User Manual

DA14585 Voice RCU Software Manual

UM-B-086

Abstract

This document describes the software of the DA14585 Voice Remote Control Unit reference design application, based on the DA14585 Bluetooth 5.0 SoC with Audio Interface.

Contents

 © 2022 Renesas Electronics

© 2022 Renesas Electronics

Figures

© 2022 Renesas Electronics

Tables

© 2022 Renesas Electronics

1 Terms and Definitions

ADPCM	Adaptive Differential Pulse-Code Modulation
ATT	ATTribute (protocol)
BLE	Bluetooth low energy
CCC	Client Characteristic Configuration
DIS	Device Information Service
DLE	Data packet Length Extension
EEPROM	Electrically Erasable Programmable Read Only Memory
FSM	Finite State Machine
GAP	Generic Access Profile
GATT	Generic ATTribute profile
GPIO	General Purpose Input Output
GUI	Graphical User Interface
HID	Human Interface Device
HOGP	HID Over GATT Profile
I2C	Inter-Integrated Circuit (bus)
IMA	Interactive Multimedia Association
L2CAP	Logical Link Control and Adaptation Protocol
LED	Light Emitting Diode
LLD	Low Level Driver
MITM	Man In The Middle
NVM	Non-Volatile Memory (Flash or EEPROM)
PCB	Printed Circuit Board
PCM	Pulse Code Modulation
PDM	Pulse Density Modulation
PTT	Push To Talk
RAM	Random Access Memory
RCU	Remote Control Unit
SoC	System on a Chip
SPI	Serial Peripheral Interface
SWD	Serial Wire Debug
USB	Universal Serial Bus
UUID	Universally Unique IDentifier

© 2022 Renesas Electronics

2 References

[1] DA14585 Bluetooth 5.0 SoC with Audio Interface, Datasheet, Dialog Semiconductor.

[2] UM-B-080 DA14585/586 SDK 6 Software Developer's Guide, User Manual, Dialog Semiconductor.

[3] UM-B-079 DA14585/586 SDK 6 Software Platform Reference, User Manual, Dialog Semiconductor.

[4] UM-B-049 DA1458x Getting started with Development Kit - Pro, User Manual, Dialog Semiconductor.

[5] UM-B-009 DA14580 Keyboard reference application, User Manual, Dialog Semiconductor.

[6] UM-B-037 DA14580 Remote control BLE & IR reference application, User Manual, Dialog Semiconductor.

[7] AN-B-10 DA1458x using SUOTA, Application Note, Dialog Semiconductor.

[8] HID over GATT Profile, Specification, Bluetooth SIG.

[9] HID Service, Specification, Bluetooth SIG.

[10] Battery Service, Specification, Bluetooth SIG.

[11] Device Information Service, Specification, Bluetooth SIG.

[12] Bluetooth core 5.0, Specification, Bluetooth SIG.

[13] Bluetooth SIG website: Specification Adopted Documents, Bluetooth SIG.

[14] Universal Serial Bus - Device Class Definition for Human Interface, Specification, USB Implementers' Forum.

[15] Universal Serial Bus - HID Usage Tables, Specification, USB Implementers' Forum.

[16] UM-B-087 DA14585 Voice RCU Hardware Manual, User manual, Dialog Semiconductor.

 © 2022 Renesas Electronics

3 Introduction

This document describes the software of the Voice Remote Control Unit reference design, which is based on the DA14585 Bluetooth® low energy 5.0 SoC with Audio Interface. This user guide describes the software architecture and the software components that can be optionally included to implement various features of the Voice RCU reference design. It also describes the functionality of each component, as well as the mechanism to enable and configure it. Finally, it briefly describes the hardware components used in the Voice RCU reference design. The hardware is described in detail in the Voice RCU hardware manual [16].

The developer is suggested to get familiar with the DA14585 software and hardware, looking into the development kit [4], software developer's guide [2], the software platform reference manual [3] and the DA14580 Keyboard Reference application [5].

4 Features

The Voice RCU reference design supports the following features:

- 12 keys in a 4 x 3 matrix (customizable)
- On-board non-volatile Flash memory for storing the firmware and the bonding information
- Simultaneous key presses
- Programmable key debouncing
- Key de-ghosting
- Audio capturing using a PDM microphone
- 16-bit, 8 kHz or 16 kHz Audio IMA ADPCM encoder
- Adaptive audio sampling rate
- Audio data transfer over HOGP or custom Dialog Audio BLE service
- Audio buffering capabilities exceeding 1 second
- Pointing device functionality using a gyro/accelerometer sensor
- Pointing device functionality using a trackpad
- Advanced user input using a touch slider
- Infra-Red (IR) LED transmitter
- Two LEDs for indicating RCU state
- A magnetic buzzer for indicating RCU state
- Software Update Over The Air (SUOTA)
- Low external component count
- Ultra-low power operation
- Host demo application with audio capabilities for Android platform

The reference application is based on the HID over GATT Profile [8]. It is an adaptation of the USB HID specification for operation over a Bluetooth Low Energy wireless link. The HID over GATT profile requires the Generic Attribute Profile (GATT), the Battery Service and the Device Information Service. The remote application implements the HID Device role. The GATT role is Server and the GAP role is Peripheral.

The reference application exposes the following services:

- Device Information Service
- HID Over GATT Profile (HOGP)
- Battery Service
- GAP Service
- GATT Service

© 2022 Renesas Electronics

- SUOTA Service (optional)
- Custom Dialog Audio Service (optional)

For more information see the HID over GATT Profile [8], the HID Service Specification [9], the Device Information Specification [11] and the Battery Service Specification [10]. The Bluetooth Core 5.0 Specification [12] contains detailed information about GATT and GAP.

5 System Architecture

The system architecture of the Voice RCU reference design is depicted in Figure 1.

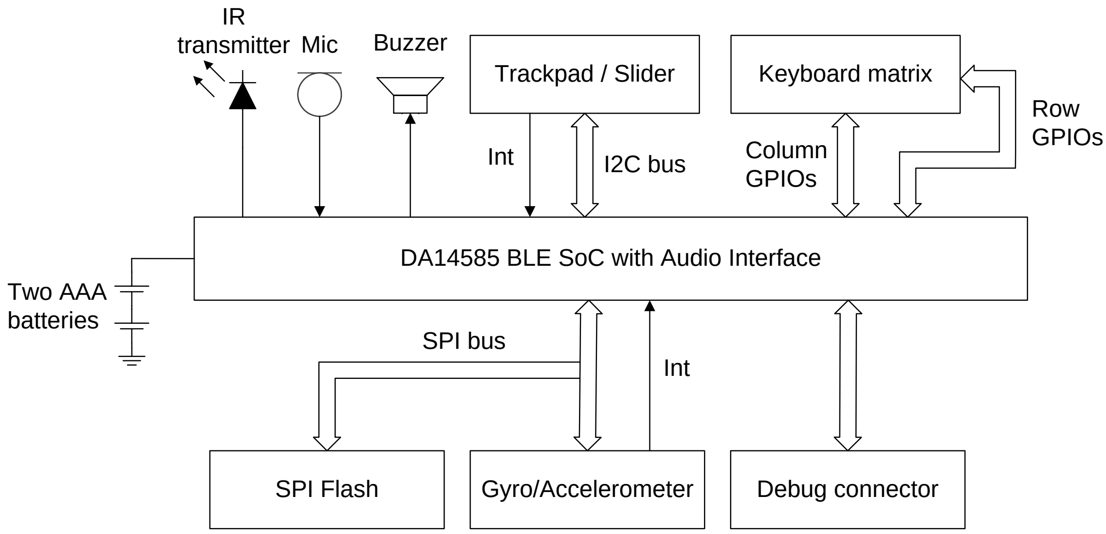


Figure 1: System Block Diagram

The system is based on DA14585 Bluetooth low energy SoC with Audio Interface. All peripheral devices are directly connected to the DA14585, minimizing the external component count.

Two AAA batteries connected in series supply a nominal voltage of 3 V to the system power rail. DA14585 and all peripheral devices are powered by this rail.

The system features two data buses:

- The I2C bus, to which the trackpad or slider controller is connected. Slider and trackpad cannot coexist due to physical limitations.
- The SPI bus which is shared between the SPI Flash memory and the gyro/accelerometer sensor. Care must be taken to keep the chip select (CS) signal of the gyro/accelerometer sensor de-asserted during in-system programming of the SPI Flash memory.

The rest of the peripheral devices are connected to DA14585 GPIO pins. The peripheral devices used in the RCU reference design are listed in Table 1.

Table 1: Voice RCU Key Peripheral Components

Block	Part Number	Interface	Comments
Flash Memory	Macronix MX25R2035F	SPI	External Flash memory with SPI interface used to store the firmware and the bonding data
Key Matrix	-	7 GPIOs	12 keys, 4 x 3 key configuration
Gyro/Accelerometer	Bosch BMI160	SPI, interrupt line	Accelerometer and gyro sensor used for the pointing device functionality
Trackpad Controller	Azoteq IQS572	I2C, interrupt line	Trackpad controller used in the trackpad add-on module

© 2022 Renesas Electronics

Block	Part Number	Interface	Comments
Slider/Scroll Wheel Controller	Azoteq IQS263	I2C, interrupt line	Trackpad controller used in the scroll wheel add-on module
Buzzer	CUI CSS-I4B20-SMT	1 GPIO	Magnetic buzzer for sound indications
IR LED	Kingbright WP7113F3BT	1 GPIO	IR LED transmitter
Microphone	Knowles SPK0838HT4H-B	PDM	PDM microphone for audio capture
Debug Connector	-	SWD, UART	2 x 5 pins 1.27 mm pitch female connector with SWD debug and UART interface

The reference design board features a debug connector which can be used to connect a debugger to the CPU, perform in-system programming of the SPI Flash memory and connect to a console to log the debugging messages. The pinout of the debug connection is depicted in Table 2.

Table 2: Debug Connector Pinout

Pin	Signal	Pin	Signal
1	UART Tx	2	UART Rx
3	Debugger SWCLK	4	Debugger SWDIO
5	-	6	Reset
7	Power supply (Note 1)	8	GND
9	Power supply (Note 1)	10	GND

Note 1 The system power supply can be switched between battery supply and debug connector supply, using the switch located at the left side of the RCU reference design.

The reference design board also features a test connector, the pinout of which is depicted in Table 3.

Table 3: Test Connector Pinout

Pin	Signal	Pin	Signal
1	-	2	-
3	GND	4	GND
5	GND	6	GND
7	P2_6 (I2C SCL)	8	P0_7 (Buzzer)
9	P2_7 (touchpad interrupt)	10	P2_5 (I2C SDA)
11	P2_4 (Keyboard row 4)	12	P2_8 (Keyboard column 1)
13	P2_3 (Keyboard row 3)	14	P0_2 (Keyboard column 3)
15	P2_2 (Keyboard row 2)	16	P0_1 (Keyboard column 2)
17	P2_1 (Keyboard row 1)	18	SWCLK
19	SWDIO	20	VBAT3V

© 2022 Renesas Electronics

6 Using the Reference Design Board

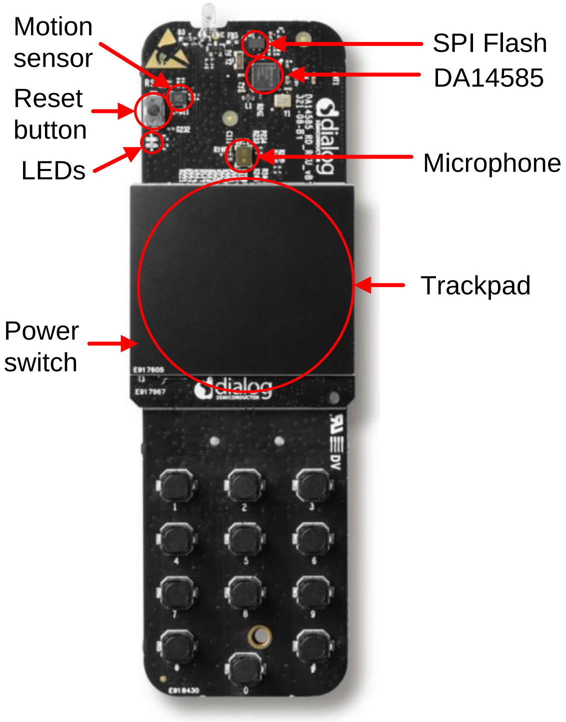


Figure 2: Voice RCU with Trackpad

Connect the debugger as shown in Figure 3 and follow the steps in Table 4 to build and download the firmware. Section 6.3 explains how to use the hardware.

6.1 Connecting the Debugger

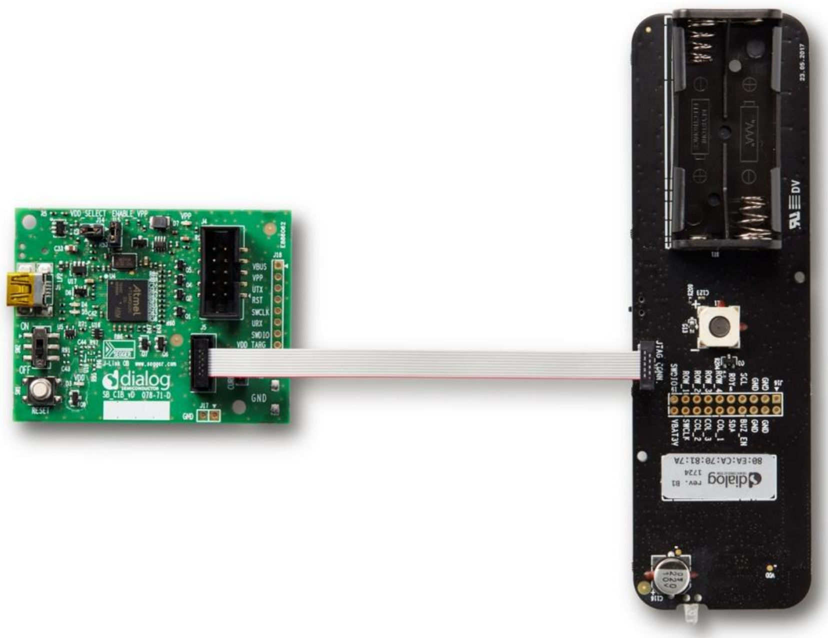
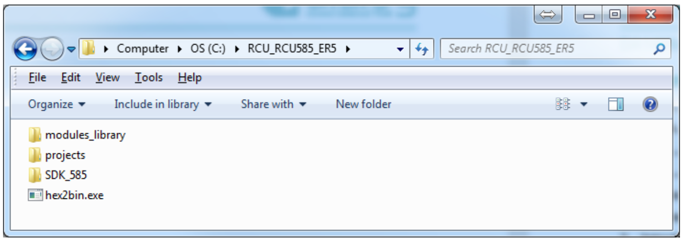
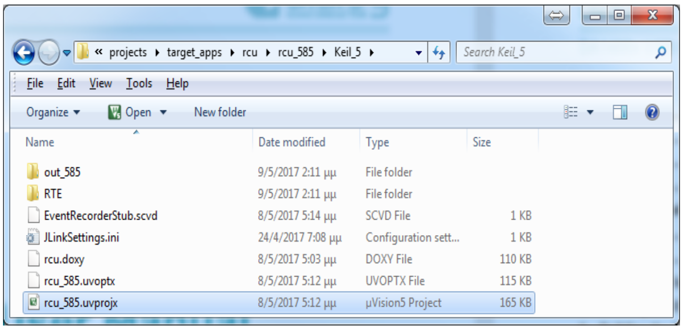
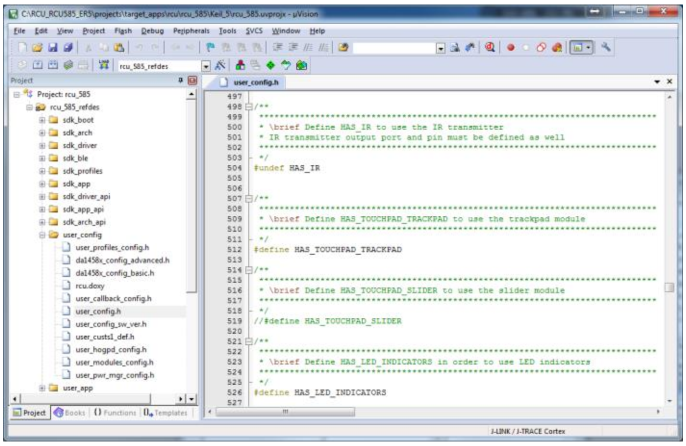
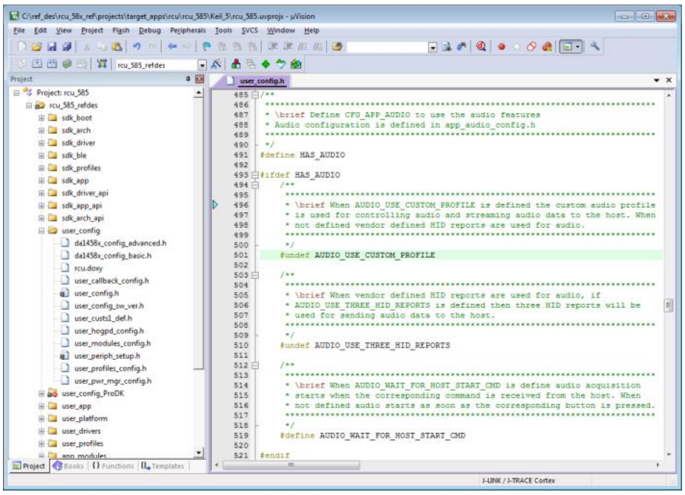


Figure 3: Connecting the Debugger

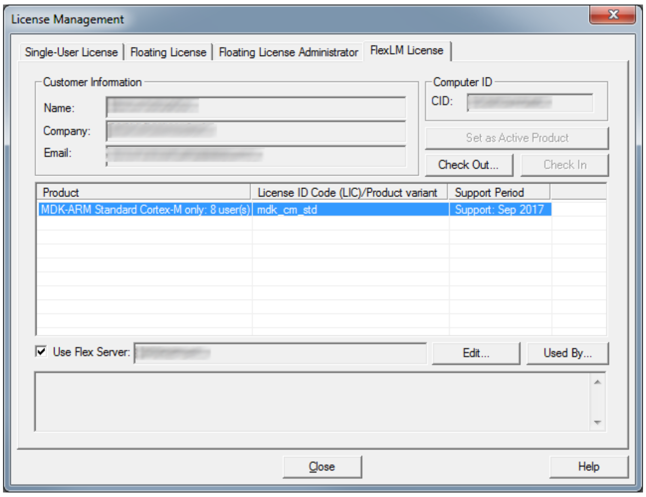
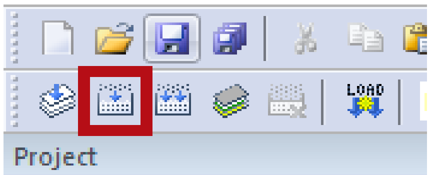
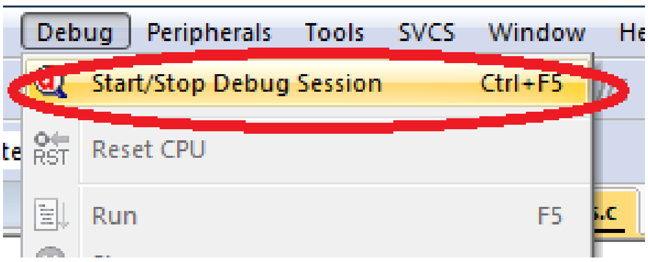
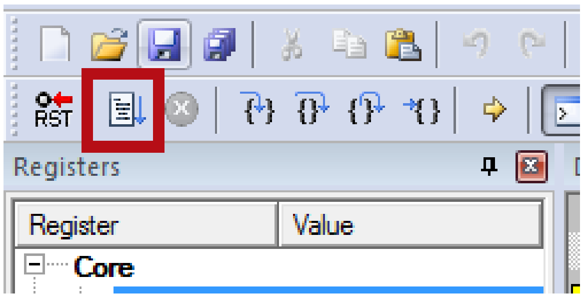
 © 2022 Renesas Electronics

6.2 Building and Downloading the Firmware

Table 4: Steps for Building and Downloading the Firmware

1	After you unzip the release archive, you will find these contents.	
2	Double click file `rcu_585.uvprojx` in `projects\target_apps\rcu\rcu_585\Keil_5`.	
3	Open file `user_config.h` from the `user_config` group in the Keil development environment and make sure either `HAS_TOUCHPAD_TRACKPAD` or `HAS_TOUCHPAD_SLIDER` is defined, depending your RCU configuration.	
4	If you want to use the audio feature with custom HID reports instead of the custom Dialog Audio service, make sure that `AUDIO_USE_CUSTOM_PROFILE` is undefined.	

 © 2022 Renesas Electronics

5	Make sure you have a license for code size over 32K, by opening menu **License Management** from the File menu. If no license is shown, please contact Keil to obtain a license.	
6	Build the project by pressing key **F7**, or click the **Build** button. If you want to: ● Build and download the firmware to other hardware, follow the steps in Table 42. ● Convert the output to a SUOTA image follow the steps in Appendix C.	
7	Start a debugging session by pressing **Ctrl+F5** or by using the Debug menu.	
8	Press key **F5** or click the **Run** button to start code execution.	

 © 2022 Renesas Electronics

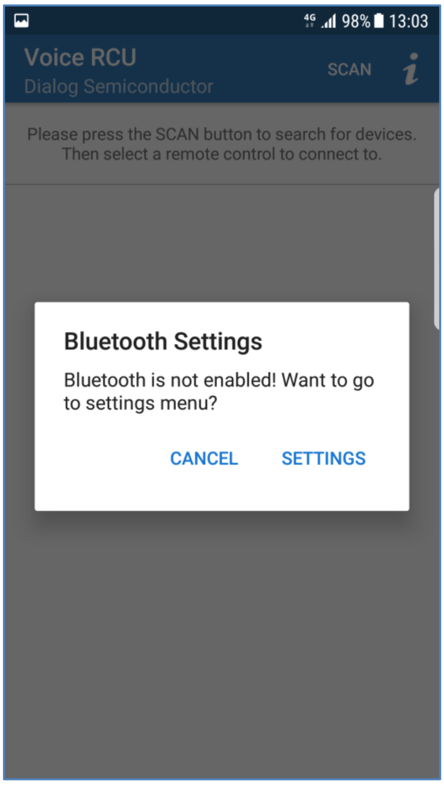
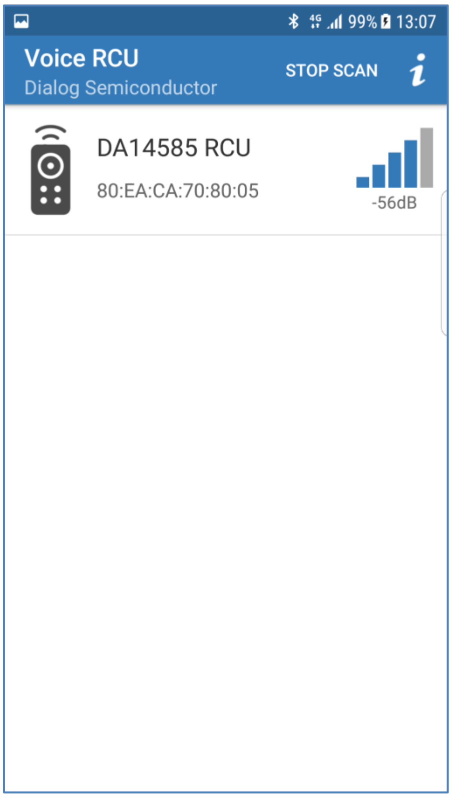
6.3 Using the Hardware

6.3.1 Connecting and Testing the Keypad and Sound

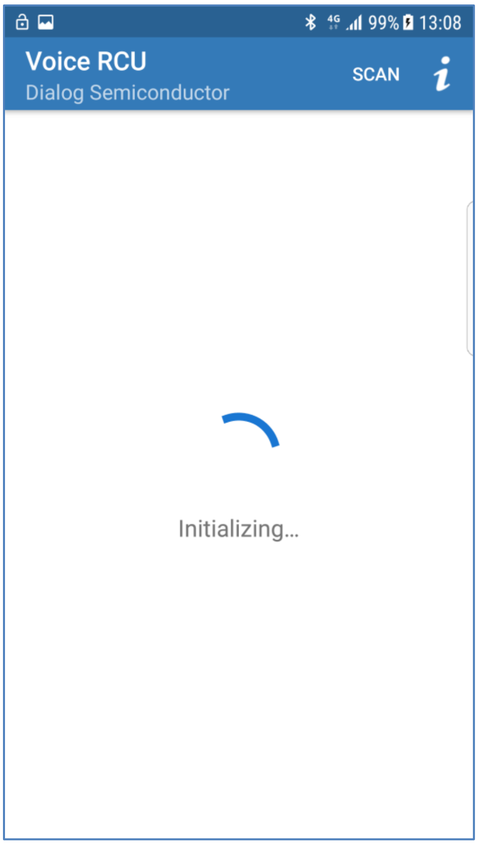
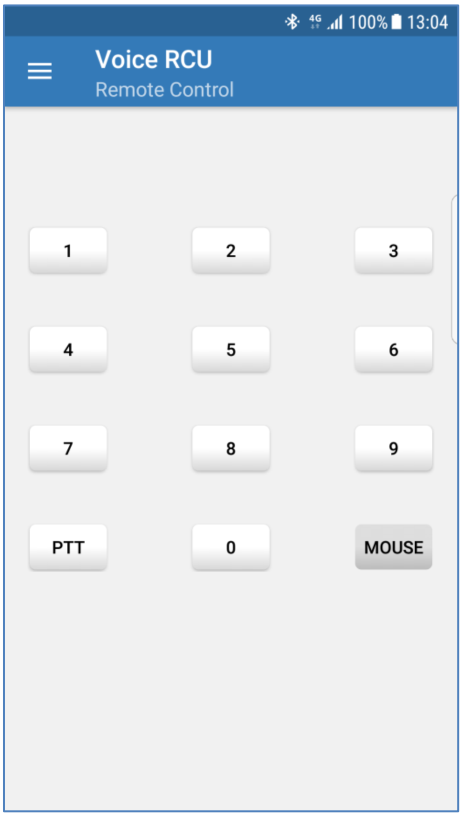
Table 5: Steps for Connecting and Testing the Keypad and Sound

1	Install the **Dialog Voice RCU** app from the Google Play Store.	
2	Open the application on your Android device.	

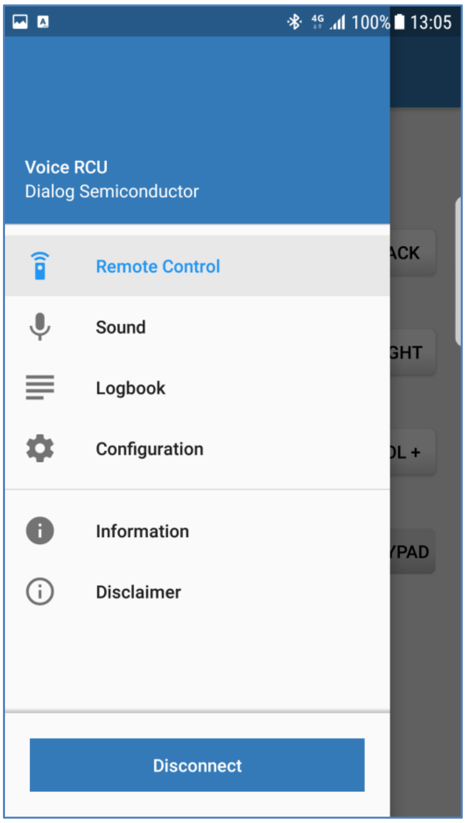
 © 2022 Renesas Electronics

3	Make sure Bluetooth is enabled.	
4	Tap on **SCAN** and press a button on the RCU so that it starts advertising. If the RCU was already paired with your device and you want to reconnect the RCU from scratch, follow the steps in Table 41 first.	

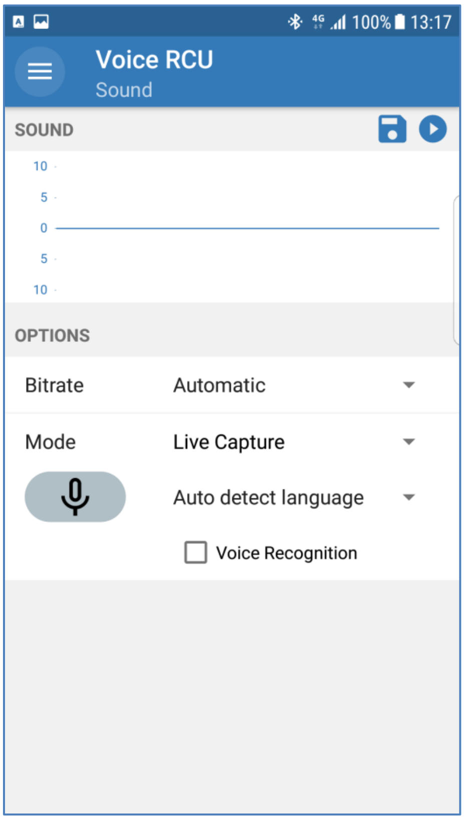
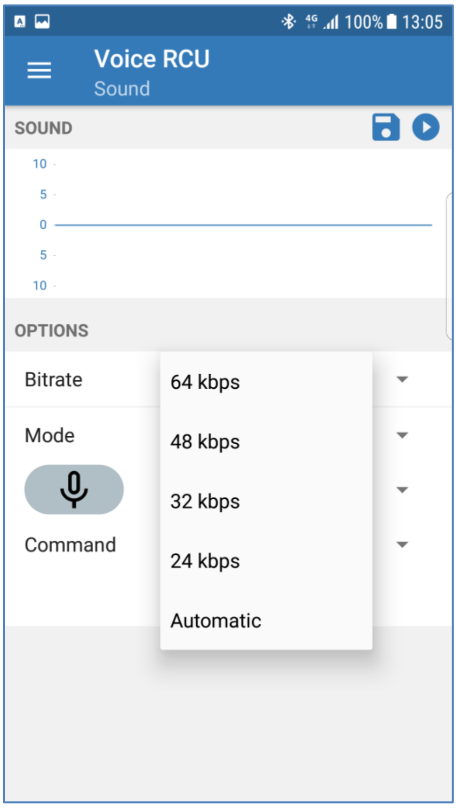
 © 2022 Renesas Electronics

5	Select the RCU and wait while it connects and initializes.	
6	The screen where you can test the keypad appears.	

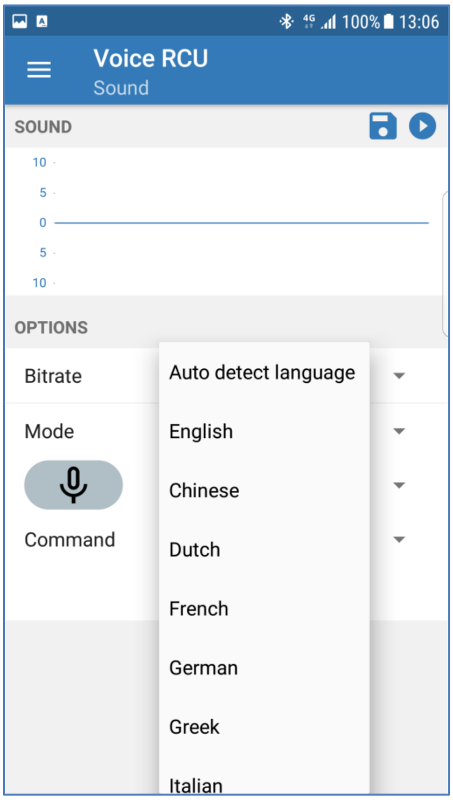
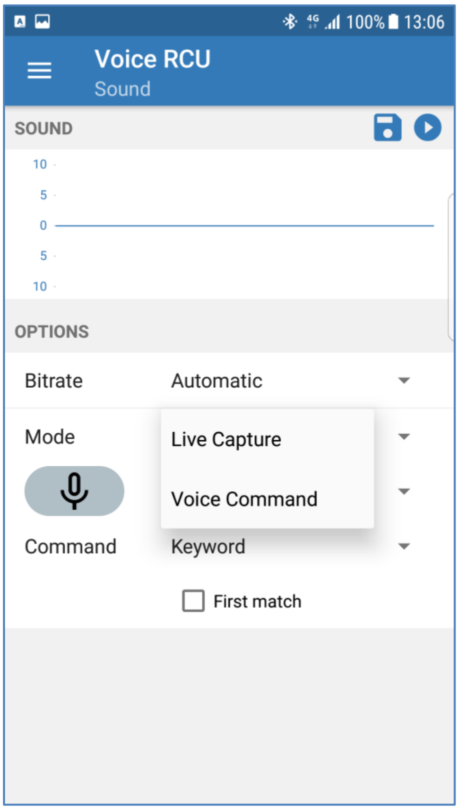
 © 2022 Renesas Electronics

7	Press # on the RCU to toggle between the **MOUSE** and **KEYPAD** mappings.	
8	Tap on the app menu and select **Sound** to test the audio functionality.	

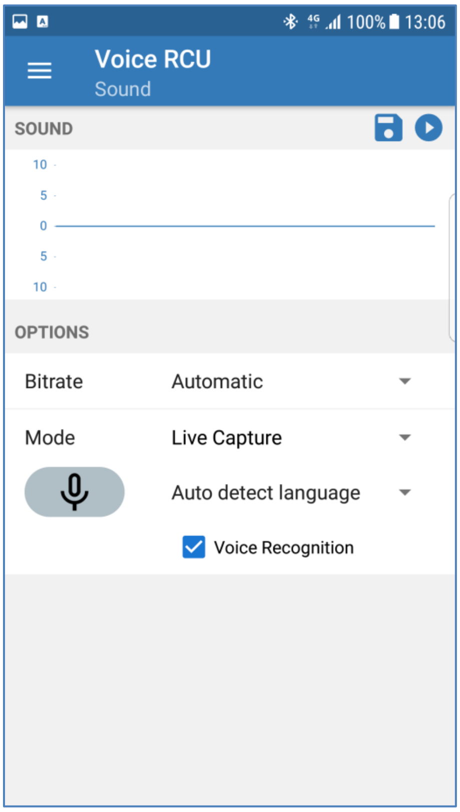
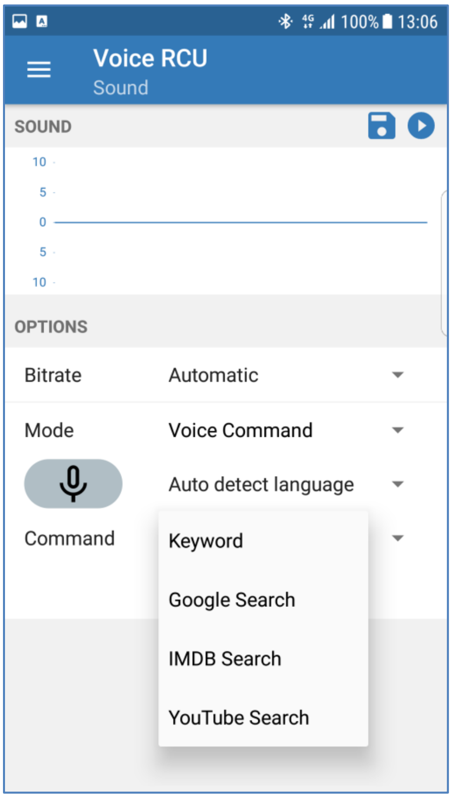
 © 2022 Renesas Electronics

9	The **Sound** screen appears.	
10	Select the required audio bit rate. When **Automatic** is chosen, the bit rate adapts dynamically.	

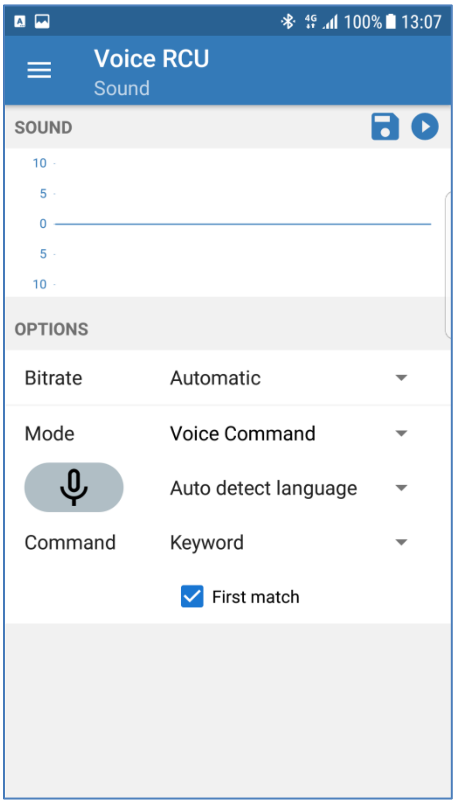
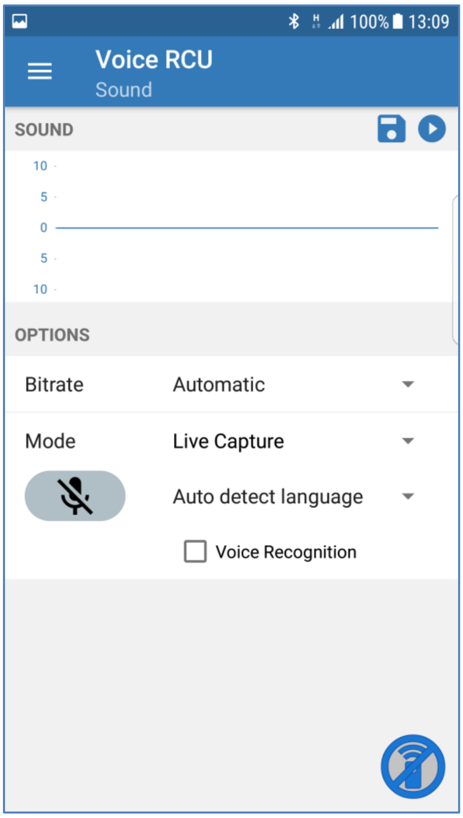
 © 2022 Renesas Electronics

11	Let the language be automatically detected or select among English, Chinese, Dutch, French, German, Greek, Italian, Japanese, Korean, Russian, and Spanish.	
12	Select the operating mode of the application: ● **Live Capture**: record audio and playback (see step 13). ● **Voice Command**: record audio, recognize and process the command (see step 14).	

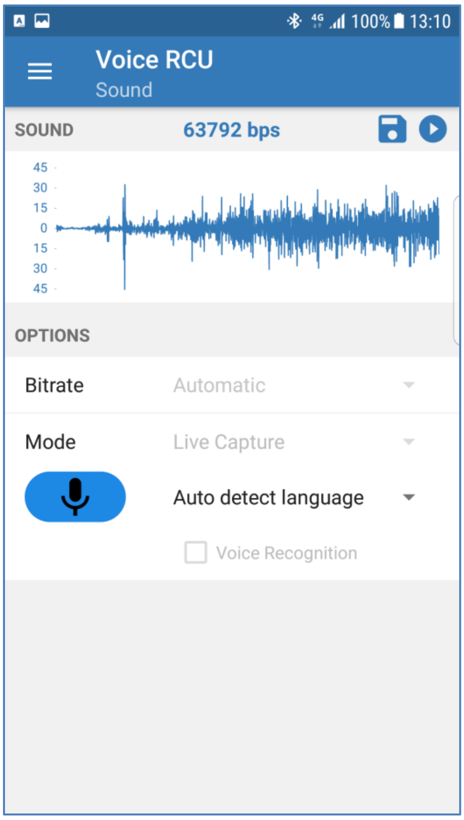
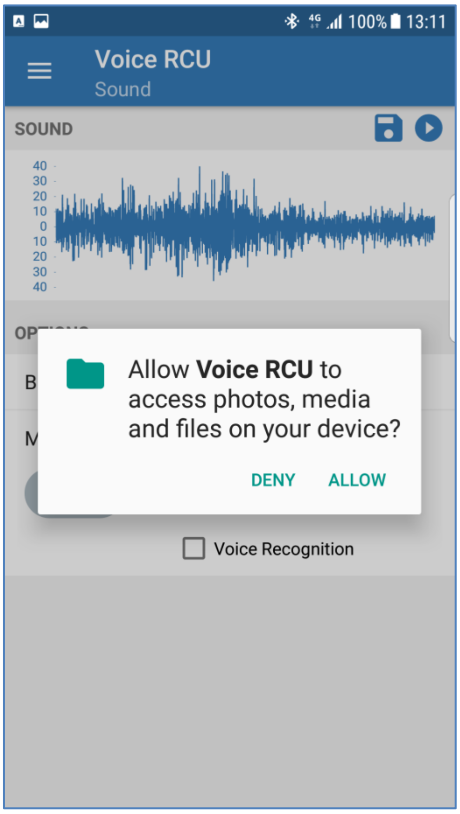
 © 2022 Renesas Electronics

13	When **Live Capture** is chosen, you may attempt **Voice Recognition** after the audio is captured.	
14	When **Voice Command** is chosen, you may select where to submit your keyword.	

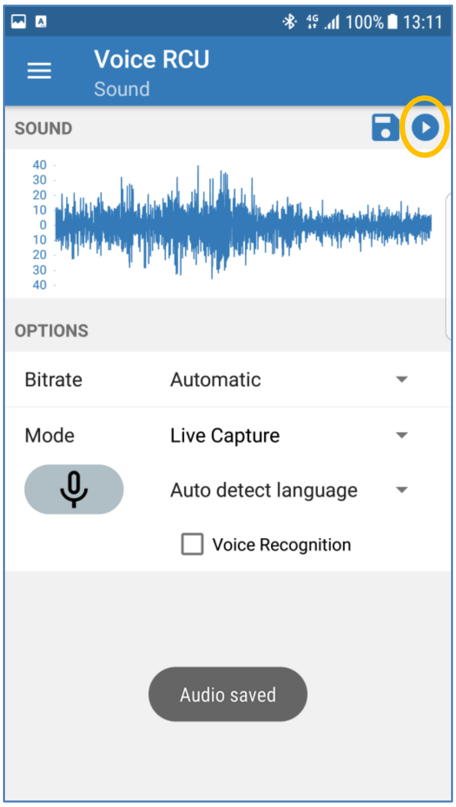
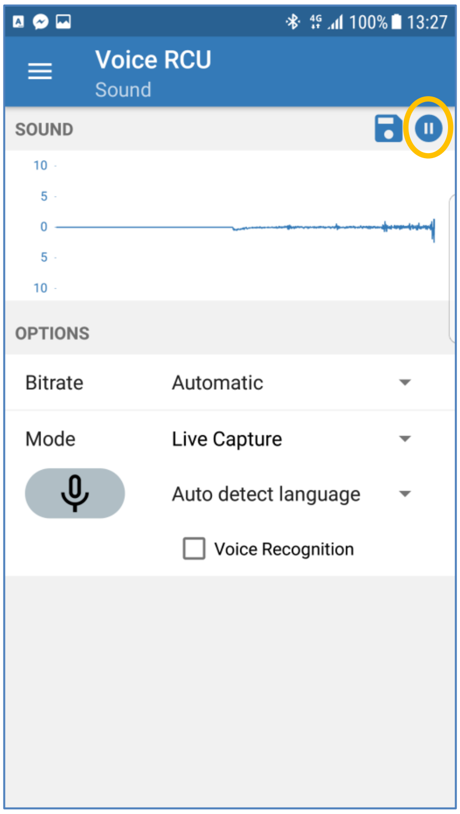
 © 2022 Renesas Electronics

15	You may also opt to automatically open the first result that is returned.	
16	If at any point the RCU is disconnected, the **Microphone** button will be disabled. Press any key to reconnect the microphone.	

 © 2022 Renesas Electronics

17	Press the **Microphone** button to start capturing audio and press it again to stop. The asterisk (*) button on the RCU can be used as PTT (Push To Talk). Press it and keep it pressed to capture audio. Release it to stop.	
18	You may use the diskette button to save the audio. If you receive this warning, select "ALLOW".	

© 2022 Renesas Electronics

19	When audio is saved, a notification appears for a short time. You may use the **Play** button to play it back.	
20	While the audio is played back, you may use the **Pause** button to stop it.	

© 2022 Renesas Electronics

6.3.2 Testing the Motion, Trackpad, Slider, IR and LED functionality

The motion, trackpad, slider, IR and LED functionality depends on definitions in file `user_config.h`:

- **Motion:** To test the motion data that the BMI160 sensor provides, you need a Nexus device with a special driver installed and `HAS_MOTION` must be defined (key **#** on the RCU activates the motion data and key **1** serves as the click button).
- **Trackpad:** To test the trackpad `HAS_TOUCHPAD_TRACKPAD` must be defined.
- **IR transmitter:** To test the IR transmitter you need an RC5 compatible receiver and `HAS_IR` must be defined.
- **LEDs:** To test the LEDs `HAS_LED_INDICATORS` must be defined. The functionality of the green and red LEDs is defined in `projects\target_apps\rcu\rcu_585\src\config\app_leds_config.h` as listed in Table 6. To use the ramp on/off values `LED_USE_RAMP_FEATURE` must be defined.

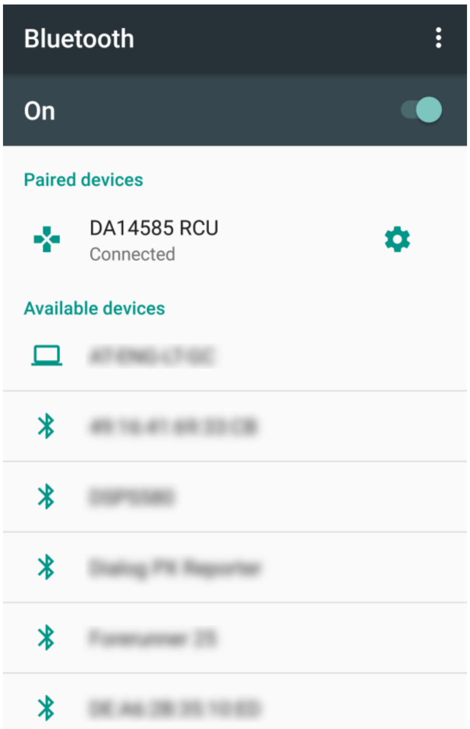
Table 6: LED Functionality

Condition	LED	Blinks	On Time (ms)	Period (ms)	Ramp On Time (ms)	Ramp Off Time (ms)
RCU connected	Green	1	100	-	-	-
RCU disconnected	Green	3	50	150	-	-
RCU advertising	Red	Continuous	500	1000	200	200
Connection in progress	Green	Continuous	20	100	-	-
Battery low	Red (with Green off)	15	400	2000	200	200
SUOTA in progress	Red	Continuous	150	1000	-	-
Motion active	Green	Continuous	50	1000	-	-
IR on	Red	1	1000	-	-	-
IR off	Red	2	100	200	-	-

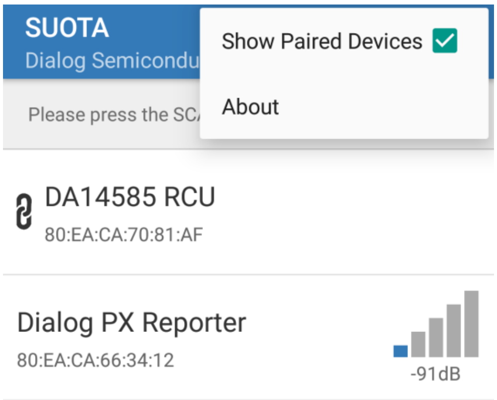
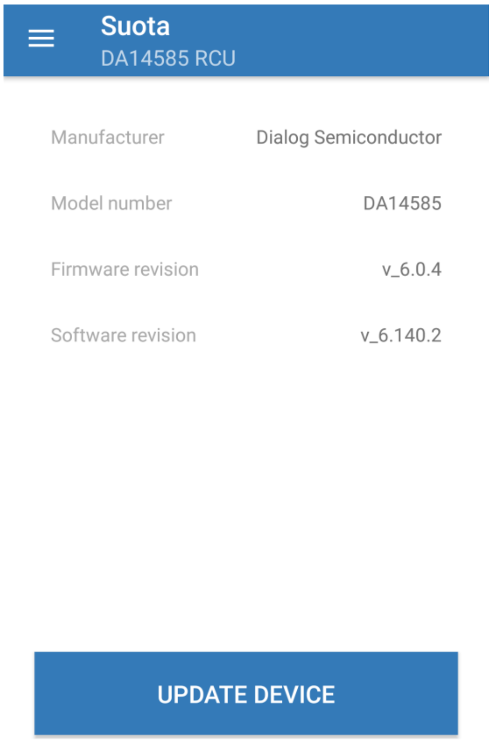
© 2022 Renesas Electronics

6.3.3 Firmware Updating Using SUOTA

Table 7: Steps for Firmware Updating Using SUOTA

1	Install the **Dialog SUOTA** app from the Google Play Store.	
2	Make sure that the RCU is properly paired in the Android Bluetooth settings.	

 © 2022 Renesas Electronics

3	Open the application on your Android device and make sure that **Show Paired Devices** is selected in the application menu.	
4	Select **DA14585 RCU** and proceed with the firmware update as described in AN-B-10 (see Ref. [7]). Refer to Appendix C for creating the SUOTA firmware image.	

 © 2022 Renesas Electronics

7 Software Architecture

7.1 General Description

The software architecture of the Voice RCU is depicted in Figure 4.

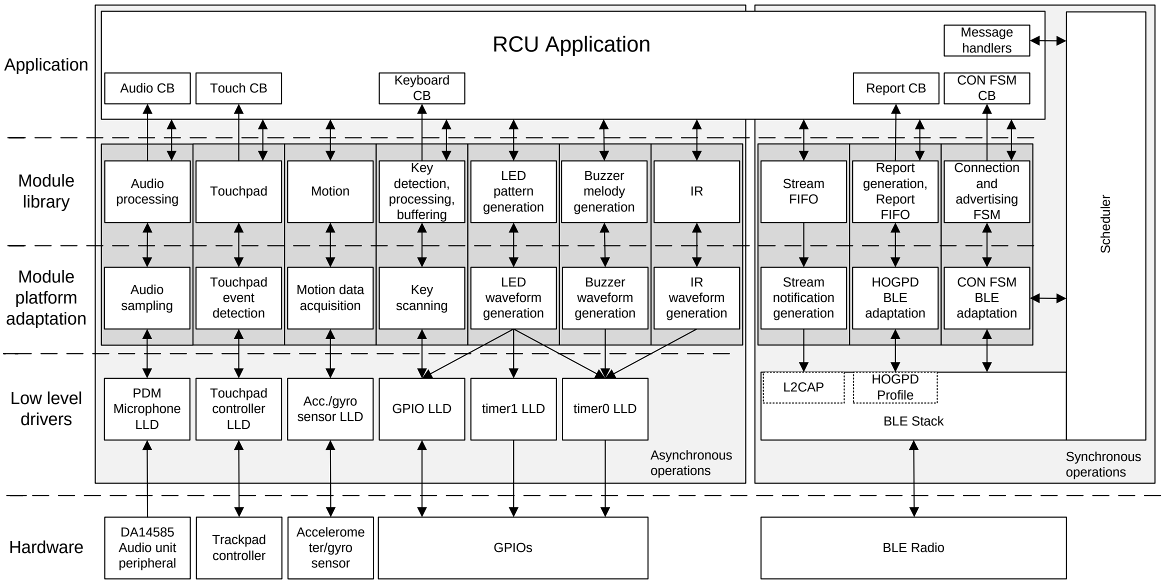


Figure 4: Software Architecture Diagram

The RCU software consists of:

- **Low Level Drivers** (LLDs) for accessing the hardware peripherals
- **Modules**, each implementing a specific function
- **RCU application**, implementing the RCU functionality using the API provided by the modules and the DA14585 SDK.

The RCU software consists of a synchronous and an asynchronous part.

The **synchronous part** is controlled by a scheduler, which is used for scheduling kernel tasks, processing events and delivering messages to the tasks. The scheduler is called from the main application loop. The RCU application runs as a task and registers message handlers to the kernel. The scheduler calls these message handlers to deliver messages to the application. Some of these messages may be handled by the modules, the rest of them are handled by the RCU application. The RCU application can also send messages to other tasks. In this case, the application must allow the main loop to run and call the scheduler to process the messages.

The **asynchronous part** is interrupt driven. Interrupt service routines only perform time critical operations, in order to keep the CPU in the interrupt context for as short a time as possible. The main processing is performed in the main loop. Kernel messages generated by asynchronous operations can be synchronized with the kernel at this point.

The following **modules** are included in the RCU reference design software:

- **Audio:** Audio samples are captured from a PDM microphone using the audio unit peripheral of DA14585.
- **Touchpad:** Action events are captured using either an Azoteq IQS572 trackpad controller or an Azoteq IQS263 slider and scroll wheel controller.
- **Motion:** Accelerometer and gyroscope data are capture using a BMI160 sensor.
- **Keyboard:** Key actions are detected using a key matrix connected to GPIO pins.
- **IR:** InfraRed (IR) waveforms are generated using an IR LED connected to a GPIO to transmit IR codes.

 © 2022 Renesas Electronics

- **LED indicators:** On/off and ramp patterns are generated to drive LEDs connected to GPIO pins.
- **Buzzer:** Melodies can be played using a magnetic buzzer connected to a GPIO pin.
- **Connection FSM:** Handles the connection with the BLE host (advertising, connection, disconnection, pairing, encryption, etc.).
- **BLE stream:** Data are streamed to the BLE host using notifications.
- **HID report:** Human Interface Device (HID) reports are generated and sent to the BLE host using the HID Over GATT profile (HOGP).

Each module can be independently included or excluded according to the needs of the application. The user RCU application uses the included modules to implement the application functionality. Since modules are isolated from each other, information cannot flow across modules. This is the task of the user application.

Example:

In a typical RCU use case the user presses the Push-To-Talk (PTT) key on the RCU to stream audio captured by the microphone to the BLE host. Three modules are used by the application to perform the operations required by this case: **Keyboard**, **Audio** and **BLE Stream**.

The **Keyboard Module** detects the key press and calls its callback function to report the key press to the application. The application checks this key and, since this is the PTT key, it commands the audio module to start capturing data.

The **Audio Module** starts the audio sampling. Audio data are buffered in the audio module's FIFO. Its callback is called whenever a certain amount of data is available in the FIFO.

The application checks if there is enough free space in the FIFO of the **BLE Stream Module**. In this case, the application gets a pointer to the free space and passes it to the encoder of the Audio Module. The encoder encodes the available data in the audio FIFO and writes the encoded data to the FIFO of the BLE Stream. The application then checks if there are data in the BLE Stream FIFO, and if so, it asks the BLE Stream Module to send the data to the BLE stack.

7.2 Low Level Drivers (LLDs)

LLDs are used for accessing hardware peripherals. Some of the required LLDs are provided by the SDK. New LLDs have been included in the scope of the RCU application for accessing peripherals, such as motion sensor or touchpad controllers. These LLDs may use other LLDs (such as I2C or SPI bus LLDs) provided by the SDK.

7.3 Modules

Each module performs a very specific function. It provides an API that enables the initialization and control of the module, as well as the data exchange between the module and the application. The API provides for example functions to initialize, start or stop the module, get data from the module or provide data to the module.

A callback function can optionally be registered to the module, to allow notification of the application, whenever there is an event to the module. This callback function can be called, for example, when the module can provide new data to the application, or when there is a change in the module's state.

Each module may use one or more LLDs to access the hardware peripherals, or have direct access to the BLE stack.

Each module consists of the following parts:

- **Module library** consisting of a set of functions implementing the functionality of the module. This part is platform independent and can be shared across projects, even for different target processors.
- **Platform adaptation layer** that is used for accessing the platform-dependent APIs. This layer consists of a set of functions that enables the module library to access the platform-specific hardware. The layer lies between the module library and the LLDs or the BLE stack.

© 2022 Renesas Electronics

The **module library** consists of files named `app_[module_name].c` and `app_[module_name].h` and implements the module API. The module library may:

- Have a function that is called from the main application's loop, for executing module specific tasks, such as running state machines.
- Use its callback to notify the application about new data or state changes.
- Register its own callbacks with LLDs, to be triggered by interrupts.

The functions of the module library call functions from the platform adaptation layer to access the LLDs and the BLE stack.

The **platform adaptation layer** consists of files named `port_[module_name].c` and `port_[module_name].h` and adapts the module library function calls to the API provided by the LLDs and the BLE stack. These APIs are platform-dependent. They depend on the platform processor and on the external hardware peripherals.

Each module has only one module library. However, it may have one or more platform adaptation layers to support multiple processors or hardware peripherals. In this case, the names of the platform adaptation files may have an additional postfix to define the hardware peripheral that they support. The module library uses a structure of pointers to platform adaptation functions, which can be modified by the application to override the default implementation.

Example:

In the **Motion Module** a Finite State Machine (FSM) is implemented in the **motion library** for handling the motion data acquisition in `file app_motion.c`. The motion library provides a function in its API (which is declared in file `app_motion.h`) that must be call periodically from the application, in order to operate the FSM.

The motion library uses the **motion platform adaptation layer** (implemented in file `port_motion.c`) to request motion data from the sensor. The motion adaptation layer adapts this call to a call in the BMI160 sensor LLD, and returns the motion data to the motion library. When a different sensor is to be used, a second motion adaptation layer can be used, that adapts the calls from the motion library to the API of LLD of the new sensor.

7.4 Including Modules in the Project

A module can be included in the project by defining the appropriate symbol in file `user_config.h`, as described in the corresponding module API header file. For example, define `HAS_AUDIO` in the file `user_config.h` to include the audio module.

During execution of the application a module must perform are six basic operations:

- **Module initialization.** This can be done by calling the module's initialization function in function `user_on_init()`.
- **Declaration of the GPIOs used by the module.** This can be done by calling the module's GPIO reservation function from within function `GPIO_reservations()`.
- **Initialization of the GPIOs used by the module.** Pins must be reinitialized every time the system wakes up from sleep. This can be done by calling the module's GPIO initialization function in function `set_pad_functions()`.
- **Execution of asynchronous operations** (e.g. state machine execution) while the BLE core is powered. This can be done by calling the appropriate module function in the function `user_on_ble_powered()`. This function must return true, if the system must remain powered, in order to process kernel messages. It can also return true to force the system to call this function again, in order to continue processing after running the main loop once.
- **Execution of asynchronous operations** (e.g. audio encoding) while the system is powered. This can be done by calling the appropriate module function in the function `user_on_system_powered()`. This function must return `APP_KEEP_POWERED`, if the system must remain powered, in order to continue processing after running the main loop once. It can also return `APP_BLE_WAKEUP`, in order to force the BLE core to wake up.

 © 2022 Renesas Electronics

- **Execution of operations upon BLE disconnection** (e.g. stop the BLE stream). This can be done by calling the appropriate module function from within function `user_on_disconnect()`.

To facilitate module integration within the application, the array `user_module_config[]` is defined in the file `user_modules_config.h`. When a module is included in the application, a structure of type `module_config_t` can be added in the `user_module_config[]` array. This structure has the following members:

- `init`: pointer to the function that initializes the module.
- `on_disconnect`: pointer to the module's function that executes actions upon BLE disconnection.
- `on_ble_powered`: pointer to the module's function that executes asynchronous operations while the BLE core is powered.
- `on_system_powered`: pointer to the module's function that executes asynchronous operations while the system is powered.
- `pins_config`: pointer to a structure that contains the module's GPIO configuration. This structure is used for GPIO reservation. If the `init_gpios` member is NULL, the structure is also used for GPIO initialization after the system wakes up.
- `init_gpios`: pointer to the module's function that initializes the GPIOs after the system wakes up. This function is used when a special initialization must be performed, that cannot be declared in file `pins_config.h`.

The application uses the `user_module_config[]` array to call the module functions provided, to reserve and initialize the GPIOs when required.

If the user application needs to perform additional actions, it can override the module's default behavior by defining a user defined function in the `user_module_config[]` array and then call the corresponding module's function from within this user defined function.

7.5 User RCU Application

The user RCU application implements the RCU functionality using the required modules and SDK API. The application establishes the following connections between the modules and the SDK to achieve the desired information flow:

- **Keyboard, Touchpad — HID Report**, to generate and send HID keyboard reports to the host upon key or touchpad events.
- **Keyboard — Audio**, to control audio sampling when the PTT key is pressed.
- **Keyboard —Motion**, to control motion activation when the motion key is pressed.
- **Keyboard — IR**, to transmit IR waveforms when a key is pressed.
- **Keyboard, Touchpad — Connection FSM**, to start advertising on a key or touchpad event.
- **Audio — BLE Stream**, to stream the encoded data to the BLE host.
- **Motion, Trackpad — BLE Stack,** to send data from these modules to the BLE stack using the SDK API.

The user RCU application is implemented in four different files:

- `user_rcu.c`
  - Implements SDK callback functions.
  - Implements main loop functions.
  - Handles BLE connection using the **Connection FSM Module**.
  - Handles BLE services.
  - Handles SUOTA operations.
  - Handles system sleep.
  - Creates and sends HID keyboard reports using the **HID Report Module**.

© 2022 Renesas Electronics

- `user_rcu_kbd.c`
  - Handles the keyboard using the **Keyboard Module**.
  - Detects audio key presses and calls the appropriate functions of `user_rcu_audio.c`.
  - Detects motion key presses and calls the appropriate functions of `user_rcu_motion.c`.
- `user_rcu_audio.c`
  - Handles audio sampling and encoding using the **Audio Module**.
  - Sends encoded audio data to the BLE host using the **BLE Stream Module**.
  - Handles audio control commands from the BLE host over the Dialog Audio Service or vendor-specific HID reports.
  - Sends control and configuration notifications to the BLE host over the custom audio profile or vendor-specific HID reports.
  - Optionally adapts the audio encoder parameters automatically to the available BLE channel bandwidth.
  - Calculates the optimal audio packet size.
- `user_rcu_motion.c`
  - Handles motion using the **Motion Module**.
  - Handles a trackpad or slider using the **Touchpad Module**.
  - Creates and sends HID mouse reports on trackpad track events.
  - Creates HID keyboard reports on trackpad or slider events using the **HID Report Module**.

The audio packet size is the number of bytes of encoded audio sent in one ATT packet. It is calculated dynamically according to the current connection interval, the `ATT_MTU`, the BLE packet length and the maximum number of packets per connection event. The packet size must be large enough to achieve the required bandwidth, achieved by reducing the L2CAP header and ATT header overhead. Its maximum size is (`ATT_MTU - 3`) bytes, since notifications are used for audio data transmission. When Data packet Length Extension (DLE) is used, the packet length can be larger than the audio packet size, so the packet size must be kept as small as possible, in order to reduce retransmission time when packets are dropped due to communication errors.

There is no dependency between audio sampling size, audio sample buffer size, number of samples encoded at a time and audio packet size. As a result, each of these parameters can be fine-tuned independently. The audio sampling size can be adjusted to optimize the interrupt frequency. The audio sample buffer size depends on the system delay from the time data samples are available to the time they are encoded. This buffer can be quite small, when the system is very responsive. The number of samples encoded at a time determines how long the CPU is occupied with the audio encoding. A higher number of samples blocks the main loop for a longer time, and requires a larger audio sample buffer size to hold the encoder input data.

In-band audio commands are supported. The application can add custom commands in-band with the encoded audio data. This is achieved using a user-defined escape character. The application can insert the escape character followed by one byte for the command and its parameters. Bits 7 to 4 contain the command opcode, while bits 3 to 0 contain the optional command parameter. The format of the commands is provided in Table 8.

Table 8: Audio Stream In-Band Commands

Command Description	Command Code	Parameter	Command Byte
Audio stream reset	0x00	-	0x00
Set IMA ADPCM mode	0x10	0x00: 64 kbit/s	0x10
		0x01: 48 kbit/s	0x11
		0x02: 32 kbit/s	0x12
		0x03: 24 kbit/s	0x13

© 2022 Renesas Electronics

Byte stuffing is used for audio data bytes that have the same value as the escape character. In that case the escape character is transmitted twice.

When audio stream begins, an `audio stream reset` command is transmitted first, followed by a `set IMA ADPCM mode` command. The ADPCM mode can be changed at any time during audio streaming by inserting a new `set IMA ADPCM mode` command in the stream data. The **Audio Module** supports in-band control characters as described in Section 8.2.1.

Option `STREAM_FIFO_NUM_OF_HIGH_PRIORITY_BYTES` can be set as described in Section 8.3.2 to ensure that there is always space available in the BLE stream for in-band commands. In that case, the function `app_stream_fifo_get_priority_write_dataptr()` can be used to insert in-band commands into the audio stream.

7.6 Configuration Files

Configuration files are used for configuring the RCU software without having to modify the source code. There are two different sets of configuration files:

- **User configuration files** located in project group `user_config`.
- **Module configuration files** located in project group `modules_config`.

Configuration files belong to the project. Each project target can have its own configuration file set. Configuration files are stored in folder `\src\config` under the project folder. These files are used for all project targets.

A new project target can be added in folder `\src\config\variants`, in its own subfolder, containing the configuration files that differ from the main project target. This subfolder must be placed before `\src\config` in the target's options C++ include paths. The compiler will first check the subfolder in the folder `\src\config\variants` for a configuration file. When it is not found, the configuration file located in `\src\config` will be used.

The configuration files used are statically included at compile time when the project is built, in order to optimize the code size of the application.

7.6.1 User Configuration Files

User configuration files are used to configure the target system and the user application. These are the standard SDK configuration files with the following modifications and additions:

1. In file `da1458x_config_advanced.h`:
   a. `CFG_NB_PRF` has been added, to set the maximum number of BLE profiles.
   b. `DB_HEAP_SZ`, `MSG_HEAP_SZ` and `NON_RET_HEAP_SZ` have been defined to optimize the memory used by the BLE stack heaps.
   c. The maximum supported TX and RX data packet lengths have been reduced to optimize the memory used by the BLE stack.
2. In file `da1458x_config_basic.h`:
   a. `CFG_APP_SECURITY` has been undefined. The **Connection FSM Module** is used instead.
3. In file `user_callback_config.h`:
   a. Structure `user_app_callback` has been modified to work with the **Connection FSM Module**.
   b. Functions have been added in structure `app_main_loop_callbacks`.
   c. Device information service, battery service, and HOGP have been added in array `user_prf_funcs[]`.
4. In file `user_config.h`:
   a. Default configuration has been adapted to the RCU application requirements.
   b. Symbols have been added for including modules.
   c. Configuration for debugging options has been added.

 © 2022 Renesas Electronics

5. In file `user_modules_config.h`:
   a. Modules not used in the RCU application have been excluded.
   b. Table `user_module_config` has been added for including modules as described in Section 7.4.
6. In file `user_periph_setup.h`:
   a. The UART pin configuration has been updated.
   b. The SPI and I2C bus pin configurations have been added.
7. In file `user_profiles_config.h`:
   a. HOGP and custom1 profile have been added.
   b. DIS information has been updated.
8. File `user_hogpd_config.h` has been added for configuring the HOGP profile.
9. File `user_pwr_mgr_config.h` has been added for configuring the power manager.

7.6.2 Module Configuration Files

Each module can be configured using a configuration file named `app_[module_name]_config.h`, without having to modify the source code of the module. This file can be used to:

- Enable or disable various features of the module (e.g. multi-bonding, in-band audio commands),
- Configure operational parameters of the module (e.g. key debouncing time, audio sampling rate),
- Set the configuration of the GPIO pins used by the module's platform adaptation layer,
- Initialize the structure of pointers to platform adaptation layer functions.

The link between the module library and the platform adaptation layer is defined in the configuration file, allowing the application to define different platform adaptation layers according to the system configuration.

The pin configuration is defined using an array of elements of type `pin_type_t`. Each array element configures one pin. Structure `pin_type_t` has the following members:

`port`:	The GPIO port of the pin.
`pin`:	The GPIO pin of the pin.
`high`:	Pin inactive level. Set to 1 for active low, set to 0 for active high.
`mode_function`:	Sets the mode (`INPUT`, `INPUT_PULLUP`, `INPUT_PULLDOWN` or `OUTPUT` as defined in `GPIO_PUPD` enumeration) and the function (as defined in `GPIO_FUNCTION` enumeration).

An enumeration could also be used to define the pin names. Let us examine, for example, the definition of the audio pins. The following enumeration is used to define the audio pin names:

```
enum audio_pin_ids {
    AUDIO_CLK_PIN,
    AUDIO_DATA_PIN,
};
```

The pins are active high, connected to the PDM peripheral, the clock pin is an output and the data pin is an input. The pin configuration array then is the following:

```
static const pin_type_t app_audio_pins[] = {
    [AUDIO_CLK_PIN]  = { .port = GPIO_PORT_1,
                         .pin = GPIO_PIN_1,
                         .high = 0,
                         .mode_function = OUTPUT | PID_PDM_CLK },
    [AUDIO_DATA_PIN] = { .port = GPIO_PORT_1,
                         .pin = GPIO_PIN_0,
                         .high = 0,
                         .mode_function = INPUT  | PID_PDM_DATA },
};
```


© 2022 Renesas Electronics

7.7 Project Folder Structure

The RCU reference design code software is organized in three folders:

- `SDK_585`: This folder contains the DA14585 SDK.
- `Modules_library`: This folder contains the modules library and the platform adaptation layer for DA14585. Each module is stored in a separate subfolder. The platform adaptation layer files for DA14585 are stored in subfolder `\port_58x`.
- `projects\target_apps\rcu`: This folder contains the RCU application. It contains the following subfolders:
  - `src`: Non DA14585-specific application code.
  - `src\drivers`: Non DA14585-specific LLDs.
  - `src\platform`: Non DA14585-specific code.
  - `rcu_585\src\profiles`: DA14585 BLE profiles that are not included in the SDK.
  - `rcu_585\src\config`: Reference design project configuration files.
  - `rcu_585\src\config\variants\ProDK`: Project configuration files for the ProDK target.
  - `rcu_585\Keil_5`: Keil project folder.
  - `rcu_585\Keil_5\out_585`: Reference design project output folder.
  - `rcu_585\Keil_5\out_585_ProDK`: Project output folder for the ProDK target.

 © 2022 Renesas Electronics

8 Modules

8.1 Keyboard Module

8.1.1 Description

The Keyboard Module enables the detection and processing of key actions of a keyboard matrix. The block diagram of the Keyboard Module is depicted in Figure 5.

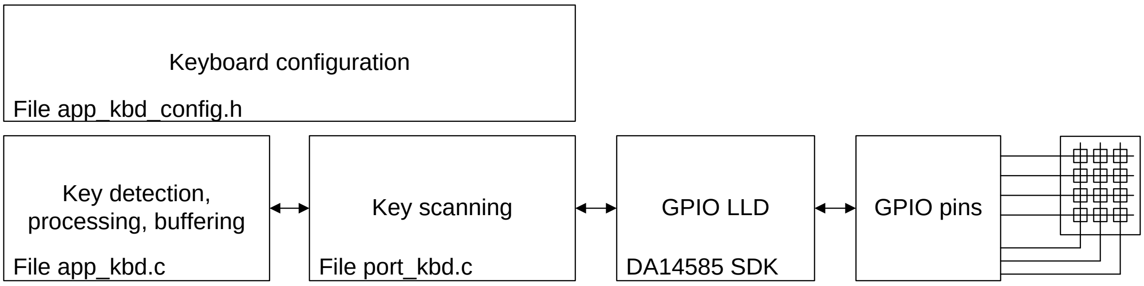


Figure 5: Keyboard Module Block Diagram

Features

- Keys organized in a 16 x 16 matrix
- Simultaneous key-presses
- Key debouncing with programmable press and release debouncing times
- Key de-ghosting
- Key event buffering
- Custom keys for application specific tasks
- Passcode mode

Keyboard Scanning

The keyboard matrix is connected to GPIO pins. Row pins are configured as outputs and column pins are configured as inputs. The internal pull-up resistor of the DA14585 GPIO pins is used to set the input to a high state when no key is pressed. The keyboard matrix is scanned one row at a time. The GPIO LLD is used to set the corresponding pin to a low state and read the status of the column pins. One scan cycle is completed when all rows have been scanned.

The keyboard scanning algorithm is implemented in file `port_kbd.c`. Implementation details can be found in Ref. [5].

Key event processing is implemented in file `app_kbd.c`. Status (pressed or release) for all keys in the key matrix is provided by file `port_kbd.c`. Keys are debounced to detect key press or release events. Key press events are de-ghosted. Valid key press or release events are recorded in the key buffer.

The application can poll the key buffer to get the code of a key event as defined in the `kbd_keymap` table. Multiple codes can be defined for each key. The user can use one or more **Fn** function keys to select the appropriate key code. The **Fn** key can be used as a **modifier key**, pressed simultaneously with the normal key, or it can be used as an **Fn lock** key. In the latter case, a second set of key codes is used for all keys until the **Fn lock** key is pressed again to unlock the **Fn** state.

Custom Key Handling

The Keyboard Module enables special handling of specific keys that are defined as custom keys. These keys are scanned, debounced and de-ghosted as normal keys but the generated key events

© 2022 Renesas Electronics

are not inserted in the key buffer. The application is notified directly to handle the key event and to perform custom actions. Key combinations can also be used to generate custom key events.

Passcode Mode

Passcode mode enables the use of the keyboard for entering a numeric passcode. The application can set the Keyboard Module in Passcode mode by calling the function `app_kbd_start_passcode()`. The Keyboard Module assembles the passcode after each numeric key press and notifies the application when the user presses the **Enter** key (keyboard: key code 0x28, keypad: key code 0x58). A custom enter key can also be defined in the configuration file of the Keyboard Module.

Callback Function

The application can register a callback function to receive notifications from the Keyboard Module. The following notifications are supported:

- **Key action notification**. A key has been pressed or released. Custom keys are not included. The key event can be acquired from the key buffer.
- **Custom key notification**. A custom key has been pressed or released.
- **Fn lock activated notification**. This is only available when the **Fn lock** feature is enabled.
- **Fn lock deactivated notification**. This is only available when the **Fn lock** feature is enabled.
- **Passcode entered notification**. In Passcode mode the user has entered the code followed by the Enter key. The passcode can be acquired by calling the `app_kbd_get_passcode()` function.

The Keyboard Module is enabled by defining the symbol `HAS_KBD` in file `user_config.h`.

Files `app_kbd.c`, `port_kbd.c`, `app_kbd_config.h`, `app_kbd_matrix.h` and `app_kbd_scan_matrix.h` must be included in the project.

8.1.2 Configuration

The following **keyboard parameters** are defined in file `app_kbd_config.h`:

- `KBD_KEYCODE_BUFFER_SIZE` sets the size of the key buffer.
- `KBD_DEBOUNCE_BUFFER_SIZE` sets the number of keys that can be debounced at the same time.
- Define `ALTERNATIVE_SCAN_TIMES_ON` to use different full and partial scan periods. Full scan cycles are used to detect any key press. Partial scan cycles are used to scan for key events only in the rows that have a key pressed.

Additional parameters are defined in the `kbd_params structure`:

- Set `scan_always_active` to `true` to force continuous keyboard scanning. The system will be forced to stay active in this case.
- Set `has_fn_lock` to `true` to have the **Fn** key functioning as **Fn lock** key. When set to `false` the **Fn** key will function as a key modifier.
- Set `passcode_enter_key` to the code of a custom key used to complete the passcode entry, in addition to the Enter keys of the keyboard and the keypad.
- Set `notify_callback` to the function that is to be called by the Keyboard Module to send a notification to the application.
- Set `key_detect_callback` to the function that is to be called by the Keyboard Module to report a key press or release. Only the key status, row and column are reported.
- Set `row_scan_time_in_us` to the delay (in μs) from row activation until the column inputs are sampled.
- Set `full_scan_cycle_in_us` to the duration (in μs) of a full scan cycle. All rows in are scanned in a full scan cycle.

 © 2022 Renesas Electronics

- Set partial_scan_cycle_in_us to the duration (in µs) of a partial scan cycle. Only rows having key activity are scanned. The remaining idle rows are monitored for key activity. When a key of an idle row is pressed a full scan cycle is initiated.
- Set press_debounce_counter_in_us to the duration (in µs) of key press debouncing. This time must be a multiple of the partial scan cycle duration.
- Set release_debounce_counter_in_us to the time (in µs) for key release debouncing. This time must be a multiple of the partial scan cycle duration.

The **key matrix pin configuration** is defined in file app_kbd_scan_matrix.h:

- KBD_NR_COLUMN_INPUTS sets the total number of key matrix column inputs.
- KBD_NR_ROW_OUTPUTS sets the total number of key matrix row outputs.
- COLUMN_INPUT_x_PORT and COLUMN_INPUT_x_PIN define the pin to which column input x is connected. x is in the range of 0 to (KBD_NR_COLUMN_INPUTS-1).
- ROW_OUTPUT_y_PORT and ROW_OUTPUT_y_PIN define the pin to which column input y is connected. y is in the range of 0 to (KBD_NR_ROW_OUTPUTS-1).

Matrix key assignment is defined in file app_kbd_matrix.h.

Key codes are defined in table kbd_keymap. Multiple key sets can be defined in this table. The first set is used by default. The user can switch to another set using **Fn** as **modifier key**. Key codes can be one of the following:

- Regular key the value of which is in the range of 1 to 255.
- Modifier key (such as **LEFT/RIGHT SHIFT**, **ALT**, **CTRL**, **GUI**), the value of which is in the range of 1 to 255. Macro KBD_MODIFIER_KEY can be used to generate the key code. Modifier keys for HID normal keyboard reports can have one of the following values:

  LEFT CTRL: 0x01
  LEFT SHIFT: 0x02
  LEFT ALT: 0x04
  LEFT GUI: 0x08 (e.g. the left Windows key)
  RIGHT CTRL: 0x10
  RIGHT SHIFT: 0x20
  RIGHT ALT: 0x40
  RIGHT GUI: 0x80 (e.g. the right Windows key)
- Special key, the value of which is in the range of 1 to 127. Macro KBD_SPECIAL_KEY can be used to generate the key code.
- Fn modifier key. This key is used to switch to another key set. Macro KBD_FN_LOCK_KEY can be used to generate the key code.
- Custom key, the value of which is in the range of 1 to 15. Macro KBD_CUSTOM_KEY can be used to generate the key code.
- KEY_UNUSED when the key in the key matrix is not used.
- K_CODE when there is no key code for the specific key but the key must be examined for ghosting.

Key combinations can be used to generate custom key events when MULTI_KEY_COMBINATIONS_ON is defined. The maximum number of keys used for a key combination is defined using the symbol MULTI_KEY_NUM_OF_KEYS. Key combinations are defined in table multi_key_combinations, which specifies the row and column of each key in a key combination. If a key is not used (e.g. two keys are needed when MULTI_KEY_NUM_OF_KEYS is 3) its row and column can be set to MULTI_KEY_NOT_USED. Macro KBD_KEY_TO_NOTIFY_CODE can be used to generate the appropriate notification code from the custom key code.

© 2022 Renesas Electronics

8.1.3 Design Considerations

Keyboard scanning is more efficient when the matrix is defined as 3 (rows) x 4 (columns), instead of 4 x 3. All matrix configuration settings then have to be transposed. A 4 x 3 matrix configuration has been selected here, because it is easier to match the key configuration to the physical keyboard.

The `kbd_params.row_scan_time_in_us` has been set to 150 μs to allow the column input signal to settle when scanning a new row. This time can be fine-tuned if external pull-up resistors are used for the column pins. When this value is set too low, the column inputs may not have settled when being sampled by the software.

8.2 Audio Module

8.2.1 Description

The Audio Module enables the processing of the audio signal captured by a PDM microphone connected to the DA14585 audio unit. The block diagram of the Audio Module is depicted in Figure 6.

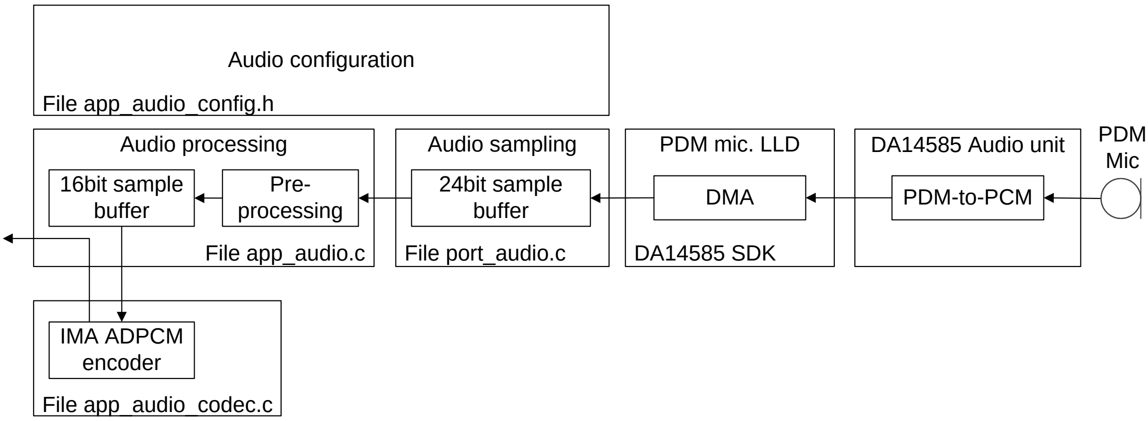


Figure 6: Audio Module Block Diagram

Sample Processing

The audio unit converts the PDM bit stream to 24-bit PCM samples. The 24-bit samples are stored left-aligned in 32-bit words, with bits 0 to 7 set to 0. The PDM microphone LLD transfers the PCM data from the audio unit output to a circular buffer using a DMA channel of the DMA controller. An interrupt is generated whenever a certain amount of samples are transferred to the circular buffer.

The 24-bit samples are preprocessed and then transferred to a smaller 16-bit sample buffer. Preprocessing may include a DC blocking filter to remove the DC component of the signal and it converts the 24-bit samples to 16-bit samples. It may also include dropping a certain amount of samples at the beginning of the audio stream to remove clicking and transient effects.

Finally, an IMA ADPCM encoder is used to encode the PCM data and deliver them to the buffer provided by the application.

Buffer Handling

The application allocates a buffer and calls `function app_audio_encode()`, passing a pointer to this buffer and the buffer size as parameters. This function performs the audio signal preprocessing and encoding, placing the encoded data in the buffer allocated by the application, and returns the number of encoded data bytes placed in the buffer. The return value may be smaller than the size of the buffer, or even 0 when there are not enough audio samples available for processing.

Optionally, the application can register a callback function, that is called in response to the interrupt generated by the DMA controller, when new data are available in the 24-bit sample buffer.

 © 2022 Renesas Electronics

In-Band Commands

The IMA ADPCM encoder also supports in-band commands. The application can add custom commands in-band with the encoded audio data. This is achieved by using a user defined escape character and byte stuffing technique, as described in Section 7.5.

During data encoding, the encoder checks whether the encoded data contains a value equal to the escape character used for in-band commands. In that case, the encoder performs byte stuffing by sending the escape character value twice.

On the host side, the decoder must detect escape characters and remove byte stuffing, when it detects two consecutive escape characters. Then it must process in-band commands, when one escape character is followed by a byte that is not equal to the escape character.

Encoding

The Audio Module can be dynamically configured to use either an 8 kHz or 16 kHz sampling rate. Adapting the sampling rate is performed by the DA14585 audio unit. Furthermore, the IMA ADPCM encoder compression can be dynamically configured to use 3 or 4 bits per sample. The resulting compressed audio bit rates are listed in Table 9.

Table 9: Compressed Audio Bit Rate

Bit Rate	Bits per Sample	Sampling Rate	IMA ADPCM Compression
64 kbit/s	16	16 kHz	4 bits per sample
48 kbit/s	16	16 kHz	3 bits per sample
32 kbit/s	16	8 kHz	4 bits per sample
24 kbit/s	16	8 kHz	3 bits per sample

The Audio Module is enabled by defining the symbol `HAS_AUDIO` in file `user_config.h`.

Files `app_audio.c`, `app_audio_codec.c`, `port_audio.c` and `app_audio_config.h` must be included in the project.

8.2.2 Configuration

The Audio Module configuration is specified in file `app_audio_config.h`. The following configuration options can be defined:

- Define `AUDIO_CONTROL_ESCAPE_VALUE` to the escape character value used for in-band commands. When `AUDIO_CONTROL_ESCAPE_VALUE` is not defined, in-band commands cannot be used and the encoder will not perform byte stuffing.
- Define `CFG_AUDIO_ADAPTIVE_RATE` to allow dynamic IMA ADPCM mode changing. When `CFG_AUDIO_ADAPTIVE_RATE` is not defined, the mode defined in `ADPCM_DEFAULT_MODE` will be used.
- Define `CFG_AUDIO_DC_BLOCK` to enable the DC blocking filter during audio signal preprocessing. The DC blocking filter must always be enabled when the DA14585 audio unit is used, because the DC component of the output signal is quite large compared to the signal amplitude.
- Define `CFG_AUDIO_CLICK_STARTUP_CLEAN` to drop a certain amount of samples at the beginning of the audio stream to remove clicking and transient effects.
- Define `CFG_AUDIO_CONFIGURABLE_SAMPLING_RATE` when the hardware audio peripheral can perform audio sampling at 8 kHz and 16 kHz. When only 16 kHz sampling is supported by the hardware peripheral, software down-sampling must be performed to use 8 kHz sampling modes.
- Define `CFG_AUDIO_USE_32BIT_SAMPLING` to support the 32-bit sampling output of the DA14585 audio unit.
- When `CFG_AUDIO_USE_32BIT_SAMPLING` is defined, `AUDIO_SAMPLING_OFFSET` defines the number of bits that the samples are shifted to the right, when they are converted into 16-bit samples. The output data of the DA14585 audio unit are located at bits 31 to 8, so `AUDIO_SAMPLING_OFFSET` must be defined as 8. Only 16 bits of the samples will be used, so the eight most significant bits

 © 2022 Renesas Electronics

will be discarded. When a number of least significant bits must be discarded to attenuate the signal, this number must be added to the AUDIO_SAMPLING_OFFSET value.

- Define AUDIO_BUFFER_NR_SLOTS to specify the size (in slots) of the 24-bit sample buffer and AUDIO_NR_SAMP_PER_SLOT to specify the number of samples per slot. The single buffer contains (AUDIO_NR_SAMP_PER_SLOT * AUDIO_BUFFER_NR_SLOTS) samples. An interrupt is generated when DMA fills a slot with data read from the audio unit. The reason why it is treated as a number of slots is to simplify buffer handling.
- AUDIO_SBUF_SIZE defines the size of the 16-bit sample buffer.
- AUDIO_NOTIFICATION_CB defines the function called when AUDIO_NR_SAMP_PER_SLOT samples have been transferred to the 24-bit sample buffer. This function is called in interrupt context.

The following debugging configuration options can be used during evaluation and testing of the Audio Module:

- Define CFG_AUDIO_EMULATE_PDM_MIC to emulate microphone input using a software generated waveform. A sine waveform is used by default.
- Define CFG_AUDIO_EMULATE_PDM_MIC_TRIANGULAR to emulate microphone input using a software generated triangular waveform.
- Define CFG_AUDIO_UART_DEBUG to print to the debug console special characters indicating the audio process progress.
- Define USE_AUDIO_MARK to use a GPIO pin to mark the audio data sampling. The GPIO pin level is high while audio data are transferred from the audio peripheral to the buffer. The GPIO pin is defined by AUDIO_MARK_PORT and AUDIO_MARK_PIN.
- Define CFG_AUDIO_DEBUG_ENC_AUDIO_TO_UART to send encoded audio data to the UART port.
- Define CFG_AUDIO_DEBUG_PDM_TO_UART to send PDM microphone samples to the UART port.

8.3 BLE Stream Module

8.3.1 Description

The BLE Stream Module enables data streaming from the BLE peripheral to the BLE host. This module achieves a high data throughput by maximizing the number of packets sent in each connection event and minimizing the processing time required.

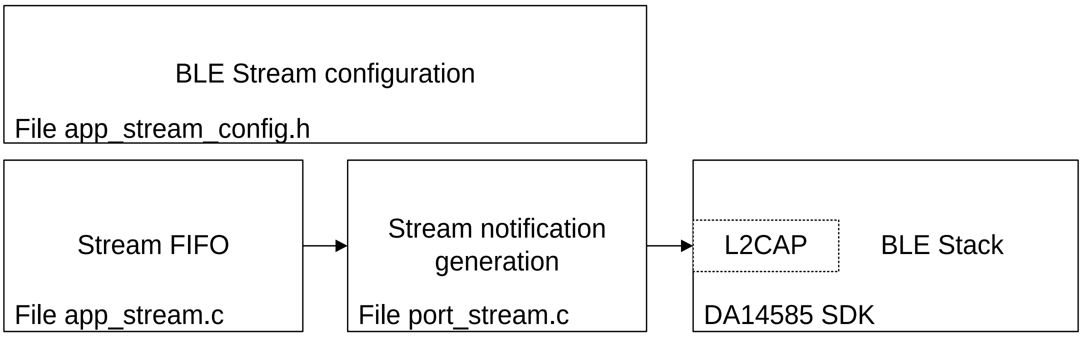


Figure 7: BLE Stream Module Block Diagram

Data are streamed to the host using GATT notifications. To reduce the processing time, the BLE Stream Module - instead of using the corresponding GATT profile - delivers the notification packets directly to L2CAP using the appropriate handle, which is acquired when the GATT server database is created.

The BLE Stream Module uses a FIFO to buffer the data before sending them to L2CAP. Data from this FIFO are copied to the L2CAP buffers before each connection event. The FIFO can be packet based or non-packet based. In both cases the packet size cannot exceed (ATT MTU – 3) bytes.

 © 2022 Renesas Electronics

- A **packet based FIFO** consists of a list of data packets. The application can place a packet in the FIFO. The BLE stream will transfer this packet to the L2CAP buffers. The packet size of a packet based FIFO can be predefined or variable.
- A **non-packet based FIFO** consists of a circular buffer. The application can add data to the FIFO without any constraint on the data length. The BLE stream creates packets of data and sends them to L2CAP. The size of the packets is configurable at run time.

The BLE Stream Module is enabled by defining symbol `HAS_BLE_STREAM` in the file `user_config.h`.

Files `app_stream.c`, `port_stream.c` and `app_stream_config.h` must be included in the project.

 © 2022 Renesas Electronics

8.3.2 Configuration

The BLE Stream Module configuration is specified in file `app_stream_config.h`.

- When `CFG_APP_STREAM_PACKET_BASED` is defined, a packet based stream FIFO is used. In this case:
    - `APP_STREAM_MAX_PACKET_FIFO_LEN` sets the maximum number of packets in the FIFO.
    - When `CFG_APP_STREAM_FIFO_PREDEFINED` is defined, the size of each packet in the FIFO is predefined to the value of `APP_STREAM_PACKET_SIZE`.
    - When `CFG_APP_STREAM_FIFO_PREDEFINED` is not defined, packets of variable length can be added to the FIFO. The memory for each packet is allocated from a buffer, the size of which is set to `APP_STREAM_FIFO_SIZE`.
- When `CFG_APP_STREAM_PACKET_BASED` is not defined, a non-packet based stream FIFO is used. In this case:
    - `APP_STREAM_FIFO_SIZE` sets the size of the FIFO.
    - `STREAM_FIFO_NUM_OF_HIGH_PRIORITY_BYTES` defines the number of bytes reserved for high priority data, such as audio in-band commands. These bytes are never used when adding normal data to the FIFO. Function `app_stream_fifo_get_priority_write_dataptr()` can be used to add high priority data to the FIFO.

8.3.3 Design Considerations

Packet based FIFO implementation is simple. Memory used for FIFO can be split in many segments allowing easier handling by the linker. However, the packet size is fixed and must be considered by the application when adding packets to the FIFO.

Non-packet based FIFO implementation is more complex, but allows decoupling of the BLE packet size from the data added to the FIFO. The packet size can be changed dynamically without the need to reconstruct the data packets.

The number of L2CAP buffers must be carefully selected to allow the storage of an adequate number of data packets, to take advantage of the maximum number of packets per connection event. This number can be set by defining `CFG_NUM_OF_BLE_TX_BUFFERS` in file `da1458x_config_advanced.h`.

For example, when the packet length is 27 and `ATT MTU` is 77, three link layer packets (one ATT packet) are needed to send 74 bytes (payload: 20, 27, 27 bytes). When the number of link layer buffers is five, only one ATT packet can be placed in the link layer buffers, limiting the number of packets per connection event to three. Therefore, to maximize the throughput, the number of link layer buffers must be incremented to at least six, holding at least the packets for the next two connection events.

© 2022 Renesas Electronics

8.4 Motion Module

8.4.1 Description

The Motion Module is used for acquiring data from a combined gyroscope/accelerometer sensor.

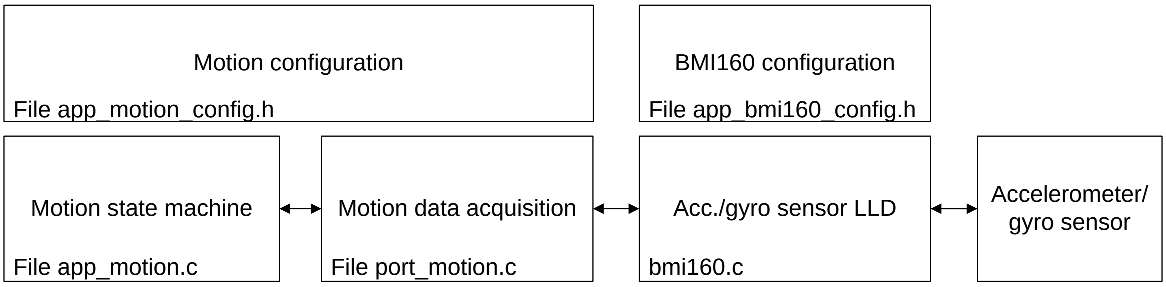


Figure 8: Motion Module Block Diagram

A Finite State Machine (FSM) is used for waking up, initializing, handling and powering down the sensor. The sensor LLD is used to access the sensor. An adaptation layer between the FSM and the LLD is implemented in file `port_motion.c` to condition the raw data acquired by the sensor.

The Motion Module is enabled by defining symbol `HAS_MOTION` in the file `user_config.h`.

Files `app_motion.c`, `port_motion.c`, `bmi160.c`, `bmi160_support.c`, `app_bmi160_config.h` and `app_motion_config.h` must be included in the project.

8.4.2 Configuration

The BMI160 sensor configuration is set in file `app_bmi160_config.h`:

- `MOTION_IF` sets the sensor communication interface, either SPI or I2C.
- Define `INCLUDE_BMI160ACC` to enable the accelerometer.
- Define `INCLUDE_BMI160GYR` to enable the gyroscope.
- Define `INCLUDE_BMI160TEM` to enable the temperature measurement.

The Motion Module configuration is defined in file `app_motion_config.h`:

- The gyroscope/accelerometer sensor type is selected by defining the corresponding symbol:
  - `BMI160` for the BMI160 sensor.
  - `BMI055` for the BMI055 sensor.
- Sensor placement (top or bottom side of the PCB, placement rotation) is specified in order to assign the measurements acquired for the sensor to the proper physical axis (X, Y or Z)
  - Define `MOTION_PCB_BOTTOM` when the sensor is placed at the bottom side of the PCB.
  - Set `MOTION_ROTATION` to the rotation angle of the sensor in degrees (0, 90, 180 or 270).
- Set `MOTION_DEACTIVATION_TIMEOUT_IN_MS` to the time (in ms) that the sensor will remain active after calling the stop function to deactivate it. This timeout is used when the motion feature is deactivated for a short time. In that case the sensor remains active to ensure smooth operation.

 © 2022 Renesas Electronics

8.5 Touchpad Module

8.5.1 Description

Overview

The Touchpad Module offers interaction with trackpad/touch devices and provides an application layer for managing a large variety of touch events and gestures, as well as forwarding these to the user application for further processing. The block diagram of the Touchpad Module is depicted in Figure 9.

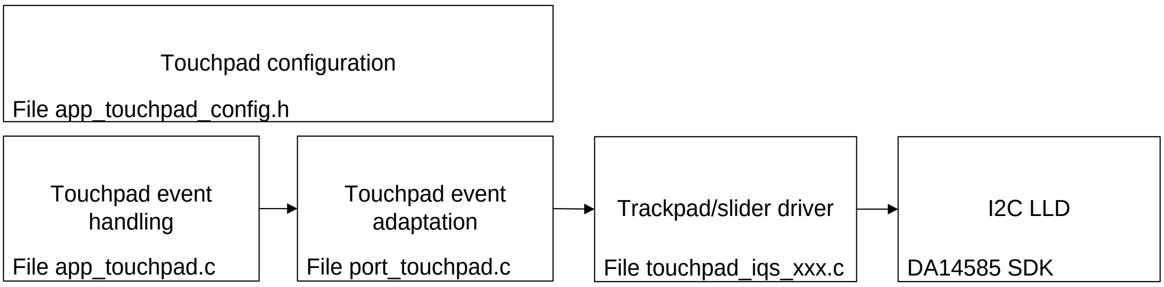


Figure 9: Touchpad Module Block Diagram

Depending on the configuration, the Touchpad Module may support either trackpad modules (defining `HAS_TOUCHPAD_TRACKPAD` will enable the driving of Azoteq's IQS572 IC) or simple slider touch modules (defining `HAS_TOUCHPAD_SLIDER` will enable the driving of Azoteq's IQS263 IC).

In order to use the Touchpad Module, the user must either define `HAS_TOUCHPAD_TRACKPAD` or `HAS_TOUCHPAD_SLIDER` in the file `user_config.h`.

The files `app_touchpad.c` and `port_touchpad.c` must be included in the project.

Operation

Apart from the initialization and de-initialization functions, the Touchpad Module provides a polling function, that can be used by the user application to poll it periodically for any existing touch events. When there are any new touch events, the module polling will trigger the execution of the touchpad module's callback functions. Configuration and registration of the touch event callback functions must be done in the file `app_touchpad_config.h`.

As mentioned above, the Touchpad Module can either work with trackpad ICs or with simpler slider ICs. The two variations of the RCU touch controller ICs can be seen below:

Touchpad Operation

The touch events produced by the Touchpad Module consist of a touch action and a set of coordinates related to the touch action. The supported touch actions are listed below:

- `APP_TOUCHPAD_RESET`
- `APP_TOUCHPAD_RELEASE`
- `APP_TOUCHPAD_TRACK_STARTED`
- `APP_TOUCHPAD_TRACKING`
- `APP_TOUCHPAD_TRACK_STOPPED`
- `APP_TOUCHPAD_TAP_UP`
- `APP_TOUCHPAD_TAP_DOWN`
- `APP_TOUCHPAD_TAP_RIGHT`

 © 2022 Renesas Electronics

- APP_TOUCHPAD_TAP_LEFT
- APP_TOUCHPAD_SINGLE_FINGER_TAP
- APP_TOUCHPAD_TOUCH_AND_HOLD
- APP_TOUCHPAD_SWIPE_LEFT
- APP_TOUCHPAD_SWIPE_RIGHT
- APP_TOUCHPAD_SWIPE_UP
- APP_TOUCHPAD_SWIPE_DOWN
- APP_TOUCHPAD_ZOOM_IN
- APP_TOUCHPAD_ZOOM_OUT
- APP_TOUCHPAD_DOUBLE_FINGER_TAP
- APP_TOUCHPAD_SCROLL_UP
- APP_TOUCHPAD_SCROLL_DOWN
- APP_TOUCHPAD_SCROLL_RIGHT
- APP_TOUCHPAD_SCROLL_LEFT
- APP_TOUCHPAD_FLICK_LEFT
- APP_TOUCHPAD_FLICK_RIGHT
- APP_TOUCHPAD_ROTATE_LEFT
- APP_TOUCHPAD_ROTATE_RIGHT
- APP_TOUCHPAD_NO_EVENT

The Touchpad Module uses a special callback function `user_touchpad_special_eventCB()` to notify the user application of touch events.

Trackpad Operation

When the trackpad variation is used, tracking actions are handled separately. Additionally, the Touchpad Module uses an internal state machine to determine whether tracking has started, is ongoing or has stopped.

The tracking state machine uses a separate callback function `user_touchpad_track_eventCB()` to notify the user application of tracking related events. Figure 10 represents the tracking FSM.

In addition to the separate tracking callback function, the Touchpad Module also provides the function `app_touchpad_get_last_track_info()`. When called, this function returns the last absolute coordinates and the coordinate differences between the newest tracking data and the oldest tracking data (deltaX and deltaY) and refreshes the oldest tracking data with the current ones. This function helps translating the movement of the finger on the trackpad into HID Mouse reports.

 © 2022 Renesas Electronics

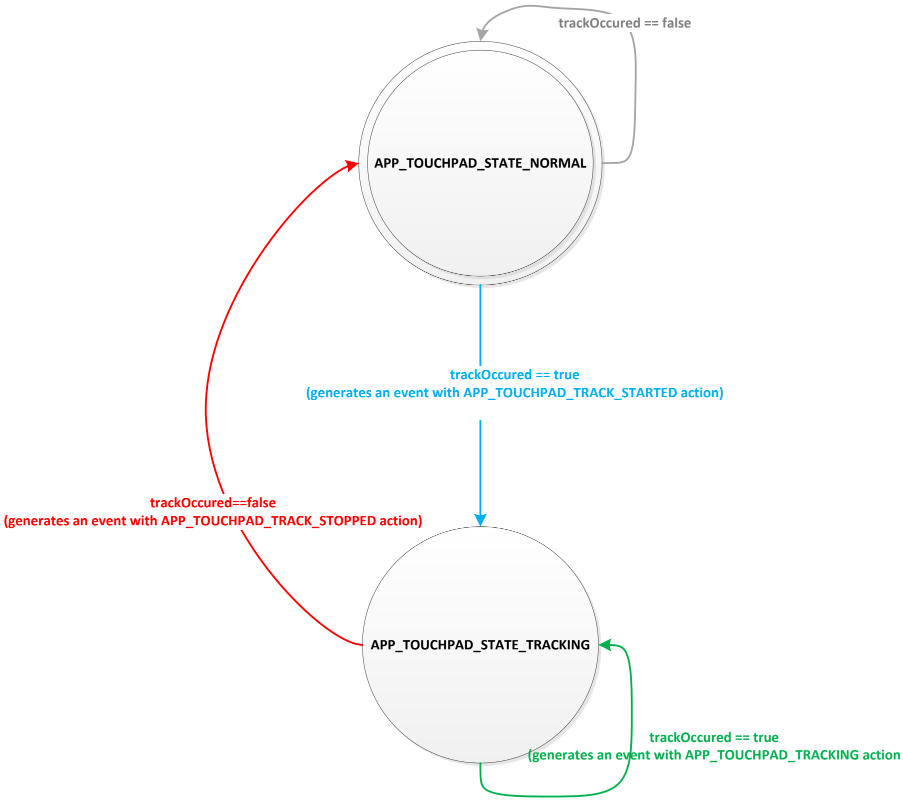


Figure 10 Touchpad Module - Tracking State Machine

8.5.2 Configuration

The Touchpad Module configuration can be modified via the file `app_touchpad_config.h`.

- `TOUCH_PAD_MODULE` defines which touchpad IC is connected to the RCU motherboard.
  - When `HAS_TOUCHPAD_TRACKPAD` is defined the Azoteq IQS572 trackpad IC will be used
  - When `HAS_TOUCHPAD_SLIDER` is defined the Azoteq IQS263 slider IC will be used.
- When `TOUCHPAD_STATIC_EVENT_SETUP` is defined, the user can configure the touch events of which the application can be notified, at compile time. Otherwise, these events can be modified during run time.
- `APP_TOUCH_MAX_SLIDE_ACCUMULATED_EVENTS`: When the slider IC is used, this symbol defines the minimum number of slide events needed to notify the user application about a valid slide event. This number defines the slide sensitivity.
- `APP_TOUCH_COORDS_*`: These symbols are specific for the IQS263 slider IC and describe the tap direction boundaries (Up, Down, Left, Right).
- `TOUCH_INT_PORT`, `TOUCH_INT_PIN`: These symbols define the GPIO pin that is used to receive interrupts from the touch ICs.
- `TOUCHPAD_INT_POLARITY`: Depending on the Touchpad Module that is used, this symbol defines the interrupt polarity of the touch ICs.
- `user_touchpad_track_eventCB`: Pointer to the user application function that will handle the incoming tracking events (if tracking is supported).
- `user_touchpad_special_eventCB`: Pointer to the user application function that will handle all incoming touch events (except tracking events).
- `app_touchpad_funcs`: A structure of platform specific functions required for the module's proper behavior and functionality.

 © 2022 Renesas Electronics

8.6 IR Module

8.6.1 Description

The IR Module is used for transmitting key codes using an IR LED connected to a GPIO pin of the DA14585.

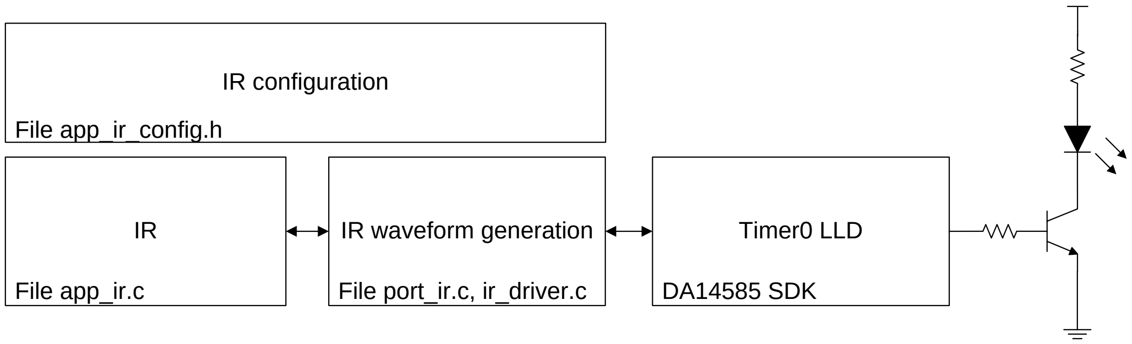


Figure 11: IR Module Block Diagram

The IR Module uses Timer 0 of the DA14585 to generate the waveform that drives the IR transmitter. The module can be configured to generate waveforms compatible with various protocols such as Philips RC-5, Sony SIRC and NEC.

The following waveform parameters can be defined in the configuration file to implement the waveform generation for a specific protocol:

- Carrier frequency
- Carrier duty-cycle
- Number of carrier cycles for logic 1 mark
- Number of carrier cycles for logic 1 space
- Number of carrier cycles for logic 0 mark
- Number of carrier cycles for logic 0 space
- A function must be implemented for the specific protocol to send the proper sequence of 1 and 0 symbols to form the IR message.

Detailed IR functionality description and implementation is provided in Ref. [6].

The IR Module is enabled by defining symbol `HAS_IR` in the file `user_config.h`.

Files `app_ir.c`, `port_ir.c`, `ir_driver.c` and `app_ir_config.h` must be included in the project.

8.6.2 Configuration

The IR Module parameters are defined in the file `app_ir_config.h`.

The main configuration is defined in structure `ir_params`. The following parameters can be set:

- `use_ble_sync`: The IR transmission is synchronized with the BLE end event. The IR transmitter is activated after the completion of the BLE event.
- `max_repeat`: The maximum number of times a key code is transmitted when the transmission is not stopped by the application (e.g. while the corresponding key is kept pressed).
- `protocol_params`: A pointer to a structure of type `ir_protocol_params_t`, which holds the parameters used for the waveform generation according to the protocol used. The structure `ir_protocol_params_t` has the following members:
    - `timer_freq`: The frequency (in MHz) of Timer 0 used for the IR waveform generation (2 MHz, 4 MHz, 8 MHz or 16 MHz).

© 2022 Renesas Electronics

- `carrier_period`: The IR carrier clock period (in timer clock cycles).
- `carrier_on_time`: The IR carrier on-time (in timer clock cycles) which defines the carrier duty cycle.
- `logic_one_mark`: The IR logic 1 mark duration (in carrier clock cycles).
- `logic_one_space`: The IR logic 1 space duration (in carrier clock cycles).
- `logic_zero_mark`: The IR logic 0 mark duration (in carrier clock cycles).
- `logic_zero_space`: The IR logic 0 space duration (in carrier clock cycles).
- `repeat_time`: The message repeat time (in carrier clock cycles).
- `repeat_type`: The message repeat type, either `IR_REPEAT_FROM_CODE_FIFO` or `IR_REPEAT_FROM_REPEAT_FIFO`.

- `send_command_callback`: Pointer to the function that called for constructing the IR message to be transmitted.

8.6.3 Design Considerations

Only the Philips RC-5 protocol has been implemented and tested. Example code and configuration for Sony SIRC and NEC protocols is provided for reference. Parameters `protocol_params` and `send_command_callback` can be customized to implement the required IR protocol.

8.7 GPIO Keys Module

8.7.1 Description

The GPIO Keys Module initializes, checks and processes GPIOs for key presses or releases. The block diagram of the GPIO Keys Module is depicted in Figure 12.

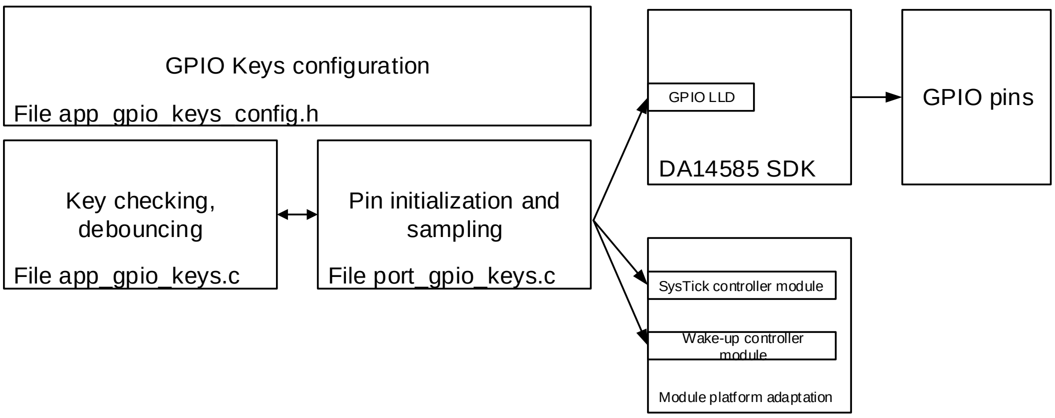


Figure 12: GPIO Keys Module Block Diagram

The GPIO Keys Module has the following features:

- Supports maximum eight keys
- Selectable polarity (active low or active high)
- Key debouncing with configurable debouncing time and sampling period

The keys are connected to GPIO pins which are configured as inputs. When no key is pressed, the internal resistors of the DA14585 pins define the appropriate input level:

- Active-low keys: a pull-up resistor is used to define a high input level.
- Active-high keys: a pull-down resistor is used to define a low input level.

© 2022 Renesas Electronics

The keys are sampled using the SysTick Controller Module described in Section 8.16 and the Wakeup Controller Module described in Section 8.14. The GPIO LLD is used to read the status of the pins. One sampling cycle is completed when the status of all pins has been read.

The key sampling is implemented in the file `port_gpio_keys.c`.

The keys are debounced in the file `app_gpio_keys.c` in order to detect key press or release events.

The application can register a callback function to receive notifications of key press/release events.

The GPIO Keys Module is enabled by defining symbol `HAS_GPIO_KEYS` in the file `user_config.h`.

Files `app_gpio_keys.c`, `port_gpio_keys.c` and `app_gpio_keys_config.h` must be included in the project.

8.7.2 Configuration

The GPIO Keys Module parameters are defined in the file `app_gpio_keys_config.h`.

- `GPIO_NUM_OF_KEYS` defines the number of keys to be used.
- `GPIO_KEY_x_PORT` and `GPIO_KEY_x_PIN` define the pin to which key x is connected. x is in the range of 0 to (`GPIO_NUM_OF_KEYS - 1`).
- `GPIO_KEY_x_POLARITY` defines the polarity (`ACTIVE_LOW` or `ACTIVE_HIGH`) of the pin to which key x is connected. x is in the range of 0 to (`GPIO_NUM_OF_KEYS - 1`).
- `GPIO_DEBOUNCE_TIME_IN_MS` defines the time (in ms) during which samples are collected to determine the status of a key.
- `GPIO_DEBOUNCE_PERIOD_IN_US` defines the sampling period (in μs) for key debouncing.
- `APP_GPIO_KEYS_NOTIFICATION_CB`: Pointer to the function called to notify the application that a key has been pressed or released. This function is called in interrupt context during debouncing.

Key names are defined in the `gpio_key` enumeration. These names should be used to reference the keys when checking their status or handling a notification.

8.7.3 Design Considerations

The number of contiguous similar samples needed to decide the status of a key during debouncing is (`GPIO_DEBOUNCE_TIME_IN_MS * 1000 / GPIO_DEBOUNCE_PERIOD_IN_US`).

© 2022 Renesas Electronics

8.8 HID Report Module

8.8.1 Description

The HID Report Module enables the generation, buffering and transmission of HID reports via the HID Over GATT Profile (HOGP).

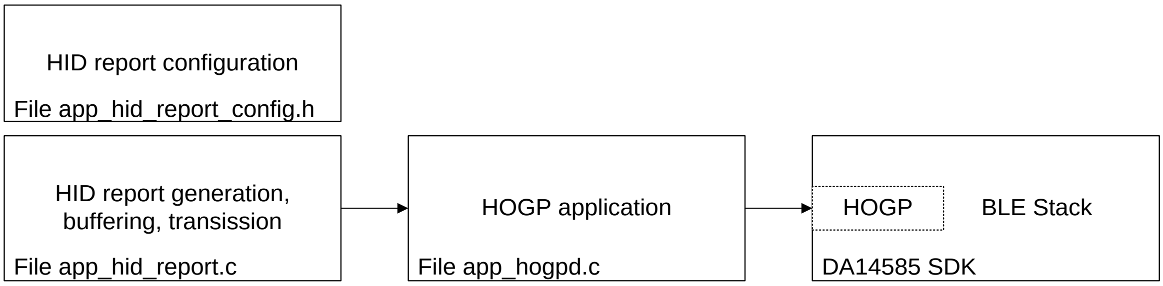


Figure 13: HID Report Module Block Diagram

The HID Report Module has the following features:

- Normal HID keyboard report handling
- Normal HID keyboard report rollover handling
- Extended HID keyboard report handling
- Report FIFO

HID reports can be added in the FIFO using their HID report index. Reports are sent from the FIFO to the HOGP. A single FIFO is used for all HID reports. The memory space of the FIFO is pre-allocated.

Special handling is implemented for normal and extended keyboard reports. The differences between these report types are described in Ref. [5].

New reports are created for key events by appending the new key event to the events reported in the previous report of the same type. Keys are added to the report when they are pressed and removed when they are released.

Key rollover is implemented for normal keyboard reports, when more than six keys are pressed simultaneously.

The HID Report Module is enabled by defining symbol `HAS_HID_REPORT` in the file `user_config.h`.

Files `app_hid_report.c` and `app_hid_report_config.h` must be included in the project.

8.8.2 Configuration

The HID Report Module configuration is specified in file `app_hid_report_config.h`. The following configuration options can be defined:

- `HID_REPORT_FIFO_SIZE` sets the number of HID key reports that can be stored in the report FIFO.
- `HID_REPORT_MAX_REPORT_SIZE` sets the maximum size of the reports that can be stored in the report FIFO.
- `HID_REPORT_ROLL_OVER_BUF_SIZE` sets the size of the rollover buffer. It must be greater than 6, since this is the maximum number of simultaneous keys that can be reported in a normal HID keyboard report. `HID_REPORT_ROLL_OVER_BUF_SIZE` must be greater or equal to the number of keys that can be detected simultaneously by the Keyboard Module as defined in Section 8.1.2.
- Define `HID_REPORT_HISTORY` to enable reporting of any events happened while disconnected.
- `HID_REPORT_NORMAL_REPORT_IDX` sets the index of the normal HID keyboard report.
- `HID_REPORT_NORMAL_REPORT_SIZE` sets the size of the normal HID keyboard report.

© 2022 Renesas Electronics

- HID_REPORT_EXTENDED_REPORT_IDX sets the index of the extended HID keyboard report.
- HID_REPORT_EXTENDED_REPORT_SIZE sets the size of the extended HID keyboard report.
- APP_HID_REPORT_SEND_CB points to the callback function which the HID Report Module calls to notify the application that a report has been sent to HOGP.

8.9 Advertising FSM Module

8.9.1 Description

Overview

The Advertising FSM Module implements a straightforward BLE Finite State Machine (FSM) that manages BLE advertising related events and functions. It provides a simple API to the Connection FSM Module for easy advertising management: procedures for starting and stopping advertising as well as giving back notifications of every advertising related event.

Advertising FSM States

The Advertising FSM states are listed below:

- ADV_IDLE: The Advertising FSM is *Idle*. It is waiting for incoming advertising related events.
- ADV_UNDIRECTED: The FSM is currently doing *Undirected Advertising*. Pairing is allowed during this advertising type. For more information, see the array adv_fsm_config.adv_params [ADV_SETTING_UNDIRECTED] in the Advertising FSM configuration file app_adv_fsm_config.h.
- ADV_UNDIRECTED_LIM: The FSM is currently doing *Undirected Limited Advertising*. Pairing is not allowed during this advertising type. This advertising type is usually used when the device wants to reconnect to a known host that does not have a public address. For more information, see the configuration array adv_fsm_config.adv_params[ADV_SETTING_UNDIRECTED_LIM] in the Advertising FSM configuration file app_adv_fsm_config.h.
- ADV_UNDIRECTED_NO_PAIRING: The FSM is currently doing *Undirected Advertising*. Pairing is not allowed during this advertising type. For more information, see the configuration array adv_fsm_config.adv_params[ADV_SETTING_UNDIRECTED_NO_PAIR] in the Advertising FSM configuration file app_adv_fsm_config.h.
- ADV_UNDIRECTED_SLOW: The FSM is currently doing *Slow Undirected Advertising*. Pairing is allowed during this advertising type. For more information, see the configuration array adv_fsm_config.adv_params[ADV_SETTING_SLOW] in the Advertising FSM configuration file app_adv_fsm_config.h.
- ADV_UNDIRECTED_SPECIAL: The FSM is currently doing *Special Undirected Advertising*. Pairing is allowed during this advertising type. The payload of this advertising type is set by using the app_adv_fsm_set_special_adv_data() function. For more information, see the configuration array adv_fsm_config.adv_params[ADV_SETTING_SPECIAL] in the Advertising FSM configuration file app_adv_fsm_config.h.
- ADV_FSM_EVENT_PENDING: The FSM is currently waiting for a previous operation to be completed in order to handle a pending event, queued by the application or the Connection FSM Module.
- ADV_DIRECTED: The FSM is currently performing *Directed Advertising* to a host.

Advertising FSM Events

The Advertising FSM events are listed below. Events marked as *output events* are events generated by the Advertising FSM Module, while the rest of the events (marked as *input events*) are used by the Connection FSM Module or the user application to drive the Advertising FSM.

- Output Events:
  - UND_ADV_COMPLETED: This event will occur when an Undirected Advertising cycle completes.

 © 2022 Renesas Electronics

- DIR_ADV_COMPLETED: This event will occur when a Directed Advertising cycle completes.
- UND_ADV_TIMED_OUT: This event will occur when the Undirected Advertising Timer expires, which will trigger the Advertising FSM Module to stop the ongoing Undirected Advertising cycle (which will eventually result in an UND_ADV_COMPLETED event).
- DIR_ADV_INTERRUPTED: This event will occur when an ongoing Directed Advertising cycle gets interrupted, due to a connection request.
- UND_ADV_INTERRUPTED: This event will occur when an ongoing Undirected Advertising cycle gets interrupted, due to a connection request.

● Input Events:
  - START_ADV: This event is used to tell the Advertising FSM Module to start an Undirected Advertising cycle, using the settings provided in the configuration table adv_fsm_config.adv_params[ADV_SETTING_UNDIRECTED].
  - START_ADV_LIM: This event is used in order to tell the Advertising FSM Module to start an Undirected Limited Advertising cycle, using the settings provided in the configuration table adv_fsm_config.adv_params[ADV_SETTING_UNDIRECTED_LIM].
  - START_ADV_NO_PAIR: This event is used to tell the Advertising FSM Module to start an Undirected Advertising cycle with no Pairing Allowed, using the settings provided in the configuration table adv_fsm_config.adv_params[ADV_SETTING_UNDIRECTED_NO_PAIR].
  - START_ADV_DIR: This event is used to tell the Advertising FSM Module to start a Directed Advertising cycle.
  - STOP_ADV: This event is used to tell the Advertising FSM Module to stop any ongoing advertising cycle.
  - START_ADV_SPECIAL: This event is used to tell the Advertising FSM Module to start an Undirected Special Advertising cycle.

Advertising FSM State Transitions

For better understanding of the Advertising FSM Module operation, state transition tables (Table 10 to Table 24) and a state transition diagram (Figure 14) demonstrate all possible state transitions.

Table 10: ADV_IDLE State Transitions

Event	Description	Next State
START_ADV	The FSM will start Undirected Advertising.	ADV_UNDIRECTED
START_ADV_DIR	The FSM will start Directed Advertising.	ADV_DIRECTED
START_ADV_NO_PAIR	The FSM will start Undirected Advertising with No Pairing Allowed.	ADVERTISING_ST
START_ADV_LIM	The FSM will start Undirected Limited Advertising.	ADV_UNDIRECTED_NO_PAIRING
STOP_ADV	A STOP_ADV event while the FSM is in ADV_IDLE state will not have any effect, since there is no ongoing advertising cycle.	ADV_IDLE
START_ADV_SPECIAL	If special advertising is supported, the FSM will start Undirected Special Advertising.	ADV_SPECIAL

Table 11: ADV_DIRECTED State Transitions

Event	Description	Next State
DIR_ADV_INTERRUPTED	The ongoing Directed Advertising cycle was interrupted by a connection request. The FSM will send a notification that directed advertising ended (ADV_FSM_DIR_ADV_ENDED) and go to ADV_IDLE state.	ADV_IDLE

© 2022 Renesas Electronics

Event	Description	Next State
DIR_ADV_COMPLETED	The ongoing Directed Advertising cycle was completed. When the counter of directed advertising repeats has reached the configured value adv_fsm_config.directed_advertising_repeats and adv_fsm_config.disable_undirected_advertise is true, the FSM will send a notification that directed advertising has ended and go to ADV_IDLE state.	ADV_IDLE
DIR_ADV_COMPLETED	The ongoing Directed Advertising was completed. When the counter of directed advertising repeats has reached the configured value adv_fsm_config.directed_advertising_repeats and adv_fsm_config.disable_undirected_advertise is false, the FSM will start Undirected Advertising with No Pairing Allowed and send a notification that Undirected Advertising has started (ADV_FSM_UND_ADV_STARTED).	ADV_UNDIRECTED_NO_PAIRING
START_ADV	When one of these events occurs during Directed Advertising, the FSM will save the event as a pending event and move to the ADV_FSM_EVENT_PENDING state, waiting for the ongoing advertising cycle to end and continue with the saved pending event.	ADV_FSM_EVENT_PENDING
START_ADV_LIM		
START_ADV_NO_PAIR		
START_ADV_DIR		
STOP_ADV		

Table 12: ADV_UNDIRECTED State Transitions

Event	Description	Next State
UND_ADV_COMPLETED	An ongoing Undirected Advertising cycle was completed (stopped). When Slow Undirected Advertising is supported (adv_params[ADV_SETTING_SLOW].discoverable_timeout is non-zero), the FSM will start Slow Undirected Advertising.	ADV_SLOW
UND_ADV_COMPLETED	An ongoing Undirected Advertising cycle was completed (stopped). When Slow Undirected Advertising is NOT supported (adv_params[ADV_SETTING_SLOW].discoverable_timeout is zero) the FSM will go to ADV_IDLE state.	ADV_IDLE
UND_ADV_INTERRUPTED	The ongoing Undirected Advertising was interrupted due to a connection request. The FSM will clear the Undirected Advertising Timer, send a notification that Undirected Advertising has ended (ADV_FSM_UND_ADV_ENDED) and go to ADV_IDLE state.	ADV_IDLE
UND_ADV_TIMED_OUT	The Undirected Advertising Timer has expired. The FSM will stop the ongoing Undirected Advertising cycle and stay in the same state, waiting for the UND_ADV_COMPLETED event.	ADV_UNDIRECTED
START_ADV	When one of these events occurs during Undirected Advertising, the FSM will save the event as a pending event, clear the Undirected Advertising Timer, stop the ongoing Undirected Advertising and move to the ADV_FSM_EVENT_PENDING state, waiting for the ongoing advertising cycle to end and continue with the saved pending event.	ADV_FSM_EVENT_PENDING
START_ADV_LIM		
START_ADV_NO_PAIR		
START_ADV_DIR		
STOP_ADV		
START_ADV_SPECIAL		

 © 2022 Renesas Electronics

Table 13: ADV_UNDIRECTED_LIM State Transitions

Event	Description	Next State
UND_ADV_COMPLETED	An ongoing Undirected Advertising cycle was completed (stopped). When the configuration field adv_fsm_config.disable_undirected_advertise is true, the FSM will send a notification that Undirected Limited Advertising has ended (ADV_FSM_UND_ADV_LIM_ENDED) and go to ADV_IDLE state.	ADV_IDLE
UND_ADV_COMPLETED	An ongoing Undirected Advertising cycle was completed (stopped). When the configuration field adv_fsm_config.disable_undirected_advertise is false, the FSM will start Undirected Advertising with No Pairing Allowed and send a notification that Undirected Advertising has started (ADV_FSM_UND_ADV_STARTED).	ADV_UNDIRECTED_NO_PAIRING
UND_ADV_INTERRUPTED	The ongoing Undirected Advertising cycle was interrupted due to a connection request. The FSM will clear the Undirected Advertising Timer, send a notification that Undirected Limited Advertising has ended (ADV_FSM_UND_ADV_LIM_ENDED) and go to ADV_IDLE state.	ADV_IDLE
UND_ADV_TIMED_OUT	The Undirected Advertising Timer has expired. The FSM will stop the ongoing Undirected Advertising cycle and stay in the same state, waiting for the UND_ADV_COMPLETED event.	ADV_UNDIRECTED_LIM
START_ADV START_ADV_LIM START_ADV_NO_PAIR START_ADV_DIR STOP_ADV START_ADV_SPECIAL	When one of these events occurs during Undirected Limited Advertising, the FSM will save the event as a pending event, clear the Undirected Advertising Timer, stop the ongoing Undirected Advertising cycle and move to the ADV_FSM_EVENT_PENDING state, waiting for the ongoing advertising cycle to end and continue with the saved pending event	ADV_FSM_EVENT_PENDING

Table 14: ADV_UNDIRECTED_NO_PAIRING State Transitions

Event	Description	Next State
UND_ADV_COMPLETED	An ongoing Undirected Advertising cycle was completed (stopped). When Slow Undirected Advertising is supported (adv_params[ADV_SETTING_SLOW)].discoverable_timeout is non-zero), the FSM will start Slow Undirected Advertising and send a notification that Undirected Advertising has started (ADV_FSM_UND_ADV_STARTED).	ADV_SLOW
UND_ADV_COMPLETED	An ongoing undirected advertising cycle was completed (stopped). When Slow Undirected Advertising is NOT supported (adv_params[ADV_SETTING_SLOW)].discoverable_timeout is zero), the FSM will send a notification that Undirected Advertising has ended (ADV_FSM_UND_ADV_ENDED) and go to ADV_IDLE state.	ADV_IDLE
UND_ADV_INTERRUPTED	The ongoing Undirected Advertising was interrupted due to a connection request. The FSM will clear the Undirected Advertising Timer, send a notification that Undirected Advertising has ended (ADV_FSM_UND_ADV_ENDED) and go to ADV_IDLE state.	ADV_IDLE
UND_ADV_TIMED_OUT	The Undirected Advertising Timer has expired. The FSM will stop the ongoing Undirected Advertising cycle and stay in the same state, waiting for the UND_ADV_COMPLETED event.	ADV_UNDIRECTED_NO_PAIR

 © 2022 Renesas Electronics

Event	Description	Next State
START_ADV START_ADV_LIM START_ADV_NO_PAIR START_ADV_DIR STOP_ADV START_ADV_SPECIAL	When one of these events occurs during Undirected Advertising, the FSM will save the event as a pending event, clear the Undirected Advertising Timer, stop the ongoing Undirected Advertising cycle and move to the ADV_FSM_EVENT_PENDING state, waiting for the ongoing advertising to end and continue with the saved pending event.	ADV_FSM_EVENT_PENDING

Table 15: ADV_UNDIRECTED_SLOW State Transitions

Event	Description	Next State
UND_ADV_COMPLETED	An ongoing undirected advertising cycle was completed (stopped). The FSM will send a notification that Undirected Advertising has ended (ADV_FSM_UND_ADV_ENDED) and go to ADV_IDLE state.	ADV_IDLE
UND_ADV_INTERRUPTED	The ongoing Undirected Advertising was interrupted due to a connection request. The FSM will clear the Undirected Advertising Timer, send a notification that Undirected Advertising has ended (ADV_FSM_UND_ADV_ENDED) and go to ADV_IDLE state.	ADV_IDLE
UND_ADV_TIMED_OUT	The Undirected Advertising Timer has expired. The FSM will stop the ongoing Undirected Advertising cycle and stay in the same state, waiting for the UND_ADV_COMPLETED event.	ADV_SLOW
START_ADV START_ADV_LIM START_ADV_NO_PAIR START_ADV_DIR STOP_ADV START_ADV_SPECIAL	When one of these events occurs during Undirected Advertising, the FSM will save the event as a pending event, clear the Undirected Advertising Timer, stop the ongoing Undirected Advertising cycle and move to the ADV_FSM_EVENT_PENDING state, waiting for the ongoing advertising to end and continue with the saved pending event.	ADV_FSM_EVENT_PENDING

Table 16: ADV_FSM_EVENT_PENDING State Transitions

Event	Description	Next State
UND_ADV_COMPLETED	While in FSM pending event state an Undirected Advertising cycle was completed. The FSM will send a notification that Undirected Advertising has ended (ADV_FSM_UND_ADV_ENDED). When there is a valid pending event (START_ADV_* or STOP_ADV), the FSM will process that event exactly like when being in ADV_IDLE state.	The next state depends on the saved/pending event. See the ADV_IDLE state transition table for each event (START_ADV, START_ADV_LIM, START_ADV_DIR, START_ADV_NO_PAIR, START_ADV_SPECIAL, STOP_ADV)
DIR_ADV_COMPLETED	While in FSM pending event state a Directed Advertising cycle was completed. The FSM will send a notification that Directed Advertising has ended (ADV_FSM_DIR_ADV_ENDED). When there is a valid pending event (START_ADV_* or STOP_ADV), the FSM will process that event exactly like when being in ADV_IDLE state.	The next state depends on the saved/pending event. See the ADV_IDLE state transition table for each case of pending event (START_ADV,

 © 2022 Renesas Electronics

Event	Description	Next State
		START_ADV_LIM, START_ADV_DIR, START_ADV_NO_PAIR, START_ADV_SPECIAL, STOP_ADV)
DIR_ADV_INTERRUPTED	An ongoing Directed Advertising cycle was interrupted due to a connection request. The FSM will clear the pending event(set it to ADV_NO_EVENT), send a notification that Directed Advertising has ended (ADV_FSM_DIR_ADV_ENDED) and go to ADV_IDLE state.	ADV_IDLE
UND_ADV_INTERRUPTED	An ongoing Undirected Advertising cycle was interrupted due to a connection request. The FSM will clear the pending event (set it to ADV_NO_EVENT), send a notification that Undirected Advertising has ended (ADV_FSM_UND_ADV_ENDED) and go to ADV_IDLE state.	ADV_IDLE
START_ADV START_ADV_LIM START_ADV_NO_PAIR START_ADV_DIR STOP_ADV START_ADV_SPECIAL	When one of these events occurs during the pending event state, the FSM will save that event as pending and stay in the same state, waiting for a trigger by one of the *_ADV_COMPLETED events.	ADV_FSM_EVENT_ PENDING

© 2022 Renesas Electronics

For a better view of the Advertising FSM Module, Figure 14 presents a state transition diagram.

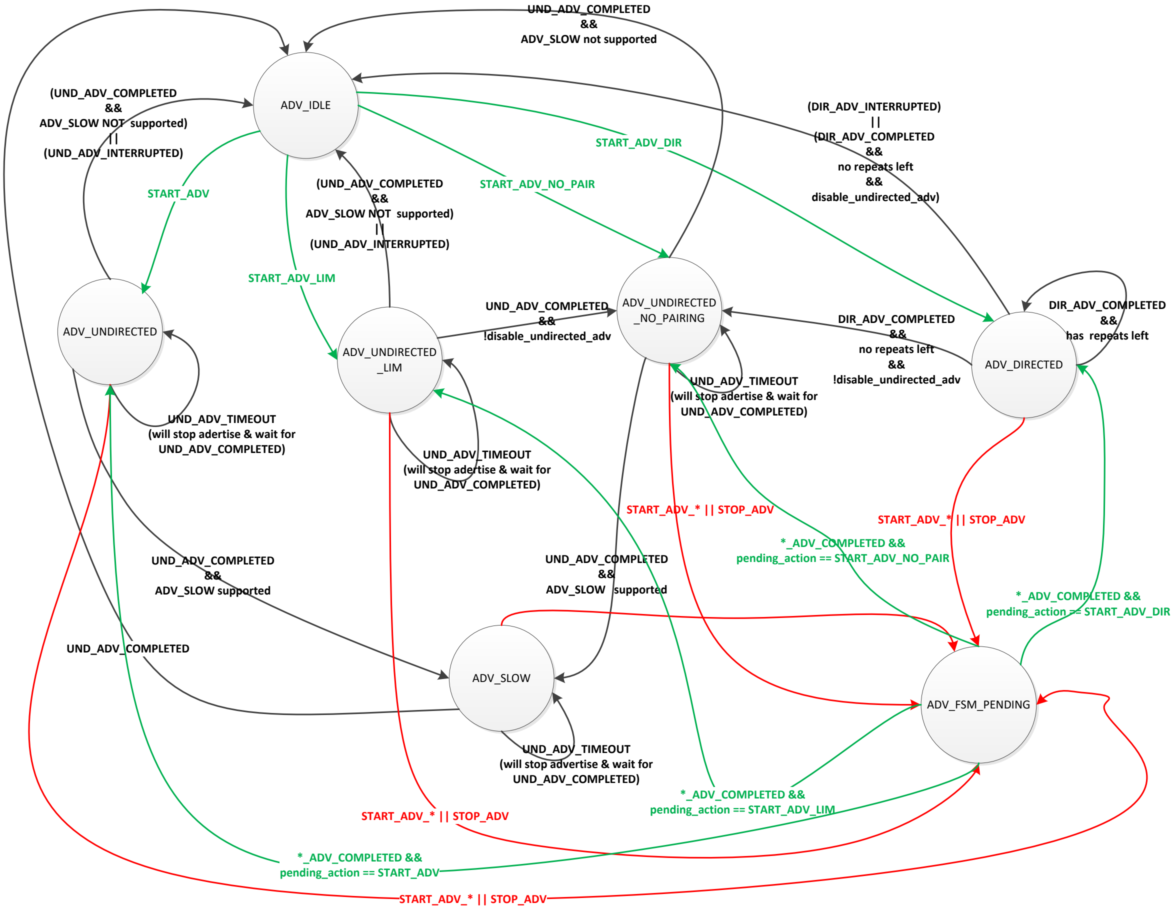


Figure 14: Advertising FSM State Transition Diagram

8.9.2 Configuration

The Advertising FSM Module can be configured via the `app_adv_fsm_config.h` header file.

- `APP_DFLT_DEVICE_NAME`: This macro defines the default device name when advertising.
- `AUTO_APPEND_DEVICE_NAME_IN_ADV_DATA`: If this symbol is defined, the `APP_DFLT_DEVICE_NAME` will be automatically appended in advertising data. If it does not fit, a partial name will be appended in the advertising data. The full name will be appended in the scan response data.
- `APP_ADV_DATA`: The default Undirected Advertising payload.
- `HAS_SPECIAL_ADVERTISING`: When defined, this symbol enables the usage of Special Undirected Advertising.
- The `adv_fsm_config` structure contains configuration information of the Advertising FSM:
  - `app_adv_notification_callback`: A callback function that should be registered in order to handle incoming notifications from the Advertising FSM Module.
  - `disable_undirected_advertise`: When this flag is set to `true`, the FSM will not start Undirected Advertising after Directed or Limited Undirected Advertising has ended.
  - `directed_advertising_repeats`: This parameter sets the number of the repeats of Directed Advertising (implicitly sets the duration of Directed Advertising).
  - `disable_advertising_timeout`: When this flag is set to `true`, this parameter disables the usage of the Undirected Advertising timer.

 © 2022 Renesas Electronics

- The adv_params array inside the adv_fsm_config structure, contains Undirected Advertising configuration parameters for the following Undirected Advertising types: ADV_SETTING_SPECIAL, ADV_SETTING_SLOW, ADV_SETTING_UNDIRECTED_LIM, ADV_SETTING_UNDIRECTED, ADV_SETTING_UNDIRECTED_NO_PAIR.

  For each advertising type 'x' the following parameters can be set:
  - adv_params[x].discoverable_timeout: For Undirected Advertising configuration 'x' this parameter sets the duration of Undirected Advertising for that configuration. Set this parameter to zero to disable the advertising type specified by 'x'.
  - adv_params[x].adv_int_min: For Undirected Advertising configuration 'x' this parameter sets the minimum advertising interval in milliseconds.
  - adv_params[x].adv_int_max: For Undirected Advertising configuration 'x' this parameter sets the maximum advertising interval in milliseconds.
  - adv_params[x].adv_mode: For Undirected Advertising configuration 'x' this parameter sets undirected advertising mode
  - adv_params[x].adv_data: For Undirected Advertising configuration 'x' this parameter sets the undirected advertising data
  - adv_params[x].adv_data_length: For Undirected Advertising configuration 'x' this parameter sets the undirected advertising data length
  - adv_params[x].scan_rsp_data: For Undirected Advertising configuration 'x' this parameter sets the undirected advertising scan response data
  - adv_params[x].scan_rsp_data_length: For Undirected Advertising configuration 'x' this parameter sets the undirected advertising scan response data length
- app_adv_fsm_funcs: A structure of platform specific advertising functions needed for proper functionality of the Advertising FSM Module.

8.10 Connection FSM Module

8.10.1 Description

Overview

The Connection FSM Module implements a straightforward BLE Finite State Machine (FSM) that manages BLE connection, disconnection, advertising, bonding, pairing etc. The module provides a set of functions meant to be used by the user application to feed the FSM with events, as well as give feedback (indications) back to the application after the events have been handled.

The following sub-sections describe the states of the Connection FSM, as well as all possible events that may occur during the device's operation. For advertising related functions the Connection FSM collaborates closely with the Advertising FSM, which is described in Section 8.9.

Connection FSM States

This section enumerates and describes all possible Connection FSM states and events that may occur during the device's operation, as well as all the possible state transitions after the events have been handled.

The states of the Connection FSM are the following:

- IDLE_ST: The Connection FSM is idle. There are no active connection, advertising, bonding, pairing procedures. The FSM either waits for an event to occur or for previous uncompleted operations to end.
- ADVERTISING_ST: The FSM is advertising. It will be in this state as long as there is an ongoing advertising cycle. The FSM will leave this state when the advertising ends, either because of an incoming connection that will interrupt the advertising (will go to CONNECTION_IN_PROGRESS_ST) or because the advertising cycle has completed (will go to IDLE_ST).

 © 2022 Renesas Electronics

- CONNECTION_IN_PROGRESS_ST: The FSM is processing a connection. The FSM will go to this state whenever there is a connection in progress and wait for one of the following events:
  - Connection completed (will go to CONNECTED_ST).
  - Connection cancelled (will go to DISCONNECTED_INIT_ST).
  - Disconnection (will start advertising and go to ADVERTISING_ST).
  - Pairing request (will go to CONNECTED_PAIRING_ST).
  - Power off (will terminate the ongoing connection and go to POWEROFF_ST).
- CONNECTED_PAIRING_ST: The FSM is connected and there is an ongoing pairing procedure.
- CONNECTED_ST: The FSM is successfully connected to a host.
- DISCONNECTED_INIT_ST: The FSM will go to this state when waiting for a disconnection to complete.
- POWEROFF_ST: The FSM will go to this state after a power-off event during an active connection. It will start the power-off timer and wait for a shutdown event or for power-off timer expiration. Then it will request disconnection and go to WAITING_DISCONNECTION_AFTER_POWEROFF state.
- WAITING_DISCONNECTION_AFTER_POWEROFF: The FSM will go to this state after a shutdown event or power-off timer expiration when in POWEROFF_ST, and will wait there until there is a successful disconnection.

Connection FSM Events

The Connection FSM events are listed below.

- ADV_COMPLETED_EVT: This event indicates that an advertising cycle has completed.
- ALT_PAIR_TIMER_EXP_EVT: This event indicates that the alternate pairing timer has expired, which is used to give the device time to switch to a new host.
- CONN_CANCELLED_EVT: This event indicates that the connection in progress was cancelled.
- CONN_CMP_EVT: This event indicates that a connection has been completed successfully.
- CONN_REQ_EVT: This event indicates an incoming connection request from a peer device.
- CONN_UPD_RESP_EVT: This event indicates that the connection parameters have been updated.
- DISCONNECT_CMP_EVT: This event indicates that an ongoing connection has been terminated.
- INIT_EVT: This event indicates that the Connection FSM Module must be initialized.
- NEW_HOST_EVT: This event indicates a paring request from an unknown host (i.e. one that was not previously bonded).
- PAIRING_REQ_EVT: This event indicates an incoming pairing request from a host.
- PARAM_UPD_TIMEOUT_EVT: This event indicates that the required time has passed since the connection was established and a connection parameter request can now be sent to the host.
- PASSCODE_TIMEOUT_EVT: This event indicates that the RCU has not sent a passcode witin a specified time.
- PASSKEY_ENTERED: This event indicates that the host is waiting for the RCU to provide the displayed passcode in order to complete pairing with MITM authentication.
- POWEROFF_EVT: This event indicates that the device will be powered off, because it has been inactive for a specified time.
- POWEROFF_TIMEOUT_EVT: This event indicates that the power-off timer has expired.
- SHUTDOWN_EVT: This event indicates that the application is ready to proceed with disconnection.
- START_PAIRING_EVT: This event indicates that the application wishes to pair with a new host.
- SWITCH_EVT: This event indicates that the application has requested to switch to a known (i.e. previously bonded) host. This event is supported when multi-bonding is enabled.
- USER_EVT: This custom event is used to indicate user actions (RCU: button push or touch event).

 © 2022 Renesas Electronics

Connection FSM State Transitions

For every FSM state described above, state transition tables (Table 17 to Table 24) show all the expected events, how they are handled and the resulting state transitions.

Table 17: IDLE_ST State Transitions

Event	Description	Next State
INIT_EVT	When the NORMALLY_CONNECTABLE configuration is enabled (set to 1), the FSM will never be in IDLE_ST state and always try to reconnect by doing Directed or Limited Undirected Advertising in ADVERTISING_ST state.	ADVERTISING_ST
	When the NORMALLY_CONNECTABLE configuration is disabled (set to 0), the FSM will just send an IDLE indication to the user application.	IDLE_ST
USER_EVT	A custom user event has occurred (RCU: button push or touch event). The FSM will send an indication that it is exiting IDLE_ST state, and start advertising in order to reconnect to a previously connected host (if any).	ADVERTISING_ST
SWITCH_EVT	The application has requested to switch to a known host. The FSM will send an indication that it is exiting IDLE_ST state, and start advertising in order to reconnect to a previously connected host (if any).	ADVERTISING_ST
START_PAIRING_EVT	The application wishes to pair with a new host. The FSM will start default Undirected Advertising.	ADVERTISING_ST
ADV_COMPLETED_EVT	An advertising cycle has completed. The FSM will continue advertising with the same type.	ADVERTISING_ST
POWEROFF_EVT	The device will be powered off. Since the FSM is already in IDLE_ST state, there are no additional actions to perform and it stays in the same state.	IDLE_ST

Table 18: ADVERTISING_ST State Transitions

Event	Description	Next State
SWITCH_EVT	The application has requested to switch to a known host. The FSM will start Limited Undirected Advertising in order to reconnect to a previously connected host (if any).	ADVERTISING_ST
START_PAIRING_EVT	The application wishes to pair with a new host. The FSM will start default Undirected Advertising.	ADVERTISING_ST
ADV_COMPLETED_EVT	An advertising cycle has completed. The FSM will go to IDLE_ST state, since advertising is over.	IDLE_ST
POWEROFF_EVT	The device will be powered off. The FSM will stop advertising and stay in the same state, waiting for an ADV_COMPLETED_EVT event. Then it will switch to IDLE_ST state.	ADVERTISING_ST
CONN_REQ_EVT	A connection request was received from a peer device. The FSM will interrupt advertising and will move to CONNECTION_IN_PROGRESS_ST state in order to manage the incoming connection.	CONNECTION_IN_PROGRESS_ST
ALT_PAIR_TIMER_EXP_EVT	Pairing with a new host was not completed within the specified time. The FSM will stop advertising and move to IDLE_ST state.	IDLE_ST

 © 2022 Renesas Electronics

Table 19: CONNECTION_IN_PROGRESS_ST State Transitions

Event	Description	Next State
ALT_PAIR_TIMER_EXP_EVT	Pairing with a new host was not completed within the specified time. The FSM will send a disconnection request and go to DISCONNECTED_INIT_ST state, waiting for a DISCONNECT_CMP_EVT event. Then it will start advertising with no pairing.	DISCONNECTED_INIT_ST
PAIRING_REQ_EVT	A pairing request has occurred while a connection is in progress. The FSM will check whether the pairing request can be accepted. When it is accepted, the FSM will send a positive pairing response and switch to CONNECTED_PAIRING_ST state.	CONNECTED_PAIRING_ST
	Otherwise, when the pairing request cannot be accepted, the FSM will send a negative pairing response and stay in the same state.	CONNECTION_IN_PROGRESS_ST
NEW_HOST_EVT	A pairing request was received from an unknown host. The FSM will send a ‘re-initialization’ indication to the application and stay in the same state, waiting for a PAIRING_REQ_EVT event.	CONNECTION_IN_PROGRESS_ST
CONN_CANCELLED_EVT	The connection in progress was cancelled. The FSM will send a disconnection request and switch to DISCONNECTED_INIT_ST state, waiting for a DISCONNECT_CMP_EVT event.	DISCONNECTED_INIT_ST
CONN_CMP_EVT	A connection has been made successfully. The FSM will send a ‘connection complete’ indication to the application and switch to CONNECTED_ST state.	CONNECTED_ST
DISCONNECT_CMP_EVT	The ongoing connection has been terminated. The FSM will start (continue) advertising as before the connection began and switch to ADVERTISING_ST state.	ADVERTISING_ST
POWEROFF_EVT	The device will be powered off. The FSM will send a disconnection request start the power-off timer (when available) and switch to POWEROFF_ST state.	POWEROFF_ST
START_PAIRING_EVT	The application wishes to pair with a new host. The FSM will send a disconnection request and move to DISCONNECTED_INIT_ST state, waiting for a DISCONNECT_CMP_EVT event. Then it will start default Undirected Advertising.	DISCONNECTED_INIT_ST
SWITCH_EVT	The application has requested to switch to a known host. The FSM will send a disconnection request and move to DISCONNECTED_INIT_ST state, waiting for a DISCONNECT_CMP_EVT event. Then it will start advertising to reconnect to a previously connected host (if any).	DISCONNECTED_INIT_ST

Table 20: CONNECTED_PAIRING_ST State Transitions

Event	Description	Next State
PASSKEY_ENTERED	The host is waiting for the RCU to provide the displayed passcode in order to pair securely. The FSM will start a passcode timer and stay in the same state.	CONNECTED_PAIRING_ST
CONN_CMP_EVT	A connection has been completed successfully. The FSM will check whether the host can be accepted. When it is accepted, the FSM will switch to CONNECTED_ST state.	CONNECTED_ST
	Otherwise, when the host cannot be accepted, the FSM will send a disconnection request and stay in the same	CONNECTED_PAIRING_ST

© 2022 Renesas Electronics

Event	Description	Next State
	state, waitng for a DISCONNECT_CMP_EVT event.	
DISCONNECT_CMP_EVT	The ongoing connection has been terminated. The FSM will start default advertising.	ADVERTISING_ST
PASSCODE_TIMEOUT_EVT	The RCU has not sent a passcode within the specified time. The FSM will send a disconnection request and go to DISCONNECTED_INIT_ST state, waiting for a DISCONNECT_CMP_EVT event.	DISCONNECTED_INIT_ST
POWEROFF_EVT	The device will be power off. The FSM will send a disconnection request, start the power-off timer (when available) and switch to POWEROFF_ST state.	POWEROFF_ST
START_PAIRING_EVT	The application wishes to pair with a new host. The FSM will send a disconnect request and go to DISCONNECTED_INIT_ST state, waiting for DISCONNECT_CMP_EVT event. Then it will start default Undirected Advertising.	DISCONNECTED_INIT_ST
SWITCH_EVT	The application has requested to switch to a known host. The FSM will send a disconnection request and move to DISCONNECTED_INIT_ST state, waiting for a DISCONNECT_CMP_EVT event. Then it will start advertising to reconnect to a previously connected host (if any).	DISCONNECTED_INIT_ST

Table 21: CONNECTED_ST State Transitions

Event	Description	Next State
PAIRING_REQ_EVT	A pairing request occurred while connected to a host. The FSM will check whether the pairing request can be accepted. When it is accepted, the FSM will move to CONNECTED_PAIRING_ST state.	CONNECTED_PAIRING_ST
	Otherwise, when the pairing request is not accepted, the FSM will send a disconnection request and stay in the same state, waiting for a DISCONNECT_CMP_EVT event.	CONNECTED_ST
CONN_UPD_RESP_EVT	The device has received a connection update response from the host. The FSM will stay in the same state.	CONNECTED_ST
DISCONNECT_CMP_EVT	The ongoing connection has been terminated. When advertising after disconnection is supported, the FSM will start default advertising for reconnection and move to ADVERTISING_ST state.	ADVERTISING_ST
	Otherwise, when advertising after disconnection is not supported, the FSM will indicate to the application that it is going idle and move to IDLE_ST state.	IDLE_ST
PARAM_UPD_TIMEOUT_EVT	The host has not responded to a connection parameter update request within the specified time. The FSM will resend the update request and stay in the same state.	CONNECTED_ST
START_PAIRING_EVT	The application wishes to pair with a new host. The FSM will send a disconnection request and go to DISCONNECTED_INIT_STATE, waiting for a DISCONNECT_CMP_EVT event. Then it will start default Undirected Advertising.	DISCONNECTED_INIT_ST
SWITCH_EVT	The application has requested to switch to a known host. The FSM will send a disconnection request, stop the parameter update timer (when running) and move to DISCONNECTED_INIT_ST state, waiting for a DISCONNECT_CMP_EVT event. Then it will start advertising	DISCONNECTED_INIT_ST

© 2022 Renesas Electronics

Event	Description	Next State
	to reconnect to a previously connected host (if any).	
POWEROFF_EVT	The device will be powered off. The FSM will stop the parameter update timer and start the power-off timer. It will **not** send a disconnection request, allowing the application to perform any required cleanup actions. The FSM will move to POWEROFF_ST state, waiting for the application to send a SHUTDOWN_EVT, in order to proceed with the disconnection.	POWEROFF_ST

Table 22: POWEROFF_ST State Transitions

Event	Description	Next State
CONN_REQ_EVT	A connection request was received from a peer device. The FSM will send a disconnection request and stay in the same state.	POWEROFF_ST
POWEROFF_TIMEOUT_EVT	The power-off timer has expired. The FSM will send a disconnection request and go to WAITING_DISCONNECTION_AFTER_POWEROFF state, waiting for a DISCONNECT_CMP_EVT event.	WAITING_DISCONNECTION_AFTER_POWEROFF
SHUTDOWN_EVT	The application is ready to proceed with disconnection. The FSM will stop the power-off timer (when running), send a disconnection request and move to WAITING_DISCONNECTION_AFTER_POWEROFF state, waiting for a DISCONNECT_CMP_EVT event.	WAITING_DISCONNECTION_AFTER_POWEROFF
DISCONNECT_CMP_EVT	The ongoing connection has been terminated. The FSM will indicate to the application that it is going idle and move to IDLE_ST state.	IDLE_ST

Table 23: DISCONNECTED_INIT_ST State Transitions

Event	Description	Next State
DISCONNECT_CMP_EVT	The ongoing connection has been terminated. The FSM will start advertising as it was last queued before disconnecting.	ADVERTISING_ST
POWEROFF_EVT	The device will be powered off. The FSM will move to POWEROFF_ST state.	POWEROFF_ST

Table 24: WAITING_DISCONNECTION_AFTER_POWEROFF State Transitions

Event	Description	Next State
DISCONNECT_CMP_EVT	The ongoing connection has been terminated. The FSM will indicate to the application that it is going idle and will move to IDLE_ST state.	IDLE_ST

© 2022 Renesas Electronics

8.10.2 Configuration

The Connection FSM Module configuration is specified in file `app_con_fsm_config.h` as follows:

- `USE_L2CAP_CONN_UPDATE_REQ`: When this symbol is defined, `L2CAP_CONN_PARAM_UPDATE_REQ` will be sent instead of `LL_CONNECTION_PARAM_REQ`.
- `MBOND_LOAD_INFO_AT_INIT`: When this symbol is defined the device will load all bonding information into the retention RAM at power-up. This eliminates subsequent reads and reduces power consumption.
- `MAX_BOND_PEER`: Defines the maximum number of hosts that can be handled by the multi-bonding mechanism.
- `FORCE_CONNECT_TO_HOST_ON`: When this symbol is defined, force-connecting to a specific host is enabled. The application can use an index to store the bonding data of a host to a specific location in the memory. The same index can be used to force the reconnection only to the specific host. `FORCE_CONNECT_NUM_OF_HOSTS` must be defined as well.
- `FORCE_CONNECT_NUM_OF_HOSTS`: Defines the number of hosts that can be handled when the `FORCE_CONNECT_TO_HOST_ON` feature is used.
- `notification_info`: This structure contains the configuration for the storage of the attribute values that must be stored for each host in the non-volatile memory, together with the bonding data. One 32-bit word is used for all attributes. For each attribute the following parameters must be defined:
  - `position`: The bit position in the 32-bit word.
  - `num_of_bits`: Number of bits used for the attribute in the 32-bit word.
  - `uuid`: The UUID of the attribute.
  - `type`: The type of the attribute, either `CCC_TYPE` or `ATTR_TYPE`.
  - `num_of_atts`: Number of attributes.
  - `length`: The length of the attribute in the database.
  - `default_value`: The default value of the attribute. It must fit in `num_of_bits`.
- `con_fsm_params`: This structure contains a variety of configuration parameters of the Connection FSM Module:
  - `disable_bonding_data_storage`: When set to `true`, bonding data will not be saved in Non-Volatile Memory (NVM).
  - `has_multi_bond`: When set to `true`, the multi-bonding feature of the Connection FSM Module is enabled, where the device can bond with multiple hosts and switch between them.
  - `use_pref_conn_params`: When set to `true`, the Connection FSM Module will send a `CONNECTION_PARAM_UPDATE_REQUEST` after completion of the connection.
  - `disable_advertise_after_disconnection`: When set to `true`, the Connection FSM Module will not start advertising after disconnecting from an active connection.
  - `has_mitm`: When set to `true`, the Connection FSM Module will use MITM authentication mode.
  - `has_passcode_timeout`: When set to `true` and when MITM authentication is supported (`has_mitm` is set to `true`), timeout checking will be enabled during passcode entry.
  - `has_nv_rom`: When set to `true`, the Connection FSM Module will use Non-Volatile Memory (NVM) to store bonding data.
  - `has_white_list`: When set to `true`, the Connection FSM Module will use white-listing.
    Note: `has_virtual_white_list` should not be set to `true` when `has_white_list` is enabled.
  - `has_virtual_white_list`: When set to `true`, the usage of a Virtual White List is enabled, which provides support for hosts with resolvable random addresses. In that case, all addresses are filtered by software and not by hardware.

 © 2022 Renesas Electronics

 - has_security_request_send: When set to true, the Connection FSM Module will send a SECURITY_REQUEST when connected to a host.
 - has_usage_counters: When set to true, the usage counters mechanism is enabled. Usage counters are used during the bonding of a new host in order to determine the oldest entry in the NVM that will be used for storing the bonding data of the new host. If usage counters are not used then the next new host will be stored in the next empty position in the NVM. If all positions are used then the new host will be stored in the last position of the NVM.
 - has_smart_rssi_pairing: When set to true, the Smart RSSI Pairing feature of the Connection FSM Module is enabled. In that case, the device will only pair with hosts that have at least a predefined RSSI value (hosts that are in close range to the device).
 - smart_pairing_rssi_threshold: When has_smart_rssi_pairing is set to true, this parameter defines the minimum RSSI threshold that is needed for a pairing host to be accepted (how close the host needs to be to the device).
 - enc_safeguard_timeout: Defines the time (in ms) to wait for a PAIRING_REQ or ENC_REQ when connecting to a host. When a PAIRING_REQ or ENC_REQ is not received the connection is dropped.
 - passcode_timeout: Defines the timeout value (in ms) until the passcode is entered (when has_passcode_timeout is set to true).
 - notification_timeout: Defines the time (in ms) to wait for the last notifications to be sent to the host before disconnecting (set to 0 if the device can disconnect immediately).
 - time_to_request_param_upd: When use_pref_conn_params is set to true, this parameter defines the time (in ms) needed to request an update of connection parameters.
 - alt_pair_disconn_time: During host-switching this parameter determines the time (in ms) to block the previous host in order to allow a new host to connect.
 - param_update: Structure defining the parameter update timing, with the following members:

   .preferred_conn_interval_min: Preferred minimum connection interval (in ms).

   .preferred_conn_interval_max: Preferred maximum connection interval (in ms).

   .preferred_conn_latency: Preferred connection latency (in number of skipped events).

   .preferred_conn_timeout: Preferred connection timeout (in ms).
 - state_update_callback: Pointer to the callback function that is called when the connection state has changed to *connected*, *connection in progress*, *disconnected*, *off*, *passcode entry started*, or when connection information of a previous host is cleared (re-initialized).
 - attr_update_callback: Pointer to the callback function that is called when bonding data are read from the non-volatile memory so that the service database is updated.
- app_con_fsm_funcs: A structure of platform-specific functions of the Connection FSM Module.

 © 2022 Renesas Electronics

8.11 LED Indicators Module

8.11.1 Description

The LED Indicators Module enables the generation of visual indications by generating patterns using LEDs connected to GPIO pins or using custom LED drivers.

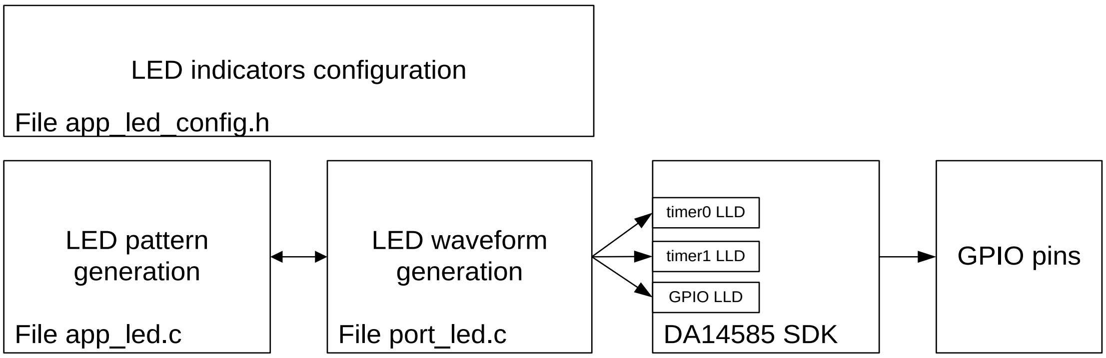


Figure 15: LED Indicators Module Block Diagram

The LED Indicators Module has the following features:

- Configurable blink patterns
- Multi-LED patterns
- Ramp-up and ramp-down using PWM output to generate a ‘breathing’ effect
- LEDs connected to custom drivers (other than GPIO pins)

A number of LEDs can be configured to be used by the LED Indicators Module. These LEDs can be connected to GPIO pins. Each GPIO pin can source current to the LED (active high) or sink current from the LED (active low) as depicted in Figure 16.

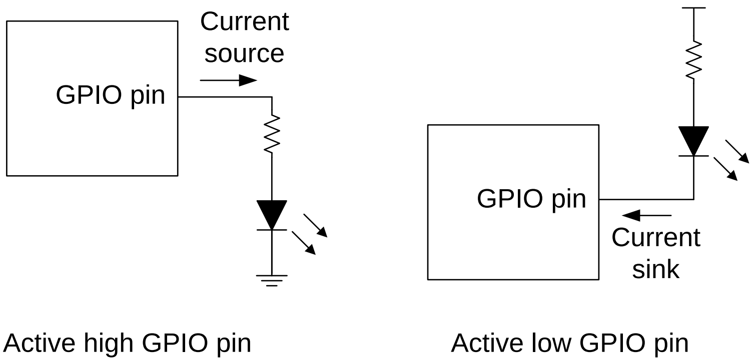


Figure 16: LED Connection to GPIO Pin

The GPIO LLD is used to turn the LED on or off.

The Ramp feature is implemented using Timer 2 to generate a PWM signal to dim the LED and Timer 0 to progressively change the duty cycle of the PWM signal to ramp-up or ramp-down the brightness of the LED. The PWM frequency and rate of change of the PWM duty cycle is configurable. The Ramp feature can be used for only one LED at a time.

LEDs can also be connected to custom LED drivers instead of GPIO pins. In this case a custom callback function is used to control the state of the LED.

 © 2022 Renesas Electronics

Various LED patterns can be defined to indicate the status of the device. One or more LEDs can be used for each pattern. On, off, ramp-up, ramp-down times as well as mode, time offset from the beginning and number of repetitions can be individually configured for each LED of each pattern.

LED patterns can have high or low priority. When two patterns share the same LEDs, the one with high priority LED cannot be interrupted by the other. In this case, the second pattern will remain pending and it will be started when the first one has finished.

Time offset is used when multiple LEDs are used in a pattern, to define the relative offset between the waveforms of the various LEDs in the pattern.

A table containing the LED configuration is defined for each pattern. This table is passed as parameter to all functions of the LED Indicators Module that control the LED states.

The **LED mode** can be one of the following:

- `LED_BLINK`: LED is turned on and then off for a specified number of times.
- `LED_NO_BLINK`: LED is turned on and then off once.
- `LED_TURN_OFF`: Turn LED off in case is left on by another pattern.
- `LED_DOUBLE_BLINK`: LED is turned on and then off twice for a specified number of times.

Up to two software timers can be used to generate LED patterns. A timer is assigned to each LED. LEDs that are sharing the same software timer cannot be used at the same time.

The **blink period** for `LED_BLINK` and `LED_DOUBLE_BLINK` modes is equal to (`on_time + off_time`). The `ramp_on_time` and `ramp_off_time` are part of `on_time`.

The LED Indicators Module is enabled by defining symbol `HAS_LED_INDICATORS` in the file `user_config.h`.

Files `app_led.c`, `port_leds.c` and `app_leds_config.h` must be included in the project.

8.11.2 Configuration

The LED Indicators Module configuration is specified in file `app_leds_config.h`. The following configuration options can be defined:

- Define symbol `LED_USE_RAMP_FEATURE` to enable the Ramp Up/Down feature. Timer 0 and Timer 2 are used for this feature. When the Ramp feature is enabled the following parameters must be defined to configure the ramp waveforms that drive the LEDs:
    - `LED_RAMP_PWM_FREQUENCY_IN_HZ`: The frequency (in Hz) of the PWM that is used for dimming the LED.
    - `LED_RAMP_PWM_MIN_DC`: The minimum duty cycle (in %) of the PWM.
    - `LED_RAMP_STEP_PERIOD_IN_MS`: The step period (in ms) used for increasing or decreasing the PWM duty cycle.
- Define symbol `LED_USE_DOUBLE_BLINK_FEATURE` to enable the `LED_DOUBLE_BLINK LED` mode.

The LED names are defined in the `led_id_t` enumeration. These names can be used to reference the LEDs when defining the GPIO pin configuration and indication patterns.

The LED GPIO pin configuration is defined in table `app_led_pins`. The following parameters ere defined for each LED that is connected to a GPIO pin:

- `port`: The GPIO port of the LED pin.
- `pin`: The GPIO pin of the LED pin.
- `high`: Defines the polarity of the GPIO pin:
    - `LED_ACTIVE_HIGH:` if the GPIO pin is sourcing the LED current.
    - `LED_ACTIVE_LOW:` if the GPIO pin is sinking the LED current
- `mode_function`: Must always be `OUTPUT | PID_GPIO`.

© 2022 Renesas Electronics

Table led_pads defines whether the LEDs are connected to a GPIO pin or to a custom LED driver.

- When the corresponding LED is connected to a GPIO pin, the parameter type must be assigned to LED_GPIO.
- When the LED is connected to a custom driver, the parameter callback must be assigned to the function controlling the LED state.

For each LED pattern a table of type led_params_t must be defined. This table has one entry for each LED of the pattern. The following parameters can be set:

- id: The LED identifier as defined in the led_id_t enumeration.
- mode: The LED mode. Can be one of LED_BLINK, LED_NO_BLINK, LED_TURN_OFF or LED_DOUBLE_BLINK
- high_priority: Must be set to true to define the LED as high priority.
- on_time: The time (in ms) that the LED will remain on.
- off_time: The time (in ms) that the LED will remain off.
- ramp_on_time: The duration (in ms) of the ramp-up waveform.
- ramp_off_time: The duration (in ms) of the ramp-down waveform.
- count: The number of repetitions in LED_BLINK and LED_DOUBLE_BLINK modes.
- delay: The time offset (in ms) of the LED in the pattern.

8.11.3 Design Considerations

When the duty cycle of the PWM signal driving an LED is very small, the LED may appear to be off. When during ramp-up or ramp-down the LED appears to be off for a part of the ramp time, set the LED_RAMP_PWM_MIN_DC to a larger value, so that the PWM duty cycle steps are properly recalculated.

The LED pattern priority can be used for non-critical indicators. An example of such an indicator is the low battery LED. Its priority can be set to low. When an important event must be indicated (such as a connection or disconnection from the host), its priority can be set to high in order to interrupt the low battery indication.

© 2022 Renesas Electronics

8.12 Sound Indicator Module

8.12.1 Description

The Sound Indicator (buzzer) Module gives the user application the ability to generate simple musical melodies. The block diagram of the Sound Indicator Module Block diagram is depicted in Figure 17.

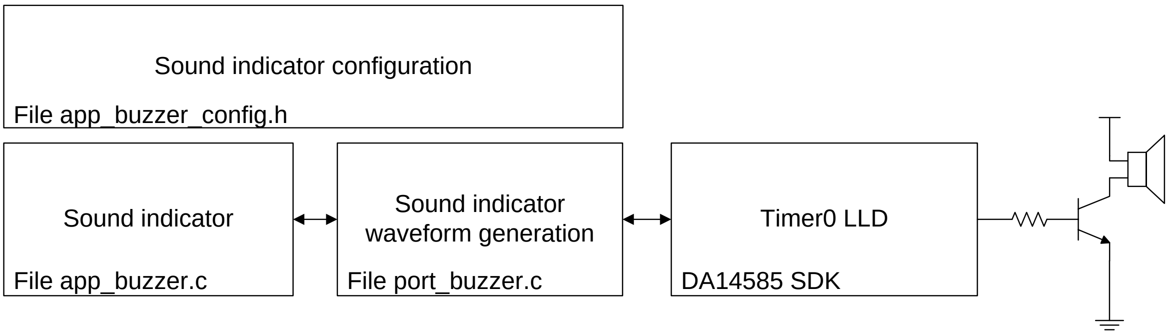


Figure 17: Sound Indicator Module Block Diagram

The user application needs to provide an array containing the notes of the melody. The module reproduces each note until the melody ends.

The Sound Indicator Module is enabled by defining symbol `HAS_SOUND_INDICATION` in the file `user_config.h`.

Files `app_buzzer.c`, `port_buzzer.c` and `app_buzzer_config.h` must be included in the project.

8.12.2 Configuration

The Sound Indicator Module configuration is specified in file `app_buzzer_config.h`.

- `BUZZER_MAX_MELODY_NOTES`: This symbol defines the max number of musical notes per melody
- `BUZZER_MAX_MELODY_LENGTH`: This symbol defines the maximum size of a melody array. Each melody array contains the length of the melody plus the notes of the melody (`BUZZER_MAX_MELODY_NOTES + 1`).
- `BUZZER_NOTE_DURATION_TICKS`: This symbol defines the duration of each note contained in a melody
- `buzzerMelodies[][]`: This array contains a number of different melodies. The user may alter the predefined melodies and/or add more melodies, depending on application requirements.
- `app_buzzer_funcs`: This structure registers the platform functions required by the Sound Indicator Module in order to function properly.

 © 2022 Renesas Electronics

8.13 Power Manager Module

8.13.1 Description

The Power Manager Module enables the use of the inactivity timer from different application modules. The block diagram of the Power Manager Module is depicted in Figure 18.

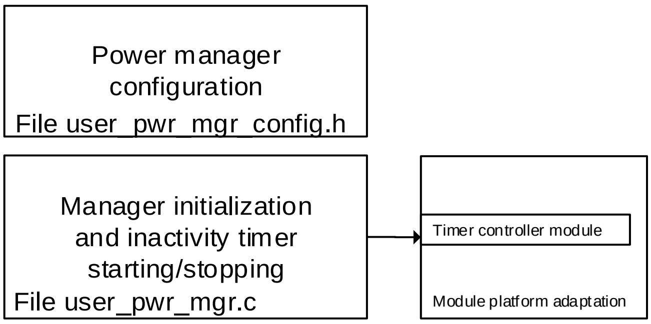


Figure 18: Power Manager Module Block Diagram

The Power Manager Module has a configurable inactivity timeout.

The inactivity timer is managed by calling the Timer Controller Module, described in Section 8.15, from the file `user_pwr_mgr.c`.

The application can register a callback function to receive a notification on expiration of the inactivity timeout, to stop everything and go to sleep.

The Power Manager Module is enabled by defining symbol `HAS_PWR_MGR` in the file `user_config.h`.

Files `user_pwr_mgr.c` and `user_pwr_mgr_config.h` must be included in the project.

8.13.2 Configuration

The Power Manager Module configuration parameters are defined in structure `pwr_mgr_params` in the file `user_pwr_mgr_config.h`:

- `inactivity_timeout`: Defines the idle time (in ms) until the inactivity timer expires.
- `inactivity_callback`: Defines the function that is called to notify the application that the inactivity timer has expired. This function is called in interrupt context.

8.13.3 Design Considerations

When extended timer support is not enabled in the Timer Controller Module, the maximum inactivity timeout is `KE_TIMER_DELAY_MAX`.

 © 2022 Renesas Electronics

8.14 Wakeup Controller Module

8.14.1 Description

The Wakeup Controller Module allows the use of the wakeup controller by several application modules. The block diagram of the Wakeup Controller Module is depicted in Figure 19.

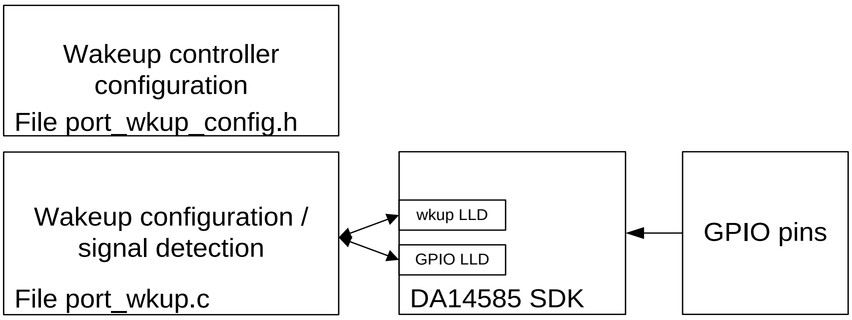


Figure 19: Wakeup Controller Module Block Diagram

Each application module can register one or more GPIO pins that can wake up the system, and a callback function that is called when the Wakeup Controller Module detects a system wakeup by one of these GPIOs. The Wakeup Controller Module disables the wakeup source before calling the callback function of the application module. This is done in interrupt context. Re-enabling the wakeup source is the responsibility of the application module.

Application modules that have registered a single pin as a wakeup source, are checked first and their callback functions are called as needed. When no single-pin wakeup source is found, the callback functions are called of all application modules that have registered multiple wakeup pins. The application module is responsible for deciding whether it should handle the system wakeup event.

The Wakeup Controller Module can be configured to always call the callback functions of application modules that have registered multiple pins as wakeup sources, even when single-pin wakeup sources have been detected.

The Wakeup Controller Module is enabled by defining symbol `HAS_WKUP` in the file `user_config.h`.

Files `port_wkup.c` and `port_wkup_config.h` must be included in the project.

8.14.2 Configuration

The Wakeup Controller Module configuration is specified in file `port_wkup_config.h`.

Wakeup channels are defined in the `port_wkup_channel` enumeration. Each application module can use one or more channels to define GPIOs as wakeup sources and the corresponding callback functions.

The wakeup channel configuration is defined in table `wkup_config`. For each channel the following parameters can be set:

- `callback`: The pointer to the callback function that will be called when the system wakes up.
- `single_pin_input`: Set to `true` when only one GPIO input is used by the application module to wake up the system.

  When `single_pin_input` is set to `true`:
  - `config.pin_config.port`: The port of the GPIO input.
  - `config.pin_config.pin`: The pin of the GPIO input.
  - `config.pin_config.polarity`: Set to `WKUPCT_PIN_POLARITY_LOW` if the system must be woken up by a high-to-low transition. Otherwise set to `WKUPCT_PIN_POLARITY_HIGH`

© 2022 Renesas Electronics

When single_pin_input is set to false:

   - config.pin_mask: The Wakeup Controller mask for all the pins. Use macro WKUP_MASK or WKUPCT_PIN_SELECT to generate the mask.

- Symbol WKUP_ALWAYS_CONTINUE_WITH_MASKS can be defined to always call the callbacks of channels with multiple GPIO pin wakeup sources, even when a channel with a single-pin wakeup source has been found.

8.15 Timer Controller Module

8.15.1 Description

The Timer Controller Module simplifies usage of Kernel timers from different application modules. The block diagram of the Timer Controller Module is depicted in Figure 20.

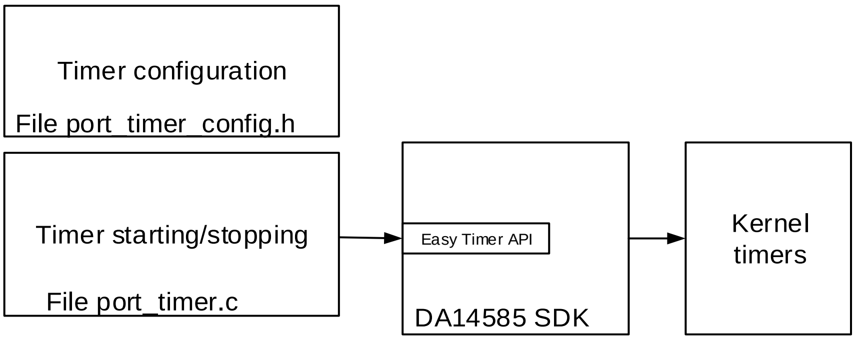


Figure 20: Timer Controller Module Block Diagram

The Timer Controller Module has the following features:

- Optional returning of the current value of a timer when it is cleared.
- Extended timers (longer than 5 minutes).

The Easy Timer API is used to create new timers, modify the delay of existing timers and cancel active timers.

The extended timers and getting the current value of a timer are implemented in file port_timer.c.

Other application modules can register a callback function to be called when their timer expires.

The Timer Controller Module is enabled by defining symbol HAS_TIMERS in the file user_config.h.

Files port_timer.c and port_timer_config.h must be included in the project.

8.15.2 Configuration

Timer parameters are defined in the file port_timer_config.h.

- EXTENDED_TIMERS_ON: Enables the support for extended timers.
- port_timer_cbs[]: Contains pointers to the callback functions that are called when the timers expire.

Timer names are defined in the port_timer_ids enumeration. These names should be used when setting or clearing a timer.

8.15.3 Design Considerations

The maximum delay for non-extended timers is KE_TIMER_DELAY_MAX, which is defined in the file app_easy_timer.h.

 © 2022 Renesas Electronics

8.16 SysTick Controller Module

8.16.1 Description

The SysTick Controller Module enables sharing of the SysTick timer among different application modules. The block diagram of the SysTick Controller Module is depicted in Figure 21.

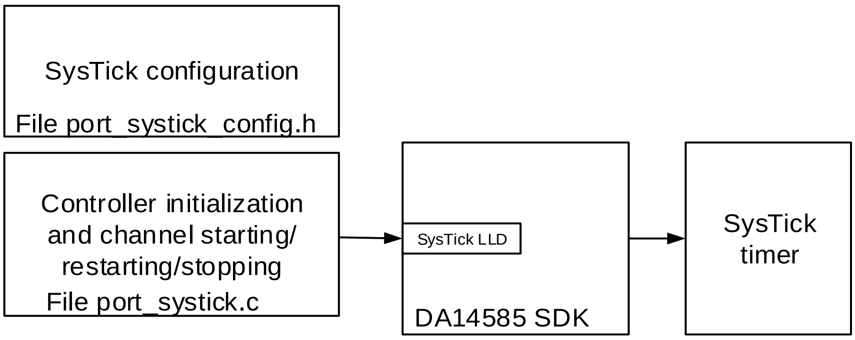


Figure 21: SysTick Controller Module Block Diagram

The SysTick Controller Module has the following features:

- One channel per application module that uses the SysTick timer.
- Configurable base period and channel periods.

The SysTick LLD is used with the base period to create interrupts.

The channels are implemented in the file `port_systick.c`.

Other application modules can register a callback function to be called when their channel period expires.

The SysTick Controller Module is enabled by defining symbol `HAS_SYSTICK` in file `user_config.h`.

Files `port_systick.c` and `port_systick_config.h` must be included in the project.

8.16.2 Configuration

SysTick Controller Module parameters are defined in the file `port_systick_config.h`:

- `SYSTICK_PERIOD_IN_US`: Defines the base period (in µs) for interrupts.
- `systick_config[]`: Contains pointers to the callback functions that are called when the channel periods expire.

Channel names are defined in the `port_systick_channel` enumeration. These names should be used when starting, restarting or stopping a channel.

8.16.3 Design Considerations

As long as at least one channel is active, the SysTick timer is started and active mode is forced. When all channels are stopped, the SysTick timer is stopped and sleep mode is restored.

 © 2022 Renesas Electronics

9 BLE Services

9.1 Dialog Audio Service

The Voice RCU reference design uses a custom service named Dialog Audio Service for the audio feature. The service includes the characteristics listed in Table 25.

Table 25: Dialog Audio Service Characteristics

Description	UUID	Properties (Note 1)	Size (B)
Dialog Audio Service attribute	D803-992D-913D-B0B6-034C-9450-8B10-1DBC	RD	16
Control point characteristic	6ABC-6F0F-0BDC-C2B2-7147-E8EF-AE41-050C	WR, NTF	20
Device configuration characteristic	4695-AC87-3D0E-C6BF-D74D-219C-9F28-F9FD	RD	20
Audio data report characteristic	D13D-0724-3261-878E-E44C-200D-9109-2C8D	NTF	`MAX_MTU_SIZE - 3`

Note 1 RD: read, WR: write, NTF: notify, IND: indicate.

9.1.1 Control Point Characteristic

9.1.1.1 Control Point Commands

The host sends control commands to the Voice RCU by writing the Control Point Characteristic. The available commands are described in Table 26 to Table 30.

Table 26: Control Point Command Structure

Offset (B)	Name	Description
0	Audio stream enable	1: Enable audio stream, 0: Disable audio stream
1	Command	See Table 27 for the list of commands.
2	Command parameters length	The length in bytes of the command parameters. See Table 27 for the values corresponding to each command.
3 to 19	Command parameters	See Table 28, Table 29 and Table 30.

Table 27: Control Point Commands

Value	Name	Parameter Length (B)	Description
0x00	Audio stream enable/disable	0	Enable or disable the audio stream according to the value of the first byte of the control point structure.
0x0M	Change audio mode	0	Change the audio mode: ● M = 2: 64 kbit/s ● M = 3: 48 kbit/s ● M = 4: 32 kbit/s ● M = 5: 24 kbit/s ● M = 6 automatic See Table 9 for audio mode parameters.
0x1M	Force audio mode	0	Same as 0x0M. All 0x0M commands following a 0x1M are ignored.

© 2022 Renesas Electronics

Value	Name	Parameter Length (B)	Description
0x81	Set ATT packet size	4	Set the ATT packet size used for audio data. See Table 28 for command parameters.
0x82	Set connection parameters	8	Trigger a connection parameter update request from the device. See Table 29 for command parameters.
0x83	Read configuration	0	Read the device configuration. A device configuration notification must be sent.
0x84	System reset	0	Issue a system reset
0x85	Emulate key press	4 or 12	Emulate key presses. See Table 30 for command parameters.

Table 28: Set ATT Packet Size Command Parameters

Offset (B)	Parameter	Description
0	Max ATT packet size (LSB)	The maximum ATT packet size. Must be less or equal than MTU. This is used to limit the maximum audio packet size since MTU cannot be changed more than once.
1	Max ATT packet size (MSB)	
2	Fixed ATT packet size (LSB)	The fixed ATT packet size. When set to 0, the audio packet size is calculated automatically, as described in Section 7.5. Otherwise the audio packet size is set to: (`Fixed ATT packet size - 3`).
3	Fixed ATT packet size (MSB)	

Table 29: Set Connection Parameters Command Parameters

Offset (B)	Parameter	Description
0	Min connection interval (LSB)	The minimum connection interval in steps of 1.25 ms.
1	Min connection interval (MSB)	
2	Max connection interval (LSB)	The maximum connection interval in steps of 1.25 ms.
3	Max connection interval (MSB)	
4	Slave latency (LSB)	The slave latency (number of skipped events).
5	Slave latency (MSB)	
6	Supervision timeout (LSB)	The supervision timeout in steps of 10 ms.
7	Supervision timeout (MSB)	

Table 30: Emulate Key Press Command Parameters

Offset (B)	Parameter	Description
0	Initial delay (LSB)	The delay (in ms) before starting the key press emulation.
1	Initial delay (MSB)	
2	Starting column	Consecutive key presses can be emulated. The sequence starts at the key in the starting row and column. The next key is in the next column of the starting row. When the ending column is reached, the row is incremented by one and the column is reset to the starting column. This procedure is repeated until the ending row and column are reached.
3	Starting row	
4	Ending column	
5	Ending row	
6	Press duration (LSB)	The time interval (in ms) between key press and key release.
7	Press duration (MSB)	
8	Repeat counter (LSB)	The total number of emulated key presses.

© 2022 Renesas Electronics

Offset (B)	Parameter	Description
9	Repeat counter (MSB)	
10	Repeat interval (LSB)	The time interval (in ms) between a key release and the next key press.
11	Repeat interval (MSB)	

9.1.1.2 Control Point Notifications

The Voice RCU sends control commands and status reports to the host as notifications, as described in Table 31 to Table 36.

Table 31: Control Point Notification Structure

Offset (B)	Name	Description
0	Audio stream enable	1: Enable audio stream, 0: Disable audio stream
1	Notification type	See Table 32 for the list of notifications.
2	Notification parameters length	The length in bytes of the command parameters. See Table 32 for the values corresponding to each notification.
3 to 19	Notification parameters	See Table 38, Table 33, Table 34, Table 35 and Table 36.

Table 32: Control Point Notifications

Value	Name	Parameter length (B)	Description
0	Audio stream enable/disable command	0	Audio stream enable/disable command according to the value of the first byte of the control point structure. The command may optionally contain configuration parameters as listed in Table 38.
1	Audio stream configuration report	14	This report contains the configuration of the audio stream. See Table 38 for report field descriptions.
2	Keyboard key report	10	This report contains the data of the normal and consumer HID keyboard reports. These reports are used for displaying the key events in the **Remote Controls** Android application. See Table 33 for report field descriptions.
3	Debug info report	7	This report contains debug information for the audio stream. This information is displayed in the logbook page of the **Remote Controls** Android application. See Table 34 for report field descriptions.
4	-		Reserved for backwards compatibility.
5	Connection parameter report	6	This report contains the active BLE connection parameter. It is sent to the host upon connection and whenever new connection parameters are applied. See Table 35 for report field descriptions.
6	ATT packet size report	4	This report contains the ATT MTU and the active audio ATT packet size. It is sent to the host whenever the MTU is changed or the audio ATT packet size is recalculated. See Table 36 for report field descriptions.

Table 33: Keyboard Key Report Fields

Offset (B)	Parameter	Description
0 to 7	Keyboard report data	The data of the HID keyboard report 8 bytes for normal keyboard reports or 3 bytes for consumer keyboard reports.
9	Keyboard layout	1: Number pad page

 © 2022 Renesas Electronics

Offset (B)	Parameter	Description
		2: Navigation page

Table 34: Debug Info Report Fields

Offset (B)	Parameter	Description
0	-	Reserved for backwards compatibility. Must be 0.
1	-	Reserved for backwards compatibility. Must be 3.
2	Stream buffer underrun	Number of dropped frames due to stream buffer underrun.
3	Audio buffer underrun	Number of dropped frames due to audio samples buffer underrun.
4	Stream buffer size	The size of the stream buffer.
5	Stream buffer write pointer	The write pointer to the stream buffer.
6	Stream buffer read pointer	The read pointer to the stream buffer.

Table 35: Connection Parameter Report Fields

Offset (B)	Parameter	Description
0	Connection interval (LSB)	Current BLE connection interval in steps of 1.25 ms.
1	Connection interval (MSB)	
2	Slave latency (LSB)	Current BLE connection slave latency (number of skipped events).
3	Slave latency (MSB)	
4	Supervision timeout (LSB)	Current BLE connection supervision timeout in steps of 10 ms.
5	Supervision timeout (MSB)	

Table 36: ATT packet size report fields

Offset (B)	Parameter	Description
0	ATT packet size (LSB)	Current ATT packet size in bytes.
1	ATT packet size (MSB)	
2	ATT MTU size (LSB)	Current ATT MTU size in bytes.
3	ATT MTU size (MSB)	

9.1.2 Device Configuration Characteristic

The host can read the value of the Device Configuration Characteristic to get the RCU configuration. The structure of the characteristic value is depicted in Table 37.

Table 37: Device Configuration Structure

Offset (B)	Name	Description
0	-	Reserved. Must be 0.
1	-	Reserved. Must be 1.
2	Configuration parameters length	14
3 to 19	Configuration parameters	See Table 38

© 2022 Renesas Electronics

Table 38: Audio Stream Configuration Report Fields

Offset (B)	Parameter	Description
0	Audio stream format	Reserved for HID reports when the Dialog Audio Service is not used. Must be 0.
1	RCU audio stream features	Bit 0 1: In-band control information is used Bit 1 1: Adaptive rate control is used Bit 2 1: Non-packet based audio data 0: Packet based audio data Bit 3 1: Enhanced command set in control point command characteristic as described in Table 27 is supported. Current ATT packet size, ATT MTU size and BLE connection parameters must also be reported as described below. 0: Only the *Audio stream enable/disable* command is supported.
2	ADPCM mode	Bit 4:5 2: Automatic, 1: Fixed Bit 0:3 Current (or fixed) ADPCM mode: 0: 64 kbit/s 1: 48 kbit/s 2: 32 kbit/s 3: 24 kbit/s Value 0 is reserved for backwards compatibility (no ADPCM mode information available).
3	Keyboard layout	1: Number pad page. 2: Navigation page 0: Reserved for backwards compatibility with the DA14582 RCU reference design.
4	ATT packet size (LSB)	Current ATT packet size in bytes.
5	ATT packet size (MSB)	
6	ATT MTU size (LSB)	Current ATT MTU size in bytes.
7	ATT MTU size (MSB)	
8	Connection interval (LSB)	Current BLE connection interval in steps of 1.25 ms.
9	Connection interval (MSB)	
10	Slave latency (LSB)	Current BLE connection slave latency (number of skipped events).
11	Slave latency (MSB)	
12	Supervision timeout (LSB)	Current BLE connection supervision timeout in steps of 10 ms.
13	Supervision timeout (MSB)e	

9.1.3 Audio Data Report Characteristic

The RCU transfers the audio data to the host by sending notifications using the Audio Data Report Characteristic. The length of the data in the notification may be fixed or variable, depending on the configuration of the BLE Stream Module. A variable length can be set using the *Set ATT packet size* command as defined in Table 27. The maximum length of the audio data is (`ATT MTU size – 3`).

© 2022 Renesas Electronics

9.2 HID Over GATT Profile

9.2.1 Description

The HID Over GATT Profile (HOGP) is used to send keyboard, audio and pointing device data to the host. Five types of HID reports are used:

- **Normal keyboard input reports** as defined in Ref. [14]: Key modifiers and keys with code up to 0x65 as defined in Ref. [15] can be reported.
- **Keyboard LED output reports** as defined in Ref. [14]: This report is always included for compatibility reasons, even when keyboard LED functionality is not implemented.
- **Consumer key input reports**: Keys not included in normal keyboard reports can be reported.
- **Mouse input reports**: These reports are used for sending trackpad events to the host.
- **Vendor-defined HID input reports**: These reports are used for sending gyro/accelerometer sensor data to the host. Audio data can also be sent to the host using one or more vendor-defined reports.

Standard keyboard and mouse reports are supported by practically all host operating systems. Therefore, no additional development is needed on the host side for key and trackpad events. On the other hand, there is no default implementation for vendor-defined reports. Special software must be developed to handle audio and gyro/accelerometer data.

The report map (report descriptor) must be defined as described in Ref. [14]. A report descriptor describes each piece of data that the device generates and what the data is actually measuring. Report IDs are used in the report map to identify reports. A zero-based index is used in HOGP to handle HID reports. Report map IDs are associated with HOGP indexes in the HOGP configuration file `user_hogpd_config.h`.

9.2.1.1 Vendor-Defined Reports for Audio Stream

Up to five vendor-defined HID reports are used for the audio stream feature as described in Table 39.

Table 39: Vendor-Defined HID Reports for Audio Stream Functionality

Name	Report ID	Report Type	Size (B)	Description
STREAM_CTRL_OUT	4	Output	20	The host uses this report to send command to the RCU as specified in Table 26.
STREAM_CTRL_IN	5	input	20	The RCU uses this report to send commands and configuration notifications to the host as specified in Table 31. The host can read this report to get the RCU configuration as specified in Table 37.
AUDIO_DATA_1	6 (Note 1)	input	`ATT MTU size - 3`	The RCU uses these reports to send audio data to the host as specified in Section 9.1.3. Either only the first or all three reports can be used depending on the configuration of the application.
AUDIO_DATA_2	7	input	`ATT MTU size - 3`	
AUDIO_DATA_3	8	input	`ATT MTU size - 3`	

Note 1 When only one HID report is used for audio data then the report ID is specified in device configuration.

When using HID reports for audio data the `Audio stream format` field of the device configuration notification (specified in Table 37) must be set to the proper value:

- **Set to 0:** Use all three audio reports (AUDIO_DATA_1, AUDIO_DATA_2 and AUDIO_DATA_3) for sending audio data notifications. The report IDs must have values 6, 7 and 8, respectively. This configuration enables backwards compatibility with previous RCU reference designs.

 © 2022 Renesas Electronics

- **Set to report ID**: Use one specific report type for sending audio data notifications. The report ID can have any value.

9.2.1.2 Vendor-Defined Report for Gyro/Accelerometer Sensor

A vendor-defined HID report is used for sending data from the gyro/accelerometer sensor to the host. The report ID is fixed and set to 8. The structure of the notification is depicted in Table 40.

Table 40: Vendor Defined HID Reports for Gyro/Accelerometer Sensor

Offset (B)	Name	Description
0 to 3	Timestamp	A 32-bit counter is used for data timestamping. The counter is incremented by one before the transmission of each notification.
4 to 5	Temperature	The sensor temperature as provided by the gyro/accelerometer sensor.
6 to 7	Accelerometer X	Accelerometer X,Y and Z data.
8 to 9	Accelerometer Y	
10 to 11	Accelerometer Z	
12 to 13	Gyro X	Gyroscope X,Y and Z data.
14 to 15	Gyro Y	
16 to 17	Gyro Z	

18 to 19	Button state	The state of the buttons used for mouse clicks. Currently only the left mouse button is supported.
		`1: Button is pressed.`
		`0: Button is released.`

9.2.2 Configuration

The HOGP configuration is defined in file `user_hogpd_config.h`.

Structure `hogpd_params` defines the configuration parameters of HOGP:

- `boot_protocol_mode`: Set to true to use boot protocol mode is used as defined in [14]
- `batt_external_report`: Set to true to use external report reference to battery service
- remote_wakeup: Set to true to use remote wakeup mode. Remote Host may not handle properly remote wakeup when the inactivity timeout is on. Some Hosts do not expect to receive `LL_TERMINATE_IND` from wakeup capable devices while they are sleeping.
- `normally_connectable`: Set to true to set normally connectable mode. Inactivity timeout cannot be used in this mode. The device is always in advertising mode when not connected to the host.
- `store_attribute_callback`: This callback function is call when a HOGP attribute or CCC is changed to store the new value in the non-volatile memory. This value will be restored every time the device is connected to the host.
- `hogpd_indexes`: This enumeration defines the zero-based index of all HID reports. The last entry named `HID_NUM_OF_REPORTS` defines the total number of reports.
- `hogpd_reports`: This table defines the configuration for every HID report. Each entry contains the following parameters:
  - `id`: The report ID as defined in the HID report map
  - `size`: The report size in bytes
  - `cfg`: Set the report configuration by setting the following bit masks:
  - `HOGPD_CFG_REPORT_IN`: HID input report. Set `HOGPD_CFG_REPORT_WR` to enable report write capabilities
  - `HOGPD_CFG_REPORT_OUT`: HID output report

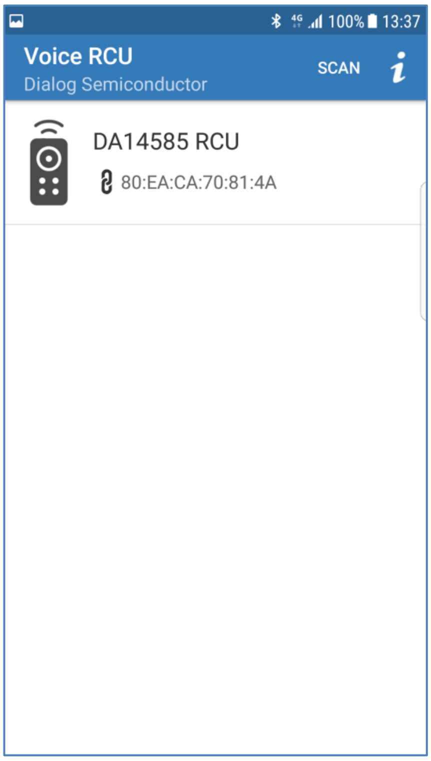
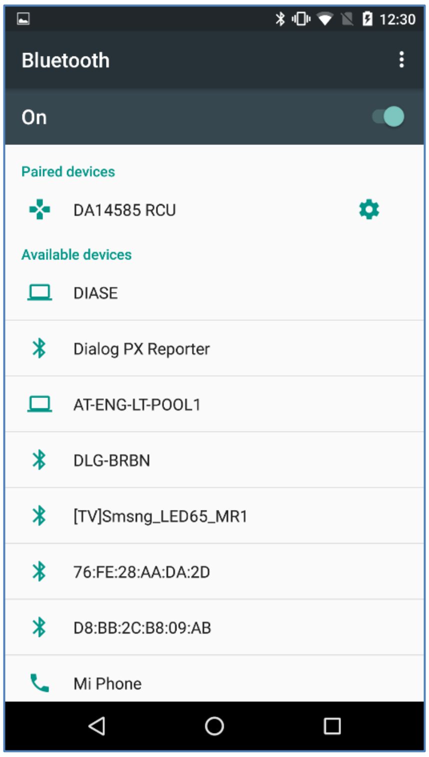
© 2022 Renesas Electronics

  - HOGPD_CFG_REPORT_FEAT: HID feature report
  - read_callback: This callback function is called when the report is read by the host. This function must provide the report data.
  - write_callback: This callback function is called when the report is written by the host
- report_map: The report map (report descriptor) as defined in Ref. [14]. The following reports are included in the reference design application:
  - **Normal keyboard report (report ID 1)**: This is the first report and should not be modified.
  - **Keyboard LED report (report ID 2):** This is the second report and should not be modified even if keyboard LEDs are not used for compatibility reasons.
  - **Consumer key report (report ID 3)**: A report consisting of three bytes has been included as an example to demonstrate how to include various consumer keys, although many of them are not used in the application and could be removed. A set of symbols has been defined to facilitate the use of consumer keys in the application: [key_name]_BYTE, [key_name]_BIT, [key_name]_MASK and [key_name]_CODE.
  - **Five vendor-defined reports for audio data (report IDs 4, 5, 6, 7 and 8)**: These reports are only used when the Dialog Audio Service is not used.
  - **One vendor-defined report for motion data (report ID 9):** This report is used for sending gyro/accelerometer data to the host.
  - **A mouse report (report ID 0x1A)**: This report is used for sending trackpad events to the host. A consumer usage page is included, although it is not used, for compatibility reasons.

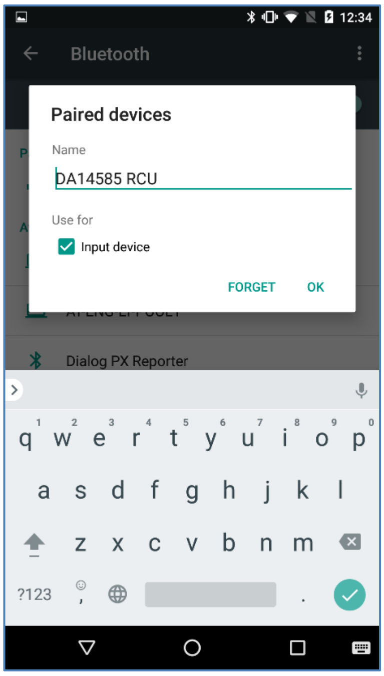
© 2022 Renesas Electronics

Appendix A Reconnect the RCU from Scratch

Table 41: Reconnect the RCU from Scratch

1	If the RCU was already paired with your Android device, it will show up with a chain icon next to its BD address.	
2	Press keys **7** and **9** on the RCU simultaneously to clear its bonding data and open the Bluetooth settings of your device.	

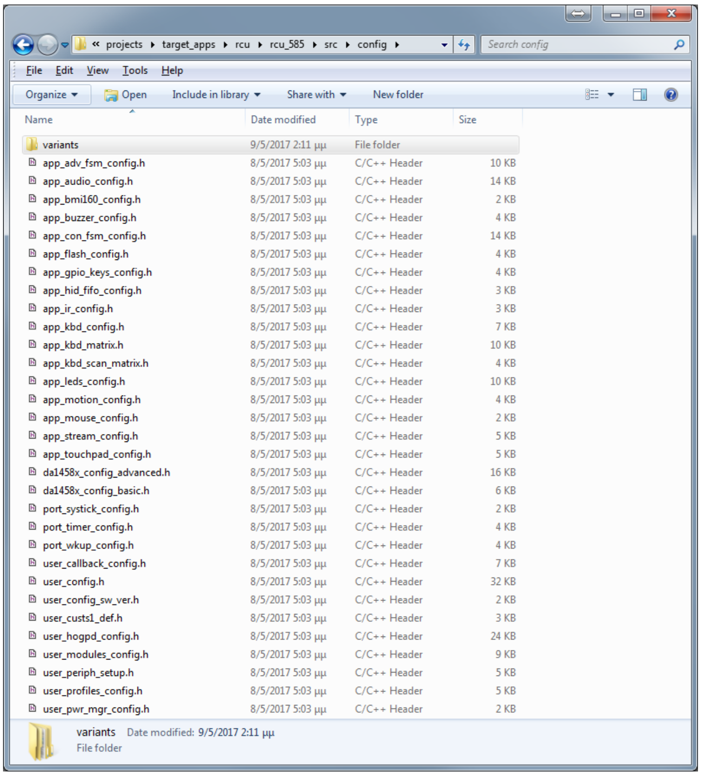
© 2022 Renesas Electronics

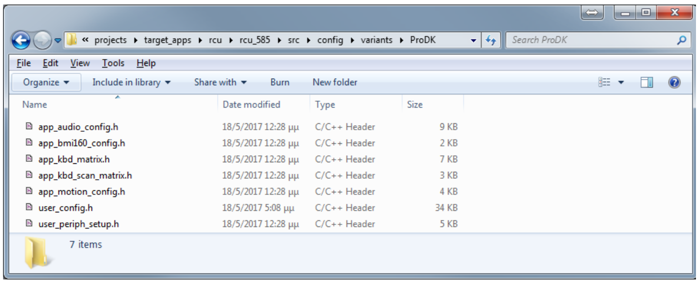
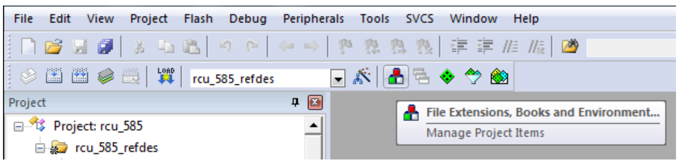
3	Tap the cogwheel icon next to the RCU name and select **FORGET** in the screen that pops up.	

 © 2022 Renesas Electronics

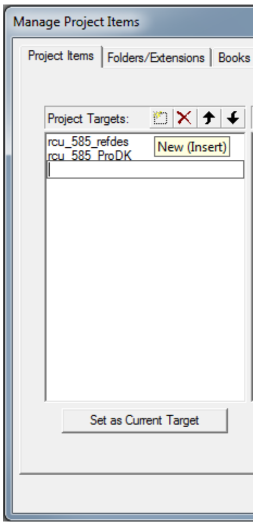
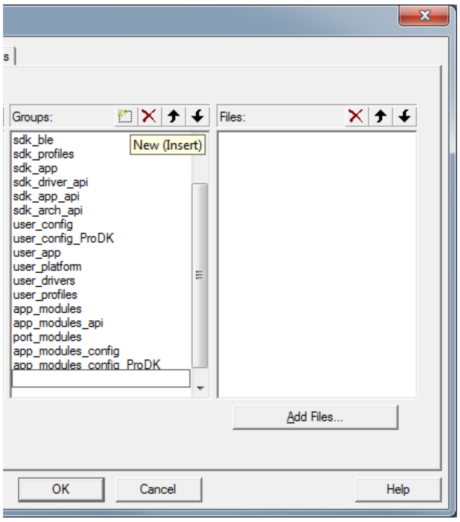
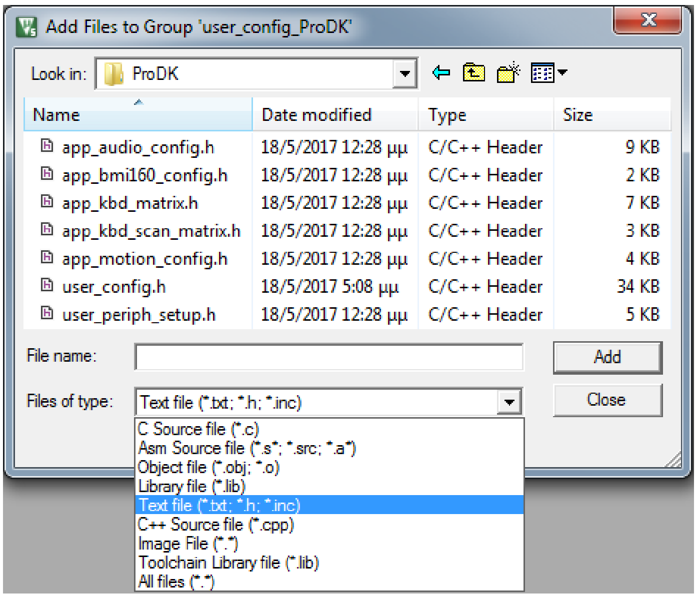
Appendix B Build and Download the Firmware to Other Hardware

Table 42: Build and Download the Firmware to Other Hardware

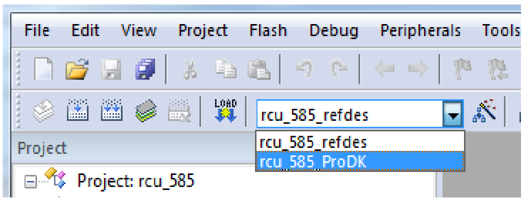
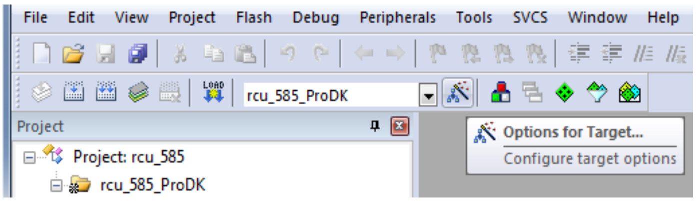
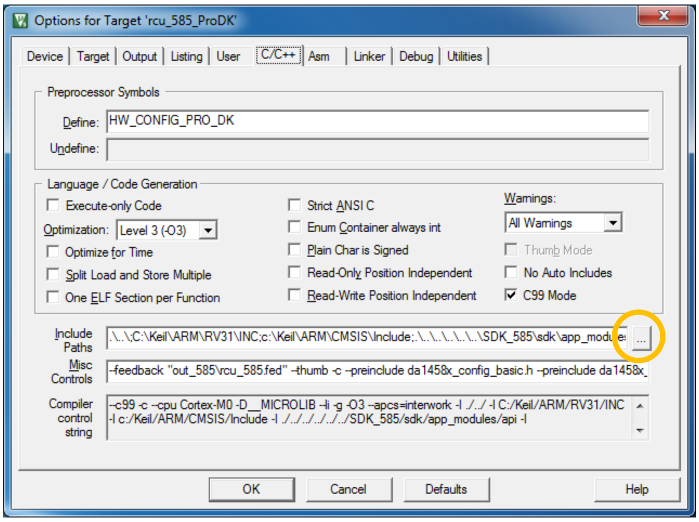
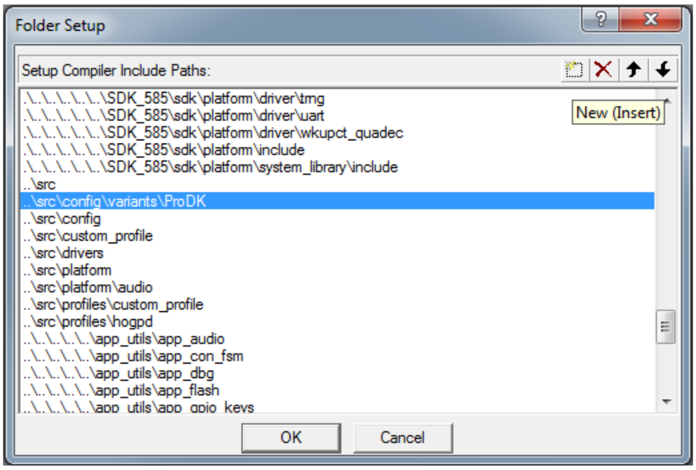
<table>
<tr><td>1</td><td>The default configuration files can be found in folder <code>projects\target_apps\rcu\rcu_585\src\config</code>.</td><td>
</td></tr>
<tr><td>2</td><td>To override some configuration files for a different target, first copy them to a new subfolder in folder <code>\variants</code>.

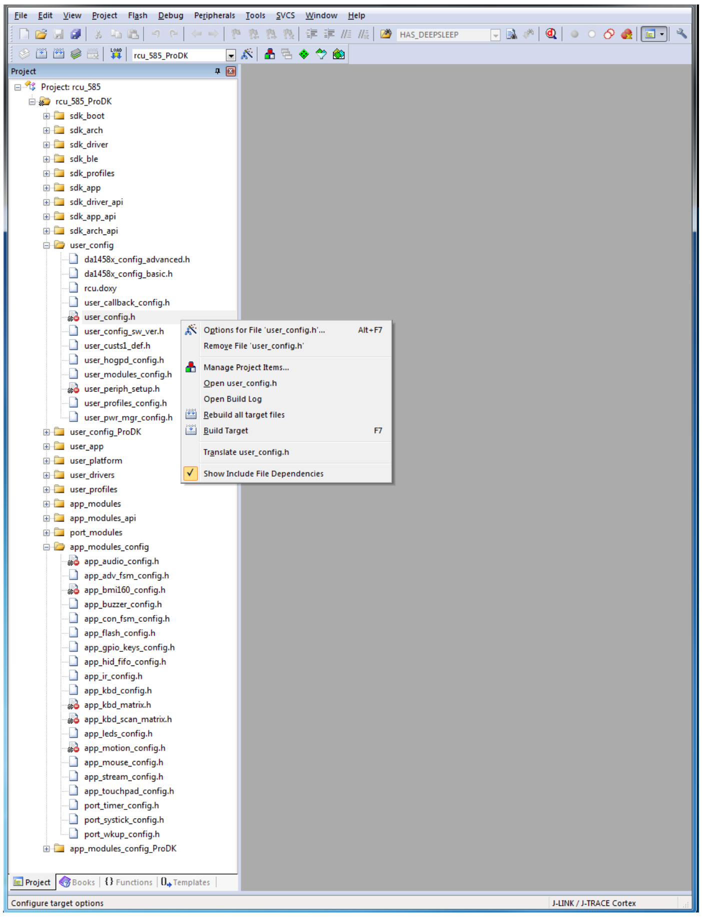
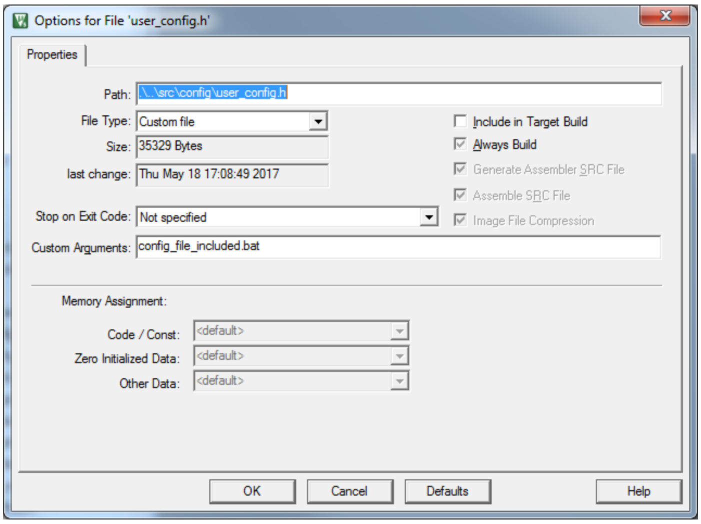
The contents of folder <code>variants\ProDK</code> are shown here and explained in Section 7.6.</td><td>
</td></tr>
<tr><td>3</td><td>Click icon File Extensions, Books and Environment… on the Build toolbar.</td><td>
</td></tr>
</table>

© 2022 Renesas Electronics

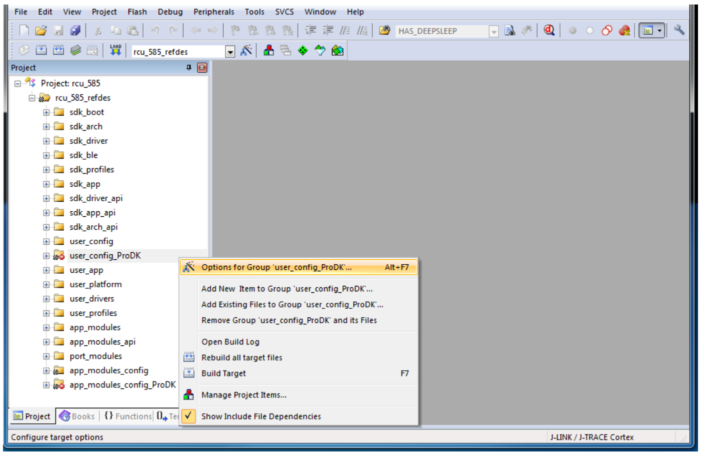
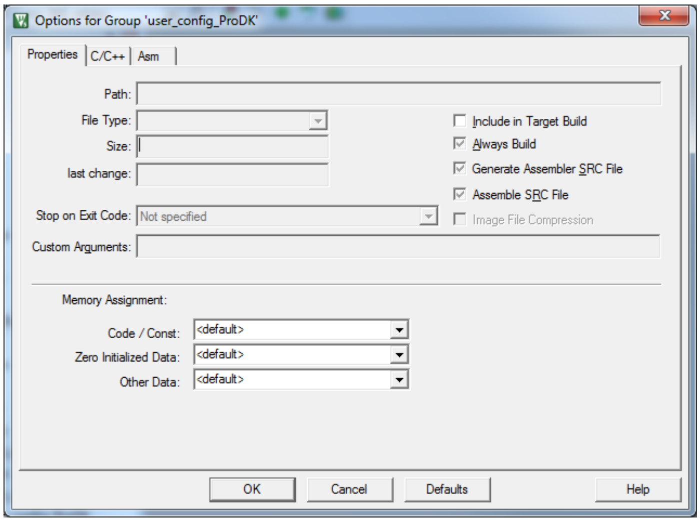
<table>
<tr>
<td>4</td>
<td>The Manage Project Items window appears. Click button New (Insert) in the Project Targets column and enter the name of your target device.</td>
<td>
</td>
</tr>
<tr>
<td>5</td>
<td>Use button New (Insert) in the Groups column to create two groups for the user configuration files (e.g. <code>user_config_xxx</code>) and the modules configuration files (e.g. <code>app_modules_config_xxx</code>).</td>
<td>
</td>
</tr>
<tr>
<td>6</td>
<td>Click button Add Files… to open the Add Files to Group window. Browse to the folder containing your configuration files, select them as Text files and click button Add to the add:
Files <code>da1458x_config_*.h</code> and <code>user_*.h</code> to group <code>user_config_xxx</code>.
Files <code>app_*.h</code> and <code>port_*_config.h</code> to group <code>app_modules_config_xxx</code>.</td>
<td>
</td>
</tr>
</table>

© 2022 Renesas Electronics

7	Select your target.	
8	Open the target's options from the Build toolbar.	
9	Navigate to the C/C++ tab and click the **ellipsis (…)** button next to the Include Paths.	
10	Click button **New (Insert)** to add the relative path of your configuration files above the default path (`..\src\config`).	

11	Select the files you copied from the default `user_config` and `modules_config` groups and open their options by right clicking or pressing **Alt+F7**.	
12	Make sure that option **Include in Target Build** is **not** checked. Do the same for the **groups** that were created previously for other targets.	

 © 2022 Renesas Electronics

13	Change to the other targets, select **your groups** and open their options by right clicking or pressing **Alt+F7**.	
14	Again make sure that option **Include in Target Build** is **not** checked.	

 © 2022 Renesas Electronics

B.1 ProDK Kit Configuration

The ProDK development kit can be used with DA14585 in a QFN40 package.

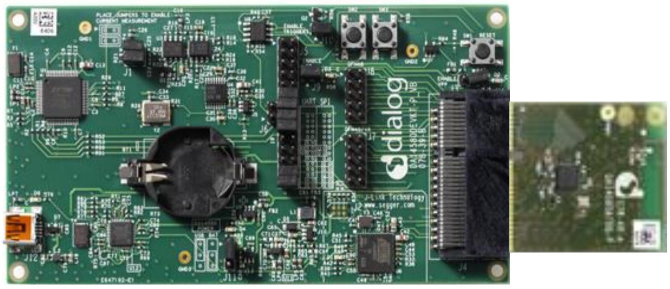


Figure 22: ProDK with a QFN40 DA14585

Its specific configuration is included in seven header files described in the next sub-sections.

B.1.1 app_audio_config.h

A PDM microphone can be connected to J5 and J7 of the ProDK.

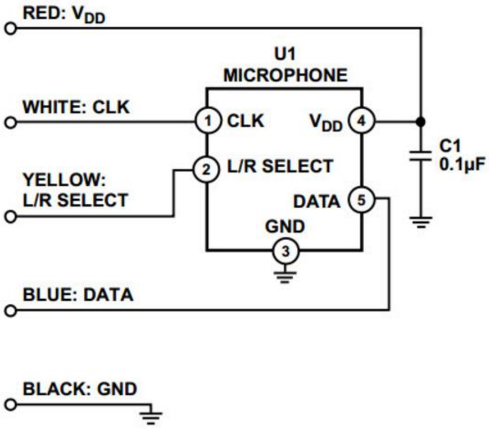


Figure 23: Microphone Pins and Cables

The audio pin configuration is:

```
static const pin_type_t app_audio_pins[] = {
    [AUDIO_CLK_PIN]  = {.port = GPIO_PORT_2, .pin = GPIO_PIN_1, .high = 0, .mode_function = OUTPUT | PID_PDM_CLK },
    [AUDIO_DATA_PIN] = {.port = GPIO_PORT_2, .pin = GPIO_PIN_0, .high = 0, .mode_function = INPUT  | PID_PDM_DATA},
};
```

Table 43: Microphone Connection

Microphone		ProDK	
Cable Color	**Function**	**Pin**	**Function**
White	CLK	J7_2	P2_1
Yellow	L/R SELECT	J5_2	GND
Black	GND	J5_2	GND
Red	VDD	J5_1	VBAT_580
Blue	DATA	J7_1	P2_0

 © 2022 Renesas Electronics

B.1.2 app_bmi160_config.h

A BMI160 Shuttle Board can be connected to the I2C or SPI interface depending on the definition of MOTION_IF in the file.

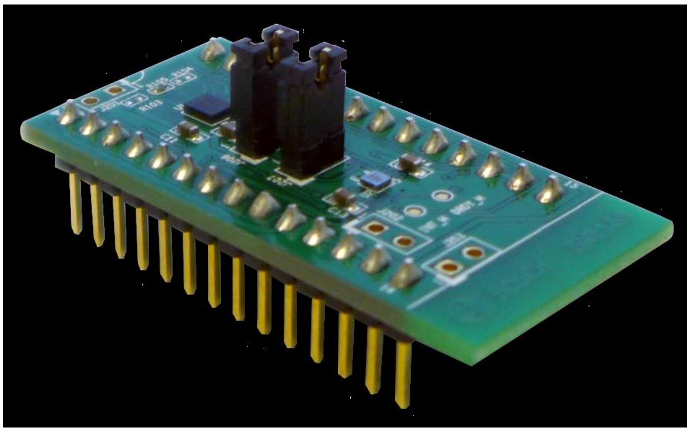


Figure 24: BMI160 Shuttle Board

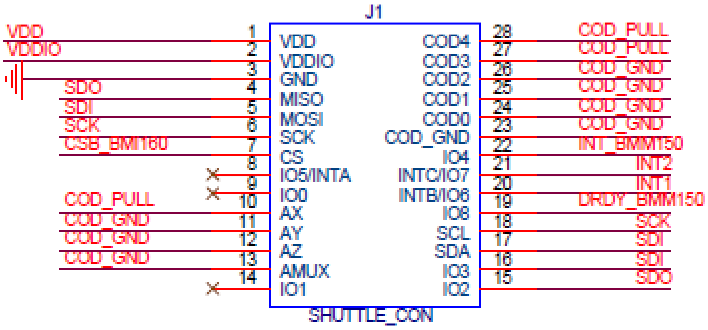


Figure 25: Shuttle Board Connector

© 2022 Renesas Electronics

B.1.3 app_kbd_matrix.h

A 4 x 4 keyboard can be connected to the ProDK.

Figure 26: 4 x 4 Keyboard

The key map is:

```
static const uint16_t kbd_keymap[KBD_NR_SETS][KBD_NR_ROW_OUTPUTS][KBD_NR_COLUMN_INPUTS] =
{
  {
    /* 0          1       2           3
      ---------------------------------------------
      1           2       3           Combination key
      4           5       6           Power
      7           8       9           Vol+
      AUDIO       0       MOTION      Vol-
      -----------------------------------------------*/
    { 0x001E,     0x001F, 0x0020,     COMB_MOD_KEY, }, // 0
    { 0x0021,     0x0022, 0x0023,     POWER_KEY     }, // 1
    { 0x0024,     0x0025, 0x0026,     VOL_PLUS_KEY, }, // 2
    { AUDIO_KEY,  0x0027, MOTION_KEY, VOL_MINUS_KEY,}, // 3
  }
};
```

Table 44: Special Keys

Key	Function
*	Audio
#	Motion
A	Combination
B	Power
C	Volume +
D	Volume -

 © 2022 Renesas Electronics

The IR key map is defined in Table 47.

B.1.4 app_kbd_scan_matrix.h

The keyboard can be connected to J7 of the ProDK.

Table 45: Keyboard Definitions

Name	Value
KBD_NR_COLUMN_INPUTS	4
KBD_NR_ROW_OUTPUTS	4
COLUMN_INPUT_0_PORT	2
COLUMN_INPUT_0_PIN	8
COLUMN_INPUT_1_PORT	2
COLUMN_INPUT_1_PIN	6
COLUMN_INPUT_2_PORT	2
COLUMN_INPUT_2_PIN	4
COLUMN_INPUT_3_PORT	2
COLUMN_INPUT_3_PIN	3
ROW_OUTPUT_0_PORT	2
ROW_OUTPUT_0_PIN	7
ROW_OUTPUT_1_PORT	2
ROW_OUTPUT_1_PIN	5
ROW_OUTPUT_2_PORT	2
ROW_OUTPUT_2_PIN	2
ROW_OUTPUT_3_PORT	2
ROW_OUTPUT_3_PIN	9
POWER_BUTTON_COLUMN	3
POWER_BUTTON_ROW	1
DELAYED_WAKEUP_COLUMN	0
DELAYED_WAKEUP_ROW	0

Table 46: Keyboard Connection

Keyboard Line (Note 1)	Keyboard Function	J7 Pin	P2 pin
1	Row 0	8	7
2	Row 1	6	5
3	Row 2	3	2
4	Row 3	10	9
5	Column 0	9	8
6	Column 1	7	6
7	Column 2	5	4
8	Column 3	4	3

Note 1 The keyboard lines are numbered 1 to 8 from left to right.

© 2022 Renesas Electronics

B.1.5 app_motion_config.h

When `MOTION_IF = SPI`, the chip select pin configuration is:

```
static const pin_type_t app_motion_cs_pin[] = {
    [MOTION_SPI_CS_PIN]  = {.port = GPIO_PORT_0, .pin = GPIO_PIN_2, .high = 1, .mode_function = INPUT_PULLUP | PID_GPIO },
};
```

B.1.6 user_config.h

The user can define/undefine specific identifiers to include/exclude specific modules.

Table 47: Module Configuration

<table>
<tr><th>Definition</th><th>Functionality</th><th>Notes</th></tr>
<tr><td>HAS_KBD</td><td>Keyboard matrix scanner</td><td>Enables the HID report FIFO, wakeup controller sharing and timer handling.
Can emulate BLE packet loss by turning the RF radio off.</td></tr>
<tr><td>HAS_GPIO_KEYS</td><td>Keys connected to GPIO pins</td><td>Enables wakeup controller sharing and SysTick sharing.</td></tr>
<tr><td>HAS_AUDIO</td><td>Audio</td><td>Enables the BLE stream and logs audio and stream buffer statistics.
Uses the Dialog Audio service or vendor-defined HID reports to control audio and stream audio data to the host.
Acquisition starts when the corresponding command is received from the host.
Can use vendor-defined HID reports and start acquisition as soon as the corresponding button is pressed.</td></tr>
<tr><td>HAS_MOTION</td><td>Motion sensor</td><td>Enables timer handling.</td></tr>
<tr><td>HAS_IR</td><td>IR transmitter</td><td>The key map is:<pre>
static const uint16_t kbd_ir_keymap[][] =
{
  /*
      0   1   2   3
      ----
      1   2   3   unused
      4   5   6   unused
      7   8   9   vol+
      unused   0   unused   vol-
      ------------------------------------*/

    { 0x0001,   0x0002,   0x0003,   0x0000},   // 0
    { 0x0004,   0x0005,   0x0006,   0x0000},   // 1
    { 0x0007,   0x0008,   0x0009,   0x0010},   // 2
    { 0x0000,   0x0000,   0x0000,   0x0011},   // 3
};
</pre></td></tr>
<tr><td>HAS_TOUCHPAD_TRACKPAD</td><td>Trackpad</td><td>Enables wakeup controller sharing.</td></tr>
<tr><td>HAS_TOUCHPAD_SLIDER</td><td>Slider</td><td>Enables the HID report FIFO and wakeup controller sharing.</td></tr>
<tr><td>HAS_LED_INDICATORS</td><td>LED indicators</td><td>Enables timer handling.</td></tr>
<tr><td>HAS_SOUND_INDICATION</td><td>Buzzer as sound indicator</td><td></td></tr>
<tr><td>HAS_MOUSE</td><td>Mouse sensor</td><td>Is not implemented.</td></tr>
<tr><td>HAS_CONNECTION_FSM</td><td>Pairing/bonding to one or multiple hosts</td><td>Enables timer handling.</td></tr>
<tr><td>HAS_ACTION_INACTIVITY_TIMEOUT</td><td>SysTick hit some time after the last user action or wakeup interrupt</td><td>Enables SysTick sharing.
The timeout is 100 ms.</td></tr>
</table>

 © 2022 Renesas Electronics

Definition	Functionality	Notes
HAS_PWR_MGR	Power management	Enables timer handling.
HAS_POWERUP_BUTTON	Keyboard button turns the system on or off	Can detect the power button being pressed for more than some time.
HAS_SPI_FLASH_STORAGE	SPI flash used for storing parameters	The bonding info base address is 0x39000. The debug info base address is 0x3A000.
HAS_I2C_EEPROM_STORAGE	I2C EEPROM used for storing parameters	The slave device address is 0x50, the addressing mode is 7-bits and the address size is 2 bytes. The EEPROM size is 8192 bytes and its page size is 32 bytes. The speed is fast (400 kbit/s). The bonding info base address is 0.
HAS_DEEPSLEEP	Device goes into deep sleep when idle	

B.1.7 user_periph_setup.h

B.1.7.1 Shuttle Board Connection over I2C

The Shuttle Board can be connected to J5 and J7 of the ProDK.

Table 48: I2C Pin Configuration

Definition	Value
I2C_SDA_PORT	GPIO_PORT_2
I2C_SDA_PIN	GPIO_PIN_0
I2C_SCL_PORT	GPIO_PORT_2
I2C_SCL_PIN	GPIO_PIN_1

Table 49: Shuttle Board Connection over I2C

Shuttle Board		ProDK	
Pin	Function	Pin	Function
1	VDD	J5_1	VBAT_580
2	VDDIO	J5_1	VBAT_580
3	GND	J5_2	GND
15	Address Select	J5_2	GND
17	SDA	J7_1	P2_0
18	SCL	J7_1	P2_1

 © 2022 Renesas Electronics

B.1.7.2 Shuttle Board Connection over SPI

The Shuttle Board can be connected to J5 and J6 of the ProDK.

Table 50: SPI Pin Definitions

Definition	Value
SPI_CLK_PORT	GPIO_PORT_0
SPI_CLK_PIN	GPIO_PIN_0
SPI_DO_PORT	GPIO_PORT_0
SPI_DO_PIN	GPIO_PIN_6
SPI_DI_PORT	GPIO_PORT_0
SPI_DI_PIN	GPIO_PIN_5

Table 51: Shuttle Board Connection over SPI

Shuttle Board		ProDK (Note 1)		SPI	
Pin	Function	Pin	Function	Pin	Function
1	VDD	J5_1	VBAT_580		
2	VDDIO	J5_1	VBAT_580		
3	GND	J5_2	GND		
4	SDO	J5_13	P0_5	J6_2	MISO
5	SDI	J5_15	P0_6	J6_1	MOSI
6	SCK	J5_21	P0_0	J5_22	CLK
7	CS	J5_9	P0_2		

Note 1 Jumpers for J5 and J6 must be configured for SPI as described in the ProDK user manual (Ref. [4]).

© 2022 Renesas Electronics

Appendix C Create SUOTA Image

Folder `projects\target_apps\rcu\rcu_585` contains a Windows batch file (`mk_suota_img.bat`) that creates a SUOTA image from `rcu_585.hex`, which is the output of building the firmware in Keil.

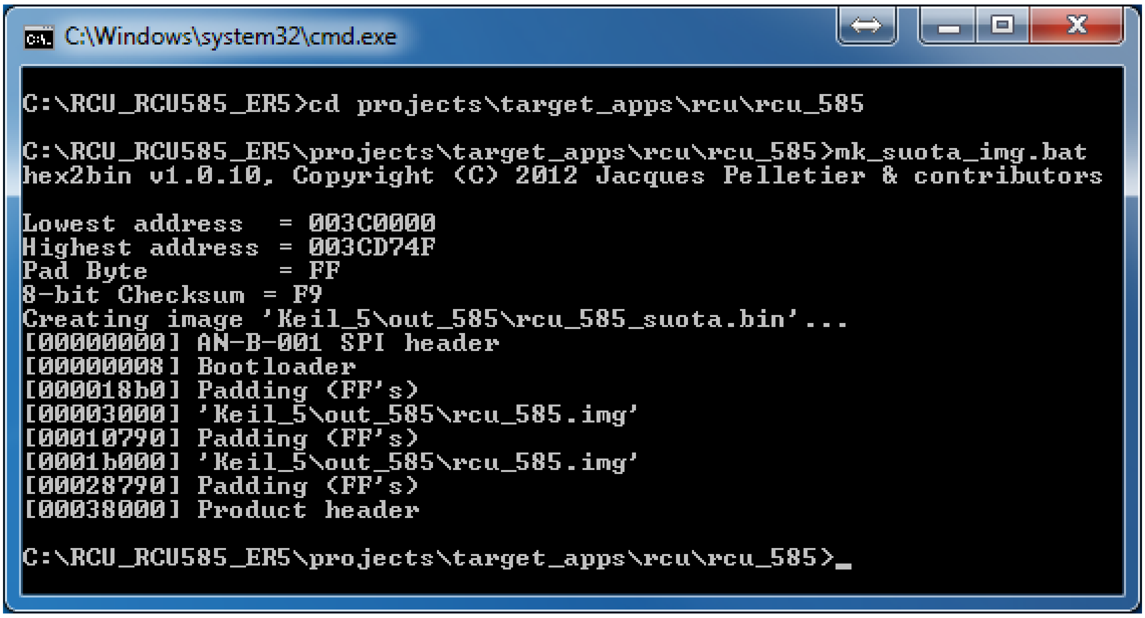


Figure 27: Create SUOTA Image

The SUOTA image (`rcu_585.img`) is stored in `projects\target_apps\rcu\rcu_585\Keil_5\out_585` and can be used to update the RCU software over the air from an iOS or Android device, as described in Application Note AN-B-10 (Ref. [7]).

 © 2022 Renesas Electronics

Appendix D Slider Gestures

Table 52: Slider Gestures

Slide your finger counterclockwise to decrease the volume.	
Slide your finger clockwise to increase the volume.	

 © 2022 Renesas Electronics

Slide your finger right to left to mute/unmute.	
Slide your finger left to right to make a selection.	
Tap on the points shown to move up, down, left or right.	

 © 2022 Renesas Electronics

Revision History

Revision	Date	Description
1.1	24-Dec-2021	Updated logo, disclaimer, copyright.
1.0	21-Jul-2017	Initial version.

 © 2022 Renesas Electronics

Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

© 2022 Renesas Electronics