fields	
fsp_err_t(*	open)(motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg)
fsp_err_t(*	close)(motor_speed_ctrl_t *const p_ctrl)
fsp_err_t(*	reset)(motor_speed_ctrl_t *const p_ctrl)
fsp_err_t(*	run)(motor_speed_ctrl_t *const p_ctrl)
fsp_err_t(*	speedReferenceSet)(motor_speed_ctrl_t *const p_ctrl, float const speed_reference_rpm)
fsp_err_t(*	positionReferenceSet)(motor_speed_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position_data)

fsp_err_t(*	parameterSet)(motor_speed_ctrl_t *const p_ctrl, motor_speed_input_t const *const p_st_input)
fsp_err_t(*	speedControl)(motor_speed_ctrl_t *const p_ctrl)
fsp_err_t(*	parameterGet)(motor_speed_ctrl_t *const p_ctrl, motor_speed_output_t *const p_st_output)
fsp_err_t(*	parameterUpdate)(motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg)

Field Documentation

◆ open

fsp_err_t(* motor_speed_api_t::open) (motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg)

Initialize the motor speed module.

Implemented as

- RM_MOTOR_SPEED_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ close

fsp_err_t(* motor_speed_api_t::close) (motor_speed_ctrl_t *const p_ctrl)

Close (Finish) the motor speed module.

Implemented as

- RM_MOTOR_SPEED_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ reset

fsp_err_t(* motor_speed_api_t::reset) (motor_speed_ctrl_t *const p_ctrl)

Reset(Stop) the motor speed module.

Implemented as

- RM_MOTOR_SPEED_Reset()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ run

fsp_err_t(* motor_speed_api_t::run) (motor_speed_ctrl_t *const p_ctrl)

Activate the motor speed control.

Implemented as

- RM_MOTOR_SPEED_Run()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ speedReferenceSet

fsp_err_t(* motor_speed_api_t::speedReferenceSet) (motor_speed_ctrl_t *const p_ctrl, float const speed_reference_rpm)

Set (Input) speed reference into the motor speed module.

Implemented as

- RM_MOTOR_SPEED_SpeedReferenceSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	speed_refernce_rpm	Speed reference [rpm]

◆ positionReferenceSet

fsp_err_t(* motor_speed_api_t::positionReferenceSet) (motor_speed_ctrl_t *const p_ctrl, motor_speed_position_data_t const *const p_position_data)

Set (Input) position reference and control mode

Implemented as

- RM_MOTOR_SPEED_PositionReferenceSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_position_data	Pointer to structure position data

◆ parameterSet

fsp_err_t(* motor_speed_api_t::parameterSet) (motor_speed_ctrl_t *const p_ctrl, motor_speed_input_t const *const p_st_input)

Set (Input) speed parameters into the motor speed module.

Implemented as

- RM_MOTOR_SPEED_ParameterSet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_st_input	Pointer to structure to input parameters.

◆ speedControl

fsp_err_t(* motor_speed_api_t::speedControl) (motor_speed_ctrl_t *const p_ctrl)

Calculate current reference

Implemented as

- RM_MOTOR_SPEED_SpeedControl()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ parameterGet

fsp_err_t(* motor_speed_api_t::parameterGet) (motor_speed_ctrl_t *const p_ctrl, motor_speed_output_t *const p_st_output)

Get speed control output parameters

Implemented as

- RM_MOTOR_SPEED_ParameterGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_st_output	Pointer to get speed control parameters

◆ parameterUpdate

fsp_err_t(* motor_speed_api_t::parameterUpdate) (motor_speed_ctrl_t *const p_ctrl, motor_speed_cfg_t const *const p_cfg)

Update Parameters for the calculation in the motor speed module.

Implemented as

- RM_MOTOR_SPEED_ParameterUpdate()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure include update parameters.

◆ motor_speed_instance_t

struct motor_speed_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
motor_speed_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
motor_speed_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
motor_speed_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ motor_speed_ctrl_t

typedef void motor_speed_ctrl_t
Control block. Allocate an instance specific control block to pass into the API calls. **Implemented as** ◦ motor_speed_ctrl_t

Enumeration Type Documentation

◆ motor_speed_event_t

enum motor_speed_event_t

Events that can trigger a callback function

Enumerator	
MOTOR_SPEED_EVENT_FORWARD	Event forward speed control.
MOTOR_SPEED_EVENT_BACKWARD	Event backward speed control.
MOTOR_SPEED_EVENT_ENCODER_CYCLIC	Event encoder cyclic.
MOTOR_SPEED_EVENT_ENCODER_ADJUST	Event encoder adjust.

◆ motor_speed_loop_mode_t

enum motor_speed_loop_mode_t

Enumerator	
MOTOR_SPEED_LOOP_MODE_SPEED	Speed control mode.
MOTOR_SPEED_LOOP_MODE_POSITION	Position control mode.

◆ motor_speed_step_t

enum motor_speed_step_t

Enumerator	
MOTOR_SPEED_STEP_DISABLE	Position control works without step mode.
MOTOR_SPEED_STEP_ENABLE	Position control works with step mode.

4.3.66 Touch Middleware Interface

Interfaces

Detailed Description

Interface for Touch Middleware functions.

Summary

The TOUCH interface provides TOUCH functionality.

The TOUCH interface can be implemented by:

- Capacitive Touch Middleware (rm_touch)

Data Structures

struct	touch_button_cfg_t
struct	touch_slider_cfg_t
struct	touch_wheel_cfg_t
struct	touch_pad_cfg_t
struct	touch_cfg_t
struct	touch_sensitivity_info_t
struct	touch_api_t
struct	touch_instance_t

Macros

#define	TOUCH_COUNT_MAX Value of Maximum count.
#define	TOUCH_OFF_VALUE Value of Non-touch.

Typedefs

typedef void	touch_ctrl_t

typedef struct st_ctsu_callback_args	touch_callback_args_t

Data Structure Documentation

◆ touch_button_cfg_t

struct touch_button_cfg_t		
Configuration of each button		
Data Fields		
uint8_t	elem_index	Element number used by this button.
uint16_t	threshold	Touch/non-touch judgment threshold.
uint16_t	hysteresis	Threshold hysteresis for chattering prevention.

◆ touch_slider_cfg_t

struct touch_slider_cfg_t		
Configuration of each slider		
Data Fields		
uint8_t const *	p_elem_index	Element number array used by this slider.
uint8_t	num_elements	Number of elements used by this slider.
uint16_t	threshold	Position calculation start threshold value.

◆ touch_wheel_cfg_t

struct touch_wheel_cfg_t		
Configuration of each wheel		
Data Fields		
uint8_t const *	p_elem_index	Element number array used by this wheel.
uint8_t	num_elements	Number of elements used by this wheel.
uint16_t	threshold	Position calculation start threshold value.

◆ touch_pad_cfg_t

struct touch_pad_cfg_t
Configuration of each pads

Data Fields		
uint8_t const *	p_elem_index_rx	RX of element number arrays used by this pad.
uint8_t const *	p_elem_index_tx	TX of element number arrays used by this pad.
uint8_t	num_elements	Number of elements used by this pad.
uint16_t	threshold	Coordinate calculation threshold value.
uint16_t	rx_pixel	rx coordinate resolution
uint16_t	tx_pixel	tx coordinate resolution
uint8_t	max_touch	Maximum number of touch judgments used by the pad.
uint8_t	num_drift	Number of pad drift.

◆ touch_cfg_t

struct touch_cfg_t

User configuration structure, used in open function

Data Fields		
touch_button_cfg_t const *	p_buttons	Pointer to array of button configuration.
touch_slider_cfg_t const *	p_sliders	Pointer to array of slider configuration.
touch_wheel_cfg_t const *	p_wheels	Pointer to array of wheel configuration.
touch_pad_cfg_t const *	p_pad	Pointer of pad configuration.
uint8_t	num_buttons	Number of buttons.
uint8_t	num_sliders	Number of sliders.
uint8_t	num_wheels	Number of wheels.
uint8_t	on_freq	The cumulative number of determinations of ON.
uint8_t	off_freq	The cumulative number of determinations of OFF.
uint16_t	drift_freq	Base value drift frequency. [0 : no use].
uint16_t	cancel_freq	Maximum continuous ON. [0 : no use].
uint8_t	number	Configuration number for QE monitor.

ctsu_instance_t const *	p_ctsu_instance	Pointer to CTSU instance.
uart_instance_t const *	p_uart_instance	Pointer to UART instance.
void const *	p_context	User defined context passed into callback function.
void const *	p_extend	Pointer to extended configuration by instance of interface.

◆ touch_sensitivity_info_t

struct touch_sensitivity_info_t		
Configuration of each touch sensitivity information		
Data Fields		
uint16_t *	p_touch_sensitivity_ratio	Pointer to sensitivity ratio array.
uint16_t	old_threshold_ratio	Old threshold ratio.
uint16_t	new_threshold_ratio	New threshold ratio.
uint8_t	new_hysteresis_ratio	New hysteresis ratio.

◆ touch_api_t

struct touch_api_t	
Functions implemented at the HAL layer will follow this API.	
Data Fields	
fsp_err_t(*	open)(touch_ctrl_t *const p_ctrl, touch_cfg_t const *const p_cfg)
fsp_err_t(*	scanStart)(touch_ctrl_t *const p_ctrl)
fsp_err_t(*	dataGet)(touch_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position)
fsp_err_t(*	scanStop)(ctsu_ctrl_t *const p_ctrl)
fsp_err_t(*	padDataGet)(touch_ctrl_t *const p_ctrl, uint16_t *p_pad_rx_coordinate, uint16_t *p_pad_tx_coordinate, uint8_t *p_pad_num_touch)
fsp_err_t(*	callbackSet)(touch_ctrl_t *const p_api_ctrl, void(*p_callback)(touch_callback_args_t *), void const *const p_context, touch_callback_args_t *const p_callback_memory)

fsp_err_t(*	close)(touch_ctrl_t *const p_ctrl)
fsp_err_t(*	sensitivityRatioGet)(touch_ctrl_t *const p_ctrl, touch_sensitivity_info_t *p_touch_sensitivity_info)
fsp_err_t(*	thresholdAdjust)(touch_ctrl_t *const p_ctrl, touch_sensitivity_info_t *p_touch_sensitivity_info)
fsp_err_t(*	driftControl)(touch_ctrl_t *const p_ctrl, uint16_t input_drift_freq)

Field Documentation

◆ open

fsp_err_t(* touch_api_t::open) (touch_ctrl_t *const p_ctrl, touch_cfg_t const *const p_cfg)

Open driver.

Implemented as

- RM_TOUCH_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ scanStart

fsp_err_t(* touch_api_t::scanStart) (touch_ctrl_t *const p_ctrl)

Scan start.

Implemented as

- RM_TOUCH_ScanStart()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ dataGet

fsp_err_t(* touch_api_t::dataGet) (touch_ctrl_t *const p_ctrl, uint64_t *p_button_status, uint16_t *p_slider_position, uint16_t *p_wheel_position)

Data get.

Implemented as

- RM_TOUCH_DataGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_button_status	Pointer to get data bitmap.
[out]	p_slider_position	Pointer to get data array.
[out]	p_wheel_position	Pointer to get data array.

◆ scanStop

fsp_err_t(* touch_api_t::scanStop) (ctsu_ctrl_t *const p_ctrl)

ScanStop.

Implemented as

- RM_TOUCH_ScanStop()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ padDataGet

fsp_err_t(* touch_api_t::padDataGet) (touch_ctrl_t *const p_ctrl, uint16_t *p_pad_rx_coordinate, uint16_t *p_pad_tx_coordinate, uint8_t *p_pad_num_touch)

pad data get.

Implemented as

- RM_TOUCH_PadDataGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[out]	p_pad_rx_coordinate	Pointer to get coordinate of receiver side.
[out]	p_pad_tx_coordinate	Pointer to get coordinate of transmitter side.
[out]	p_pad_num_touch	Pointer to get touch count.

◆ callbackSet

fsp_err_t(* touch_api_t::callbackSet) (touch_ctrl_t *const p_api_ctrl, void(*p_callback)(touch_callback_args_t *), void const *const p_context, touch_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- RM_TOUCH_CallbackSet()

Parameters

[in]	p_ctrl	Pointer to the CTSU control block.
[in]	p_callback	Callback function
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* touch_api_t::close) (touch_ctrl_t *const p_ctrl)

Close driver.

Implemented as

- RM_TOUCH_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ sensitivityRatioGet

fsp_err_t(* touch_api_t::sensitivityRatioGet) (touch_ctrl_t *const p_ctrl, touch_sensitivity_info_t *p_touch_sensitivity_info)

Sensitivity ratio get.

Implemented as

- RM_TOUCH_SensitivityRatioGet()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in,out]	p_touch_sensitivity_info	Pointer to touch sensitivity structure.

◆ thresholdAdjust

fsp_err_t(* touch_api_t::thresholdAdjust) (touch_ctrl_t *const p_ctrl, touch_sensitivity_info_t *p_touch_sensitivity_info)

Threshold adjust.

Implemented as

- RM_TOUCH_ThresholdAdjust()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_touch_sensitivity_info	Pointer to touch sensitivity structure.

◆ driftControl

fsp_err_t(* touch_api_t::driftControl) (touch_ctrl_t *const p_ctrl, uint16_t input_drift_freq)

Drift control.

Implemented as

- RM_TOUCH_DriftControl()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	input_drift_freq	Drift frequency value.

◆ touch_instance_t

struct touch_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields

touch_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
touch_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
touch_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ touch_ctrl_t

typedef void touch_ctrl_t
Control block. Allocate an instance specific control block to pass into the API calls. **Implemented as** ◦ touch_instance_ctrl_t

◆ touch_callback_args_t

typedef struct st_ctsu_callback_args touch_callback_args_t
Callback function parameter data

4.3.67 Virtual EEPROM Interface

Interfaces

Detailed Description

Interface for Virtual EEPROM access.

Summary

The Virtual EEPROM Port configures a fail-safe key value store designed for microcontrollers on top of a lower level storage device.

Implemented by: Virtual EEPROM (rm_vee_flash)

Data Structures

struct	rm_vee_callback_args_t
struct	rm_vee_cfg_t

struct	rm_vee_api_t
struct	rm_vee_instance_t

Typedefs

typedef void	rm_vee_ctrl_t

Enumerations

enum	rm_vee_state_t

Data Structure Documentation

◆ rm_vee_callback_args_t

struct rm_vee_callback_args_t

User configuration structure, used in open function

Data Fields		
rm_vee_state_t	state	State of the Virtual EEPROM.
void const *	p_context	Placeholder for user data. Set in rm_vee_api_t::open function in::rm_vee_cfg_t.

◆ rm_vee_cfg_t

struct rm_vee_cfg_t

User configuration structure, used in open function

Data Fields

uint32_t	start_addr
	Start address to be used for Virtual EEPROM memory.
uint32_t	num_segments
	Number of segments to divide the volume into.
uint32_t	total_size
	Total size of the volume.
uint32_t	ref_data_size
	Size of the reference data stored at the end of the segment.

uint32_t	record_max_id
	Maximum record ID that can be used.
uint16_t *	rec_offset
	Pointer to buffer used for record offset caching.
void(*	p_callback)(rm_vee_callback_args_t *p_args)
	Callback provided when a Virtual EEPROM event occurs.
void const *	p_context
	Placeholder for user data.
void const *	p_extend
	Pointer to hardware dependent configuration.

◆ **rm_vee_api_t**

struct rm_vee_api_t	
Virtual EEPROM interface API.	
Data Fields	
fsp_err_t(*	open)(rm_vee_ctrl_t *const p_ctrl, rm_vee_cfg_t const *const p_cfg)
fsp_err_t(*	recordWrite)(rm_vee_ctrl_t *const p_ctrl, uint32_t const rec_id, uint8_t const *const p_rec_data, uint32_t num_bytes)
fsp_err_t(*	recordPtrGet)(rm_vee_ctrl_t *const p_ctrl, uint32_t rec_id, uint8_t **const pp_rec_data, uint32_t *const p_num_bytes)
fsp_err_t(*	refDataWrite)(rm_vee_ctrl_t *const p_ctrl, uint8_t const *const p_ref_data)

fsp_err_t(*	refDataPtrGet)(rm_vee_ctrl_t *const p_ctrl, uint8_t **const pp_ref_data)
fsp_err_t(*	statusGet)(rm_vee_ctrl_t *const p_ctrl, rm_vee_status_t *const p_status)
fsp_err_t(*	refresh)(rm_vee_ctrl_t *const p_ctrl)
fsp_err_t(*	format)(rm_vee_ctrl_t *const p_ctrl, uint8_t const *const p_ref_data)
fsp_err_t(*	callbackSet)(rm_vee_ctrl_t *const p_api_ctrl, void(*p_callback)(rm_vee_callback_args_t *), void const *const p_context, rm_vee_callback_args_t *const p_callback_memory)
fsp_err_t(*	close)(rm_vee_ctrl_t *const p_ctrl)

Field Documentation

◆ open

fsp_err_t(* rm_vee_api_t::open) (rm_vee_ctrl_t *const p_ctrl, rm_vee_cfg_t const *const p_cfg)

Initializes the driver's internal structures and opens the Flash driver.

Implemented as

- RM_VEE_FLASH_Open

Parameters

[in]	p_ctrl	Pointer to control block. Must be declared by user. Elements set here.
[in]	p_cfg	Pointer to configuration structure. All elements of this structure must be set by user.

◆ recordWrite

fsp_err_t(* rm_vee_api_t::recordWrite) (rm_vee_ctrl_t *const p_ctrl, uint32_t const rec_id, uint8_t const *const p_rec_data, uint32_t num_bytes)

Writes a record to data flash.

Implemented as

- RM_VEE_FLASH_RecordWrite

Parameters

[in]	p_ctrl	Pointer to control block.
[in]	rec_id	ID of record to write.
[in]	p_rec_data	Pointer to record data to write.
[in]	num_bytes	Length of data to write.

◆ recordPtrGet

fsp_err_t(* rm_vee_api_t::recordPtrGet) (rm_vee_ctrl_t *const p_ctrl, uint32_t rec_id, uint8_t **const pp_rec_data, uint32_t *const p_num_bytes)

This function gets the pointer to the most recent version of a record specified by ID.

Implemented as

- RM_VEE_FLASH_RecordPtrGet

Parameters

[in]	p_ctrl	Pointer to control block.
[in]	rec_id	ID of record to locate.
[in]	pp_rec_data	Pointer to set to the most recent version of the record.
[in]	p_num_bytes	Variable to load with record length.

◆ refDataWrite

fsp_err_t(* rm_vee_api_t::refDataWrite) (rm_vee_ctrl_t *const p_ctrl, uint8_t const *const p_ref_data)

Writes new Reference data to the reference update area.

Implemented as

- RM_VEE_FLASH_RefDataWrite

Parameters

[in]	p_ctrl	Pointer to control block.
[in]	p_ref_data	Pointer to data to write to the reference data update area.

◆ refDataPtrGet

fsp_err_t(* rm_vee_api_t::refDataPtrGet) (rm_vee_ctrl_t *const p_ctrl, uint8_t **const pp_ref_data)

Gets a pointer to the most recent reference data.

Implemented as

- RM_VEE_FLASH_RefDataPtrGet

Parameters

[in]	p_ctrl	Pointer to control block.
[in]	pp_ref_data	Pointer to set to the most recent valid reference data.

◆ statusGet

fsp_err_t(* rm_vee_api_t::statusGet) (rm_vee_ctrl_t *const p_ctrl, rm_vee_status_t *const p_status)

Get the current status of the VEE driver.

Implemented as

- RM_VEE_FLASH_StatusGet

Parameters

[in]	p_ctrl	Pointer to control block.
[in]	p_status	Pointer to store the current status of the VEE driver.

◆ refresh

fsp_err_t(* rm_vee_api_t::refresh) (rm_vee_ctrl_t *const p_ctrl)

Manually start a refresh operation.

Implemented as

- RM_VEE_FLASH_Refresh

Parameters

[in]	p_ctrl	Pointer to control block.

◆ format

fsp_err_t(* rm_vee_api_t::format) (rm_vee_ctrl_t *const p_ctrl, uint8_t const *const p_ref_data)

Format the Virtual EEPROM.

Implemented as

- RM_VEE_FLASH_Format

Parameters

[in]	p_ctrl	Pointer to control block.
[in]	p_ref_data	Optional pointer to reference data to write during format.

◆ callbackSet

fsp_err_t(* rm_vee_api_t::callbackSet) (rm_vee_ctrl_t *const p_api_ctrl, void(*p_callback)(rm_vee_callback_args_t *), void const *const p_context, rm_vee_callback_args_t *const p_callback_memory)

Specify callback function and optional context pointer and working memory pointer.

Implemented as

- RM_VEE_FLASH_CallbackSet()

Parameters

[in]	p_ctrl	Control block set in rm_vee_api_t::open call.
[in]	p_callback	Callback function to register
[in]	p_context	Pointer to send to callback function
[in]	p_working_memory	Pointer to volatile memory where callback structure can be allocated. Callback arguments allocated here are only valid during the callback.

◆ close

fsp_err_t(* rm_vee_api_t::close) (rm_vee_ctrl_t *const p_ctrl)

Closes the module and lower level storage device.

Implemented as

- RM_VEE_FLASH_Close

Parameters

[in]	p_ctrl	Control block set in rm_vee_api_t::open call.

◆ rm_vee_instance_t

struct rm_vee_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
rm_vee_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
rm_vee_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
rm_vee_api_t const *	p_api	Pointer to the API structure for

		this instance.

Typedef Documentation

◆ **rm_vee_ctrl_t**

typedef void rm_vee_ctrl_t
Virtual EEPROM API control block. Allocate an instance specific control block to pass into the VEE API calls. **Implemented as** ◦ rm_vee_flash_instance_ctrl_t

Enumeration Type Documentation

◆ **rm_vee_state_t**

enum rm_vee_state_t

Enumerator	
RM_VEE_STATE_READY	Ready.
RM_VEE_STATE_BUSY	Operation in progress.
RM_VEE_STATE_REFRESH	Refresh operation in progress.
RM_VEE_STATE_OVERFLOW	The amount of data written exceeds the space available.
RM_VEE_STATE_HARDWARE_FAIL	Lower level hardware failure.

4.3.68 ZMOD4XXX Middleware Interface

Interfaces

Detailed Description

Interface for ZMOD4XXX Middleware functions.

Summary

The ZMOD4XXX interface provides ZMOD4XXX functionality.

The ZMOD4XXX interface can be implemented by:

- ZMOD4XXX Sensor Middleware (rm_zmod4xxx)

Data Structures

struct	rm_zmod4xxx_callback_args_t
struct	rm_zmod4xxx_raw_data_t
struct	rm_zmod4xxx_iaq_1st_data_t
struct	rm_zmod4xxx_iaq_2nd_data_t
struct	rm_zmod4xxx_odor_data_t
struct	rm_zmod4xxx_sulfur_odor_data_t
struct	rm_zmod4xxx_oaq_1st_data_t
struct	rm_zmod4xxx_oaq_2nd_data_t
struct	rm_zmod4xxx_cfg_t
struct	rm_zmod4xxx_api_t
struct	rm_zmod4xxx_instance_t

Typedefs

typedef void	rm_zmod4xxx_ctrl_t

Enumerations

enum	rm_zmod4xxx_event_t
enum	rm_zmod4xxx_sulfur_odor_t

Data Structure Documentation

◆ rm_zmod4xxx_callback_args_t

struct rm_zmod4xxx_callback_args_t
ZMOD4XXX sensor API callback parameter definition

◆ rm_zmod4xxx_raw_data_t

struct rm_zmod4xxx_raw_data_t
ZMOD4XXX raw data structure

◆ rm_zmod4xxx_iaq_1st_data_t

struct rm_zmod4xxx_iaq_1st_data_t		
ZMOD4XXX IAQ 1st gen data structure		
Data Fields		
float	rmox	MOx resistance.
float	rcda	CDA resistance.
float	iaq	IAQ index.
float	tvoc	TVOC concentration (mg/m^3).
float	etoh	EtOH concentration (ppm).
float	eco2	eCO2 concentration (ppm).

◆ **rm_zmod4xxx_iaq_2nd_data_t**

struct rm_zmod4xxx_iaq_2nd_data_t		
ZMOD4XXX IAQ 2nd gen data structure		
Data Fields		
float	rmox[13]	MOx resistance.
float	log_rcda	log10 of CDA resistance.
float	iaq	IAQ index.
float	tvoc	TVOC concentration (mg/m^3).
float	etoh	EtOH concentration (ppm).
float	eco2	eCO2 concentration (ppm).

◆ **rm_zmod4xxx_odor_data_t**

struct rm_zmod4xxx_odor_data_t		
ZMOD4XXX Odor structure		
Data Fields		
bool	control_signal	Control signal input for odor lib.
float	odor	Concentration ratio for odor lib.

◆ **rm_zmod4xxx_sulfur_odor_data_t**

struct rm_zmod4xxx_sulfur_odor_data_t		
ZMOD4XXX Sulfur-Odor structure		
Data Fields		
float	rmox[9]	MOx resistance.
float	intensity	odor intensity rating ranges from 0.0 to 5.0 for sulfur lib
rm_zmod4xxx_sulfur_odor_t	odor	sulfur_odor classification for lib

◆ rm_zmod4xxx_oaq_1st_data_t

struct rm_zmod4xxx_oaq_1st_data_t		
ZMOD4XXX OAQ 1st gen data structure		
Data Fields		
float	rmox[15]	MOx resistance.
float	aiq	Air Quality.

◆ rm_zmod4xxx_oaq_2nd_data_t

struct rm_zmod4xxx_oaq_2nd_data_t		
ZMOD4XXX OAQ 2nd gen data structure		
Data Fields		
float	rmox[8]	MOx resistance.
float	ozone_concentration	The ozone concentration in part-per-billion.
uint16_t	fast_aqi	1-minute average of the Air Quality Index according to the EPA standard based on ozone
uint16_t	epa_aqi	The Air Quality Index according to the EPA standard based on ozone.

◆ rm_zmod4xxx_cfg_t

struct rm_zmod4xxx_cfg_t	
ZMOD4XXX configuration block	
Data Fields	
rm_comms_instance_t const *	p_comms_instance
	Pointer to Communications Middleware instance.
void const *	p_irq_instance
	Pointer to IRQ instance.
void const *	p_zmod4xxx_device
	[DEPRECATED] This will be removed in FSP v4.0.0. Please use rm_zmod4xxx_lib_extended_cfg_t in rm_zmod4xxx.h

void const *	p_zmod4xxx_handle
	[DEPRECATED] This will be removed in FSP v4.0.0. Please use rm_zmod4xxx_lib_extended_cfg_t in rm_zmod4xxx.h
void const *	p_zmod4xxx_results
	[DEPRECATED] This will be removed in FSP v4.0.0. Please use rm_zmod4xxx_lib_extended_cfg_t in rm_zmod4xxx.h
void const *	p_context
	Pointer to the user-provided context.
void const *	p_extend
	Pointer to extended configuration by instance of interface.
void(*	p_comms_callback)(rm_zmod4xxx_callback_args_t *p_args)
	I2C Communications callback.
void(*	p_irq_callback)(rm_zmod4xxx_callback_args_t *p_args)
	IRQ callback.

◆ rm_zmod4xxx_api_t

struct rm_zmod4xxx_api_t	
ZMOD4XXX APIs	
Data Fields	
fsp_err_t(*	open)(rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_cfg_t const *const p_cfg)
fsp_err_t(*	measurementStart)(rm_zmod4xxx_ctrl_t *const p_api_ctrl)
fsp_err_t(*	measurementStop)(rm_zmod4xxx_ctrl_t *const p_api_ctrl)

fsp_err_t(*	statusCheck)(rm_zmod4xxx_ctrl_t *const p_api_ctrl)
fsp_err_t(*	read)(rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data)
fsp_err_t(*	iaq1stGenDataCalculate)(rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_1st_data_t *const p_zmod4xxx_data)
fsp_err_t(*	iaq2ndGenDataCalculate)(rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_2nd_data_t *const p_zmod4xxx_data)
fsp_err_t(*	odorDataCalculate)(rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_odor_data_t *const p_zmod4xxx_data)
fsp_err_t(*	sulfurOdorDataCalculate)(rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_sulfur_odor_data_t *const p_zmod4xxx_data)
fsp_err_t(*	oaq1stGenDataCalculate)(rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_oaq_1st_data_t *const p_zmod4xxx_data)
fsp_err_t(*	oaq2ndGenDataCalculate)(rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_oaq_2nd_data_t *const p_zmod4xxx_data)
fsp_err_t(*	temperatureAndHumiditySet)(rm_zmod4xxx_ctrl_t *const p_api_ctrl, float temperature, float humidity)
fsp_err_t(*	close)(rm_zmod4xxx_ctrl_t *const p_api_ctrl)

Field Documentation

◆ open

fsp_err_t(* rm_zmod4xxx_api_t::open) (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_cfg_t const *const p_cfg)

Open sensor.

Implemented as

- RM_ZMOD4XXX_Open()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to configuration structure.

◆ measurementStart

fsp_err_t(* rm_zmod4xxx_api_t::measurementStart) (rm_zmod4xxx_ctrl_t *const p_api_ctrl)

Start measurement

Implemented as

- RM_ZMOD4XXX_MeasurementStart()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ measurementStop

fsp_err_t(* rm_zmod4xxx_api_t::measurementStop) (rm_zmod4xxx_ctrl_t *const p_api_ctrl)

Stop measurement

Implemented as

- RM_ZMOD4XXX_MeasurementStop()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ statusCheck

fsp_err_t(* rm_zmod4xxx_api_t::statusCheck) (rm_zmod4xxx_ctrl_t *const p_api_ctrl)

Read status of the sensor

Implemented as

- RM_ZMOD4XXX_StatusCheck()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ read

fsp_err_t(* rm_zmod4xxx_api_t::read) (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data)

Read ADC data.

Implemented as

- RM_ZMOD4XXX_Read()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_raw_data	Pointer to raw data structure.

◆ iaq1stGenDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::iaq1stGenDataCalculate) (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_1st_data_t *const p_zmod4xxx_data)

Calculate IAQ 1st Gen. values from ADC data.

Implemented as

- RM_ZMOD4XXX_Iaq1stGenDataCalculate()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_raw_data	Pointer to raw data.
[in]	p_zmod4xxx_data	Pointer to ZMOD4XXXX data structure.

◆ iaq2ndGenDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::iaq2ndGenDataCalculate) (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_iaq_2nd_data_t *const p_zmod4xxx_data)

Calculate IAQ 2nd Gen. values from ADC data.

Implemented as

- RM_ZMOD4XXX_Iaq2ndGenDataCalculate()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_raw_data	Pointer to raw data.
[in]	p_zmod4xxx_data	Pointer to ZMOD4XXXX data structure.

◆ odorDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::odorDataCalculate) (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_odor_data_t *const p_zmod4xxx_data)

Calculate Odor values from ADC data.

Implemented as

- RM_ZMOD4XXX_OdorDataCalculate()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_raw_data	Pointer to raw data.
[in]	p_zmod4xxx_data	Pointer to ZMOD4XXXX data structure.

◆ sulfurOdorDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::sulfurOdorDataCalculate) (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_sulfur_odor_data_t *const p_zmod4xxx_data)

Calculate Sulfur Odor values from ADC data.

Implemented as

- RM_ZMOD4XXX_SulfurOdorDataCalculate()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_raw_data	Pointer to raw data.
[in]	p_zmod4xxx_data	Pointer to ZMOD4XXXX data structure.

◆ oaq1stGenDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::oaq1stGenDataCalculate) (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_oaq_1st_data_t *const p_zmod4xxx_data)

Calculate OAQ 1st Gen. values from ADC data.

Implemented as

- RM_ZMOD4XXX_Oaq1stGenDataCalculate()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_raw_data	Pointer to raw data.
[in]	p_zmod4xxx_data	Pointer to ZMOD4XXXX data structure.

◆ oaq2ndGenDataCalculate

fsp_err_t(* rm_zmod4xxx_api_t::oaq2ndGenDataCalculate) (rm_zmod4xxx_ctrl_t *const p_api_ctrl, rm_zmod4xxx_raw_data_t *const p_raw_data, rm_zmod4xxx_oaq_2nd_data_t *const p_zmod4xxx_data)

Calculate OAQ 2nd Gen. values from ADC data.

Implemented as

- RM_ZMOD4XXX_Oaq2ndGenDataCalculate()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	p_raw_data	Pointer to raw data.
[in]	p_zmod4xxx_data	Pointer to ZMOD4XXXX data structure.

◆ temperatureAndHumiditySet

fsp_err_t(* rm_zmod4xxx_api_t::temperatureAndHumiditySet) (rm_zmod4xxx_ctrl_t *const p_api_ctrl, float temperature, float humidity)

Set temperature and humidity.

Implemented as

- RM_ZMOD4XXX_TemperatureAndHumiditySet()

Parameters

[in]	p_api_ctrl	Pointer to control structure.
[in]	temperature	Temperature (deg C).
[in]	humidity	Humidity (percent).

◆ close

fsp_err_t(* rm_zmod4xxx_api_t::close) (rm_zmod4xxx_ctrl_t *const p_api_ctrl)

Close the sensor

Implemented as

- RM_ZMOD4XXX_Close()

Parameters

[in]	p_api_ctrl	Pointer to control structure.

◆ rm_zmod4xxx_instance_t

struct rm_zmod4xxx_instance_t

ZMOD4XXX instance

Data Fields		
rm_zmod4xxx_ctrl_t *	p_ctrl	Pointer to the control structure for this instance
rm_zmod4xxx_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance
rm_zmod4xxx_api_t const *	p_api	Pointer to the API structure for this instance

Typedef Documentation

◆ rm_zmod4xxx_ctrl_t

typedef void rm_zmod4xxx_ctrl_t

ZMOD4xxx Control block. Allocate an instance specific control block to pass into the API calls.

Implemented as

- rm_zmod4xxx_instance_ctrl_t

Enumeration Type Documentation

◆ rm_zmod4xxx_event_t

enum rm_zmod4xxx_event_t

Event in the callback function

◆ rm_zmod4xxx_sulfur_odor_t

enum rm_zmod4xxx_sulfur_odor_t

Sulfur-Odor status

4.3.69 SCE Interface

Interfaces

Detailed Description

Interface for Secure Crypto Engine (SCE) functions.

Summary

The SCE interface provides SCE functionality.

The SCE interface can be implemented by:

- Secure Crypto Engine (r_sce_protected)

Data Structures

struct	sce_byte_data_t
struct	sce_aes_wrapped_key_t
struct	sce_hmac_sha_wrapped_key_t
struct	sce_rsa1024_public_wrapped_key_t
struct	sce_rsa1024_private_wrapped_key_t
struct	sce_rsa2048_public_wrapped_key_t
struct	sce_rsa2048_private_wrapped_key_t
struct	sce_rsa3072_public_wrapped_key_t
struct	sce_rsa4096_public_wrapped_key_t
struct	sce_rsa1024_wrapped_pair_key_t
struct	sce_rsa2048_wrapped_pair_key_t
struct	sce_ecc_public_wrapped_key_t
struct	sce_ecc_private_wrapped_key_t
struct	sce_ecc_wrapped_pair_key_t
struct	sce_ecdh_wrapped_key_t
struct	sce_key_update_key_t
struct	sce_aes_handle_t
struct	sce_gcm_handle_t
struct	sce_ccm_handle_t
struct	sce_cmac_handle_t
struct	sce_sha_md5_handle_t

struct	sce_hmac_sha_handle_t
struct	sce_ecdh_handle_t
struct	sce_cfg_t
struct	sce_api_t
struct	sce_instance_t

Typedefs

typedef sce_byte_data_t	sce_rsa_byte_data_t byte data More...
typedef sce_byte_data_t	sce_ecdsa_byte_data_t byte data More...
typedef void	sce_ctrl_t

Enumerations

enum	lifecycle_t

Data Structure Documentation

◆ sce_byte_data_t

struct sce_byte_data_t

Byte data structure

Data Fields

uint8_t *	pdata	pointer
uint32_t	data_length	data_length
uint32_t	data_type	data type

◆ sce_aes_wrapped_key_t

struct sce_aes_wrapped_key_t

AES wrapped key data structure. DO NOT MODIFY.

Data Fields

uint32_t	type	key type
uint32_t	value[SCE_TLS_AES256_KEY_INDEX_WORD_SIZE]	wrapped key value

◆ sce_hmac_sha_wrapped_key_t

struct sce_hmac_sha_wrapped_key_t		
HMAC-SHA wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	key type
uint32_t	value[SCE_TLS_HMAC_KEY_INDEX_WORD_SIZE]	wrapped key value

◆ sce_rsa1024_public_wrapped_key_t

struct sce_rsa1024_public_wrapped_key_t		
RSA 1024bit public wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	key type
struct sce_rsa1024_public_wrapped_key_t	value	

◆ sce_rsa1024_private_wrapped_key_t

struct sce_rsa1024_private_wrapped_key_t		
RSA 1024bit private wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	key type
struct sce_rsa1024_private_wrapped_key_t	value	

◆ sce_rsa2048_public_wrapped_key_t

struct sce_rsa2048_public_wrapped_key_t		
RSA 2048bit public wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	Key type.
struct sce_rsa2048_public_wrapped_key_t	value	

◆ sce_rsa2048_private_wrapped_key_t

struct sce_rsa2048_private_wrapped_key_t		
RSA 2048bit private wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	key type

struct sce_rsa2048_private_wrapped_key_t	value	

◆ **sce_rsa3072_public_wrapped_key_t**

struct sce_rsa3072_public_wrapped_key_t		
RSA 3072bit public wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	Key type.
struct sce_rsa3072_public_wrapped_key_t	value	

◆ **sce_rsa4096_public_wrapped_key_t**

struct sce_rsa4096_public_wrapped_key_t		
RSA 4096bit public wrapped key data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	Key type.
struct sce_rsa4096_public_wrapped_key_t	value	

◆ **sce_rsa1024_wrapped_pair_key_t**

struct sce_rsa1024_wrapped_pair_key_t		
RSA 1024bit wrapped key pair structure. DO NOT MODIFY.		
Data Fields		
sce_rsa1024_private_wrapped_key_t	priv_key	RSA 1024-bit private wrapped key.
sce_rsa1024_public_wrapped_key_t	pub_key	RSA 1024-bit public wrapped key.

◆ **sce_rsa2048_wrapped_pair_key_t**

struct sce_rsa2048_wrapped_pair_key_t		
RSA 2048bit wrapped key pair structure. DO NOT MODIFY.		
Data Fields		
sce_rsa2048_private_wrapped_key_t	priv_key	RSA 2048-bit private wrapped key.
sce_rsa2048_public_wrapped_key_t	pub_key	RSA 2048-bit public wrapped key.

◆ **sce_ecc_public_wrapped_key_t**

struct sce_ecc_public_wrapped_key_t		
ECC P-192/224/256 public wrapped key data structure		
Data Fields		
uint32_t	type	key type
struct sce_ecc_public_wrapped_key_t	value	

◆ sce_ecc_private_wrapped_key_t

struct sce_ecc_private_wrapped_key_t		
ECC P-192/224/256 private wrapped key data structure		
Data Fields		
uint32_t	type	key type
uint32_t	value[HW_SCE_ECC_PRIVATE_KEY_MANAGEMENT_INFO_WORD_SIZE]	wrapped key value

◆ sce_ecc_wrapped_pair_key_t

struct sce_ecc_wrapped_pair_key_t		
ECC P-192/224/256 wrapped key pair structure		
Data Fields		
sce_ecc_private_wrapped_key_t	priv_key	ECC private wrapped key.
sce_ecc_public_wrapped_key_t	pub_key	ECC public wrapped key.

◆ sce_ecdh_wrapped_key_t

struct sce_ecdh_wrapped_key_t		
ECDH wrapped key data structure		
Data Fields		
uint32_t	type	key type
uint32_t	value[HW_SCE_SHARED_SECRET_KEY_INDEX_WORD_SIZE]	wrapped key value

◆ sce_key_update_key_t

struct sce_key_update_key_t		
Update key ring index data structure. DO NOT MODIFY.		
Data Fields		
uint32_t	type	key type
uint32_t	value[HW_SCE_UPDATE_KEY_RING_INDEX_WORD_SIZE]	wrapped key value

◆ sce_aes_handle_t

struct sce_aes_handle_t		
The work area for AES. DO NOT MODIFY.		
Data Fields		
uint32_t	id	serial number of this handle
sce_aes_wrapped_key_t	wrapped_key	wrapped key
uint32_t	current_input_data_size	text size under encryption / decryption
uint8_t	last_1_block_as_fraction[HW_SCE_AES_BLOCK_BYTE_SIZE]	text array less than the block long
uint8_t	last_2_block_as_fraction[HW_SCE_AES_BLOCK_BYTE_SIZE *2]	reserved
uint8_t	current_initial_vector[HW_SCE_AES_CBC_IV_BYTE_SIZE]	current initialization vector used in CBC mode
uint8_t	flag_call_init	control flag of calling function

◆ sce_gcm_handle_t

struct sce_gcm_handle_t		
The work area for GCM. DO NOT MODIFY.		
Data Fields		
uint32_t	id	serial number of this handle
sce_aes_wrapped_key_t	wrapped_key	wrapped key
uint8_t	gcm_buffer[HW_SCE_AES_BLOCK_BYTE_SIZE]	text array less than the block long
uint8_t	gcm_aad_buffer[HW_SCE_AES_GCM_AAD_BLOCK_BYTE_SIZE]	AAD array less than the block long.
uint32_t	all_received_length	entire length of text
uint32_t	all_received_aad_length	entire length of text
uint32_t	buffering_length	text array length less than the block long
uint32_t	buffering_aad_length	AAD array length less than the block long.
uint8_t	flag_call_init	control flag of calling function
uint8_t	flag_update_input_data	control flag of next input data

◆ sce_ccm_handle_t

struct sce_ccm_handle_t
The work area for CCM. DO NOT MODIFY.
Data Fields

uint32_t	id	serial number of this handle
sce_aes_wrapped_key_t	wrapped_key	wrapped key
uint8_t	formatted_data[HW_SCE_AES_CCM_B_FORMAT_BYTE_SIZE]	formatted data area
uint8_t	counter[HW_SCE_AES_CCM_COUNTER_BYTE_SIZE]	counter of CTR mode
uint8_t	ccm_buffer[HW_SCE_AES_BLOCK_BYTE_SIZE]	text array less than the block long
uint32_t	all_received_length	entire length of text
uint32_t	buffering_length	text array length less than the block long
uint8_t	flag_call_init	control flag of calling function

◆ sce_cmac_handle_t

struct sce_cmac_handle_t		
The work area for CMAC. DO NOT MODIFY.		
Data Fields		
uint32_t	id	serial number of this handle
sce_aes_wrapped_key_t	wrapped_key	wrapped key
uint8_t	cmac_buffer[HW_SCE_AES_BLOCK_BYTE_SIZE]	message array less than the block long
uint32_t	all_received_length	entire length of message
uint32_t	buffering_length	message array length less than the block long
uint8_t	flag_call_init	control flag of calling function

◆ sce_sha_md5_handle_t

struct sce_sha_md5_handle_t		
The work area for SHA. DO NOT MODIFY.		
Data Fields		
uint32_t	id	serial number of this handle
uint8_t	sha_buffer[HW_SCE_SHA256_HASH_LENGTH_BYTE_SIZE *4]	message array length less than the block long
uint32_t	all_received_length	entire length of message
uint32_t	buffering_length	message array length less than the block long
uint8_t	current_hash[HW_SCE_SHA256_HASH_LENGTH_BYTE_SIZE]	last hash value
uint8_t	flag_call_init	control flag of calling function

◆ sce_hmac_sha_handle_t

struct sce_hmac_sha_handle_t		
The work area for HMAC-SHA. DO NOT MODIFY.		
Data Fields		
uint32_t	id	serial number of this handle
sce_hmac_sha_wrapped_key_t	wrapped_key	wrapped key
uint8_t	hmac_buffer[HW_SCE_SHA256_HASH_LENGTH_BYTE_SIZE *4]	message array length less than the block long
uint32_t	all_received_length	entire length of message
uint32_t	buffering_length	message array length less than the block long
uint8_t	flag_call_init	control flag of calling function

◆ sce_ecdh_handle_t

struct sce_ecdh_handle_t		
The work area for ECDH		
Data Fields		
uint32_t	id	serial number of this handle
uint32_t	flag_use_key_id	control frag that the key_id has already used or not
uint32_t	key_id	serial number of the wrapped key
uint32_t	key_type	key type
uint8_t	flag_call_init	control flag of calling function
uint8_t	flag_call_make_public	control flag of calling function
uint8_t	flag_call_read_public	control flag of calling function
uint8_t	flag_call_shared_secret	control flag of calling function

◆ sce_cfg_t

struct sce_cfg_t		
User configuration structure, used in open function		
Data Fields		
lifecycle_t	lifecycle	Data lifecycle.

◆ sce_api_t

struct sce_api_t
Functions implemented at the HAL layer will follow this API.

Data Fields

fsp_err_t(*	open)(sce_ctrl_t *const p_ctrl, sce_cfg_t const *const p_cfg)
fsp_err_t(*	close)(sce_ctrl_t *const p_ctrl)
fsp_err_t(*	softwareReset)(void)
fsp_err_t(*	randomNumberGenerate)(uint32_t *random)
fsp_err_t(*	AES128_WrappedKeyGenerate)(sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	AES256_WrappedKeyGenerate)(sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	AES128_EncryptedKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	AES256_EncryptedKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	AES128ECB_EncryptInit)(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	AES128ECB_EncryptUpdate)(sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)
fsp_err_t(*	AES128ECB_EncryptFinal)(sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)
fsp_err_t(*	AES128ECB_DecryptInit)(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	AES128ECB_DecryptUpdate)(sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

fsp_err_t(*	AES128ECB_DecryptFinal)(sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)
fsp_err_t(*	AES256ECB_EncryptInit)(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	AES256ECB_EncryptUpdate)(sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)
fsp_err_t(*	AES256ECB_EncryptFinal)(sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)
fsp_err_t(*	AES256ECB_DecryptInit)(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	AES256ECB_DecryptUpdate)(sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)
fsp_err_t(*	AES256ECB_DecryptFinal)(sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)
fsp_err_t(*	AES128CBC_EncryptInit)(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)
fsp_err_t(*	AES128CBC_EncryptUpdate)(sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)
fsp_err_t(*	AES128CBC_EncryptFinal)(sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)
fsp_err_t(*	AES128CBC_DecryptInit)(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)
fsp_err_t(*	AES128CBC_DecryptUpdate)(sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)
fsp_err_t(*	AES128CBC_DecryptFinal)(sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)

fsp_err_t(*	AES256CBC_EncryptInit)(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)
fsp_err_t(*	AES256CBC_EncryptUpdate)(sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)
fsp_err_t(*	AES256CBC_EncryptFinal)(sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)
fsp_err_t(*	AES256CBC_DecryptInit)(sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)
fsp_err_t(*	AES256CBC_DecryptUpdate)(sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)
fsp_err_t(*	AES256CBC_DecryptFinal)(sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)
fsp_err_t(*	AES128GCM_EncryptInit)(sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)
fsp_err_t(*	AES128GCM_EncryptUpdate)(sce_gcm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_data_length, uint8_t *aad, uint32_t aad_length)
fsp_err_t(*	AES128GCM_EncryptFinal)(sce_gcm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_data_length, uint8_t *atag)
fsp_err_t(*	AES128GCM_DecryptInit)(sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)
fsp_err_t(*	AES128GCM_DecryptUpdate)(sce_gcm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_data_length, uint8_t *aad, uint32_t aad_length)
fsp_err_t(*	AES128GCM_DecryptFinal)(sce_gcm_handle_t *handle, uint8_t *plain, uint32_t *plain_data_length, uint8_t *atag, uint32_t atag_length)

fsp_err_t(*	AES256GCM_EncryptInit)(sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)
fsp_err_t(*	AES256GCM_EncryptUpdate)(sce_gcm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_data_length, uint8_t *aad, uint32_t aad_length)
fsp_err_t(*	AES256GCM_EncryptFinal)(sce_gcm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_data_length, uint8_t *atag)
fsp_err_t(*	AES256GCM_DecryptInit)(sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)
fsp_err_t(*	AES256GCM_DecryptUpdate)(sce_gcm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_data_length, uint8_t *aad, uint32_t aad_length)
fsp_err_t(*	AES256GCM_DecryptFinal)(sce_gcm_handle_t *handle, uint8_t *plain, uint32_t *plain_data_length, uint8_t *atag, uint32_t atag_length)
fsp_err_t(*	AES128CCM_EncryptInit)(sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length)
fsp_err_t(*	AES128CCM_EncryptUpdate)(sce_ccm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)
fsp_err_t(*	AES128CCM_EncryptFinal)(sce_ccm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length, uint8_t *mac, uint32_t mac_length)
fsp_err_t(*	AES128CCM_DecryptInit)(sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length)
fsp_err_t(*	AES128CCM_DecryptUpdate)(sce_ccm_handle_t *handle, uint8_t

	*cipher, uint8_t *plain, uint32_t cipher_length)
fsp_err_t(*	AES128CCM_DecryptFinal)(sce_ccm_handle_t *handle, uint8_t *plain, uint32_t *plain_length, uint8_t *mac, uint32_t mac_length)
fsp_err_t(*	AES256CCM_EncryptInit)(sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length)
fsp_err_t(*	AES256CCM_EncryptUpdate)(sce_ccm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)
fsp_err_t(*	AES256CCM_EncryptFinal)(sce_ccm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length, uint8_t *mac, uint32_t mac_length)
fsp_err_t(*	AES256CCM_DecryptInit)(sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length)
fsp_err_t(*	AES256CCM_DecryptUpdate)(sce_ccm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)
fsp_err_t(*	AES256CCM_DecryptFinal)(sce_ccm_handle_t *handle, uint8_t *plain, uint32_t *plain_length, uint8_t *mac, uint32_t mac_length)
fsp_err_t(*	AES128CMAC_GenerateInit)(sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	AES128CMAC_GenerateUpdate)(sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)
fsp_err_t(*	AES128CMAC_GenerateFinal)(sce_cmac_handle_t *handle, uint8_t *mac)
fsp_err_t(*	AES128CMAC_VerifyInit)(sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	AES128CMAC_VerifyUpdate)(sce_cmac_handle_t *handle, uint8_t

	*message, uint32_t message_length)
fsp_err_t(*	AES128CMAC_VerifyFinal)(sce_cmac_handle_t *handle, uint8_t *mac, uint32_t mac_length)
fsp_err_t(*	AES256CMAC_GenerateInit)(sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	AES256CMAC_GenerateUpdate)(sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)
fsp_err_t(*	AES256CMAC_GenerateFinal)(sce_cmac_handle_t *handle, uint8_t *mac)
fsp_err_t(*	AES256CMAC_VerifyInit)(sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	AES256CMAC_VerifyUpdate)(sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)
fsp_err_t(*	AES256CMAC_VerifyFinal)(sce_cmac_handle_t *handle, uint8_t *mac, uint32_t mac_length)
fsp_err_t(*	SHA256_Init)(sce_sha_md5_handle_t *handle)
fsp_err_t(*	SHA256_Update)(sce_sha_md5_handle_t *handle, uint8_t *message, uint32_t message_length)
fsp_err_t(*	SHA256_Final)(sce_sha_md5_handle_t *handle, uint8_t *digest, uint32_t *digest_length)
fsp_err_t(*	RSA1024_WrappedKeyPairGenerate)(sce_rsa1024_wrapped_pair_key_t *wrapped_pair_key)
fsp_err_t(*	RSA2048_WrappedKeyPairGenerate)(sce_rsa2048_wrapped_pair_key_t *wrapped_pair_key)
fsp_err_t(*	RSA1024_EncryptedPublicKeyWrap)(uint8_t *initial_vector, uint8_t

	*encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa1024_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	RSA1024_EncryptedPrivateKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa1024_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	RSA2048_EncryptedPublicKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa2048_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	RSA2048_EncryptedPrivateKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa2048_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	RSA3072_EncryptedPublicKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa3072_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	RSA4096_EncryptedPublicKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa4096_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	RSASSA_PKCS1024_SignatureGenerate)(sce_rsa_byte_data_t *message_hash, sce_rsa_byte_data_t *signature, sce_rsa1024_private_wrapped_key_t *wrapped_key, uint8_t hash_type)
fsp_err_t(*	RSASSA_PKCS2048_SignatureGenerate)(sce_rsa_byte_data_t *message_hash, sce_rsa_byte_data_t *signature, sce_rsa2048_private_wrapped_key_t *wrapped_key, uint8_t hash_type)
fsp_err_t(*	RSASSA_PKCS1024_SignatureVerify)(sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa1024_public_wrapped_key_t *wrapped_key, uint8_t hash_type)
fsp_err_t(*	RSASSA_PKCS2048_SignatureVerify)(sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa2048_public_wrapped_key_t *wrapped_key, uint8_t hash_type)

fsp_err_t(*	RSASSA_PKCS3072_SignatureVerify)(sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa3072_public_wrapped_key_t *wrapped_key, uint8_t hash_type)
fsp_err_t(*	RSASSA_PKCS4096_SignatureVerify)(sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa4096_public_wrapped_key_t *wrapped_key, uint8_t hash_type)
fsp_err_t(*	RSAES_PKCS1024_Encrypt)(sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa1024_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	RSAES_PKCS2048_Encrypt)(sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa2048_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	RSAES_PKCS3072_Encrypt)(sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa3072_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	RSAES_PKCS4096_Encrypt)(sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa4096_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	RSAES_PKCS1024_Decrypt)(sce_rsa_byte_data_t *cipher, sce_rsa_byte_data_t *plain, sce_rsa1024_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	RSAES_PKCS2048_Decrypt)(sce_rsa_byte_data_t *cipher, sce_rsa_byte_data_t *plain, sce_rsa2048_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	SHA256HMAC_EncryptedKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_hmac_sha_wrapped_key_t *wrapped_key)
fsp_err_t(*	SHA256HMAC_GenerateInit)(sce_hmac_sha_handle_t *handle, sce_hmac_sha_wrapped_key_t *wrapped_key)
fsp_err_t(*	SHA256HMAC_GenerateUpdate)(sce_hmac_sha_handle_t *handle,

	uint8_t *message, uint32_t message_length)
fsp_err_t(*	SHA256HMAC_GenerateFinal)(sce_hmac_sha_handle_t *handle, uint8_t *mac)
fsp_err_t(*	SHA256HMAC_VerifyInit)(sce_hmac_sha_handle_t *handle, sce_hmac_sha_wrapped_key_t *wrapped_key)
fsp_err_t(*	SHA256HMAC_VerifyUpdate)(sce_hmac_sha_handle_t *handle, uint8_t *message, uint32_t message_length)
fsp_err_t(*	SHA256HMAC_VerifyFinal)(sce_hmac_sha_handle_t *handle, uint8_t *mac, uint32_t mac_length)
fsp_err_t(*	ECC_secp192r1_WrappedKeyPairGenerate)(sce_ecc_wrapped_pair_key_t *wrapped_pair_key)
fsp_err_t(*	ECC_secp224r1_WrappedKeyPairGenerate)(sce_ecc_wrapped_pair_key_t *wrapped_pair_key)
fsp_err_t(*	ECC_secp256r1_WrappedKeyPairGenerate)(sce_ecc_wrapped_pair_key_t *wrapped_pair_key)
fsp_err_t(*	ECC_secp384r1_WrappedKeyPairGenerate)(sce_ecc_wrapped_pair_key_t *wrapped_pair_key)
fsp_err_t(*	ECC_secp192r1_EncryptedPublicKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECC_secp224r1_EncryptedPublicKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECC_secp256r1_EncryptedPublicKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECC_secp384r1_EncryptedPublicKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key,

	sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECC_secp192r1_EncryptedPrivateKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECC_secp224r1_EncryptedPrivateKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECC_secp256r1_EncryptedPrivateKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECC_secp384r1_EncryptedPrivateKeyWrap)(uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECDSA_secp192r1_SignatureGenerate)(sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECDSA_secp224r1_SignatureGenerate)(sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECDSA_secp256r1_SignatureGenerate)(sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECDSA_secp384r1_SignatureGenerate)(sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECDSA_secp192r1_SignatureVerify)(sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECDSA_secp224r1_SignatureVerify)(sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)

fsp_err_t(*	ECDSA_secp256r1_SignatureVerify)(sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECDSA_secp384r1_SignatureVerify)(sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECDH_secp256r1_Init)(sce_ecdh_handle_t *handle, uint32_t key_type, uint32_t use_key_id)
fsp_err_t(*	ECDH_secp256r1_PublicKeySign)(sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, sce_ecc_private_wrapped_key_t *ecc_private_wrapped_key, uint8_t *public_key, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECDH_secp256r1_PublicKeyVerify)(sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, uint8_t *public_key_data, sce_ecdsa_byte_data_t *signature, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECDH_secp256r1_PublicKeyReadWithoutSignature)(sce_ecdh_handle_t *handle, uint8_t *public_key_data, sce_ecc_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	ECDH_secp256r1_SharedSecretCalculate)(sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, sce_ecc_private_wrapped_key_t *ecc_private_wrapped_key, sce_ecdh_wrapped_key_t *shared_secret_wrapped_key)
fsp_err_t(*	ECDH_secp256r1_KeyDerivation)(sce_ecdh_handle_t *handle, sce_ecdh_wrapped_key_t *shared_secret_wrapped_key, uint32_t key_type, uint32_t kdf_type, uint8_t *other_info, uint32_t other_info_length, sce_hmac_sha_wrapped_key_t *salt_wrapped_key, sce_aes_wrapped_key_t *wrapped_key)
fsp_err_t(*	TLS_RootCertificateRSA2048PublicKeyInstall)(uint8_t *encrypted_provisioning_key, uint8_t *initial_vector, uint8_t *encrypted_key, sce_tls_ca_certification_public_wrapped_key_t *wrapped_key)
fsp_err_t(*	TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate

	)(sce_tls_p256_ecc_wrapped_key_t *tls_p256_ecc_wrapped_key, uint8_t *ephemeral_ecdh_public_key)
fsp_err_t(*	TLS_RootCertificateVerify)(uint32_t public_key_type, uint8_t *certificate, uint32_t certificate_length, uint32_t public_key_n_start_position, uint32_t public_key_n_end_position, uint32_t public_key_e_start_position, uint32_t public_key_e_end_position, uint8_t *signature, uint32_t *encrypted_root_public_key)
fsp_err_t(*	TLS_CertificateVerify)(uint32_t public_key_type, uint32_t *encrypted_input_public_key, uint8_t *certificate, uint32_t certificate_length, uint8_t *signature, uint32_t public_key_n_start_position, uint32_t public_key_n_end_position, uint32_t public_key_e_start_position, uint32_t public_key_e_end_position, uint32_t *encrypted_output_public_key)
fsp_err_t(*	TLS_PreMasterSecretGenerateForRSA2048)(uint32_t *sce_pre_master_secret)
fsp_err_t(*	TLS_MasterSecretGenerate)(uint32_t select_cipher_suite, uint32_t *sce_pre_master_secret, uint8_t *client_random, uint8_t *server_random, uint32_t *sce_master_secret)
fsp_err_t(*	TLS_PreMasterSecretEncryptWithRSA2048)(uint32_t *encrypted_public_key, uint32_t *sce_pre_master_secret, uint8_t *encrypted_pre_master_secret)
fsp_err_t(*	TLS_SessionKeyGenerate)(uint32_t select_cipher_suite, uint32_t *sce_master_secret, uint8_t *client_random, uint8_t *server_random, uint8_t *nonce_explicit, sce_hmac_sha_wrapped_key_t *client_mac_wrapped_key, sce_hmac_sha_wrapped_key_t *server_mac_wrapped_key, sce_aes_wrapped_key_t *client_crypto_wrapped_key, sce_aes_wrapped_key_t *server_crypto_wrapped_key, uint8_t *client_initial_vector, uint8_t *server_initial_vector)
fsp_err_t(*	TLS_VerifyDataGenerate)(uint32_t select_verify_data, uint32_t *sce_master_secret, uint8_t *hand_shake_hash, uint8_t *verify_data)
fsp_err_t(*	TLS_ServerKeyExchangeVerify)(uint32_t public_key_type, uint8_t *client_random, uint8_t *server_random, uint8_t *server_ephemeral_ecdh_public_key, uint8_t *server_key_exchange_signature, uint32_t *encrypted_public_key, uint32_t *encrypted_ephemeral_ecdh_public_key)

fsp_err_t(*	TLS_PreMasterSecretGenerateForECC_secp256r1)(uint32_t *encrypted_public_key, sce_tls_p256_ecc_wrapped_key_t *tls_p256_ecc_wrapped_key, uint32_t *sce_pre_master_secret)

Field Documentation

◆ open

fsp_err_t(* sce_api_t::open) (sce_ctrl_t *const p_ctrl, sce_cfg_t const *const p_cfg)

Enables use of SCE functionality.

Implemented as

- R_SCE_Open()

Parameters

[in]	p_ctrl	Pointer to control structure.
[in]	p_cfg	Pointer to pin configuration structure.

◆ close

fsp_err_t(* sce_api_t::close) (sce_ctrl_t *const p_ctrl)

Stops supply of power to the SCE.

Implemented as

- R_SCE_Close()

Parameters

[in]	p_ctrl	Pointer to control structure.

◆ softwareReset

fsp_err_t(* sce_api_t::softwareReset) (void)

Software reset to SCE.

Implemented as

- R_SCE_SoftwareReset()

◆ randomNumberGenerate

fsp_err_t(* sce_api_t::randomNumberGenerate) (uint32_t *random)

Generates 4 words random number.

Implemented as

- R_SCE_RandomNumberGenerate()

Parameters

[in,out]	random	Stores 4words (16 bytes) random data.

◆ AES128_WrappedKeyGenerate

fsp_err_t(* sce_api_t::AES128_WrappedKeyGenerate) (sce_aes_wrapped_key_t *wrapped_key)

This API outputs 128-bit AES wrapped key.

Implemented as

- R_SCE_AES128_WrappedKeyGenerate()

Parameters

[in,out]	wrapped_key	128-bit AES wrapped key

◆ AES256_WrappedKeyGenerate

fsp_err_t(* sce_api_t::AES256_WrappedKeyGenerate) (sce_aes_wrapped_key_t *wrapped_key)

This API outputs 256-bit AES wrapped key.

Implemented as

- R_SCE_AES256_WrappedKeyGenerate()

Parameters

[in,out]	wrapped_key	256-bit AES wrapped key

◆ AES128_EncryptedKeyWrap

fsp_err_t(* sce_api_t::AES128_EncryptedKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_aes_wrapped_key_t *wrapped_key)

This API outputs 128-bit AES wrapped key.

Implemented as

- R_SCE_AES128_EncryptedKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	128-bit AES wrapped key

◆ AES256_EncryptedKeyWrap

fsp_err_t(* sce_api_t::AES256_EncryptedKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_aes_wrapped_key_t *wrapped_key)

This API outputs 256-bit AES wrapped key.

Implemented as

- R_SCE_AES256_EncryptedKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	256-bit AES wrapped key

◆ AES128ECB_EncryptInit

fsp_err_t(* sce_api_t::AES128ECB_EncryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES128ECB encryption.

Implemented as

- R_SCE_AES128ECB_EncryptInit()

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	128-bit AES wrapped key

◆ AES128ECB_EncryptUpdate

fsp_err_t(* sce_api_t::AES128ECB_EncryptUpdate) (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

Update AES128ECB encryption.

Implemented as

- R_SCE_AES128ECB_EncryptUpdate()

Parameters

[in,out]	handle	AES handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in,out]	plain_length	plaintext data length (must be a multiple of 16)

◆ AES128ECB_EncryptFinal

fsp_err_t(* sce_api_t::AES128ECB_EncryptFinal) (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)

Finalize AES128ECB encryption.

Implemented as

- R_SCE_AES128ECB_EncryptFinal()

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	cipher	ciphertext data area (nothing ever written here)
[in,out]	cipher_length	ciphertext data length (0 always written here)

◆ AES128ECB_DecryptInit

fsp_err_t(* sce_api_t::AES128ECB_DecryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES128ECB decryption.

Implemented as

- R_SCE_AES128ECB_DecryptInit()

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	128-bit AES wrapped key

◆ AES128ECB_DecryptUpdate

fsp_err_t(* sce_api_t::AES128ECB_DecryptUpdate) (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

Update AES128ECB decryption.

Implemented as

- R_SCE_AES128ECB_DecryptUpdate()

Parameters

[in,out]	handle	AES handler (work area)
[in]	cipher	ciphertext data area
[in,out]	plain	plaintext data area
[in,out]	cipher_length	ciphertext data length (must be a multiple of 16)

◆ AES128ECB_DecryptFinal

fsp_err_t(* sce_api_t::AES128ECB_DecryptFinal) (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)

Finalize AES128ECB decryption.

Implemented as

- R_SCE_AES128ECB_DecryptFinal()

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	plain	plaintext data area (nothing ever written here)
[in,out]	plain_length	plaintext data length (0 always written here)

◆ AES256ECB_EncryptInit

fsp_err_t(* sce_api_t::AES256ECB_EncryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES256ECB encryption.

Implemented as

- R_SCE_AES256ECB_EncryptInit()

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	256-bit AES wrapped key

◆ AES256ECB_EncryptUpdate

fsp_err_t(* sce_api_t::AES256ECB_EncryptUpdate) (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

Update AES256ECB encryption.

Implemented as

- R_SCE_AES256ECB_EncryptUpdate()

Parameters

[in,out]	handle	AES handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in,out]	plain_length	plaintext data length (must be a multiple of 16)

◆ AES256ECB_EncryptFinal

fsp_err_t(* sce_api_t::AES256ECB_EncryptFinal) (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)

Finalize AES256ECB encryption.

Implemented as

- R_SCE_AES256ECB_EncryptFinal()

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	cipher	ciphertext data area (nothing ever written here)
[in,out]	cipher_length	ciphertext data length (0 always written here)

◆ AES256ECB_DecryptInit

fsp_err_t(* sce_api_t::AES256ECB_DecryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES256ECB decryption.

Implemented as

- R_SCE_AES256ECB_DecryptInit()

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	256-bit AES wrapped key

◆ AES256ECB_DecryptUpdate

fsp_err_t(* sce_api_t::AES256ECB_DecryptUpdate) (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

Update AES256ECB decryption.

Implemented as

- R_SCE_AES256ECB_DecryptUpdate()

Parameters

[in,out]	handle	AES handler (work area)
[in]	cipher	ciphertext data area
[in,out]	plain	plaintext data area
[in,out]	cipher_length	ciphertext data length (must be a multiple of 16)

◆ AES256ECB_DecryptFinal

fsp_err_t(* sce_api_t::AES256ECB_DecryptFinal) (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)

Finalize AES256ECB decryption.

Implemented as

- R_SCE_AES256ECB_DecryptFinal()

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	plain	plaintext data area (nothing ever written here)
[in,out]	plain_length	plaintext data length (0 always written here)

◆ AES128CBC_EncryptInit

fsp_err_t(* sce_api_t::AES128CBC_EncryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)

Initialize AES128CBC encryption.

Implemented as

- R_SCE_AES128CBC_EncryptInit()

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	128-bit AES wrapped key
[in]	initial_vector	initial vector area (16byte)

◆ AES128CBC_EncryptUpdate

fsp_err_t(* sce_api_t::AES128CBC_EncryptUpdate) (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

Update AES128CBC encryption.

Implemented as

- R_SCE_AES128CBC_EncryptUpdate()

Parameters

[in,out]	handle	AES handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in,out]	plain_length	plaintext data length (must be a multiple of 16)

◆ AES128CBC_EncryptFinal

fsp_err_t(* sce_api_t::AES128CBC_EncryptFinal) (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)

Finalize AES128CBC encryption.

Implemented as

- R_SCE_AES128CBC_EncryptFinal()

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	cipher	ciphertext data area (nothing ever written here)
[in,out]	cipher_length	ciphertext data length (0 always written here)

◆ AES128CBC_DecryptInit

fsp_err_t(* sce_api_t::AES128CBC_DecryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)

Initialize AES128CBC decryption.

Implemented as

- R_SCE_AES128CBC_DecryptInit()

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	128-bit AES wrapped key
[in]	initial_vector	initial vector area (16byte)

◆ AES128CBC_DecryptUpdate

fsp_err_t(* sce_api_t::AES128CBC_DecryptUpdate) (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

Update AES128CBC decryption.

Implemented as

- R_SCE_AES128CBC_DecryptUpdate()

Parameters

[in,out]	handle	AES handler (work area)
[in]	cipher	ciphertext data area
[in,out]	plain	plaintext data area
[in,out]	cipher_length	ciphertext data length (must be a multiple of 16)

◆ AES128CBC_DecryptFinal

fsp_err_t(* sce_api_t::AES128CBC_DecryptFinal) (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)

Finalize AES128CBC decryption.

Implemented as

- R_SCE_AES128CBC_DecryptFinal()

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	plain	plaintext data area (nothing ever written here)
[in,out]	plain_length	plaintext data length (0 always written here)

◆ AES256CBC_EncryptInit

fsp_err_t(* sce_api_t::AES256CBC_EncryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)

Initialize AES256CBC encryption.

Implemented as

- R_SCE_AES256CBC_EncryptInit()

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	256-bit AES wrapped key
[in]	initial_vector	initial vector area (16byte)

◆ AES256CBC_EncryptUpdate

fsp_err_t(* sce_api_t::AES256CBC_EncryptUpdate) (sce_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

Update AES256CBC encryption.

Implemented as

- R_SCE_AES256CBC_EncryptUpdate()

Parameters

[in,out]	handle	AES handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in,out]	plain_length	plaintext data length (must be a multiple of 16)

◆ AES256CBC_EncryptFinal

fsp_err_t(* sce_api_t::AES256CBC_EncryptFinal) (sce_aes_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length)

Finalize AES256CBC encryption.

Implemented as

- R_SCE_AES256CBC_EncryptFinal()

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	cipher	ciphertext data area (nothing ever written here)
[in,out]	cipher_length	ciphertext data length (0 always written here)

◆ AES256CBC_DecryptInit

fsp_err_t(* sce_api_t::AES256CBC_DecryptInit) (sce_aes_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector)

Initialize AES256CBC decryption.

Implemented as

- R_SCE_AES256CBC_DecryptInit()

Parameters

[in,out]	handle	AES handler (work area)
[in]	wrapped_key	256-bit AES wrapped key
[in]	initial_vector	initial vector area (16byte)

◆ AES256CBC_DecryptUpdate

fsp_err_t(* sce_api_t::AES256CBC_DecryptUpdate) (sce_aes_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

Update AES256CBC decryption.

Implemented as

- R_SCE_AES256CBC_DecryptUpdate()

Parameters

[in,out]	handle	AES handler (work area)
[in]	cipher	ciphertext data area
[in,out]	plain	plaintext data area
[in,out]	cipher_length	ciphertext data length (must be a multiple of 16)

◆ AES256CBC_DecryptFinal

fsp_err_t(* sce_api_t::AES256CBC_DecryptFinal) (sce_aes_handle_t *handle, uint8_t *plain, uint32_t *plain_length)

Finalize AES256CBC decryption.

Implemented as

- R_SCE_AES256CBC_DecryptFinal()

Parameters

[in,out]	handle	AES handler (work area)
[in,out]	plain	plaintext data area (nothing ever written here)
[in,out]	plain_length	plaintext data length (0 always written here)

◆ AES128GCM_EncryptInit

fsp_err_t(* sce_api_t::AES128GCM_EncryptInit) (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)

Initialize AES128GCM encryption.

Implemented as

- R_SCE_AES128GCM_EncryptInit()

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in]	wrapped_key	128-bit AES wrapped key
[in]	initial_vector	initialization vector area (initial_vector_length byte)
[in]	initial_vector_length	initialization vector length (1 ore more bytes)

◆ AES128GCM_EncryptUpdate

fsp_err_t(* sce_api_t::AES128GCM_EncryptUpdate) (sce_gcm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_data_length, uint8_t *aad, uint32_t aad_length)

Update AES128GCM encryption.

Implemented as

- R_SCE_AES128GCM_EncryptUpdate()

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in]	plain_data_length	plaintext data length (0 or more bytes)
[in]	aad	additional authentication data (aad_length byte)
[in]	aad_length	additional authentication data length (0 or more bytes)

◆ AES128GCM_EncryptFinal

fsp_err_t(* sce_api_t::AES128GCM_EncryptFinal) (sce_gcm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_data_length, uint8_t *atag)

Finalize AES128GCM encryption.

Implemented as

- R_SCE_AES128GCM_EncryptFinal()

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in,out]	cipher	ciphertext data area (cipher_data_length byte)
[in,out]	cipher_data_length	ciphertext data length (0 always written here)
[in,out]	atag	authentication tag area

◆ AES128GCM_DecryptInit

fsp_err_t(* sce_api_t::AES128GCM_DecryptInit) (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)

Initialize AES128GCM decryption.

Implemented as

- R_SCE_AES128GCM_DecryptInit()

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in]	wrapped_key	128-bit AES wrapped key
[in]	initial_vector	initialization vector area (initial_vector_length byte)
[in]	initial_vector_length	initialization vector length (1 ore more bytes)

◆ AES128GCM_DecryptUpdate

fsp_err_t(* sce_api_t::AES128GCM_DecryptUpdate) (sce_gcm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_data_length, uint8_t *aad, uint32_t aad_length)

Update AES128GCM decryption.

Implemented as

- R_SCE_AES128GCM_DecryptUpdate()

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in,out]	cipher	ciphertext data area
[in]	plain	plaintext data area
[in]	cipher_data_length	ciphertext data length (0 or more bytes)
[in]	aad	additional authentication data (aad_length byte)
[in]	aad_length	additional authentication data length (0 or more bytes)

◆ AES128GCM_DecryptFinal

fsp_err_t(* sce_api_t::AES128GCM_DecryptFinal) (sce_gcm_handle_t *handle, uint8_t *plain, uint32_t *plain_data_length, uint8_t *atag, uint32_t atag_length)

Finalize AES128GCM decryption.

Implemented as

- R_SCE_AES128GCM_DecryptFinal()

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in,out]	plain	plaintext data area (cipher_data_length byte)
[in,out]	plain_data_length	plaintext data length (0 always written here)
[in,out]	atag	authentication tag area (atag_length byte)
[in]	atag_length	authentication tag length (4,8,12,13,14,15,16 bytes)

◆ AES256GCM_EncryptInit

fsp_err_t(* sce_api_t::AES256GCM_EncryptInit) (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)

Initialize AES256GCM encryption.

Implemented as

- R_SCE_AES256GCM_EncryptInit()

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in]	wrapped_key	256-bit AES wrapped key
[in]	initial_vector	initialization vector area (initial_vector_length byte)
[in]	initial_vector_length	initialization vector length (1 ore more bytes)

◆ AES256GCM_EncryptUpdate

fsp_err_t(* sce_api_t::AES256GCM_EncryptUpdate) (sce_gcm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_data_length, uint8_t *aad, uint32_t aad_length)

Update AES256GCM encryption.

Implemented as

- R_SCE_AES256GCM_EncryptUpdate()

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in]	plain_data_length	plaintext data length (0 or more bytes)
[in]	aad	additional authentication data (aad_length byte)
[in]	aad_length	additional authentication data length (0 or more bytes)

◆ AES256GCM_EncryptFinal

fsp_err_t(* sce_api_t::AES256GCM_EncryptFinal) (sce_gcm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_data_length, uint8_t *atag)

Finalize AES256GCM encryption.

Implemented as

- R_SCE_AES256GCM_EncryptFinal()

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in,out]	cipher	ciphertext data area (cipher_data_length byte)
[in,out]	cipher_data_length	ciphertext data length (0 always written here)
[in,out]	atag	authentication tag area

◆ AES256GCM_DecryptInit

fsp_err_t(* sce_api_t::AES256GCM_DecryptInit) (sce_gcm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *initial_vector, uint32_t initial_vector_length)

Initialize AES256GCM decryption.

Implemented as

- R_SCE_AES256GCM_DecryptInit()

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in]	wrapped_key	256-bit AES wrapped key
[in]	initial_vector	initialization vector area (initial_vector_length byte)
[in]	initial_vector_length	initialization vector length (1 ore more bytes)

◆ AES256GCM_DecryptUpdate

fsp_err_t(* sce_api_t::AES256GCM_DecryptUpdate) (sce_gcm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_data_length, uint8_t *aad, uint32_t aad_length)

Update AES256GCM decryption.

Implemented as

- R_SCE_AES256GCM_DecryptUpdate()

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in,out]	cipher	ciphertext data area
[in]	plain	plaintext data area
[in]	cipher_data_length	ciphertext data length (0 or more bytes)
[in]	aad	additional authentication data (aad_length byte)
[in]	aad_length	additional authentication data length (0 or more bytes)

◆ AES256GCM_DecryptFinal

fsp_err_t(* sce_api_t::AES256GCM_DecryptFinal) (sce_gcm_handle_t *handle, uint8_t *plain, uint32_t *plain_data_length, uint8_t *atag, uint32_t atag_length)

Finalize AES256GCM decryption.

Implemented as

- R_SCE_AES256GCM_DecryptFinal()

Parameters

[in,out]	handle	AES-GCM handler (work area)
[in,out]	plain	plaintext data area (cipher_data_length byte)
[in,out]	plain_data_length	plaintext data length (0 always written here)
[in,out]	atag	authentication tag area (atag_length byte)
[in]	atag_length	authentication tag length (4,8,12,13,14,15,16 bytes)

◆ AES128CCM_EncryptInit

fsp_err_t(* sce_api_t::AES128CCM_EncryptInit) (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length)

Initialize AES128CCM encryption.

Implemented as

- R_SCE_AES128CCM_EncryptInit()

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	wrapped_key	128-bit AES wrapped key
[in]	nonce	Nonce
[in]	nonce_length	Nonce data length (7 to 13 bytes)
[in]	adata	additional authentication data
[in]	a_length	additional authentication data length (0 to 110 bytes)
[in]	payload_length	Payload length (any number of bytes)
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

◆ AES128CCM_EncryptUpdate

fsp_err_t(* sce_api_t::AES128CCM_EncryptUpdate) (sce_ccm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

Update AES128CCM encryption.

Implemented as

- R_SCE_AES128CCM_EncryptUpdate()

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in]	plain_length	plaintext data length

◆ AES128CCM_EncryptFinal

fsp_err_t(* sce_api_t::AES128CCM_EncryptFinal) (sce_ccm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length, uint8_t *mac, uint32_t mac_length)

Finalize AES128CCM encryption.

Implemented as

- R_SCE_AES128CCM_EncryptFinal()

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in,out]	cipher	ciphertext data area
[in,out]	cipher_length	ciphertext data length
[in,out]	mac	MAC area
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

◆ AES128CCM_DecryptInit

fsp_err_t(* sce_api_t::AES128CCM_DecryptInit) (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length)

Initialize AES128CCM decryption.

Implemented as

- R_SCE_AES128CCM_DecryptInit()

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	wrapped_key	128-bit AES wrapped key
[in]	nonce	Nonce
[in]	nonce_length	Nonce data length (7 to 13 bytes)
[in]	adata	additional authentication data
[in]	a_length	additional authentication data length (0 to 110 bytes)
[in]	payload_length	Payload length (any number of bytes)
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

◆ AES128CCM_DecryptUpdate

fsp_err_t(* sce_api_t::AES128CCM_DecryptUpdate) (sce_ccm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

Update AES128CCM decryption.

Implemented as

- R_SCE_AES128CCM_DecryptUpdate()

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	cipher	ciphertext data area
[in,out]	plain	plaintext data area
[in]	cipher_length	ciphertext data length

◆ AES128CCM_DecryptFinal

fsp_err_t(* sce_api_t::AES128CCM_DecryptFinal) (sce_ccm_handle_t *handle, uint8_t *plain, uint32_t *plain_length, uint8_t *mac, uint32_t mac_length)

Finalize AES128CCM decryption.

Implemented as

- R_SCE_AES128CCM_DecryptFinal()

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in,out]	plain	plaintext data area
[in,out]	plain_length	plaintext data length
[in]	mac	MAC area
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

◆ AES256CCM_EncryptInit

fsp_err_t(* sce_api_t::AES256CCM_EncryptInit) (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length)

Initialize AES256CCM encryption.

Implemented as

- R_SCE_AES256CCM_EncryptInit()

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	wrapped_key	256-bit AES wrapped key
[in]	nonce	Nonce
[in]	nonce_length	Nonce data length (7 to 13 bytes)
[in]	adata	additional authentication data
[in]	a_length	additional authentication data length (0 to 110 bytes)
[in]	payload_length	Payload length (any number of bytes)
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

◆ AES256CCM_EncryptUpdate

fsp_err_t(* sce_api_t::AES256CCM_EncryptUpdate) (sce_ccm_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length)

Update AES256CCM encryption.

Implemented as

- R_SCE_AES256CCM_EncryptUpdate()

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	plain	plaintext data area
[in,out]	cipher	ciphertext data area
[in]	plain_length	plaintext data length

◆ AES256CCM_EncryptFinal

fsp_err_t(* sce_api_t::AES256CCM_EncryptFinal) (sce_ccm_handle_t *handle, uint8_t *cipher, uint32_t *cipher_length, uint8_t *mac, uint32_t mac_length)

Finalize AES256CCM encryption.

Implemented as

- R_SCE_AES256CCM_EncryptFinal()

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in,out]	cipher	ciphertext data area
[in,out]	cipher_length	ciphertext data length
[in,out]	mac	MAC area
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

◆ AES256CCM_DecryptInit

fsp_err_t(* sce_api_t::AES256CCM_DecryptInit) (sce_ccm_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key, uint8_t *nonce, uint32_t nonce_length, uint8_t *adata, uint8_t a_length, uint32_t payload_length, uint32_t mac_length)

Initialize AES256CCM decryption.

Implemented as

- R_SCE_AES256CCM_DecryptInit()

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	wrapped_key	256-bit AES wrapped key
[in]	nonce	Nonce
[in]	nonce_length	Nonce data length (7 to 13 bytes)
[in]	adata	additional authentication data
[in]	a_length	additional authentication data length (0 to 110 bytes)
[in]	payload_length	Payload length (any number of bytes)
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

◆ AES256CCM_DecryptUpdate

fsp_err_t(* sce_api_t::AES256CCM_DecryptUpdate) (sce_ccm_handle_t *handle, uint8_t *cipher, uint8_t *plain, uint32_t cipher_length)

Update AES256CCM decryption.

Implemented as

- R_SCE_AES256CCM_DecryptUpdate()

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in]	cipher	ciphertext data area
[in,out]	plain	plaintext data area
[in]	cipher_length	ciphertext data length

◆ AES256CCM_DecryptFinal

fsp_err_t(* sce_api_t::AES256CCM_DecryptFinal) (sce_ccm_handle_t *handle, uint8_t *plain, uint32_t *plain_length, uint8_t *mac, uint32_t mac_length)

Finalize AES256CCM decryption.

Implemented as

- R_SCE_AES256CCM_DecryptFinal()

Parameters

[in,out]	handle	AES-CCM handler (work area)
[in,out]	plain	plaintext data area
[in,out]	plain_length	plaintext data length
[in]	mac	MAC area
[in]	mac_length	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

◆ AES128CMAC_GenerateInit

fsp_err_t(* sce_api_t::AES128CMAC_GenerateInit) (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES128CMAC generation.

Implemented as

- R_SCE_AES128CMAC_GenerateInit()

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	wrapped_key	128-bit AES wrapped key

◆ AES128CMAC_GenerateUpdate

fsp_err_t(* sce_api_t::AES128CMAC_GenerateUpdate) (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)

Update AES128CMAC generation.

Implemented as

- R_SCE_AES128CMAC_GenerateUpdate()

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	message	message data area (message_length byte)
[in]	message_length	message data length (0 or more bytes)

◆ AES128CMAC_GenerateFinal

fsp_err_t(* sce_api_t::AES128CMAC_GenerateFinal) (sce_cmac_handle_t *handle, uint8_t *mac)

Finalize AES128CMAC generation.

Implemented as

- R_SCE_AES128CMAC_GenerateFinal()

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in,out]	mac	MAC data area (16byte)

◆ AES128CMAC_VerifyInit

fsp_err_t(* sce_api_t::AES128CMAC_VerifyInit) (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES128CMAC verification.

Implemented as

- R_SCE_AES128CMAC_VerifyInit()

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	wrapped_key	128-bit AES wrapped key

◆ AES128CMAC_VerifyUpdate

fsp_err_t(* sce_api_t::AES128CMAC_VerifyUpdate) (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)

Update AES128CMAC verification.

Implemented as

- R_SCE_AES128CMAC_VerifyUpdate()

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	message	message data area (message_length byte)
[in]	message_length	message data length (0 or more bytes)

◆ AES128CMAC_VerifyFinal

fsp_err_t(* sce_api_t::AES128CMAC_VerifyFinal) (sce_cmac_handle_t *handle, uint8_t *mac, uint32_t mac_length)

Finalize AES128CMAC verification.

Implemented as

- R_SCE_AES128CMAC_VerifyFinal()

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in,out]	mac	MAC data area (mac_length byte)
[in,out]	mac_length	MAC data length (2 to 16 bytes)

◆ AES256CMAC_GenerateInit

fsp_err_t(* sce_api_t::AES256CMAC_GenerateInit) (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES256CMAC generation.

Implemented as

- R_SCE_AES256CMAC_GenerateInit()

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	wrapped_key	256-bit AES wrapped key

◆ AES256CMAC_GenerateUpdate

fsp_err_t(* sce_api_t::AES256CMAC_GenerateUpdate) (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)

Update AES256CMAC generation.

Implemented as

- R_SCE_AES256CMAC_GenerateUpdate()

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	message	message data area (message_length byte)
[in]	message_length	message data length (0 or more bytes)

◆ AES256CMAC_GenerateFinal

fsp_err_t(* sce_api_t::AES256CMAC_GenerateFinal) (sce_cmac_handle_t *handle, uint8_t *mac)

Finalize AES256CMAC generation.

Implemented as

- R_SCE_AES256CMAC_GenerateFinal()

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in,out]	mac	MAC data area (16byte)

◆ AES256CMAC_VerifyInit

fsp_err_t(* sce_api_t::AES256CMAC_VerifyInit) (sce_cmac_handle_t *handle, sce_aes_wrapped_key_t *wrapped_key)

Initialize AES256CMAC verification.

Implemented as

- R_SCE_AES256CMAC_VerifyInit()

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	wrapped_key	256-bit AES wrapped key

◆ AES256CMAC_VerifyUpdate

fsp_err_t(* sce_api_t::AES256CMAC_VerifyUpdate) (sce_cmac_handle_t *handle, uint8_t *message, uint32_t message_length)

Update AES256CMAC verification.

Implemented as

- R_SCE_AES256CMAC_VerifyUpdate()

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in]	message	message data area (message_length byte)
[in]	message_length	message data length (0 or more bytes)

◆ AES256CMAC_VerifyFinal

fsp_err_t(* sce_api_t::AES256CMAC_VerifyFinal) (sce_cmac_handle_t *handle, uint8_t *mac, uint32_t mac_length)

Finalize AES256CMAC verification.

Implemented as

- R_SCE_AES256CMAC_VerifyFinal()

Parameters

[in,out]	handle	AES-CMAC handler (work area)
[in,out]	mac	MAC data area (mac_length byte)
[in,out]	mac_length	MAC data length (2 to 16 bytes)

◆ SHA256_Init

fsp_err_t(* sce_api_t::SHA256_Init) (sce_sha_md5_handle_t *handle)

Initialize SHA-256 Calculation.

Implemented as

- R_SCE_SHA256_Init()

Parameters

[in,out]	handle	SHA handler (work area)

◆ SHA256_Update

fsp_err_t(* sce_api_t::SHA256_Update) (sce_sha_md5_handle_t *handle, uint8_t *message, uint32_t message_length)

Update SHA-256 Calculation.

Implemented as

- R_SCE_SHA256_Update()

Parameters

[in,out]	handle	SHA handler (work area)
[in]	message	message data area
[in]	message_length	message data length

◆ SHA256_Final

fsp_err_t(* sce_api_t::SHA256_Final) (sce_sha_md5_handle_t *handle, uint8_t *digest, uint32_t *digest_length)

Finalize SHA-256 Calculation.

Implemented as

- R_SCE_SHA256_Final()

Parameters

[in,out]	handle	SHA handler (work area)
[in,out]	digest	hasha data area
[in,out]	digest_length	hash data length (32bytes)

◆ RSA1024_WrappedKeyPairGenerate

fsp_err_t(* sce_api_t::RSA1024_WrappedKeyPairGenerate) (sce_rsa1024_wrapped_pair_key_t *wrapped_pair_key)

This API outputs 1024-bit RSA wrapped pair key.

Implemented as

- R_SCE_RSA1024_WrappedKeyPairGenerate()

Parameters

[in,out]	wrapped_key	128-bit AES wrapped key

◆ RSA2048_WrappedKeyPairGenerate

fsp_err_t(* sce_api_t::RSA2048_WrappedKeyPairGenerate) (sce_rsa2048_wrapped_pair_key_t *wrapped_pair_key)

This API outputs 2048-bit RSA wrapped pair key.

Implemented as

- R_SCE_RSA2048_WrappedKeyPairGenerate()

Parameters

[in,out]	wrapped_key	128-bit AES wrapped key

◆ RSA1024_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::RSA1024_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa1024_public_wrapped_key_t *wrapped_key)

This API outputs 1024-bit RSA public wrapped key.

Implemented as

- R_SCE_RSA1024_EncryptedPublicKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	1024-bit RSA public wrapped key

◆ RSA1024_EncryptedPrivateKeyWrap

fsp_err_t(* sce_api_t::RSA1024_EncryptedPrivateKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa1024_private_wrapped_key_t *wrapped_key)

This API outputs 1024-bit RSA private wrapped key.

Implemented as

- R_SCE_RSA1024_EncryptedPrivateKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	1024-bit RSA private wrapped key

◆ RSA2048_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::RSA2048_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa2048_public_wrapped_key_t *wrapped_key)

This API outputs 2048-bit RSA public wrapped key.

Implemented as

- R_SCE_RSA2048_EncryptedPublicKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	2048-bit RSA public wrapped key

◆ RSA2048_EncryptedPrivateKeyWrap

fsp_err_t(* sce_api_t::RSA2048_EncryptedPrivateKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa2048_private_wrapped_key_t *wrapped_key)

This API outputs 2048-bit RSA private wrapped key.

Implemented as

- R_SCE_RSA2048_EncryptedPrivateKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	2048-bit RSA private wrapped key

◆ RSA3072_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::RSA3072_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa3072_public_wrapped_key_t *wrapped_key)

This API outputs 3072-bit RSA public wrapped key.

Implemented as

- R_SCE_RSA3072_EncryptedPublicKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	3072-bit RSA public wrapped key

◆ RSA4096_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::RSA4096_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_rsa4096_public_wrapped_key_t *wrapped_key)

This API outputs 4096-bit RSA public wrapped key.

Implemented as

- R_SCE_RSA4096_EncryptedPublicKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	4096-bit RSA public wrapped key

◆ RSASSA_PKCS1024_SignatureGenerate

fsp_err_t(* sce_api_t::RSASSA_PKCS1024_SignatureGenerate) (sce_rsa_byte_data_t *message_hash, sce_rsa_byte_data_t *signature, sce_rsa1024_private_wrapped_key_t *wrapped_key, uint8_t hash_type)

RSASSA-PKCS1-V1_5 signature generation.

Implemented as

- R_SCE_RSASSA_PKCS1024_SignatureGenerate()

Parameters

[in]	message_hash	Message or hash value to which to attach signature
[in,out]	signature	Signature text storage destination information
[in]	wrapped_key	Inputs the 1024-bit RSA private wrapped key.
[in]	hash_type	Only HW_SCE_RSA_HASH_SHA256 is supported

◆ RSASSA_PKCS2048_SignatureGenerate

fsp_err_t(* sce_api_t::RSASSA_PKCS2048_SignatureGenerate) (sce_rsa_byte_data_t *message_hash, sce_rsa_byte_data_t *signature, sce_rsa2048_private_wrapped_key_t *wrapped_key, uint8_t hash_type)

RSASSA-PKCS1-V1_5 signature generation.

Implemented as

- R_SCE_RSASSA_PKCS2048_SignatureGenerate()

Parameters

[in]	message_hash	Message or hash value to which to attach signature
[in,out]	signature	Signature text storage destination information
[in]	wrapped_key	Inputs the 2048-bit RSA private wrapped key.
[in]	hash_type	Only HW_SCE_RSA_HASH_SHA256 is supported

◆ RSASSA_PKCS1024_SignatureVerify

fsp_err_t(* sce_api_t::RSASSA_PKCS1024_SignatureVerify) (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa1024_public_wrapped_key_t *wrapped_key, uint8_t hash_type)

RSASSA-PKCS1-V1_5 signature verification.

Implemented as

- R_SCE_RSASSA_PKCS1024_SignatureVerify()

Parameters

[in]	signature	Signature text information to verify
[in]	message_hash	Message text or hash value to verify
[in]	wrapped_key	Inputs the 1024-bit RSA public wrapped key.
[in]	hash_type	Only HW_SCE_RSA_HASH_SHA256 is supported

◆ RSASSA_PKCS2048_SignatureVerify

fsp_err_t(* sce_api_t::RSASSA_PKCS2048_SignatureVerify) (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa2048_public_wrapped_key_t *wrapped_key, uint8_t hash_type)

RSASSA-PKCS1-V1_5 signature verification.

Implemented as

- R_SCE_RSASSA_PKCS2048_SignatureVerify()

Parameters

[in]	signature	Signature text information to verify
[in]	message_hash	Message text or hash value to verify
[in]	wrapped_key	Inputs the 2048-bit RSA public wrapped key.
[in]	hash_type	Only HW_SCE_RSA_HASH_SHA256 is supported

◆ RSASSA_PKCS3072_SignatureVerify

fsp_err_t(* sce_api_t::RSASSA_PKCS3072_SignatureVerify) (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa3072_public_wrapped_key_t *wrapped_key, uint8_t hash_type)

RSASSA-PKCS1-V1_5 signature verification.

Implemented as

- R_SCE_RSASSA_PKCS3072_SignatureVerify()

Parameters

[in]	signature	Signature text information to verify
[in]	message_hash	Message text or hash value to verify
[in]	wrapped_key	Inputs the 3072-bit RSA public wrapped key.
[in]	hash_type	Only HW_SCE_RSA_HASH_SHA256 is supported

◆ RSASSA_PKCS4096_SignatureVerify

fsp_err_t(* sce_api_t::RSASSA_PKCS4096_SignatureVerify) (sce_rsa_byte_data_t *signature, sce_rsa_byte_data_t *message_hash, sce_rsa4096_public_wrapped_key_t *wrapped_key, uint8_t hash_type)

RSASSA-PKCS1-V1_5 signature verification.

Implemented as

- R_SCE_RSASSA_PKCS4096_SignatureVerify()

Parameters

[in]	signature	Signature text information to verify
[in]	message_hash	Message text or hash value to verify
[in]	wrapped_key	Inputs the 4096-bit RSA public wrapped key.
[in]	hash_type	Only HW_SCE_RSA_HASH_SHA256 is supported

◆ RSAES_PKCS1024_Encrypt

fsp_err_t(* sce_api_t::RSAES_PKCS1024_Encrypt) (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa1024_public_wrapped_key_t *wrapped_key)

RSAES-PKCS1-V1_5 encryption.

Implemented as

- R_SCE_RSAES_PKCS1024_Encrypt()

Parameters

[in]	plain	plaintext
[in,out]	cipher	ciphertext
[in]	wrapped_key	Inputs the 1024-bit RSA public wrapped key.

◆ RSAES_PKCS2048_Encrypt

fsp_err_t(* sce_api_t::RSAES_PKCS2048_Encrypt) (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa2048_public_wrapped_key_t *wrapped_key)

RSAES-PKCS1-V1_5 encryption.

Implemented as

- R_SCE_RSAES_PKCS2048_Encrypt()

Parameters

[in]	plain	plaintext
[in,out]	cipher	ciphertext
[in]	wrapped_key	Inputs the 2048-bit RSA public wrapped key.

◆ RSAES_PKCS3072_Encrypt

fsp_err_t(* sce_api_t::RSAES_PKCS3072_Encrypt) (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa3072_public_wrapped_key_t *wrapped_key)

RSAES-PKCS1-V1_5 encryption.

Implemented as

- R_SCE_RSAES_PKCS3072_Encrypt()

Parameters

[in]	plain	plaintext
[in,out]	cipher	ciphertext
[in]	wrapped_key	Inputs the 3072-bit RSA public wrapped key.

◆ RSAES_PKCS4096_Encrypt

fsp_err_t(* sce_api_t::RSAES_PKCS4096_Encrypt) (sce_rsa_byte_data_t *plain, sce_rsa_byte_data_t *cipher, sce_rsa4096_public_wrapped_key_t *wrapped_key)

RSAES-PKCS1-V1_5 encryption.

Implemented as

- R_SCE_RSAES_PKCS4096_Encrypt()

Parameters

[in]	plain	plaintext
[in,out]	cipher	ciphertext
[in]	wrapped_key	Inputs the 4096-bit RSA public wrapped key.

◆ RSAES_PKCS1024_Decrypt

fsp_err_t(* sce_api_t::RSAES_PKCS1024_Decrypt) (sce_rsa_byte_data_t *cipher, sce_rsa_byte_data_t *plain, sce_rsa1024_private_wrapped_key_t *wrapped_key)

RSAES-PKCS1-V1_5 decryption.

Implemented as

- R_SCE_RSAES_PKCS1024_Decrypt()

Parameters

[in]	cipher	ciphertext
[in,out]	plain	plaintext
[in]	wrapped_key	Inputs the 1024-bit RSA private wrapped key.

◆ RSAES_PKCS2048_Decrypt

fsp_err_t(* sce_api_t::RSAES_PKCS2048_Decrypt) (sce_rsa_byte_data_t *cipher, sce_rsa_byte_data_t *plain, sce_rsa2048_private_wrapped_key_t *wrapped_key)

RSAES-PKCS1-V1_5 decryption.

Implemented as

- R_SCE_RSAES_PKCS2048_Decrypt()

Parameters

[in]	cipher	ciphertext
[in,out]	plain	plaintext
[in]	wrapped_key	Inputs the 2048-bit RSA private wrapped key.

◆ SHA256HMAC_EncryptedKeyWrap

fsp_err_t(* sce_api_t::SHA256HMAC_EncryptedKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_hmac_sha_wrapped_key_t *wrapped_key)

This API outputs HMAC-SHA256 wrapped key.

Implemented as

- R_SCE_SHA256HMAC_EncryptedKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	HMAC-SHA256 wrapped key

◆ SHA256HMAC_GenerateInit

fsp_err_t(* sce_api_t::SHA256HMAC_GenerateInit) (sce_hmac_sha_handle_t *handle, sce_hmac_sha_wrapped_key_t *wrapped_key)

Initialize HMAC-SHA256 generation.

Implemented as

- R_SCE_SHA256HMAC_GenerateInit()

Parameters

[in,out]	handle	SHA-HMAC handler (work area)
[in]	wrapped_key	MAC wrapped key

◆ SHA256HMAC_GenerateUpdate

fsp_err_t(* sce_api_t::SHA256HMAC_GenerateUpdate) (sce_hmac_sha_handle_t *handle, uint8_t *message, uint32_t message_length)

Update HMAC-SHA256 generation.

Implemented as

- R_SCE_SHA256HMAC_GenerateUpdate()

Parameters

[in,out]	handle	SHA-HMAC handle (work area)
[in]	message	Message area
[in]	message_length	Message length

◆ SHA256HMAC_GenerateFinal

fsp_err_t(* sce_api_t::SHA256HMAC_GenerateFinal) (sce_hmac_sha_handle_t *handle, uint8_t *mac)

Finalize HMAC-SHA256 generation.

Implemented as

- R_SCE_SHA256HMAC_GenerateFinal()

Parameters

[in,out]	handle	SHA-HMAC handle (work area)
[in,out]	mac	HMAC area (32 bytes)

◆ SHA256HMAC_VerifyInit

fsp_err_t(* sce_api_t::SHA256HMAC_VerifyInit) (sce_hmac_sha_handle_t *handle, sce_hmac_sha_wrapped_key_t *wrapped_key)

Initialize HMAC-SHA256 verification.

Implemented as

- R_SCE_SHA256HMAC_VerifyInit()

Parameters

[in,out]	handle	SHA-HMAC handler (work area)
[in]	wrapped_key	MAC wrapped key

◆ SHA256HMAC_VerifyUpdate

fsp_err_t(* sce_api_t::SHA256HMAC_VerifyUpdate) (sce_hmac_sha_handle_t *handle, uint8_t *message, uint32_t message_length)

Update HMAC-SHA256 verification.

Implemented as

- R_SCE_SHA256HMAC_VerifyUpdate()

Parameters

[in,out]	handle	SHA-HMAC handle (work area)
[in]	message	Message area
[in]	message_length	Message length

◆ SHA256HMAC_VerifyFinal

fsp_err_t(* sce_api_t::SHA256HMAC_VerifyFinal) (sce_hmac_sha_handle_t *handle, uint8_t *mac, uint32_t mac_length)

Finalize HMAC-SHA256 verification.

Implemented as

- R_SCE_SHA256HMAC_VerifyFinal()

Parameters

[in,out]	handle	SHA-HMAC handle (work area)
[in]	mac	HMAC area
[in]	mac_length	HMAC length

◆ ECC_secp192r1_WrappedKeyPairGenerate

fsp_err_t(* sce_api_t::ECC_secp192r1_WrappedKeyPairGenerate) (sce_ecc_wrapped_pair_key_t *wrapped_pair_key)

This API outputs secp192r1 wrapped pair key.

Implemented as

- R_SCE_ECC_secp192r1_WrappedKeyPairGenerate()

Parameters

[in,out]	wrapped_pair_key	Wrapped pair key for secp192r1 public key and private key pair

◆ ECC_secp224r1_WrappedKeyPairGenerate

fsp_err_t(* sce_api_t::ECC_secp224r1_WrappedKeyPairGenerate) (sce_ecc_wrapped_pair_key_t *wrapped_pair_key)

This API outputs secp224r1 wrapped pair key.

Implemented as

- R_SCE_ECC_secp224r1_WrappedKeyPairGenerate()

Parameters

[in,out]	wrapped_pair_key	Wrapped pair key for secp224r1 public key and private key pair

◆ ECC_secp256r1_WrappedKeyPairGenerate

fsp_err_t(* sce_api_t::ECC_secp256r1_WrappedKeyPairGenerate) (sce_ecc_wrapped_pair_key_t *wrapped_pair_key)

This API outputs secp256r1 wrapped pair key.

Implemented as

- R_SCE_ECC_secp256r1_WrappedKeyPairGenerate()

Parameters

[in,out]	wrapped_pair_key	Wrapped pair key for secp256r1 public key and private key pair

◆ ECC_secp384r1_WrappedKeyPairGenerate

fsp_err_t(* sce_api_t::ECC_secp384r1_WrappedKeyPairGenerate) (sce_ecc_wrapped_pair_key_t *wrapped_pair_key)

This API outputs secp384r1 wrapped pair key.

Implemented as

- R_SCE_ECC_secp384r1_WrappedKeyPairGenerate()

Parameters

[in,out]	wrapped_pair_key	Wrapped pair key for secp384r1 public key and private key pair

◆ ECC_secp192r1_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::ECC_secp192r1_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)

This API outputs secp192r1 public wrapped key.

Implemented as

- R_SCE_ECC_secp192r1_EncryptedPublicKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp192r1 public wrapped key

◆ ECC_secp224r1_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::ECC_secp224r1_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)

This API outputs secp224r1 public wrapped key.

Implemented as

- R_SCE_ECC_secp224r1_EncryptedPublicKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp224r1 public wrapped key

◆ ECC_secp256r1_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::ECC_secp256r1_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)

This API outputs secp256r1 public wrapped key.

Implemented as

- R_SCE_ECC_secp256r1_EncryptedPublicKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp256r1 public wrapped key

◆ ECC_secp384r1_EncryptedPublicKeyWrap

fsp_err_t(* sce_api_t::ECC_secp384r1_EncryptedPublicKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_public_wrapped_key_t *wrapped_key)

This API outputs secp384r1 public wrapped key.

Implemented as

- R_SCE_ECC_secp384r1_EncryptedPublicKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp384r1 public wrapped key

◆ ECC_secp192r1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_api_t::ECC_secp192r1_EncryptedPrivateKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)

This API outputs secp192r1 private wrapped key.

Implemented as

- R_SCE_ECC_secp192r1_EncryptedPrivateKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp192r1 private wrapped key

◆ ECC_secp224r1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_api_t::ECC_secp224r1_EncryptedPrivateKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)

This API outputs secp224r1 private wrapped key.

Implemented as

- R_SCE_ECC_secp224r1_EncryptedPrivateKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp224r1 private wrapped key

◆ ECC_secp256r1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_api_t::ECC_secp256r1_EncryptedPrivateKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)

This API outputs secp256r1 private wrapped key.

Implemented as

- R_SCE_ECC_secp256r1_EncryptedPrivateKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp256r1 private wrapped key

◆ ECC_secp384r1_EncryptedPrivateKeyWrap

fsp_err_t(* sce_api_t::ECC_secp384r1_EncryptedPrivateKeyWrap) (uint8_t *initial_vector, uint8_t *encrypted_key, sce_key_update_key_t *key_update_key, sce_ecc_private_wrapped_key_t *wrapped_key)

This API outputs secp384r1 private wrapped key.

Implemented as

- R_SCE_ECC_secp384r1_EncryptedPrivateKeyWrap()

Parameters

[in]	initial_vector	Initialization vector when generating encrypted_key
[in]	encrypted_key	User key encryptedand MAC appended
[in]	key_update_key	Key update keyring
[in,out]	wrapped_key	secp384r1 private wrapped key

◆ ECDSA_secp192r1_SignatureGenerate

fsp_err_t(* sce_api_t::ECDSA_secp192r1_SignatureGenerate) (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)

ECDSA signature generation.

Implemented as

- R_SCE_ECDSA_secp192r1_SignatureGenerate()

Parameters

[in]	message_hash	Message or hash value to which to attach signature
[in,out]	signature	Signature text storage destination information
[in]	wrapped_key	Input wrapped key of secp192r1 private key.

◆ ECDSA_secp224r1_SignatureGenerate

fsp_err_t(* sce_api_t::ECDSA_secp224r1_SignatureGenerate) (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)

ECDSA signature generation.

Implemented as

- R_SCE_ECDSA_secp224r1_SignatureGenerate()

Parameters

[in]	message_hash	Message or hash value to which to attach signature
[in,out]	signature	Signature text storage destination information
[in]	wrapped_key	Input wrapped key of secp224r1 private key.

◆ ECDSA_secp256r1_SignatureGenerate

fsp_err_t(* sce_api_t::ECDSA_secp256r1_SignatureGenerate) (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)

ECDSA signature generation.

Implemented as

- R_SCE_ECDSA_secp256r1_SignatureGenerate()

Parameters

[in]	message_hash	Message or hash value to which to attach signature
[in,out]	signature	Signature text storage destination information
[in]	wrapped_key	Input wrapped key of secp256r1 private key.

◆ ECDSA_secp384r1_SignatureGenerate

fsp_err_t(* sce_api_t::ECDSA_secp384r1_SignatureGenerate) (sce_ecdsa_byte_data_t *message_hash, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)

ECDSA signature generation.

Implemented as

- R_SCE_ECDSA_secp384r1_SignatureGenerate()

Parameters

[in]	message_hash	Message or hash value to which to attach signature
[in,out]	signature	Signature text storage destination information
[in]	wrapped_key	Input wrapped key of secp384r1 private key.

◆ ECDSA_secp192r1_SignatureVerify

fsp_err_t(* sce_api_t::ECDSA_secp192r1_SignatureVerify) (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)

ECDSA signature verification.

Implemented as

- R_SCE_ECDSA_secp192r1_SignatureVerify()

Parameters

[in]	signature	Signature text information to be verified
[in,out]	message_hash	Message or hash value to be verified
[in]	wrapped_key	Input wrapped key of secp192r1 public key.

◆ ECDSA_secp224r1_SignatureVerify

fsp_err_t(* sce_api_t::ECDSA_secp224r1_SignatureVerify) (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)

ECDSA signature verification.

Implemented as

- R_SCE_ECDSA_secp224r1_SignatureVerify()

Parameters

[in]	signature	Signature text information to be verified
[in,out]	message_hash	Message or hash value to be verified
[in]	wrapped_key	Input wrapped key of secp224r1 public key.

◆ ECDSA_secp256r1_SignatureVerify

fsp_err_t(* sce_api_t::ECDSA_secp256r1_SignatureVerify) (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)

ECDSA signature verification.

Implemented as

- R_SCE_ECDSA_secp256r1_SignatureVerify()

Parameters

[in]	signature	Signature text information to be verified
[in,out]	message_hash	Message or hash value to be verified
[in]	wrapped_key	Input wrapped key of secp256r1 public key.

◆ ECDSA_secp384r1_SignatureVerify

fsp_err_t(* sce_api_t::ECDSA_secp384r1_SignatureVerify) (sce_ecdsa_byte_data_t *signature, sce_ecdsa_byte_data_t *message_hash, sce_ecc_public_wrapped_key_t *wrapped_key)

ECDSA signature verification.

Implemented as

- R_SCE_ECDSA_secp384r1_SignatureVerify()

Parameters

[in]	signature	Signature text information to be verified
[in,out]	message_hash	Message or hash value to be verified
[in]	wrapped_key	Input wrapped key of secp384r1 public key.

◆ ECDH_secp256r1_Init

fsp_err_t(* sce_api_t::ECDH_secp256r1_Init) (sce_ecdh_handle_t *handle, uint32_t key_type, uint32_t use_key_id)

secp256r1 ECDH Initialization.

Implemented as

- R_SCE_ECDH_secp256r1_Init()

Parameters

[in,out]	handle	ECDH handler (work area)
[in]	key_type	Key exchange type
[in]	use_key_id	use key_id or not

◆ ECDH_secp256r1_PublicKeySign

fsp_err_t(* sce_api_t::ECDH_secp256r1_PublicKeySign) (sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, sce_ecc_private_wrapped_key_t *ecc_private_wrapped_key, uint8_t *public_key, sce_ecdsa_byte_data_t *signature, sce_ecc_private_wrapped_key_t *wrapped_key)

secp256r1 ECDH public key Signature.

Implemented as

- R_SCE_ECDH_secp256r1_PublicKeySign()

Parameters

[in,out]	handle	ECDH handler (work area)
[in]	ecc_public_wrapped_key	For ECDHE, input a null pointer. For ECDH, input the wrapped key of a secp256r1 public key.
[in]	ecc_private_wrapped_key	secp256r1 private key for signature generation
[in,out]	public_key	User secp256r1 public key (512-bit) for key exchange.
[in,out]	signature	Signature text storage destination information
[in,out]	wrapped_key	For ECDHE, a private wrapped key generated from a random number. Not output for ECDH.

◆ ECDH_secp256r1_PublicKeyVerify

fsp_err_t(* sce_api_t::ECDH_secp256r1_PublicKeyVerify) (sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, uint8_t *public_key_data, sce_ecdsa_byte_data_t *signature, sce_ecc_public_wrapped_key_t *wrapped_key)

secp256r1 ECDH public key verification.

Implemented as

- R_SCE_ECDH_secp256r1_PublicKeyVerify()

Parameters

[in,out]	handle	ECDH handler (work area)
[in]	ecc_public_wrapped_key	Public wrapped key area for signature verification
[in]	public_key_data	secp256r1 public key (512-bit)
[in]	signature	ECDSA secp256r1 signature of ecc_public_wrapped_key
[in,out]	wrapped_key	wrapped key of ecc_public_wrapped_key

◆ ECDH_secp256r1_PublicKeyReadWithoutSignature

fsp_err_t(* sce_api_t::ECDH_secp256r1_PublicKeyReadWithoutSignature) (sce_ecdh_handle_t *handle, uint8_t *public_key_data, sce_ecc_public_wrapped_key_t *wrapped_key)

Output the key index of QeU without signature verification.

Implemented as

- R_SCE_ECDH_secp256r1_PublicKeyReadWithoutSignature()

Parameters

[in,out]	handle	ECDH handler (work area)
[in]	public_key_data	secp256r1 public key (512-bit). When key_id is used: key_id (8-bit) || public key (512-bit)
[in,out]	wrapped_key	wrapped key of ecc_public_wrapped_key

◆ ECDH_secp256r1_SharedSecretCalculate

fsp_err_t(* sce_api_t::ECDH_secp256r1_SharedSecretCalculate) (sce_ecdh_handle_t *handle, sce_ecc_public_wrapped_key_t *ecc_public_wrapped_key, sce_ecc_private_wrapped_key_t *ecc_private_wrapped_key, sce_ecdh_wrapped_key_t *shared_secret_wrapped_key)

secp256r1 ECDH shared secret calculation.

Implemented as

- R_SCE_ECDH_secp256r1_SharedSecretCalculate()

Parameters

[in,out]	handle	ECDH handler (work area)
[in]	ecc_public_wrapped_key	Public wrapped key
[in]	ecc_private_wrapped_key	Private wrapped key
[in,out]	shared_secret_wrapped_key	Wrapped key of shared secret Z calculated by ECDH key exchange

◆ ECDH_secp256r1_KeyDerivation

fsp_err_t(* sce_api_t::ECDH_secp256r1_KeyDerivation) (sce_ecdh_handle_t *handle, sce_ecdh_wrapped_key_t *shared_secret_wrapped_key, uint32_t key_type, uint32_t kdf_type, uint8_t *other_info, uint32_t other_info_length, sce_hmac_sha_wrapped_key_t *salt_wrapped_key, sce_aes_wrapped_key_t *wrapped_key)

secp256r1 ECDH key derivation.

Implemented as

- R_SCE_ECDH_secp256r1_KeyDerivation()

Parameters

[in,out]	handle	ECDH handler (work area)
[in]	shared_secret_wrapped_key	Z wrapped key calculated by R_SCE_ECDH_secp256r1_SharedSecretCalculate
[in]	key_type	Derived key type
[in]	kdf_type	Algorithm used for key derivation calculation
[in]	other_info	Additional data used for key derivation calculation
[in]	other_info_length	Data length of other_info
[in]	salt_wrapped_key	Salt wrapped key
[in,out]	wrapped_key	Wrapped key corresponding to key_type.

◆ TLS_RootCertificateRSA2048PublicKeyInstall

fsp_err_t(* sce_api_t::TLS_RootCertificateRSA2048PublicKeyInstall) (uint8_t *encrypted_provisioning_key, uint8_t *initial_vector, uint8_t *encrypted_key, sce_tls_ca_certification_public_wrapped_key_t *wrapped_key)

Generate TLS RSA Public key index data

Implemented as

- R_SCE_TLS_RootCertificateRSA2048PublicKeyInstall()

Parameters

[in]	encrypted_provisioning_key	the provisioning key includes encrypted CBC/CBC-MAC key for user key
[in]	initial_vector	the initial_vector for user key CBC encrypt
[in]	encrypted_key	the user key encrypted with AES128-ECB mode
[out]	wrapped_key	the user Key Generation Information (141 words) of RSA2048 bit

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

◆ TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate

fsp_err_t(* sce_api_t::TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate) (sce_tls_p256_ecc_wrapped_key_t *tls_p256_ecc_wrapped_key, uint8_t *ephemeral_ecdh_public_key)

Generate TLS ECC key pair

Implemented as

- R_SCE_TLS_ECC_secp256r1_EphemeralWrappedKeyPairGenerate()

Parameters

[in]	tls_p256_ecc_wrapped_key	P256 ECC key index for TLS
[in]	ephemeral_ecdh_public_key	ephemeral ECDH public key

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

◆ TLS_RootCertificateVerify

fsp_err_t(* sce_api_t::TLS_RootCertificateVerify) (uint32_t public_key_type, uint8_t *certificate, uint32_t certificate_length, uint32_t public_key_n_start_position, uint32_t public_key_n_end_position, uint32_t public_key_e_start_position, uint32_t public_key_e_end_position, uint8_t *signature, uint32_t *encrypted_root_public_key)

Verify root CA certificate.

Implemented as

- R_SCE_TLS_RootCertificateVerify()

Parameters

[in]	public_key_type	key type
[in]	certificate	certificates.
[in]	certificate_length	byte size of certificates.
[in]	public_key_n_start_position	start position of public key n.
[in]	public_key_n_end_position	end position of public key n.
[in]	public_key_e_start_position	start position of public key e.
[in]	public_key_e_end_position	end position of public key e.
[in]	signature	signature for certificates.
[out]	encrypted_root_public_key	public key for RSA 2048bit.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

◆ TLS_CertificateVerify

fsp_err_t(* sce_api_t::TLS_CertificateVerify) (uint32_t public_key_type, uint32_t *encrypted_input_public_key, uint8_t *certificate, uint32_t certificate_length, uint8_t *signature, uint32_t public_key_n_start_position, uint32_t public_key_n_end_position, uint32_t public_key_e_start_position, uint32_t public_key_e_end_position, uint32_t *encrypted_output_public_key)

Verify server certificate and intermediate certificate.

Implemented as

- R_SCE_TLS_CertificateVerify()

Parameters

[in]	public_key_type	key type
[in]	input_public_key	public key.
[in]	certificate	certificates.
[in]	certificate_length	byte size of certificates.
[in]	signature	signature for certificates.
[in]	public_key_n_start_position	start position of public key n.
[in]	public_key_n_end_position	end position of public key n.
[in]	public_key_e_start_position	start position of public key e.
[in]	public_key_e_end_position	end position of public key e.
[out]	encrypted_output_public_key	public key for RSA 2048bit.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

◆ TLS_PreMasterSecretGenerateForRSA2048

fsp_err_t(* sce_api_t::TLS_PreMasterSecretGenerateForRSA2048) (uint32_t *sce_pre_master_secret)

Generate encrypted pre-master secret.

Implemented as

- R_SCE_TLS_PreMasterSecretGenerateForRSA2048()

Parameters

[out]	sce_pre_master_secret	pre-master secret value for SCE.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

◆ TLS_MasterSecretGenerate

fsp_err_t(* sce_api_t::TLS_MasterSecretGenerate) (uint32_t select_cipher_suite, uint32_t *sce_pre_master_secret, uint8_t *client_random, uint8_t *server_random, uint32_t *sce_master_secret)

Generate encrypted master secret.

Implemented as

- R_SCE_TLS_MasterSecretGenerate()

Parameters

[in]	select_cipher_suite	cipher suite type
[in]	sce_pre_master_secret	pre-master secret value for SCE.
[in]	client_random	random value reported ClientHello.
[in]	server_random	random value reported ServerHello.
[out]	sce_master_secret	master secret value for SCE.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

◆ TLS_PreMasterSecretEncryptWithRSA2048

fsp_err_t(* sce_api_t::TLS_PreMasterSecretEncryptWithRSA2048) (uint32_t *encrypted_public_key, uint32_t *sce_pre_master_secret, uint8_t *encrypted_pre_master_secret)

Output the result encrypted pre-master secret with RSA 2048bit

Implemented as

- R_SCE_TLS_PreMasterSecretEncryptWithRSA2048()

Parameters

[in]	encrypted_public_key	public key data.
[in]	sce_pre_master_secret	pre-master secret value.
[out]	encrypted_pre_master_secret	the value encrypted pre-master secret.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

◆ TLS_SessionKeyGenerate

fsp_err_t(* sce_api_t::TLS_SessionKeyGenerate) (uint32_t select_cipher_suite, uint32_t *sce_master_secret, uint8_t *client_random, uint8_t *server_random, uint8_t *nonce_explicit, sce_hmac_sha_wrapped_key_t *client_mac_wrapped_key, sce_hmac_sha_wrapped_key_t *server_mac_wrapped_key, sce_aes_wrapped_key_t *client_crypto_wrapped_key, sce_aes_wrapped_key_t *server_crypto_wrapped_key, uint8_t *client_initial_vector, uint8_t *server_initial_vector)

Output various key information.

Implemented as

- R_SCE_TLS_SessionKeyGenerate()

Parameters

[in]	select_cipher_suite	Key suite information number.
[in]	sce_master_secret	master secret value.
[in]	client_random	random value reported ClientHello.
[in]	server_random	random value reported ServerHello.
[in]	nonce_explicit	nonce value

[out]	client_mac_wrapped_key	the mac key during communication from client to server.
[out]	server_mac_wrapped_key	the mac key during communication from server to client.
[out]	client_crypto_wrapped_key	the crypto key during communication from client to server.
[out]	server_crypto_wrapped_key	the crypto key during communication from server to client.
[in]	client_initial_vector	not use.
[in]	server_initial_vector	not use.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

◆ TLS_VerifyDataGenerate

fsp_err_t(* sce_api_t::TLS_VerifyDataGenerate) (uint32_t select_verify_data, uint32_t *sce_master_secret, uint8_t *hand_shake_hash, uint8_t *verify_data)

Generate verify data.

Implemented as

- R_SCE_TLS_VerifyDataGenerate()

Parameters

[in]	select_verify_data	Select Client/Server data.
[in]	sce_master_secret	master secret data.
[in]	hand_shake_hash	TLS hand shake message SHA256 HASH value.
[out]	verify_data	verify data.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

◆ TLS_ServerKeyExchangeVerify

fsp_err_t(* sce_api_t::TLS_ServerKeyExchangeVerify) (uint32_t public_key_type, uint8_t *client_random, uint8_t *server_random, uint8_t *server_ephemeral_ecdh_public_key, uint8_t *server_key_exchange_signature, uint32_t *encrypted_public_key, uint32_t *encrypted_ephemeral_ecdh_public_key)

Retrives ECDH public key.

Implemented as

- R_SCE_TLS_ServerKeyExchangeVerify()

Parameters

[in]	public_key_type	key type
[in]	client_random	random value reported ClientHello.
[in]	server_random	random value reported ServerHello.
[in]	server_ephemeral_ecdh_public_key	Ephemeral ECDH public key from Server.
[in]	server_key_exchange_signature	Server Key Exchange sigunature.
[in]	encrypted_public_key	encrypted public key.
[out]	encrypted_ephemeral_ecdh_public_key	encrypted Ephemeral ECDH public key.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

◆ TLS_PreMasterSecretGenerateForECC_secp256r1

fsp_err_t(* sce_api_t::TLS_PreMasterSecretGenerateForECC_secp256r1) (uint32_t *encrypted_public_key, sce_tls_p256_ecc_wrapped_key_t *tls_p256_ecc_wrapped_key, uint32_t *sce_pre_master_secret)

Generate encrypted pre-master secret.

Implemented as

- R_SCE_TLS_PreMasterSecretGenerateForECC_secp256r1()

Parameters

[in]	encrypted_public_key	encrypted public key
[in]	tls_p256_ecc_wrapped_key	P-256 ECC key index.
[out]	sce_pre_master_secret	encrypted pre-master secret value for SCE.

Return values

FSP_SUCCESS	Normal termination
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
FSP_ERR_CRYPTO_SCE_FAIL	An internal error occurred.

◆ sce_instance_t

struct sce_instance_t

This structure encompasses everything that is needed to use an instance of this interface.

Data Fields		
sce_ctrl_t *	p_ctrl	Pointer to the control structure for this instance.
sce_cfg_t const *	p_cfg	Pointer to the configuration structure for this instance.
sce_api_t const *	p_api	Pointer to the API structure for this instance.

Typedef Documentation

◆ sce_rsa_byte_data_t

typedef sce_byte_data_t sce_rsa_byte_data_t

byte data

RSA byte data structure

◆ sce_ecdsa_byte_data_t

typedef sce_byte_data_t sce_ecdsa_byte_data_t
byte data ECDSA byte data structure

◆ sce_ctrl_t

typedef void sce_ctrl_t
SCE Control block. Allocate an instance specific control block to pass into the API calls. **Implemented as** ◦ sce_instance_ctrl_t

Enumeration Type Documentation

◆ lifecycle_t

enum lifecycle_t
Data lifecycle

Chapter 5 Copyright

Renesas FSP

User's Manual

Publication Date: Revision 2.50 Dec.3.2021

Renesas FSP v3.5.0

User’s Manual

Renesas Electronics Corporation