RZ/T2M

Getting Started with Flexible Software Package

Introduction

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

Target Device

RZ/T2M

Contents

1. Introduction

1.1 Overview

This application note describes how to use the Renesas Flexible Software Package (FSP) running on the Cortex®-R52 (hereinafter referred to as CR52) incorporated on RZ/T2M.

1.2 Introduction to FSP

1.2.1 Purpose

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

1.2.2 e2 studio IDE

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

1.2.3 FSP Smart Configurator

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

For creating RZ project, the FSP SC can currently be used with

- IAR EWARM with IAR toolchain for Arm

1.2.4 FSP Documentation

The related file “FSP Documentation” contains HTML documentations describing the features, APIs and usage notes regarding the BSP and HAL drivers implemented as FSP modules and interfaces. After clicking the “index.html” in “FSP Documentation” to open the introduction page on your html browser, the reference documents for utilizing each FSP module and interface can be read from “API Reference” menu.

1.3 Related documentation files

The related documentation files are shown in the followings.

1.3.1 Renesas Starter Kit+ User’s Manual

This Getting Started Guide refers the following “Evaluation Board Kit User’s Manual”.

- RZ/T2M Group Renesas Starter Kit+ for RZ/T2M User’s Manual
  - Document Number: R20UT4939

1.3.2 FSP Documentation

This Getting Started Guide refers the following “FSP Documentation”.

- RZ/T2M Flexible Software Package Documentation
  - File name: fsp_documentation_vx.x.x.zip

1.4 Starting Development Introduction

FSP application project can be created by e2 studio or FSP SC (for IAR EWARM), and this Getting Started includes tutorial for both tools; the chapters you should read changes.

e2 studio users should read the following chapters:

- Chapter 2 ”Set up Renesas Starter kit+ for RZ/T2M CPU Board”
- Chapter 3 “e2 studio setup”
- Chapter 4 “Tutorial: Your First RZ MPU Project - Blinky”
- Chapter 6 “FSP Configuration Users Guide”

FSP SC users (for IAR EWARM users) should read the following chapters:

- Chapter 2 ”Set up Renesas Starter kit+ for RZ/T2M CPU Board”
- Chapter 5 “FSP Smart Configurator User Guide”
- Chapter 6 “FSP Configuration Users Guide”

The summary of each chapter is shown below.

- Chapter 2 ”Set up Renesas Starter kit+ for RZ/T2M CPU Board”
  - ➢ Explains how to setup Renesas Starter kit+ for RZ/T2 CPU Board to proceed the tutorials in Chapter 4 and 5.
- Chapter 3 “e2 studio setup”
  - ➢ Explains the setup of e2 studio for utilizing FSP.
- Chapter 4 “Tutorial: Your First RZ MPU Project - Blinky”
  - ➢ Explains the tutorial with minimal steps to create, run, and debug a FSP project by using e2 studio.
- Chapter 5 “FSP Smart Configurator User Guide”
  - ➢ Explains the tutorial with minimal steps to create a FSP project as IAR EWARM project by using the FSP Smart Configurator and to run and debug the created IAR EWARM project.
- Chapter 6 “FSP Configuration Users Guide”
  - ➢ Explains how to create and configure a FSP project in detail.
  - ➢ The explanation is described based on e2 studio, but most of the explanations are applied the FSP smart configurator.

2. Set up Renesas Starter kit+ for RZ/T2M CPU Board

2.1 Obtaining an Renesas Starter Kit+

To develop application with RZ/T2M FSP, start with Renesas Starter Kit+ (RSK).

The Renesas Starter Kit+ for RZ/T2M CPU Board (hereafter, called RSK+RZT2M board) are designed to seamlessly integrate with the e2 studio.

Ordering information, User Manuals, and other related documents for RSK+RZT2M board are available. Please contact Renesas to get them.

2.2 System Configuration

Below is an example of a typical system configuration of RSK+RZT2M board.

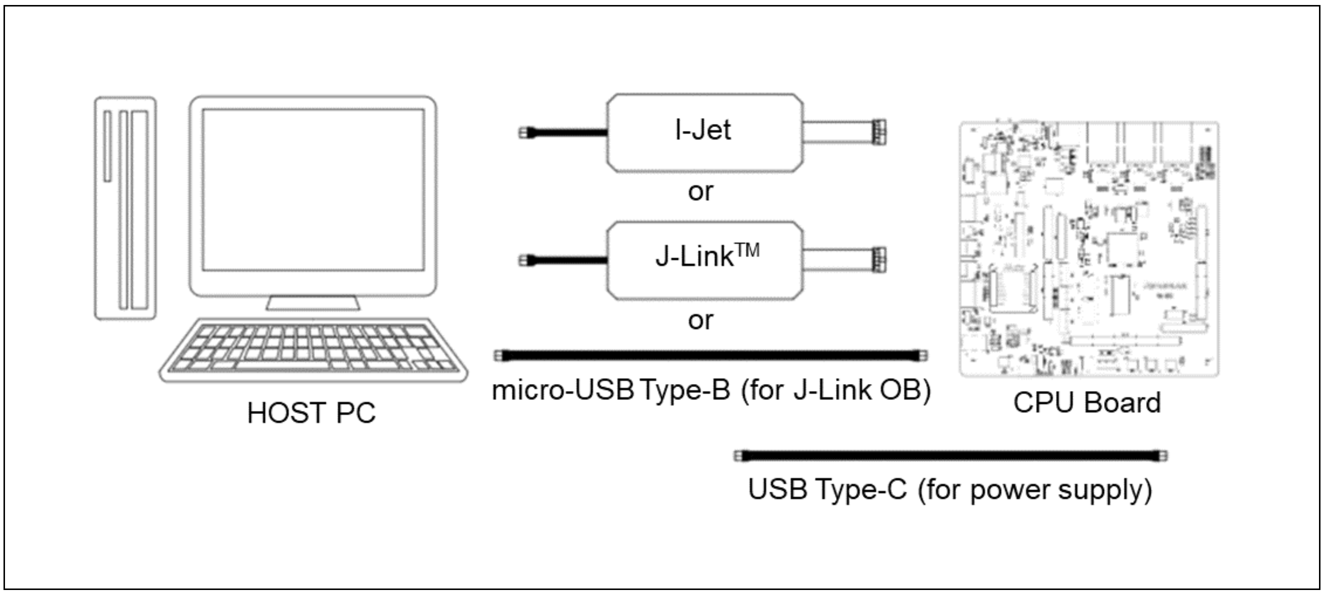

I-Jet
or
J-Link™
or
micro-USB Type-B (for J-Link OB)
HOST PC
CPU Board
USB Type-C (for power supply)

Figure 1: System Configuration Example – RSK+RZT2M Board

For the details, please refer to the related document “Renesas Starter Kit+ User’s Manual”.

2.3 Supported Emulator

2.3.1 SEGGER J-Link

SEGGER J-Link can be used on Renesas e2 studio only for debugging on RZ/T2M device.

Renesas e2 studio supports the following emulators.

- J-Link EDU V11 and later
- J-Link BASE V11 and later
- J-Link PLUS V11 and later
- J-Link WiFi V1 and later
- J-Link ULTRA+ V5 and later
- J-Link PRO V5 and later
- J-Link OB-S124 V1.00

Renesas has tested debugging RZ/T2M with J-Link BASE V11 and J-Link OB-S124.

For the details on SEGGER J-Link, please see SEGGER website.

2.3.2 IAR I-Jet

IAR I-jet can be used on IAR EWARM only for debugging on RZ/T2M device.

For the details on IAR I-jet, please see IAR Systems website.

2.4 Board Setup

2.4.1 Boot MODE

The operation mode settings for the RSK+RZT2M board are as follows.

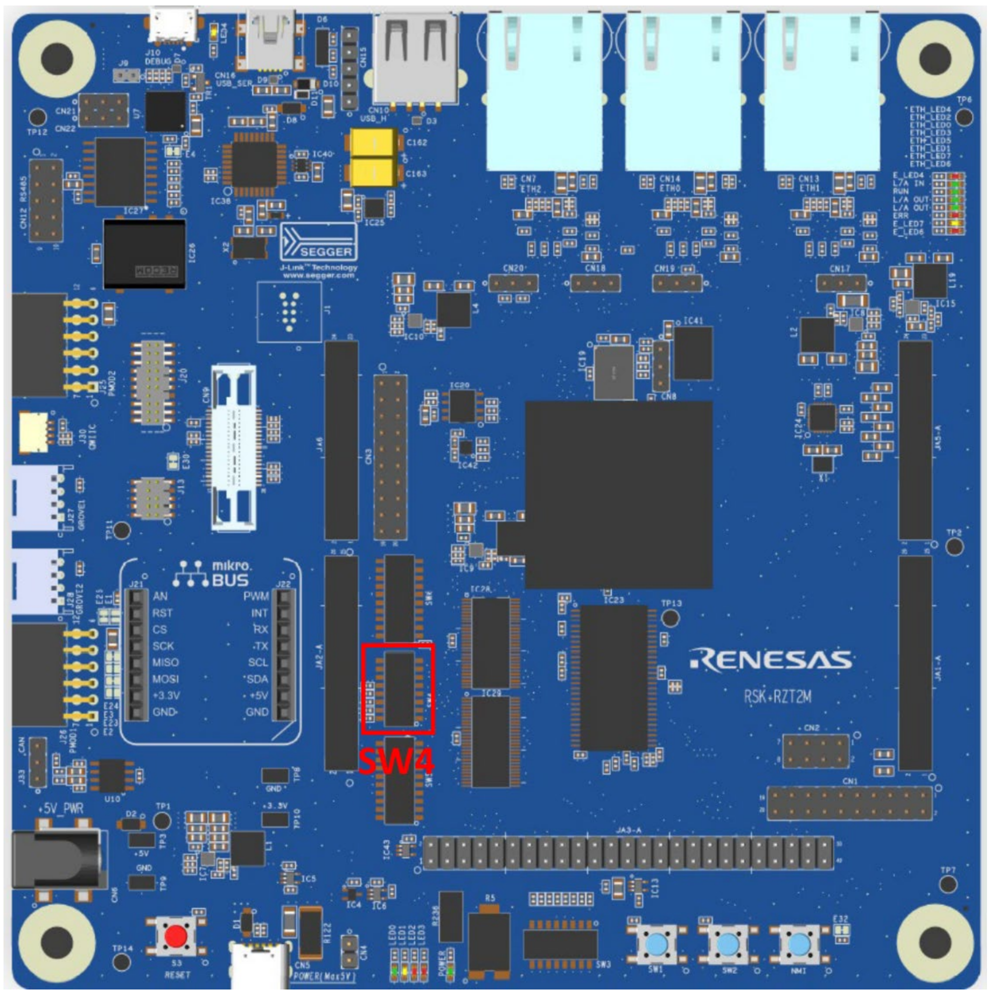


Figure 2: Boot MODE

SW	Setting	Description
SW4.1	ON	16-bit bus boot mode (NOR Flash)
SW4.2	OFF	
SW4.3	ON	
SW4.4	ON	JTAG Authentication by Hash is disabled.
SW4.5	ON	ATCM 0 wait Valid for CPU operating frequency equal to or less than 400MHz.

2.4.2 Debugger connection

If you use JTAG connection with I-Jet or J-Link,

1) Short the jumper pin (J9) for switching the debug connection so that RSK+RZT2M can use the emulator connected to JTAG connector (J20).
2) Connect the emulator (J-Link or I-jet) to a free USB port on your computer.
3) Connect the IAR I-Jet to the RSK+RZT2M board ensuring that it is plugged in to the header "J20".

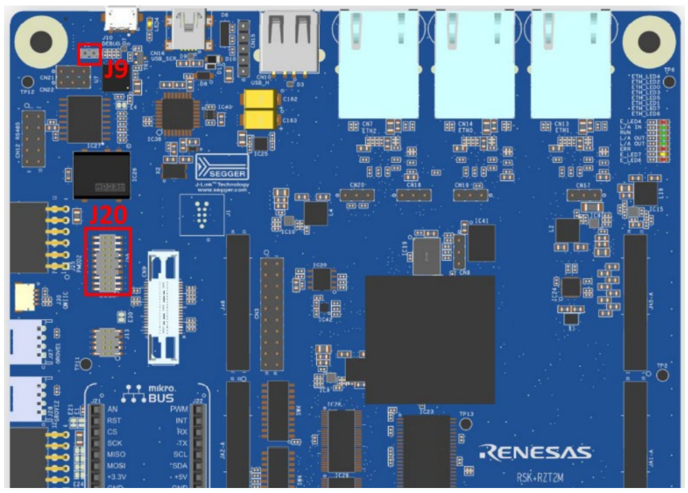


Figure 3: JTAG connection

If you use J-Link OB on RSK+RZT2M,

1) Open the jumper pin (J9) for switching the debug connection so that RSK+RZT2M can use J-Link OB on the board.
2) Connect the micro-USB type-B to J-Link OB USB connector (J10), and then the LED4 is lighted.

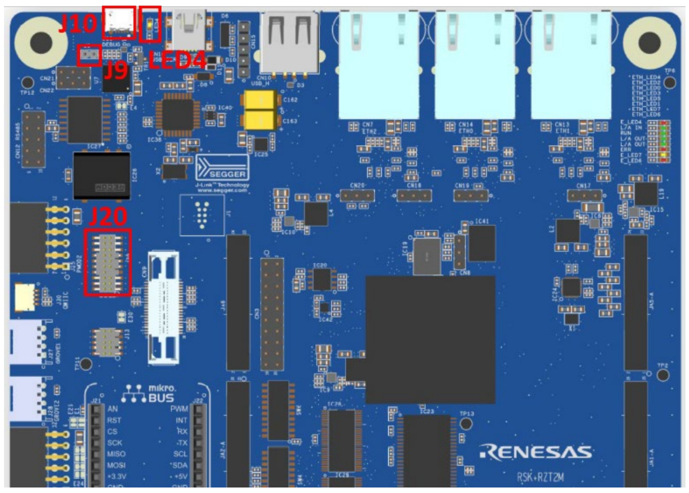


Figure 4: J-Link OB connection

2.4.3 Power Supply

Power is supplied using a USB cable (Type-C) or an AC / DC adapter.

- When using a USB cable (Type-C), connect it to the USB connector "CN5" of the RSK+RZT2M board.
- When connecting the AC / DC adapter, connect it to the USB connector "CN6" of the RSK+RZT2M board.

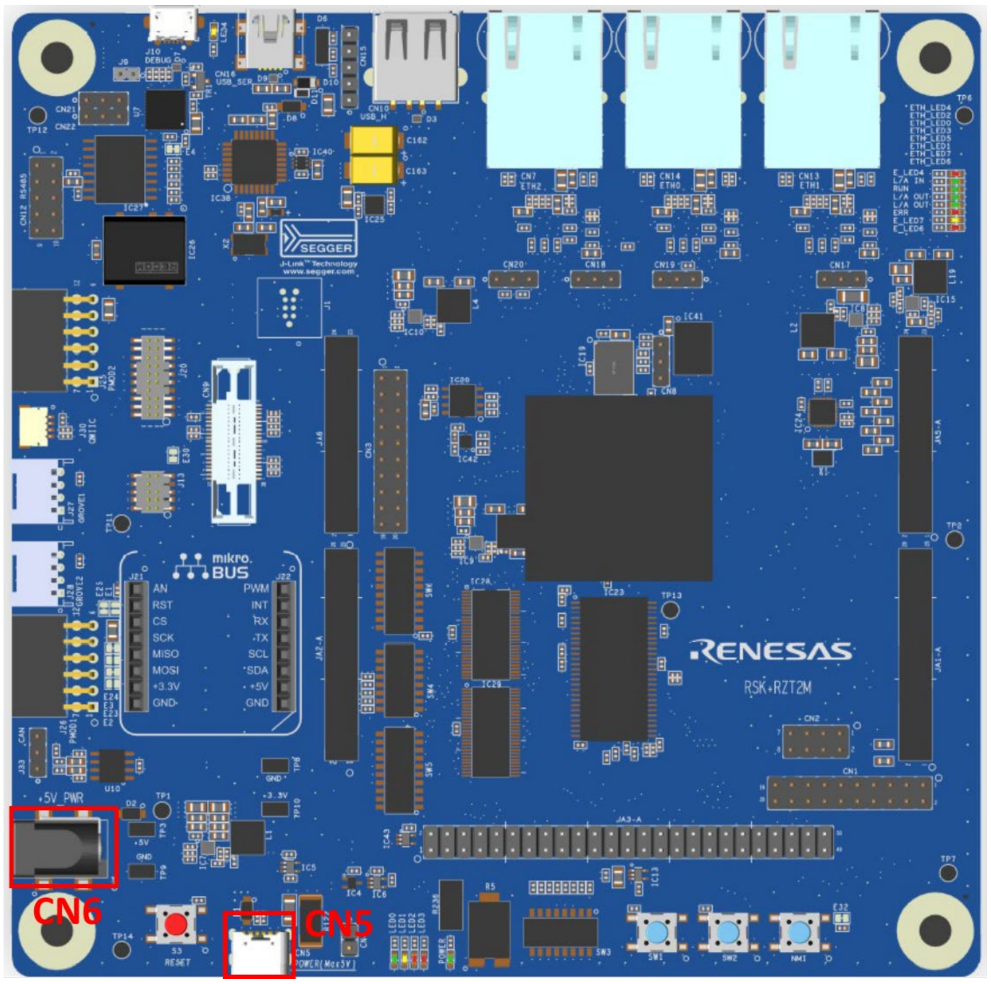


Figure 5: Power Supply

3. e2 studio setup

3.1 What is e2 studio?

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

When developing for RZ MPUs, e2 studio hosts the Renesas Flexible Software Package (FSP). FSP provides a wide range of time saving tools to simplify the selection, configuration, and management of modules and threads, to easily implement complex applications.

3.2 e2 studio Prerequisites

3.2.1 Windows PC Requirements

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

For Windows 64-bit version

- IBM PC/AT compatible
  - Windows® 10 (64-bit version)
  - Windows® 8.1 (64-bit version)
- Memory capacity: We recommend 8 GB or more. At least 4 GB.
- Capacity of hard disk: At least 2 GB of free space.
- Display: Graphics resolution should be at least 1024 x 768, and the mode should display at least 65,536 colors.
- Interface: USB 2.0
- Microsoft Visual C++ 2010 SP1 runtime library *1
- Microsoft Visual C++ 2015-2019 runtime library *1

*1. This software will be installed at the same time as the e^2 studio.

3.2.2 Installing e2 studio, platform installer and FSP package.

Detailed installation instructions for the e2 studio and the FSP are available on the Renesas website. Review the release notes for e2 studio to ensure that the e2 studio version supports the selected FSP version. The starting version of the installer includes all features of the RZ MPUs.

3.2.3 Choosing a Toolchain

The GNU ARM Embedded Toolchain (version 9.3.1.20200408) is required.

If the version of the toolchain has not been installed, please download the toolchain from ARM Developer website, and install it.

3.2.4 Licensing

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

4. Tutorial: Your First RZ MPU Project - Blinky

4.1 Tutorial Blinky

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

4.2 What Does Blinky Do?

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

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

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

4.3 Create a New Project for Blinky

The creation and configuration of an RZ/T C/C++ FSP Project is the first step in the creation of an application. The base RZ/T2M pack includes a pre-written Blinky example application.

Follow these steps to create an RZ MPU project:

1. In e2 studio, click **File** > **New** > **C/C++ Project**.

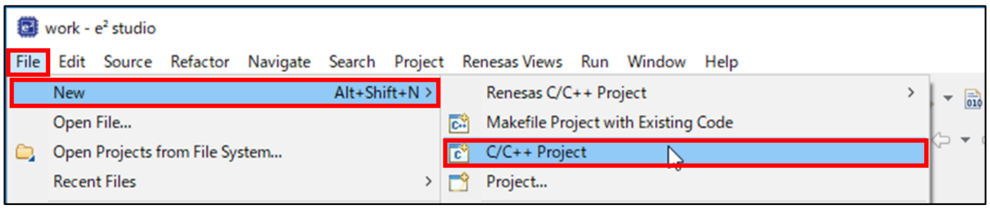
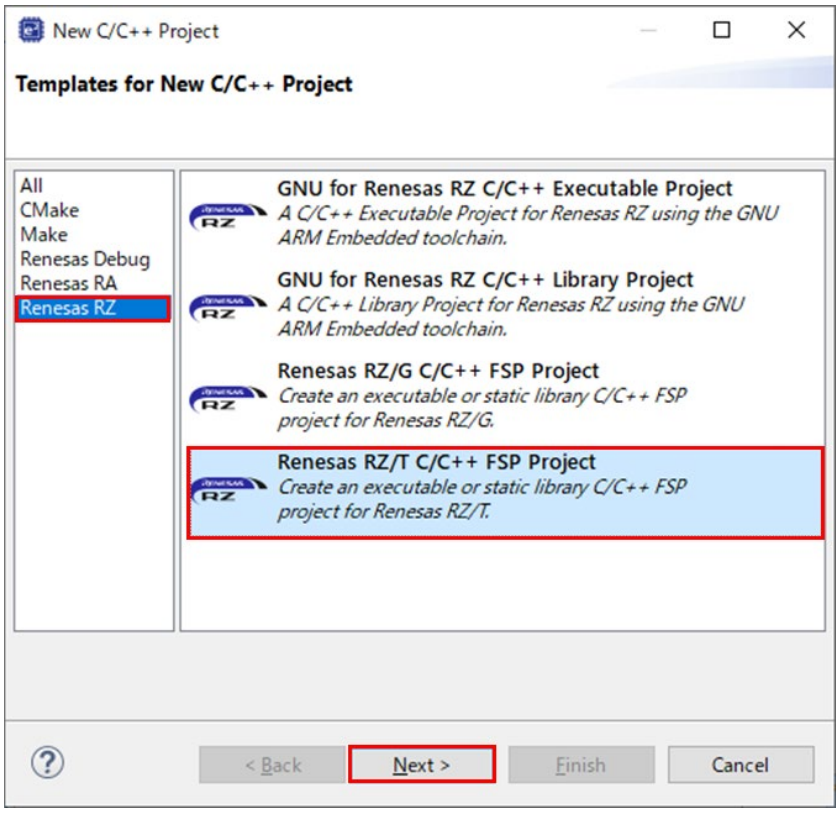


Figure 6: New C/C++ Project

2. Select Renesas RZ > Renesas RZ/T C/C++ FSP Project and Click Next.


New C/C++ Project

Templates for New C/C++ Project

All
CMake
Make
Renesas Debug
Renesas RA
Renesas RZ

GNU for Renesas RZ C/C++ Executable Project
A C/C++ Executable Project for Renesas RZ using the GNU ARM Embedded toolchain.

GNU for Renesas RZ C/C++ Library Project
A C/C++ Library Project for Renesas RZ using the GNU ARM Embedded toolchain.

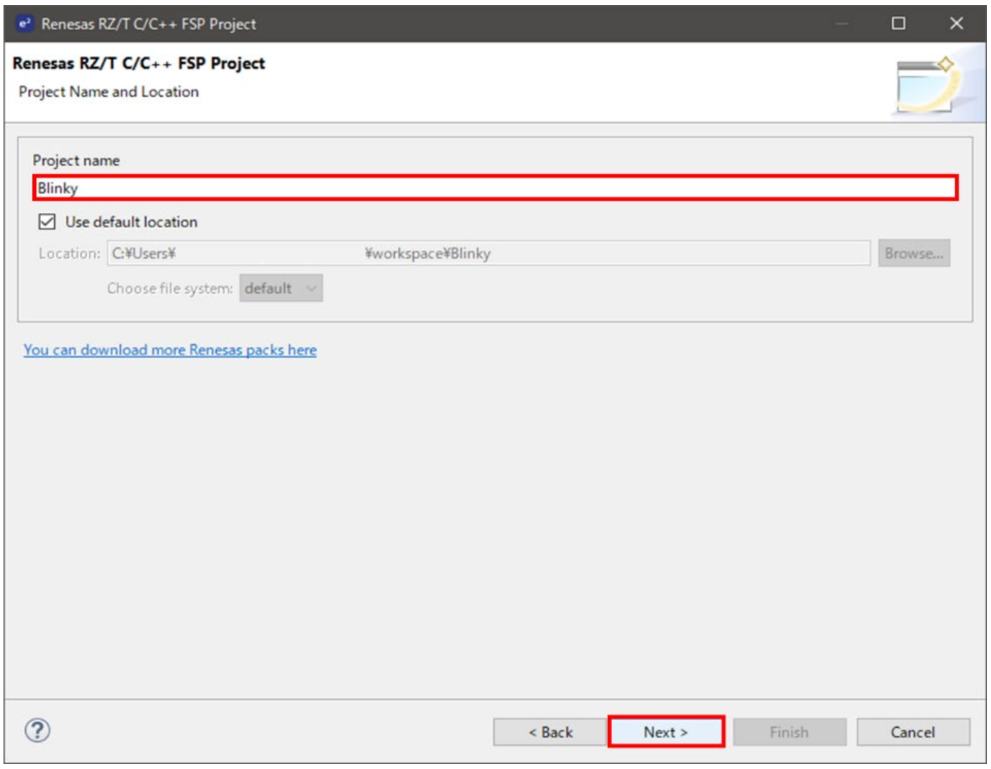
Renesas RZ/G C/C++ FSP Project
Create an executable or static library C/C++ FSP project for Renesas RZ/G.

Renesas RZ/T C/C++ FSP Project
Create an executable or static library C/C++ FSP project for Renesas RZ/T.

Figure 7: Renesas RZ/T C/C++ FSP Project

(Continued on next page)

3. Assign a name to this new project. Blinky is a good name to use for this tutorial.
4. Click **Next**. The Project Configuration window shows your selection.



Renesas RZ/T C/C++ FSP Project

Renesas RZ/T C/C++ FSP Project

Project Name and Location

Project name: Blinky

☑ Use default location

Location: C:¥Users¥ ¥workspace¥Blinky

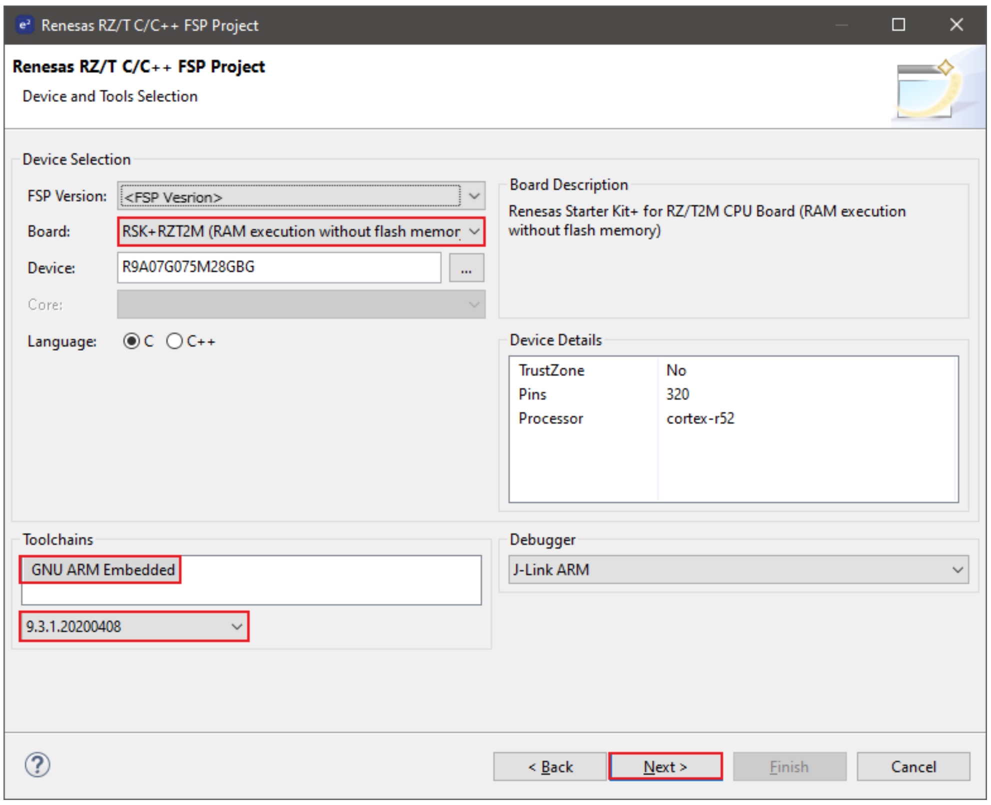
Choose file system: default

You can download more Renesas packs here

Figure 8 : e2 studio Project Configuration window (part 1)

(Continued on next page)

5. Select the board support package by selecting the name of your board from the drop-down list.
   In this tutorial, please select the **RSK+RZT2M (RAM execution without flash memory)**.
6. Select **GNU ARM Embedded** in Toolchains and version is **9.3.1.20200408** as its toolchain version, and click **Next**.
   - If there is NOT the 9.3.1.20200408 version of GNU ARM Embedded Toolchain, please download the version of the toolchain from ARM Developer website and install it.


Renesas RZ/T C/C++ FSP Project

Renesas RZ/T C/C++ FSP Project
Device and Tools Selection

Device Selection
FSP Version: <FSP Vesrion>
Board: RSK+RZT2M (RAM execution without flash memor
Device: R9A07G075M28GBG
Core:
Language: [✓ C] C++

Board Description
Renesas Starter Kit+ for RZ/T2M CPU Board (RAM execution without flash memory)

Device Details

TrustZone	No
Pins	320
Processor	cortex-r52

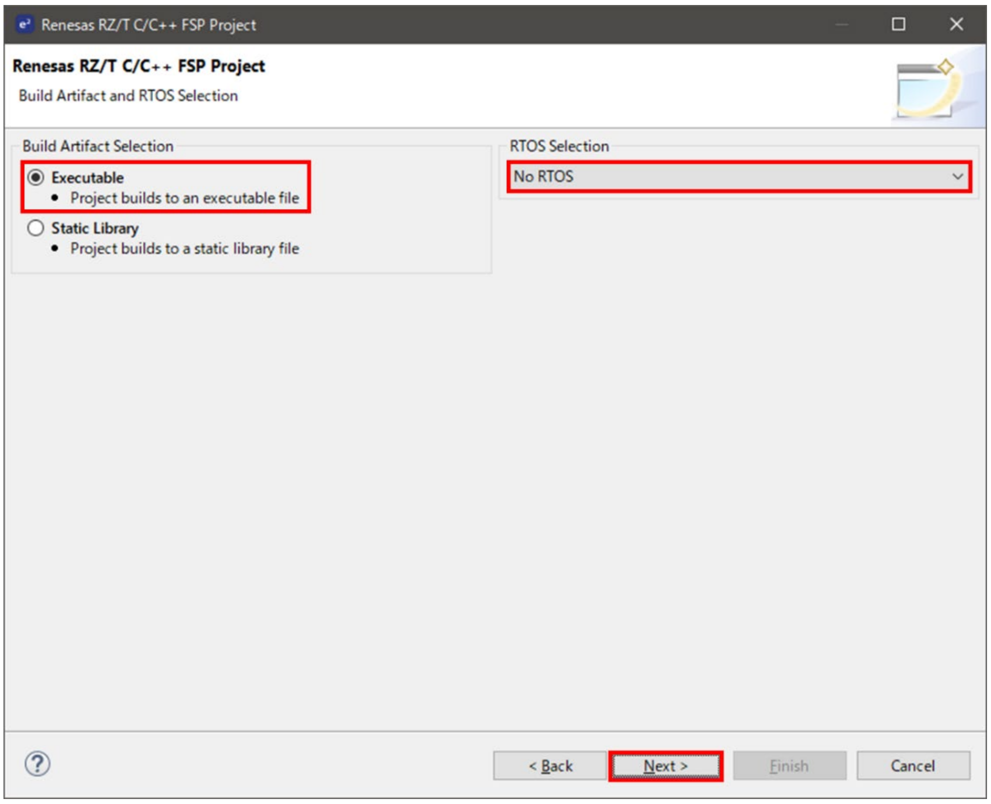
Toolchains
GNU ARM Embedded
9.3.1.20200408

Debugger
J-Link ARM

Figure 9 : e2 studio Project Configuration window (part 2)

(Continued on next page)

7. Select the **build artifact** and RTOS.



Renesas RZ/T C/C++ FSP Project

Build Artifact and RTOS Selection

Build Artifact Selection

(●) Executable
- Project builds to an executable file

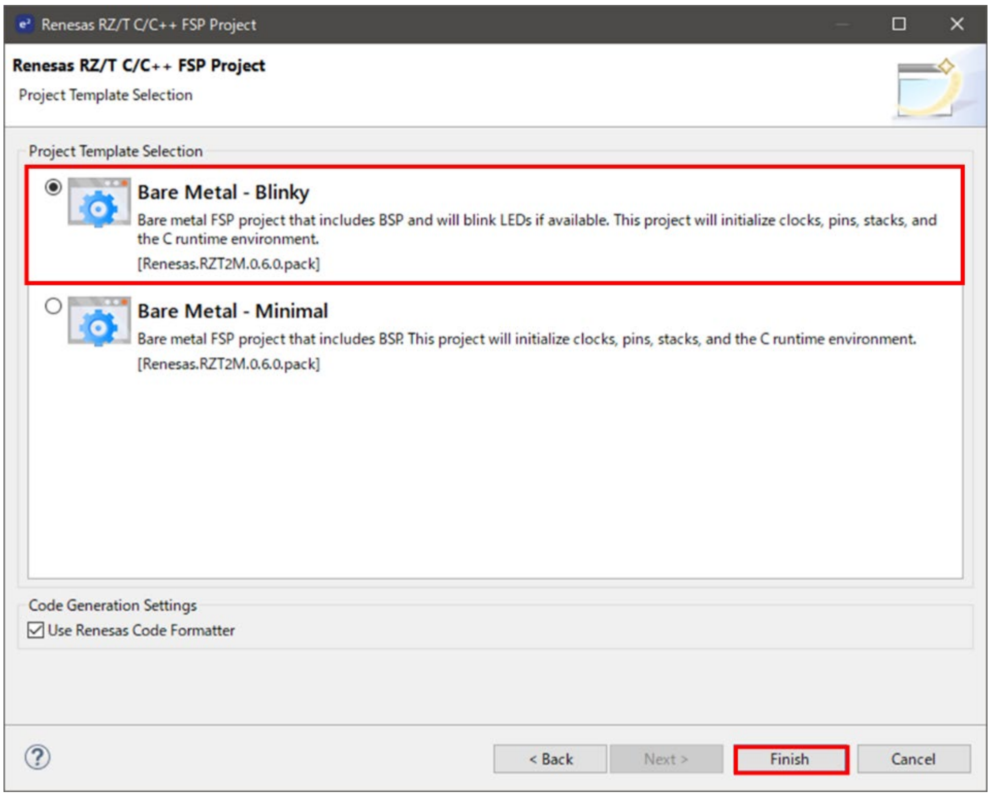
( ) Static Library
- Project builds to a static library file

RTOS Selection: No RTOS

Figure 10 : e2 studio Project Configuration window (part 3)

(Continued on next page)

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


Renesas RZ/T C/C++ FSP Project

Renesas RZ/T C/C++ FSP Project

Project Template Selection

Project Template Selection

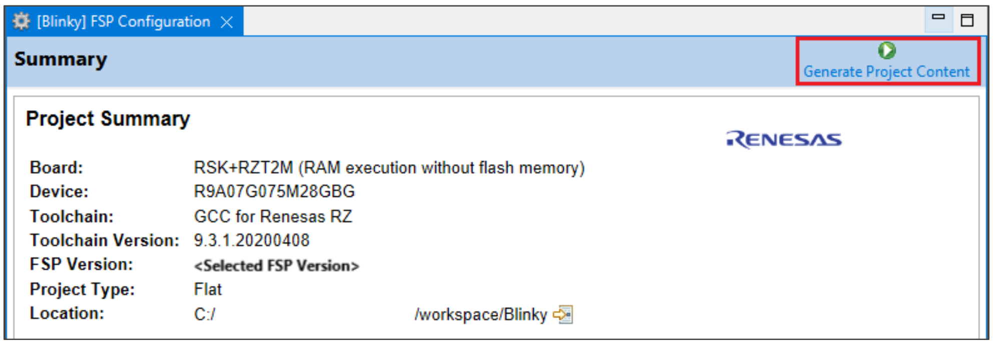
(●) **Bare Metal - Blinky**
Bare metal FSP project that includes BSP and will blink LEDs if available. This project will initialize clocks, pins, stacks, and the C runtime environment.
[Renesas.RZT2M.0.6.0.pack]

( ) **Bare Metal - Minimal**
Bare metal FSP project that includes BSP. This project will initialize clocks, pins, stacks, and the C runtime environment.
[Renesas.RZT2M.0.6.0.pack]

Code Generation Settings
☑ Use Renesas Code Formatter

Figure 11 : e2 studio Project Configuration window (part 4)

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



Project Summary

Board:	RSK+RZT2M (RAM execution without flash memory)
Device:	R9A07G075M28GBG
Toolchain:	GCC for Renesas RZ
Toolchain Version:	9.3.1.20200408
FSP Version:	<Selected FSP Version>
Project Type:	Flat
Location:	C:/ /workspace/Blinky

Figure 12 : e2 studio Project Configuration tab

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

4.3.1 Details about the Blinky Configuration

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

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

4.3.2 Configuring the Blinky Clocks

By selecting the Blinky template, the clocks are configured by e2 studio for the Blinky application.

The clock configuration tab (see 6.3.3 Configuring Clocks) shows the Blinky clock configuration. The Blinky clock configuration is stored in the BSP clock configuration file.

4.3.3 Configuring the Blinky Pins

By selecting the Blinky template, the GPIO pins used to toggle two LED0~1 are configured by e2 studio for the Blinky application.

The pin configuration tab shows the pin configuration for the Blinky application (see 6.3.4 Configuring Pins). The Blinky pin configuration is stored in the BSP configuration file.

4.3.4 Configuring the Parameters for Blinky Components

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

- r_ioport

To see the configuration parameters for any of the components, check the Properties tab in the HAL window for the respective drivers (see 6.5 Adding and Configuring HAL Drivers).

4.3.5 Where is main()?

The main function is located in < project >/rzt_gen/main.c. It is one of the files that are generated during the project creation stage and only contains a call to hal_entry(). For more information on generated files, see 6.5 Adding and Configuring HAL Drivers.

4.3.6 Blinky Example Code

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

The application performs the following steps:

1. Get the LED information for the selected board by **bsp_leds_t** structure.
2. Initialize output level information (**pin_level** variable) for LED pin to LOW.
3. Use **R_BSP_PinWrite ((bsp_io_port_pin_t) pin, pin_level)** to set the output level information to the LED pin.
4. Toggle the output level information for LED pin.
5. **R_BSP_SoftwareDelay(delay, bsp_delay_units)** waits for a certain period of time. Then run #3 again.

4.4 Build the Blinky Project

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

There are three ways to build a project:

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

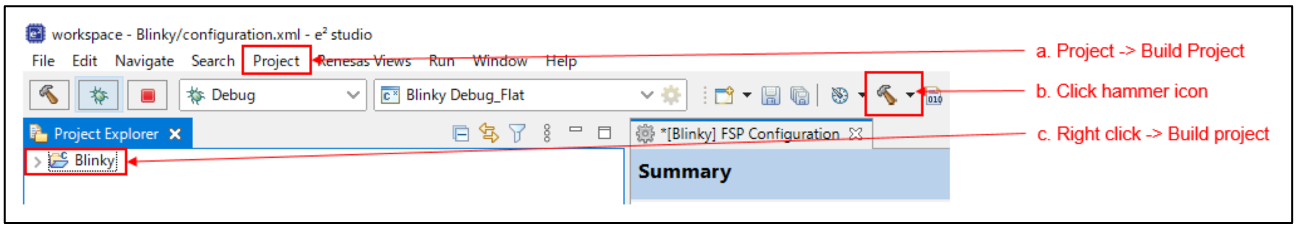
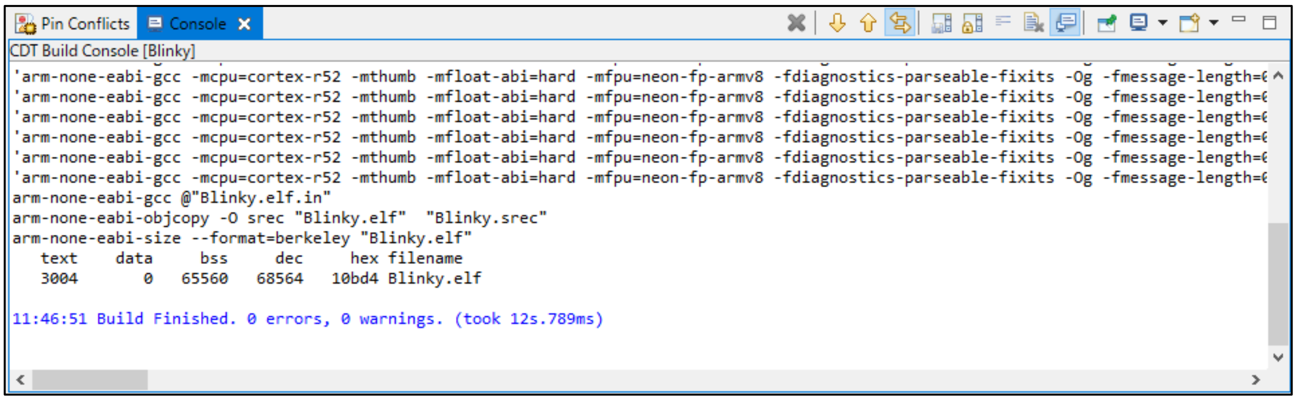

workspace - Blinky/configuration.xml - e² studio
File Edit Navigate Search Project Renesas Views Run Window Help
Debug
Blinky Debug_Flat
Project Explorer
Blinky
*[Blinky] FSP Configuration
Summary
a. Project -> Build Project
b. Click hammer icon
c. Right click -> Build project

Figure 13 : e2 studio Project Explorer window

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



```
'arm-none-eabi-gcc -mcpu=cortex-r52 -mthumb -mfloat-abi=hard -mfpu=neon-fp-armv8 -fdiagnostics-parseable-fixits -Og -fmessage-length=
'arm-none-eabi-gcc -mcpu=cortex-r52 -mthumb -mfloat-abi=hard -mfpu=neon-fp-armv8 -fdiagnostics-parseable-fixits -Og -fmessage-length=
'arm-none-eabi-gcc -mcpu=cortex-r52 -mthumb -mfloat-abi=hard -mfpu=neon-fp-armv8 -fdiagnostics-parseable-fixits -Og -fmessage-length=
'arm-none-eabi-gcc -mcpu=cortex-r52 -mthumb -mfloat-abi=hard -mfpu=neon-fp-armv8 -fdiagnostics-parseable-fixits -Og -fmessage-length=
'arm-none-eabi-gcc -mcpu=cortex-r52 -mthumb -mfloat-abi=hard -mfpu=neon-fp-armv8 -fdiagnostics-parseable-fixits -Og -fmessage-length=
'arm-none-eabi-gcc -mcpu=cortex-r52 -mthumb -mfloat-abi=hard -mfpu=neon-fp-armv8 -fdiagnostics-parseable-fixits -Og -fmessage-length=
arm-none-eabi-gcc @"Blinky.elf.in"
arm-none-eabi-objcopy -O srec "Blinky.elf"  "Blinky.srec"
arm-none-eabi-size --format=berkeley "Blinky.elf"
   text    data     bss     dec     hex filename
   3004       0   65560   68564   10bd4 Blinky.elf

11:46:51 Build Finished. 0 errors, 0 warnings. (took 12s.789ms)
```

Figure 14 : e2 studio Project Build console

4.5 Debug the Blinky Project

4.5.1 Debug prerequisites

To debug the project on a board, you need the followings:

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

Applications run from the internal ram of your microprocessor. To run or debug the application, the application must first be programmed to ram by JTAG debugger.

RSK+RZT2M board has an JTAG header and requires an external JTAG debugger to the header.

4.5.2 Debug steps

To debug the Blinky application, follow these steps:

1. Configure the debugger for your project by clicking **Run** > **Debugger Configurations** ... or by selecting the drop-down menu next to the bug icon and selecting **Debugger Configuration**s ...

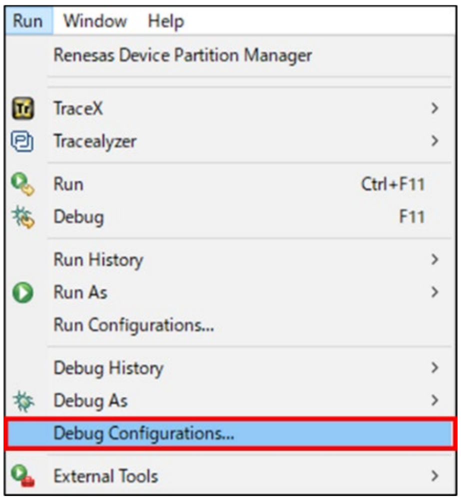


Figure 15 : e2 studio Debug icon

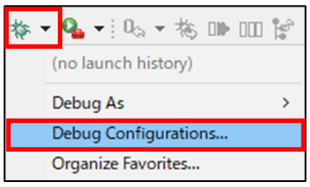
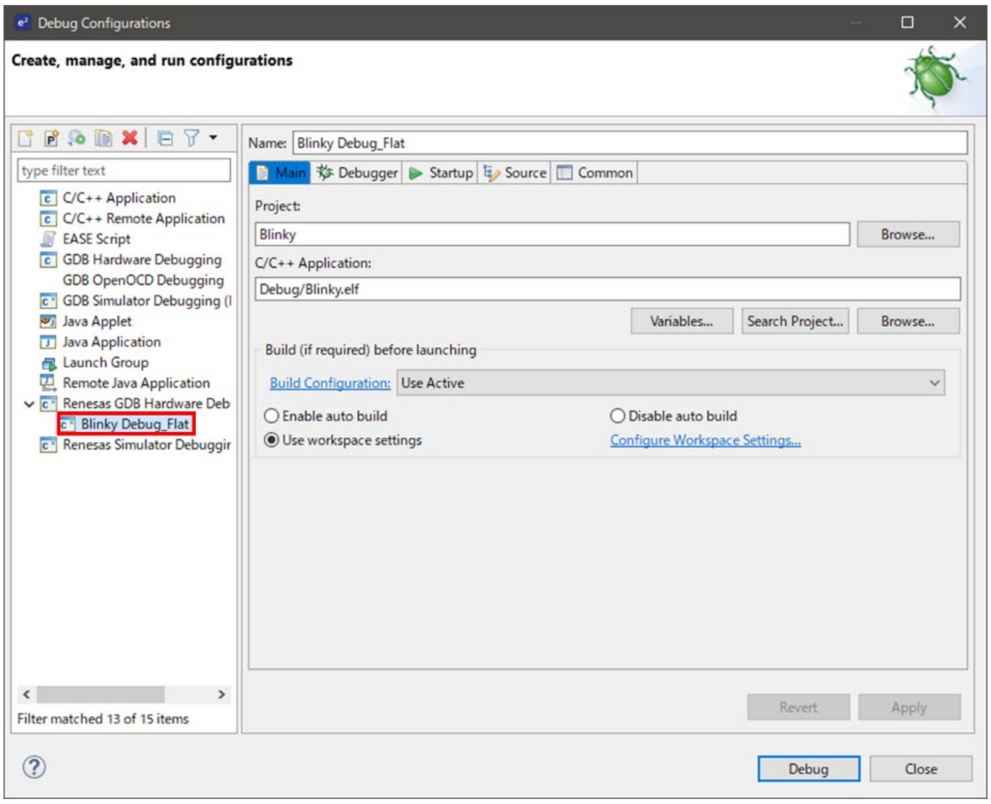


Figure 16 : e2 studio Debugger Configurations selection option

(Continued on next page)

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

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



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Remote Application
- EASE Script
- GDB Hardware Debugging
- GDB OpenOCD Debugging
- GDB Simulator Debugging (I
- Java Applet
- Java Application
- Launch Group
- Remote Java Application
- Renesas GDB Hardware Deb
  - Blinky Debug_Flat
- Renesas Simulator Debuggir

Filter matched 13 of 15 items

Name: Blinky Debug_Flat

Main | Debugger | Startup | Source | Common

Project: Blinky

C/C++ Application: Debug/Blinky.elf

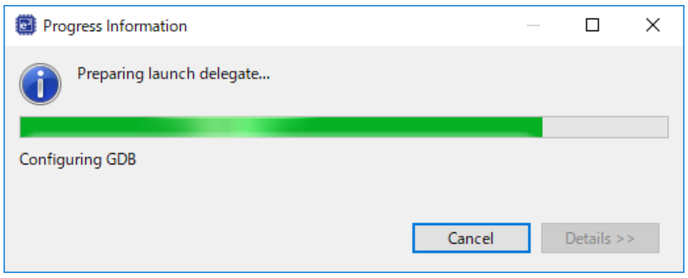
Build (if required) before launching

Build Configuration: Use Active

Enable auto build | Disable auto build

[✓ Use workspace settings] | Configure Workspace Settings...

Figure 17 : e2 studio Debugger Configurations window with Blinky project (1)



Progress Information

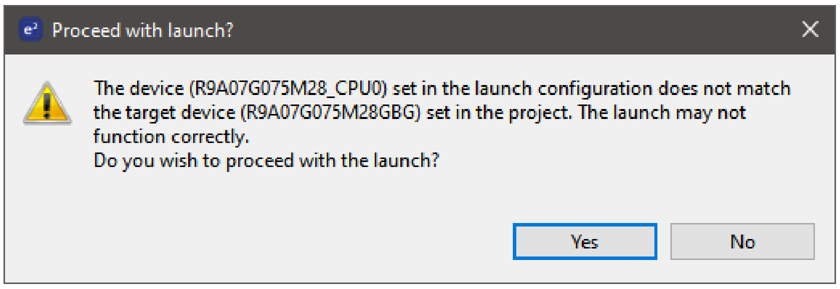
Preparing launch delegate...

Configuring GDB

Figure 18: Start Debugging

Warning:

When the following dialog box is shown, please click **Yes** to proceed the launch.


Proceed with launch?

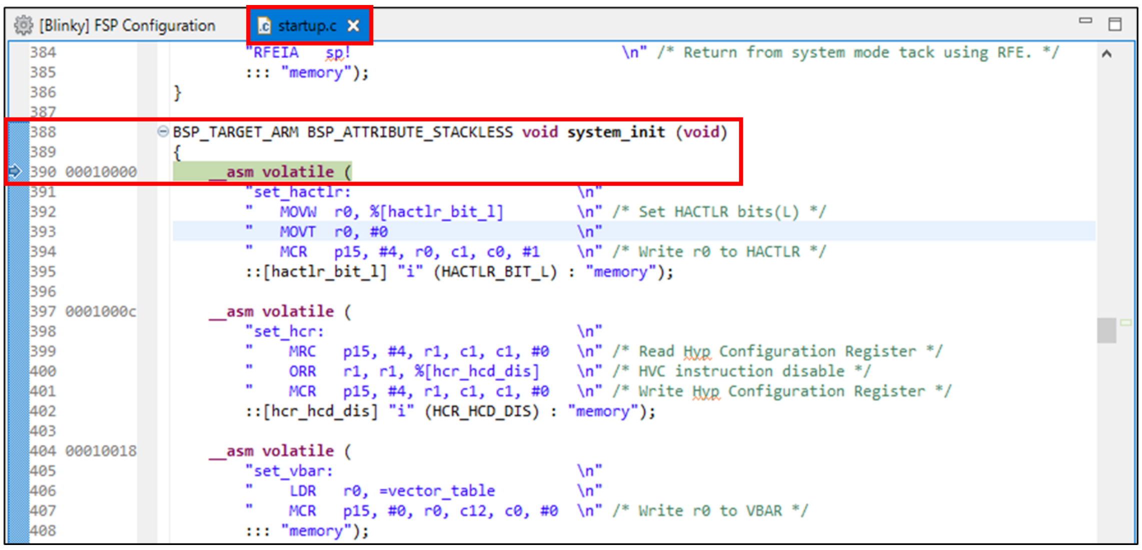
The device (R9A07G075M28_CPU0) set in the launch configuration does not match the target device (R9A07G075M28GBG) set in the project. The launch may not function correctly.
Do you wish to proceed with the launch?

Figure 19: e2 studio debug issue target device is not matched

4.5.3 Details about the Debug Process

In debug mode, e2 studio executes the following tasks:

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



```
                  "RFEIA   sp!                       \n" /* Return from system mode tack using RFE. */
                  ::: "memory");
          }

          BSP_TARGET_ARM BSP_ATTRIBUTE_STACKLESS void system_init (void)
          {
00010000      __asm volatile (
                  "set_hactlr:                       \n"
                  "   MOVW  r0, %[hactlr_bit_l]      \n" /* Set HACTLR bits(L) */
                  "   MOVT  r0, #0                   \n"
                  "   MCR   p15, #4, r0, c1, c0, #1  \n" /* Write r0 to HACTLR */
                  ::[hactlr_bit_l] "i" (HACTLR_BIT_L) : "memory");

0001000c      __asm volatile (
                  "set_hcr:                          \n"
                  "    MRC   p15, #4, r1, c1, c1, #0 \n" /* Read Hyp Configuration Register */
                  "    ORR   r1, r1, %[hcr_hcd_dis]  \n" /* HVC instruction disable */
                  "    MCR   p15, #4, r1, c1, c1, #0 \n" /* Write Hyp Configuration Register */
                  ::[hcr_hcd_dis] "i" (HCR_HCD_DIS) : "memory");

00010018      __asm volatile (
                  "set_vbar:                         \n"
                  "    LDR   r0, =vector_table       \n"
                  "    MCR   p15, #0, r0, c12, c0, #0 \n" /* Write r0 to VBAR */
                  ::: "memory");
```

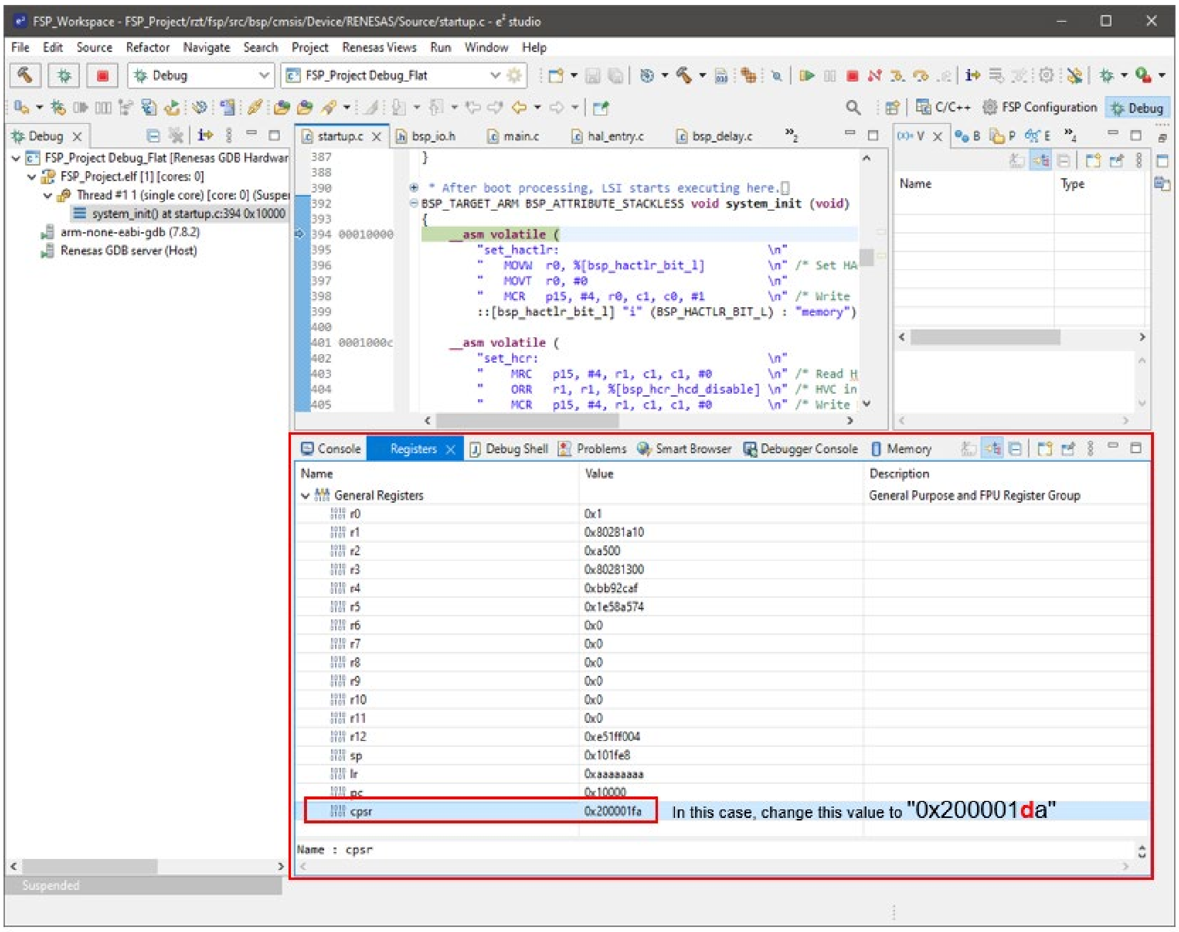
Figure 20 : e2 studio Debugger memory window

4.5.4 Change CPSR register value

Before running the loaded program, please change the CPSR register of CR52 general register on **Registers** tabs.

- Please change the “T” register bit (bit 5 in CPSR register), which is Thumb execution state bit, from “1” to “0” to switch the instruction mode from “Thumb” to “Arm”.
    - For example, when the register value is “0x000001fa”, set it to “0x000001da”.

Please note that the program halts at Default_Handler() when running if the value of “T” bit in CPSR register is not changed.



```
}

 * After boot processing, LSI starts executing here.
BSP_TARGET_ARM BSP_ATTRIBUTE_STACKLESS void system_init (void)
{
    __asm volatile (
        "set_hactlr:                                   \n"
        "    MOVW  r0, %[bsp_hactlr_bit_l]             \n" /* Set HA
        "    MOVT  r0, #0                              \n"
        "    MCR   p15, #4, r0, c1, c0, #1             \n" /* Write
        ::[bsp_hactlr_bit_l] "i" (BSP_HACTLR_BIT_L) : "memory")

    __asm volatile (
        "set_hcr:                                      \n"
        "    MRC   p15, #4, r1, c1, c1, #0             \n" /* Read H
        "    ORR   r1, r1, %[bsp_hcr_hcd_disable]      \n" /* HVC in
        "    MCR   p15, #4, r1, c1, c1, #0             \n" /* Write
```

Name	Value	Description
General Registers		General Purpose and FPU Register Group
r0	0x1	
r1	0x80281a10	
r2	0xa500	
r3	0x80281300	
r4	0xbb92caf	
r5	0x1e58a574	
r6	0x0	
r7	0x0	
r8	0x0	
r9	0x0	
r10	0x0	
r11	0x0	
r12	0xe51ff004	
sp	0x101fe8	
lr	0xaaaaaaaa	
pc	0x10000	
cpsr	0x200001fa	In this case, change this value to "0x200001da"

```
Name : cpsr
```

Figure 21 : CPSR register of CR52 generic register on Registers tab

4.6 Run the Blinky Project

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

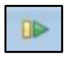


Figure 22 : e2 studio Debugger Play icon

The LEDs on the board LED 0~1 should now be blinking.

5. FSP Smart Configurator User Guide

5.1 What is FSP Smart Configurator?

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

For creating RZ/T2M project, the FSP SC can currently be used with

- IAR EWARM with IAR toolchain for Arm

Projects can be configured, and the project content generated in the same way as in e2 studio. Please refer to 5.2 Configuring a Project section for more details.

5.2 Tutorial Blinky

The goal of this tutorial is to quickly get acquainted with the Flexible Platform by moving through the steps of creating a simple application using FSP SC and 3rd party IDE and running that application on an RZ MPU board.

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

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

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

5.3 Using Smart Configurator with IAR EWARM

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

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

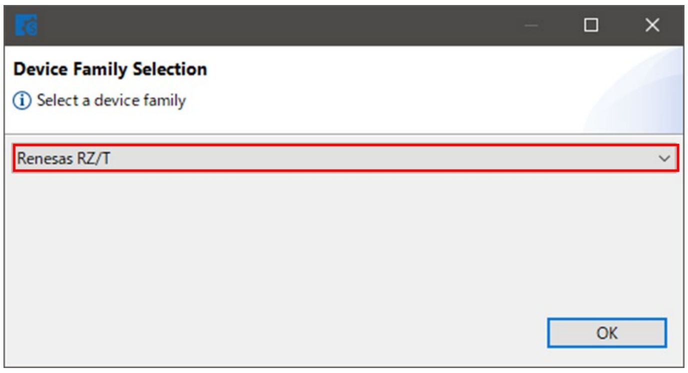
5.3.1 Prerequisites

- IAR EWARM installed and licensed.
  - ➢ Please refer to IAR systems website regarding IAR EWARM.
- FSP SC and FSP Pack installed.
  - ➢ Please refer to Renesas website regarding to FSP SC and FSP Pack.

5.3.2 Create new project

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

1. Start the FSP Smart Configurator.
2. Select "Renesas RZ/T" device family (if the device family selection window is shown.)
3. Enter a project folder and project name.

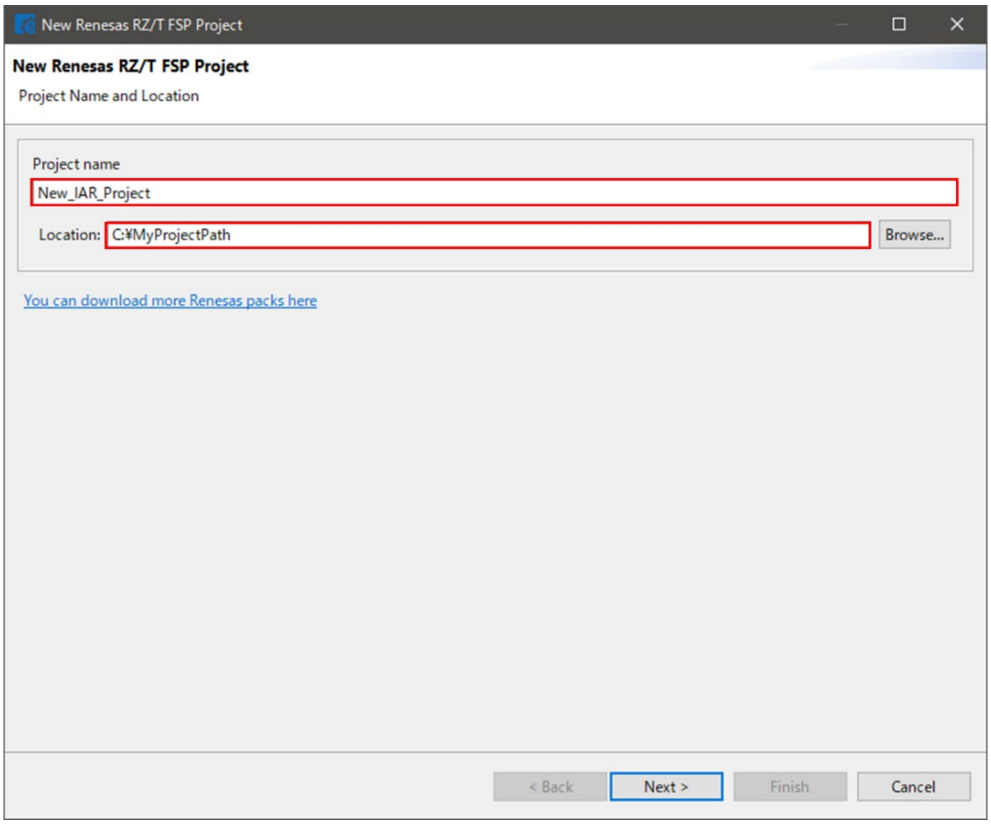


Device Family Selection

Select a device family

Renesas RZ/T

Figure 23 : FSP SC device family selection



New Renesas RZ/T FSP Project

Project Name and Location

Project name: New_IAR_Project

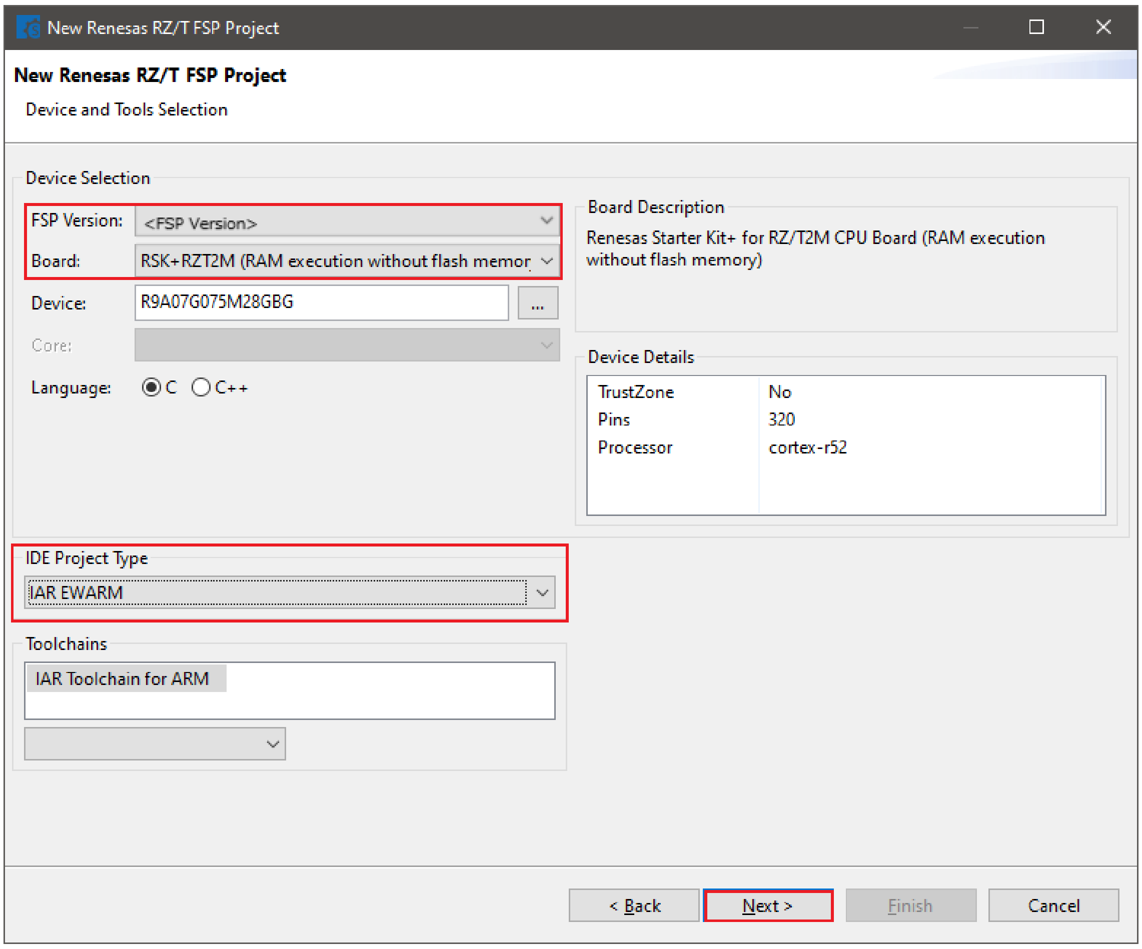
Location: C:¥MyProjectPath

You can download more Renesas packs here

Figure 24 : FSP SC project settings

(Continued on next page)

4. Select the **FSP** version.
5. Select the **Board** for your application.
   - ➢ You can select an existing RZ MPU Evaluation Kit or select Custom User Board for any of the RZ MPU devices with your own BSP definition.
   - ➢ Here, select **RSK+RZT2M (RAM execution without flash memory)** for proceeding the following tutorial.
6. Select **IDE Project Type**.
   - ➢ Here, select **IAR EWARM**.
     - ✧ As the Toolchain, IAR Toolchain for ARM is preselected.
7. Click **Next**.


New Renesas RZ/T FSP Project

New Renesas RZ/T FSP Project

Device and Tools Selection

Device Selection

FSP Version: <FSP Version>

Board: RSK+RZT2M (RAM execution without flash memor

Device: R9A07G075M28GBG

Core:

Language: [✓ C] C++

Board Description

Renesas Starter Kit+ for RZ/T2M CPU Board (RAM execution without flash memory)

Device Details

TrustZone	No
Pins	320
Processor	cortex-r52

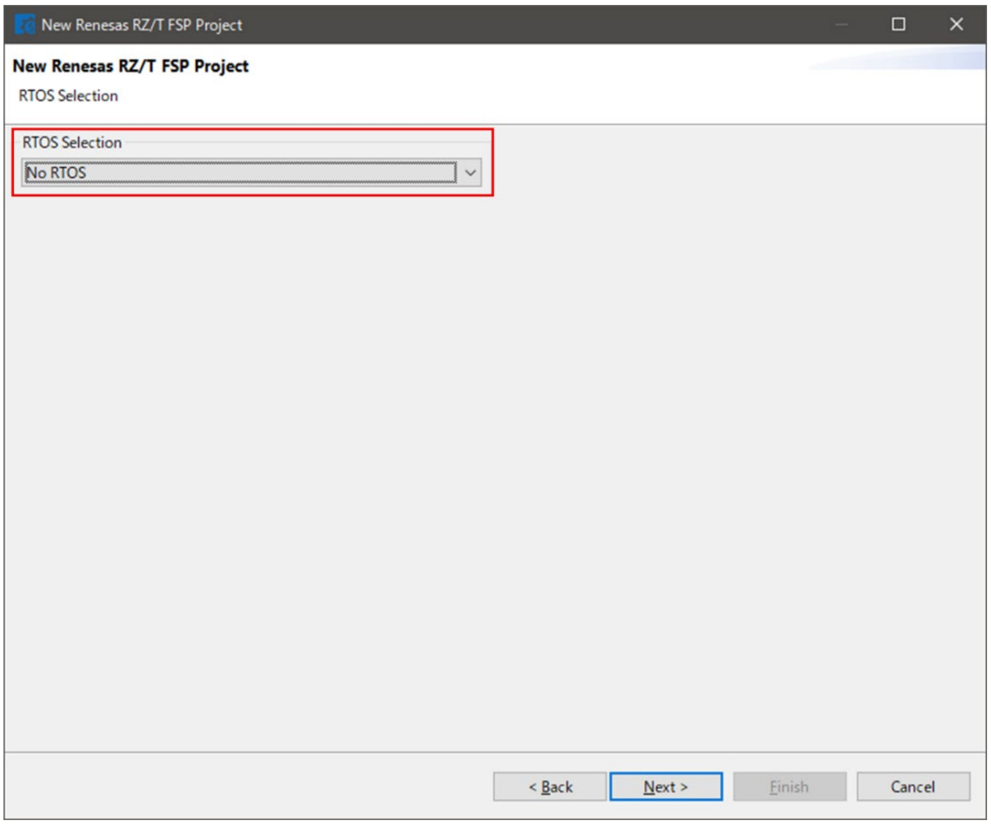
IDE Project Type: IAR EWARM

Toolchains: IAR Toolchain for ARM

Figure 25 : Target device and IDE selections

(Continued on next page)

8. Select **RTOS**.
   - Here, select **No RTOS** for proceeding the following tutorial.
9. Click **Next**.


New Renesas RZ/T FSP Project

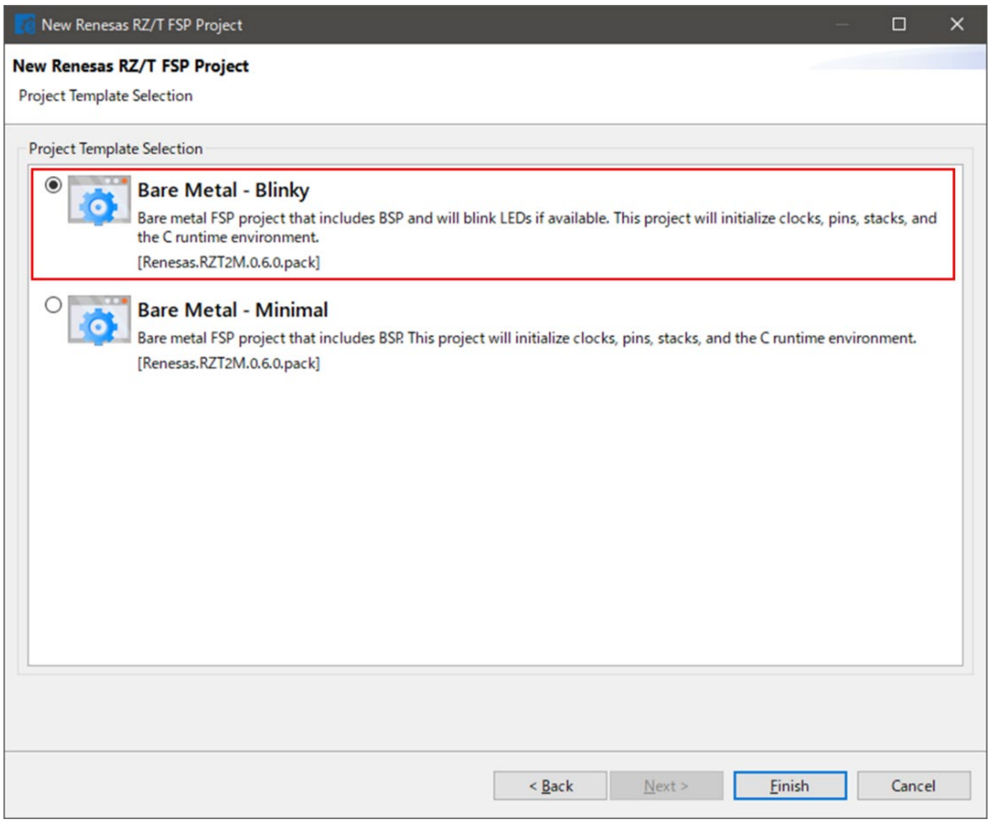
RTOS Selection

RTOS Selection: No RTOS

Figure 26 : RTOS Selection

(Continued on next page)

10. Select a **project template** from the list of available templates.
    - ➢ By default, this screen shows the templates that are included in your current RZ/T MPU Pack.
    - ➢ Here, select Bare Metal - Blinky for proceeding the following tutorial.
        - ✧ If you want to develop your own application, select the basic template for your board, **Bare Metal - Minimal**.
11. Click **Finish.**


New Renesas RZ/T FSP Project
New Renesas RZ/T FSP Project
Project Template Selection
Project Template Selection

(●) Bare Metal - Blinky
Bare metal FSP project that includes BSP and will blink LEDs if available. This project will initialize clocks, pins, stacks, and the C runtime environment.
[Renesas.RZT2M.0.6.0.pack]

( ) Bare Metal - Minimal
Bare metal FSP project that includes BSP. This project will initialize clocks, pins, stacks, and the C runtime environment.
[Renesas.RZT2M.0.6.0.pack]

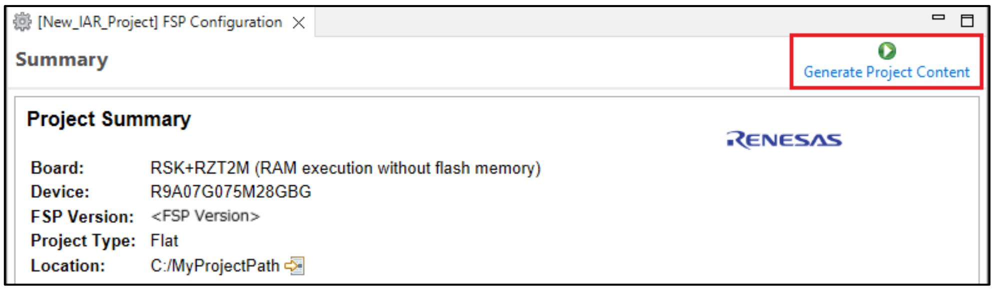
< Back
Next >
Finish
Cancel

Figure 27 : Template Selection

(Continued on next page)

12. **Configure** the FSP configuration by referring to Chapter 6.3 “Configuring a Project”.
    - Here, skips this configuration step for proceeding the following tutorial.

13. On completion of the FSP configuration, press **Generate Project Content**.



Project Summary

Board:	RSK+RZT2M (RAM execution without flash memory)
Device:	R9A07G075M28GBG
FSP Version:	<FSP Version>
Project Type:	Flat
Location:	C:/MyProjectPath

Figure 28 : FSP Project Configuration and Generation

A new IAR EWARM project file will be generated in the project path.

5.3.3 Build the project

Click on **Project** -> **Make** from menu bar or **Make** button on tool bar to build.

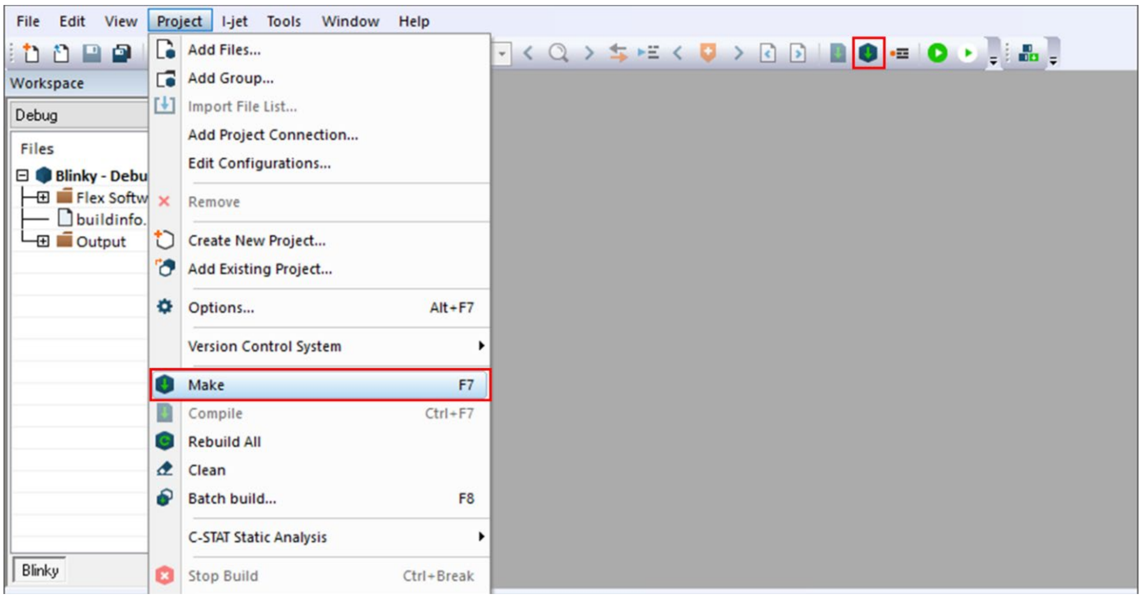
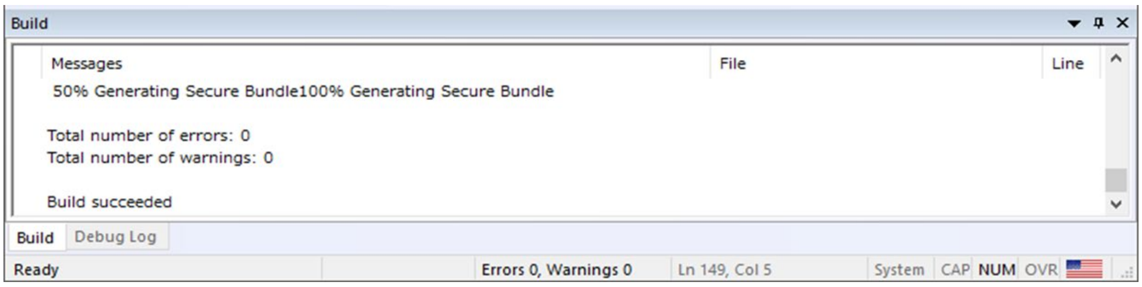


Figure 29 : Make button



```
50% Generating Secure Bundle100% Generating Secure Bundle

Total number of errors: 0
Total number of warnings: 0

Build succeeded
```

Figure 30 : Build Message Console

Once the build is completed, the build message is displayed in the Build Console window that displays compilation target files and the number of error/warnings.

5.3.4 Download & Debug the Project

Click on **Project** -> **Download and debug** from menu bar or **Download and Debug** button on tool bar to download and debug.

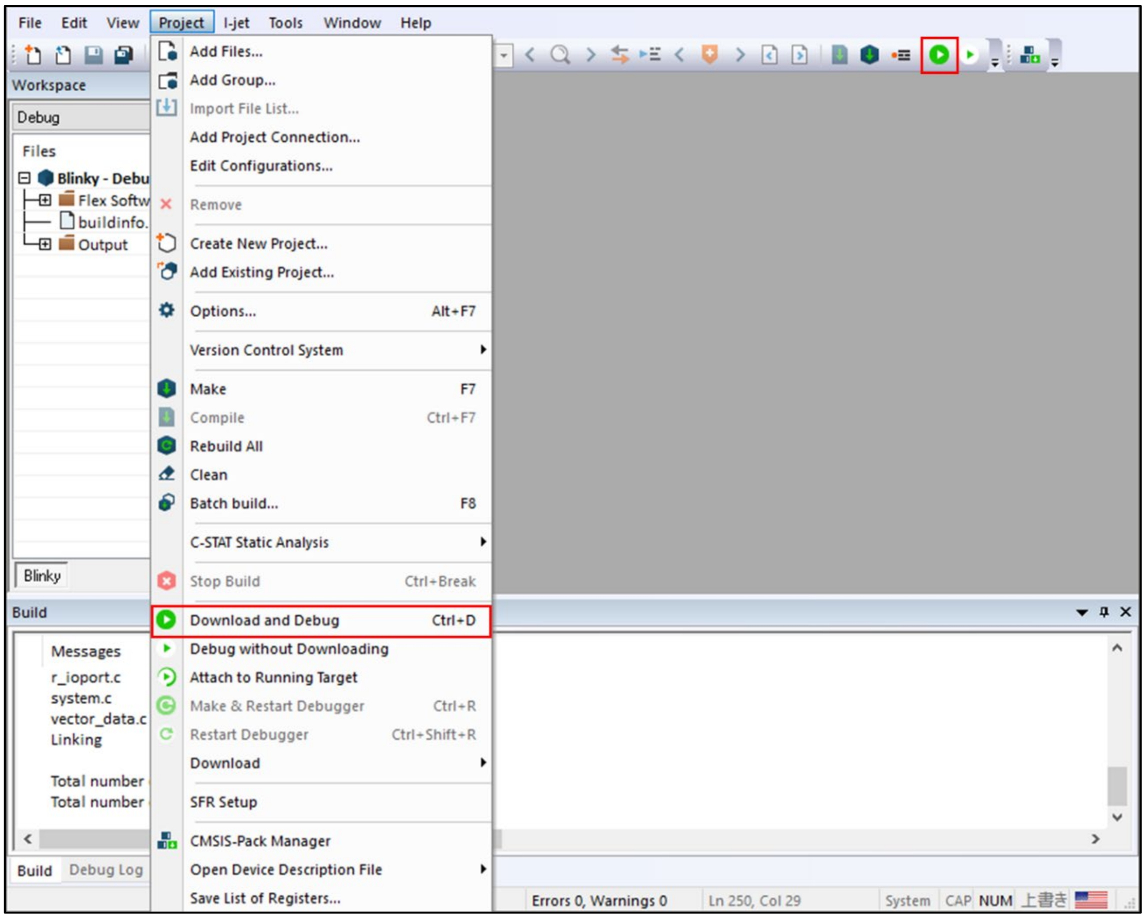
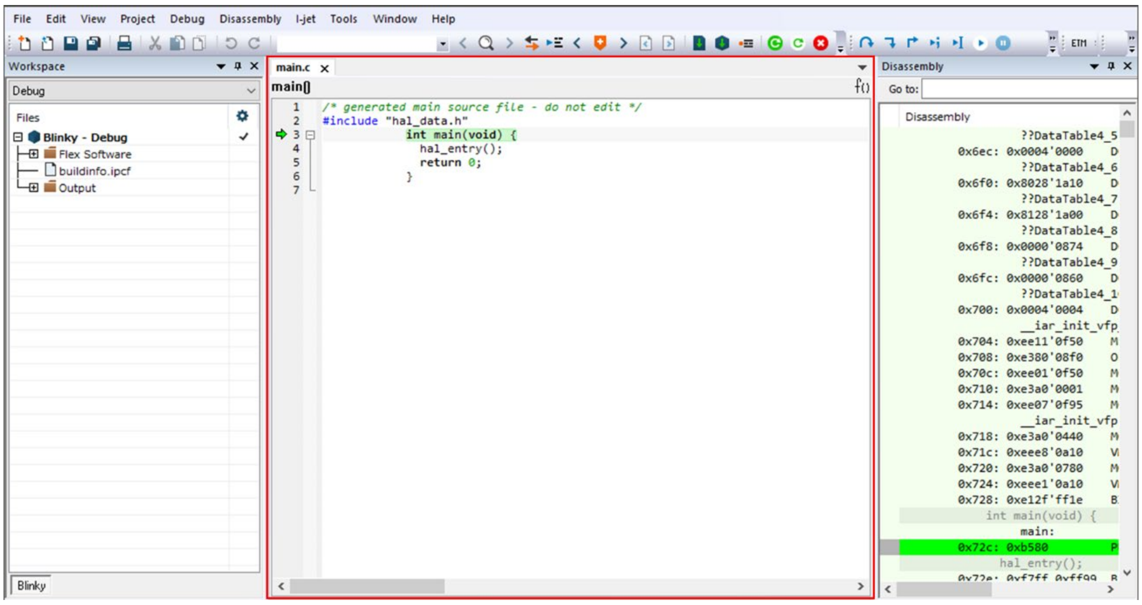


Figure 31 : Download and Debug button

(Continued on next page)

Once the download is completed and the debug is started, the program breaks at the beginning of **main** in **main.c**.

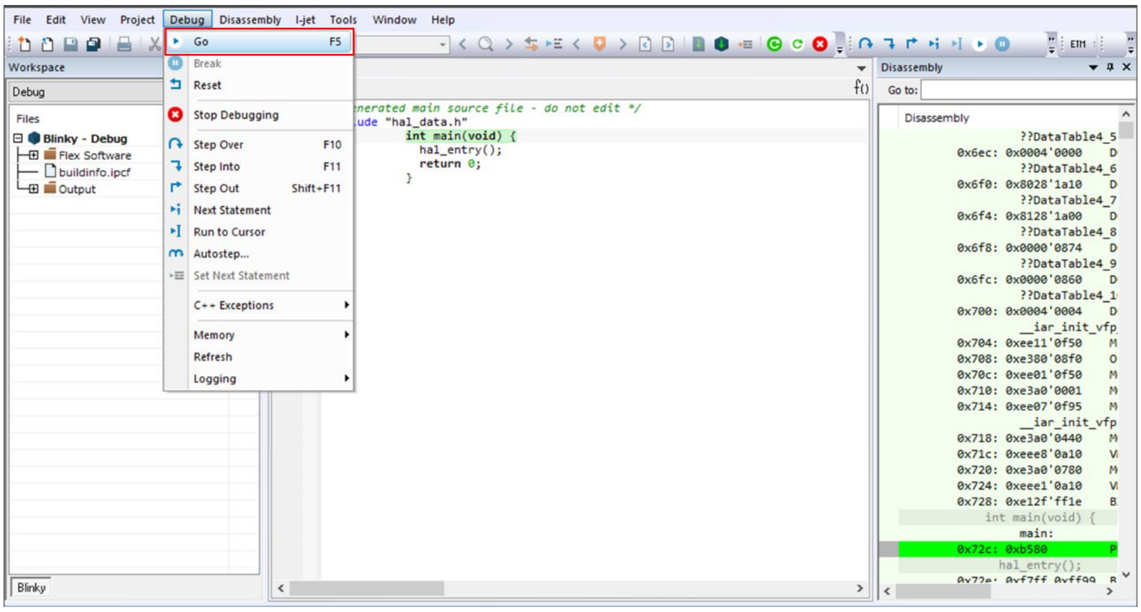


```
/* generated main source file - do not edit */
#include "hal_data.h"
        int main(void) {
            hal_entry();
            return 0;
        }
```

```
Disassembly
                  ??DataTable4_5
0x6ec: 0x0004'0000    D
                  ??DataTable4_6
0x6f0: 0x8028'1a10    D
                  ??DataTable4_7
0x6f4: 0x8128'1a00    D
                  ??DataTable4_8
0x6f8: 0x0000'0874    D
                  ??DataTable4_9
0x6fc: 0x0000'0860    D
                  ??DataTable4_1
0x700: 0x0004'0004    D
                  __iar_init_vfp
0x704: 0xee11'0f50    M
0x708: 0xe380'08f0    O
0x70c: 0xee01'0f50    M
0x710: 0xe3a0'0001    M
0x714: 0xee07'0f95    M
                  __iar_init_vfp
0x718: 0xe3a0'0440    M
0x71c: 0xeee8'0a10    V
0x720: 0xe3a0'0780    M
0x724: 0xeee1'0a10    V
0x728: 0xe12f'ff1e    B
      int main(void) {
                  main:
0x72c: 0xb580         P
          hal_entry();
```

Figure 32 : Starting Debug

Click on **Debug->Go** from menu bar or **Go** button on tool bar to run this program.



```
nerated main source file - do not edit */
ude "hal_data.h"
        int main(void) {
            hal_entry();
            return 0;
        }
```

Figure 33 : Go button

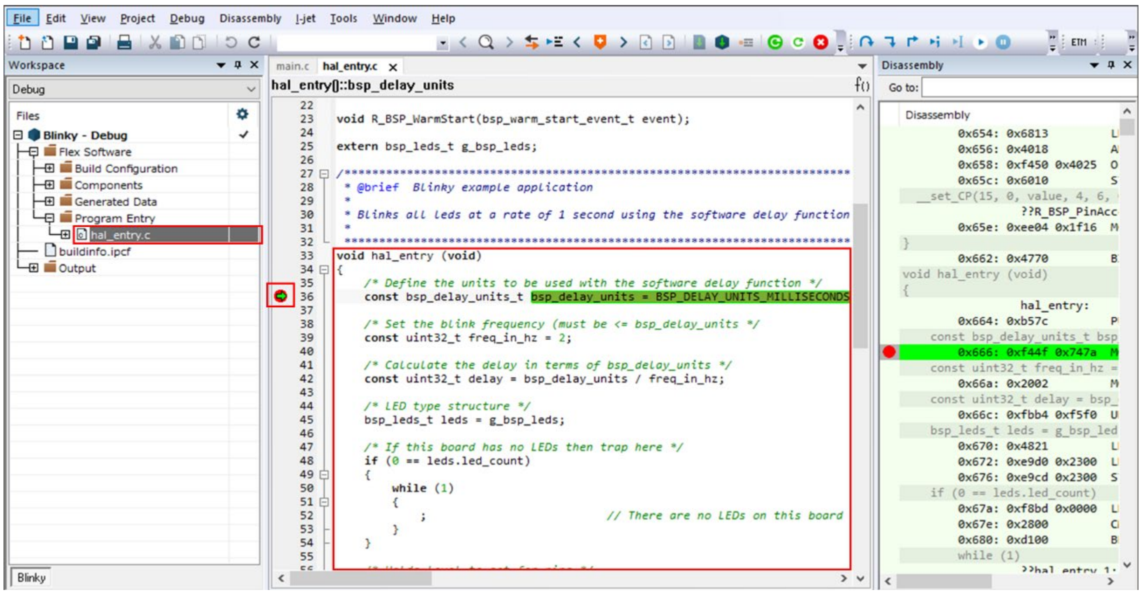
(Continued on next page)

The blinky application is stored in the **hal_entry.c** file. This file is generated by FSP SC when you select the Blinky Project template and is located in the project's src/ folder. In EWARM workspace view, the **hal_entry.c** is registered **Flex Software > Program Entry**.

The application performs the following steps:

1. Get the LED information for the selected board by **bsp_leds_t** structure.
2. Initialize output level information (**pin_level** variable) for LED pin to LOW.
3. Use **R_BSP_PinWrite ((bsp_io_port_pin_t) pin, pin_level)** to set the output level information to the LED pin.
4. Toggle the output level information for LED pin.
5. **R_BSP_SoftwareDelay(delay, bsp_delay_units)** waits for a certain period of time. Then run #3 again.

On debugging on EWARM, the break point can be set by click the left space next to line number.



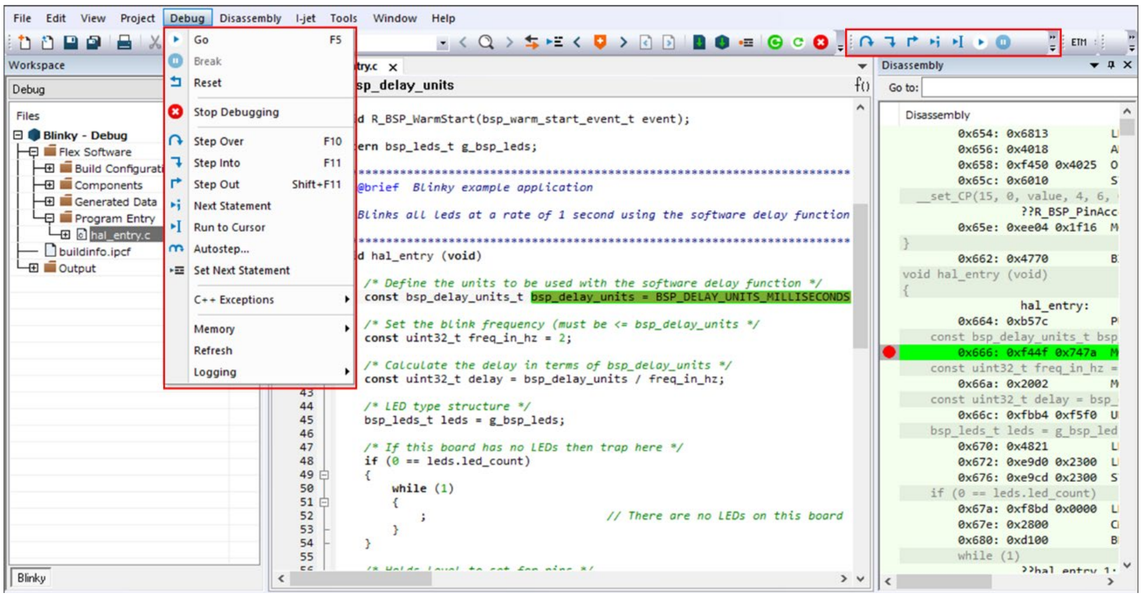
```
void R_BSP_WarmStart(bsp_warm_start_event_t event);

extern bsp_leds_t g_bsp_leds;

/*******************************************************************************
 * @brief  Blinky example application
 *
 * Blinks all leds at a rate of 1 second using the software delay function
 *
 *******************************************************************************
void hal_entry (void)
{
    /* Define the units to be used with the software delay function */
    const bsp_delay_units_t bsp_delay_units = BSP_DELAY_UNITS_MILLISECONDS

    /* Set the blink frequency (must be <= bsp_delay_units */
    const uint32_t freq_in_hz = 2;

    /* Calculate the delay in terms of bsp_delay_units */
    const uint32_t delay = bsp_delay_units / freq_in_hz;

    /* LED type structure */
    bsp_leds_t leds = g_bsp_leds;

    /* If this board has no LEDs then trap here */
    if (0 == leds.led_count)
    {
        while (1)
        {
            ;                               // There are no LEDs on this board
        }
    }
```

```
Disassembly
            0x654: 0x6813          L
            0x656: 0x4018          A
            0x658: 0xf450 0x4025   O
            0x65c: 0x6010          S
  __set_CP(15, 0, value, 4, 6,
                      ??R_BSP_PinAcc
            0x65e: 0xee04 0x1f16   M
}
            0x662: 0x4770          B
void hal_entry (void)
{
                      hal_entry:
            0x664: 0xb57c          P
    const bsp_delay_units_t bsp
            0x666: 0xf44f 0x747a   M
    const uint32_t freq_in_hz =
            0x66a: 0x2002          M
    const uint32_t delay = bsp_
            0x66c: 0xfbb4 0xf5f0   U
    bsp_leds_t leds = g_bsp_led
            0x670: 0x4821          L
            0x672: 0xe9d0 0x2300   L
            0x676: 0xe9cd 0x2300   S
    if (0 == leds.led_count)
            0x67a: 0xf8bd 0x0000   L
            0x67e: 0x2800          C
            0x680: 0xd100          B
        while (1)
```

Figure 34 : hal_entry.c and setting breakpoint

(Continued on next page)

By using the break point and the **Debug** menu or **Debug** tool bar, you can check the behavior of the Blinky application step by step.



```
d R_BSP_WarmStart(bsp_warm_start_event_t event);

ern bsp_leds_t g_bsp_leds;

*****************************************************************************
@brief  Blinky example application

Blinks all leds at a rate of 1 second using the software delay function

*****************************************************************************
d hal_entry (void)

  /* Define the units to be used with the software delay function */
  const bsp_delay_units_t bsp_delay_units = BSP_DELAY_UNITS_MILLISECONDS

  /* Set the blink frequency (must be <= bsp_delay_units */
  const uint32_t freq_in_hz = 2;

  /* Calculate the delay in terms of bsp_delay_units */
  const uint32_t delay = bsp_delay_units / freq_in_hz;

  /* LED type structure */
  bsp_leds_t leds = g_bsp_leds;

  /* If this board has no LEDs then trap here */
  if (0 == leds.led_count)
  {
      while (1)
      {
          ;                           // There are no LEDs on this board
      }
  }
```

```
Disassembly
          0x654: 0x6813           L
          0x656: 0x4018           A
          0x658: 0xf450 0x4025    O
          0x65c: 0x6010           S
   __set_CP(15, 0, value, 4, 6,
                ??R_BSP_PinAcc
          0x65e: 0xee04 0x1f16    M
}
          0x662: 0x4770           B
void hal_entry (void)
{
                hal_entry:
          0x664: 0xb57c           P
    const bsp_delay_units_t bsp
          0x666: 0xf44f 0x747a    M
    const uint32_t freq_in_hz =
          0x66a: 0x2002           M
    const uint32_t delay = bsp_
          0x66c: 0xfbb4 0xf5f0    U
    bsp_leds_t leds = g_bsp_led
          0x670: 0x4821           L
          0x672: 0xe9d0 0x2300    L
          0x676: 0xe9cd 0x2300    S
    if (0 == leds.led_count)
          0x67a: 0xf8bd 0x0000    L
          0x67e: 0x2800           C
          0x680: 0xd100           B
        while (1)
```

Figure 35 : Debug menu

5.4 Re-configuring project with FSP SC

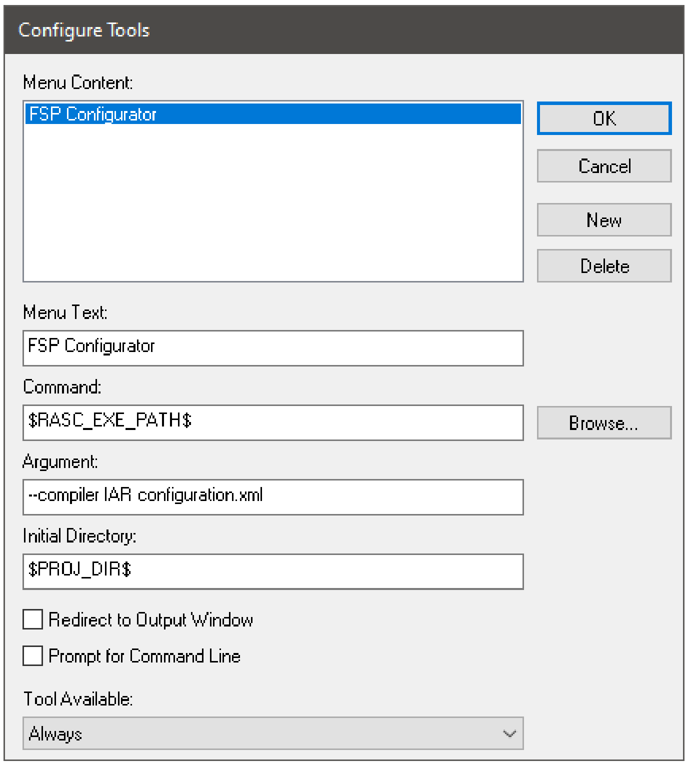
For proceeding the tutorial with Blinky project, the FSP configuration steps of the Blinky project was skipped in this chapter. The FSP SC can be launched from IAR EWARM or command prompt, and the FSP project configuration can be re-configured by FSP SC.

There are two ways to launch FSP Smart Configurator with exiting project.

5.4.1 Launch FSP Smart Configurator from IAR EWARM

1. Select "Tools -> Configure Tools…"
2. Select "New" and fill in the fields as follows:

• Menu Content:	FSP SC Smart Configurator:
• Menu Text	FSP Smart Configurator
• Command	$RASC_EXE_PATH$
• Argument	--compiler IAR configuration.xml
• Initial Directory	$PROJ_DIR$


Configure Tools

Menu Content: FSP Configurator

Menu Text: FSP Configurator

Command: $RASC_EXE_PATH$

Argument: --compiler IAR configuration.xml

Initial Directory: $PROJ_DIR$

☐ Redirect to Output Window

☐ Prompt for Command Line

Tool Available: Always

Figure 36 : Settings to launch FSPSC from EWARM

5.4.2 Launch from the command prompt.

1. Open command prompt window.
2. Move to the folder where the created project is located.
3. Execute the following command.
   - {FSP Smart Configurator installation folder} \ eclipse \ rasc.exe --compiler IAR configuration.xml

6. FSP Configuration Users Guide

6.1 What is a Project?

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

1. **Create a Project**
2. **Configuring a Project**

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

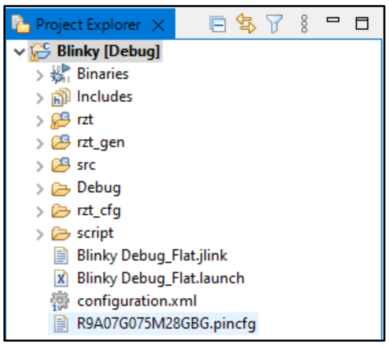


Figure 37 : e2 studio Project Configuration file

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

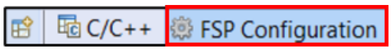
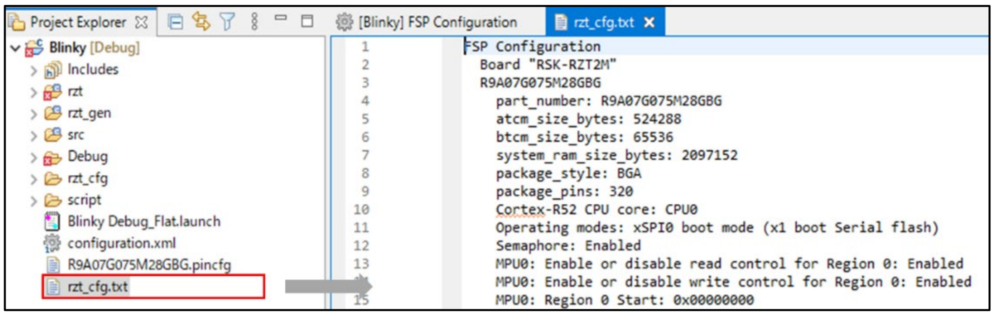


Figure 38 : e2 studio FSP Configuration Perspective

(Continued on next page)

Note:

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



```
FSP Configuration
  Board "RSK-RZT2M"
  R9A07G075M28GBG
    part_number: R9A07G075M28GBG
    atcm_size_bytes: 524288
    btcm_size_bytes: 65536
    system_ram_size_bytes: 2097152
    package_style: BGA
    package_pins: 320
    Cortex-R52 CPU core: CPU0
    Operating modes: xSPI0 boot mode (x1 boot Serial flash)
    Semaphore: Enabled
    MPU0: Enable or disable read control for Region 0: Enabled
    MPU0: Enable or disable write control for Region 0: Enabled
    MPU0: Region 0 Start: 0x00000000
```

Figure 39 : RZ Project Report

The RZ Project Editor has several tabs. The configuration steps and options for individual tabs are discussed in the following sections.

Note:

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

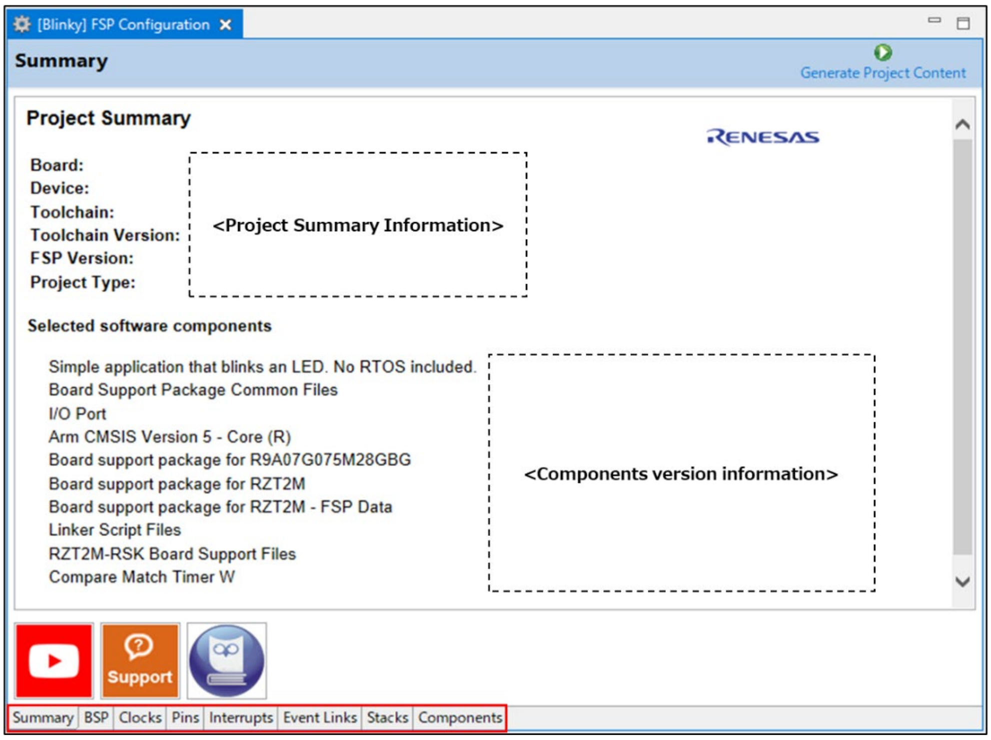

[Blinky] FSP Configuration
Summary
Generate Project Content
Project Summary
Board:
Device:
Toolchain:
Toolchain Version:
FSP Version:
Project Type:
<Project Summary Information>
Selected software components
Simple application that blinks an LED. No RTOS included.
Board Support Package Common Files
I/O Port
Arm CMSIS Version 5 - Core (R)
Board support package for R9A07G075M28GBG
Board support package for RZT2M
Board support package for RZT2M - FSP Data
Linker Script Files
RZT2M-RSK Board Support Files
Compare Match Timer W
<Components version information>
Support
Summary
BSP
Clocks
Pins
Interrupts
Event Links
Stacks
Components

Figure 40 : RZ Project Editor tabs

6.2 Create a Project

6.2.1 Creating a New Project

For RZ MPU applications, generate a new project using the following steps:

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

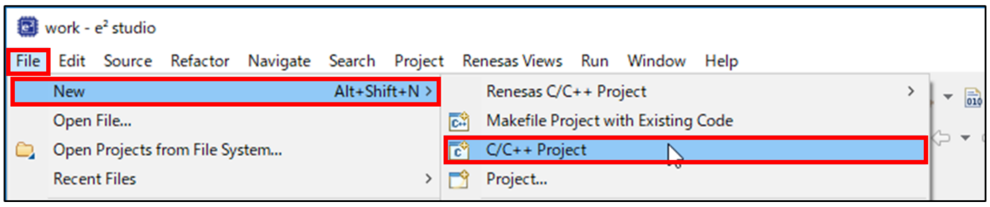
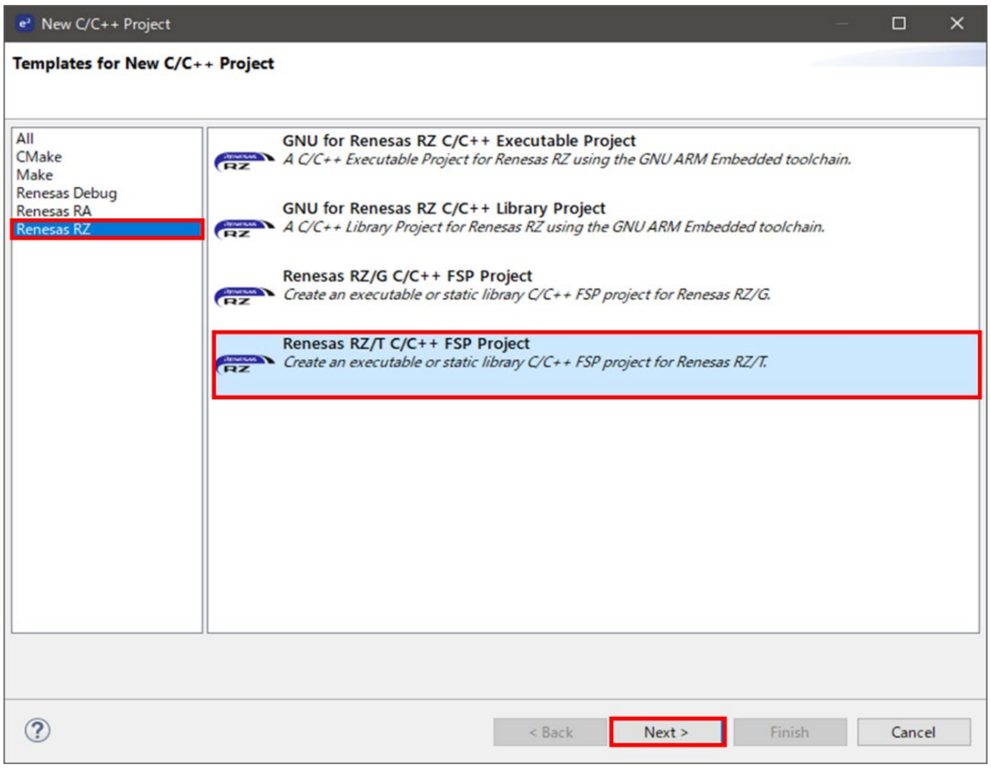


Figure 41 : New RZ MPU Project

2. Then click on the **Renesas RZ/T C/C++ FSP Project** template for the type of project you are creating.



GNU for Renesas RZ C/C++ Executable Project
A C/C++ Executable Project for Renesas RZ using the GNU ARM Embedded toolchain.

GNU for Renesas RZ C/C++ Library Project
A C/C++ Library Project for Renesas RZ using the GNU ARM Embedded toolchain.

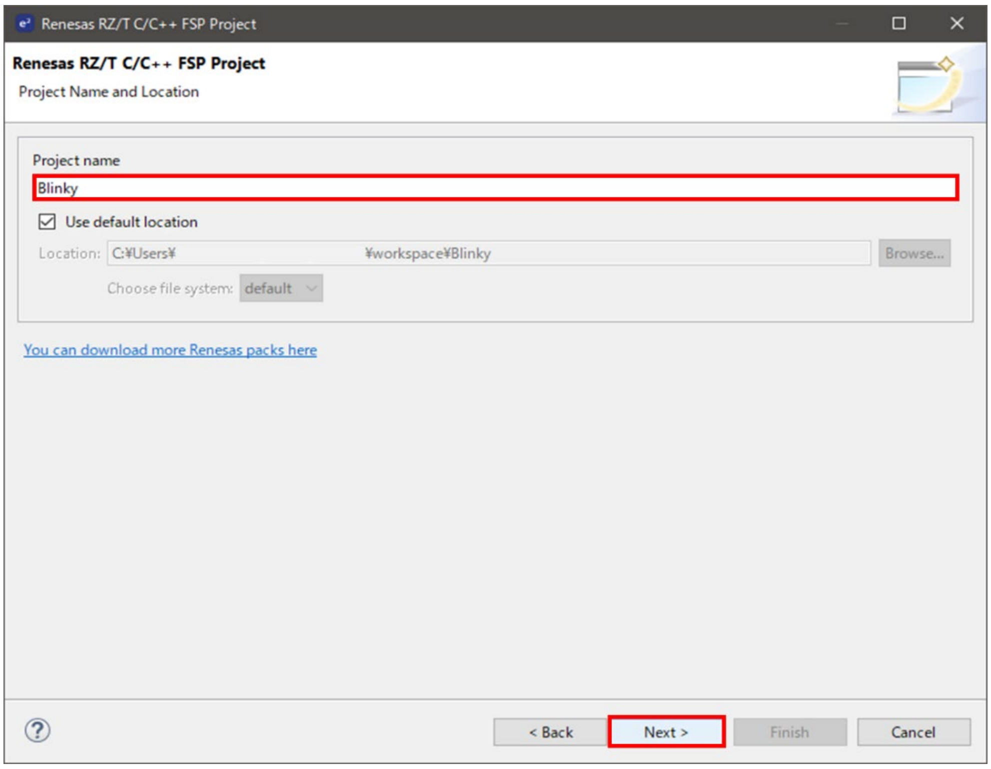
Renesas RZ/G C/C++ FSP Project
Create an executable or static library C/C++ FSP project for Renesas RZ/G.

Renesas RZ/T C/C++ FSP Project
Create an executable or static library C/C++ FSP project for Renesas RZ/T.

Figure 42 : New Project Templates

(Continued on next page)

3. Select a project name and location.
4. Click Next.



Renesas RZ/T C/C++ FSP Project

Renesas RZ/T C/C++ FSP Project

Project Name and Location

Project name: Blinky

☑ Use default location

Location: C:¥Users¥ ¥workspace¥Blinky

Choose file system: default

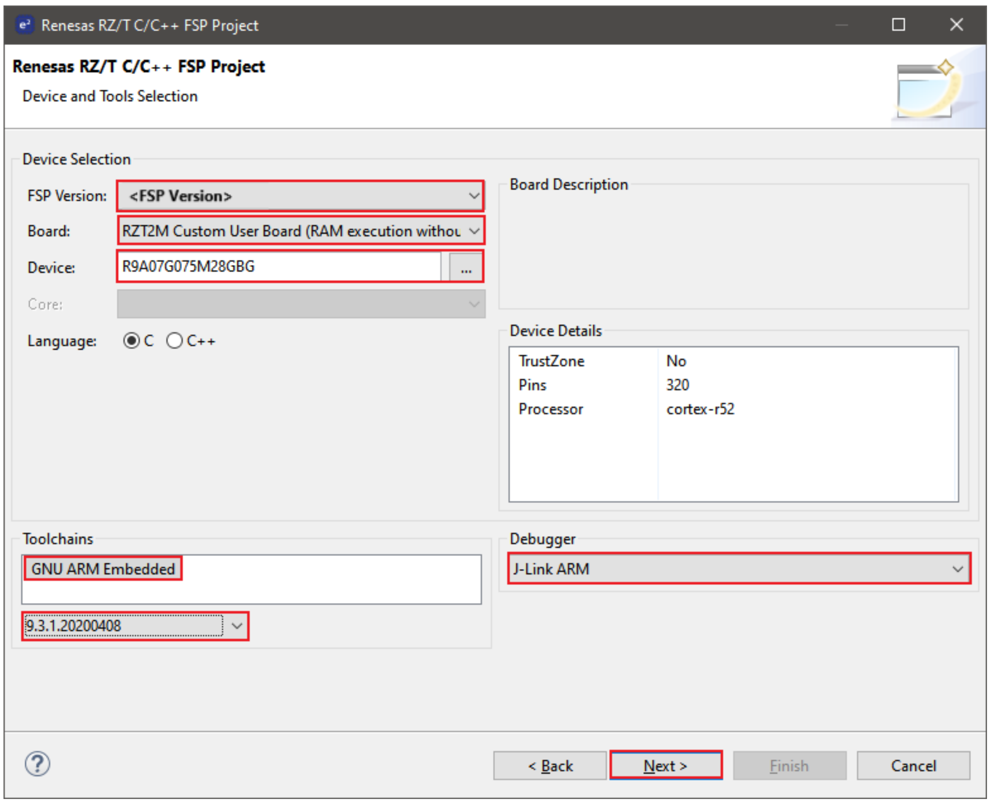
You can download more Renesas packs here

Figure 43 : RZ MPU Project Generator (Screen 1)

6.2.2 Selecting a Board and Toolchain

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

1. Select the **FSP version**.
2. Select the **Board** and **Device** for your application.
    - You can select an existing RZ MPU Evaluation Kit or select **Custom User Board** for any of the RZ MPU devices with your own BSP definition.
    - The **Device** is automatically populated based on the **Board** selection.
    - Please change the **Device** when using **Custom User Board** in board selections.
    - After changing the **Device**, please select the **Custom User Board** depending on the boot mode which you require.
3. The **Toolchain** selection defaults to **GNU Arm Embedded**.
4. Select the **Toolchain version**.
    - This should default to the installed toolchain version.
5. Select the **Debugger**.
    - The J-Link Arm Debugger is preselected.
6. Click **Next**.


Renesas RZ/T C/C++ FSP Project

Renesas RZ/T C/C++ FSP Project

Device and Tools Selection

Device Selection

FSP Version: <FSP Version>

Board: RZT2M Custom User Board (RAM execution withou

Device: R9A07G075M28GBG

Core:

Language: [✓ C] C++

Board Description

Device Details

TrustZone	No
Pins	320
Processor	cortex-r52

Toolchains

GNU ARM Embedded

9.3.1.20200408

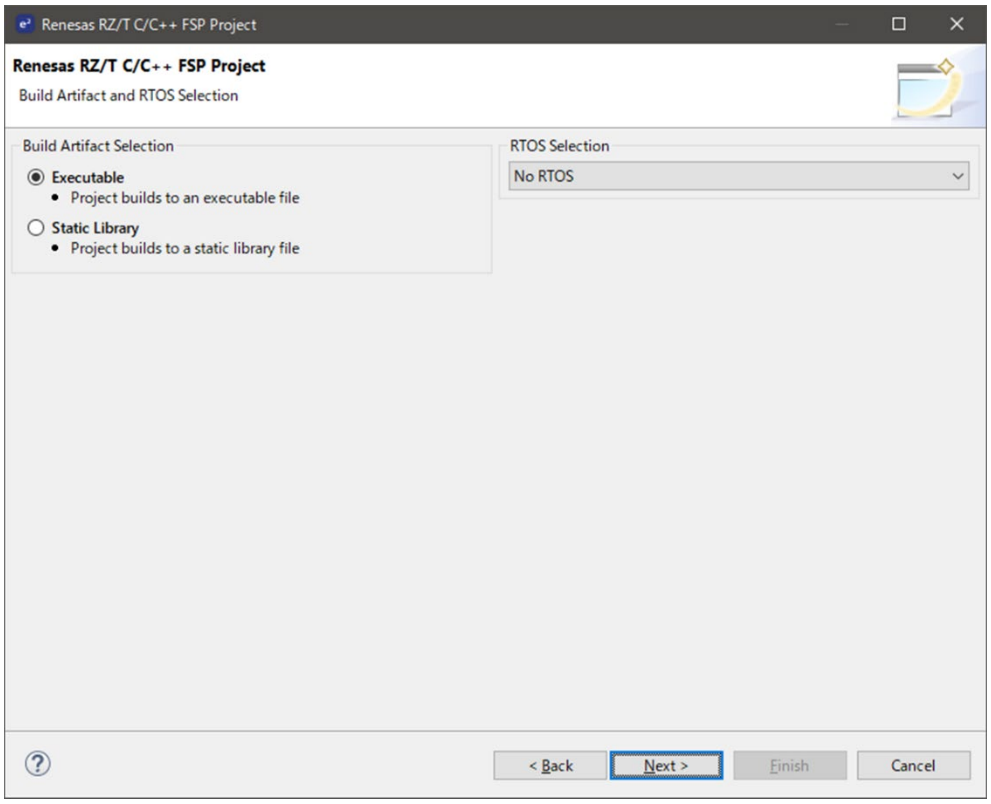
Debugger

J-Link ARM

Figure 44 : RZ MPU Project Generator (Screen 2)

6.2.3 Selecting a Project Template

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



Renesas RZ/T C/C++ FSP Project

Renesas RZ/T C/C++ FSP Project

Build Artifact and RTOS Selection

Build Artifact Selection

(●) **Executable**
- Project builds to an executable file

( ) **Static Library**
- Project builds to a static library file

RTOS Selection: No RTOS

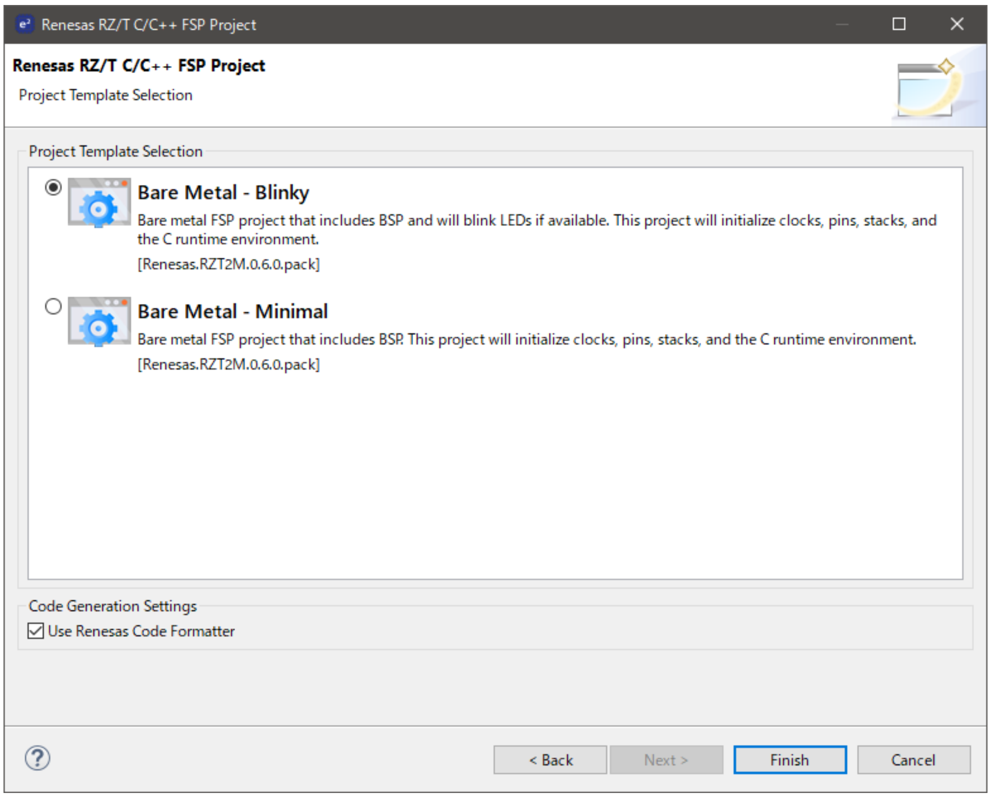
Figure 45 : RZ MPU Project Generator (Screen 3)

(Continued on next page)

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

Note:

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



Renesas RZ/T C/C++ FSP Project

Project Template Selection

Project Template Selection

[✓] **Bare Metal - Blinky**
Bare metal FSP project that includes BSP and will blink LEDs if available. This project will initialize clocks, pins, stacks, and the C runtime environment.
[Renesas.RZT2M.0.6.0.pack]

[ ] **Bare Metal - Minimal**
Bare metal FSP project that includes BSP. This project will initialize clocks, pins, stacks, and the C runtime environment.
[Renesas.RZT2M.0.6.0.pack]

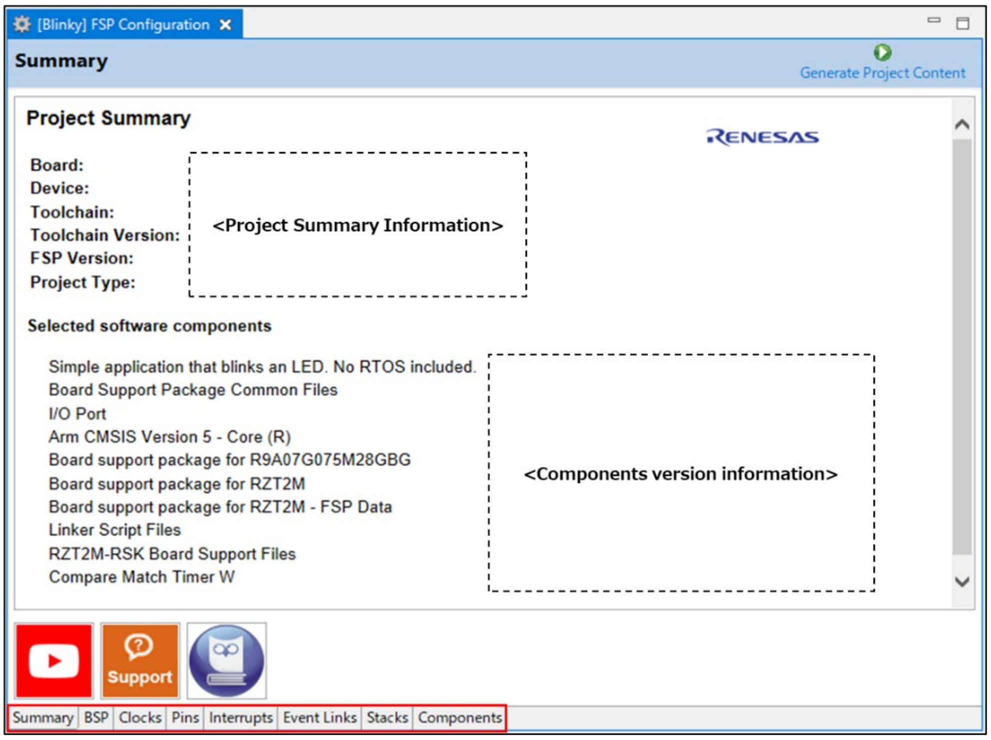
Code Generation Settings

[✓] Use Renesas Code Formatter

Figure 46 : RZ MPU Project Generator (Screen 4)

(Continued on next page)

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


[Blinky] FSP Configuration
Summary
Generate Project Content
Support
Summary | BSP | Clocks | Pins | Interrupts | Event Links | Stacks | Components

Project Summary

Board:
Device:
Toolchain:
Toolchain Version:
FSP Version:
Project Type:

<Project Summary Information>

Selected software components

Simple application that blinks an LED. No RTOS included.
Board Support Package Common Files
I/O Port
Arm CMSIS Version 5 - Core (R)
Board support package for R9A07G075M28GBG
Board support package for RZT2M
Board support package for RZT2M - FSP Data
Linker Script Files
RZT2M-RSK Board Support Files
Compare Match Timer W

<Components version information>

Figure 47 : RZ MPU Project Editor and available editor tabs

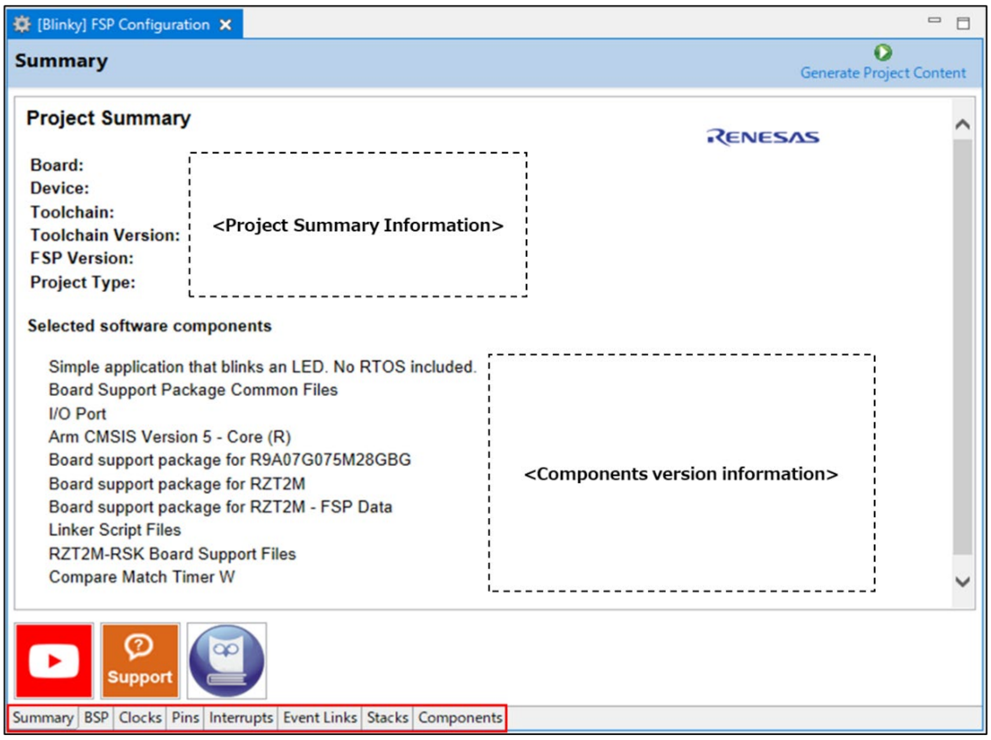
On the bottom of the RZ MPU Project Editor view, you can find the tabs for configuring multiple aspects of your project:

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

6.3 Configuring a Project

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

6.3.1 Summary Tab



Project Summary

Board:
Device:
Toolchain:
Toolchain Version:
FSP Version:
Project Type:

<Project Summary Information>

Selected software components

Simple application that blinks an LED. No RTOS included.
Board Support Package Common Files
I/O Port
Arm CMSIS Version 5 - Core (R)
Board support package for R9A07G075M28GBG
Board support package for RZT2M
Board support package for RZT2M - FSP Data
Linker Script Files
RZT2M-RSK Board Support Files
Compare Match Timer W

<Components version information>

Figure 48 : Configuration Summary tab

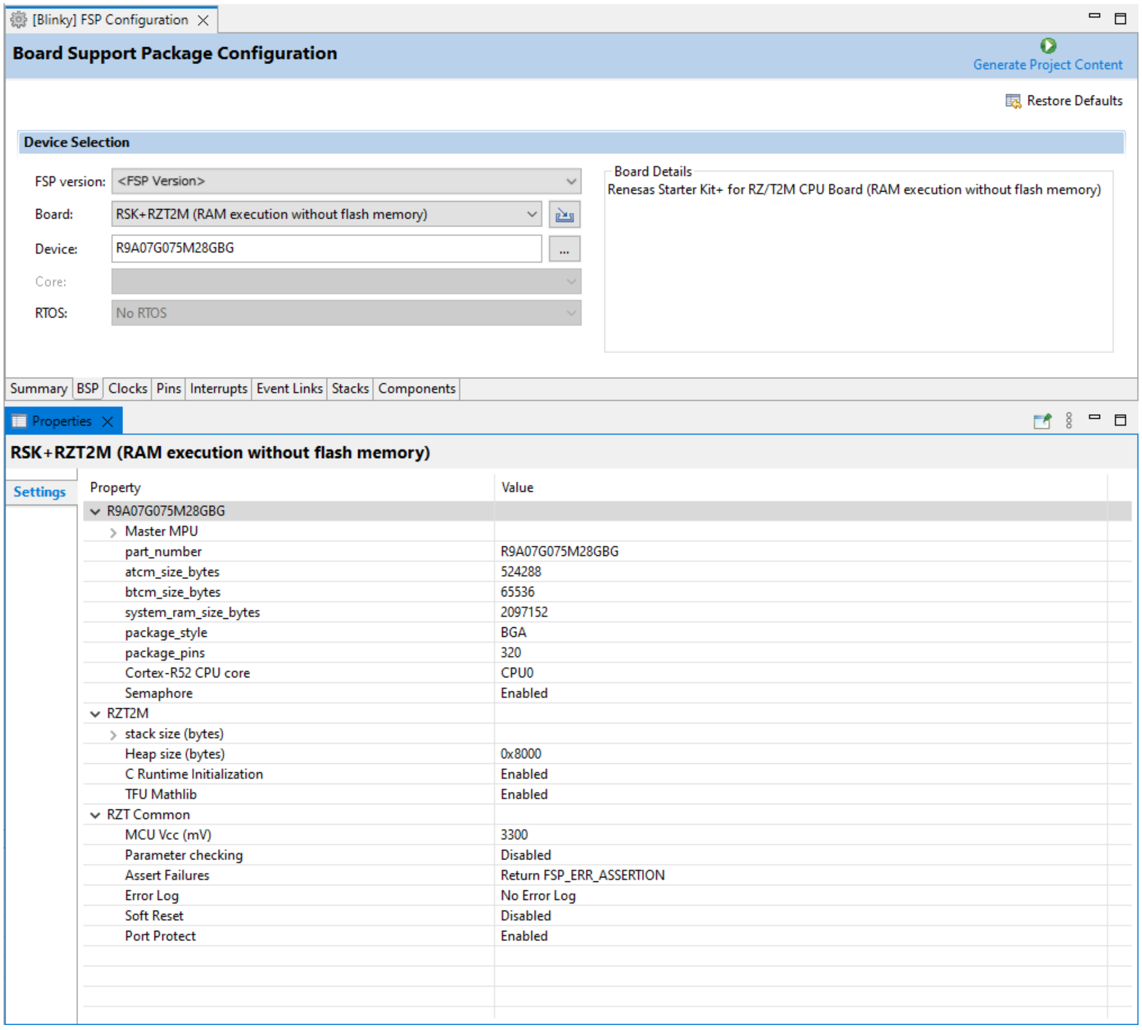
The **Summary** tab, seen in the above figure, identifies all the key elements and components of a project. It shows the target board, the device, toolchain and FSP version. Additionally, it provides a list of all the selected software components and modules used by the project. This is a more convenient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

6.3.2 Configuring the BSP

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

Note:

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



FSP version:	<FSP Version>
Board:	RSK+RZT2M (RAM execution without flash memory)
Device:	R9A07G075M28GBG
Core:	
RTOS:	No RTOS

Board Details
Renesas Starter Kit+ for RZ/T2M CPU Board (RAM execution without flash memory)

RSK+RZT2M (RAM execution without flash memory)

Property	Value
R9A07G075M28GBG	
Master MPU	
part_number	R9A07G075M28GBG
atcm_size_bytes	524288
btcm_size_bytes	65536
system_ram_size_bytes	2097152
package_style	BGA
package_pins	320
Cortex-R52 CPU core	CPU0
Semaphore	Enabled
RZT2M	
stack size (bytes)	
Heap size (bytes)	0x8000
C Runtime Initialization	Enabled
TFU Mathlib	Enabled
RZT Common	
MCU Vcc (mV)	3300
Parameter checking	Disabled
Assert Failures	Return FSP_ERR_ASSERTION
Error Log	No Error Log
Soft Reset	Disabled
Port Protect	Enabled

Figure 49 : Configuration BSP tab

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

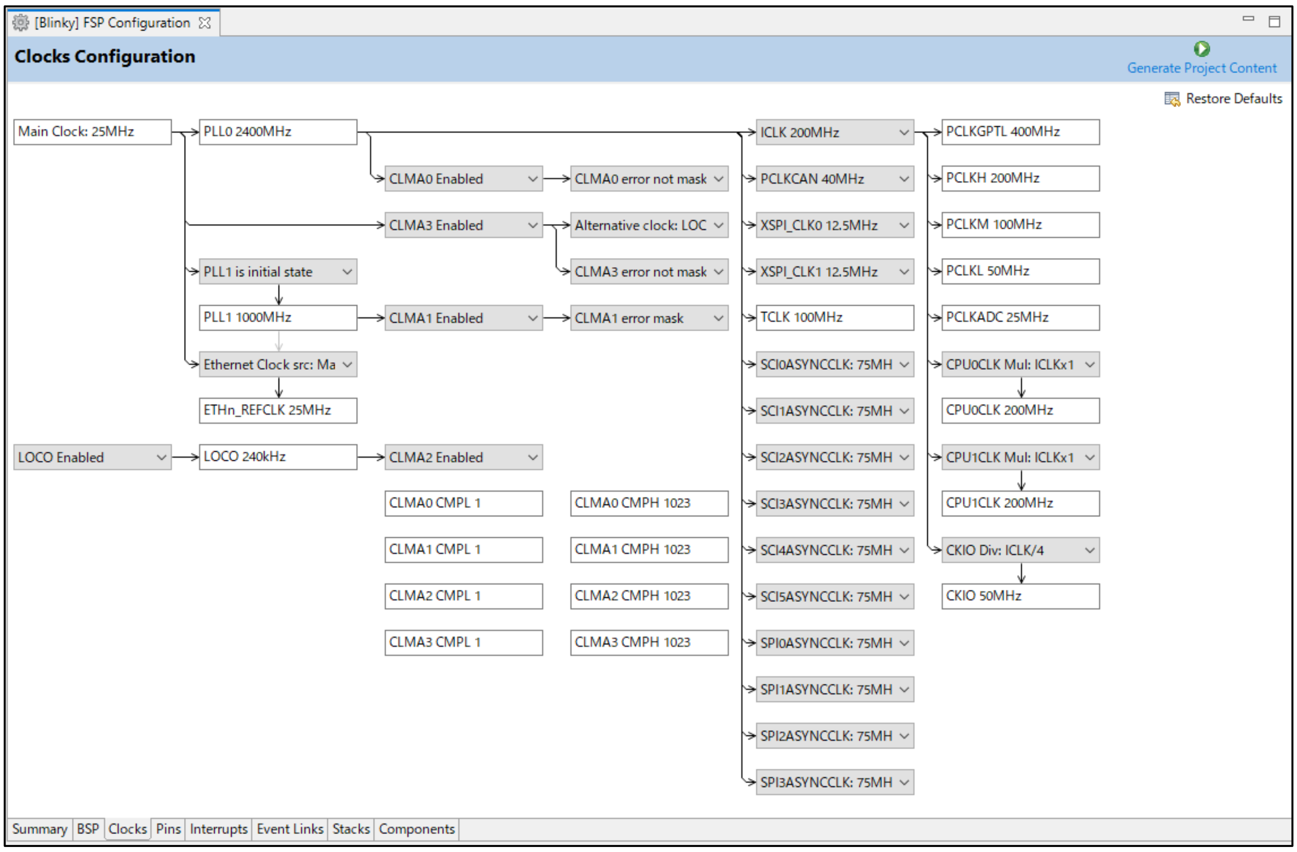
When you click the **Generate Project Content** button, the BSP configuration contents are written to rzt_cfg/fsp_cfg/bsp/bsp_cfg.h This file is created if it does not already exist.

Warning:

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

6.3.3 Configuring Clocks

The Clocks tab presents a graphical view of the MPU's clock tree, allowing the various clock dividers and sources to be modified.



Main Clock: 25MHz
PLL0 2400MHz
CLMA0 Enabled
CLMA0 error not mask
CLMA3 Enabled
Alternative clock: LOC
CLMA3 error not mask
PLL1 is initial state
PLL1 1000MHz
CLMA1 Enabled
CLMA1 error mask
Ethernet Clock src: Ma
ETHn_REFCLK 25MHz
LOCO Enabled
LOCO 240kHz
CLMA2 Enabled
CLMA0 CMPL 1
CLMA0 CMPH 1023
CLMA1 CMPL 1
CLMA1 CMPH 1023
CLMA2 CMPL 1
CLMA2 CMPH 1023
CLMA3 CMPL 1
CLMA3 CMPH 1023
ICLK 200MHz
PCLKCAN 40MHz
XSPI_CLK0 12.5MHz
XSPI_CLK1 12.5MHz
TCLK 100MHz
SCI0ASYNCCLK: 75MH
SCI1ASYNCCLK: 75MH
SCI2ASYNCCLK: 75MH
SCI3ASYNCCLK: 75MH
SCI4ASYNCCLK: 75MH
SCI5ASYNCCLK: 75MH
SPI0ASYNCCLK: 75MH
SPI1ASYNCCLK: 75MH
SPI2ASYNCCLK: 75MH
SPI3ASYNCCLK: 75MH
PCLKGPTL 400MHz
PCLKH 200MHz
PCLKM 100MHz
PCLKL 50MHz
PCLKADC 25MHz
CPU0CLK Mul: ICLKx1
CPU0CLK 200MHz
CPU1CLK Mul: ICLKx1
CPU1CLK 200MHz
CKIO Div: ICLK/4
CKIO 50MHz

Figure 50 : Configuration Clocks tab

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

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

Warning:

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

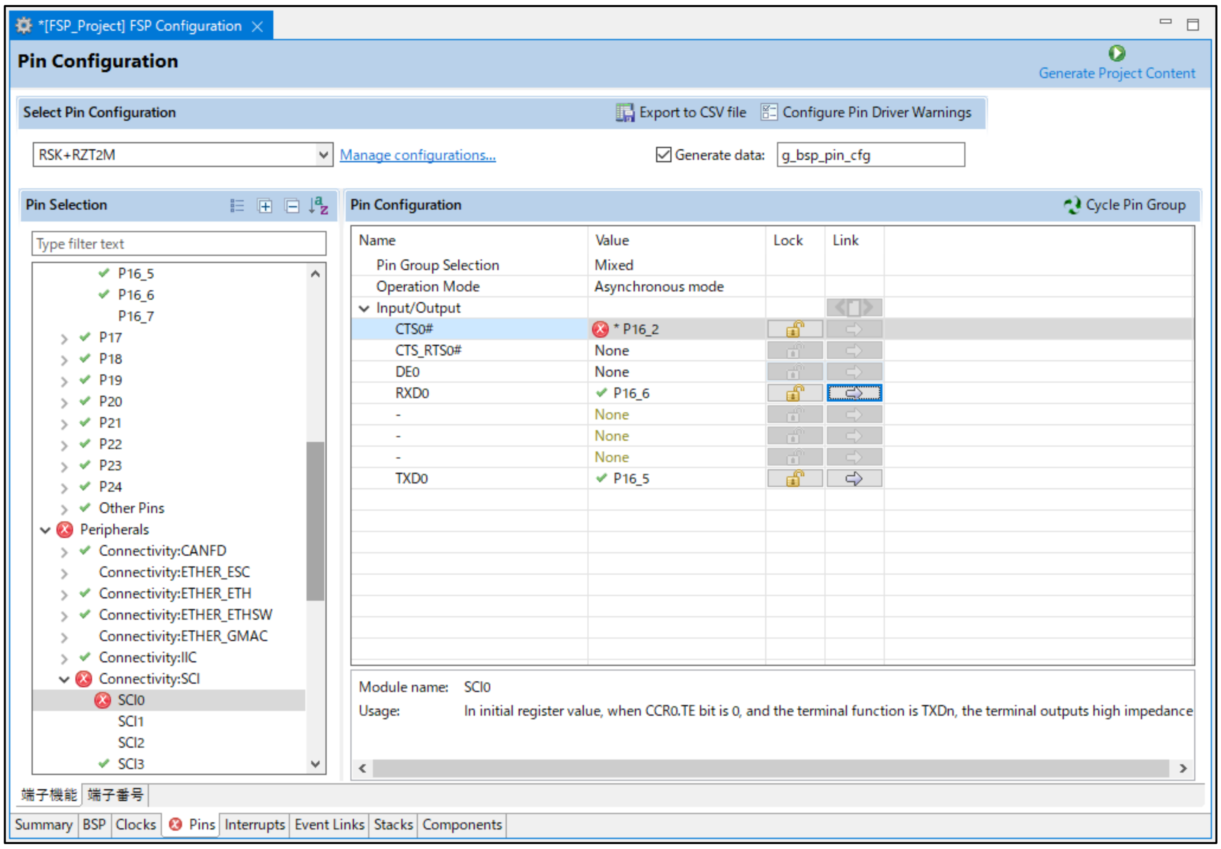
6.3.4 Configuring Pins

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

Note:

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

The **Pins** tab simplifies the configuration of large packages with highly multiplexed pins by highlighting errors and presenting the options for each pin or for each peripheral. If you selected a project template for a specific board such as RSK+RZT2M, some peripherals connected on the board are preselected.



RSK+RZT2M

☑ Generate data: g_bsp_pin_cfg

Pin Selection:

- ✓ P16_5
- ✓ P16_6
- P16_7
- ✓ P17
- ✓ P18
- ✓ P19
- ✓ P20
- ✓ P21
- ✓ P22
- ✓ P23
- ✓ P24
- ✓ Other Pins
- ✗ Peripherals
  - ✓ Connectivity:CANFD
  - Connectivity:ETHER_ESC
  - ✓ Connectivity:ETHER_ETH
  - ✓ Connectivity:ETHER_ETHSW
  - Connectivity:ETHER_GMAC
  - ✓ Connectivity:IIC
  - ✗ Connectivity:SCI
    - ✗ SCI0
    - SCI1
    - SCI2
    - ✓ SCI3

Name	Value	Lock	Link
Pin Group Selection	Mixed		
Operation Mode	Asynchronous mode		
Input/Output			
CTS0#	✗ * P16_2		
CTS_RTS0#	None		
DE0	None		
RXD0	✓ P16_6		
-	None		
-	None		
-	None		
TXD0	✓ P16_5		

Module name: SCI0

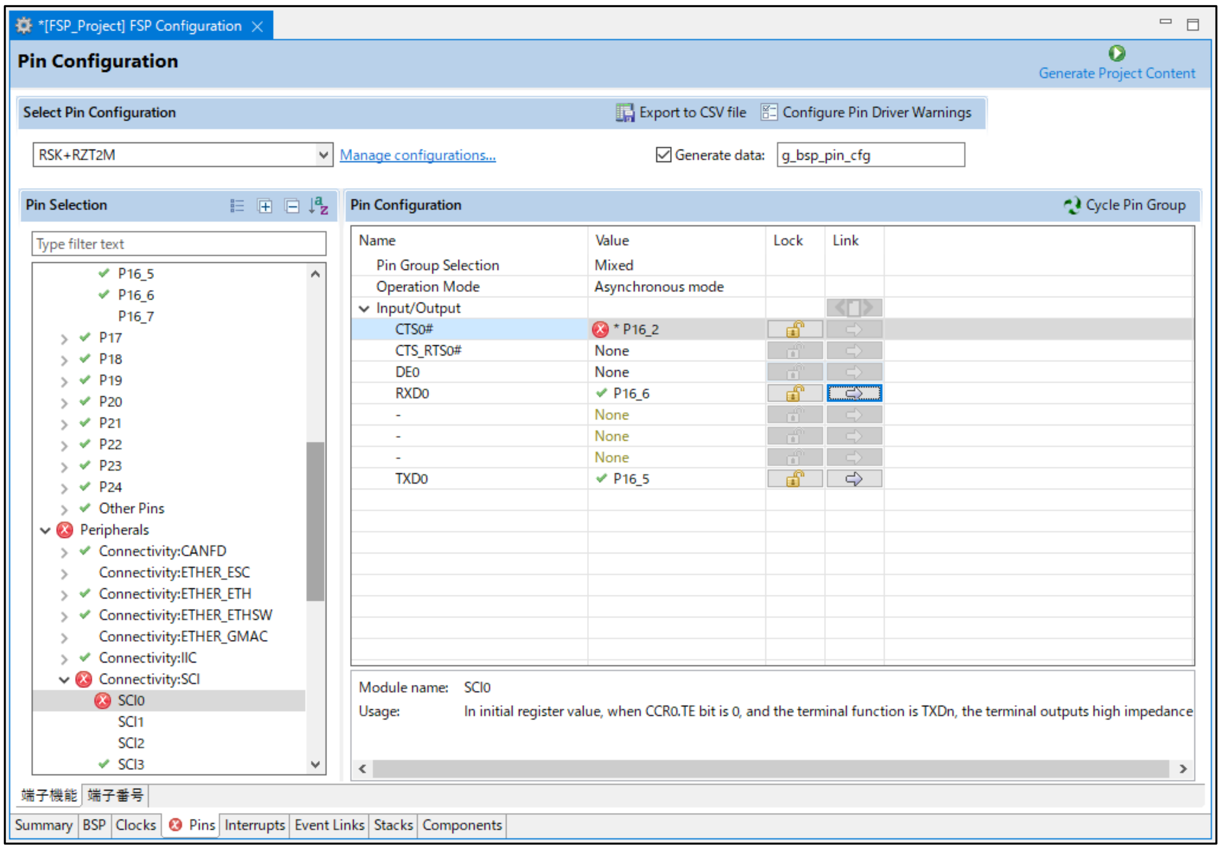
Usage: In initial register value, when CCR0.TE bit is 0, and the terminal function is TXDn, the terminal outputs high impedance

Figure 51: Pin Configuration

(Continued on next page)

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

In the example shown below, port P162 is already used by the GPIO, and the attempt to connect this port to the Serial Communications Interface (SCI) results in a dangling connection error. To fix this error, select another port from the pin drop-down list or disable the GPIO in the **Pin Selection** pane on the left side of the tab.



RSK+RZT2M

Generate data: g_bsp_pin_cfg

Pin Selection:

- P16_5
- P16_6
- P16_7
- P17
- P18
- P19
- P20
- P21
- P22
- P23
- P24
- Other Pins
- Peripherals
  - Connectivity:CANFD
  - Connectivity:ETHER_ESC
  - Connectivity:ETHER_ETH
  - Connectivity:ETHER_ETHSW
  - Connectivity:ETHER_GMAC
  - Connectivity:IIC
  - Connectivity:SCI
    - SCI0
    - SCI1
    - SCI2
    - SCI3

Name	Value	Lock	Link
Pin Group Selection	Mixed		
Operation Mode	Asynchronous mode		
Input/Output			
CTS0#	* P16_2		
CTS_RTS0#	None		
DE0	None		
RXD0	P16_6		
-	None		
-	None		
-	None		
TXD0	P16_5		

Module name: SCI0

Usage: In initial register value, when CCR0.TE bit is 0, and the terminal function is TXDn, the terminal outputs high impedance

Figure 52: conflict checker in Pin Configuration

(Continued on next page)

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

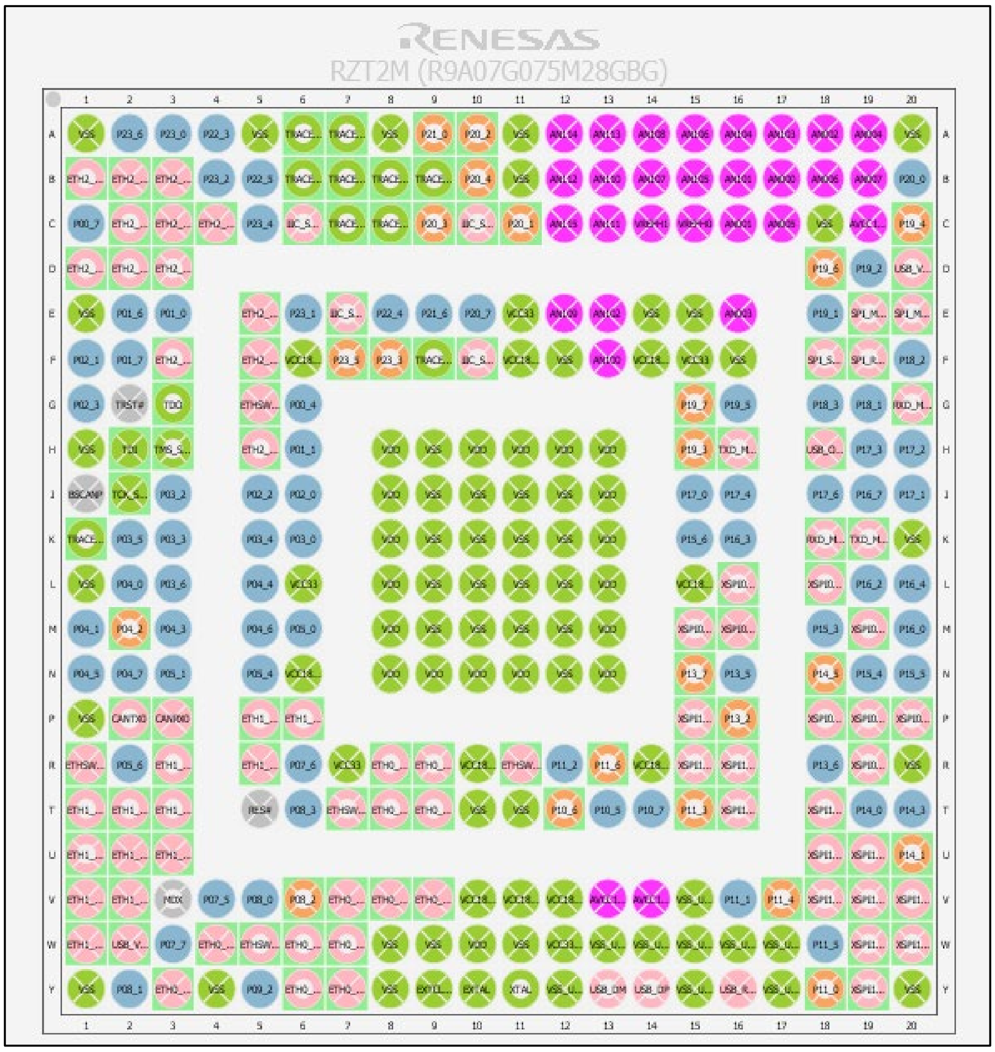


Figure 53: Pin configurator package view

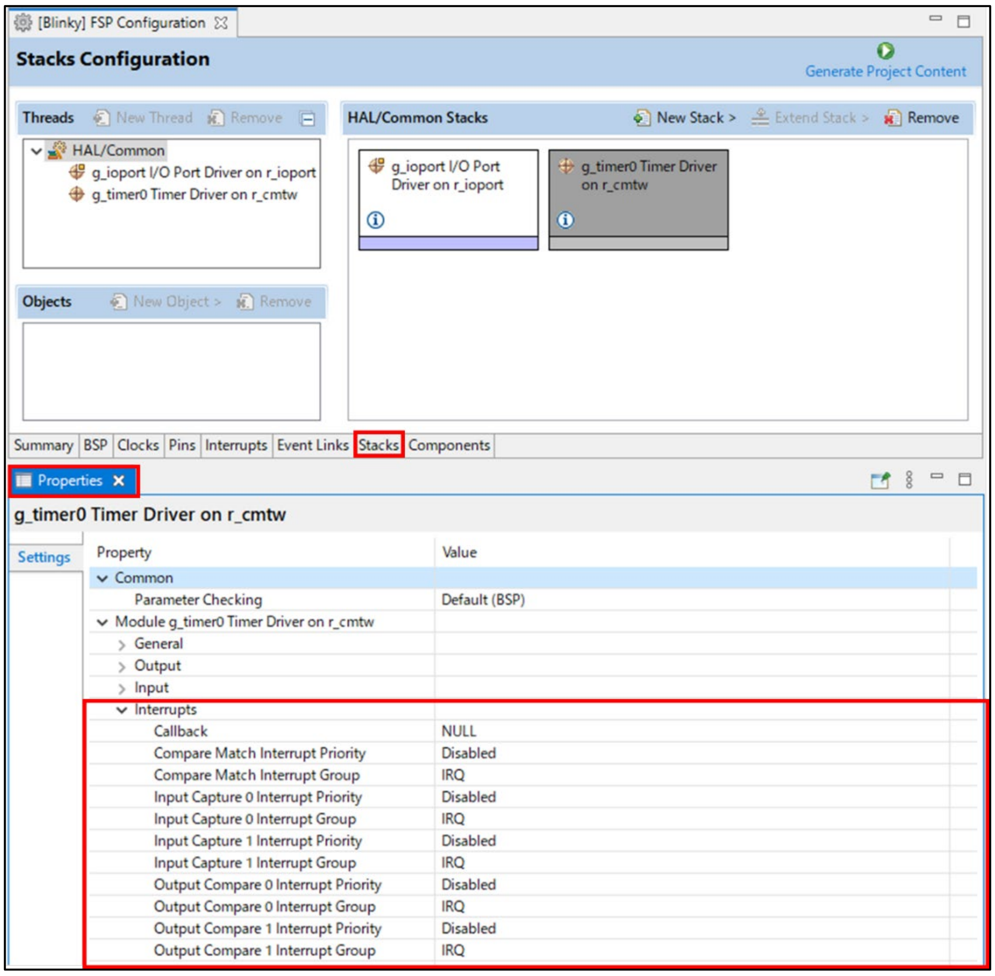
When you click the **Generate Project Content button**, the pin configuration contents are written to:
rzt_gen\bsp_pin_cfg.h
This file will be created if it does not already exist.

Warning:

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

6.4 Configuring Interrupts from the Stacks Tab

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

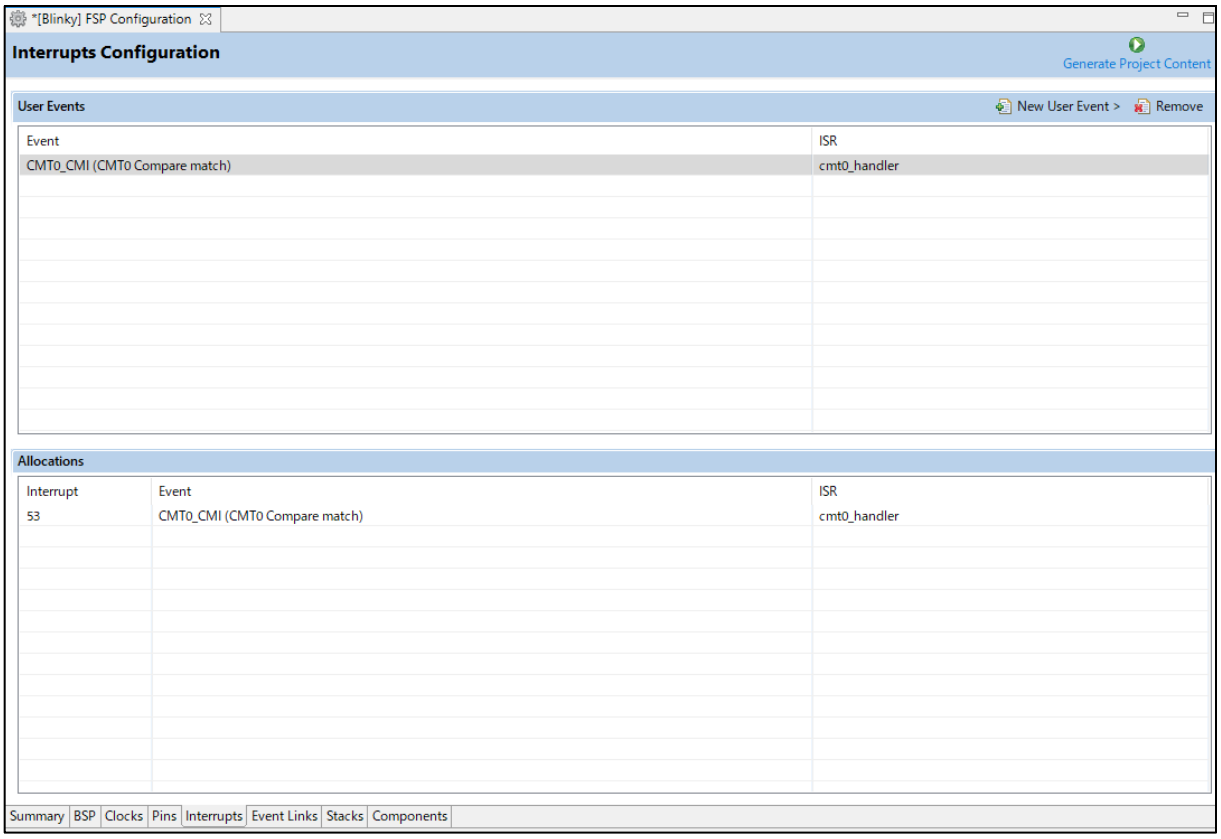


Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_timer0 Timer Driver on r_cmtw	
General	
Output	
Input	
Interrupts	
Callback	NULL
Compare Match Interrupt Priority	Disabled
Compare Match Interrupt Group	IRQ
Input Capture 0 Interrupt Priority	Disabled
Input Capture 0 Interrupt Group	IRQ
Input Capture 1 Interrupt Priority	Disabled
Input Capture 1 Interrupt Group	IRQ
Output Compare 0 Interrupt Priority	Disabled
Output Compare 0 Interrupt Group	IRQ
Output Compare 1 Interrupt Priority	Disabled
Output Compare 1 Interrupt Group	IRQ

Figure 54 : Configuring Interrupts in the Stacks tab

6.4.1 Creating Interrupts from the Interrupts Tab

On the **Interrupts** tab, the interrupt of the driver selected in the **Stacks** tab is registered.



Event	ISR
CMT0_CMI (CMT0 Compare match)	cmt0_handler

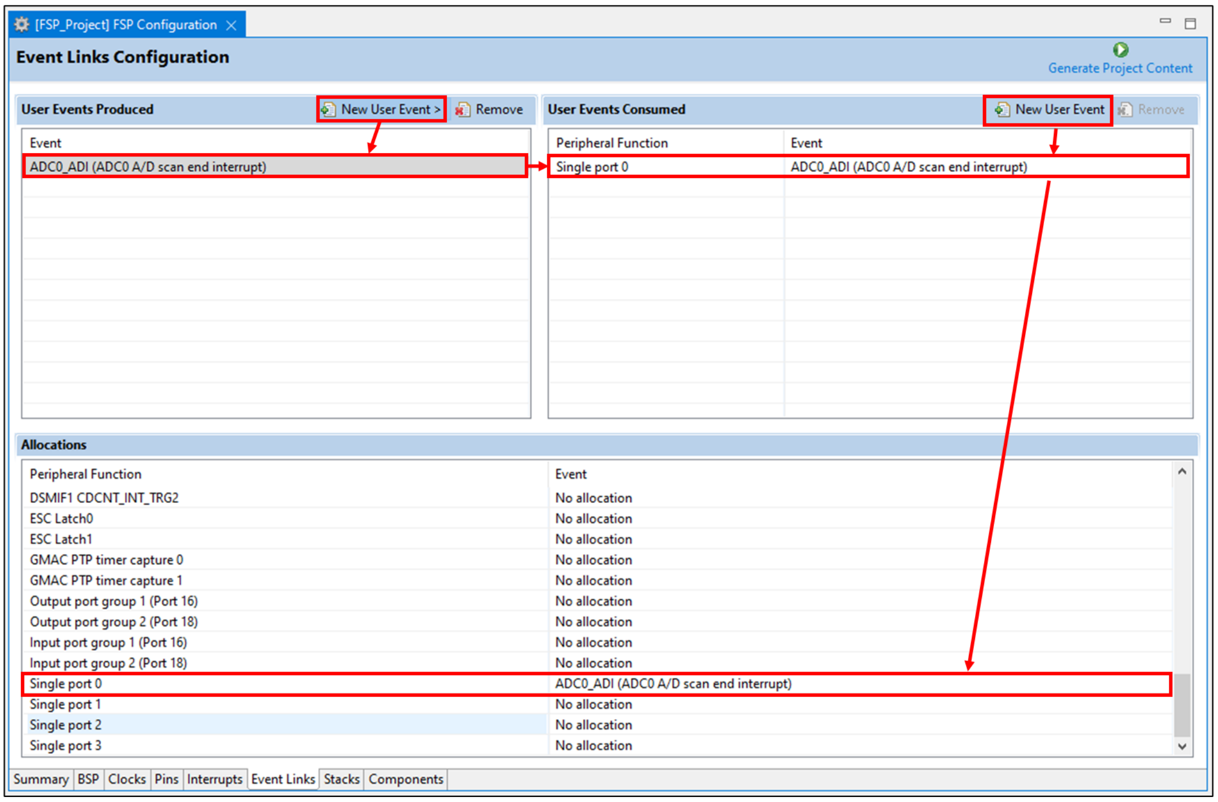
Interrupt	Event	ISR
53	CMT0_CMI (CMT0 Compare match)	cmt0_handler

Figure 55 : Configuring interrupt in Interrupt Tab

And the user can add a peripheral interrupt created by the user’s own. This can be done by adding a new event via the **New User Event** button.

6.4.2 Viewing Event Links

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



Event
ADC0_ADI (ADC0 A/D scan end interrupt)

Peripheral Function	Event
Single port 0	ADC0_ADI (ADC0 A/D scan end interrupt)

Peripheral Function	Event
DSMIF1 CDCNT_INT_TRG2	No allocation
ESC Latch0	No allocation
ESC Latch1	No allocation
GMAC PTP timer capture 0	No allocation
GMAC PTP timer capture 1	No allocation
Output port group 1 (Port 16)	No allocation
Output port group 2 (Port 18)	No allocation
Input port group 1 (Port 16)	No allocation
Input port group 2 (Port 18)	No allocation
Single port 0	ADC0_ADI (ADC0 A/D scan end interrupt)
Single port 1	No allocation
Single port 2	No allocation
Single port 3	No allocation

Figure 56 : Viewing Event Links

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

Note:

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

6.5 Adding and Configuring HAL Drivers

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

1. Click on the HAL/Common icon in the **Stacks** pane. The Modules pane changes to **HAL/Common** Stacks.
2. Click New Stack to see a drop-down list of HAL level drivers available in the FSP.
3. Select a driver from the menu **New Stack > Driver**.

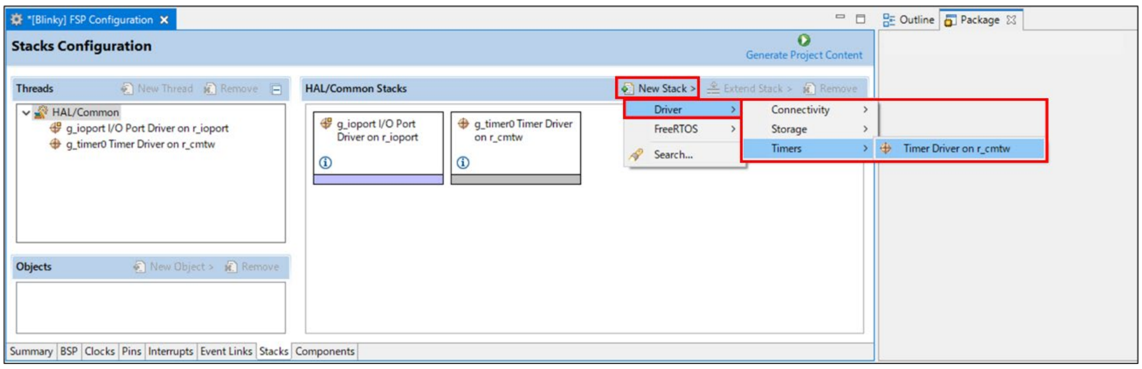


Figure 57 : e2 studio Project configurator - Adding drivers

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

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

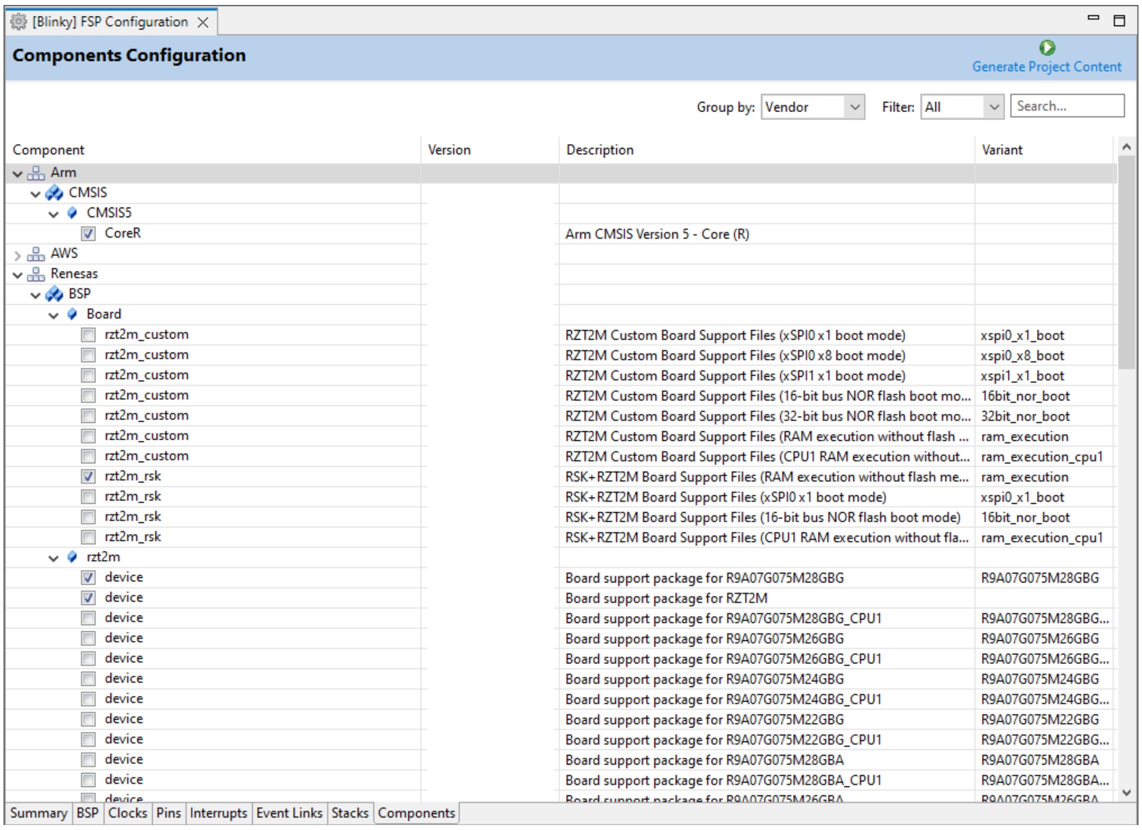
- The selected driver module and its files to the rzt/fsp directory
- The main() function and configuration structures and header files for your application as shown in the table below.

File	**Contents**	**Overwritten by Generate Project Content?**
rzt_gen/main.c	Contains main() calling generated and user code. When called, the BSP already has Initialized the MPU.	Yes
rzt_gen/hal_data.c	Configuration structures for HAL Driver only modules.	Yes
rzt_gen/hal_data.h	Header file for HAL driver only modules.	Yes
src/hal_entry.c	User entry point for HAL Driver only code. Add your code here.	No

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

6.6 Reviewing and Adding Components

The **Components** tab enables the individual modules required by the application to be included or excluded. Modules common to all RZ/T MPU projects are preselected. All modules that are necessary for the modules selected in the **Stacks** tab are included automatically. You can include or exclude additional modules by ticking the box next to the required component.



Group by: Vendor Filter: All Search...

Component	Version	Description	Variant
Arm			
CMSIS			
CMSIS5			
✓ CoreR		Arm CMSIS Version 5 - Core (R)	
AWS			
Renesas			
BSP			
Board			
☐ rzt2m_custom		RZT2M Custom Board Support Files (xSPI0 x1 boot mode)	xspi0_x1_boot
☐ rzt2m_custom		RZT2M Custom Board Support Files (xSPI0 x8 boot mode)	xspi0_x8_boot
☐ rzt2m_custom		RZT2M Custom Board Support Files (xSPI1 x1 boot mode)	xspi1_x1_boot
☐ rzt2m_custom		RZT2M Custom Board Support Files (16-bit bus NOR flash boot mo...	16bit_nor_boot
☐ rzt2m_custom		RZT2M Custom Board Support Files (32-bit bus NOR flash boot mo...	32bit_nor_boot
☐ rzt2m_custom		RZT2M Custom Board Support Files (RAM execution without flash ...	ram_execution
☐ rzt2m_custom		RZT2M Custom Board Support Files (CPU1 RAM execution without...	ram_execution_cpu1
✓ rzt2m_rsk		RSK+RZT2M Board Support Files (RAM execution without flash me...	ram_execution
☐ rzt2m_rsk		RSK+RZT2M Board Support Files (xSPI0 x1 boot mode)	xspi0_x1_boot
☐ rzt2m_rsk		RSK+RZT2M Board Support Files (16-bit bus NOR flash boot mode)	16bit_nor_boot
☐ rzt2m_rsk		RSK+RZT2M Board Support Files (CPU1 RAM execution without fla...	ram_execution_cpu1
rzt2m			
✓ device		Board support package for R9A07G075M28GBG	R9A07G075M28GBG
✓ device		Board support package for RZT2M	
☐ device		Board support package for R9A07G075M28GBG_CPU1	R9A07G075M28GBG...
☐ device		Board support package for R9A07G075M26GBG	R9A07G075M26GBG
☐ device		Board support package for R9A07G075M26GBG_CPU1	R9A07G075M26GBG...
☐ device		Board support package for R9A07G075M24GBG	R9A07G075M24GBG
☐ device		Board support package for R9A07G075M24GBG_CPU1	R9A07G075M24GBG...
☐ device		Board support package for R9A07G075M22GBG	R9A07G075M22GBG
☐ device		Board support package for R9A07G075M22GBG_CPU1	R9A07G075M22GBG...
☐ device		Board support package for R9A07G075M28GBA	R9A07G075M28GBA
☐ device		Board support package for R9A07G075M28GBA_CPU1	R9A07G075M28GBA...
☐ device		Board support package for R9A07G075M26GBA	R9A07G075M26GBA

Figure 58 : Components Tab

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

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

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

Appendix. Known Issues

This chapter describes the known issues regarding the current version of FSP and related platform software.

Most of the issues may requires users to follow some manual operations to resolve the issues or to avoid the problems caused by the issues. Please follow the operations in the description of the issues if you use the features related the issues.

FSP Configuration

This section describes the known issues regarding the FSP Configuration.

e2 studio and FSP SC have various configuration features worked on GUI with FSP.

Regarding the overview of each configuration feature (GUI tab) provided as a part of FSP configuration in e2 studio and FSP SC, please see the chapter 6. “FSP Configuration Users Guide”.

Stacks tab

- **“r_gmac” may be showed as “r_ether” incorrectly.**

In Stacks tab, “r_gmac” may be showed as “r_ether” incorrectly. Please replace the “r_ether” as “r_gmac”.

FSP Modules

This section describes the known issues regarding the FSP modules.

The FSP provides HAL drivers and BSP configured by FSP Configuration on e2 studio and FSP SC.

Regarding their features, usage notes and API references, please see the related file “FSP Documentation”.

Serial Communication Interface (SCI) UART

- **“R_SCI_UART_BaudCalculate()” of “r_sci_uart” module properly works ONLY when its clock source is SCInASYNCCLK and its frequency is 96MHz.**

The “R_SCI_UART_BaudCalculate()” of “r_sci_uart” module works ONLY when its clock source is “SCInASYNCCLK” and its frequency is “96MHz”; therefore, when the module uses “PCLKM” as its clock source or the frequency is not 96MHz, the API function will be not work properly.

Serial Peripheral Interface

- **“R_SPI_CalculateBitrate()” of “r_spi” module properly works ONLY when its clock source is SPInASYNCCLK and its frequency is 96MHz.**

The “R_SPI_BaudCalculate()” of “r_spi” module works ONLY when its clock source is “SPInASYNCCLK” and its frequency is “96MHz”; therefore, when the module uses “PCLKM” as its clock source or the frequency is not 96MHz, the API function will be not work properly.

Appendix. Tool Software Limitations

This section describes the limitations regarding the tool software (e2 studio, FSP SC) to create and debug FSP projects.

FSP Smart Configurator

- **When installing, please install in the following installation folder.**

C:\Renesas\rzt\sc_v2022-04_fsp_v1.0.0

When sharing a project between different PCs, build errors will occur if the installation folders are different.

e2 studio

- **If [Reset] button is pushed immediately after setting a breakpoint, the program will not stop at the set breakpoint.**

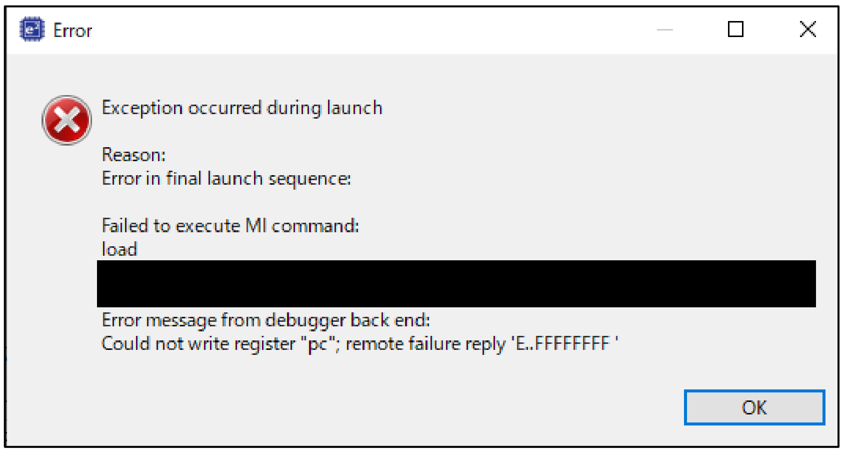
Please run the program once before pressing the [Reset] button.

- **Before pressing the reset button on the board, disconnect the e2 studio connection first.**

If the reset button is pressed on the board while connected with e2 studio, debugging will not be able to continue.

- **If the following error is displayed when connecting the debugger or when downloading the program, click the [OK] button to close the dialog and try connecting again.**

An error has occurred because the program download to the NOR flash area has failed. The download is successful on the second connection.



```
Exception occurred during launch

Reason:
Error in final launch sequence:

Failed to execute MI command:
load
[redacted]
Error message from debugger back end:
Could not write register "pc"; remote failure reply 'E..FFFFFFFF '
```

(Continued on next page)

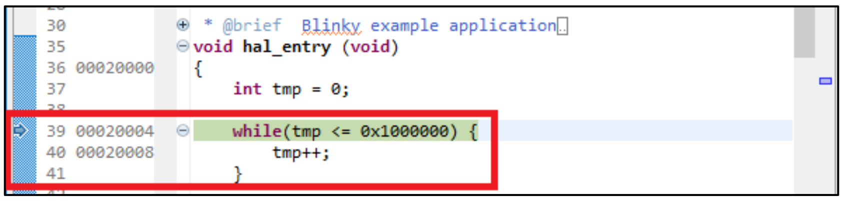
- **The user program cannot be stopped immediately after the device boot process.**

Immediately after the device boot process (boot code), the program cannot be stopped at the beginning of the user program (loader program).

When debugging, please note the following two points.

1. Operation of [Reset] button

When [Reset] button is pushed, the debugger stops the user program (loader program) about 100ms after the device boot process (boot code). When debugging the program immediately after the boot process (boot code), insert the loop part at the beginning of the "hal_entry" function as shown below.



```
 * @brief  Blinky example application
void hal_entry (void)
{
    int tmp = 0;

    while(tmp <= 0x1000000) {
        tmp++;
    }
```

2. Operation immediately after connecting to the debugger

It looks like the program stopped at the entry point, but it actually forces the entry point to set the PC value after the program runs. The program may not operate normally even if it is executed as it is. Please press the [Reset] button before running the program.

Revision History

Rev.	Date	Description	
		Page	Summary
1.0.0	June.7.22	-	First Edition issued

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 V_{IL} (Max.) and V_{IH} (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 V_{IL} (Max.) and V_{IH} (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.