

RX Family

Guide for Migration to RX FSP

Introduction

This application note gives an overview of the RX Flexible Software Package (RX FSP) from Renesas and describes how to port a project that was created by using FIT modules and the code generation feature of the Smart Configurator, both of which are software tools provided by Renesas, to a project generated by using the newer RX FSP.

Target Device

- RX family

Contents

1. Software for the RX Family	6
2. What is the RX FSP?	6
2.1 FSP Modules	6
2.2 FSP Development Environment and FSP Stacks	7
2.3 Procedure for Creating an FSP Project	8
2.4 FSP Project Window	15
2.4.1 Project Explorer	16
2.4.2 RX Project Configuration Editor View	16
2.4.3 Visualization View	17
2.4.4 Properties View	18
2.5 Locations of Settings for an FSP Project	18
2.5.1 Basic Configuration	18
2.5.2 Clock Configuration	20
2.5.3 Pin Configuration	21
2.5.4 Interrupt Configuration	22
2.5.5 Event Link Configuration	22
2.5.6 Linker Section Mapping Configuration	23
2.5.7 Component (Module) Configuration and Stack Configuration	23
2.6 Structure of the FSP Project Files	25
2.7 How to Check the Information on the API of an FSP Module	38
3. Migration to the FSP Development Environment	39
3.1 Differences in Handling between the Smart Configurator and FSP Configuration Perspectives	39
3.1.1 Checking the Overview and Component State	40
3.1.2 Selecting the Board and Device	41
3.1.3 Clock Configuration	42
3.1.4 Debug Interface Configuration	44
3.1.5 Component Configuration	45
3.1.6 Pin Configuration	47
3.1.7 Interrupt Configuration	49
3.1.8 Event Link Configuration	50
4. Comparison of Software Specifications	52
4.1 Comparison of the Functions Supported by the Code Generator of the Smart Configurator, FIT Modules, and FSP Modules	52
4.1.1 Comparison of the Supported Peripheral Functions	52
4.1.2 Comparison of the Supported Middleware Functions	55
4.2 Comparison of Initialization	57
4.2.1 Comparison of the BSP Specifications between the FIT and FSP Modules	57
4.2.2 Alternative Measures for Processing Not Available in the FSP Module	59
4.2.2.1 CPU-Related Processing	59
4.2.2.2 Arithmetic Unit for Trigonometric Functions	59
4.2.2.3 Waiting for Stabilization of the VBATT Voltage	59
4.2.2.4 Stopping the Clocks Not to be Used as Clock Sources	60

4.2.2.5	Initializing the STDIO Library	60
4.2.2.6	Initializing Bus Priority	60
4.2.2.7	Enabling or Disabling the Use of the User Stack	61
4.2.2.8	Switching to User Mode	61
4.2.2.9	Closing the STDIO Library	61
4.2.3	Initializing the BSP in the FSP.....	62
4.3	Comparison of the Individual Module Specifications.....	65
4.3.1	Compare Match Timer (CMT)	65
4.3.1.1	CMT Module of the FSP	65
4.3.1.2	Comparison of the CMT Modules of FIT and the FSP	66
4.3.1.3	Comparison of the API Functions for the CMT of the Code Generator and those of the FSP Module	67
4.3.2	Serial Communications Interface (RSCI)	68
4.3.2.1	SCI UART Module of the FSP	68
4.3.2.2	Comparison of the RSCI Module of FIT and the UART Module of the FSP	69
4.3.2.3	Comparison of the API Functions for the SCI/SCIF Asynchronous Mode of the Code Generator and those for the UART Module of the FSP	70
5.	Steps in Porting a Pre-FSP Project to an FSP Project	72
5.1	Creating a New FSP Project	72
5.2	Porting the Basic Settings	72
5.3	Porting the Clock Configuration.....	76
5.4	Porting Modules.....	77
5.5	Porting an Interrupt Configuration	80
5.5.1	Interrupt Configuration by Adding a Module.....	81
5.5.2	Interrupt Configuration through Direct Addition.....	83
5.6	Porting a Pin Configuration	84
5.7	Porting the Source Code	87
5.8	Adding iodef.h.....	87
5.9	Replacing Processing with FSP Module Processing	87
5.9.1	Replacing Processing by FIT Modules with Processing by FSP Modules.....	87
5.9.2	Replacing Processing Created by the Code Generator with Processing by FSP Modules	88
6.	Examples of Porting FIT Applications to FSP Projects.....	89
6.1	Example of Porting an FIT-Based CMT Sample Program	89
6.1.1	Structure of the Folders and Files in the Source Project and the Targets for Porting	89
6.1.2	Overview of the Processing in the Source Project	89
6.1.3	Creating the Destination Project for Porting.....	92
6.1.4	Adding CMT Modules.....	92
6.1.5	Structure of the Folders and Files in the Project after Porting and the Targets for Modification	94
6.1.6	Key Points in Porting	94
6.1.7	Processing after Replacement of the Target Files for Modification	94
6.1.8	Replacement of Processing by the FIT Modules with Processing by the FSP Modules in this Example of Porting	97
6.1.8.1	Comparison of the Main Processing	98
6.1.8.2	Comparison of the Callback Functions	98
6.2	Example of Porting an FIT-Based RSCI Sample Program	99
6.2.1	Structure of the Folders and Files in the Source Project and the Targets for Porting	99

6.2.2	Overview of the Processing in the Source Project	99
6.2.3	Creating the Destination Project for Porting	105
6.2.4	Adding an SCI Module	105
6.2.5	Adding the Port Settings.....	111
6.2.6	Structure of the Folders and Files in the Project after Porting and the Targets for Modification	113
6.2.7	Key Points in Porting	113
6.2.8	Processing after Replacement of the Target Files for Modification	113
6.2.9	Replacement of Processing by the FIT Module with Processing by the FSP Module in this Example of Porting	117
6.2.9.1	Comparison of the Main Processing	118
6.2.9.2	Comparison of the Callback Function	119
7.	Examples of Porting Applications Created by the Code Generator to FSP Projects.....	120
7.1	Example of Porting a Sample Program Using the CMT Component Created by the Code Generator	120
7.1.1	Structure of the Folders and Files in the Source Project and the Targets for Porting	120
7.1.2	Overview of the Processing in the Source Project	120
7.1.3	Creating the Destination Project for Porting	126
7.1.4	Adding a CMT Module	126
7.1.5	Adding an ICU Module	128
7.1.6	Adding the Port Settings.....	133
7.1.7	Structure of the Folders and Files in the Project after Porting and the Targets for Modification	137
7.1.8	Key Points in Porting	137
7.1.9	Processing after Replacement of the Target Files for Modification	138
7.1.10	Replacement of Processing Created by the Code Generator with Processing by the FSP Modules in this Example of Porting	140
7.1.10.1	Comparison of the Processing for Initializing Modules	140
7.1.10.2	Comparison of the Main Processing	141
7.1.10.3	Comparison of the Callback Functions	141
7.2	Example of Porting a Sample Program Using the SCI Component Created by the Code Generator	142
7.2.1	Structure of the Folders and Files in the Source Project and the Targets for Porting	142
7.2.2	Overview of the Processing in the Source Project	143
7.2.3	Creating the Destination Project for Porting	147
7.2.4	Adding an SCI Module	147
7.2.5	Adding the Port Settings.....	152
7.2.6	Structure of the Folders and Files in the Project after Porting and the Target for Modification	155
7.2.7	Key Points in Porting	155
7.2.8	Processing after Replacement of the Target File for Modification	156
7.2.9	Replacement of Processing Created by the Code Generator with Processing by the FSP Module in this Example of Porting	158
7.2.9.1	Comparison of the Processing for Initializing Modules	159
7.2.9.2	Comparison of the Main Processing	159
7.2.9.3	Comparison of the Callback Functions	160
8.	Examples of Porting Projects Created without Using the Smart Configurator to FSP Projects	161
8.1	Example of Porting a Sample Program that Does Not Use Interrupt Functions	161
8.1.1	Structure of the Folders and Files in the Source Project and the Targets for Porting	161

8.1.2	Overview of the Processing in the Source Project	161
8.1.3	Creating the Destination Project for Porting	164
8.1.4	Structure of the Folders and Files in the Project after Porting and the Targets for Modification	164
8.1.5	Key Points in Porting	165
8.1.6	Processing after Replacement of the Target Files for Modification	165
8.2	Example of Porting a Sample Program Using Interrupt Functions	168
8.2.1	Structure of the Folders and Files in the Source Project and the Targets for Porting	168
8.2.2	Overview of the Processing in the Source Project	168
8.2.3	Creating the Destination Project for Porting	173
8.2.4	Adding the Interrupt Configuration	174
8.2.5	Structure of the Folders and Files in the Project after Porting and the Targets for Modification	176
8.2.6	Key Points in Porting	177
8.2.7	Processing after Replacement of the Target Files for Modification	177
9.	Notes on Porting to an FSP Project	182
9.1	Addition and Conflicts of a BSP	182
9.2	Differences between and Conflicts with the I/O Register Definition Files	182
9.3	Porting to a Device that Does Not Provide the Same Peripheral Functions as the Source Device ...	186
10.	Reference Documents	187
	Revision History	188

1. Software for the RX Family

Renesas has been providing two types of software package to be used in the development of user systems: the Code Generator and various FIT software modules. The Code Generator is a feature of the [RX Smart Configurator](#) and automatically generates driver code. FIT software modules are a group of modules that are based on the specifications of the [Firmware Integration Technology](#) (hereafter called FIT) and are provided in the form of the [RX Driver Package](#). For new devices of the RX family to be released from January of 2026, we will be providing a new software package, the [RX Flexible Software Package](#) (hereafter called the RX FSP), to offer more consistent and competitive experiences to users, so please make use of it.

2. What is the RX FSP?

The Renesas Flexible Software Package (FSP) is a software package designed to deliver user-friendly, scalable, and high-quality software for embedded system designs employing MCU devices of the Renesas RX family. The FSP contains various software components such as the board support package (BSP), hardware abstraction layer (HAL) drivers for access to peripheral functions, FreeRTOS for supporting multitasking, and basic middleware stacks.

- ✓ The FSP supports the new Protect Zone and other innovative security features to prevent the leakage of important data and programs and enables the quick development of secure IoT devices in various ways with the use of HAL drivers, FreeRTOS, and other versatile middleware stacks.
- ✓ The FSP uses FreeRTOS, supporting multi-threaded application environments as the base for development, but it also gives flexibility to the user, such as bare-metal programming and the use of a desired RTOS, existing software resources, and third-party ecosystem solutions.
- ✓ The e² studio integrated development environment provides an intuitive configurator and supports intelligent code generation through use of the FSP.

FreeRTOS is a trademark of Amazon Web Services, Inc.

<http://www.openrtos.net/copyright.html>

2.1 FSP Modules

The FSP consists of the board support package (BSP), peripheral function modules, and middleware modules. Modules are the core components of the FSP. The simplest FSP application consists of a BSP and a single module above which the user application is placed. Note that the structure of the BSP in the FSP differs from that of the BSP of FIT.

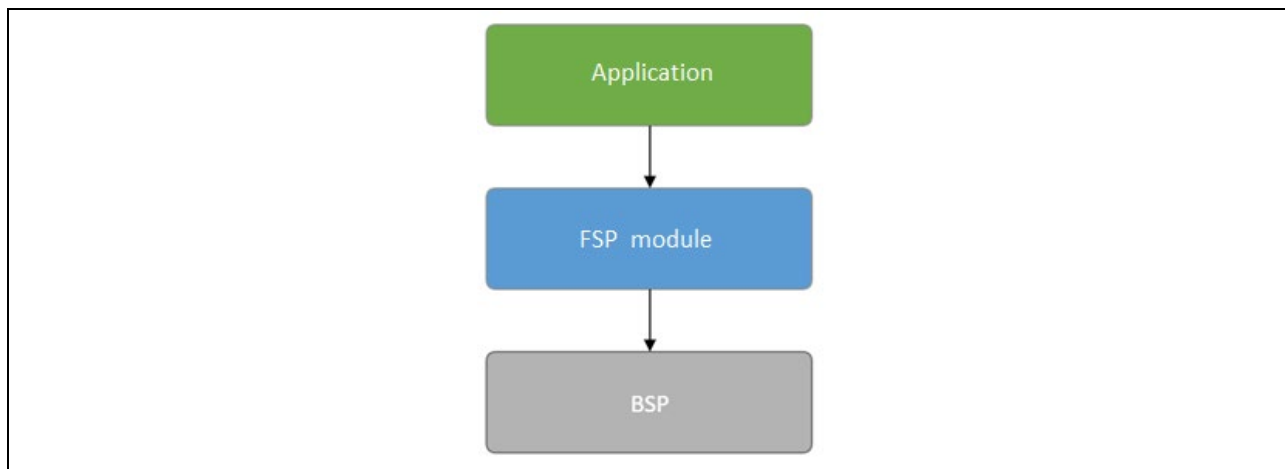


Figure 2-1 Minimum Structure of an FSP Application

Modules can handle various operations but all modules share the basic concept of a module providing functionality to upper-layer software and sending requests to lower-layer software or peripheral function hardware.

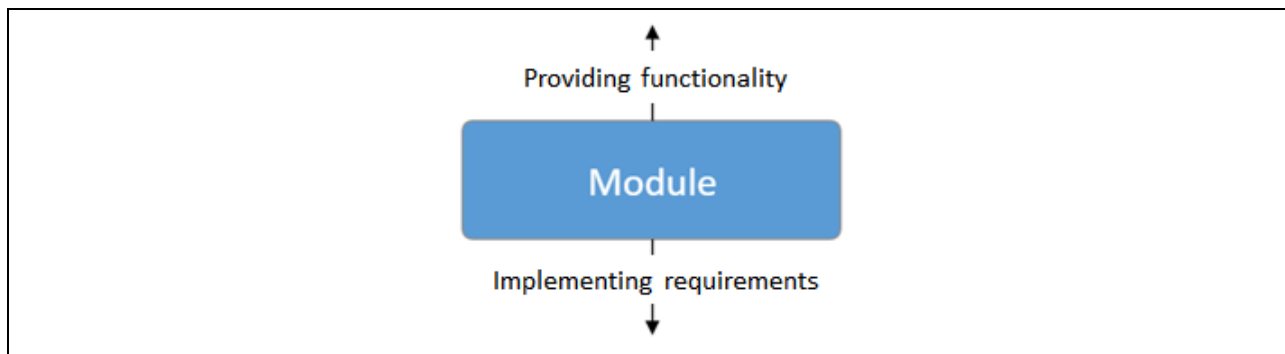


Figure 2-2 FSP Module

Functional use cases determine the amounts of functionality provided by modules.

The BSP is the foundation for FSP modules and provides features such as determination of the MCU to be used and configuration of clocks, interrupts, and pins.

FSP modules do not have functional compatibility with FIT modules or the source code generated by the RX Smart Configurator, the software Renesas currently provides. The APIs also have different specifications. In addition, the structure of the register definition files used in a C/C++ project created with the FSP (hereafter called an FSP project) differs from that of the register definition file (iodefine.h) used in a project created with FIT modules and the Code Generator of the RX Smart Configurator. The FSP provides the register definitions as "renesas.h", which is composed of multiple files of register definitions for the individual peripheral functions. For the differences in the specifications between modules, see Chapter 4, Comparison of Software Specifications.

2.2 FSP Development Environment and FSP Stacks

The FSP offers many tools for improving efficiency in developing projects for MCU devices of the Renesas RX family. The e² studio IDE provides a familiar development environment for managing all major steps of development such as project creation, module selection and configuration, code development, code generation, and debugging.

FSP stacks are a feature of the FSP. An FSP stack is a group of modules that are necessary to implement a functionality or a use case. With the FSP, a combination of modules forms an FSP stack. An FSP stack can be configured by matching the requirements of a selected module with the offerings of another module. For example, a use case involving SPI communications may use interrupts or use the DTC or DMAC. To use the DTC, the SPI module requires a module that provides the transfer interfaces for data transmission and reception. The requirements of the transfer interfaces can be satisfied by the DTC driver module. When the DTC is to be used in UART communications or SD/MMC communications as well as SPI communications, configuring the FSP stacks accordingly enables sharing of the DTC module by multiple modules.

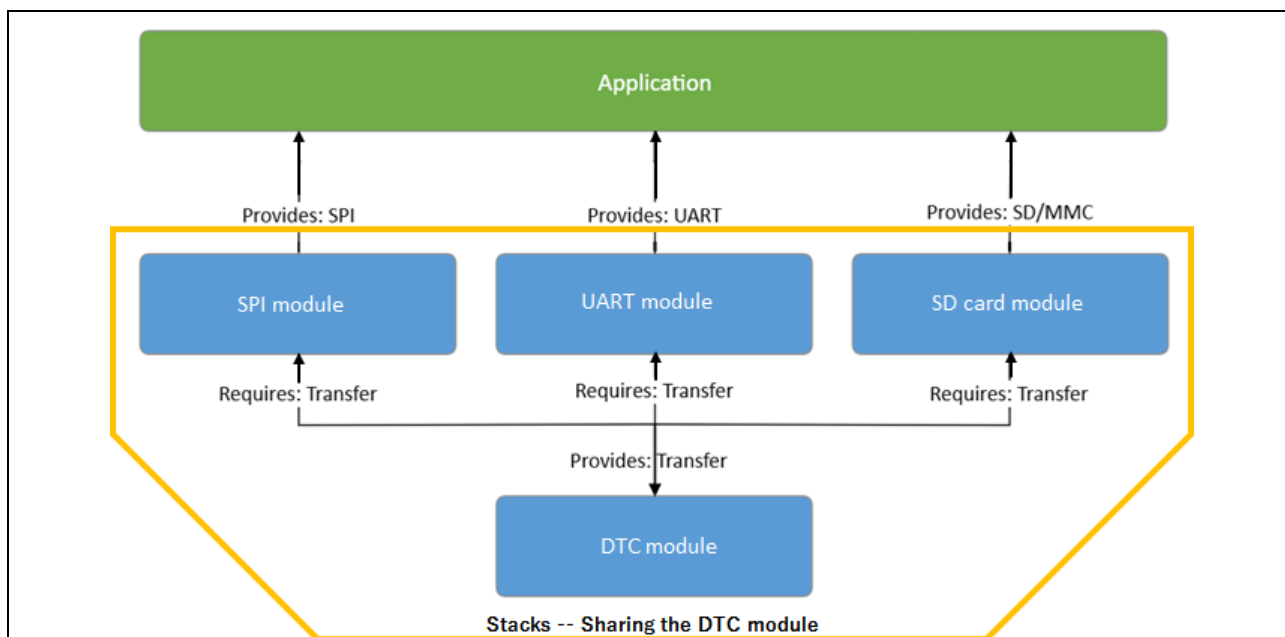


Figure 2-3 Overview of FSP Stacks

The ability to form a stack of modules ensures the flexibility of the architecture as a whole. When multiple modules include the same functionality, problems will occur if multiple user designs with different application features require the use of that functionality. To ensure the reusability of the modules, a dependent module has to be replaceable by other modules that offer the same functionality. The FSP architecture provides this flexibility by using FSP interfaces for swapping modules in and out.

2.3 Procedure for Creating an FSP Project

The following describes the procedure for creating a C/C++ project by using the FSP.

(1) Starting the e² studio

Start the e² studio to specify a workspace. After it has started, select [File] → [New] → [Renesas C/C++ Project] → [Renesas RX] to open the project creation wizard.

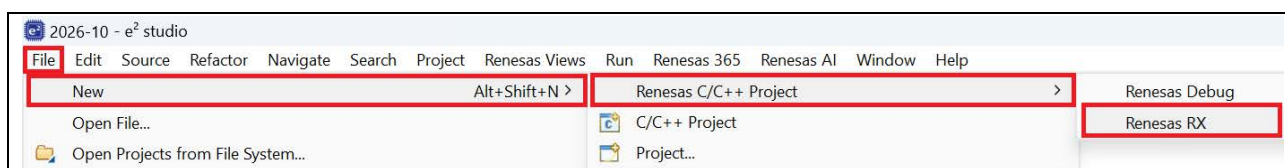


Figure 2-4 Creating a New Project

(2) Selecting an FSP project

Select [Renesas RX C/C++ FSP Project] in the project creation wizard and click the [Next] button.

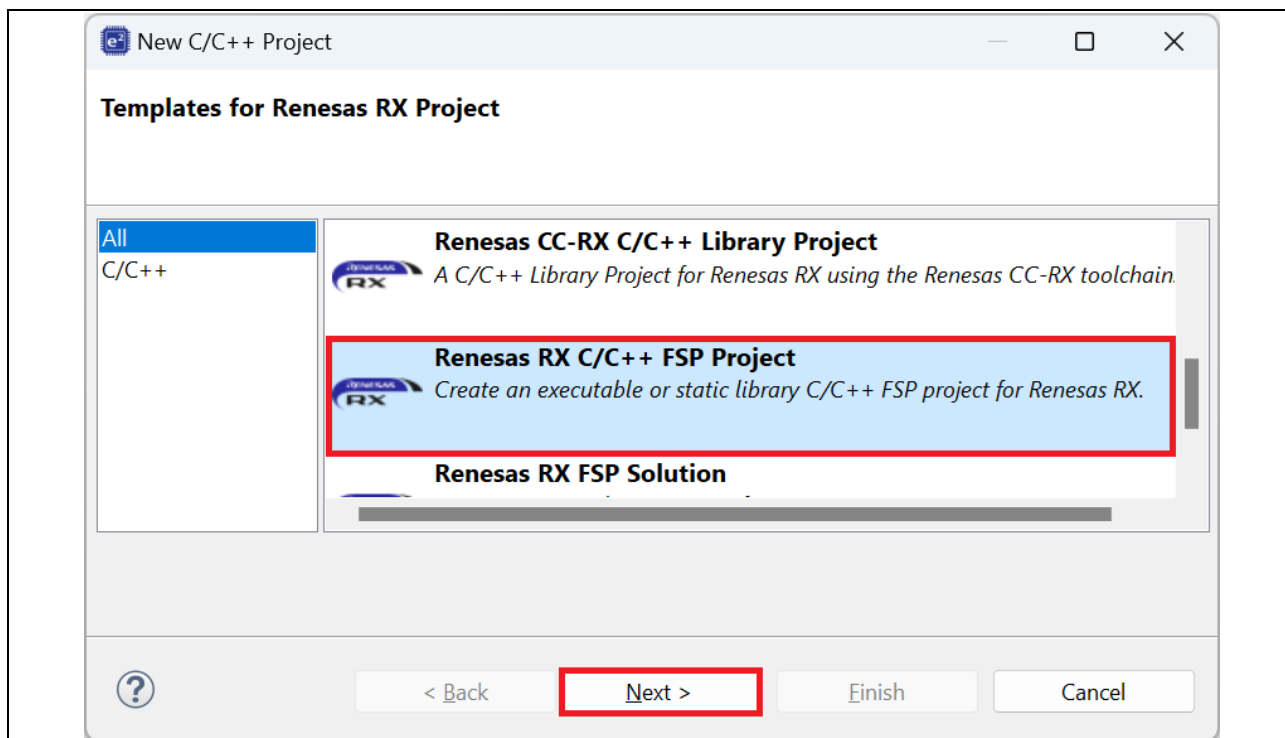


Figure 2-5 Selecting a Project Template

(3) Specifying a project name

Enter a project name and click on the [Next] button to proceed to the next step.

Example of a project name for an RX C/C++ FSP project: "FSP_example"

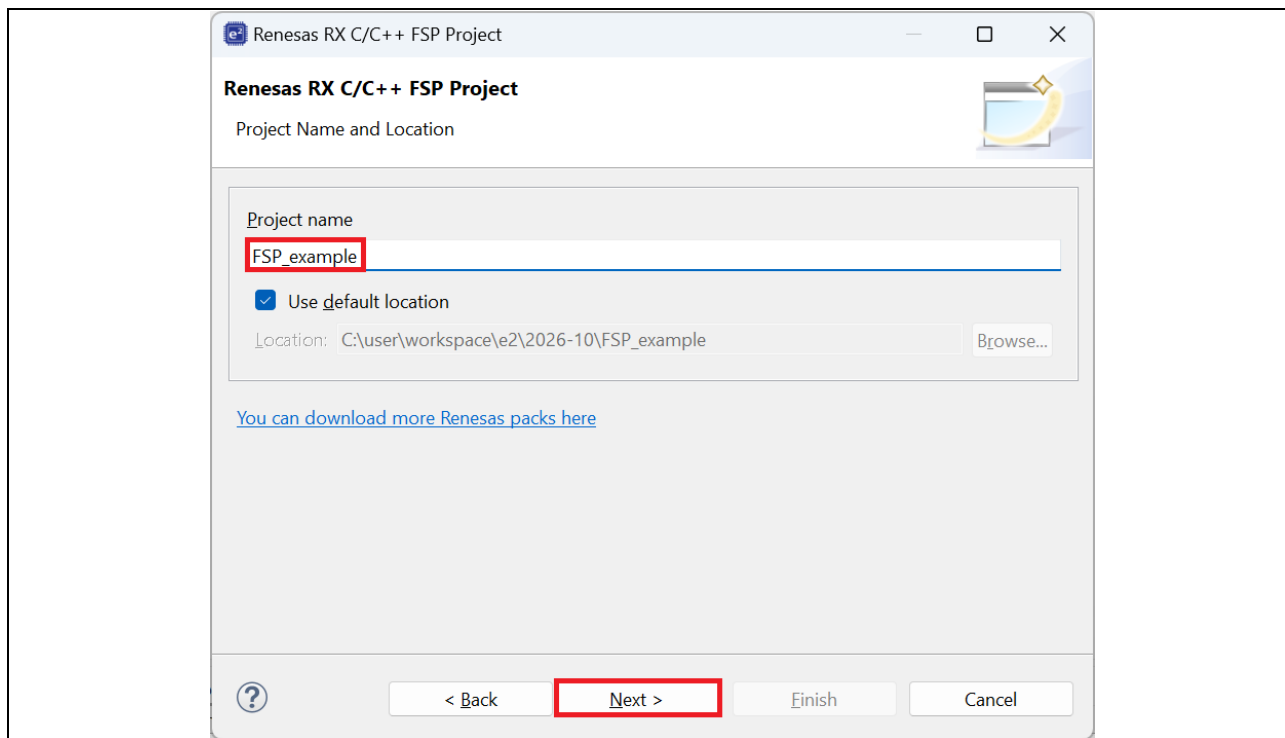


Figure 2-6 Creating a Renesas RX C/C++ Project

(4) Selecting the device, toolchain, and debugger

Make selections for [FSP Version], [Board], [Device], and [Language] under [Device Selection].

Selecting a board automatically sets the device to the one mounted on the board.

Select the type of the project for [IDE Project Type], the toolchain and version to be used for [Toolchains], and the debugger to be used for [Debugger]. After that, click on the [Next] button.

Renesas RX C/C++ FSP Project

Device and Tools Selection

Device Selection

FSP Version: 1.0.0

Board: EK-RX74M

Device: R5K574MCBxAE

Core: RXv3e

Language: ☒ C ☐ C++

Board Description

Evaluation kit for RX74M MCU Group

Visit <https://renesas.com/ek-rx74m> to get kit user's manual, quick start guide, errata, design package, example projects, etc.

Device Details

Protect Zone	No
Pins	320

IDE Project Type

e² studio managed build

Debugger

E2 Lite (RX)

Toolchains

GCC for Renesas RX

Renesas CC-RX

v3.08.00 [Manage Toolchains...](#)

[? < Back](#) **Next >** [Finish](#) [Cancel](#)

Figure 2-7 Selecting the Device, IDE Project Type, Toolchain, and Debugger

(5) Selecting the project type

Select the type of project for [Project type Selection].

Leave it as "Flat (Non-Protect Zone)" and click on the [Next] button.

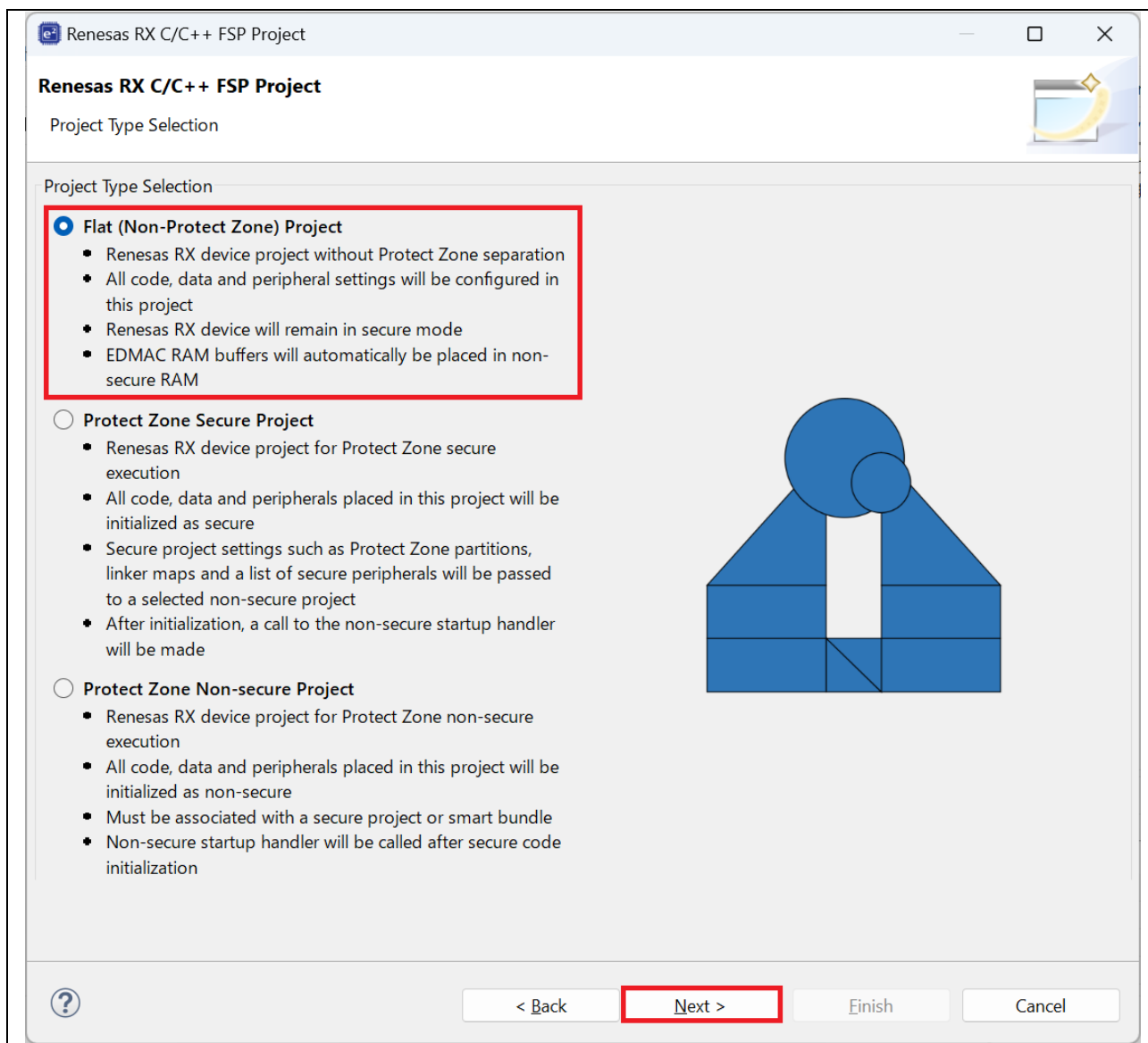


Figure 2-8 Selecting the Project Type

(6) Selecting "Preceding Project" or "Smart Bundle Selection"

Leave the selection as "None" and click on the [Next] button.

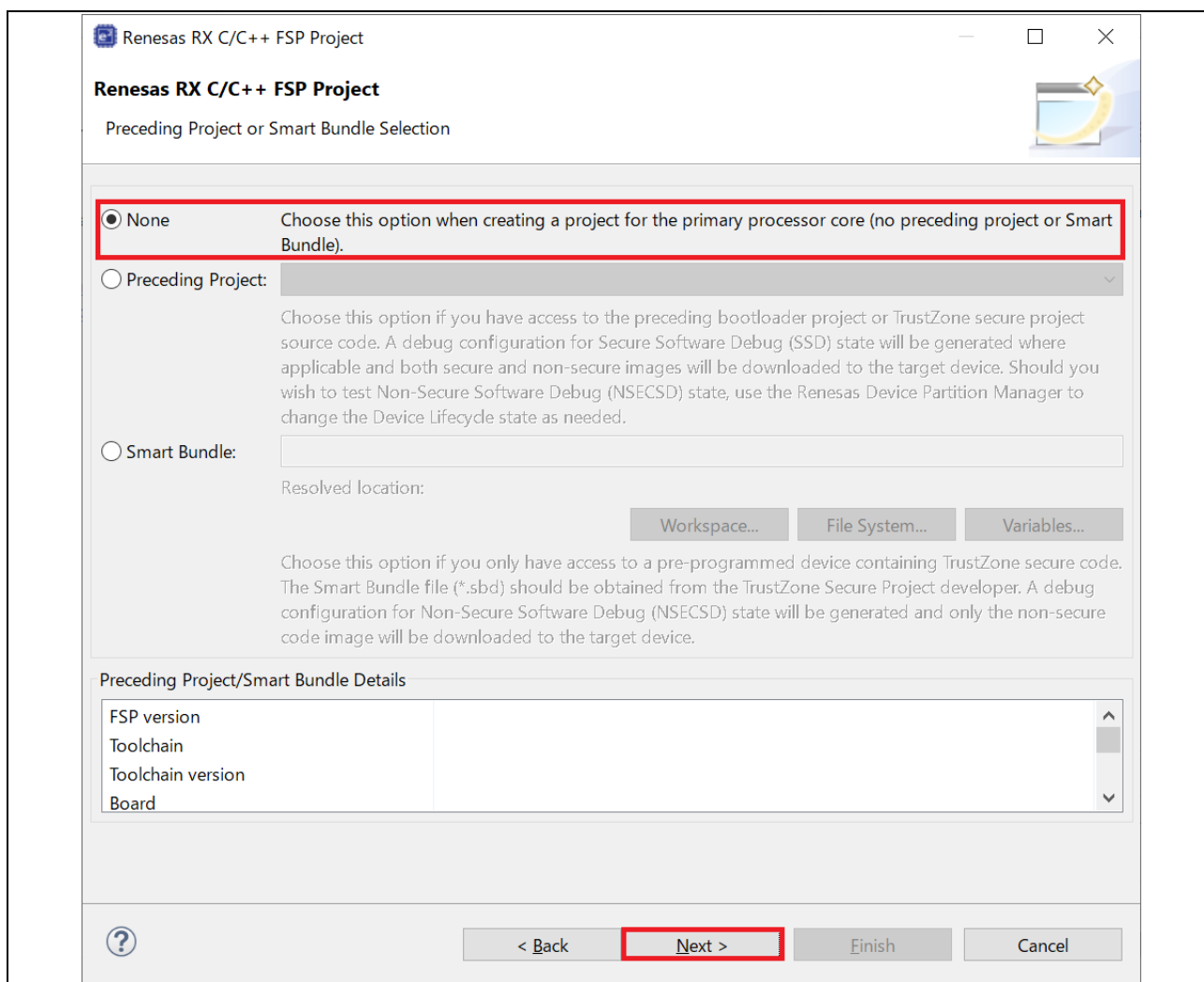


Figure 2-9 Selecting "Preceding Project" or "Smart Bundle Selection"

(7) Selecting the project type and RTOS

Select the type of project to be created for [Build Artifact Selection].

Select whether to use the RTOS for [RTOS Selection]. Leave it as "No RTOS" and click on the [Next] button.

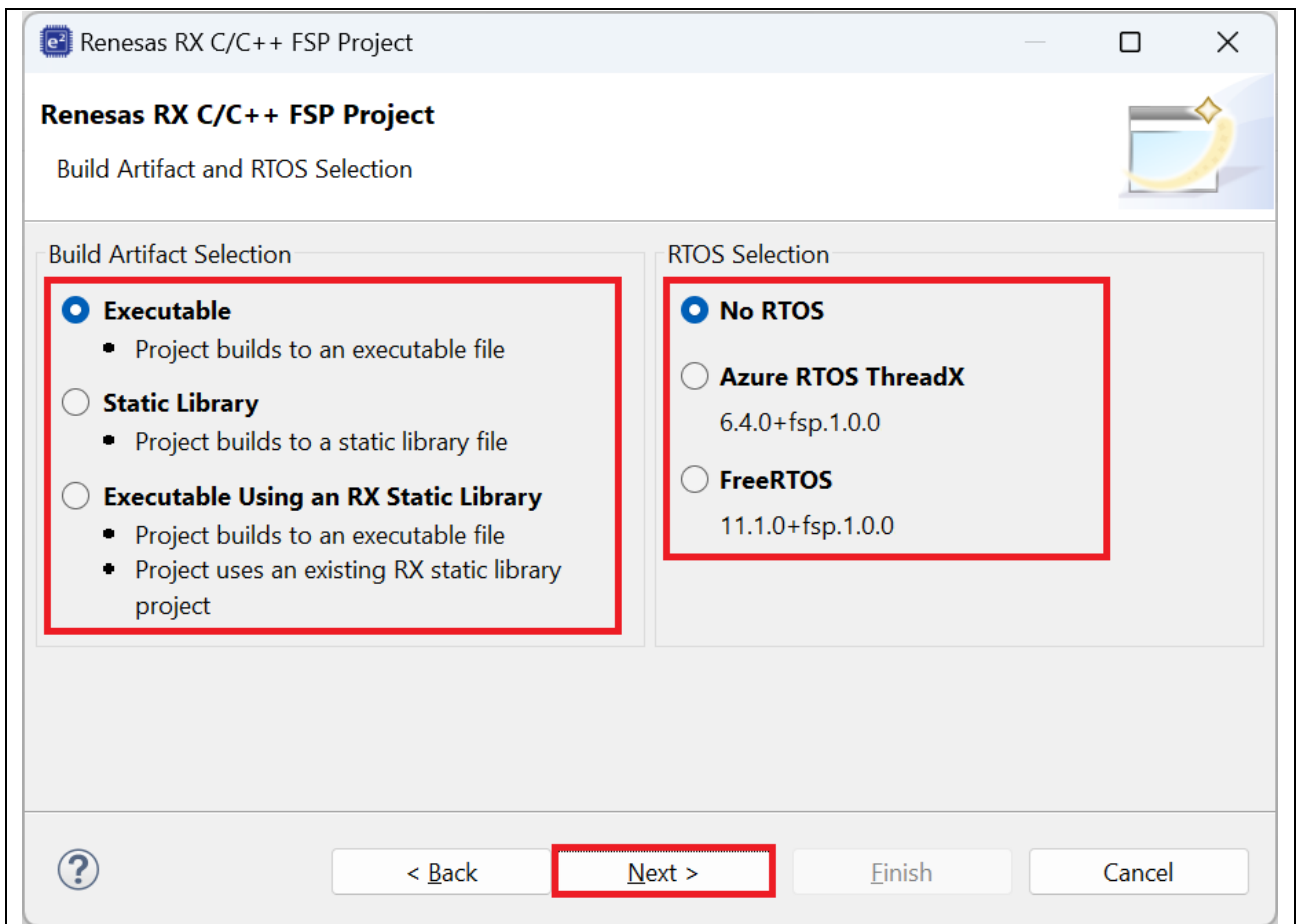


Figure 2-10 Selecting the Project Type and RTOS

(8) Selecting a project template

Select a template for [Project Templates Selection]. Click on the [Finish] button.

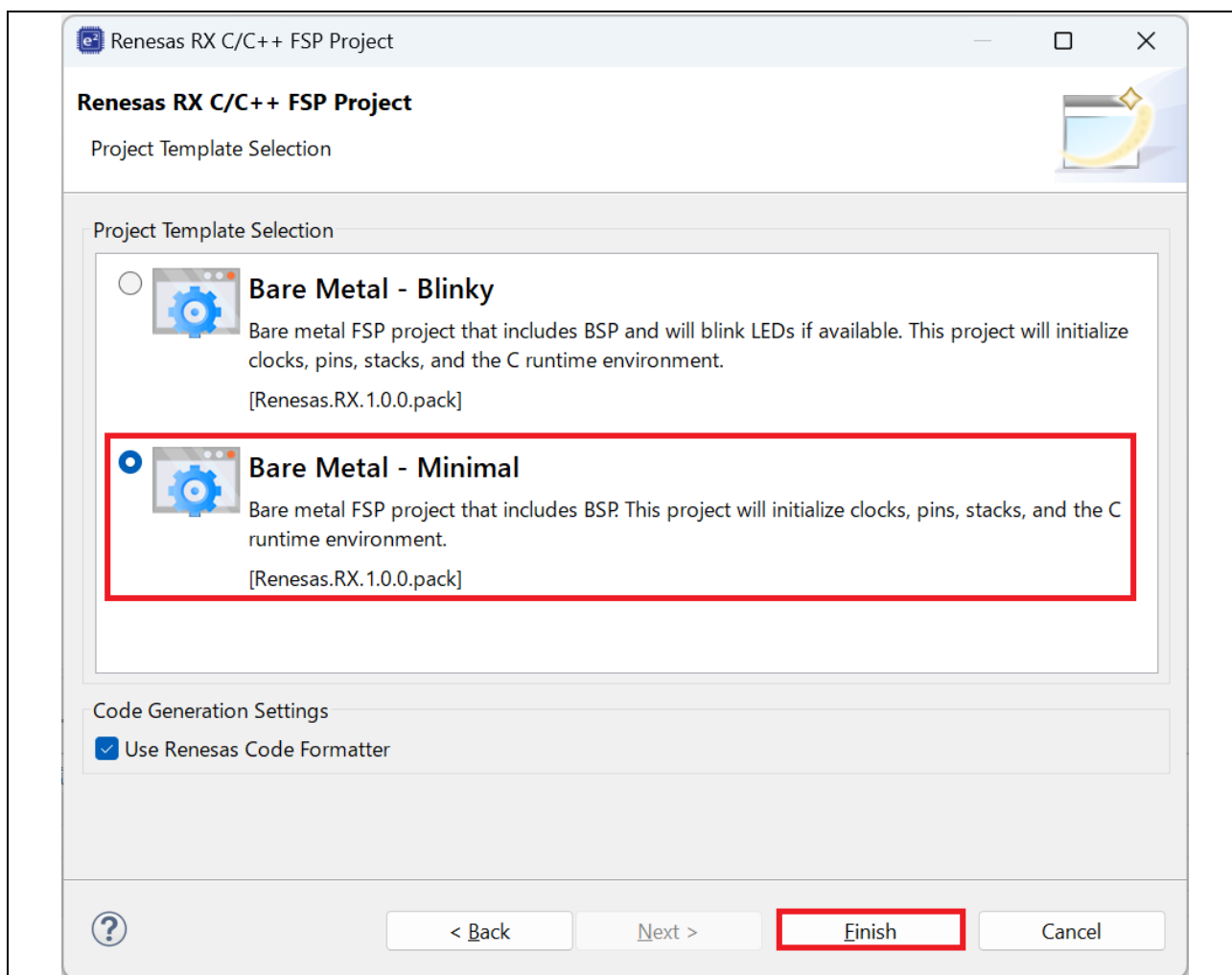


Figure 2-11 Selecting a Project Template

(9) Selecting the [FSP Configuration] perspective

Select whether to open the [FSP Configuration] perspective. Click on the [Open Perspective] button.

In Eclipse IDEs, a perspective is a predefined layout of multiple views combined.

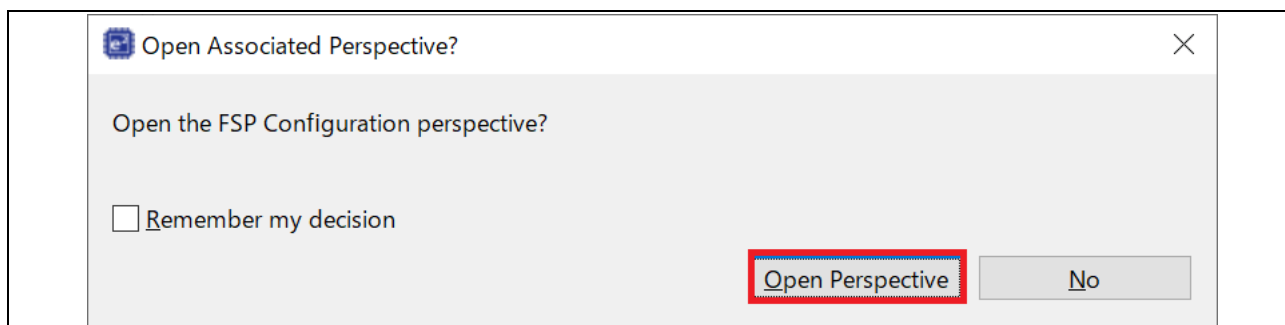


Figure 2-12 Selecting Whether to Open the [FSP Configuration] Perspective

(10) Completing creation of a new project

After successful creation of a new C/C++ project, it will be opened in the [FSP Configuration] perspective.

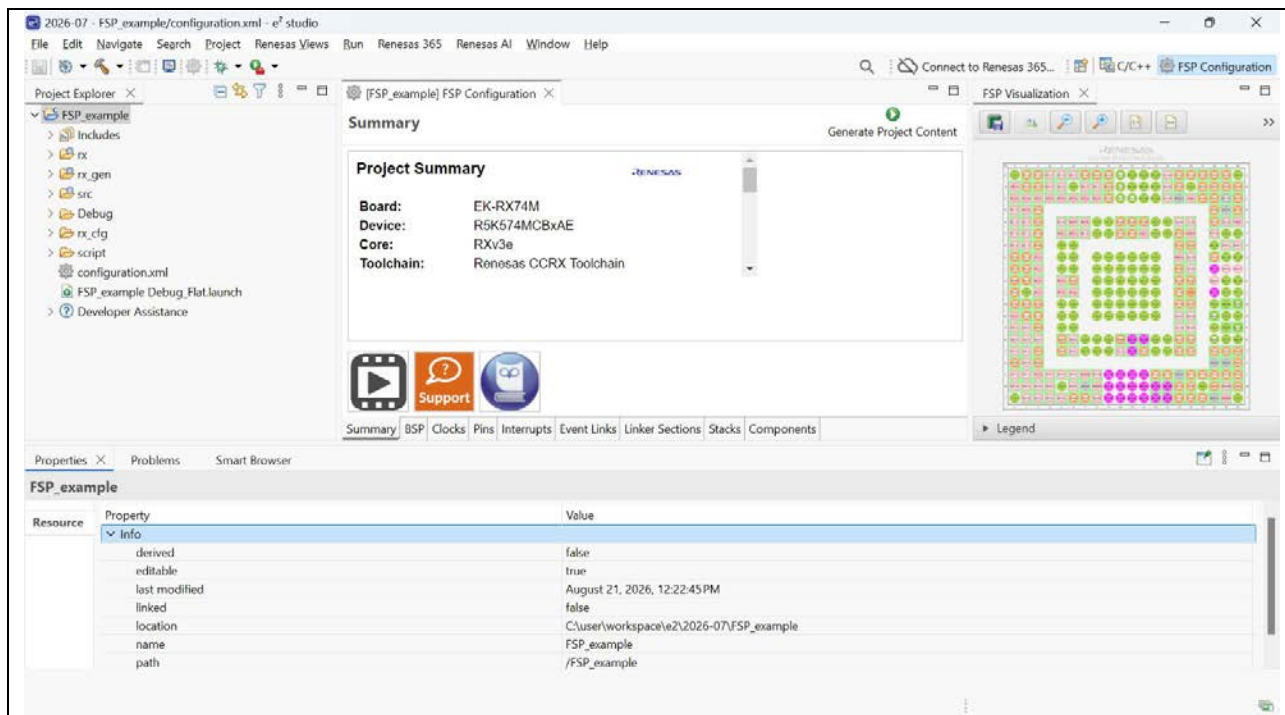


Figure 2-13 [FSP Configuration] Perspective

2.4 FSP Project Window

The following shows the layout of the [FSP Configuration] perspective.

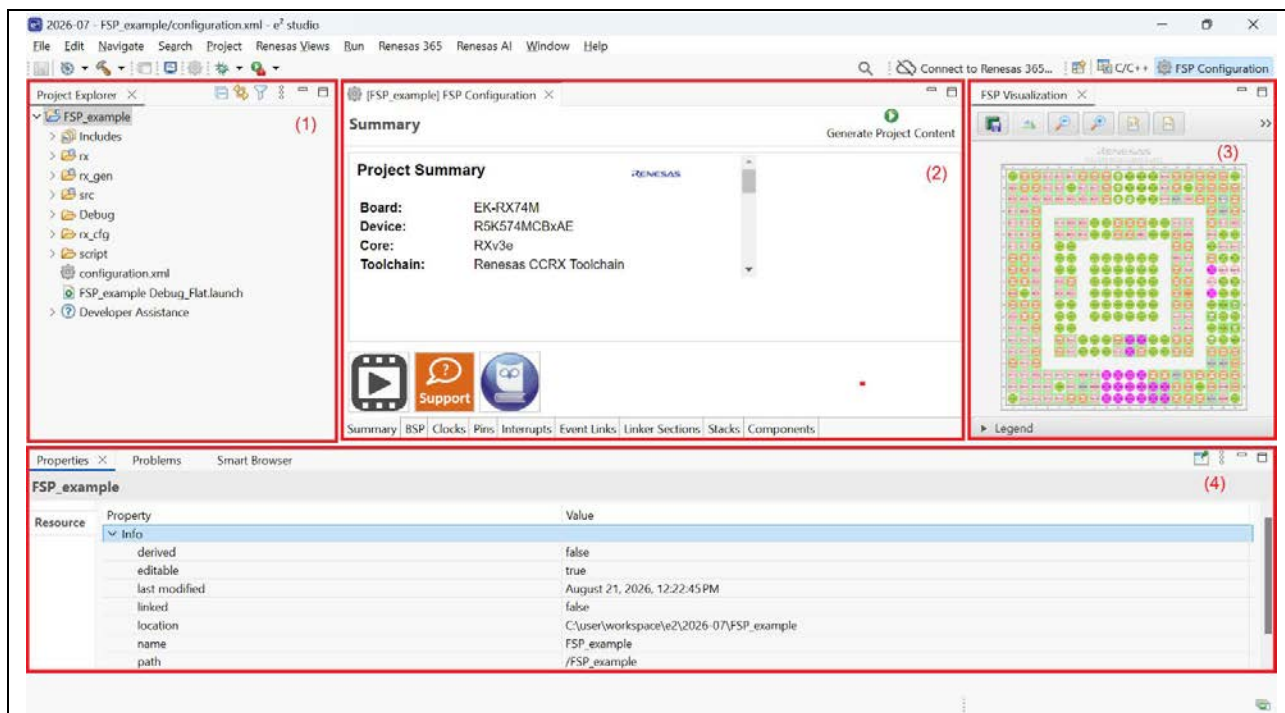


Figure 2-14 Layout of the [FSP Configuration] Perspective

- (1) [Project Explorer]
- (2) [RX Project Configuration Editor] view
- (3) [Visualization] view
- (4) [Properties] view

2.4.1 Project Explorer

The structure of the folders and files in the project is shown in a tree form.

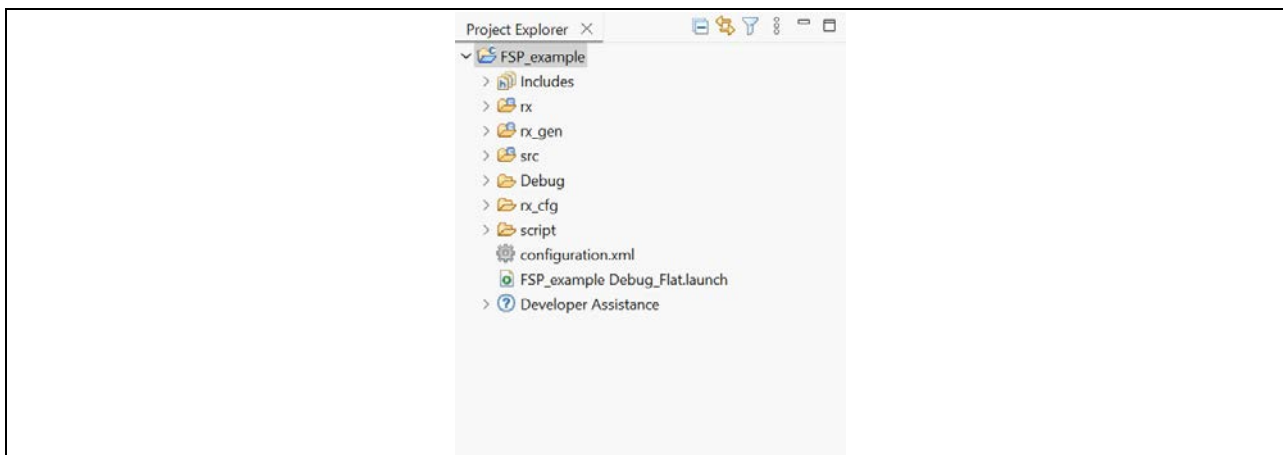


Figure 2-15 Project Explorer

If [Project Explorer] is not open, select [Window] → [Show View] → [Project Explorer] from the e² studio menu.

2.4.2 RX Project Configuration Editor View

This view consists of nine pages: [Summary], [Board Support Package Configuration] (hereafter called [BSP]), [Clocks Configuration], [Pin Configuration], [Interrupts Configuration], [Event Links Configuration], [Linker Section Mapping Configuration], [Stacks Configuration], and [Components Configuration]. Click on a tab to select the corresponding page, and the display will change to show the contents of the selected tabbed page.

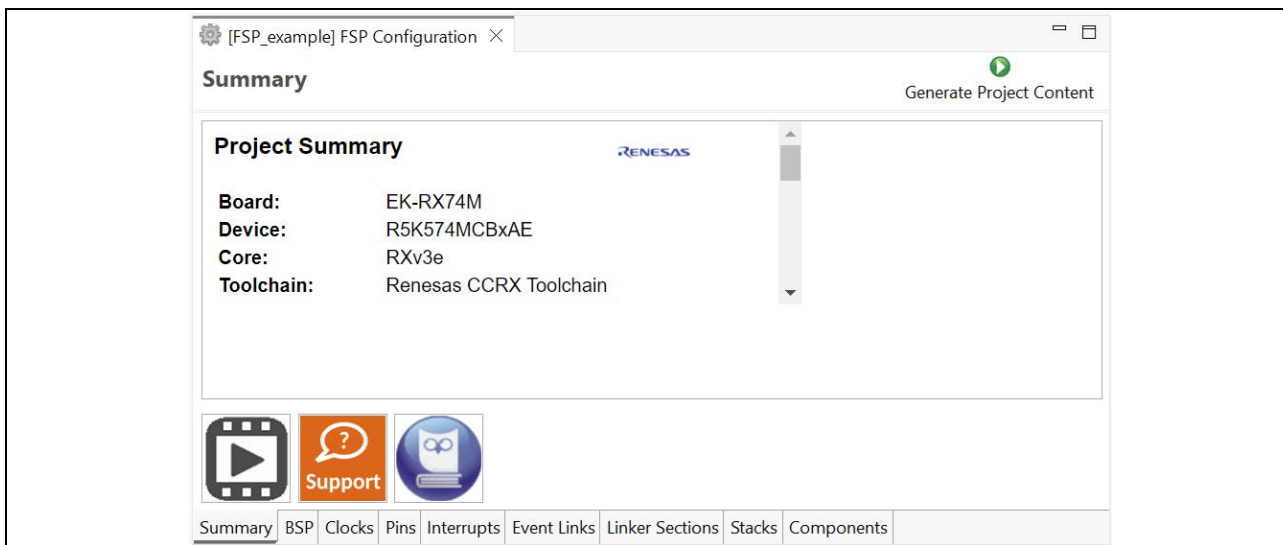


Figure 2-16 [RX Project Configuration Editor] View

If this view is not open, right-click on the name of the configuration file (configuration.xml) in the [Project Explorer] and select [Open] from the context menu.

2.4.3 Visualization View

This view shows the states of pins on a diagram of the MCU or MPU package. The settings of pins can be modified here.

The types of information shown in the initial state are set to [Pin Function] and [Module Name], but the pin information to be displayed can be switched by selecting from the list through the use of the [>>] button in the top-right region of the view.

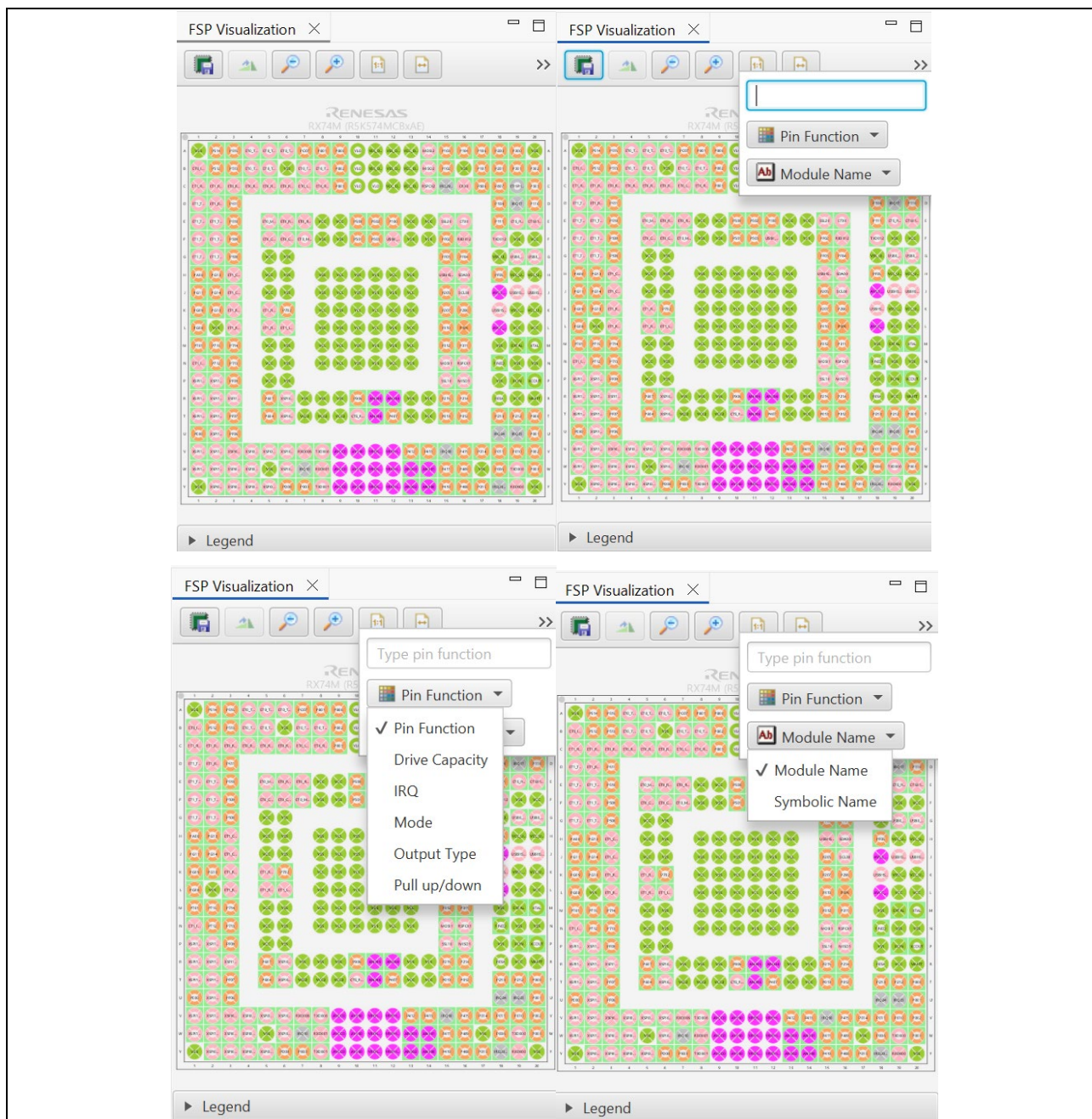


Figure 2-17 [Visualization] View

If this view is not open, select [Renesas Views] → [C/C++] → [FSP Visualization] from the e² studio menu.

2.4.4 Properties View

The following shows the [Properties] view. When a module is selected on the [BSP] or [Stacks Configuration] page and the module has property information, the details of the properties will be displayed. The settings can be modified from the [Properties] view.



Figure 2-18 [Properties] View (Initial State)



Figure 2-19 [Properties] View (When the [BSP] Page is Selected)

If this view is not open, select [Window] → [Show View] → [Properties] from the e² studio menu.

2.5 Locations of Settings for an FSP Project

The [RX Project Configuration Editor] view and [Properties] view are used to make most settings for an FSP project.

2.5.1 Basic Configuration

The [BSP] page can be used to modify the FSP version, board, or device even after the project has been created.

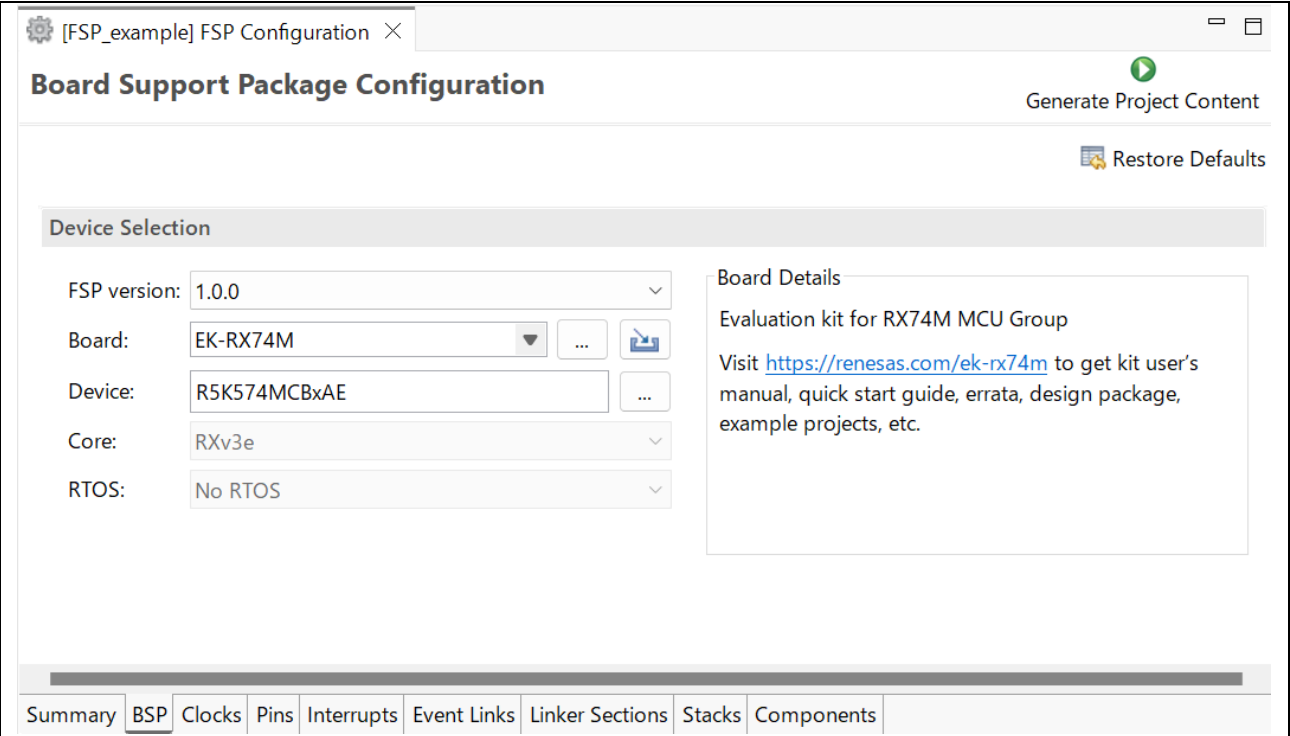


Figure 2-20 [BSP] Page

When the [BSP] page is selected, the [Properties] view shows the device-specific configuration items, such as "Security" or "Clocks", and also shows the basic configuration items for [RX Common]. Any settings that are required must be made.

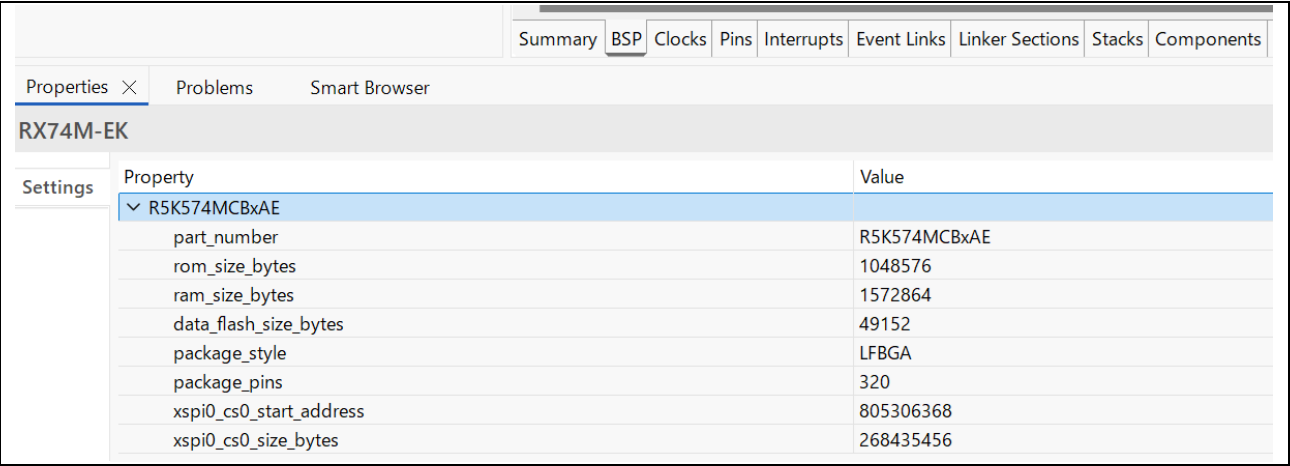


Figure 2-21 [Properties] View for the BSP

The following lists the configuration items under [RX Common].

Table 2.1 Basic Configuration Items

No.	Label for the Basic Configuration Item	Overview of the Configuration Item	Initial Value
1	Main stack size (bytes)	Size of the stack for use by the system	0x400
2	Heap size (bytes)	Size of the heap for use by the system	0
3	Protect zone root stack size (bytes)	Size of the stack for use by the root world of the protect zone.	0x800
4	MCU Vcc (mV)	Voltage supplied to the MCU	3300
5	Parameter checking	Enabling or disabling checking of the API parameters	"Disabled"
6	Assert Failures	Operation of the FSP_ASSERT() macro	"Return FSP_ERR_ASSERTION"
7	Clock Registers not Reset Values during Startup	Setting or non-setting of the clock-related registers to the reset values at the time of startup	"Disabled"
8	Main Oscillator Populated	Enabling or disabling oscillation of the main clock oscillator	"Populated" (oscillated)
9	PFS Protect	Enabling or disabling locking of the PFS registers	"Enabled" (locking is maintained)
10	C Runtime Initialization	Initializing or non-initializing of the C runtime	"Enabled" (initialized)
11	Early BSP Initialization	Enabling or disabling initialization to use BSP functions before C runtime initialization	"Disabled" (not initialized)
12	Main Oscillator Clock Source	Oscillation source for the main clock oscillator	"Crystal or Resonator"
13	Subclock Populated	Oscillation or stopping of the subclock oscillator	"Not Populated" (oscillation is stopped)
14	Subclock Drive (Drive capacitance availability varies by MCU)	Drive capacitance availability of the subclock oscillator	"Standard/Normal mode"
15	Subclock Stabilization Time (ms)	Oscillation stabilization time for the subclock oscillator	1000

For example, to increase the size of the stack for use by the system by 256 bytes, change "Main stack size (bytes)" to 0x500.

Note that the basic configuration items depend on the selected device or FSP version.

2.5.2 Clock Configuration

The [Clocks Configuration] page has the items for clock-related configuration, and they require settings to suit your system environment.

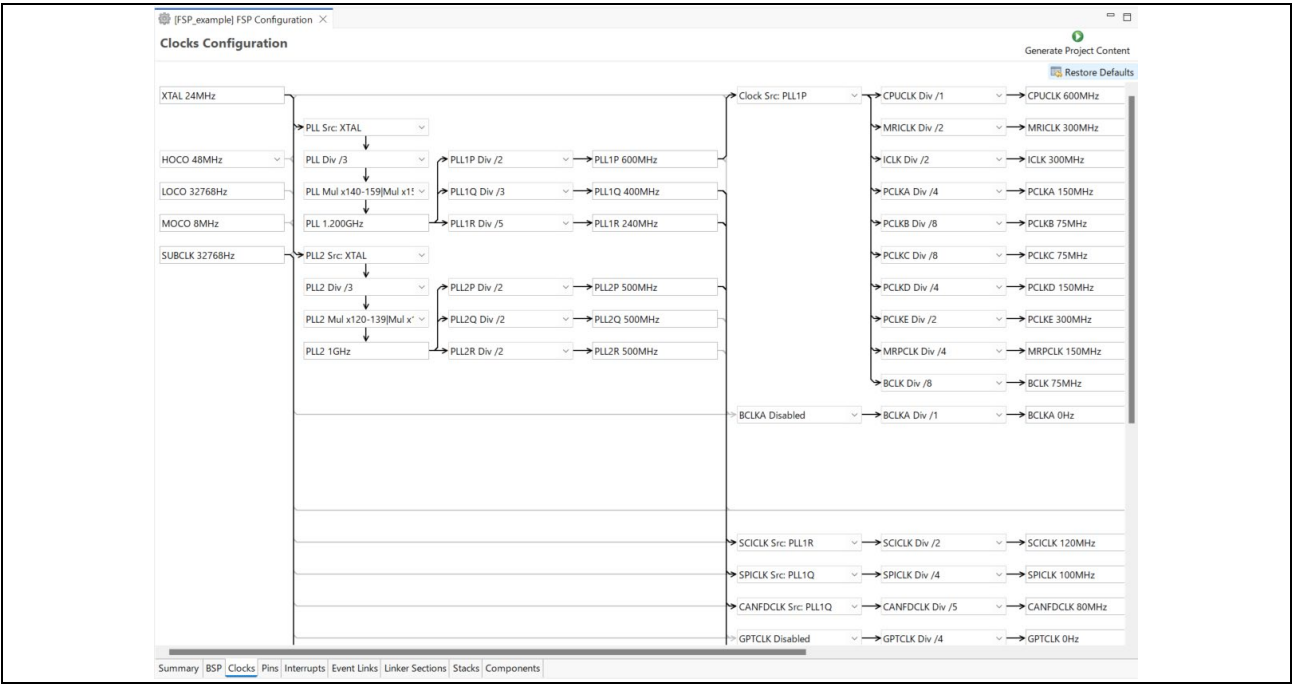


Figure 2-22 Example of the [Clocks Configuration] Page

For example, to change the division ratio for the system clock (ICLK) to 4, change the setting from "ICLK Div /2" to "ICLK Div /4".

Note that the clock configuration items depend on the selected device or FSP version.

2.5.3 Pin Configuration

The [Pin Configuration] page has the items for pin function assignment and any required settings must be made. Pin assignment can also be set in the [Visualization] view.

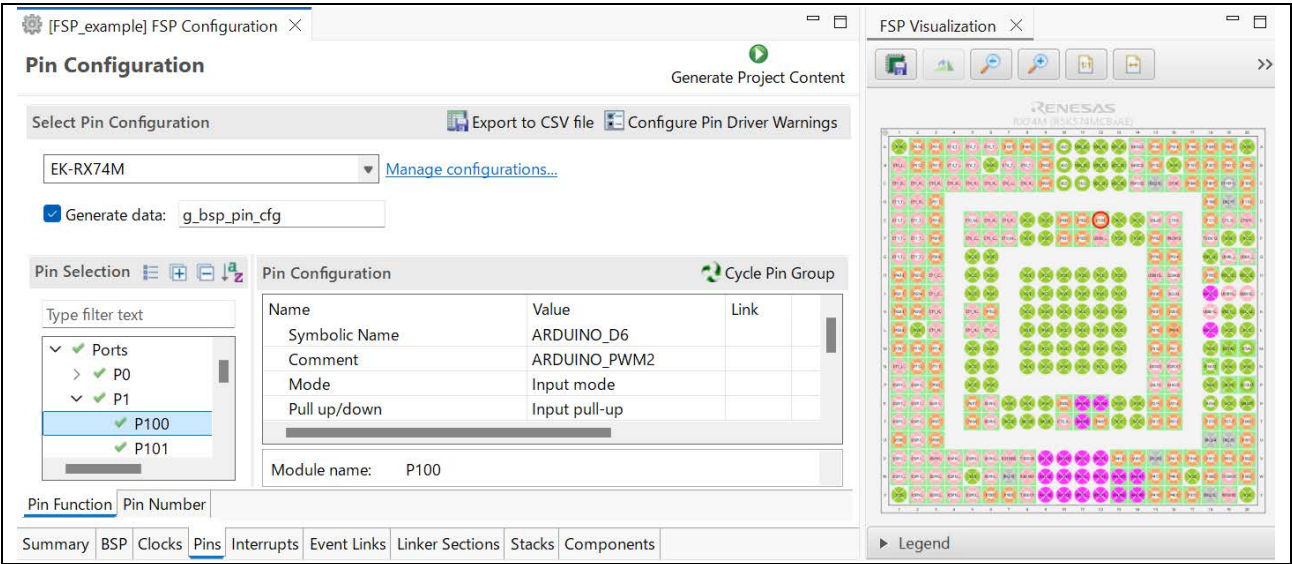


Figure 2-23 [Pin Configuration] Page and [Visualization] View

For example, to change P300 to an input port pin, change "Mode" to "Input mode".

Note that the port configuration items depend on the selected device or FSP version, and the initial values depend on the assignment of the pin functions.

2.5.4 Interrupt Configuration

The [Interrupts Configuration] page can be used to configure interrupts. This page requires setting if interrupts are to be used. If the use of interrupts is specified when a component is added in the component configuration step to be described later, the configuration of the interrupts to be used by the added module is automatically reflected on this page.

[FSP_example] FSP Configuration ×

Interrupts Configuration Generate Project Content

User Events New User Event > Remove

Event	ISR

Allocations

Interrupt	Event	ISR	Priority
28	ID:28 CMT0 CMI (Compare match)	cmt_cm_int_isr	Priority 31 (high...)

Summary BSP Clocks Pins **Interrupts** Event Links Linker Sections Stacks Components

Figure 2-24 [Interrupts Configuration] Page

2.5.5 Event Link Configuration

The [Event Links Configuration] page can be used to configure event links. This page requires setting if the event link function is to be used.

[FSP_example] FSP Configuration ×

Event Links Configuration Generate Project Content

User Events Produced New User Event > Remove

Event

User Events Consumed New User Event > Remove

Peripheral Function	Event

Allocations

Peripheral Function	Event
CMT1	No allocation
CMTW0	No allocation
TMR0	No allocation
TMR1	No allocation

Summary BSP Clocks Pins **Event Links** Linker Sections Stacks Components

Figure 2-25 [Event Links Configuration] Page

2.5.6 Linker Section Mapping Configuration

The [Linker Section Mapping Configuration] page can be used to configure the linker sections.

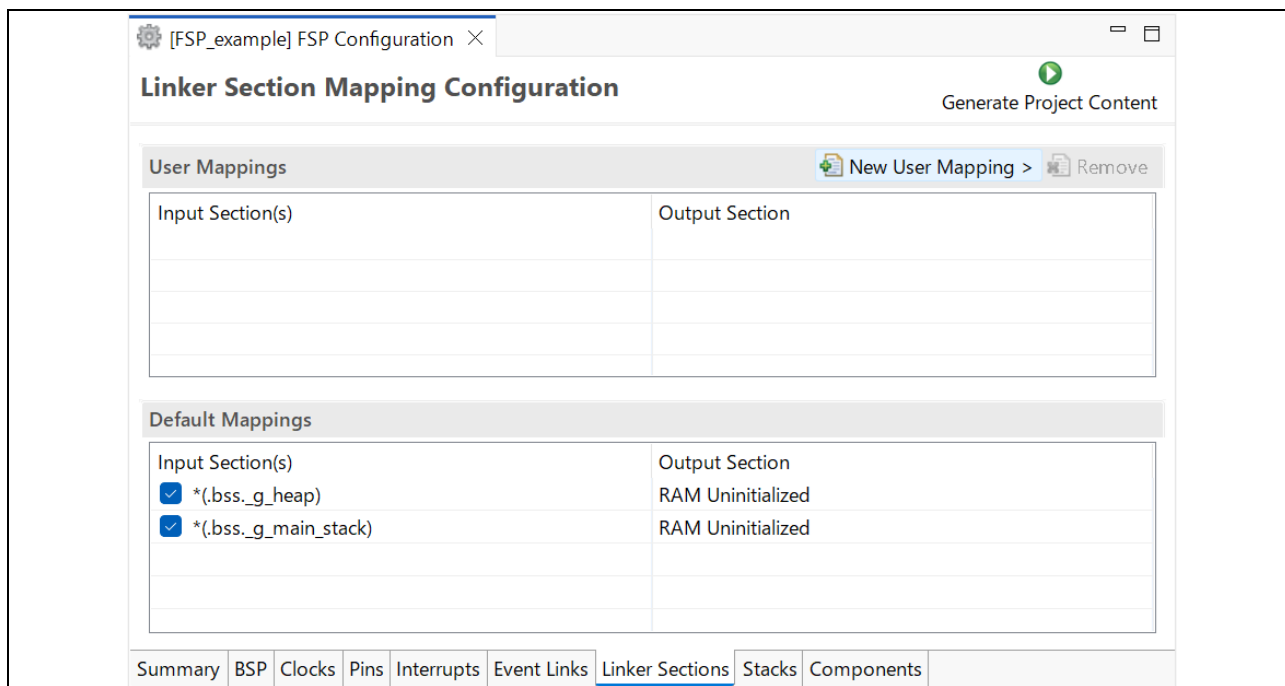


Figure 2-26 [Linker Section Mapping Configuration] Page

2.5.7 Component (Module) Configuration and Stack Configuration

The [Stacks Configuration] page can be used to add components (modules) and make detailed settings. To make detailed settings, use the [Properties] view while the target module is selected. FSP stacks are generated according to the specified information.

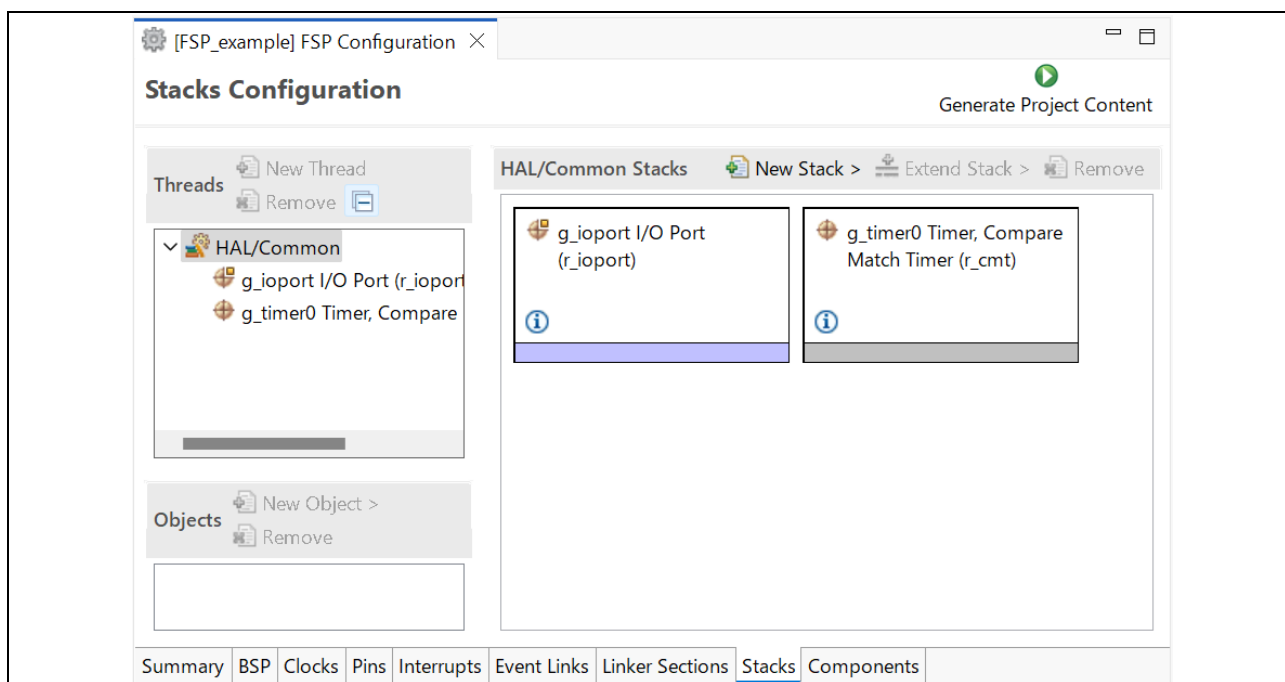
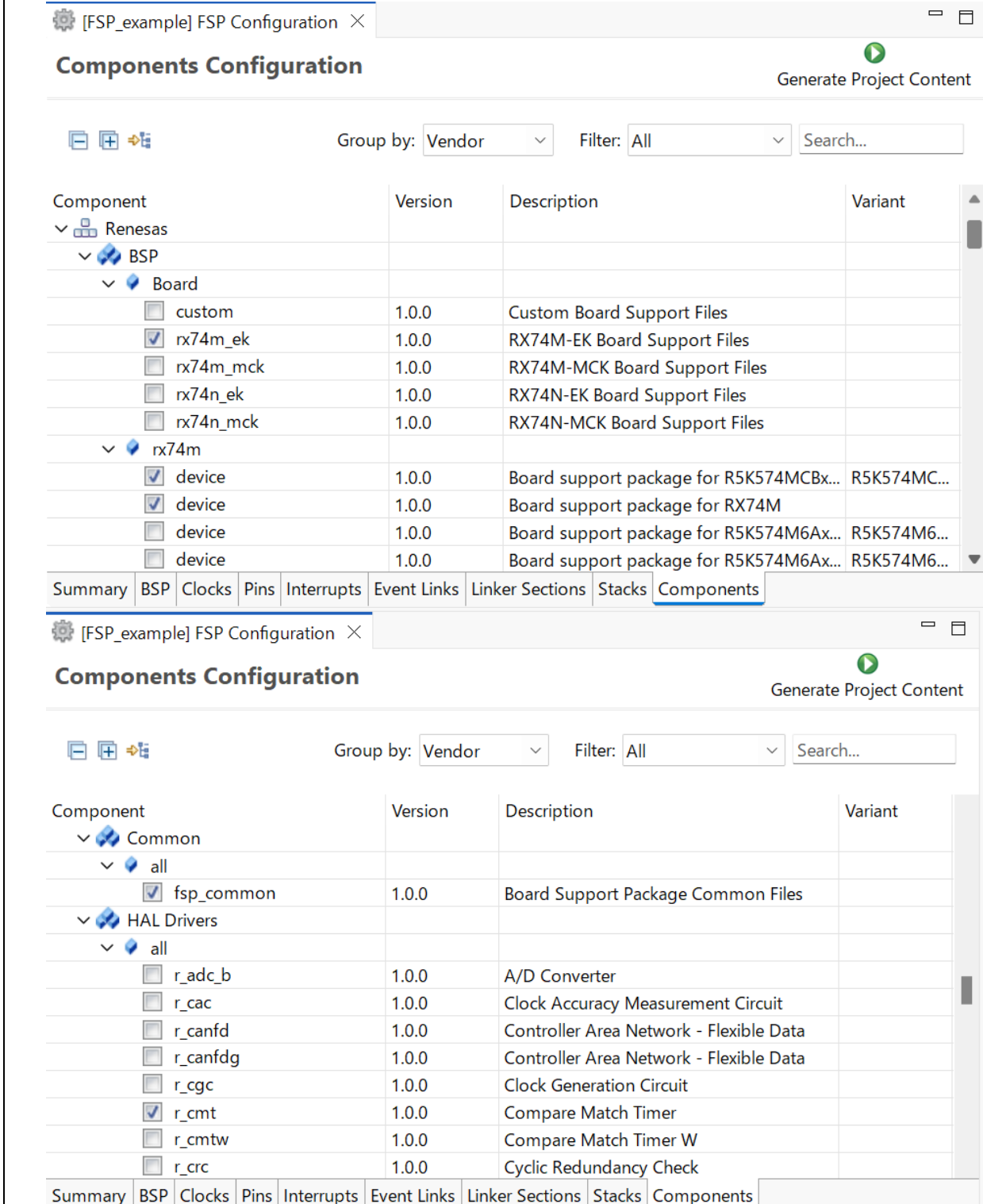


Figure 2-27 [Stacks Configuration] Page

Although modules can also be added or deleted on the [Components Configuration] page, the information of the modules added or deleted in this way is not reflected on the [Stacks Configuration] page. Therefore, adding or deleting a module on the [Components Configuration] page requires re-making the settings on the [Stacks Configuration] page. Addition or deleting of modules on the [Stacks Configuration] page is recommended.



Components Configuration

Generate Project Content

Group by: Vendor Filter: All Search...

Component	Version	Description	Variant
▼ Renesas			
▼ BSP			
▼ Board			
☐ custom	1.0.0	Custom Board Support Files	
☒ rx74m_ek	1.0.0	RX74M-EK Board Support Files	
☐ rx74m_mck	1.0.0	RX74M-MCK Board Support Files	
☐ rx74n_ek	1.0.0	RX74N-EK Board Support Files	
☐ rx74n_mck	1.0.0	RX74N-MCK Board Support Files	
▼ rx74m			
☒ device	1.0.0	Board support package for R5K574MCBx...	R5K574MC...
☒ device	1.0.0	Board support package for RX74M	
☐ device	1.0.0	Board support package for R5K574M6Ax...	R5K574M6...
☐ device	1.0.0	Board support package for R5K574M6Ax...	R5K574M6...

Summary | **BSP** | Clocks | Pins | Interrupts | Event Links | Linker Sections | Stacks | **Components**

Components Configuration

Generate Project Content

Group by: Vendor Filter: All Search...

Component	Version	Description	Variant
▼ Common			
▼ all			
☒ fsp_common	1.0.0	Board Support Package Common Files	
▼ HAL Drivers			
▼ all			
☐ r_adc_b	1.0.0	A/D Converter	
☐ r_cac	1.0.0	Clock Accuracy Measurement Circuit	
☐ r_canfd	1.0.0	Controller Area Network - Flexible Data	
☐ r_canfdg	1.0.0	Controller Area Network - Flexible Data	
☐ r_cgc	1.0.0	Clock Generation Circuit	
☒ r_cmt	1.0.0	Compare Match Timer	
☐ r_cmtw	1.0.0	Compare Match Timer W	
☐ r_crc	1.0.0	Cyclic Redundancy Check	

Summary | **BSP** | Clocks | Pins | Interrupts | Event Links | Linker Sections | Stacks | **Components**

Figure 2-28 [Components Configuration] Page

2.6 Structure of the FSP Project Files

The following shows the structure of the FSP project files.

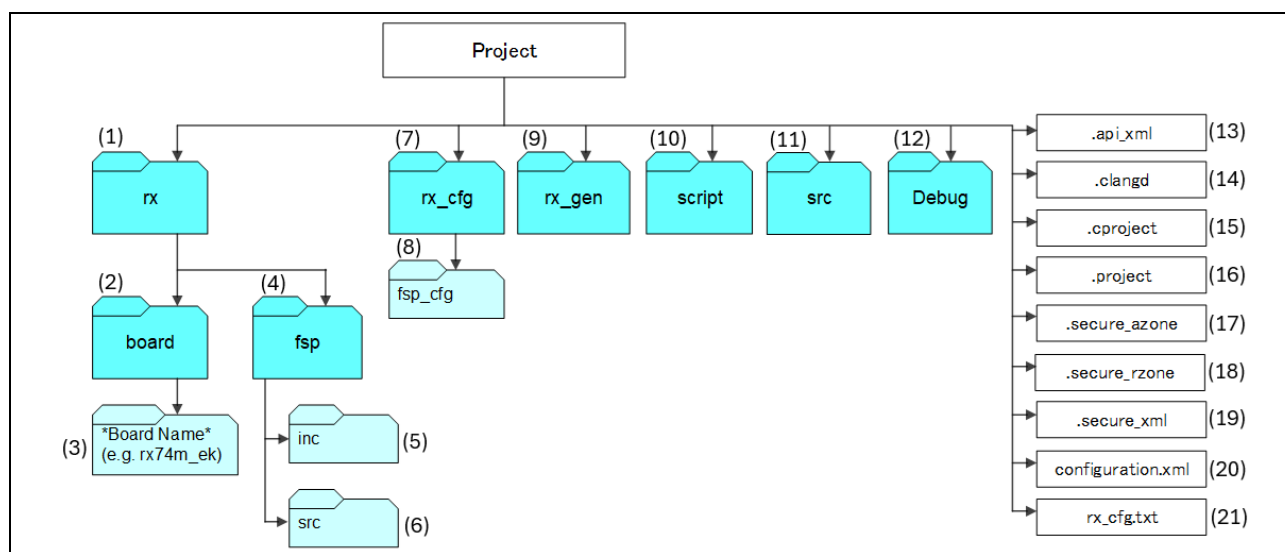


Figure 2-29 Structure of the FSP Project Files

Table 2.2 Descriptions of the Individual Folders of the Project

No.	Folder Name	Description
1	rx	Base folder
2	board	Base folder for output of the files related to the board that was selected when the project was created
3	*Board Name* (e.g., rx74m_ek)	Folder containing the definitions related to the board. The board selected at the time of creating the project determines the folder name that is output. Example: If the RX74M-EK board was selected, the folder name will be rx74m_ek.
4	fsp	Base folder for the FSP
5	inc	Folder for output of the header files used to include the definitions of the FSP modules that were selected in component configuration
6	src	Folder for output of the source files containing the definitions of the FSP modules that were selected in component configuration
7	rx_cfg	Base folder for output of the header files containing the definitions in which the settings by the RX project configuration editor are reflected
8	fsp_cfg	Folder for output of the header files containing the definitions in which the settings by the RX project configuration editor are reflected
9	rx_gen	Folder for output of the source code generated by the RX project configuration editor
10	script	Folder for output of the linker script files
11	src	Folder for output of the hal_entry.c file, which contains a function that is equivalent to the main function of the user application
12	Debug	Folder for output of the header files and linker script need to link. Build results, such as object files, are also output here after the build.

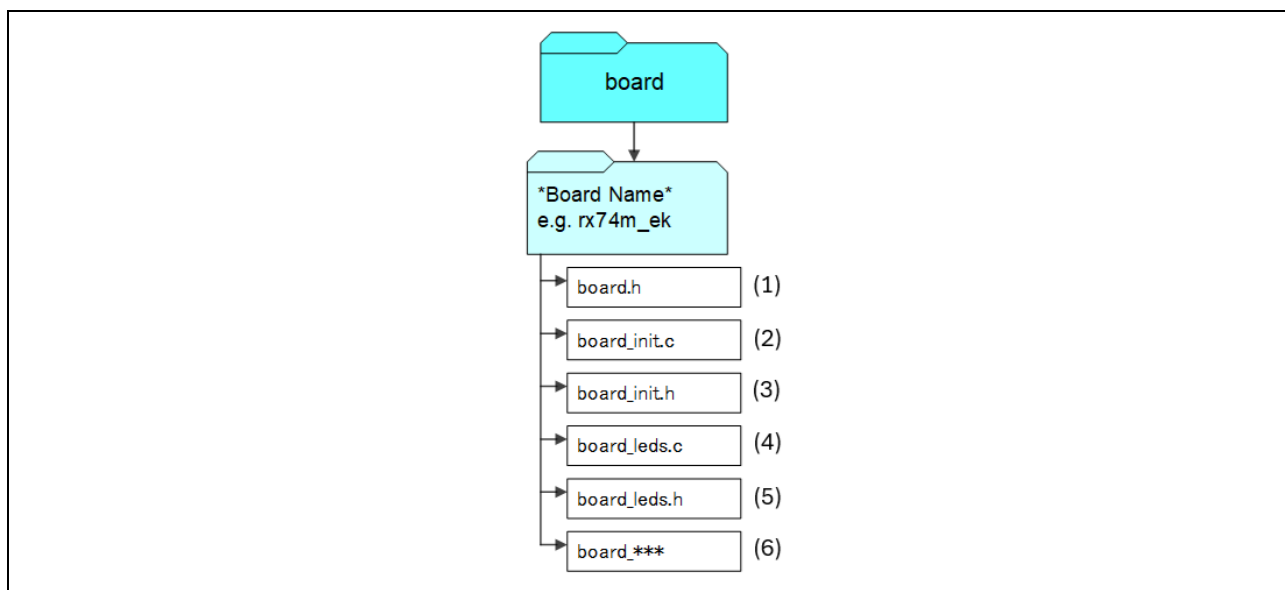


Figure 2-30 Structure of the Files in the board Folder

Table 2.3 Descriptions of the Individual Files in the board Folder

No.	File Name	Description
1	board.h	Base header file for including the header files related to the board
2	board_init.c	Source file defining the board-specific initialization processing
3	board_init.h	Header file containing the definitions related to the board-specific initialization processing
4	board_leds.c	Source file containing the definitions related to the LEDs on the board
5	board_leds.h	Header file containing the definitions related to the LEDs on the board
6	board_*** (e.g., board_ethernet_phy.h)	Source file or header file that is output when the selected board requires definitions other than those listed above

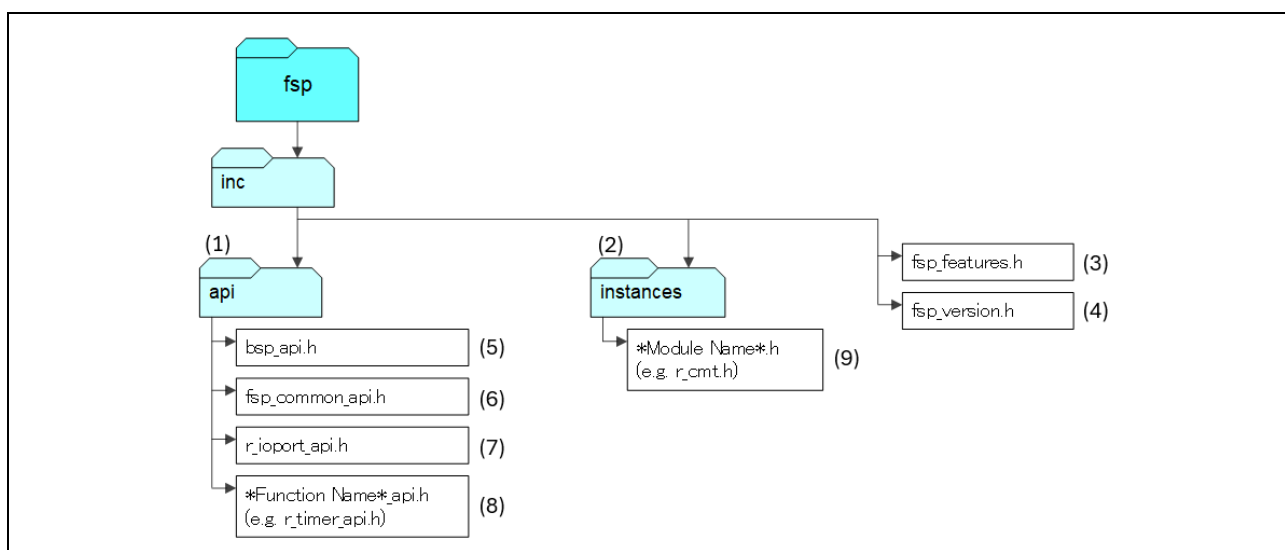


Figure 2-31 Structure of the Files in the fsp inc Folder

Table 2.4 Descriptions of the Individual Folders in the inc Folder

No.	Folder Name	Description
1	api	Folder for output of the header files containing the definitions of the structures used in the API for the modules and functions that were specified with the RX project configuration editor
2	instances	Folder for output of the header files containing the definitions of the API functions used for the modules that were specified with the RX project configuration editor

Table 2.5 Descriptions of the Individual Files in the inc Folder

No.	File Name	Description
3	fsp_features.h	Header file containing the definitions related to the features supported by the FSP
4	fsp_version.h	Header file containing the definition of the version of the FSP

Table 2.6 Descriptions of the Individual Files in the api Folder

No.	File Name	Description
5	bsp_api.h	Header file containing the definitions related to the API of the BSP
6	fsp_common_api.h	Header file containing the common definitions for the FSP
7	r_ioport_api.h	Header file containing the definitions related to the API of the IOPORT
8	*Function Name*_api.h (e.g., r_timer_api.h)	Header file containing the definitions related to the API for the function that was selected in component configuration. The selected function determines the name of the file that is output. Example: If a timer function was selected, the file name will be r_timer_api.h.

Table 2.7 Descriptions of the Individual Files in the Instances Folder

No.	File Name	Description
9	*Module Name*.h (e.g., r_cmt.h)	Header file containing the definitions related to the API for the module that was selected in component configuration. The selected module determines the name of the file that is output. Example: If the CMT module (r_cmt) was selected, the file name will be r_cmt.h.

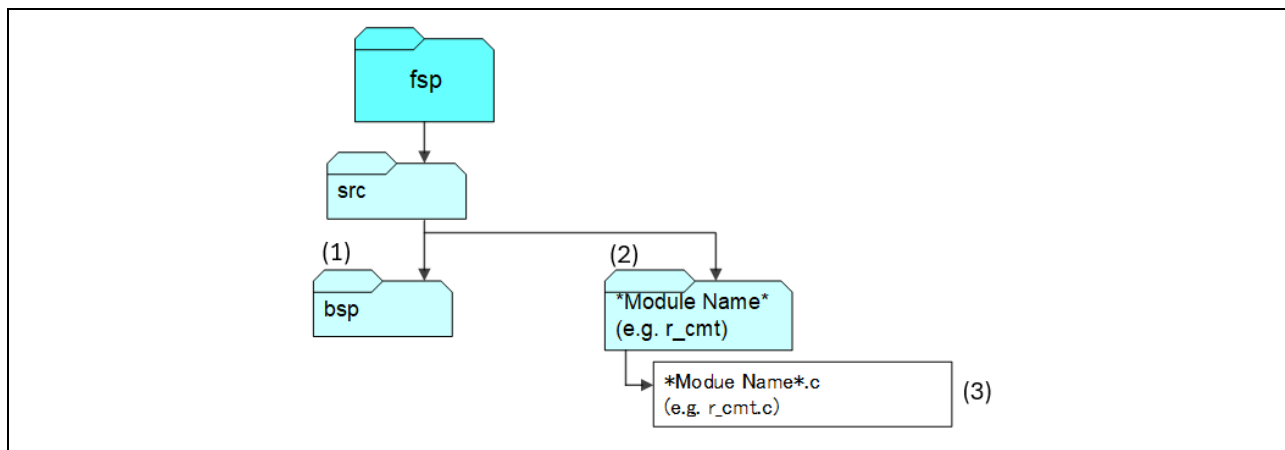


Figure 2-32 Structure of the Files in the fsp src Folder

Table 2.8 Descriptions of the Individual Folders in the src Folder

No.	Folder Name	Description
1	bsp	Base folder for output of the BSP files
2	*Module Name* (e.g., r_cmt)	Folder for output of the file for the module specified with the RX project configuration editor. The selected module determines the name of the folder that is output. Example: If the CMT module (r_cmt) was selected, the folder name will be r_cmt.

Table 2.9 Descriptions of the Individual Files in the *Module Name* Folder

No.	File Name	Description
3	*Module Name*.c (e.g., r_cmt.c)	Source file containing the definitions related to the API for the module selected in component configuration. The selected module determines the name of the file that is output. Example: If the CMT module (r_cmt) was selected, the file name will be r_cmt.c.

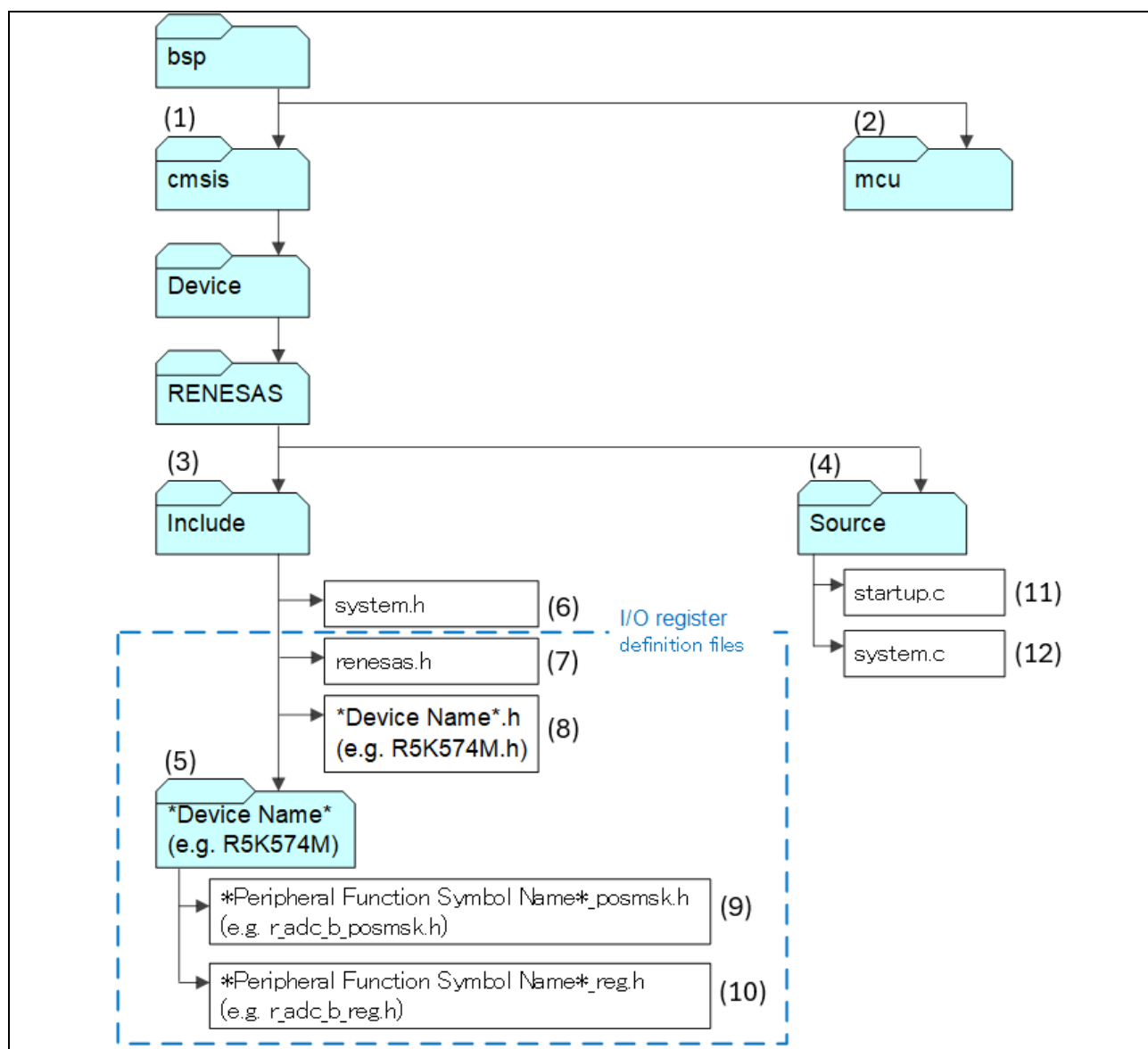


Figure 2-33 Structure of the Files in the bsp Folder

Table 2.10 Descriptions of the Individual Folders in the bsp Folder

No.	Folder Name	Description
1	cmsis	Base folder for output of the I/O register definitions and the source files for startup processing
2	mcu	Folder for output of the files containing the BSP definitions
3	Include	Base folder for output of the header files containing the I/O register definitions
4	Source	Folder for output of the source files related to the startup processing
5	*Device Name* (e.g., R5K574M)	Folder for output of the header files containing the definitions of the I/O registers in the individual peripheral functions of the device that was selected when the project was created. The selected device determines the name of the folder that is output. Example: If the RX74M was selected, the folder name will be R5K574M.

Table 2.11 Descriptions of the Individual Files in the Include Folder

No.	File Name	Description
6	system.h	Header file containing the definitions and processing related to initialization of the system
7	renesas.h	Header file that serves as the starting point for a group of I/O register definition files in the FSP. This file has an include statement for *Device Name*.h.
8	*Device Name*.h (e.g., R5K574M.h)	Base header file for the group of I/O register definition files for the corresponding device. This file has include statements for the header files for all peripheral functions, which are stored in the *Device Name* folder. The name of the selected device determines the name of the file that is output. Example: If the RX74M was selected, the file name will be R5K574M.h.
9	*Peripheral Function Symbol Name*_posmsk.h (e.g., r_adc_b_posmsk.h)	Header file containing the definitions of the individual bit-field locations and masks for the registers in each peripheral function. The symbol name of the peripheral function determines the name of the file that is output. Example: The file name for the A/D converter will be r_adc_b_posmsk.h.
10	*Peripheral Function Symbol Name*_reg.h (e.g., r_adc_b_reg.h)	Header file containing the definitions of the structures representing the registers in each peripheral function. The symbol name of the peripheral function determines the name of the file that is output. Example: The file name for the A/D converter will be r_adc_b_reg.h.

Table 2.12 Descriptions of the Individual Files in the Source Folder

No.	File Name	Description
11	startup.c	Source file containing the definitions of the reset and exception vectors and the startup processing
12	system.c	Source file containing the definitions and processing related to initialization of the system

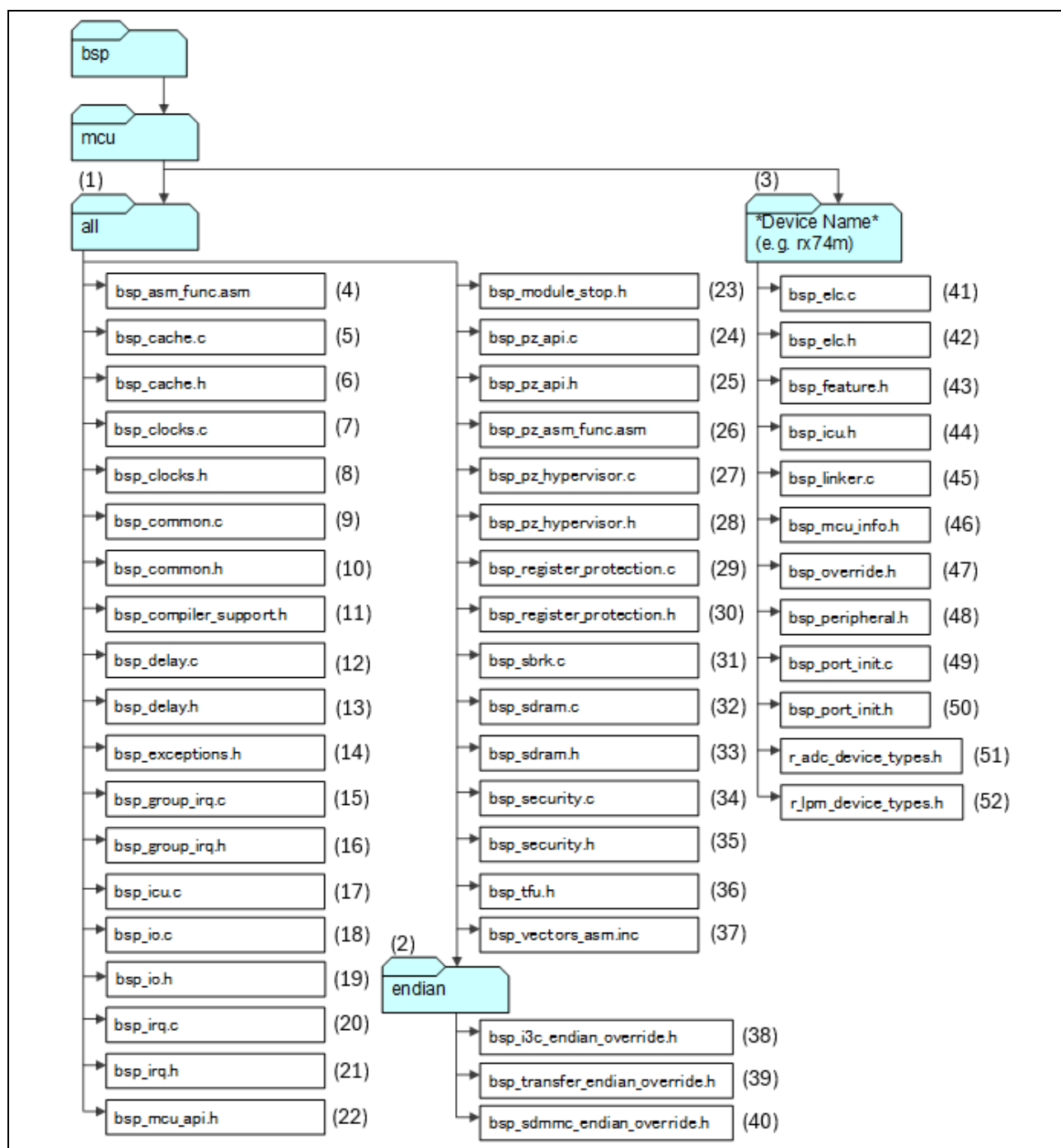


Figure 2-34 Structure of the Files in the mcu Folder

Table 2.13 Descriptions of the Individual Folders in the mcu Folder

No.	Folder Name	Description
1	all	Folder for output of the BSP definitions used in common for all devices
2	endian	Folder for output of the files containing the definitions related to endianness
3	*Device Name* (e.g., rx74m)	Folder for output of the files containing the device-specific BSP definitions. The selected device determines the name of the folder that is output. Example: If the RX74M was selected, the folder name will be rx74m.

Table 2.14 Descriptions of the Individual Files in the all Folder (1)

No.	File Name	Description
4	bsp_asm_func.asm	Source file containing the definitions of the assembler functions
5	bsp_cache.c	Source file containing the definitions related to cache control
6	bsp_cache.h	Header file containing the definitions related to cache control
7	bsp_clocks.c	Source file containing the definitions related to the settings of the clock generation circuit
8	bsp_clocks.h	Header file containing the definitions related to the settings of the clock generation circuit
9	bsp_common.c	Source file containing the common BSP definitions
10	bsp_common.h	Header file containing the common BSP definitions
11	bsp_compiler_support.h	Header file containing the definitions related to the compiler
12	bsp_delay.c	Source file containing the definitions related to the delay processing of software
13	bsp_delay.h	Header file containing the definitions related to the delay processing of software
14	bsp_exceptions.h	Header file containing the definitions related to exception processing
15	bsp_group_irq.c	Source file containing the definitions related to the group interrupts
16	bsp_group_irq.h	Header file containing the definitions related to the group interrupts
17	bsp_icu.c	Source file containing the definitions related to the interrupt configuration
18	bsp_io.c	Source file containing the definitions related to the I/O ports
19	bsp_io.h	Header file containing the definitions related to the I/O ports
20	bsp_irq.c	Source file containing the definitions related to interrupt processing
21	bsp_irq.h	Header file containing the definitions related to interrupt processing
22	bsp_mcu_api.h	Header file containing the definitions related to the API functions for the BSP used in common for all devices
23	bsp_module_stop.h	Header file containing the definitions related to the module stop control registers for the low power consumption function
24	bsp_pz_api.c	Source file containing the definitions related to world movement for the Protect Zone
25	bsp_pz_api.h	Header file containing the definitions related to world movement for the Protect Zone
26	bsp_pz_asm_func.asm	Source file containing the definitions of the assembler functions for the Protect Zone control
27	bsp_pz_hypervisor.c	Source file containing the definitions related to the Protect Zone control
28	bsp_pz_hypervisor.h	Header file containing the definitions related to the Protect Zone control
29	bsp_register_protection.c	Source file containing the definitions related to the register write protection function

Table 2.15 Descriptions of the Individual Files in the all Folder (2)

No.	File Name	Description
30	bsp_register_protection.h	Header file containing the definitions related to the register write protection function
31	bsp_sbrk.c	Source file containing the definition of the sbrk function, which is a low-level interface function
32	bsp_sdram.c	Source file containing the definitions related to the SDRAM
33	bsp_sdram.h	Header file containing the definitions related to the SDRAM
34	bsp_security.c	Source file containing the definitions related to initialization of the security functions
35	bsp_security.h	Header file containing the definitions related to initialization of the security functions
36	bsp_tfu.h	Header file containing the definitions related to the arithmetic unit for trigonometric functions
37	bsp_vectors_asm.inc	Include file containing the definitions related to the vectors

Table 2.16 Descriptions of the Individual Files in the endian Folder

No.	File Name	Description
38	bsp_i3c_endian_override.h	Header file containing the definitions related to the endianness of the I3C
39	bsp_sdmmc_endian_override.h	Header file containing the definitions related to the endianness of the SDMMC
40	bsp_transfer_endian_override.h	Header file containing the definitions related to the endianness of the transfer

Table 2.17 Descriptions of the Individual Files in the *Device Name* Folder

No.	File Name	Description
41	bsp_elc.c	Source file containing the definitions related to the event link controller
42	bsp_elc.h	Header file containing the definitions related to the event link controller
43	bsp_feature.h	Header file containing the definitions related to the functions supported by the device
44	bsp_icu.h	Header file containing the definitions related to the interrupt configuration
45	bsp_linker.c	Source file containing the definitions related to the option settings
46	bsp_mcu_info.h	Header file containing the definitions related to the device information
47	bsp_override.h	Header file containing the definitions related to the modifications of clock configuration or interrupt configuration
48	bsp_peripheral.h	Header file containing the definitions related to the peripheral functions
49	bsp_port_init.c	Source file containing the initialization of the port direction registers (PDR)
50	bsp_port_init.h	Header file containing the initialization of the port direction registers (PDR)
51	r_adc_device_types.h	Header file containing the definitions related to the A/D converter
52	r_lpm_device_types.h	Header file containing the definitions related to the Low Power Consumption

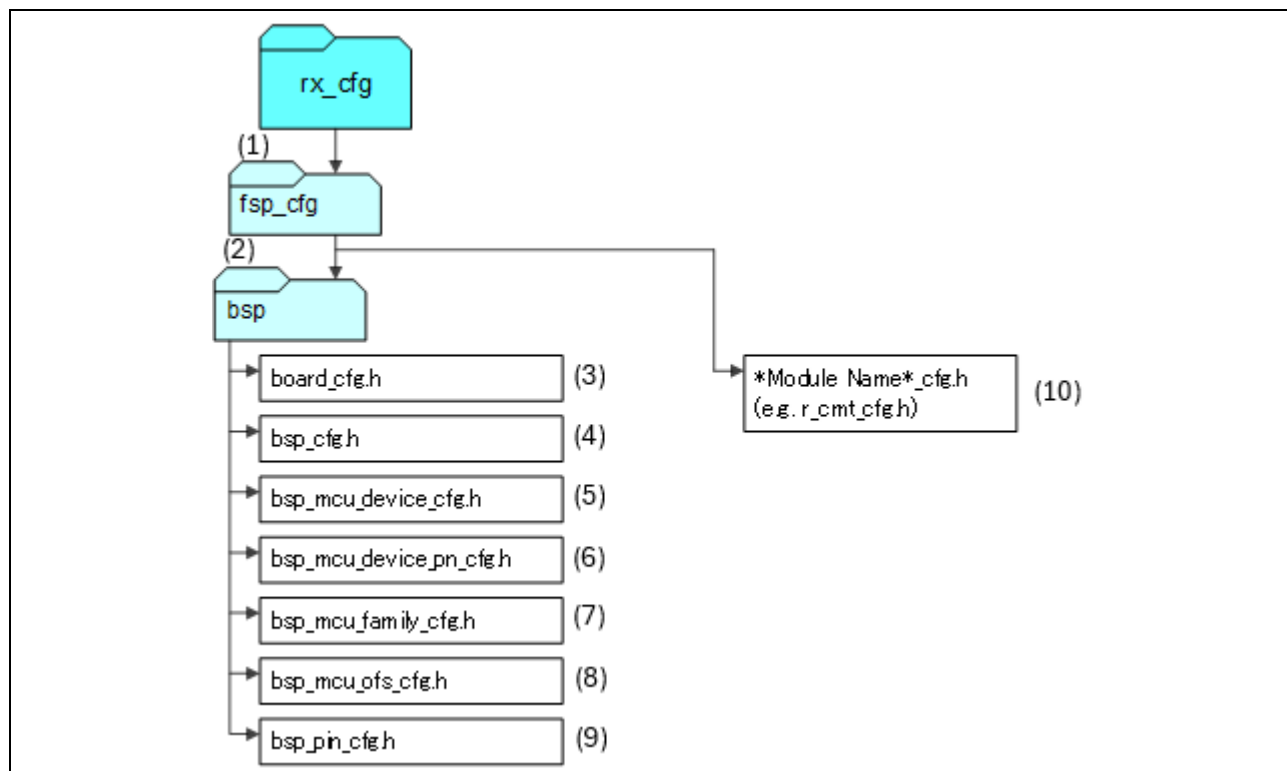
**Figure 2-35 Structure of the Files in the rx_cfg Folder**

Table 2.18 Descriptions of the Individual Folders in the rx_cfg Folder

No.	Folder Name	Description
1	fsp_cfg	Base folder for output of the files containing the definitions that reflect the settings of the modules specified with the RX project configuration editor
2	bsp	Folder for output of the files containing the definitions that reflect the settings of the BSP

Table 2.19 Descriptions of the Individual Files in the bsp Folder

No.	File Name	Description
3	board_cfg.h	Header file containing the include path to board.h
4	bsp_cfg.h	Header file containing the definitions that reflect the settings of the BSP
5	bsp_mcu_device_cfg.h	Header file containing the definitions that reflect the settings related to the particular MCU device
6	bsp_mcu_device_pn_cfg.h	Header file containing the definitions related to the package of the particular MCU device
7	bsp_mcu_family_cfg.h	Header file containing the definitions that reflect the settings related to the whole MCU family
8	bsp_mcu_ofs_cfg.h	Header file containing the definitions related to the option-setting memory of the particular MCU device
9	bsp_pin_cfg.h	Header file containing the definitions that reflect the settings related to the pin configuration

Table 2.20 Descriptions of the Individual Files in the fsp_cfg Folder

No.	File Name	Description
10	*Module Name*_cfg.h (e.g., r_cmt_cfg.h)	Header file containing the definitions related to the settings of the module selected in component configuration. The selected module determines the name of the file that is output. Example: If the CMT module (r_cmt) was selected, the file name will be r_cmt_cfg.h.

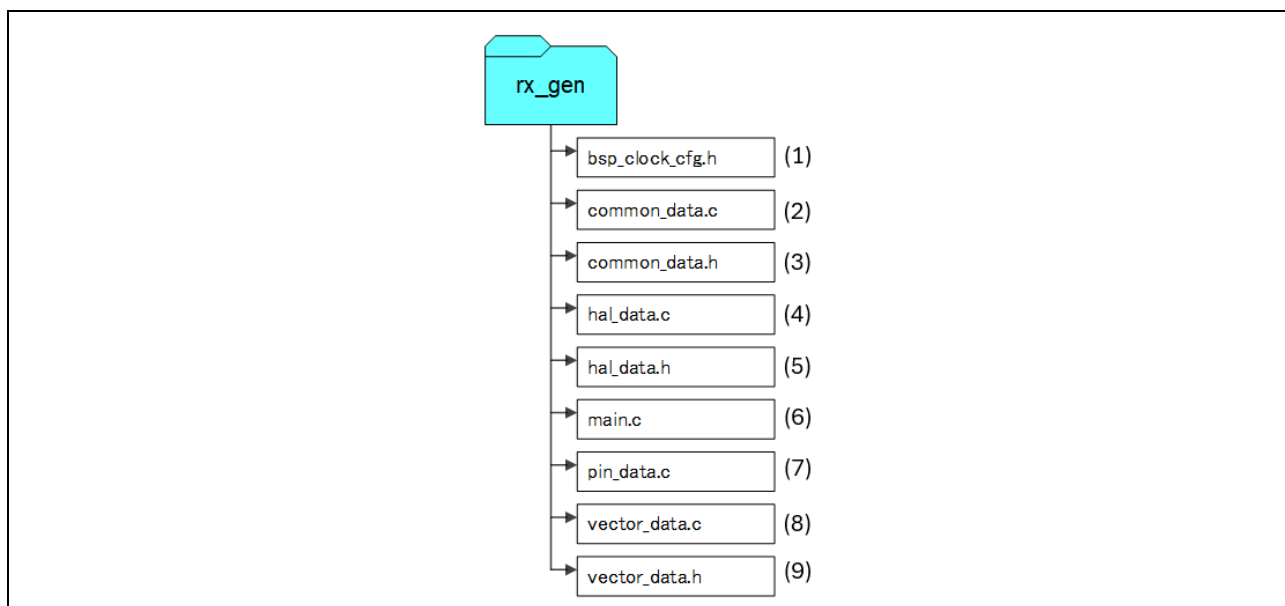
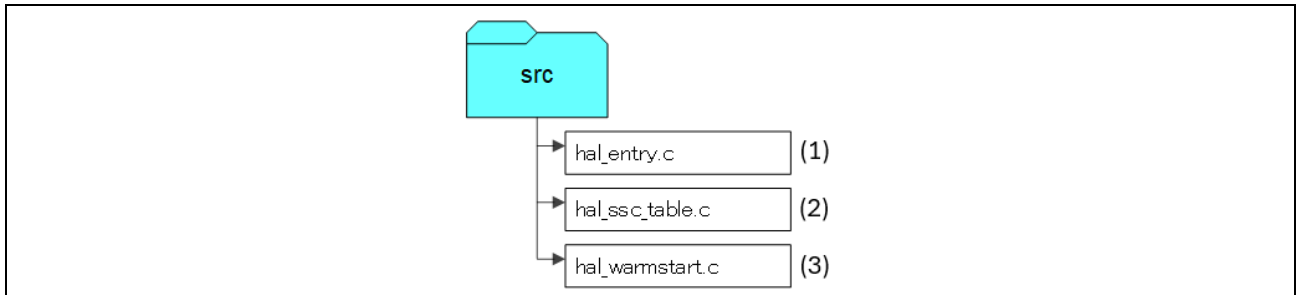


Figure 2-36 Structure of the Files in the rx_gen Folder

Table 2.21 Descriptions of the Individual Files in the rx_gen Folder

No.	File Name	Description
1	bsp_clock_cfg.h	Header file containing the definitions that reflect the settings in the clock configuration
2	common_data.c	Source file containing the common FSP definitions that reflect the settings by the RX project configuration editor
3	common_data.h	Header file containing the common FSP definitions that reflect the settings by the RX project configuration editor
4	hal_data.c	Source file containing the definitions that reflect the settings of the modules selected in component configuration
5	hal_data.h	Header file containing the definitions that reflect the settings of the modules selected in component configuration
6	main.c	Source file defining the main function
7	pin_data.c	Source file containing the definitions that reflect the settings in pin configuration
8	vector_data.c	Source file containing the definitions of the interrupt vector table and interrupt processing that reflect the settings in interrupt configuration
9	vector_data.h	Header file containing the definitions of the interrupt vector table and interrupt processing that reflect the settings in interrupt configuration

**Figure 2-37 Structure of the Files in the src Folder****Table 2.22 Descriptions of the Individual Files in the src Folder**

No.	File Name	Description
1	hal_entry.c	Source file containing the definition of a function that is equivalent to the main function of the FSP project
2	hal_ssc_table.c	Source file containing the definition related to the handler for secure world
3	hal_warmstart.c	Source file containing the definitions related to the warmstart function

2.7 How to Check the Information on the API of an FSP Module

The detailed information on the API of an FSP module can be obtained as follows.

On the [Stacks Configuration] page, click on the information icon of the module for which you require detailed API information. Alternatively, right-click on the module and select [Module Resources...] from the context menu.

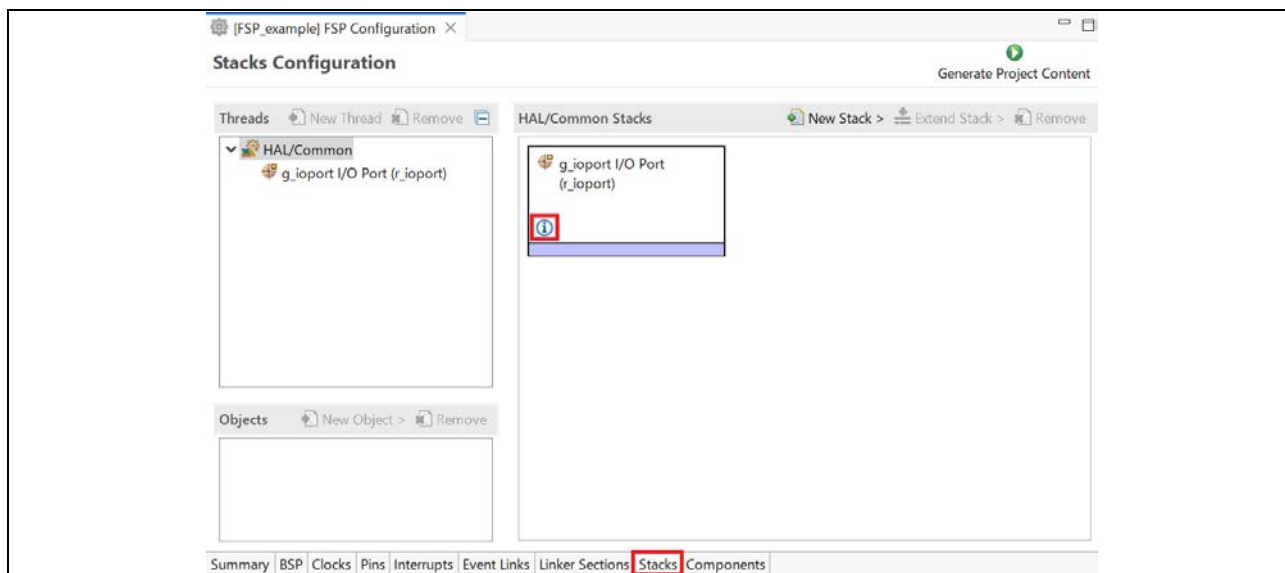


Figure 2-38 [Stacks Configuration] Page

The documentation page for the API of the selected module will be displayed.

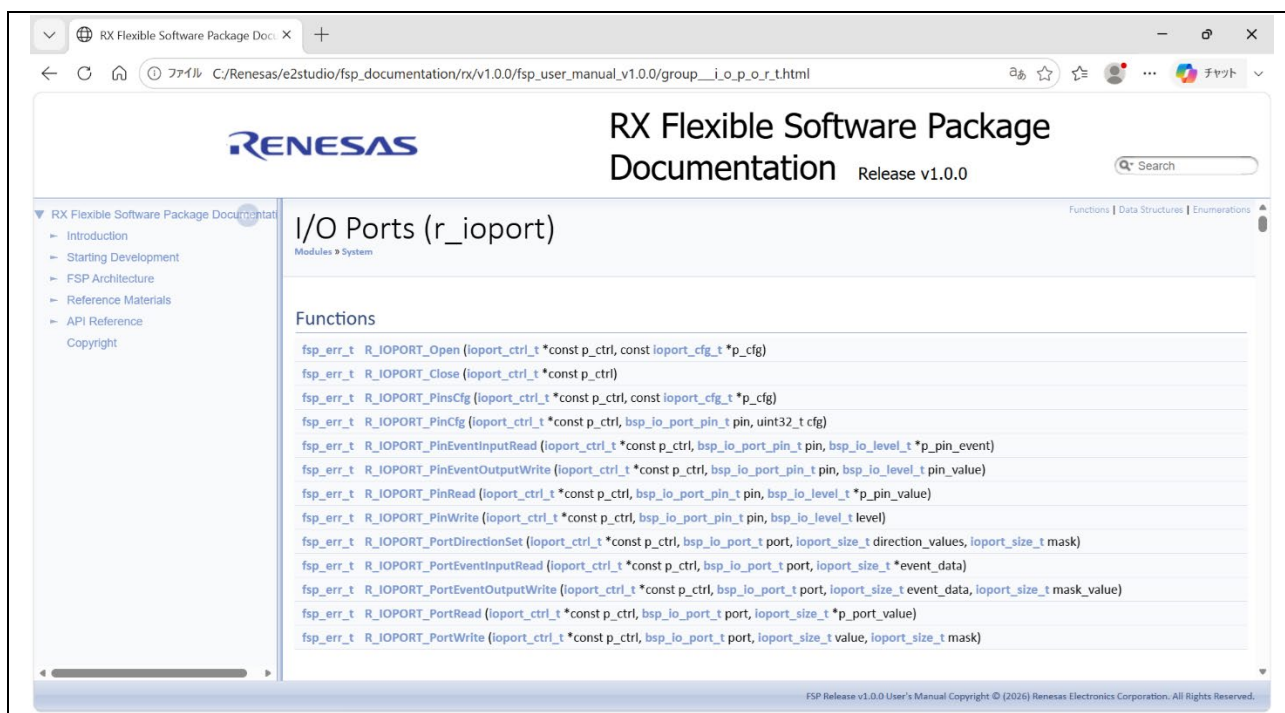


Figure 2-39 Help Information on a Module in the FSP Documentation

3. Migration to the FSP Development Environment

This chapter describes migration from an existing environment using the Smart Configurator of the e² studio to an FSP project environment.

3.1 Differences in Handling between the Smart Configurator and FSP Configuration Perspectives

The following describes the changes in handling from the [Smart Configurator] perspective to the [FSP Configuration] perspective. The [Smart Configurator] view corresponds to the [RX Project Configuration Editor] view (red boxes in the following figures) and the [MCU/MPU Package] view corresponds to the [Visualization] view (blue boxes in the figures).

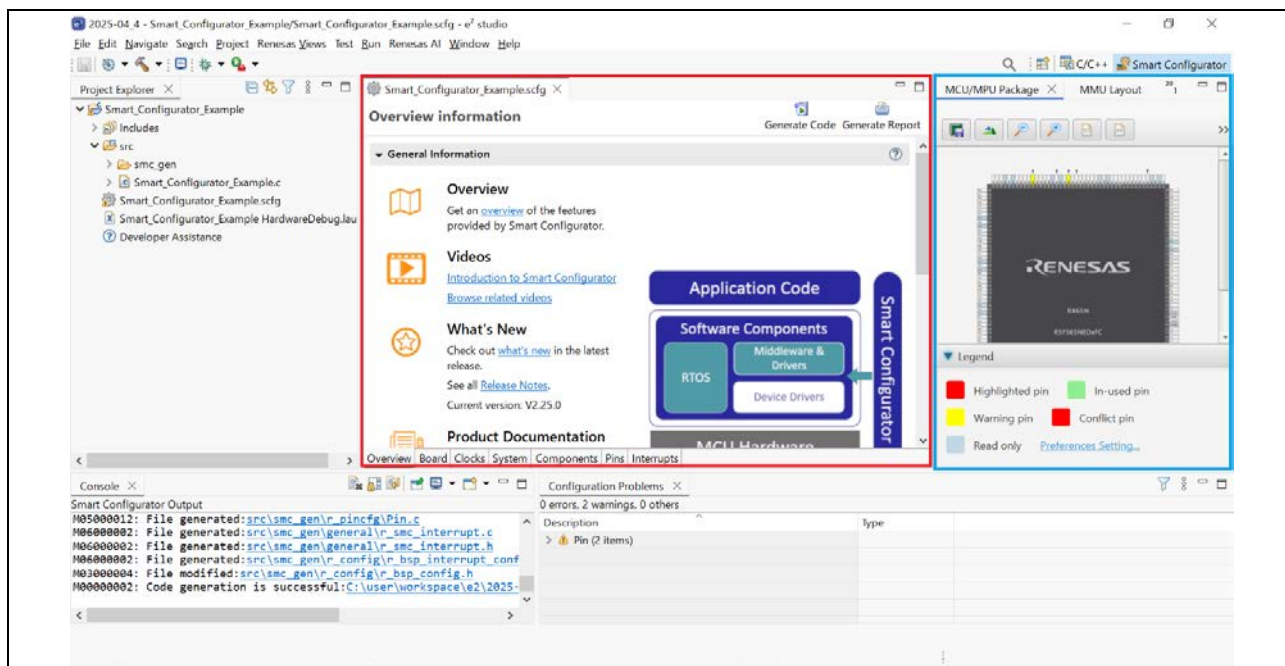


Figure 3-1 [Smart Configurator] Perspective

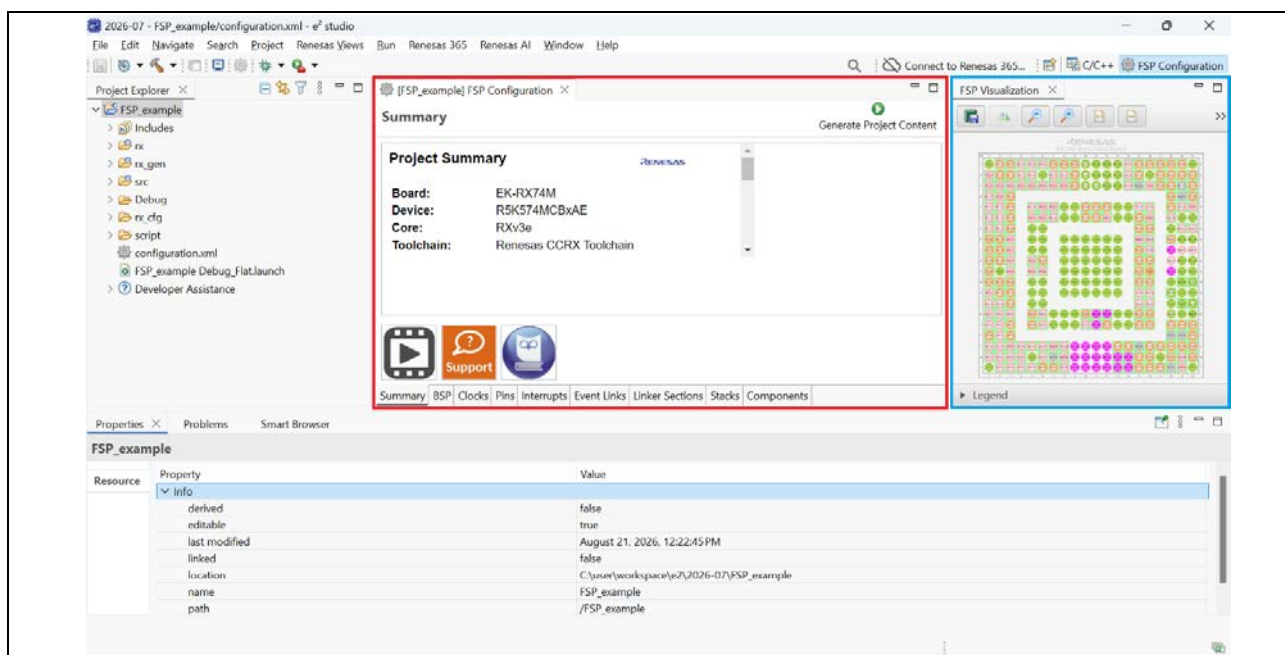


Figure 3-2 [FSP Configuration] Perspective

3.1.1 Checking the Overview and Component State

In the [Smart Configurator] view, the [Overview] page is used to check information such as the functional overview, introductory video, latest information, and FAQ for the Smart Configurator, and the state of the components and versions of the modules used in the project.

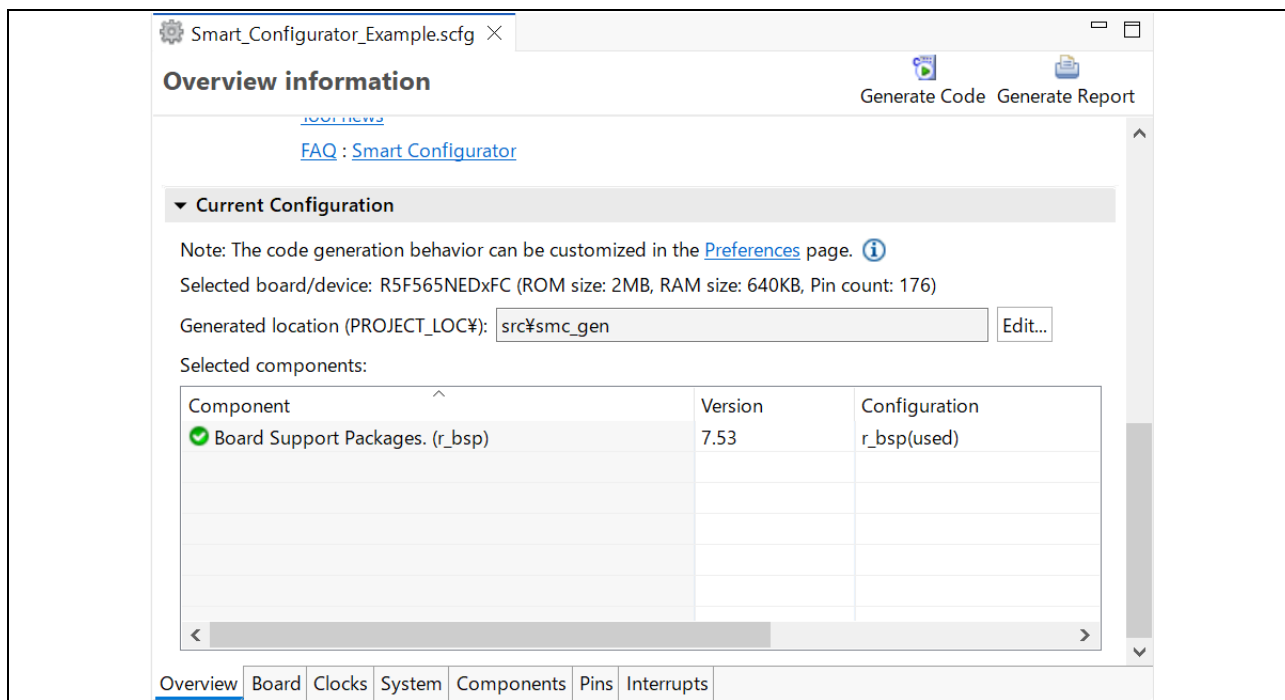


Figure 3-3 [Overview] Page of the [Smart Configurator] View

In the [RX Project Configuration Editor] view of the FSP, the [Summary] page is used to check information such as the versions of the board, device, and compiler, as well as the state and versions of the components used in the project.

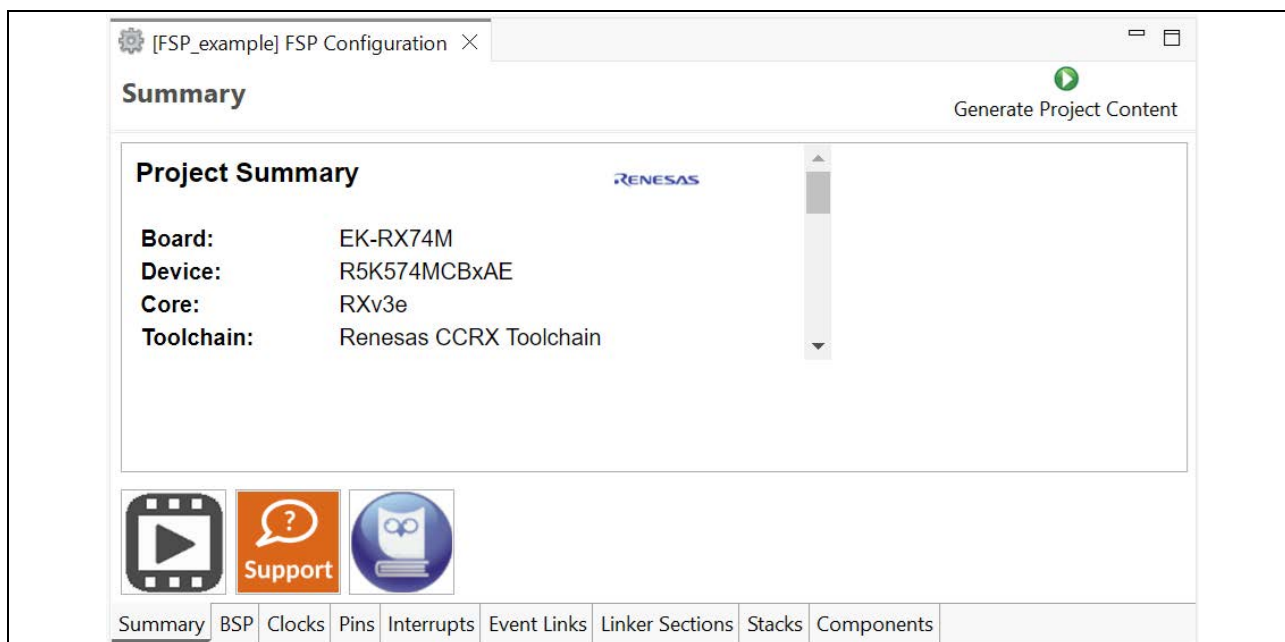


Figure 3-4 [Summary] Page

3.1.2 Selecting the Board and Device

In the [Smart Configurator] view, the [Board] page is used to select the board and device.

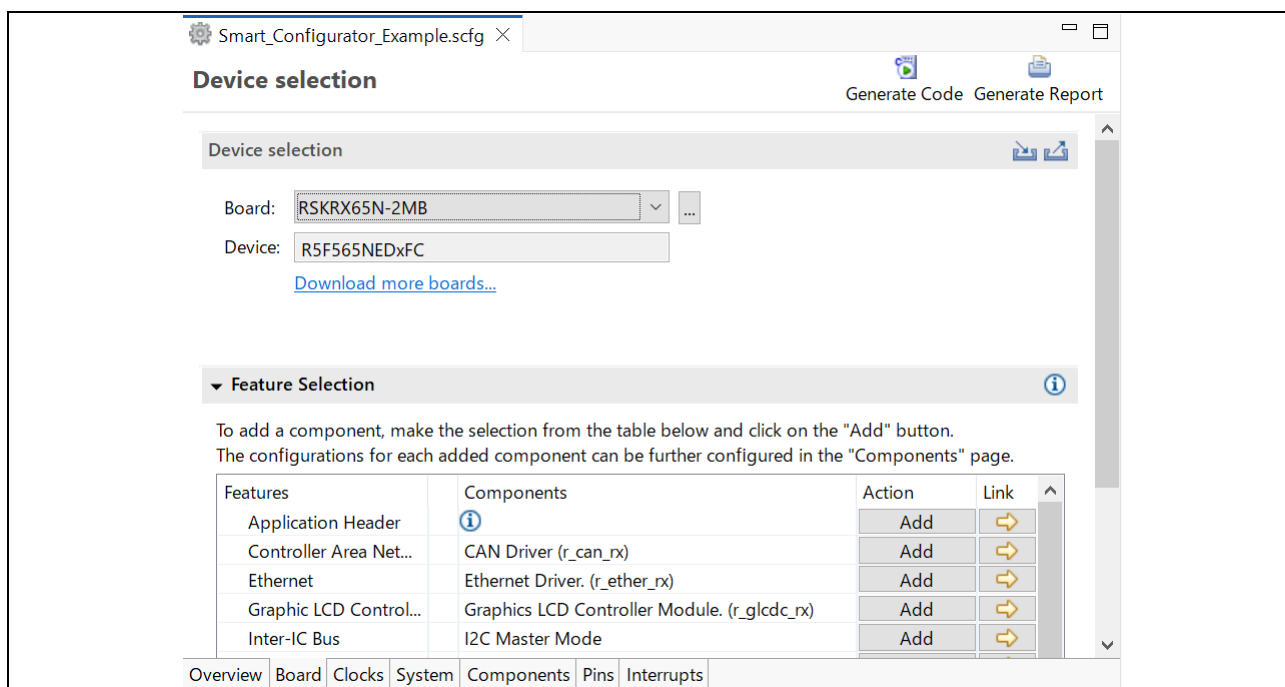


Figure 3-5 [Board] Page of the [Smart Configurator] View

In the [RX Project Configuration Editor] view, the [BSP] page is used to select the version of the FSP, the board, and the device.

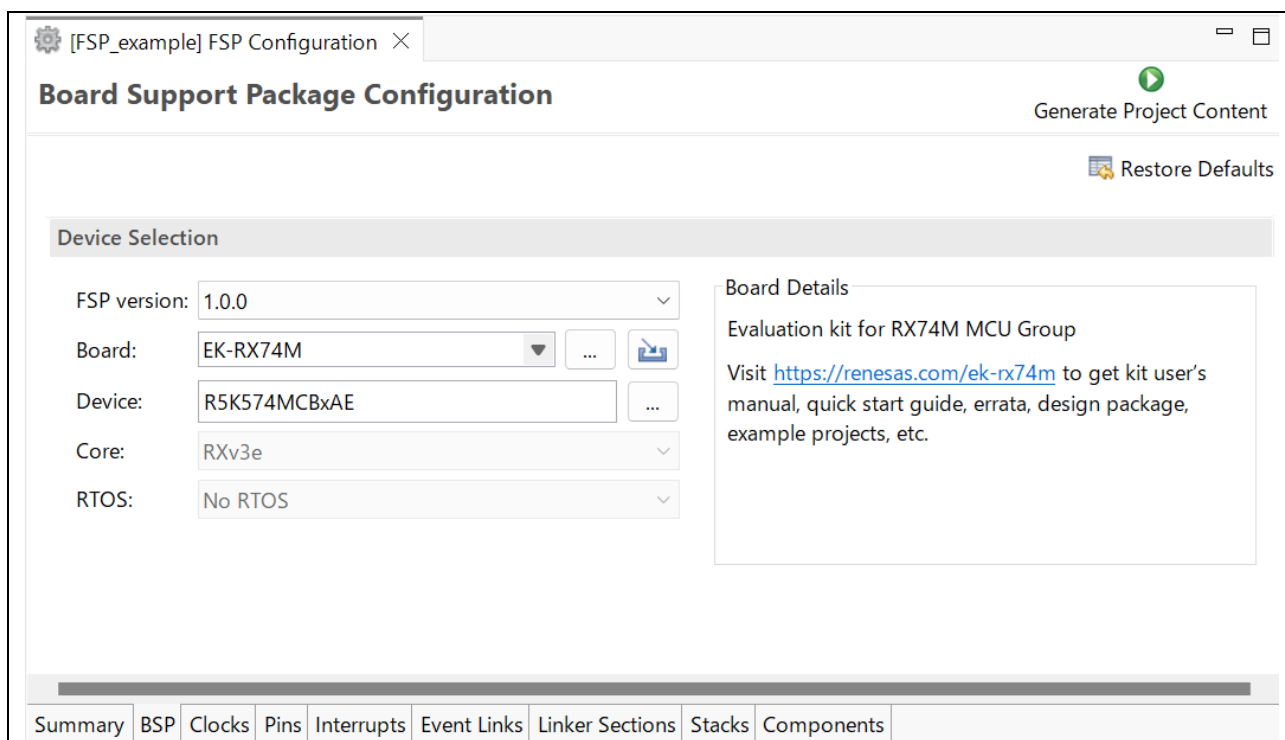


Figure 3-6 [BSP] Page

3.1.3 Clock Configuration

In the [Smart Configurator] view, the [Clocks] page is used to configure the clocks.

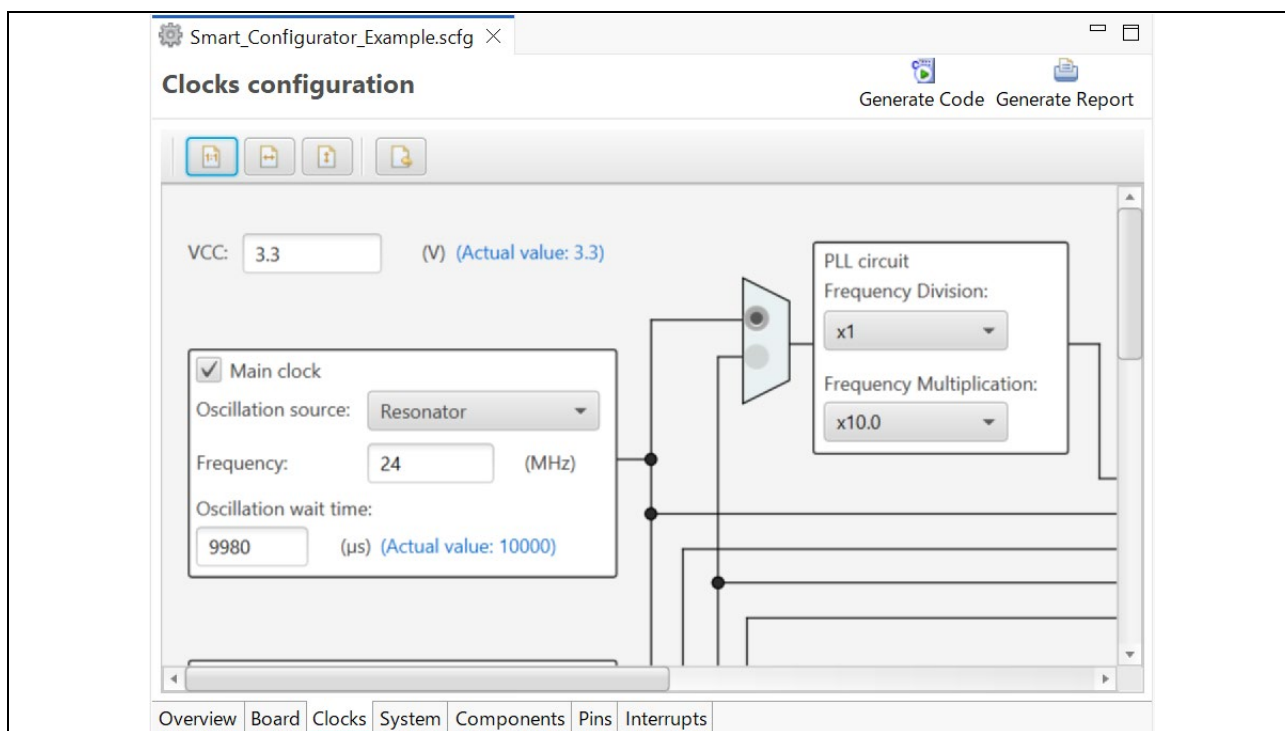


Figure 3-7 [Clocks] Page of the [Smart Configurator] View

In the [RX Project Configuration Editor] view, the [Clocks Configuration] page is used to configure the clocks. However, note that oscillation of any clock that works as a clock source can be stopped in the [Smart Configurator] view but clock oscillation cannot be stopped on the [Clocks Configuration] page.

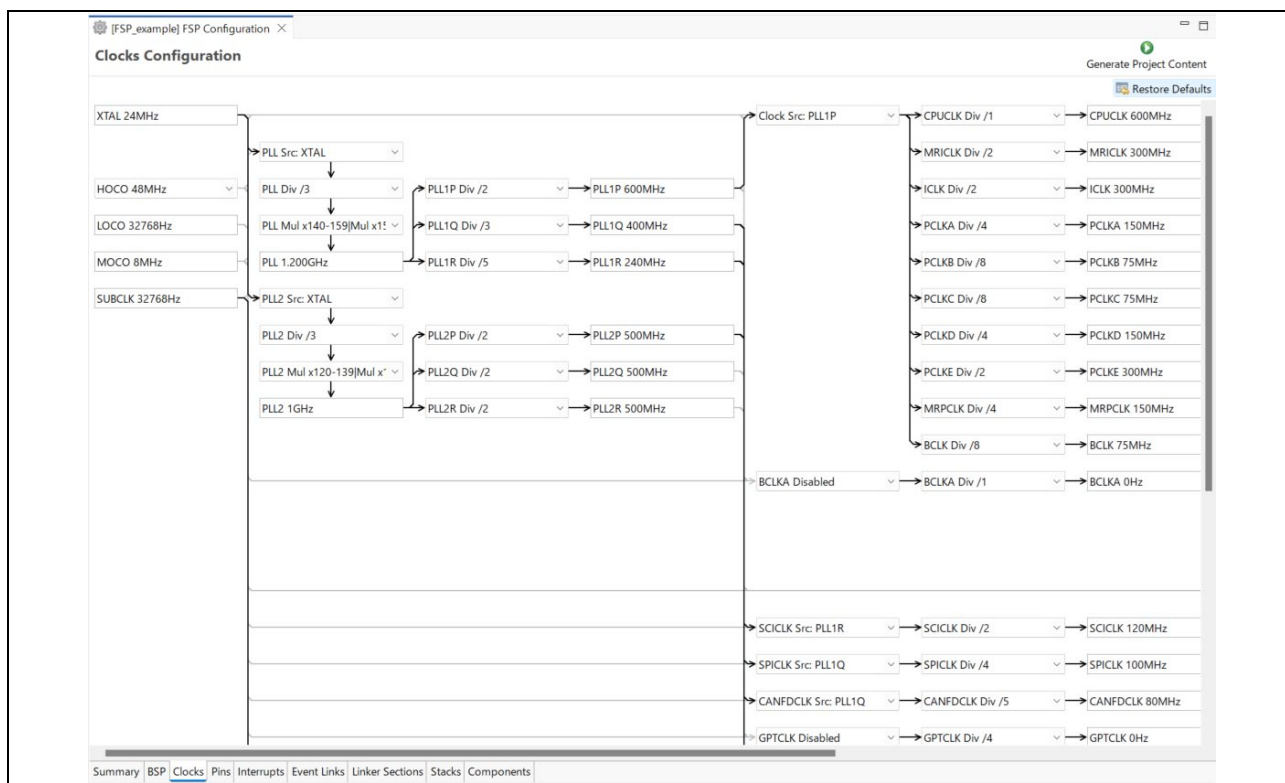


Figure 3-8 [Clocks Configuration] Page

Oscillation can only be started or stopped for the clock sources that are selectable in the [Properties] view when the [BSP] page is selected.

Summary **BSP** Clocks Pins Interrupts Event Links Linker Sections Stacks Components

Properties  Problems Smart Browser

EK-RX74M

Settings	Property	Value
	Clock Registers not Reset Values during Startup	Disabled
	Main Oscillator Populated	Populated
	PFS Protect	Enabled
	C Runtime Initialization	Enabled
	Early BSP Initialization	Disabled
	Main Oscillator Clock Source	Crystal or Resonator
	Subclock Populated	Populated
	Subclock Drive (Drive capacitance availability varies by MCU)	Standard/Normal mode
	Subclock Stabilization Time (ms)	1000

**Figure 3-9 Oscillation Start and Stop Settings in the [Properties] View
When the [BSP] Page is Selected**

Oscillation cannot be stopped for clock sources that cannot be selected in the [Properties] view. Such sources include the on-chip oscillators such as the HOCO and LOCO. To stop such a clock source, implement user-specified processing to stop oscillation at an appropriate location in the source code such as one of those listed in the following examples of locations for implementation.

Table 3.1 Examples of Locations for Implementing Processing to Stop Oscillation

No.	File Name	Location for Implementation
1	bsp_clocks.c	In the bsp_clock_init function (clock initialization function) ^{Note}
2	hal_warmstart.c	In a conditional branch with BSP_WARM_START_POST_CLOCK (immediately after clock initialization) or BSP_WARM_START_POST_C in the R_BSP_WarmStart function
3	hal_entry.c	In the hal_entry function (after completion of startup)

Note: If the FSP version is changed and [Generate Project Content] is executed after processing is implemented in this file, the code for the processing will be lost. Implement the processing again in such cases.

3.1.4 Debug Interface Configuration

In the [Smart Configurator] view, the [System] page is used to configure the debug interface.

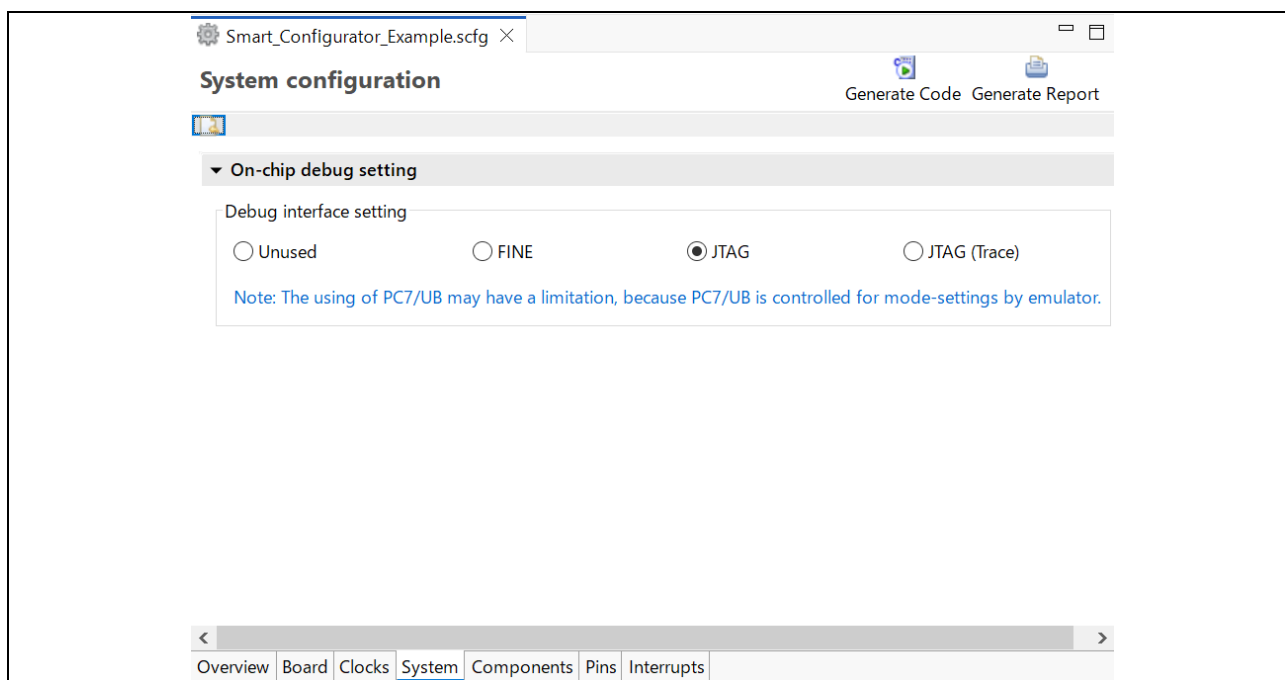


Figure 3-10 [System] Page of the [Smart Configurator] View

The [RX Project Configuration Editor] view does not have a page for configuring a debug interface. The pins for use in the debug interface should be set in the [Visualization] view or on the [Pin Configuration] page (see section 3.1.6, Pin Configuration). The type of debug interface should be set as part of the debugger configuration in the [Debug Configurations] window.

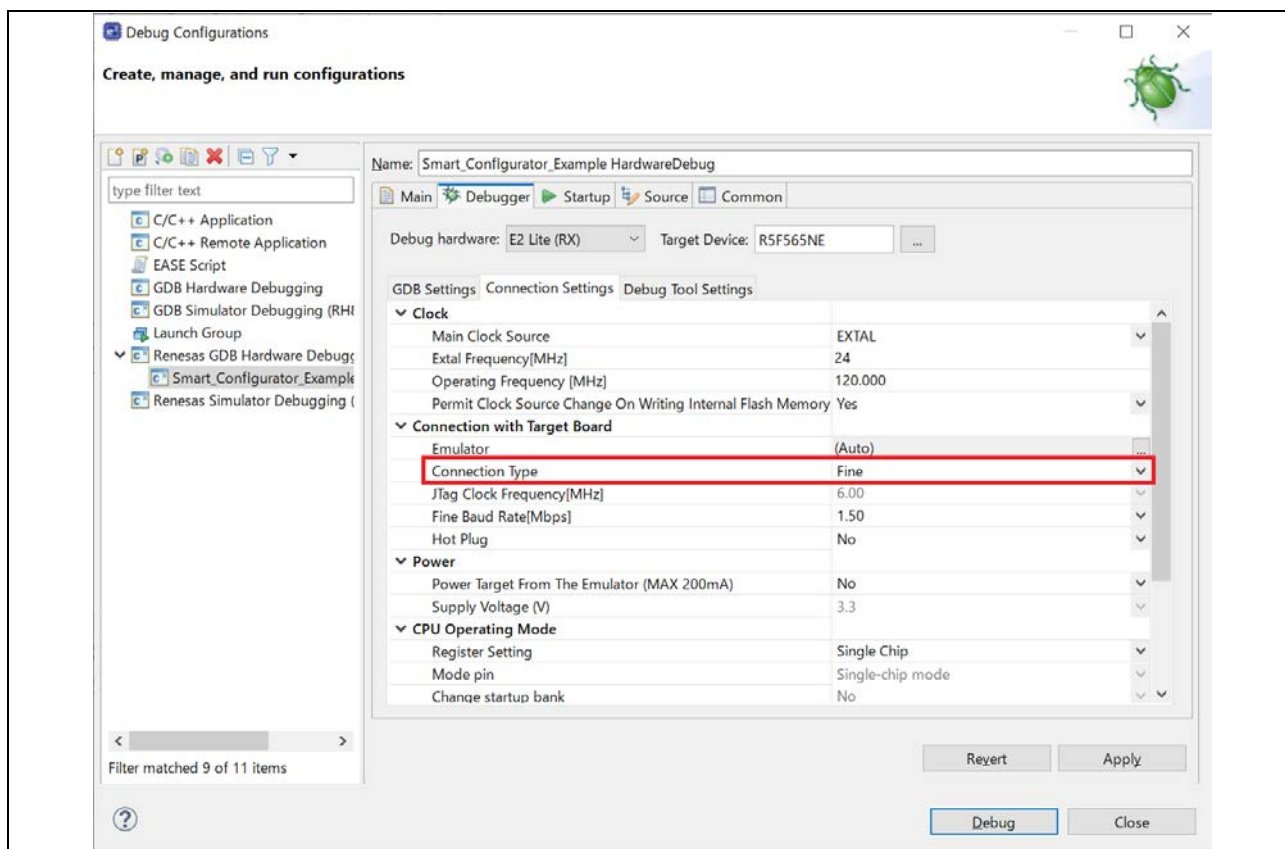


Figure 3-11 [Debug Configurations] Window

3.1.5 Component Configuration

In the [Smart Configurator] view, the [Board] or [Components] page is used to add a component. Either of two types of components is selectable: [Firmware Integration Technology] (FIT modules) or [Code Generator] (Code Generator components), and details of the component can be set in [Property] after the component module has been selected on the [Components] page.

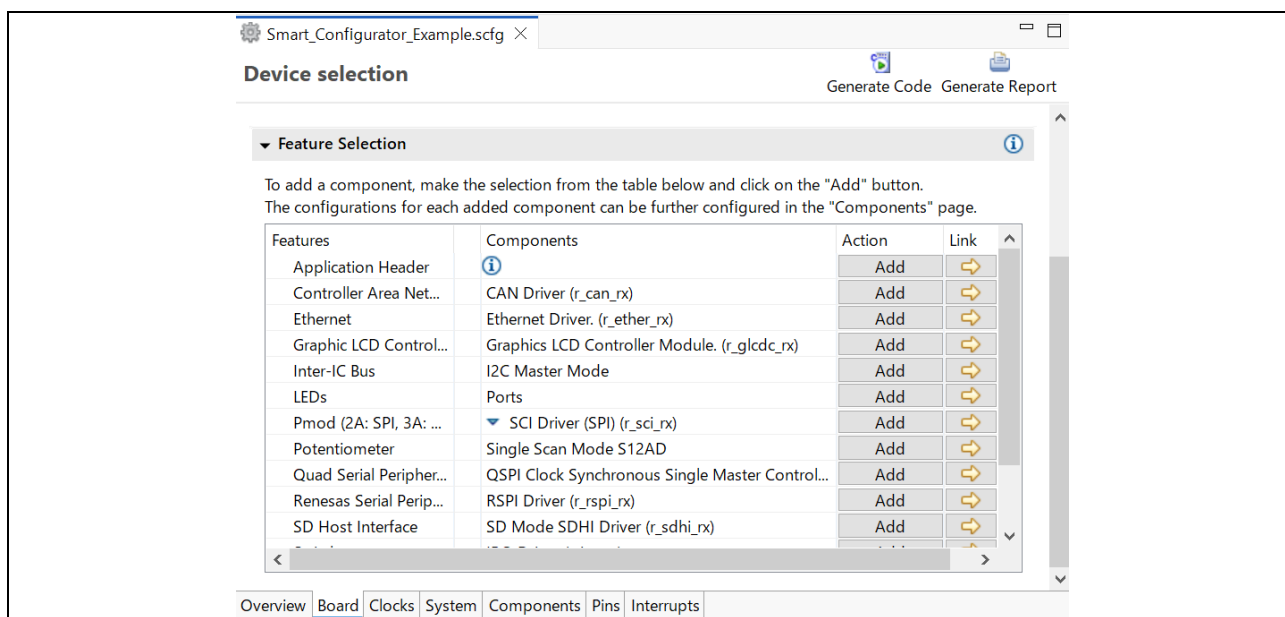


Figure 3-12 Selecting Functions on the [Board] Page of the [Smart Configurator] View

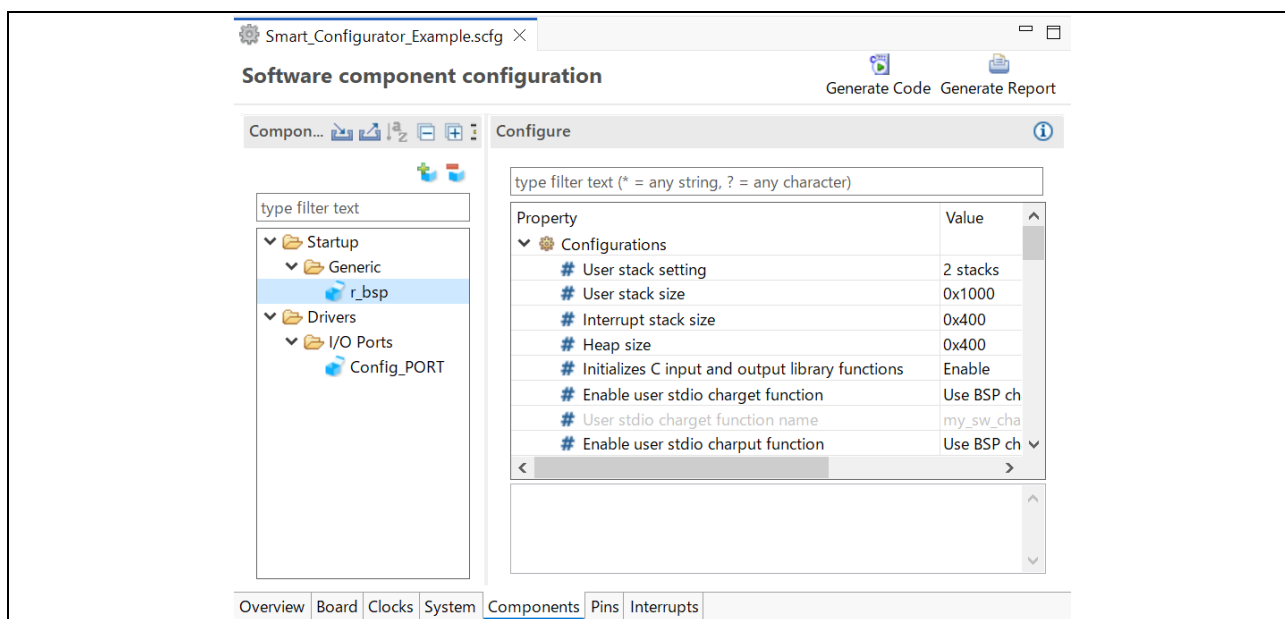


Figure 3-13 BSP Properties on the [Components] Page of the [Smart Configurator] View

In the [RX Project Configuration Editor] view, the [Stacks Configuration] or [Components Configuration] page is used to add a component (a module). If a component is added on the [Components Configuration] page, settings on the [Stacks Configuration] page are also necessary. In most cases, therefore, add a component on the [Stacks Configuration] page. Details of the component can be set in the [Properties] view. To make detailed settings of the BSP, select the [BSP] page and then set details in the [Properties] view. For a component other than the BSP, select the module of the target component on the [Stacks Configuration] page and then make detailed settings in the [Properties] view.

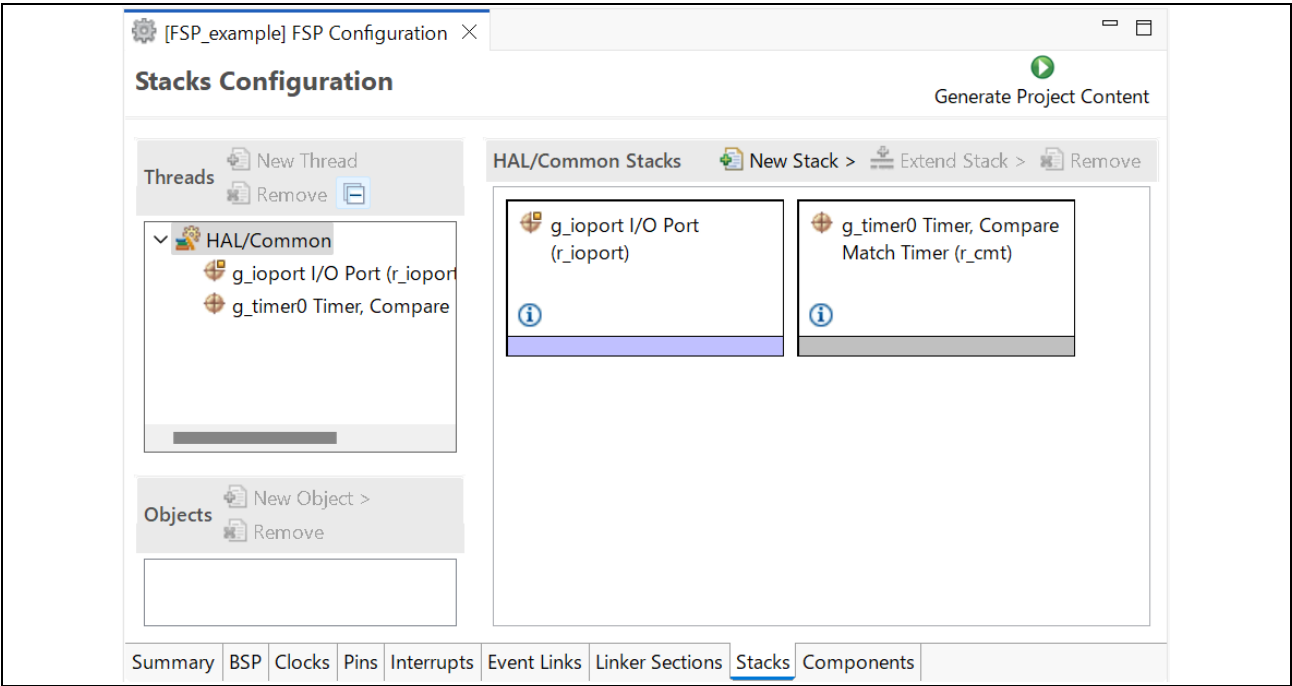


Figure 3-14 [Stacks Configuration] Page

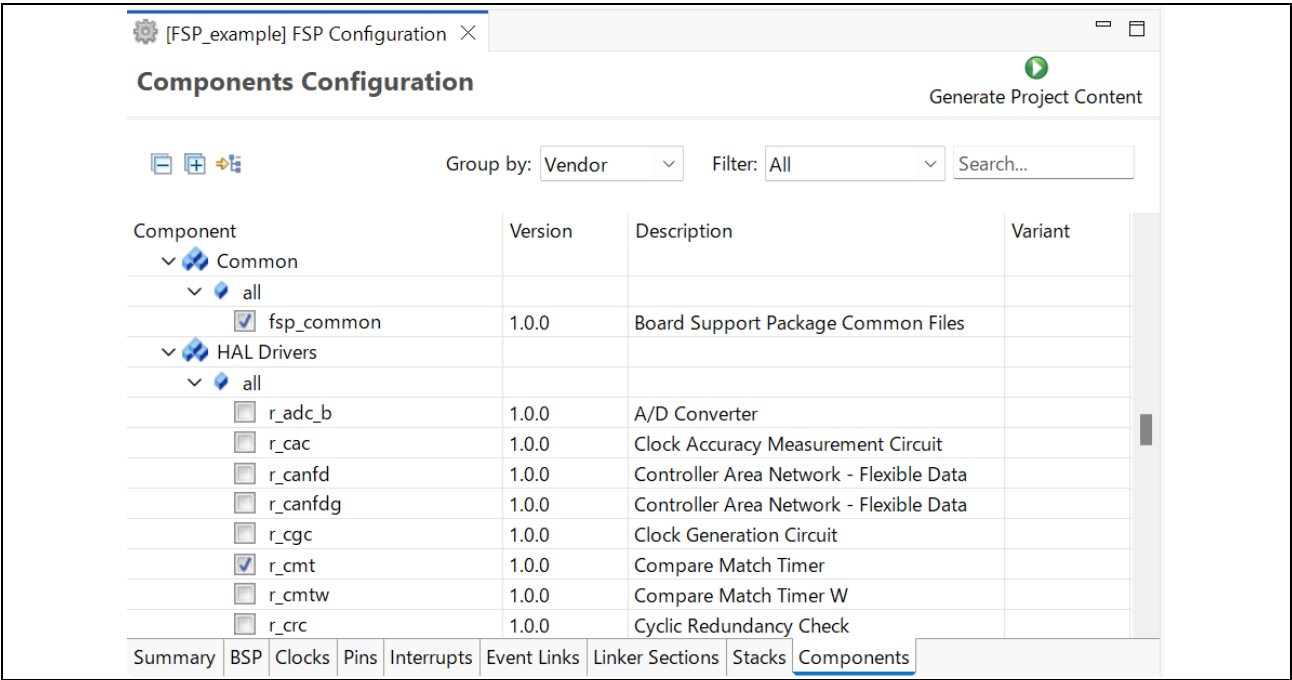


Figure 3-15 [Components Configuration] Page

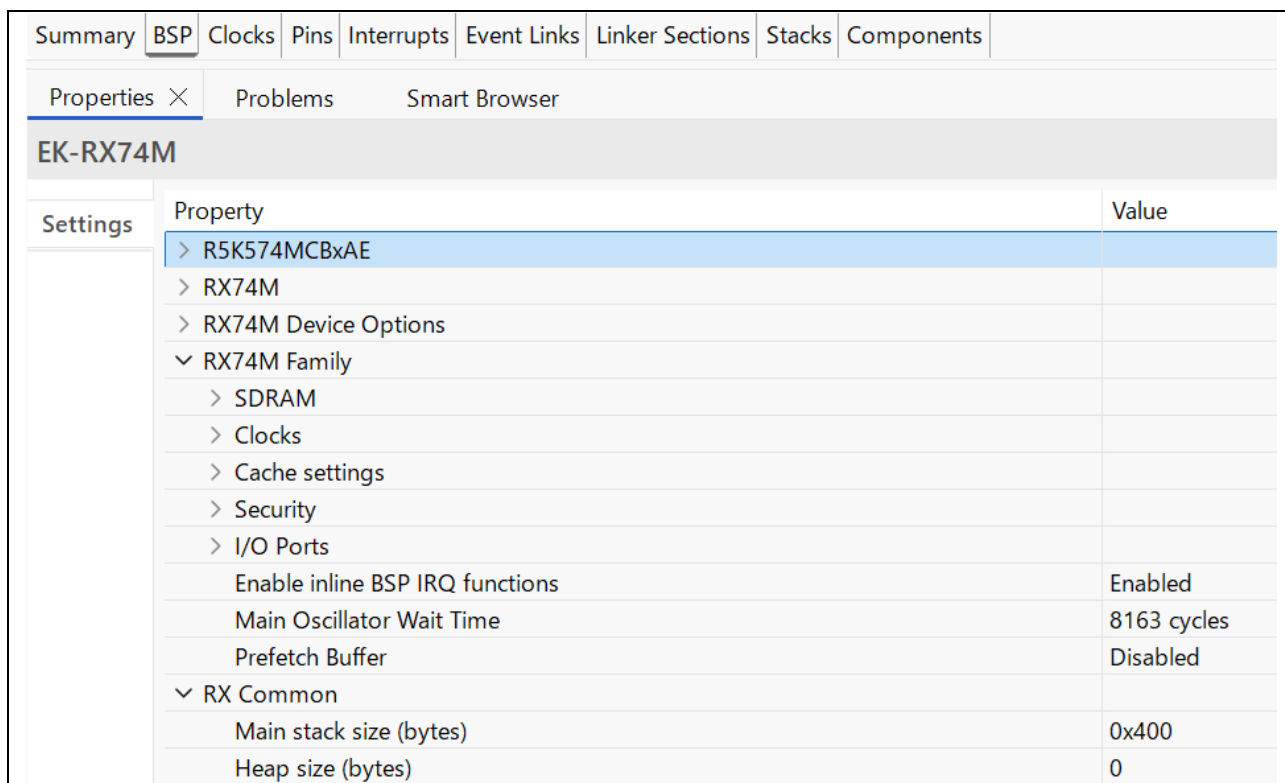


Figure 3-16 [Properties] View When the [BSP] Page is Selected

3.1.6 Pin Configuration

In the [Smart Configurator] view, the [Pins] page or [MCU/MPU Package] view is used to assign pin functions.

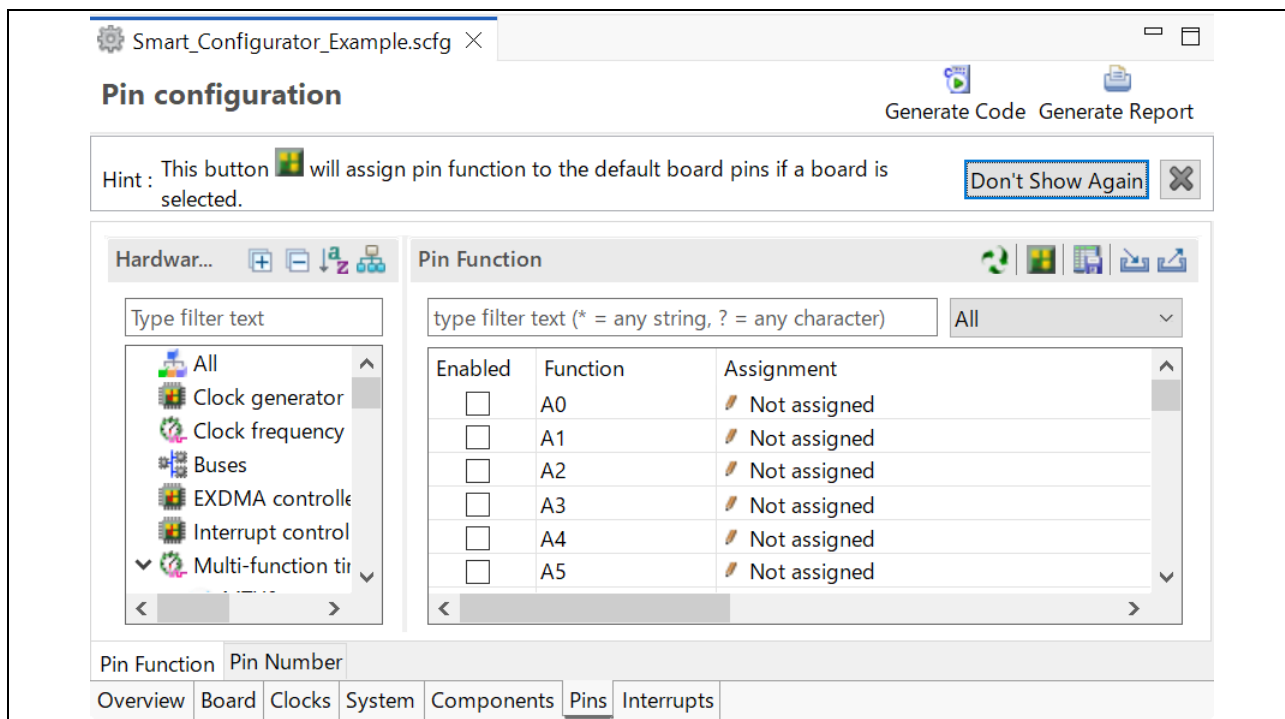


Figure 3-17 [Pins] Page of the [Smart Configurator] View

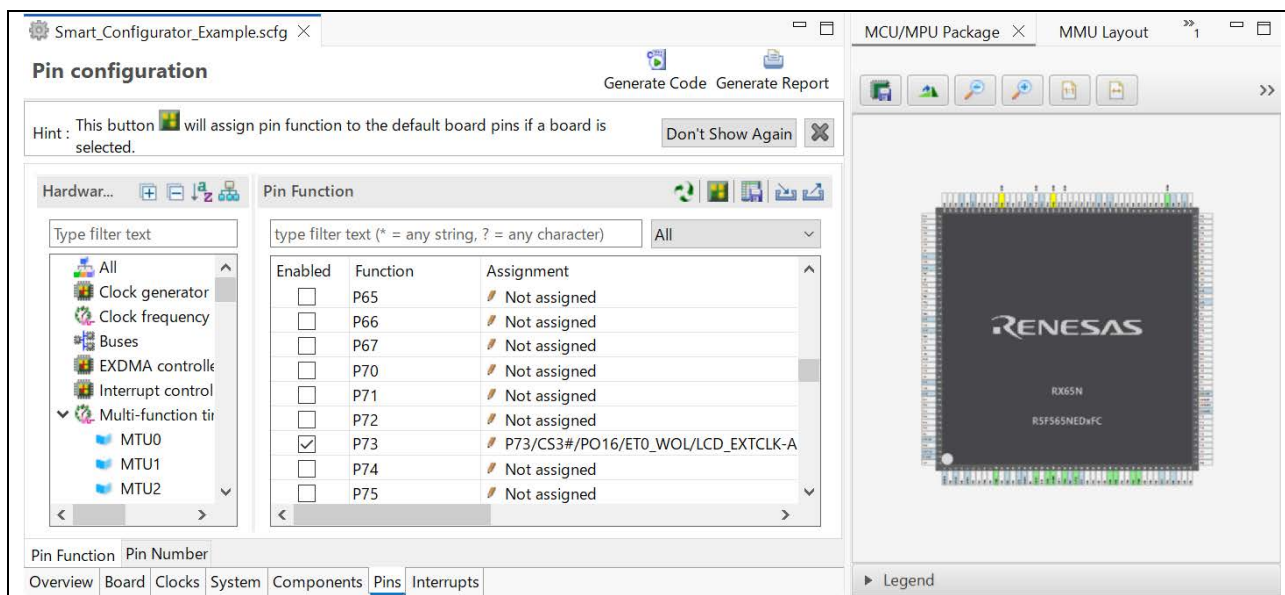


Figure 3-18 [Pins] Page of the [Smart Configurator] View and [MCU/MPU Package] View

In the [RX Project Configuration Editor] view, the [Pin Configuration] page or [Visualization] view is used to assign pin functions.

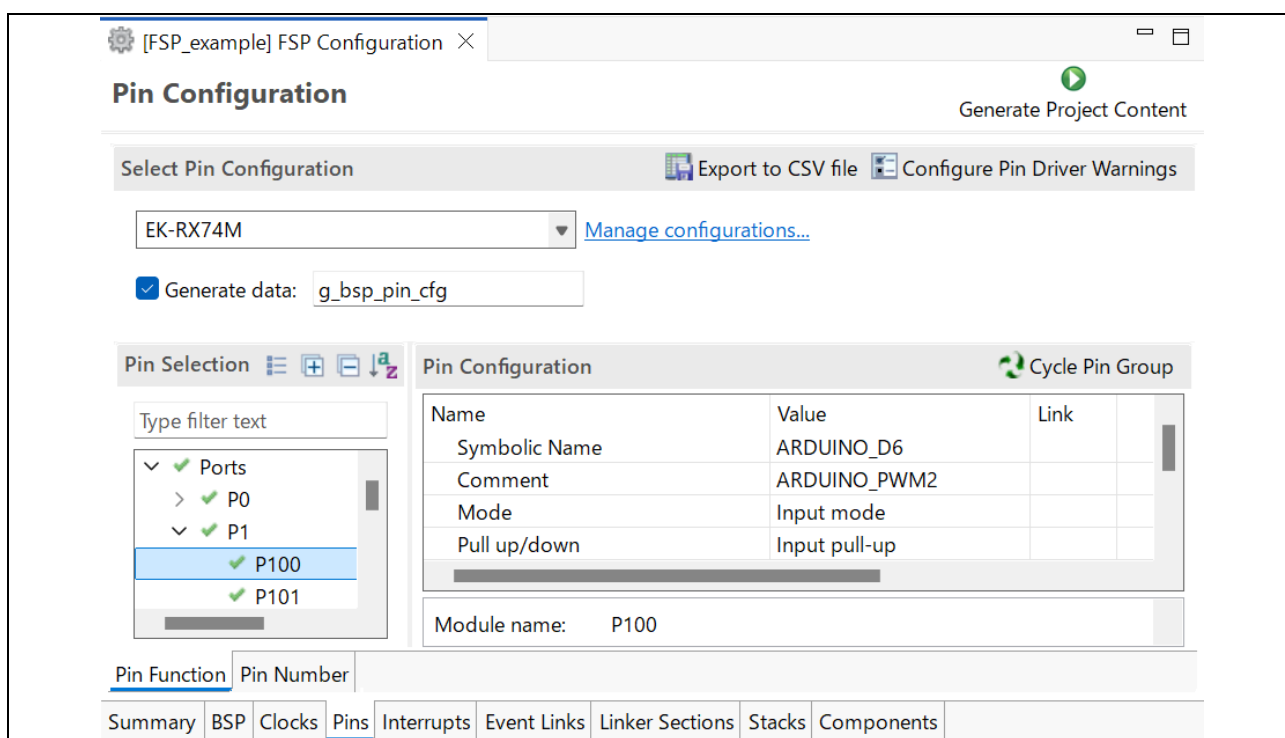


Figure 3-19 [Pin Configuration] Page

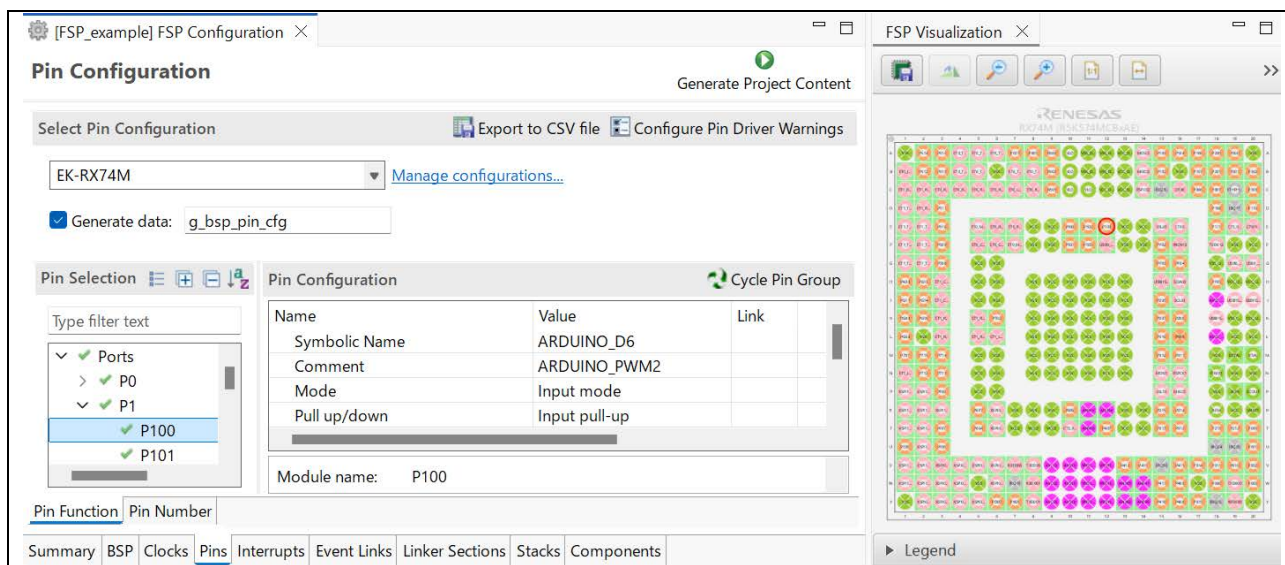


Figure 3-20 [Pin Configuration] Page and [Visualization] View

3.1.7 Interrupt Configuration

In the [Smart Configurator] view, the [Interrupts] page is used to configure interrupts.

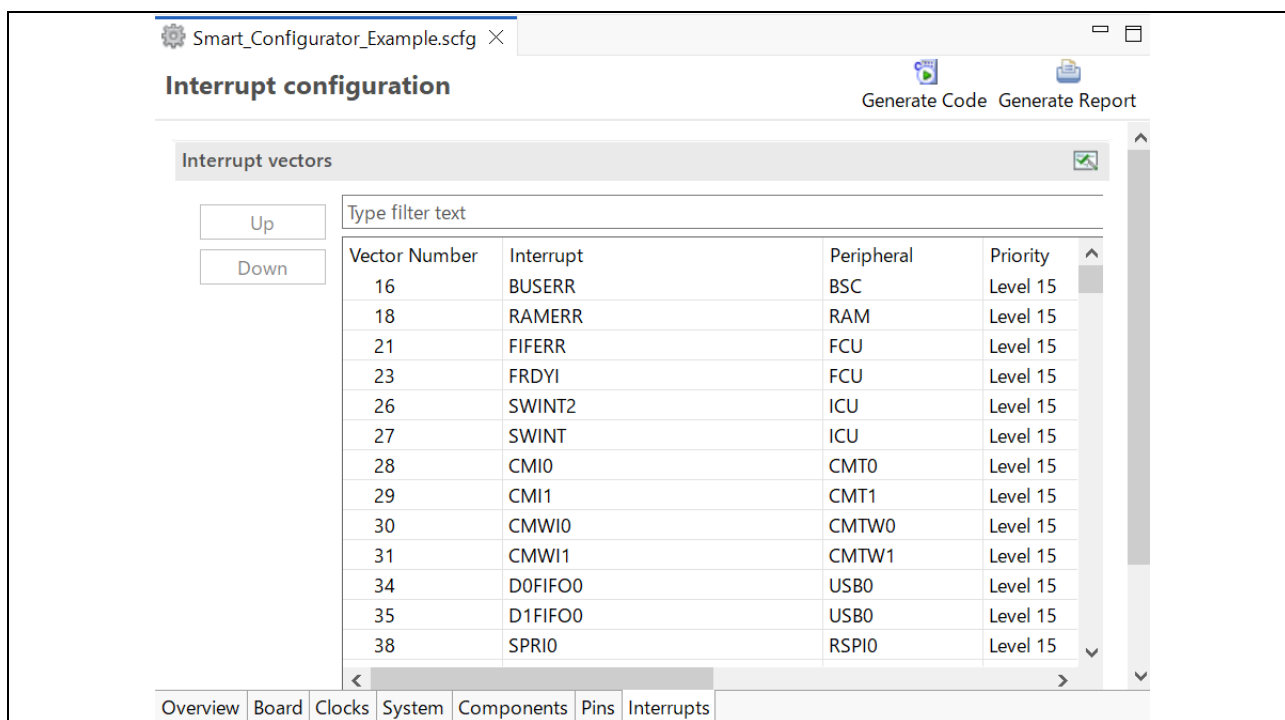


Figure 3-21 [Interrupts] Page of the [Smart Configurator] View

In the [RX Project Configuration Editor] view, the [Interrupts Configuration] page is used to configure interrupts.

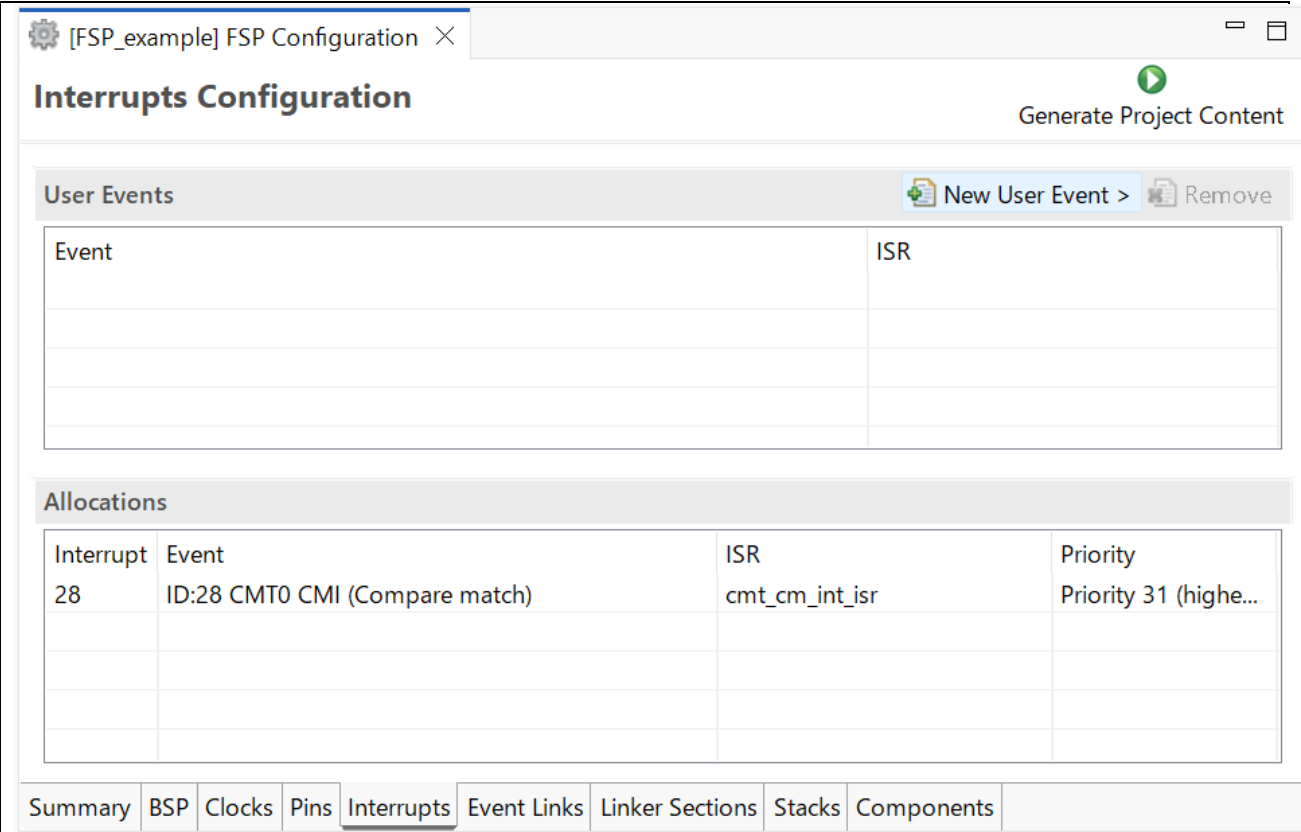


Figure 3-22 [Interrupts Configuration] Page

3.1.8 Event Link Configuration

In the [Smart Configurator] view, the event link function can be configured by adding the event link controller of the Code Generator type or the ELC module of the Firmware Integration Technology type on the [Components] page and then making settings in [Property].

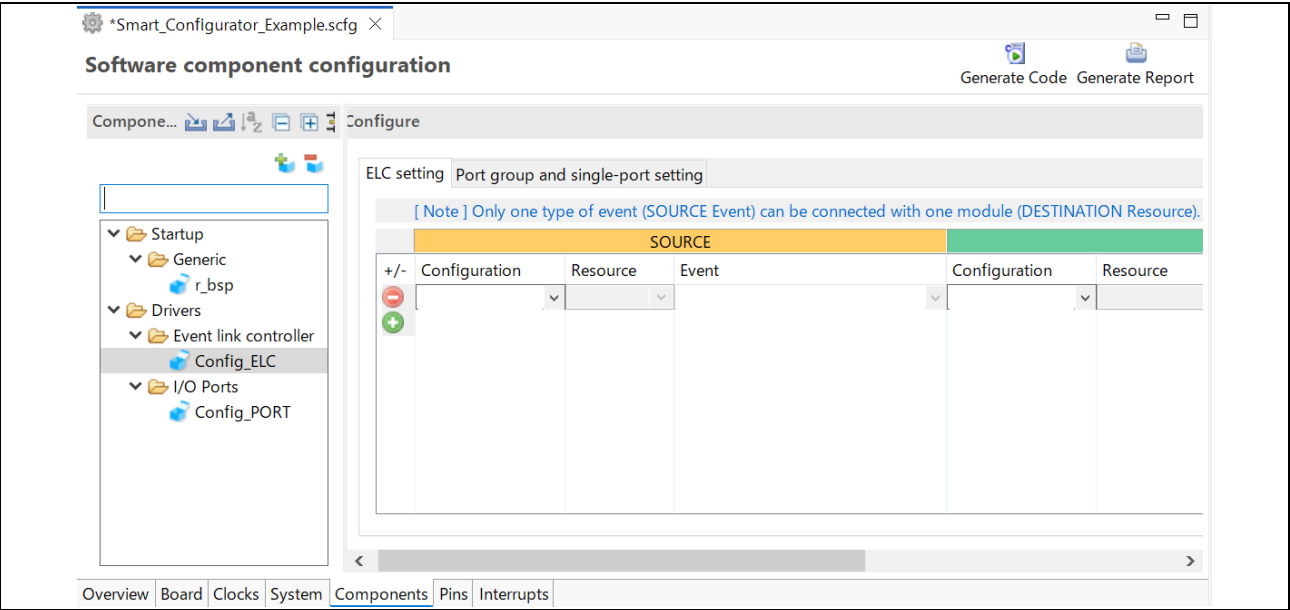


Figure 3-23 Config_ELC on the [Components] Page of the [Smart Configurator] View

In the [RX Project Configuration Editor] view, the [Event Links Configuration] page is used to configure event links.

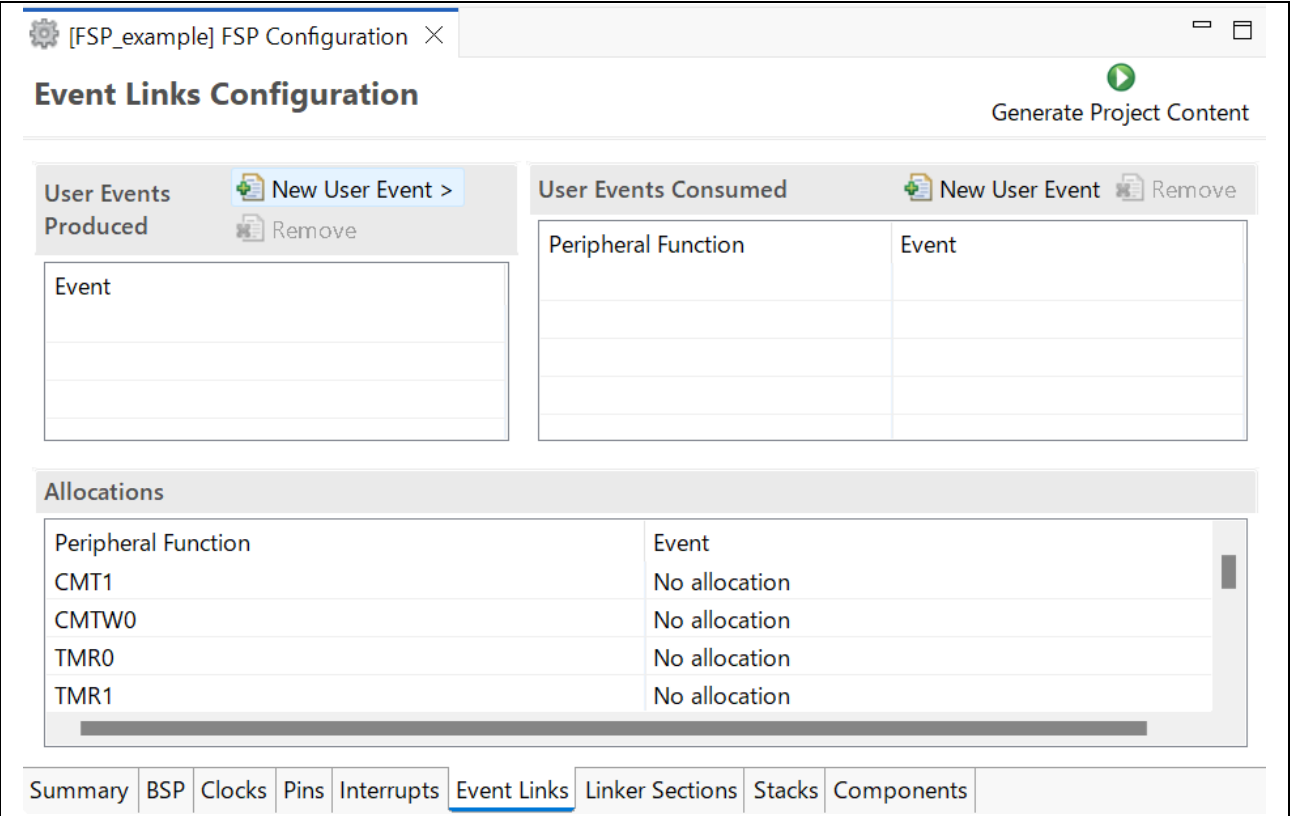


Figure 3-24 [Event Links Configuration] Page

4. Comparison of Software Specifications

4.1 Comparison of the Functions Supported by the Code Generator of the Smart Configurator, FIT Modules, and FSP Modules

4.1.1 Comparison of the Supported Peripheral Functions

The following shows whether and how the individual peripheral functions are supported by the Code Generator, FIT modules, and FSP modules.

Table 4.1 Comparison of the Supported Peripheral Functions (1)

Function Name	Code Generator	FIT Module	FSP Module
Basic settings	— (r_bsp is used)	r_bsp	bsp fsp_common
Battery backup function (VBATT)	—	r_vbatt_rx	(Contained in bsp)
Voltage detection circuit (LVD)	Same name as that to the left	r_lvd_rx	r_pvd
Low power consumption (LPC)	Same name as that to the left	r_lpc_rx	r_lpm
Interrupt controller (IRQ)	Same name as that to the left	r_irq_rx	r_icu
Data transfer controller (DTC)	Same name as that to the left	r_dtc_rx	r_dtc
DMA controller (DMAC)	Same name as that to the left	r_dmaca_rx	r_dmac
I/O ports	I/O port	r_gpio_rx	r_ioport
Multi-function pin controller (MPC)	—	r_mpc_rx	— (Note)
Compare match timer (CMT)	Same name as that to the left	r_cmt_rx	r_cmt
Compare match timer W (CMTW)	Compare match timer	r_cmtw_rx	r_cmtw
Realtime clock (RTC)	Same name as that to the left	r_rtc_rx	r_rtc
Low power timer (LPT)	Same name as that to the left	r_lpt_rx	— (Note)
Independent watchdog timer (IWDT)	Watchdog timer	r_iwdt_rx	r_iwdt
Watchdog timer (WDTA)	Same name as that to the left	r_wdt_rx	r_wdt
8-bit timer (TMR)	Same name as that to the left	—	r_tmr
Port output enable (POE)	Same name as that to the left	—	— (Note)
GPTW port output enable (POEG)	Port output enable	—	r_poeg
General PWM timer (GPTW)	General PWM timer (GPT)	—	r_gpt r_gpt_three_phase
Multi-function timer pulse unit (MTU)	Normal mode timer	—	— (Note)
	PWM mode timer	—	— (Note)
	Phase counting mode timer	—	— (Note)
	Complementary PWM mode timer	—	— (Note)
	Motor	—	— (Note)
	Dead time compensation counter	—	— (Note)
Programmable pulse generator (PPG)	Same name as that to the left	—	— (Note)
Buses	Same name as that to the left	—	(Contained in bsp)
Clock frequency accuracy measurement circuit	Same name as that to the left	—	r_cac
Comparator	Same name as that to the left	—	— (Note)
CRC calculator (CRC)	Same name as that to the left	—	r_crc
Data operation circuit	Same name as that to the left	—	r_doc

Note: To be supported in the future.

Table 4.2 Comparison of the Supported Peripheral Functions (2)

Function Name	Code Generator	FIT Module	FSP Module
Serial communications interface (SCI)	SCI/SCIF asynchronous mode	r_sci_rx, r_scif_rx	— (Note)
	SCI/SCIF clock synchronous mode		
	Smart card interface mode	—	— (Note)
	I2C master mode	r_sci_iic_rx	— (Note)
	SPI clock synchronous mode (3-wire method)	—	— (Note)
Serial communications interface (RSCI)	SCI/SCIF asynchronous mode	r_rsci_rx	r_sci_b_uart
	SCI/SCIF clock synchronous mode		— (Note)
	Smart card interface mode	—	— (Note)
	I2C master mode	r_rsci_iic_rx	r_sci_b_i2c
	SPI clock synchronous mode (3-wire method)	—	r_sci_b_spi
I2C bus interface (RIIC)	I2C master mode	r_riic_rx	— (Note)
	I2C slave mode		— (Note)
High-speed I2C bus interface (RIICHS)	—	r_riichs_rx	r_iic_b_master r_iic_b_slave
I3C bus interface (RI3C)	—	r_i3c_rx	r_i3c
Serial peripheral interface (RSPI)	SPI clock synchronous mode (3-wire method)	r_rsapi_rx	— (Note)
	SPI operation mode (4-wire method)		
Serial peripheral interface (RSPIA)	—	r_rsapia_rx	r_spi_b
Quad serial peripheral interface (QSPI)	—	r_qspix_smstr_rx	— (Note)
Quad-SPI memory interface (QSPIX)	—	r_qspix_rx	— (Note)
USB2.0FS host/function module (USB)	—	r_usb_basic	r_usb_basic
		r_usb_basic_mini	— (Note)
		r_usb_hmsc	r_usb_hmsc
		r_usb_hmsc_mini	— (Note)
		r_usb_hcdc	r_usb_hcdc
		r_usb_hcdc_mini	— (Note)
		r_usb_hhid	r_usb_hhid
		r_usb_hhid_mini	— (Note)
		r_usb_pmsc	r_usb_pmsc
		r_usb_pmsc_mini	— (Note)
		r_usb_pcdc	r_usb_pcdc
		r_usb_pcdc_mini	— (Note)
		r_usb_phid	r_usb_phid
		r_usb_phid_mini	— (Note)

Note: To be supported in the future.

Table 4.3 Comparison of the Supported Peripheral Functions (3)

Function Name	Code Generator	FIT Module	FSP Module
PTP module for the Ethernet controller (EPTPC)	—	r_ptp_rx	r_gptp
EPTPC Light module (EPTPC Light: device driver for enhanced Ethernet function)	—	r_ptp_light_rx	— (Note)
EtherCAT slave controller (ESC)	—	r_ecat_rx	r_ethercat_phy
Ethernet controller (ETHERC)	—	r_ether_rx	r_ether r_ether_phy
Controller area network (CAN)	—	r_can_rx	—
Controller area network (CANFD)	—	r_canfd_rx	r_canfd r_canfd_f
Controller area network (RSCAN)	—	r_rscan_rx	— (Note)
IrDA interface	—	r_irda_sci_rx	— (Note)
Parallel data capture unit (PDC)	—	r_pdc_rx	r_pdc
SD host interface (SDHI)	—	r_sdhi_rx	r_sdhi
SD slave interface (SDSI)	—	r_sdsi_rx	— (Note)
MultiMediaCard interface (MMCIF)	—	r_mmcif_rx	— (Note)
Delta-sigma modulator interface (DSMIF)	Same name as that to the left	r_dsmif_rx	r_dsmif
Remote control signal receiver	Same name as that to the left	—	— (Note)
12-bit A/D converter (S12AD)	Single scan mode S12AD	r_s12ad_rx	— (Note)
	Continuous scan mode S12AD		
	Group scan mode S12AD		
12-bit D/A converter (R12DA)	D/A converter	r_dac_rx	r_dac
Analog front end (AFE)	Same name as that to the left	—	— (Note)
Sampling rate converter (SRC)	—	r_src_api_rx	— (Note)
Serial sound interface (SSIE)	—	r_ssi_api_rx	r_ssi
Flash memory (FLASH)	—	r_flash_rx	— (Note)
Trusted Secure IP (TSIP)	—	r_tsip_rx	— (Note)
Renesas Secure IP (RSIP)	—	r_rsip_protected_rx	r_rsip_protected
		r_rsip_cm_rx	— (Note)
LCD controller/driver (LCDC)	LCD controller	r_lcdc_rx	— (Note)
Graphic LCD controller (GLCDC)	—	r_glcddc_rx	— (Note)
Unique ID reading	—	r_uid_rx	— (Note)
Byte queue buffer (data management)	—	r_byteq	— (Note)
Long queue buffer (data management)	—	r_longq	— (Note)
Capacitive touch sensor (CTSU)	—	r_ctsu_qe	— (Note)
Event link controller (ELC)	Same name as that to the left	r_elc_rx	r_elc
RX DRW2D driver	—	r_drw2d_rx	— (Note)

Note: To be supported in the future.

4.1.2 Comparison of the Supported Middleware Functions

The following shows whether FIT modules and FSP modules for middleware are provided for the individual functions.

Table 4.4 Comparison of the Supported Middleware Functions (1)

Classification of Functions		FIT Module Name	FIT Module	FSP Module
Communi- cations	TCP/IP	lwIP (lightweight IP) Driver	r_lwip_driver_rx	rm_lwip_ether rm_lwip_sys_b aremetal rm_lwip_sys_fr eertos
		lwIP (lightweight IP)	r_lwip_rx	
	Wi-Fi	US159-DA16XXXMEVZ Wi-Fi Control Module	r_wifi_da16xxx	___ (Note)
	BLE	BLE Module (BLE)	r_ble_rx23w	___ (Note)
		QE Utility Module (Profiles)	r_ble_qe_utility	___ (Note)
		Mesh Module (Mesh)	r_mesh_rx23w	___ (Note)
		US159-DA14531EVZ BLE Control Module	r_ble_da1453x_rx	___ (Note)
	Cellular	RYZ014A/GM01Q Cellular Module Control Module	r_cellular	___ (Note)
	NFC	PTX NFC Control Module	r_nfc_ptx_rx	___ (Note)
File system device driver	SD memory card	SD memory card driver module	r_sdc_sdmem_rx	___ (Note)
		SPI Mode SD Memory Card Driver Firmware Integration Technology	r_sdc_spi_rx	___ (Note)
	Data flash memory driver	Flash Memory Data Management Module	r_datfrx_rx	___ (Note)
	Serial NOR flash memory	Clock Synchronous Control Module for Serial Flash Memory Access	r_flash_spi	___ (Note)
	SPI serial EEPROM driver	Clock Synchronous Control Module for EEPROM Access	r_eeeprom_spi	___ (Note)
	Memory access driver interface	Memory Access Driver Interface Module	r_memdrv_rx	___ (Note)
Graphic processing	JPEG	JPEG Decoder Module	r_jpegd_rx	___ (Note)
		JPEG Encoder Module	r_jpege_rx	___ (Note)
Graphics		Aeropoint module	r_aeropoint_rx	___ (Note)
		emWin module	r_emwin_rx	___ (Note)
		QE for Display module	r_qe_display_rx	___ (Note)

Note: To be supported in the future.

Table 4.5 Comparison of the Supported Middleware Functions (2)

Classification of Functions		FIT Module Name	FIT Module	FSP Module
Reality AI	Reality AI UART communication module	Reality AI UART Communication Module	rm_comms_uart_rx	___ (Note)
	Reality AI data collector control module	Reality AI Data Collector Control Module	rm_rai_data_collect or_rx	___ (Note)
	Reality AI data shipper control module	Reality AI Data Shipper Control Module	rm_rai_data_shippe r_rx	___ (Note)
	Reality AI USB PCDC communication module	Reality AI USB PCDC Communication Module	rm_comms_usb_pc dc_rx	rm_comms_usb_pc dc
Software library	Sound	Sound Playback/Compression System (Original ADPCM Codec) [M3S-S2-Tiny] Module	r_s2_rx	___ (Note)
	Open source FAT file system	Open Source FAT File System [M3S-TFAT-Tiny] Module	r_tfat_rx	rm_freertos_plus_fat
		M3S-TFAT-Tiny Memory Driver Interface Module	r_tfat_driver_rx	___ (Note)
Firmware update		Firmware update Module	r_fwup	___ (Note)
MCU boot		MCUboot Module	rm_mcuboot	rm_mcuboot_port
Firmware update communication		Firmware Update Communication Module	r_fwupcomm	___ (Note)
Sensor	Touch sensor module	QE Touch Module	rm_touch_qe	___ (Note)

Note: To be supported in the future.

4.2 Comparison of Initialization

4.2.1 Comparison of the BSP Specifications between the FIT and FSP Modules

The following shows the differences in initialization of the BSP between the FIT and FSP modules. For initialization of the BSP module by FIT, refer to section 2.2, Initialization, in the [Board Support Package Module Using Firmware Integration Technology](#).

Table 4.6 Comparison of the Initialization Specifications (1)

No.	Initialization Item of the BSP Module of FIT	Availability of the Equivalent Processing in the BSP Module of the FSP	Alternative Measures for Non-available Processing
1	Setting stack addresses	Yes	
2	Setting the start address of the interrupt vector table	Yes	
3	Setting the start address of the exception vector table (Notes: 1)	Yes	
4	Initializing the FPU (Notes: 1) R_BSP_SET_FPSW()	No (Notes: 3)	See 4.2.2.1.
5	Initializing the double-precision floating-point coprocessor (Notes: 1) R_BSP_SET_DPSW()	No (Notes: 3)	See 4.2.2.1.
6	Initializing the arithmetic unit for trigonometric functions (Notes: 1) R_BSP_INIT_TFU() or bsp_tfu_init()	No (Notes: 3)	See 4.2.2.2.
7	Waiting for stabilization of the VBATT voltage (Notes: 1) vbatt_voltage_stability_wait()	No (Notes: 3)	See 4.2.2.3.
8	Setting the system clock mcu_clock_setup()	Yes	
	Setting the clock division ratio and multiplication factor	Yes	
	Stopping the clocks not to be used as clock sources	Some processes are not available. The processes for stopping the HOCO and LOCO are not implemented.	See 4.2.2.4.
	Waiting for stabilization of the clocks to be used as clock sources	Yes	
	Switching the system clock	Yes	
9	Warm start function (PRE) (Notes: 2) User_Warm_start_func_pre()	Yes	
10	Initializing the C runtime environment _INIT_SCT()	Yes	
	Filling the area for uninitialized data with 0	Yes	
	Copying data to the area for initialized data	Yes	
11	Initializing the variables of BSP bsp_ram_initialize()	Yes	
12	Warm start function (POST) (Notes: 2) User_Warm_start_func_post()	Yes	
13	Initializing the STDIO library (Notes: 2) init_iolib()	No (Notes: 3)	See 4.2.2.5.

Notes: 1. Skipping this step may be possible, depending on the target MCU.

Notes: 2. The operation depends on the r_bsp_config.h settings.

Notes: 3. Specifications as of v1.0.0

Table 4.7 Comparison of the Initialization Specifications (2)

No.	Initialization Item of the BSP Module of FIT	Availability of the Equivalent Processing in the BSP Module of the FSP	Alternative Measures for Non-available Processing
14	Initializing the interrupt callback functions bsp_interrupt_open()	Yes	
	Initializing the interrupt callback functions	Yes, but only for group interrupts	
	Assigning interrupt sources to the software configurable interrupts (Note 1)	Yes, but only for the interrupts configured on the [Interrupts Configuration] page	
15	Initializing register protection bsp_register_protect_open()	Yes	
16	Initializing bus priority (Notes: 2) bsp_bus_priority_initialize()	No (Notes: 4)	See 4.2.2.6.
17	Initializing hardware and pins hardware_setup()	Yes	
	Enabling the pins used for peripheral functions	Yes, but only for the pins configured on the [Pin Configuration] page	
	Initializing the non-existent ports bsp_non_existent_port_init()	Yes	
18	Initializing the PSW R_BSP_SET_PSW()	Yes	
	Enabling or disabling the use of the user stack (Notes: 2)	No (Notes: 4)	See 4.2.2.7.
	Enabling acceptance of interrupts by the CPU (Notes: 3)	Yes	
19	Switching to user mode (Notes: 2) R_BSP_CHG_PMUSR()	No (Notes: 4)	See 4.2.2.8.
20	Jumping to the main function	Yes	
21	Closing the STDIO library (Notes: 2) close_all()	No (Notes: 4)	See 4.2.2.9.

Notes: 1. Skipping this step may be possible, depending on the target MCU.

Notes: 2. The operation depends on the r_bsp_config.h settings.

Notes: 3. Only acceptance of interrupts by the CPU is enabled. The interrupts to be generated in peripheral functions require individual enabling.

Notes: 4. Specifications as of v1.0.0

4.2.2 Alternative Measures for Processing Not Available in the FSP Module

4.2.2.1 CPU-Related Processing

When this processing is necessary, refer to the chapter describing the CPU in the User's Manual: Hardware of the target device, compiler manual, and source code of the BSP module of FIT and implement user-specified processing at an appropriate location, such as one of those listed in the following examples of locations for implementation.

Table 4.8 Examples of Locations for Implementing CPU-Related Processing

No.	File Name	Location for Implementation
1	system.c	In the SystemInit function ^(Note)
2	hal_warmstart.c	In a conditional branch with BSP_WARM_START_RESET (immediately before clock initialization), BSP_WARM_START_POST_CLOCK (immediately after clock initialization), or BSP_WARM_START_POST_C in the R_BSP_WarmStart function
3	hal_entry.c	In the hal_entry function (after completion of startup)

Note: If the FSP version is changed and [Generate Project Content] is executed after processing is implemented in this file, the code for the processing will be lost. Implement the processing again in such cases.

4.2.2.2 Arithmetic Unit for Trigonometric Functions

When this processing is necessary, refer to the chapter describing the arithmetic unit for trigonometric functions in the User's Manual: Hardware of the target device, compiler manual, and source code of the BSP module of FIT and implement user-specified processing at an appropriate location such as one of those listed in the following examples of locations for implementation.

Table 4.9 Examples of Locations for Implementing Processing Related to the Arithmetic Unit for Trigonometric Functions

No.	File Name	Location for Implementation
1	system.c	In the SystemInit function ^(Note)
2	hal_warmstart.c	In a conditional branch with BSP_WARM_START_RESET (immediately before clock initialization), BSP_WARM_START_POST_CLOCK (immediately after clock initialization), or BSP_WARM_START_POST_C in the R_BSP_WarmStart function
3	hal_entry.c	In the hal_entry function (after completion of startup)

Note: If the FSP version is changed and [Generate Project Content] is executed after processing is implemented in this file, the code for the processing will be lost. Implement the processing again in such cases.

4.2.2.3 Waiting for Stabilization of the VBATT Voltage

When this processing is necessary, refer to the chapter describing the battery backup function in the User's Manual: Hardware of the target device and the source code of the BSP module of FIT, and implement user-specified processing at an appropriate location, such as one of those listed in the following examples of locations for implementation.

Table 4.10 Examples of Locations for Implementing Processing to Wait for Stabilization of the VBATT Voltage

No.	File Name	Location for Implementation
1	system.c	Before execution of the bsp_clock_init function (clock initialization function) in the SystemInit function ^(Note)
2	hal_warmstart.c	In a conditional branch with BSP_WARM_START_RESET (immediately before clock initialization) in the R_BSP_WarmStart function

Note: If the FSP version is changed and [Generate Project Content] is executed after processing is implemented in this file, the code for the processing will be lost. Implement the processing again in such cases.

4.2.2.4 Stopping the Clocks Not to be Used as Clock Sources

When this processing is necessary, refer to the chapter describing the clock generation circuit in the User's Manual: Hardware of the target device and the source code of the BSP module of FIT and implement user-specified processing at an appropriate location such as one of those listed in the following examples of locations for implementation.

Table 4.11 Examples of Locations for Implementing Processing to Stop Oscillation

No.	File Name	Location for Implementation
1	bsp_clocks.c	In the bsp_clock_init function (clock initialization function) ^(Note)
2	hal_warmstart.c	In a conditional branch with BSP_WARM_START_POST_CLOCK (immediately after clock initialization) or BSP_WARM_START_POST_C in the R_BSP_WarmStart function
3	hal_entry.c	In the hal_entry function (after completion of startup)

Note: If the FSP version is changed and [Generate Project Content] is executed after processing is implemented in this file, the code for the processing will be lost. Implement the processing again in such cases.

4.2.2.5 Initializing the STDIO Library

When this processing is necessary, refer to the manual of the compiler used and the source code of the BSP module of FIT and implement user-specified processing at an appropriate location such as one of those listed in the following examples of locations for implementation.

Table 4.12 Examples of Locations for Implementing Processing to Initialize the STDIO Library

No.	File Name	Location for Implementation
1	hal_warmstart.c	In a conditional branch with BSP_WARM_START_POST_C in the R_BSP_WarmStart function
2	hal_entry.c	In the hal_entry function (after completion of startup)

4.2.2.6 Initializing Bus Priority

When this processing is necessary, refer to the chapter describing the buses in the User's Manual: Hardware of the target device and the source code of the BSP module of FIT and implement user-specified processing at an appropriate location such as one of those listed in the following examples of locations for implementation.

Table 4.13 Examples of Locations for Implementing Processing to Initialize Bus Priority

No.	File Name	Location for Implementation
1	hal_warmstart.c	In a conditional branch with BSP_WARM_START_POST_C in the R_BSP_WarmStart function
2	hal_entry.c	In the hal_entry function (after completion of startup)

4.2.2.7 Enabling or Disabling the Use of the User Stack

When this processing is necessary, refer to the chapter describing the CPU in the User's Manual: Hardware of the target device, compiler manual, and source code of the BSP module of FIT, and implement user-specified processing at an appropriate location such as one of those listed in the following examples of locations for implementation.

Table 4.14 Examples of Locations for Implementing Processing to Enable or Disable the Use of the User Stack

No.	File Name	Location for Implementation
1	hal_warmstart.c	In a conditional branch with BSP_WARM_START_POST_C in the R_BSP_WarmStart function
2	hal_entry.c	In the hal_entry function (after completion of startup)

4.2.2.8 Switching to User Mode

When this processing is necessary, refer to the chapter describing the CPU in the User's Manual: Hardware of the target device, compiler manual, and source code of the BSP module of FIT, and implement user-specified processing at an appropriate location such as the one shown in the following example of implementation location.

Table4.15 Example of Location for Implementing Processing to Switch to User Mode

No.	File Name	Location for Implementation
1	hal_entry.c	In the hal_entry function (after completion of startup)

4.2.2.9 Closing the STDIO Library

When this processing is necessary, refer to the manual of the compiler used and the source code of the BSP module of FIT and implement user-specified processing at an appropriate location such as the one shown in the following example of implementation location.

Table 4.16 Example of Location for Implementing Processing to Close the STDIO Library

No.	File Name	Location for Implementation
1	hal_entry.c	In the hal_entry function (after completion of startup)

4.2.3 Initializing the BSP in the FSP

The BSP in the FSP sets the reset vector of the MCU to indicate the Reset_Handler function. The Reset_Handler function (startup function) executes various types of initialization processing to prepare the MCU for executing the user application. The following flowcharts show the operation of the startup function and system clock settings. the code is created based on the processing by RX FSP v1.0.

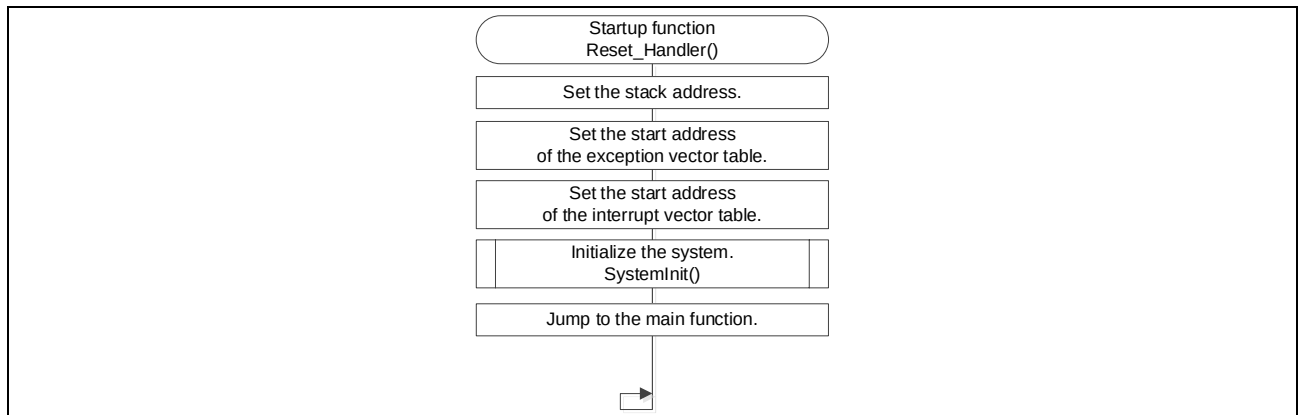


Figure 4-1 Flowchart of Startup Function

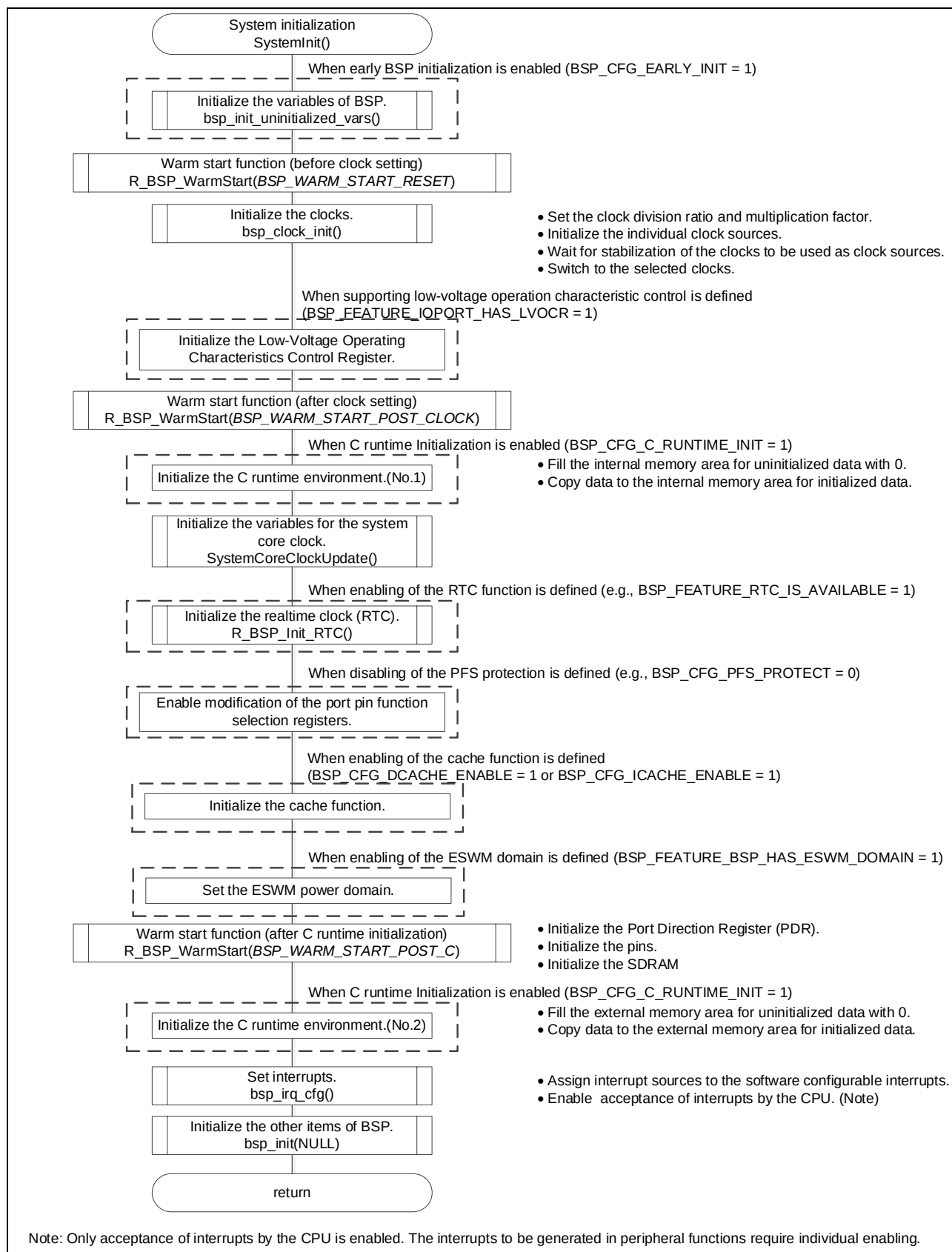


Figure 4-2 Flowchart of System Initialization

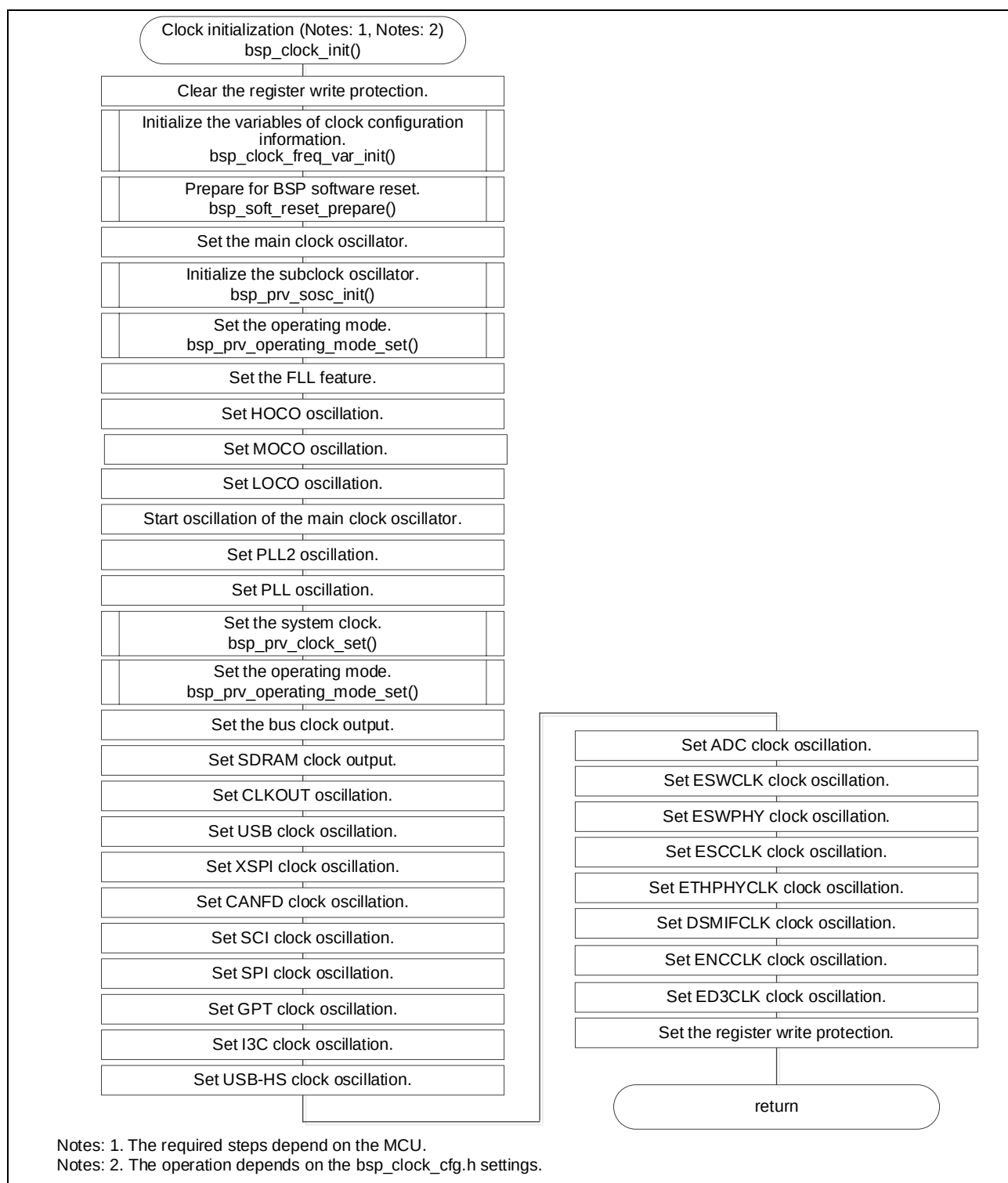


Figure 4-3 Flowchart of Clock Initialization

4.3 Comparison of the Individual Module Specifications

The API functions of FIT modules, Code Generator components, and FSP modules are not compatible with each other.

The following describes the API specifications of some modules.

4.3.1 Compare Match Timer (CMT)

4.3.1.1 CMT Module of the FSP

Table 4.17 API Functions for the CMT Module of the FSP

No.	Function Definition	Functional Overview
1	fsp_err_t R_CMT_Open(timer_ctrl_t * const p_ctrl, timer_cfg_t const * const p_cfg)	Initializes the CMT timer module.
		Pointer to the control information
		Pointer to the configuration information
2	fsp_err_t R_CMT_Stop(timer_ctrl_t * const p_ctrl)	Stops the timer.
		Pointer to the control information
3	fsp_err_t R_CMT_Start(timer_ctrl_t * const p_ctrl)	Starts the timer.
		Pointer to the control information
4	fsp_err_t R_CMT_Reset(timer_ctrl_t * const p_ctrl)	Resets the counter value to 0.
		Pointer to the control information
5	fsp_err_t R_CMT_PeriodSet(timer_ctrl_t * const p_ctrl, uint32_t const period_counts)	Updates the period. The new period is immediately reflected and the counter is reset to 0.
		Pointer to the control information
		Value to set for counting to control the period
6	fsp_err_t R_CMT_InfoGet(timer_ctrl_t * const p_ctrl, timer_info_t * const p_info)	Gets timer information and stores it in the variable specified by the p_info pointer.
		Pointer to the control information
		Pointer to the variable for storing the acquired information
7	fsp_err_t R_CMT_StatusGet(timer_ctrl_t * const p_ctrl, timer_status_t * const p_status)	Gets the current state of the timer and stores it in the variable specified by the p_status pointer.
		Pointer to the control information
		Pointer to the variable for storing the acquired state
8	fsp_err_t R_CMT_CounterSet(timer_ctrl_t * const p_ctrl, uint32_t counter)	Sets the counter value.
		Pointer to the control information
		Counter value to be set
9	fsp_err_t R_CMT_CallbackSet(timer_ctrl_t * const p_api_ctrl, void (* p_callback)(timer_callback_args_t *), void const * const p_context, timer_callback_args_t * const p_callback_memory)	Updates the user callback function by using the option for providing memory for the callback argument structure.
		Pointer to the API control information
		Pointer to the callback function
		Context to be passed to the user processing from the callback function
10	fsp_err_t R_CMT_Close(timer_ctrl_t * const p_ctrl)	Stops the counter and clears the internal driver data.
		Pointer to the control information

Note: APIs not supported in RX FSP v1.0.0 are omitted.

4.3.1.2 Comparison of the CMT Modules of FIT and the FSP

The following shows the correspondence between the API functions of FIT and the FSP for the CMT modules. The API functions listed in the "API of the FIT Module" column can be replaced with those listed in the "API of the FSP Module" column. For details of the API functions for the CMT module of FIT, refer to Chapter 3, API Functions, in the [CMT Module Using Firmware Integration Technology](#). For details of the API functions for the CMT module of the FSP, refer to the [RX Flexible Software Package Documentation](#) → API Reference → Modules → Timers → [Compare Match Timer \(r_cmt\)](#).

Table 4.18 Correspondence of the Functions of the CMT Modules (1)

No.	API of the FIT Module	API of the FSP Module
1	R_CMT_CreatePeriodic()	R_CMT_Open() p_cfg->mode = TIMER_MODE_PERIODIC R_CMT_Start()
2	R_CMT_CreatePeriodicAssignChannelPriority()	R_CMT_Open() p_cfg->mode = TIMER_MODE_PERIODIC p_cfg->cycle_end_ipl = (x) (x = 0 to 15) R_CMT_Start()
3	R_CMT_CreateOneShot()	R_CMT_Open() p_cfg->mode = TIMER_MODE_ONE_SHOT R_CMT_Start()
4	R_CMT_CreateOneShotAssignChannelPriority()	R_CMT_Open() p_cfg->mode = TIMER_MODE_ONE_SHOT p_cfg->cycle_end_ipl = (x) (x = 0 to 15) R_CMT_Start()
5	R_CMT_Stop()	R_CMT_Stop()
6	R_CMT_Control()	Some of the functionality is provided by the following functions.
	Command = CMT_RX_CMD_IS_CHANNEL_COUNTING	R_CMT_StatusGet()
	Command = CMT_RX_CMD_PAUSE	R_CMT_Stop()
	Command = CMT_RX_CMD_RESUME	R_CMT_Start()
	Command = CMT_RX_CMD_RESTART	R_CMT_Start()
6	Command = CMT_RX_CMD_GET_NUM_CHANNELS	No corresponding function ^(Note)
	Command = CMT_RX_CMD_SET_PRIORITY	R_CMT_Open() p_cfg->cycle_end_ipl = (x) (x = 0 to 15)
	Command = CMT_RX_CMD_GET_PRIORITY	No corresponding function ^(Note)
7	R_CMT_GetVersion()	Version numbers are not assigned to the individual modules. The following function can be used to check the version of the FSP. R_FSP_VersionGet()

Note: If a similar function is necessary, refer to the source code of the FIT module and implement user-specified processing.

Table 4.19 Correspondence of the Functions of the CMT Modules (2)

No.	Interrupt and Callback Functions of the FIT Module	Interrupt and Callback Functions of the FSP Module
1	cmtn_isr() (n = channel number)	cmt_cm_int_isr()
2	Specify a user-defined callback function as the second argument "callback" of the following API functions. R_CMT_CreatePeriodic() R_CMT_CreatePeriodicAssignChannelPriority() R_CMT_CreateOneShot() R_CMT_CreateOneShotAssignChannelPriority()	R_CMT_Open() Specify a user-defined function as p_cfg->p_callback.

4.3.1.3 Comparison of the API Functions for the CMT of the Code Generator and those of the FSP Module

The following shows the correspondence between the API functions of the Code Generator and FSP for the CMT. The API functions listed in the "API for the CMT of the Code Generator" column can be replaced with those listed in the "API of the FSP Module" column. For details of the API functions for the CMT of the Code Generator, see section 4.2.6, Compare Match Timer, in the [Smart Configurator User's Manual: RX API Reference](#). For details of the API functions for the CMT module of the FSP, refer to the [RX Flexible Software Package Documentation](#) → API Reference → Modules → Timers → [Compare Match Timer \(r_cmt\)](#).

Table 4.20 Correspondence with the API Functions for the CMT of the Code Generator

No.	API for the CMT of the Code Generator	API of the FSP Module
1	R_<Config_CMT0>_Create()	R_CMT_Open()
2	R_<Config_CMT0>_Start()	R_CMT_Start()
3	R_<Config_CMT0>_Stop()	R_CMT_Stop()
4	R_<Config_CMT0>_UserInit()	No corresponding function (Note)

Note: Many API functions provided by the Smart Configurator do not have arguments because variables for them cannot be specified through the GUI. Therefore, the UserInit function for initialization is provided to specify or modify variables in the user program. The FSP does not support such usage and has no equivalent function.

Table 4.21 Correspondence with the Interrupt Function for the CMT of the Code Generator

No.	Interrupt Function for the CMT of the Code Generator	Interrupt Function of the FSP Module
1	r_<Config_CMT0>_cmin_interrupt User-defined processing in the function	cmt_cm_int_isr() Replace the processing with a callback function. R_CMT_Open() Specify a user-defined function as p_cfg->p_callback.

4.3.2 Serial Communications Interface (RSCI)

4.3.2.1 SCI UART Module of the FSP

Table 4.22 API Functions for the SCI UART Module of the FSP (1)

No.	Function Definition	Functional Overview
1	fsp_err_t R_SCI_B_UART_Open(uart_ctrl_t * const p_api_ctrl, uart_cfg_t const * const p_cfg)	Configures the UART driver based on the input settings. Reception is enabled at the end of this function if it is set to be enabled at the time of compilation.
		Pointer to the API control information
		Pointer to the configuration information
2	fsp_err_t R_SCI_B_UART_Read(uart_ctrl_t * const p_api_ctrl, uint8_t * const p_dest, uint32_t const bytes)	Receives the specified number of bytes and stores them in the reception buffer.
		Pointer to the API control information
		Pointer to the reception buffer
3	fsp_err_t R_SCI_B_UART_Write(uart_ctrl_t * const p_api_ctrl, uint8_t const * const p_src, uint32_t const bytes)	Transmits the specified number of bytes from the transmission buffer.
		Pointer to the API control information
		Pointer to the transmission buffer
4	fsp_err_t R_SCI_B_UART_BaudSet(uart_ctrl_t * const p_api_ctrl, void const * const p_baud_setting)	Updates the bit-rate by using the clock selected in the Open function.
		Pointer to the API control information
		Pointer to the bit-rate configuration information
5	fsp_err_t R_SCI_B_UART_InfoGet(uart_ctrl_t * const p_api_ctrl, uart_info_t * const p_info)	Gets the driver information such as the maximum number of bytes that can be received or transmitted at a time.
		Pointer to the API control information
		Pointer to the variable for storing the acquired information
6	fsp_err_t R_SCI_B_UART_Close(uart_ctrl_t * const p_api_ctrl)	Aborts all communications in progress, disables the interrupts, the receiver, and the transmitter, closes the lower-level transfer drivers that are in use, and turns the power to the peripheral functions off.
		Pointer to the API control information
7	fsp_err_t R_SCI_B_UART_Abort(uart_ctrl_t * const p_api_ctrl, uart_dir_t communication_to_abort)	Aborts the transfer in progress. In the case of transmission, the processing is aborted after transmission of the current character.
		Pointer to the API control information
		Variable for specifying the direction of transfer to be aborted
8	fsp_err_t R_SCI_B_UART_BaudCalculate(uint32_t baudrate, bool bitrate_modulation, uint32_t baud_rate_error_x_1000, sci_b_baud_setting_t * const p_baud_setting)	Calculates the values to be set in the registers related to bit-rate settings. Appropriate values to be set in the registers are evaluated and determined.
		Bit rate (bps)
		Setting for enabling or disabling bit-rate modulation
		Maximum bit-rate error
		Pointer to the location for storing the calculated bit-rate configuration information

Table 4.23 API Functions for the SCI UART Module of the FSP (2)

No.	Function Definition	Functional Overview
9	fsp_err_t R_SCI_B_UART_CallbackSet(uart_ctrl_t * const p_api_ctrl, void (* p_callback)(uart_callback_args_t *), void const * const p_context, uart_callback_args_t * const p_callback_memory)	Updates the user callback function by using the option for providing memory for the callback argument structure.
		Pointer to the API control information
		Pointer to the callback function
		Context to be passed to the user processing from the callback function
10	fsp_err_t R_SCI_B_UART_ReadStop(uart_ctrl_t * const p_api_ctrl, uint32_t * remaining_bytes)	Aborts the reading in progress.
		Pointer to the API control information
		Number of remaining bytes

Note: APIs not supported in RX FSP v1.0.0 are omitted.

4.3.2.2 Comparison of the RSCI Module of FIT and the UART Module of the FSP

The following shows the correspondence of API functions between the RSCI module of FIT and the SCI UART module of the FSP. The API functions listed in the "API of the FIT Module" column can be replaced with those listed in the "API of the FSP Module" column. For details of the API functions for the RSCI module of FIT, refer to Chapter 3, API Functions, in the [RSCI Module Using Firmware Integration Technology](#). For details of the API functions for the UART module of the FSP, refer to the [RX Flexible Software Package Documentation](#) → API Reference → Modules → Connectivity → [UART \(r_sci_b_uart\)](#).

Table 4.24 Correspondence with the Functions of the RSCI Module (1)

No.	API of the FIT Module	API of the FSP Module
1	R_RSCI_Open()	R_SCI_B_UART_Open()
2	R_RSCI_Close()	R_SCI_B_UART_Close()
3	R_RSCI_Send()	R_SCI_B_UART_Write()
4	R_RSCI_Receive()	R_SCI_B_UART_Read()
5	R_RSCI_SendReceive()	No corresponding function (Note)
6	R_RSCI_Control()	Some of the functionality is provided by the following functions.
	Cmd = SCI_CMD_CHANGE_BAUD	R_SCI_B_UART_BaudSet()
	Cmd = SCI_CMD_CHANGE_TX_FIFO_THRESH	No corresponding function (Note)
	Cmd = SCI_CMD_CHANGE_RX_FIFO_THRESH	No corresponding function (Note)
	Cmd = SCI_CMD_SET_RXI_PRIORITY	R_SCI_B_UART_Open() p_cfg->rxi_ipl = (x) (x = 0 to 15)
	Cmd = SCI_CMD_SET_TXI_PRIORITY	R_SCI_B_UART_Open() p_cfg->txi_ipl = (x) (x = 0 to 15)
	Cmd = SCI_CMD_EN_NOISE_CANCEL	No corresponding function (Note)
	Cmd = CMT_RX_CMD_GET_PRIORITY	No corresponding function (Note)
	Cmd = SCI_CMD_OUTPUT_BAUD_CLK	No corresponding function (Note)
	Cmd = SCI_CMD_START_BIT_EDGE	No corresponding function (Note)
	Cmd = SCI_CMD_GENERATE_BREAK	No corresponding function (Note)
	Cmd = SCI_CMD_COMPARE_RECEIVED_DATA	No corresponding function (Note)
	Cmd = SCI_CMD_TX_Q_FLUSH	No corresponding function (Note)
	Cmd = SCI_CMD_RX_Q_FLUSH	No corresponding function (Note)
	Cmd = SCI_CMD_TX_Q_BYTES_FREE	No corresponding function (Note)

Note: If a similar function is necessary, refer to the source code of the FIT module and implement user-specified processing.

Table 4.25 Correspondence with the Functions of the RSCI Module (2)

No.	API of the FIT Module	API of the FSP Module
6	Cmd = SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ	No corresponding function (Note)
	Cmd = SCI_CMD_EN_CTS_IN	No corresponding function (Note)
	Cmd = SCI_CMD_CHECK_XFER_DONE	No corresponding function (Note)
	Cmd = SCI_CMD_ABORT_XFER	No corresponding function (Note)
	Cmd = SCI_CMD_XFER_LSB_FIRST	No corresponding function (Note)
	Cmd = SCI_CMD_XFER_MSB_FIRST	No corresponding function (Note)
	Cmd = SCI_CMD_INVERT_DATA	No corresponding function (Note)
	Cmd = SCI_CMD_CHANGE_SPI_MODE	No corresponding function (Note)
	Cmd = SCI_CMD_CHECK_TX_DONE	No corresponding function (Note)
	Cmd = SCI_CMD_CHECK_RX_DONE	No corresponding function (Note)
	Cmd = SCI_CMD_CHECK_RX_SYNC_DONE	No corresponding function (Note)
7	R_RSCI_GetVersion()	Version numbers are not assigned to the individual modules. The following function can be used to check the version of the FSP. R_FSP_VersionGet()

Note: If a similar function is necessary, refer to the source code of the FIT module and implement user-specified processing.

Table 4.26 Correspondence with the Functions of the RSCI Module (3)

No.	Interrupt and Callback Functions of the FIT Module	Interrupt and Callback Functions of the FSP Module
1	rscin_txi_isr() (n = channel number)	sci_b_uart_txi_isr()
2	rscin_tei_isr() (n = channel number)	sci_b_uart_tei_isr()
3	rscin_rxi_isr() (n = channel number)	sci_b_uart_rxi_isr()
4	rscin_eri_isr() (n = channel number)	sci_b_uart_eri_isr()
5	Specify a user-defined callback function as the fourth argument "p_callback" of the following API function. R_RSCI_Open()	Specify a user-defined callback function in the following API function. R_SCI_B_UART_Open() Alternatively, specify it as the second argument "p_callback" of the following API function. R_SCI_B_UART_CallbackSet()

4.3.2.3 Comparison of the API Functions for the SCI/SCIF Asynchronous Mode of the Code Generator and those for the UART Module of the FSP

The following shows the correspondence between the API functions for the SCI/SCIF asynchronous mode of the Code Generator and those for the SCI UART module of the FSP. The API functions listed in the "API for the SCI/SCIF Asynchronous Mode of the Code Generator" column can be replaced with those listed in the "API of the FSP Module" column. For details of the API functions for the SCI/SCIF asynchronous mode of the Code Generator, see section 4.2.31, SCI/SCIF Asynchronous Mode, in the [Smart Configurator User's Manual: RX API Reference](#). For details of the API functions for the UART module of the FSP, refer to the [RX Flexible Software Package Documentation](#) → API Reference → Modules → Connectivity → [UART \(r_sci_b_uart\)](#).

Table 4.27 Correspondence with the API Functions for the SCI/SCIF Asynchronous Mode of the Code Generator

No.	API for the SCI/SCIF Asynchronous Mode of the Code Generator	API of the FSP Module
1	R_<Config_SCI0>_Create()	R_SCI_B_UART_Open()
2	R_<Config_SCI0>_Start()	R_SCI_B_UART_Open()
3	R_<Config_SCI0>_Stop()	R_SCI_B_UART_Close()
4	R_<Config_SCI0>_Serial_Send()	R_SCI_B_UART_Write()
5	R_<Config_SCI0>_Serial_Receive()	R_SCI_B_UART_Read()
6	R_<Config_SCI0>_Serial_Multiprocessor_Send()	No corresponding function
7	R_<Config_SCI0>_Serial_Multiprocessor_Receive()	No corresponding function
8	R_<Config_SCI0>_Create_UserInit()	No corresponding function (Note)

Note: Many API functions provided by the Smart Configurator do not have arguments because variables for them cannot be specified through the GUI. Therefore, the UserInit function for initialization is provided to specify or modify variables in the user program. The FSP does not support such usage and has no equivalent function.

Table 4.28 Correspondence with the Interrupt and Callback Functions for the SCI/SCIF Asynchronous Mode of the Code Generator

No.	Interrupt and Callback Functions for the SCI/SCIF Asynchronous Mode of the Code Generator	Interrupt and Callback Functions of the FSP Module
1	r_<Config_SCI0>_transmitend_interrupt()	sci_b_uart_tei_isr()
2	r_<Config_SCI0>_transmit_interrupt()	sci_b_uart_txi_isr()
3	r_<Config_SCI0>_receive_interrupt()	sci_b_uart_rxi_isr()
4	r_<Config_SCI0>_receiveerror_interrupt()	sci_b_uart_eri_isr()
5	r_<Config_SCI0>_callback_transmitend() User-defined processing in the function	Replace the function with a user-defined callback function. Specify a user-defined callback function in the following API function. R_SCI_B_UART_Open() Alternatively, specify it as the second argument "p_callback" of the following API function. R_SCI_B_UART_CallbackSet()
6	r_<Config_SCI0>_callback_receiveend() User-defined processing in the function	Replace the function with a user-defined callback function. Specify a user-defined callback function in the following API function. R_SCI_B_UART_Open() Alternatively, specify it as the second argument "p_callback" of the following API function. R_SCI_B_UART_CallbackSet()
7	r_<Config_SCI0>_callback_receiveerror() User-defined processing in the function	Replace the function with a user-defined callback function. Specify a user-defined callback function in the following API function. R_SCI_B_UART_Open() Alternatively, specify it as the second argument "p_callback" of the following API function. R_SCI_B_UART_CallbackSet()

5. Steps in Porting a Pre-FSP Project to an FSP Project

Use the steps listed in the table below to port a project using FIT modules (hereafter called an FIT project) or a project using the Code Generator (hereafter called a code-generator project) to an FSP project.

Table 5.1 Steps in Porting an FIT Project or a Code-Generator Project to an FSP Project

No.	Step	Condition	Reference Section
1	Create a new FSP project.		5.1
2	Make basic settings for the FSP project.		5.2
3	Configure the clocks for the FSP project.		5.3
4	Add FSP modules.		5.4
5	Configure the interrupts for the FSP project.	Interrupts are to be used by peripheral functions for which FSP modules are not used	5.5
6	Add the processing for pin settings by using the IOPORT module.		5.6
7	Port the required sections of the source code.		5.7
8	Add iodef.h to the FSP project.	Porting includes processing for access to I/O registers.	5.8
9	Replace the processing by the FIT modules or the processing created by the Code Generator with processing by FSP modules.		5.9

The steps involved in porting a project created without using the Smart Configurator to an FSP project are listed below.

Table 5.2 Steps in Porting a Project Created without Using the Smart Configurator to an FSP Project

No.	Step	Condition	Reference Section
1	Create a new FSP project.		5.1
2	Make basic settings for the FSP project.		5.2
3	Configure the clocks for the FSP project.		5.3
4	Configure the interrupts for the FSP project.	Interrupts are to be used.	5.5
5	Port the required sections of the source code.		5.7
6	Add the processing for pin settings by using the IOPORT module.	Porting does not include processing for pin settings.	5.6
7	Add iodef.h to the FSP project.	Porting includes processing for access to I/O registers.	5.8

5.1 Creating a New FSP Project

See section 2.3, Procedure for Creating an FSP Project, and create a new FSP project.

5.2 Porting the Basic Settings

A BSP is automatically included in an FSP project when the project is created. The BSP contains the startup processing. For details of the startup processing, see section 4.2.3, Initializing the BSP in the FSP.

The startup processing of the BSP executed after release from the reset state makes basic settings such as for the stack configuration, vector table configuration, clock configuration, and interrupt configuration. Therefore, the items under "RX Common" in the [Properties] view on the [BSP] page require selection to suit the target board. Modify the configurations as required to suit the conditions of usage of the source project to be ported.

The following shows the properties displayed on the [BSP] page of the GUI.

Summary **BSP** Clocks Pins Interrupts Event Links Linker Sections Stacks Components

Properties Problems Smart Browser

EK-RX74M

Settings

Property	Value
▼ RX Common	
Main stack size (bytes)	0x400
Heap size (bytes)	0
Protect zone root stack size (bytes)	0x800
MCU Vcc (mV)	3300
Parameter checking	Disabled
Assert Failures	Return FSP_ERR_ASSERTION
Clock Registers not Reset Values during Startup	Disabled
Main Oscillator Populated	Populated
PFS Protect	Enabled

Figure 5-1 BSP Properties

The information selected for the GUI items under "RX Common" will be reflected in the macro definitions in bsp_cfg.h, which is to be generated in the rx_cfg folder after generation of the project code (after execution of [Generate Project Content]).

The following shows the contents of bsp_cfg.h after generation with the above configuration.

Project Explorer ×	bsp_cfg.h ×
FSP_example [Debug] Includes rx rx_gen src Debug rx_cfg fsp_cfg bsp board_cfg.h bsp_cfg.h bsp_mcu_device_cfg.h	<pre> 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 30 #if defined(_RX_BOOT_IMAGE) 31 #define BSP_CFG_BOOT_IMAGE (1) 32 #endif 33 #define BSP_CFG_MCU_VCC_MV (3300) 34 #define BSP_CFG_STACK_MAIN_BYTES (0x400) 35 #define BSP_CFG_HEAP_BYTES (0) 36 #define BSP_CFG_PZ_ROOT_STACK_SIZE (0x800) 37 #define BSP_CFG_PARAM_CHECKING_ENABLE (0) 38 #define BSP_CFG_ASSERT (0) 39 40 #define BSP_CFG_PFS_PROTECT ((1)) 41 #define BSP_CFG_C_RUNTIME_INIT ((1)) 42 #define BSP_CFG_EARLY_INIT ((0)) 43 #define BSP_CFG_STARTUP_CLOCK_REG_NOT_RESET ((0)) </pre>

Figure 5-2 Macro Definitions in bsp_cfg.h

The following shows an example of settings for porting.

Table 5.3 Items and Values of Settings for Porting from an FIT Project or a Code-Generator Project (1)

No.	Item and Value that were Set in the BSP Component	Item and Value that Should be Set for "RX Common"
1	"User stack setting" is set to "2 stacks". (USP and ISP)	None
2	"User stack size" is set to "0x200".	None
3	"Interrupt stack size" is set to "0x200".	Set "Main stack size (byte)" to "0x200".
4	"Heap size" is set to "0x400".	Set "Heap size (byte)" to "0x400".
5	"Initializes C input and output library functions" is set to "Disabled".	None
6	"Enable user stdio charget function" is set to "Use BSP charget() function".	None
7	"Enable user stdio charput function" is set to "Use BSP charput() function".	None
8	"Processor Mode" is set to "Stay in Supervisor mode".	None
9	"ID code 1" to "ID code 4" are set to desired values.	None
10	"Serial programmer connection select" is set to "Enabled".	None
11	"ROM Cache Enable Register" is set to "Enabled".	None
12	"TMEF – TM Enable Flag Register" is set to a desired value.	None
13	"FAW – Flash Access Window Setting Register" is set to a desired value.	None
14	"ROMCODE – ROM Code Protection Register" is set to a desired value.	None
15	"Select the startup bank" is set to "Bank 1: FFC00000h~FFDFFFFFh; Bank 0: FFE00000h~FFFFFFFFh".	None
16	"Lock function select" is set to "Use default locking (non-RTOS)".	None
17	"Enable user warm start callback (PRE)" is set to "Unused".	None
18	"Enable user warm start callback (POST)" is set to "Unused".	None
19	"Parameter checking" is set to "Enabled".	Set "Parameter checking" to "Enabled".
20	"Extended Bus Master 1st Priority Selection" is set to "GLCDC graphics 1 data read".	None
21	"Extended Bus Master 2nd Priority Selection" is set to "GLCDC graphics 2 data read".	None
22	"Extended Bus Master 3rd Priority Selection" is set to "DRW2D texture data read".	None
23	"Extended Bus Master 4th Priority Selection" is set to "DRW2D frame buffer data read write and display list data read".	None
24	"Extended Bus Master 5th Priority Selection" is set to "EDMAC".	None

Table 5.4 Items and Values of Settings for Porting from an FIT Project or a Code-Generator Project (2)

No.	Item and Value that were Set in the BSP Component	Item and Value that Should be Set for "RX Common"
25	"Non-cacheable area 0 select" is set to "Disabled".	None
26	"Non-cacheable area 1 select" is set to "Disabled".	None
27	"Interrupt Priority Level When FIT Module Interrupts Are Disabled" is set to "Priority level 15 (highest)".	None
28	"Software Interrupt Unit1 (SWINT1)" is set to "Unused".	None
29	"Software Interrupt Unit2 (SWINT2)" is set to "Unused".	None
30	"Serial terminal select" is set to "Disabled".	None
31	"Select whether to enable bus priority initialization." is set to "Disabled".	None
32	"Select whether it is bootloader project." is set to "Not bootloader project".	None

Table 5.5 Items and Values of Settings for Porting from an FIT Project or a Code-Generator Project (3)

No.	Item and Value that were Set on the [Clocks] Page	Item and Value that Should be Set for "RX Common"
1	"Main clock" is selected to enable oscillation.	Set "Main Oscillator Populated" to "Populated" to enable oscillation.
2	"Resonator" is selected for the oscillation source of the main clock.	Set "Main Oscillator Clock Source" to "Crystal or Resonator".
3	"Sub-clock" is deselected to disable oscillation.	Set "Subclock Populated" to "Not Populated" to disable oscillation.
4	"Oscillator drive capacity" for "Sub-clock" is set to "Standard CL".	Set "Subclock Drive (Drive capacitance availability varies by MCU)" to "Standard/Normal mode".
5	"Oscillation wait time" for "Sub-clock" is set to "2000 ms".	Set "Subclock Stabilization Time (ms)" to "2000".
6	"IWDG-dedicated clock" is selected to enable oscillation.	None

Table 5.6 Items and Values of Settings for Porting from a Project Created without Using the Smart Configurator

No.	Specifications in the Source Project	Item and Value that Should be Set for "RX Common"
1	The uestack (USP) is set to 0x200.	None (USP cannot be used.)
2	The istack (ISP) is set to 0x200.	Set "Main stack size (byte)" to "0x200".
3	The main clock oscillator is used.	Set "Main Oscillator Populated" to "Populated" to enable oscillation.
4	The main clock oscillator is mounted on the board.	Set "Main Oscillator Clock Source" to "Crystal or Resonator".
5	The subclock oscillator is not used.	Set "Subclock Populated" to "Not Populated" to disable oscillation.

5.3 Porting the Clock Configuration

In an FSP project, the system clock setting process in the startup processing executed after release from the reset state makes settings of the registers for the clock generation circuit based on the clock configuration specified on the [Clocks Configuration] page in the same way as an FIT project or a code-generator project. For details of the BSP startup processing in the FSP, see section 4.2.3, Initializing the BSP in the FSP.

The following shows the [Clocks Configuration] page.

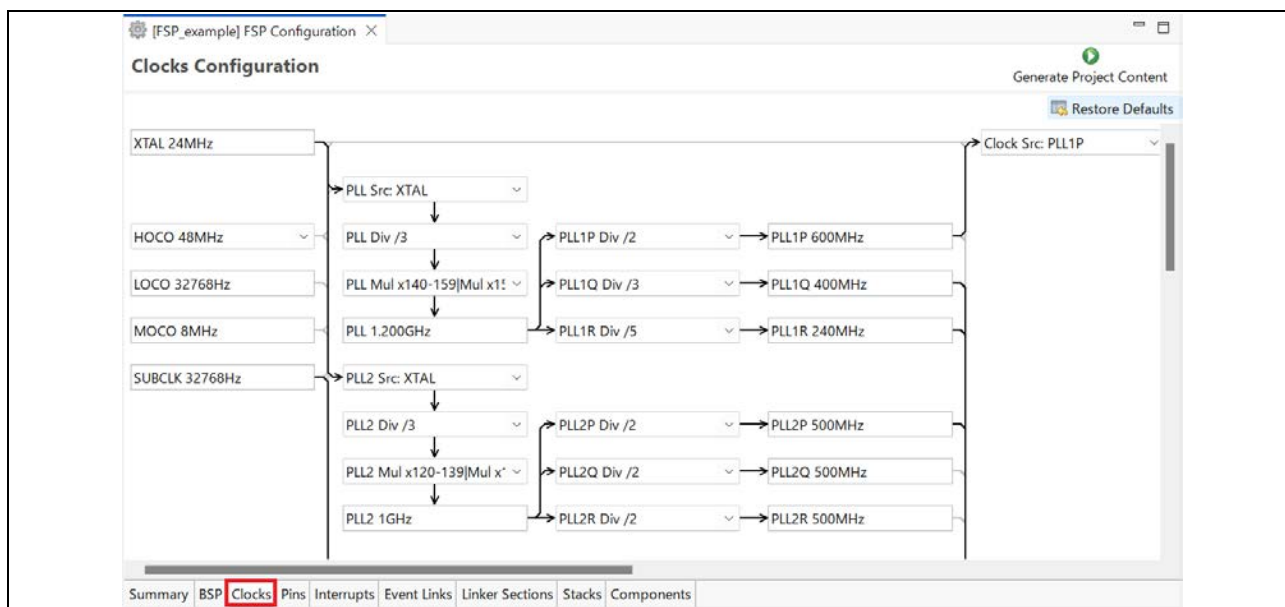


Figure 5-3 [Clocks Configuration] Page

The information selected on the [Clocks Configuration] page will be reflected in the macro definitions in bsp_clock_cfg.h to be generated in the rx_gen folder after generation of project code (after execution of [Generate Project Content]).

The following shows the contents of bsp_clock_cfg.h after generation with the above configuration.

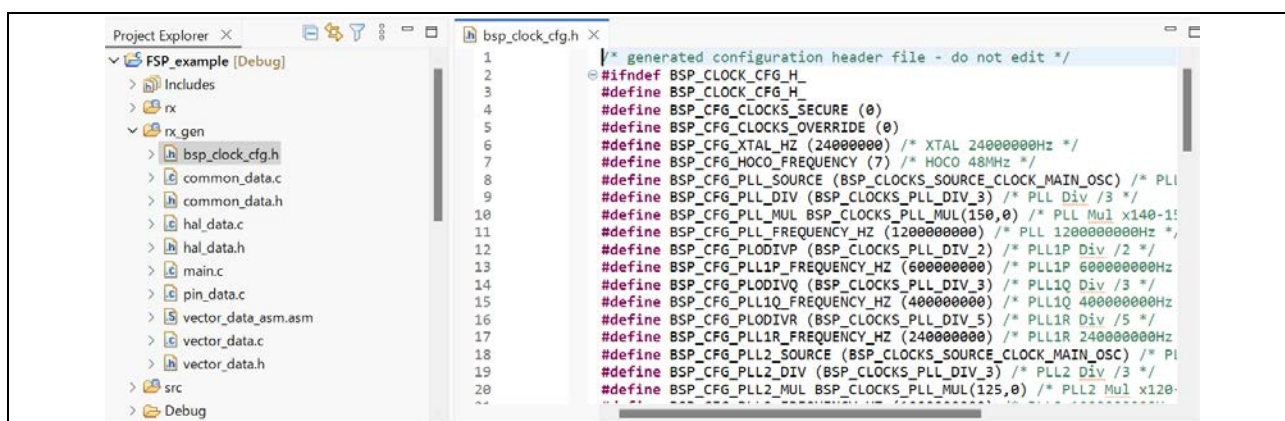


Figure 5-4 Macro Definitions in bsp_clock_cfg.h

Clocks are configured based on the macro values defined in bsp_clock_cfg.h, so the configuration on the [Clocks Configuration] page has to match the clock configuration specified on the page for clocks or the clock setting processing in the source project for porting. Do not directly port the clock setting processing of the source project because the processing will be in conflict with the clock configuration in the BSP of the FSP project.

5.4 Porting Modules

Add the modules equivalent to those used in the source project for porting to the FSP project. Stacks must be assigned to the individual modules in the FSP. Use the [Stacks Configuration] page to add modules.

The following shows the [Stacks Configuration] page. Only the IOPORT module is added in the initial state after generating a project.

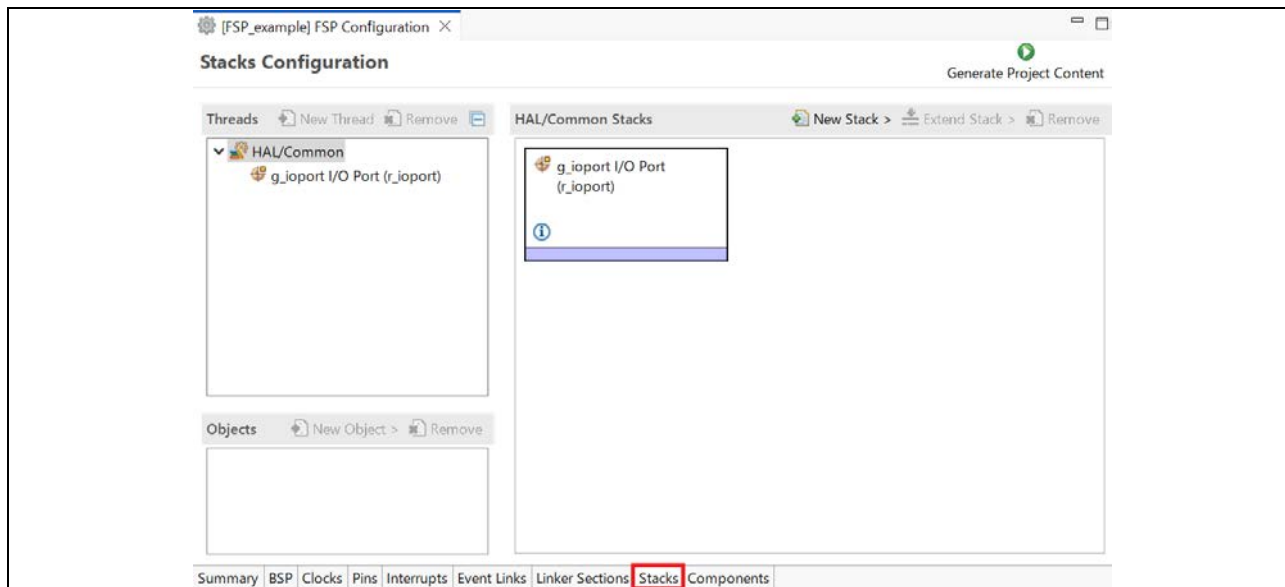


Figure 5-5 [Stacks Configuration] Page

To add a module, press the [New Stack] button and then select the desired function. From the opened list, select the module to be added to the [HAL/Common Stacks] list. For example, the CMT module can be added by selecting [New Stack] → [Timers] → [Timer, Compare Match Timer (r_cmt)].

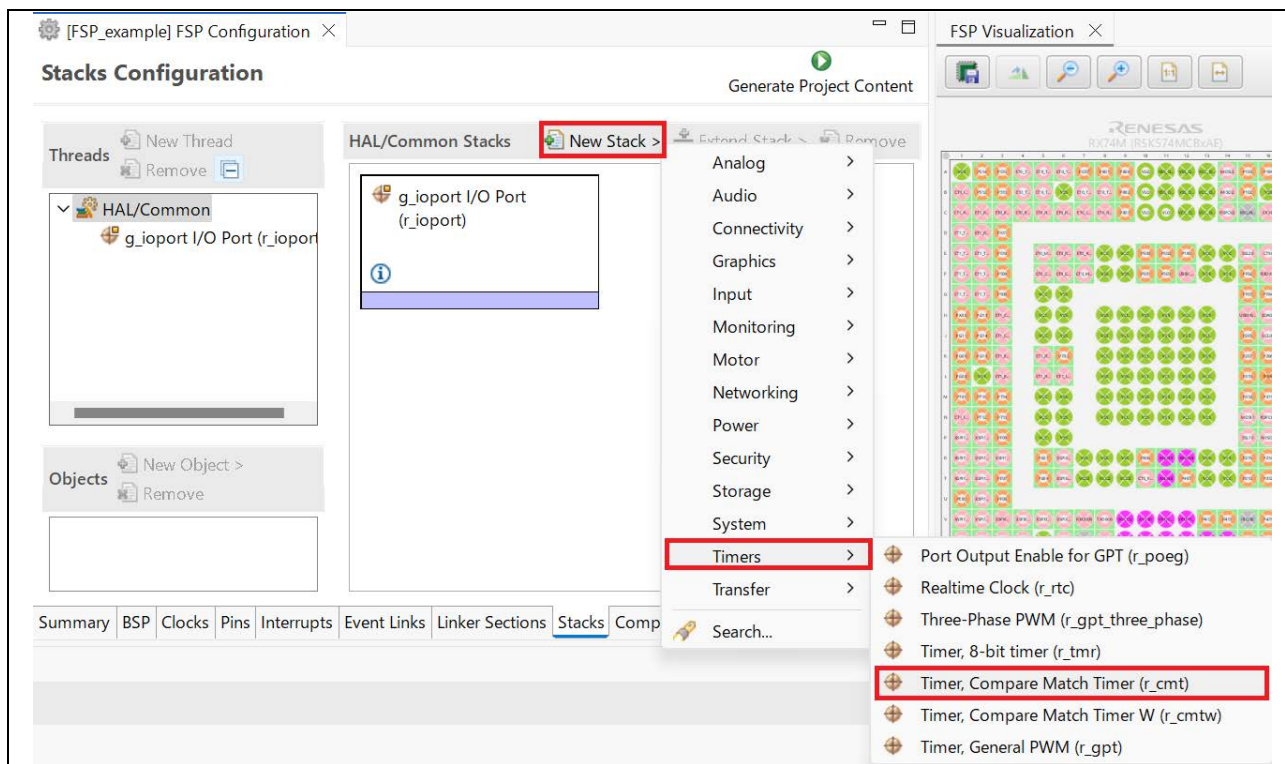


Figure 5-6 Selecting a Timer

The selected CMT module is added to [HAL/Common Stacks].

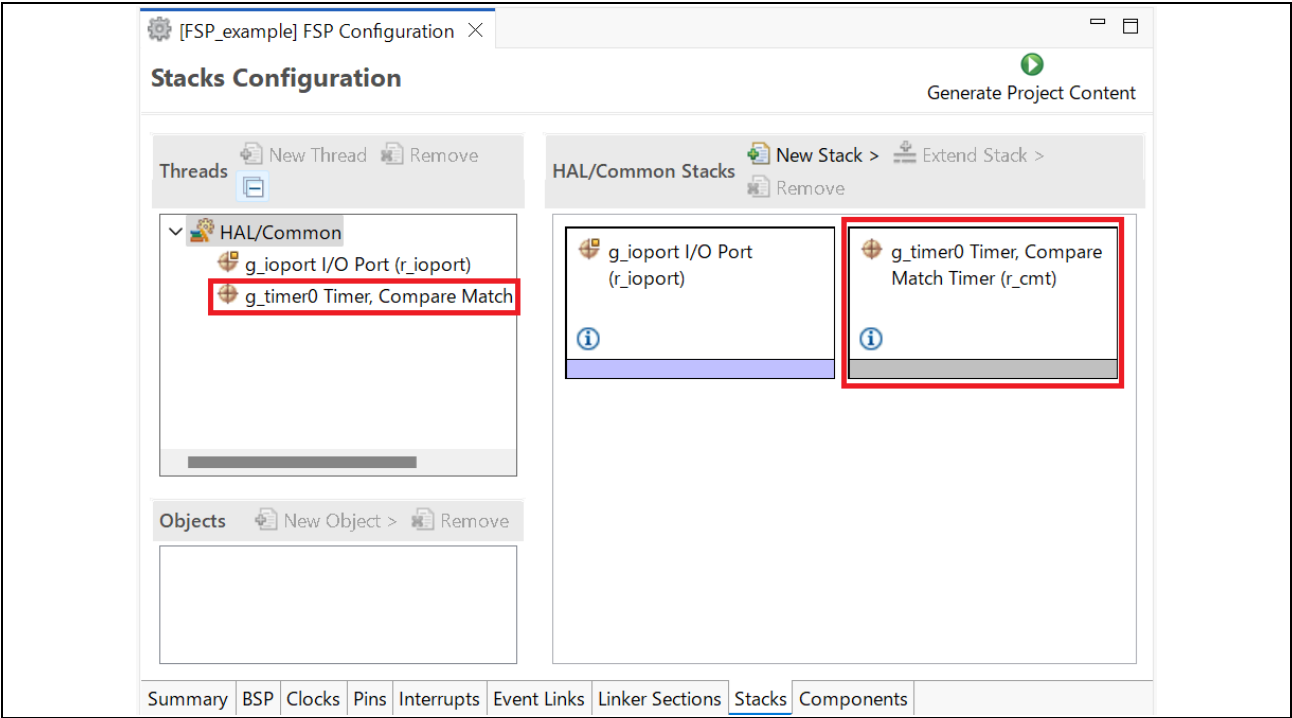


Figure 5-7 Adding the CMT Module (r_cmt)

If the module requires detailed settings, select the added module on the [Stacks Configuration] page and modify the settings in the [Properties] view.

The following shows the properties of the CMT module.

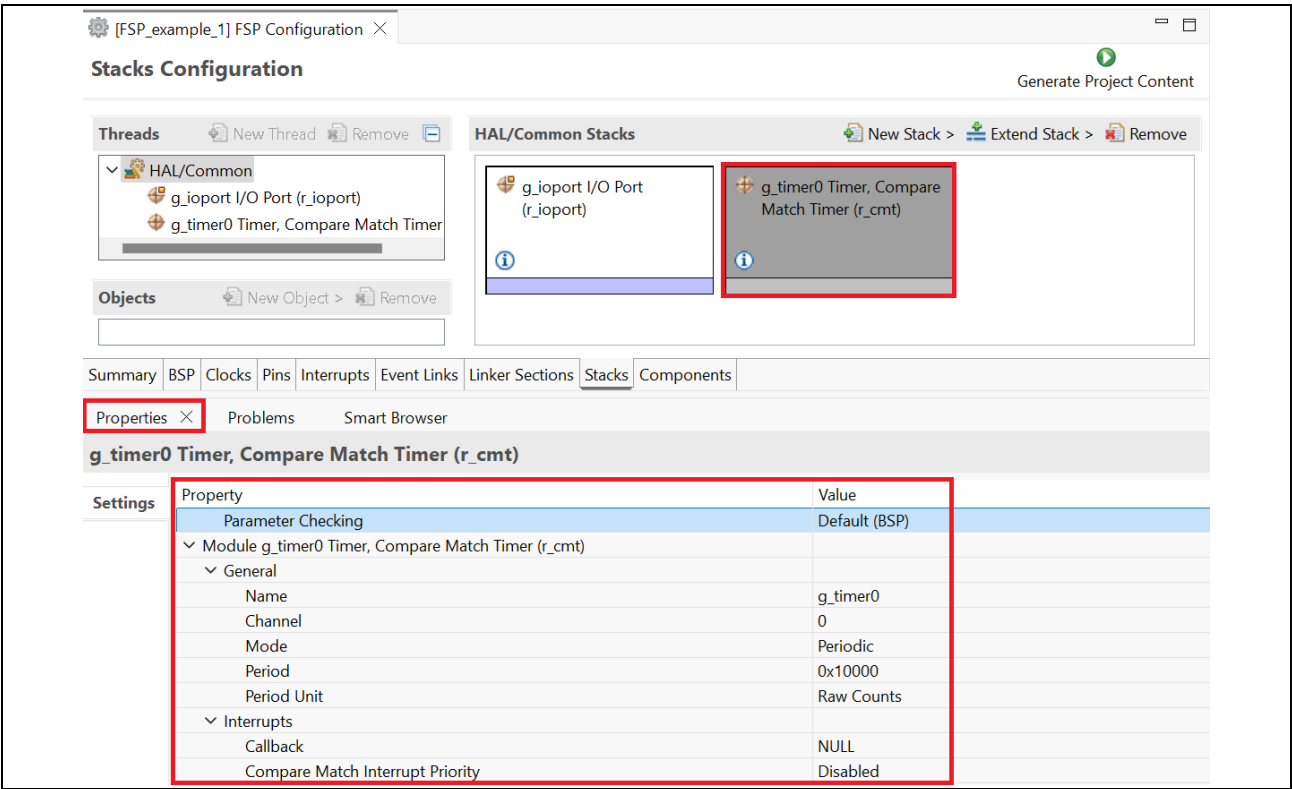


Figure 5-8 Properties of the CMT Module (r_cmt)

Click on the [Generate Project Content] button to add the module to the project.

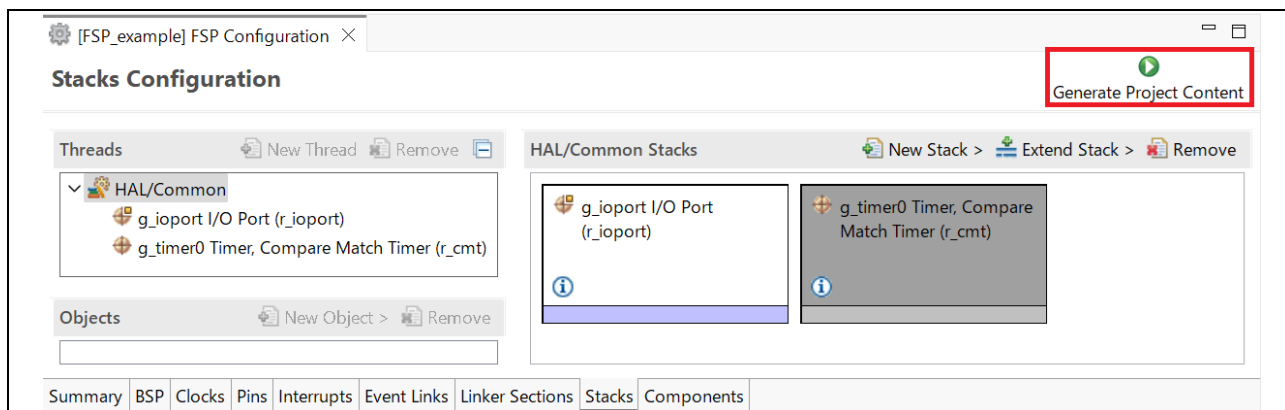


Figure 5-9 [Generate Project Content]

The definitions related to the added module will be output to hal_data.h and hal_data.c in the rx_gen folder.

The code in the red boxes in the following figures will be added when the CMT module is added without changing its properties from the initial state. If the properties are changed, the code to be added will depend on the selected module and the settings for the properties.

rx_gen\hal_data.h

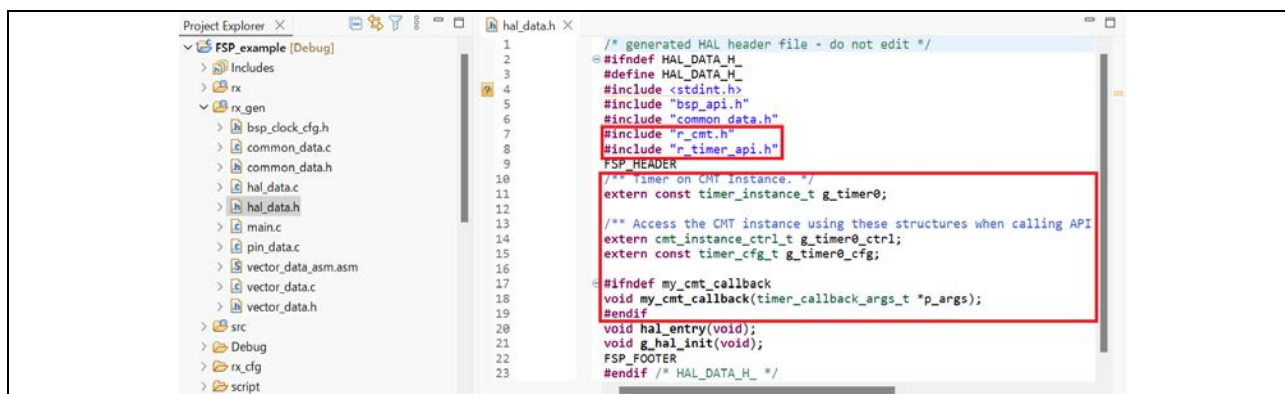


Figure 5-10 Definitions in hal_data.h

rx_gen\hal_data.c

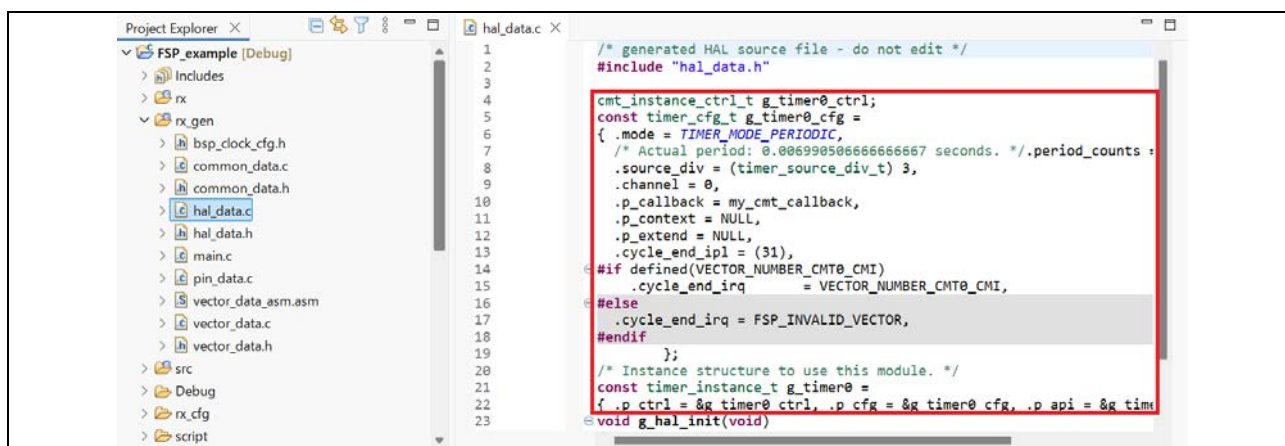


Figure 5-11 Definitions in hal_data.c

A list of components can be checked on the [Components Configuration] page.

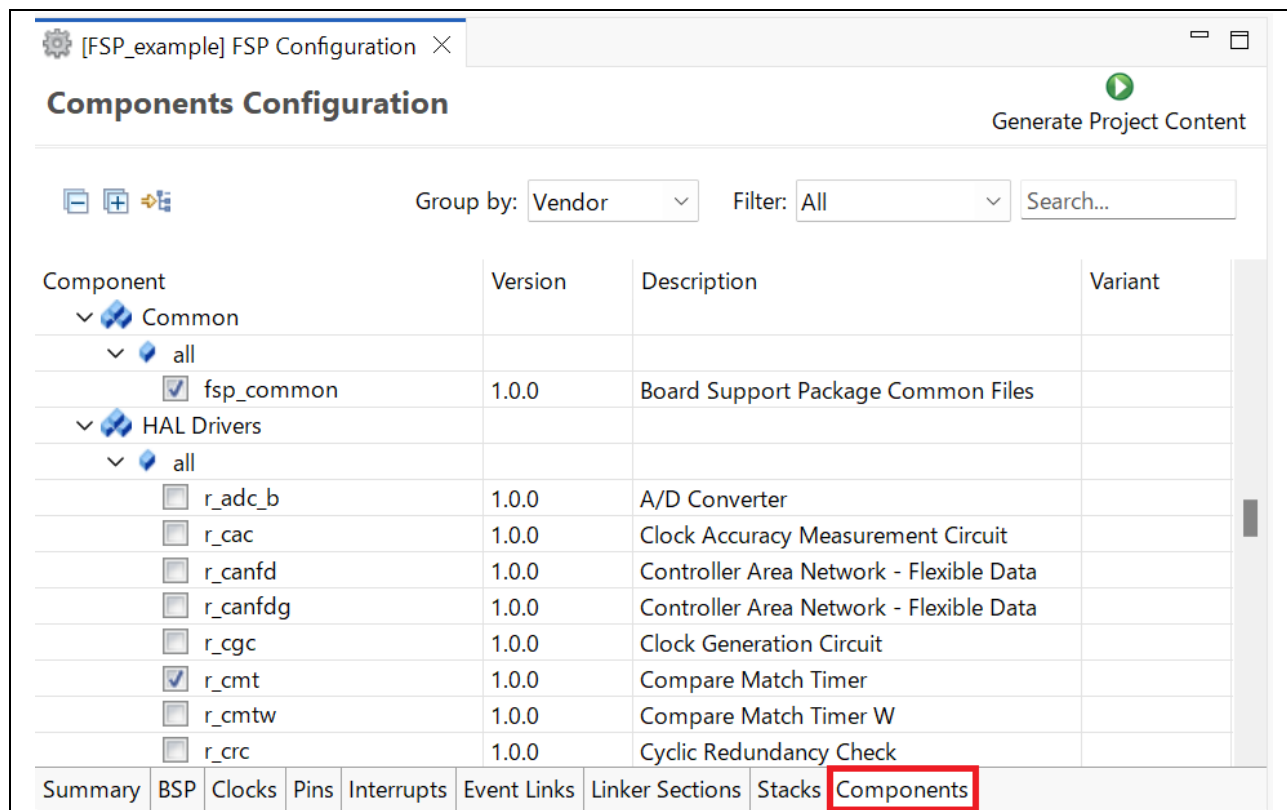


Figure 5-12 List of Components

5.5 Porting an Interrupt Configuration

In an FSP project, the BSP holds the definitions of the exception vector table and interrupt vector table. The FSP has functionality for automatically generating the declarations of interrupt functions to be registered in the vector table according to the interrupt configuration. Generation of the interrupt functions depends on the method of interrupt configuration. For details, see section 5.5.1, Interrupt Configuration by Adding a Module, and section 5.5.2, Interrupt Configuration through Direct Addition.

The following shows the [Interrupts Configuration] page.

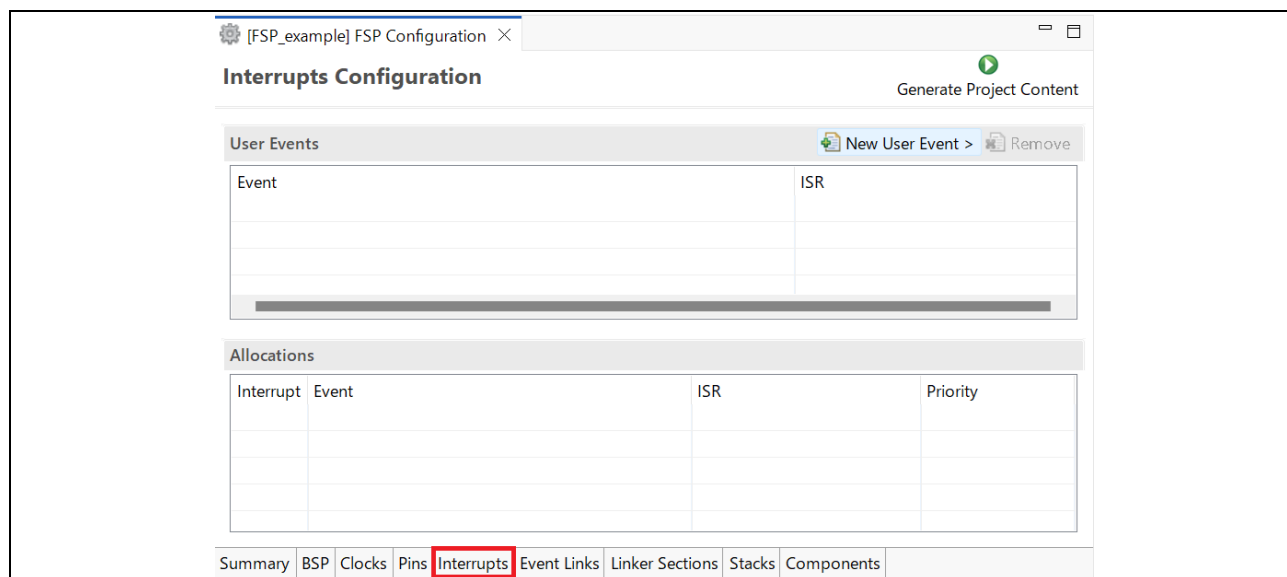


Figure 5-13 [Interrupts Configuration] Page

5.5.1 Interrupt Configuration by Adding a Module

If the settings related to interrupts (settings of callback functions or interrupt priority levels) are enabled when a module is added, information on the interrupts is added according to the settings.

Assume that the settings related to the interrupt of the CMT module are enabled in the [Properties] view as shown below.

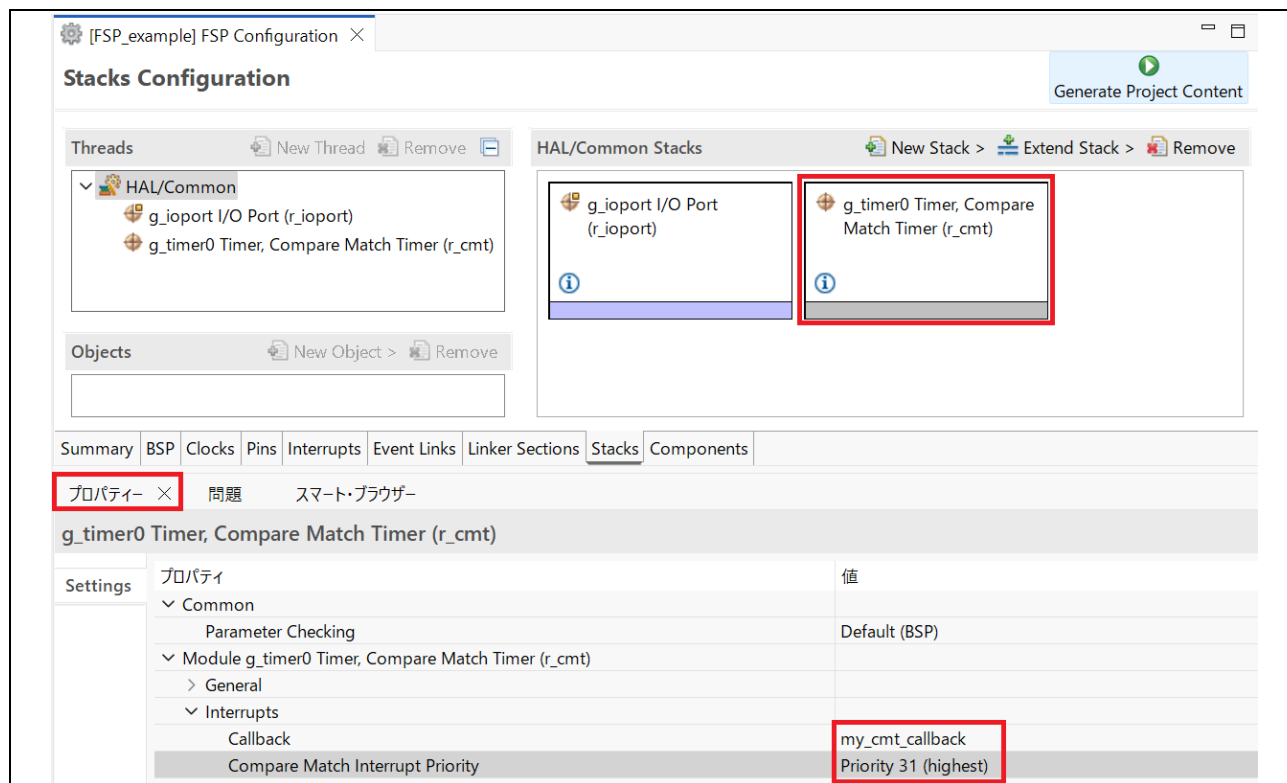


Figure 5-14 Settings for Enabling the CMT Interrupt on the [Stacks Configuration] Page

The information on the use of the CMT0 interrupt is registered as shown below. The name of the interrupt function contained in the module is used without change.

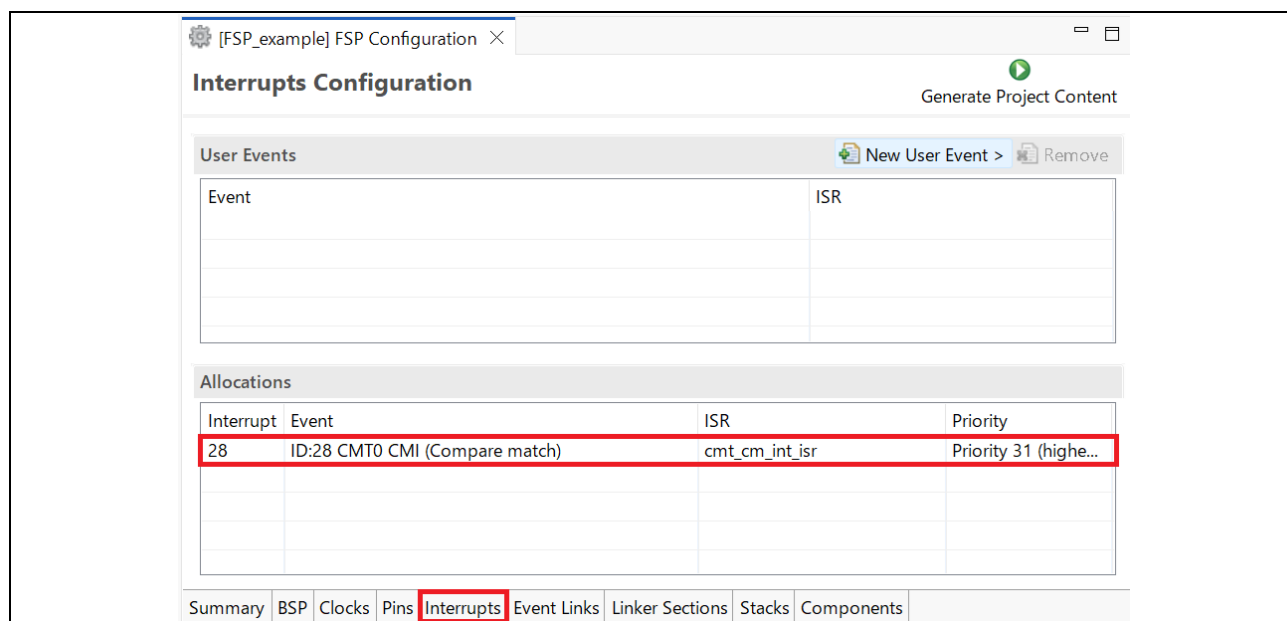


Figure 5-15 Registered CMT Interrupt on the [Interrupts Configuration] Page

Execute [Generate Project Content] to generate the definition of the interrupt function and output the declaration of the interrupt function to vector_data.h. The following screenshot shows vector_data.h after the interrupt function has been output with the above settings.

rx_gen\vector_data.h

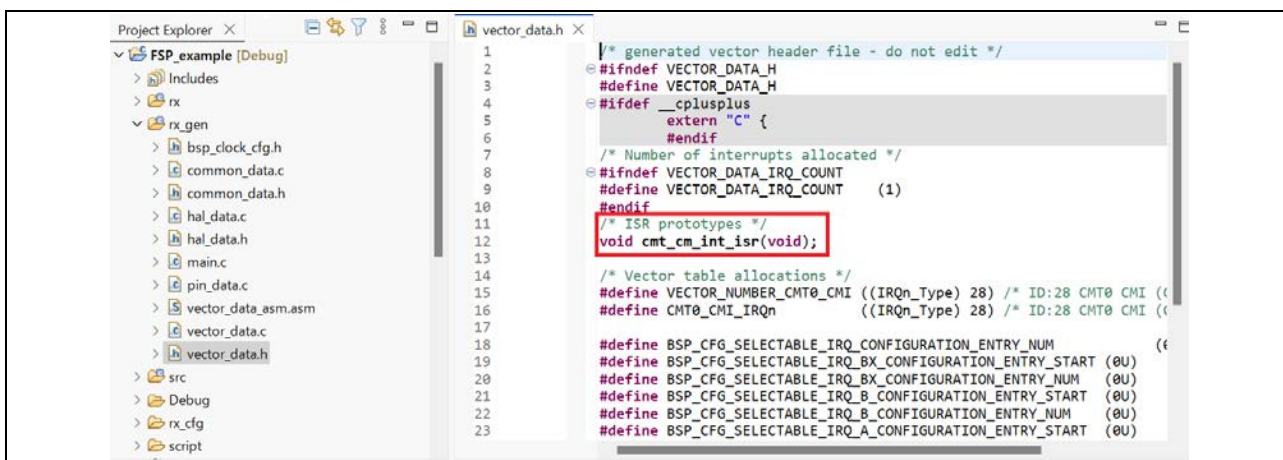


Figure 5-16 Declaration of the Interrupt Function in vector_data.h

The declaration of the callback function is output to hal_data.h.

rx_gen\hal_data.h

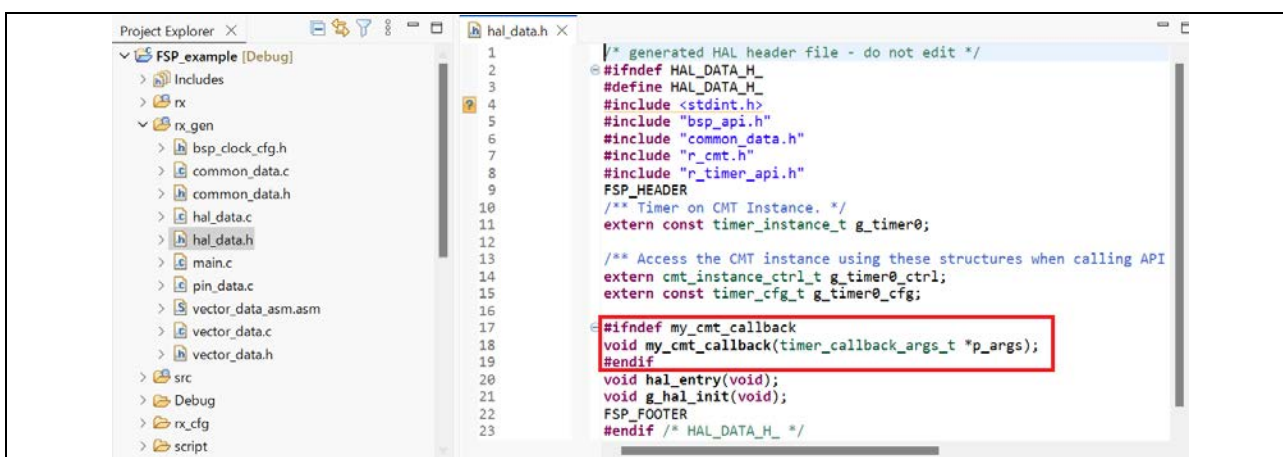


Figure 5-17 Declaration of the Callback Function in hal_data.h

The body of the interrupt function is defined in the module.

rx\fsp\src\r_cmt\r_cmt.c

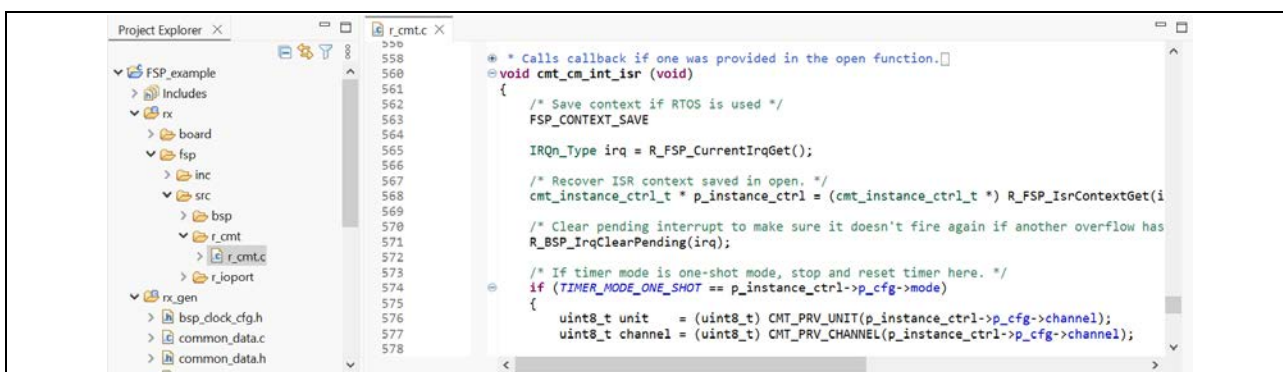


Figure 5-18 Interrupt Function in r_cmt.c

For the callback function, however, only the declaration is output and the user has to create the body of the callback function.

When an FSP module is not to be used for a peripheral function but an interrupt from the module is to be used, refer to section 5.5.2, Interrupt Configuration through Direct Addition.

5.5.2 Interrupt Configuration through Direct Addition

Use the [New User Event] button to select the source of the interrupt to be added and enter the interrupt function name. After that, execute [Generate Project Content] to generate the definition of the interrupt function and output the declaration of the interrupt function to vector_data.h.

Assume that the CMT0 is selected as the interrupt source and the CMT0 interrupt function is defined on the [Interrupts Configuration] page as shown below.

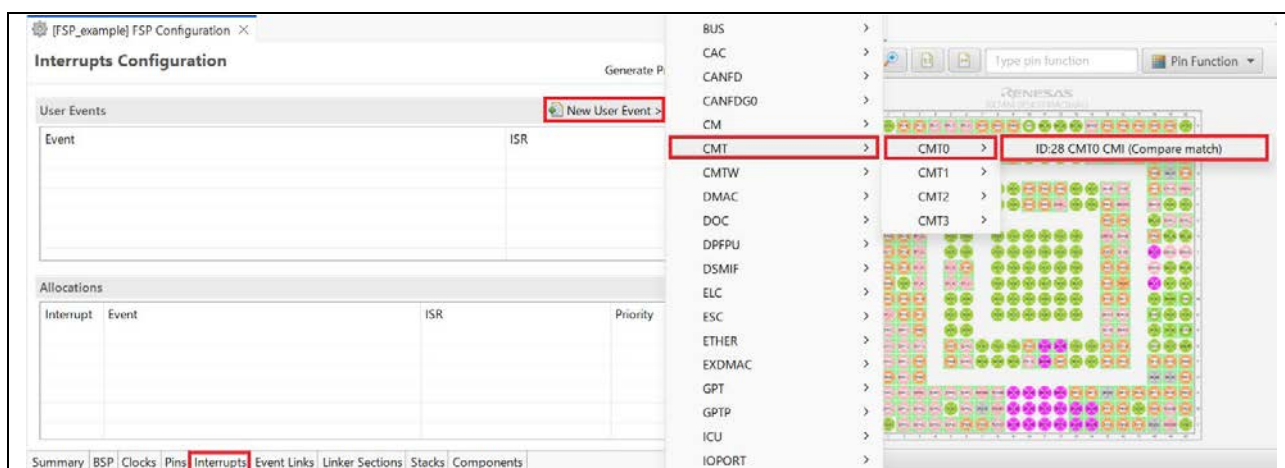


Figure 5-19 Selecting the CMT0 as the Interrupt Source on the [Interrupts Configuration] Page

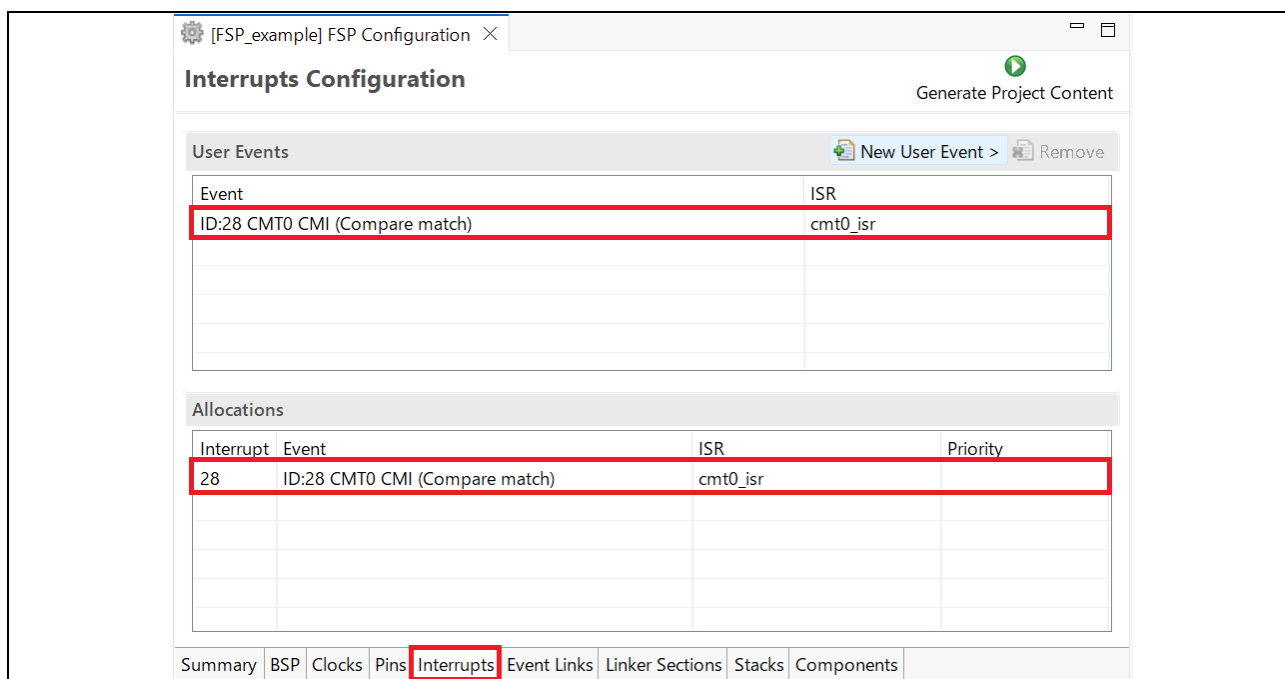


Figure 5-20 Adding the CMT0 Interrupt on the [Interrupts Configuration] Page

The declaration of the interrupt function is output to the generated vector_data.h.

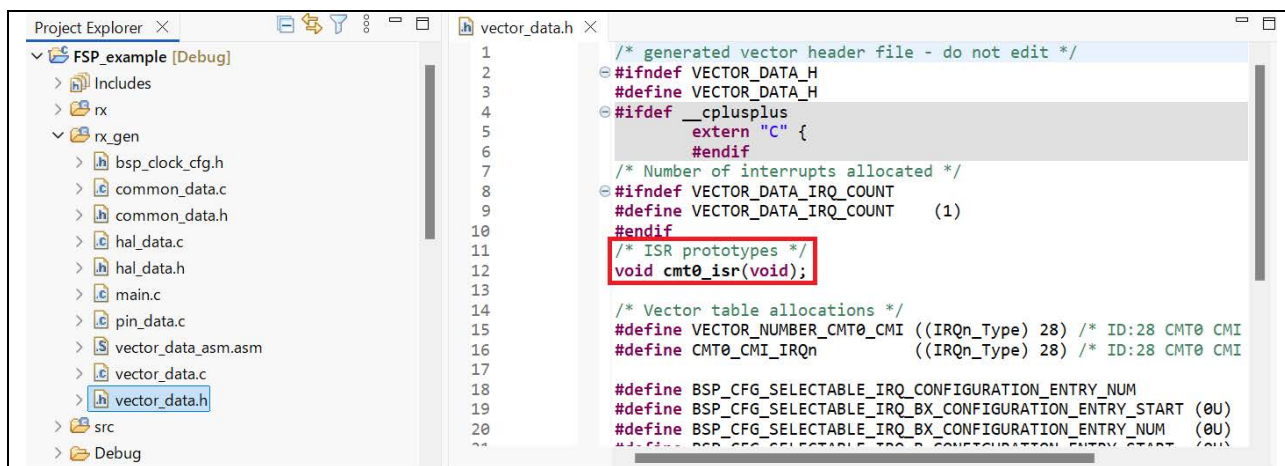


Figure 5-21 Declaration of the CMT0 Interrupt Function in vector_data.h

Only the declaration of the interrupt function is generated and the user has to create the body of the interrupt function.

Do not directly port the declarations of the interrupt functions from the source project because they will conflict with the vector table definitions and interrupt declarations in the BSP of the FSP project.

5.6 Porting a Pin Configuration

In an FSP project, the IOPORT module, which is the module for configuring the pins, is automatically added at the time a project is generated.

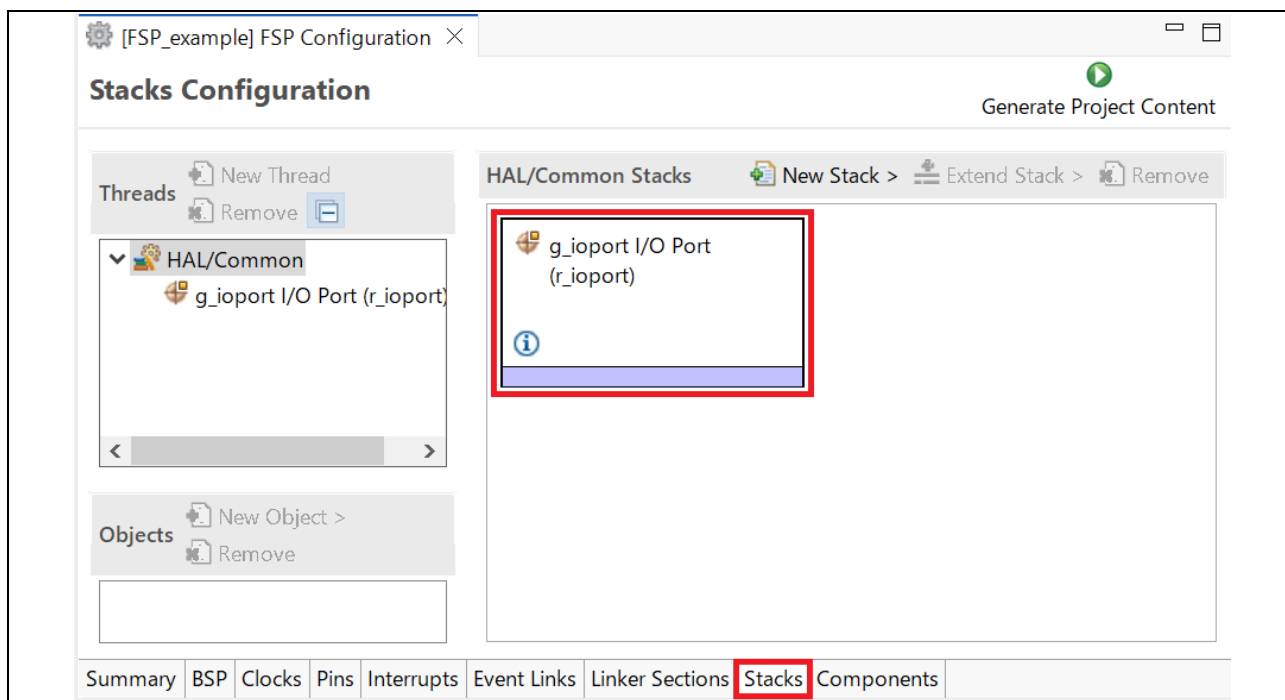


Figure 5-22 IOPORT Module on the [Stacks Configuration] Page

The [Pin Configuration] page can be used to configure the settings of the individual pins.

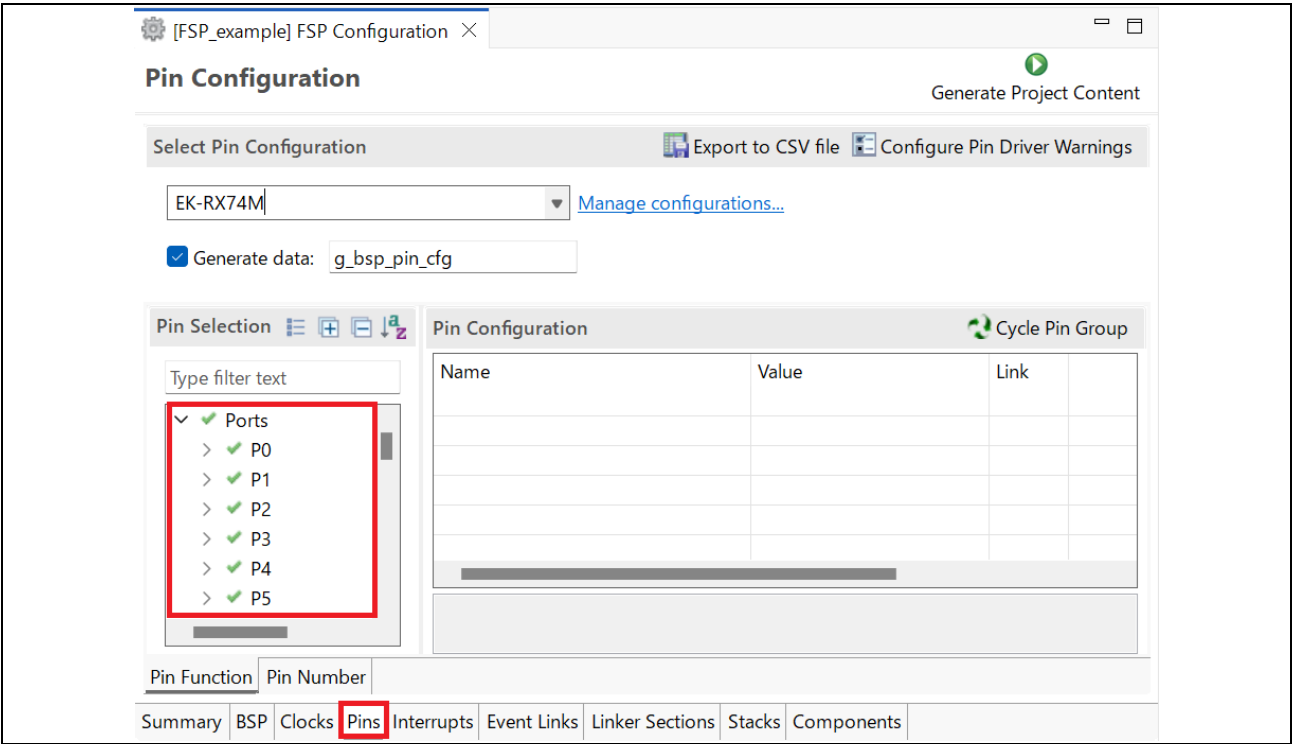


Figure 5-23 [Pin Configuration] Page

Select port pins on the [Pin Configuration] page and configure them in the [Pin Configuration] panel.

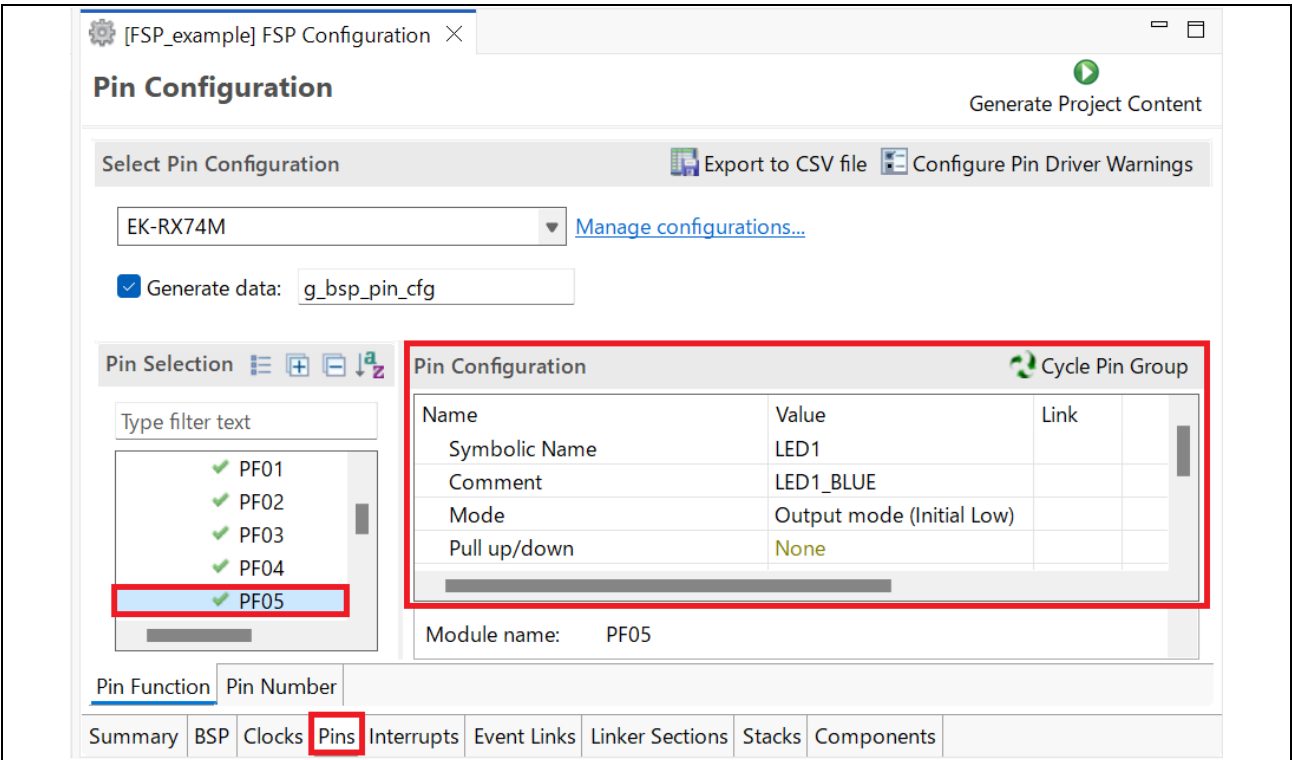


Figure 5-24 [Pin Configuration]

After that, click on the [Generate Project Content] button to output the definitions related to the pin configuration to `pin_data.c`, `common_data.h`, and `common_data.c`.

rx_gen\pin_data.c

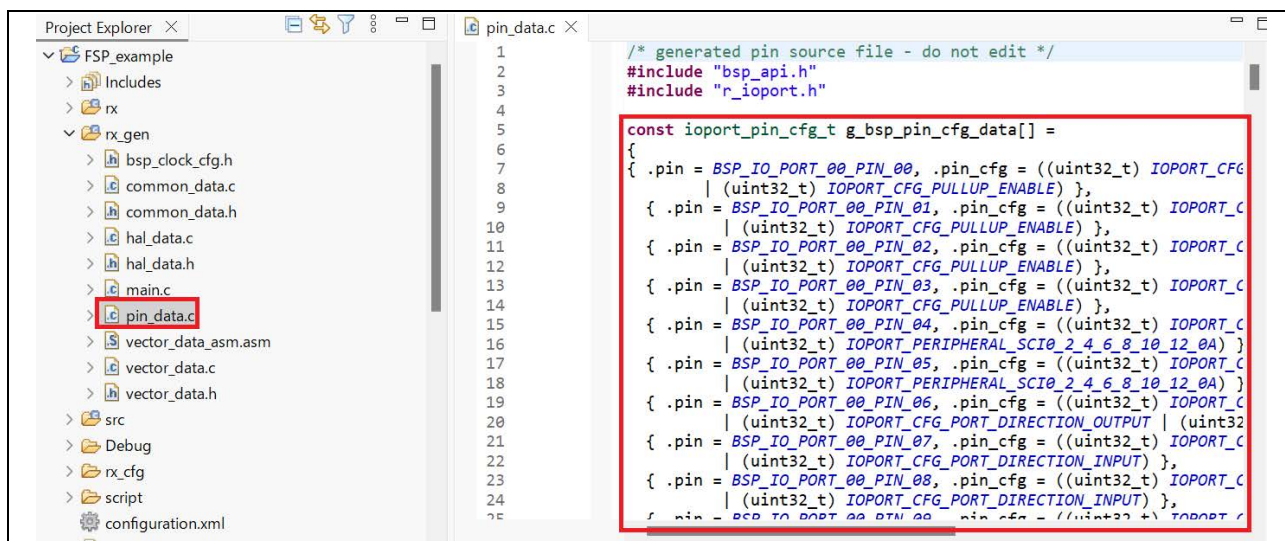


Figure 5-25 Definitions in `pin_data.c`

```
rx gen\common data.h
```

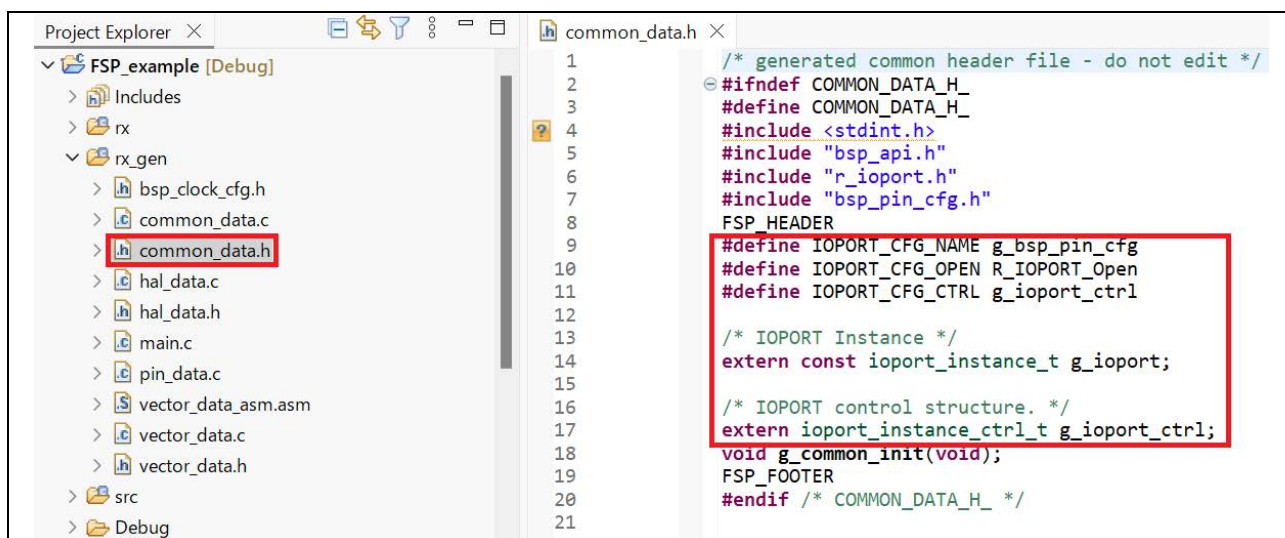


Figure 5-26 Definitions in `common_data.h`

```
rx gen\common data.c
```

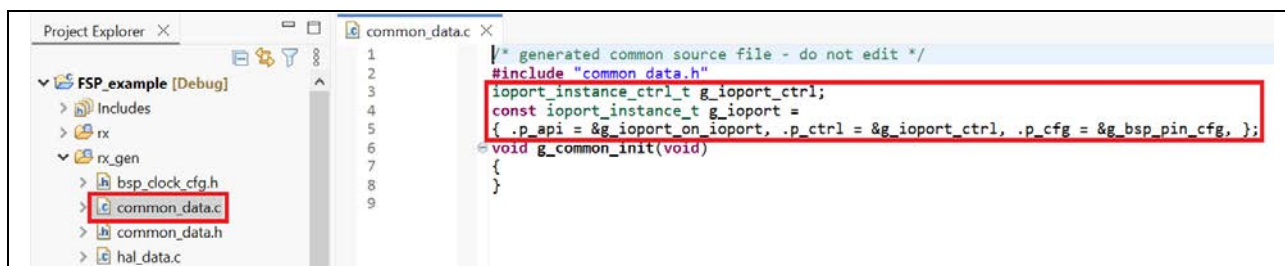


Figure 5-27 Definitions in common `data.c`

The startup processing uses the above definitions to initialize the pins. For an overview of the BSP startup processing, see section 4.2.3, Initializing the BSP in the FSP. To modify the pin configuration, use the API functions of the IOPORT module.

5.7 Porting the Source Code

Add the required sections of the source code from the FIT project to `hal_entry.c` or add them to a separate new file and add the file to the `src` folder immediately under the project. For the `src` folder and `hal_entry.c`, see section 2.6, Structure of the FSP Project Files.

5.8 Adding `iodefine.h`

Separately from the destination project for porting, create a new executable project for the same device as that for the destination project. Add `iodefine.h` in the "generate" folder of the newly created project to the `src` folder immediately under the destination project.

5.9 Replacing Processing with FSP Module Processing

5.9.1 Replacing Processing by FIT Modules with Processing by FSP Modules

Check the specifications of the API functions of the FIT modules in use and the FSP modules to be used, and replace the processing by FIT modules with processing by FSP modules.

The following shows an example of replacing the processing to set up operation by the CMT module.

Processing by the CMT module of FIT

```
bool      cmt_result;
uint32_t  cmt_hdl;

/* Create a timer tick with callback at a rate of 2Hz. */
cmt_result = R_CMT_CreatePeriodic(2, cmt_cb, &cmt_hdl);
```

Processing by the CMT module of the FSP

```
fsp_err_t err = FSP_SUCCESS;

/* Create a timer tick with callback at a rate of 2Hz. */
/* Called the R_CMT_Open to initialize the module for CMT0. */
err = g_timer0.p_api->open(&g_timer0_ctrl, &g_timer0_cfg);

/* Handle any errors. This function should be defined by the user. */
assert(FSP_SUCCESS == err);

/* Called the R_CMT_Start to start the timer for CMT0. */
(void) g_timer0.p_api->start(&g_timer0_ctrl);
```

5.9.2 Replacing Processing Created by the Code Generator with Processing by FSP Modules

Check the specifications of the API functions of the Code Generator in use and the FSP modules to be used, and replace the processing created by the Code Generator with the processing by FSP modules.

The following shows an example of replacement of an API function for the compare match timer of the Code Generator.

Processing by the API function for the compare match timer

```
R_Config_CMT0_Start();
```

Processing by the CMT module of the FSP

```
fsp_err_t err = FSP_SUCCESS;

/* Create a timer tick with callback at a rate of 2Hz. */
/* Called the R_CMT_Open to initialize the module for CMT0. */
err = g_timer0.p_api->open(&g_timer0_ctrl, &g_timer0_cfg);

/* Handle any errors. This function should be defined by the user. */
assert(FSP_SUCCESS == err);

/* Called the R_CMT_Start to start the timer for CMT0. */
(void) g_timer0.p_api->start(&g_timer0_ctrl);
```

6. Examples of Porting FIT Applications to FSP Projects

This chapter takes two projects using FIT modules as examples and describes the methods of porting them to FSP projects.

6.1 Example of Porting an FIT-Based CMT Sample Program

The following describes the procedure for porting the sample program of a CMT demonstration project to an FSP project. For the sample program, the `cmt_demo_rskrx72m` project (hereafter called the source project) in [CMT FIT](#) is used.

6.1.1 Structure of the Folders and Files in the Source Project and the Targets for Porting

The following shows the structure of the folders and files in the source project and the target files for porting. `main.c` and `rskrx72m.h` are the targets for porting.

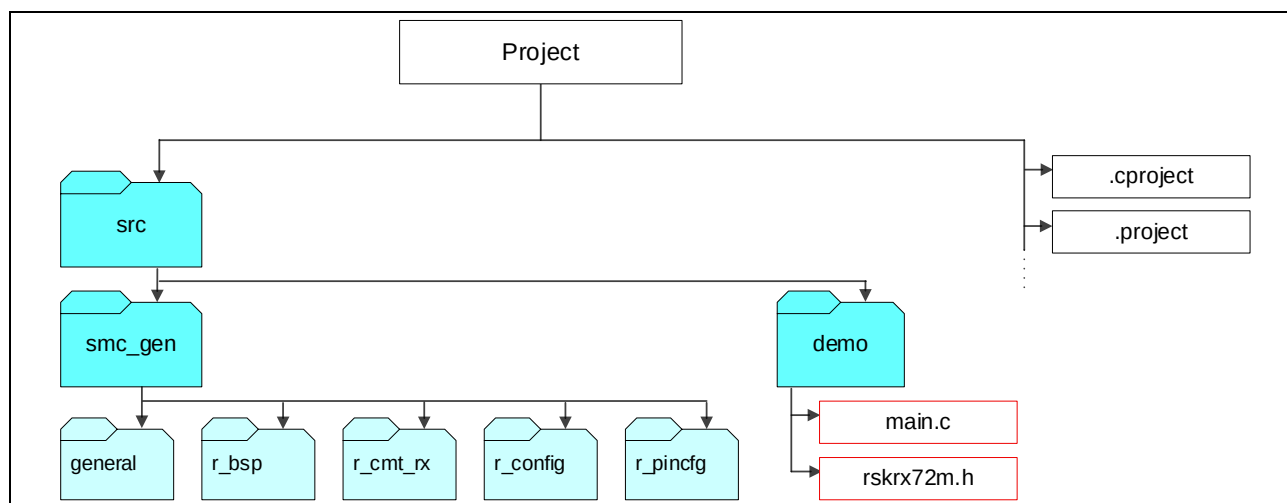


Figure 6-1 Structure of the Folders and Files in the Source Project and the Targets for Porting

6.1.2 Overview of the Processing in the Source Project

The following listing is of `main.c` in the source project.

`src\demo\main.c`

```

/*****
Includes  <System Includes> , "Project Includes"
*****/

#include "r_cmt_rx_if.h"
#include "rskrx72m.h"

/*****
Private global variables and functions
*****/

void main(void);
void cmt_cb(void *pdata);

/*****

```

```

* Function Name : main
* Description   : Assigns a CMT timer channel to the callback function cmt_cb.
* Arguments     : none
* Return value  : none
*****/
void main()
{
    bool      cmt_result;
    uint32_t  cmt_hdl;

    /* Enable LEDs. */
    /* Start with LEDs off. */
    LED0 = LED_OFF;
    LED1 = LED_OFF;

    /* Set LED pins as outputs. */
    LED0_PDR = 1;
    LED1_PDR = 1;

    /* Create a timer tick with callback at a rate of 2Hz. */
    cmt_result = R_CMT_CreatePeriodic(2, cmt_cb, &cmt_hdl);
    cmt_result = R_CMT_CreatePeriodic(2, cmt_cb, &cmt_hdl);

    if (false == cmt_result)
    {
        R_BSP_NOP(); //Handle your error here.
    }

    /* End of demo, spin here */
    while(1);
}

/*****
* Function Name : cmt_cb
* Description   : This is the callback function that was assigned to the CMT
*                 driver in main. It is called by the CMT driver at a 2 Hz rate
*                 and toggles LED0 with each call generating 1Hz blink rate.
* Arguments     : pdata - pointer to void
* Return value  : none
*****/
void cmt_cb(void *pdata)

```

```

{
    uint32_t channel_num;

    /* This is an example of how to dereference the channel number when
     * more than one channel is used. Channel numbers can be 0 to 3. */
    channel_num = *((uint32_t *)pdata);

    switch(channel_num)
    {
    case 0:
        LED0 = ~LED0; // Toggle LED0 with each callback
        break;

    case 1:
        LED1 = ~LED1; // Toggle LED1 with each callback
        break;

    default:
        break;
    }
}

```

rskrx72m.h is included as a file related to the demonstration. The following shows the contents of rskrx72m.h, in which definitions from iodef.h are used.

src\demo\rskrx72m.h

```

#ifndef RSKRX72M_H
#define RSKRX72M_H

/* Local defines */
#define LED_ON            (0)
#define LED_OFF          (1)
#define SET_BIT_HIGH     (1)
#define SET_BIT_LOW      (0)
#define SET_BYTE_HIGH    (0xFF)
#define SET_BYTE_LOW     (0x00)

/* Switches */
#define SW_ACTIVE         0
#define SW1               PORT4.PIDR.BIT.B5
#define SW2               PORT4.PIDR.BIT.B4

```

```

#define SW3          PORT0.PIDR.BIT.B7
#define SW1_PDR      PORT4.PDR.BIT.B5
#define SW2_PDR      PORT4.PDR.BIT.B4
#define SW3_PDR      PORT0.PDR.BIT.B7
#define SW1_PMR      PORT4.PMR.BIT.B5
#define SW2_PMR      PORT4.PMR.BIT.B4
#define SW3_PMR      PORT0.PMR.BIT.B7

/* LEDs */
#define LED0          PORT4.PODR.BIT.B2
#define LED1          PORTH.PODR.BIT.B0
#define LED2          PORTN.PODR.BIT.B4
#define LED3          PORT8.PODR.BIT.B5
#define LED0_PDR      PORT4.PDR.BIT.B2
#define LED1_PDR      PORTH.PDR.BIT.B0
#define LED2_PDR      PORTN.PDR.BIT.B4
#define LED3_PDR      PORT8.PDR.BIT.B5

#endif /* RSKRX72M_H */

```

The program in the source project uses the `R_CMT_CreatePeriodic` function twice to acquire unused channels of the CMT and activate them to operate at a frequency of 2 Hz. This execution generates compare match interrupts at a frequency of 2 Hz from CMT0 and CMT1 and causes LEDs to blink.

6.1.3 Creating the Destination Project for Porting

Use the procedure described in Chapter 5, Steps in Porting a Pre-FSP Project to an FSP Project, to create the destination project for porting. The new FSP project created with reference to the text from section 5.1, Creating a New FSP Project to section 5.4, Porting Modules, is used as the base project.

6.1.4 Adding CMT Modules

The CMT module of the FSP does not have a function for acquiring a channel for use from among the unused channels, which FIT provides in the form of the `R_CMT_CreatePeriodic` function in the CMT module. Therefore, the channels to be used must be determined and defined in advance.

Modify the callback function and interrupt priority level in the properties of the CMT module with reference to section 5.4, Porting Modules, and then use the following procedure to add two CMT modules for handling two channels (CMT0 and CMT1) to the project.

From the [New Stack] button, repeat selecting [Timers] → [Timer, Compare Match Timer (r_cmt)] twice to add `g_timer0` for CMT0 and `g_timer1` for CMT1 to the project.

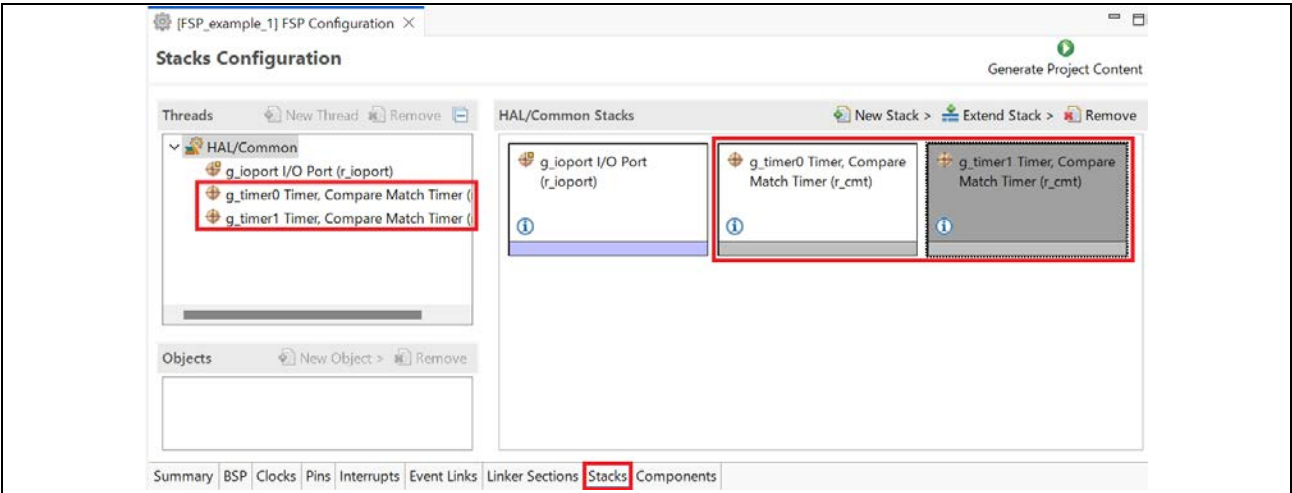


Figure 6-2 Adding CMT Modules (r_cmt) (CMT0 and CMT1) on the [Stacks Configuration] Page

g_timer0 is used for CMT0. For the properties, set "Period" and "Period Unit" to "2" and "Hertz" and specify the callback function name and interrupt priority level to match those of the CMT sample program of FIT as shown below.

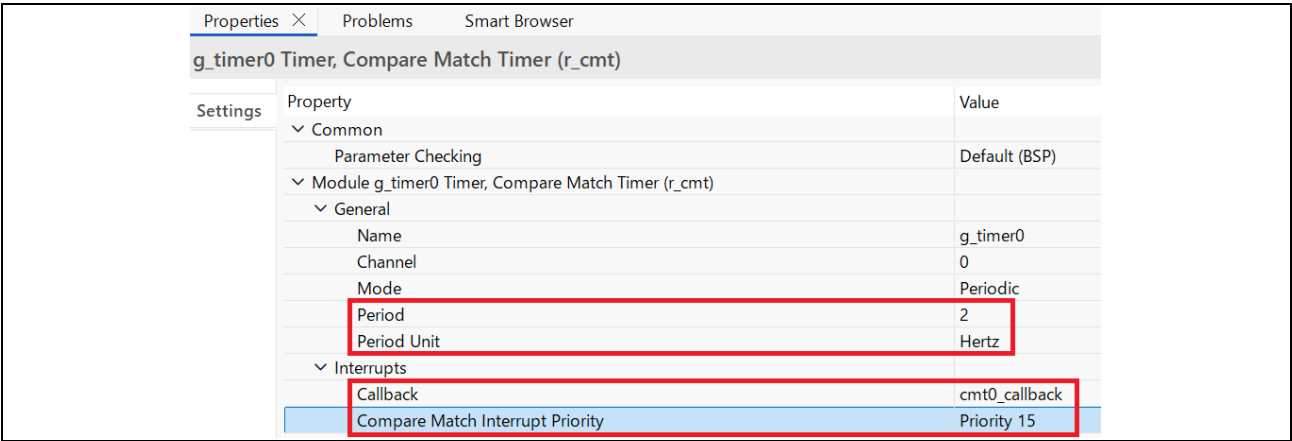


Figure 6-3 r_cmt g_timer0 (CMT0) Properties

g_timer1 is used for CMT1. For the properties, set "Channel" to "1" and "Period" and "Period Unit" to "2" and "Hertz" and specify the callback function name and interrupt priority level to match those of the CMT sample program of FIT as shown below.

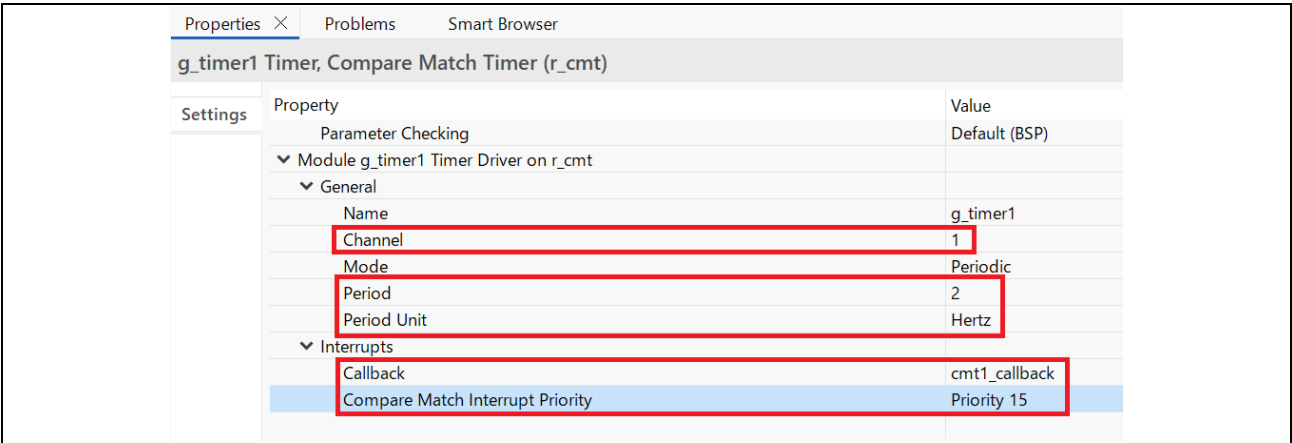


Figure 6-4 r_cmt g_timer1 (CMT1) Properties

6.1.5 Structure of the Folders and Files in the Project after Porting and the Targets for Modification

The following shows the structure of the folders and files in the project after porting and the target files for modification. `hal_entry.c` and ported `rskrx72m.h` are the targets for modification. For `iodef.h`, add the file created in the new executable project.

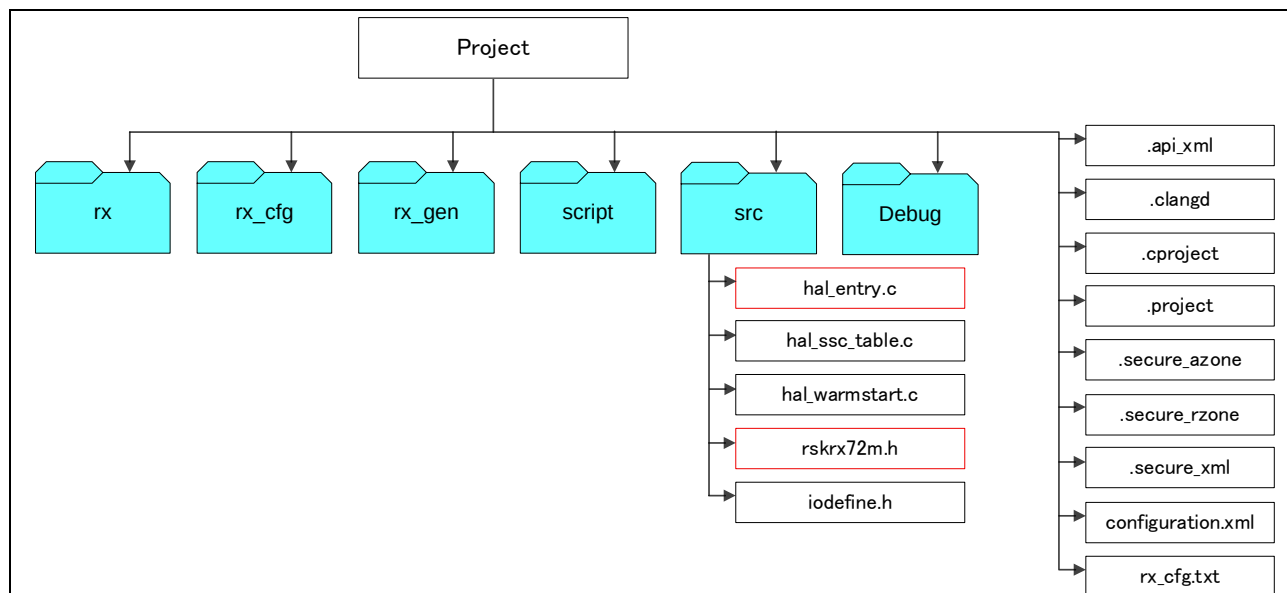


Figure 6-5 Structure of the Folders and Files in the Project after Porting and the Targets for Modification

6.1.6 Key Points in Porting

Use the procedure described in section 5.7, Porting the Source Code, to port the processing of the main function body and the CMT callback function in `main.c` to `hal_entry.c`. Also port `rskrx72m.h`.

Some of the definitions in `rskrx72m.h` use definitions provided by `iodef.h`, so add `iodef.h` by following the procedure described in section 5.8, Adding `iodef.h`. As the target device differs between the source and destination projects, some definitions to be used will require modification. Additionally, some definition names have been changed to avoid conflicts with definitions provided by the FSP.

Refer to section 5.9, Replacing Processing with FSP Module Processing, and replace the processing by the FIT module with the equivalent processing by the FSP.

6.1.7 Processing after Replacement of the Target Files for Modification

The following listing is of the final contents of `hal_entry.c` and `rskrx72m.h` after modification.

`hal_entry.c` (The final results of addition are shown in red.)

```

#include "hal_data.h"
#include "rskrx72m.h"

FSP_CPP_HEADER
FSP_CPP_FOOTER

/*****
 * main() is generated by the RX Configuration editor and is used to generate

```

```

* threads if an RTOS is used. This function is called by main() when no
* RTOS is used.
*****/
void hal_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err = FSP_SUCCESS;

    /* Enable LEDs. */
    /* Start with LEDs off. */
    LED0 = LED_OFF;
    LED1_PODR = LED_OFF;

    /* Set LED pins as outputs. */
    LED0_PDR = 1;
    LED1_PDR = 1;

    /* Create a timer tick with callback at a rate of 2Hz. */
    /* Called the R_CMT_Open to initialize the module for CMT0. */
    err = g_timer0.p_api->open(&g_timer0_ctrl, &g_timer0_cfg);

    /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);

    /* Called the R_CMT_Start to start the timer for CMT0. */
    (void) g_timer0.p_api->start(&g_timer0_ctrl);

    /* Called the R_CMT_Open to initialize the module for CMT1. */
    err = g_timer1.p_api->open(&g_timer1_ctrl, &g_timer1_cfg);

    /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);

    /* Called the R_CMT_Start to start the timer for CMT1. */
    (void) g_timer1.p_api->start(&g_timer1_ctrl);

    /* End of demo, spin here */
    while(1);

#ifdef BSP_PZ_SECURE_BUILD
    /* Enter non-secure code */

```

```

    R_BSP_NonSecureEnter();
#endif
}

void cmt0_callback(timer_callback_args_t *p_args)
{
    if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
        /* Add application code to be called periodically here. */
        LED0 = ~LED0; // Toggle LED0 with each callback
    }
}

void cmt1_callback(timer_callback_args_t *p_args)
{
    if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
        /* Add application code to be called periodically here. */
        LED1_PODR = ~LED1_PODR; // Toggle LED1 with each callback
    }
}

```

rskrx72m.h (The final results of modification are shown in red.)

```

#include "iodefine.h"

#ifndef RSKRX72M_H
#define RSKRX72M_H

/* Local defines */
#define LED_ON          (0)
#define LED_OFF         (1)
#define SET_BIT_HIGH    (1)
#define SET_BIT_LOW     (0)
#define SET_BYTE_HIGH   (0xFF)
#define SET_BYTE_LOW    (0x00)

/* Switches */
#define SW_ACTIVE        0
#define SW1_PIDR         PORT4.PIDR.BIT.B5
#define SW2              PORT4.PIDR.BIT.B4
#define SW3              PORT0.PIDR.BIT.B7

```

```

#define SW1_PDR          PORT4.PDR.BIT.B5
#define SW2_PDR          PORT4.PDR.BIT.B4
#define SW3_PDR          PORT0.PDR.BIT.B7
#define SW1_PMR          PORT4.PMR.BIT.B5
#define SW2_PMR          PORT4.PMR.BIT.B4
#define SW3_PMR          PORT0.PMR.BIT.B7

/* LEDs */
#define LED0              PORTF.PODR.BIT.PODR06
#define LED1_PODR         PORT8.PODR.BIT.PODR07
#define LED2_PODR         PORTN.PODR.BIT.B4
#define LED3_PODR         PORT8.PODR.BIT.B5
#define LED0_PDR          PORTF.PDR.BIT.PDR06
#define LED1_PDR          PORT8.PDR.BIT.PDR07
#define LED2_PDR          PORTN.PDR.BIT.B4
#define LED3_PDR          PORT8.PDR.BIT.B5

#endif /* RSKRX72M_H */

```

The program after porting uses the R_CMT_Open and R_CMT_Start functions twice to activate the CMT0 and CMT1 channels to operate at a frequency of 2 Hz. This execution generates compare match interrupts at a frequency of 2 Hz from CMT0 and CMT1 and causes LEDs to blink.

6.1.8 Replacement of Processing by the FIT Modules with Processing by the FSP Modules in this Example of Porting

The following figures compare the processing by the FIT modules before porting and the processing by the FSP modules after porting and replacement.

6.1.8.1 Comparison of the Main Processing

Before porting (FIT)	After porting (FSP)
<pre> void main() { bool cmt_result; uint32_t cmt_hdl; /* Enable LEDs. */ /* Start with LEDs off. */ LED0 = LED_OFF; LED1 = LED_OFF; /* Set LED pins as outputs. */ LED0_PDR = 1; LED1_PDR = 1; /* Create a timer tick with callback at a rate of 2Hz. */ cmt_result = R_CMT_CreatePeriodic(2, cmt_cb, &cmt_hdl); cmt_result = R_CMT_CreatePeriodic(2, cmt_cb, &cmt_hdl); if (false == cmt_result) { R_BSP_NOP(); //Handle your error here. } /* End of demo, spin here */ while(1); } </pre>	<pre> void hal_entry(void) { /* TODO: add your own code here */ fsp_err_t err = FSP_SUCCESS; /* Enable LEDs. */ /* Start with LEDs off. */ LED0 = LED_OFF; LED1_PODR = LED_OFF; /* Set LED pins as outputs. */ LED0_PDR = 1; LED1_PDR = 1; /* Create a timer tick with callback at a rate of 2Hz. */ /* Called the R_CMT_Open to initialize the module for CMT0. */ err = g_timer0.p_api->open(&g_timer0_ctrl, &g_timer0_cfg); /* Handle any errors. This function should be defined by the user. */ assert(FSP_SUCCESS == err); /* Called the R_CMT_Start to start the timer for CMT0. */ (void) g_timer0.p_api->start(&g_timer0_ctrl); /* Called the R_CMT_Open to initialize the module for CMT1. */ err = g_timer1.p_api->open(&g_timer1_ctrl, &g_timer1_cfg); /* Handle any errors. This function should be defined by the user. */ assert(FSP_SUCCESS == err); /* Called the R_CMT_Start to start the timer for CMT1. */ (void) g_timer1.p_api->start(&g_timer1_ctrl); /* End of demo, spin here */ while(1); #if BSP_PZ_SECURE_BUILD /* Enter non-secure code */ R_BSP_NonSecureEnter(); #endif } </pre>

Figure 6-6 Main Processing

6.1.8.2 Comparison of the Callback Functions

Before porting (FIT)	After porting (FSP)
<pre> void cmt_cb(void *pdata) { uint32_t channel_num; /* This is an example of how to dereference the channel number when more than * one channel is used. Channel numbers can be 0 to 3. */ channel_num = *((uint32_t *)pdata); switch(channel_num) { case 0: LED0 = ~LED0; // Toggle LED0 with each callback break; case 1: LED1 = ~LED1; // Toggle LED1 with each callback break; default: break; } } </pre>	<pre> void cmt0_callback(timer_callback_args_t *p_args) { if (TIMER_EVENT_CYCLE_END == p_args->event) { /* Add application code to be called periodically here. */ LED0 = ~LED0; // Toggle LED0 with each callback } } void cmt1_callback(timer_callback_args_t *p_args) { if (TIMER_EVENT_CYCLE_END == p_args->event) { /* Add application code to be called periodically here. */ LED1_PODR = ~LED1_PODR; // Toggle LED1 with each callback } } </pre>

Figure 6-7 Callback Functions

6.2 Example of Porting an FIT-Based RSCI Sample Program

The following describes the procedure for porting the sample program of an RSCI demonstration project to an FSP project. For the sample program, the rsci_demo_rskrx671 project (hereafter called the source project) in [RSCI FIT](#) is used.

6.2.1 Structure of the Folders and Files in the Source Project and the Targets for Porting

The following shows the structure of the folders and files in the source project and the target files for porting. main.c and rskrx671.h are the targets for porting.

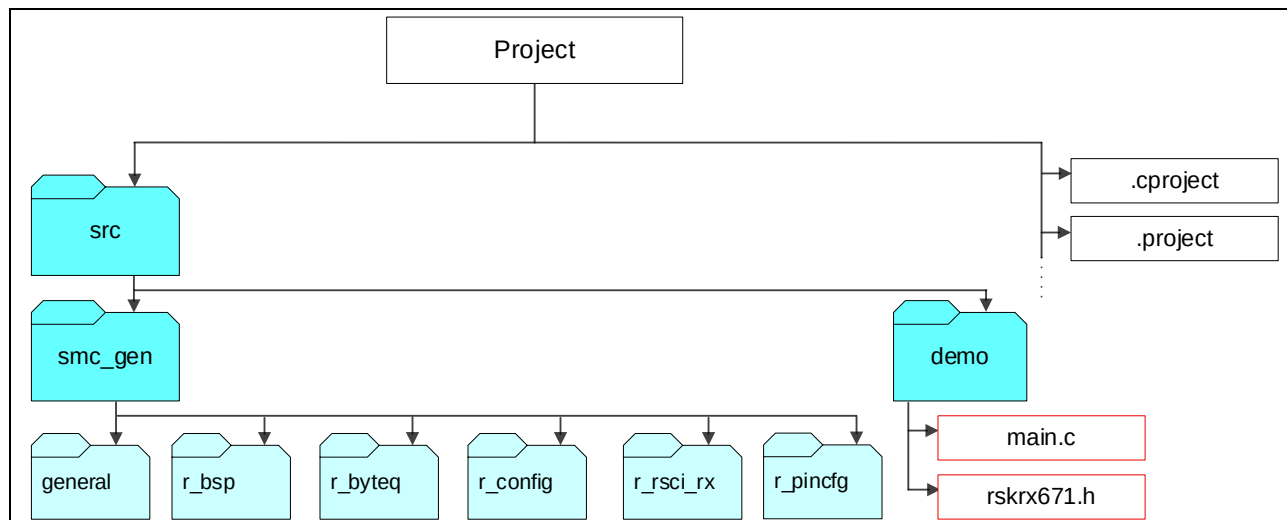


Figure 6-8 Structure of the Folders and Files in the Source Project and the Targets for Porting

6.2.2 Overview of the Processing in the Source Project

The following listing is of main.c in the source project.

src\demo\main.c

```

/*****
Includes  <System Includes> , "Project Includes"
*****/
#include <stdio.h>          /* For sprintf */
#include "platform.h"       /* Located in the FIT BSP module */
#include "r_rsci_rx_if.h"   /* The RSCI module API interface file. */
#include "r_byteq_if.h"    /* The BYTEQ module API interface file. */
#include "r_rsci_rx_config.h" /* User configurable options for the RSCI
                             module */
#include "rskrx671.h"       /* Include user define */

/*****
Private global variables and functions
*****/
void my_rsci_callback(void *pArgs);
static void rsci_rx671_init_ports(void);

```

```

/* Handle storage. Needs to persist as long as RSCI calls are going to be
   made.*/
static rsci_hdl_t  g_my_rsci_handle;

/*****
 * Function Name: main()
 * Description  : Main loop for demo.
 * Arguments    : none
 * Return Value : none
 *****/
void main(void)
{
    LED0_PDR = 1;
    rsci_cfg_t  my_rsci_config;
    rsci_err_t  my_rsci_err;
    uint32_t    my_version;
    uint8_t     version_str[12] = {0};
    uint8_t     my_char;

    /* Initialize the I/O port pins for communication on this RSCI channel.
     * This is specific to the MCU and ports chosen. For the RSKRX671 we will
     * use the RSCI channel connected to the USB serial port emulation. */
    rsci_rx671_init_ports();

    /* Set up the configuration data structure for asynchronous (UART)
       operation. */
    my_rsci_config.async.baud_rate    = 115200;
    my_rsci_config.async.clk_src      = RSCI_CLK_INT;
    my_rsci_config.async.data_size    = RSCI_DATA_8BIT;
    my_rsci_config.async.parity_type  = RSCI_NONE_PARITY;
    my_rsci_config.async.msb_first    = false;
    my_rsci_config.async.invert_data  = false;
    my_rsci_config.async.stop_bits    = RSCI_STOPBITS_1;
    my_rsci_config.async.int_priority = 3;    /* 1=lowest, 15=highest */

    /* OPEN ASYNC CHANNEL
     * Provide address of the config structure,
     * the callback function to be assigned,
     * and the location for the handle to be stored.*/
    my_rsci_err = R_RSCI_Open(RSCI_CH10, RSCI_MODE_ASYNC, &my_rsci_config,
                             my_rsci_callback, &g_my_rsci_handle);

```

```
/* If there were an error this would demonstrate error detection of API
calls. */
if (RSCI_SUCCESS != my_rsci_err)
{
    R_BSP_NOP();    /* Your error handling code would go here. */
}

/* Get module software version using the API function provided. */
my_version = R_RSCI_GetVersion();

/* Demonstrate parsing and formatting of version number. */
sprintf((char *)version_str, "RSCIV%1.0hu.%2.2hu¥r¥n",
        ((my_version >> 16)&0xf), (my_version & 0xff));

printf("Initializes the RSCI successfully¥n¥s",version_str);

/* Wait to get a character from the terminal so we know that it is ready. */
do
{
    my_rsci_err = R_RSCI_Receive(g_my_rsci_handle, &my_char, 1);
} while (RSCI_ERR_INSUFFICIENT_DATA == my_rsci_err);

/* Now send the version string out through the serial port to the
terminal. */
R_RSCI_Send(g_my_rsci_handle, version_str, 12);

/* Repeat forever. */
while(1)
{
    /* Read next character from the serial port receive queue.
    * Repeat read if needed until data is available in the queue. */
    do
    {
        my_rsci_err = R_RSCI_Receive(g_my_rsci_handle, &my_char, 1);
    } while (RSCI_ERR_INSUFFICIENT_DATA == my_rsci_err);

    /* Echo 1 character of the received data back to the sender. */
    printf("%c", (char)(my_char));
    R_RSCI_Send(g_my_rsci_handle, &my_char, 1);
}
```

```

} /* End of function main. */

/*****
 * Function Name: my_rsci_callback
 * Description  : This is a template for an RSCI Async Mode callback function.
 * Arguments    : pArgs -
 *                pointer to rsci_cb_args_t structure cast to a void. Structure
 *                contains event and associated data.
 * Return Value : none
 *****/
void my_rsci_callback(void *pArgs)
{
    rsci_cb_args_t  *args;

    args = (rsci_cb_args_t *)pArgs;

    if (args->event == RSCI_EVT_RX_CHAR)
    {
        /* From RXI interrupt; received character data is in args->byte */
        LED0 = ~LED0;          // Toggle LED to demonstrate callback execution.
        R_BSP_NOP();
    }
    else if (args->event == RSCI_EVT_RXBUF_OVFL)
    {
        /* From RXI interrupt; rx queue is full; 'lost' data is in args->byte
           You will need to increase buffer size or reduce baud rate */
        R_BSP_NOP();
    }
    else if (args->event == RSCI_EVT_OVFL_ERR)
    {
        /* From receiver overflow error interrupt; error data is in args->byte
           Error condition is cleared in calling interrupt routine */
        R_BSP_NOP();
    }
    else if (args->event == RSCI_EVT_FRAMING_ERR)
    {
        /* From receiver framing error interrupt; error data is in args->byte
           Error condition is cleared in calling interrupt routine */
        R_BSP_NOP();
    }
    else if (args->event == RSCI_EVT_PARITY_ERR)

```

```

    {
        /* From receiver parity error interrupt; error data is in args->byte
           Error condition is cleared in calling interrupt routine */
        R_BSP_NOP();
    }
}

/*****
* Function Name: rsci_rx671_init_ports
* Description  : This function initializes the port pins associated with RSCI
*                channel 10, which is connected to the USB serial port Virtual
*                COM Interface.
* Arguments    : none
* Return Value : none
*****/
static void rsci_rx671_init_ports(void)
{
    /* Use BSP function to unlock the MPC register. */
    R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);

    /* Initialize channel 2 TXD and RXD pins */
    PORT8.PODR.BIT.B7 = 0;      // Set P87(T16) signal state low
                                // (power-up default)
    MPC.P87PFS.BYTE = 0x2C;     // Assign P87(T16) to TXD010 function
    MPC.P86PFS.BYTE = 0x2C;     // Assign P86(T17) to RXD010 function

    /* Re-lock the MPC register. */
    R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_MPC);

    PORT8.PDR.BIT.B7 = 1;      // Set P87(T16) direction to output
                                // (for use as TXD)
    PORT8.PDR.BIT.B6 = 0;      // Set P86(T17) direction to input
                                // (for use as RXD)
    PORT8.PMR.BIT.B7 = 1;      // Set P87(T16) mode to peripheral operation
    PORT8.PMR.BIT.B6 = 1;      // Set P86(T17) mode to peripheral operation

    return;
}

```

rskrx671.h is included as a file related to the demonstration. The following shows the contents of rskrx671.h.

src\demo\rskrx671.h

```

#ifndef RSKRX671_H
#define RSKRX671_H

/* Local defines */
#define LED_ON          (0)
#define LED_OFF        (1)
#define SET_BIT_HIGH    (1)
#define SET_BIT_LOW     (0)
#define SET_BYTE_HIGH   (0xFF)
#define SET_BYTE_LOW    (0x00)

/* Switches */
#define SW_ACTIVE        0
#define SW1              PORT9.PIDR.BIT.B1
#define SW2              PORT9.PIDR.BIT.B2
#define SW3              PORT0.PIDR.BIT.B7
#define SW1_PDR          PORT9.PDR.BIT.B1
#define SW2_PDR          PORT9.PDR.BIT.B2
#define SW3_PDR          PORT0.PDR.BIT.B7
#define SW1_PMR          PORT9.PMR.BIT.B1
#define SW2_PMR          PORT9.PMR.BIT.B2
#define SW3_PMR          PORT0.PMR.BIT.B7

/* LEDs */
#define LED0             PORT1.PODR.BIT.B7
#define LED1             PORTF.PODR.BIT.B5
#define LED2             PORT0.PODR.BIT.B3
#define LED3             PORT0.PODR.BIT.B5
#define LED0_PDR         PORT1.PDR.BIT.B7
#define LED1_PDR         PORTF.PDR.BIT.B5
#define LED2_PDR         PORT0.PDR.BIT.B3
#define LED3_PDR         PORT0.PDR.BIT.B5

#endif /* RSKRX671_H */

```

The program in the source project uses RSCI10 that is set to work as a UART to communicate with the terminal on the PC which is running terminal emulation. After confirming connection with the terminal, the program executes the R_RSCI_Receive function to wait for the reception of 1 byte of data from the terminal. After reception of this data from the terminal, the R_RSCI_Send function is used to send the character string indicating the version number of the RSCI module to the terminal. The R_RSCI_Receive function for waiting for reception and the R_RSCI_Send function for transmission are then repeatedly executed to handle loopback operation of receiving data from the terminal and returning the data to the terminal. The RSCI module of FIT uses the BYTEQ module of FIT to allocate a transmission buffer and a reception buffer.

6.2.3 Creating the Destination Project for Porting

Use the procedure described in Chapter 5, Steps in Porting a Pre-FSP Project to an FSP Project, to create the destination project for porting. The new FSP project created with reference to the text from section 5.1, Creating a New FSP Project to section 5.4, Porting Modules, is used as the base project.

6.2.4 Adding an SCI Module

Use the following procedure to add an SCI module for one channel to the project.

From the [New Stack] button, select [Connectivity] → [UART (r_sci_b_uart)].

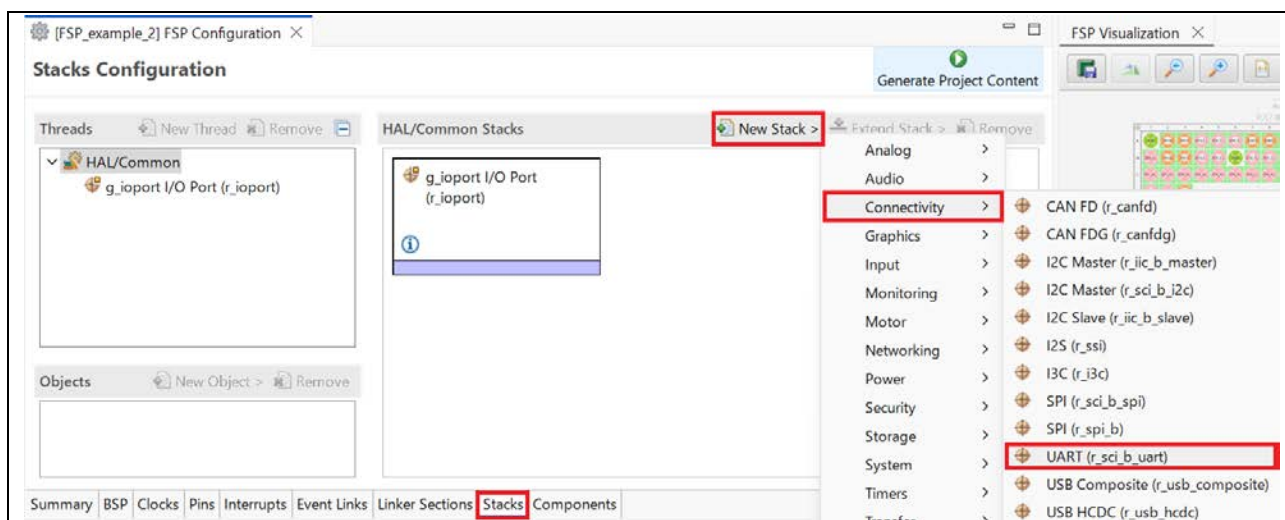


Figure 6-9 Selecting UART (SCI) on the [Stacks Configuration] Page

"g_uart0 (r_sci_b_uart)" for the SCI will be added to the [Stacks Configuration] page.

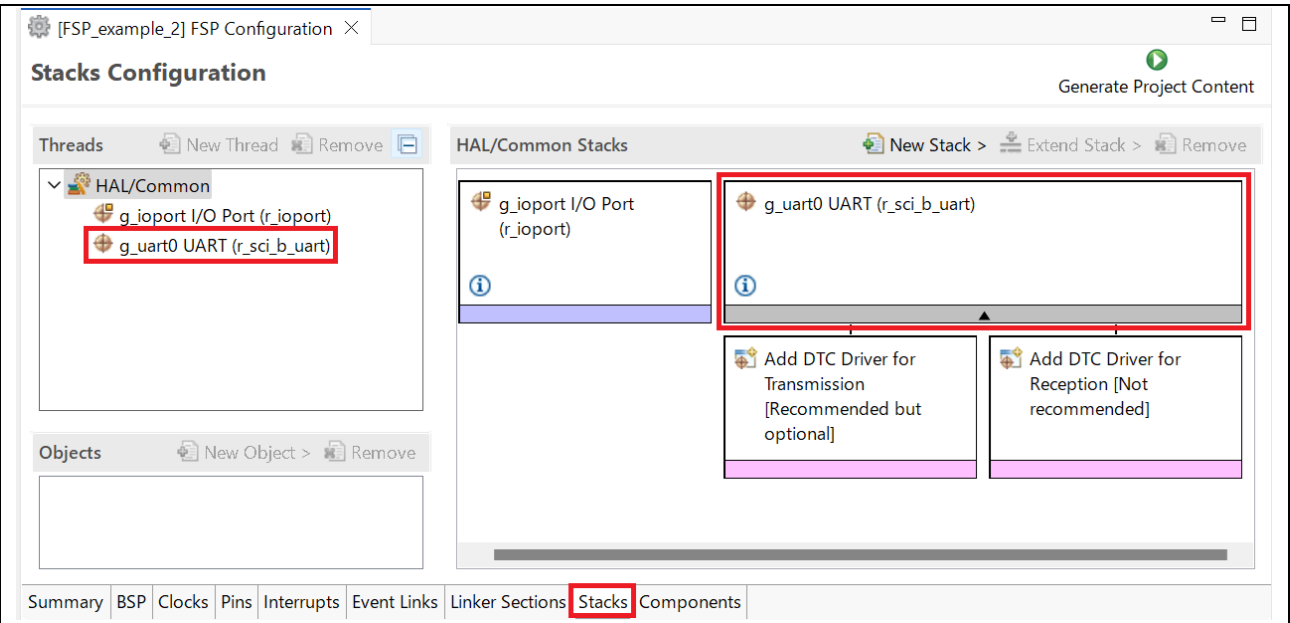


Figure 6-10 UART (SCI) Module (r_sci_b_uart) Added to the [Stacks Configuration] Page

Set the properties of the SCI module to match the RSCI component settings and R_RSCI_Open function settings (115200 bps, 8-bit data, no parity, 1 stop bit, and no flow control) in the source project.

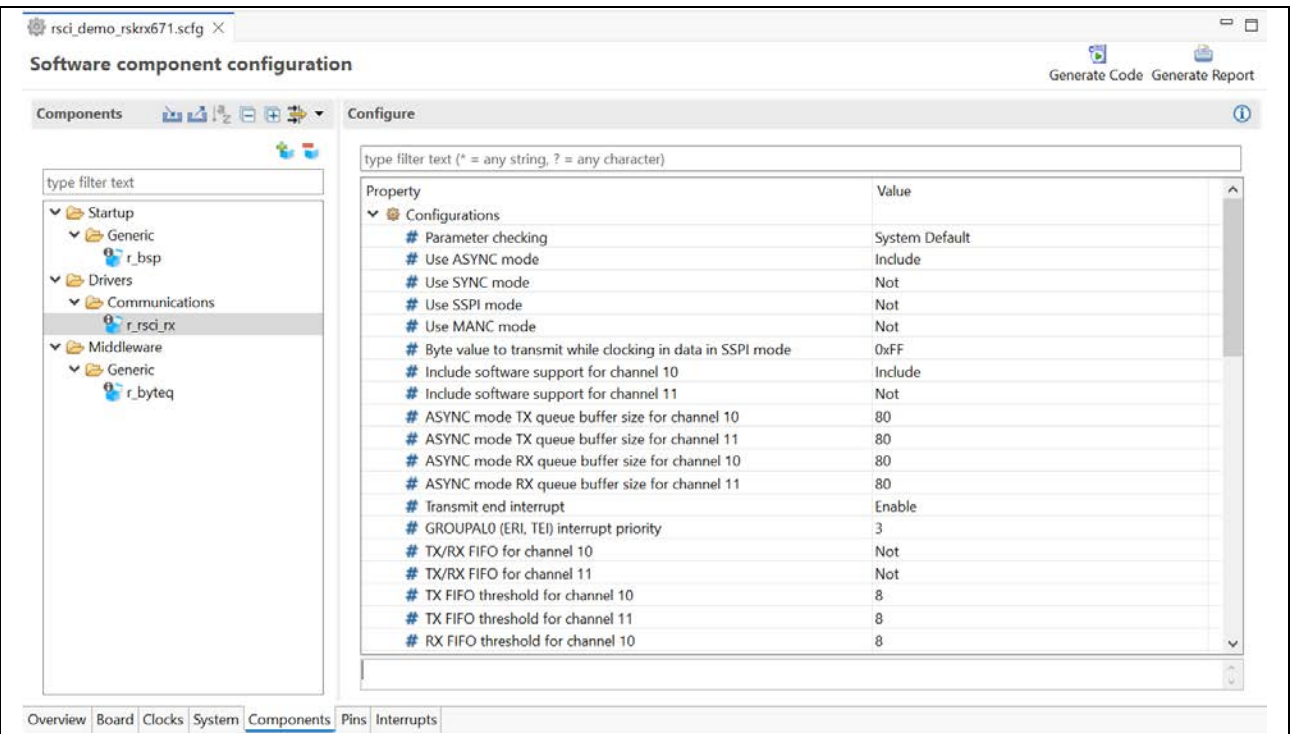


Figure 6-11 r_sci_rx Properties on the [Components] Page

Set the properties of g_uart0 as follows.

Table 6.1 Items and Values of Settings for Porting (1)

No.	Item and Value that were Set in the RSCI Component	Item and Value that Should be Set as g_uart0 Properties
1	"Parameter checking" is set to "System Default".	Set "Parameter checking" to "Default (BSP)".
2	"Use ASYNC mode" is set to "Include".	None
3	"Use SYNC mode" is set to "Not".	None
4	"Use SSPI mode" is set to "Not".	None
5	"Use MANC mode" is set to "Not".	None
6	"Byte value to transmit while clocking in data in SSPI mode" is set to "0xFF".	None
7	"Include software support for channel 10" is set to "Include".	Set "General" > "Channel" to "12". This change is necessary because the target device and board differ from those for the source project.
8	"Include software support for channel 11" is set to "Not".	None: This setting is not necessary because it is not used.
9	"ASYNC mode TX queue buffer size for channel 10" is set to "80".	None
10	"ASYNC mode TX queue buffer size for channel 11" is set to "80".	None
11	"ASYNC mode RX queue buffer size for channel 10" is set to "80".	None
12	"ASYNC mode RX queue buffer size for channel 11" is set to "80".	None
13	"Transmit end interrupt" is set to "Enable".	None
14	"GROUPAL0 (ERI, TEI) interrupt priority" is set to "3".	Set "Interrupts" > "Transmit End Interrupt Priority" to "3". Set "Interrupts" > "Error Interrupt Priority" to "3".
15	"TX/RX FIFO for channel 10" is set to "Not".	Set "Common" > "FIFO Support" to "Disable".
16	"TX/RX FIFO for channel 11" is set to "Not".	None: This setting is not necessary because it is not used.
17	"TX FIFO threshold for channel 10" is set to "8".	None
18	"TX FIFO threshold for channel 11" is set to "8".	None
19	"RX FIFO threshold for channel 10" is set to "8".	None
20	"RX FIFO threshold for channel 11" is set to "8".	None
21	"Received data match function for channel 10" is set to "Not".	None
22	"Received data match function for channel 11" is set to "Not".	None
23	"Receive data sampling timing adjustment for channel 10" is set to "Disabled".	None
24	"Transmit signal transition timing adjustment for channel 10" is set to "Disabled".	None

Table 6.2 Items and Values of Settings for Porting (2)

No.	Item and Value that were Set as p_cfg for the R_RSCI_Open Function	Item and Value that Should be Set as g_uart0 Properties
1	my_rsci_config.async.baud_rate = 115200	Set "Baud" > "Baud Rate" to "115200".
2	my_rsci_config.async.clk_src = RSCI_CLK_INT	Set "Extra" > "Clock source" to "Internal Clock".
3	my_rsci_config.async.data_size = RSCI_DATA_8BIT	Set "General" > "Data Bits" to "8bits".
4	my_rsci_config.async.parity_type = RSCI_NONE_PARITY	Set "General" > "Parity" to "None".
5	my_rsci_config.async.msb_first = false	None
6	my_rsci_config.async.invert_data = false	None
7	my_rsci_config.async.stop_bits = RSCI_STOPBITS_1	Set "General" > "Stop Bits" to "1bit".
8	my_rsci_config.async.int_priority = 3	Set "Interrupts" > "Transmit Data Empty Interrupt Priority" to "3".
		Set "Interrupts" > "Receive Interrupt Priority" to "3".

The following shows the properties of g_uart0 after setting the above values and using the same callback function name as before porting.

Properties Problems Smart Browser

g_uart0 UART (r_sci_b_uart)

Settings	Property	Value
	Module g_uart0 UART (r_sci_b_uart)	
	General	
	Name	g_uart0
	Channel	12
	Data Bits	8bits
	Parity	None
	Stop Bits	1bit
	Baud	
	Baud Rate	115200
	Baud Rate Modulation	Disabled
	Max Error (%)	5
	Flow Control	
	Extra	
	Interrupts	
	Callback	my_rsci_callback
	Receive Interrupt Priority	Priority 3
	Transmit Data Empty Interrupt Priority	Priority 3
	Transmit End Interrupt Priority	Priority 3
	Error Interrupt Priority	Priority 3

Figure 6-12 Properties of the UART (SCI) Module (r_sci_b_uart)

According to the settings of the callback function and interrupt priority levels, the information on the use of RSCI12 interrupts is registered as shown below. The names of the interrupt functions contained in the module are used without change.

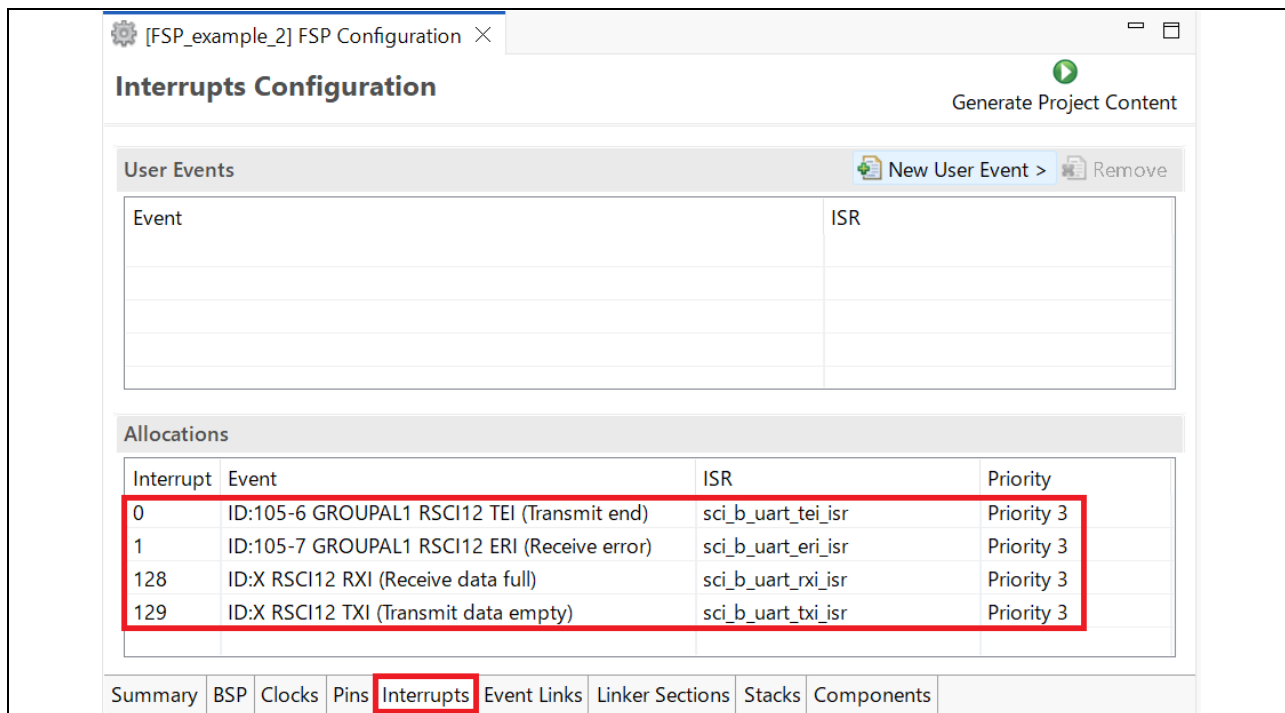


Figure 6-13 Registered UART (SCI) Interrupts on the [Interrupts Configuration] Page

Execute [Generate Project Content] to generate the definitions of the interrupt functions and output the declarations of the interrupt functions to vector_data.h. The following screenshot shows vector_data.h after the interrupt functions have been output with the above configuration.

rx_gen\vector_data.h

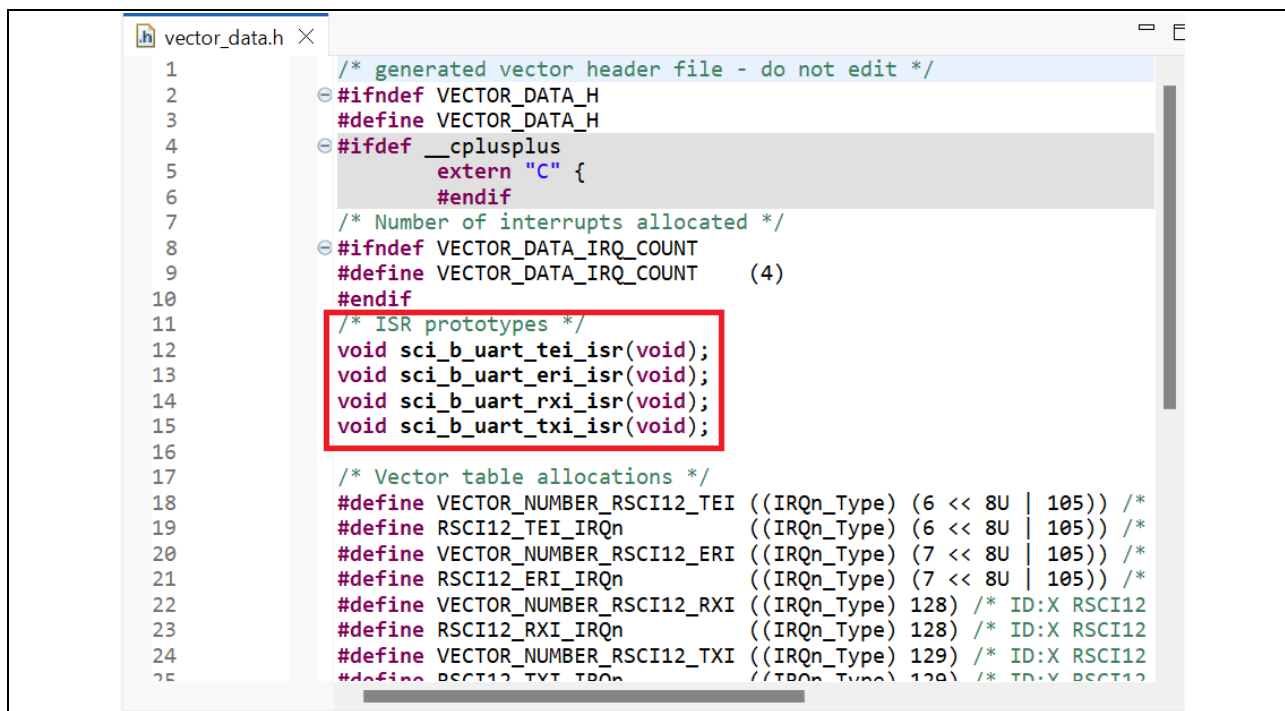
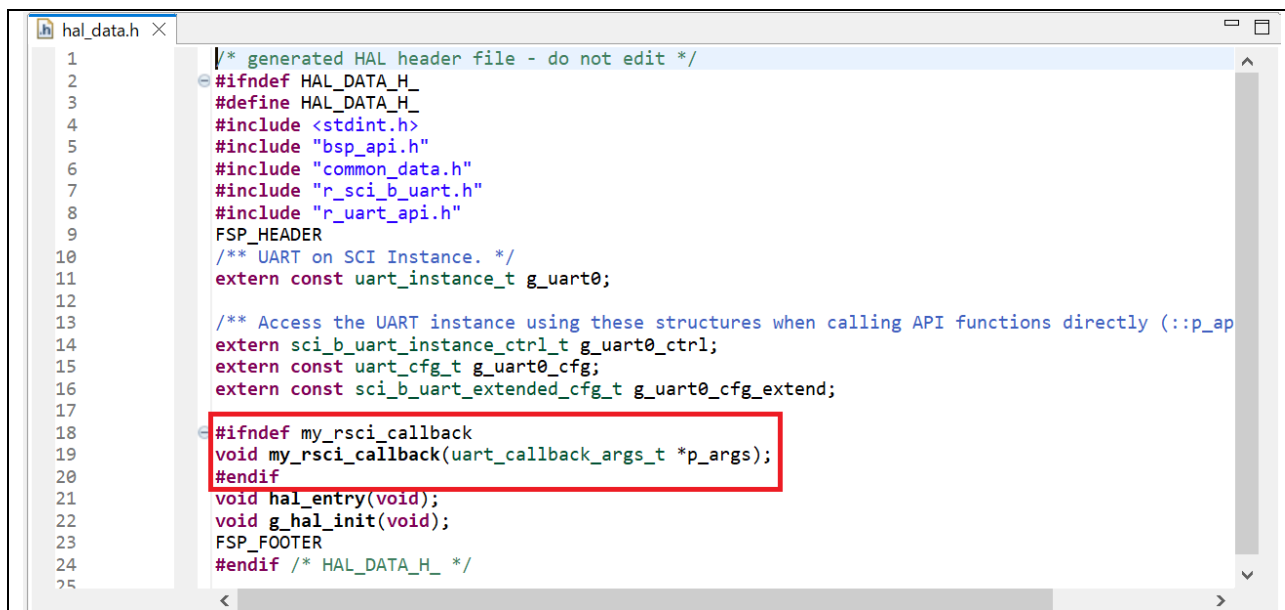


Figure 6-14 Declarations of the UART (SCI) Interrupt Functions in vector_data.h

The declaration of the callback function is output to hal_data.h.

rx_gen\hal_data.h



```

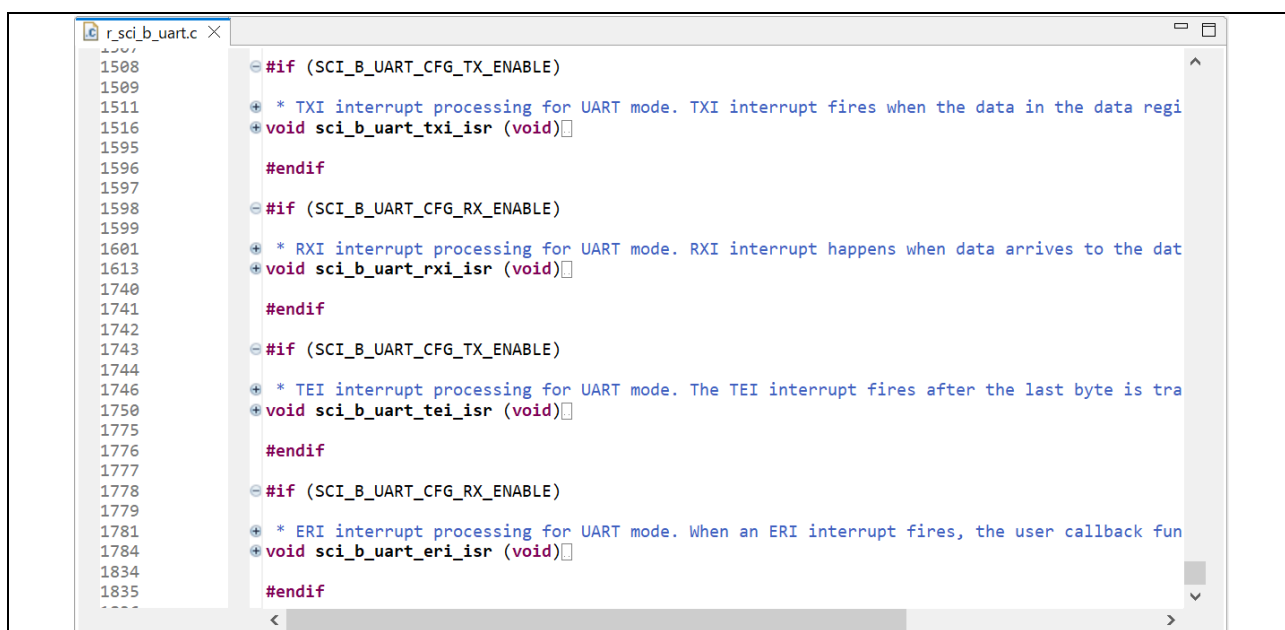
1  /* generated HAL header file - do not edit */
2  #ifndef HAL_DATA_H_
3  #define HAL_DATA_H_
4  #include <stdint.h>
5  #include "bsp_api.h"
6  #include "common_data.h"
7  #include "r_sci_b_uart.h"
8  #include "r_uart_api.h"
9  FSP_HEADER
10 /** UART on SCI Instance. */
11 extern const uart_instance_t g_uart0;
12
13 /** Access the UART instance using these structures when calling API functions directly (::p_ap
14 extern sci_b_uart_instance_ctrl_t g_uart0_ctrl;
15 extern const uart_cfg_t g_uart0_cfg;
16 extern const sci_b_uart_extended_cfg_t g_uart0_cfg_extend;
17
18 #ifndef my_rsci_callback
19 void my_rsci_callback(uart_callback_args_t *p_args);
20 #endif
21 void hal_entry(void);
22 void g_hal_init(void);
23 FSP_FOOTER
24 #endif /* HAL_DATA_H_ */
25

```

Figure 6-15 Declaration of the UART (SCI) Callback Function in hal_data.h

The bodies of the interrupt functions are defined in the module.

rx\fsp\src\r_sci_b_uart\r_sci_b_uart.c



```

1507
1508 #if (SCI_B_UART_CFG_TX_ENABLE)
1509
1511 * TXI interrupt processing for UART mode. TXI interrupt fires when the data in the data regi
1516 void sci_b_uart_txi_isr (void)
1595
1596 #endif
1597
1598 #if (SCI_B_UART_CFG_RX_ENABLE)
1599
1601 * RXI interrupt processing for UART mode. RXI interrupt happens when data arrives to the dat
1613 void sci_b_uart_rxi_isr (void)
1740
1741 #endif
1742
1743 #if (SCI_B_UART_CFG_TX_ENABLE)
1744
1746 * TEI interrupt processing for UART mode. The TEI interrupt fires after the last byte is tra
1750 void sci_b_uart_tei_isr (void)
1775
1776 #endif
1777
1778 #if (SCI_B_UART_CFG_RX_ENABLE)
1779
1781 * ERI interrupt processing for UART mode. When an ERI interrupt fires, the user callback fun
1784 void sci_b_uart_eri_isr (void)
1834
1835 #endif
1836

```

Figure 6-16 Interrupt Functions in r_sci_b_uart.c

For the callback function, however, only the declaration is output and the user has to create the body of the callback function.

6.2.5 Adding the Port Settings

The source project uses the initialization function to configure the pins related to the SCI. In the destination project, make settings on the [Pin Configuration] page in the [Properties] view of the SCI module so that initialization by the IOPORT module, to be called from the startup processing, is used for the pin settings.

Pins should be assigned to RXD and TXD under "Pins" in the [Properties] view of g_uart0. Note that if you select a board when creating the project, peripheral functions and pins may be assigned from the start based on the board information.

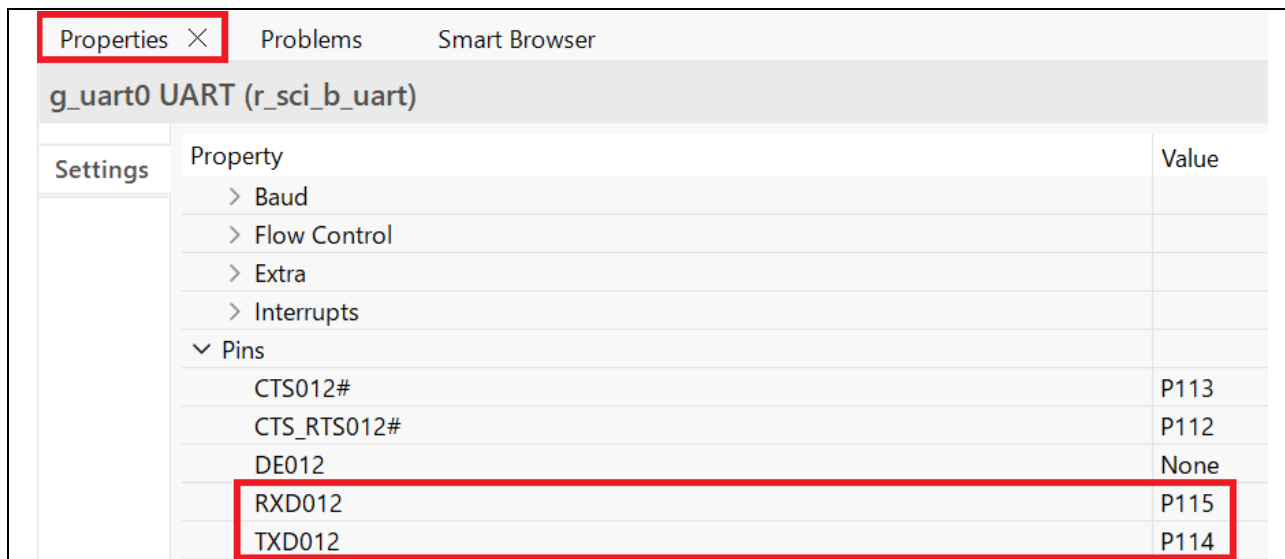


Figure 6-17 "Pins" in the [Properties] View of the UART (SCI) Module (r_sci_b_uart)

Click on the button for pin configuration to switch to the [Pin Configuration] page.



Figure 6-18 Button for Pin Configuration in the [Properties] View of the UART (SCI) Module (r_sci_b_uart)

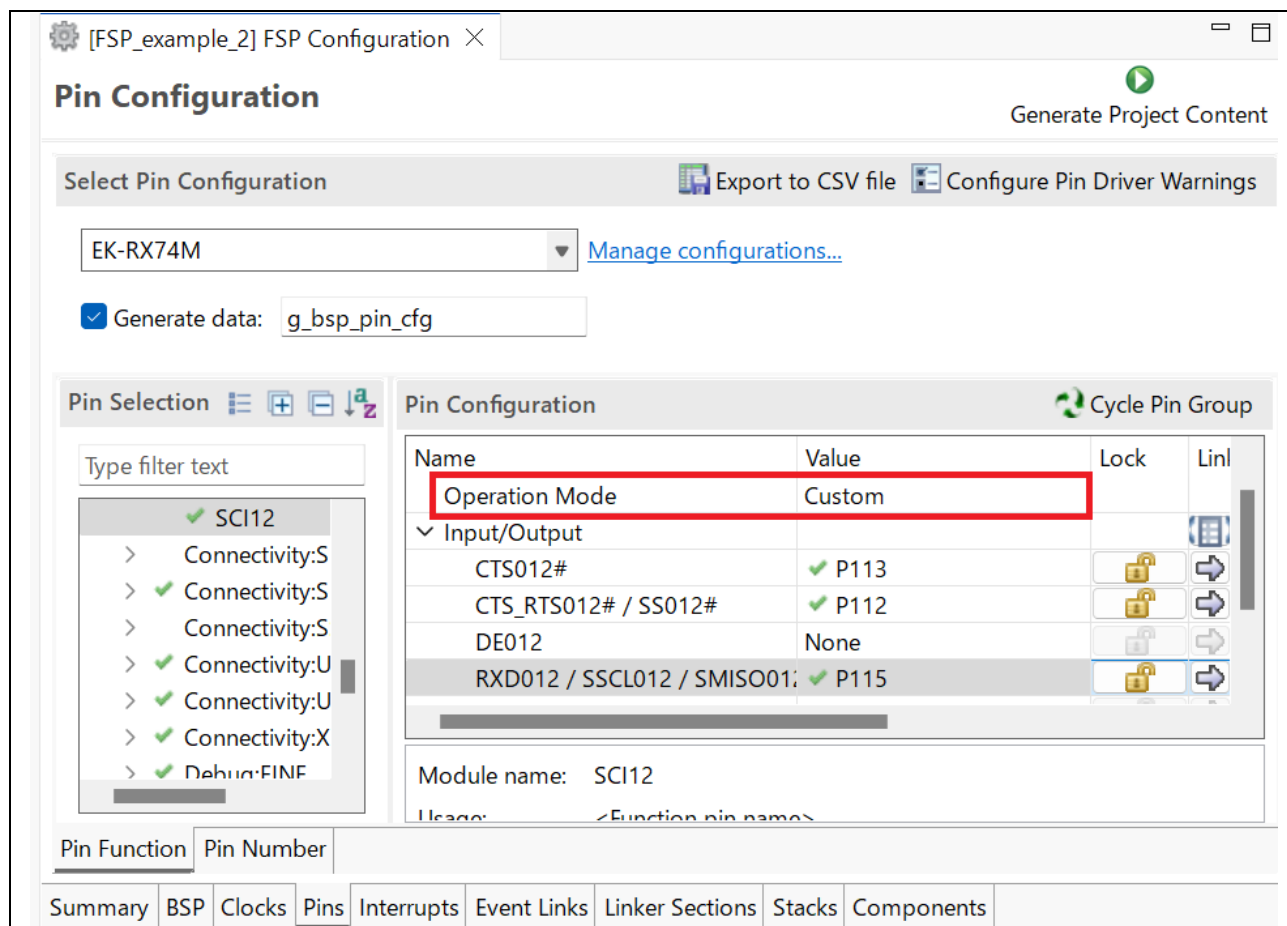


Figure 6-19 SCI12 Operation Mode Initial Settings on the [Pin Configuration] Page

Change "Operation Mode" to "Asynchronous UART" and the pins will be assigned to RXD and TXD.

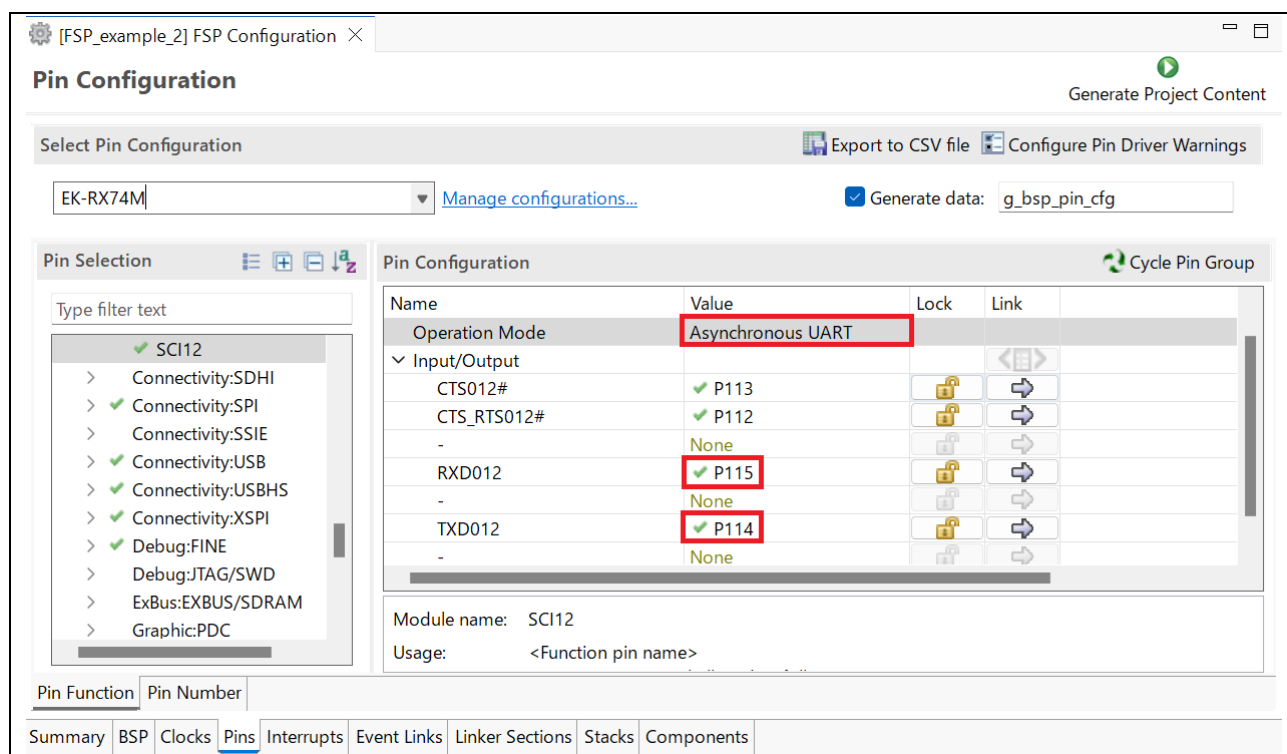


Figure 6-20 Changing the SCI12 Operation Mode to UART on the [Pin Configuration] Page

6.2.6 Structure of the Folders and Files in the Project after Porting and the Targets for Modification

The following shows the structure of the folders and files in the project after porting and the target files for modification. `hal_entry.c` and ported `rskrx671.h` are the targets for modification. For `iodefine.h`, add the file created in the new executable project.

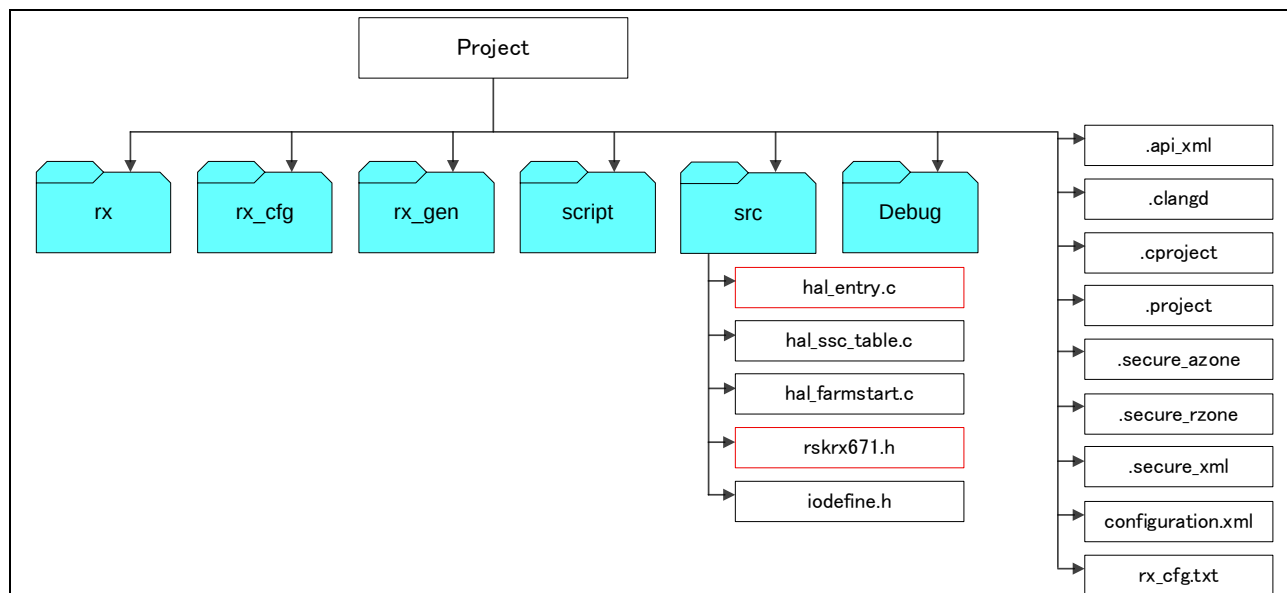


Figure 6-21 Structure of the Folders and Files in the Project after Porting and the Targets for Modification

6.2.7 Key Points in Porting

Use the procedure described in section 5.7, Porting the Source Code, to port the processing of the main function body and the SCI callback function in `main.c` to `hal_entry.c`. Also port `rskrx671.h`.

Some of the definitions in `rskrx671.h` use definitions provided by `iodefine.h`, so add `iodefine.h` by following the procedure described in section 5.8, Adding `iodefine.h`. Delete the initialization processing of the pins related to the SCI because this is now done by the IOPORT module. As the target device differs between the source and destination projects, some definitions to be used will require modification. Additionally, some definition names have been changed to avoid conflicts with definitions provided by the FSP.

Refer to section 5.9, Replacing Processing with FSP Module Processing, and replace the processing by the FIT module with the equivalent processing by the FSP.

6.2.8 Processing after Replacement of the Target Files for Modification

The following listing is of the final contents of `hal_entry.c` and `rskrx671.h` after modification.

`hal_entry.c` (The final results of addition are shown in red.)

```

#include "hal_data.h"
#include <stdio.h>          /* For sprintf */
#include "rskrx671.h"      /* Include user define */

uint8_t g_dest[32];
uint8_t g_src[32] = {"Ready"};
uint32_t g_transfer_complete = 0;
  
```

```

uint32_t g_receive_complete = 0;

static void rsci_rx671_init_ports(void);

FSP_CPP_HEADER
FSP_CPP_FOOTER

/*****
 * main() is generated by the RX Configuration editor and is used to generate
 * threads if an RTOS is used. This function is called by main() when no
 * RTOS is used.
 *****/
void hal_entry(void)
{
    /* TODO: add your own code here */
    LED0_PDR = 1;

    fsp_err_t err = FSP_SUCCESS;

    /* Called the R_SCI_B_UART_Open to open the transfer instance
       with initial configuration. */
    err = R_SCI_B_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
    assert(FSP_SUCCESS == err);

    /* Called the R_SCI_B_UART_Read to read data from UART. */
    err = R_SCI_B_UART_Read(&g_uart0_ctrl, g_dest, 1);
    assert(FSP_SUCCESS == err);

    /* Wait to get a character from the terminal so we know that it is ready. */
    while (!g_receive_complete)
    {
    }

    /* Called the R_SCI_B_UART_Write to write data to UART. */
    err = g_uart0.p_api->write(&g_uart0_ctrl, g_src, 12);
    assert(FSP_SUCCESS == err);

    while (!g_transfer_complete)
    {
    }
}

```

```

/* Repeat forever. */
while(1)
{
    g_receive_complete = 0;
    g_transfer_complete = 0;

    /* Read next character from the serial port receive queue.
     * Repeat read if needed until data is available in the queue. */
    /* Called the R_SCI_B_UART_Read to read data from UART. */
    err = g_uart0.p_api->read(&g_uart0_ctrl, g_dest, 1);
    assert(FSP_SUCCESS == err);

    while (!g_receive_complete)
    {
    }

    /* Echo 1 character of the received data back to the sender. */
    printf("%c", (char)(g_dest[0]));
    g_src[0] = g_dest[0];
    /* Called the R_SCI_B_UART_Write to write data to UART. */
    err = g_uart0.p_api->write(&g_uart0_ctrl, g_src, 1);
    assert(FSP_SUCCESS == err);

    while (!g_transfer_complete)
    {
    }
}

#if BSP_PZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

/*****
 * Function Name: my_rsci_callback
 * Description  : This is a template for an RSCI Async Mode callback function.
 * Arguments    : pArgs -
 *                 pointer to rsci_cb_args_t structure cast to a void. Structure
 *                 contains event and associated data.
 * Return Value : none
*****/

```

```

*****/
void my_rsci_callback(uart_callback_args_t *p_args)
{
    /* Handle the UART event */
    switch (p_args->event)
    {
        /* Received a character */
        case UART_EVENT_RX_CHAR:
        {
            LED0 = ~LED0;          // Toggle LED to demonstrate callback execution.
            break;
        }

        /* Receive complete */
        case UART_EVENT_RX_COMPLETE:
        {
            g_receive_complete = 1;
            break;
        }

        /* Transmit complete */
        case UART_EVENT_TX_COMPLETE:
        {
            g_transfer_complete = 1;
            break;
        }

        default:
        {
        }
    }
}

```

rskrx671.h (The final results of modification are shown in red.)

```

#include "iodefine.h"

#ifndef RSKRX671_H
#define RSKRX671_H

/* Local defines */

```

```

#define LED_ON          (0)
#define LED_OFF         (1)
#define SET_BIT_HIGH    (1)
#define SET_BIT_LOW     (0)
#define SET_BYTE_HIGH   (0xFF)
#define SET_BYTE_LOW    (0x00)

/* Switches */
#define SW_ACTIVE        0
#define SW1_PIDR         PORT9.PIDR.BIT.B1
#define SW2              PORT9.PIDR.BIT.B2
#define SW3              PORT0.PIDR.BIT.B7
#define SW1_PDR          PORT9.PDR.BIT.B1
#define SW2_PDR          PORT9.PDR.BIT.B2
#define SW3_PDR          PORT0.PDR.BIT.B7
#define SW1_PMR          PORT9.PMR.BIT.B1
#define SW2_PMR          PORT9.PMR.BIT.B2
#define SW3_PMR          PORT0.PMR.BIT.B7

/* LEDs */
#define LED0             PORTF.PODR.BIT.PODR06
#define LED1_PODR        PORTF.PODR.BIT.B5
#define LED2_PODR        PORT0.PODR.BIT.B3
#define LED3_PODR        PORT0.PODR.BIT.B5
#define LED0_PDR         PORTF.PDR.BIT.PDR06
#define LED1_PDR         PORTF.PDR.BIT.B5
#define LED2_PDR         PORT0.PDR.BIT.B3
#define LED3_PDR         PORT0.PDR.BIT.B5

#endif /* RSKRX671_H */

```

The program after porting uses RSCI2 that is set to work as a UART to communicate with the terminal on the PC which is running terminal emulation. After confirming connection with the terminal, the program executes the R_RSCI_Read function to wait for the reception of 1 byte of data from the terminal. After reception of this data from the terminal, the R_RSCI_Write function is used to send a specified character string to the terminal instead of the version number of the SCI module because version numbers are not assigned to the individual FSP modules. The R_RSCI_Read function for waiting for reception and the R_RSCI_Write function for transmission are then repeatedly executed to handle loopback operation of receiving data from the terminal and returning the data to the terminal.

6.2.9 Replacement of Processing by the FIT Module with Processing by the FSP Module in this Example of Porting

The following figures compare the processing by the FIT module before porting and the processing by the FSP module after porting and replacement.

6.2.9.1 Comparison of the Main Processing

Before porting (FIT)	After porting (FSP)
<pre> void main(void) { LED0_PDR = 1; rsci_cfg_t my_rsci_config; rsci_err_t my_rsci_err; uint32_t my_version; uint8_t version_str[12] = {0}; uint8_t my_char; /* Initialize the I/O port pins for communication on this RSCI channel. * This is specific to the MCU and ports chosen. For the RSKRX671 we will use the * RSCI channel connected to the USB serial port emulation. */ rsci_rx671_init_ports(); /* Set up the configuration data structure for asynchronous (UART) operation. */ my_rsci_config.async.baud_rate = 115200; my_rsci_config.async.clk_src = RSCI_CLK_INT; my_rsci_config.async.data_size = RSCI_DATA_8BIT; my_rsci_config.async.parity_type = RSCI_NONE_PARITY; my_rsci_config.async.msb_first = false; my_rsci_config.async.invert_data = false; my_rsci_config.async.stop_bits = RSCI_STOPBITS_1; my_rsci_config.async.int_priority = 3; /* 1lowest, 15highest */ /* OPEN ASYNC CHANNEL * Provide address of the config structure, * the callback function to be assigned, * and the location for the handle to be stored.*/ my_rsci_err = R_RSCI_Open(RSCI_CH10, RSCI_MODE_ASYNC, &my_rsci_config, my_rsci_callback, &g_my_rsci_handle); /* If there were an error this would demonstrate error detection of API calls. */ if (RSCI_SUCCESS != my_rsci_err) { R_BSP_NOP(); /* Your error handling code would go here. */ } } </pre>	<pre> void hal_entry(void) { /* 1000: add your own code here */ LED0_PDR = 1; fsp_err_t err = FSP_SUCCESS; /* Called the R_SCI_B_UART_Open to open the transfer instance * with initial configuration. */ err = g_uart0.p_api->open(&g_uart0_ctrl1, &g_uart0_cfg); assert(FSP_SUCCESS == err); } </pre>

Figure 6-22 Main Processing (1)

Before porting (FIT)	After porting (FSP)
<pre> /* Get module software version using the API function provided. */ my_version = R_RSCI_GetVersion(); /* Demonstrate parsing and formatting of version number. */ sprintf((char *)version_str, "RSCIV%1.0hu.%2.2hu\r\n", ((my_version >> 16)&0xf), (my_version & 0xff)); printf("Initializes the RSCI successfully\n%s", version_str); /* Wait to get a character from the terminal so we know that it is ready. */ do { my_rsci_err = R_RSCI_Receive(g_my_rsci_handle, &my_char, 1); } while (RSCI_ERR_INSUFFICIENT_DATA == my_rsci_err); /* Now send the version string out through the serial port to the terminal. */ R_RSCI_Send(g_my_rsci_handle, version_str, 12); /* Repeat forever. */ while(1) { /* Read next character from the serial port receive queue. * Repeat read if needed until data is available in the queue. */ do { my_rsci_err = R_RSCI_Receive(g_my_rsci_handle, &my_char, 1); } while (RSCI_ERR_INSUFFICIENT_DATA == my_rsci_err); /* Echo 1 character of the received data back to the sender. */ printf("%c", (char)my_char); R_RSCI_Send(g_my_rsci_handle, &my_char, 1); } /* End of function main. */ </pre>	<pre> /* Called the R_SCI_B_UART_Read to read data from UART. */ err = g_uart0.p_api->read(&g_uart0_ctrl1, g_dest, 1); assert(FSP_SUCCESS == err); /* Wait to get a character from the terminal so we know that it is ready. */ while (!g_receive_complete) { } /* Called the R_SCI_B_UART_Write to write data to UART. */ err = g_uart0.p_api->write(&g_uart0_ctrl1, g_src, 12); assert(FSP_SUCCESS == err); while (!g_transfer_complete) { } /* Repeat forever. */ while(1) { g_receive_complete = 0; g_transfer_complete = 0; /* Read next character from the serial port receive queue. * Repeat read if needed until data is available in the queue. */ /* Called the R_SCI_B_UART_Read to read data from UART. */ err = g_uart0.p_api->read(&g_uart0_ctrl1, g_dest, 1); assert(FSP_SUCCESS == err); while (!g_receive_complete) { } /* Echo 1 character of the received data back to the sender. */ printf("%c", (char)g_dest[0]); g_src[0] = g_dest[0]; /* Called the R_SCI_B_UART_Write to write data to UART. */ err = g_uart0.p_api->write(&g_uart0_ctrl1, g_src, 1); assert(FSP_SUCCESS == err); while (!g_transfer_complete) { } } #ifdef BSP_PZ_SECURE_BUILD /* Enter non-secure code */ R_BSP_NonSecureEnter(); #endif } </pre>

Figure 6-23 Main Processing (2)

6.2.9.2 Comparison of the Callback Function

Before porting (FIT)	After porting (FSP)
<pre> void my_rsci_callback(void *pArgs) { rsci_cb_args_t *args; args = (rsci_cb_args_t *)pArgs; if (args->event == RSCI_EVT_RX_CHAR) { /* From RXI interrupt; received character data is in args->byte */ LED0 = ~LED0; // Toggle LED to demonstrate callback execution. R_BSP_NOP(); } else if (args->event == RSCI_EVT_RXBUF_OVFL) { /* From RXI interrupt; rx queue is full; 'lost' data is in args->byte You will need to increase buffer size or reduce baud rate */ R_BSP_NOP(); } else if (args->event == RSCI_EVT_OVFL_ERR) { /* From receiver overflow error interrupt; error data is in args->byte Error condition is cleared in calling interrupt routine */ R_BSP_NOP(); } else if (args->event == RSCI_EVT_FRAMING_ERR) { /* From receiver framing error interrupt; error data is in args->byte Error condition is cleared in calling interrupt routine */ R_BSP_NOP(); } else if (args->event == RSCI_EVT_PARITY_ERR) { /* From receiver parity error interrupt; error data is in args->byte Error condition is cleared in calling interrupt routine */ R_BSP_NOP(); } } </pre>	<pre> void my_rsci_callback(uart_callback_args_t *p_args) { /* Handle the UART event */ switch (p_args->event) { /* Received a character */ case UART_EVENT_RX_CHAR: { LED0 = ~LED0; // Toggle LED to demonstrate callback execution. break; } /* Receive complete */ case UART_EVENT_RX_COMPLETE: { g_receive_complete = 1; break; } /* Transmit complete */ case UART_EVENT_TX_COMPLETE: { g_transfer_complete = 1; break; } default: { } } } </pre>

Figure 6-24 Callback Function

7. Examples of Porting Applications Created by the Code Generator to FSP Projects

This chapter takes two applications created by using the Code Generator as examples and describes the methods of porting them to FSP projects.

7.1 Example of Porting a Sample Program Using the CMT Component Created by the Code Generator

The following describes the procedure for porting a sample program that uses the functions of the CMT component created by the Code Generator to an FSP project. For the sample program, the FPB_RX140_Blink project (hereafter called the source project) in [RX140 Group Fast Prototyping Board for RX140 LED Blink Control Program](#) is used.

7.1.1 Structure of the Folders and Files in the Source Project and the Targets for Porting

The following shows the structure of the folders and files in the source project and the target files for porting. FPB_RX140_Blink_main.c, FPB_RX140_Blink_def.h, Config_CMT0_user.c, Config_ICU_user.c, and Config_PORT_user.c are the targets for porting.

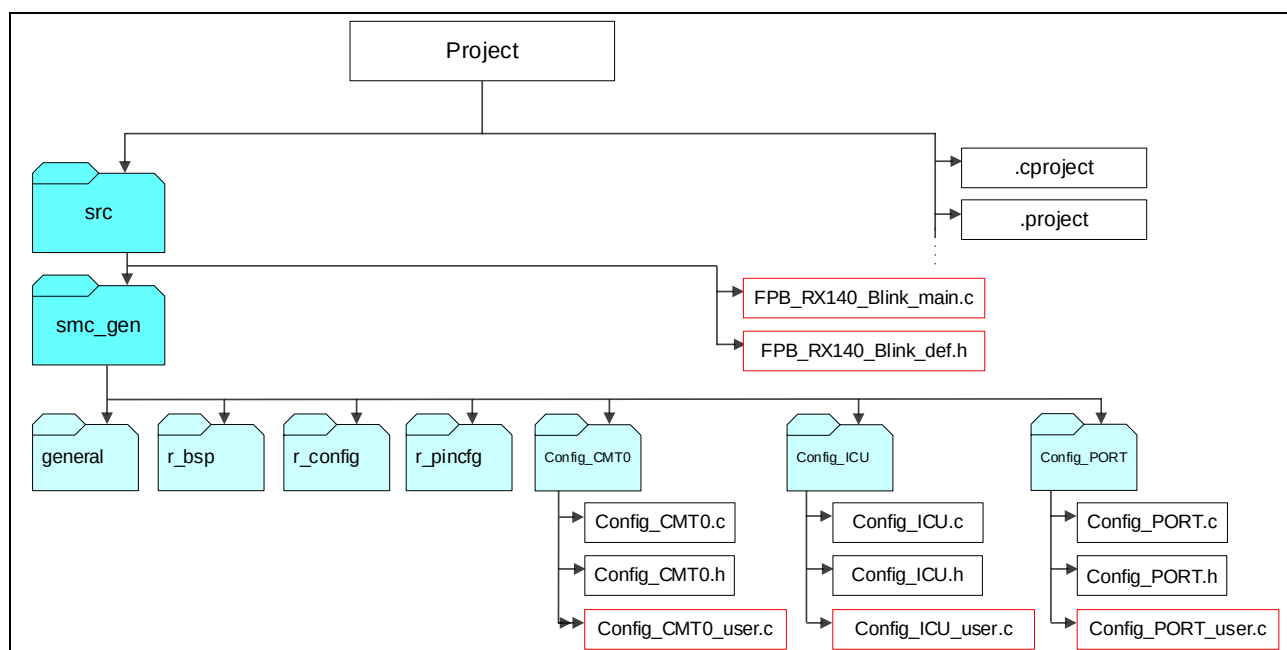


Figure 7-1 Structure of the Folders and Files in the Source Project and the Targets for Porting

7.1.2 Overview of the Processing in the Source Project

The following listing is of FPB_RX140_Blink_main.c in the source project.

src\FPB_RX140_Blink_main.c

```

/*****
Includes  <System Includes> , "Project Includes"
*****/

#include "r_smc_entry.h"
#include "FPB_RX140_Blink_def.h"

void main(void);

```

```

void main(void)
{
    LED1 = LED_OFF;
    LED2 = LED_OFF;

    if (SW1_PUSH == SW1) /*SW1_PUSH = Manual Mode, Other = Auto Mode*/
    {
        R_Config_ICU_IRQ0_Start();
    }
    else
    {
        R_Config_CMT0_Start();
    }
    while(1)
    {
        ; /* Blink LED0, LED1 Alternately by Interrupt */
    }
}

```

FPB_RX140_Blink_def.h is included as a related file. The following shows the contents of FPB_RX140_Blink_def.h.

src\FPB_RX140_Blink_def.h

```

/*****
Macro definitions
*****/

#ifndef FPB_RX140_BLINK_DEF_H_
#define FPB_RX140_BLINK_DEF_H_

/* General Values */
#define LED_ON      (1)
#define LED_OFF     (0)
#define SW1_PUSH    (0)
#define SW1_RELEASE (1)

/* Switches */
#define SW1          (PORT3.PIDR.BIT.B0)

/* LED port settings */
#define LED1          (PORT2.PODR.BIT.B0)
#define LED2          (PORT3.PODR.BIT.B2)

```

(Omitted)

```
#endif /* FPB_RX140_BLINK_DEF_H_ */
```

The CMT initialization processing is executed by the following function at the time of startup.

src\smc_gen\Config_CMT0\Config_CMT0.c

```
void R_Config_CMT0_Create(void)
{
    /* Disable CMI0 interrupt */
    IEN(CMT0, CMI0) = 0U;

    /* Cancel CMT stop state in LPC */
    MSTP(CMT0) = 0U;

    /* Set control registers */
    CMT0.CMCR.WORD = _0003_CMT_CMCR_CLOCK_PCLK512 | _0040_CMT_CMCR_CMIE_ENABLE |
    _0080_CMT_CMCR_DEFAULT;

    /* Set compare match register */
    CMT0.CMCOR = _7A11_CMT0_CMCOR_VALUE;

    /* Set CMI0 priority level */
    IPR(CMT0, CMI0) = _0F_CMT_PRIORITY_LEVEL15;

    R_Config_CMT0_Create_UserInit();
}
```

The ICU initialization processing is executed by the following function at the time of startup.

src\smc_gen\Config_ICU\Config_ICU.c

```
void R_Config_ICU_Create(void)
{
    /* Disable IRQ interrupts */
    ICU.IER[0x08].BYTE = 0x00U;

    /* Disable software interrupt */
    IEN(ICU, SWINT) = 0U;

    /* Disable IRQ digital filter */
    ICU.IRQFLTE0.BYTE &= ~(_01_ICU_IRQ0_FILTER_ENABLE);

    /* Set IRQ digital filter sampling clock */
```

```

ICU.IRQFLTC0.BIT.FCLKSEL0 = _0003_ICU_IRQ_FILTER_PCLK_64;

/* Set IRQ0 pin */
MPC.P30PFS.BYTE = 0x40U;
PORT3.PDR.BYTE &= 0xFEU;
PORT3.PMR.BYTE &= 0xFEU;

/* Set IRQ detection type */
ICU.IRQCR[0].BYTE = _04_ICU_IRQ_EDGE_FALLING;
IR(ICU, IRQ0) = 0U;

/* Enable IRQ digital filter */
ICU.IRQFLTE0.BYTE |= (_01_ICU_IRQ0_FILTER_ENABLE);

/* Set IRQ priority level */
IPR(ICU, IRQ0) = _0F_ICU_PRIORITY_LEVEL15;

R_Config_ICU_Create_UserInit();
}

```

The PORT initialization processing is executed by the following function at the time of startup.

src\smc_gen\Config_Port\Config_Port.c

```

void R_Config_PORT_Create(void)
{
    /* Set PORT2 registers */
    PORT2.PODR.BYTE = _00_Pm0_OUTPUT_0;
    PORT2.ODR0.BYTE = _00_Pm0_CMOS_OUTPUT | _00_Pm1_CMOS_OUTPUT;
    PORT2.ODR1.BYTE = _00_Pm6_CMOS_OUTPUT | _00_Pm7_CMOS_OUTPUT;
    PORT2.PMR.BYTE = _00_Pm0_PIN_GPIO;
    PORT2.PDR.BYTE = _01_Pm0_MODE_OUTPUT | _3C_PDR2_DEFAULT;

    /* Set PORT3 registers */
    PORT3.PODR.BYTE = _00_Pm2_OUTPUT_0;
    PORT3.ODR0.BYTE = _00_Pm0_CMOS_OUTPUT | _00_Pm1_CMOS_OUTPUT |
    _00_Pm2_CMOS_OUTPUT;
    PORT3.ODR1.BYTE = _00_Pm4_CMOS_OUTPUT | _00_Pm6_CMOS_OUTPUT |
    _00_Pm7_CMOS_OUTPUT;
    PORT3.PMR.BYTE = _00_Pm2_PIN_GPIO;
    PORT3.PDR.BYTE = _04_Pm2_MODE_OUTPUT | _08_PDR3_DEFAULT;

    R_Config_PORT_Create_UserInit();
}

```

}

In addition, the following related processing is implemented as the user processing for use with the code from the Code Generator.

src\smc_gen\Config_CMT0\Config_CMT0_user.c

```
#if FAST_INTERRUPT_VECTOR == VECT_CMT0_CMI0
#pragma interrupt r_Config_CMT0_cmi0_interrupt(vect=VECT(CMT0,CMI0),fint)
#else
#pragma interrupt r_Config_CMT0_cmi0_interrupt(vect=VECT(CMT0,CMI0))
#endif
static void r_Config_CMT0_cmi0_interrupt(void)
{
    /* Start user code for r_Config_CMT0_cmi0_interrupt.
       Do not edit comment generated here */
    LED1 = LED2;
    LED2 = ~LED2;
    /* End user code. Do not edit comment generated here */
}
```

src\smc_gen\Config_ICU\Config_ICU_user.c

```
#if FAST_INTERRUPT_VECTOR == VECT_ICU_IRQ0
#pragma interrupt r_Config_ICU_irq0_interrupt(vect=VECT(ICU,IRQ0),fint)
#else
#pragma interrupt r_Config_ICU_irq0_interrupt(vect=VECT(ICU,IRQ0))
#endif
static void r_Config_ICU_irq0_interrupt(void)
{
    /* Start user code for r_Config_ICU_irq0_interrupt.
       Do not edit comment generated here */
    LED1 = LED2;
    LED2 = ~LED2;
    /* End user code. Do not edit comment generated here */
}
```

There is no special initialization processing for the pin settings.

src\smc_gen\Config_Port\Config_Port_user.c

```
void R_Config_PORT_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    /* End user code. Do not edit comment generated here */
}
```

The program in the source project uses a switch and LEDs. After release from the reset state, input from the switch changes the operating mode of the sample program. As long as the switch is not pressed after a reset, the operating mode is set to Auto Mode, which causes the LEDs to blink at a frequency of 2 Hz. If the switch is pressed, the operating mode is set to Manual Mode which turns the LEDs on and off every time the switch is pressed. The Code Generator is used to create code for three functions: CMT, IRQ interrupts, and port control.

7.1.3 Creating the Destination Project for Porting

Use the procedure described in Chapter 5, Steps in Porting a Pre-FSP Project to an FSP Project, to create the destination project for porting. The new FSP project created with reference to the text from section 5.1, Creating a New FSP Project to section 5.4, Porting Modules is used as the base project.

There is no need to add a module for port control because the I/O port module (r_ioport_b) is already implemented in the FSP project. The CMT and IRQ require the addition of the corresponding FSP modules.

7.1.4 Adding a CMT Module

Use the following procedure to add a CMT module for one channel (CMT0) with property settings to match the CMT settings made by the Code Generator.

From the [New Stack] button, select [Timers] → [Timer, Compare Match Timer (r_cmt)] to add g_timer0 for CMT0 to the project.

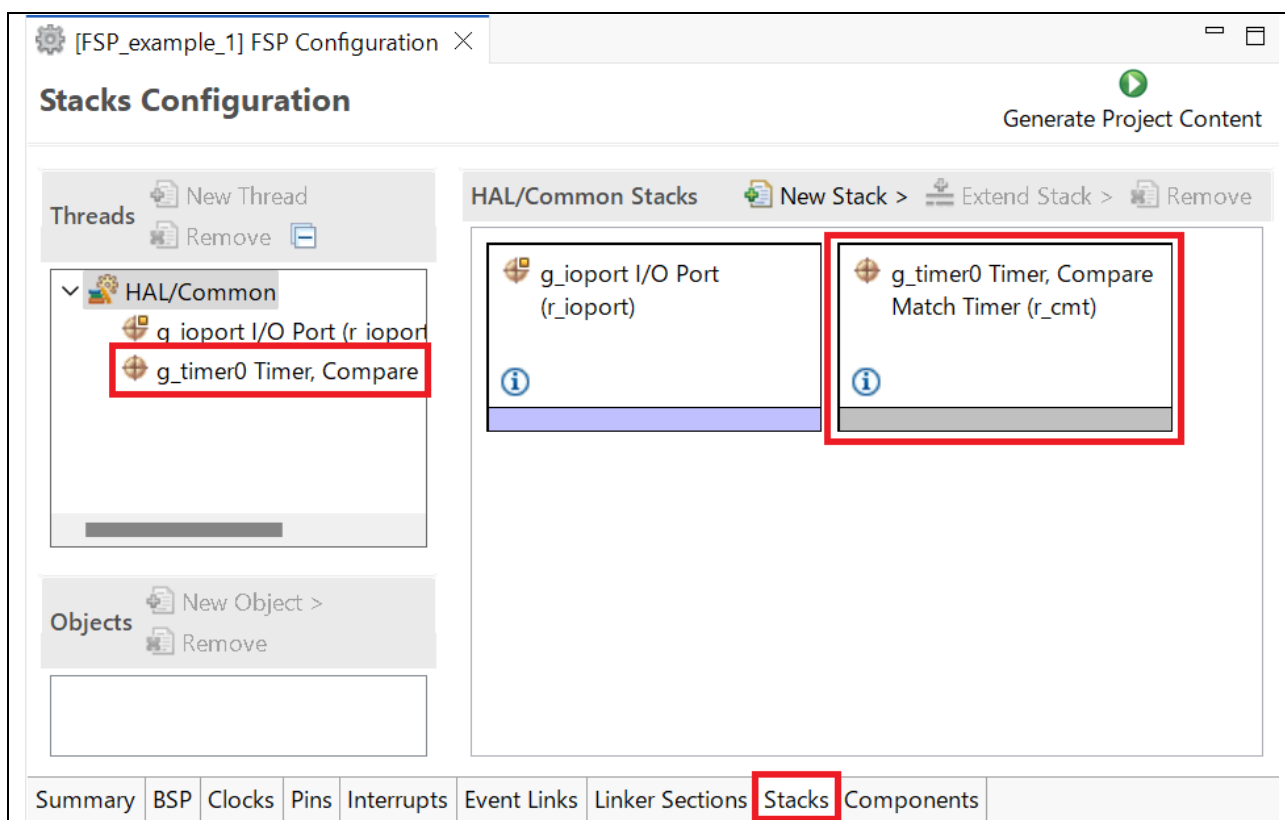


Figure 7-2 Adding a CMT Module (r_cmt) (CMT0) on the [Stacks Configuration] Page

Set the properties of g_timer0 for CMT0 to match the CMT settings made by the Code Generator.

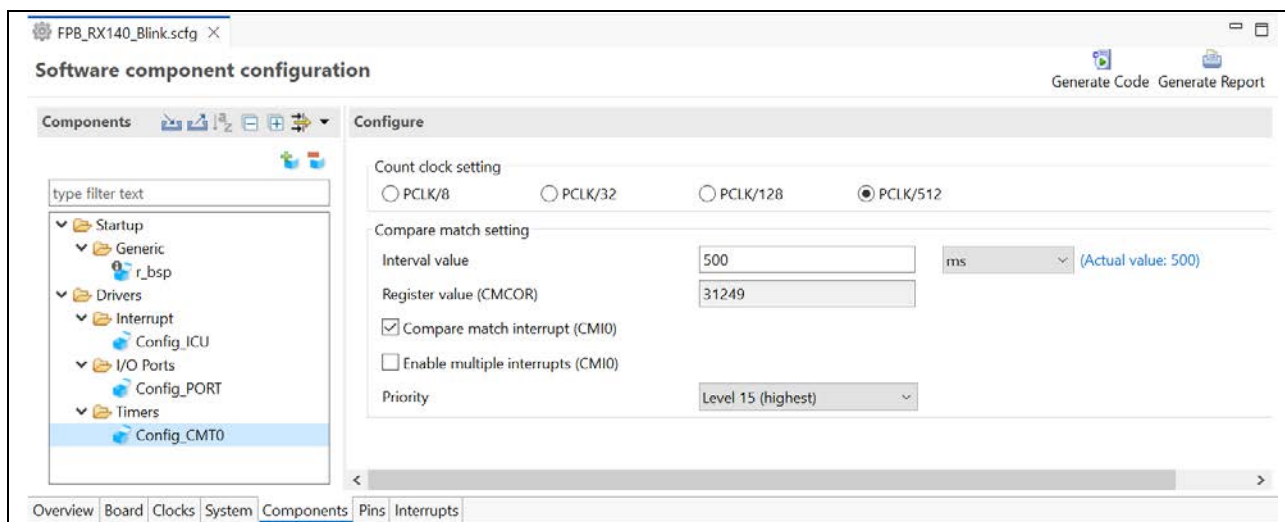


Figure 7-3 CMT Settings Made by the Code Generator on the [Components] Page

g_timer0 is used for CMT0. For the properties, set "Period" and "Period Unit" to "2" and "Hertz" and specify the callback function name and interrupt priority level to match those of the sample program.

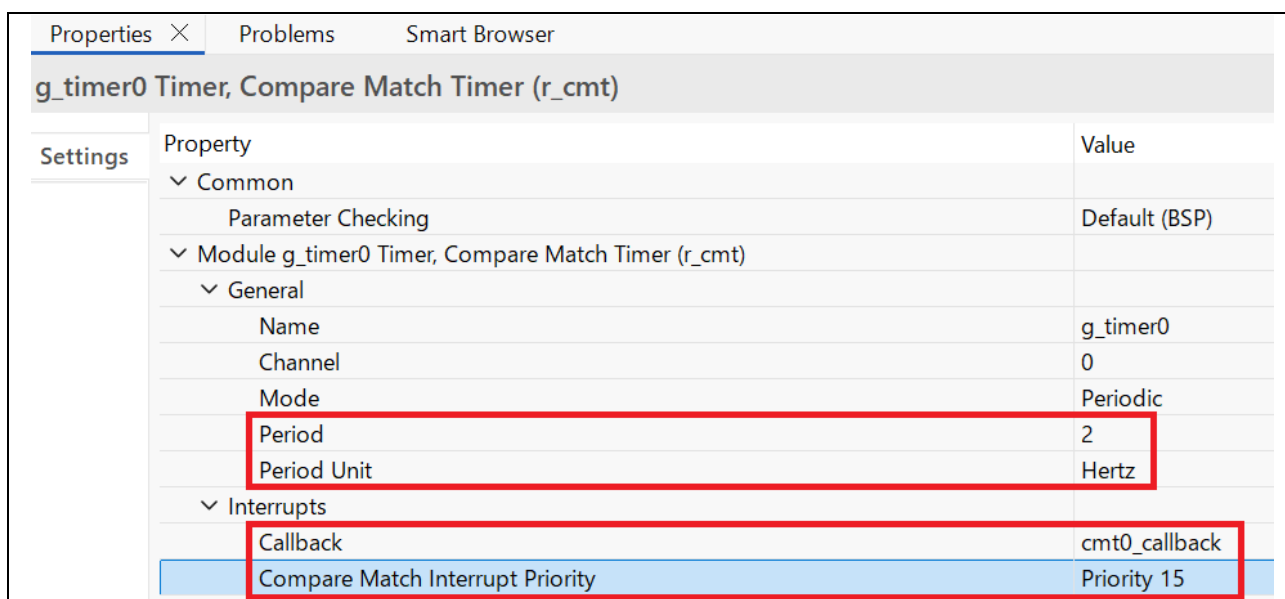


Figure 7-4 r_cmt g_timer0 (CMT0) Properties

According to the settings of the callback function and interrupt priority level, the information on the use of the CMT0 interrupt is registered as shown below. The name of the interrupt function contained in the module is used without change.

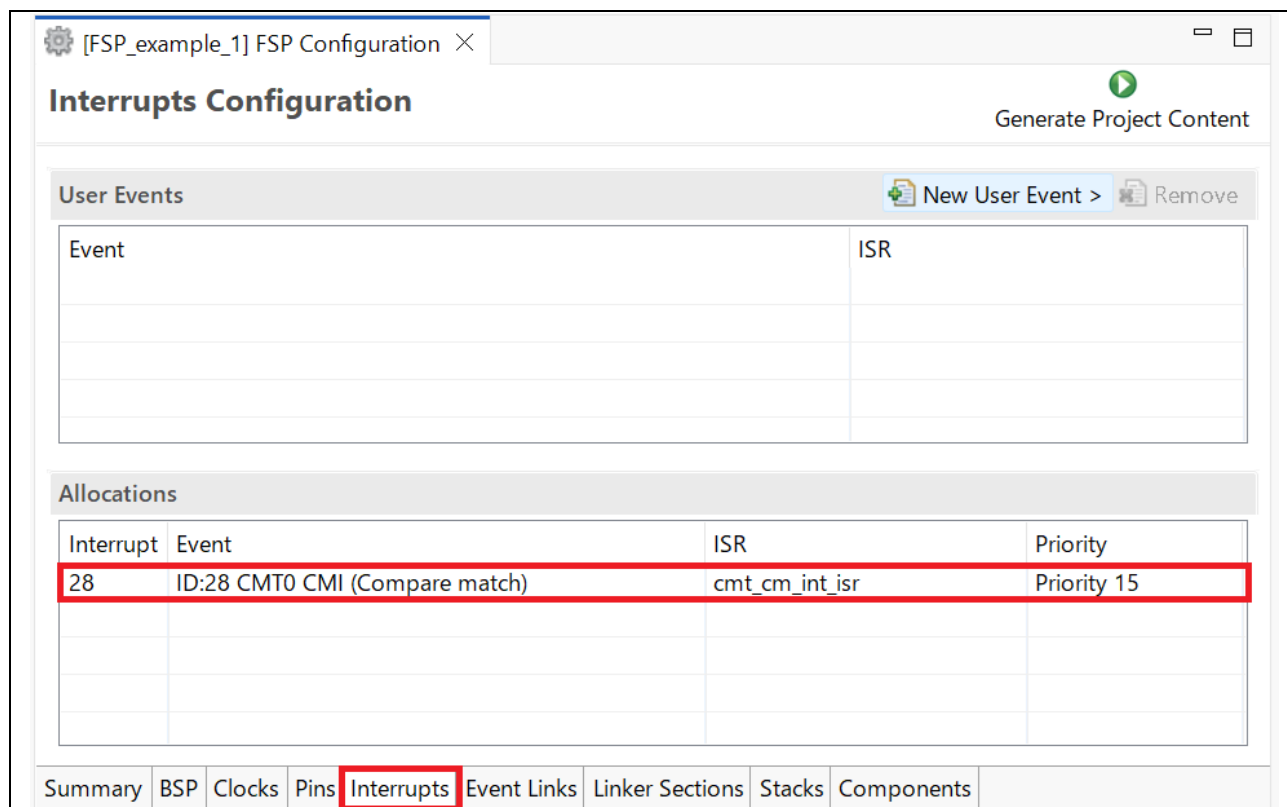


Figure 7-5 Registered CMT0 Interrupt on the [Interrupts Configuration] Page

7.1.5 Adding an ICU Module

Next, select [Input] → [External IRQ (r_icu)] from the [New Stack] button.

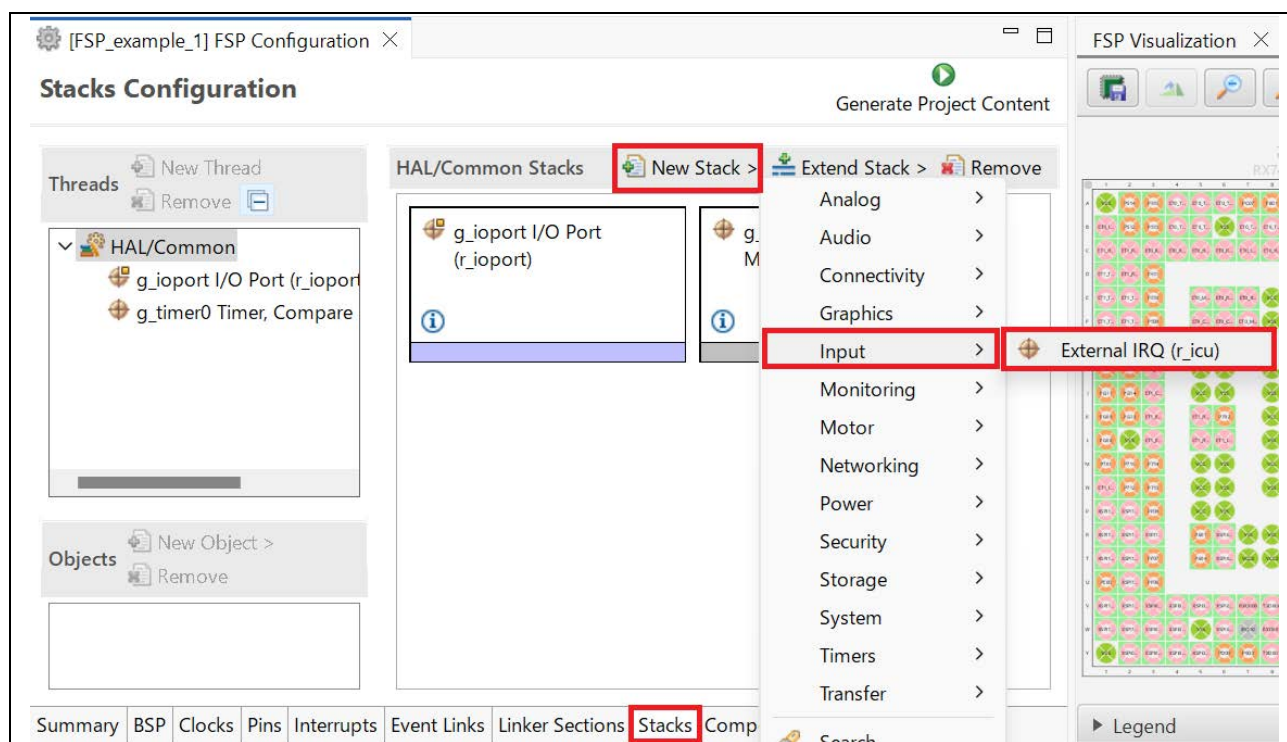


Figure 7-6 Selecting External IRQ on the [Stacks Configuration] Page

"g_external_irq0 (r_icu)" for the external IRQ will be added to the [Stacks Configuration] page.

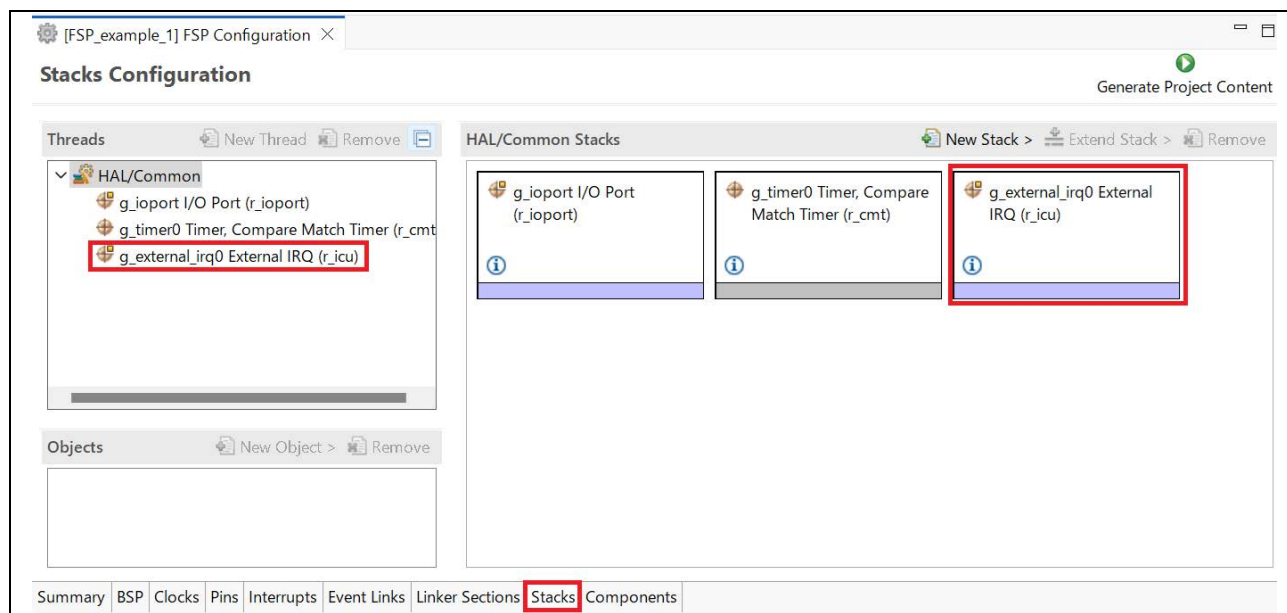


Figure 7-7 Adding an External IRQ Module (r_icu) on the [Stacks Configuration] Page

Set the properties of g_external_irq0 for IRQ0 to match the IRQ settings made by the Code Generator.

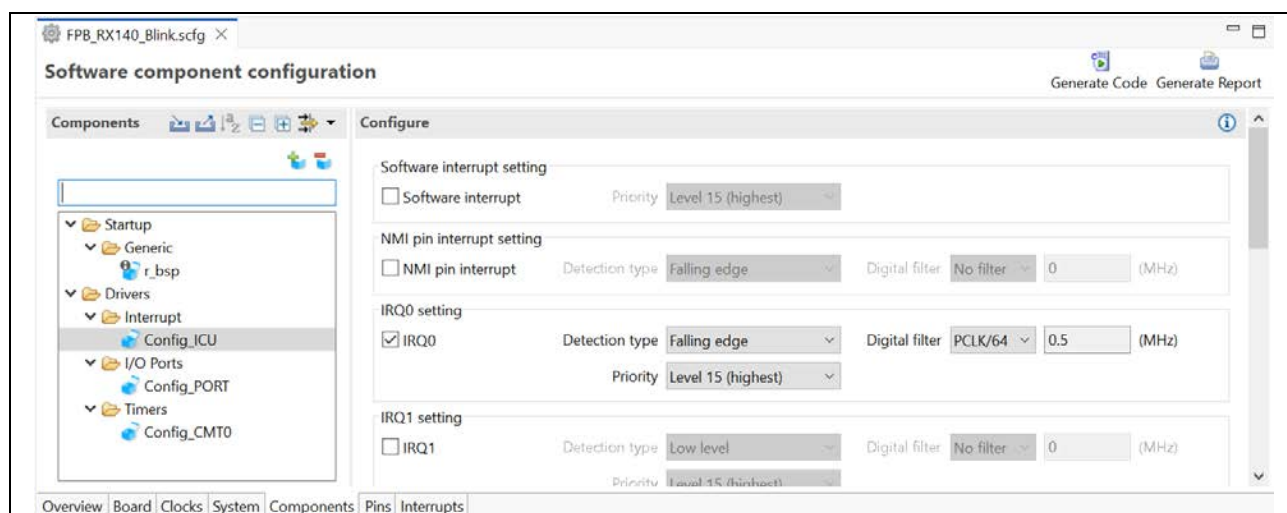


Figure 7-8 IRQ Settings Made by the Code Generator on the [Components] Page

Set the properties of g_external_irq0 as follows.

Table 7.1 Items and Values of Settings for Porting

No.	Item and Value that were Set in the IRQ Component	Item and Value that Should be Set as g_external_irq0 Properties
1	IRQ0 setting "IRQ0" is selected.	Set "Channel" to "30".
2	IRQ0 setting "Detection type" is set to "Falling edge".	Set "Trigger" to "Falling".
3	IRQ0 setting "Digital filter" is set to "PCLK/64".	Set "Digital Filtering" to "Enabled". Set "Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)" to "PCLK / 64".
4	IRQ0 setting "Priority" is set to "Level 15 (highest)".	Set "Pin Interrupt Priority" to "Priority 15".

The following shows the properties of g_external_irq0 after setting of the above values and the callback function name.

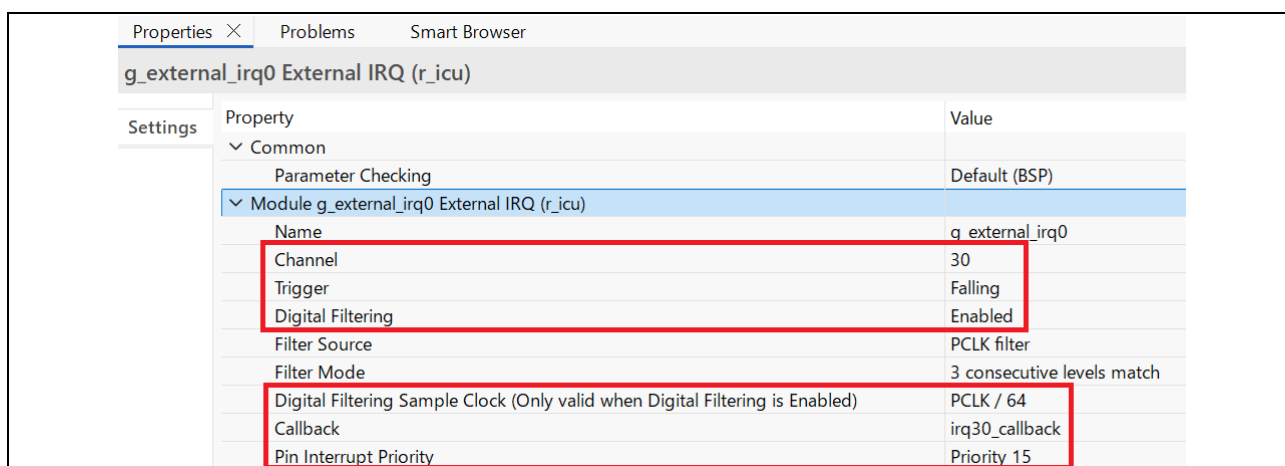


Figure 7-9 Properties of the ICU (IRQ) Module (r_icu)

According to the settings of the callback function and interrupt priority level of IRQ30, the information on the use of the IRQ interrupt is registered as shown below. The name of the interrupt function contained in the module is used without change.

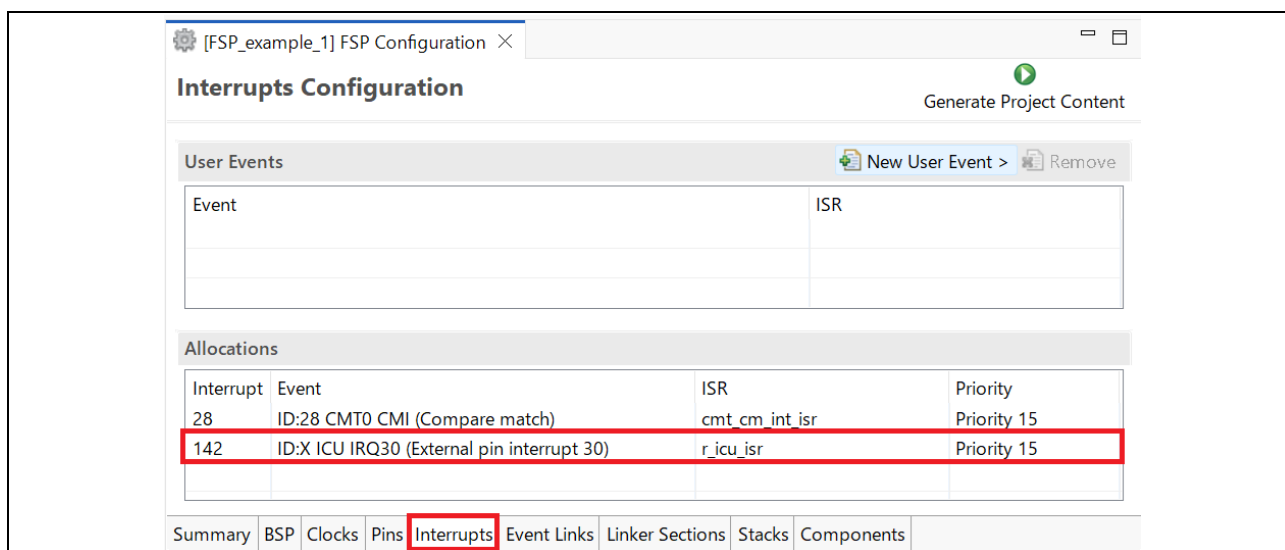


Figure 7-10 Registered IRQ30 Interrupt on the [Interrupts Configuration] Page

The following screenshot shows vector_data.h after the interrupt functions have been output with the above settings.

rx_gen\vector_data.h

```

1  /* generated vector header file - do not edit */
2  #ifndef VECTOR_DATA_H
3  #define VECTOR_DATA_H
4  #ifdef __cplusplus
5      extern "C" {
6      #endif
7  /* Number of interrupts allocated */
8  #ifndef VECTOR_DATA_IRQ_COUNT
9  #define VECTOR_DATA_IRQ_COUNT    (2)
10 #endif
11 /* ISR prototypes */
12 void cmt_cm_int_isr(void);
13 void r_icu_isr(void);
14
15 /* Vector table allocations */
16 #define VECTOR_NUMBER_CMT0_CMI ((IRQn_Type) 28) /* ID:28 CMT0 CMI
17 #define CMT0_CMI_IRQn          ((IRQn_Type) 28) /* ID:28 CMT0 CMI
18 #define VECTOR_NUMBER_ICU_IRQ30 ((IRQn_Type) 142) /* ID:X ICU IRQ30
19 #define ICU_IRQ30_IRQn         ((IRQn_Type) 142) /* ID:X ICU IRQ30
20
21

```

Figure 7-11 Declarations of the CMT and IRQ Interrupt Functions in vector_data.h

The declaration of the callback function of the CMT is output to hal_data.h.

rx_gen\hal_data.h

```

6  #include "common_data.h"
7  #include "r_cmt.h"
8  #include "r_timer_api.h"
9  FSP_HEADER
10 /** Timer on CMT Instance. */
11 extern const timer_instance_t g_timer0;
12
13 /** Access the CMT instance using these structures when calling API functions directly */
14 extern cmt_instance_ctrl_t g_timer0_ctrl;
15 extern const timer_cfg_t g_timer0_cfg;
16
17 #ifndef cmt0_callback
18 void cmt0_callback(timer_callback_args_t *p_args);
19 #endif
20 void hal_entry(void);
21 void g_hal_init(void);
22 FSP_FOOTER
23 #endif /* HAL_DATA_H */

```

Figure 7-12 Declaration of the CMT Callback Function in hal_data.h

The declaration of the callback function of IRQ30 is output to common_data.h.

rx_gen\common_data.h

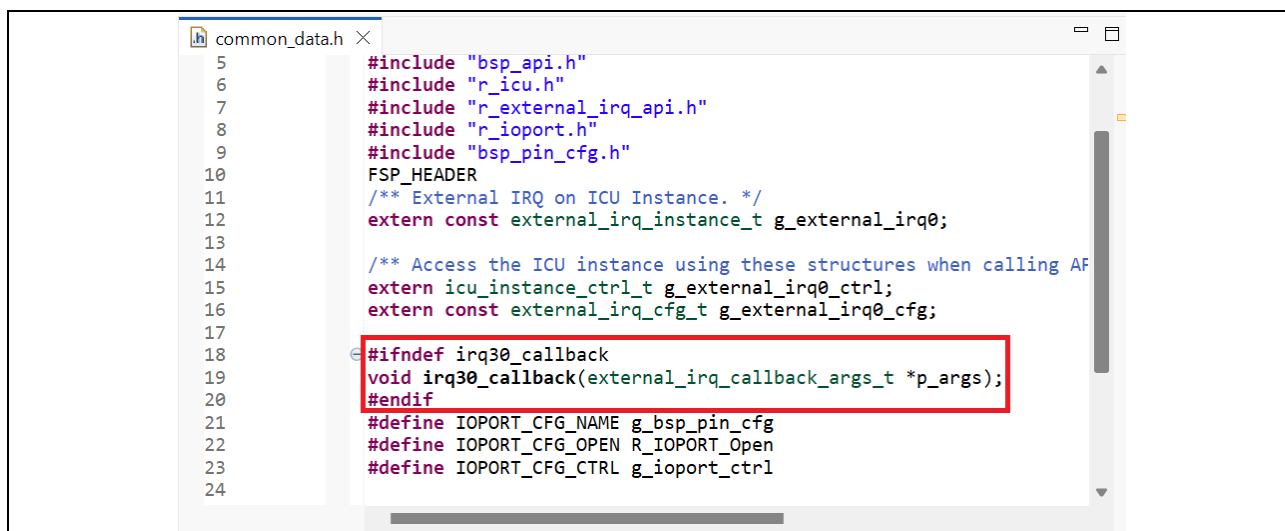


Figure 7-13 Declaration of the IRQ Callback Function in common_data.h

The bodies of the interrupt functions are defined in the modules.

rx\fsp\src\r_cmt\r_cmt.c

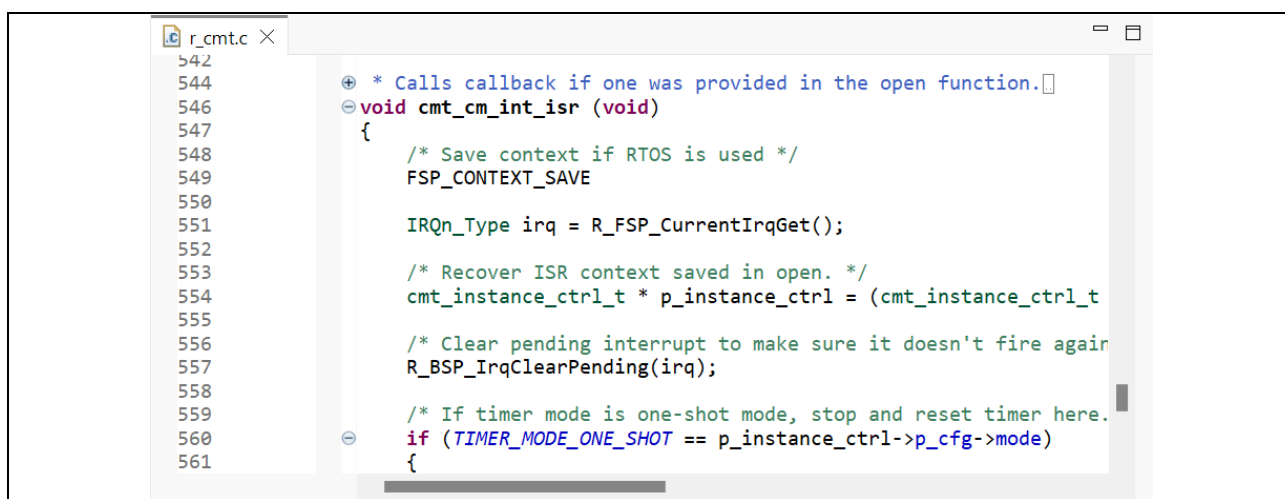


Figure 7-14 Interrupt Function in r_cmt.c

rx\fsp\src\r_icu\r_icu.c

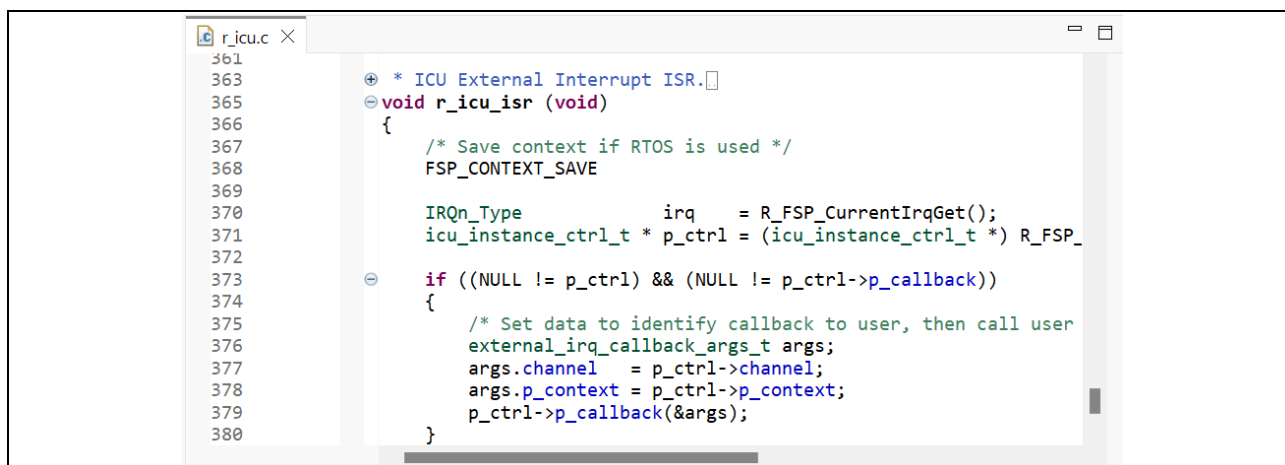


Figure 7-15 Interrupt Function in r_icu.c

For the callback functions, however, only the declarations are output, and the user has to create the bodies of the callback functions.

7.1.6 Adding the Port Settings

The source project uses Config_Port to configure the output pins for use with LEDs (the pin for use with the switch can be used in its initial state, so the pin is not initialized by Config_Port). In the destination project, make settings on the [Pin Configuration] page so that initialization by the IOPORT module to be called from the startup processing is used for the pin settings.

P20 and P32 were configured in Config_Port of the source project.

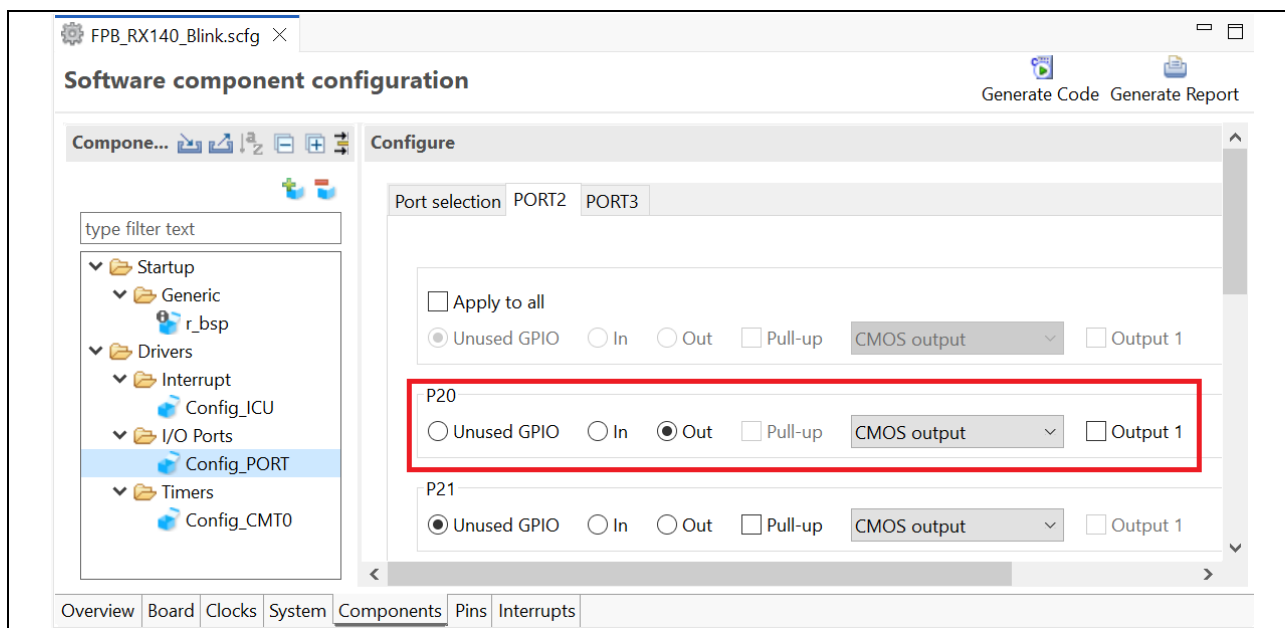


Figure 7-16 Port2 Configuration in Config_Port

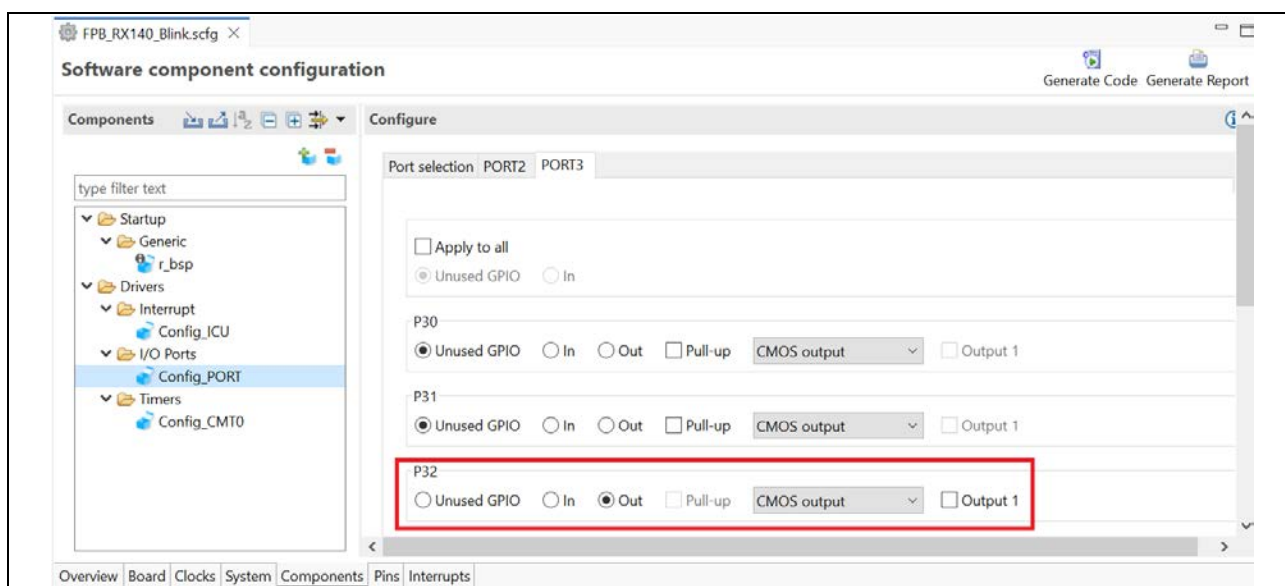


Figure 7-17 Port3 Configuration in Config_Port

Assume that PF05 and P807 are to be used in the destination project because the device and board will change in the destination project..Configure them on the [Pin Configuration] page. In EK-RX74M, PF05 and P807 are configured by default for LEDs.

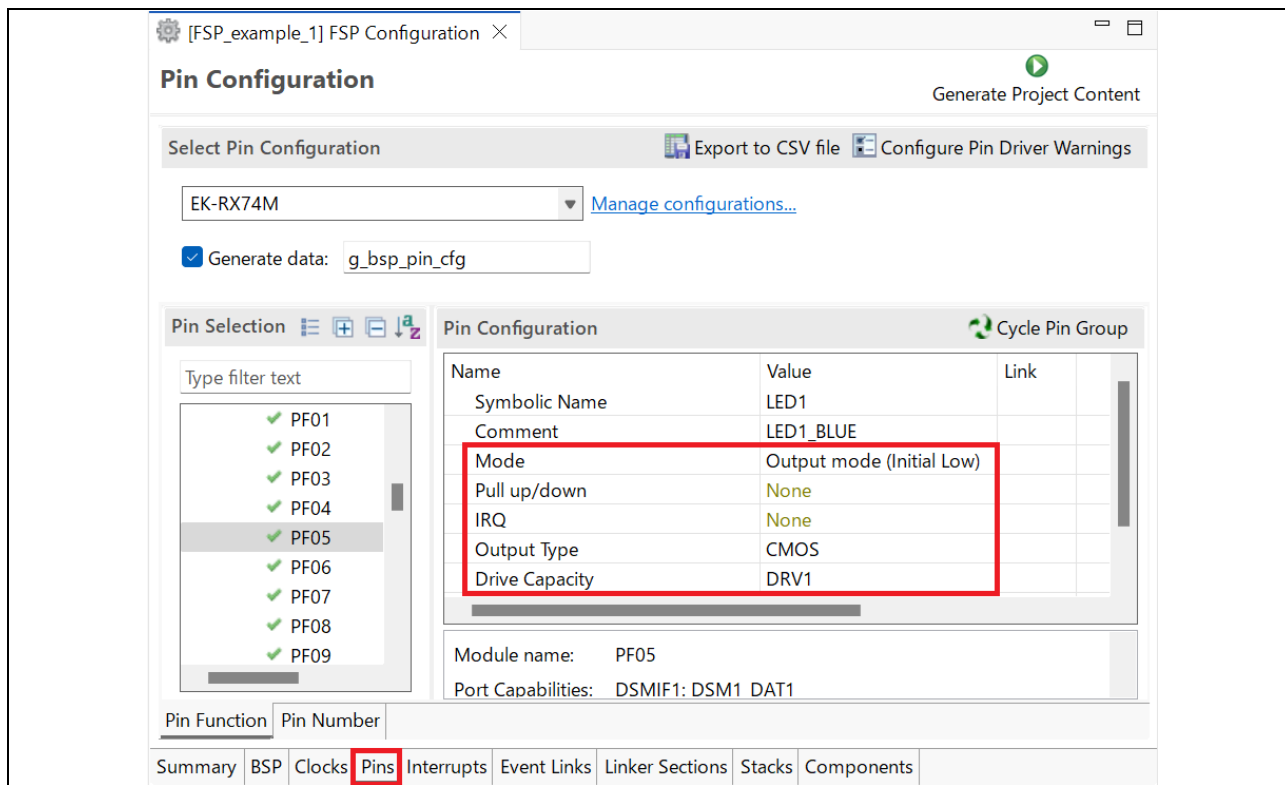


Figure 7-18 Changing the PF05 Mode to Output Mode on the [Pin Configuration] Page

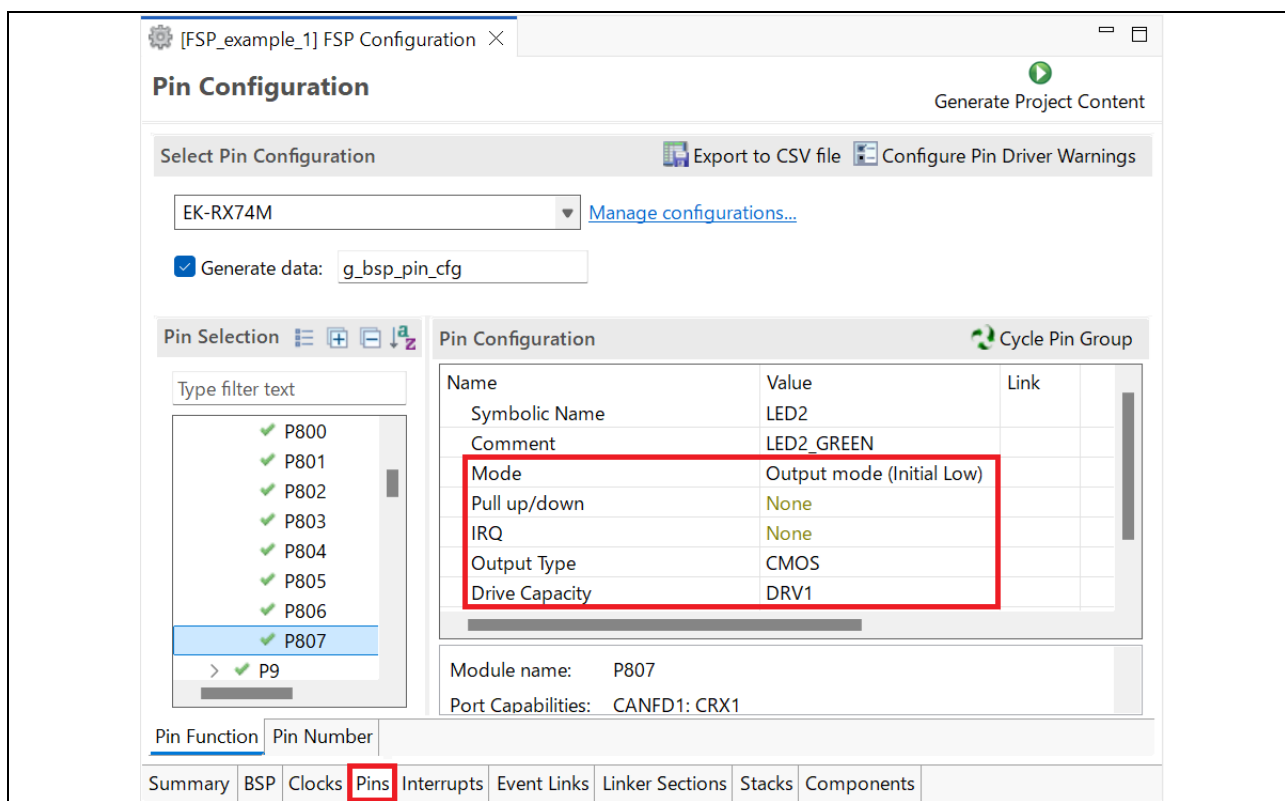


Figure 7-19 Changing the P807 Mode to Output Mode on the [Pin Configuration] Page

Regarding the SW terminal, since P414 is used as the IRQ pin, the configuration is performed via g_external_irq. In EK-RX74M, P414 is initialized as the pin for IRQ30.

Properties × Problems Smart Browser		
g_external_irq0 External IRQ (r_icu)		
Settings	Property	Value
	▼ Common	
	Parameter Checking	Default (BSP)
	▼ Module g_external_irq0 External IRQ (r_icu)	
	Name	g_external_irq0
	Channel	30
	Trigger	Falling
	Digital Filtering	Enabled
	Filter Source	PCLK filter
	Filter Mode	3 consecutive levels match
	Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK / 64
	Callback	irq30_callback
	Pin Interrupt Priority	Priority 15
	▼ Pins	
	IRQ30	P414
	IRQ30-DS	None

Figure 7-20 "Pins" in the [Properties] View of the ICU Module (r_icu)

To assign the pin for use by the ICU module, click on the button for pin configuration to switch to the [Pin Configuration] page.

g_external_irq0 External IRQ (r_icu)		
Settings	Property	Value
	Trigger	Falling
	Digital Filtering	Enabled
	Filter Source	PCLK filter
	Filter Mode	3 consecutive levels match
	Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK / 64
	Callback	irq30_callback
	Pin Interrupt Priority	Priority 15
	▼ Pins	
	IRQ30	P414
	IRQ30-DS	None

Figure 7-21 Button for Pin Configuration in the [Properties] View of the ICU Module (r_icu)

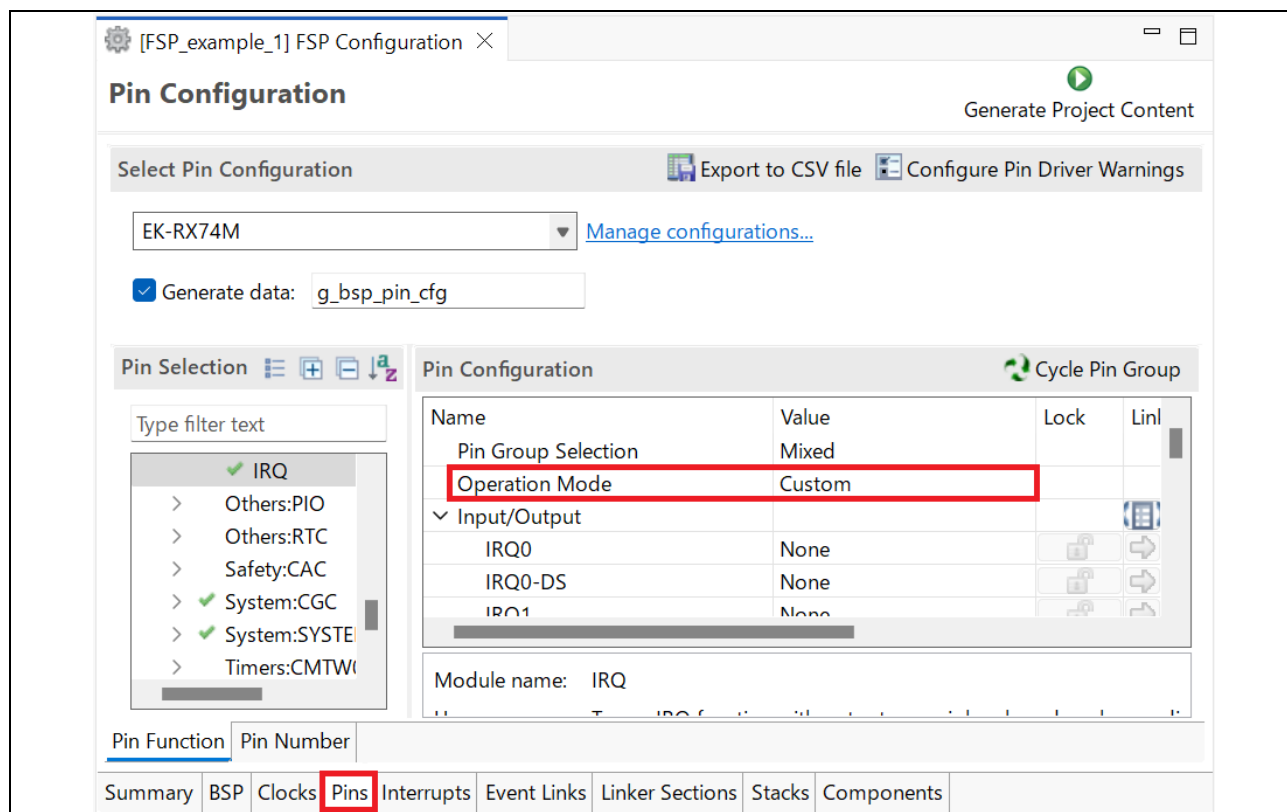


Figure 7-22 IRQ Enabled on the [Pin Configuration] Page

Change "Operation Mode" to "Custom" and assign port pin from any assignable pin.

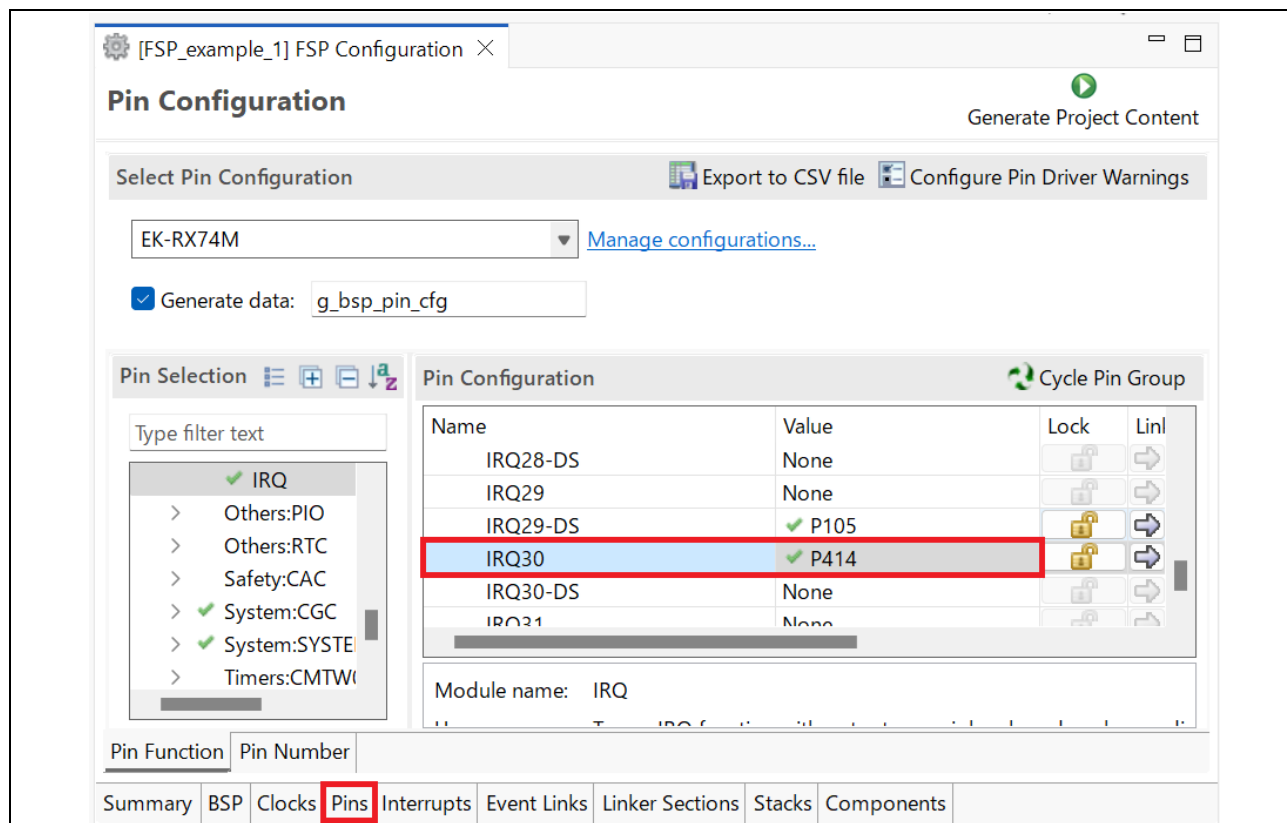


Figure 7-23 Enabling the IRQ and Assigning P414 to IRQ30 on the [Pin Configuration] Page

7.1.7 Structure of the Folders and Files in the Project after Porting and the Targets for Modification

The following shows the structure of the folders and files in the project after porting and the target files for modification. `hal_entry.c` and ported `rskrx671.h` are the targets for modification. For `iodef.h`, add the file created in the new executable project.

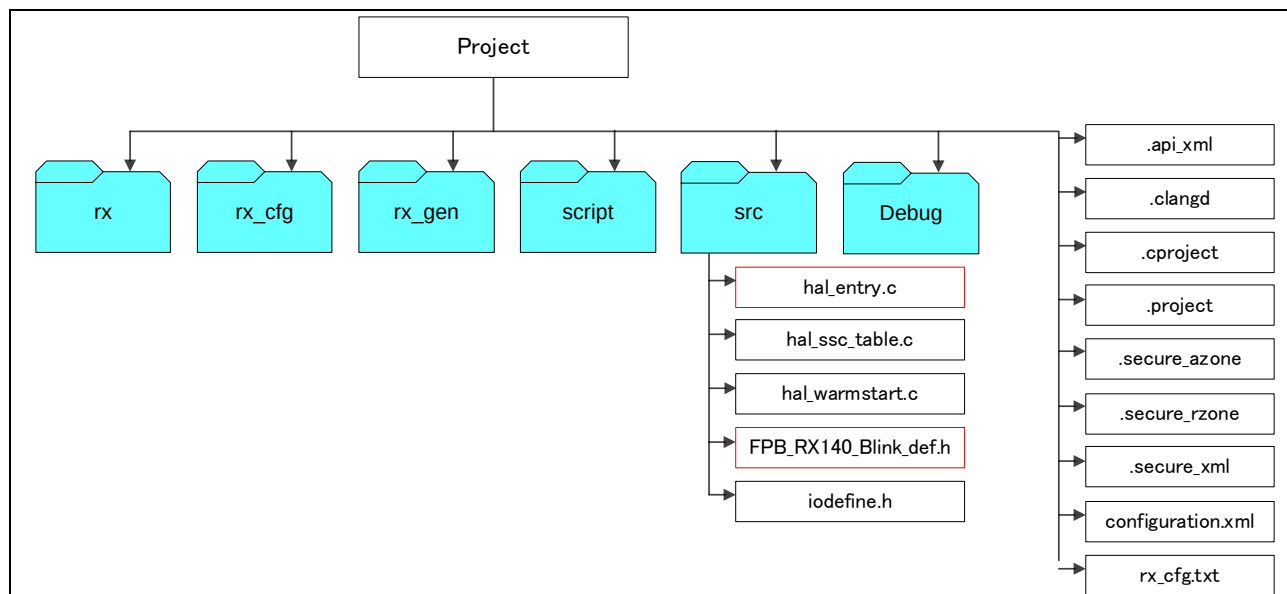


Figure 7-24 Structure of the Folders and Files in the Project after Porting and the Targets for Modification

7.1.8 Key Points in Porting

Use the procedure described in section 5.7, Porting the Source Code, to port the processing of the main function body in `main.c` and the user processing for the CMT and ICU as callback functions to `hal_entry.c`. The initialization processing at the time of startup is not ported because it is replaced with the processing provided by modules. `FPB_RX140_Blink_def.h` also requires porting.

Some of the definitions in `FPB_RX140_Blink_def.h` use definitions provided by `iodef.h`, so add `iodef.h` by following the procedure described in section 5.8, Adding `iodef.h`. As the target device differs between the source and destination projects, some definitions to be used will require modification. Additionally, some definition names have been changed to avoid conflicts with definitions provided by the FSP.

Refer to section 5.9, Replacing Processing with FSP Module Processing, and replace the processing created by the Code Generator with the equivalent processing by the FSP.

7.1.9 Processing after Replacement of the Target Files for Modification

The following listing is of the final contents of hal_entry.c and FPB_RX140_Blink_def.h after modification.

hal_entry.c (The final results of addition are shown in red.)

```
#include "hal_data.h"
#include "FPB_RX140_Blink_def.h"

FSP_CPP_HEADER
FSP_CPP_FOOTER

/*****
 * main() is generated by the RX Configuration editor and is used to generate
 * threads if an RTOS is used. This function is called by main() when no
 * RTOS is used.
 *****/
void hal_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err = FSP_SUCCESS;

    LED1_PODR = LED_OFF;
    LED2_PODR = LED_OFF;

    if (SW1_PUSH == SW1_PIDR) /*SW1_PUSH = Manual Mode, Other = Auto Mode*/
    {
        /* Configure the external interrupt. */
        fsp_err_t err = R_ICU_ExternalIrqOpen(&g_external_irq0_ctrl,
                                             &g_external_irq0_cfg);

        assert(FSP_SUCCESS == err);

        /* Enable the external interrupt. */
        /* Enable not required when used with ELC or DMAC. */
        err = R_ICU_ExternalIrqEnable(&g_external_irq0_ctrl);
        assert(FSP_SUCCESS == err);
    }
    else
    {
        /* Create a timer tick with callback at a rate of 2Hz. */
        /* Initializes the module for CMT0. */
        err = R_CMT_Open(&g_timer0_ctrl, &g_timer0_cfg);
    }
}
```

```

    /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);

    /* Start the timer. */
    (void) R_CMT_Start(&g_timer0_ctrl);
}
while(1)
{
    ; /* Blink LED1, LED2 Alternately by Interrupt */
}

#if BSP_PZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

void cmt0_callback(timer_callback_args_t *p_args)
{
    if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
        /* Add application code to be called periodically here. */
        LED1_PODR = LED2_PODR;
        LED2_PODR = ~LED2_PODR;
    }
}

void irq0_callback(external_irq_callback_args_t *p_args)
{
    (void) p_args;

    LED1_PODR = LED2_PODR;
    LED2_PODR = ~LED2_PODR;
}

```

FPB_RX140_Blink_def.h (The final results of modification are shown in red.)

```

#include "iodefine.h"

/*****
Macro definitions

```

```

*****/

#ifndef FPB_RX140_BLINK_DEF_H_
#define FPB_RX140_BLINK_DEF_H_

/* General Values */
#define LED_ON      (1)
#define LED_OFF     (0)
#define SW1_PUSH    (0)
#define SW1_RELEASE (1)

/* Switches */
#define SW1_PIDR      (PORT5.PIDR.BIT.PIDR00)

/* LED port settings */
#define LED1_PODR     (PORTF.PODR.BIT.PODR06)
#define LED2_PODR     (PORT8.PODR.BIT.PODR07)
(Omitted)
#endif /* FPB_RX140_BLINK_DEF_H_ */

```

7.1.10 Replacement of Processing Created by the Code Generator with Processing by the FSP Modules in this Example of Porting

The following figures compare the processing created by the Code Generator before porting and the processing by the FSP modules after porting and replacement.

7.1.10.1 Comparison of the Processing for Initializing Modules

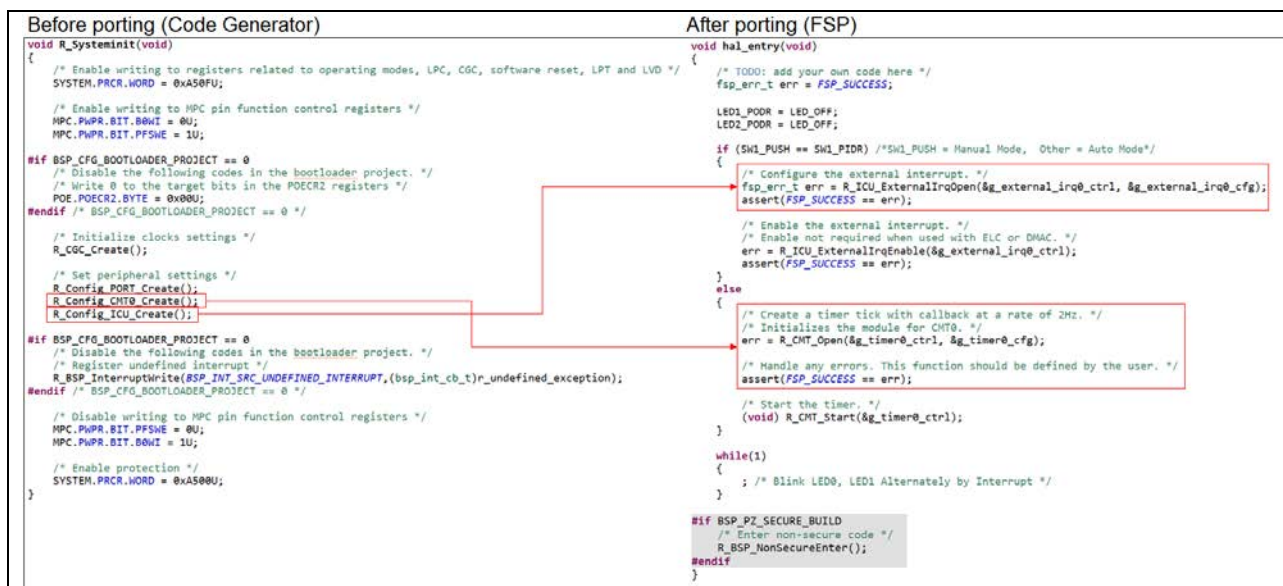


Figure 7-25 Initialization Processing

7.1.10.2 Comparison of the Main Processing

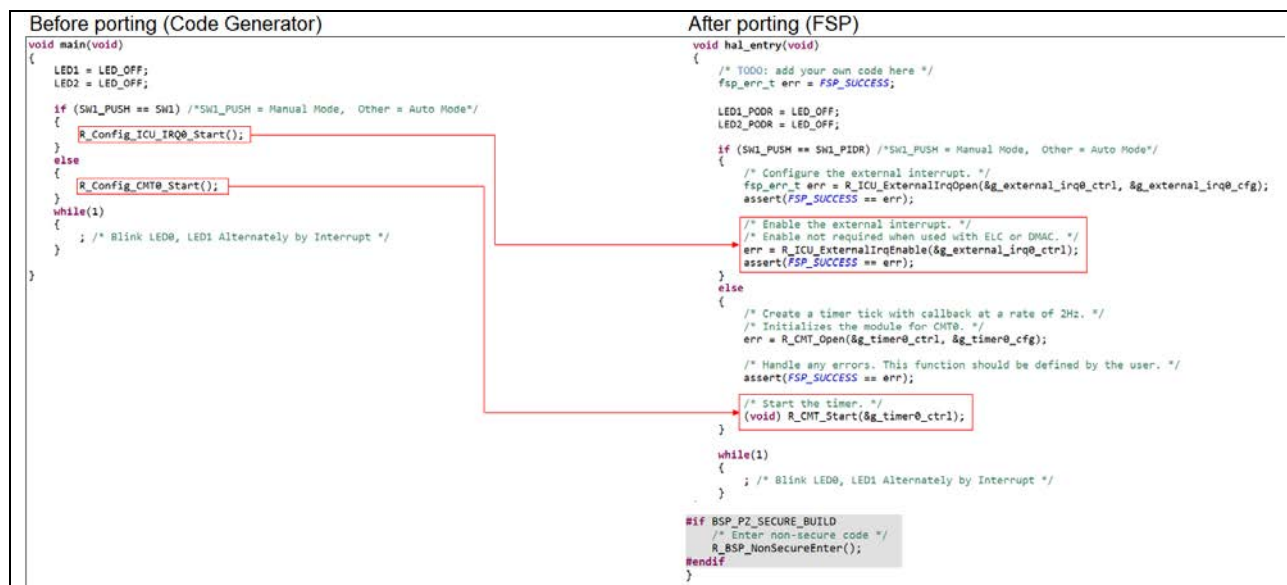


Figure 7-26 Main Processing

7.1.10.3 Comparison of the Callback Functions



Figure 7-27 Callback Functions

7.2 Example of Porting a Sample Program Using the SCI Component Created by the Code Generator

The following describes the procedure for porting a sample program that uses the functions of the SCI (asynchronous mode) component created by the Code Generator (hereafter called the source project) to an FSP project.

7.2.1 Structure of the Folders and Files in the Source Project and the Targets for Porting

The following shows the structure of the folders and files in the source project and the target files for porting. `main.c` and `Config_SCI2_user.c` are the targets for porting.

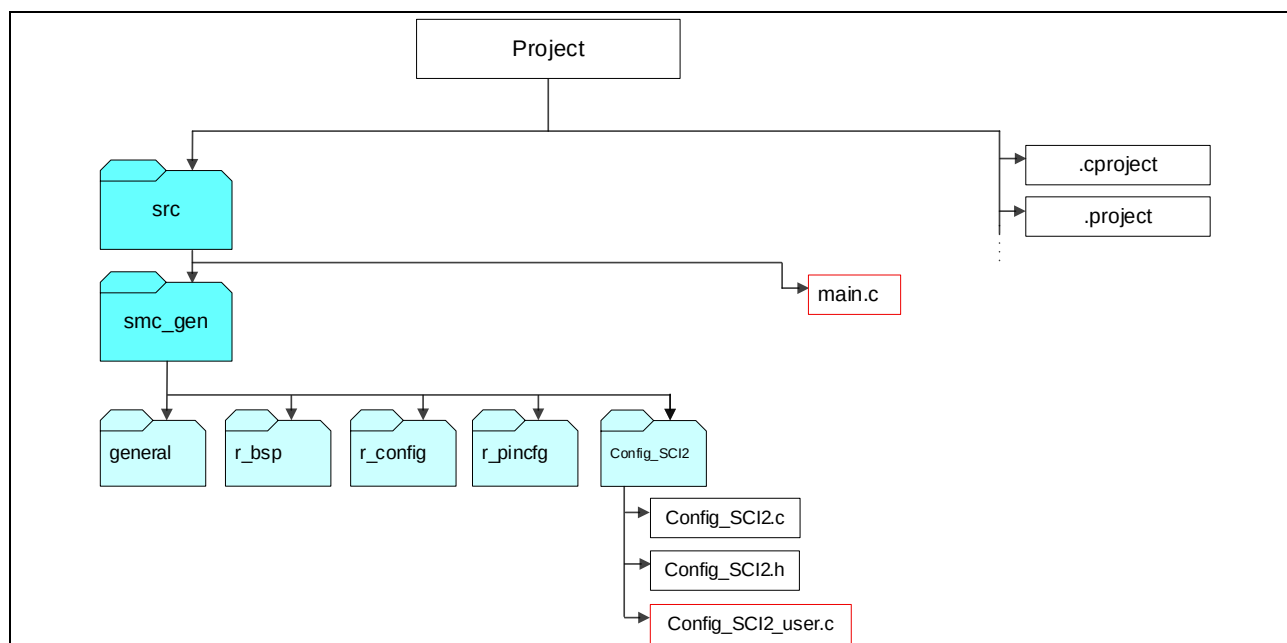


Figure 7-28 Structure of the Folders and Files in the Source Project and the Targets for Porting

7.2.2 Overview of the Processing in the Source Project

The following listing is of main.c.

src\main.c

```

/*****
Includes  <System Includes> , "Project Includes"
*****/
#include "r_smc_entry.h"

/*****
Private global variables and functions
*****/
void main(void);

/*****
* Function Name: main
* Description  : This function implements main function.
* Arguments    : None
* Return Value : None
*****/
void main(void)
{
    /* Set up SCI2 receive buffer */
    R_Config_SCI2_Serial_Receive((uint8_t *)&g_rx_char, 1);

    /* Enable SCI2 operations */
    R_Config_SCI2_Start();

    while (1U)
    {
        /* Check the received character */
        if (true == g_tx_flag)
        {
            g_tx_flag = false;

            /* Transmit the data */
            if (MD_OK == R_Config_SCI2_Serial_Send((uint8_t *)&g_tx_char, 1))
            {
                while (false == g_tx_done)
                {
                    /* Wait */

```

```

    }
  }
}
}
}

```

The SCI initialization processing is executed by the following function at the time of startup.

src\smc_gen\Config_SCI2\Config_SCI2.c

```

void R_Config_SCI2_Create(void)
{
    /* Cancel SCI stop state */
    MSTP(SCI2) = 0U;

    /* Set interrupt priority */
    IPR(SCI2, RXI2) = _0F_SCI_PRIORITY_LEVEL15;
    IPR(SCI2, TXI2) = _0F_SCI_PRIORITY_LEVEL15;

    /* Clear the control register */
    SCI2.SCR.BYTE = 0x00U;

    /* Set clock enable */
    SCI2.SCR.BYTE = _00_SCI_INTERNAL_SCK_UNUSED;

    /* Clear the SIMR1.IICM, SPMR.CKPH, and CKPOL bit, and set SPMR */
    SCI2.SIMR1.BIT.IICM = 0U;
    SCI2.SPMR.BYTE = _00_SCI_RTS | _00_SCI_CLOCK_NOT_INVERTED |
    _00_SCI_CLOCK_NOT_DELAYED;

    /* Set control registers */
    SCI2.SMR.BYTE = _01_SCI_CLOCK_PCLK_4 | _00_SCI_MULTI_PROCESSOR_DISABLE |
    _00_SCI_STOP_1 | _00_SCI_PARITY_DISABLE |
    _00_SCI_DATA_LENGTH_8 | _00_SCI_ASYNCHRONOUS_OR_I2C_MODE;
    SCI2.SCMR.BYTE = _00_SCI_SERIAL_MODE | _00_SCI_DATA_INVERT_NONE |
    _00_SCI_DATA_LSB_FIRST | _10_SCI_DATA_LENGTH_8_OR_7 |
    _62_SCI_SCMR_DEFAULT;
    SCI2.SEMR.BYTE = _00_SCI_BIT_MODULATION_DISABLE | _08_SCI_6_BASE_CLOCK |
    _00_SCI_NOISE_FILTER_DISABLE | _00_SCI_LOW_LEVEL_START_BIT;

    /* Set bit rate */
    SCI2.BRR = 0x40U;
}

```

```

/* Set RXD2 pin */
MPC.P52PFS.BYTE = 0x0AU;
PORT5.PMR.BYTE |= 0x04U;

/* Set TXD2 pin */
MPC.P50PFS.BYTE = 0x0AU;
PORT5.PODR.BYTE |= 0x01U;
PORT5.PDR.BYTE |= 0x01U;

R_Config_SCI2_Create_UserInit();
}

```

In addition, the following related processing is implemented as the user processing for use with the code from the Code Generator.

src\smc_gen\Config_SCI2\Config_SCI2_user.c

```

(Omitted)
/* Start user code for global. Do not edit comment generated here */
/* Global used to receive a character from the PC terminal */
uint8_t g_rx_char;

/* Global used to transmit a character to the PC terminal */
uint8_t g_tx_char;

/* Flag used to control transmission to PC terminal */
volatile uint8_t g_tx_flag = false;

/* Flag used locally to detect transmission complete */
volatile uint8_t g_tx_done;
/* End user code. Do not edit comment generated here */
(Omitted)
void R_Config_SCI2_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    /* End user code. Do not edit comment generated here */
}
(Omitted)
/*****
* Function Name: r_Config_SCI2_callback_transmitend
* Description  : This function is a callback function when SCI2 finishes
*                transmission
*****/

```

```

* Arguments      : None
* Return Value   : None
*****/

static void r_Config_SCI2_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI2_callback_transmitend. Do not edit
       comment generated here */
    g_tx_done = true;
    /* End user code. Do not edit comment generated here */
}

/*****
* Function Name: r_Config_SCI2_callback_receiveend
* Description   : This function is a callback function when SCI2 finishes
*                  reception
* Arguments     : None
* Return Value  : None
*****/

static void r_Config_SCI2_callback_receiveend(void)
{
    /* Start user code for r_Config_SCI2_callback_transmitend. Do not edit
       comment generated here */
    g_tx_char = g_rx_char;
    g_tx_flag = true;

    /* Set up SCI2 receive buffer */
    R_Config_SCI2_Serial_Receive((uint8_t *)&g_rx_char, 1);
    /* End user code. Do not edit comment generated here */
}

/*****
* Function Name: r_Config_SCI2_callback_receiveerror
* Description   : This function is a callback function when SCI2 reception encounters
error
* Arguments     : None
* Return Value  : None
*****/

static void r_Config_SCI2_callback_receiveerror(void)
{
    /* Start user code for r_Config_SCI2_callback_transmitend. Do not edit

```

```

comment generated here */
/* End user code. Do not edit comment generated here */
}

/* Start user code for adding. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

```

The program in the source project receives data from the terminal on the PC and sends the received data to the terminal.

7.2.3 Creating the Destination Project for Porting

Use the procedure described in Chapter 5, Steps in Porting a Pre-FSP Project to an FSP Project, to create the destination project for porting. The new FSP project created with reference to the text from section 5.1, Creating a New FSP Project to section 5.4, Porting Modules is used as the base project.

7.2.4 Adding an SCI Module

Use the following procedure to add an SCI module for one channel to the project.

From the [New Stack] button, select [Connectivity] → [UART (r_sci_b_uart)].

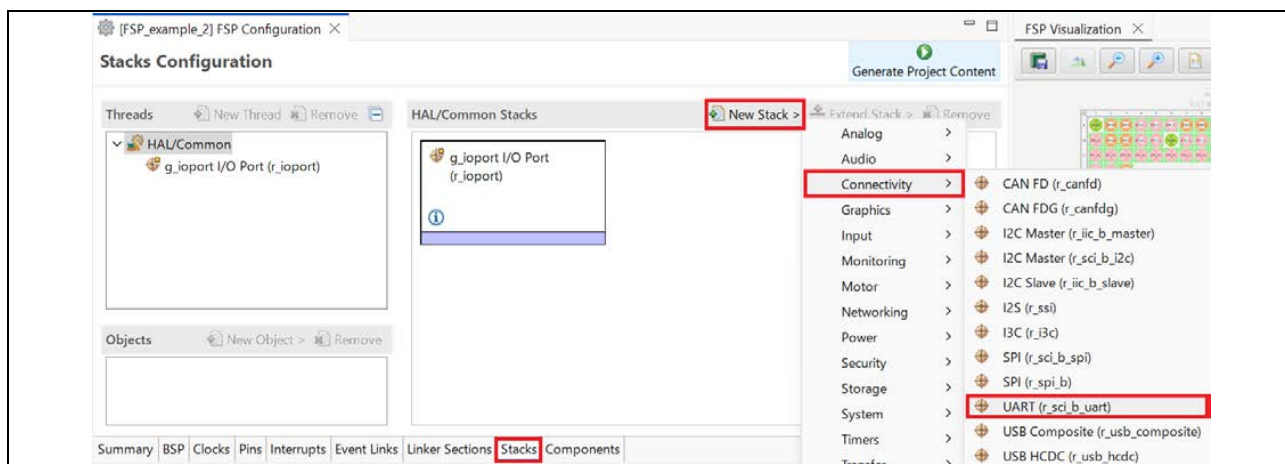


Figure 7-29 Selecting UART (SCI) on the [Stacks Configuration] Page

"g_uart0 (r_sci_b_uart)" for the SCI will be added to the [Stacks Configuration] page.

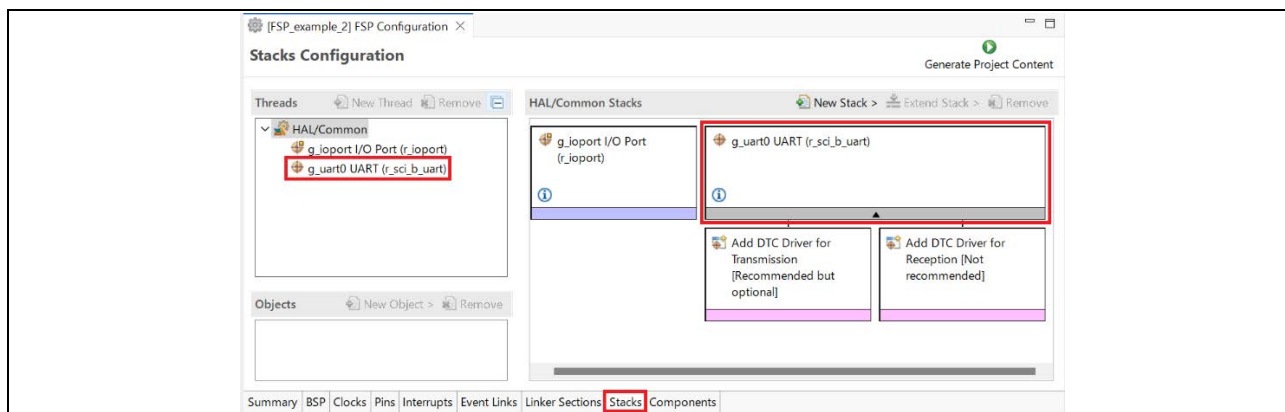


Figure 7-30 UART (SCI) Module (r_sci_b_uart) Added to the [Stacks Configuration] Page

Set the properties of g_uart0 to match the SCI settings made by the Code Generator.

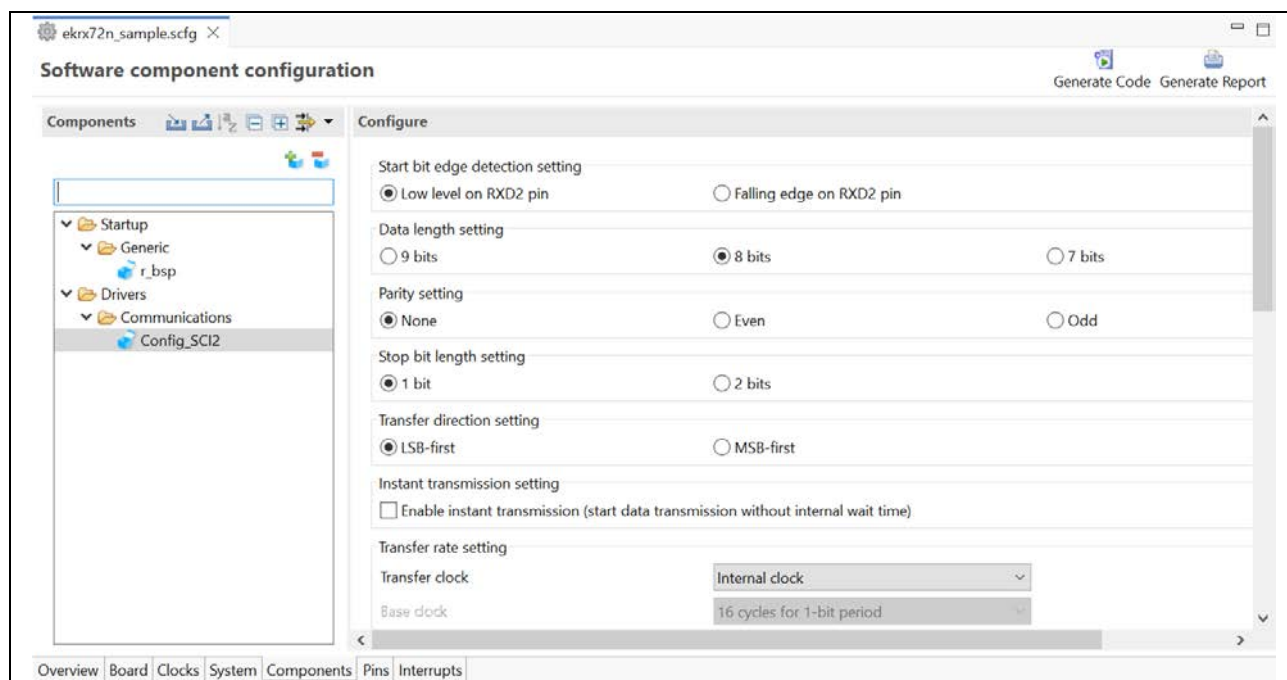


Figure 7-31 SCI Settings Made by the Code Generator on the [Components] Page

Set the properties of g_uart0 as follows.

Table 7.2 Items and Values of Settings for Porting (1)

No.	Item and Value that were Set in the SCI Component	Item and Value that Should be Set as g_uart0 Properties
1	Start bit edge detection setting "Low level on RXD2 pin" is selected.	Set "Extra" > "Start bit detection" to "Low Level".
2	Data length setting "8 bits" is selected.	Set "General" > "Data Bits" to "8bits".
3	Parity setting "None" is selected.	Set "General" > "Parity" to "None".
4	Stop bit length setting "1 bit" is selected.	Set "General" > "Stop Bits" to "1bit".
5	Transfer direction setting "LSB-first" is selected.	None
6	Transfer rate setting "Transfer clock" is set to "Internal clock". "Basic clock" is set to "16 cycles for 1-bit period". "Bit rate" is set to "38400 bps". "Enable modulation duty correction" is deselected. "SCK2 pin function" is set to "SCK2 is not used".	Set "Extra" > "Clock Source" to "Internal Clock".
		None
		Set "Baud" > "Baud Rate" to "38400".
		Set "Baud" > "Baud Rate Modulation" to "Disabled".
		None
7	Noise filter setting "Enable noise filter" is deselected.	Set "Extra" > "Noise Filter" to "Disable".

Table 7.3 Items and Values of Settings for Porting (2)

No.	Item and Value that were Set in the SCI Component	Item and Value that Should be Set as g_uart0 Properties
8	Hardware flow control setting "None" is selected.	Set "Common" > "Flow Control Support" to "Disable".
9	Data match detection setting "Enable data match detection" is deselected.	None
10	Data handling setting "Transmit data handling" is set to "Data handled in interrupt service routine". "Receive data handling" is set to "Data handled in interrupt service routine".	None
11	Interrupt setting "TXI2 priority" is set to "Level 15 (highest)". "RXI2 priority" is set to "Level 15 (highest)". "Enable reception error interrupt (ERI2)" is selected. "TEI2, ERI2 priority (Group BL0)" is set to "Level 15 (highest)".	Set "Interrupts" > "Transmit Data Empty Interrupt Priority" to "Priority 15".
		Set "Interrupts" > "Receive Interrupt Priority" to "Priority 15".
		None
		Set "Interrupts" > "Transmit End Interrupt Priority" to "Priority 15".
		Set "Interrupts" > "Error Interrupt Priority" to "Priority 15".
12	Multiple interrupts setting All items are deselected.	None
13	Callback function setting "Transmission end" is selected. "Reception end" is selected. "Reception error" is selected.	Set "Interrupts" > "Callback" to "my_rsci_callback".

Table 7.4 Items and Values of Settings for Porting (3)

No.	Other Settings in the SCI Component	Item and Value that Should be Set as g_uart0 Properties
1	SCI/SCIF Asynchronous Mode Work mode: Transmission/Reception	None
2	Resource: SCI2	Set "General" > "Channel" to "12". This change is necessary because the target device and board differ from those for the source project.

The following shows the properties of g_uart0 after setting of the above values.

Properties Problems Smart Browser

g_uart0 UART (r_sci_b_uart)

Settings

Property	Value
Module g_uart0 UART (r_sci_b_uart)	
General	
Name	g_uart0
Channel	12
Data Bits	8bits
Parity	None
Stop Bits	1bit
Baud	
Baud Rate	38400
Baud Rate Modulation	Disabled
Max Error (%)	5
Flow Control	
Extra	
RS-485	
Clock Source	Internal Clock
Start bit detection	Low Level
Noise Filter	Disable
Receive FIFO Trigger Level	Max
Interrupts	
Callback	my_rsci_callback
Receive Interrupt Priority	Priority 15
Transmit Data Empty Interrupt Priority	Priority 15
Transmit End Interrupt Priority	Priority 15
Error Interrupt Priority	Priority 15

Figure 7-32 Properties of the UART (SCI) Module (r_sci_b_uart)

According to the settings of the callback function and interrupt priority levels, the information on the use of RSCI12 interrupts is registered as shown below. The names of the interrupt functions contained in the module are used without change.

[FSP_example_2] FSP Configuration

Interrupts Configuration

Generate Project Content

User Events

New User Event > Remove

Event	ISR

Allocations

Interrupt	Event	ISR	Priority
0	ID:105-6 GROUPAL1 RSCI12 TEI (Transmit end)	sci_b_uart_tei_isr	Priority 15
1	ID:105-7 GROUPAL1 RSCI12 ERI (Receive error)	sci_b_uart_eri_isr	Priority 15
128	ID:X RSCI12 RXI (Receive data full)	sci_b_uart_rxi_isr	Priority 15
129	ID:X RSCI12 TXI (Transmit data empty)	sci_b_uart_txi_isr	Priority 15

Summary BSP Clocks Pins **Interrupts** Event Links Linker Sections Stacks Components

Figure 7-33 Registered UART (SCI) Interrupts on the [Interrupts Configuration] Page

Execute [Generate Project Content] to generate the definitions of the interrupt functions and output the declarations of the interrupt functions to vector_data.h. The following screenshot shows vector_data.h after the interrupt functions have been output with the above configuration.

rx_gen\vector_data.h

```

1  /* generated vector header file - do not edit */
2  #ifndef VECTOR_DATA_H
3  #define VECTOR_DATA_H
4  #ifdef __cplusplus
5      extern "C" {
6      #endif
7
8      /* Number of interrupts allocated */
9  #ifndef VECTOR_DATA_IRQ_COUNT
10     #define VECTOR_DATA_IRQ_COUNT (4)
11 #endif
12
13 /* ISR prototypes */
14 void sci_b_uart_tei_isr(void);
15 void sci_b_uart_eri_isr(void);
16 void sci_b_uart_rxi_isr(void);
17 void sci_b_uart_txi_isr(void);
18
19 /* Vector table allocations */
20 #define VECTOR_NUMBER_RSCI12_TEI ((IRQn_Type) (6 << 8U | 105)) /*
21 #define RSCI12_TEI_IRQn          ((IRQn_Type) (6 << 8U | 105)) /*
22 #define VECTOR_NUMBER_RSCI12_ERI ((IRQn_Type) (7 << 8U | 105)) /*
23 #define RSCI12_ERI_IRQn          ((IRQn_Type) (7 << 8U | 105)) /*
24 #define VECTOR_NUMBER_RSCI12_RXI ((IRQn_Type) 128) /* ID:X RSCI12
25 #define RSCI12_RXI_IRQn          ((IRQn_Type) 128) /* ID:X RSCI12
26 #define VECTOR_NUMBER_RSCI12_TXI ((IRQn_Type) 129) /* ID:X RSCI12
27 #define RSCI12_TXI_IRQn          ((IRQn_Type) 129) /* ID:X RSCI12

```

Figure 7-34 Declarations of the UART (SCI) Interrupt Functions in vector_data.h

The declaration of the callback function is output to hal_data.h.

rx_gen\hal_data.h

```

1  /* generated HAL header file - do not edit */
2  #ifndef HAL_DATA_H
3  #define HAL_DATA_H
4  #include <stdint.h>
5  #include "bsp_api.h"
6  #include "common_data.h"
7  #include "r_sci_b_uart.h"
8  #include "r_uart_api.h"
9
10 FSP_HEADER
11 /** UART on SCI Instance. */
12 extern const uart_instance_t g_uart0;
13
14 /** Access the UART instance using these structures when calling API functions directly (::p_ap
15 extern sci_b_uart_instance_ctrl_t g_uart0_ctrl;
16 extern const uart_cfg_t g_uart0_cfg;
17 extern const sci_b_uart_extended_cfg_t g_uart0_cfg_extend;
18
19 #ifndef my_rsci_callback
20 void my_rsci_callback(uart_callback_args_t *p_args);
21 #endif
22 void hal_entry(void);
23 void g_hal_init(void);
24 FSP_FOOTER
25 #endif /* HAL_DATA_H */

```

Figure 7-35 Declaration of the UART (SCI) Callback Function in hal_data.h

The bodies of the interrupt functions are defined in the module.

rx\fsp\src\r_sci_b_uart\r_sci_b_uart.c

```

1508  #if (SCI_B_UART_CFG_TX_ENABLE)
1509
1511  * TXI interrupt processing for UART mode. TXI interrupt fires when the data in the data regi
1516  void sci_b_uart_txi_isr (void)
1595
1596  #endif
1597
1598  #if (SCI_B_UART_CFG_RX_ENABLE)
1599
1601  * RXI interrupt processing for UART mode. RXI interrupt happens when data arrives to the dat
1613  void sci_b_uart_rxi_isr (void)
1740
1741  #endif
1742
1743  #if (SCI_B_UART_CFG_TX_ENABLE)
1744
1746  * TEI interrupt processing for UART mode. The TEI interrupt fires after the last byte is tra
1750  void sci_b_uart_tei_isr (void)
1775
1776  #endif
1777
1778  #if (SCI_B_UART_CFG_RX_ENABLE)
1779
1781  * ERI interrupt processing for UART mode. When an ERI interrupt fires, the user callback fun
1784  void sci_b_uart_eri_isr (void)
1834
1835  #endif
  
```

Figure 7-36 Interrupt Functions in r_sci_b_uart.c

For the callback function, however, only the declaration is output, and the user has to create the body of the callback function.

7.2.5 Adding the Port Settings

The source project uses Config_Port to configure the SCI pins. In the destination project, make settings on the [Pin Configuration] page so that initialization by the IOPORT module to be called from the startup processing is used for the pin settings.

Assignment of P52 to RXD2 and P50 to TXD2 in SCI2 was specified in the source project but this port pin assignment is changed in the destination project.

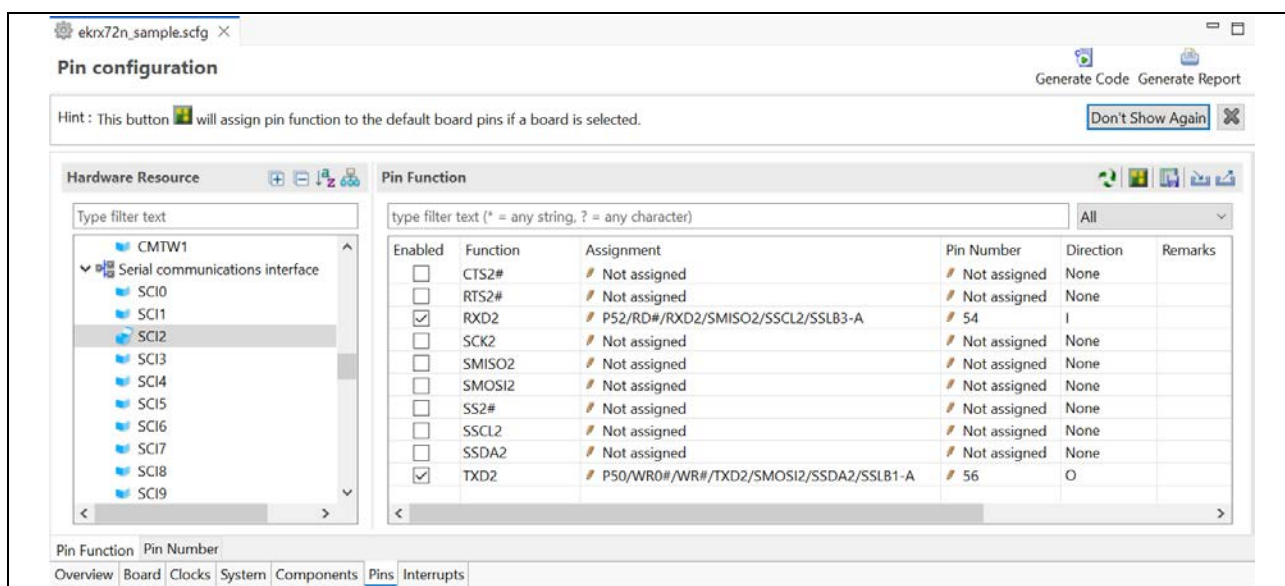


Figure 7-37 SCI Pin Configuration on the [Pins] Page

Pins should be assigned to RXD and TXD under "Pins" in the [Properties] view of g_uart0. Note that if you select a board when creating the project, peripheral functions and pins may be assigned from the start based on the board information.

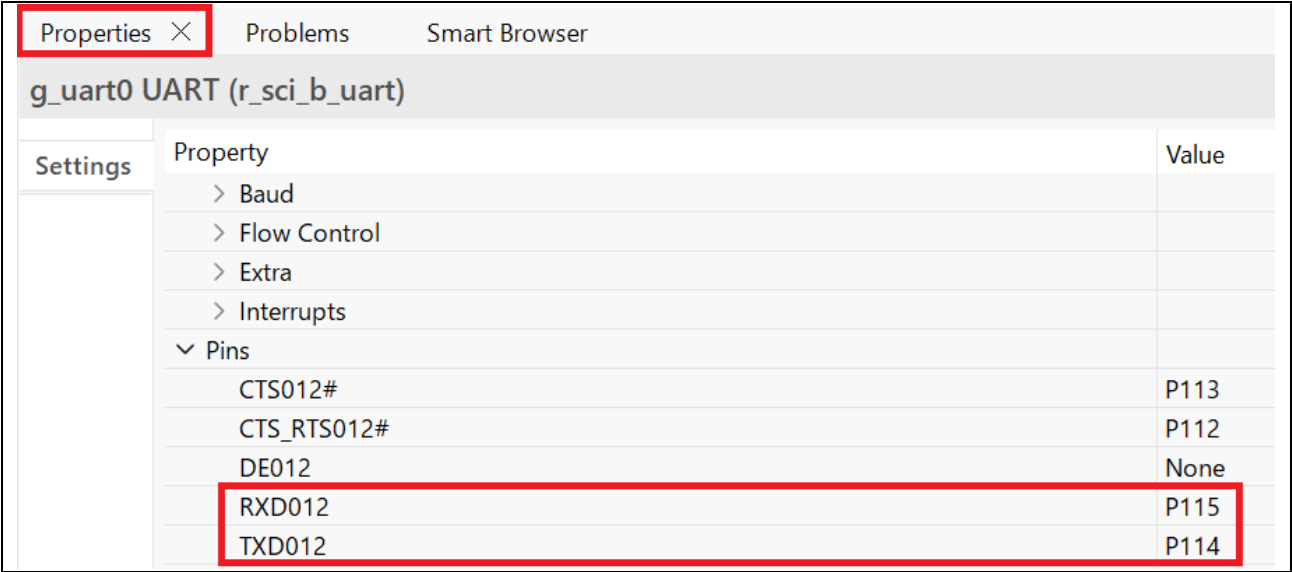


Figure 7-38 "Pins" in the [Properties] View of the UART (SCI) Module (r_sci_b_uart)

Click on the button for pin configuration to switch to the [Pin Configuration] page.

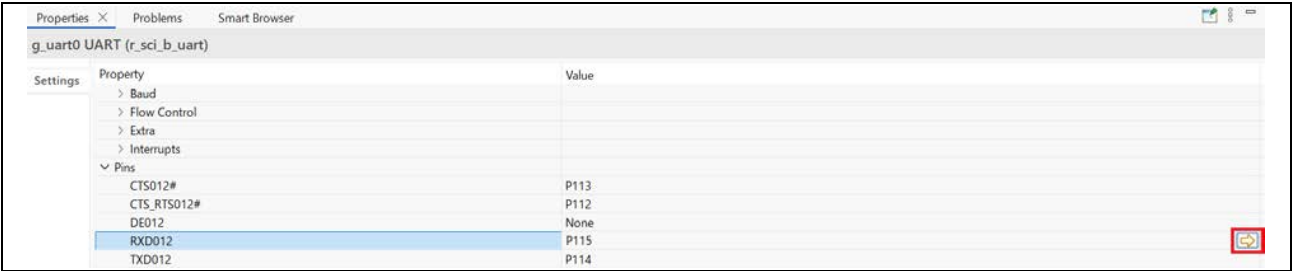


Figure 7-39 Button for Pin Configuration in the [Properties] View of the UART (SCI) Module (r_sci_b_uart)

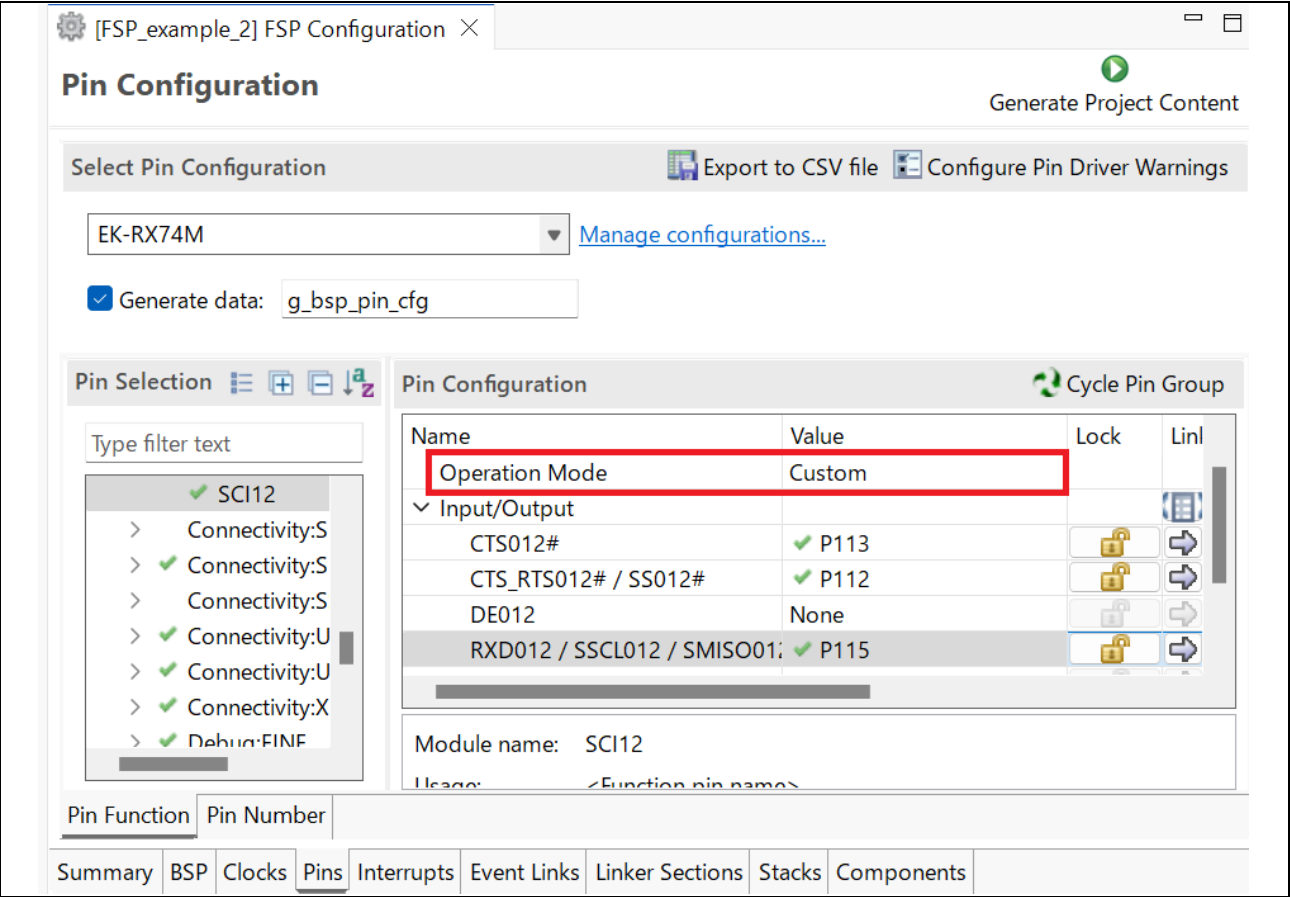


Figure 7-40 SCI12 Operation Mode Initial Settings on the [Pin Configuration] Page

In EK-RX74M, the Operation Mode is initially set to "Custom," and the RXD and TXD terminals are assigned to ports that match the board specifications. Change "Operation Mode" to "Asynchronous UART", and the pins will be assigned to RXD and TXD.

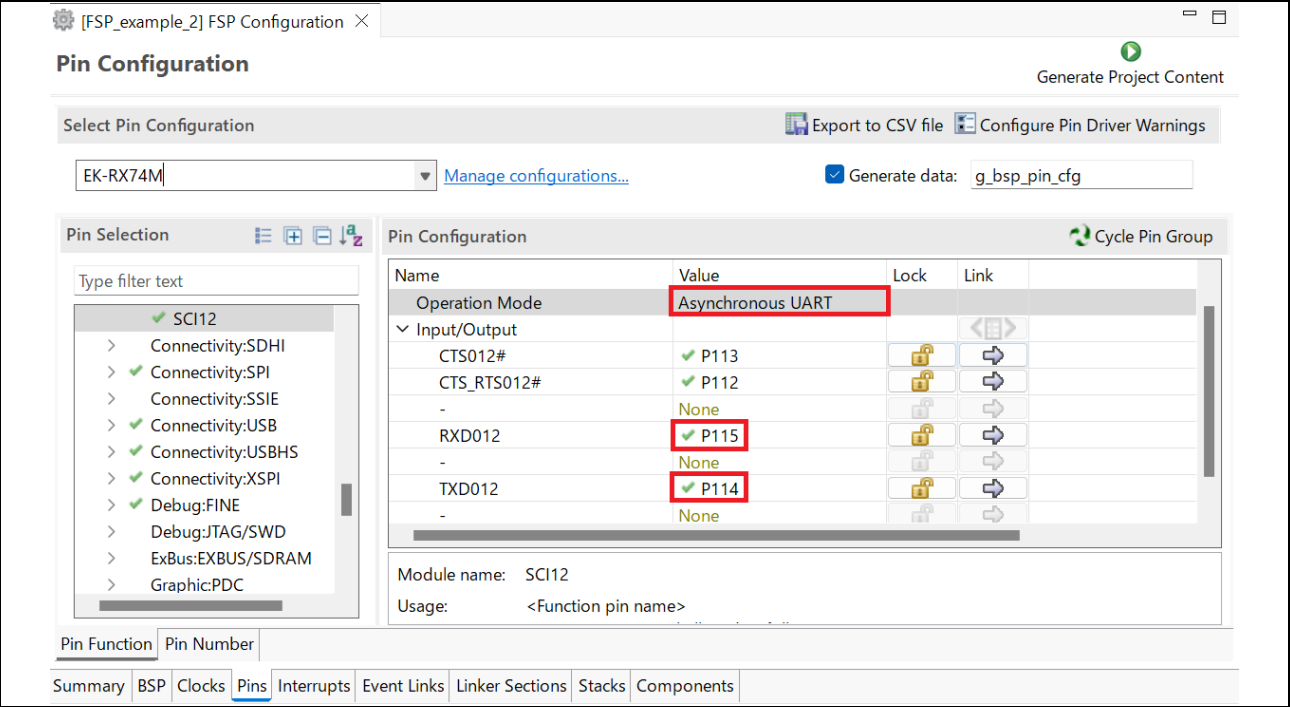


Figure 7-41 Changing the SCI12 Operation Mode to UART on the [Pin Configuration] Page

7.2.6 Structure of the Folders and Files in the Project after Porting and the Target for Modification

The following shows the structure of the folders and files in the project after porting and the target file for modification. `hal_entry.c` is the target for modification.

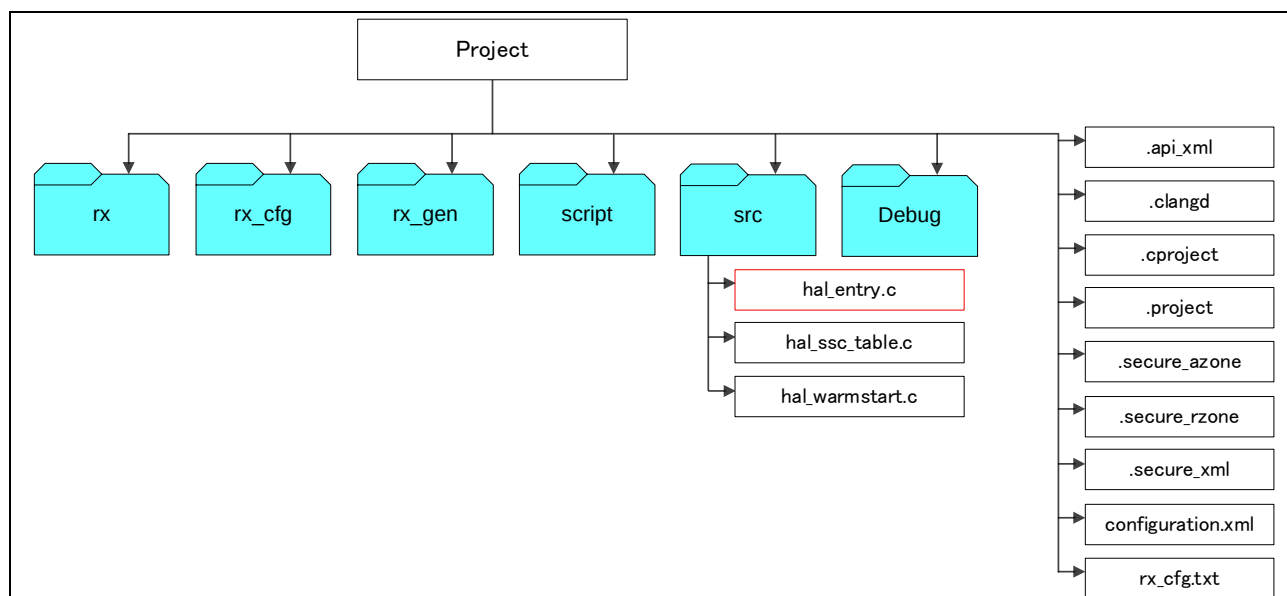


Figure 7-42 Structure of the Folders and Files in the Project after Porting and the Target for Modification

7.2.7 Key Points in Porting

Use the procedure described in section 5.7, Porting the Source Code, to port the processing of the main function body in `main.c` and the user processing for the SCI as a callback function to `hal_entry.c`. The initialization processing at the time of startup is not ported because it is replaced with the processing provided by modules.

Refer to section 5.9, Replacing Processing with FSP Module Processing, and replace the processing created by the Code Generator with the equivalent processing by the FSP.

7.2.8 Processing after Replacement of the Target File for Modification

The following listing is of the final contents of hal_entry.c after modification.

hal_entry.c (The final results of addition are shown in red.)

```
#include "hal_data.h"

/* Global used to receive a character from the PC terminal */
uint8_t g_rx_char;

/* Global used to transmit a character to the PC terminal */
uint8_t g_tx_char;

/* Flag used to control transmission to PC terminal */
volatile uint8_t g_tx_flag = false;

/* Flag used locally to detect transmission complete */
volatile uint8_t g_tx_done;

FSP_CPP_HEADER
FSP_CPP_FOOTER

/*****
 * main() is generated by the RX Configuration editor and is used to generate
 * threads if an RTOS is used. This function is called by main() when no
 * RTOS is used.
 *****/
void hal_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err = FSP_SUCCESS;

    /* Called the R_SCI_B_UART_Open to open the transfer instance
       with initial configuration. */
    err = g_uart0.p_api->open(&g_uart0_ctrl, &g_uart0_cfg);
    assert(FSP_SUCCESS == err);

    /* Called the R_SCI_B_UART_Read to read data from UART. */
    err = g_uart0.p_api->read(&g_uart0_ctrl, &g_rx_char, 1);
    assert(FSP_SUCCESS == err);

    while (1U)
```

```

{
    /* Check the received character */
    if (true == g_tx_flag)
    {
        g_tx_flag = false;

        /* Transmit the data */
        /* Called the R_SCI_B_UART_Write to write data to UART. */
        err = g_uart0.p_api->write(&g_uart0_ctrl, &g_tx_char, 1);
        assert(FSP_SUCCESS == err);

        while (false == g_tx_done)
        {
            /* Wait */
        }
    }
}

#ifdef BSP_PZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

/*****
 * Function Name: my_rsci_callback
 * Description  : This is a template for an RSCI Async Mode callback function.
 * Arguments    : pArgs -
 *                pointer to rsci_cb_args_t structure cast to a void. Structure
 *                contains event and associated data.
 * Return Value : none
 *****/
void my_rsci_callback(uart_callback_args_t *p_args)
{
    fsp_err_t err = FSP_SUCCESS;

    /* Handle the UART event */
    switch (p_args->event)
    {
        /* Received a character */
        case UART_EVENT_RX_CHAR:
    }

```

```
{
    break;
}

/* Receive complete */
case UART_EVENT_RX_COMPLETE:
{
    g_tx_char = g_rx_char;
    g_tx_flag = true;

    /* Called the R_SCI_B_UART_Read to read data from UART. */
    err = g_uart0.p_api->read(&g_uart0_ctrl, &g_rx_char, 1);
    assert(FSP_SUCCESS == err);
    break;
}

/* Transmit complete */
case UART_EVENT_TX_COMPLETE:
{
    g_tx_done = true;
    break;
}

default:
{
}
}
}
```

7.2.9 Replacement of Processing Created by the Code Generator with Processing by the FSP Module in this Example of Porting

The following figures compare the processing created by the Code Generator before porting and the processing by the FSP module after porting and replacement.

7.2.9.1 Comparison of the Processing for Initializing Modules

Before porting (Code Generator)	After porting (FSP)
<pre> void R_SystemInit(void) { /* Enable writing to registers related to operating modes, LPC, CGC and software reset */ SYSTEM.PRCR.WORD = 0xA5080U; /* Enable writing to MPC pin function control registers */ MPC.PWPR.BIT.B0W1 = 0U; MPC.PWPR.BIT.PFSWE = 1U; #if BSP_CFG_BOOTLOADER_PROJECT == 0 /* Disable the following codes in the bootloader project. */ /* Write 0 to the target bits in the POECR2 registers */ POE3.POECR2.WORD = 0x0000U; #endif /* BSP_CFG_BOOTLOADER_PROJECT == 0 */ /* Initialize clocks settings */ R_CGC_Create(); /* Set peripheral settings */ R_Config_SCI2_Create(); /* Set interrupt settings */ R_Interrupt_Create(); #if BSP_CFG_BOOTLOADER_PROJECT == 0 /* Disable the following codes in the bootloader project. */ /* Register undefined interrupt */ R_BSP_InterruptWrite(BSP_INT_SRC_UNDEFINED_INTERRUPT, (bsp_int_cb_t)r_undefined_exception); #endif /* BSP_CFG_BOOTLOADER_PROJECT == 0 */ /* Register group B10 interrupt TEI2 (SCI2) */ R_BSP_InterruptWrite(BSP_INT_SRC_B10_SCI2_TEI2, (bsp_int_cb_t)r_config_sci2_transmitend_interrupt); /* Register group B10 interrupt ERI2 (SCI2) */ R_BSP_InterruptWrite(BSP_INT_SRC_B10_SCI2_ERI2, (bsp_int_cb_t)r_config_sci2_receiveerror_interrupt); /* Disable writing to MPC pin function control registers */ MPC.PWPR.BIT.B0W1 = 0U; MPC.PWPR.BIT.PFSWE = 1U; /* Enable protection */ SYSTEM.PRCR.WORD = 0xA5080U; } </pre>	<pre> void hal_entry(void) { /* TODO: add your own code here */ fsp_err_t err = FSP_SUCCESS; /* Called the R_SCI_B_UART_Open to open the transfer instance with initial configuration. */ err = g_uart0_p_api->open(&g_uart0_ctrl, &g_uart0_cfg); assert(FSP_SUCCESS == err); /* Called the R_SCI_B_UART_Read to read data from UART. */ err = g_uart0_p_api->read(&g_uart0_ctrl, &g_rx_char, 1); assert(FSP_SUCCESS == err); while (1U) { /* Check the received character */ if (true == g_tx_flag) { g_tx_flag = false; /* Transmit the data */ /* Called the R_SCI_B_UART_Write to write data to UART. */ err = g_uart0_p_api->write(&g_uart0_ctrl, &g_tx_char, 1); assert(FSP_SUCCESS == err); while (false == g_tx_done) { /* Wait */ } } } #if BSP_PZ_SECURE_BUILD /* Enter non-secure code */ R_BSP_NonSecureEnter(); #endif } </pre>

Figure 7-43 Initialization Processing

7.2.9.2 Comparison of the Main Processing

Before porting (Code Generator)	After porting (FSP)
<pre> void main(void) { /* Set up SCI2 receive buffer */ R_Config_SCI2_Serial_Receive((uint8_t *)&g_rx_char, 1); /* Enable SCI2 operations */ R_Config_SCI2_Start(); while (1U) { /* Check the received character */ if (true == g_tx_flag) { g_tx_flag = false; /* Transmit the data */ if (MD_OK == R_Config_SCI2_Serial_Send((uint8_t *)&g_tx_char, 1)) { while (false == g_tx_done) { /* Wait */ } } } } } </pre>	<pre> void hal_entry(void) { /* TODO: add your own code here */ fsp_err_t err = FSP_SUCCESS; /* Called the R_SCI_B_UART_Open to open the transfer instance with initial configuration. */ err = g_uart0_p_api->open(&g_uart0_ctrl, &g_uart0_cfg); assert(FSP_SUCCESS == err); /* Called the R_SCI_B_UART_Read to read data from UART. */ err = g_uart0_p_api->read(&g_uart0_ctrl, &g_rx_char, 1); assert(FSP_SUCCESS == err); while (1U) { /* Check the received character */ if (true == g_tx_flag) { g_tx_flag = false; /* Transmit the data */ /* Called the R_SCI_B_UART_Write to write data to UART. */ err = g_uart0_p_api->write(&g_uart0_ctrl, &g_tx_char, 1); assert(FSP_SUCCESS == err); while (false == g_tx_done) { /* Wait */ } } } #if BSP_PZ_SECURE_BUILD /* Enter non-secure code */ R_BSP_NonSecureEnter(); #endif } </pre>

Figure 7-44 Main Processing

7.2.9.3 Comparison of the Callback Functions

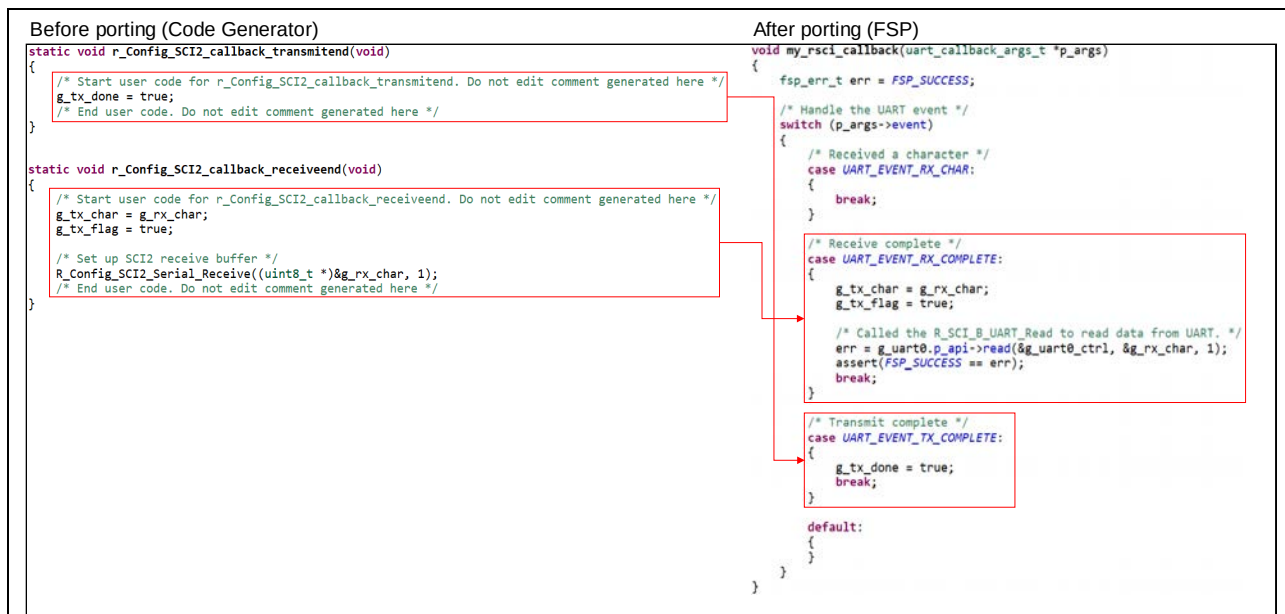


Figure 7-45 Callback Functions

8. Examples of Porting Projects Created without Using the Smart Configurator to FSP Projects

This chapter takes two sample programs that are created without using the Smart Configurator based on [the Initial Settings Example application note](#) as examples and describes the methods of porting them to FSP projects.

8.1 Example of Porting a Sample Program that Does Not Use Interrupt Functions

The following describes the procedure for porting a sample program that does not use interrupt functions (hereafter called the source project) to an FSP project.

8.1.1 Structure of the Folders and Files in the Source Project and the Targets for Porting

The following shows the structure of the folders and files in the source project and the target files for porting. main.c, led_define.h, and led_control.c are the targets for porting.

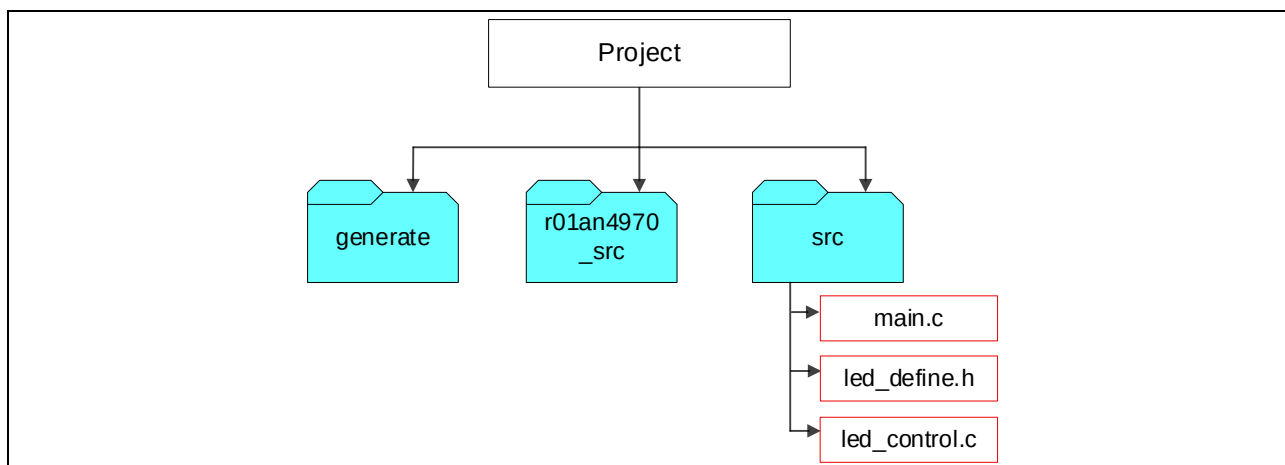


Figure 8-1 Structure of the Folders and Files in the Source Project and the Targets for Porting

8.1.2 Overview of the Processing in the Source Project

The following listing is of main.c.

src\main.c

```

#include <stdint.h>
#include "led_define.h"

static void led_blinking(void);

int main(void)
{
    /* ---- Stopping the peripherals which start operations ---- */
    R_INIT_StopModule();

    /* ---- Initialization of the non-existent ports ---- */
    R_INIT_Port_Initialize();

    /* ---- Initialization of the clock ---- */

```

```
R_INIT_Clock();

/* ---- Initialization of the ROM Cache ---- */
R_INIT_ROM_Cache();

/* Initialize for application */
led_port_init();

while(1)
{
    // TODO: add application code here
    led_blinking();
}

return 0;
}

/*****
* Function Name: led_blinking
* Description   : Blink the User LED.
* Arguments    : none
* Return Value  : none
*****/
static void led_blinking(void)
{
    uint32_t volatile i;

    led_output(LED_ON);

    /* wait */
    for(i = 0; i <= 10000000; i++)
    {

    }

    led_output(LED_OFF);

    /* wait */
    for(i = 0; i <= 10000000; i++)
    {
```

```
}
}
```

src\led_define.h

```
#ifndef LED_DEFINE_H_
#define LED_DEFINE_H_

/*****
Macro definitions
*****/

#define LED_ON          (0U)
#define LED_OFF         (1U)

/*****
Functions Prototypes
*****/

extern void led_port_init(void);
extern void led_output(uint8_t level);

#endif /* LED_DEFINE_H_ */
```

led_control.c

```
/*****
Includes  <System Includes> , "Project Includes"
*****/

#include <stdint.h>
#include "iodefine.h"
#include "led_define.h"

/*****
Macro definitions
*****/

#define PORT_OUTPUT      (1U)
#define USER_LED        (PORT4.PODR.BIT.B0)

/*****
Exported global variables and functions (to be accessed by other files)
*****/

void led_port_init(void);
void led_output(uint8_t level);
```

```

/*****
* Function Name: led_port_init
* Description  : Initialize the User LED port.
* Arguments    : none
* Return Value : none
*****/
void led_port_init(void)
{
    /* initialize port */
    USER_LED = LED_OFF;
    PORT4.PDR.BIT.B0 = PORT_OUTPUT;
    PORT4.PMR.BIT.B0 = 0U;
}

/*****
* Function Name: led_output
* Description  : Turns the User LED on or off.
* Arguments    : level : output level
* Return Value : none
*****/
void led_output(uint8_t level)
{
    USER_LED = level;
}

```

The program in the source project causes an LED to blink.

8.1.3 Creating the Destination Project for Porting

Use the procedure described in Chapter 5, Steps in Porting a Pre-FSP Project to an FSP Project, to create the destination project for porting. The new FSP project created with reference to the text from section 5.1, Creating a New FSP Project to section 5.4, Porting Modules, is used as the base project.

8.1.4 Structure of the Folders and Files in the Project after Porting and the Targets for Modification

The following shows the structure of the folders and files in the project after porting and the target files for modification. `hal_entry.c` and `led_control.c` are the targets for modification.

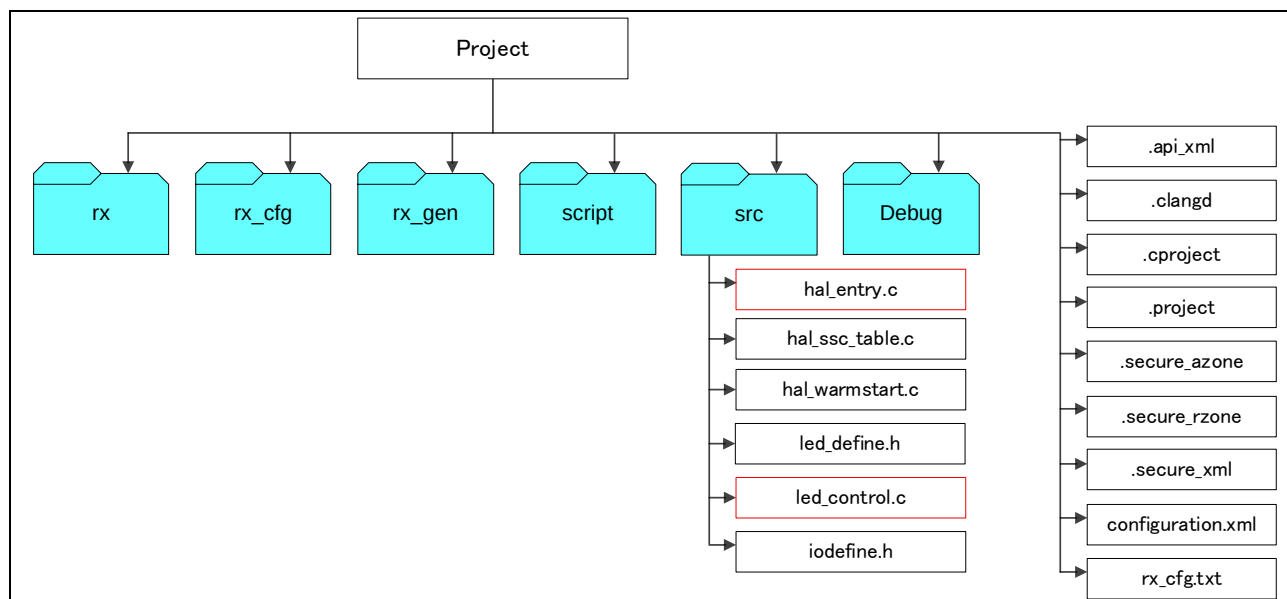


Figure 8-2 Structure of the Folders and Files in the Project after Porting and the Targets for Modification

8.1.5 Key Points in Porting

Use the procedure described in section 5.7, Porting the Source Code, to port the processing of the main function body and related processing in main.c to hal_entry.c. The processing in the main function provided in [the Initial Settings Example application note](#) contains some initialization functions, such as R_INIT_Clock() for initializing the clocks. However, in an FSP project, the initialization processing including clock configuration is completed before execution reaches the main function. If the initialization functions including R_INIT_Clock() are executed in this situation, the related settings will conflict with the clock settings made by the FSP project, so do not port the main processing provided in the *Initial Settings Example* application note. led_define.h and led_control.c also require porting. Some of the definitions in led_control.c use definitions provided by iodef.h, so add iodef.h by following the procedure described in section 5.8, Adding iodef.h. PORT40 is used in the source project but change it to PORTF06 in the destination project because the target device differs between the source and destination projects.

Refer to section 5.9, Replacing Processing with FSP Module Processing, and replace the processing of the source project with the equivalent processing by the FSP.

8.1.6 Processing after Replacement of the Target Files for Modification

The following listing is of the final contents of hal_entry.c and led_control.c after modification.

hal_entry.c (The final results of addition are shown in red.)

```
#include "hal_data.h"
#include "led_define.h"

FSP_CPP_HEADER
FSP_CPP_FOOTER

static void led_blinking(void);

/*****
```

```

* main() is generated by the RX Configuration editor and is used to generate
* threads if an RTOS is used. This function is called by main() when no
* RTOS is used.
*****/
void hal_entry(void)
{
    /* TODO: add your own code here */
    /* Initialize for application */
    led_port_init();

    while(1)
    {
        // TODO: add application code here
        led_blinking();
    }

#ifdef BSP_PZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

/*****
* Function Name: led_blinking
* Description   : Blink the User LED.
* Arguments    : none
* Return Value  : none
*****/
static void led_blinking(void)
{
    uint32_t volatile i;

    led_output(LED_ON);

    /* wait */
    for(i = 0; i <= 10000000; i++)
    {

    }

    led_output(LED_OFF);

```

```

    /* wait */
    for(i = 0; i <= 10000000; i++)
    {

    }
}

```

led_control.c (The final results of modification are shown in red.)

```

/*****
Includes  <System Includes> , "Project Includes"
*****/

#include <stdint.h>
#include "iodefine.h"
#include "led_define.h"

/*****
Macro definitions
*****/

#define PORT_OUTPUT      (1U)
#define USER_LED        (PORTF.PODR.BIT.PODR06)

/*****
Exported global variables and functions (to be accessed by other files)
*****/

void led_port_init(void);
void led_output(uint8_t level);

/*****
* Function Name: led_port_init
* Description  : Initialize the User LED port.
* Arguments    : none
* Return Value : none
*****/

void led_port_init(void)
{
    /* initialize port */
    USER_LED = LED_OFF;
    PORTF.PDR.BIT.PDR06 = PORT_OUTPUT;
    PFS.PF06PFS.BIT.PMR = 0U;
}

```

```

}

/*****
 * Function Name: led_output
 * Description  : Turns the User LED on or off.
 * Arguments    : level : output level
 * Return Value : none
 *****/
void led_output(uint8_t level)
{
    USER_LED = level;
}

```

8.2 Example of Porting a Sample Program Using Interrupt Functions

The following describes the procedure for porting a sample program that uses interrupt functions (hereafter called the source project) to an FSP project.

8.2.1 Structure of the Folders and Files in the Source Project and the Targets for Porting

The following shows the structure of the folders and files in the source project and the target files for porting. main.c, led_define.h, led_control.c, cmt_define.h, and cmt_control.c are the targets for porting.

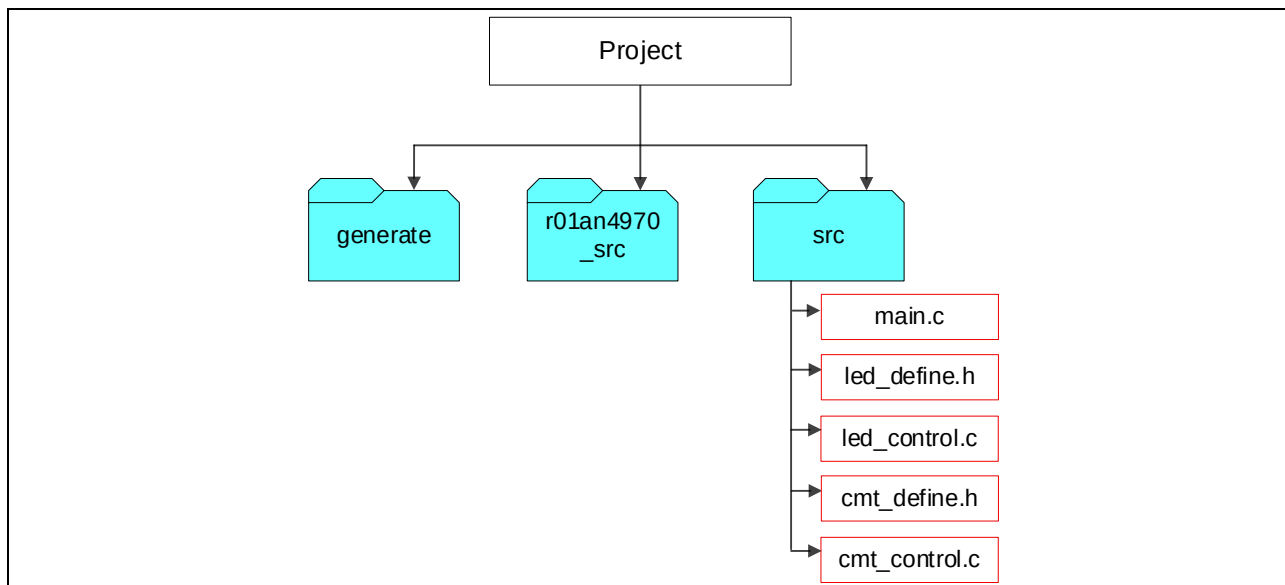


Figure 8-3 Structure of the Folders and Files in the Source Project and the Targets for Porting

8.2.2 Overview of the Processing in the Source Project

The following listing is of main.c.

src\main.c

```

#include <stdint.h>
#include "led_define.h"
#include "cmt_define.h"

```

```

int main(void)
{
    /* ---- Stopping the peripherals which start operations ---- */
    R_INIT_StopModule();

    /* ---- Initialization of the non-existent ports ---- */
    R_INIT_Port_Initialize();

    /* ---- Initialization of the clock ---- */
    R_INIT_Clock();

    /* ---- Initialization of the ROM Cache ---- */
    R_INIT_ROM_Cache();

    /* Initialize for application */
    led_port_init();
    cmt_initialize();
    cmt_start();

    while(1)
    {
        // TODO: add application code here
    }

    return 0;
}

```

led_define.h

```

#ifndef LED_DEFINE_H_
#define LED_DEFINE_H_

/*****
Macro definitions
*****/

#define LED_ON          (0U)
#define LED_OFF         (1U)

/*****
Functions Prototypes

```

```

*****/
extern void led_port_init(void);
extern void led_output(uint8_t level);
extern void led_output_inversion(void);

#endif /* LED_DEFINE_H_ */

```

led_control.c

```

/*****
Includes    <System Includes> , "Project Includes"
*****/

#include <stdint.h>
#include "iodefine.h"
#include "led_define.h"

/*****
Macro definitions
*****/

#define PORT_OUTPUT      (1U)
#define USER_LED        (PORT4.PODR.BIT.B0)

/*****
Exported global variables and functions (to be accessed by other files)
*****/

void led_port_init(void);
void led_output(uint8_t level);
void led_output_inversion(void);

/*****
* Function Name: led_port_init
* Description   : Initialize the User LED port.
* Arguments    : none
* Return Value  : none
*****/

void led_port_init(void)
{
    /* initialize port */
    USER_LED = LED_OFF;
    PORT4.PDR.BIT.B0 = PORT_OUTPUT;
    PORT4.PMR.BIT.B0 = 0U;
}

```

```

}

/*****
 * Function Name: led_output
 * Description  : Turns the User LED on or off.
 * Arguments    : level : output level
 * Return Value : none
 *****/
void led_output(uint8_t level)
{
    USER_LED = level;
}

/*****
 * Function Name: led_output_inversion
 * Description  : Inverting the LED output.
 * Arguments    : none
 * Return Value : none
 *****/
void led_output_inversion(void)
{
    if(USER_LED == LED_OFF)
    {
        led_output(LED_ON);
    }
    else
    {
        led_output(LED_OFF);
    }
}

```

cmt_define.h

```

#ifndef CMT_DEFINE_H_
#define CMT_DEFINE_H_

/*****
Functions Prototypes
 *****/
extern void cmt_initialize(void);
extern void cmt_start(void);

```

```
extern void cmt_cb_isr(void);

#endif /* CMT_DEFINE_H_ */
```

cmt_control.c

```

/*****
Includes  <System Includes> , "Project Includes"
*****/
#include <stdint.h>
#include "iodefine.h"
#include "led_define.h"

/*****
Exported global variables and functions (to be accessed by other files)
*****/
void cmt_initialize(void);
void cmt_start(void);
void cmt_cb_isr(void);

/*****
* Function Name: cmt_initialize
* Description  : Initialize the cmt.
* Arguments    : none
* Return Value : none
*****/
void cmt_initialize(void)
{
    /* initialize cmt */
    /*** Peripheral module settings ***/
    SYSTEM.PRCR.WORD = 0xA502U;
    SYSTEM.MSTPCRA.BIT.MSTPA15 = 0U;
    SYSTEM.PRCR.WORD = 0xA500U;

    IR(CMT0, CMI0) = 0U;
    IPR(CMT0, CMI0) = 0x0FU;

    /* The CMT interrupt interval is set to 0.5 seconds(2 Hz). */
    CMT0.CMCR.WORD = 0x0043U; /* PCLKB:60MHz/512 = 117.1875kHz */
    CMT0.CMCOR = 58593U - 1U;
}

```

```

/*****
* Function Name: cmt_start
* Description   : Start counting CMT.
* Arguments    : none
* Return Value  : none
*****/
void cmt_start(void)
{
    CMT.CMSTR0.BIT.STR0 = 1U;
    IEN(CMT0, CMI0) = 1U;
}

/*****
* Function Name: cmt_cb_isr
* Description   : Callback for CMT interrupt.
* Arguments    : none
* Return Value  : none
*****/
void cmt_cb_isr(void)
{
    led_output_inversion();
}

```

inthandler.c

```

#include "interrupt_handlers.h"
#include "cmt_define.h"

// CMT0 CMI0
void INT_Except_CMT0_CMI0(void)
{
    cmt_cb_isr();
}

```

The program in the source project uses the CMT to cause an LED to blink at a frequency of 2 Hz.

8.2.3 Creating the Destination Project for Porting

Use the procedure described in Chapter 5, Steps in Porting a Pre-FSP Project to an FSP Project, to create the destination project for porting. The new FSP project created with reference to the text from section 5.1, Creating a New FSP Project to section 5.4, Porting Modules, is used as the base project.

8.2.4 Adding the Interrupt Configuration

The source project uses the CMT interrupt in making the LED periodical blink. In the destination project, refer to section 5.5.2, Interrupt Configuration through Direct Addition, and configure the CMT interrupt on the [Interrupts Configuration] page.

Select [CMT0] from [New User Event] on the [Interrupts Configuration] page.

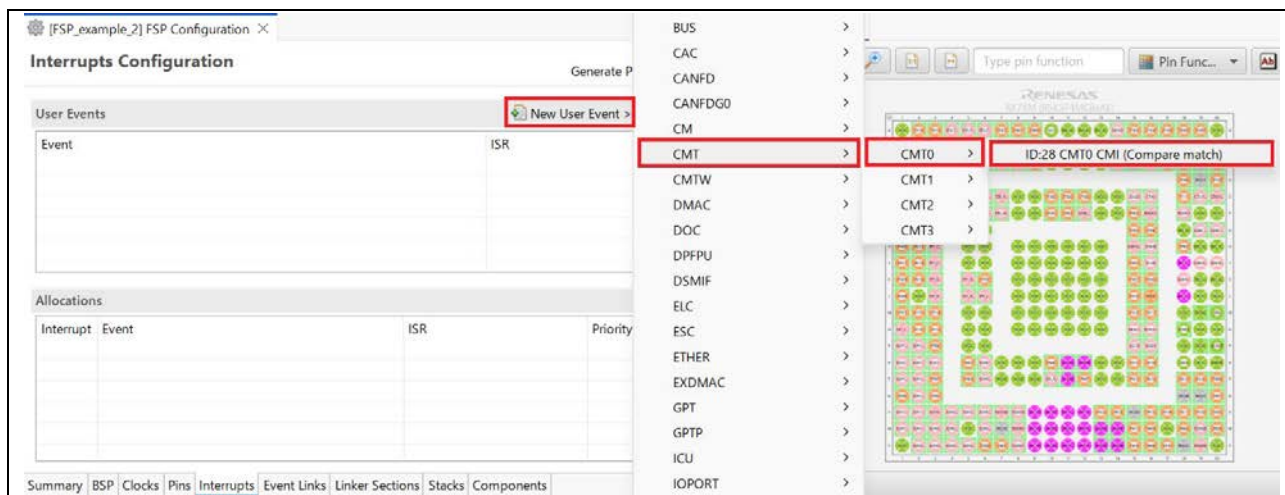


Figure 8-4 Selecting the CMT0 Interrupt Source on the [Interrupts Configuration] Page

Enter the name of the interrupt function. In this example, the name is set to "cmt0_isr". Click on the [OK] button.

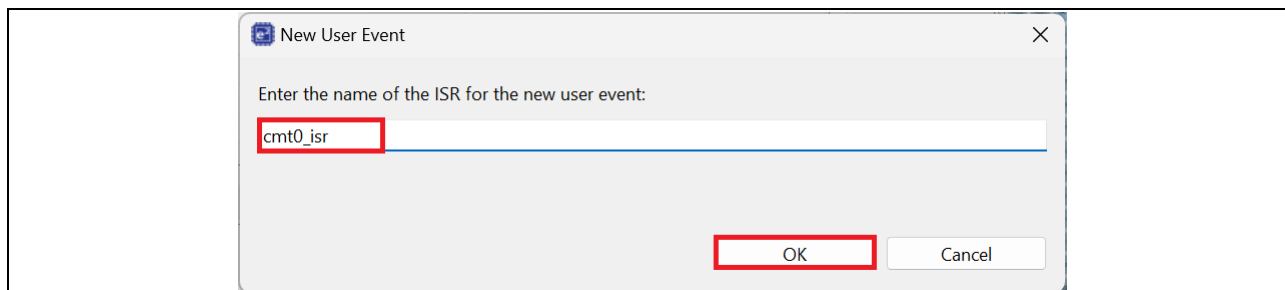


Figure 8-5 Entering the Name of the CMT0 Interrupt Function on the [Interrupts Configuration] Page

The information on the CMT0 interrupt will be registered.

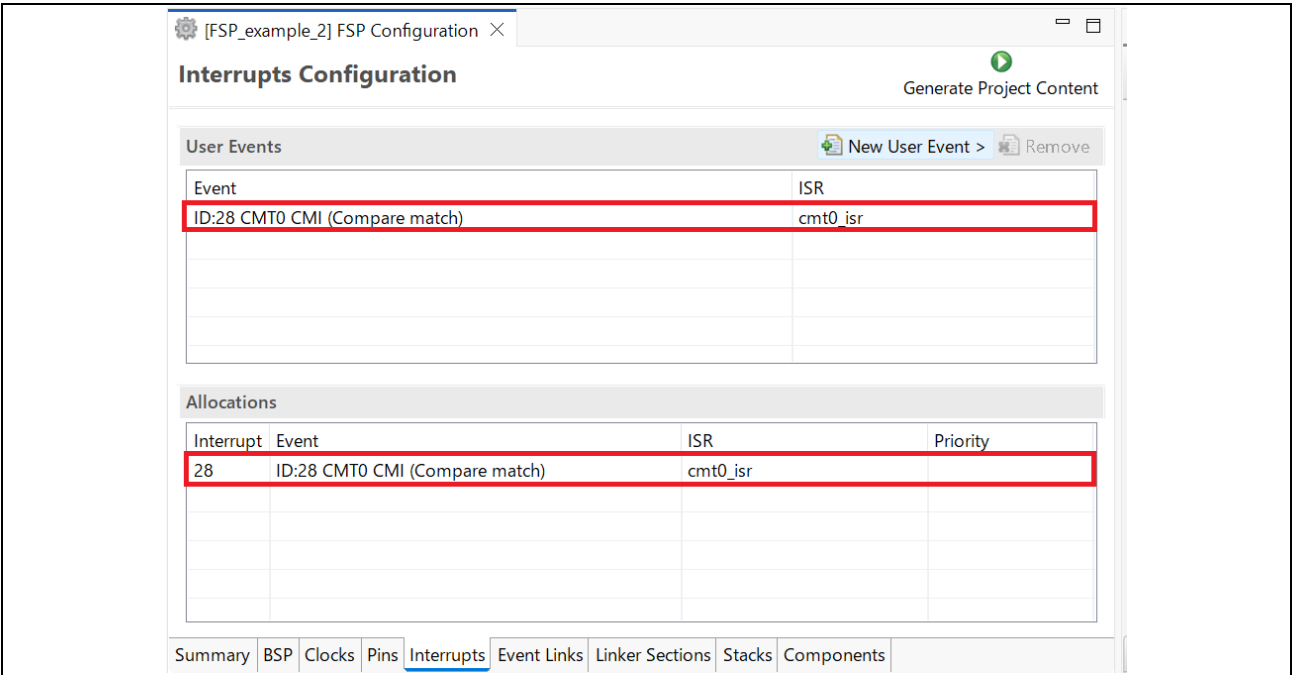


Figure 8-6 CMT0 Interrupt Added to the [Interrupts Configuration] Page

After checking the registered interrupt, execute [Generate Project Content]. The declaration of the CMT0 interrupt function will be generated.

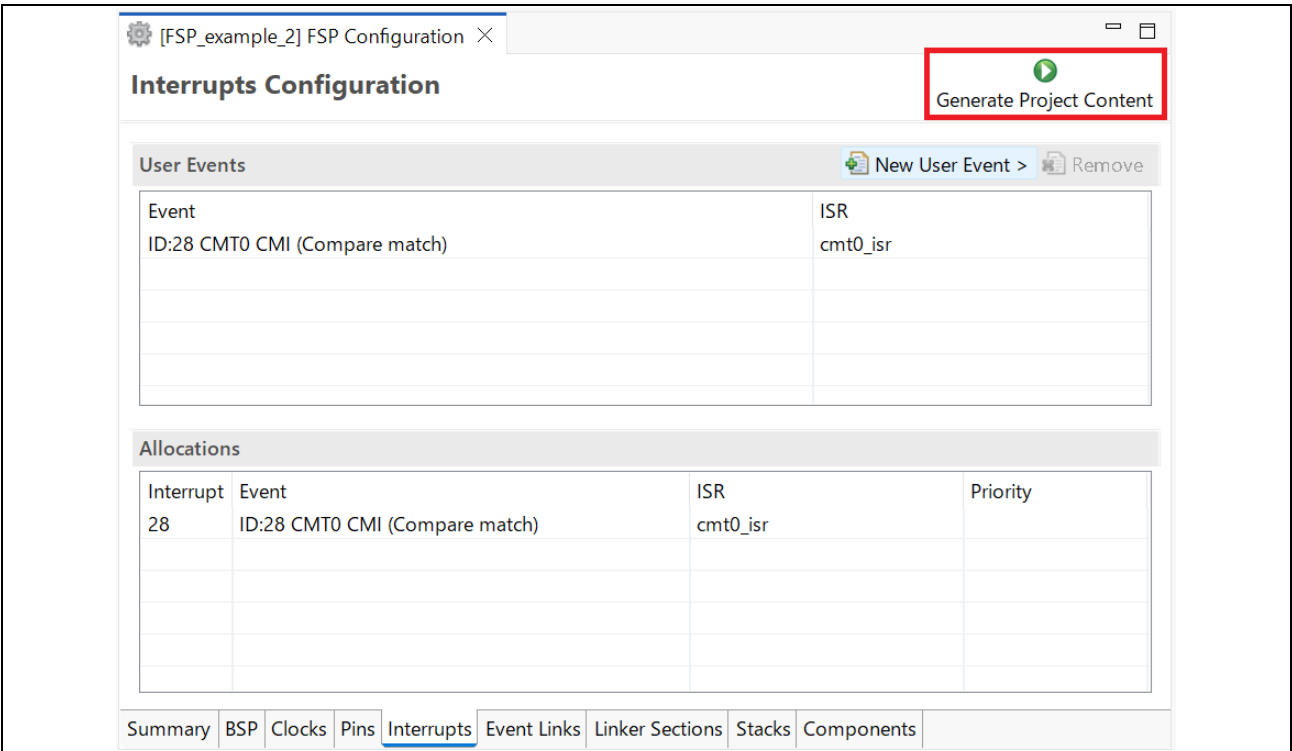


Figure 8-7 Generating the CMT0 Interrupt on the [Interrupts Configuration] Page

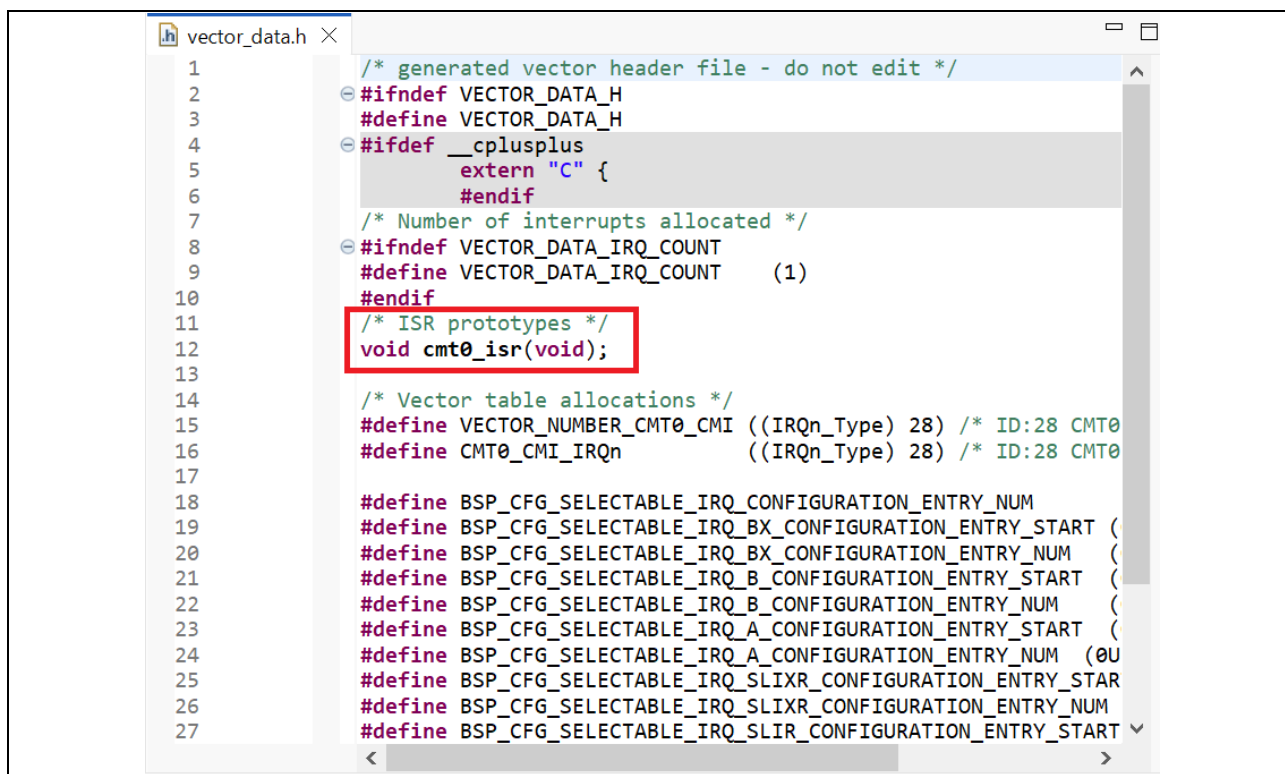


Figure 8-8 Declaration of the CMT0 Interrupt Function in vector_data.h

Only the declaration of the interrupt function is generated, and the user has to add the definition (body) of the interrupt function later.

8.2.5 Structure of the Folders and Files in the Project after Porting and the Targets for Modification

The following shows the structure of the folders and files in the project after porting and the target files for modification. hal_entry.c, led_control.c, and cmt_control.c are the targets for modification.

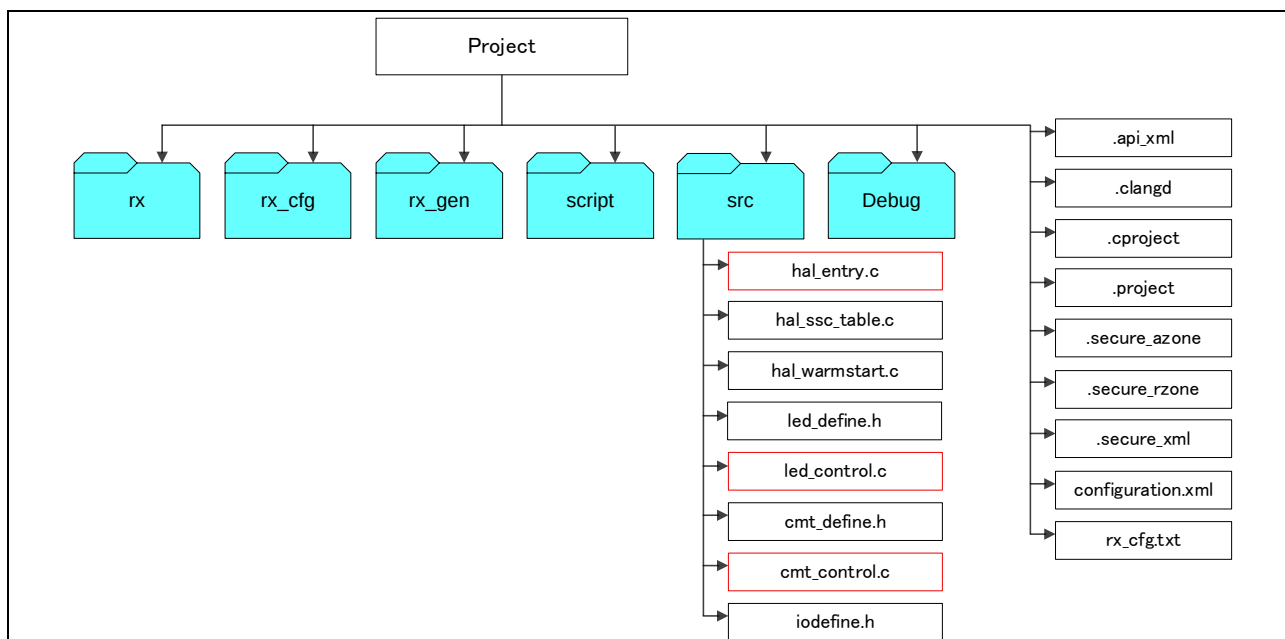


Figure 8-9 Structure of the Folders and Files in the Project after Porting and the Targets for Modification

8.2.6 Key Points in Porting

Use the procedure described in section 5.7, Porting the Source Code, to port the processing of the main function body in main.c to hal_entry.c. The processing in the main function provided in [the Initial Settings Example application note](#) contains some initialization functions, such as R_INIT_Clock() for initializing the clocks. However, in an FSP project, the initialization processing, including clock configuration, is completed before execution reaches the main function. If the initialization functions, including R_INIT_Clock(), are executed in this situation, the related settings will conflict with the clock settings made by the FSP project, so do not port the main processing provided in the *Initial Settings Example* application note.

Add the CMT interrupt function to hal_entry.c.

led_define.h, led_control.c, cmt_define.h, and cmt_control.c also require porting. Some of the definitions in led_control.c and cmt_control.c use definitions provided by iodef.h, so add iodef.h by following the procedure described in section 5.8, Adding iodef.h. PORT40 is used in the source project but change it to PORTF06 in the destination project because the target device differs between the source and destination projects.

Refer to section 5.9, Replacing Processing with FSP Module Processing, and replace the processing of the source project with the equivalent processing by the FSP.

8.2.7 Processing after Replacement of the Target Files for Modification

The following listing is of the final contents of hal_entry.c, led_control.c, and cmt_control.c after modification.

hal_entry.c (The final results of addition are shown in red.)

```
#include "hal_data.h"
#include "led_define.h"
#include "cmt_define.h"

FSP_CPP_HEADER
FSP_CPP_FOOTER

/*****
 * main() is generated by the RX Configuration editor and is used to generate
 * threads if an RTOS is used. This function is called by main() when no
 * RTOS is used.
 *****/
void hal_entry(void)
{
    /* TODO: add your own code here */
    /* Initialize for application */
    led_port_init();
    cmt_initialize();
    cmt_start();

    while(1)
    {
        // TODO: add application code here
    }
}
```

```

    }

#ifdef BSP_PZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

/*****
 * Function Name: cmt0_isr
 * Description  : Interrupt function for cmt0.
 * Arguments    : none
 * Return Value : none
 *****/
void cmt0_isr(void)
{
    cmt_cb_isr();
}

```

led_control.c (The final results of modification are shown in red.)

```

/*****
Includes  <System Includes> , "Project Includes"
*****/
#include <stdint.h>
#include "iodefine.h"
#include "led_define.h"

/*****
Macro definitions
*****/
#define PORT_OUTPUT          (1U)
#define USER_LED            (PORTF.PODR.BIT.PODR06)

/*****
Exported global variables and functions (to be accessed by other files)
*****/
void led_port_init(void);
void led_output(uint8_t level);
void led_output_inversion(uint8_t level);

```

```

/*****
* Function Name: led_port_init
* Description   : Initialize the User LED port.
* Arguments    : none
* Return Value  : none
*****/
void led_port_init(void)
{
    /* initialize port */
    USER_LED = LED_OFF;
    PORTF.PDR.BIT.PDR06 = PORT_OUTPUT;
    PFS.PF06PFS.BIT.PMR = 0U;
}

/*****
* Function Name: led_output
* Description   : Turns the User LED on or off.
* Arguments    : level : output level
* Return Value  : none
*****/
void led_output(uint8_t level)
{
    USER_LED = level;
}

/*****
* Function Name: led_output_inversion
* Description   : Inverting the LED output.
* Arguments    : none
* Return Value  : none
*****/
void led_output_inversion(void)
{
    if(USER_LED == LED_OFF)
    {
        led_output(LED_ON);
    }
    else
    {
        led_output(LED_OFF);
    }
}

```

}

cmt_control.c (The final results of modification are shown in red.)

```

/*****
Includes  <System Includes> , "Project Includes"
*****/
#include <stdint.h>
#include "iodefine.h"
#include "led_define.h"

/*****
Exported global variables and functions (to be accessed by other files)
*****/
void cmt_initialize(void);
void cmt_start(void);
void cmt_cb_isr(void);

/*****
* Function Name: cmt_initialize
* Description  : Initialize the cmt.
* Arguments    : none
* Return Value : none
*****/
void cmt_initialize(void)
{
    /* initialize cmt */
    /*** Peripheral module settings ***/
    SYSC_NS.PRCR_NS.WORD = 0xA502U;
    MSTPREG.MSTPCR.F.BIT.MSTPF15 = 0U;
    SYSC_NS.PRCR_NS.WORD = 0xA500U;

    IR(CMT0, CMI0) = 0U;
    IPR(CMT0, CMI0) = 0x0FU;

    /* The CMT interrupt interval is set to 0.5 seconds(2 Hz). */
    CMT0.CMCR.WORD = 0x0043U; /* PCLKB:60MHz/512 = 117.1875kHz */
    CMT0.CMCOR = 58593U - 1U;
}

/*****
* Function Name: cmt_start

```

```
* Description : Start counting CMT.
* Arguments   : none
* Return Value : none
*****/

void cmt_start(void)
{
    CMT.CMSTR0.BIT.STR0 = 1U;
    IEN(CMT0, CMI0) = 1U;
}

/*****
* Function Name: cmt_cb_isr
* Description : Callback for CMT interrupt.
* Arguments   : none
* Return Value : none
*****/

void cmt_cb_isr(void)
{
    led_output_inversion();
}
```

9. Notes on Porting to an FSP Project

This chapter provides notes on porting a source project created by using the RX Smart Configurator or FIT software modules, or a source project created without using these Renesas software products, to a destination project handled by using the newer FSP.

9.1 Addition and Conflicts of a BSP

A BSP is automatically added to an FSP project as the base of the project when the project is created.

The BSP contains the startup processing, which is to be executed after release from the reset state. In porting to an FSP project, it is assumed that the functions provided by the FSP project, such as the startup processing, interrupt processing, and pin configuration, are used. Accordingly, if the startup processing or interrupt processing in the source project is ported without change, this processing will be in conflict with the BSP processing, resulting in incorrect operation.

9.2 Differences between and Conflicts with the I/O Register Definition Files

The I/O registers in an existing project are only defined in `iodef.h`. However, the I/O registers in an FSP project are defined in a series of header files under `renesas.h` (see Figure 2-32, Structure of the Files in the bsp Folder).

Regarding the processing for access to I/O registers in an FSP project, reference to `renesas.h` contained in the BSP of the FSP project is recommended in most cases.

As the I/O register definitions differ between `iodef.h` and `renesas.h`, the code for access to I/O registers also differ in the way shown in the code below.

Sample code for register access with reference to `iodef.h`

```
CMT.CMSTR0.WORD = 0x0001;
WDT.WDTRR = 0x00;
```

Sample code for register access with reference to `renesas.h`

```
R_CMT->UNT->CMSTR0 = 0x0000;
R_WDT->WDTRR = 0x00;
```

Another method of porting the processing for access to I/O registers that involves the use of `iodef.h` from an existing project to an FSP project is to add `iodef.h` to the FSP project. In this case, the FSP project after porting holds two files of definitions: the existing I/O register definition file (`iodef.h`) and the I/O register definition file (`renesas.h`) for the FSP project.

Even if the definitions in both files are used together in a project, they will not conflict because of the difference of definitions between `iodef.h` and `renesas.h`. The following shows excerpts of definitions from `iodef.h`, `renesas.h`, and the header files related to `renesas.h`.

Definitions in `iodef.h`

```
(Omitted)
#define CMT (*(volatile struct st_cmt __evenaccess *)0x88000)
#define CMT0 (*(volatile struct st_cmt0 __evenaccess *)0x88002)
(Omitted)
#define WDT (*(volatile struct st_wdt __evenaccess *)0x88020)
(Omitted)
typedef struct st_cmt {
```

```

    union {
        unsigned short WORD;
        struct {
            unsigned short :14;
            unsigned short STR1:1;
            unsigned short STR0:1;
        } BIT;
    } CMSTR0;
    char          wk0[14];
    union {
        unsigned short WORD;
        struct {
            unsigned short :14;
            unsigned short STR3:1;
            unsigned short STR2:1;
        } BIT;
    } CMSTR1;
} st_cmt_t;
(Omitted)
typedef struct st_wdt {
    unsigned char  WDTRR;
    char          wk0[1];
    union {
        unsigned short WORD;
        struct {
            unsigned short :2;
            unsigned short RPSS:2;
            unsigned short :2;
            unsigned short RPES:2;
            unsigned short CKS:4;
            unsigned short :2;
            unsigned short TOPS:2;
        } BIT;
    } WDTCR;
    union {
        unsigned short WORD;
        struct {
            unsigned short REFEF:1;
            unsigned short UNDFE:1;
            unsigned short CNTVAL:14;
        } BIT;
    }

```

```

    } WDTSR;
    union {
        unsigned char BYTE;
        struct {
            unsigned char RSTIRQS:1;
            unsigned char :7;
        } BIT;
    } WDTRCR;
} st_wdt_t;
(Omitted)

```

Definitions in renesas.h

```

(Omitted)
#if BSP_MCU_GROUP_RX74M
    #include "R5K574M.h"
#elif BSP_MCU_GROUP_RX74N
    #include "R5K574N.h"
#else
    #if __has_include("renesas_internal.h")
        #include "renesas_internal.h"
    #else
        #warning "Unsupported MCU"
    #endif
#endif
(Omitted)

```

Definitions in R5F574M.h

```

(Omitted)
#include "R5F574M/r_cmt_reg.h"
(Omitted)
#include "R5F574M/r_wdt_reg.h"
(Omitted)
#define R_CMT_BASE                (0x87826000UL + BASE_NS_OFFSET)
(Omitted)
#define R_WDT_BASE                (0x87802600UL + BASE_NS_OFFSET)
(Omitted)
#define R_CMT                    ((R_CMT_Type *) R_CMT_BASE)
(Omitted)
#define R_WDT                    ((R_WDT_Type *) R_WDT_BASE)
(Omitted)

```

Definitions in r_cmt_reg.h

```

(Omitted)
typedef struct
{
    union
    {
        __IOM uint16_t CMSTR0;          /*!< (@ 0x00000000) Compare Match Timer
                                         Start Register */

        struct
        {
#ifdef defined(__BIG) && defined(__GNUC__)
            uint16_t          : 14;
            __IOM uint16_t STR1 : 1;    /*!< [1..1] Count Start 0 */
            __IOM uint16_t STR0 : 1;    /*!< [0..0] Count Start 1 */
#else
            __IOM uint16_t STR0 : 1;    /*!< [0..0] Count Start 1 */
            __IOM uint16_t STR1 : 1;    /*!< [1..1] Count Start 0 */
            uint16_t          : 14;
#endif
        } CMSTR0_b;
    };
    __IOM R_CMT_UNT_CM_Type CM[2];      /*!< (@ 0x00000002) 2 Timer Start Register
                                         Pairs */

    __IM uint16_t RESERVED;
} R_CMT_UNT_Type;                      /*!< Size = 16 (0x10) */
(Omitted)

```

Definitions in r_wdt_reg.h

```

(Omitted)
typedef struct                                /*!< (@0x87802600) R_WDT Structure */
{
    union
    {
        __IOM uint8_t WDTRR;    /*!< (@ 0x00000000) WDT Refresh Register */

        struct
        {
            __IOM uint8_t WDTRR : 8;
        } WDTRR_b;
    };
}

```

```

};
__IM uint8_t RESERVED;
(Omitted)
union
{
    __IOM uint8_t WDTRCR;          /*!< (@ 0x00000006) WDT Reset Control */

    struct
    {
#ifdef defined(__BIG) && defined(__GNUC__)
        __IOM uint8_t RSTIRQS : 1; /*!< [7..7] Reset Interrupt Request Select */
        uint8_t          : 7;
#else
        uint8_t          : 7;
        __IOM uint8_t RSTIRQS : 1; /*!< [7..7] Reset Interrupt Request Select */
#endif
    } WDTRCR_b;
};
__IM uint8_t RESERVED1;
__IM uint16_t RESERVED2;
} R_WDT_Type;                    /*!< Size = 10 (0xa) */
(Omitted)

```

9.3 Porting to a Device that Does Not Provide the Same Peripheral Functions as the Source Device

If the target RX device of the destination project does not provide the same peripheral functions as those of the target device of the source project, replacing the device drivers is clearly not possible. In such cases, a new program has to be developed for the RX device of the destination project.

10. Reference Documents

- RX Flexible Software Package Documentation
- Firmware Integration Technology User's Manual (R01AN1833)
- RX Family CMT Module Using Firmware Integration Technology (R01AN1856)
- RX Family RSCI Module Using Firmware Integration Technology (R01AN5759)
- Smart Configurator User's Manual: RX API Reference (R20UT4360)
- RX72N Group Initial Settings Example Application Note (R01AN4970)
- RX140 Group Fast Prototyping Board for RX140 LED Blink Control Program Application Note (R01AN7266)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct.09.26	—	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.

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/.