User's Manual

Renesas e² studio 2021-04 or higher

User's Manual: Quick Start Guide

Renesas MCU
RA Family

Renesas Electronics
www.renesas.com

Rev.1.00 Jul 2021

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity.
   Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between VIL (Max.) and VIH (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between VIL (Max.) and VIH (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system- evaluation test for the given product.

Table of Contents

1. Overview

Renesas e² studio is the Integrated Development Environment for Renesas embedded microcontrollers. e² studio is based on the industry-standard open-source Eclipse IDE framework and the C/C++ Development Tooling (CDT) project, covering build (editor, compiler, and linker control) and debug phases with extended GNU Debug (GDB) interface support.

The e² studio IDE provides support for the Renesas Flexible Software Package (FSP), which is an optimized software package designed to provide easy to use, scalable, high quality software for embedded system design. The primary goal of FSP is to provide lightweight, efficient drivers that meet common use cases in embedded systems.

The e² studio IDE includes multiple Graphical User Interface (GUI) wizards for auto-generating code, including and configuring existing drivers, configuring build and debug options, and running the applications you create. Driver documentation is integrated in the form of tooltips, which are available in the code editor view.

The Renesas FSP support is included in e² studio releases 2021-04 (64-bit) and higher. Multiple views and editors are available to support specifically Renesas RA microcontrollers and the open-source GNU Arm Embedded Toolchain.

The e² studio IDE also supports Arm® TrustZone® technology. Arm® TrustZone® technology divides the system and the application into Secure and Non-Secure partitions. e² studio helps users to set up new TrustZone enabled projects. It also provides debugging features for both secure and non-secure applications on Renesas devices with Arm® TrustZone® technology. Refer to this link for more information about RA Arm® TrustZone® tools, https://www.renesas.com/sg/en/document/apn/ra-arm-trustzone-tooling-primer.

This user manual targets Non-TrustZone devices and Flat (Non-TrustZone) Projects in TrustZone devices.

When using a 3rd-party IDE and toolchain, you can use the Renesas RA Smart Configurator to configure the software system (BSP, drivers, RTOS and middleware) for a Renesas RA microcontroller.

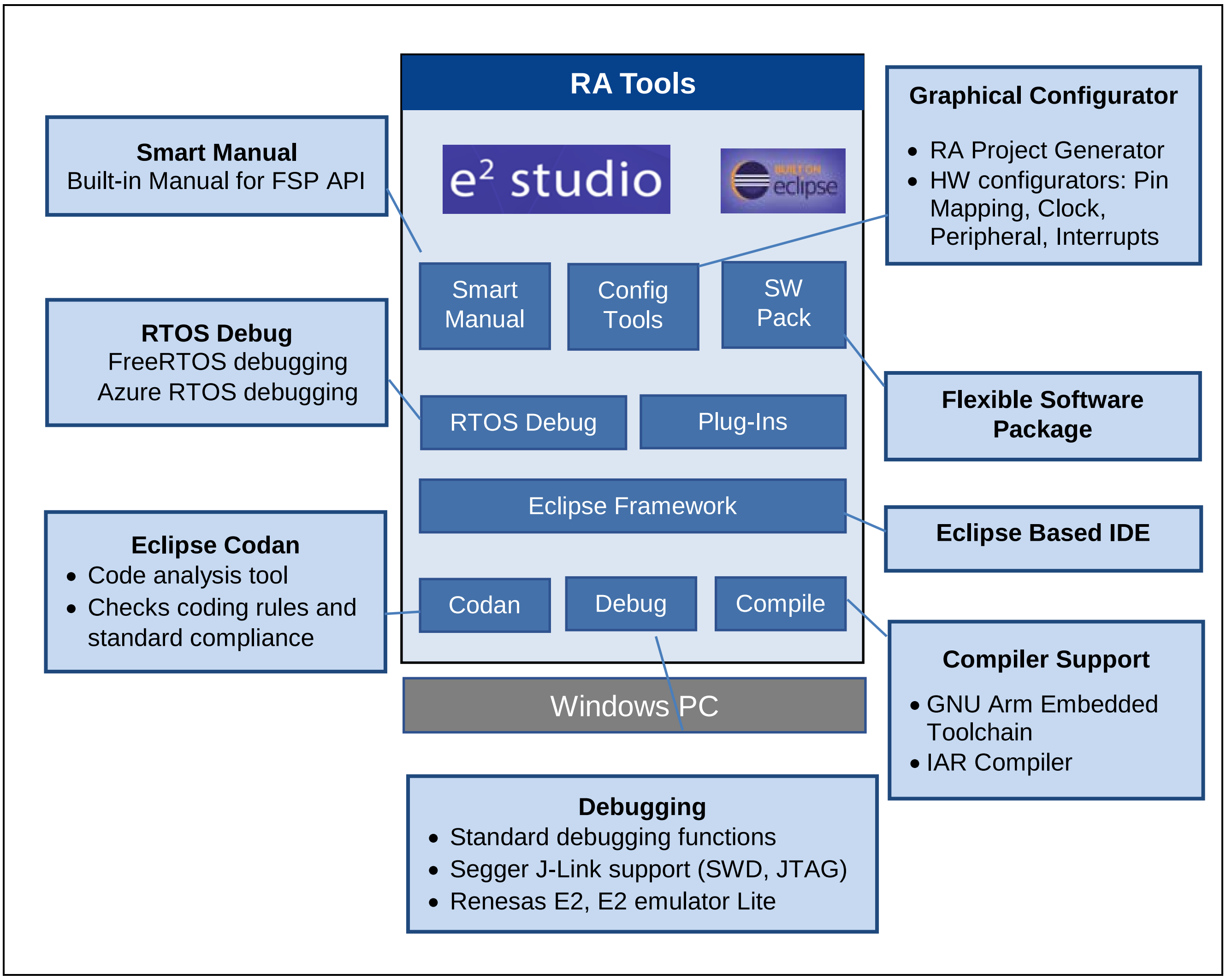

Figure 1. Renesas RA in e² Studio

1.1 System Configuration

A typical system configuration includes a host machine and a target board as shown below.

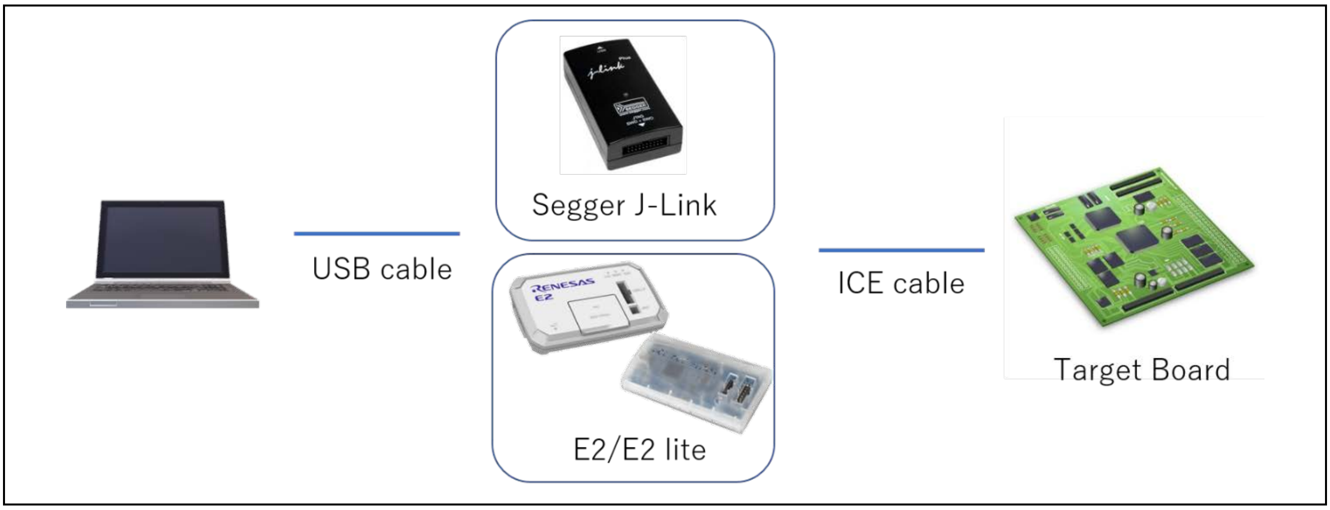

Figure 2. System Configuration

1.2 System Requirements

1.2.1 Hardware Environment:

- Processor: At least 1 GHz (hyper-threading/multi-core CPU)
- Main Memory: At least 2 GB of free memory space
- Hard disk Capacity: At least 2 GB of free space
- Display: Resolution at least 1,024 x 768; at least 65,536 colors
- Interface: USB 2.0 (High-speed/Full-speed, High-speed recommended)

1.2.2 Operating Environment:

Architecture	Windows	e² studio
64-bit version	Windows 10	2021-04

Note: 64-bit OS is required for e² studio 2021-04 and higher.

1.3 Supported Toolchains

- GNU Arm Embedded Toolchain (version: 9-2020-q2-update)
- IAR Compiler 8.50.9 or later
- Arm Compiler (version: 6.16 or later)

1.4 Supported Emulator Devices

- Segger J-Link, E2, E2 emulator Lite

1.5 Outline of RA Project Development

This document provides detailed instructions on how to start developing with Renesas RA. The main steps are outlined below. Understanding these main steps can you relate better to the procedures described in Sections 3 and 4.

- Generating a RA project
- Configuring the RA project to fit hardware specifications such as clock, ICU, and pin functions
- Configuring FreeRTOS
- Configuring Azure RTOS
- Configuring the BSP (selecting HAL driver models)
- Adding user code
- Building the project
- Configuring the debugger and launching debugging

2. Installation

The development tools can be installed using either the “FSP with e² studio Installer” or the standard e² studio Installer.

2.1 Installing the FSP with e² studio Installer

The FSP with e² studio Installer includes the e² studio tool, FSP packs, GCC toolchain, and other tools required to use this software. To download and install the FSP with e² studio Installer, follow the steps below:

Visit the GitHub page of Flexible Software Package (FSP) for Renesas RA MCU Family:
https://github.com/renesas/fsp/releases

Select FSP with e² studio Installer (for example, setup_fsp<version>_e2s_<version >.exe) and click the link to download directly.

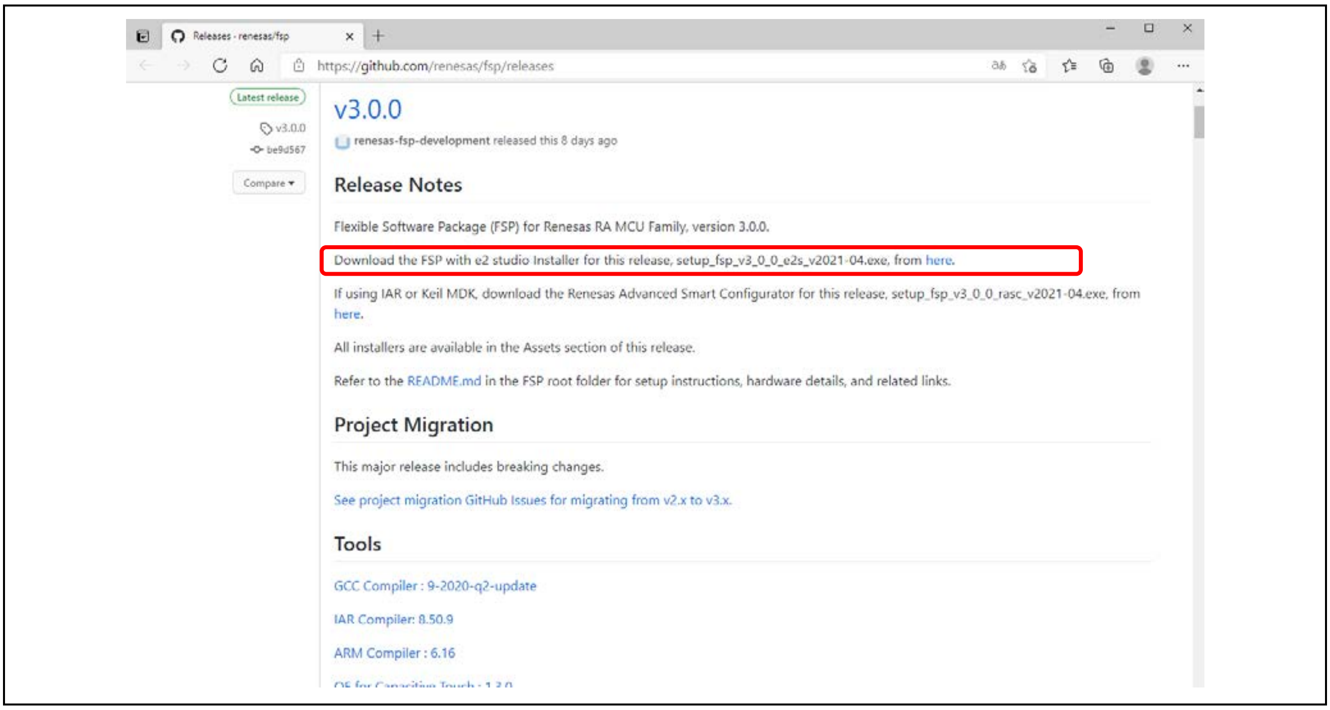

Figure 3. Installation – Download the FSP Package

Run the installation file.

In the **Select Install Type** page, if you would like to customize the components to be installed, choose **Custom Install** then click **Next**.

New users are recommended to select the **Quick Install** option to minimize the configuration steps. This option will install e² studio, FSP, and GCC ARM Embedded by default. If **Quick Install** is selected, the last step will not be shown.

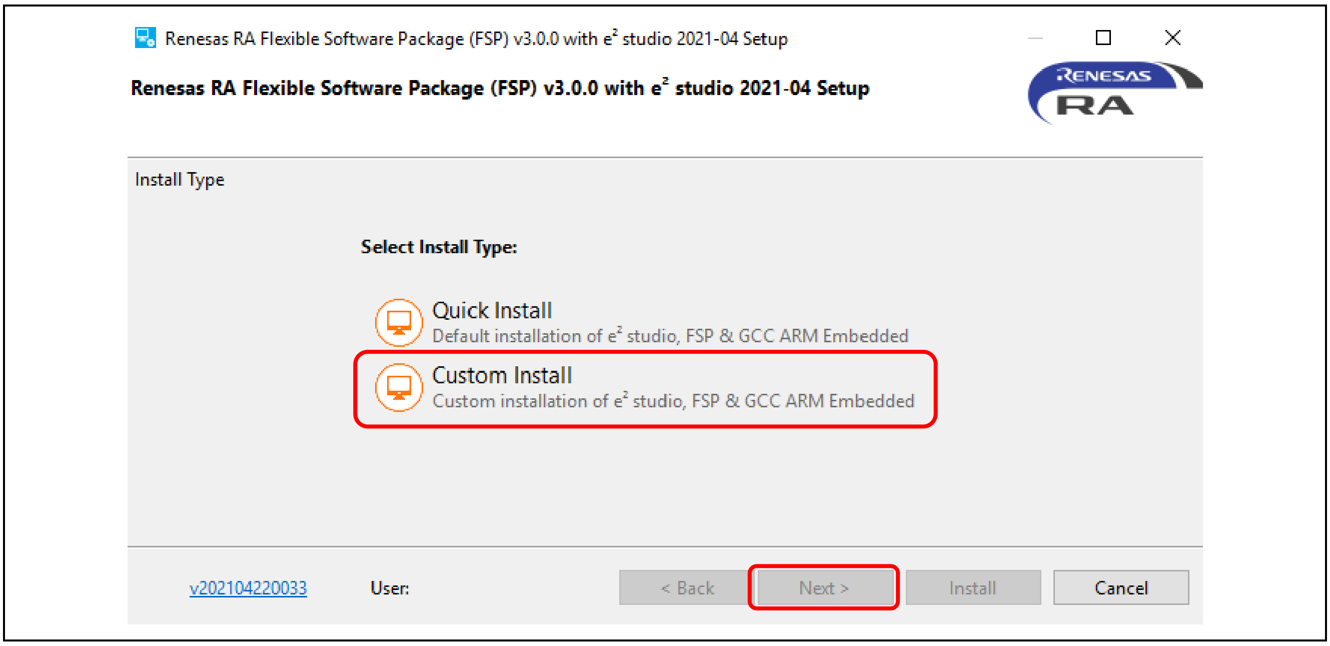

Figure 4. Installation – Select Install Type

In the welcome page, you may use the default folder or change it by clicking **[Change…]**. Click **Next** to continue.

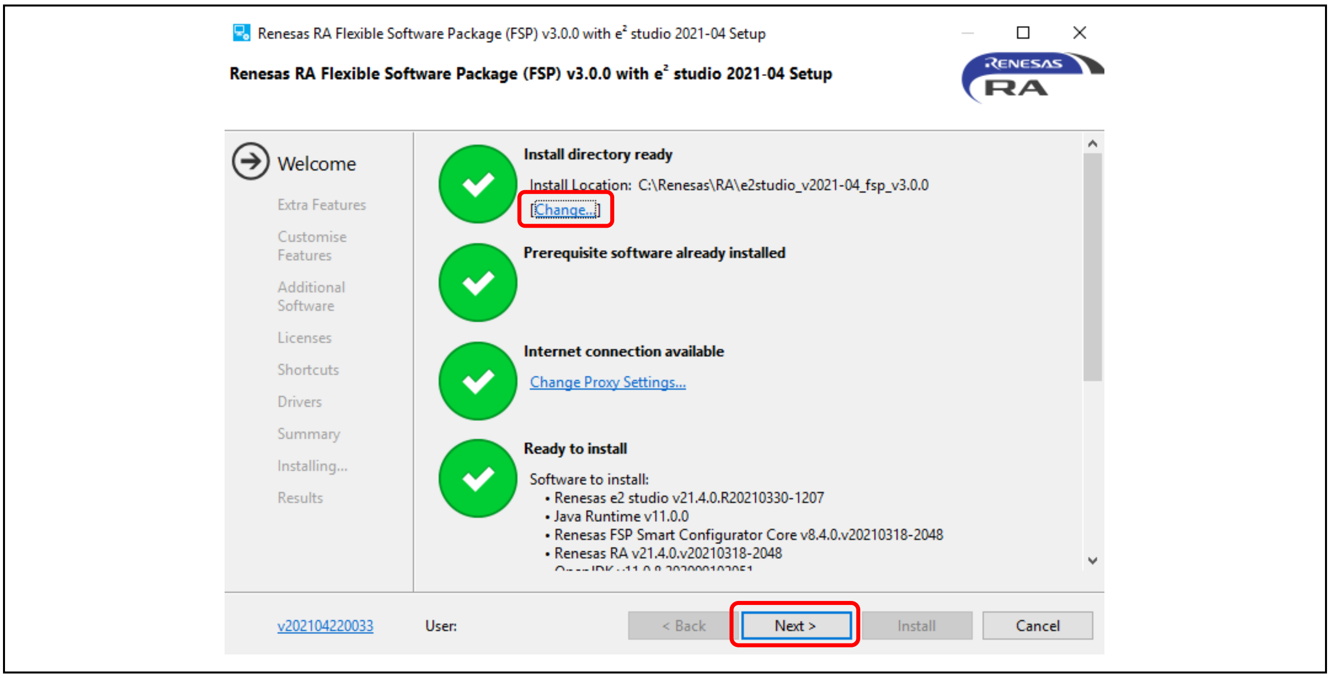

Figure 5. Installation – Welcome Page

In the **Extra Features** page, click the checkboxes for selected options, then click **Next**.

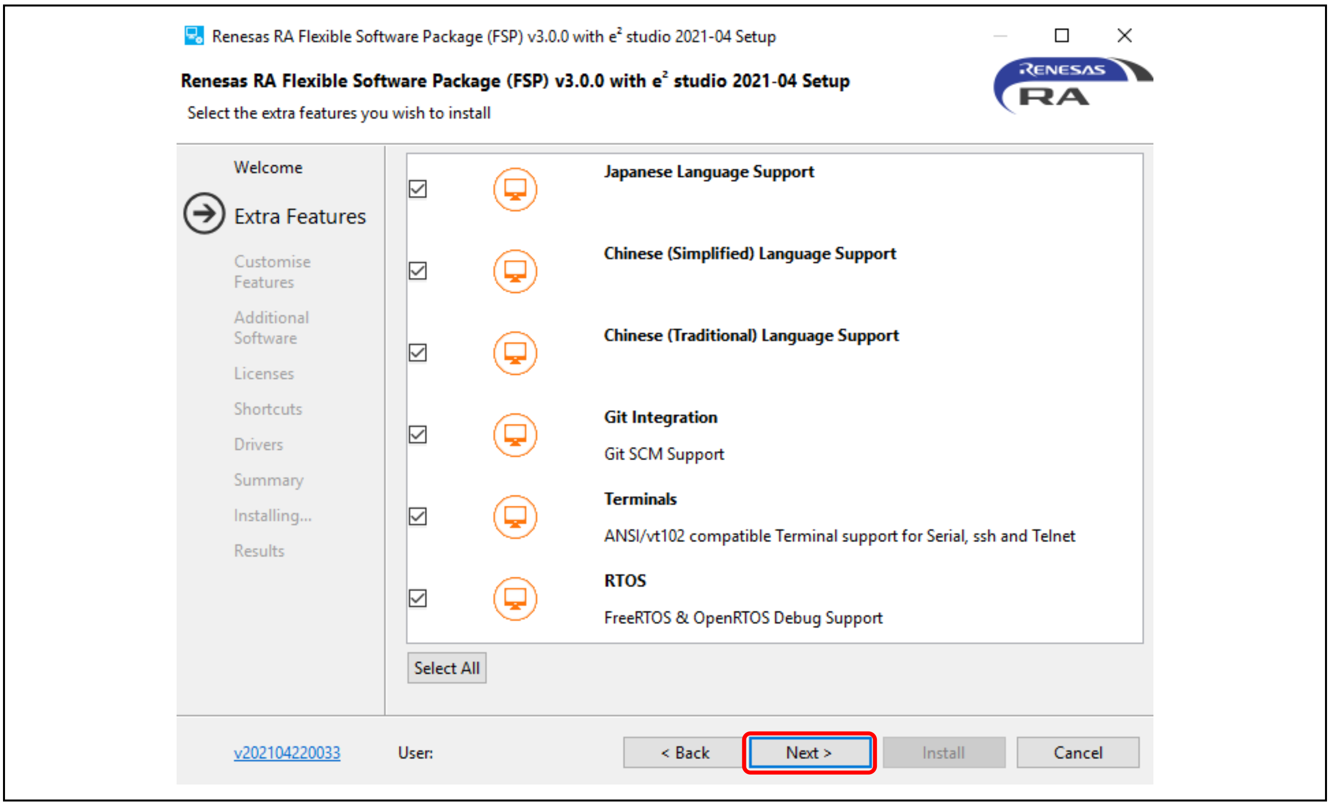

Figure 6. Installation - Extra Features

In the **Customise Features** page, select the components you want to install and click the **Next** button to continue.

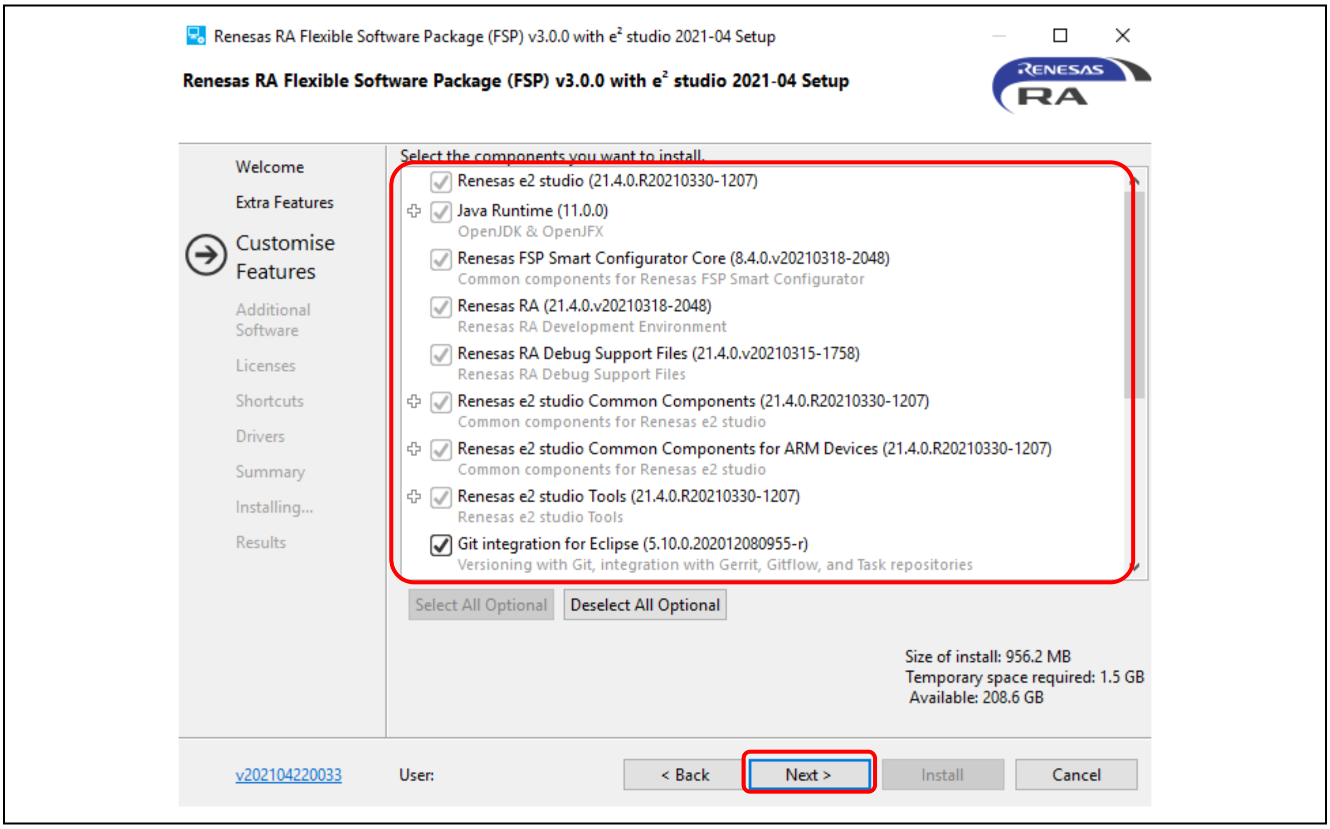

Figure 7. Installation - Customise Features

In the **Additional Software** page, select the **GNU ARM Embedded 9.3.1 2020q2** to be installed, then click **Next**. This page will not be shown if you select **Quick Install**.

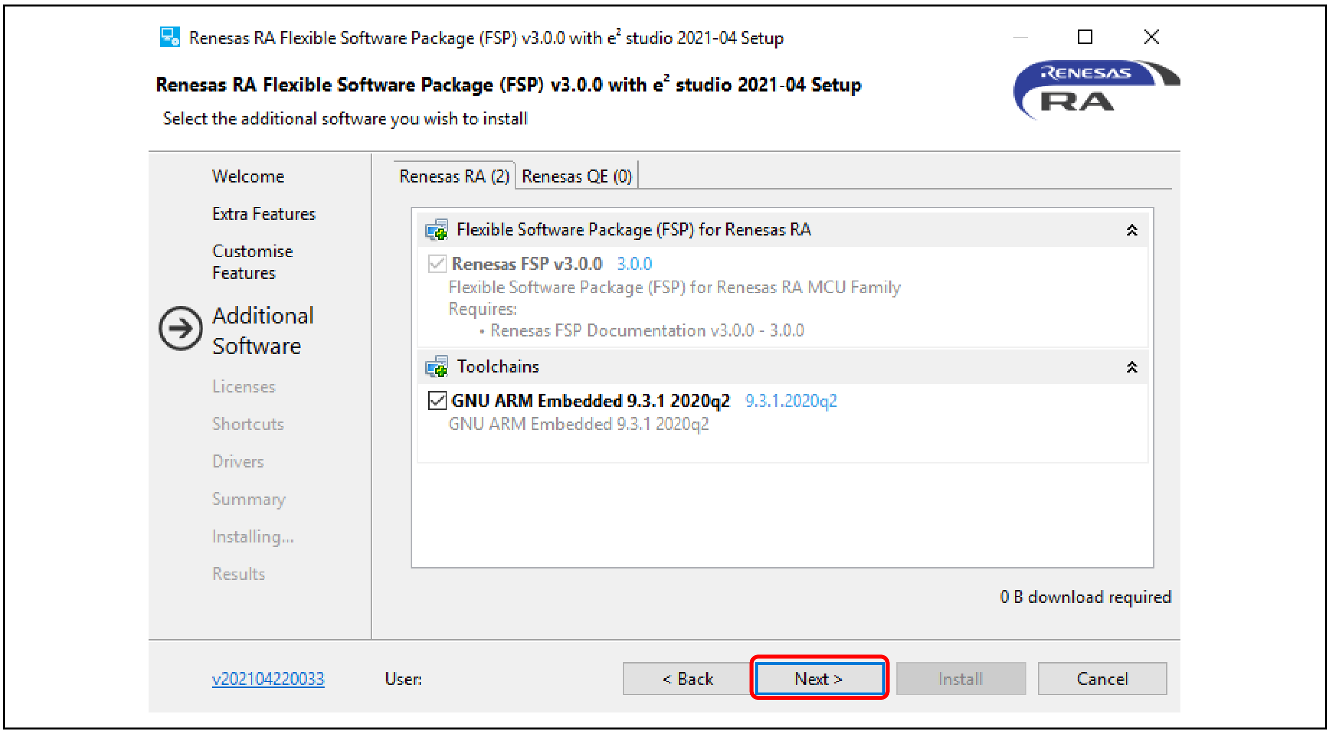

Figure 8. Installation – Select Additional Software

Click the checkbox to accept the license agreement, then click **Next** to continue.

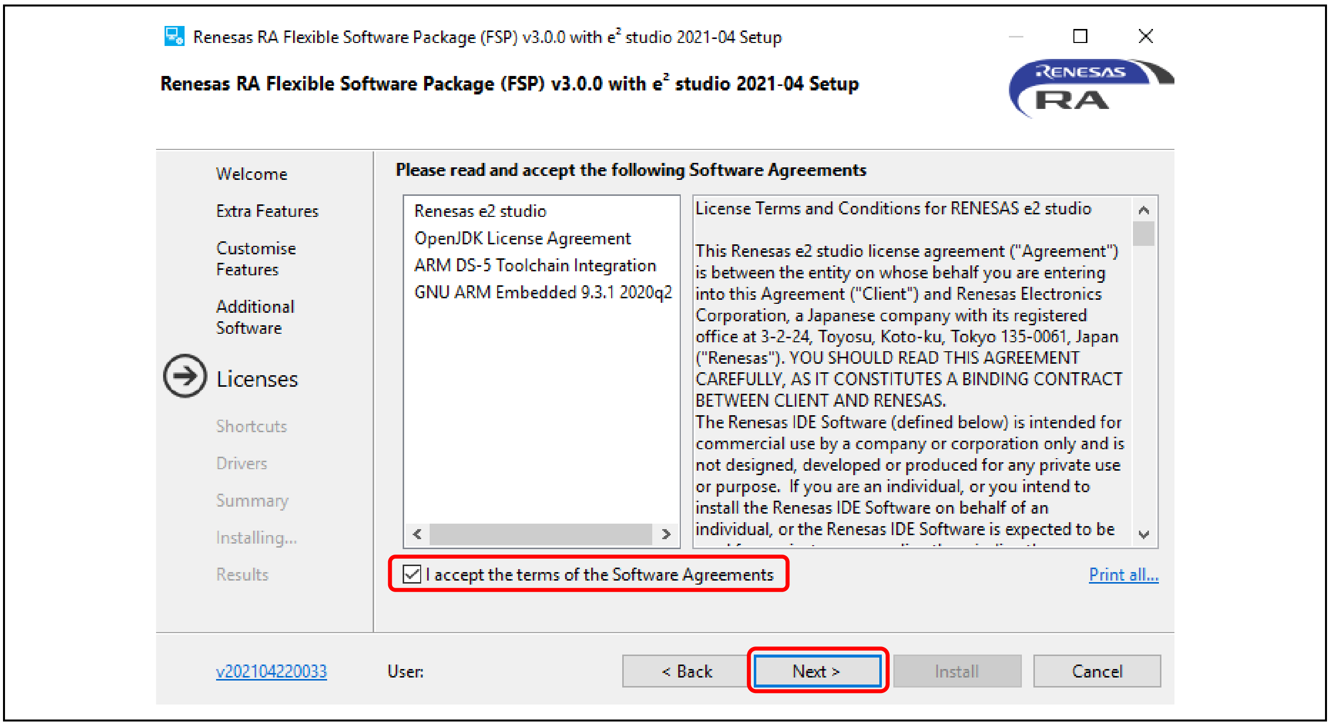

Figure 9. Installation – Software Agreements

In the **Shortcuts** page, select the shortcut name for the start menu and click **Next** button to continue.

Note: If you already have installed e² studio in another location, renaming this installation is recommended to distinguish it from the other e² studio(s).

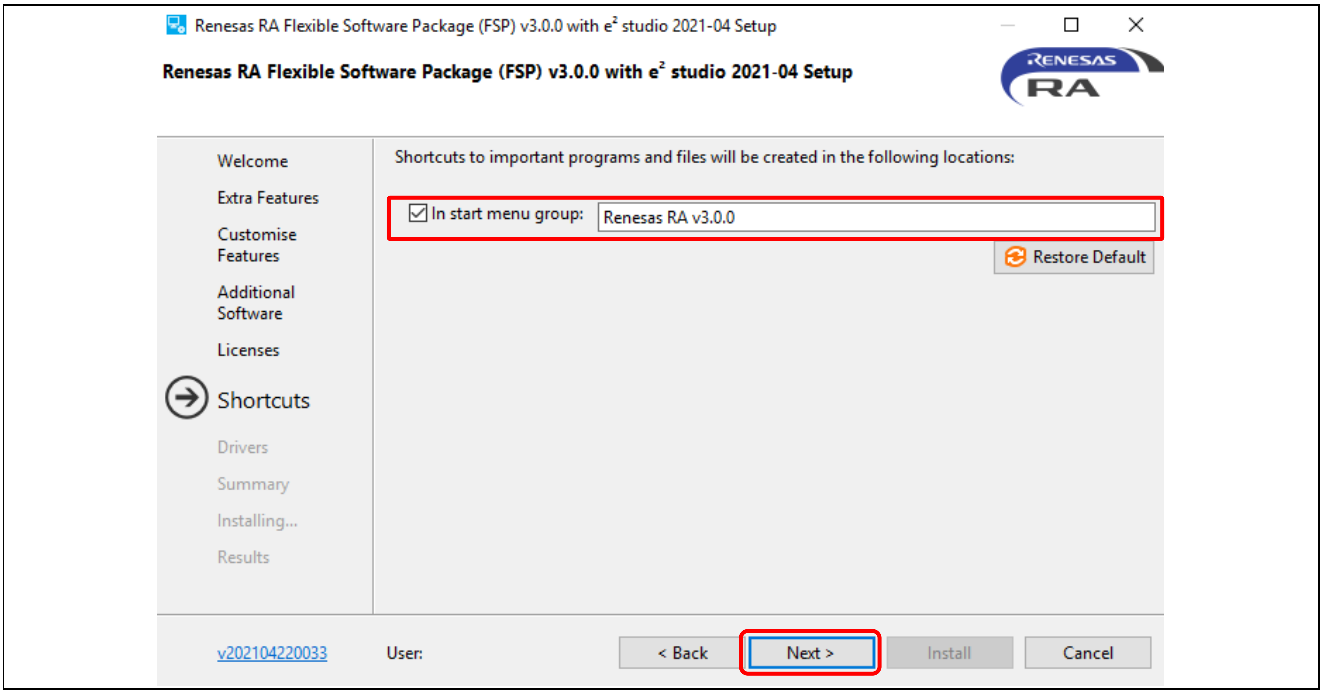

Figure 10. Installation – Shortcuts

Check the **Summary** and click **Install** to continue.

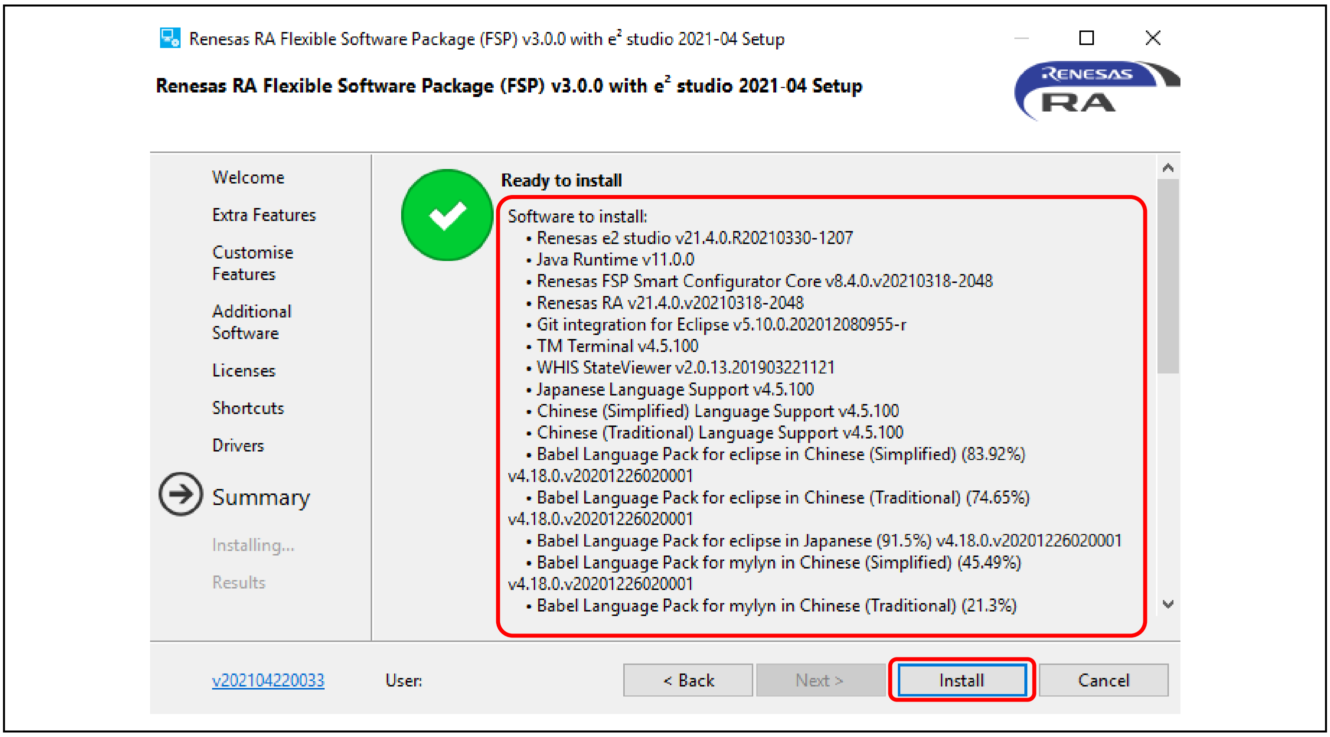

Figure 11. Installation – Summary

Click **OK** to finish the installation.

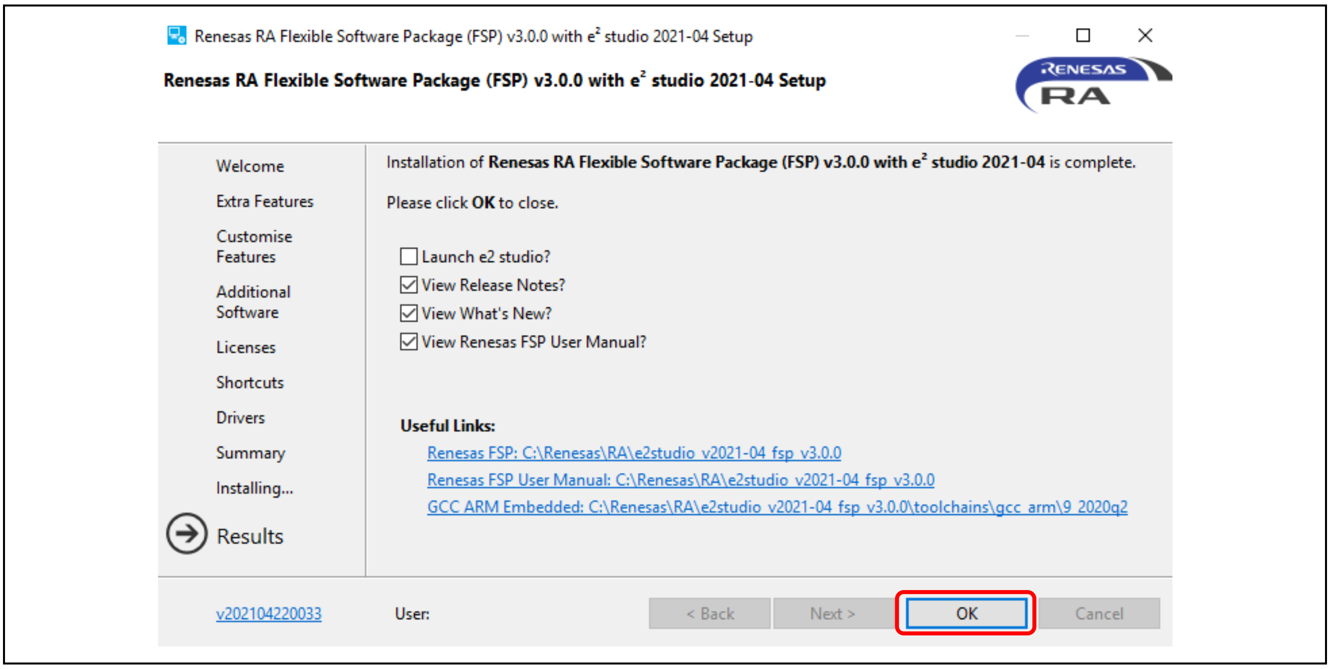

Figure 12. Installation – Complete Installation

2.2 Installing e² studio and FSP Independently

This section describes installation of the following components independently.

- e² studio IDE
- GCC ARM Embedded Compiler
- Renesas Flexible Software Package (FSP)

2.2.1 Installing e² studio

To install e² studio for RA, follow these steps:

1. Download e² studio 2021-04 (64-bit version) offline installer from https://www.renesas.com/e2studio
2. Unzip the download file and run the e² studio installer to invoke the e² studio installation wizard page.
3. If e² studio was installed in your PC, the options to modify, remove the existing version, and install e² studio to a different location will be shown. It is possible to install multiple versions of e² studio by selecting **Install to a different location**. Click the **Next** button to continue.

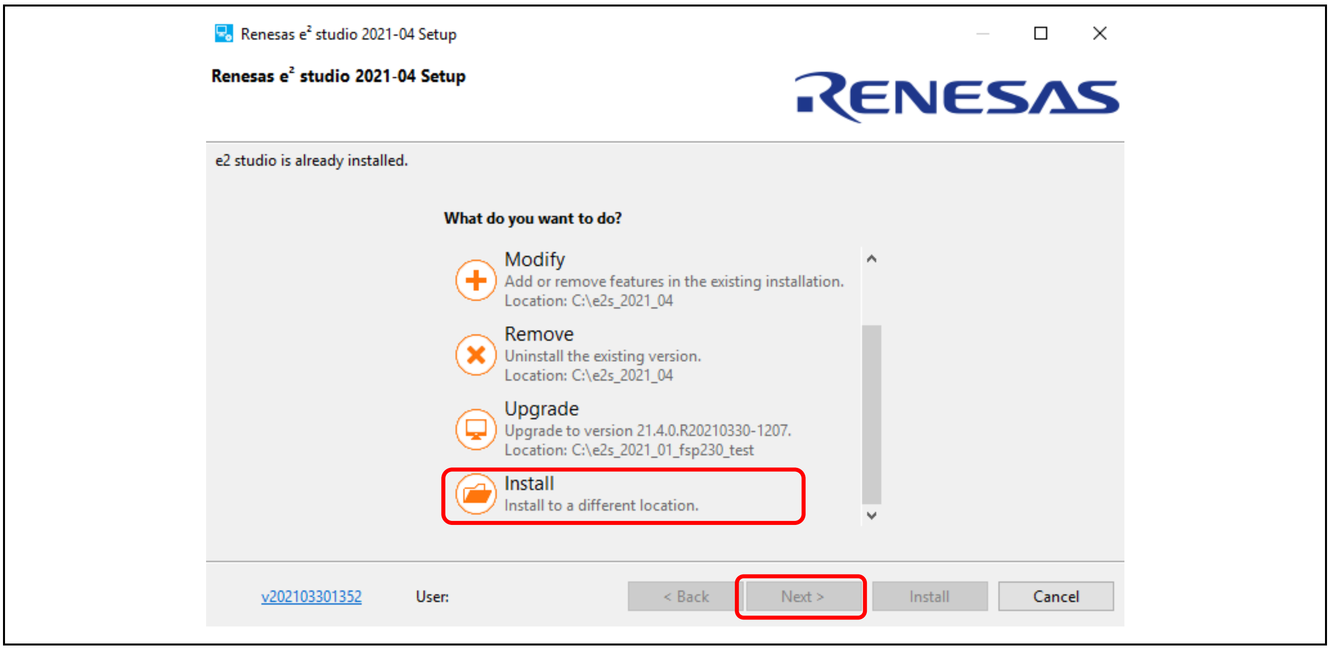

Figure 13. Install Multiple Versions of e² Studio

4. In the **Welcome** page, the default installation location is set to: `C:\Renesas\e2_studio`. You can click **[Change…]** to modify it. Click the **Next** button to continue.

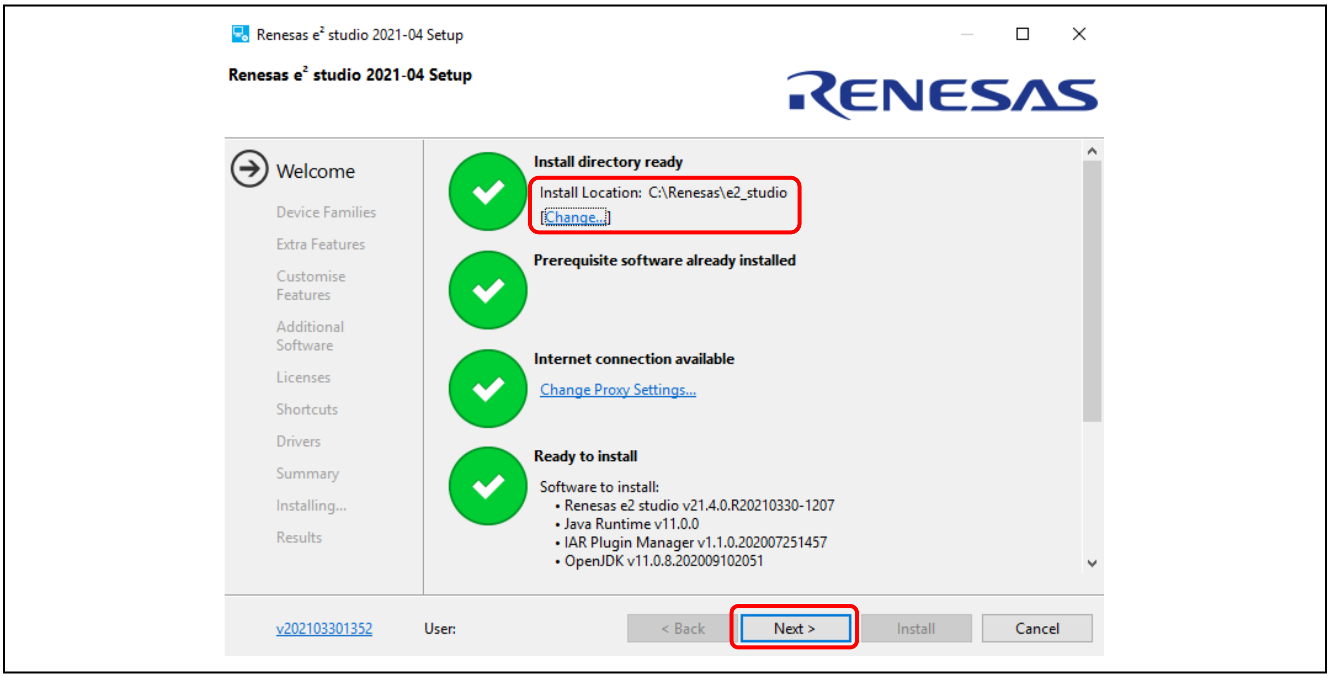

Figure 14. Installation – Welcome Page

5. **Device Families** page:
   Check the checkbox for **RA**. Checkboxes of other device families are optional.
   Click the **Next** button to continue.

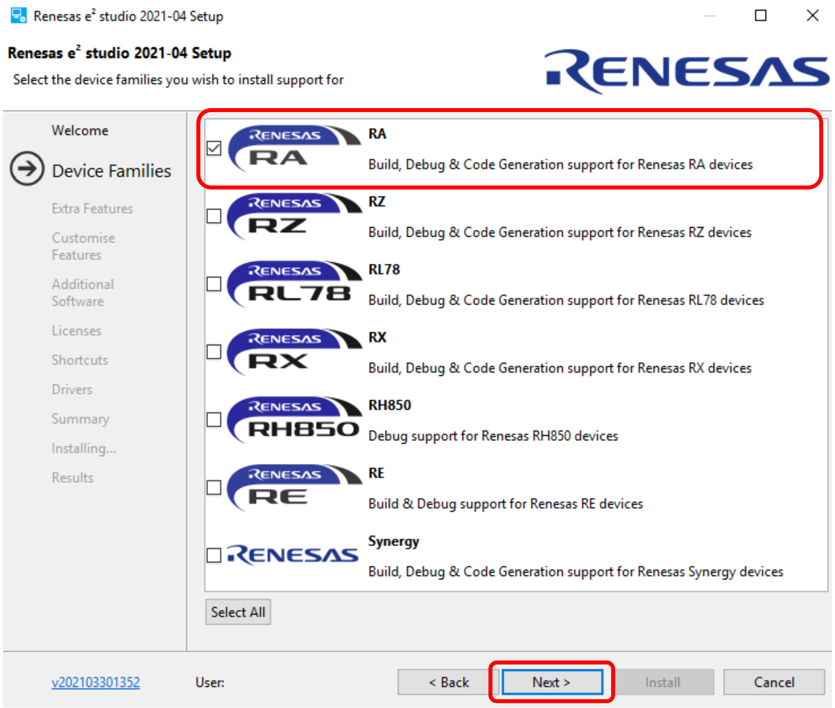

Figure 15. Installation – Device Families Page

6. **Extra Features** page:
   Select **Extra Features** (that is, Language support, Git Integration, RTOS support, and so on) to install.
   For non-English language users, select the language to support at this step.
   Click the **Next** button to continue.

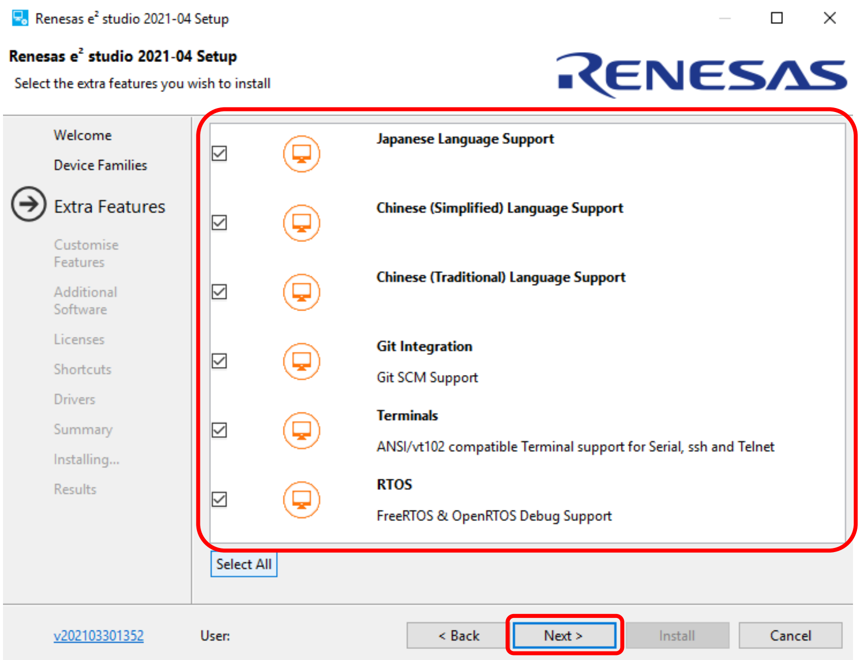

Figure 16. Installation – Extra Features Page

7. **Customise Features** page:
   Ensure that **Renesas RA Family Support** is checked.
   Click the **Next** button to continue.

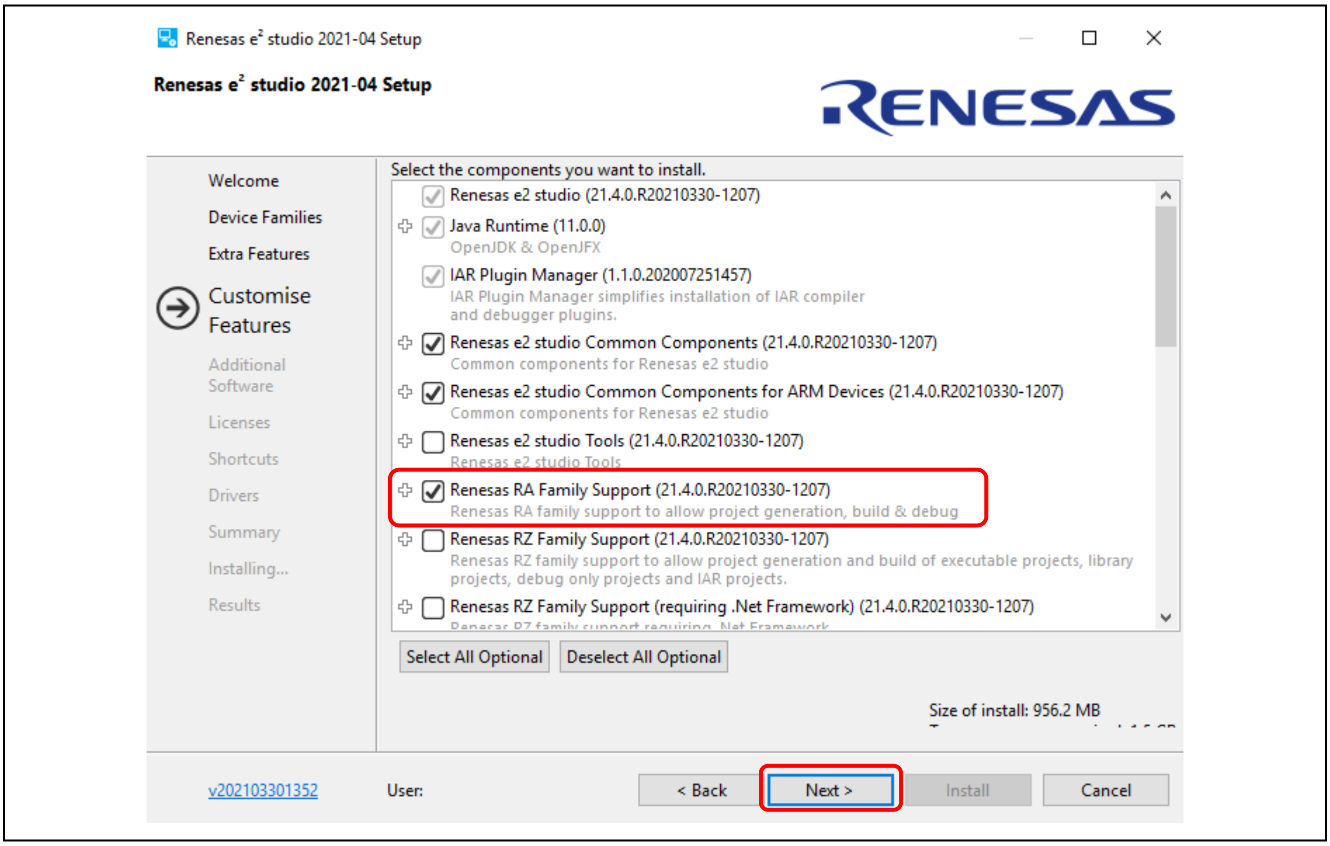

Figure 17. Installation – Customise Features Page

8. Additional Software page:
   Select the **GCC Toolchains & Utilities** tab and check the **GNU Arm Embedded 9.3.1 2020q2** checkbox to install the GNU Arm Embedded toolchain.
   Select the **Renesas FSP** tab and check the checkbox to install the latest version of FSP for Renesas RA.
   Click the **Next** button to continue.
   Note: If no internet access is available, additional software installation must be skipped because the software catalog cannot be downloaded. You can continue installation, anyway.

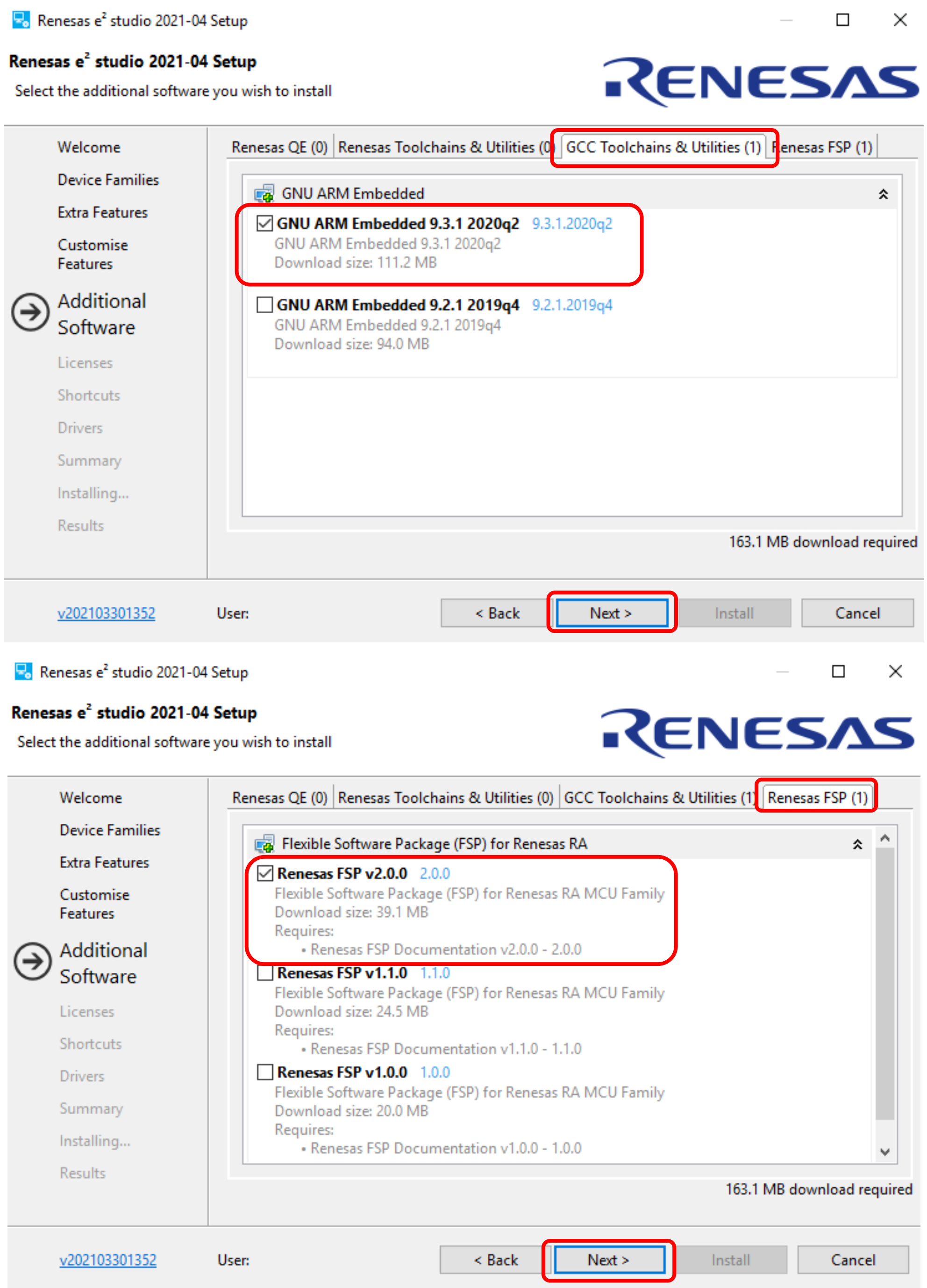

Figure 18. Installation – Additional Software Page

9. Licenses

   Read and accept the software license agreement to proceed with the **Next** button.

   Please note that you must accept the license agreement, or installation cannot proceed.

10. Shortcuts

    Select the shortcut name for the Start menu and click **Next** button to continue.

11. Summary

    Click the **Install** button to install Renesas e² studio.

12. Installing…

    The installation will start. Depending on the items selected in the **Additional Software** dialog, new dialogs may open to proceed with the installation of these software packages.

2.2.2 Setting Up the GNU Arm Embedded Toolchain

The GNU Arm Embedded Toolchain can be installed during e² studio installation. Alternatively, after e² studio has been installed, the GNU Arm Embedded Toolchain can be installed separately.

To install GNU Arm Embedded Toolchain, follow these steps:

1. Download version 9-2020-q2- update of the GNU Arm Embedded Toolchain supported by Renesas RA (`gcc-arm-none-eabi-9-2020-q2-update-win32.exe`) from https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads
2. Run the installer to install the GNU Arm Embedded Toolchain on the host machine.
3. Select the installation language. Click **Yes** in the installation confirmation dialog.
4. Keep all default settings in the installation wizard.
5. When the **Install wizard Complete** dialog appears, check the box **Add path to environment variable**, andclick **Finish** to complete the installation.

2.2.3 Installing the Renesas RA Flexible Software Package (FSP)

The Renesas RA Flexible Software Package (FSP) can be installed during e² studio installation. Alternatively, after e² studio has been installed, the FSP can be installed separately.

To install the FSP, follow these steps:

1. Visit the GitHub page of Flexible Software Package (FSP) for Renesas RA MCU Family: https://github.com/renesas/fsp/releases
2. Find and download the latest FSP packs installer (for example, `FSP_Packs_<version>.exe`). The FSP Package Installer includes the driver library, HTML User's Manual and a readme file.

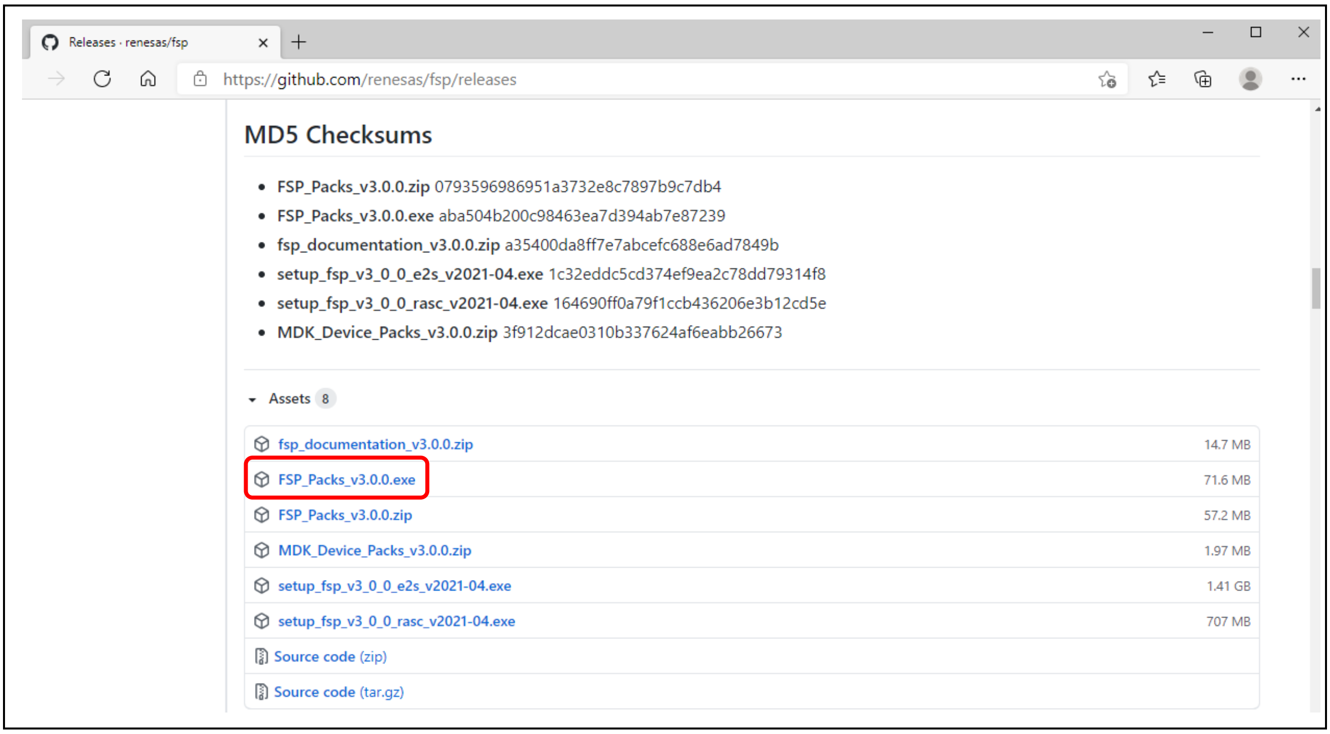

Figure 19. Installation – Download The FSP Packs

3. Make sure that a compatible e² studio was installed and closed during this installation.
4. Run the FSP packs installer and click **Next** to continue.
5. Click **I Agree** to accept the agreement.

6. Browse to the folder where e^2 studio is installed (for example, `C:\Renesas\e2_studio`) and click **Install**.

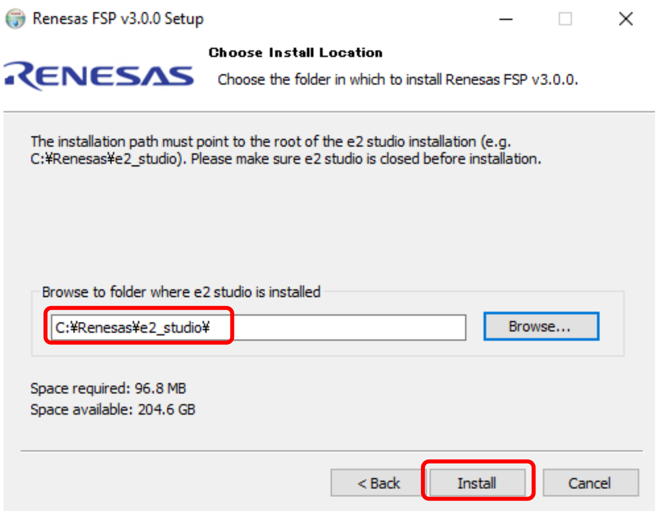

Figure 20. Choose Install Location

7. Click **Finish** to finish the installation.

2.3 Uninstalling e^2 studio

Users can uninstall e^2 studio by following typical steps to uninstall a program in the Windows OS.

1. Click on **Start** → **Control Panel** → **Programs and Features**.
2. From the currently installed programs list, choose **e^2 studio** and click the **Uninstall** button.
3. Click **Uninstall** to confirm the deletion in the **Uninstall** dialog.

At the end of the uninstallation, e^2 studio will be deleted from the installed location and the shortcut in Windows menu will be removed.

Note: If you have installed e^2 studio at multiple locations, you may not be able to find the uninstaller in **Apps & features** of the **Control Panel**. In such cases, launch the e^2 studio uninstaller located at: `{e² studio installed folder}/uninstall/uninstall.exe`.

2.4 Updating e^2 studio

To update e^2 studio, run the new version of e^2 studio installer (either **FSP with e^2 studio installer** or standard e^2 studio installer). Download the installer according to Section 2.

Please note that you should not overwrite an existing installation. Prior to the IDE upgrade, users must uninstall the old version of e^2 studio. However, to keep both old and new e^2 studio versions, you can create a new folder as installation destination for the new e^2 studio version.

2.5 Updating FSP

To update FSP, run the new version of FSP installer. Please download the installer according to Section 2.2.3.

2.6 Installing RA SC for Keil MDK and IAR EWARM

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

To download and install the RA SC Installer, follow the steps below:

1. Visit the GitHub page of Flexible Software Package (FSP) for Renesas RA MCU Family:
   https://github.com/renesas/fsp/releases
   Search for the RA SC installer and download it (for example, setup_fsp<version>_rasc_<version>.exe).

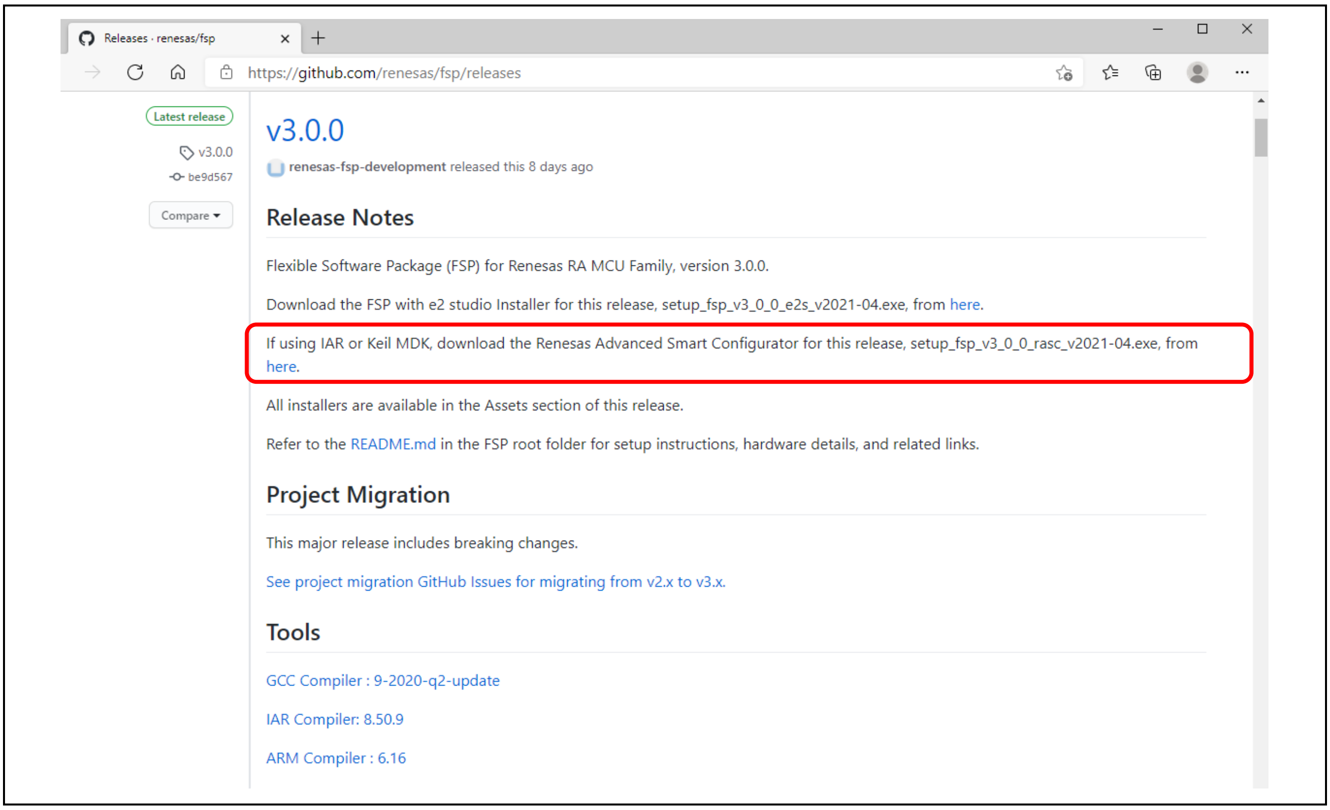

Figure 21. Installation – Download The RA SC Package

2. Run the installation file.
   In the **Welcome** page, you may use the default folder or change it by clicking **[Change…]**. Click **Next** to continue.

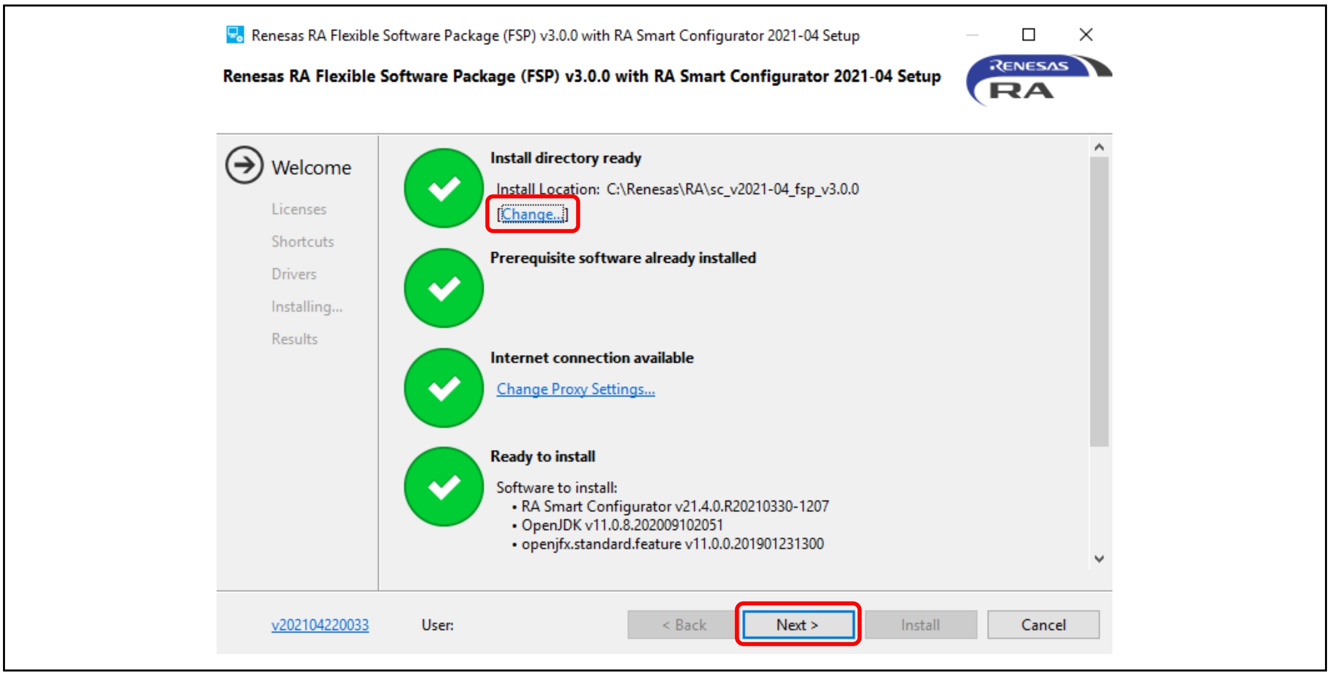

Figure 22. Installation – Welcome Page

3. Click the checkbox to accept the license agreement, and click **Next** to continue.

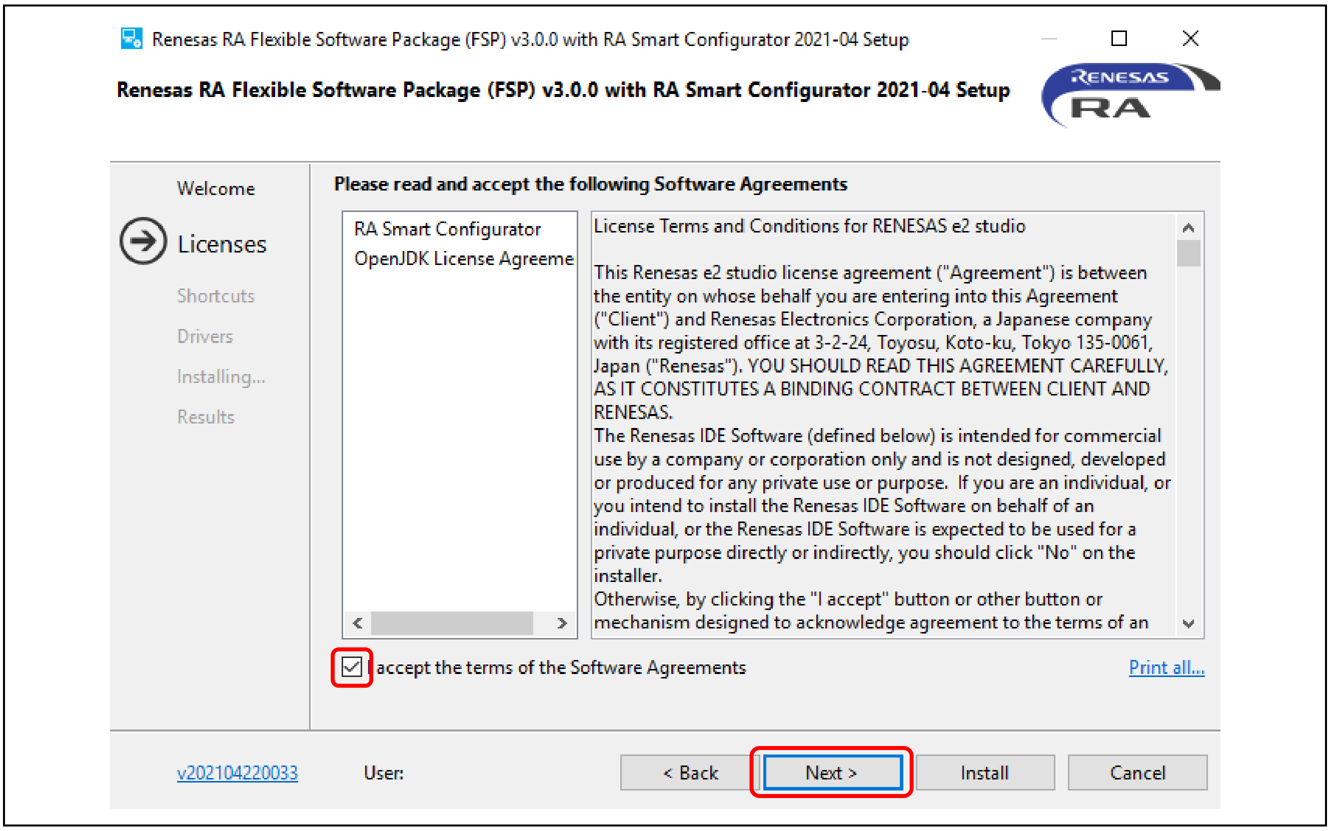

Figure 23. Installation – Software Agreements

4. Check the **Shortcuts** and click **Install** to continue.

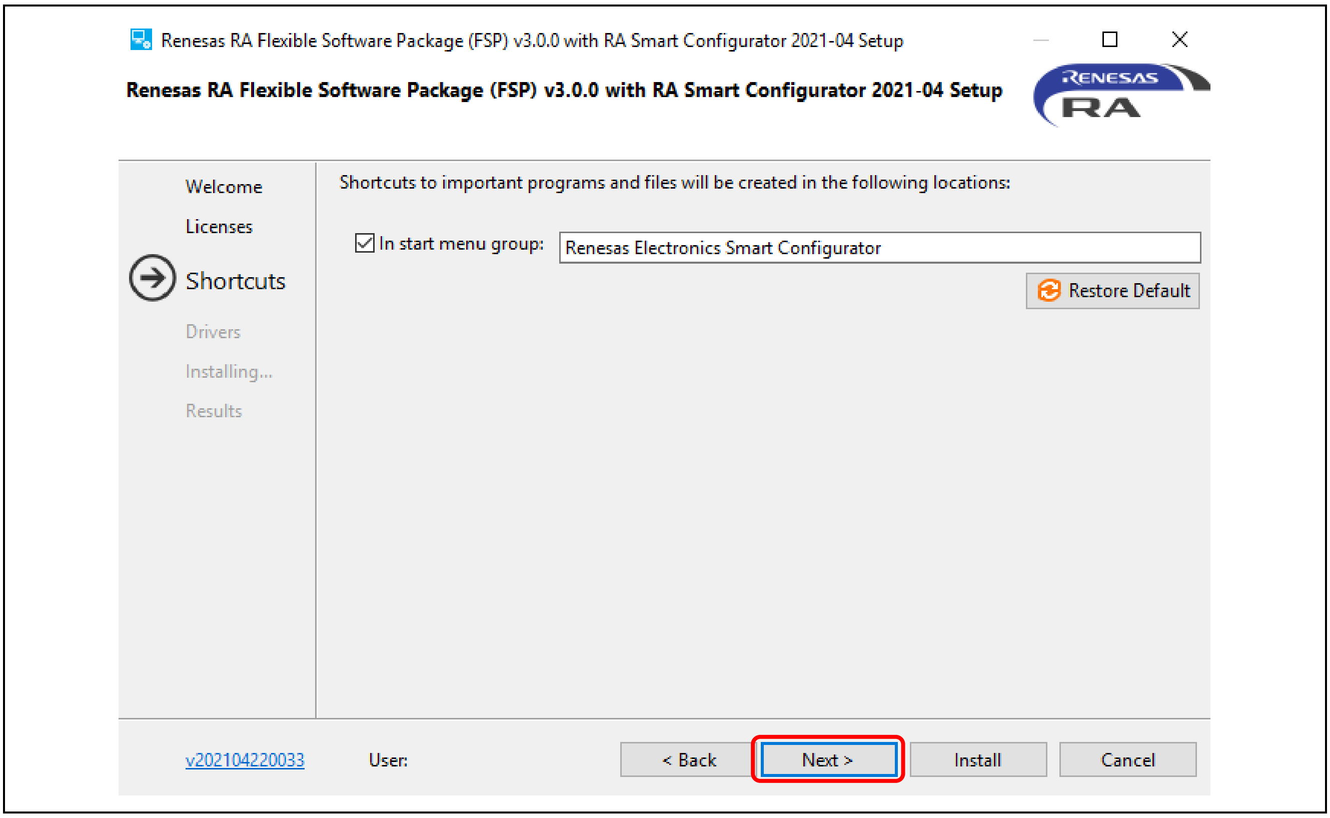

Figure 24. Installation – Shortcut

5. Click **OK** to finish the installation.

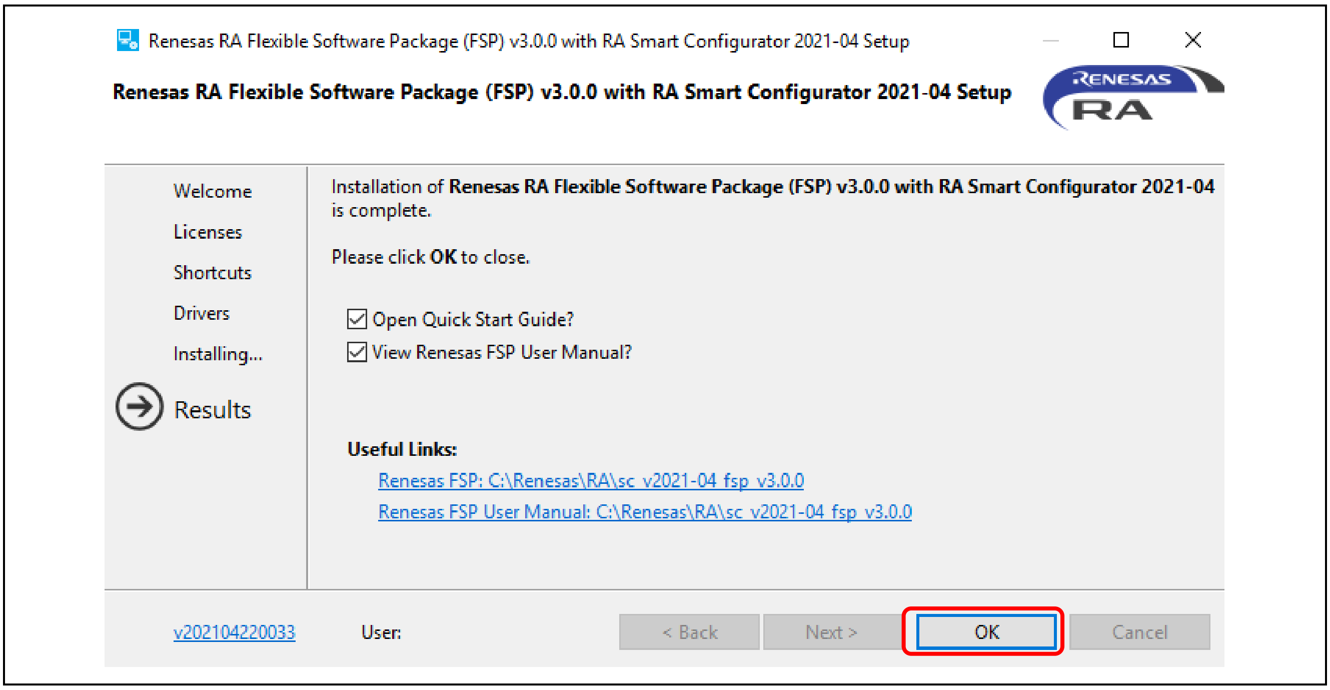

Figure 25. Installation – Complete Installation

Note: For more information on how to use RASC with Keil MDK and IAR EWARM, please refer to *RA SC User Guide for MDK and IAR* in the FSP document: https://renesas.github.io/fsp/_s_t_a_r_t__d_e_v.html#RASC-MDK-IAR-user-guide

3. Project Generation

This section describes the creation of a new RA project. The e² studio includes a wizard to help create a new RA project quickly. This is achieved by the ability of the wizard to match the project to a particular RA device and board.

The project generator can set up the pin configurations, interrupts, clock configurations, and the necessary driver software.

As a prerequisite, the FSP and the toolchain must be installed on the host machine as described in section 2.

3.1 Generating a New RA Project for a Non-TrustZone device

This section describes how to generate a RA project for a non-TrustZone device. For a TrustZone device, please refer to section 3.2.

A simple project generation wizard is available in e² studio to generate a new RA project with a project name and the associated device and board, including board-level drivers.

Start the e² studio application and choose a workspace folder in the Workspace Launcher. To configure a new RA project, follow these steps:

1. Select **File** → **New** → **Renesas C/C++ Project** → **Renesas RA**.

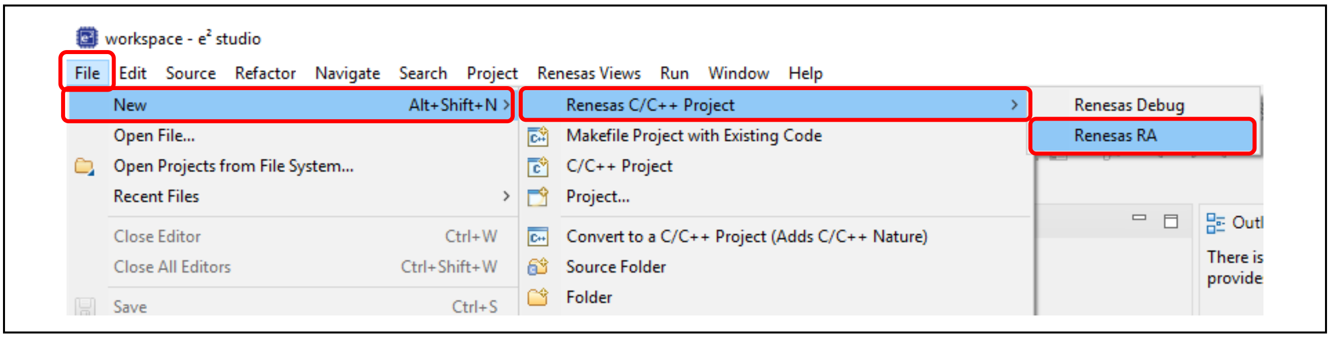

Figure 26. Project Generation – New Project Creation

2. Select **Renesas RA**: **Renesas RA C/C++ Project** template. Click **Next** to continue.

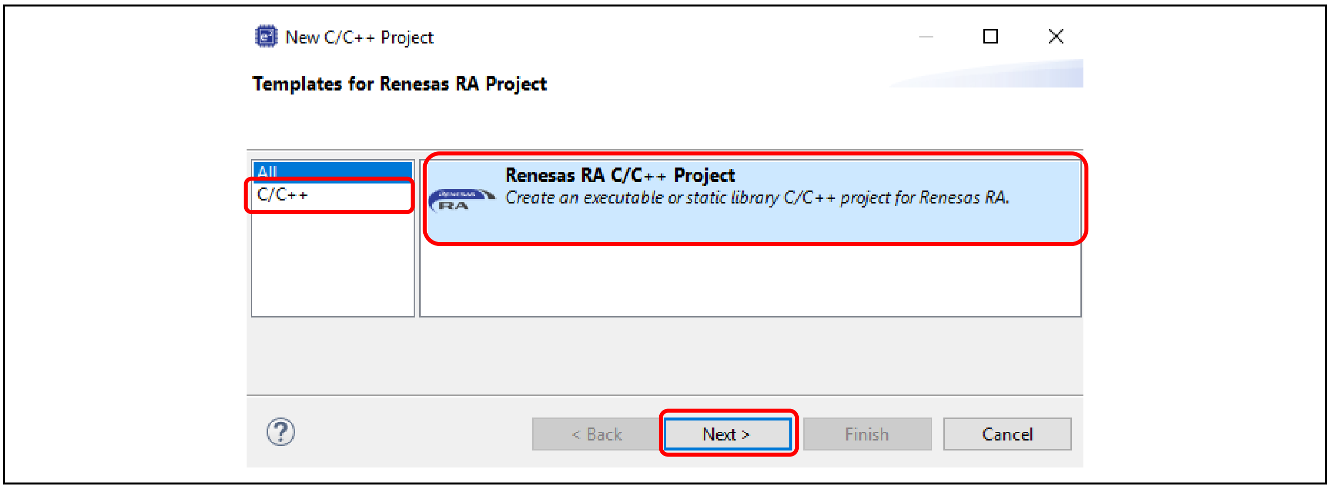

Figure 27. Project Generation – Select RA Project

3. In the project generation wizard, enter the following project information:
   Project name: Enter a name, for example, **RA_Tutorial**.
   Use default location: Checked. If you want to create a project in a different location, uncheck this checkbox and enter a new location.
   Click **Next** to continue.

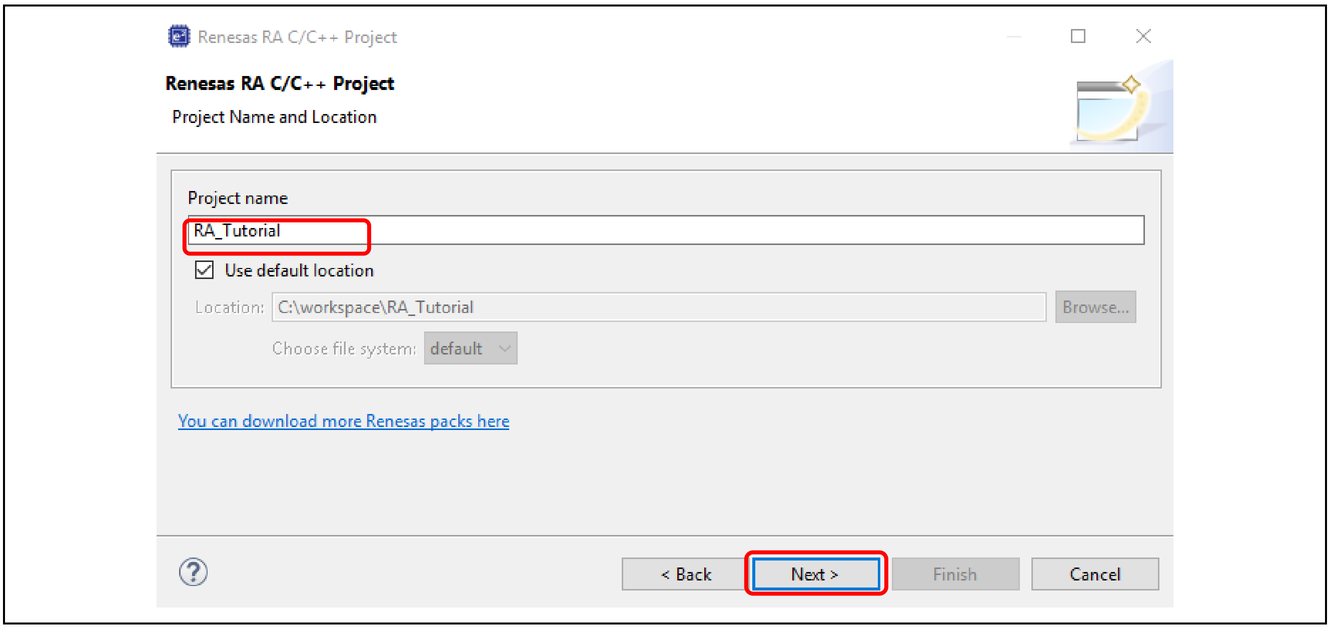

Figure 28. Project Generation – New RA Project Generation Wizard

4. In the device selection dialog, enter device and tool information:
   Board: EK-RA6M3.
   Toolchain version: Latest GNU Arm Embedded Toolchain approved for use with Renesas RA (for example, GCC ARM Embedded 9.3.1.20200408).
   Debugger: J-Link (ARM).
   Keep all other fields as default.
   Click **Next** to continue.

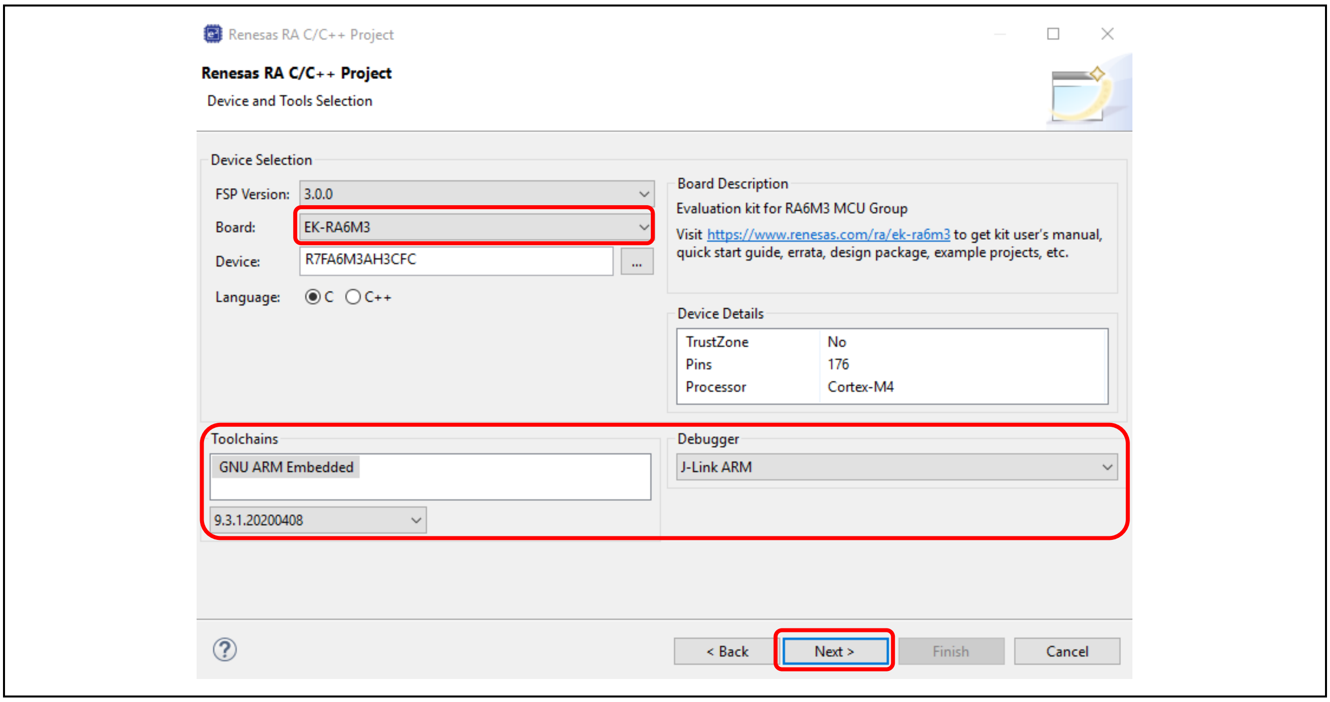

Figure 29. Project Generation – Device Selection

5. **Build Artifact Selection**: **Executable**
   RTOS Selection: **No RTOS**

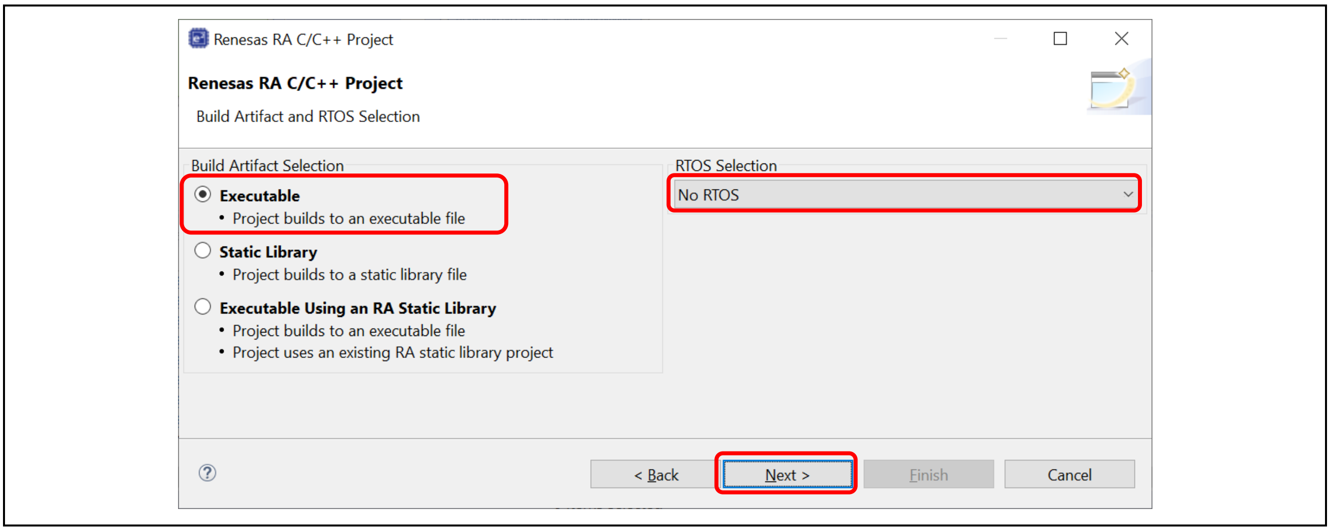

Figure 30. Artifact and RTOS Selection

6. In the project template dialog, select a project template, for example, **Blinky**.

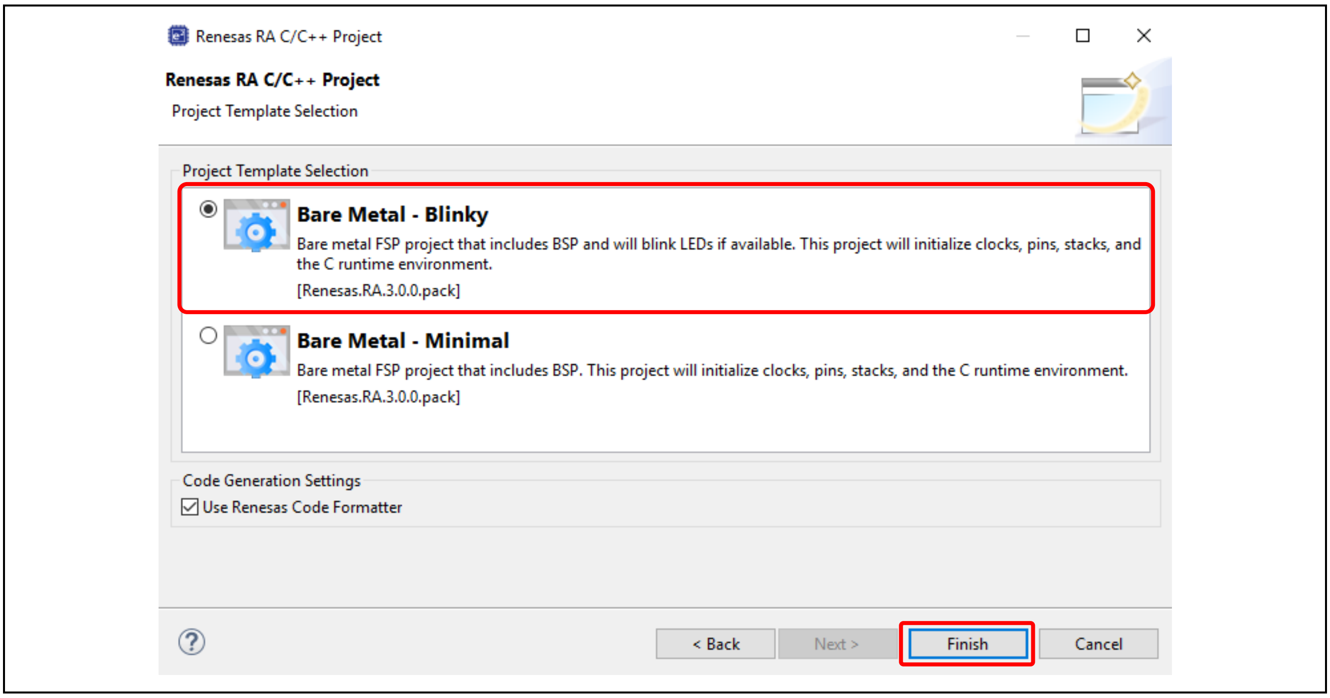

Figure 31. Project Generation – Project Template

7. Click the **Finish** button to create a new project.
   You may be prompted to open the **FSP Configuration** perspective. Click **Yes** to open the perspective. (In Eclipse, a ‘perspective’ is a predetermined arrangement of panes and views.)
   e² studio creates a new project with various views. Among them are the **Project Explorer View**, the **RA Project Configuration Editor**, and the **Visualization View**.

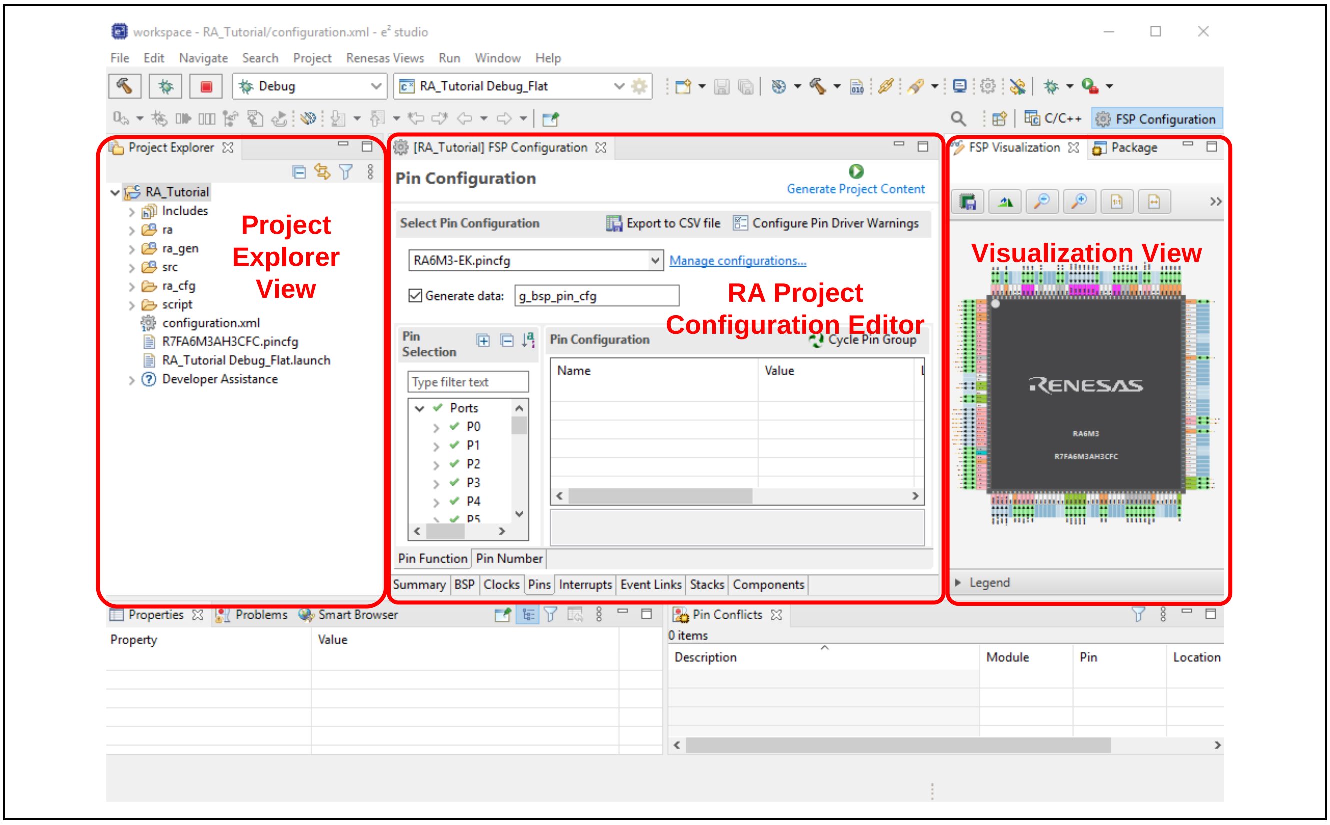

Figure 32. Project Generation – New Project Creation View

3.2 Generating a New RA Project for a TrustZone device

This section guides a user through generating an RA project for a TrustZone device. For a non-TrustZone device, please refer to section 3.1

TrustZone enabled projects for both secure and non-secure applications on with Arm® TrustZone® technology. Refer to this link for more information about RA Arm® TrustZone® tools, https://www.renesas.com/sg/en/document/apn/ra-arm-trustzone-tooling-primer.

The Flat (Non-TrustZone) Project type will create a self-contained ELF executable file without security partitioning, suitable for immediate execution on the target device.

3.2.1 Flat (Non-TrustZone) Project

To create a new Flat (Non-TrustZone) project, follow these steps:

1. From the menu, select **File** → **New** → **Renesas C/C++ Project** → **Renesas RA**.
2. Select **Renesas RA**: **Renesas RA C/C++ Project** template. Click **Next** to continue.
3. In the project generation wizard, enter the following project information:

   Project name: Enter a name, for example, **RA_Flat**.

   Use default location: Checked. If you want to create a project in a different location, uncheck this checkbox and enter a new location.

   Click the **Next** button to continue.

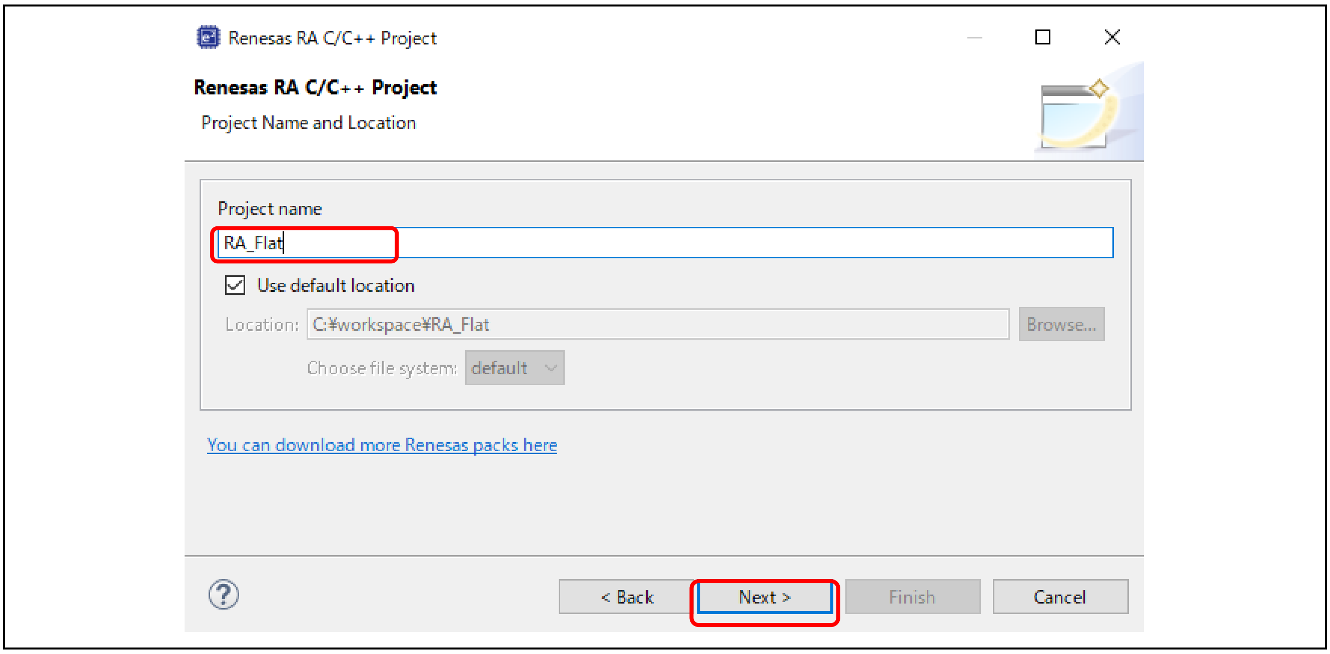

Figure 33. Project Generation – New RA Project Generation Wizard

4. In the **Device and Tools Selection** dialog, enter device and tool information:
   Board: **EK-RA6M4**
   Toolchain version: Latest GNU Arm Embedded Toolchain approved for use with Renesas RA (for example, GCC ARM Embedded 9.3.1.20200408)
   Debugger: **J-Link (ARM)**
   Keep all other fields as default
   Click **Next** to continue.

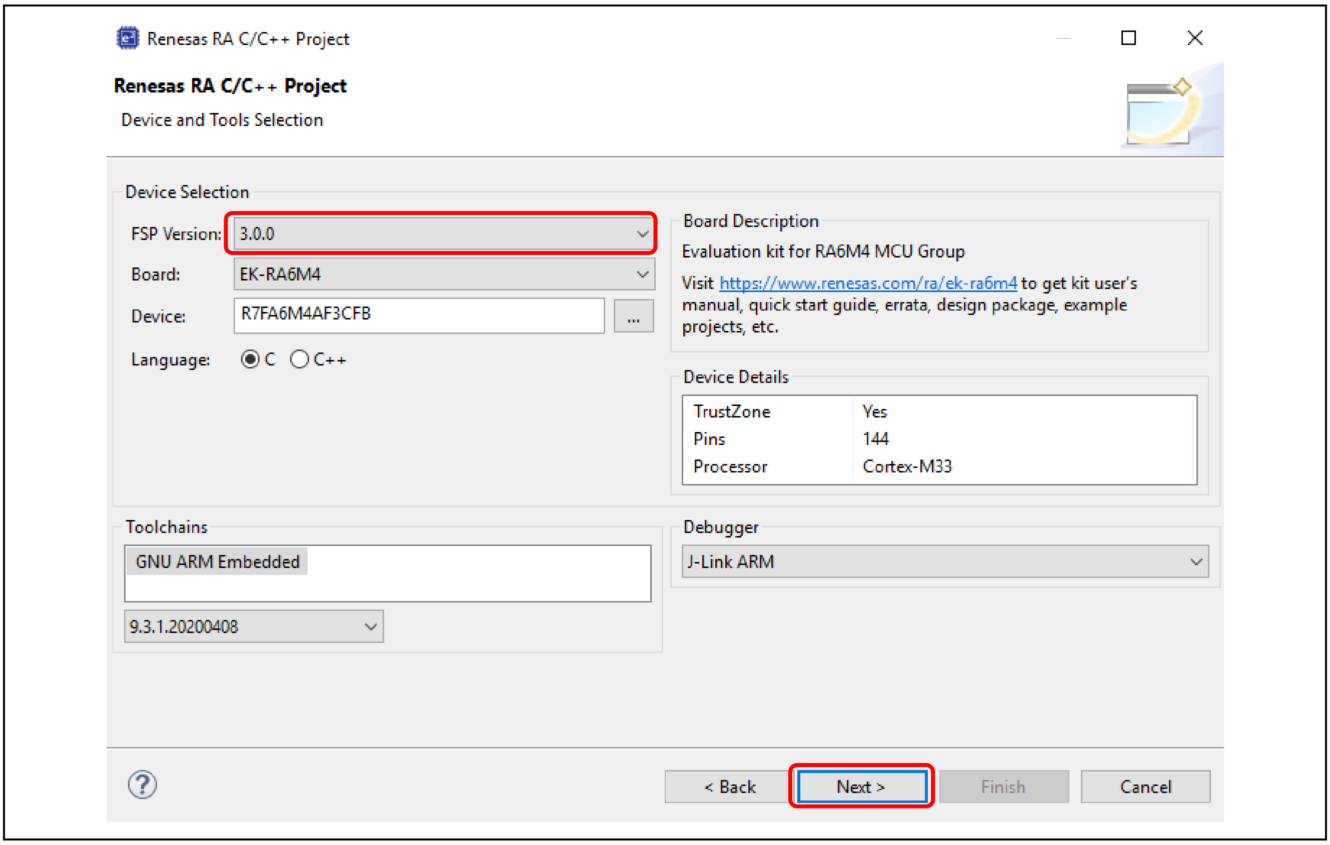

Figure 34. Project Generation – Device Selection

5. Project type selection: **Flat (Non-TrustZone) Project**

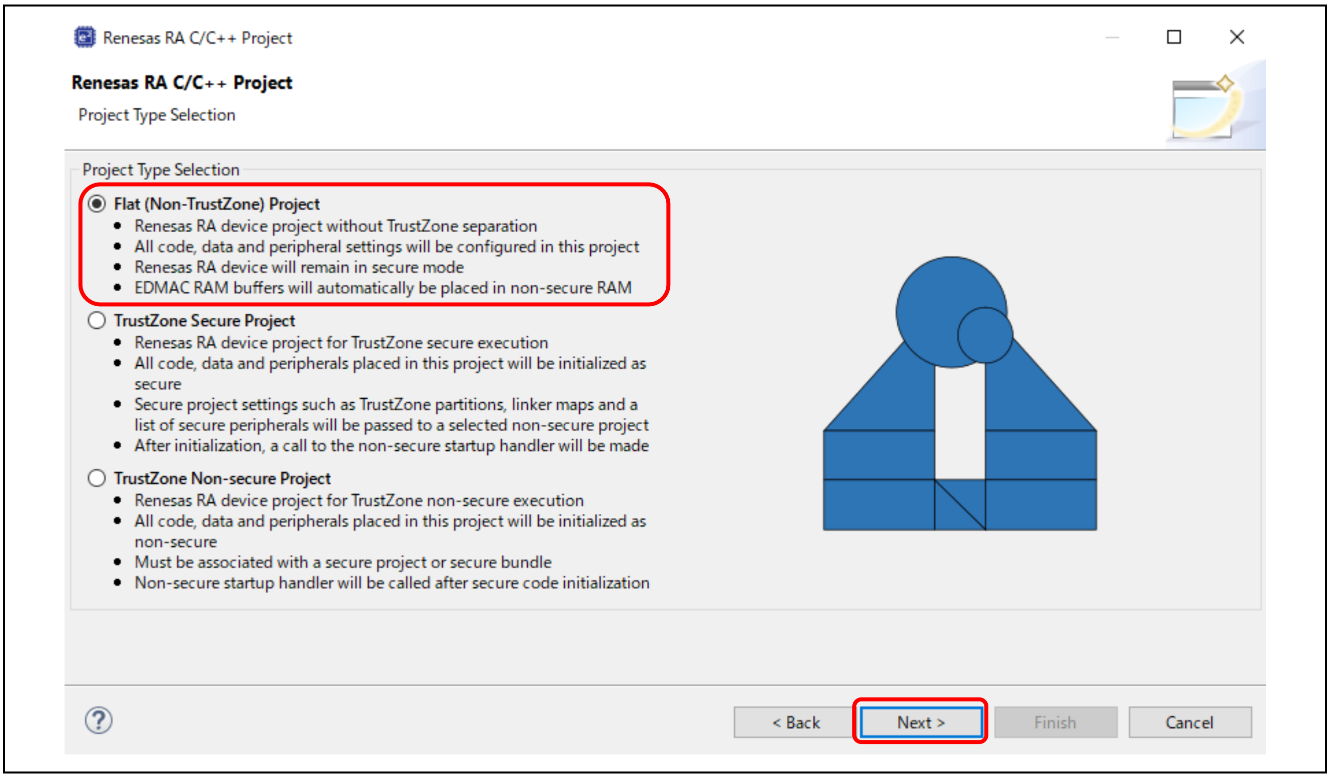

Figure 35. Project Type Selection

6. Build Artifact Selection: **Executable**
   RTOS Selection: **No RTOS**

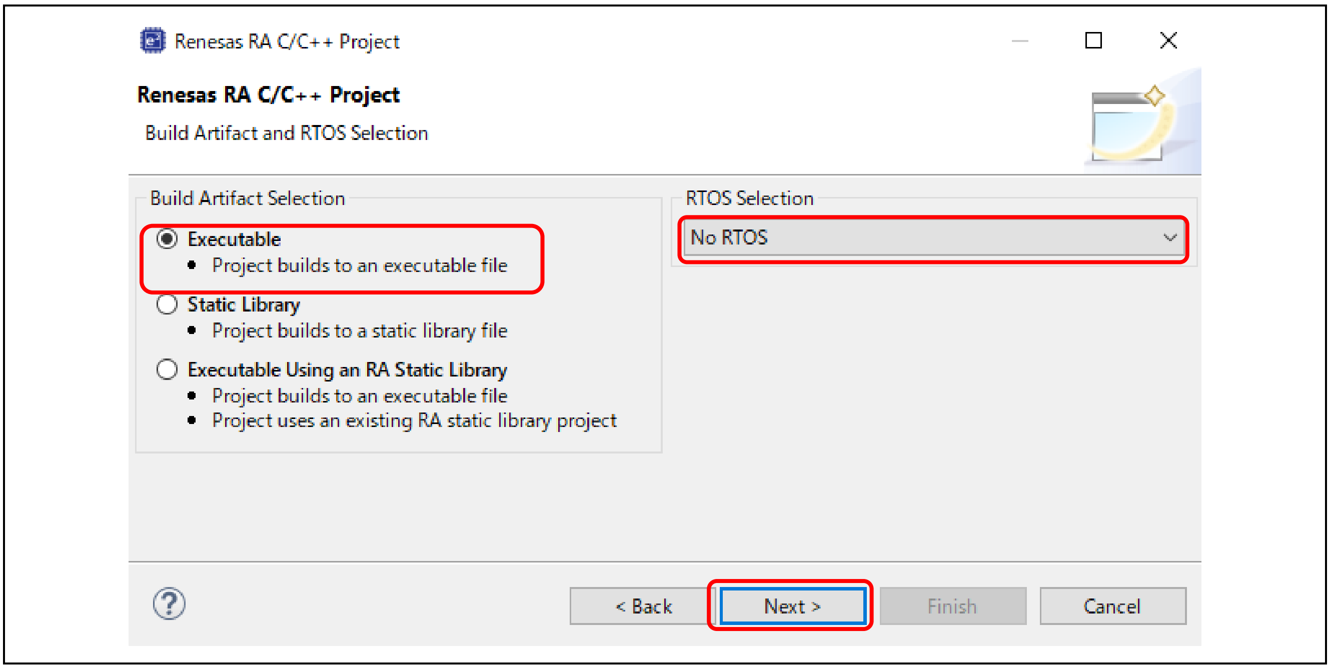

Figure 36. Artifact and RTOS Selection

7. In the **Project Template Selection** dialog, select a project template, for example, **Blinky**.

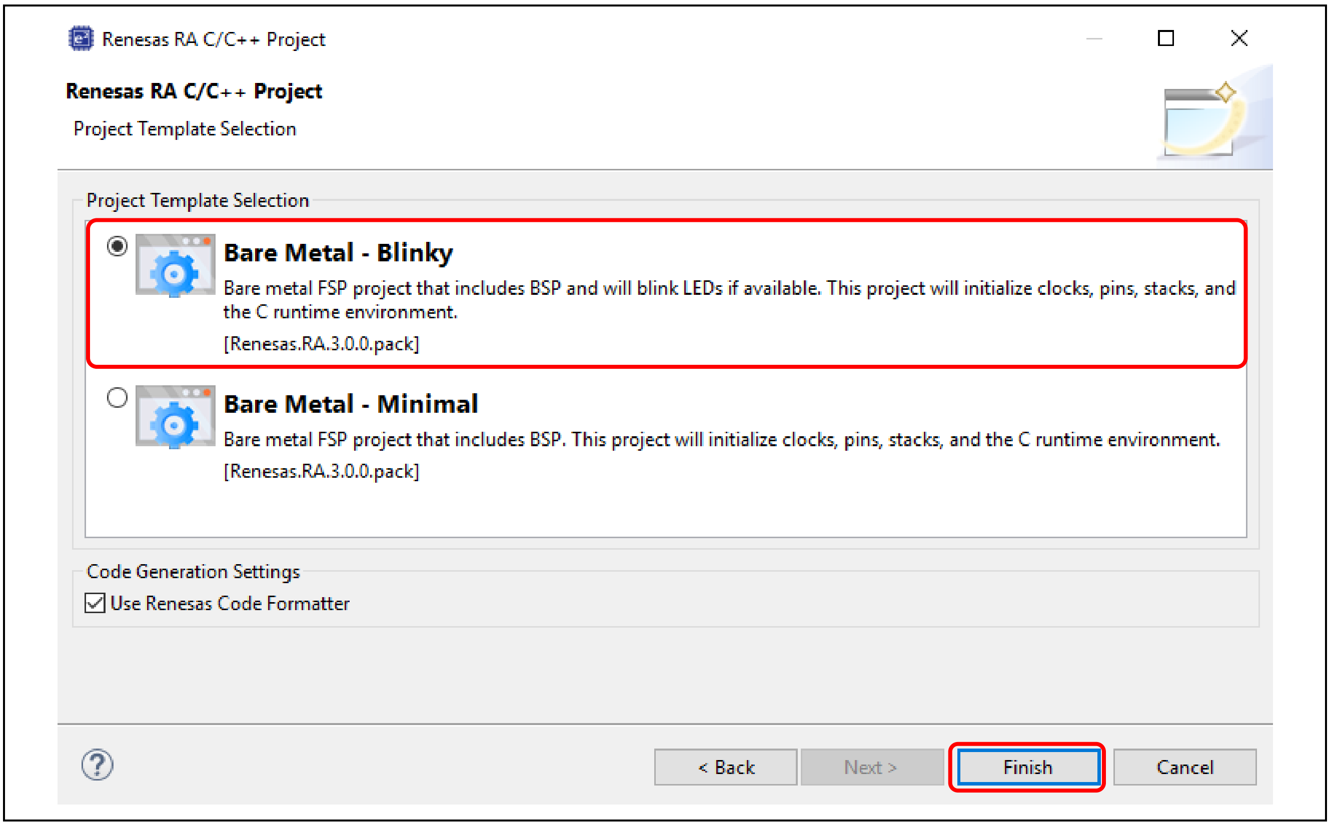

Figure 37. Project Generation – Project Template

8. Right-click on project name and select **Build Project**. Project is built without error.

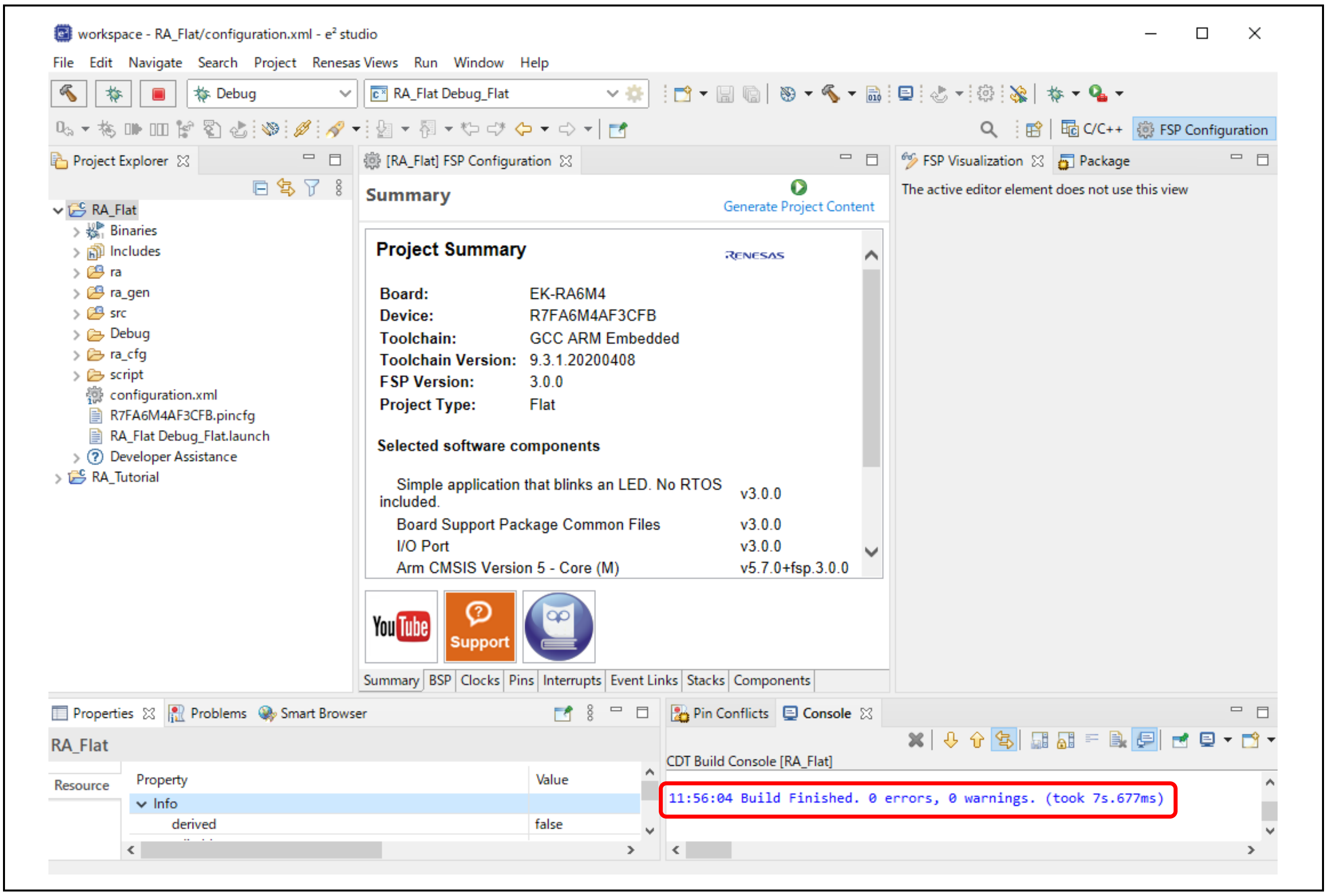

Figure 38. Project Is Built Successfully

3.3 Importing an Existing RA Project

To import an existing RA Project, please follow below steps,

1. Click **File** → **Import**.

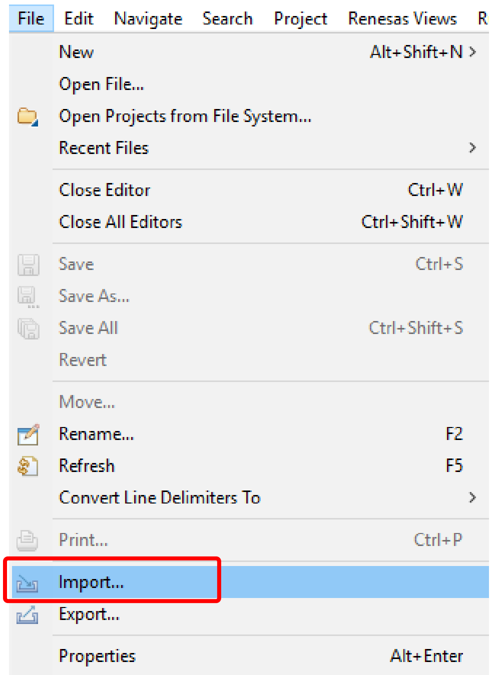

Figure 39. Import Project

2. In the **Import** dialog, select **General** → **Existing Projects into Workspace**. Click **Next**.

Note: To rename the project to be imported, select **General** → **Rename & Import Existing Projects into Workspace** instead.

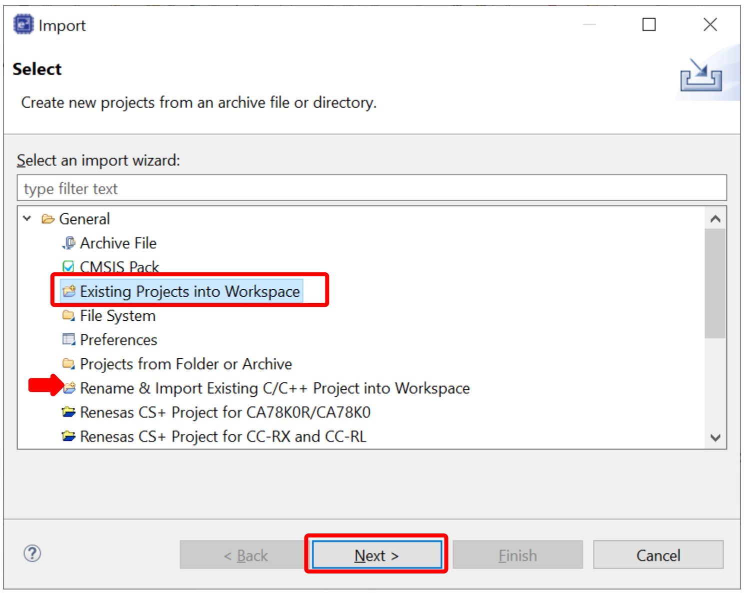

Figure 40. Select Type Of Import

3. In the **Import Projects** dialog, select **Select archive file:** then **Browse…** to browse to the compressed file (.zip) containing the project.

   If the existing project is stored in a folder, then **Select root directory:** should be selected.

4. Select the project to import and click **Finish**.

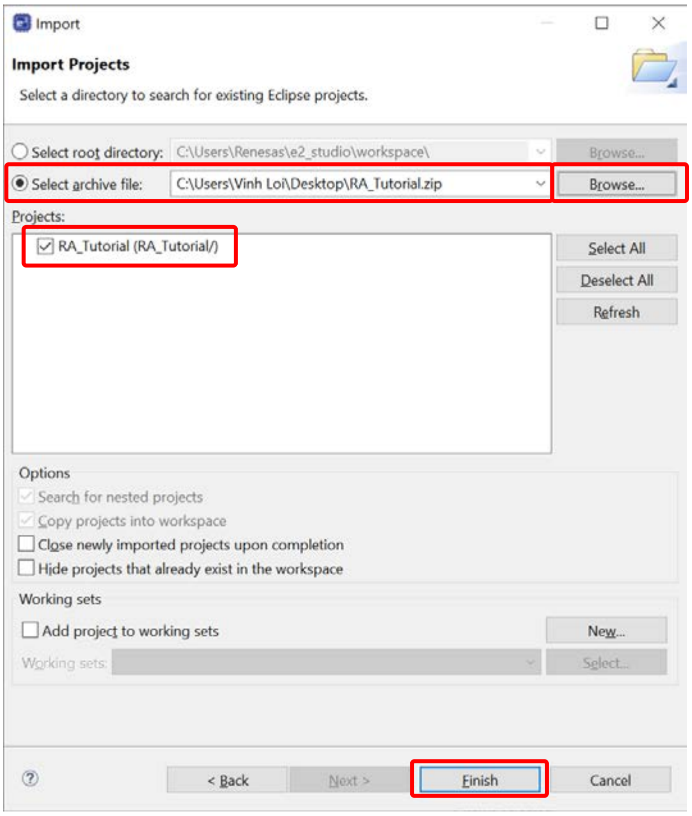

Figure 41. Select the Project in the Compressed File

5. The project will be imported to e² studio.

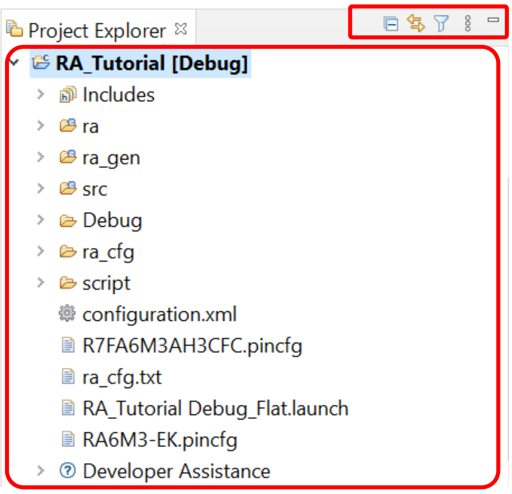

Figure 42. Imported Project

3.4 Generating and Using an RA Static Library

This section describes how to generate an RA static library project and an executable project that references the library project.

3.4.1 Creating the Static Library Project

The following steps show an example of how to create an RA static library project.

1. Select **File** → **New** → **Renesas C/C++ Project** → **Renesas RA**.

2. Select the **Renesas RA C/C++ Project** template. Click **Next** to continue.

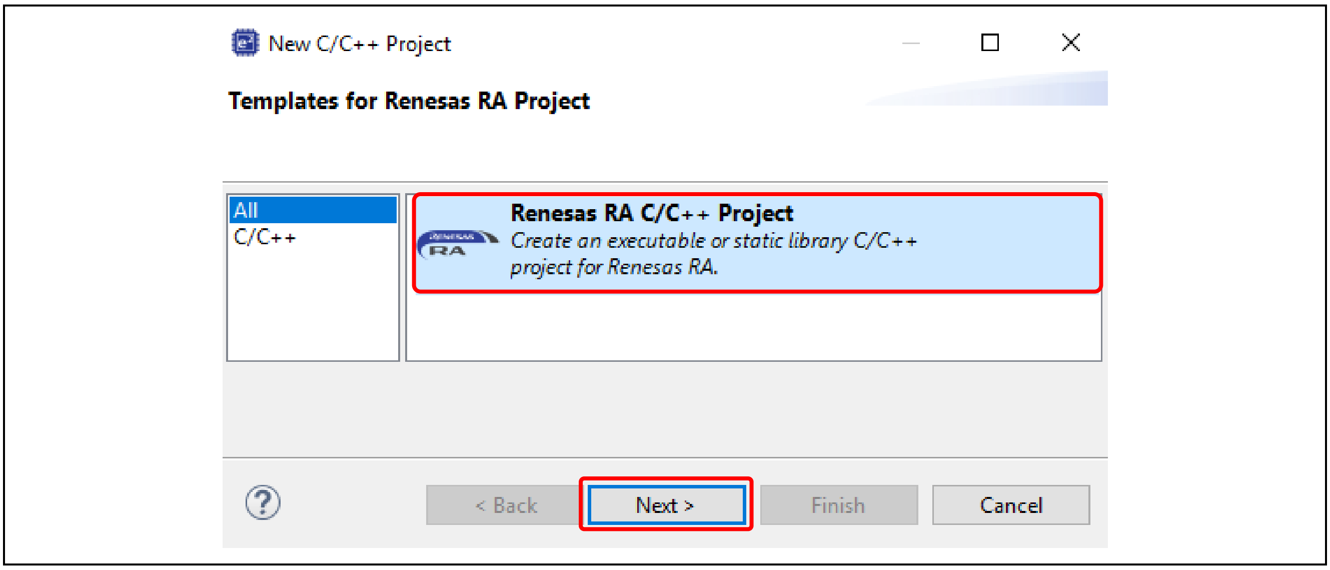

Figure 43. Project Generation – Select Library Project Template

3. On the project details page, enter a name for the static lib project (for example, **RA_Lib**) and click **Next**.

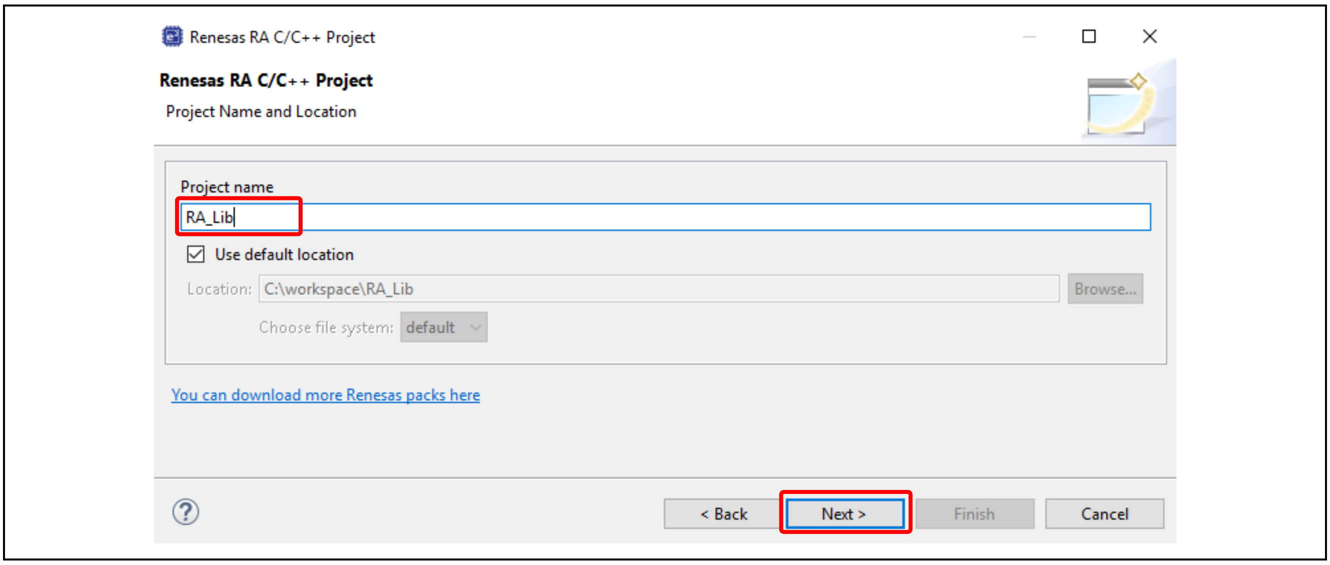

Figure 44. Library Project Configuration

4. In the **Device and Tool Selection** dialog, select a board (here we will use EK-RA6M3). Keep everything else as default and click **Next**.

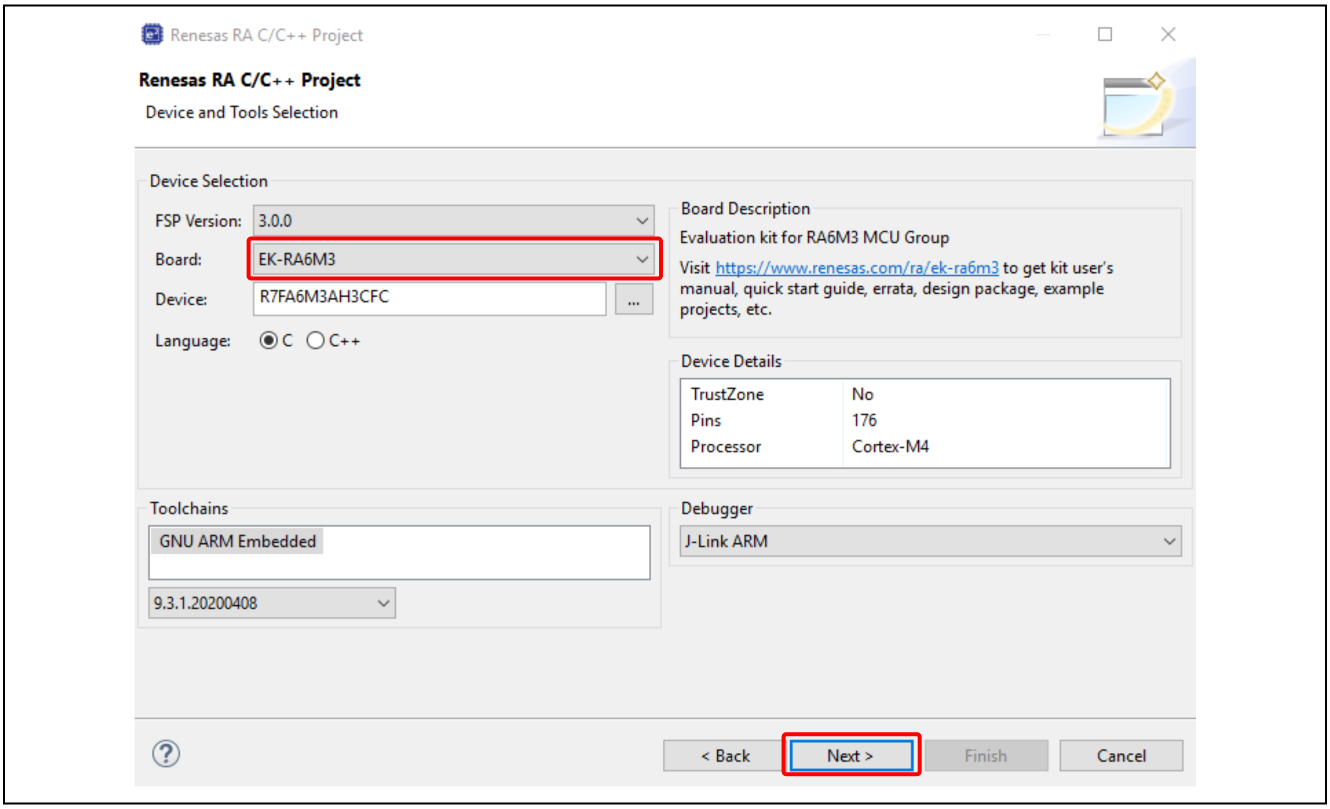

Figure 45. Select Device And Toolchain

5. **Build Artifact Selection**: **Static Library**
   RTOS Selection: **No RTOS**
   Click **Next** to continue.

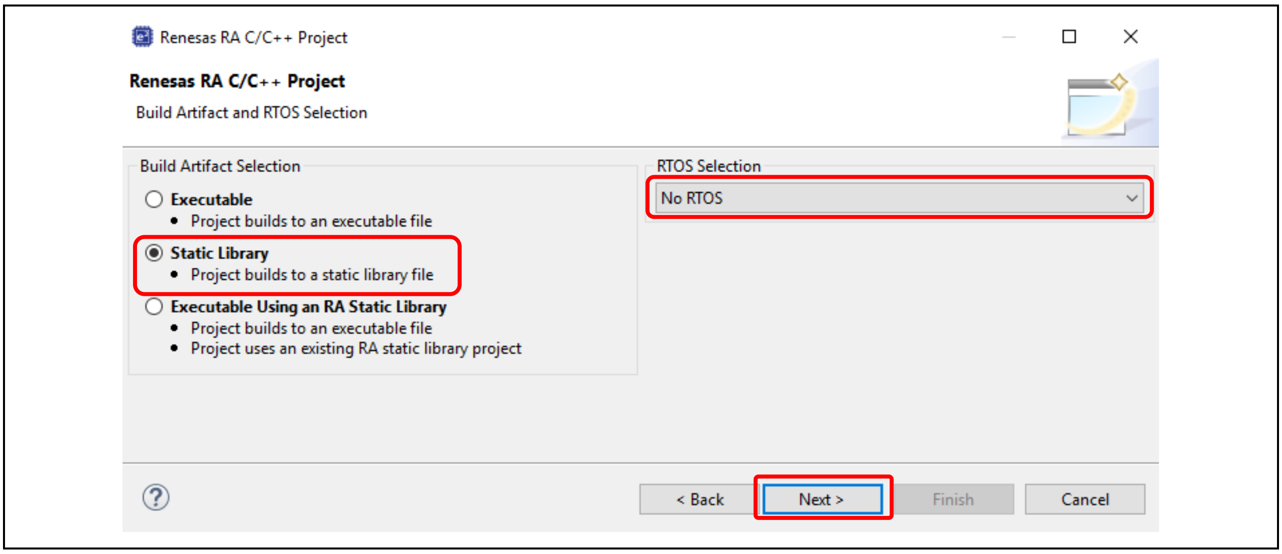

Figure 46. Artifact and RTOS Selection

6. In the project template dialog, select **Bare Metal - Blinky**, then click **Finish** to create the project.

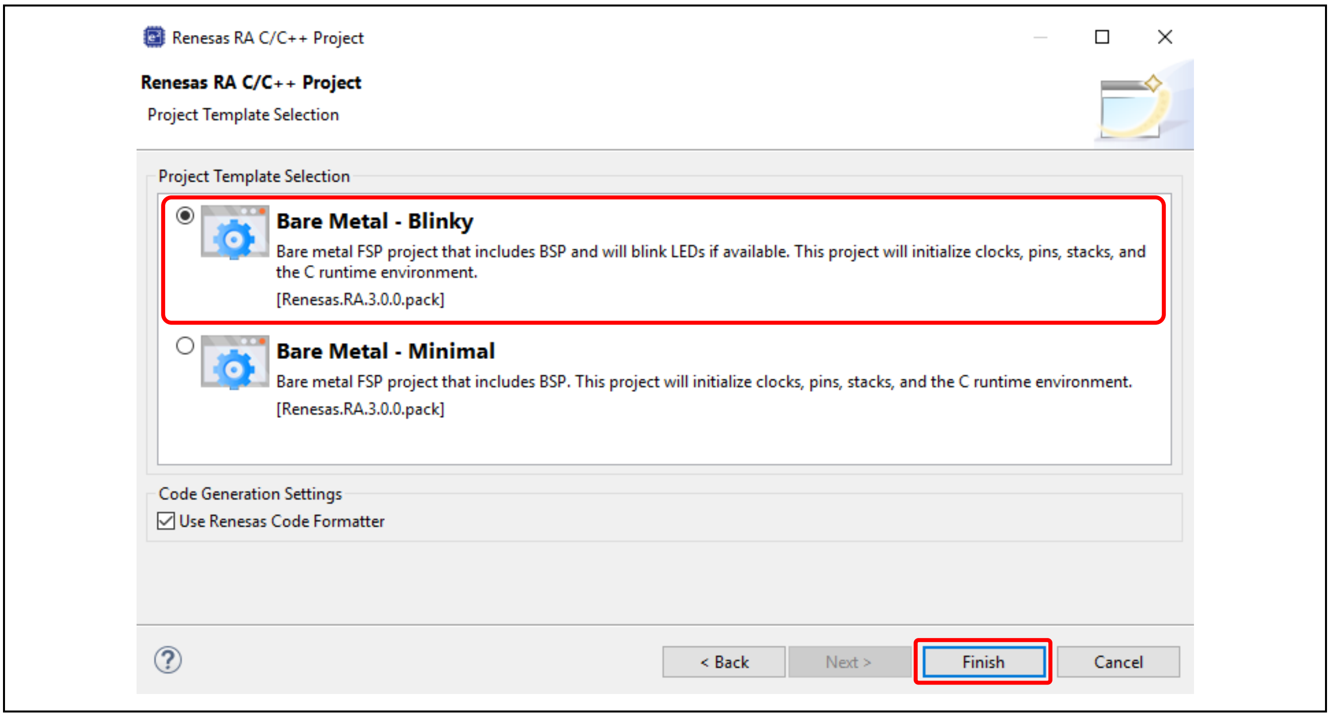

Figure 47. Select Project Template For Library

7. The e² studio may prompt you to switch to **FSP Configuration** perspective. Click **Open Perspective** to open it.
8. Click **Generate Project Content**

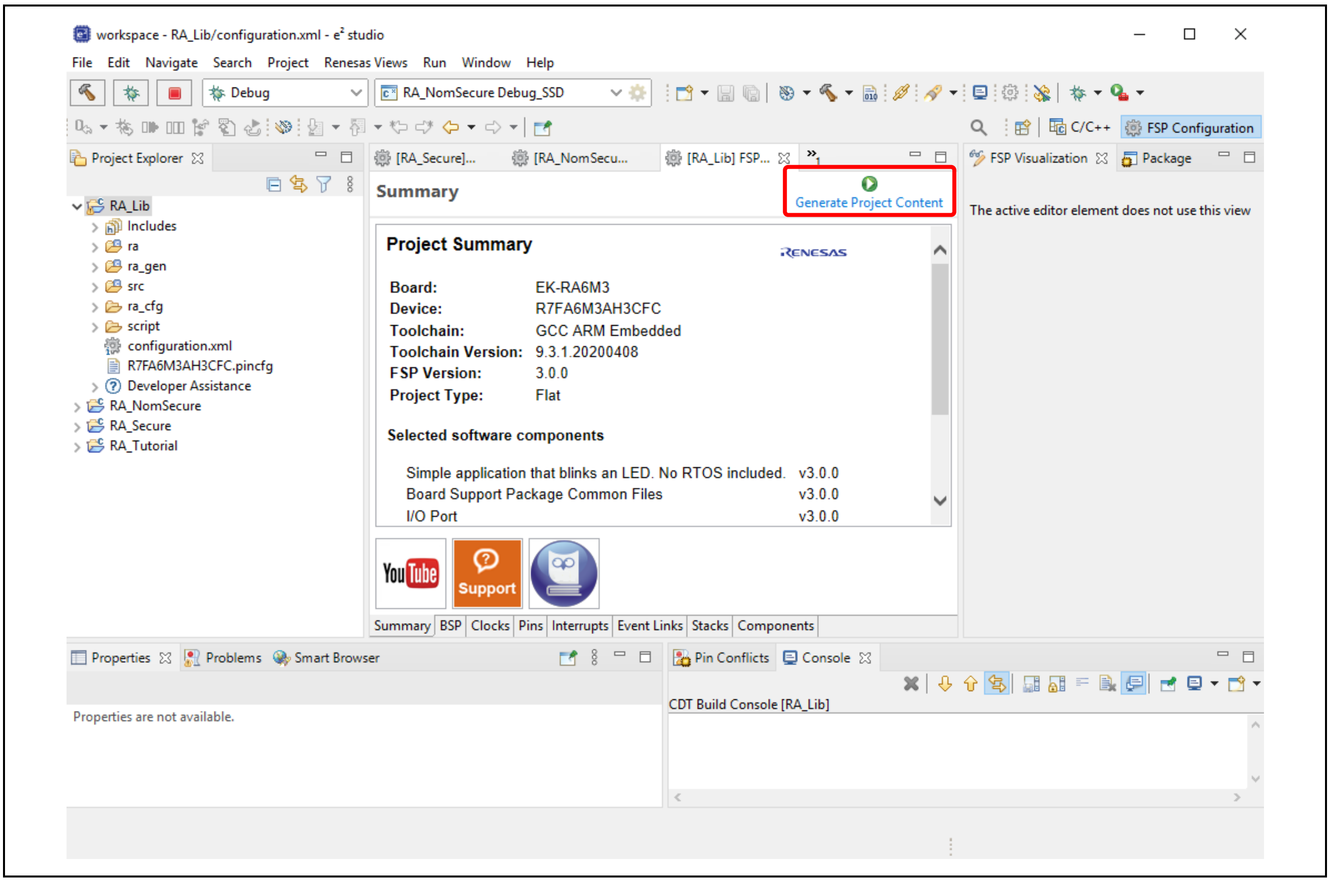

Figure 48. Generate Library Project Content

9. From the **Project Explorer** window, open `hal_entry.c` under `RA_Lib\src\`.

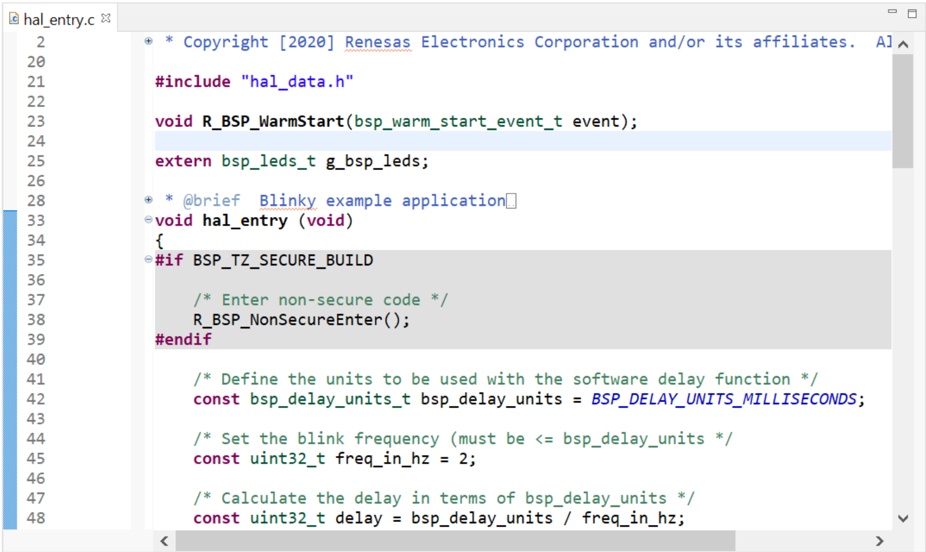

Figure 49. Old hal_entry.c

Then rename the function `hal_entry()` to `hal_entry_lib()` and add a declaration for `hal_entry_lib()`.

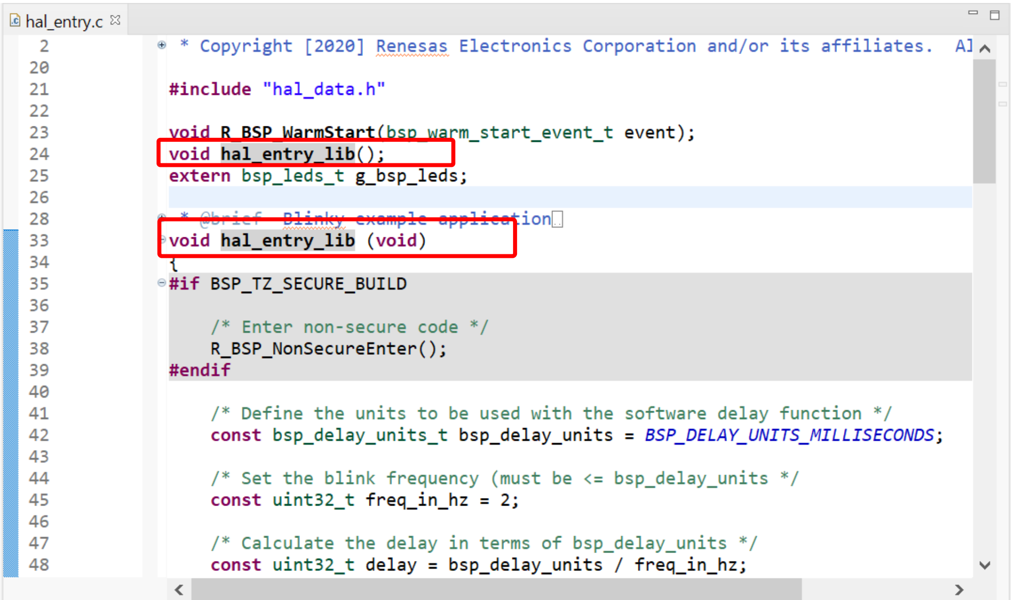

Figure 50. New hal_entry.c

10. Build the Library Project. The build outputs a static library file `libRA_Lib.a`.

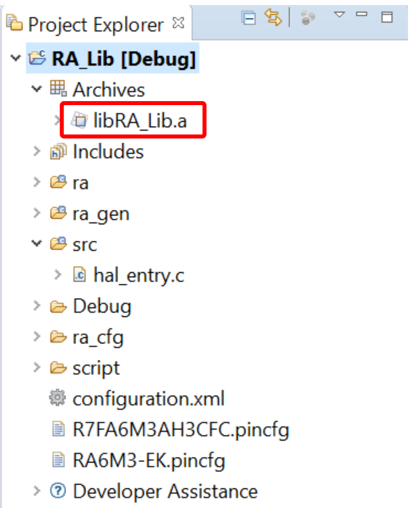

Figure 51. Built Static Library

3.4.2 Using Static Library in Executable Project

This section shows how to use the static library created in the previous section (3.4.1) in a RA executable project by performing the following steps:

- Create a RA C executable project
- Modify the source code to call a function `hal_entry_lib()` declared in the static library project
- Build and run the RA C executable project

Follow the following steps:

1. Select **File** → **New** → **Renesas C/C++ Project** → **Renesas RA**.
2. Select **Renesas RA C/C++ Project** template. Click **Next** to continue.
3. Enter project name: **RA_App** and click **Next** to continue.
4. In the **Device and Tool Selection** dialog, select a board (here we'll use EK-RA6M3). Keep everything else as default and click **Next.**
5. On the **Build Artifact and RTOS Selection** page, select **Executable**. In RTOS selection, select **No RTOS**. Click **Next**

   Note: On the **Build Artifact and RTOS Selection** page, **Executable Using an RA Static Library** is unavailable.

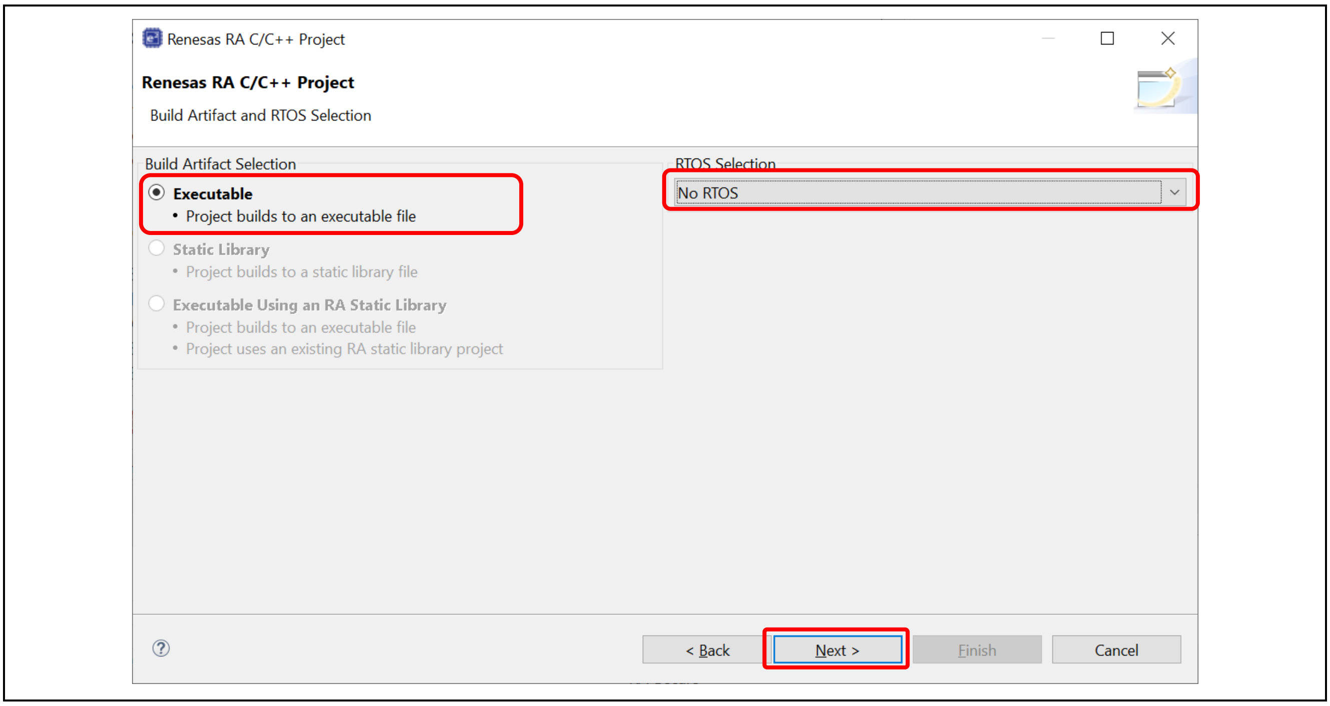

Figure 52. Artifact and RTOS Selection

6. In the project template dialog, select **Bare Metal - Minimal**. Click the **Finish** button to create the project.

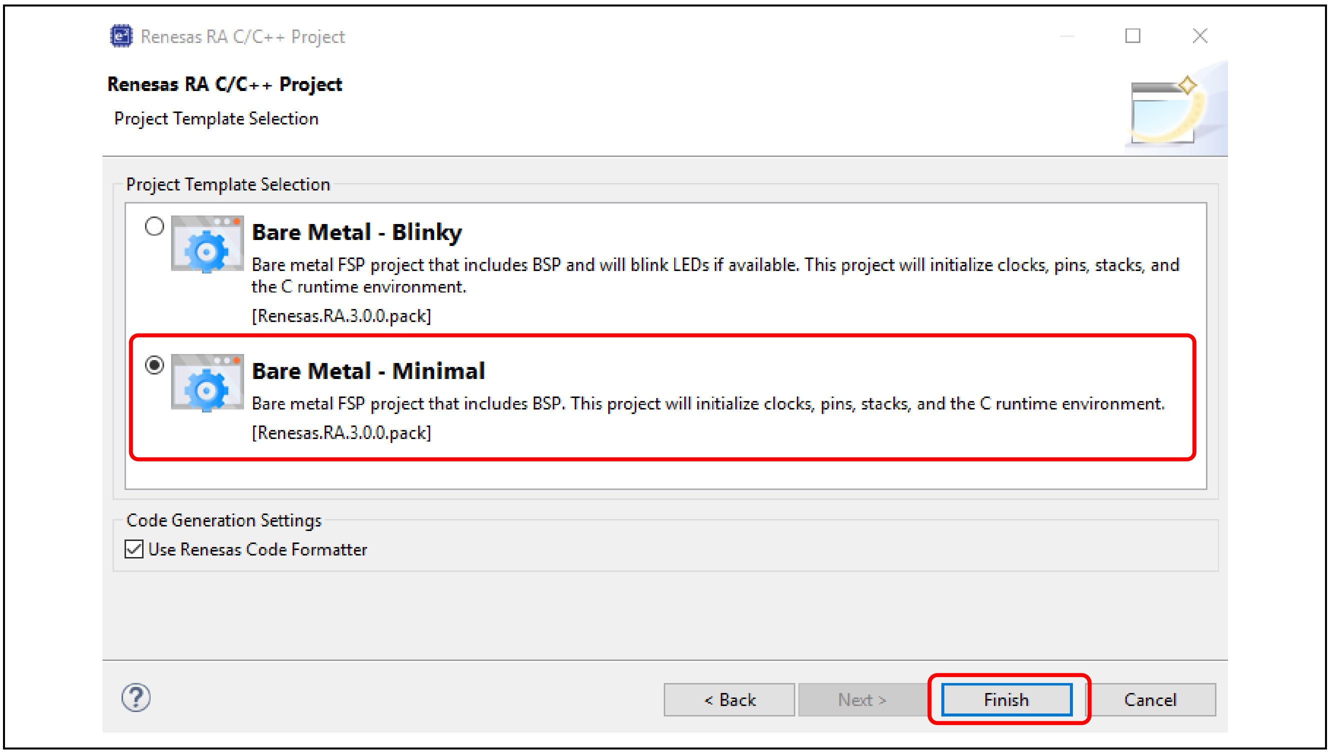

Figure 53. Project Generation – Project Template

7. Add the Existing RA Library.
   Settings for the Project Properties.
   — Select and add **Settings** → **Libraries** → **Libraries (-l)** → **RA_Lib**
   — Select and add **Settings** → **Libraries** → **Libraries search path (-L)** → **"${workspace_loc:/RA_Lib/Debug}"**

— Select **Apply and Close**

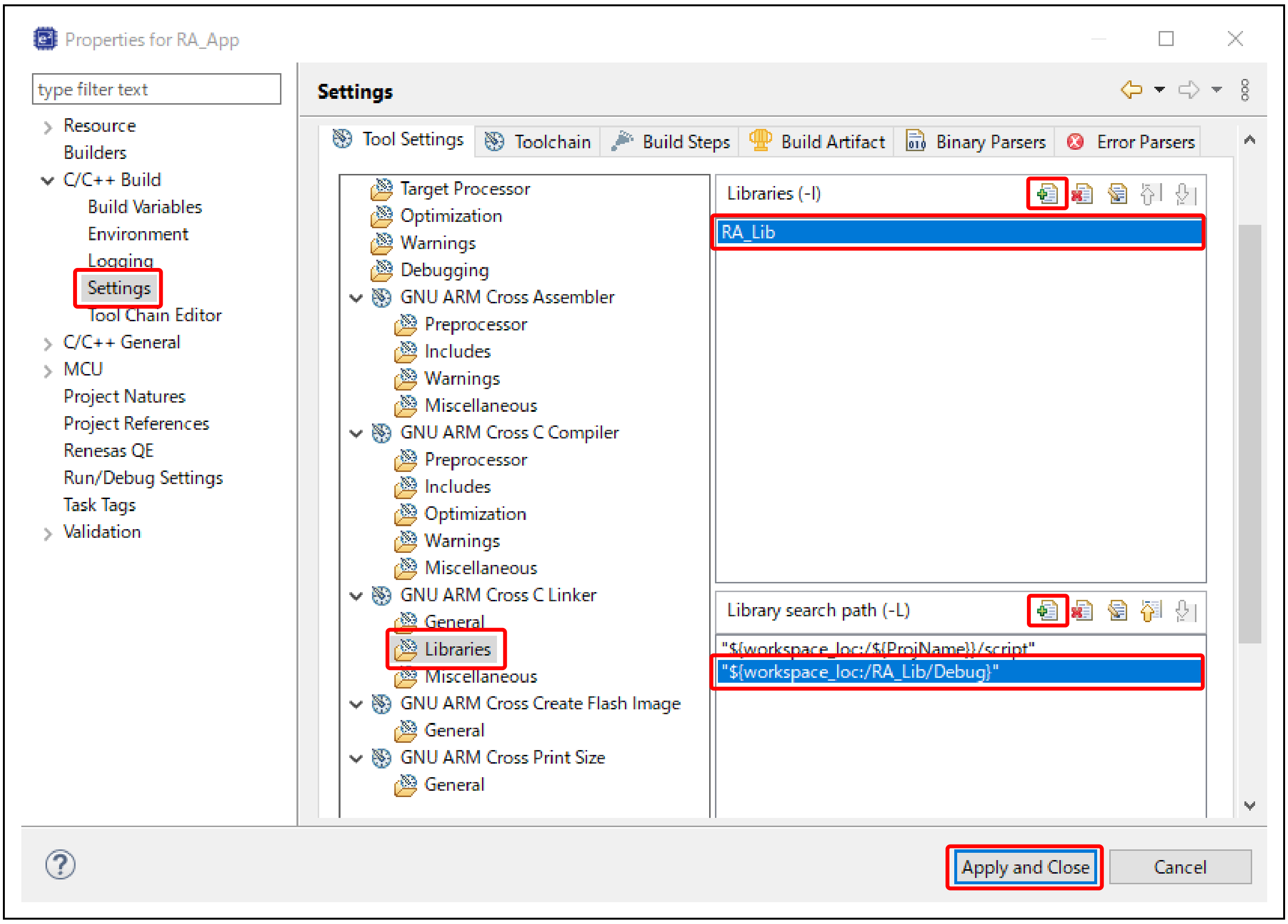

Figure 54. Settings the Project Properties

8. From the **Project Explorer** window, open `hal_entry.c` under `RA_App\src\`.

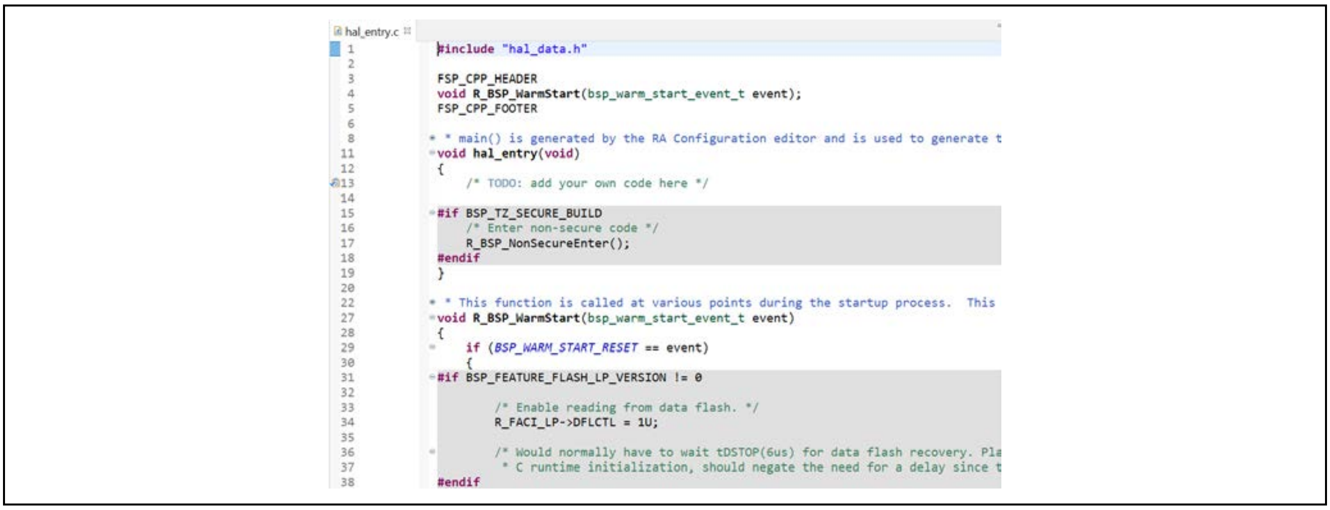

Figure 55. Old hal_entry.c

Remove function `R_BSP_WarmStart()` and its declaration.

Add codes to call the LED blinking library function `hal_entry_lib()` in the `hal_entry()` function and add a declaration for the library function.

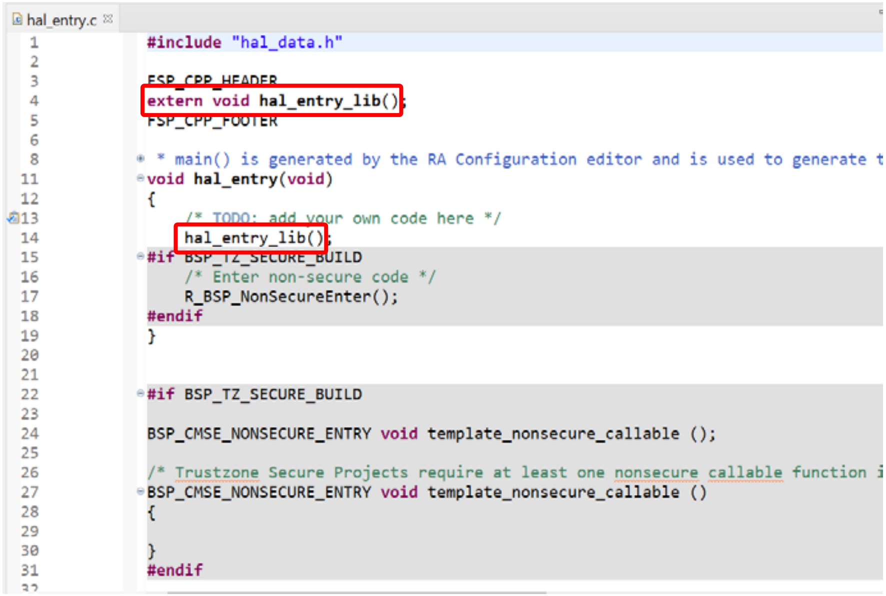

Figure 56 New hal_entry.c

9. Build the application project.
10. Set a breakpoint where the library function `hal_entry_lib()` is called. Run the RA_App project. When the program stops at the breakpoint, resume it. Confirm that the library function which blinks the LEDs (for example, `hal_entry_lib()`) is executed.

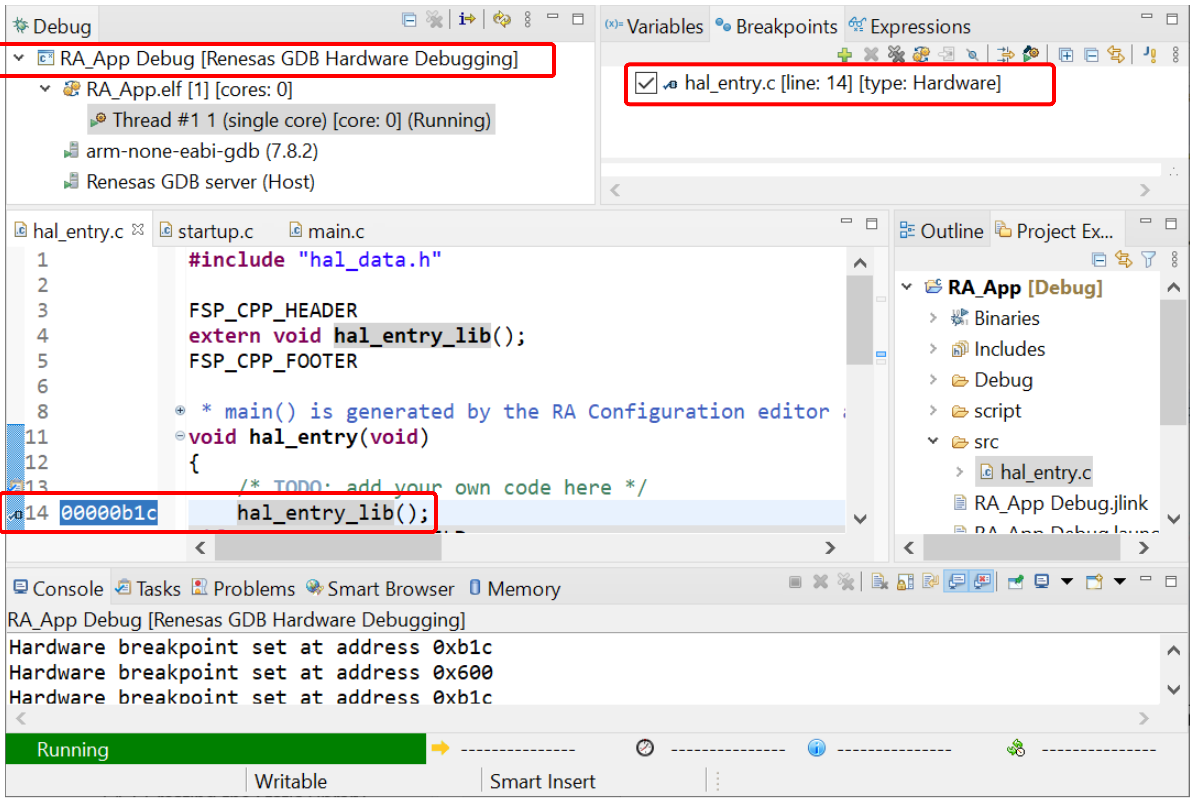

Figure 57. Application Project Executing Library Function

3.5 RA Project Configuration Editor

The **RA Project Configuration** editor view displays the current project configuration settings. The settings are saved in the file `configuration.xml`. The project configuration settings are grouped into multiple pages that allow you to set several configurable aspects of the project such as how pins and clocks are set up and which drivers are included. Drivers can range from simple hardware-level drivers to RTOS-aware applications. Multi-thread specific components like mutexes, semaphores, and events can be configured.

To edit the project configuration, make sure that:

- **FSP Configuration** perspective is selected in the upper right-hand corner of the e[2] studio window or click **Window → Perspective → Open Perspective → Other… → RA Configuration**
- The `configuration.xml` file is opened.

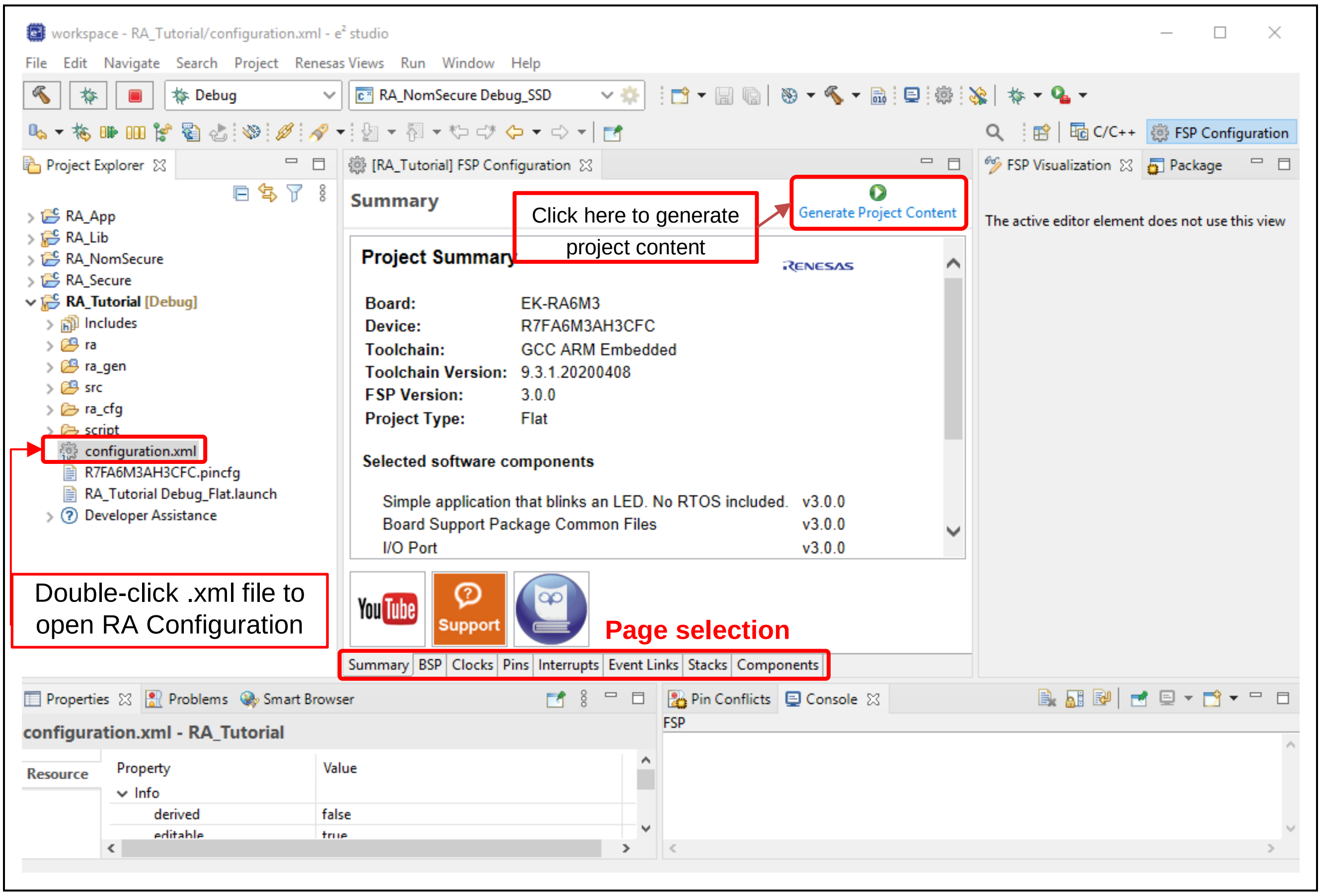

Figure 58. RA Project Configuration – RA Project Configuration View

There are eight pages (or tabs) in **RA Project Configuration** editor.

The **Summary** page contains a project-specific summary information.

The **BSP** page allows users to select the FSP version, the type of RA board, and the device.

The configuration steps and options for the **Clocks**, **Pins**, **Interrupts**, **Event Links**, **Stacks**, and **Components** pages are discussed in the following sections.

3.5.1 Summary Page

The **Summary** page contains a project-specific summary which includes details of the currently selected device, board, and RA software components. There are also useful links to the 'Renesas Presents' YouTube channel and the FSP user manual.

If you add new threads and modules/objects to a thread, this information will be also shown in the **Summary** page.

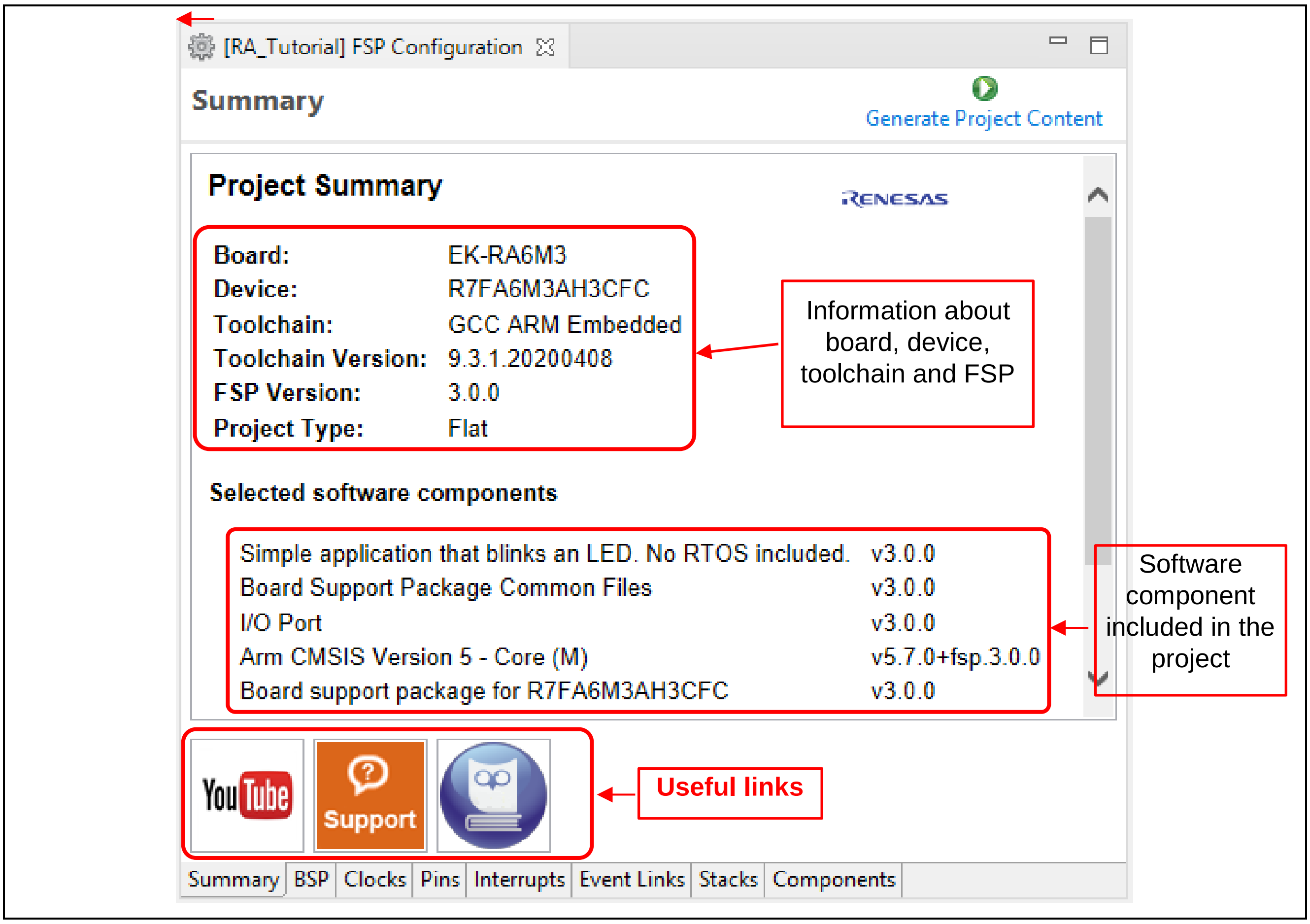

Figure 59. Summary Page

3.5.2 BSP Page

The **BSP** Page allows user to select the FSP version, board, and device. You can also import the CMSIS board information from this page.

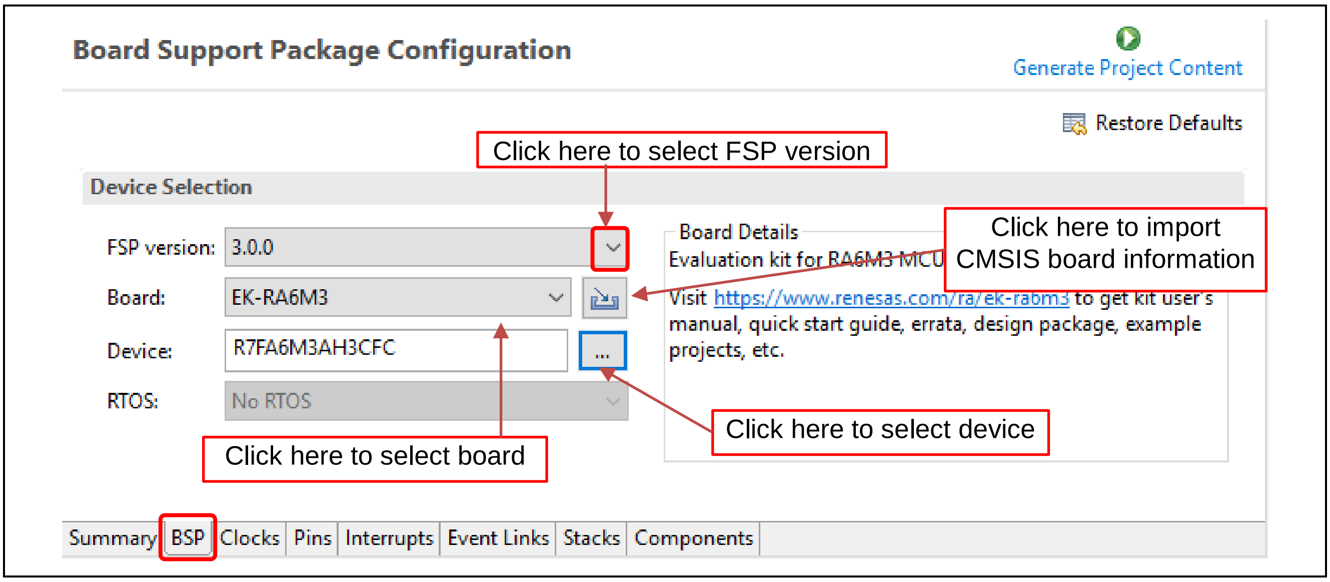

Figure 60. RA Project Configuration – BSP Page

When the **BSP** page is selected, the e² studio **Properties** view will display the available properties for the selected board. These properties may be modified in the **Properties** view as necessary according to the requirements of the project.

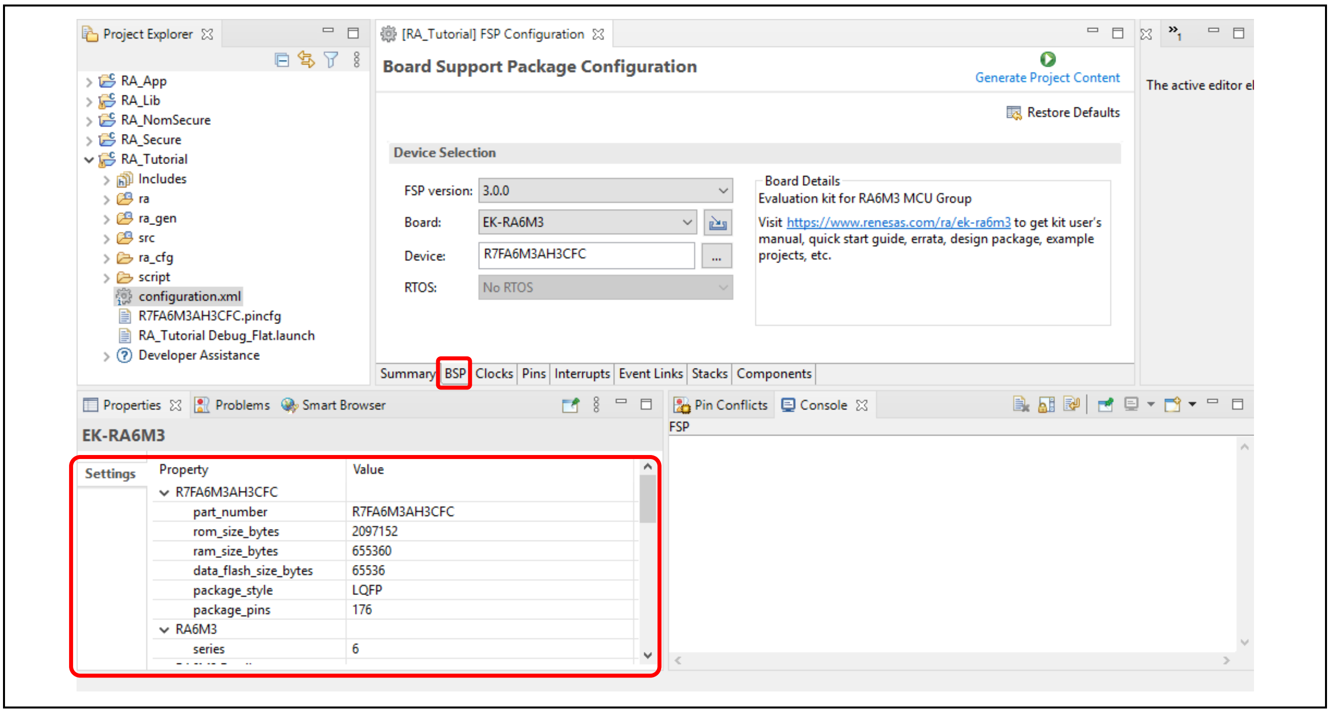

Figure 61. Board Properties

3.5.3 Clocks Configuration Page

The **Clocks Configuration** page sets up the initial clocking for the application. Clock sources, PLL settings, and clock divider settings can be selected for each of the output clocks.

For details on the Clock Generation Circuit (CGC), see the RA hardware user's manual. To update the project, follow these steps:

1. Select a value in the drop-down list for the clock setting on GUI.
   Note: If the value goes out of range, it will turn red and the ! mark will be displayed.

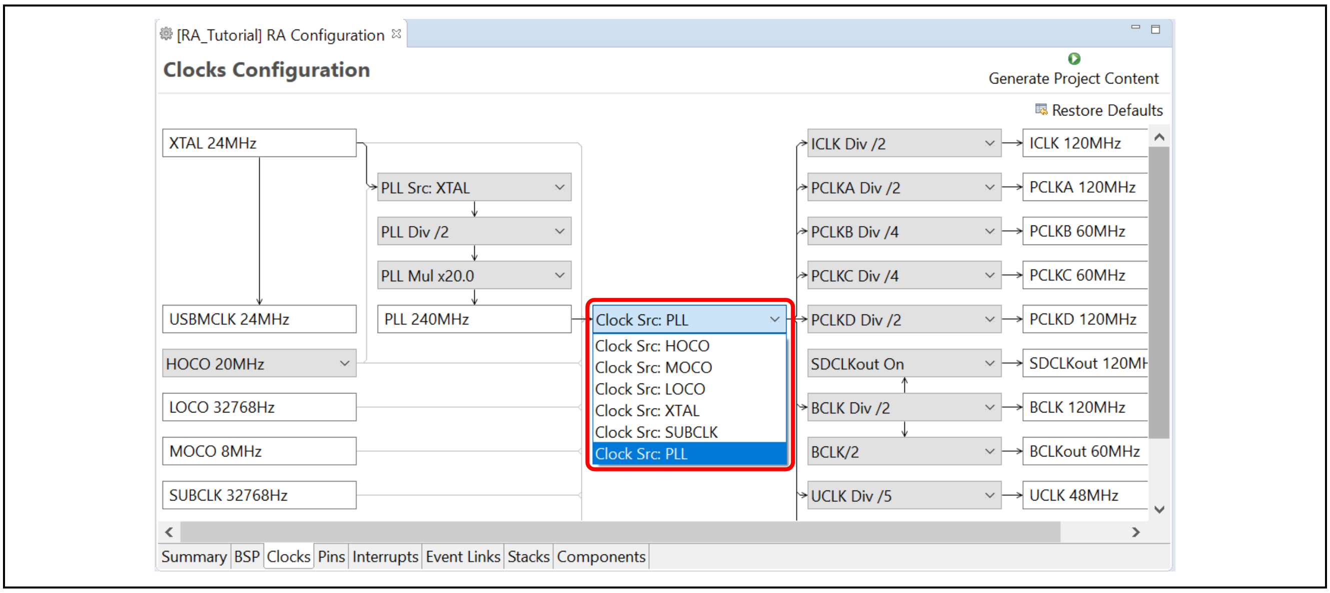

Figure 62. RA Project Configuration – Clocks Configuration

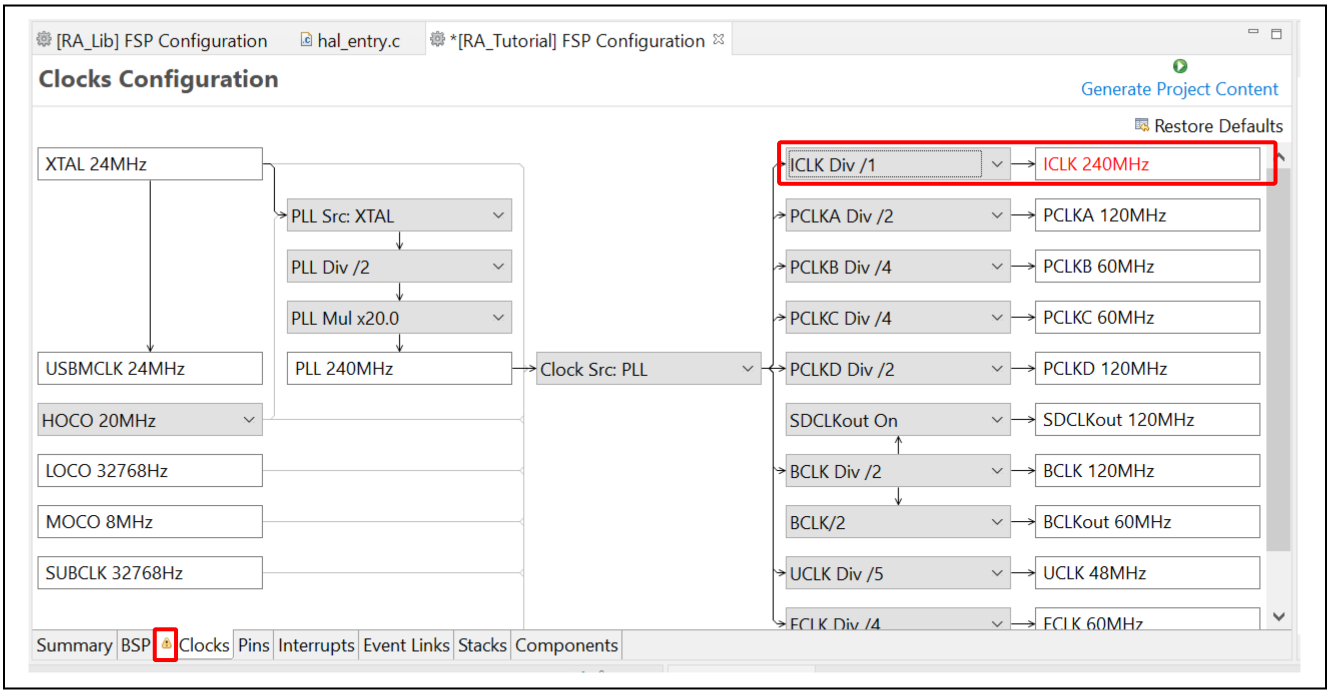

Figure 63. Value Goes Out of Range

2. Save the Project Configuration settings, for example by using the **Ctrl-S** shortcut.
3. Click the **Generate Project Content** button.

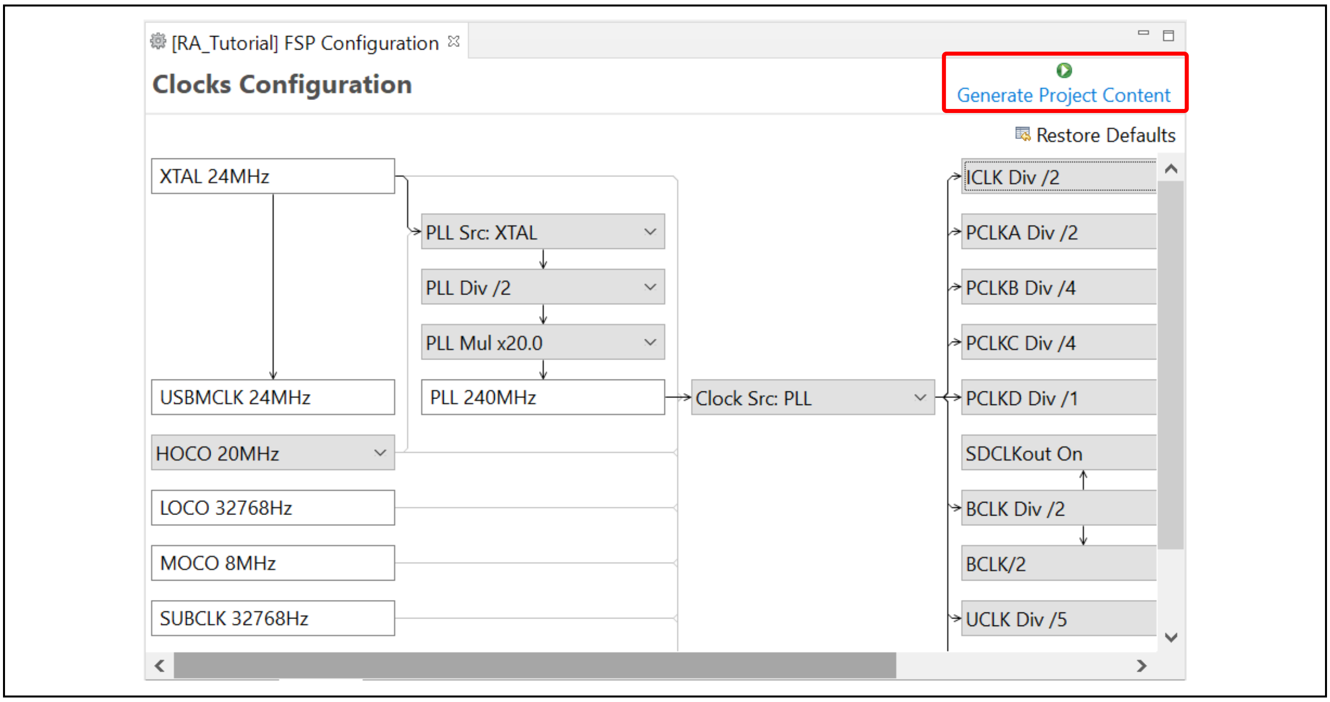

Figure 64. Generate Project Content

4. The file `bsp_clock_cfg.h` is updated with the selected clock configuration.

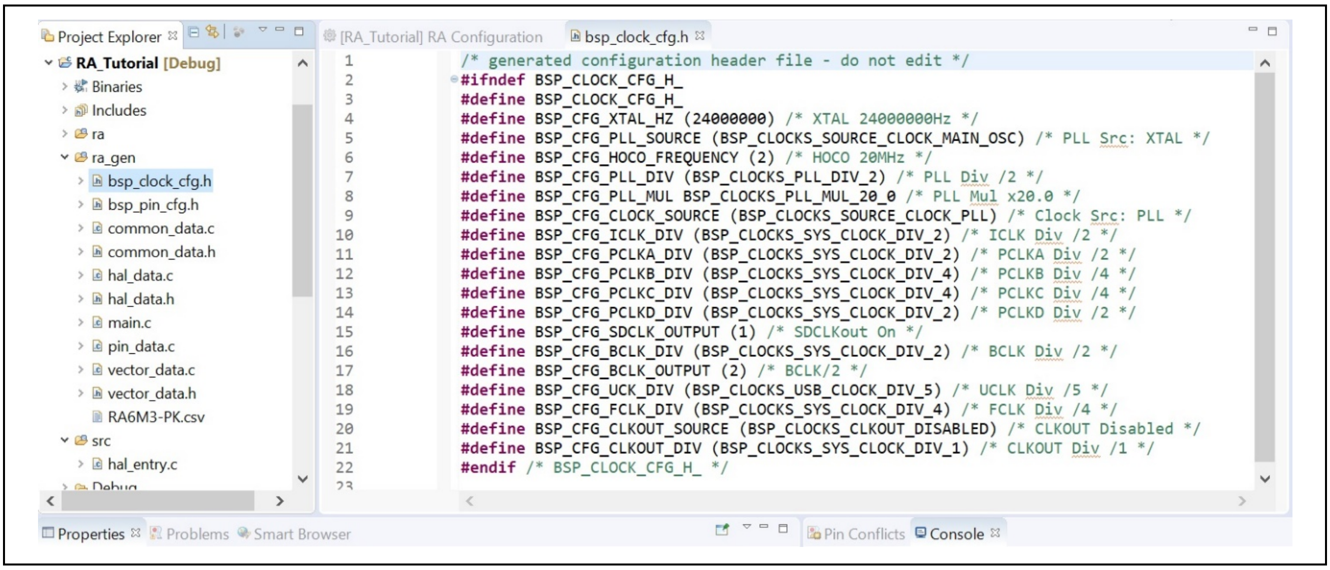

Figure 65. bsp_clock_cfg.h is Updated

3.5.4 Pin Configuration Page

The **Pin Configuration** page provides a graphical user interface for generating the pin configuration settings for the project.

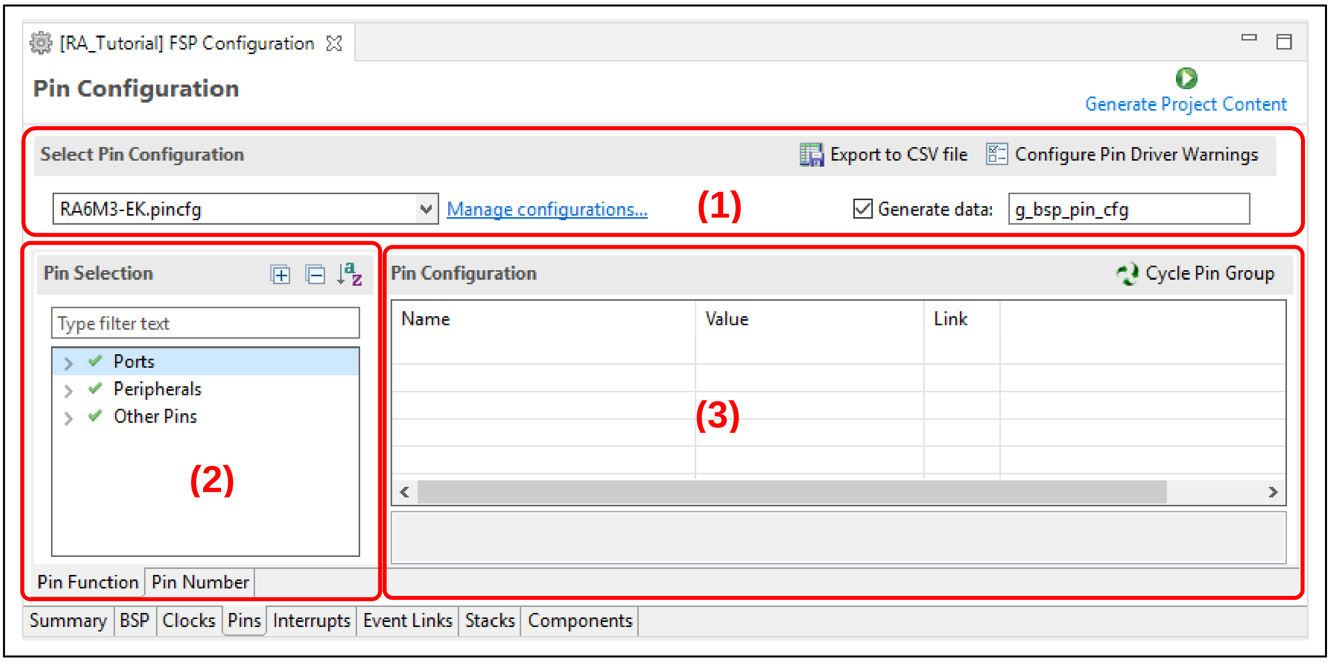

Figure 66. RA Project Configuration – Pin Configuration GUI

The **Pin Configuration** window consists of 3 parts:

1. **Select Pin Configuration**: Selects the pin configuration file and specifies the name for the associated data structure. Multiple pin configurations can be set as follows:
   a. Create a new `.pincfg` file (for example, `NewName.pincfg`) in Project Explorer by copying an existing one.
   b. Select the new `.pincfg` file (for example, `NewName.pincfg`) in the **Select pin configuration** dialog box.
   c. Check the **Generate data** check-box and give the new pin configuration a unique data structure name in the text field.
   d. The multiple pin configurations will be created in different data structures.
2. **Pin Selection**: Selects pin or peripheral that will be set up.
3. **Pin Configuration**: Sets up for function/property of the selected pin/peripheral.

The best way to configure pins is to configure the peripherals to be used in the project using the steps below:

1. Select a peripheral in the **Pin Selection** pane, for example, **Connectivity:SCI** → **SCI4**. The configuration for this peripheral will be shown in the **Pin Configuration** pane.
2. Select an **Operation Mode** for the peripheral, for example, **Simple SPI**.
3. Select the pins you would like to use for the input/output functions of the selected peripheral in the selected mode.

   Note1: The lock functionality prevents the pin assignment from changing, but there is no impact to the generated code whether you lock or don't lock.

   Note2: You can jump to the port setting by clicking the arrow on the right side of the pin.

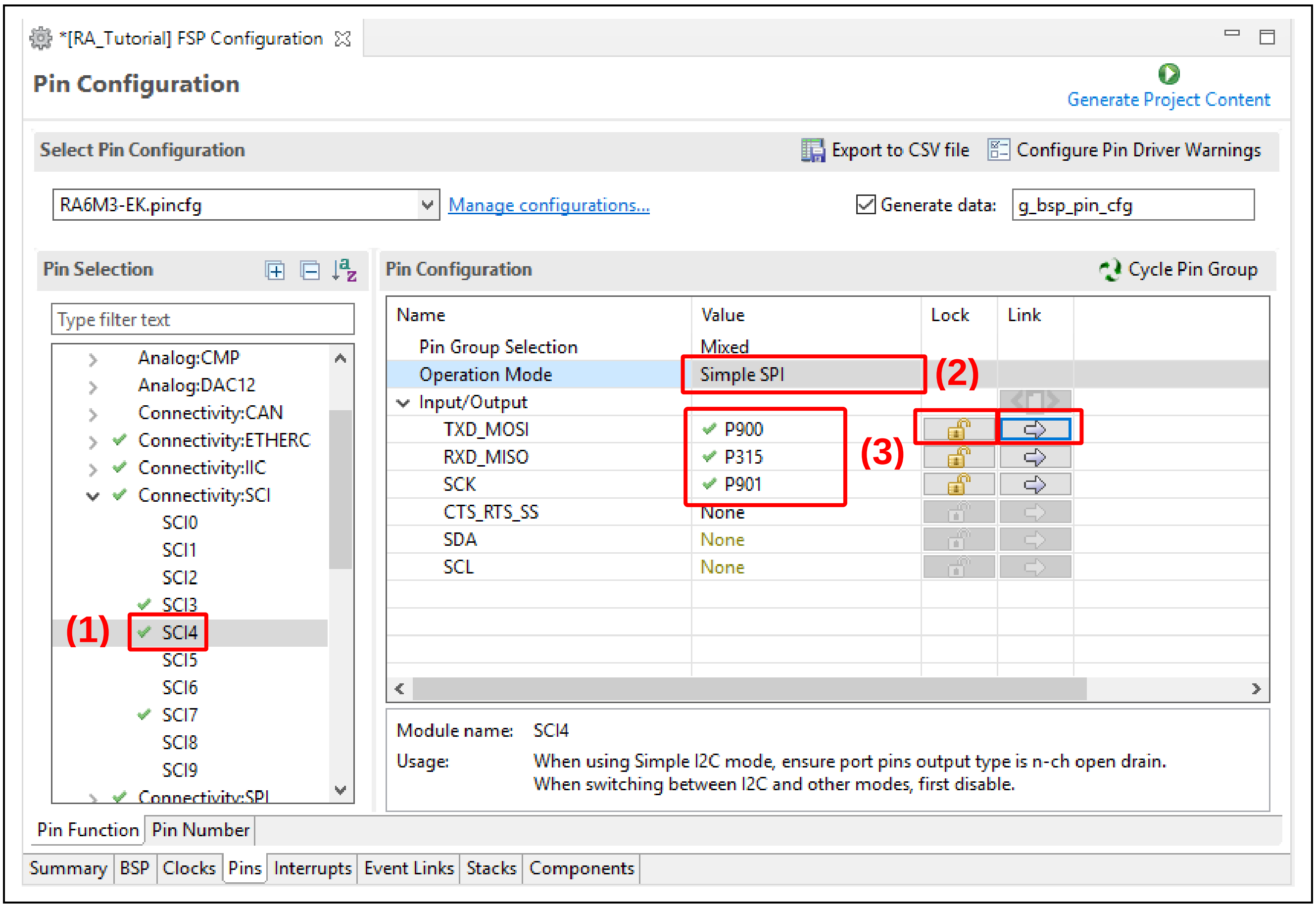

Figure 67. RA Project Configuration – Pin Configuration Setting (By Peripheral)

A single pin can also be set up following the steps below:

1. Select a pin in the **Pin Selection** pane, for example, **Ports** → **P0** → **P003**. The configuration for this pin will be shown in the **Pin Configuration** pane.
2. Enter properties for this pin, for example:

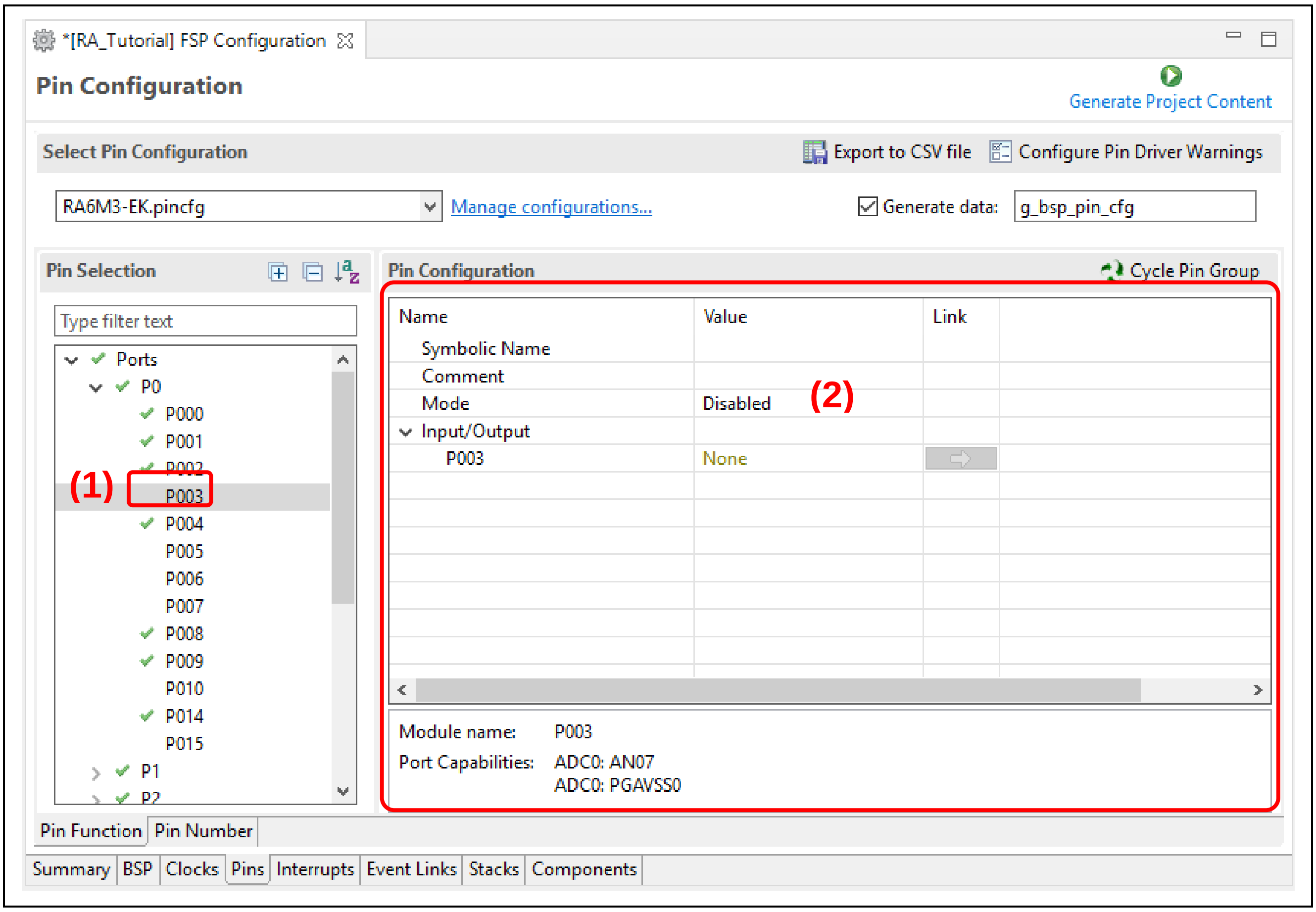

Figure 68. RA Project Configuration – Pin Configuration Setting (By Single Pin)

3. The **Visualization** view shows this pin change.

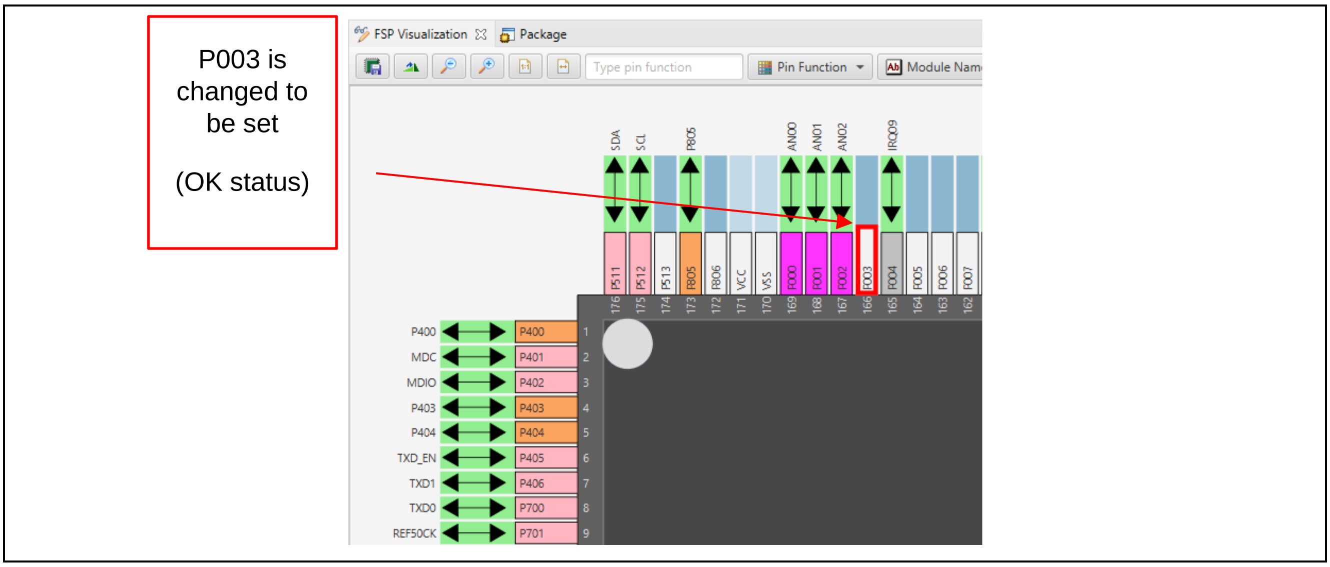

Figure 69. RA Project Configuration – Package View (Connection Status)

It is possible to migrate a pin configuration from one device to another device on this page. Use the **Import a pin configuration** button on the toolbar to perform this migration. This function allows migration of the pin configuration to the new device while retaining the user setup.

To import an existing pin configuration to the current project, click **Manage configurations…** → **Import** and select the pin configuration file to import.

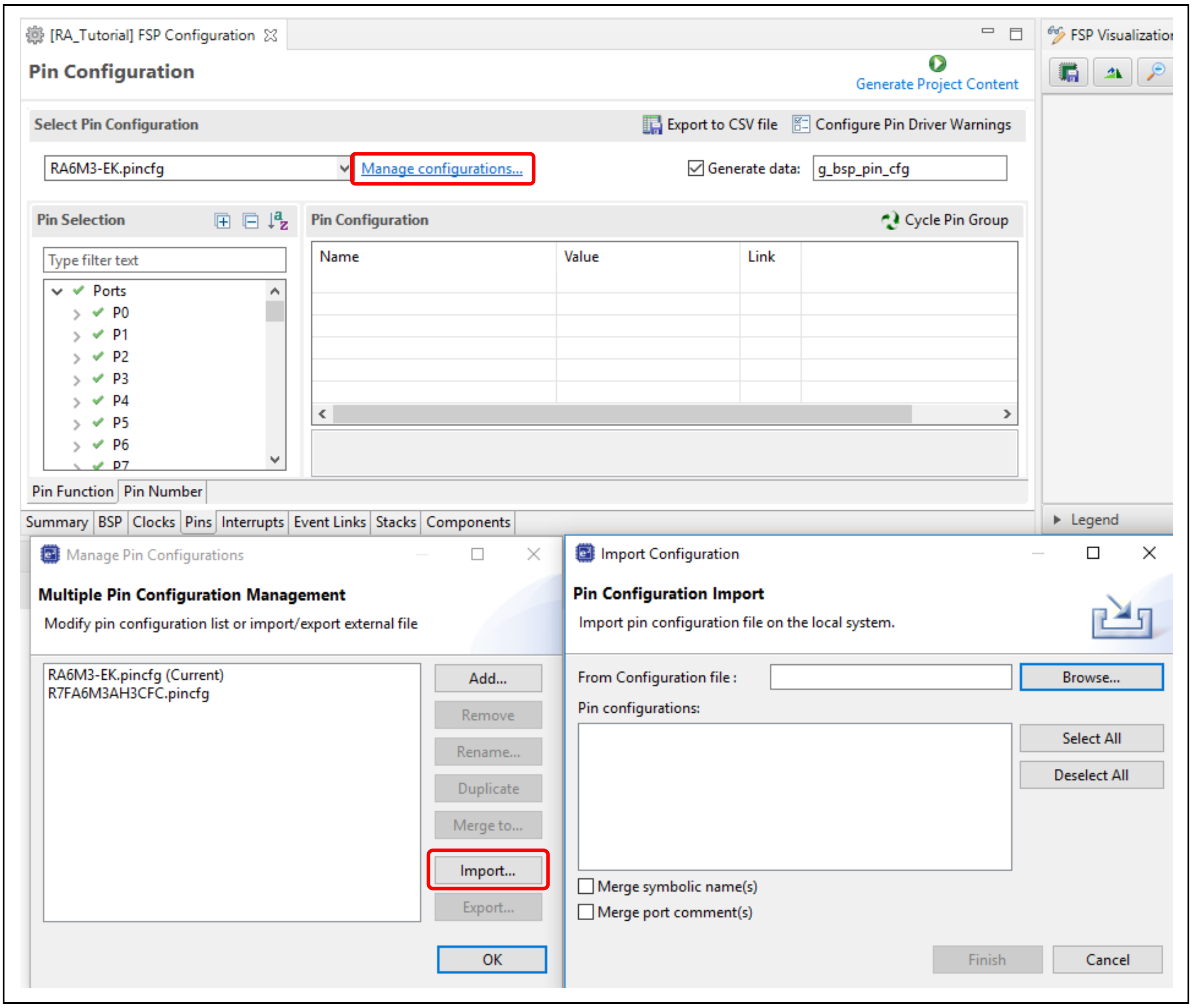

Figure 70. Import An Existing Pin Configuration To The Current Project

The import function might prompt the user about conflicts and provide the following options for the user:

- Cancel the import operation
- Ignore the conflicts and import the conflicting settings anyway
- Continue the import operation without importing the conflicting settings.

3.5.5 Stacks Configuration Page

The **Stack Configuration** page allows you to:

- Configure threads within a RA project.
- Add RA modules and objects to a thread.
- Modify module and object properties in the Properties View.

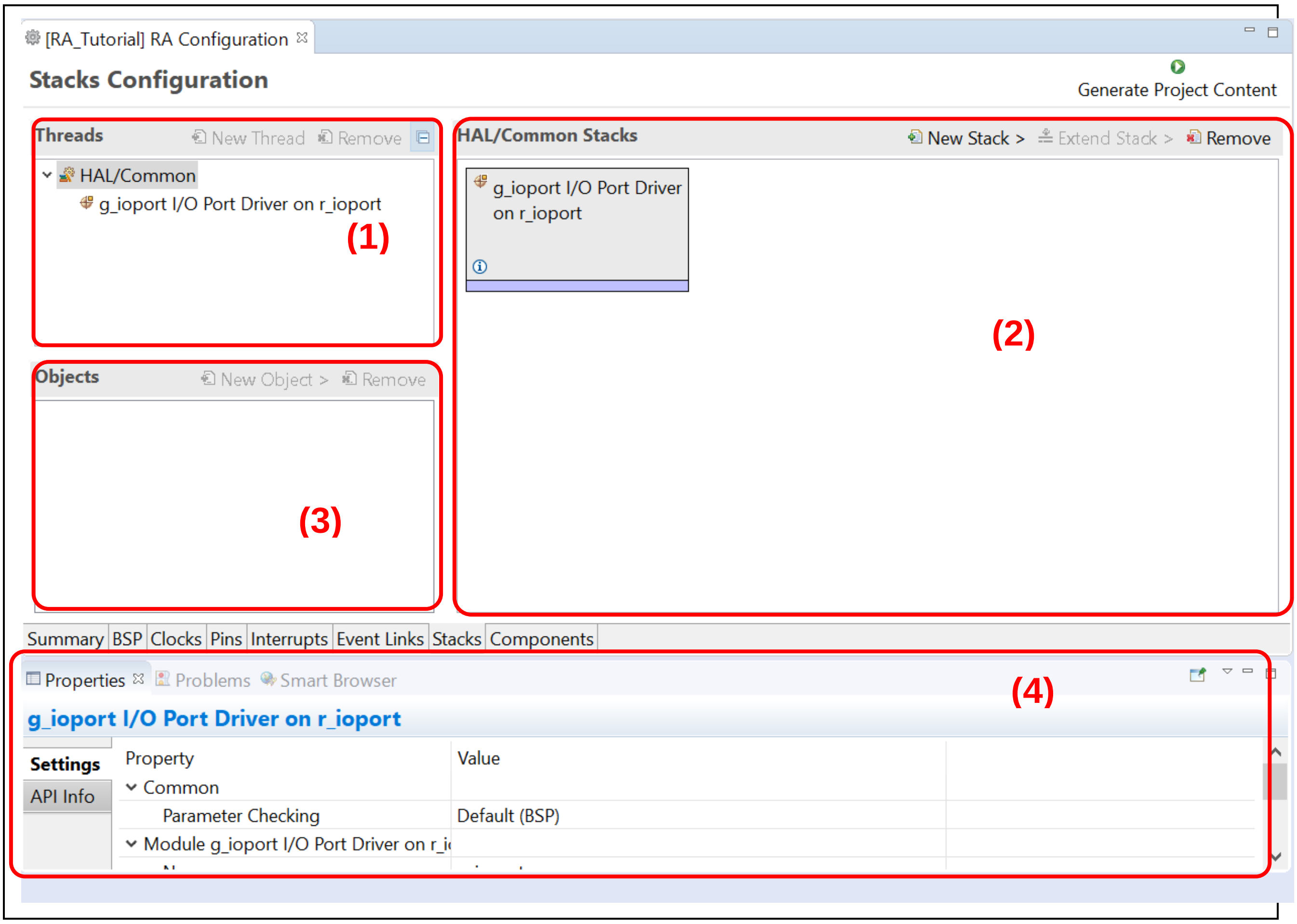

Figure 71. RA Project Configuration – Stacks Configuration GUI

The **Stacks Configuration** page consists of 3 panes:

1. **Threads** pane: Add/remove threads. Details are explained in Section 6.
2. **Stacks** pane: Add/remove FSP module instances, that is, IO port, SCI, UART, etc.
3. **Objects** pane: Add/remove kernel objects. Details are explained in Section 6.

In addition, the **Properties** view supports the **Threads Configuration** and is used to modify module/object properties.

A module can be added to the existing project following the steps below:

1. Select a thread, such as **HAL/Common**. The modules and objects in this thread are shown.
2. In the **Stacks** pane, click **New Stack** to add a module to the thread, that is, **New Stack** → **Driver** → **Monitoring** → **Clock Accuracy Circuit Driver on r_cac**.
3. Click the **Generate Project Content** button to generate the source code content.
4. The **Properties** view shows the properties of the selected module. Users can change them according to their requirements.

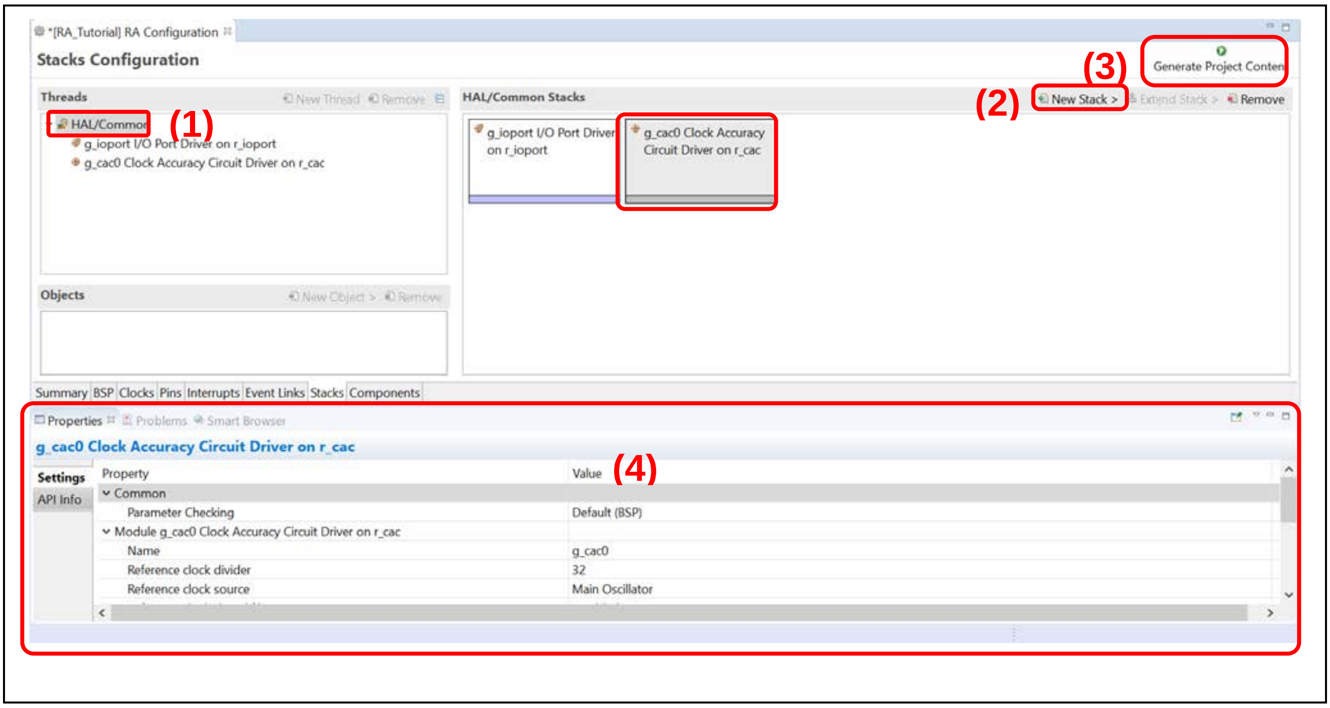

Figure 72. RA Project Configuration – Add New Module To Thread

Note: For another example, refer to section 6.1 which describes the procedure to add General Purpose Timer (GPT) module to the Blinky Thread.

An added module may require dependent modules or configuration settings. Necessary dependent modules will be added automatically. Optional dependent modules should be added manually by the user. In this case, users should click on the suggested modules to add and to configure its properties (for example, **Add New Stack → Middleware → USB → USB PCDC driver on r_usb_pcdc**).

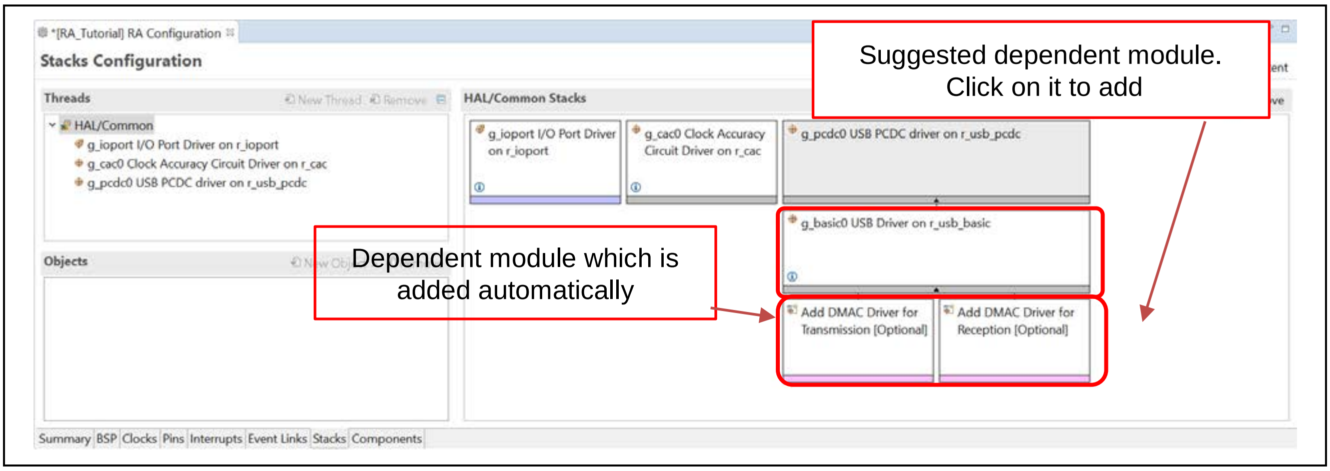

Figure 73. RA Project Configuration – Dependent Module

A module or a module stack can also be added by performing a copy-and-paste operation in the **Threads Configuration** page. Right-click on a module and select **Copy** to copy it. Right-click in the stack pane of the same or a different thread in the same project and select **Paste**. A copy-and-paste operation is also available.

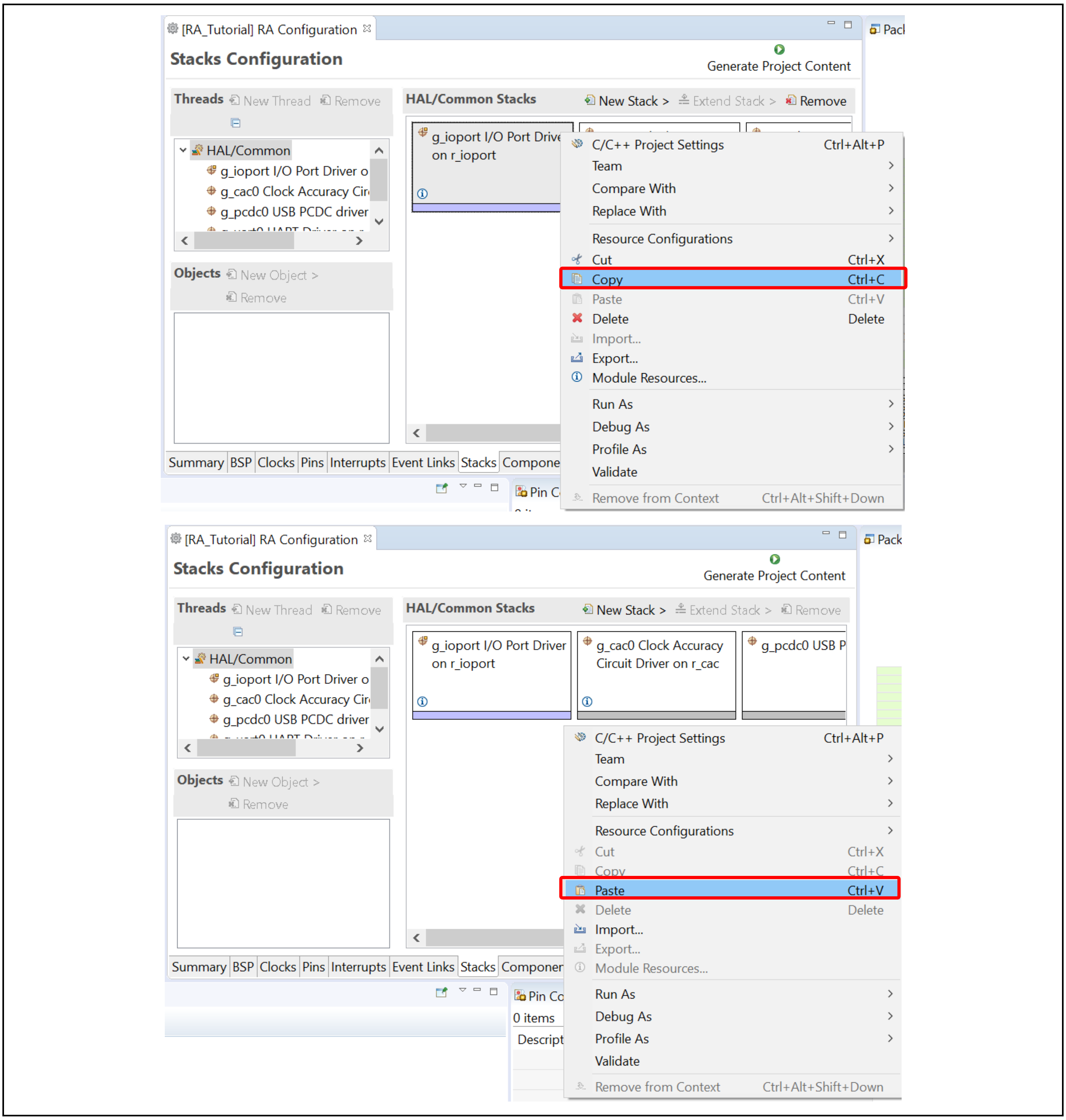

Figure 74. Copy and Paste Operation

There will be a name conflict between the old module instance and the new one. Renaming one of the module instances will solve the problem.

Note that only the copy-and-paste of g_ioport will have no name conflict.

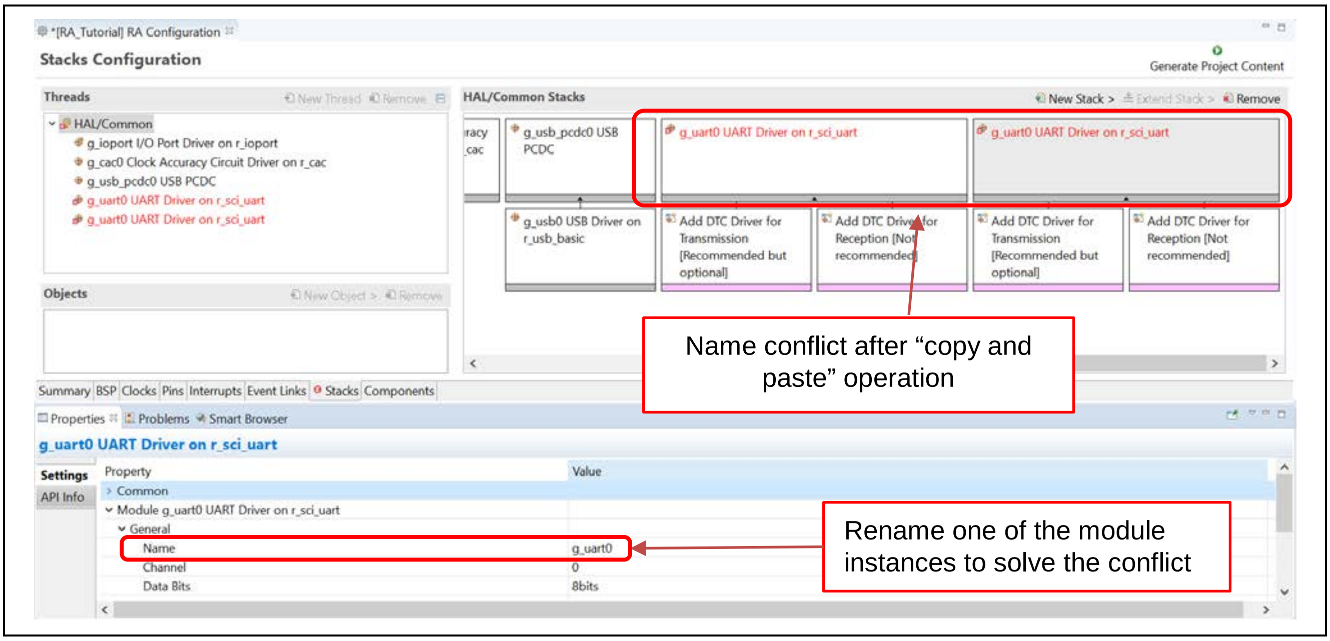

Figure 75. Module Instance Name Conflict

A module or a module stack can also be added by performing the export and import operation in the **Stacks Configuration** page. Right-click on a module and select **Export…** to export the configuration of the module to an XML file. Right-click in the stack pane of the same or a different thread in the same project and select **Import…** to import the configuration from the exported XML file. The name conflict can be solved by renaming one of the module instances.

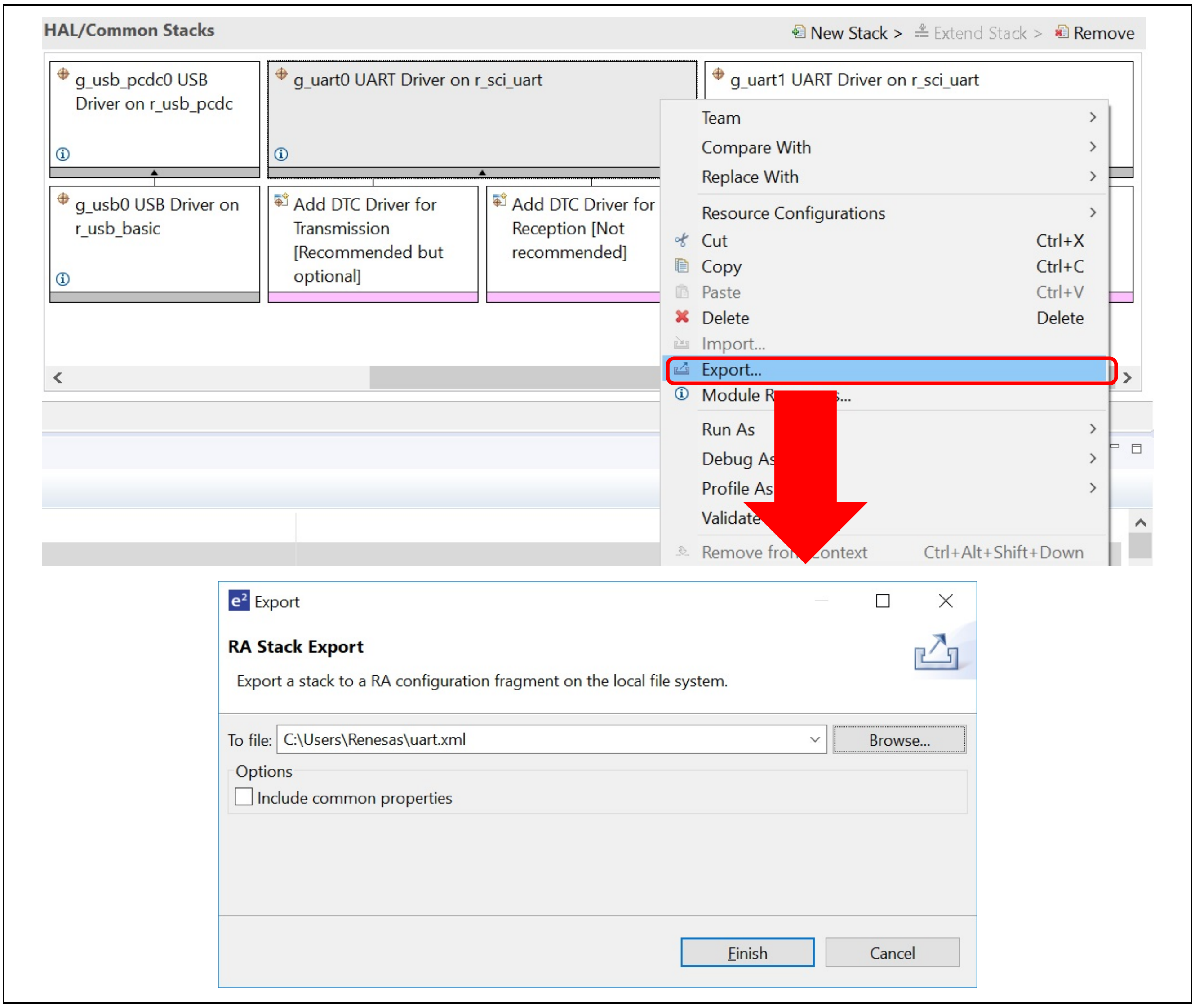

Figure 76. Export the RA Stack

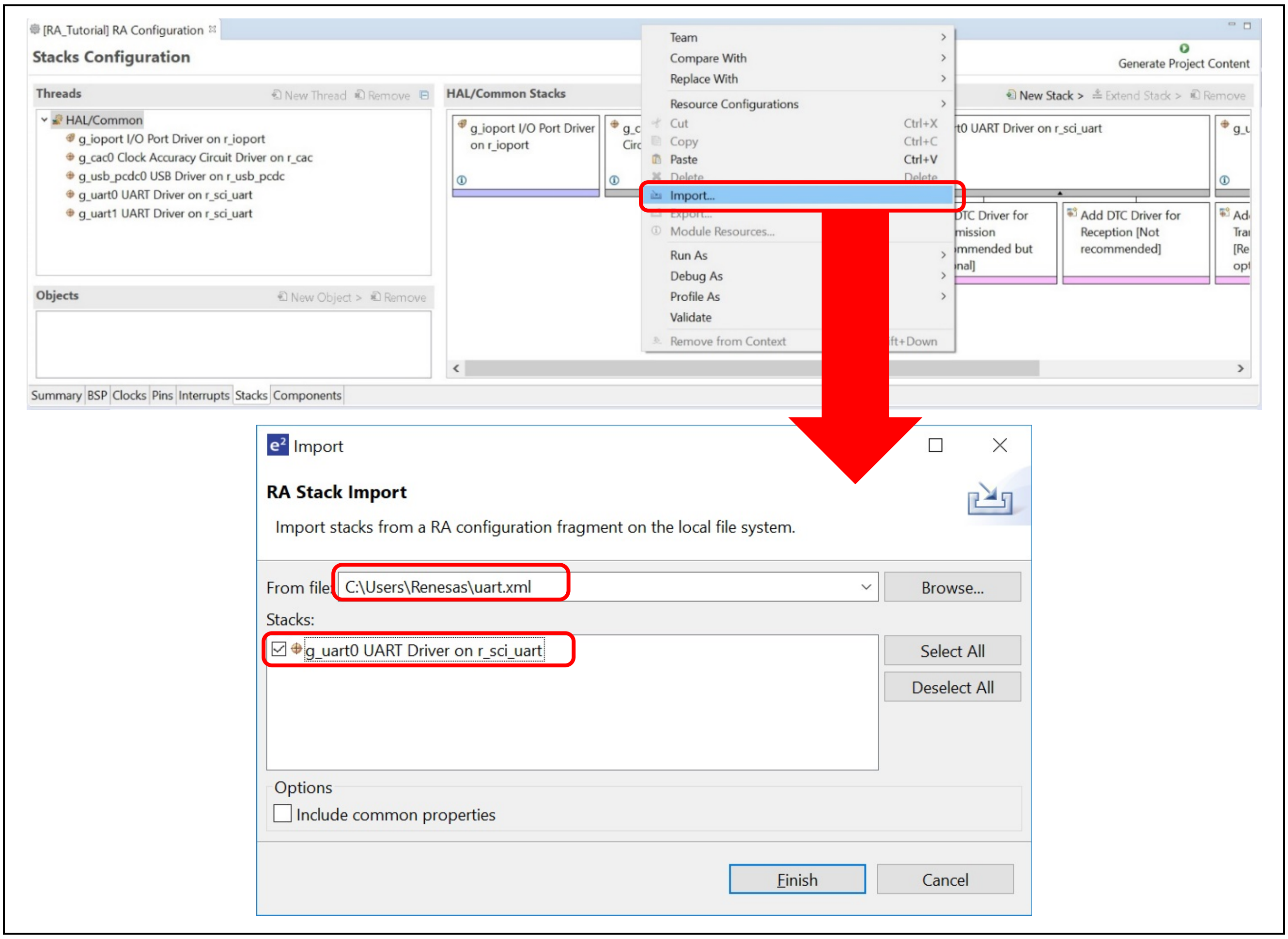

Figure 77. Import the RA Stack

3.5.6 Components Configuration Page

The **Components Configuration** page enables the individual modules required by the application to be included or excluded.

Modules common to all RA projects are preselected (for example **HAL Drivers** → **all** → **r_cac**).

All modules that are necessary for the drivers selected in the **Stacks** page are included automatically. You can include or exclude additional modules by checking the box next to the required component.

Note: The primary way of adding modules to an application is by using the **Stacks** page. The **Components** page is primarily used as a list of components available in the installed FSPs.

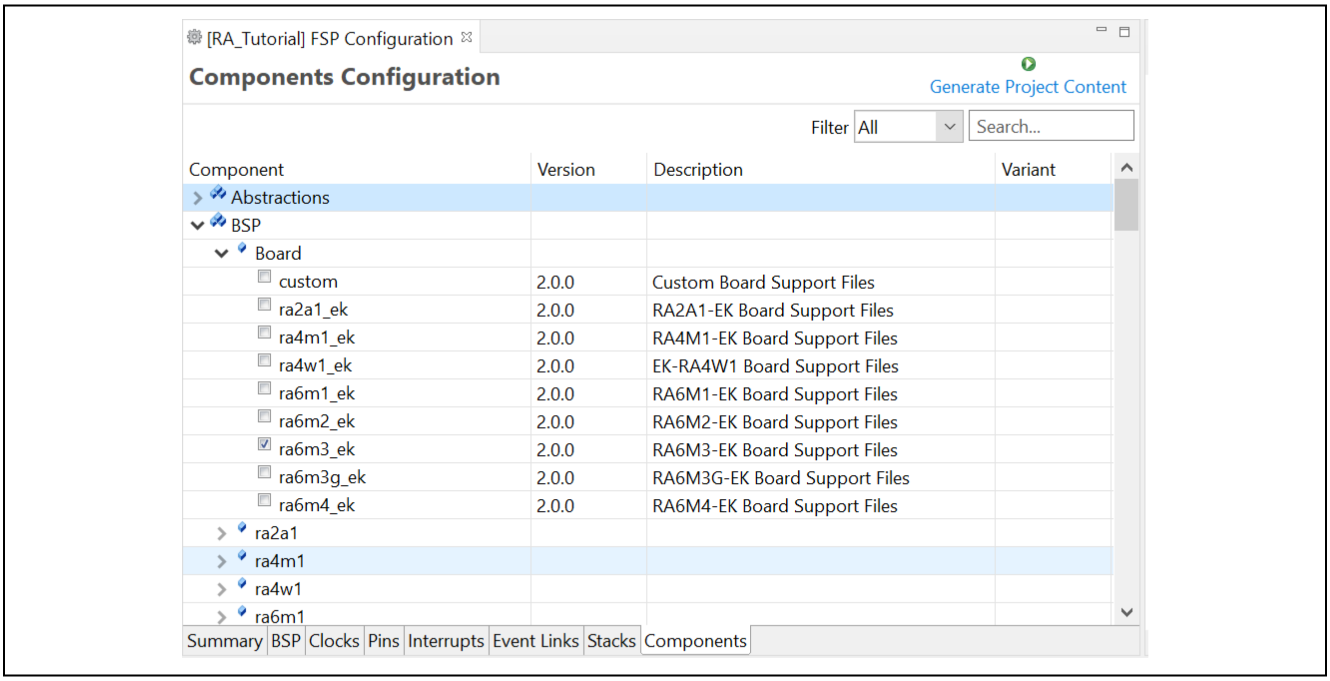

Figure 78. RA Project Configuration – Components Configuration

3.5.7 Interrupt Configuration Page

The **Interrupt** page allows the management of Events (interrupts) and ISRs (Interrupt Service Routines) for use with the RA interrupt framework.

The **Interrupt** page consists of 2 panes:

1. The **User Events** pane shows a list of events that have been created manually by user.
2. The **Allocations** pane shows a list of events that have been provided by instantiated RA modules in section 3.5.5.

In each pane, the **Event** column contains event names. The **ISR** column contains subscriber for the corresponding event in **Event** column.

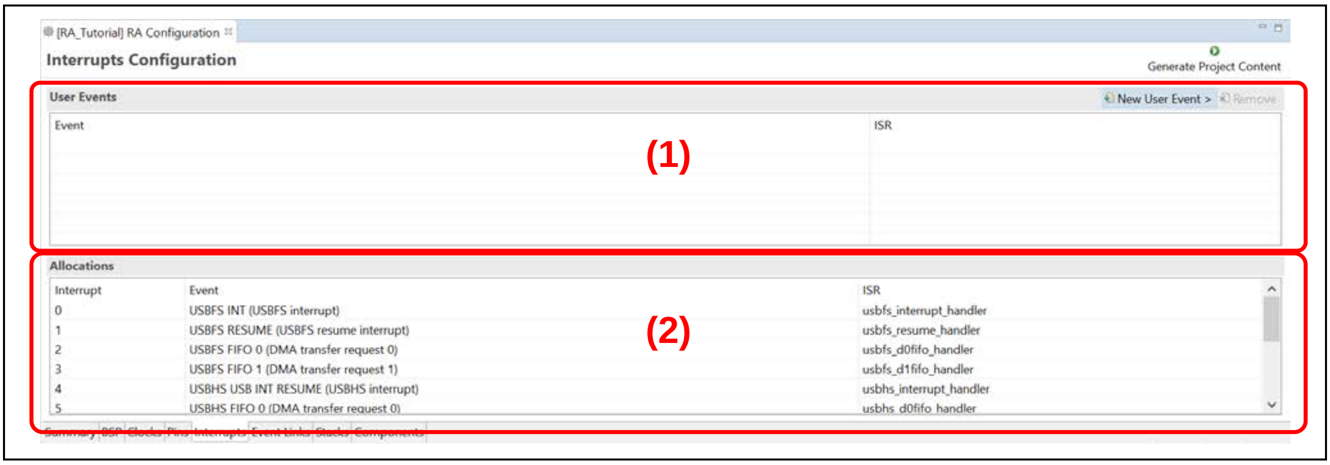

Figure 79. RA Project Configuration – Interrupt Page

A user event and its ISR can be created manually by clicking on the button **New User Event**, then selecting an event to create.

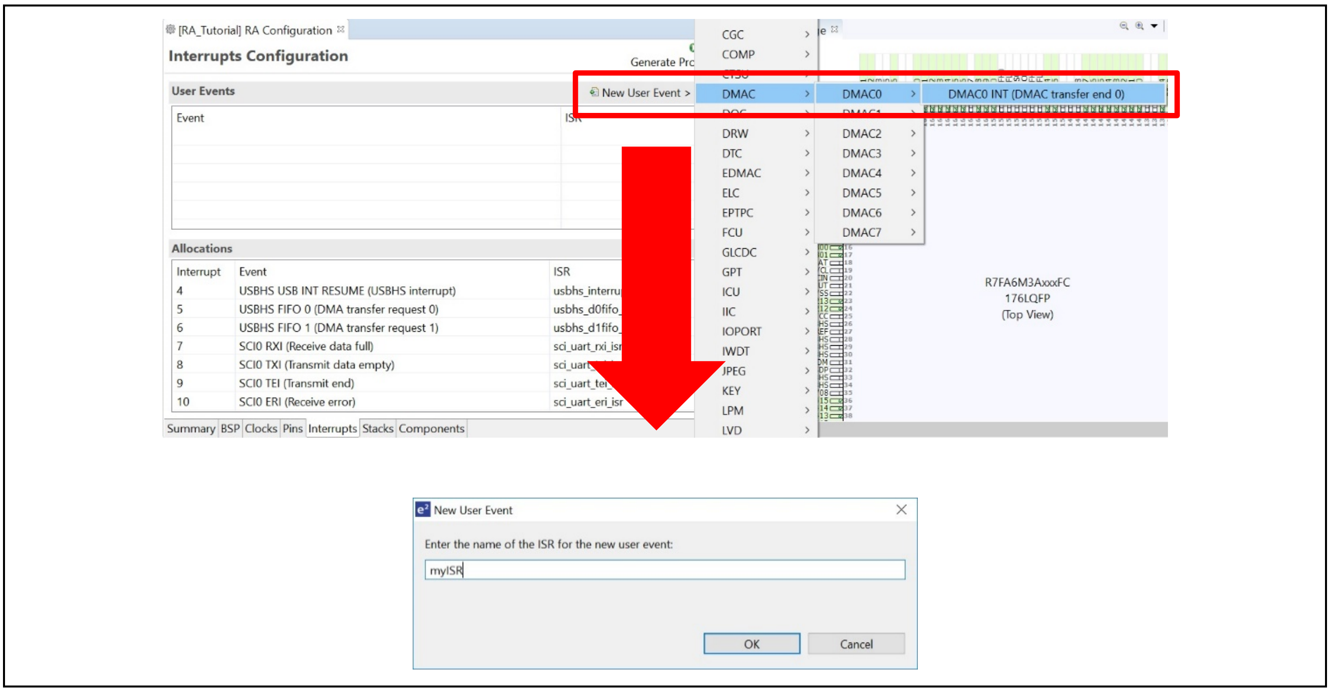

Figure 80. Interrupt Page – Adding A New User Event

The newly created event will be displayed in the **User Events** pane.

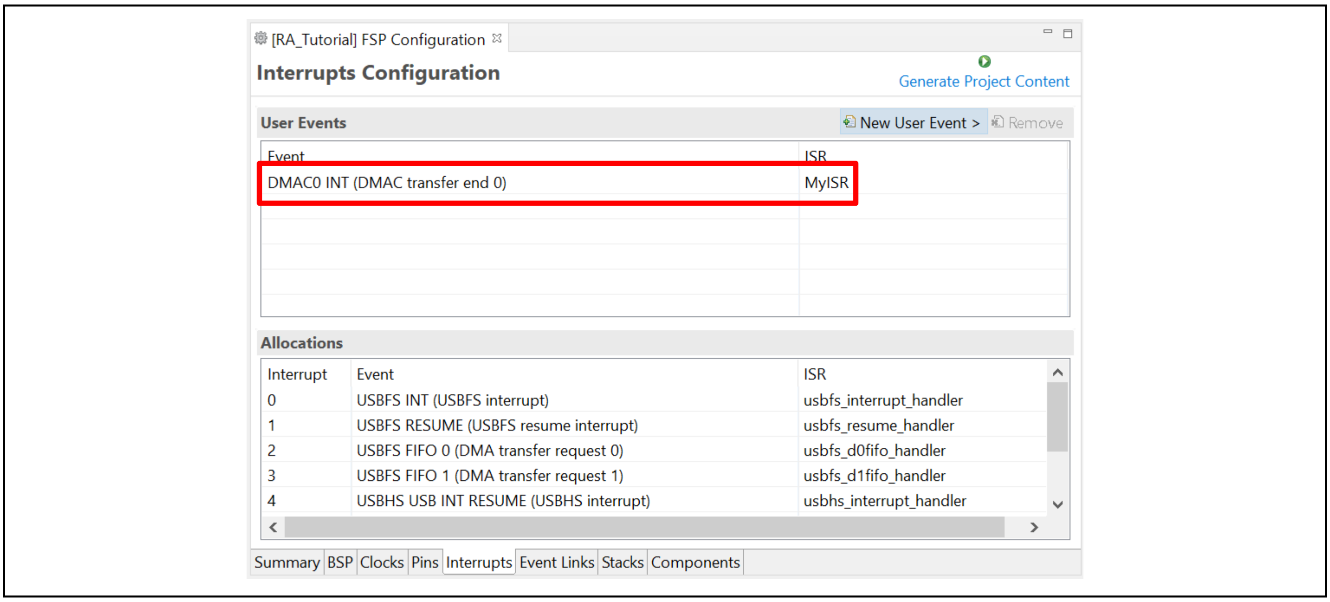

Figure 81. Interrupt Page – Generate Source Code

To remove a user event, select the event and click the button in the **User Events** pane (events added by instantiated RA modules in the **Allocations** pane cannot be removed).

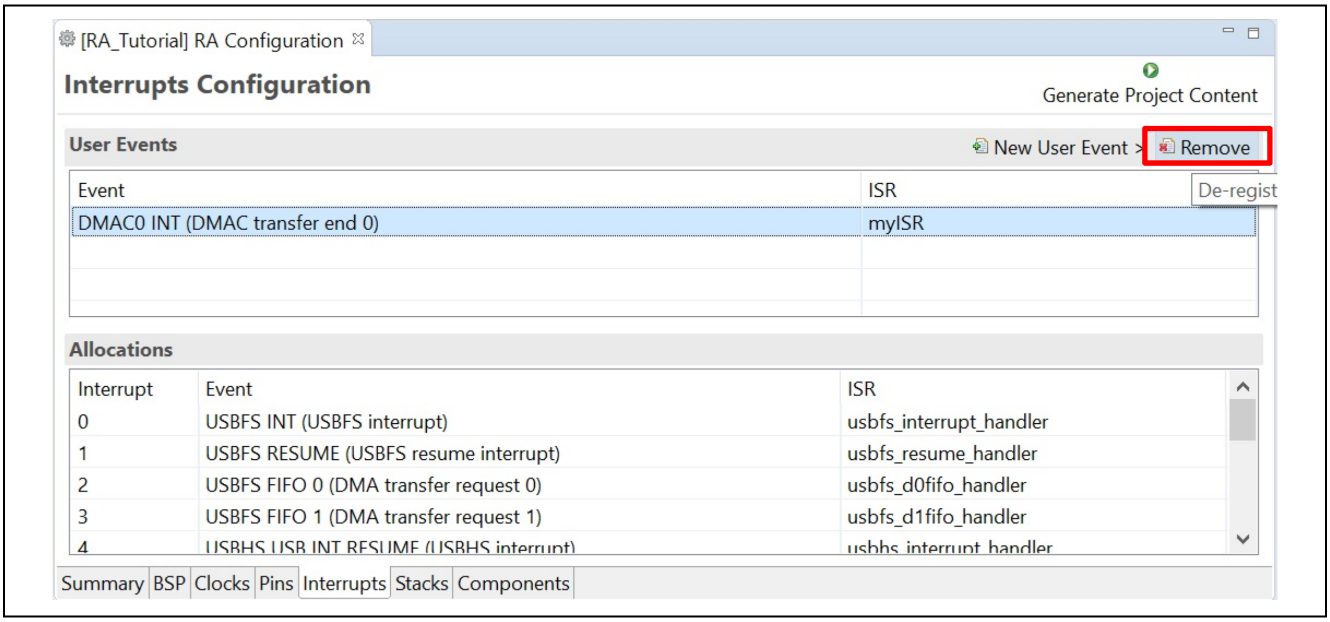

Figure 82. Remove User Events

3.5.8 Event Links Configuration Page

The **Event Links** page allows the user to specify how non-FSP drivers within an RA project make use of the Event Link Controller (ELC). The UI allows the user to declare that such a driver might produce a set of ELC events or consume a set of ELC events via a set of peripheral functions.

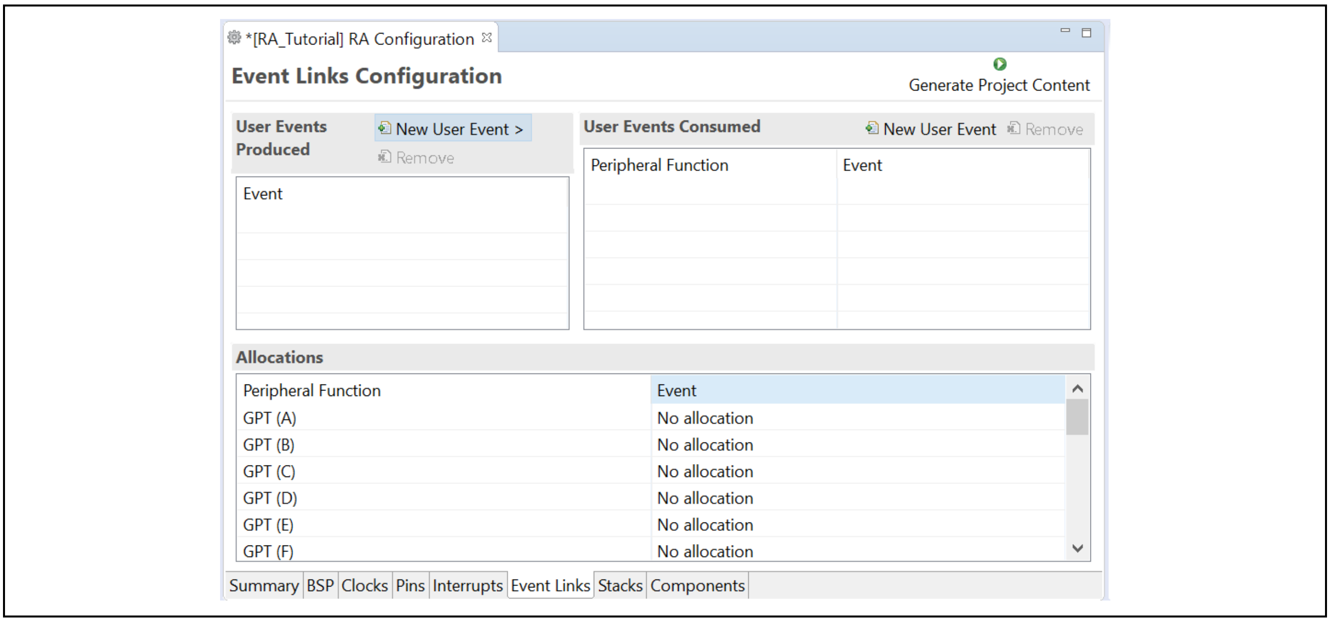

Figure 83. Event Links Page

To declare a user-produced event:

1. Select the **New User Event** button on the **User Events Produced** table
2. A cascading menu appears containing all the ELC events supported by the selected RA device
3. The **User Events Produced** list will be updated with the event selected from the menu

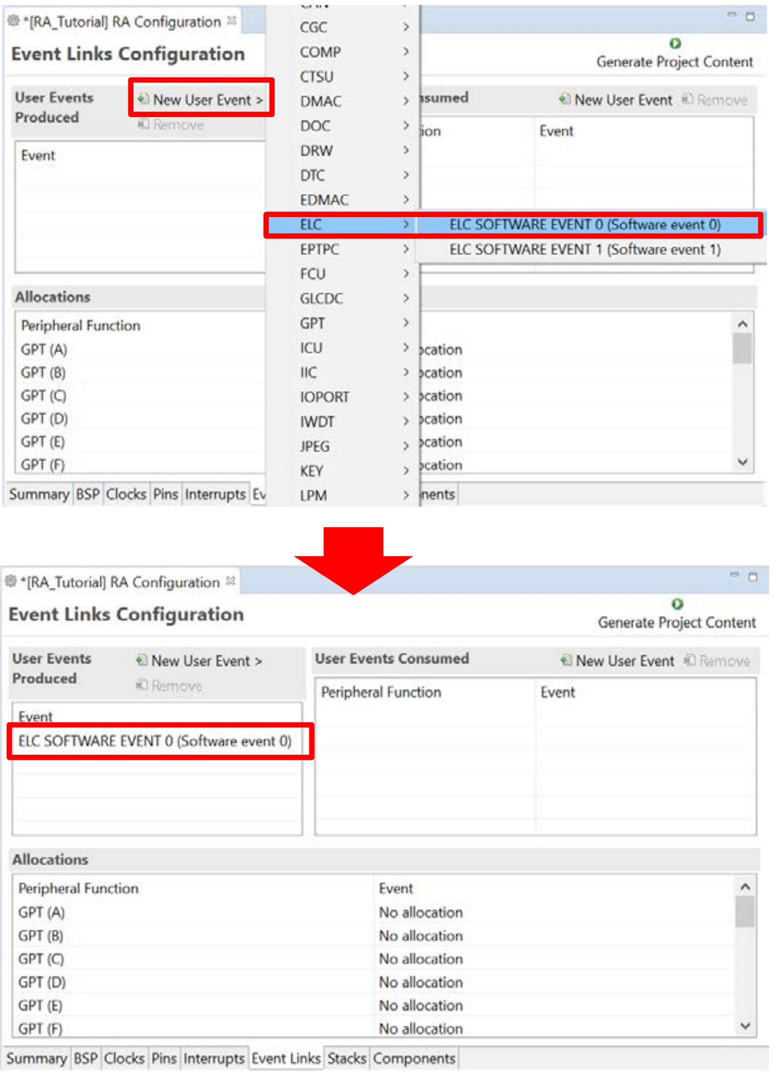

Figure 84. Declare User-Produced Event

To declare a user-consumed event:

1. Select the **New User Event** button on the **User Events Consumed** table
2. A **New User Event** dialog should open with a combo box containing the available peripheral functions and available ELC events. Select a **Peripheral Function** and **Event**, then click **OK**
3. The selected event will be allocated to the selected peripheral function within the RA configuration and should be visible within the **User Events Consumed** list and the **Allocations** list on the **Event Links** page.

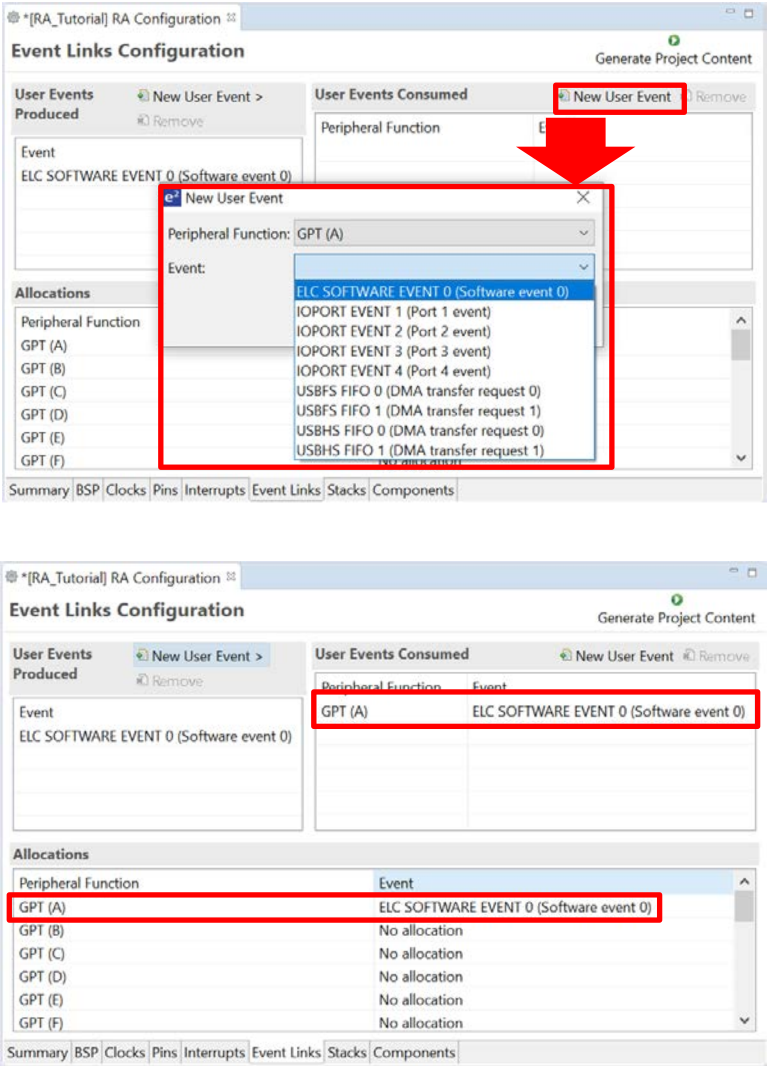

Figure 85. Declare A User-Consumed Event

3.6 Editor Hover

e² studio supports hover in the text editor. This function can be enabled/disabled via **Window** → **Preferences** → **C/C++** → **Editor** → **Hovers**.

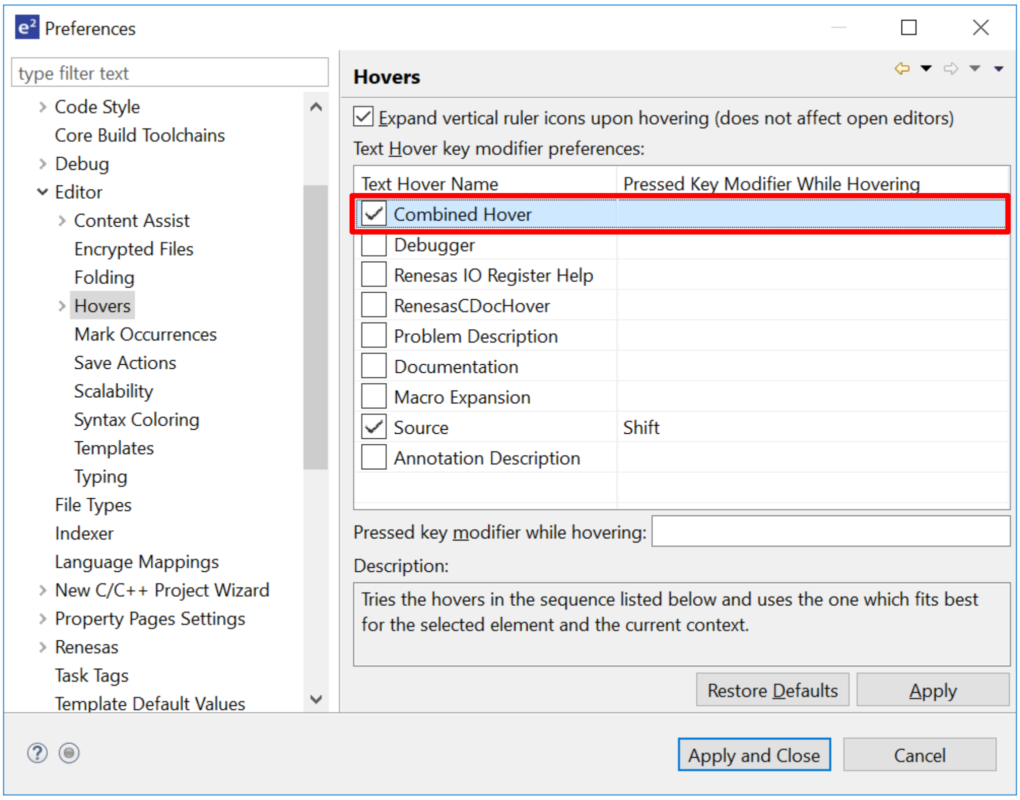

Figure 86. Hover Settings

To enable hover, check **Combined Hover**. To disable, uncheck it. This function is enabled by default.

Hover function allows user to view detailed information about any identifiers in the source code: hover the mouse over an identifier and check the pop-up.

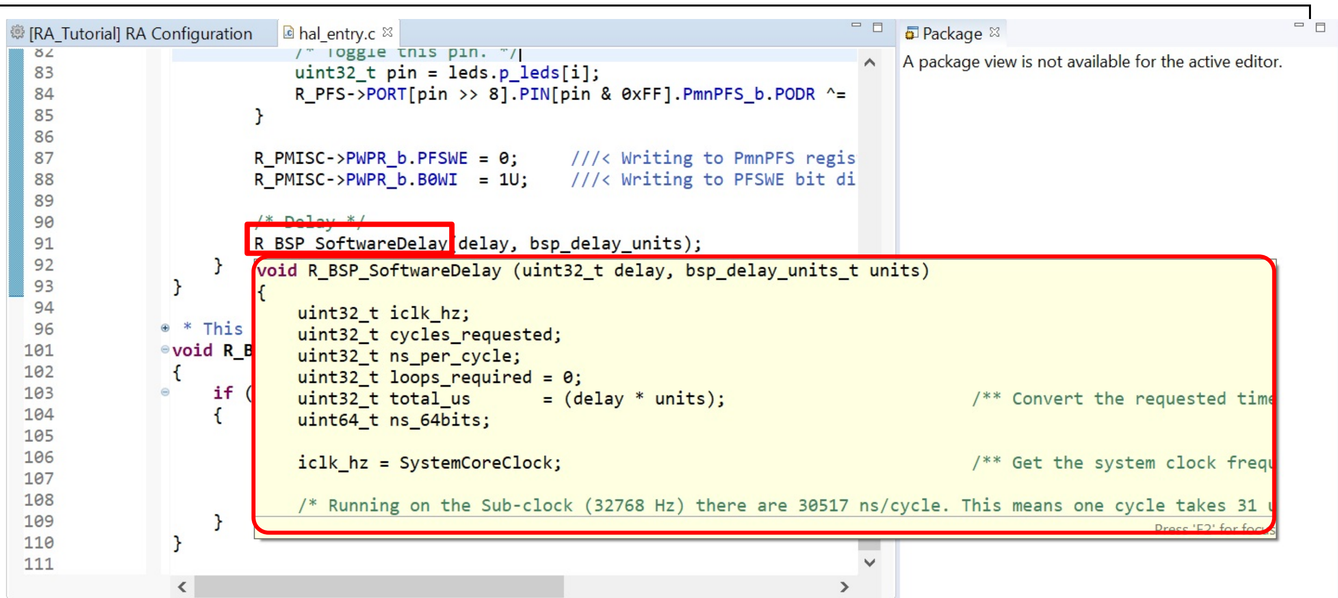

Figure 87. Information from Hover Function

4. Building

This section describes the build configurations and key build features in e² studio.

4.1 Build Configurations

The default build option is generated when a project is created. It can usually be used to build the project.

However, if changing build options is necessary (for example, toolchain version or optimization options), please follow the following steps before building the project.

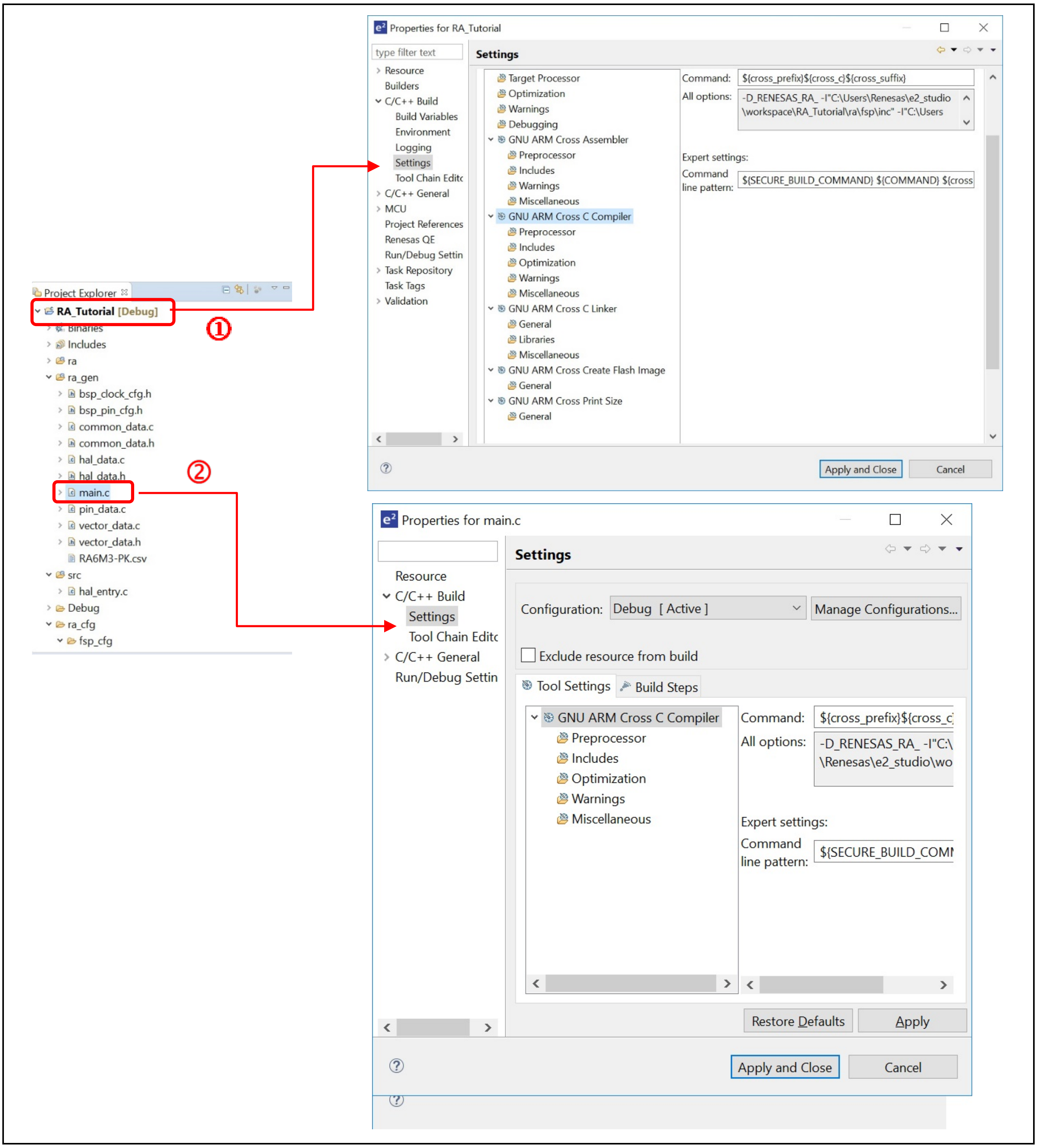

Figure 88. Build – Properties for RA Project And main.c Source File

Build options can be accessed in the properties window of a project or a source file.

1. ① Set the focus at the project name or ② set the focus at the source file name.
2. Right-click to select **Properties** or use shortcut keys **Alt** + **Enter** to open the properties dialog.
3. Click **C/C++ Build** → **Settings** option to view or edit the configuration settings.

The **Properties** window is supported at project and source level. The **Properties** window for projects supports more configurations which apply across all the files within the same project.

4.2 Building a Sample Project

Follow the steps below to build the project.

1. In **Project Explorer**, click the RA project to bring it into focus.
2. Click **Project** → **Build Project** or the  icon to build this project.
3. Confirm that there are no errors after build is finished.

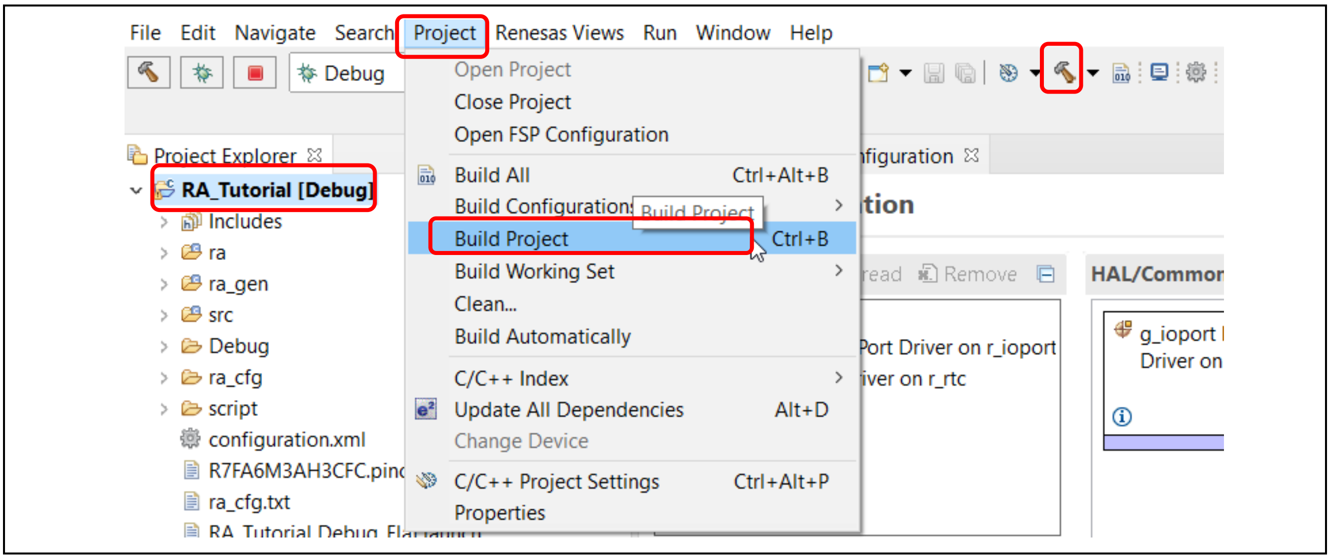

Figure 89. Build – Building A Sample Project

4.3 Saving the Build Settings Report

Project build settings in e² studio IDE can be saved to a file using the **Project Reporter** feature.

1. Right-click in the **Project Explorer** view to pop up the context menu.
2. Select **Save build settings report** to save the build settings report.

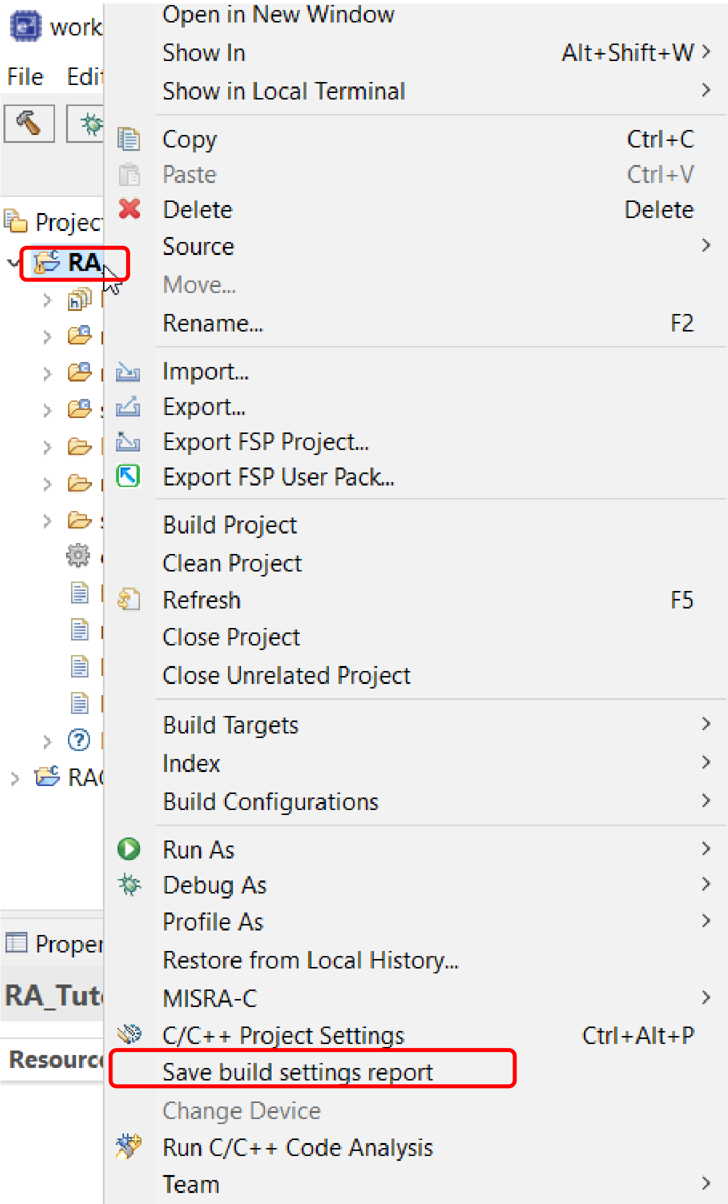

Figure 90. Build – Saving the Build Settings Report

5. Debugging

This section describes how to use the debug configuration and key debugging features for e² studio. Figure 91 refers to the RA project built in Section 4.2 Building a Sample Project and based on the following hardware configuration: J-link ARM emulator and RA6M3 EK board.

Right-click on any perspective icon, select **Show Text** to show the name of each icon.

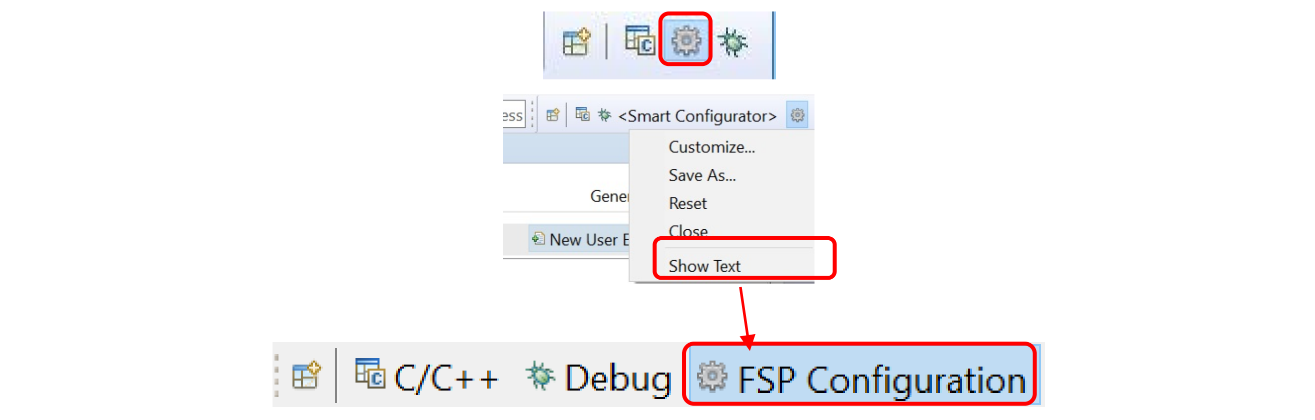

Figure 91 Debug – Switch To Debug Perspective

Open the RA project in e² studio and click **Debug** to switch to the **Debug** perspective.

As discussed earlier, a Perspective in Eclipse defines the layout of panes and views in the **Workbench** window. Each perspective consists of a combination of views, menus and toolbars that enable the user to perform a specific task. For instance,

- The **Debug** perspective has views that enable the user to debug the program
- The **RA Configuration** perspective together with `configuration.xml` in the editor window will open the RA configuration, as well as the **Package** and **Properties** views for project configuration settings
- The **C/C++** perspective has views that help the user to develop C/C++ programs.

If a user attempts to connect the debugger when not in the **Debug** perspective, e² studio will prompt the user to switch to the **Debug** perspective.

One or more perspectives can exist in a single Workbench setup. User can customize them or add new perspectives.

5.1 Changing an Existing Debug Configuration

A default debug configuration is automatically created the first time a specific RA project is built. An existing debug configuration can be changed as follows.

1. Click the project name in the **Project Explorer** view to set focus.
2. Click **Run** → **Debug Configurations…** or the  icon **(downward arrow)** → **Debug Configurations…** to open the **Debug Configurations** window.

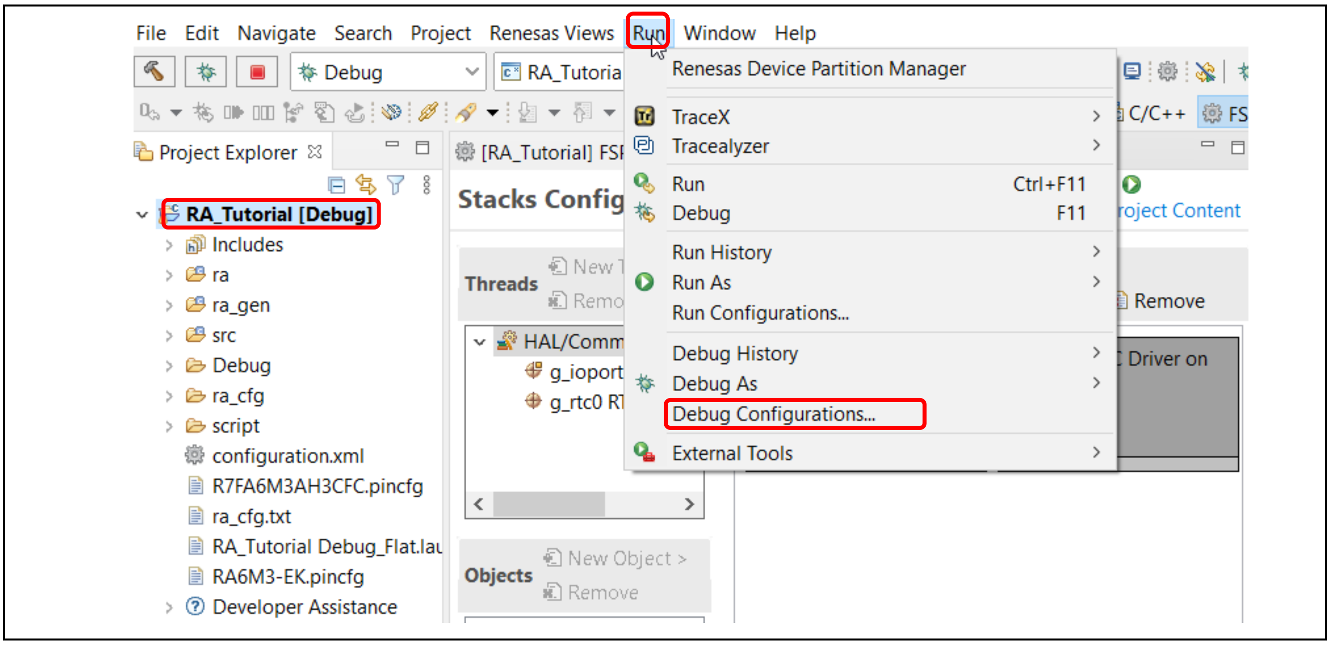

Figure 92. Debug – Opening the Debug Configurations Window

3. In the **Debug Configurations** windows, expand the **Renesas GDB Hardware Debugging** debug configuration and click on the existing debug configuration (for example, **RA_Tutorial Debug_Flat**).
4. Go to the **Main** tab and browse to add the load module (that is, **RA_Tutorial.elf**) located in the project build folder.

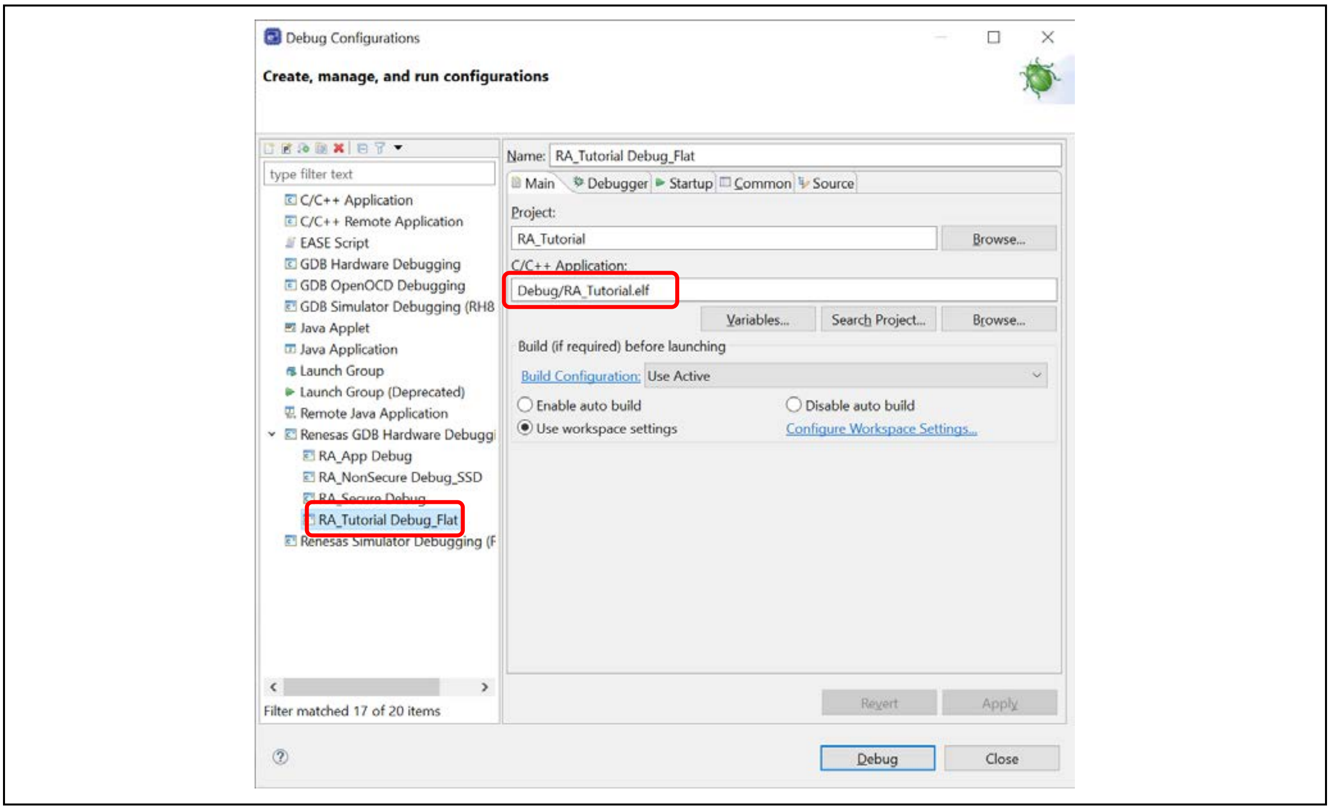

Figure 93. Debug – Selecting the Load Module

5. Switch to the **Debugger** tab, set J-Link ARM and R7FA6M3AH as the target device.
   Debug Hardware: J-link ARM
   Target Device: R7FA6M3AH

6. Click the **Apply** button to confirm the settings.
7. Click the **Debug** button to execute the debug launch configuration to connect to the J-Link and the RA board.

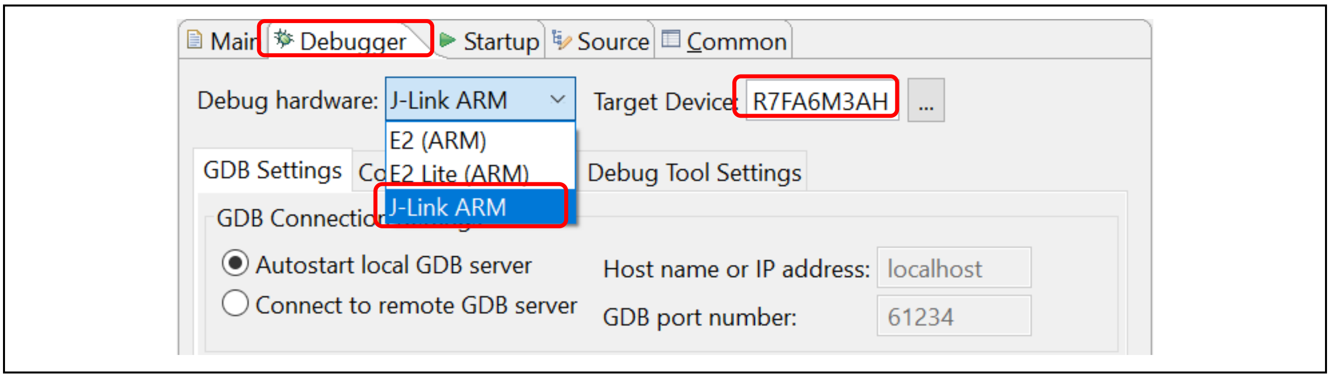

Figure 94. Debug – Changing The Connection Settings

8. For a successful connection, the **Debug** view shows the target debugging information in a tree hierarchy. The program entry point is set at `Reset_Handler()` in `startup.c`.

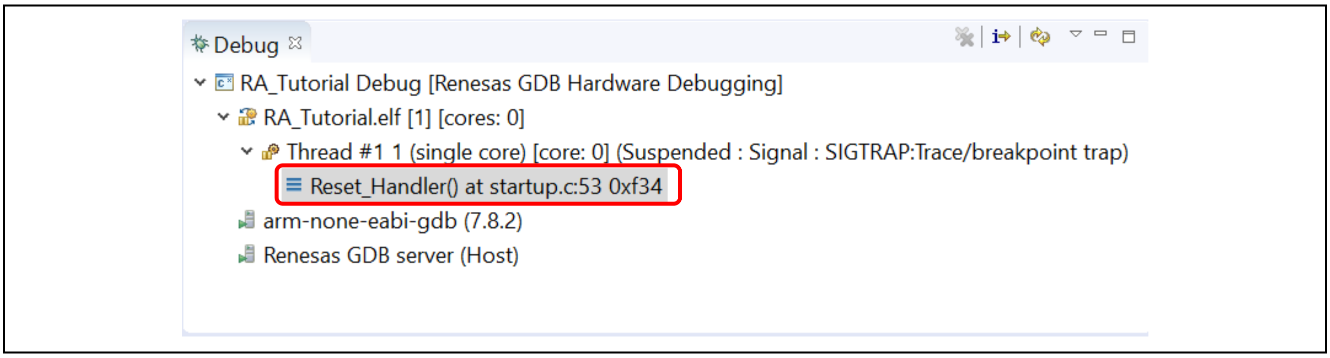

Figure 95. Debug – User Target Connection In The Debug View

5.2 Creating a New Debug Configurations

The simplest way to create a new debug configuration is by duplicating an existing one. It can be done by the following the steps below.

1. Open the **Debug Configuration** window (refer to Figure 92).
2. In the **Debug Configurations** window, select a debug configuration (for example, **RA_Tutorial Debug**) and click the  icon (which duplicates the currently selected launch configuration). A new debug launch configuration (for example, **RA_Tutorial Debug (1)**) is created.
3. The new debug configuration can be configured as described in section 5.1.

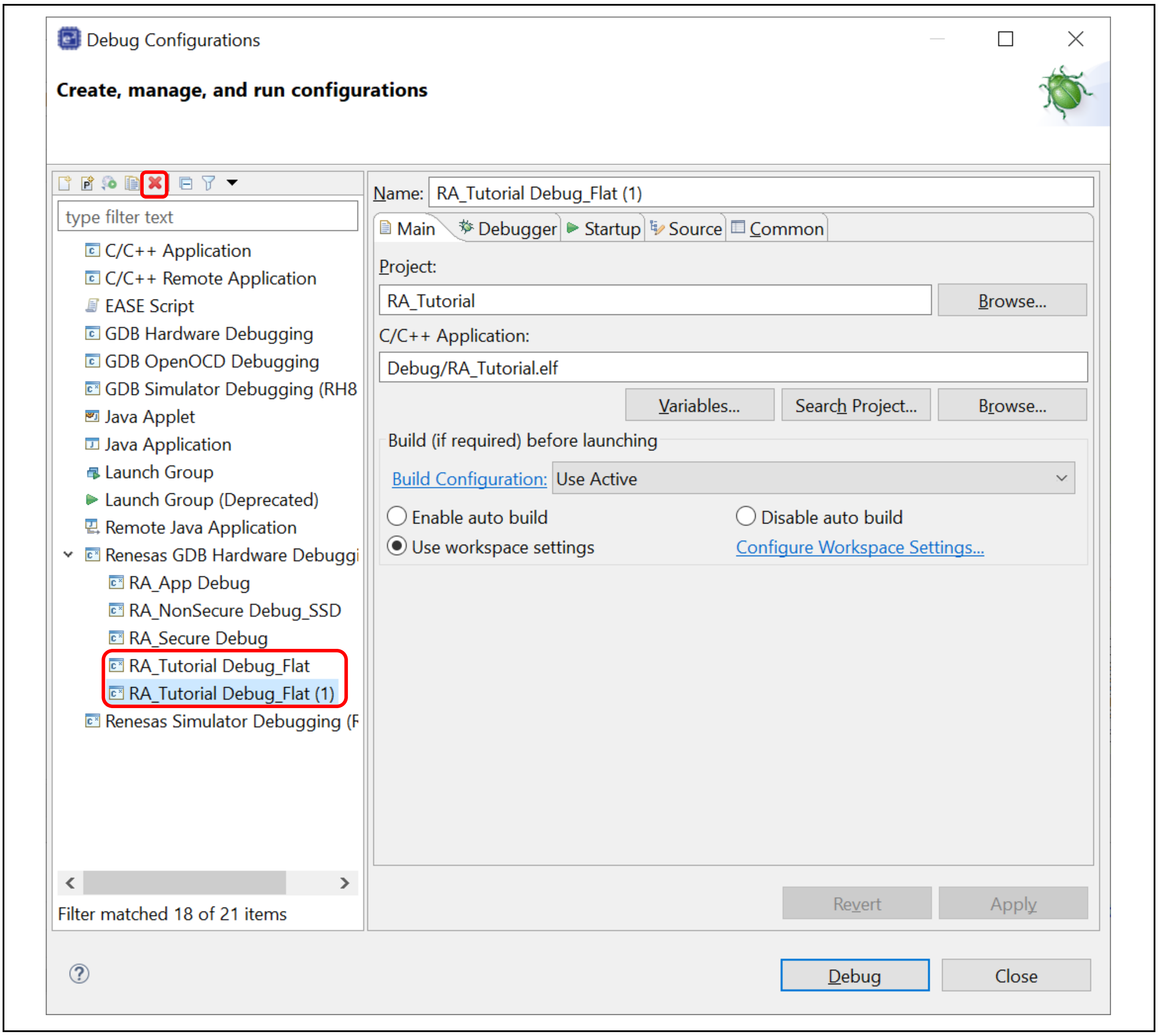

Figure 96. Debug – Duplicating A Selected Debug Launch Configuration

5.3 Basic Debugging Features

This section explains the typical Debug views supported in e² studio.

- Standard GDB Debug (supported by Eclipse IDE framework): Breakpoints, Expressions, Registers, Memory, Disassembly and Variables (MMU view is not supported in RA).
- Renesas Extension to Standard GDB Debug: IO Registers, Eventpoints, Trace and Fault Status.

The following are some useful toolbars in the **Debug** view:

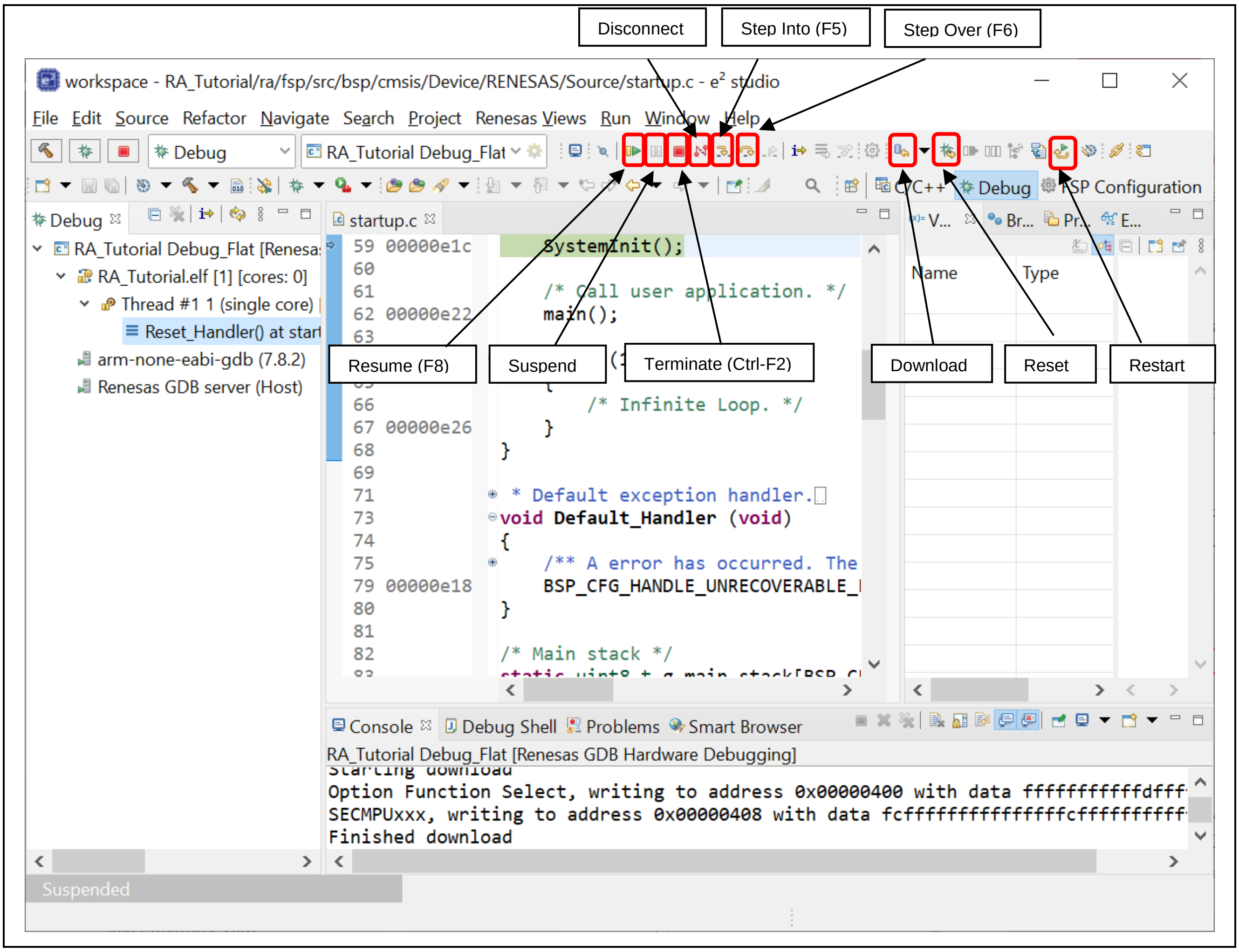

Figure 97. Debug – Useful Toolbars In Debug Views

Run the program by clicking the  button or pressing **F8**.

The program execution can be suspended by breakpoint or by clicking the  button. When the program execution is suspended, you can perform the following operations:

-  button or **F5** can be used for stepping into the next method call at the currently executing line of code.
-  button or **F6** can be used for stepping over the next method call (executing but without entering it) at the currently executing line of code.
-  button can be clicked again to resume program execution.

To stop the debugging process, the  button can be clicked to end the selected debug session and/or process or the  button can be clicked to disconnect the debugger from the selected process.

The other operations are as follows:

- The  button can be clicked to reset and run the program. It may stop at `main()` if the breakpoint is configured in the **Debug** configuration.
- The  button can be clicked to reset the program to its entry point at the PowerOn Reset.
- The  button is used for re-downloading the binary file to the target system.

5.3.1 Debug View

The **Debug** view is shows executed functions per thread.

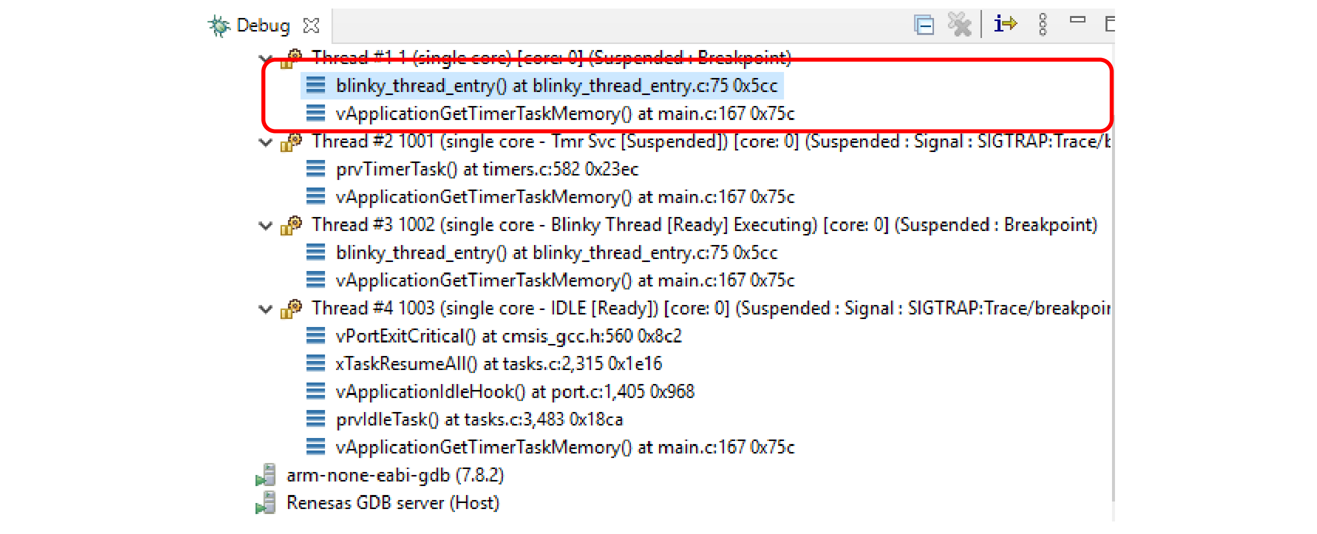

Figure 98. Debug – Debug View

Setting the Display of Executed Functions in the **Debug** view feature is set in the **Debug Configurations** dialog box.

1. Select the **Run > Debug Configurations** menu and open the **Debug Configurations** dialog box.
2. Select the **Debugger** tab and the **Debug Tool Settings** tab.
3. Set **RTOS Integration in Debug View** to **Yes**. If you select **No**, this feature is not available. This selection by default is set to **Yes**.

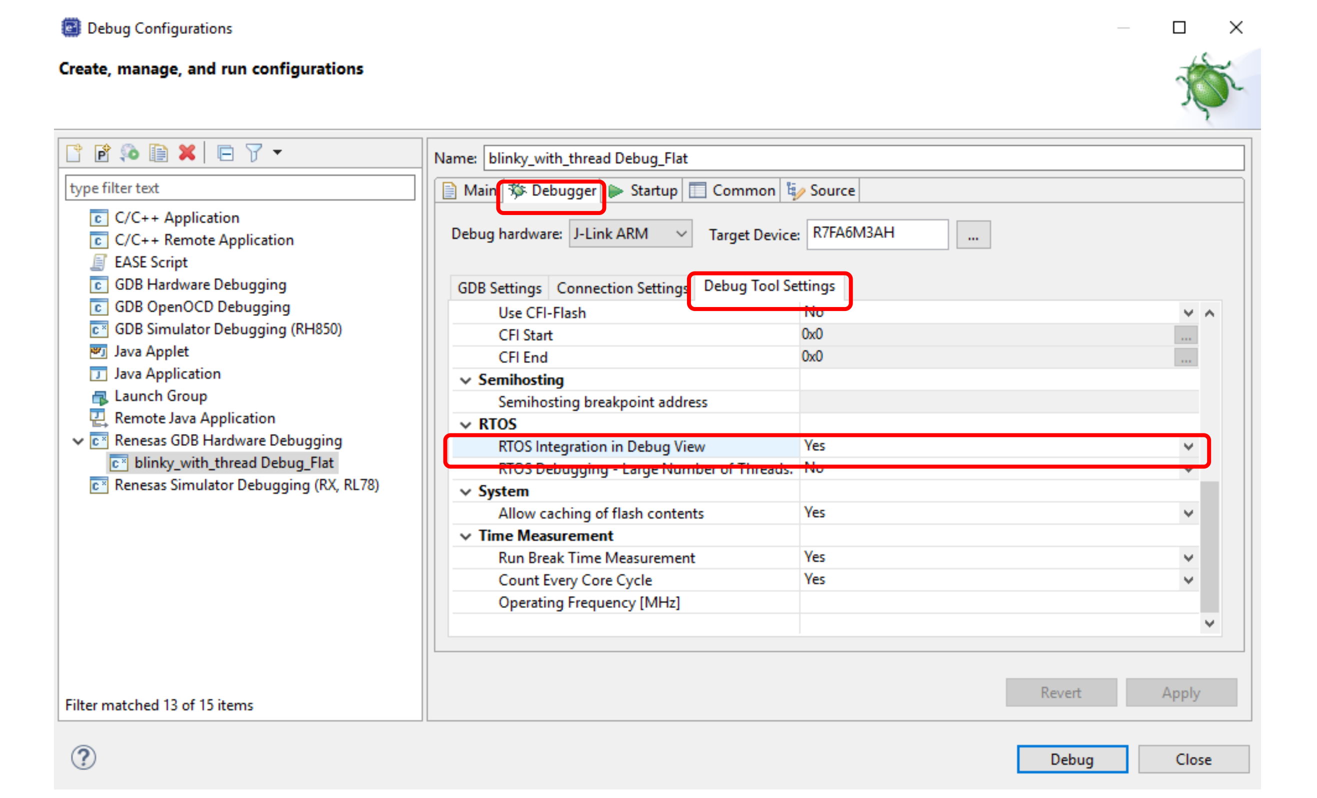

Figure 99. Setting – Debug View

5.3.2 Breakpoints View

The **Breakpoints** view stores the breakpoints that were set on executable lines of a program. If a breakpoint is enabled during debugging, the execution suspends before that line of code executes. e[2] studio allows software and hardware breakpoints to be set explicitly in the IDE. Any breakpoints added by double-clicking on the marker bar are by default hardware breakpoints. If the hardware resources are not there then the breakpoint setting will fail. If a hardware breakpoint setting fails, an error message will prompt the user to switch to a software breakpoint.

To select a hardware or software breakpoint:

1. Right-click on the marker bar to pop up the context menu. For a hardware breakpoint, select **Breakpoint Types** → **e[2] studio Breakpoint**. For a software breakpoint, select **Breakpoint Types** → **C/C++ Breakpoints**.

To set a breakpoint:

1. As an example, in `startup.c` at line 62, double-click on the marker bar located in the left margin of the **C/C++ Editor** pane to set a breakpoint. A dot (Hardware breakpoint) or (Software breakpoint) is displayed in the marker bar depending on the **Breakpoint Type** selected. **Breakpoint Type** is hardware breakpoint by default.
2. Alternatively, right-click at the marker bar to choose **Toggle Hardware Breakpoint** or **Toggle Software Breakpoint** to set a hardware breakpoint or a software breakpoint .
3. Click **Windows** → **Show View** → **Breakpoints** or icon (or use shortcut key **Alt + Shift + Q**, **B**) to open the **Breakpoints** view to view the corresponding breakpoints set. Breakpoints can be enabled and disabled in the **Breakpoints** view.

To disable breakpoints, users can choose to disable specific breakpoints or to skip all breakpoints:

1. To disable a specific breakpoint, right-click on the Software breakpoint or Hardware breakpoint located in the left margin of the C/C++ Editor pane and select **Disable Breakpoint**, or uncheck the related line in the Breakpoints view. A disabled breakpoint is displayed as a white dot (or).
2. To skip all breakpoints, click on the icon in the **Breakpoints** view. A blue dot with a backslash will appear in the editor pane as well as in the **Breakpoints** view.

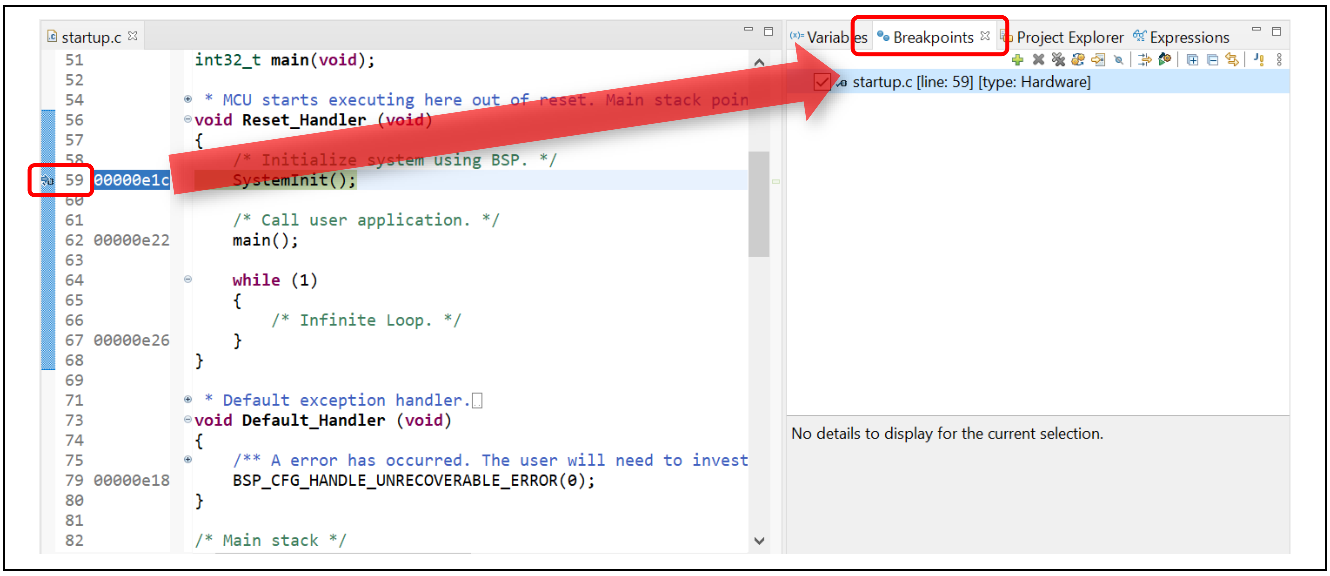

Figure 100. Debug – Breakpoints View

5.3.3 Expressions View

The **Expressions** view monitors the value of global variables, static variables, or local variables during debugging.

Follow the steps below to watch a variable:

1. Click **Windows** → **Show View** → **Expressions** or icon to open the **Expressions** view.
2. Drag and drop a variable (for example, `g_fsp_version` in `bsp_common.c`) to the **Expressions** view.
3. Alternatively, right-click the variable to select the **Add Watch Expression…** menu item to add it to the **Expressions** view.

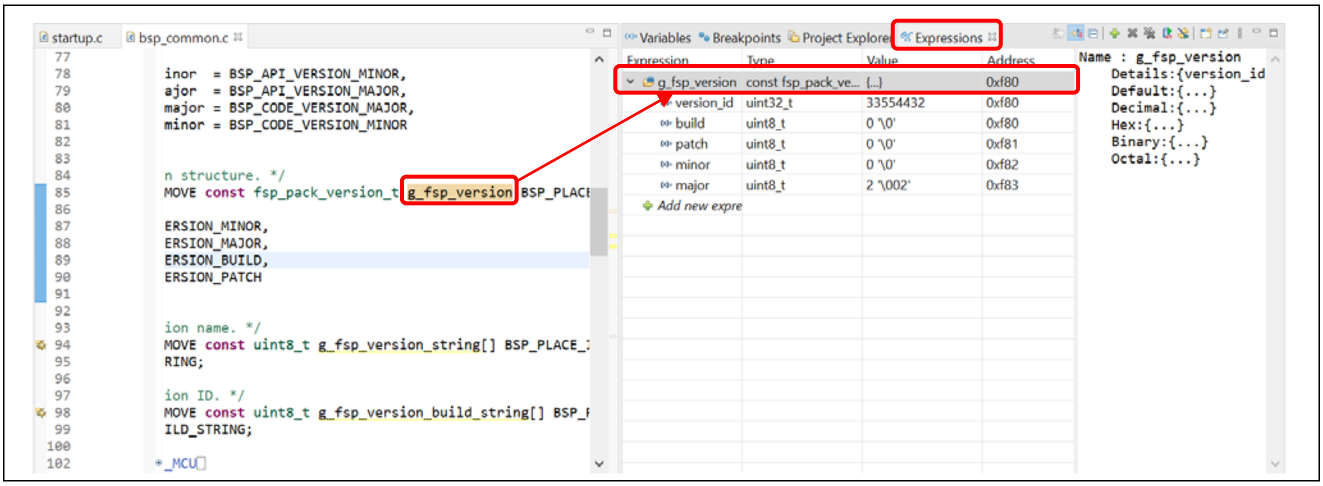

Figure 101. Debug - Expressions View

5.3.4 Registers View

The **Registers** view lists the information about the general registers in RA. Changed values are highlighted when the program stops.

1. Click **Windows** → **Show View** → **Registers** or icon to open the **Registers** view.
2. Click a register to view the values in different radix format.

Values that have been changed are highlighted (for example, in yellow) in the **Registers** view when the program stops.

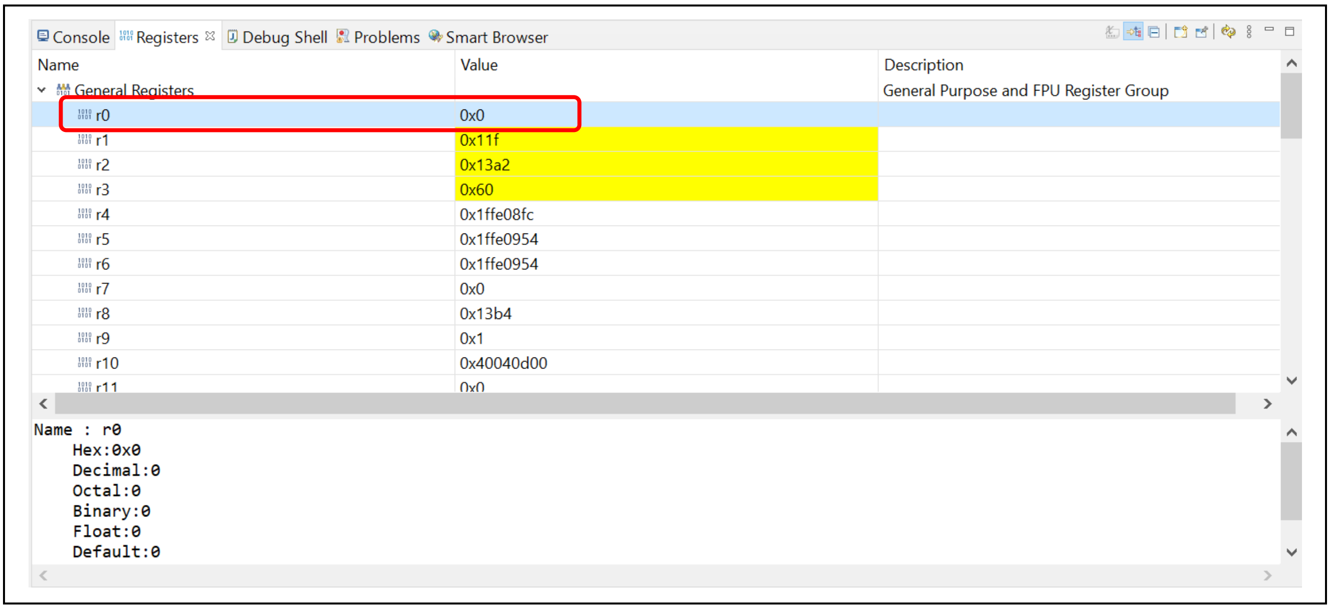

Figure 102. Debug – Registers View

5.3.5 Memory View

The **Memory** view allows users to view and edit the memory presented in “memory monitors”. Each monitor represents a section of memory specified by its location called “base address”. The memory data in each memory monitor can be presented in different “memory renderings”, which are the predefined data formats (for example, Hex integer, signed integer, unsigned integer, or ASCII image).

To view the memory of a variable (for example, `g_fsp_version_build_string`):

1. Click **Windows** → **Show View** → **Memory** to open the **Memory** view.
2. Click the icon to open the Monitor Memory dialog box. Enter the address of the variable `&g_fsp_version_build_string`.

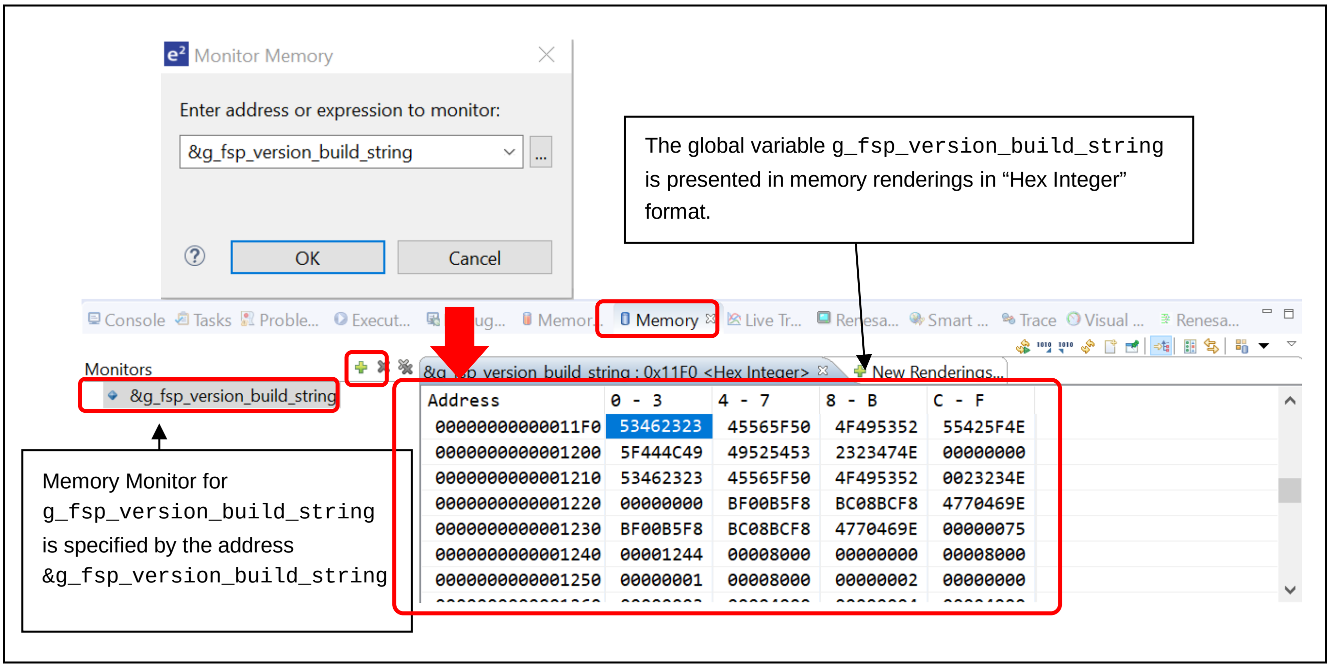

Figure 103. Debug – Memory View

To add a new rendering format (for example, ASCII) for the variable `g_fsp_version_build_string`:

Click the tab New Renderings… to select **ASCII** to add the rendering. This creates a new tab named **&g_fsp_version_build_string <ASCII>** next to the tab **&g_fsp_version_build_string <Hex Integer>**.

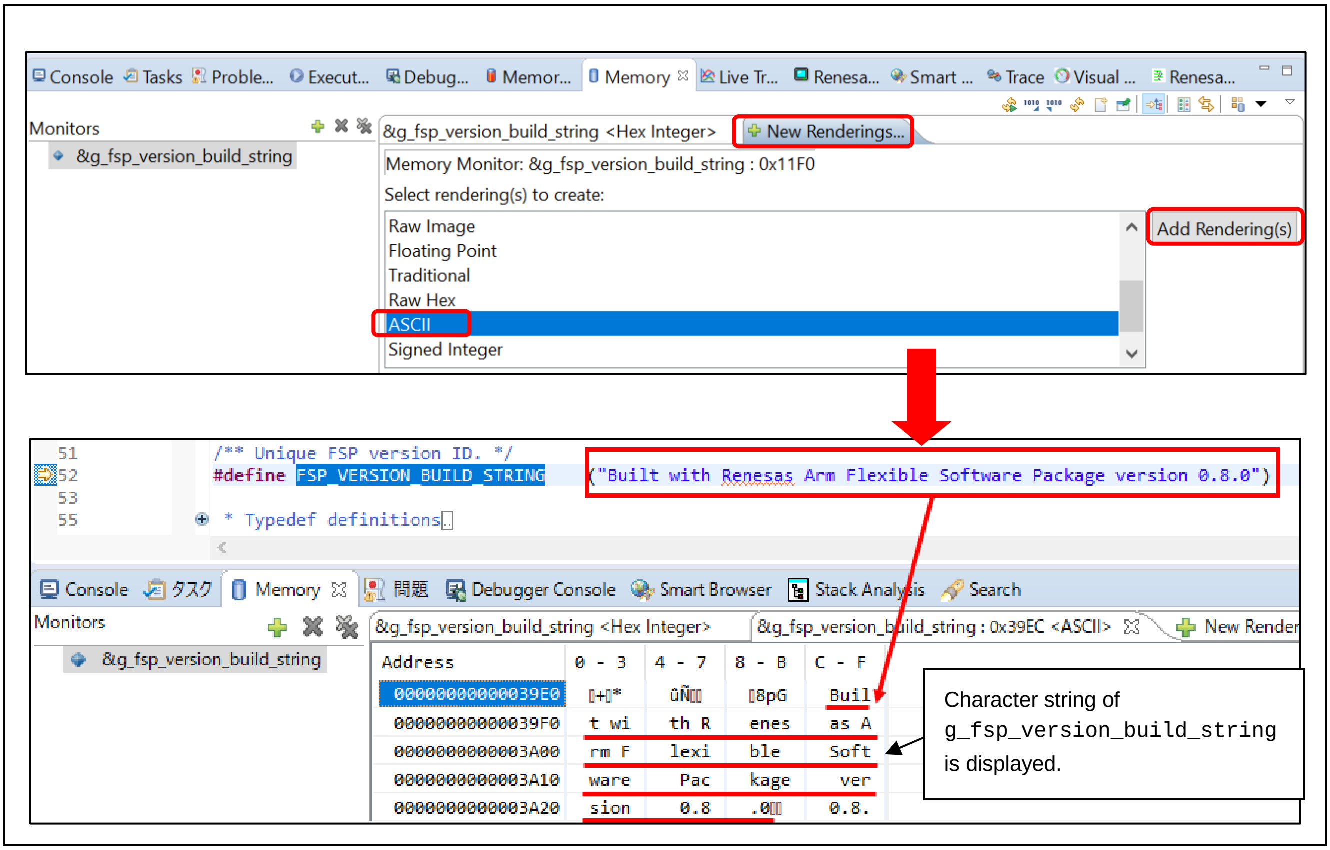

Figure 104. Debug – New Rendering In Memory View

5.3.6 Memory Usage view

Memory Usage will be used to get the information of (`*.map`) file or library list file (`,`) from a project. This will list out the total memory size, usage of ROM and RAM ratio and detailed information of sections, objects, symbols, module, vector, and cross reference used in project.

From version 7.3, e² studio supports the graphical view to show usage in the ROM and RAM memory areas.

To show the **Memory Usage** view, click menu **Window** → **Show View** → **Other…** → **C/C++**. In the **Show View** dialog, select **Memory Usage** and click **Open**.

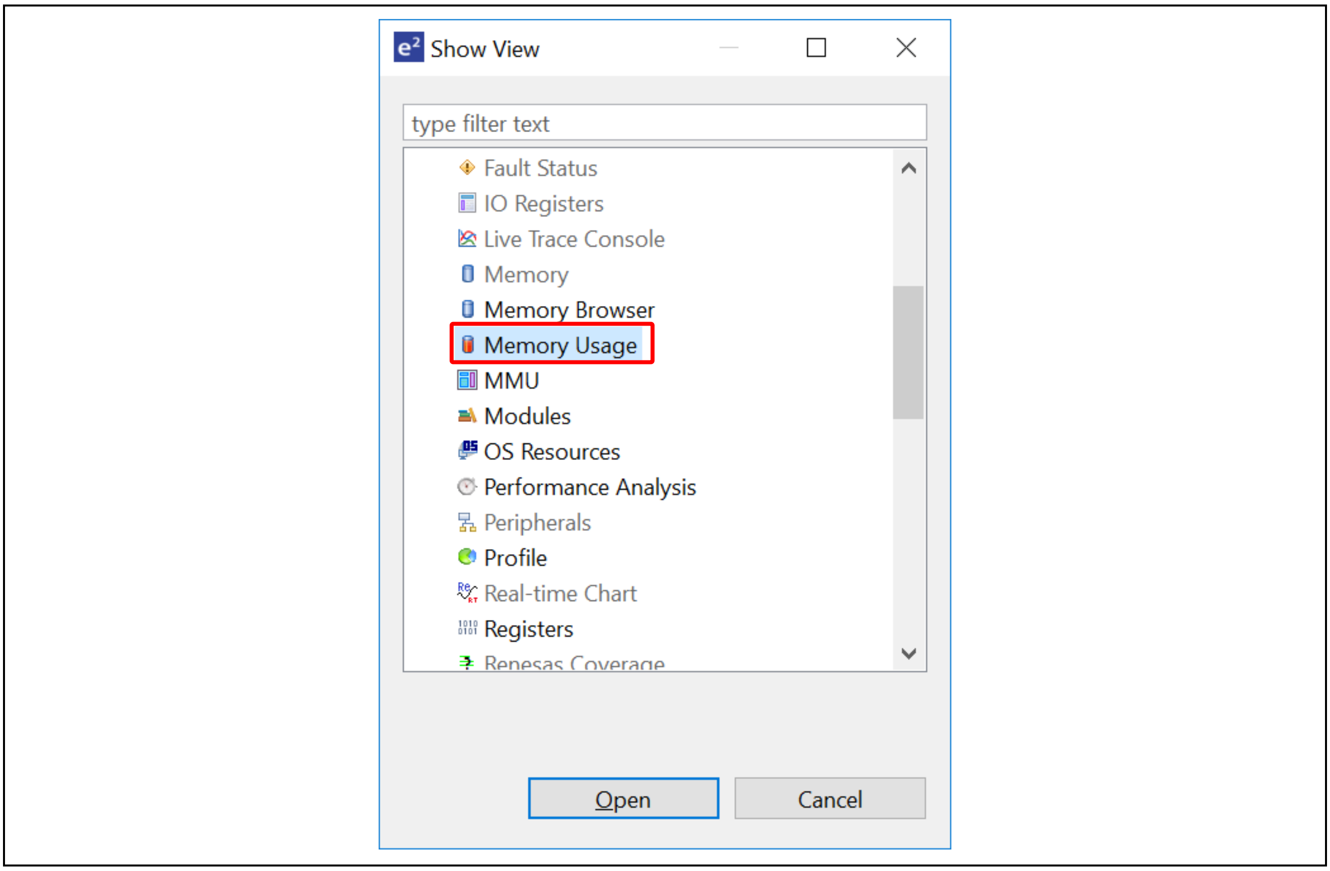

Figure 105. Show Memory Usage View

The **Memory Usage** view has three regions: (1) Group **Size** region, (2) **Memory Region Usage** region (**Device Memory Usage** region is not supported yet), (3) Detail table region.

Note: When the selected project does not contain the linker script file or there is no region defined in the linker script file, the **Memory Region Usage** region will display a warning message: "Linker script file is invalid".

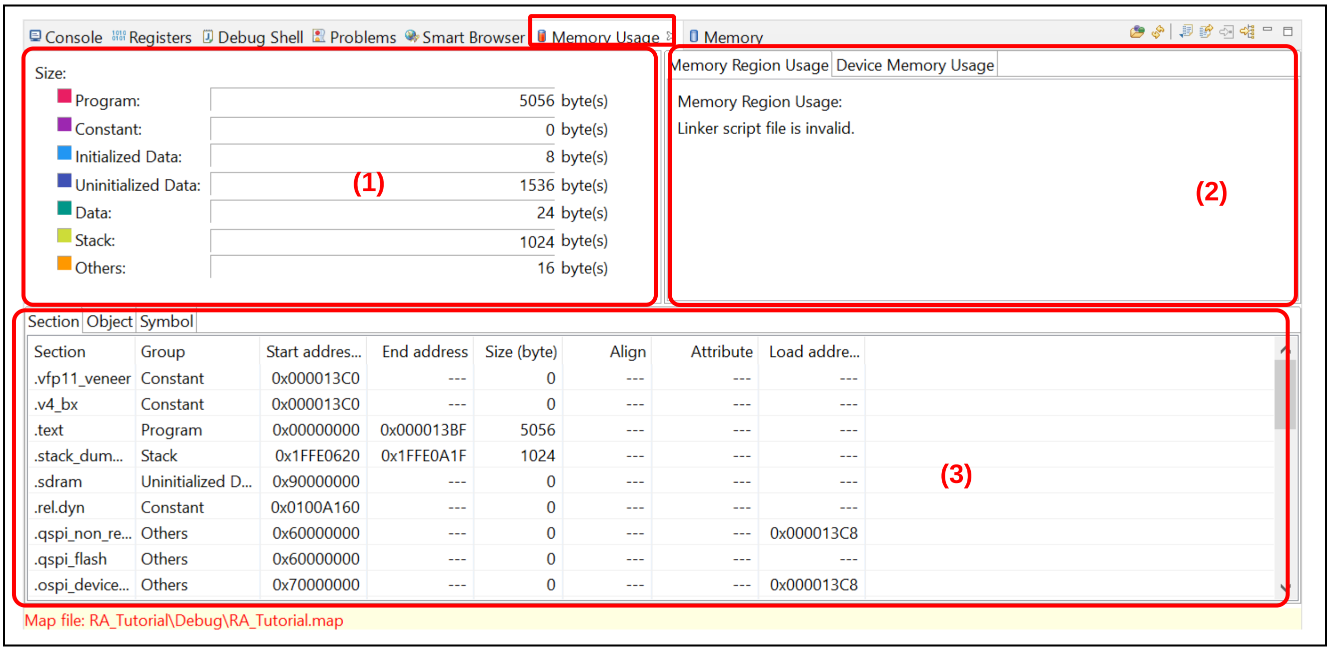

Figure 106. Regions of Memory Usage Views

Following operations are supported in the **Memory Usage** view:

: Choose a map or library list file for **Memory Usage** display.

: Refresh all information of **Memory Usage** view.

: Export data for all tabs in the **Detail** table region

: Open `*.map` or `*.lbp` file in Editor (there is no library list file in the RA library project).

: Open the Map file output page of selected project.

: Open the Section page of selected project.

5.3.7 Disassembly View

The **Disassembly** view shows the loaded program as assembler instructions mixed with the source code for comparison. The currently executing line is highlighted by an arrow marker in the view. In the **Disassembly** view, users can set breakpoints at assembler instructions, enable or disable these breakpoints, step through the disassembly instructions and even jump to a specific instruction in the program.

To view both C and assembly codes in a mixed mode:

1. Click **Windows** → **Show View** → **Disassembly** to open the **Disassembly** view.
2. Click the icon to enable synchronization between assembly source and the C source (active debug context).
3. In **Disassembly** view, right-click on the address column to select **Show Opcodes** and **Show Function Offsets**.

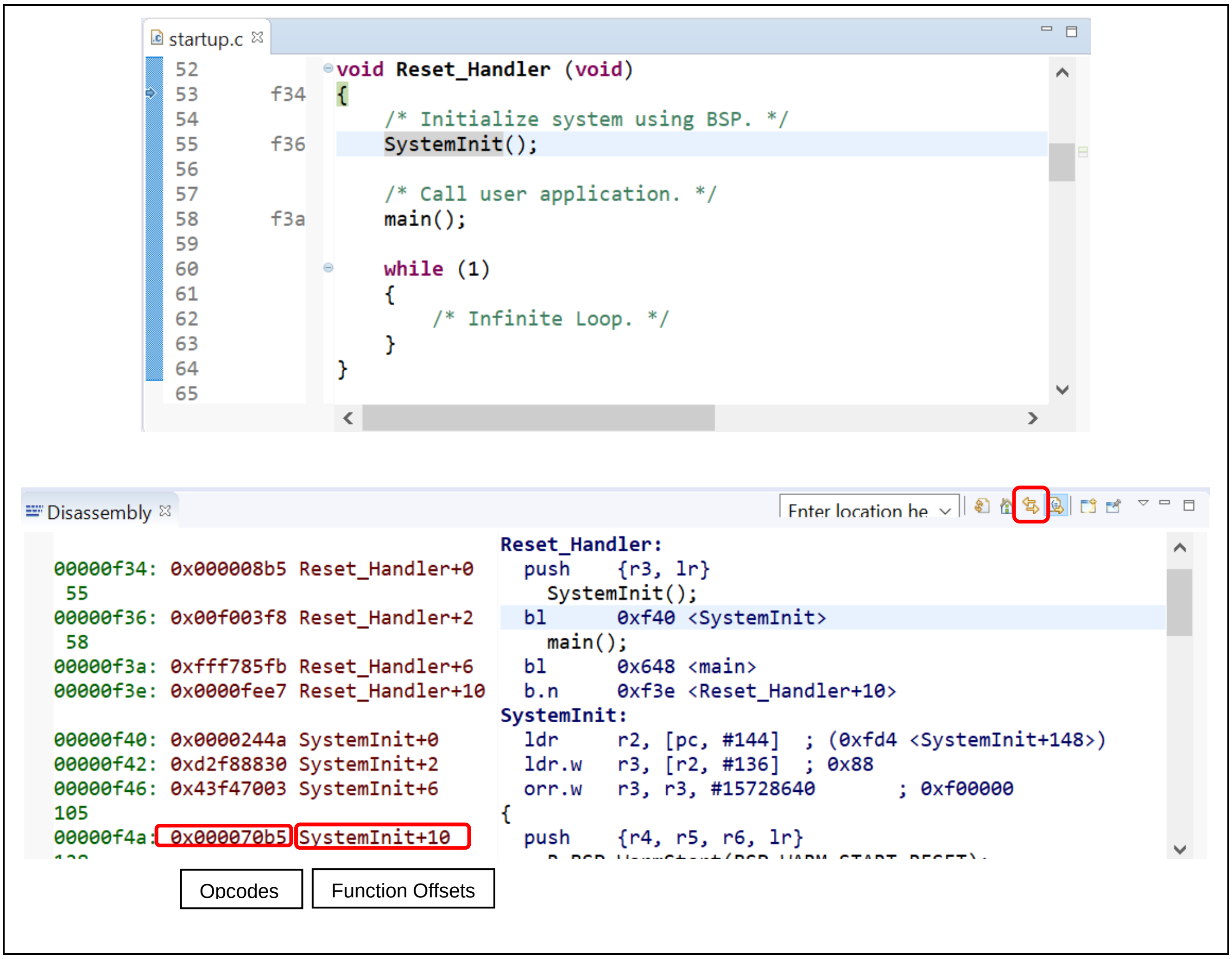

Figure 107. Debug – Disassembly View

5.3.8 Variables View

The **Variables** view displays all the valid local variables in the current program scope.

To observe a local variable (for example, `leds` for function `hal_entry ()`):

1. Click **Windows** → **Show View** → **Variables** to open the **Variables** view.
2. Step into the function `hal_entry ()` to view the local variable **timeout** value.

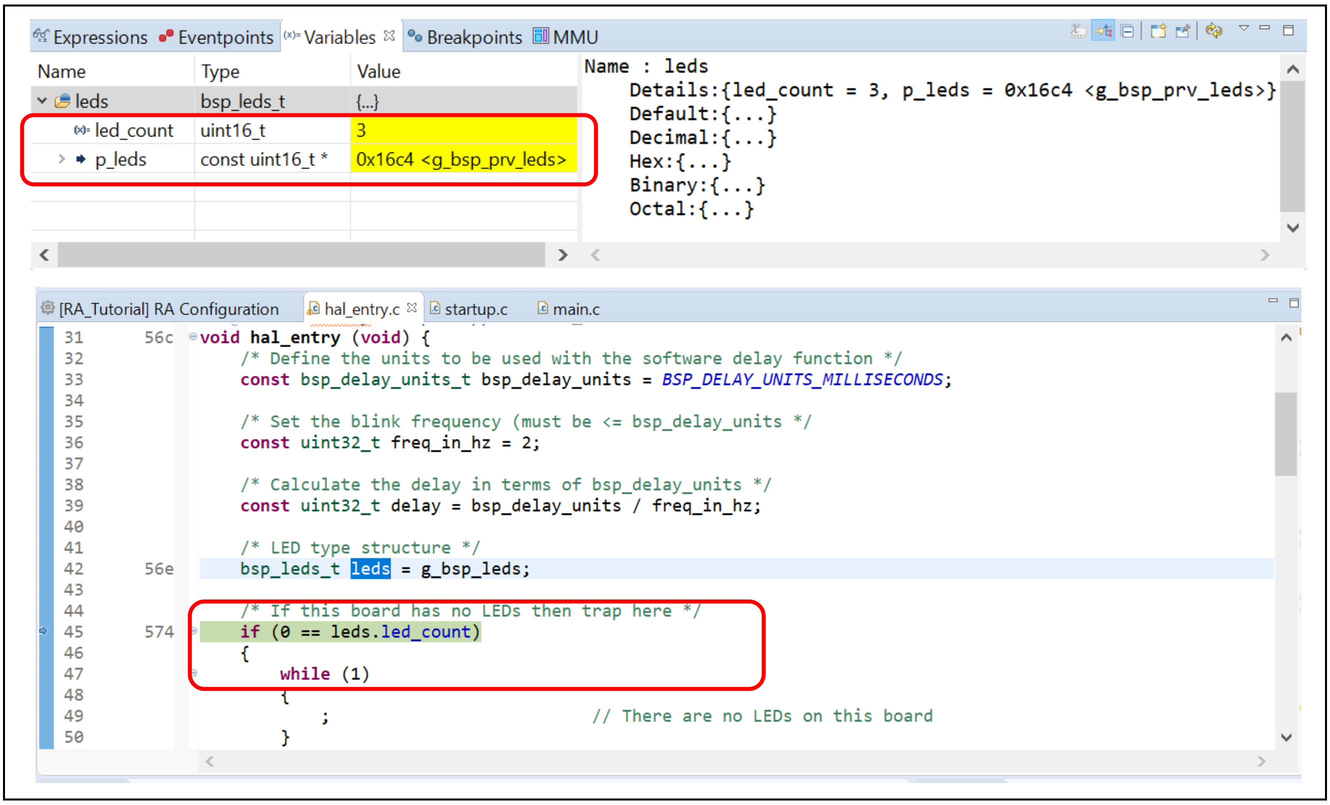

Figure 108. Debug – Variables View

5.3.9 IO Registers View

The IO Registers are also known as the Special Function Registers (SFRs). The **IO Registers** view displays all the registers defined in a target-specific IO file. Users can further customize the **IO Registers** view by adding specific IO registers to the **Selected Registers** pane.

To view selected IO registers:

1. Click **Renesas Views** → **Debug** → **IO Registers** to open the **IO Registers** view.
2. Under the **All Registers** tab, locate a module (for example, CAC) in the **IO Registers** view. Expand its IO register list.
3. Drag and drop its registers (the CAICR and CASTR) to the **Selected Registers** pane. A green dot next to the IO register indicates the status of being a selected register.
4. Switch to the **Selected Registers** tab to view the selected IO Registers.

The expanded IO register list may take more time to load in the **All Registers** pane. Hence, it is advisable to customize and view multiple selected IO registers from the **Selected Registers** pane.

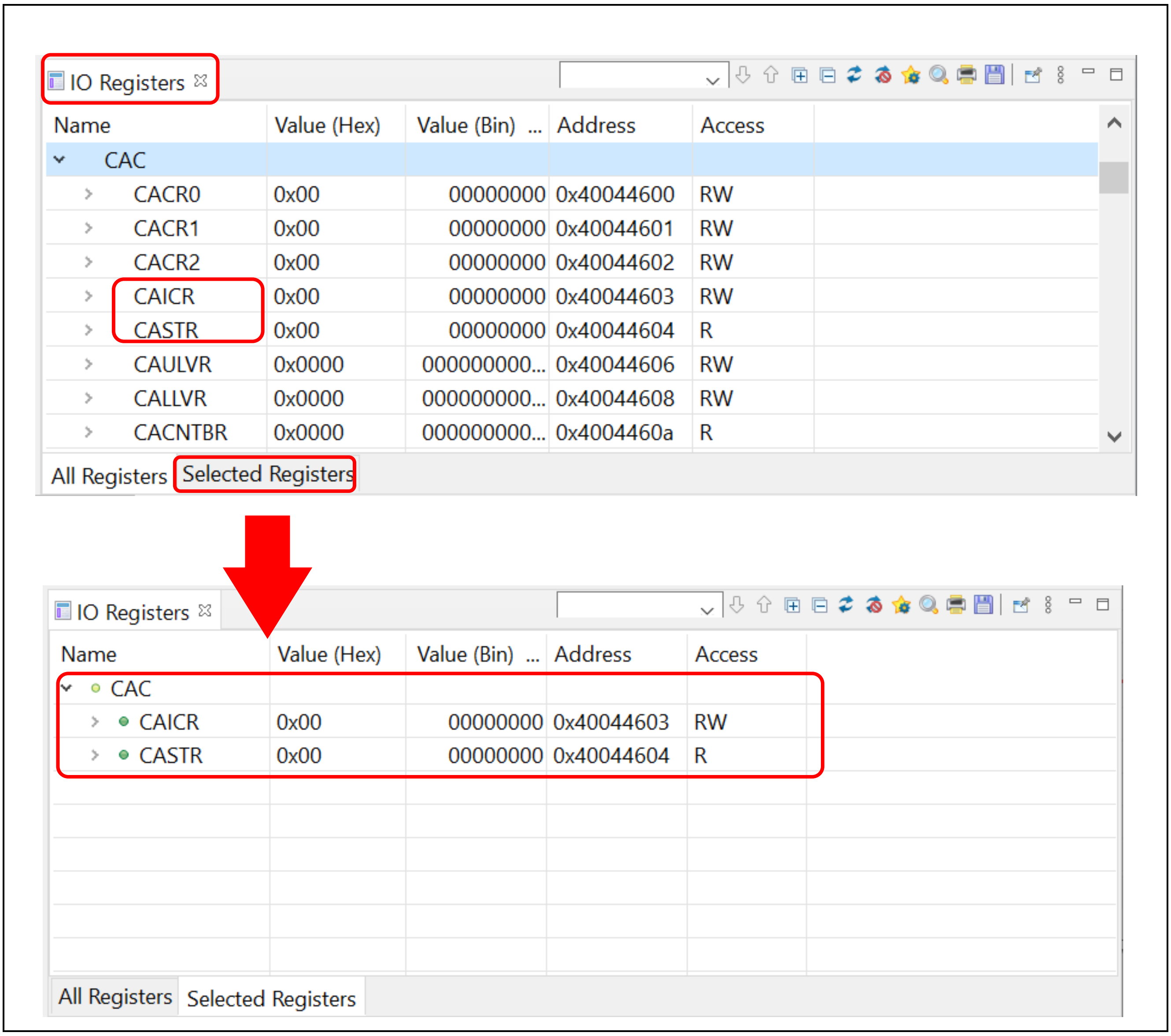

Figure 109. Debug – IO Registers View

5.3.10 Eventpoints View

An ‘event’ refers to a combination of conditions set for executing break or trace features during program execution. The **Eventpoints** view enables users to set up or view defined events of different categories, for example, trace start, trace stop, or event break.

Data access event break is supported for RA projects. The emulator detects access under a specified condition to a specified address or a specified address range. This allows complex address and data matching criteria to be set up.

Event combination (OR, AND (cumulative), and Sequential) can be applied to two or more events.

Table 1. Event combination

Event combination	Explanation
OR	The condition is met when any one of the specified events occurs.
AND (cumulative)	The condition is met when all of the specified events occur regardless of the timing.
Sequential	The condition is met when the specified events occur in a specified order.

To set an event break for a global variable when address/data is matched (for example, when `g_bsp_leds` is accessed):

Click **Renesas Views** → **Debug** → **Eventpoints** to open the **Eventpoints** view.

Double-click the **Event Break** option to open the **Edit Event Break** dialog box.

Click the **Add…** button to continue.

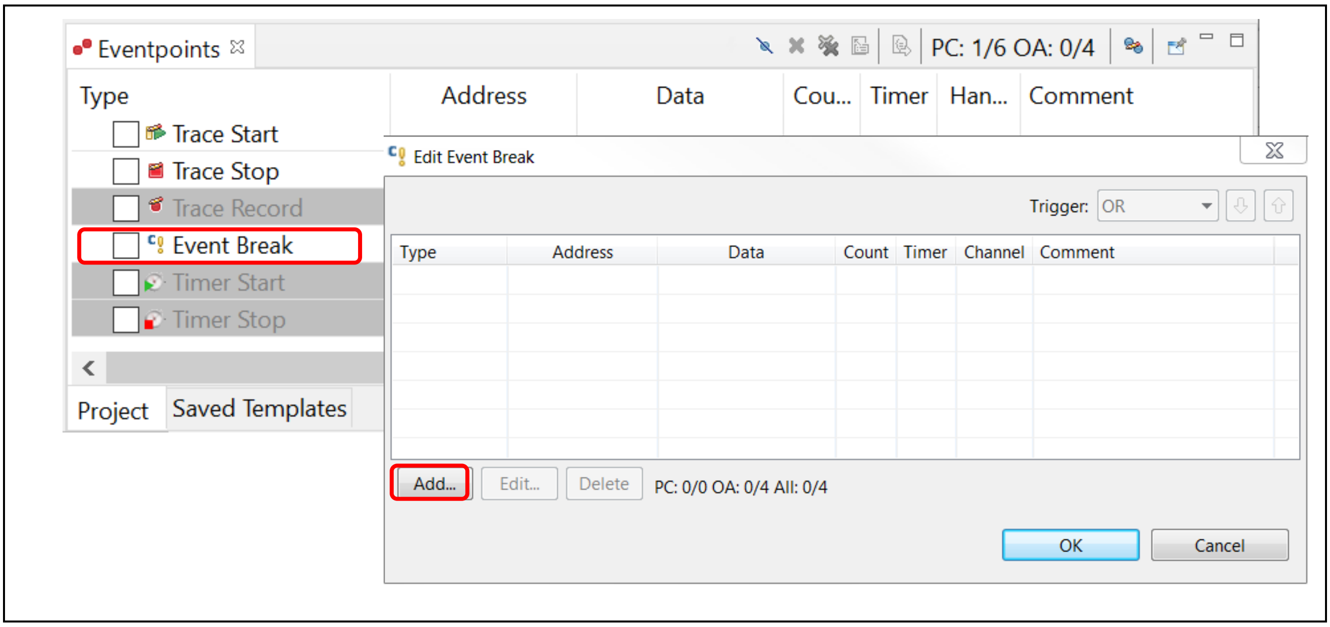

Figure 110. Debug – Eventpoints View (1/2)

Select the **Data Access** eventpoint type.

Go to the **Address Settings** tab and click the **…** icon to browse for the symbol **g_bsp_leds**. (The address of this global variable is `&g_bsp_leds`.)

Next, switch to the **Data Access Settings** tab and set the **Read/Write** selection to **Read**.

Click **OK** to proceed.

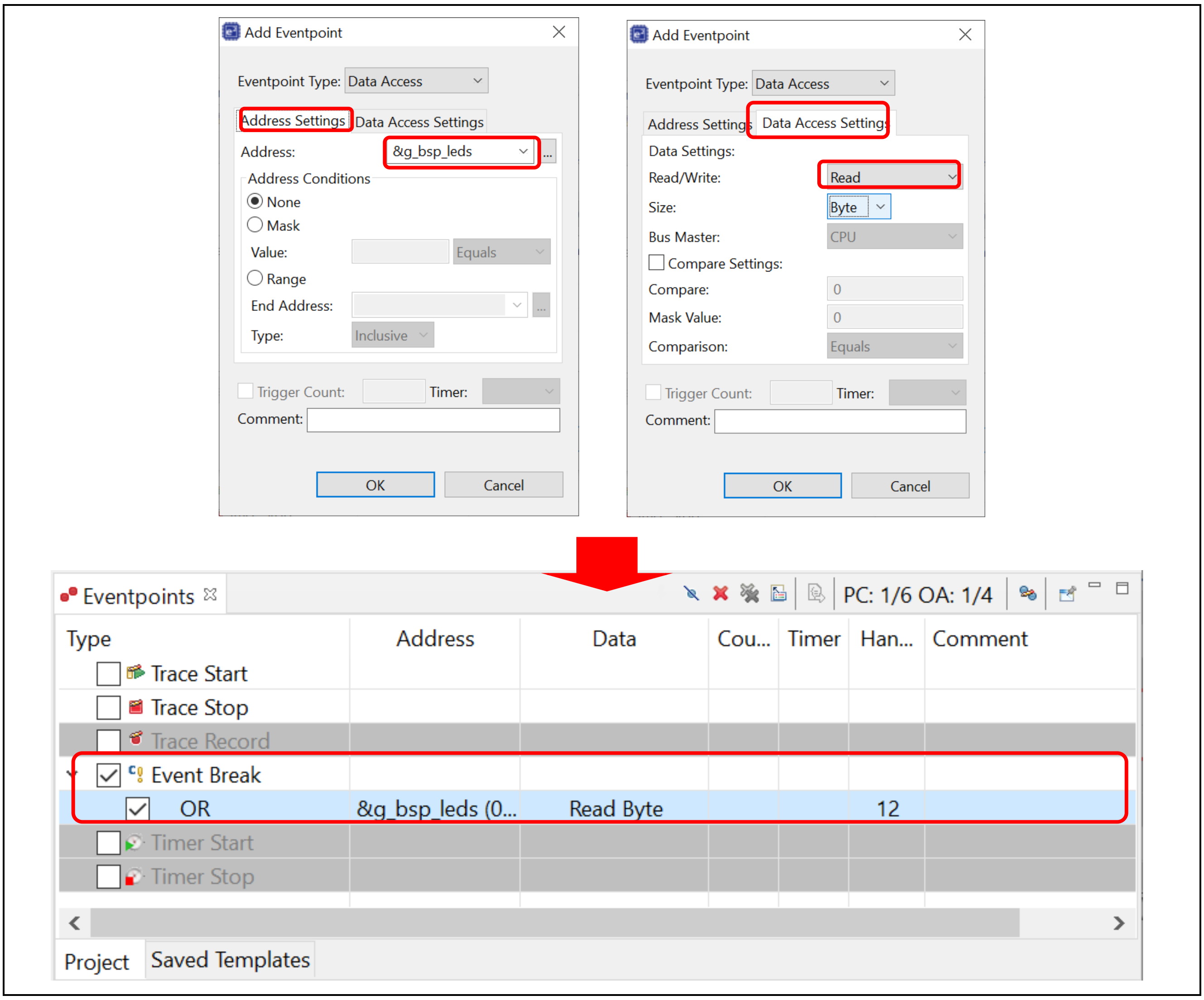

Figure 111. Debug – Eventpoints View (2/2)

Perform a reset to execute the program from the start.

The figure below shows that when the variable `g_bsp_leds` is accessed (read), the program stops.

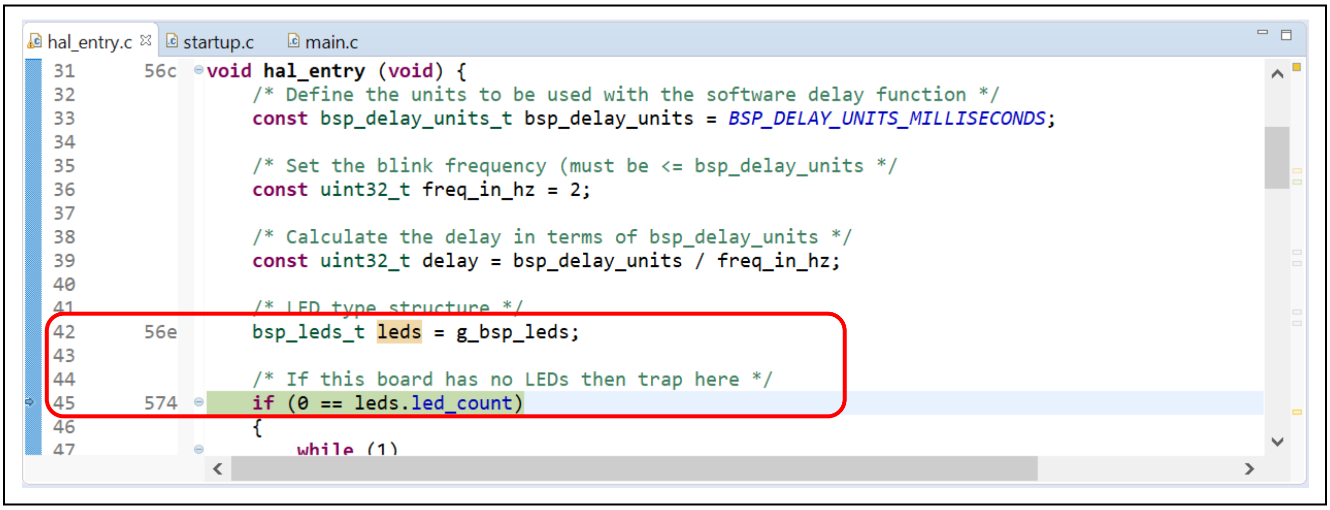

Figure 112. Debug – Execution of Event Break

5.3.11 Trace View

Tracing means the acquisition of bus information per cycle from the trace memory during user program execution. The acquired trace information is displayed in the **Trace** view. It helps users to track the program execution flow to search for and examine the points where problems arise.

The trace buffer is limited, therefore older trace data is overwritten with new data after the buffer has become full.

To set a trace until the program is suspended, users can do as following:

1. Click **Renesas Views** → **Debug** → **Trace** to open the **Trace** view.
2. Turn on the **Trace** view by selecting the  icon.

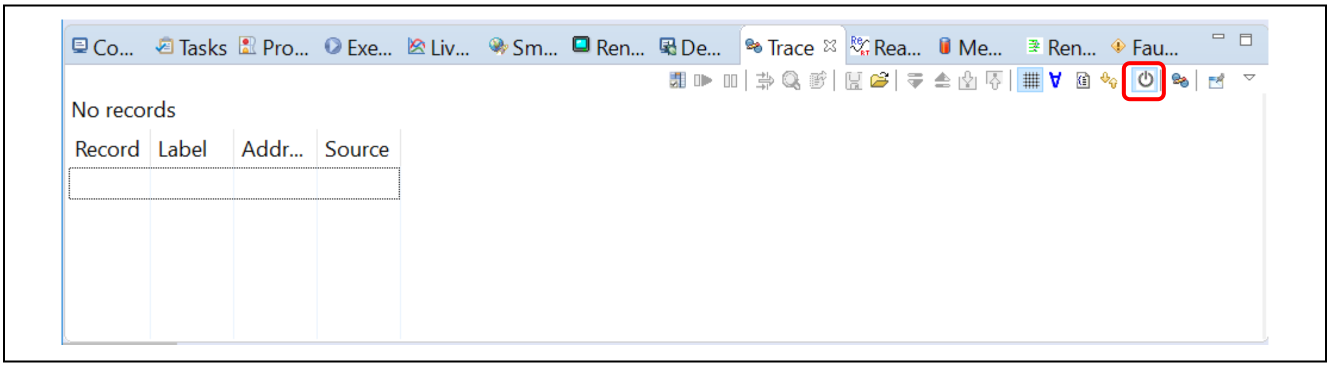

Figure 113. Debug – Turn On Trace View

3. Execute the program and stop program execution by using a breakpoint or by pressing the **Suspend** button on the **Debug** toolbar. The content stored in trace memory at that point in time is displayed as trace result.
4. Select the display mode by clicking on the corresponding button. The following figure shows the trace result before the `main()` function is executed.

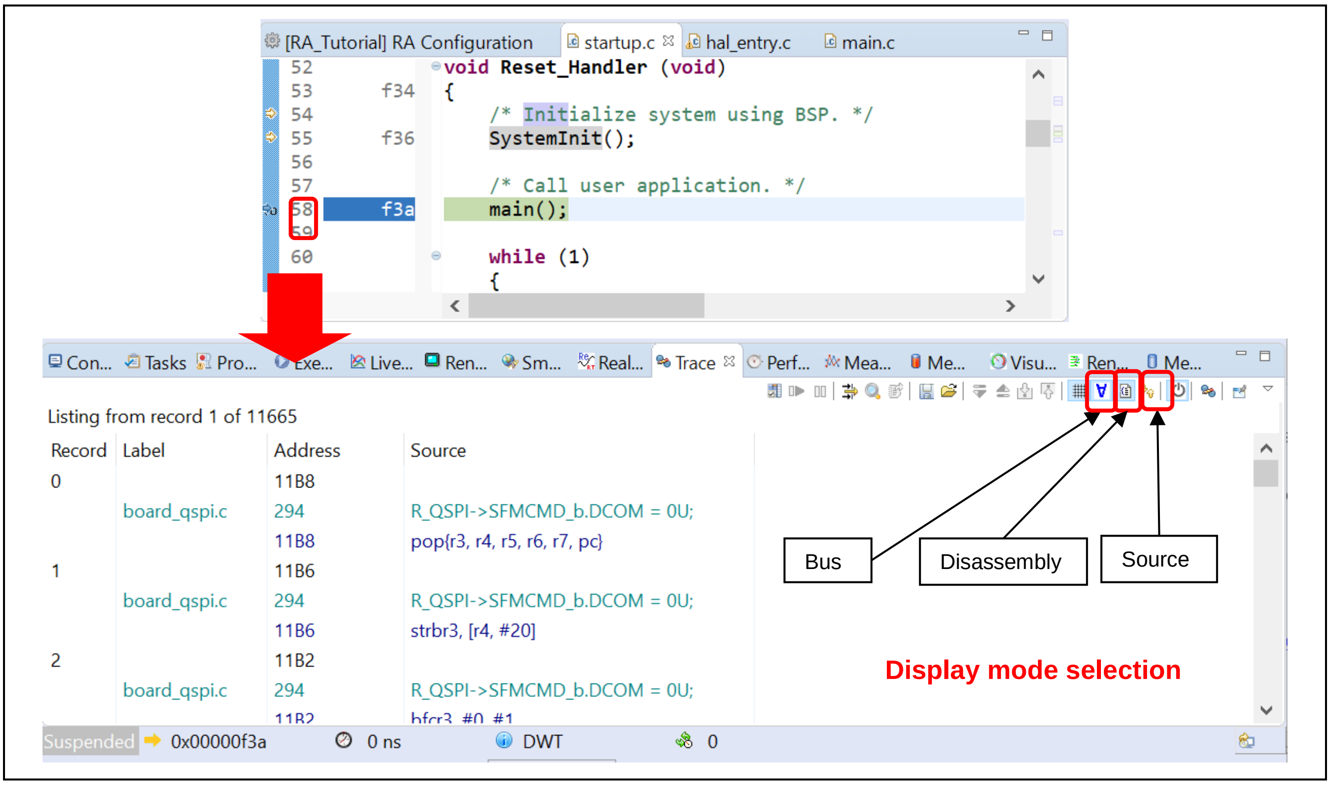

Figure 114. Debug – Select Display Mode In Trace View

The trace records are displayed from oldest data to latest data by default. The display order can be changed by clicking the button.

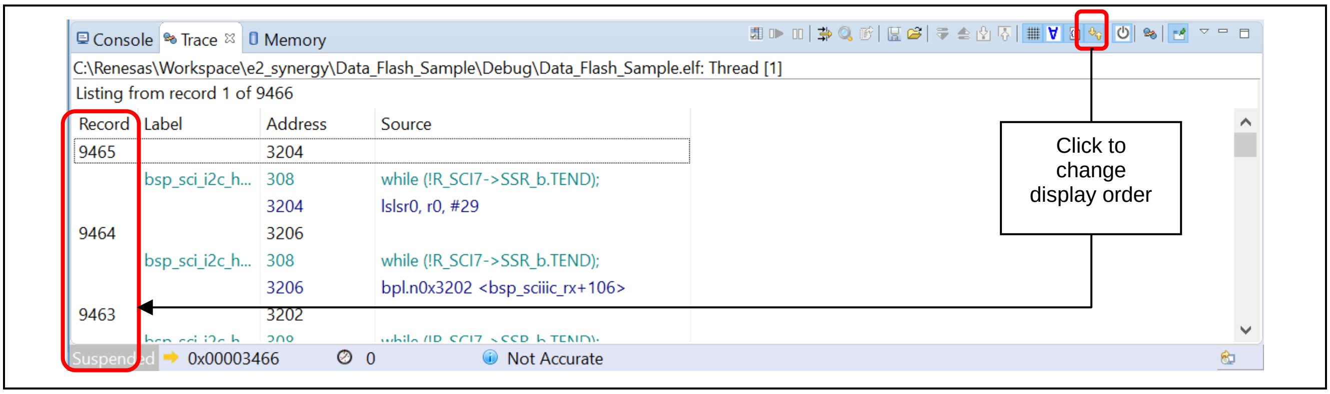

Figure 115. Changing Display Order

The trace result can be filtered by clicking on button. You can select filtering by **Record** and/or **Address**.

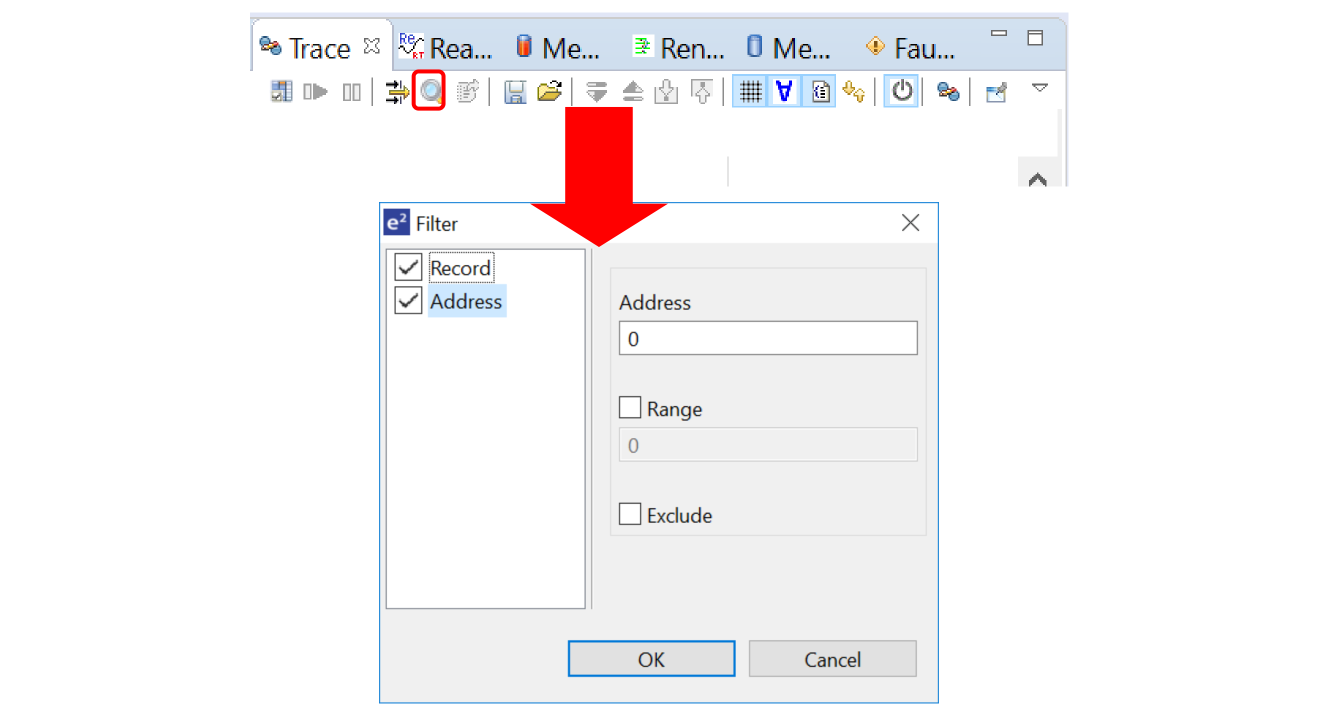

Figure 116. Debug – Filter Trace Result

The trace result can be saved to a `.csv` file (with the inclusion of bus, assembly, and source information). **Trace** view also allows loading trace results from a `.csv` file.

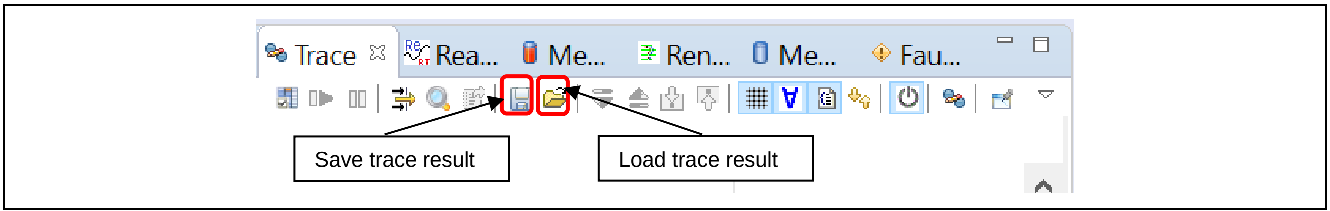

Figure 117. Debug – Save And Load Trace Result

5.3.12 Fault Status View

The **Fault Status** view shows the bit status of several fault status registers and the value of the key register to the user when a hardware fault crash occurs. When a hardware fault occurs, the bits of the register related to the cause of the fault are checked and the r0, r1, r2, r3, r12, lr, pc, and psr register values are displayed. This is shown in figure below. This function is available in e² studio v5.2 and above.

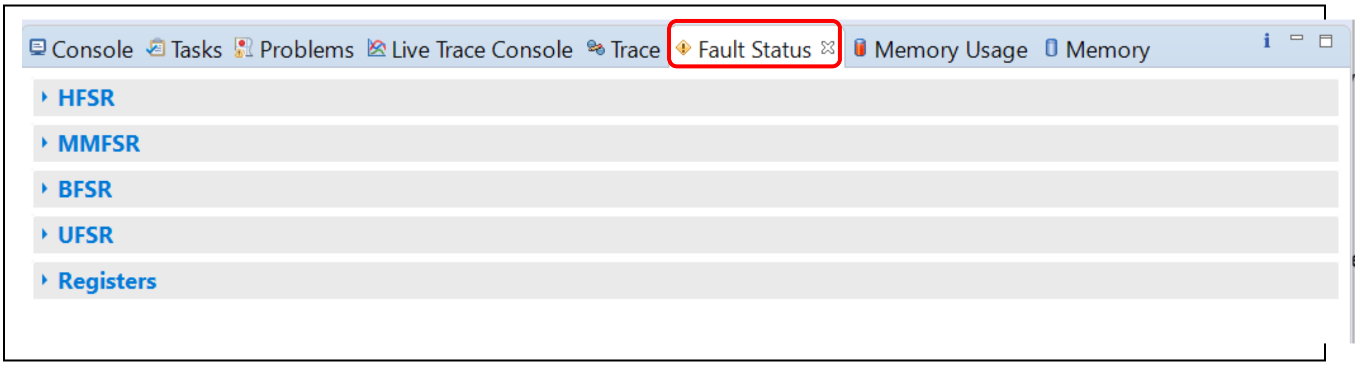

Figure 118. Fault Status - No Hardware Fault

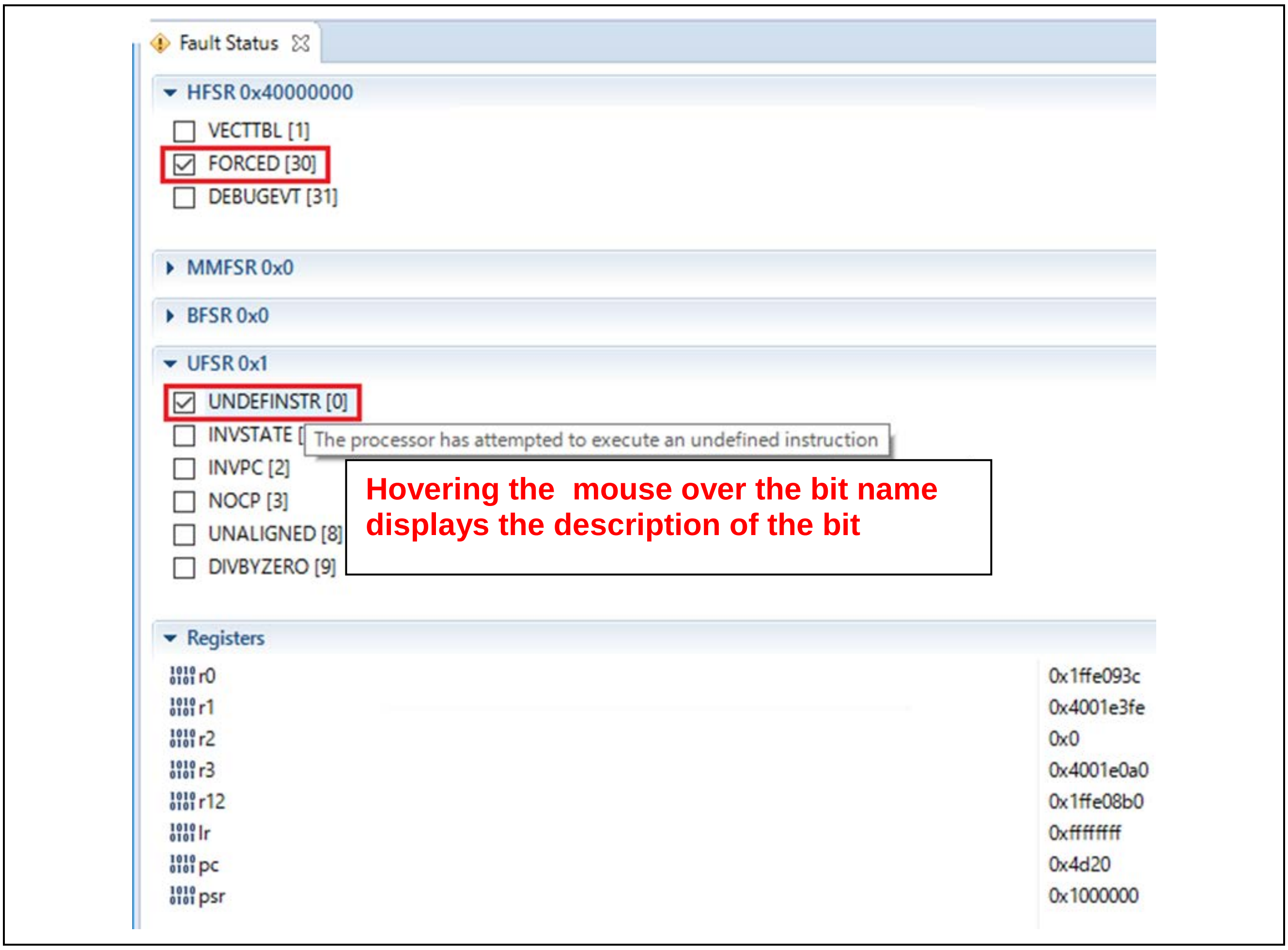

Figure 119. Fault Status Hardware Fault Occurred

5.3.13 Run Break Timer

The **Run Break Timer** feature allows the user to see the last execution performance on the status bar. When the program is suspended, the user can check the current program counter (PC), the last execution timing either in time or CPU cycles, and the accuracy or measurement method used.

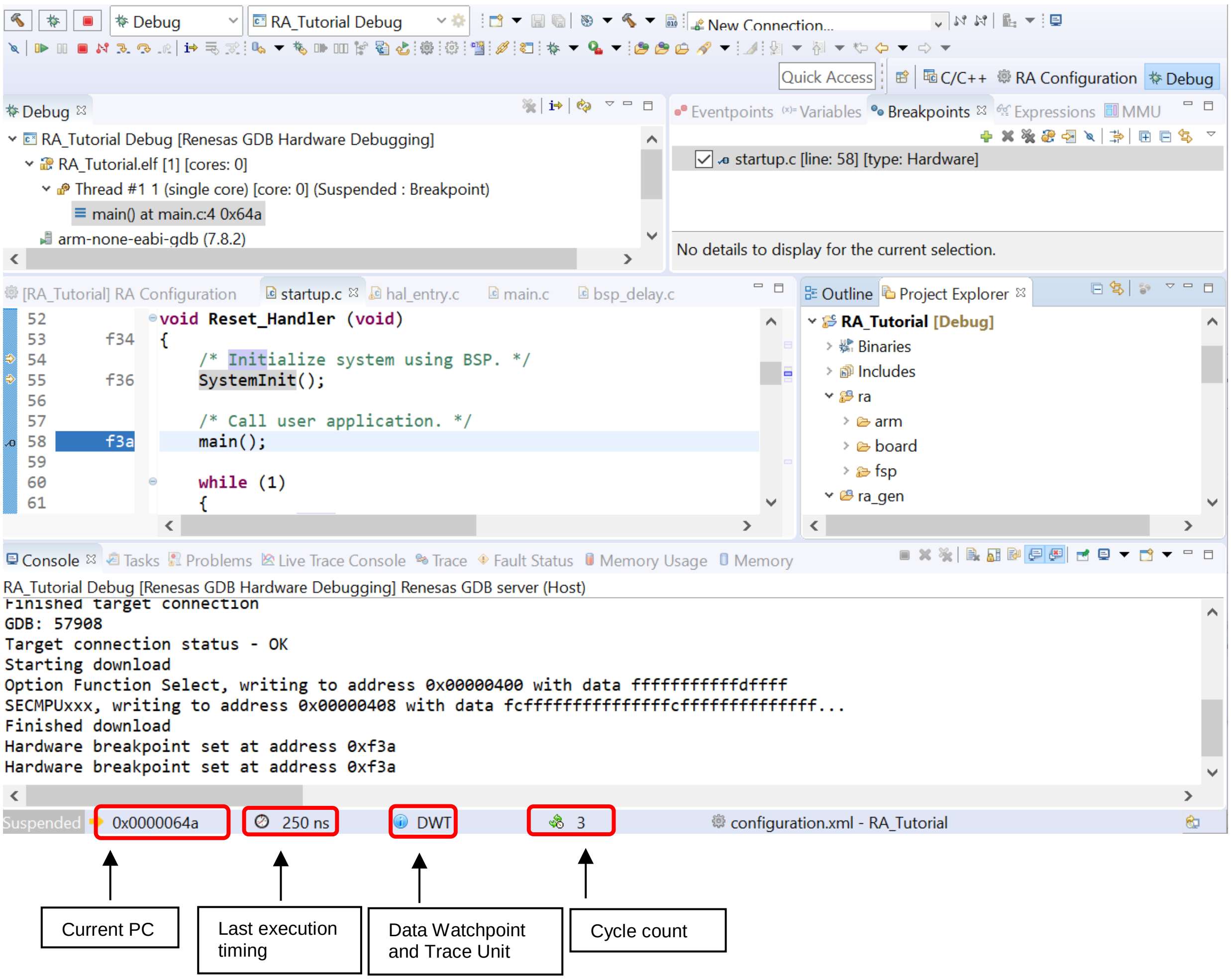

Figure 120. Run Break Timer Shows the Last Execution Performance

The following table shows the support of **Run Break Timer** feature available for various RA devices.

Table 2. Support for Run Break Timer

Device	Debugger	Support
RA2 Series (Cortex-M23)	J-Link	System Time
RA4, RA6 Series	J-Link	Data Watchpoint and Trace Unit (DWT) – Cycle Count and number of overflows calculated using the System Time

The **Run Break Timer** feature is supported in e² studio v7.3.0 and higher versions. For updates in the specification, refer to the e² studio release note at https://www.renesas.com/e2studio.

6. Setting up a FreeRTOS Application

This example shows how to generate and build an RA project to include FreeRTOS objects and the General Purpose Timer (GPT) module using the project template **FreeRTOS – Blinky – Static Allocation**.

6.1 General Purpose Timer Example in FreeRTOS

In the **FreeRTOS – Blinky – Static Allocation** RA project from **Project Template Selection**, LEDs are blinked by putting a task for a short delay before toggling the LEDs state.

In this example, instead of a delay, the Blinky Thread waits for a semaphore and a timer interrupt (generated by GPT) which sets this semaphore every 1 second so that thread can resume.

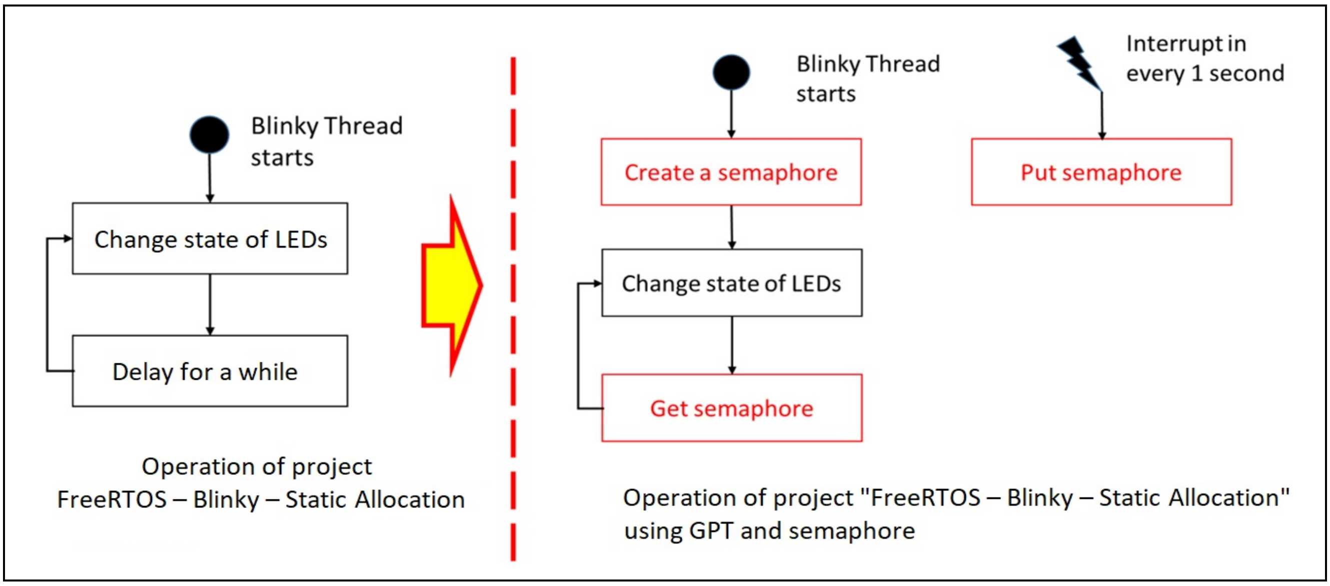

Figure 121. Setting Up a FreeRTOS Application – Introduction

6.2 Creating the Sample Project

To create a sample FreeRTOS project with GPT and semaphore, configure the RA project as follows:

1. Invoke the **New Project** editor and follow the steps in Section 3.1 (Generating a New RA Project) to generate a new project. However, in the **Build Artifact and RTOS Selection** dialog, select **FreeRTOS** and in the **Project Template** dialog, select **FreeRTOS – Blinky – Static Allocation.**

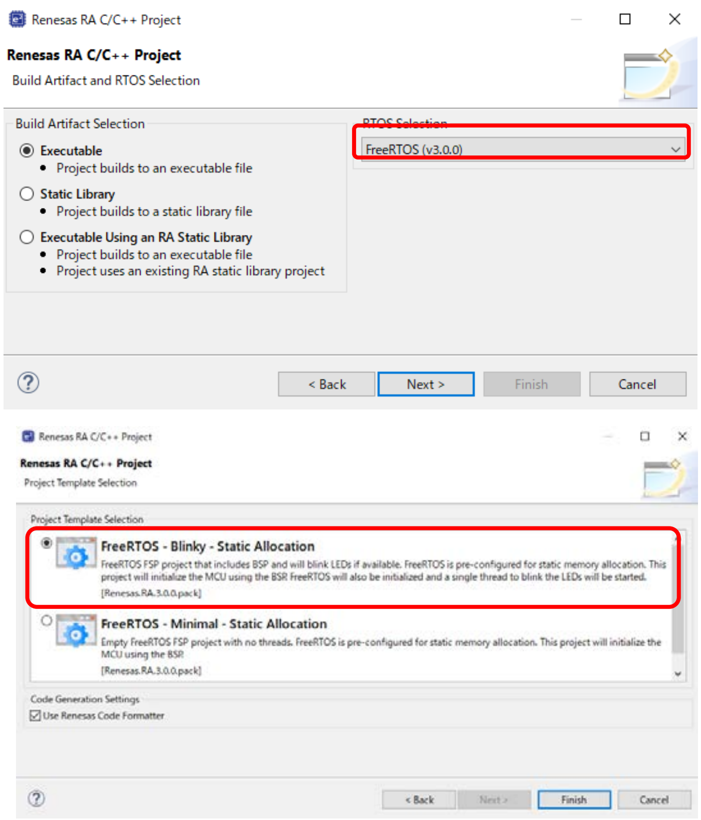

Figure 122. Setting Up a FreeRTOS Application - Create New Project

2. Open the **Stacks** page in the **RA Project Configuration**. Please refer to section 3.5.5: Stacks Configuration Page.
3. Add the GPT module to the Blinky Thread by selecting **Blinky Thread** in the **Threads** panel and selecting **New Stack → Driver → Timers → Timer Driver on r_gpt** in the **Stacks** panel.

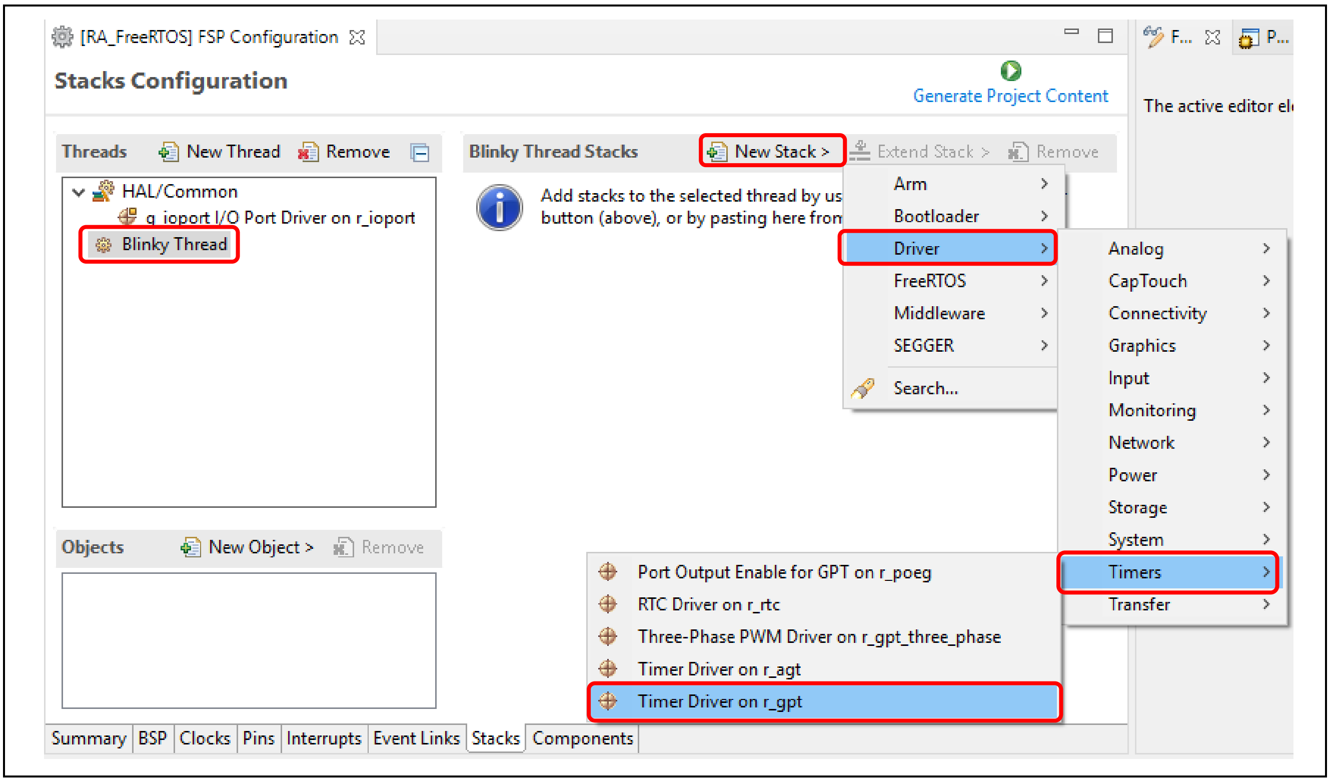

Figure 123. Setting Up a FreeRTOS Application – Adding The GPT Module

4. Configure the GPT module as follows.
   - Name: g_timer
   - Mode: Periodic
   - Period: 1
   - Period Unit: Seconds
   - Callback: gpt_callback
   - Overflow/Crest Interrupt priority: Priority 2

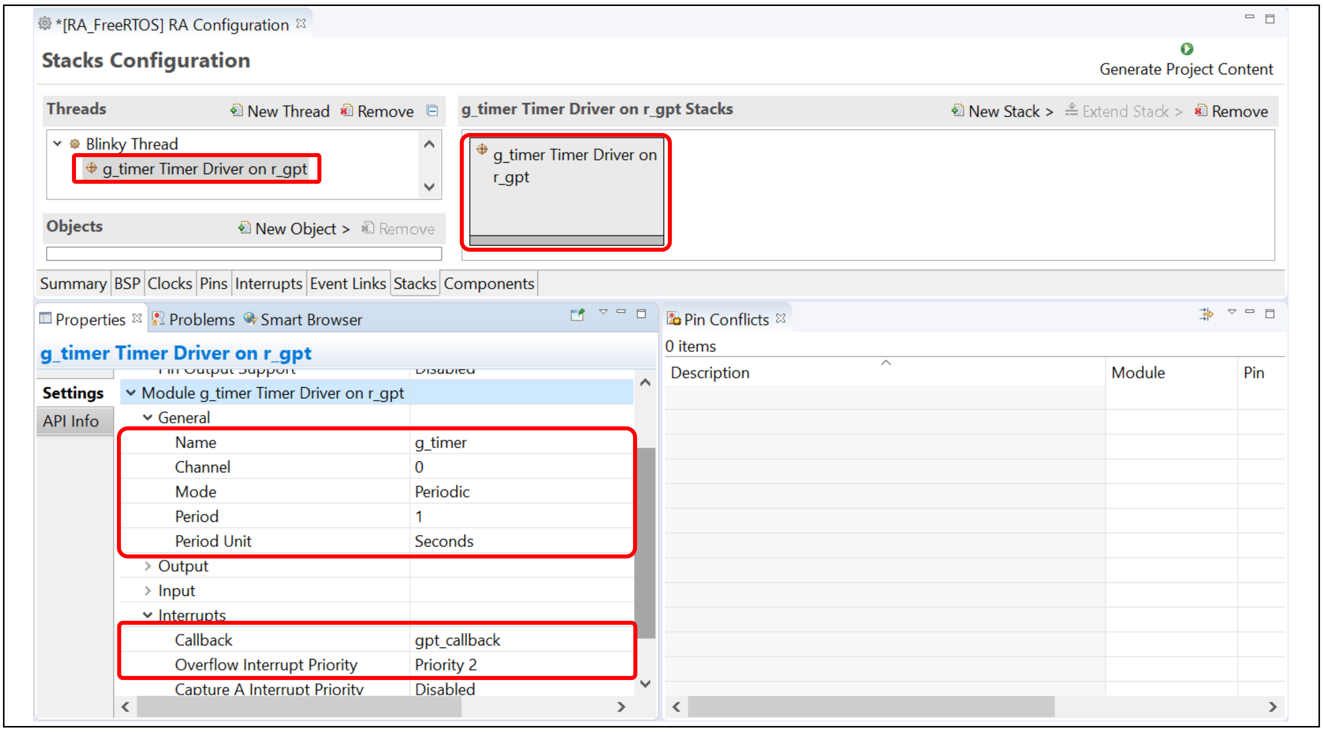

Figure 124. Setting Up a FreeRTOS Application – GPT Module Configuration

5. Add a semaphore object to the **Blinky Thread** by selecting the **Blinky Thread** in the **Threads** panel and select **New Object → Binary Semaphore** in the **Objects** panel.

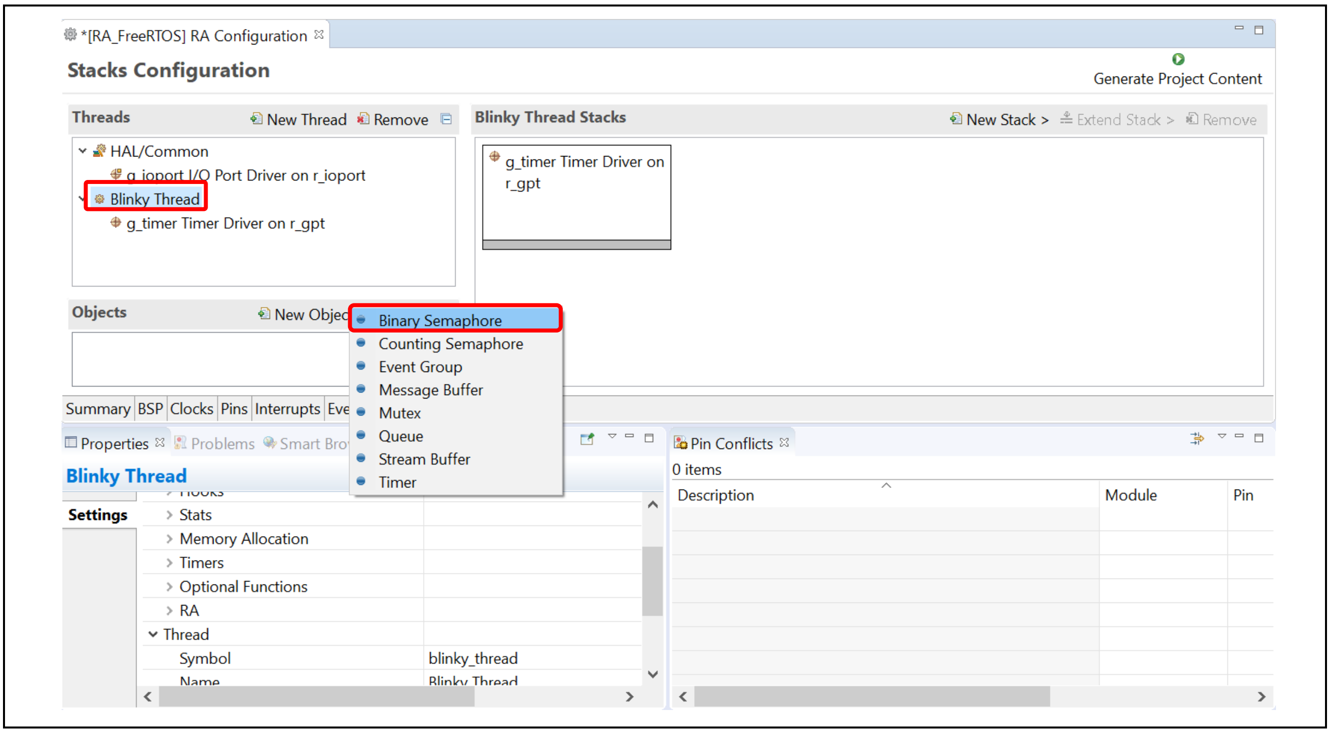

Figure 125. Setting Up a FreeRTOS Application – Adding A Semaphore Object

6. Configure this newly created semaphore as follows:
   - Name: **Blinky Semaphore**
   - Symbol: **g_blinky_semaphore**

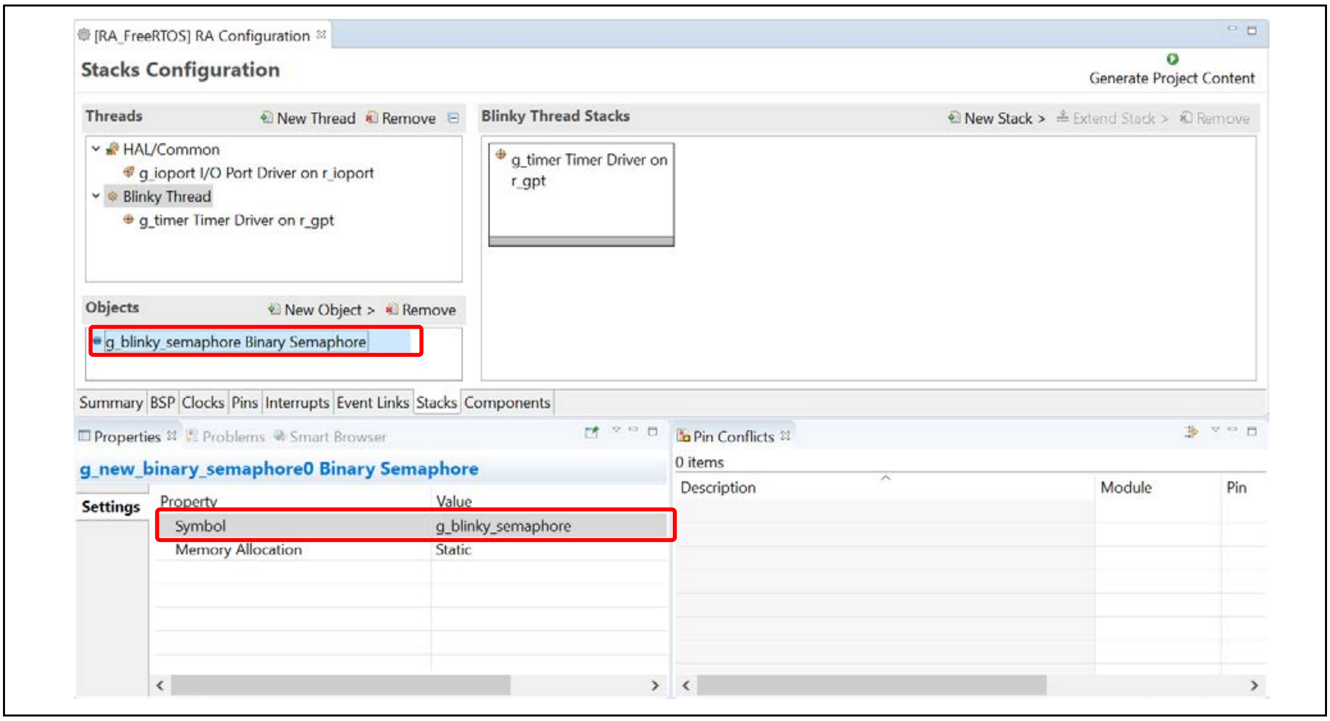

Figure 126. Setting Up a FreeRTOS Application – Semaphore Object Configuration

7. Press **Ctrl + S** to save the setting and click the **Generate Project Content** 

button to generate source code.
8. Open `src\blinky_thread_entry.c` and implement the following contents:
   — Add source code to initialize the GPT module before the `while(1)` loop in `blinky_thread_entry()`.

```
g_timer.p_api->open(g_timer.p_ctrl, g_timer.p_cfg);
g_timer.p_api->start(g_timer.p_ctrl);
```

   — Delete the task delay instruction and add code to wait for the semaphore in `blinky_thread_entry()`.

```
xSemaphoreTake(g_blinky_semaphore, portMAX_DELAY);
```

   — Implement the `gpt_callback()` function to signal the semaphore for the Blinky thread.

```
void gpt_callback(timer_callback_args_t *p_args) {
(void)p_args;
static signed portBASE_TYPE xHigherPriorityTaskWoken;
xSemaphoreGiveFromISR(g_blinky_semaphore, &xHigherPriorityTaskWoken);
}
```

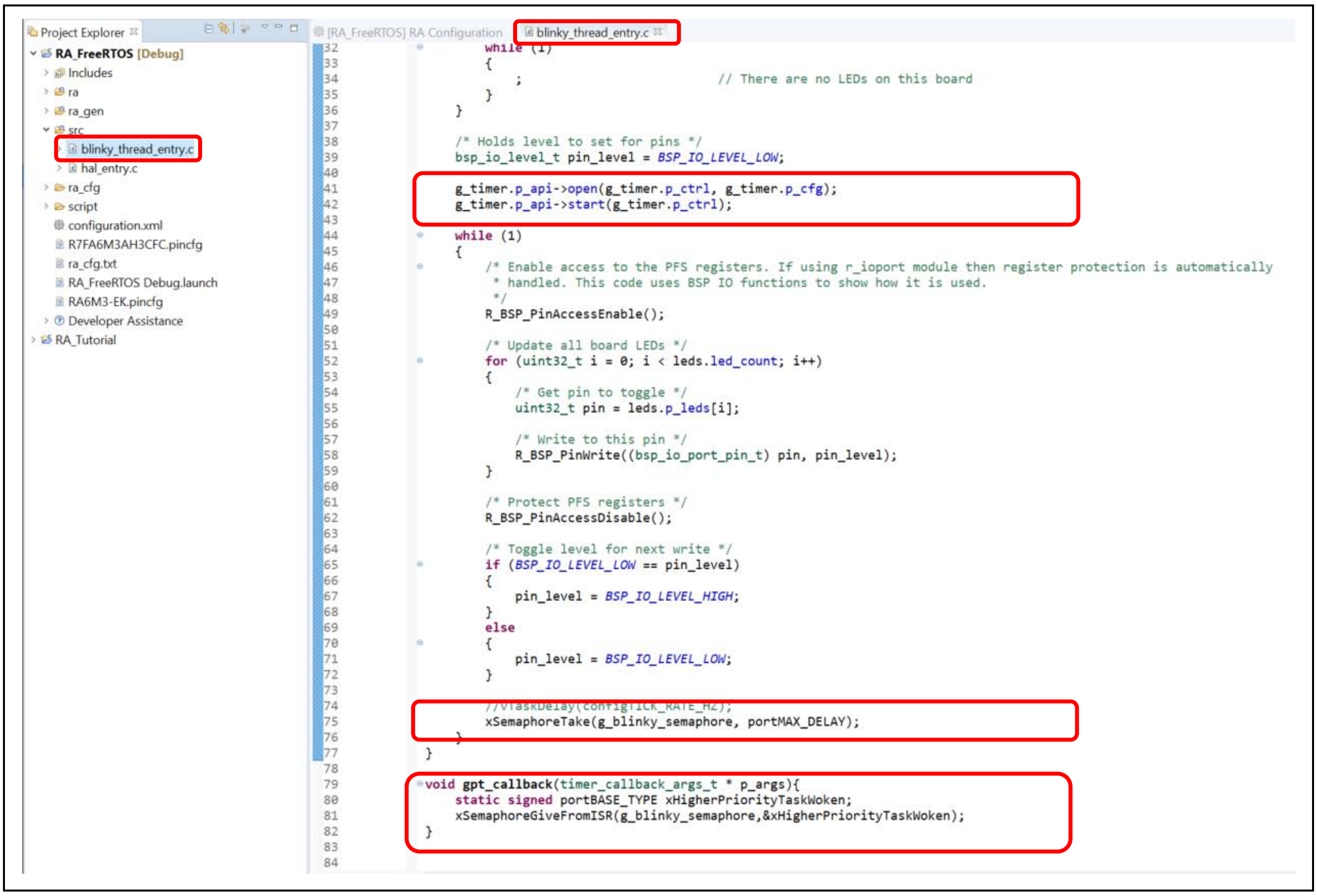

Figure 127. Setting Up a FreeRTOS Application – Adding User Source Code

9. Build and run the project on the EK-RA6M3 board. Confirm that the LEDs are turned ON/OFF every 1 second.

7. Setting up an Azure RTOS Application

This example shows how to generate and build an RA project to include Azure RTOS objects and the General Purpose Timer (GPT) module using the project template **Azure RTOS ThreadX – Blinky**.

7.1 General Purpose Timer Example in Azure RTOS

In the **Azure RTOS ThreadX – Blinky** RA project from **Project Template Selection**, LEDs are blinked by putting a task for a short delay before toggling the LEDs state .

In this example, instead of a delay, the Blinky Thread waits for a semaphore and a timer interrupt (generated by GPT) which sets this semaphore every 1 second so that thread can resume.

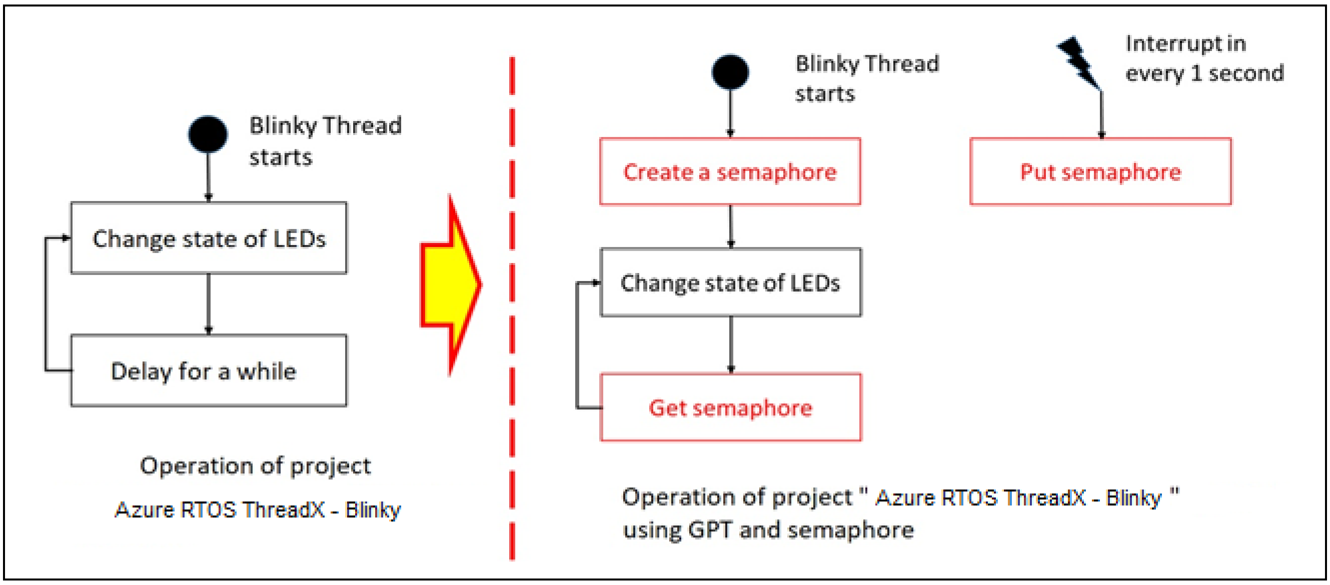

Figure 128. Setting up an Azure RTOS Application – Introduction

7.2 Creating the Sample Project

To create a sample Azure RTOS project with GPT and semaphore, configure the RA project as follows:

1. Invoke the **New Project** editor and follow the steps in Section 3.1 (Generating a New RA Project) to generate a new project. However, in the **Build Artifact and RTOS Selection** dialog, select **Azure RTOS ThreadX** and in the **Project Template** dialog, select **Azure RTOS ThreadX – Blinky.**

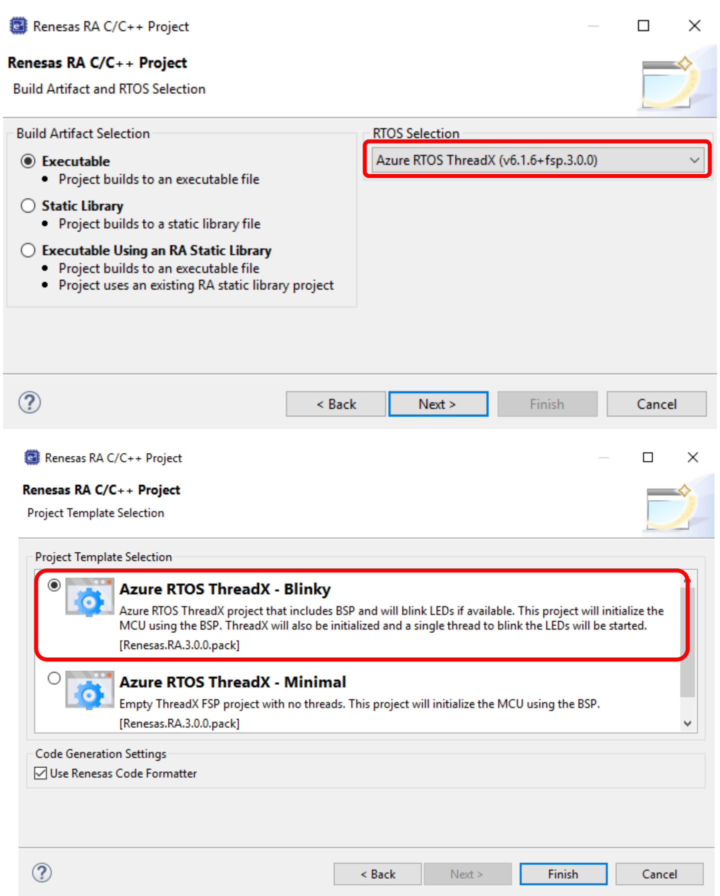

Figure 129. Setting up an Azure RTOS Application - Create New Project

2. Open the **Stacks** page in the **RA Project Configuration**. Please refer to section 3.5.5: Stacks Configuration Page.
3. Add the GPT module to the Blinky Thread by selecting **Blinky Thread** in the **Threads** panel and selecting **New Stack** → **Driver** → **Timers** → **Timer Driver on r_gpt** in the **Stacks** panel.

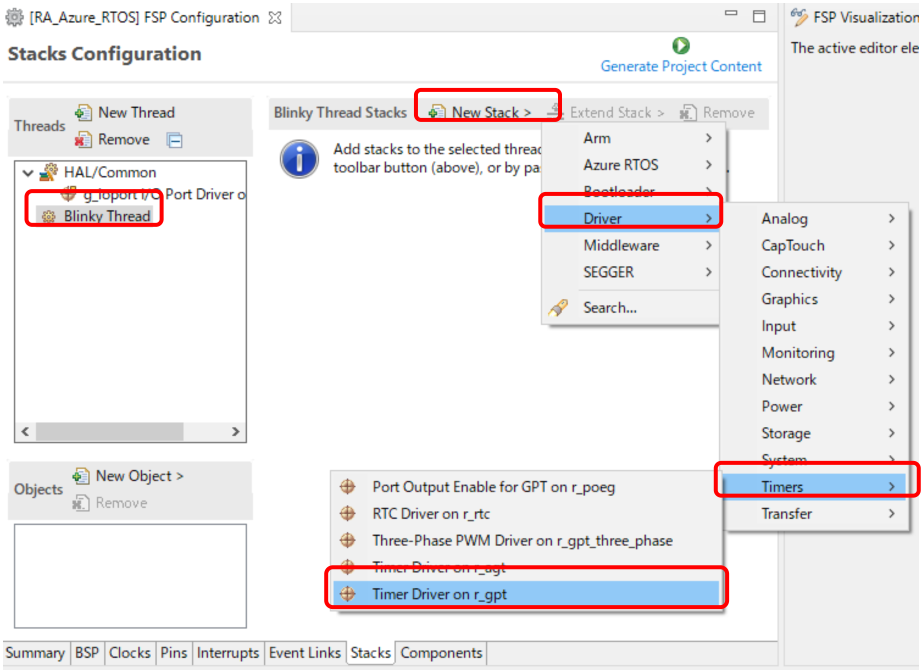

Figure 130. Setting up an Azure RTOS Application – Adding The GPT Module

4. Configure the GPT module as follows.
   - Name: g_timer
   - Mode: Periodic
   - Period: 1
   - Period Unit: Seconds
   - Callback: gpt_callback
   - Overflow/Crest Interrupt priority: Priority 2

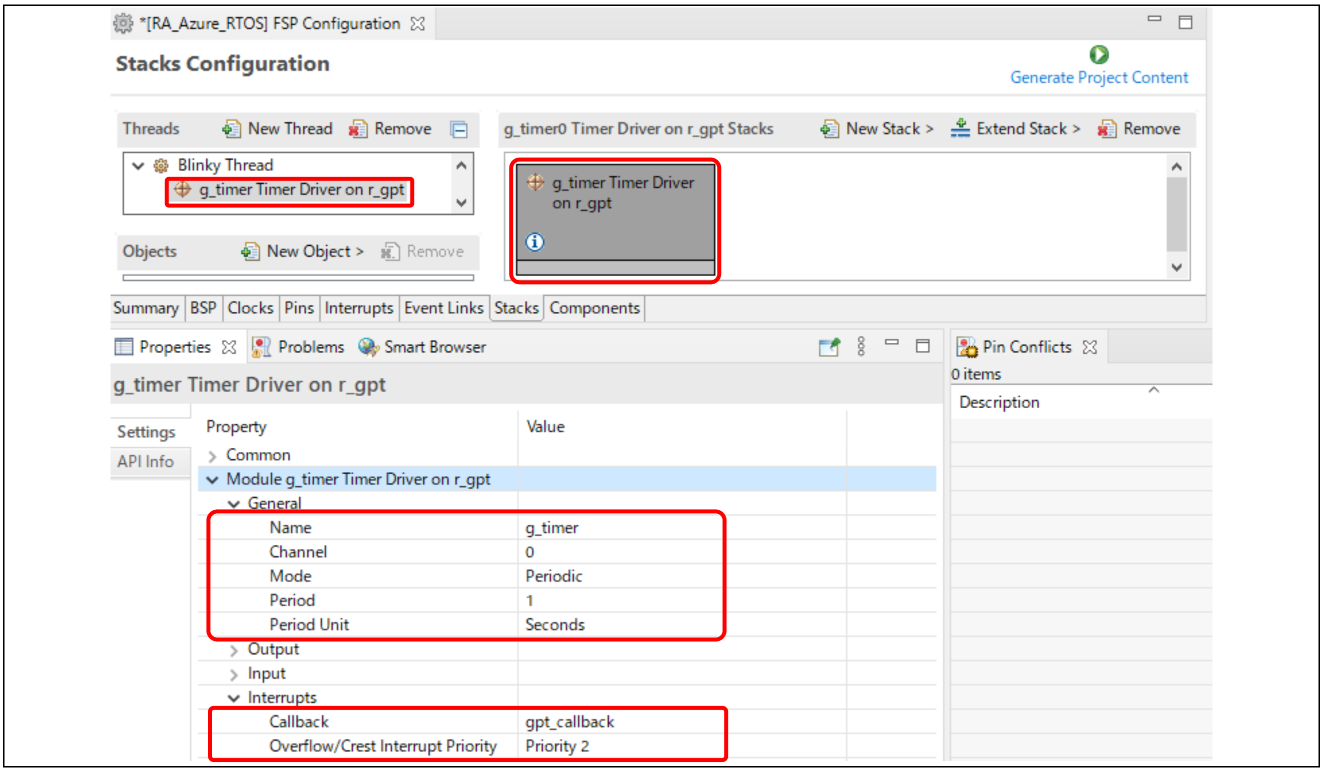

Figure 131. Setting up an Azure RTOS Application – GPT Module Configuration

5. Add a semaphore object to the **Blinky Thread** by selecting the **Blinky Thread** in the **Threads** panel and select **New Object** → **Semaphore** in the **Objects** panel.

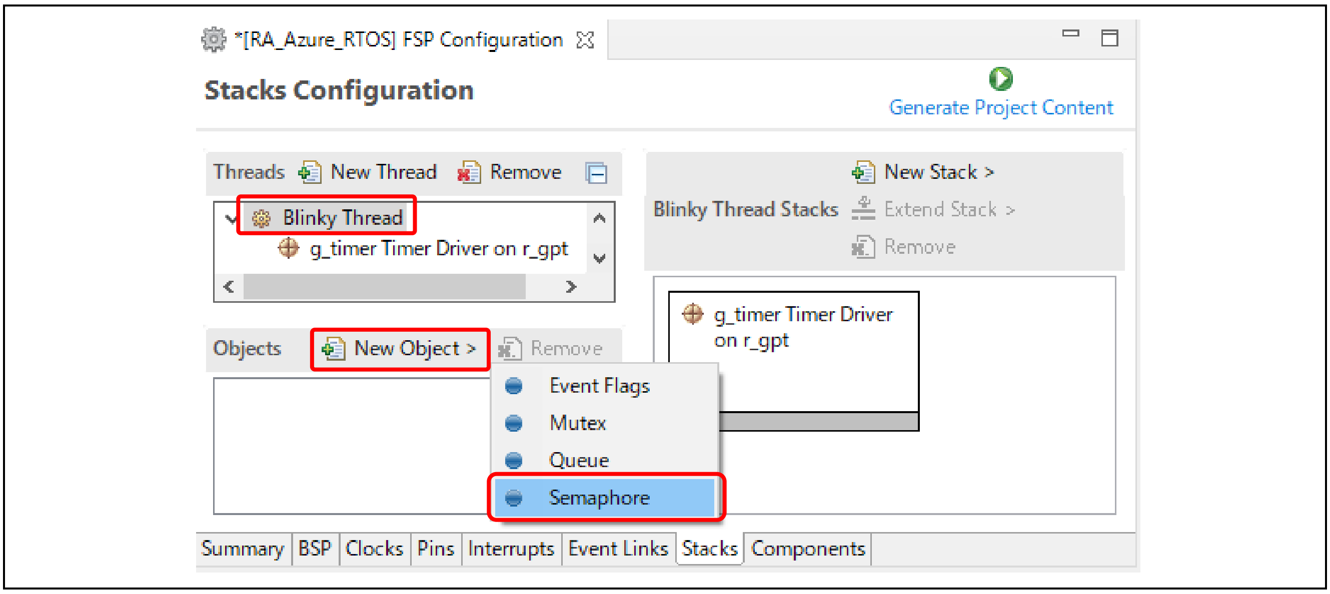

Figure 132. Setting up an Azure RTOS Application – Adding A Semaphore Object

6. Configure this newly created semaphore as follows:
   — Name: **Blinky Semaphore**
   — Symbol: **g_blinky_semaphore**

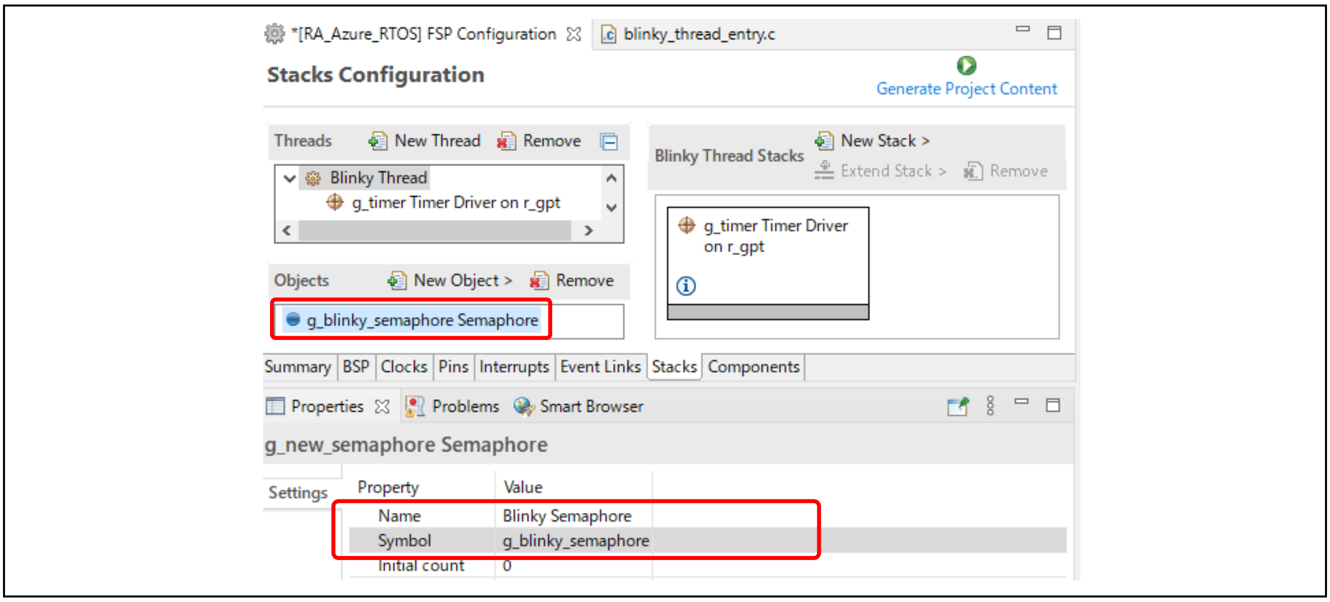

Figure 133. Setting up an Azure RTOS Application – Semaphore Object Configuration

7. Press **Ctrl + S** to save the setting and click the **Generate Project Content**  button to generate source code content.
8. Open `src\blinky_thread_entry.c` and implement the following contents:

— Add source code to initialize the GPT module before the `while(1)` loop in `blinky_thread_entry()`.

```
g_timer.p_api->open(g_timer.p_ctrl, g_timer.p_cfg);
g_timer.p_api->start(g_timer.p_ctrl);
```

— Delete the task delay instruction and add code to wait for the semaphore in `blinky_thread_entry()`.

```
tx_semaphore_get(&g_blinky_semaphore, TX_WAIT_FOREVER);
```

— Implement the gpt_callback() function to signal the semaphore for the Blinky thread.

```
void gpt_callback(timer_callback_args_t *p_args) {
(void)p_args;
tx_semaphore_put(&g_blinky_semaphore);
}
```

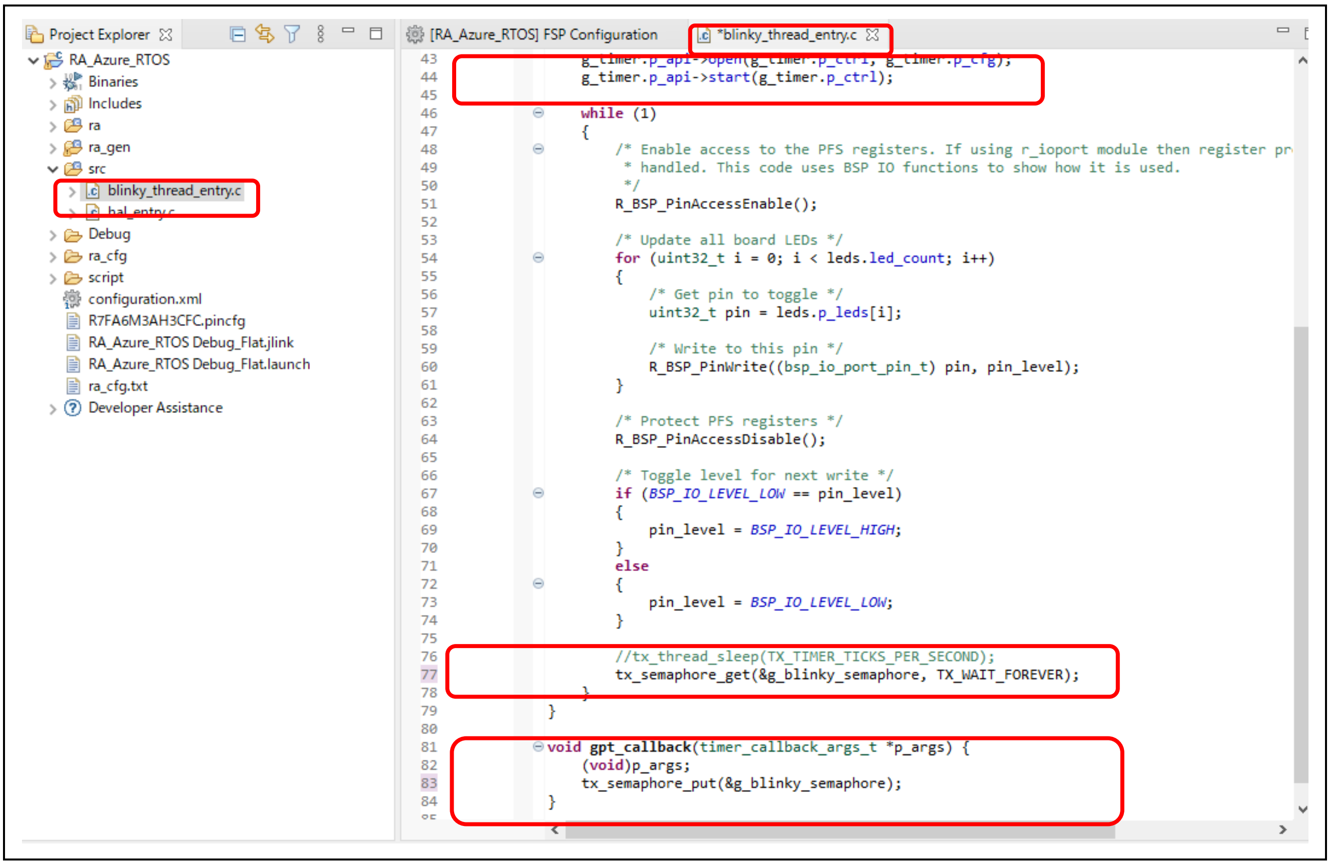

Figure 134. Setting up an Azure RTOS Application – Adding User Source Code

9. Build and run the project on the EK-RA6M3 board. Confirm that the LEDs are turned ON/OFF every 1 second.

8. Help

The help system allows users to browse, search, bookmark, and print help documentation from a separate **Help** window or **Help** view within the workbench. Users can also access an online forum dedicated to the e² studio from here.

Click on **Help** tab to open the **Help** menu.

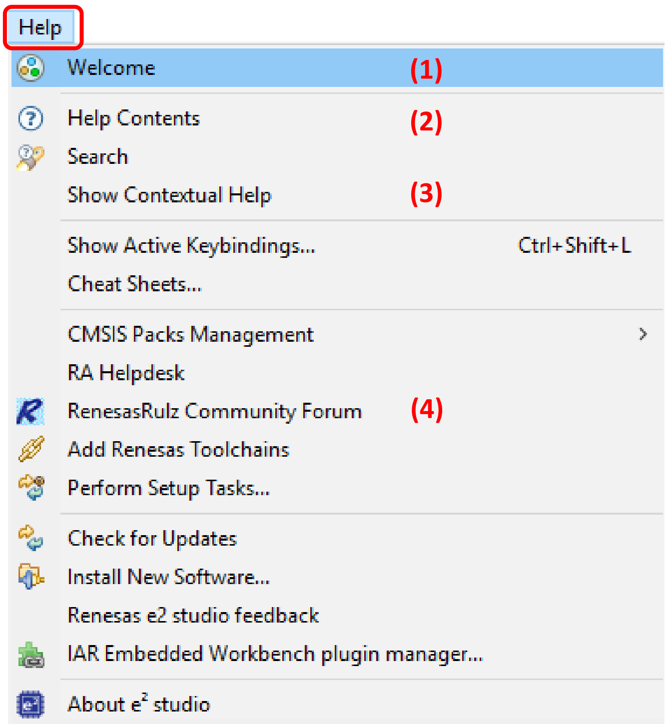

Figure 135. Help – Help Menu

Quick Help tips:

1. Click **Welcome** for an overview of the e² studio and to view Release Notes.
2. Click **Help Contents** to open a separate **Help** window with a search function.
3. Click **Show Contextual Help** to open the **Help** view within the workbench.
4. Click **RenesasRulz Community Forum** to go an online forum that is dedicated to topics and discussions related to the e² studio (Internet connection is required).

Under the **Help Contents** window, there are many useful topics such as

- The **Debugging Projects** topic which provides useful information such as debug configuration and supported number of breakpoints.
  It can be launched by clicking on the **Help** menu → **Help Contents** → **e² studio User Guide** → **Debugging Projects**.
- The **RA Contents** topic which provides information about RA project creation, using the RA Configuration Editor and FAQs.
  It can be launched by clicking on the **Help** menu → **Help Contents** → **RA Contents.**

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul.12.2021	—	First release document

Renesas e^2 studio 2021-04 or higher
User's Manual: Quick Start Guide

Publication Date: Jul.12.21

Published by: Renesas Electronics Corporation

Renesas e² studio 2021-04 or higher
Quick Start Guide

Renesas Electronics Corporation

R20UT4989EJ0100