RX Family

Board Support Package Module Using Firmware Integration Technology

Introduction

The foundation of any project that uses FIT modules is the Renesas Board Support Package (r_bsp). The r_bsp is easily configurable and provides all the code needed to get the MCU from reset to main(). The document covers conventions of the r_bsp so that users will know how to use it, configure it, and create a BSP for their own board.

Target Device

The following is a list of devices that are currently supported:

- RX110 Group
- RX111 Group
- RX113 Group
- RX130 Group
- RX231, RX230 Groups
- RX23T Group
- RX23W Group
- RX24T Group
- RX24U Group
- RX63N, RX631 Groups
- RX64M Group
- RX71M Group
- RX65N, RX651 Groups
- RX66T Group
- RX72M Group
- RX72T Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Target Compiler

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX (RX23W is not supported.)
- IAR C/C++ Compiler for Renesas RX (RX110, RX23W and RX72M are not supported.)

For details of the confirmed operation contents of each compiler, refer to “10.1 Confirmed Operation Environment".

Contents

1. Overview

Before running the user application there are a series of operations that must be performed to get the MCU set up properly. These operations, and the number of operations, will vary depending on the MCU being used. Common examples include: setting up stack(s), initializing memory, configuring system clocks, and setting up port pins. No matter the application, these steps need to be followed. To make this process easier the Renesas Board Support Package, abbreviated as r_bsp, is provided.

At the lowest level the r_bsp provides everything needed to get the user's MCU from reset to the start of their application's main() function. The r_bsp also provides common functionality that is needed by many applications. Examples of this include callbacks for exceptions and functions to enable or disable interrupts.

While every application will need to address the same steps after reset, this does not mean that the settings will be the same. Depending on the application, stack sizes will vary and which clock is used will change. The r_bsp configuration options are contained in the config header file for easy access.

Many customers start development on a Renesas development board and then transition to their own custom boards. When users move to their own custom hardware it is highly recommended they create a new BSP inside of the r_bsp. By following the same standards and rules that are used for the provided BSPs the user can get an early start on development knowing that their application code will move to their target board very easily. Details on how users can create their own BSPs are provided in this document.

1.1 Terminology

Term	Meaning
Platform	The user's development board. Used interchangeably with 'board'.
BSP	Short for Board Support Package. BSP's usually have source files related to a specific board.
Callback Function	This term refers to a function that is called when an event occurs. For example, the bus error interrupt handler is implemented in the r_bsp. The user will likely want to know when a bus error occurs. To alert the user, a callback function can be supplied to the r_bsp. When a bus error occurs the r_bsp will jump to the provided callback function and the user can handle the error. Interrupt callback functions should be kept short and be handled carefully because when they are called the MCU will still be inside of an interrupt and therefore will be delaying any pending interrupts.

1.2 File Structure

The r_bsp file structure is shown below in Figure 1.1. Underneath the root r_bsp folder there are 3 folders and 2 files. The first folder is named *doc* and contains r_bsp documentation.

The *board* folder contains the *generic* folder and the *user* folder. The *generic* folder contains source files whose settings are independent of the board and is provided for each MCU. The structures of the generic folder are shown in “Figure 1.2 Structures of generic Folder”. The *user* folder is merely a placeholder and, for example, can be used for the user boards.

The *mcu* folder has one folder per supported MCU. There is also a folder named *all* in this directory containing source that is common to all MCUs in the r_bsp.

The file *platform.h* is provided for the user to choose their current development platform. *platform.h*, in turn, selects all the proper header files from the *board* and *mcu* folders to be included in the user’s project. This is discussed in more detail in later sections. The *readme.txt* file is a standard text file that is provided with all FIT Modules that provides brief information about the r_bsp.

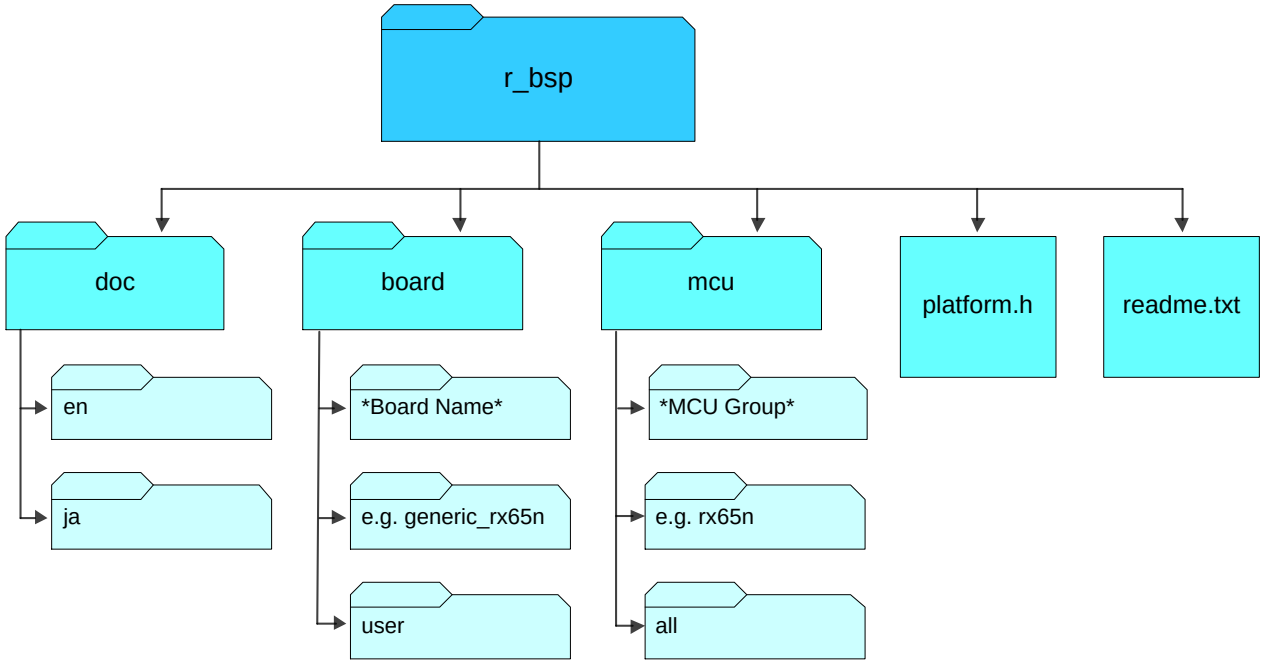


Figure 1.1 r_bsp File Structure

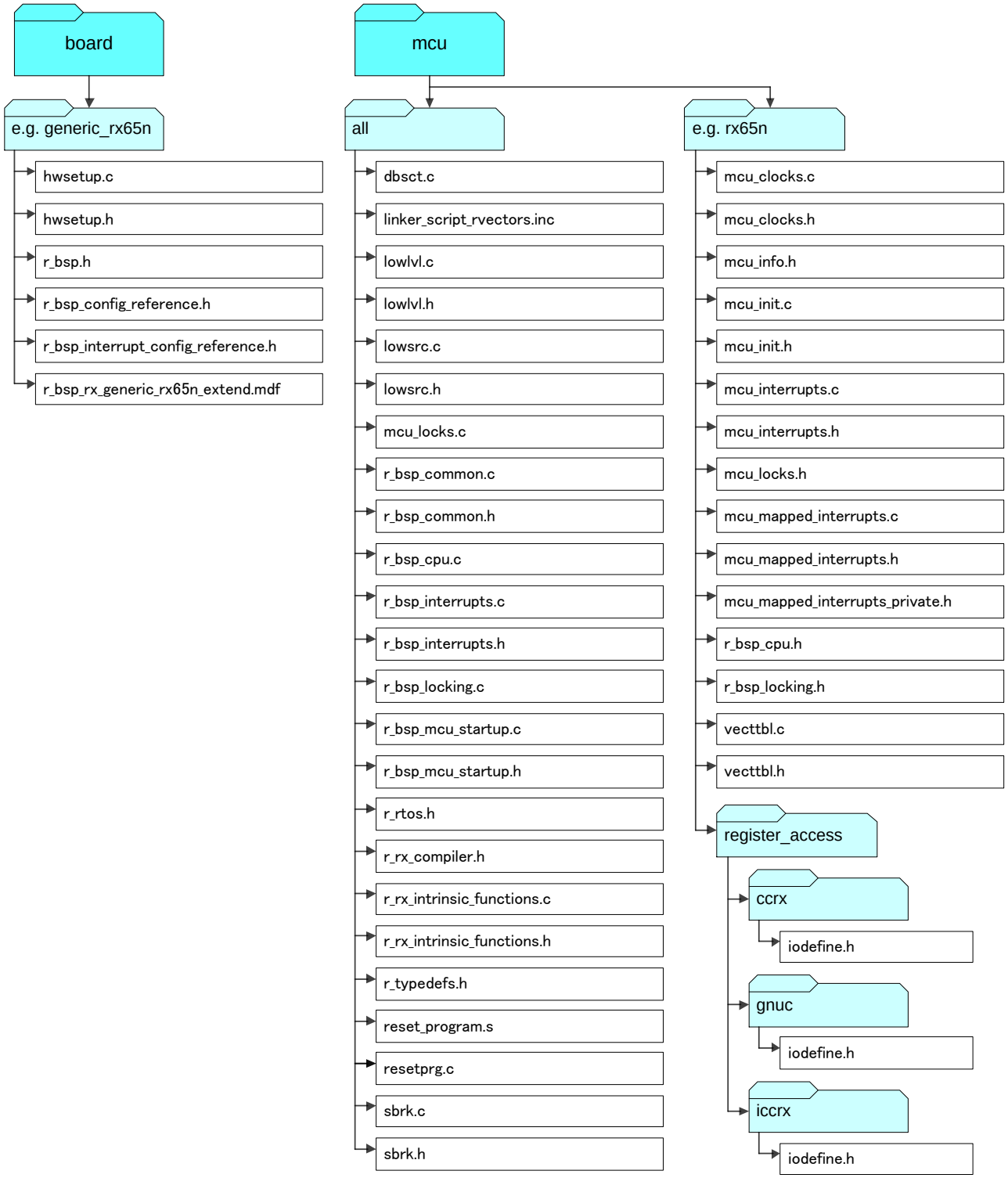


Figure 1.2 Structures of generic Folder

2. Features

This section will go into more detail on the features provided by the r_bsp.

2.1 MCU Information

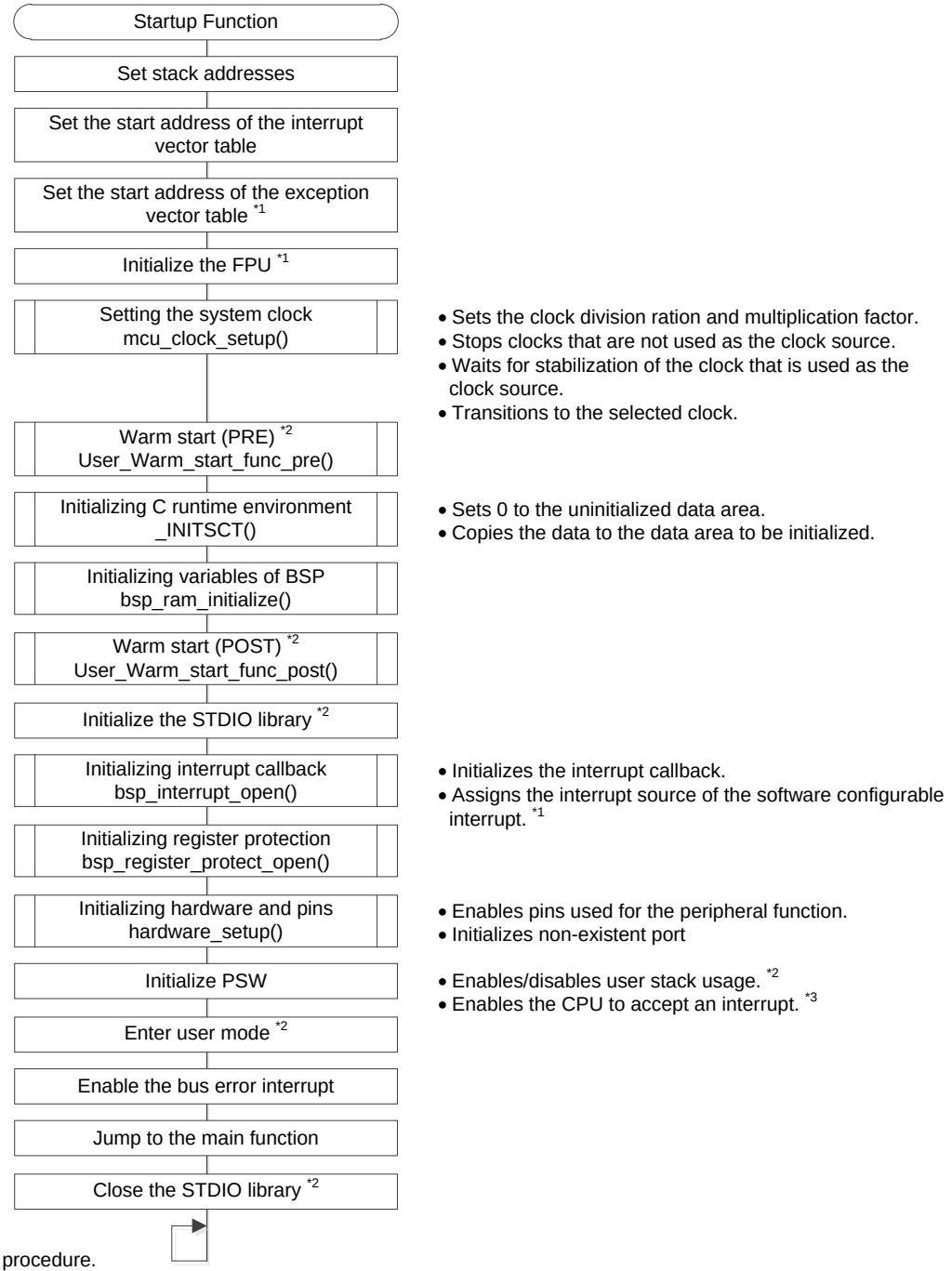
One of the main benefits of the r_bsp is that the user defines their global system settings only once, in a single place in the project. This information is defined in the r_bsp and then used by FIT Modules and user code. FIT Modules can use this information to automatically configure their code for the user's system configuration. If the r_bsp did not provide this information then the user would have to specify system information to each FIT Module separately.

Configuring the r_bsp is discussed in Section 3. The r_bsp uses this configuration information to set macro definitions in *mcu_info.h*. Each MCU may have different macros in *mcu_info.h*, but below are some common examples.

Define	Meaning
BSP_MCU_SERIES_<MCU_SERIES>	Which MCU Series this MCU belongs to. Example: BSP_MCU_SERIES_RX600 would be defined if the MCU was an RX64M.
BSP_MCU_<MCU_GROUP>	Which MCU Group this MCU belongs to. Example: BSP_MCU_RX111 would be defined if the MCU was an RX111.
BSP_PACKAGE_<PACKAGE_TYPE>	The package of the MCU. Example: BSP_PACKAGE_LQFP100 would be defined for a 100-pin LQFP package MCU.
BSP_PACKAGE_PINS	How many pins this MCU has.
BSP_ROM_SIZE_BYTES	The size of the user application ROM space in bytes.
BSP_RAM_SIZE_BYTES	The size of the RAM available to the user in bytes. In some MCUs the RAM area is not contiguous.
BSP_DATA_FLASH_SIZE_BYTES	The size of the data flash area in bytes.
BSP_<CLOCK>_HZ	There will be one of these macros for each clock on the MCU. Each macro will define that clock's frequency in hertz. Examples: BSP_LOCO_HZ defines the LOCO frequency in Hz. BSP_ICLK_HZ defines the CPU clock in Hz. BSP_PCLKB_HZ defines the Peripheral Clock B in Hz.
BSP_MCU_IPL_MAX	The maximum interrupt priority level for the MCU.
BSP_MCU_IPL_MIN	The minimum interrupt priority level for the MCU.
FIT_NO_FUNC **and** **FIT_NO_PTR**	These macros can be used as arguments in function calls to specify that nothing is being supplied for an argument. For example, if a function takes an optional argument for a callback function then FIT_NO_FUNC could be used if the user did not wish to supply a callback function. These macros are defined to point to reserved address space. This is done so that if the argument is used improperly it is easier to catch. The reason for this is that if the MCU attempts to access reserved space then a bus error will occur and the user will know immediately. If NULL was used instead then a bus error would not occur because NULL is typically defined as 0 which is a valid RAM location on the RX.

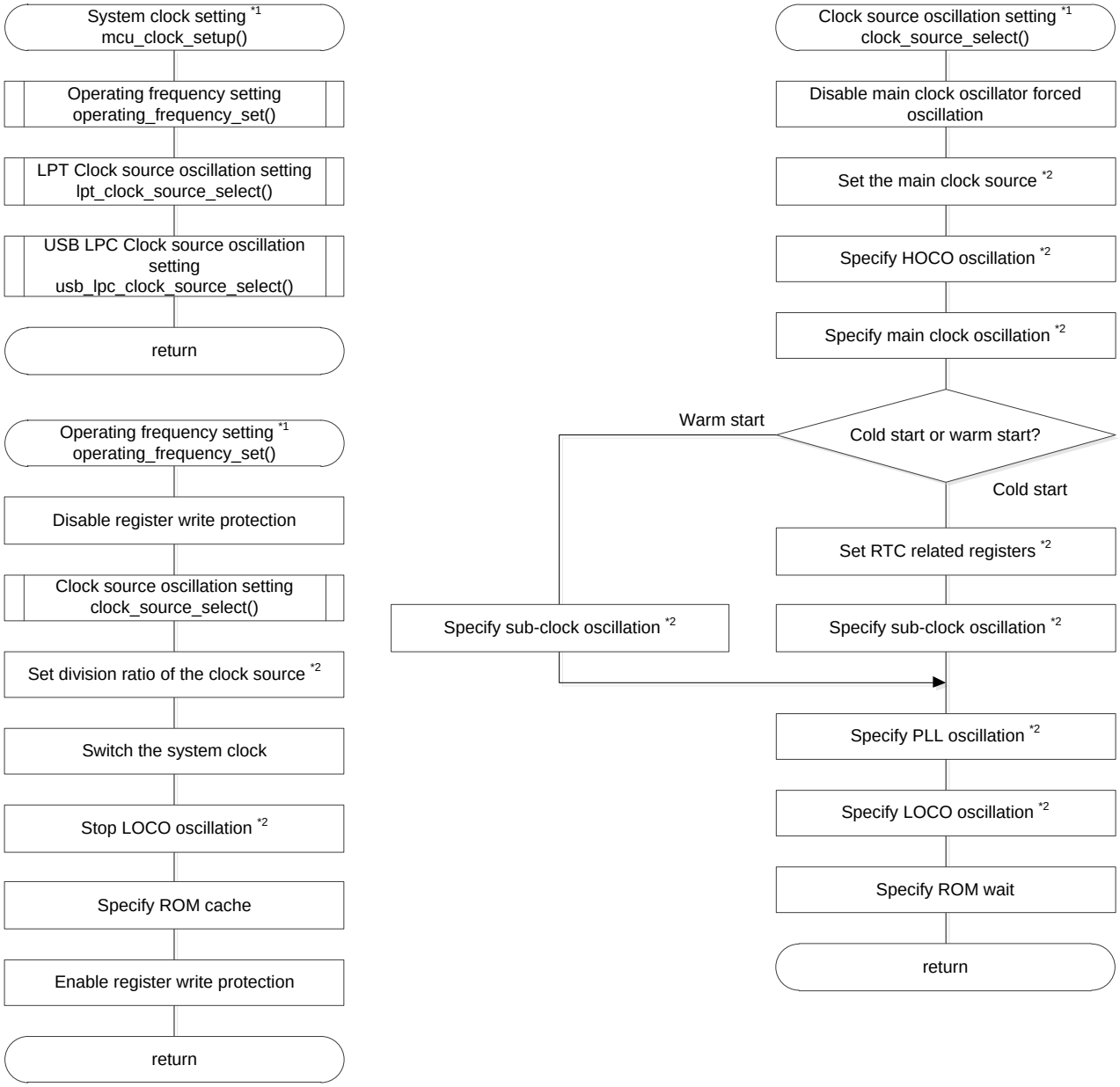
2.2 Initialization

When using the Renesas compiler and GCC, the PowerON_Reset_PC function is set as the reset vector for the MCU. When using the IAR compiler, the __iar_program_start function is set as the reset vector for the MCU.The PowerON_Reset_PC() function and the __iar_program_start function (startup function) performs a number of chip initialization actions to get the MCU ready to jump to the user's application. The flowchart below details operations of the startup function and the system clock setting.



Note 1: The MCU skips this procedure.
Note 2: The operation varies depending on the setting in the r_bsp_config.h.
Note 3: Only acceptance of CPU interrupts is enabled. Acceptance of each peripheral interrupt must be enabled separately.

Figure 2.1 Flowchart of startup function



Note 1: The procedure may vary depending on the MCU used.
Note 2: The operation varies according to settings in r_bsp_config.h.

Figure 2.2 Flowchart of System Clock Setting

2.3 Global Interrupts

Interrupts on RX MCUs are disabled out of reset. The startup function will enable interrupts before the user's application is called (see Section 2.2).

RXv1 devices have two vector tables: a relocatable vector table and a fixed vector table. As the names suggest, the relocatable vector table can be anywhere in memory and the fixed vector table is at a static location at the top of the memory map. RXv2 and RXv3 devices have two vector tables: an interrupt vector table and an exception vector table. The interrupt vector table and the exception vector table can be anywhere in memory.

The relocatable vector table and the interrupt vector table hold peripheral interrupt vectors and are pointed to by the INTB register. This register is initialized after reset in the startup function. The vectors in the relocatable vector table and the interrupt vector table are inserted by the RX toolchain. The RX toolchain knows about the user's interrupt vectors by the use of the '#pragma interrupt' directives in the user's code.

The fixed vector table holds exception vectors, the reset vector, as well as some flash-based option registers. The exception vector table holds exception vectors.

The fixed vector table and the exception vector table are defined in *vecttbl.c* along with default interrupt handlers for all exceptions, the NMI interrupt, bus errors, and undefined interrupts. The user has the option of dynamically setting callbacks (see Section 2.4) for all of these vectors using the functionality found in *mcu_interrupts.c*. The *vecttbl.c* file also takes care of setting up the User Boot reset vector when applicable.

All vectors in the fixed vector table and the exception vector table are handled in *vecttbl.c*. All vectors in the relocatable vector table and the interrupt vector table are not handled because the user will define these vectors and each application will be different. This means that in every application there will be unfilled vectors that should be taken care of in case that interrupt is triggered by accident. Many linkers support the filling of unused vectors with a static function. The undefined_interrupt_source_isr() function in *vecttbl.c* is provided for this purpose and the user is encouraged to set up the linker to fill in unused vectors with this function's address.

2.4 Interrupt Callbacks

The r_bsp provides several API functions (see Sections 5.13 and 5.14) which allow the user to be alerted when certain interrupts are triggered. This works by the user selecting the interrupt and then providing a callback function. When the interrupt is triggered the r_bsp will call the supplied callback function.

Currently, the user can choose to register callbacks for all exception interrupts in the fixed vector table and the exception vector table, the bus error interrupt, and the undefined interrupt. After the user callback function has been executed, the r_bsp interrupt handler will clear any interrupt flags as needed.

2.5 Non-Existent Port Pins

Within a MCU Group there can be many different packages with varying number of pins. For packages that have less pins than the maximum (e.g. 64 pin package in a MCU group that goes up to 144 pins), the non-bonded out pins can be initialized to lower power consumption. Based on the settings in *r_bsp_config.h*, the r_bsp will automatically initialize these non-bonded out pins during the MCU initialization procedure. This feature is implemented in the *mcu_init.c* function and is called by the hardware_setup() function.

2.6 Clock Setup

All system clocks are setup during r_bsp initialization. The clocks are configured based upon the user's settings in the *r_bsp_config.h* file (see Section 3.2.6). Clock configuration is performed prior to initializing the C runtime environment. This is done to quicken this process since some RX MCUs startup on a relatively slow clock. When selecting a clock the code in the r_bsp will implement the required delays to allow the selected clock to stabilize.

Some RX MCUs require a wait cycle to access the flash memory or the RAM. The wait cycle can be set by the MEMWAIT register or the ROMWT register. The setting values for these registers depend on the system clock or operating power control mode used. Make sure to confirm the limitations in the user's manual for setting the MEMWAIT register and the ROMWT register.

2.7 STDIO & Debug Console

When enabled (see Section 3.2.3), the STDIO library is initialized as part of the MCU initialization procedure. The r_bsp code is setup to send STDIO output to the debug console that can be viewed in e^2 studio. The source file *lowlvl.c* is responsible for sending and receiving bytes for STDIO functions and as previously stated is set up by default to use the debug console. If desired the user may redirect the STDIO charget() and/or charput() functions to their own respective functions by modifying r_bsp_config.h and enabling BSP_CFG_USER_CHARGET_ENABLED and/or BSP_CFG_USER_CHARPUT_ENABLED, and providing and replacing the my_sw_charget_function and/or my_sw_charput_function function names with the names of their own functions.
When using the Renesas compiler, it is possible to select whether STDIO is initialized or not. When using GCC and the IAR compilers, STDIO is always initialized.

2.8 Stacks Area and Heap Area

RX MCUs have two stacks that can be used: the User stack and the Interrupt stack. When both stacks are used the User stack will be used during normal execution flow and the Interrupt stack will be used during interrupt handling. Having two stacks can make it easier to figure out how much stack space to allocate since the user does not have to worry about always having enough room on the User stack for if-and-when an interrupt occurs. Some users will not want two stacks though because it is not needed in all applications and can lead to wasted RAM (i.e. space in between stacks that is not used). If only one stack is used then it will always be the Interrupt stack.
The User and Interrupt stacks and the heap are all set up and initialized after reset inside of the startup function. The sizes of the stacks and heap, and whether one or two stacks are used, is configured in *r_bsp_config.h* (see Section 3.2.2). The user also has the option of disabling the heap if desired.
When using the IAR compiler, set the stack and heap size not only with r_bsp_config.h but also with the GUI.

2.9 CPU Mode

Out of reset, RX MCUs operate in Supervisor CPU Mode. In Supervisor Mode all CPU resources and instructions are available. The user has the option (see Section 3.2.4) of transitioning to User Mode before the r_bsp code jumps to main(). In User Mode there are restrictions to any instruction capable of writing to:

- Some bits (bits IPL[3:0], PM, U, and I) in the processor status word (PSW)
- Interrupt stack pointer (ISP)
- Interrupt table register (INTB)
- Backup PSW (BPSW)
- Backup PC (BPC)
- Fast interrupt vector register (FINTV)

If the MCU executes one of these instructions while in User Mode, an exception will trigger. If the user has a callback setup (see Section 2.4) then they will be alerted by a callback function of the exception.

2.10 ID Code

RX MCUs have a 16-byte ID Code in ROM that protects the MCU's memory from being read through a debugger, or in serial boot mode, in an attempt to extract the firmware from the device. The ID Code resides in the fixed vector table or option-setting memory and can easily be set in *r_bsp_config.h* (see Section 3.2.7). For more information on available ID Code options please reference the ID Code subsection in the 'Flash Memory' or 'ROM' section of your MCU's hardware manual.

2.11 Parallel Programmer Protection

Similar to the ID Code, RX MCUs also have a 4-byte code in ROM that can protect access to the MCU's memory from parallel programmers. The user has the option of allowing reads and write, only allowing writes, and prohibiting all access. See Section 3.2.7 for information on how to enable this feature.

2.12 Endian

RX MCUs have the option of operating in big or little endian mode. The r_bsp detects the endian selected in the toolchain and will use that to appropriately set the register. The r_bsp currently detects endian from the following toolchains:

- Renesas CCRX Toolchain
- IAR Toolchain for RX
- GCC for Renesas RX

2.13 Option Function Select Registers

RX MCUs have registers stored in ROM called Option Function Select registers. These registers are used to enable certain MCU features at reset instead of having to enable them in the user's code. Examples include the ability to enable low voltage monitoring, start the HOCO oscillating, and to configure and start the IWDT.
The user can input the values to be used for these registers in *r_bsp_config.h* (see Section 3.2.7).

2.14 Trusted Memory

The trusted memory (TM) function prevents illegal reading of the area set as TM. This function is disabled by default. To enable the trusted memory function, specify with the BSP_CFG_TRUSTED_MODE_FUNCTION definition in r_bsp_config.h.
For a dual-bank device, available TM area varies according to bank mode. To switch bank mode, specify with the BSP_CFG_CODE_FLASH_BANK_MODE definition in r_bsp_config.h.

2.15 Bank Mode

The user area can be used in linear mode, which uses the user area as one area, or in dual mode, which uses the user area as dual area. These modes can be selected with the bank mode switch function. The memory mapping differs between linear mode and dual mode, and is switched depending on the mode selected. When dual mode is selected, the bank area to launch the program can be selected.
To switch bank mode, specify with the BSP_CFG_CODE_FLASH_BANK_MODE definition in r_bsp_config.h.
To select a bank to launch the program, specify with the BSP_CFG_CODE_FLASH_START_BANK definition in r_bsp_config.h.

2.16 System Wide Parameter Checking

By default FIT modules will check input parameters to be valid. This is helpful during development but some users will want to disable this for production code. The reason for this would be to save execution time and code space. In *r_bsp_config.h* there is an option to globally enable or disable parameter checking. Local modules will use this value by default but can select to override the value locally if desired. To configure this option see Section 3.2.9.

2.17 Atomic Locking

The r_bsp provides API functions to implement atomic locking. These locks can be used to protect critical areas of code as a RTOS semaphore or mutex normally would. Care should be taken when using these locks though since they do not offer the advanced features one would expect from a modern RTOS. If used incorrectly then the locks could cause a deadlock in the user's system.

In each *mcu* folder the user will find a file named *mcu_locks.h*. This contains an enum named *mcu_lock_t* which has one lock per peripheral, and peripheral channel, on the MCU. These locks can be used to mark that a peripheral has been reserved. This could be used if the user wanted to use a FIT module to control three channels of a peripheral and their own custom code for one channel. By reserving the lock for the channel they need they have removed that channel from being used by the FIT Module. These locks can also be used if the user has more than one FIT module for the same peripheral. For example, if the user had one FIT module for using the SCI in asynchronous mode and another for using the SCI in I^2C mode then these locks will prevent these two modules from trying to use the same SCI channel. There are four locking API functions provided that are detailed in Section 3.2.8. The only difference between the hardware and software locking functions is that the hardware locking functions only use locks that are defined in *mcu_locks.h*. The software locking function takes locks allocated anywhere so the user could create their own as needed. FIT Modules that need locking and do not use a MCU peripheral will also create their own locks and use the software locking routines.

The user has the option of substituting the default r_bsp locking mechanisms for their own. See Section 3.2.8 for more information.

2.18 Register Protection

RX MCUs have protect registers that protect various MCU registers from being written. Examples of registers that are protected include clock registers, low power consumption registers, the software reset register, and low voltage detection registers. The r_bsp provides API functions for easily manipulating these registers to enable or disable write access. Refer to Sections 5.7 and 5.8 for more information.

2.19 CPU Functions

API functions are provided for CPU functions such as enabling and disabling interrupts and setting the CPU's interrupt priority level. Refer to Section 5 for more information.

2.20 Group Interrupts

Multiple peripheral interrupt requests (up to 32 requests) are grouped together as one interrupt request. Interrupts are grouped depending on the peripheral operating clock (PCLKB or PCLKA) and method to detect interrupt requests (edge detection or level detection).
When the group interrupt request is generated, checking the corresponding group interrupt request register (A or B, edge or level) identifies the interrupt source.

Figure 2.3 shows the Overview of FIT Group Interrupts.
With the BSP group interrupt function, when an interrupt occurs, the preregistered function is called. The registration is done by each peripheral FIT module using the R_BSP_InterruptWrite function.

1. Each peripheral FIT module registers the interrupt callback function by calling the R_BSP_InterruptWrite function.
2. When an interrupt occurs, the FIT module calls the callback function registered in 1 above.

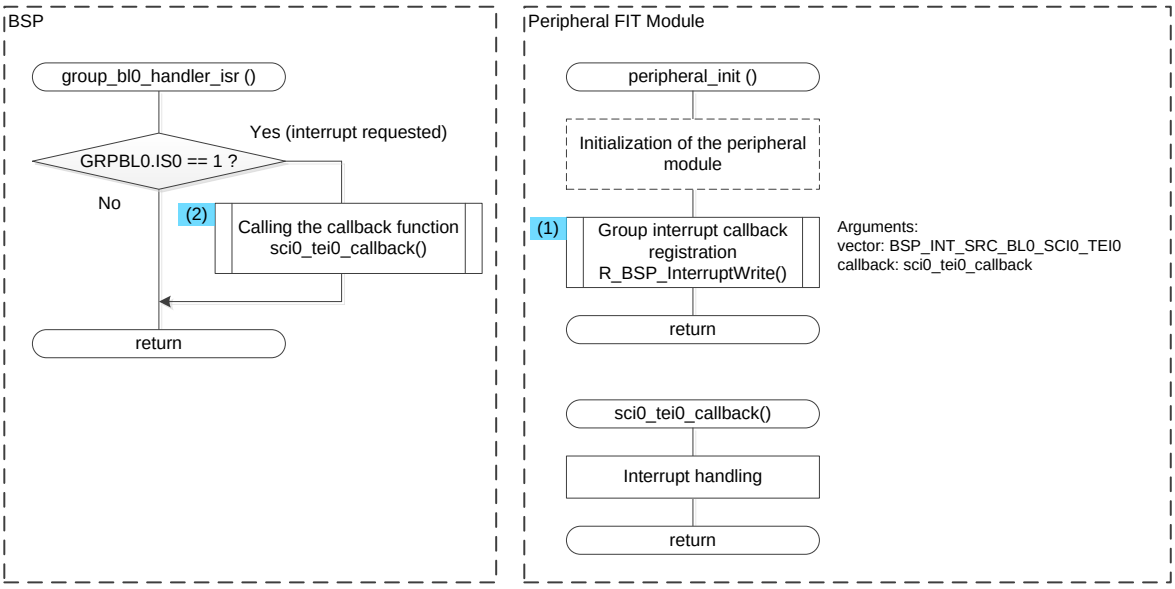


Figure 2.3 Overview of FIT Group Interrupts

2.21 Software Configurable Interrupts

Peripheral interrupt sources can be dynamically assigned to a vector number from 128 to 255. Based on the peripheral operating clock, they are divided into software configurable interrupt A and software configurable interrupt B. Software configurable interrupt B may be used for peripherals that operate in synchronization with PCLKB and can be assigned to interrupt numbers 128 to 207. Software configurable interrupt A may be used for peripherals that operate in synchronization with PCLKA and can be assigned to interrupt numbers 208 to 255.

2.22 Startup Disable

The startup disable function is the function for the user who wants to add the peripheral FIT module to the existing user project without creating a new project.
When the startup disable function is enabled, all startup processing performed by the BSP (processing in the startup function) become disabled. This prevents conflict with the user created startup processing.
This function is available only when using Renesas compiler.
Figure 2.4 shows the Overview of the Startup Disable Function, Figure 2.5 shows the Processing Disabled with the Startup Disable Function, and Figure 2.6 shows the Files Influenced by the Startup Disable Function.

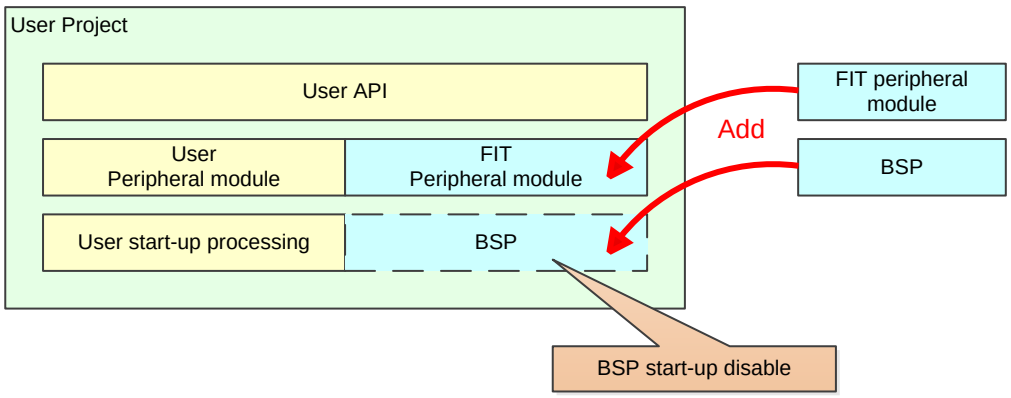


Figure 2.4 Overview of the Startup Disable Function

Note 1: The MCU skips this procedure.
Note 2: The operation varies depending on the setting in the r_bsp_config.h.

Figure 2.5 Processing Disabled with the Startup Disable Function

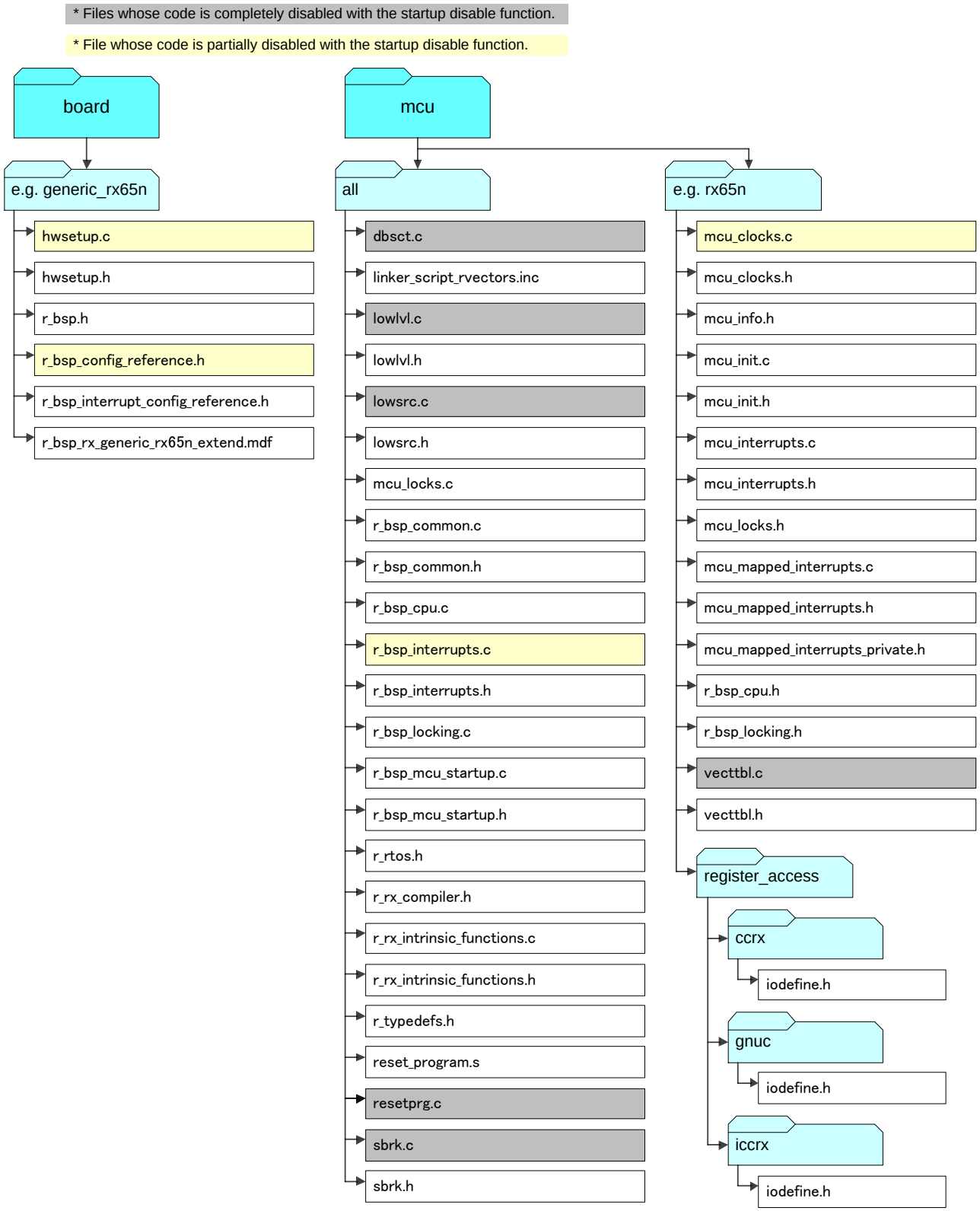


Figure 2.6 Files Influenced by the Startup Disable Function

2.22.1 Setting the Startup Disable Function

To disable the BSP startup processing, specify the setting described below. For how to implement the FIT module, refer to “9 Adding FIT Modules to the User Project”.
(1) Setting the configuration file

Disable the BSP startup processing by setting BSP_CFG_STARTUP_DISABLE to 1 in the r_bsp_config.h of the BSP.

Set the user created startup processing in the r_bsp_config.h. BSP API functions and peripheral FIT modules refer to the contents in the r_bsp_config.h. If the contents of the user startup processing and the BSP startup processing are different, the FIT module does not operate correctly.

Here is an example when mcu_info.h of the BSP has the definition of the peripheral module clock B frequency (BSP_PCLKB_HZ). The frequency of peripheral module clock B is calculated with the information (frequency of the resonator, division ratio, multiplication factor, and so on) set in r_bsp_config.h. The calculated frequency of peripheral module clock B is referred by peripheral FIT modules.

The BSP information to which FIT modules refer is generated from r_bsp_config.h. Therefore, the settings in the user startup processing and settings in r_bsp_config.h must be the same.

Figure 2.7 shows Configuration File Settings.

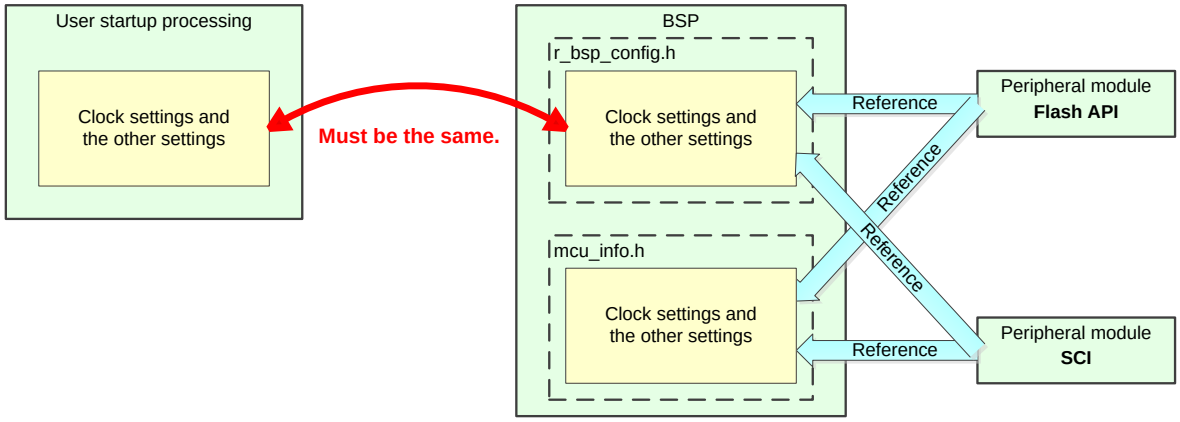


Figure 2.7 Configuration File Settings

(2) Setting for the conflicted group interrupt function

The BSP uses the group interrupt function. The function cannot be disabled since the peripheral FIT module uses it. To avoid confliction, use the group interrupt function of the BSP instead of the user's group interrupt function.

For group interrupts of the FIT module, refer to 2.20 Group Interrupts.

(3) Calling the R_BSP_StartupOpen function

The R_BSP_StartupOpen function performs initialization for the interrupt callback, register protection, and the hardware and pins. These processing are for using the BSP and peripheral FIT modules. Thus, the R_BSP_StartupOpen function must be called in the beginning of the user main function.

Refer to “5.18 R_BSP_StartupOpen()” for the R_BSP_StartupOpen function.

3. Configuration

The r_bsp provides two header files that are used for configuration. One header file is used for choosing which platform will be used. The other header file is used to configure the chosen platform.

3.1 Choosing a Platform

The r_bsp provides board support packages for many boards. Choosing which one is currently being used is done by modifying the *platform.h* header file found in the root of the r_bsp folder.

To choose a platform uncomment the #include for the board you are using. For example, to develop with a GENERIC_RX65N board, uncomment the #include for ‘./board/ generic_rx65n/r_bsp.h’ macro and make sure all other board #includes are commented out.

```
/***********************************************************************
DEFINE YOUR SYSTEM - UNCOMMENT THE INCLUDE PATH FOR THE PLATFORM YOU ARE USING.
***********************************************************************/
/* GENERIC_RX64M */
//#include "./board/generic_rx64m/r_bsp.h"

/* GENERIC_RX65N */
#include "./board/generic_rx65n/r_bsp.h"

/* GENERIC_RX66T */
//#include "./board/generic_rx66t/r_bsp.h"

/* GENERIC_RX71M */
//#include "./board/generic_rx71m/r_bsp.h"
```

3.2 Platform Configuration

Once a platform has been chosen, it will need to be configured. The user configures their platform using a file named *r_bsp_config.h.* Each platform has its own specific configuration file. This file is located in the platform’s *board* folder and is named *r_bsp_config_reference.h*. To create an *r_bsp_config.h* file the user simply needs to copy the *r_bsp_config_reference.h* file from their *board* folder, rename it to *r_bsp_config.h*, and put it somewhere in their project where it can be included. The reference configuration file is provided so that users always have a known-good configuration file if needed. It is recommended that the *r_bsp_config.h* file be stored in a folder named *r_config* in the user’s project. This is not a requirement but all FIT Modules have configuration files and having one designated location for these files makes them easy to find and easy to back up.

While each *r_bsp_config.h* file is different, there are many of the same options in each. The following sections will provide details on these configuration options. Note that each macro starts with the common prefix ‘BSP_CFG_’ which makes them easy to search for and easy for the user to identify.

When using Smart Configurator, the configuration option can be set on the software component configuration screen. The setting value is automatically reflected in *r_bsp_config.h* when adding modules to a user project.

3.2.1 MCU Product Part Number Information

The product part number for a MCU can provide the r_bsp with a lot of information about an MCU. For this reason, the beginning of the configuration file has definitions that are set based on the MCU's product part number. All of these macros have a prefix of 'BSP_CFG_MCU_PART_'. Some MCUs have more information in their product part numbers than others but the table below shows the standard set that most have.

Table 3.1 Product Part Number Defines

Define	Value	Meaning
BSP_CFG_MCU_PART_PACKAGE	See comments above #define in r_bsp_config.h.	Defines which package is being used. Depending on package sizes MCUs will have different numbers of pins and may have more or less peripherals.
BSP_CFG_MCU_PART_MEMORY_SIZE	See comments above #define in r_bsp_config.h.	Defines the sizes of ROM, RAM, and Data Flash.
BSP_CFG_MCU_PART_GROUP	See comments above #define in r_bsp_config.h.	Defines the MCU Group (e.g. RX64M, RX65N) in a MCU series.
BSP_CFG_MCU_PART_SERIES	See comments above #define in r_bsp_config.h.	Defines the MCU Series (e.g. RX600, RX200, RX100).

3.2.2 Stack & Heap Sizes

Table 3.2 Stack & Heap Defines (1/2)

Define	Value	Meaning
BSP_CFG_USER_STACK_ENABLE	0 = Use only Interrupt stack. 1 = Use Interrupt & User stacks.	Whether to use 1 stack (Interrupt stack) or 2 (Interrupt & User stack). For further explanation please see Section 2.8.
BSP_CFG_USTACK_BYTES	Size of User Stack in bytes.	Defines the size of the User stack. When using the IAR compiler, the stack size is determined by the GUI setting. Set the same value as the value set with the GUI.
BSP_CFG_ISTACK_BYTES	Size of Interrupt Stack in bytes.	Defines the size of the Interrupt stack. When using the IAR compiler, the stack size is determined by the GUI setting. Set the same value as the value set with the GUI.

Table 3.2 Stack & Heap Defines (2/2)

Define	Value	Meaning
BSP_CFG_HEAP_BYTES	Size of heap in bytes.	Defines the size of the heap. To prohibit the heap, refer to the comment above on '#define" in this definition. When using the IAR compiler, the heap size is determined by the GUI setting. Set the same value as the value set with the GUI.

3.2.3 STDIO & Debug Console

The use of the STDIO library requires extra code space, RAM space, and use of the heap. If the user does not require the use of STDIO then it is recommended to disable it and save the extra memory.

Table 3.3 STDIO & Debug Console Defines

Define	Value	Meaning
BSP_CFG_IO_LIB_ENABLE	0 = Disable use of STDIO 1 = Enable use of STDIO	Determines whether STDIO initialization functions are called at startup to set up the STDIO libraries.
BSP_CFG_USER_CHARGET_ENABLED	0 = User function is not called by charget function 1 = Specified user function is called by charget function.	Defines whether or not to redirect the charget function.
BSP_CFG_USER_CHARGET_FUNCTION	Function redirected to by charget function.	Defines the function to be called when the charget function is redirected.
BSP_CFG_USER_CHARPUT_ENABLED	0 = User function is not called by charget function 1 = Specified user function is called by charget function.	Defines whether or not to redirect the charget function.
BSP_CFG_USER_CHARPUT_FUNCTION	Function redirected to by charget function.	Defines the function to be called when the charget function is redirected.

3.2.4 CPU Modes & Boot Modes

RX MCUs have multiple boot modes including Serial Boot Mode, User Boot Mode, and Single-Chip Mode. The method for selecting boot mode varies depending on the MCU used. Some MCU select boot mode according to the level of a target pin at startup, and some others select boot mode by setting a pin and also setting a value (UB code) to the ROM.

Table 3.4 CPU Modes & Boot Modes Defines

Define	Value	Meaning
BSP_CFG_RUN_IN_USER_MODE	0 = Stay in Supervisor Mode 1 = Transition to User Mode	Out of reset RX MCUs operate in Supervisor Mode. The user has the option of transitioning to User Mode (which has limited write access to certain registers). Unless needed it is recommended to keep the MCU in Supervisor mode.
BSP_CFG_USER_BOOT_ENABLE	0 = Disable User Boot Mode 1 = Enable User Boot Mode	To enter User Boot Mode, a value in ROM must be set. If this macro defines User Boot Mode to be enabled then the r_bsp will set the appropriate ROM value.

3.2.5 RTOS

Table 3.5 RTOS Defines

Define	Value	Meaning
BSP_CFG_RTOS_USED	0 = RTOS is not used. 1 = FreeRTOS is used. 2 = embOS is used. 3 = MicroC_OS is used. 4 = RI600V4 or RI600PX is used.	Defines if a RTOS is being used in the current application. Some FIT modules may use this information for their own configuration.
BSP_CFG_RTOS_SYSTEM_TIMER	0 = CMT channel 0 1 = CMT channel 1 2 = CMT channel 2 3 = CMT channel 3	Defines the channel of CMT used for the RTOS system timer. If the RTOS is not used then this definition is disabled.

3.2.6 Clock Setup

Available clocks vary amongst RX MCUs but the same basic concepts apply to all. After reset the r_bsp will initialize the MCU clocks using the clock configuration macros found in *r_bsp_config.h.*

Table 3.6 Clock Setup Defines (1/3)

Define	Value	Meaning
BSP_CFG_CLOCK_SOURCE	0 = Low Speed On-Chip Oscillator (LOCO) 1 = High Speed On-Chip Oscillator (HOCO) 2 = Main Clock Oscillator 3 = Sub-Clock Oscillator 4 = PLL Circuit	Defines which clock source will be in use when the r_bsp code jumps to main().
BSP_CFG_MAIN_CLOCK_SOURCE	0 = Resonator 1 = External clock input	Defines which clock source will be used for the main clock oscillator.
BSP_CFG_RTC_ENABLE	0 = RTC is not used 1 = RTC is used	Defines whether to use the RTC or not.
BSP_CFG_SOSC_DRV_CAP	See the comment above #define in r_bsp_config.h.	Defines the driving ability of the sub-clock oscillator.
BSP_CFG_PLL_SCR	0 = Main clock 1 = HOCO	Defines which clock source will be used for the PLL Circuit.
BSP_CFG_USB_CLOCK_SOURCE	0 = System Clock (PLL Circuit/No division) 1 = USB PLL Circuit 2 = PLL Circuit (UDIVCLK) 3 = PPLL Circuit (PPLLDIVCLK)	Defines which clock source will be used when the USB peripheral is enabled.
BSP_CFG_LCD_CLOCK_SOURCE	0 = Low Speed On-Chip Oscillator (LOCO) 1 = High Speed On-Chip Oscillator (HOCO) 2 = Main Clock Oscillator 3 = Sub-Clock Oscillator 4 = IWDT dedicated clock (IWDTCLK)	Defines which clock source will be used when LCD is enabled.
BSP_CFG_LCD_CLOCK_ENABLE	0 = LCD Source clock is disabled 1 = LCD Source clock is enabled	Defines if clock source to the LCD is enabled.
BSP_CFG_HOCO_FREQUENCY	See the comment above #define in r_bsp_config.h.	Defines the HOCO frequency.
BSP_CFG_LPT_CLOCK_SOURCE	0 = Sub-Clock 1 = IWDT dedicated clock 2 = LPT not used	Defines which clock source will be used when the Low-Power Timer is enabled. The default value is 2 (LPT not used).

Table 3.6 Clock Setup Defines (2/3)

Define	Value	Meaning
BSP_CFG_XTAL_HZ	Input clock frequency in Hz.	Defines the input clock frequency (Resonator or External oscillator). This is used for calculating final clock speeds.
BSP_CFG_PLL_DIV	PLL Input Frequency Divider	Defines the PLL divider to be used. If the PLL is not used then this can be ignored.
BSP_CFG_PLL_MUL	PLL Frequency Multiplication Factor	Defines the PLL multiplier to be used. If the PLL is not used then this can be ignored.
BSP_CFG_UPLL_DIV	USB PLL Input Frequency Divider	Defines the USB PLL divider to be used. If the PLL is not used then this can be ignored.
BSP_CFG_UPLL_MUL	USB PLL Frequency Multiplication Factor	Defines the USB PLL multiplier to be used. If the PLL is not used then this can be ignored.
BSP_CFG_<ClockAcronym>_DIV **Examples:** **BSP_CFG_ICK_DIV** **BSP_CFG_PCKA_DIV** **BSP_CFG_PCKB_DIV** **BSP_CFG_PCKD_DIV** **BSP_CFG_FCK_DIV**	The divisor to use for this clock.	RX MCUs have a number of clock domains on-chip. Dividers can be set for each of these independently to maximize performance while minimizing power consumption. <ClockAcronym> is a placeholder for the name of the clock to be set. For example, to set the divider for the CPU clock (ICLK) then the user would set the BSP_CFG_ICK_DIV macro.
BSP_CFG_HOCO_WAIT_TIME	HOSCWTCR register setting value	Defines the high-speed on-chip oscillator wait time.
BSP_CFG_MOSC_WAIT_TIME	MOSCWTCR register setting value	Defines the main clock oscillator wait time.
BSP_CFG_SOSC_WAIT_TIME	SOSCWTCR register setting value	Defines the sub-clock oscillator wait time.
BSP_CFG_BCLK_OUTPUT	0 = BCLK is not output 1 = BCK frequency is output 2 = BCK/2 frequency is output	Defines if BCLK is output and if so what frequency is output.
BSP_CFG_SDCLK_OUTPUT	0 = SDCLK is not output 1 = BCK frequency is output	Defines if SDCLK is output.

Table 3.6 Clock Setup Defines (3/3)

Define	Value	Meaning
BSP_CFG_PPLL_DIV	PPLL Input Frequency Divider	Defines the PPLL divider to be used. If the PPLL is not used then this can be ignored.
BSP_CFG_PPLL_MUL	PPLL Frequency Multiplication Factor	Defines the PPLL multiplier to be used. If the PPLL is not used then this can be ignored.
BSP_CFG_PHY_CLOCK_SOURCE	0 = PLL Circuit 1 = PPLL Circuit 2 = Ethernet-PHY not use	Defines which clock source will be used for the External clock for Ethernet-PHY.
BSP_CFG_ESC_CLOCK_SOURCE	0 = PCLKA 1 = PPLLDIVCLK	Defines which clock source will be used for the ESC clock.
BSP_CFG_CLKOUT_SOURCE	0 = LOCO 1 = HOCO 2 = Main Clock Oscillator 3 = Sub Clock Oscillator 4 = PLL 6 = PPLL	Defines which clock source will be used when the CLKOUT pin output is enabled. If the clock is not output from the CLKOUT pin then this can be ignored.
BSP_CFG_CLKOUT_DIV	0 = x1/1 1 = x1/2 2 = x1/4 3 = x1/8 4 = x1/16	Defines the division ratio of the clock output from the CLKOUT pin. If the clock is not output from the CLKOUT pin then this can be ignored.
BSP_CFG_CLKOUT_OUTPUT	0 = CLKOUT pin output stopped. (Fixed to the low level) 1 = CLKOUT pin output enabled.	Defines whether to output the CLKOUT pin or not.
BSP_CFG_CLKOUT_RF_MAIN	0 = Resonator or External oscillator. 1 = CLKOUT_RF	Defines the clock to be input to the EXTAL terminal. Set 1 to input the output from Bluetooth-dedicated clock pin to the EXTAL pin.

3.2.7 Registers in ROM & External Memory Access Protection

Some registers are located in ROM and therefore must be set at compile-time. These include some option-setting memory registers as well as certain memory protection registers.

RX MCUs have two different mechanisms for protecting MCU memory from being read after production. The first is the use of ID codes. The RX ID code is a 16 byte value that can be used to protect the MCU from being connected to a debugger or from connecting in Serial Boot Mode. There are different settings that can be set for the ID code; please refer to the hardware manual for your device for available options. The second mechanism is a 4 byte value called ROM Code Protection. This value determines what read and write access parallel programmers have to the MCU.

Option-Setting Memory registers (i.e. OFS0, OFS1) can be set so that certain operations occur at reset. For example, the IWDT can be configured and enabled, voltage detection can be enabled, and HOCO oscillation can be enabled. When these registers are set the operations are completed before the MCU's reset vector is fetched and execution begins.

Table 3.7 ROM Register Defines (1/2)

Define	Value	Meaning
BSP_CFG_ID_CODE_LONG_1 **BSP_CFG_ID_CODE_LONG_2** **BSP_CFG_ID_CODE_LONG_3** **BSP_CFG_ID_CODE_LONG_4**	ID code setting in 4 byte units.	Defines the ID code of the MCU. The default value all 0xFF's means that protection is disabled. **Note:** If the ID code is set then it should be remembered because the code will be required if the MCU is going to be connected for debugging or in Serial Boot Mode again.
BSP_CFG_ROM_CODE_PROTECT_VALUE	0 = Read/Write access is disabled 1 = Read access is disabled Else = Read/Write access is enabled	Defines the read and write access allowed by parallel programmers.
BSP_CFG_OFS0_REG_VALUE	Value to be written to OFS0 register.	Defines the 4-byte value to be programmed into the OFS0 ROM location.
BSP_CFG_OFS1_REG_VALUE	Value to be written to OFS1 register.	Defines the 4-byte value to be programmed into the OFS1 ROM location. When HOCO oscillation is enabled, set the default value to BSP_CFG_HOCO_FREQUENCY.
BSP_CFG_TRUSTED_MODE_FUNCTION	Value to be written to TMEF register.	Defines if Trusted Mode is enabled or disabled.

Table 3.7 ROM Register Defines (2/2)

Define	Value	Meaning
BSP_CFG_FAW_REG_VALUE	Value to be written to FAW register.	Defines the 4-byte value to be programmed into the FAW ROM location.
BSP_CFG_ROMCODE_REG_VALUE	Value to be written to ROMCODE register.	Defines the 4-byte value to be programmed into the ROMCODE ROM location.
BSP_CFG_CODE_FLASH_BANK_MODE	0 = Dual mode 1 = Linear mode	Defines bank mode for a dual-bank device.
BSP_CFG_CODE_FLASH_START_BANK	0 = Launch from bank 0 1 = Launch from bank 1	Defines a bank to launch the program in dual mode for a dual-bank device. This definition is disabled for linear mode.

3.2.8 Atomic Locking

For an introduction into the r_bsp's atomic locking see 2.17. These macros allow the user to override the default locking mechanisms and implement their own. A user might wish to do this in order to replace the simple default mechanisms provided in the r_bsp with more feature rich objects such as semaphores or mutexes from their RTOS. If the user wished to do this they would first configure the r_bsp to use user defined locking mechanisms (see BSP_CFG_USER_LOCKING_ENABLED below). After that they would define BSP_CFG_USER_LOCKING_TYPE to be the type they wished to use for their locks. If using an RTOS semaphore then its type would be used here. Finally the user would need to define the four locking functions that would be used (see last 4 entries in table below). The arguments to these user defined functions have to match the arguments sent to the default locking functions. After these changes are made all locks in the user's project would be converted to the user defined locks. Whenever the r_bsp lock functions are called by user code, or FIT Module code, the user's functions would be called. At this point the user is responsible for implementing the locking features. Inside these functions the user would be free to use the more advanced locking features of their RTOS.

Table 3.8 Atomic Locking Defines (1/2)

Define	Value	Meaning
BSP_CFG_USER_LOCKING_ENABLED	0 = Use default locking mechanisms 1 = Use user defined locking mechanisms	The default locking mechanisms provided with the r_bsp do not use an RTOS and therefore do not offer some of the advanced features that a user might expect from an RTOS when using a semaphore or mutex.

Table 3.8 Atomic Locking Defines (2/2)

Define	Value	Meaning
BSP_CFG_USER_LOCKING_TYPE	Data type to be used for locks (default is *bsp_lock_t*)	If the user decides to use their own locking mechanism then the data type for their locks should be defined here. For example, if the user replaces the default locks with an RTOS semaphore or mutex then that data type would be specified here.
BSP_CFG_USER_LOCKING_HW_LOCK_FUNCTION	User defined functions to be called when r_bsp lock functions are overridden by user.	If the user is using their own locking mechanisms then the function defined by this macro will be called when R_BSP_HardwareLock() is called.
BSP_CFG_USER_LOCKING_HW_UNLOCK_FUNCTION	User defined functions to be called when r_bsp lock functions are overridden by user.	If the user is using their own locking mechanisms then the function defined by this macro will be called when R_BSP_HardwareUnlock() is called.
BSP_CFG_USER_LOCKING_SW_LOCK_FUNCTION	User defined functions to be called when r_bsp lock functions are overridden by user.	If the user is using their own locking mechanisms then the function defined by this macro will be called when R_BSP_SoftwareLock() is called.
BSP_CFG_USER_LOCKING_SW_UNLOCK_FUNCTION	User defined functions to be called when r_bsp lock functions are overridden by user.	If the user is using their own locking mechanisms then the function defined by this macro will be called when R_BSP_SoftwareUnlock() is called.

3.2.9 Parameter Checking

This macro is a global setting for enabling or disabling parameter checking. Each FIT module will also have its own local macro for this same purpose. By default the local macros will take the global value from here though they can be overridden. Therefore, the local setting has priority over this global setting. Disabling parameter checking should only be performed when inputs are known to be good and an increase in speed or decrease in code space is needed.

Table 3.9 Parameter Checking Defines

Define	Value	Meaning
BSP_CFG_PARAM_CHECKING_ENABLE	0 = Parameter checking disabled 1 = Parameter checking enabled	Defines whether the global setting for parameter checking is enabled or disabled. Local modules will take this value by default but can be locally overridden.

3.2.10 Extended Bus Master Priority Setting

Table 3.10 Extended Bus Master Priority Setting Defines

Define	Value	Meaning
BSP_CFG_EBMAPCR_1ST_PRIORITY	0 = GLCDC graphics 1 data read 1 = DRW2D texture data read 2 = DRW2D frame buffer data read write and display list data read 3 = GLCDC graphics 2 data read 4 = EDMAC Settings other than above are prohibited. Also, it is not possible to set the same value for multiple priorities.	Extended Bus Master 1st Priority Selection.
BSP_CFG_EBMAPCR_2ND_PRIORITY		Extended Bus Master 2nd Priority Selection.
BSP_CFG_EBMAPCR_3RD_PRIORITY		Extended Bus Master 3rd Priority Selection.
BSP_CFG_EBMAPCR_4TH_PRIORITY		Extended Bus Master 4th Priority Selection.
BSP_CFG_EBMAPCR_5TH_PRIORITY		Extended Bus Master 5th Priority Selection.

3.2.11 MCU Voltage

These macros set the voltage (Vcc) and the analog voltage (AVcc) supplied to the MCU by the system. Each FIT module obtains the voltage and analog voltage supplied to the MCU by referencing these macros. Specify values appropriate for the user system.

Table 3.11 MCU Voltage Defines

Define	Value	Meaning
BSP_CFG_MCU_VCC_MV	Voltage supplied to MCU (Vcc) in millivolts.	Some FIT Modules (e.g. LVD) need to know the voltage supplied to the MCU. The voltage information can be obtained from this definition.
BSP_CFG_MCU_AVCC_MV	Analog voltage supplied to MCU (AVcc) in millivolts.	Some FIT Modules (e.g. AD) need to know the analog voltage supplied to the MCU. The analog voltage information can be obtained from this definition.

3.2.12 Startup Disable

Table 3-12 Startup Disable Defines

Define	Value	Meaning
BSP_CFG_STARTUP_DISABLE	0 = BSP startup enable 1 = BSP startup disable	Defines the BSP startup processing to be enabled or disabled. When setting to be disabled, all startup processing performed by the BSP is disabled. This function is available only when using Renesas compiler.

3.2.13 Using Smart Configurator

Table 3.13 Using Smart Configurator Defines

Define	Value	Meaning
BSP_CFG_CONFIGURATOR_SELECT	0 = Do not use Smart Configurator 1 = Use Smart Configurator	Define whether Smart Configurator will be used in the current project. When BSP_CFG_CONFIGURATOR_SELECT = 1, The Smart Configurator initialization function is called.

3.2.14 Negative Voltage Input Settings for AD Pins

These macros specify whether negative voltage is input to pins AN000, AN001, AN002, AN100, AN101, AN102, PGAVSS0, and PGAVSS1 of the AD. Some FIT modules (such as the AD module) need to know whether to apply negative voltage to the input pins of the PGA when using PGA pseudo-differential input. Each FIT module obtains the negative voltage input settings of the AD pins by referencing these macros. Set values appropriate for the user system. These macros have no effect if the package of the product has no PGA differential inputs.

Table 3.14 Definitions of Negative Voltage Input Settings for AD Pins (1/2)

Definition	Value	Description
BSP_CFG_AD_NEGATIVE_VOLTAGE_INPUT_AN000	0 = Negative voltage is not to be input to the pin. 1 = Negative voltage is to be input to the pin.	Defines whether negative voltage is input to the AN000 pin of AD.
BSP_CFG_AD_NEGATIVE_VOLTAGE_INPUT_AN001	0 = Negative voltage is not to be input to the pin. 1 = Negative voltage is to be input to the pin.	Defines whether negative voltage is input to the AN001 pin of AD.
BSP_CFG_AD_NEGATIVE_VOLTAGE_INPUT_AN002	0 = Negative voltage is not to be input to the pin. 1 = Negative voltage is to be input to the pin.	Defines whether negative voltage is input to the AN002 pin of AD.
BSP_CFG_AD_NEGATIVE_VOLTAGE_INPUT_PGAVSS0	0 = Negative voltage is not to be input to the pin. 1 = Negative voltage is to be input to the pin.	Defines whether negative voltage is input to the PGAVSS0 pin of AD.

Table 3.14 Definitions of Negative Voltage Input Settings for AD Pins (2/2)

Definition	Value	Description
BSP_CFG_AD_NEGATIVE_VOLTAGE_INPUT_AN100	0 = Negative voltage is not to be input to the pin. 1 = Negative voltage is to be input to the pin.	Defines whether negative voltage is input to the AN100 pin of AD.
BSP_CFG_AD_NEGATIVE_VOLTAGE_INPUT_AN101	0 = Negative voltage is not to be input to the pin. 1 = Negative voltage is to be input to the pin.	Defines whether negative voltage is input to the AN101 pin of AD.
BSP_CFG_AD_NEGATIVE_VOLTAGE_INPUT_AN102	0 = Negative voltage is not to be input to the pin. 1 = Negative voltage is to be input to the pin.	Defines whether negative voltage is input to the AN102 pin of AD.
BSP_CFG_AD_NEGATIVE_VOLTAGE_INPUT_PGAVSS1	0 = Negative voltage is not to be input to the pin. 1 = Negative voltage is to be input to the pin.	Defines whether negative voltage is input to the PGAVSS1 pin of AD.

3.2.15 ROM Cache Function

Some RX MCUs have a ROM cache function. After a reset r_bsp initializes the ROM cache settings of the MCU using the ROM cache configuration macros found in r_bsp_config.h.

Table 3.15 ROM Cache Setting Definitions (1/2)

Definition	Value	Description
BSP_CFG_ROM_CACHE_ENABLE	0 = ROM cache operation disabled. 1 = ROM cache operation enabled.	Defines whether ROM cache operation is enabled or disabled.
BSP_CFG_NONCACHEABLE_AREA0_ENABLE	0 = Non-cacheable area 0 settings disabled. 1 = Non-cacheable area 0 settings enabled.	Defines whether non-cacheable area 0 is enabled or disabled when ROM cache operation is enabled.
BSP_CFG_NONCACHEABLE_AREA0_ADDR	Setting value of NCRG0 register	Defines the start address of non-cacheable area 0.
BSP_CFG_NONCACHEABLE_AREA0_SIZE	See comments above #define in r_bsp_config.h.	Defines the size of non-cacheable area 0.
BSP_CFG_NONCACHEABLE_AREA0 _IF_ENABLE	0 = Non-cacheable area 0 setting of IF cache disabled. 1 = Non-cacheable area 0 setting of IF cache enabled.	Defines whether the non-cacheable area for the IF cache is enabled or disabled when ROM cache operation is enabled and non-cacheable area 0 is enabled.
BSP_CFG_NONCACHEABLE_AREA0 _OA_ENABLE	0 = Non-cacheable area 0 setting of OA cache disabled. 1 = Non-cacheable area 0 setting of OA cache enabled.	Defines whether the non-cacheable area for the OA cache is enabled or disabled when ROM cache operation is enabled and non-cacheable area 0 is enabled.
BSP_CFG_NONCACHEABLE_AREA0 _DM_ENABLE	0 = Non-cacheable area 0 setting of DM cache disabled. 1 = Non-cacheable area 0 setting of DM cache enabled.	Defines whether the non-cacheable area for the DM cache is enabled or disabled when ROM cache operation is enabled and non-cacheable area 0 is enabled.

Table 3.15 ROM Cache Setting Definitions (2/2)

Definition	Value	Description
BSP_CFG_NONCACHEABLE_AREA1_ENABLE	0 = Non-cacheable area 1 settings disabled. 1 = Non-cacheable area 1 settings enabled.	Defines whether non-cacheable area 1 is enabled or disabled when ROM cache operation is enabled.
BSP_CFG_NONCACHEABLE_AREA1_ADDR	Setting value of NCRG1 register	Defines the start address of non-cacheable area 1.
BSP_CFG_NONCACHEABLE_AREA1_SIZE	See comments above #define in r_bsp_config.h.	Defines the size of non-cacheable area 1.
BSP_CFG_NONCACHEABLE_AREA1 _IF_ENABLE	0 = Non-cacheable area 1 setting of IF cache disabled. 1 = Non-cacheable area 1 setting of IF cache enabled.	Defines whether the non-cacheable area for the IF cache is enabled or disabled when ROM cache operation is enabled and non-cacheable area 1 is enabled.
BSP_CFG_NONCACHEABLE_AREA1 _OA_ENABLE	0 = Non-cacheable area 1 setting of OA cache disabled. 1 = Non-cacheable area 1 setting of OA cache enabled.	Defines whether the non-cacheable area for the OA cache is enabled or disabled when ROM cache operation is enabled and non-cacheable area 1 is enabled.
BSP_CFG_NONCACHEABLE_AREA1 _DM_ENABLE	0 = Non-cacheable area 1 setting of DM cache disabled. 1 = Non-cacheable area 1 setting of DM cache enabled.	Defines whether the non-cacheable area for the DM cache is enabled or disabled when ROM cache operation is enabled and non-cacheable area 1 is enabled.

3.2.16 Callback function at warm start

Table 3.16 Definition of callback function at warm start

Define	Value	Meaning
BSP_CFG_USER_WARM_START_CALLBACK _PRE_INITC_ENABLED	0 = User function is not called before C runtime environment has been initialized. 1 = User function is called before C runtime environment has been initialized.	Defines whether or not to call a user function before C runtime environment has been initialized.
BSP_CFG_USER_WARM_START _PRE_C_FUNCTION	Function to call before C runtime environment has been initialized.	Defines the function to be called when a user function is called before the C runtime environment has been initialized.
BSP_CFG_USER_WARM_START_CALLBACK _POST_INITC_ENABLED	0 = User function is not called after C runtime environment has been initialized. 1 = User function is called after C runtime environment has been initialized.	Defines whether or not to call a user function after C runtime environment has been initialized.
BSP_CFG_USER_WARM_START _POST_C_FUNCTION	Function to call after C runtime environment has been initialized.	Defines the function to be called when a user function is called after the C runtime environment has been initialized.

3.2.17 Board Revision

Table 3.17 Board Revision Defines

Define	Value	Meaning
BSP_CFG_BOARD_REVISION	See comments above #define in r_bsp_config.h.	There are multiple board revisions, and the specifications may differ from revision to revision. A specific board revision can be obtained based on this definition.

3.2.18 Interrupt Priority Level When FIT Module Interrupts Are Disabled

For some BSP functions, it is necessary to ensure that, while these functions are executing, interrupts from other FIT modules do not occur. By controlling the IPL, these functions disable interrupts that are at or below the specified interrupt priority level.

Table 3.18 Definition of Interrupt Priority Level When FIT Module Interrupts Are Disabled

Define	Value	Meaning
BSP_CFG_FIT_IPL_MAX	See comments above #define in r_bsp_config.h.	Defines interrupt priority level when FIT module interrupts are disabled.

4. API Information

This Driver API follows the Renesas API naming standards.

4.1 Hardware Requirements

Not Applicable.

4.2 Hardware Resource Requirements

Not Applicable.

4.3 Software Requirements

None.

4.4 Limitations

None.

4.5 Supported Toolchains

This driver is tested and working with the toolchains listed in 10.1 Confirmed Operation Environment.

4.6 Header Files

All API calls are accessed by including a single file *platform.h* which is supplied with this driver's project code.

4.7 Integer Types

This project uses ANSI C99 "Exact width integer types" in order to make the code clearer and more portable. These types are defined in *stdint.h*.

4.8 Configuration Overview

For configuration information please see Section 3.

4.9 API Data Structures

4.9.1 Software Lock

This data structure is used for implementing atomic locking on RX MCUs. The *lock* member must be 4-bytes in order to use the atomic XCHG instruction. This structure is the default type defined by the BSP_CFG_USER_LOCKING_TYPE macro.

```
typedef struct
{
    /* The actual lock. int32_t is used because this is what the xchg()
       instruction takes as parameters. */
    int32_t     lock;
} bsp_lock_t;
```

4.9.2 Interrupt Callback Parameter

This data structure is used when calling an interrupt callback function. The interrupt handler will fill in this structure, cast it as '(void *)', and then send it as the argument to the callback function.

```
typedef struct
{
    bsp_int_src_t vector;         //Which vector caused this interrupt
} bsp_int_cb_args_t;
```

4.9.3 Interrupt Control Parameter

This data structure is used when calling the R_BSP_InterruptControl function. Specify the parameter value according to the interrupt control command.

```
/* Type to be used for pdata argument in Control function. */
typedef union
{
      uint32_t ipl;     /* Used when enabling an interrupt to set that
                           interrupt's priority level */
} bsp_int_ctrl_t;
```

4.10 API Typedefs

4.10.1 Register Protection

This typedef defines the different register protection options that can be toggled. Notice that some registers are grouped together. For example, LPC, CGC, and software reset registers are all protected by the same bit. Which items, and how many, are in this typedef will vary depending on the MCU being used. Please reference r_bsp_c*pu.h* for your MCU to see the valid options for your MCU. The typedef below belongs to the RX111.

```
/* The different types of registers that can be protected. */
typedef enum
{
    /* Enables writing to the registers related to the clock generation circuit:
       SCKCR, SCKCR3, PLLCR, PLLCR2, MOSCCR, SOSCCR,LOCOCR, ILOCOCR, HOCOCR,
       OSTDCR, OSTDSR, CKOCR. */
    BSP_REG_PROTECT_CGC = 0,
    /* Enables writing to the registers related to operating modes, low power
       consumption, the clock generation circuit, and software reset: SYSCR1,
       SBYCR, MSTPCRA, MSTPCRB, MSTPCRC, OPCCR, RSTCKCR, SOPCCR, MOFCR,
MOSCWTCR,
       SWRR. */
    BSP_REG_PROTECT_LPC_CGC_SWR,
    /* Enables writing to the HOCOWTCR register. */
    BSP_REG_PROTECT_HOCOWTCR,
    /* Enables writing to the registers related to the LVD: LVCMPCR, LVDLVLR,
       LVD1CR0, LVD1CR1, LVD1SR, LVD2CR0, LVD2CR1, LVD2SR. */
    BSP_REG_PROTECT_LVD,
    /* Enables writing to MPC's PFS registers. */
    BSP_REG_PROTECT_MPC,
    /* This entry is used for getting the number of enum items. This must be the
       last entry. DO NOT REMOVE THIS ENTRY!*/
    BSP_REG_PROTECT_TOTAL_ITEMS
} bsp_reg_protect_t;
```

4.10.2 Hardware Resource Locks

This typedef defines the available hardware resource locks. For each entry in this enum one software lock will be allocated in the hardware lock array. Which items are in this list, and how many, will vary depending on the MCU chosen. The typedef below is for the RX111.

```
typedef enum
{
    BSP_LOCK_BSC = 0,
    BSP_LOCK_CAC,
    BSP_LOCK_CMT,
    BSP_LOCK_CMT0,
    BSP_LOCK_CMT1,
    BSP_LOCK_CRC,
    BSP_LOCK_DA,
    BSP_LOCK_DOC,
    BSP_LOCK_DTC,
    BSP_LOCK_ELC,
    BSP_LOCK_FLASH,
    BSP_LOCK_ICU,
    BSP_LOCK_IRQ0,
    BSP_LOCK_IRQ1,
    BSP_LOCK_IRQ2,
    BSP_LOCK_IRQ3,
    BSP_LOCK_IRQ4,
    BSP_LOCK_IRQ5,
    BSP_LOCK_IRQ6,
    BSP_LOCK_IRQ7,
    BSP_LOCK_IWDT,
    BSP_LOCK_MPC,
    BSP_LOCK_MTU,
    BSP_LOCK_MTU0,
    BSP_LOCK_MTU1,
    BSP_LOCK_MTU2,
    BSP_LOCK_MTU3,
    BSP_LOCK_MTU4,
    BSP_LOCK_MTU5,
    BSP_LOCK_POE,
    BSP_LOCK_RIIC0,
    BSP_LOCK_RSPI0,
    BSP_LOCK_RTC,
    BSP_LOCK_RTCB,
    BSP_LOCK_S12AD,
    BSP_LOCK_SCI1,
    BSP_LOCK_SCI5,
    BSP_LOCK_SCI12,
    BSP_LOCK_SYSTEM,
    BSP_LOCK_USB0,
    BSP_NUM_LOCKS   /* This entry is not a valid lock. It is used for sizing
                       g_bsp_Locks[] array below. Do not touch! */
} mcu_lock_t;
```

4.10.3 Interrupt Error Codes

This typedef defines the error codes that can be returned by the R_BSP_InterruptWrite(), R_BSP_InterruptRead(), and R_BSP_InterruptControl() functions.

The typedef below is for RX65N

The definition BSP_INT_ERR_GROUP_STILL_ENABLED is not included in MCUs which do not support group interrupts.

Some RX MCUs may support additional interrupt control commands.

```
typedef enum
{
    BSP_INT_SUCCESS = 0,
    BSP_INT_ERR_NO_REGISTERED_CALLBACK, //There is not a registered callback
                                        //for this interrupt source
    BSP_INT_ERR_INVALID_ARG,            //Illegal argument input
    BSP_INT_ERR_UNSUPPORTED             //Operation is not supported by this API
    BSP_INT_ERR_GROUP_STILL_ENABLED     //Not all group interrupts were disabled
                                        //so group interrupt was not disabled
} bsp_int_err_t;
```

4.10.4 Interrupt Control Commands

This typedef defines the available commands that can be used with the R_BSP_InterruptControl() function.

The typedef below is for RX65N.

The definitions BSP_INT_CMD_GROUP_INTERRUPT_ENABLE and BSP_INT_CMD_GROUP_INTERRUPT_DISABLE are not included in MCUs that do not support group interrupts.

Some RX MCUs may support additional interrupt control commands.

```
typedef enum
{
    BSP_INT_CMD_CALL_CALLBACK = 0, //Calls registered callback function
                                   //if one exists
    BSP_INT_CMD_INTERRUPT_ENABLE,  //Enables a give interrupt (Available for NMI
                                   //pin, FPU, and Bus Error)
    BSP_INT_CMD_INTERRUPT_DISABLE  //Disables a given interrupt (Available for
                                   //FPU, and Bus Error)
    BSP_INT_CMD_GROUP_INTERRUPT_ENABLE, //Enables a group interrupt when
                                   //a group interrupt source is given.
                                   //The pdata argument should give the IPL
                                   //to be used using the bsp_int_ctrl_t type.
                                   //If a group interrupt is enabled
                                   //multiple times with different IPL levels
                                   //it will use the highest given IPL.
   BSP_INT_CMD_GROUP_INTERRUPT_DISABLE //Disables a group interrupt when
                                   //a group interrupt source is given.
                                   //This will only disable a group
                                   //Interrupt when all interrupt sources
                                   //for that group are already disabled.
} bsp_int_cmd_t;
```

4.10.5 Interrupt Callback Function

This typedef defines the callback function type. Callback functions should have a ‘void’ return type and should take an argument of type ‘void *’.

```
typedef void (*bsp_int_cb_t)(void *);
```

4.10.6 Interrupt Sources

This typedef defines the interrupt vectors that can have callbacks registered to them. Note that the options in this typedef will vary depending on which MCU is being used. The typedef below is for the RX111. Other RX MCU's may support additional interrupt sources.

```
typedef enum
{
    BSP_INT_SRC_EXC_SUPERVISOR_INSTR = 0, //Occurs when privileged instruction
                                          //is executed in User Mode
    BSP_INT_SRC_EXC_UNDEFINED_INSTR,      //Occurs when MCU encounters an
                                          //unknown instruction
    BSP_INT_SRC_EXC_NMI_PIN,              //NMI Pin interrupt
    BSP_INT_SRC_EXC_FPU,                  //FPU exception
    BSP_INT_SRC_OSC_STOP_DETECT,          //Oscillation stop is detected
    BSP_INT_SRC_WDT_ERROR,                //WDT underflow/refresh error has
                                          //occurred
    BSP_INT_SRC_IWDT_ERROR,               //IWDT underflow/refresh error has
                                          //occurred
    BSP_INT_SRC_LVD1,                     //Voltage monitoring 1 interrupt
    BSP_INT_SRC_LVD2,                     //Voltage monitoring 2 interrupt
    BSP_INT_SRC_UNDEFINED_INTERRUPT,      //Interrupt has triggered for a vector
                                          //that user did not write a handler
                                          //for
    BSP_INT_SRC_BUS_ERROR,                //Bus error: illegal address access or
                                          //timeout
    BSP_INT_SRC_TOTAL_ITEMS               //DO NOT MODIFY! This is used for
                                          //sizing the interrupt callback array.
} bsp_int_src_t;
```

4.10.7 Unit for Software Delay

This typedef defines units which can be used with the R_BSP_SoftwareDelay function.

```
/* Available delay units. */
typedef enum
{
   BSP_DELAY_MICROSECS = 1000000, // Requested delay amount is in microseconds
   BSP_DELAY_MILLISECS = 1000,    // Requested delay amount is in milliseconds
   BSP_DELAY_SECS = 1             // Requested delay amount is in seconds
} bsp_delay_units_t;
```

4.11 Return Values

None.

4.12 Adding Driver to Your Project

Please see Section 7, Section 8, and Section 9.

4.13 Code size

The sizes of ROM, RAM and maximum stack usage associated with this module are listed below. Information is listed for a single representative device of the RX100 Series, RX200 Series, and RX600 Series, respectively.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 3 Configuration.

The values in the table below are confirmed under the following conditions.

Module Revision: r_bsp rev5.00

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00

(The option of "lang = c99" is added to the default settings of the integrated development environment.)

GCC for Renesas RX 4.8.4.201902

(The option of "-std=gnu99" is added to the default settings of the integrated development environment.)

IAR C/C++ Compiler for Renesas RX version 4.11.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings

ROM, RAM and Stack Code Sizes

Device	Category	Memory Used					
		CCRX		GCC		IAR	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX130	ROM	5,345 bytes	5,345 bytes	7,540 bytes	7,540 bytes	4,694 bytes	4,694 bytes
	RAM *1	3,030 bytes		2,836 bytes		1,556 bytes	
	STACK *2	196 bytes		-		132 bytes	
RX231	ROM	5,600 bytes	5,600 bytes	8,060 bytes	8,060 bytes	5,062 bytes	5,062 bytes
	RAM *1	6,970 bytes		6,776 bytes		1,656 bytes	
	STACK *2	200 bytes		-		132 bytes	
RX65N	ROM	8,084 bytes	8,071 bytes	13,364 bytes	13,340 bytes	9,583 bytes	9,568 bytes
	RAM *1	7,440 bytes		7,272 bytes		2,124 bytes	
	STACK *2	212 bytes		-		148 bytes	

Note 1. It is because the RAM sizes different for each compiler that the default values of stack and heap sizes different.

Note 2. The sizes of maximum usage stack of Interrupts functions is included.

4.14 “for”, “while” and “do while” statements

In this module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for the register to be reflected and so on. For this loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if the user incorporates fail-safe processing into loop processing, user can search the corresponding processing using “WAIT_LOOP”.

The following shows an example of description.

```
while statement example :
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}

for statement example :
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}

do while statement example :
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /* WAIT_LOOP */
```

5. API Functions

5.1 Summary

The following functions are included in this design:

Function	Description
R_BSP_GetVersion	Returns the version of r_bsp.
R_BSP_InterruptsDisable	Globally disables interrupts.
R_BSP_InterruptsEnable	Globally enables interrupts.
R_BSP_CpuInterruptLevelRead	Reads the CPU's Interrupt Priority Level.
R_BSP_CpuInterruptLevelWrite	Writes the CPU's Interrupt Priority Level.
R_BSP_RegisterProtectEnable	Enables write protection for selected registers.
R_BSP_RegisterProtectDisable	Disables write protection for selected registers.
R_BSP_SoftwareLock	Attempts to reserve a lock.
R_BSP_SoftwareUnlock	Releases a lock.
R_BSP_HardwareLock	Attempts to reserve a hardware peripheral lock.
R_BSP_HardwareUnlock	Releases a hardware peripheral lock.
R_BSP_InterruptWrite	Registers a callback function for an interrupt.
R_BSP_InterruptRead	Gets the callback for an interrupt if one is registered.
R_BSP_InterruptControl	Controls various interrupt operations.
R_BSP_SoftwareDelay	Delays the specified duration.
R_BSP_GetICLkFreqHz	Returns the system clock frequency specified by the r_bsp.
R_BSP_StartupOpen *1	Performs the startup processing for using the BSP.
R_BSP_VoltageLevelSetting *2	Makes settings to the voltage level setting register (VOLSR) that are necessary in order to use the USB, AD, and RIIC peripheral modules.
R_BSP_InterruptRequestEnable	Enables the specified interrupt request.
R_BSP_InterruptRequestDisable	Disables the specified interrupt request.
R_BSP_ConfigClockSetting *3	Used by Bluetooth® Low Energy Protocol Stack Basic Package (R01UW0205).

Note 1. This function is only used when the BSP startup processing is disabled.

Note 2. This function is available only on the RX66T and RX72T.

Note 3. This function is available only on the RX23W.

5.2 R_BSP_GetVersion()

Returns the current version of the r_bsp.

Format

```
uint32_t R_BSP_GetVersion(void);
```

Parameters

None.

Return Values

Version of the r_bsp.

Properties

Prototyped in file “r_bsp_common.h”.
Implemented in file “r_bsp_common.c”.

Description

This function will return the version of the currently installed r_bsp. The version number is encoded where the top 2 bytes are the major version number and the bottom 2 bytes are the minor version number. For example, Version 4.25 would be returned as 0x00040019.

Reentrant

Yes.

Example

```
uint32_t cur_version;

/* Get version of installed r_bsp. */
cur_version = R_BSP_GetVersion();

/* Check to make sure version is new enough for this application’s use. */
if (MIN_VERSION > cur_version)
{
    /* This r_bsp version is not new enough and does not have XXX feature
       that is needed by this application. Alert user. */
    ....
}
```

Special Notes:

None.

5.3 R_BSP_InterruptsDisable()

Globally disables interrupts.

Format

```
   void R_BSP_InterruptsDisable(void);
```

Parameters

None.

Return Values

None.

Properties

Prototyped in file “r_bsp_cpu.h”.
Implemented in file “r_bsp_cpu.c”

Description

This function globally disables interrupts. This is performed by clearing the ‘I’ bit in the CPU’s Processor Status Word (PSW) register.

Reentrant

Yes.

Example

```
/* Disable interrupts so that accessing this critical area will be guaranteed
   to be atomic. */
R_BSP_InterruptsDisable();

/* Access critical resource while interrupts are disabled */
....

/* End of critical area. Enable interrupts. */
R_BSP_InterruptsEnable();
```

Special Notes:

The ‘I’ bit of the PSW can only be modified when in Supervisor Mode. If the CPU is in User Mode and this function is called then a Privileged Instruction Exception will occur.

5.4 R_BSP_InterruptsEnable()

Globally enables interrupts.

Format

```
   void R_BSP_InterruptsEnable(void);
```

Parameters

None.

Return Values

None.

Properties

Prototyped in file “r_bsp_cpu.h”.
Implemented in file “r_bsp_cpu.c”.

Description

This function globally enables interrupts. This is performed by setting the ‘I’ bit in the CPU’s Processor Status Word (PSW) register.

Reentrant

Yes.

Example

```
/* Disable interrupts so that accessing this critical area will be guaranteed
   to be atomic. */
R_BSP_InterruptsDisable();

/* Access critical resource while interrupts are disabled */
....

/* End of critical area. Enable interrupts. */
R_BSP_InterruptsEnable();
```

Special Notes:

The ‘I’ bit of the PSW can only be modified when in Supervisor Mode. If the CPU is in User Mode and this function is called then a Privileged Instruction Exception will occur.

5.5 R_BSP_CpuInterruptLevelRead()

Reads the CPU’s Interrupt Priority Level.

Format

```
uint32_t R_BSP_CpuInterruptLevelRead(void);
```

Parameters

None.

Return Values

The CPU’s Interrupt Priority Level.

Properties

Prototyped in file “r_bsp_cpu.h”.
Implemented in file “r_bsp_cpu.c”.

Description

This function reads the CPU’s Interrupt Priority Level. This level is stored in the IPL bits of the Processor Status Word (PSW) register.

Reentrant

Yes.

Example

```
uint32_t cpu_ipl;

/* Read the CPU’s Interrupt Priority Level. */
cpu_ipl = R_BSP_CpuInterruptLevelRead();
```

Special Notes:

None.

5.6 R_BSP_CpuInterruptLevelWrite()

Writes the CPU's Interrupt Priority Level.

Format

```
bool R_BSP_CpuInterruptLevelWrite(uint32_t level);
```

Parameters

level

The level to write to the CPU's IPL.

Return Values

true: *Successful, CPU's IPL has been written*

false: *Failure, provided 'level' has invalid IPL value*

Properties

Prototyped in file "r_bsp_cpu.h".
Implemented in file "r_bsp_cpu.c".

Description

This function writes the CPU's Interrupt Priority Level. This level is stored in the IPL bits of the Processor Status Word (PSW) register. This function does check to make sure that the IPL being written is valid. The maximum and minimum valid settings for the CPU IPL are defined in *mcu_info.h* using the BSP_MCU_IPL_MAX and BSP_MCU_IPL_MIN macros.

Reentrant

Yes.

Example

```
/* Response time is critical during this portion of the application. Set the
   CPU's Interrupt Priority Level so that interrupts below the set
   threshold are disabled. Interrupt vectors with IPLs higher than this
   threshold will still be accepted and will not have to contend with the
   lower priority interrupts. */
if (false == R_BSP_CpuInterruptLevelWrite(HIGH_PRIORITY_THRESHOLD))
{
    /* Error in setting CPU's IPL. Invalid IPL was provided. */
    ....
}

/* Only high priority interrupts (as defined by user) will be accepted during
   this period. */
....

/* Time sensitive period is over. Set CPU's IPL back to lower value so that
   lower priority interrupts can now be serviced again. */
if (false == R_BSP_CpuInterruptLevelWrite(LOW_PRIORITY_THRESHOLD))
{
    /* Error in setting CPU's IPL. Invalid IPL was provided. */
    ....
}
```

Special Notes:

The CPU's IPL can only be modified by the user when in Supervisor Mode. If the CPU is in User Mode and this function is called then a Privileged Instruction Exception will occur.

5.7 R_BSP_RegisterProtectEnable()

Enables write protection for selected registers.

Format

```
void R_BSP_RegisterProtectEnable(bsp_reg_protect_t regs_to_protect);
```

Parameters

regs_to_protect

Which registers to enable write protection for. See Section 4.10.1.

Return Values

None.

Properties

Prototyped in file “r_bsp_cpu.h”.
Implemented in file “r_bsp_cpu.c”.

Description

This function enables write protection for the input registers. Only certain MCU registers have the ability to be write protected. To see which registers are available to be protected by this function look at the *bsp_reg_protect_t* enum in r_bsp_c*pu.h* for your MCU.

This function, and R_BSP_RegisterProtectDisable(), use counters for each entry in the *bsp_reg_protect_t* enum so that users can call these functions multiple times without problem. By controlling the IPL (interrupt priority level), interrupts that are at or below the specified interrupt priority level will be disabled while the function is executing. An example of why this is needed is shown below in the Special Notes section below.

Reentrant

No.

Example

```
/* Write access must be enabled before writing to MPC registers. */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);

/* MPC registers are now writable. */
/* Setup Port 2 Pin 6 as TXD1 for SCI1. */
MPC.P26PFS.BYTE = 0x0A;

/* Setup Port 4 Pin 2 as AD input for potentiometer. */
MPC.P42PFS.BYTE = 0x80;

/* More pin setup. */
....

/* Enable write protection for MPC registers to protect against accidental
   writes. */
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_MPC);
```

Special Notes:

This is an example showing why counters are needed for register protection.

1. The user's application calls the open function for r_module1.
2. r_module1 disables write protection for some registers that are required to be written during initialization of this module by calling R_BSP_RegisterProtectDisable(). At this point the counter for this protected registers is incremented by 1.
3. r_module1 writes to unprotected registers that were made writable in the previous step.
4. r_module1 also depends upon r_module2 and needs to call its open function, R_MODULE2_Open().
5. In the r_module2 function it also needs to write to the same protected registers as r_module1. r_module2 calls R_BSP_RegisterProtectDisable() again since it does not know that r_module1 already enabled write access to these registers. The counter for the protected register is incremented by 1 and is now 2.
6. r_module2 writes to unprotected registers that were made writable in the previous step.
7. r_module2 is done writing to the protected registers so it calls R_BSP_RegisterProtectEnable() to re-enable write protection for the registers. The counter for the protected register is decremented by 1 and is now 1. Since the counter is not 0 the code knows that it should not actually re-enable protection yet.
8. Execution goes back to R_MODULE1_Open() where it continues to write to registers. Here is where a problem can occur. If counters are not used then the call to R_BSP_RegisterProtectEnable() by r_module2 (Step #7) can prevent the registers in r_module1 from being written.
9. r_module1 is done writing to the protected registers so it calls R_BSP_RegisterProtectEnable() to re-enable write protection for the registers. The counter for the protected register is decremented by 1 and is now 0. Since the counter is 0 the API code knows that it is safe to re-enable write protection for the registers.

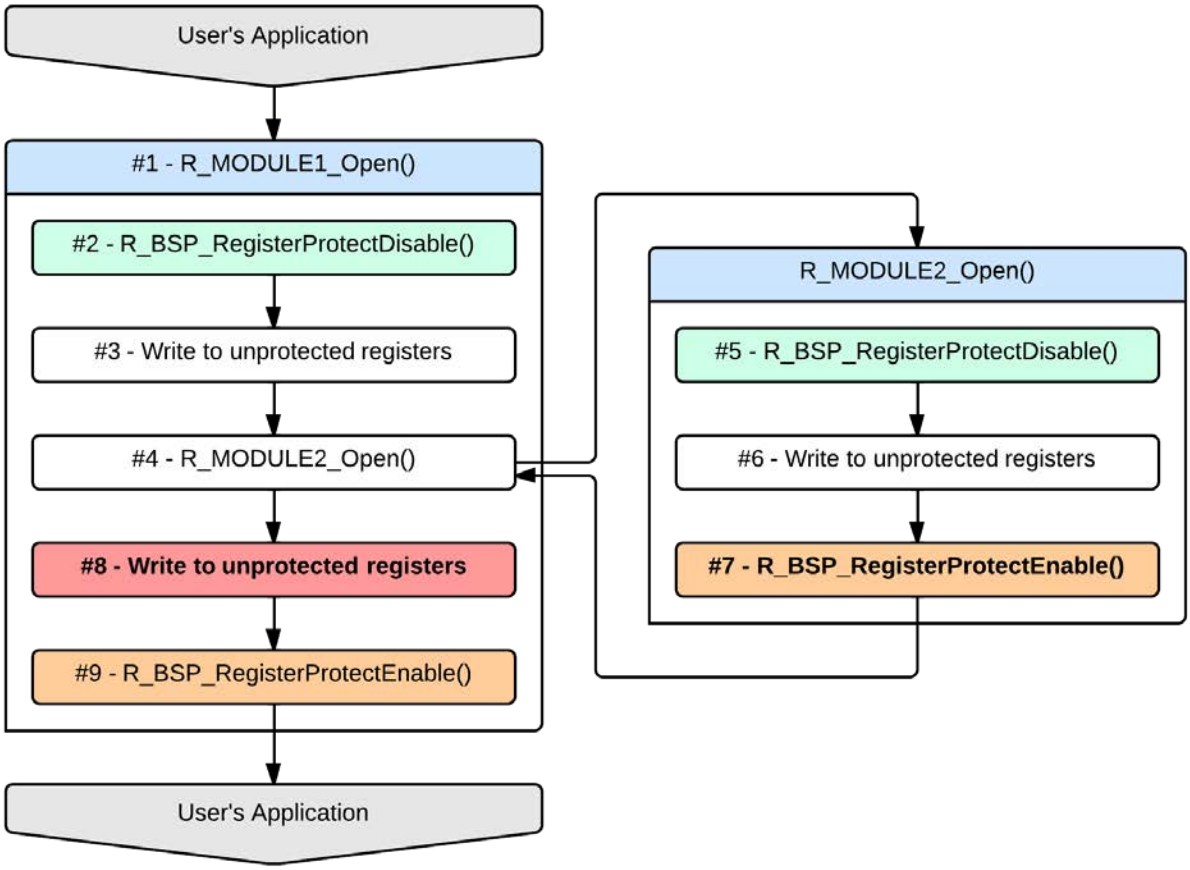


Figure 5.1 Register Protection Example

5.8 R_BSP_RegisterProtectDisable()

Disables write protection for selected registers.

Format

```
void R_BSP_RegisterProtectDisable(bsp_reg_protect_t regs_to_unprotect);
```

Parameters

regs_to_unprotect

Which registers to disable write protection for. See Section 4.10.1.

Return Values

None.

Properties

Prototyped in file “r_bsp_cpu.h”.
Implemented in file “r_bsp_cpu.c”

Description

This function disables write protection for the input registers. Only certain MCU registers have the ability to be write protected. To see which registers are available to be protected by this function look at the *bsp_reg_protect_t* enum in r_bsp_c*pu.h* for your MCU.

This function, and R_BSP_RegisterProtectEnable(), use counters for each entry in the *bsp_reg_protect_t* enum so that users can call these functions multiple times without problem. By controlling the IPL (interrupt priority level), interrupts that are at or below the specified interrupt priority level will be disabled while the function is executing. An example of why this is needed is shown in the Special Notes section of Section 5.7.

Reentrant

No.

Example

```
/* Write access must be enabled before writing to CGC registers. */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_CGC);
/* CGC registers are spread amongst two protection bits. */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_LPC_CGC_SWR);

/* CGC registers are now writable. */
/* Select PLL as clock source. */
SYSTEM.SCKCR3.WORD = 0x0400;

/* More clock setup. */
....

/* Enable write protection for CGC registers to protect against accidental
   writes. */
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_CGC);
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_LPC_CGC_SWR);
```

Special Notes:

None.

5.9 R_BSP_SoftwareLock()

Attempts to reserve a lock.

Format

```
bool R_BSP_SoftwareLock(BSP_CFG_USER_LOCKING_TYPE * const plock);
```

Parameters

plock

Pointer to lock structure with lock to try and acquire.

Return Values

true: *Successful, lock was available and acquired*

false: *Failure, lock was already acquired and is not available*

Properties

Prototyped in file “r_bsp_locking.h”.
Implemented in file “r_bsp_locking.c”

Description

This function implements an atomic locking mechanism. Locks can be used in numerous ways. Two common uses of locks are to protect critical sections of code and to protect against duplicate resource allocation. For protecting critical sections of code the user would require that the code first obtain the critical section’s lock before executing. An example of protecting against duplicate resource allocation would be if the user had two FIT modules that used the same peripheral. For example, the user may have one FIT module that uses the SCI peripheral in UART mode and another FIT module that uses the SCI peripheral in I^2C mode. To make sure that both modules cannot use the same SCI channel, locks can be used.

Care should be taken when using locks as they do not provide advanced features one might expect from an RTOS semaphore or mutex. If used improperly locks can lead to deadlock in the user’s system.

Users can override the default locking mechanisms. See Section 3.2.8 for more information.

Reentrant

Yes.

Example

This shows an example of using locks with the Virtual EEPROM code. This FIT module does not access any peripherals directly, but still needs protection against reentrancy.

```
/* Used for locking state of VEE */
static BSP_CFG_USER_LOCKING_TYPE g_vee_lock;

/******************************************************************************
* Function Name: vee_lock_state
* Description  : Tries to lock the VEE state
* Arguments    : state -
*                    Which state to try to transfer to
* Return value : VEE_SUCCESS -
*                    Successful, state taken
*                VEE_BUSY -
*                    Data flash is busy, state not taken
******************************************************************************/
static uint8_t vee_lock_state (vee_states_t state)
{
    /* Local return variable */
    uint8_t ret = VEE_SUCCESS;

    /* Try to lock VEE to change state. */
    /* Check to see if lock was successfully taken. */
    if(false == R_BSP_SoftwareLock(&g_vee_lock))
    {
```

```
        /* Another operation is on-going */
        return VEE_BUSY;
    }


    /* Check VEE status to make sure we are not interfering with another
       thread */
    if( state == VEE_READING )
    {
        /* If another read comes in while the state is reading then we are OK */
        if( ( g_vee_state != VEE_READY ) && ( g_vee_state != VEE_READING) )
        {
            /* VEE is busy */
            ret = VEE_BUSY;
        }
    }
    else
    {
        /* If we are doing something other than reading then we must be in the
           VEE_READY state */
        if( g_vee_state != VEE_READY )
        {
            /* VEE is busy */
            ret = VEE_BUSY;
        }
    }

    if( ret == VEE_SUCCESS )
    {
        /* Lock state */
        g_vee_state = state;
    }

    /* Release lock. */
    R_BSP_SoftwareUnlock(&g_vee_lock);

    return ret;
}
```

Special Notes:

None.

5.10 R_BSP_SoftwareUnlock()

Releases a lock.

Format

```
bool R_BSP_SoftwareUnlock(BSP_CFG_USER_LOCKING_TYPE * const plock);
```

Parameters

plock

Pointer to lock structure with lock to release.

Return Values

true: *Successful, lock was released. Or the lock has been already released.*

false: *Failure, lock could not be released*

Properties

Prototyped in file “r_bsp_locking.h”.
Implemented in file “r_bsp_locking.c”

Description

This function releases a lock that was previously acquired using the R_BSP_SoftwareLock() function. Please see Section 5.9 for more information on locks.

Reentrant

Yes.

Example

This shows an example of using locks for a critical section of code.

```
/* Used for locking critical section of code. */
static BSP_CFG_USER_LOCKING_TYPE g_critical_lock;

static bool critical_area_example (void)
{
    /* Try to acquire lock for executing critical section below. */
    if(false == R_BSP_SoftwareLock(&g_critical_lock))
    {
        /* Lock has already been acquired. */
        return false;
    }

    /* BEGIN CRITICAL SECTION. */

    /* Execute critical section. */
    ....

    /* END CRITICAL SECTION. */

    /* Release lock. */
    R_BSP_SoftwareUnlock(&g_critical_lock);

    return true;
}
```

Special Notes:

None.

5.11 R_BSP_HardwareLock()

Attempts to reserve a hardware peripheral lock.

Format

```
bool R_BSP_HardwareLock(mcu_lock_t const hw_index);
```

Parameters

hw_index

Index of lock to acquire from the hardware lock array.

Return Values

true: *Successful, lock was available and acquired*

false: *Failure, lock was already acquired and is not available*

Properties

Prototyped in file “r_bsp_locking.h”.
Implemented in file “r_bsp_locking.c”

Description

This function attempts to acquire the lock for a hardware resource of the MCU. Instead of sending in a pointer to a lock as with the R_BSP_SoftwareLock() function, the user sends in an index to an array that holds 1 lock per MCU hardware resource. This array is shared amongst all FIT modules and user code therefore allowing multiple FIT modules (and user code) to use the same locks. The user can see the available hardware resources by looking at the *mcu_lock_t* enum in *mcu_locks.h*. These enum values are also the index into the hardware lock array. The same atomic locking mechanisms from the R_BSP_SoftwareLock() function are used with this function as well.

Reentrant

Yes.

Example

This example shows hardware locks being used to control access to a RSPI channel.

```
/******************************************************************************
* Function Name: R_RSPI_Send
* Description  : Send data over RSPI channel.
* Arguments    : channel -
*                    Which channel to use.
*                pdata -
*                    Pointer to data to transmit
*                bytes -
*                    Number of bytes to transmit
* Return Value : true -
*                    Data sent successfully.
*                false -
*                    Could not obtain lock.
******************************************************************************/
bool R_RSPI_Send(uint8_t channel, uint8_t * pdata, uint32_t bytes)
{
    mcu_lock_t rspi_channel_lock;

    /* Check and make sure channel is valid. */
    ...

    /* Use appropriate RSPI channel lock. */
    if (0 == channel)
    {
        rspi_channel_lock = BSP_LOCK_RSPI0;
    }
    else
    {
```

```
        rspi_channel_lock = BSP_LOCK_RSPI1;
    }

    /* Attempt to obtain lock so we know we have exclusive access to RSPI
       channel. */
    if (false == R_BSP_HardwareLock(rspi_channel_lock))
    {
        /* Lock has already been acquired by another task. Need to try again
           later. */
        return false;
    }

    /* Else, lock was acquired. Continue on with send operation. */
    ...

    /* Now that send operation is completed, release hold on lock so that other
       tasks may use this RSPI channel. */
    R_BSP_HardwareUnlock(rspi_channel_lock);

    return true;
}
```

Special Notes:

Each entry in the *mcu_lock_t* enum in *mcu_locks.h* will be allocated a lock. On RX MCUs, each lock is required to be 4-bytes. If RAM space is an issue then the user can remove the entries from the *mcu_lock_t* enum they are not using. For example, if the user is not using the CRC peripheral then they could delete the BSP_LOCK_CRC entry. The user will save 4-bytes per deleted entry.

5.12 R_BSP_HardwareUnlock()

Releases a hardware peripheral lock.

Format

```
bool R_BSP_HardwareUnlock(mcu_lock_t const hw_index);
```

Parameters

hw_index

Index of lock to release from the hardware lock array.

Return Values

true: *Successful, lock was released*

false: *Failure, lock could not be released*

Properties

Prototyped in file “r_bsp_locking.h”.
Implemented in file “r_bsp_locking.c”

Description

This function attempts to release the lock for a hardware resource of the MCU that was previously acquired using the R_BSP_HardwareLock() function. For more information on hardware locks please see Section 5.11.

Reentrant

Yes.

Example

This example shows hardware locks being used to prevent duplicate hardware resource allocation. The R_SCI_Open() function takes the lock so all modules know that the SCI channel is being used. R_SCI_Close() releases the lock thereby making it available for any module to use.

```
bool R_SCI_Open(uint8_t channel, ...)
{
    mcu_lock_t sci_channel_lock;

    /* Check and make sure channel is valid. */
    ...

    /* Use appropriate RSPI channel lock. */
    if (0 == channel)
    {
        sci_channel_lock = BSP_LOCK_SCI0;
    }
    else if (1 == channel)
    {
        sci_channel_lock = BSP_LOCK_SCI1;
    }
    ... continue for other channels ...

    /* Attempt to obtain lock so we know we have exclusive access to SCI
       channel. */
    if (false == R_BSP_HardwareLock(sci_channel_lock))
    {
        /* Lock has already been acquired by another task or another FIT module.
           Need to try again later. */
        return false;
    }

    /* Else, lock was acquired. Continue on initialization. */
    ...
```

```
}


bool R_SCI_Close(uint8_t channel, ...)
{
    mcu_lock_t sci_channel_lock;

    /* Check and make sure channel is valid. */
    ...

    /* Use appropriate RSPI channel lock. */
    if (0 == channel)
    {
        sci_channel_lock = BSP_LOCK_SCI0;
    }
    else if (1 == channel)
    {
        sci_channel_lock = BSP_LOCK_SCI1;
    }
    ... continue for other channels ...

    /* Clean up and turn off this SCI channel. */
    ....

    /* Release hardware lock for this channel. */
    R_BSP_HardwareUnlock(sci_channel_lock);
}
```

Special Notes:

Each entry in the *mcu_lock_t* enum in *mcu_locks.h* will be allocated a lock. On RX MCUs, each lock is required to be 4-bytes. If RAM space is an issue then the user can remove the entries from the *mcu_lock_t* enum that they are not using. For example, if the user is not using the CRC peripheral then they could delete the BSP_LOCK_CRC entry. The user will save 4-bytes per deleted entry.

5.13 R_BSP_InterruptWrite()

Registers a callback function for an interrupt.

Format

```
bsp_int_err_t R_BSP_InterruptWrite(bsp_int_src_t vector,
                                   bsp_int_cb_t callback);
```

Parameters

vector

Which interrupt to register a callback for. See Section 4.10.6.

callback

Pointer to function to call when interrupt occurs. See Section 4.10.5.

Return Values

BSP_INT_SUCCESS: *Successful, callback has been registered*

BSP_INT_ERR_INVALID_ARG: *Invalid function address input, any previous function has been unregistered*

Properties

Prototyped in file “mcu_interrupts.h”.
Implemented in file “mcu_interrupts.c”.

Description

This function registers a callback function for an interrupt. If FIT_NO_FUNC, NULL, or any other invalid function address is passed for the callback argument then any previously registered callbacks are unregistered.

If one of the interrupts that is handled by this code is triggered then the interrupt handler will query this code to see if a valid callback function is registered. If one is found then the callback function will be called. If one is not found then the interrupt handler will clear the appropriate flag(s) and exit.

If the user has a callback function registered and wishes to no longer handle the interrupt then the user should call this function again with FIT_NO_FUNC as the *vector* parameter.

Reentrant

No.

Example

```
/* Prototype for callback function. */
void bus_error_callback(void * pdata);

void main (void)
{
    bsp_int_err_t err;

    /* Register bus_error_callback() to be called whenever a bus error occurs */
    err = R_BSP_InterruptWrite(BSP_INT_SRC_BUS_ERROR, bus_error_callback);

    if (BSP_INT_SUCCESS != err)
    {
        /* Error in registering callback. Alert user. */
        ...
    }
}

void bus_error_callback (void * pdata)
{
    /* Bus error has occurred. Handle accordingly. */
    ...
}
```

Special Notes:
Use of FIT_NO_FUNC is preferred over NULL since access to the address defined by FIT_NO_FUNC will cause a bus error which is easy for the user to catch. NULL typically resolves to 0 which is a valid address on RX MCUs.

5.14 R_BSP_InterruptRead()

Gets the callback for an interrupt if one is registered.

Format
```
   bsp_int_err_t R_BSP_InterruptRead(bsp_int_src_t vector,
                                     bsp_int_cb_t * callback);
```

Parameters
vector

Which interrupt to read the callback for. See Section 4.10.6.

callback

Pointer to where to store callback address. See Section 4.10.5.

Return Values
BSP_INT_SUCCESS:	*Successful, callback address has been returned*
BSP_INT_ERR_NO_REGISTERED_CALLBACK:	*No valid callback has been registered for this interrupt source.*

Properties
Prototyped in file “mcu_interrupts.h”.
Implemented in file “mcu_interrupts.c”.

Description
This function returns the callback function address for an interrupt if one has been registered. If a callback function has not been registered then an error is returned and nothing is stored to the *callback* address.

Reentrant
No.

Example
```
/* This function handles bus error interrupts. The address for this function
   is located in the bus error interrupt vector. */
void bus_error_isr (void)
{
    bsp_int_err_t  err;
    bsp_int_cb_t * user_callback;

    /* Bus error has occurred, see if a callback function has been registered */
    err = R_BSP_InterruptRead(BSP_INT_SRC_BUS_ERROR, user_callback);

    if (BSP_INT_SUCCESS == err)
    {
        /* Valid callback function found. Call it. */
        user_callback ();
    }

    /* Clear bus error flags. */
    ...
}
```

Special Notes:
None.

5.15 R_BSP_InterruptControl()

Controls various interrupt operations.

Format

```
bsp_int_err_t R_BSP_InterruptControl(bsp_int_src_t vector,
                                     bsp_int_cmd_t cmd,
                                     void *pdata)
```

Parameters

vector

Which interrupt to control for. See Section 4.10.6.

cmd

Interrupt control command. See Section 4.10.4.

pdata

Pointer to the argument for each command. Typecasted to void*. See Section 4.9.3.
Most of the commands do not need the argument and take FIT_NO_PTR for this parameter.
For BSP_INT_CMD_GROUP_INTERRUPT_ENABLE, specify the interrupt priority level for group interrupts as the argument.

Return Values

BSP_INT_SUCCESS:	*Successful*
BSP_INT_ERR_NO_REGISTERED_CALLBACK:	*No valid callback has been registered for this interrupt source.*
BSP_INT_ ERR_INVALID_ARG:	*The command passed is invalid.*
BSP_INT_ERR_UNSUPPORTED:	*This processing is not supported.*
BSP_INT_ERR_GROUP_STILL_ENABLED:	*Group interrupt request remains enabled.*

Properties

Prototyped in file “mcu_interrupts.h”

Description

This function controls the interrupt callback function call and enabling/disabling interrupts such as bus error interrupt, floating-point exception, NMI pin interrupt, and group interrupts.

When BSP_INT_CMD_GROUP_INTERRUPT_ENABLE is set as the interrupt control command, the interrupt request (IER) for group interrupts is enabled and also the interrupt priority level is set. The interrupt priority level set must be higher than the current level.

When BSP_INT_CMD_GROUP_INTERRUPT_DISABLE is set as the interrupt control command, the interrupt request (IER) for group interrupts is disabled. Note that the interrupt request (IER) for group interrupts cannot be disabled as long as all interrupt requests (GEN) caused by grouped interrupt sources are disabled.

Reentrant

No.

Example

```
bsp_int_err_t err;
bsp_int_ctrl_t int_ctrl;

err = BSP_INT_ERR_NO_INVALID_ARG;
int_ctrl.ipl = 0x0A;

err = R_BSP_InterruptControl(BSP_INT_SRC_BL0_SCI0_TEI0,
                             BSP_INT_CMD_GROUP_INTERRUPT_ENABLE,
                             &int_ctrl);
if (BSP_INIT_SUCCESS != err)
{
    /* NG processing */
 }
```

Special Notes:

None.

5.16 R_BSP_SoftwareDelay()

Delay the specified duration in units and return.

Format

```
bool R_BSP_SoftwareDelay(uint32_t delay, bsp_delay_units_t units)
```

Parameters

delay

The number of 'units' to delay.

units

The 'base' for the units specified. See Section 4.10.7.

Return Values

true: *True if delay executed*

false: *False if delay/units combination resulted in overflow/underflow*

Properties

Prototyped in file “r_bsp_common.h”.

Implemented in file “r_bsp_common.c”

Description

This is function that may be called for all MCU targets to implement a specific wait time.

The actual delay time is plus the overhead at a specified duration. The overhead changes under the influence of the compiler, operating frequency and ROM cache. When the operating frequency is low, or the specified duration in units of microsecond level, please note that the error becomes large.

Reentrant

No.

Example

```
bool ret;

/* Delay 5 seconds before returning */
ret = R_BSP_SoftwareDelay(5, BSP_DELAY_SECS);

if (true != ret)
{
   /* NG processing */
}

/* Delay 5 milliseconds before returning */
ret = R_BSP_SoftwareDelay(5, BSP_DELAY_MILLISECS);

if (true != ret)
{
/* NG processing */
}

/* Delay 50 microseconds before returning */
ret = R_BSP_SoftwareDelay(50, BSP_DELAY_MICROSECS);

if (true != ret)
{
/* NG processing */
}
```

Special Notes:

None.

5.17 R_BSP_GetIClkFreqHz()

Returns the system clock frequency.

Format

```
uint32_t R_BSP_GetClkFreqHz(void)
```

Parameters

None.

Return Values

System clock frequency specified by the r_bsp.

Properties

Prototyped in file “r_bsp_common.h”

Description

This function returns the system clock frequency. For example, when the system clock is set to 120 MHz in r_bsp_config_h and the r_bsp has completed to specify the clock setting, then even if the user changed the system clock frequency to 60 MHz, the return value is '60000000'.

Reentrant

Yes.

Example

```
uint32_t iclk;

iclk = R_BSP_GetIClkFreqHz();
```

Special Notes:

None.

5.18 R_BSP_StartupOpen()

Specifies settings to use the BSP and peripheral FIT modules. Call this function only when the BSP startup is disabled.

Format

```
void R_BSP_StartupOpen(void)
```

Parameters

None.

Return Values

None.

Properties

Prototyped in file “r_bsp_mcu_startup.h”

Description

This function performs initialization for the interrupt callback, register protection, and the hardware and pins. These processing are needed for using the BSP and peripheral FIT modules. Thus, this function must be called in the beginning of the main function.

Call this function only when the BSP startup is disabled.

Reentrant

No.

Example

```
void main (void)
{
   R_BSP_StartupOpen();

   ...
}
```

Special Notes:

The R_BSP_StartupOpen function performs a part of processing in the startup function. The following shows the processing.

Note 1: The MCU skips this procedure.
Note 2: The operation varies depending on the setting in the r_bsp_config.h.

Figure 5.2 Processing of the R_BSP_StartupOpen Function

5.19 R_BSP_VoltageLevelSetting()

This API function is used excessively with the RX66T and RX72T. It makes settings to the voltage level setting register (VOLSR) that are necessary in order to use the USB, AD, and RIIC peripheral modules. Call this function only when it is necessary to change the register settings.

Format

```
bool R_BSP_VoltageLevelSetting(uint8_t ctrl_ptn)
```

Parameters

ctrl_ptn

Register Setting Patterns

The following setting patterns cannot be selected at the same time.

When specifying more than one pattern at the same time, use the "|" (OR) operator.

- BSP_VOL_USB_POWEROFF and BSP_VOL_USB_POWERON
- BSP_VOL_AD_NEGATIVE_VOLTAGE_INPUT and BSP_VOL_AD_NEGATIVE_VOLTAGE_NOINPUT
- BSP_VOL_RIIC_4_5V_OROVER and BSP_VOL_RIIC_UNDER_4_5V

```
#define     BSP_VOL_USB_POWEROFF      (0x01)                  /* Updates the USBVON bit to 0. */
#define     BSP_VOL_USB_POWERON       (0x02)                  /* Updates the USBVON bit to 1. */
#define     BSP_VOL_AD_NEGATIVE_VOLTAGE_INPUT     (0x04)      /* Updates the PGAVLS bit to 0. */
#define     BSP_VOL_AD_NEGATIVE_VOLTAGE_NOINPUT (0x08)      /* Updates the PGAVLS bit to 1. */
#define     BSP_VOL_RIIC_4_5V_OROVER (0x10)                 /* Updates the RICVLS bit to 0. */
#define     BSP_VOL_RIIC_UNDER_4_5V   (0x20)                /* Updates the RICVLS bit to 1. */
```

Return Values

true: / Processing completed, register successfully updated. */*

false: / The function was called under the following conditions, so the register setting was not updated. */*

- *Setting patterns that cannot be selected at the same time were selected.*
- *A setting pattern related to the USB was selected when the USB was not in the module stop state.*
- *A setting pattern related to the AD was selected when the AD was not in the module stop state.*
- *A setting pattern related to the RIIC was selected when the RIIC was not in the module stop state.*

Properties

Prototyped in file "r_bsp_cpu.h"

Description

This function initializes the voltage level setting register (VOLSR), which is necessary in order to use the USB, AD and RIIC peripheral modules. When specifying a setting pattern related to the USB, call this function before the USB is released from the module stop state. When specifying a setting pattern related to the AD, call this function before the AD (unit 0 and unit 1) is released from the module stop state. When specifying a setting pattern related to the RIIC, call this function before the RIIC is released from the module stop state. If the function is called with a setting pattern related to the USB specified after the USB is released from the module stop state, the function returns "false" as the return value and does not update the register settings. If the function is called with a setting pattern related to the AD specified after the AD (unit 0 and unit 1) is released from the module stop state, the function returns "false" as the return value and does not update the register settings. Finally, if the function is called with a setting pattern related to the RIIC specified after the RIIC is released from the module stop state, the function returns "false" as the return value and does not update the register settings. In the BSP the initial settings are specified in accordance with the macro settings listed in 3.2.11, MCU Voltage, and 3.2.14, Negative Voltage Input Settings for AD Pins.

Reentrant

No.

Example

```
void main (void)
{
   bool ret;

   /* USBVON bit set to 1. */
   ret = R_BSP_VoltageLevelSetting(BSP_VOL_USB_POWERON);
   if (true != ret)
   {
      /* NG processing */
   }

   ...

   /* PGAVLS and USBVON bit set to 0. */
   ret = R_BSP_VoltageLevelSetting(BSP_VOL_AD_NEGATIVE_VOLTAGE_NOINPUT |
BSP_VOL_USB_POWEROFF);
   if (true != ret)
   {
      /* NG processing */
   }

   ...
}
```

Special Notes:

None.

5.20 R_BSP_InterruptRequestEnable()

Enable the specified interrupt request.

Format

```
void R_BSP_InterruptRequestEnable (uint32_t vector)
```

Parameters

vector

Interrupt vector number.

Return Values

None.

Properties

Prototyped in file “r_bsp_interrupts.h”.

Description

Enable the specified interrupt request. Calculate the corresponding IER [m].IEN [j] from the vector number of the argument, and set “1” to that bit.

The macro defined in iodefine.h can be used to the setting of the argument "vector". A description example is shown in Example.

Reentrant

Yes.

Example

```
void main(void)
{
   /* Enable interrupt of CMT0. */
   R_BSP_InterruptRequestEnable(VECT(CMT0, CMI0));
}
```

Special Notes:

When setting an immediate value for an argument “vector”, the argument must be 0 to 255.

Don't set the vector number of the reserved interrupt source to the argument.

5.21 R_BSP_InterruptRequestDisable()

Disable the specified interrupt request.

Format

```
void R_BSP_InterruptRequestDisable (uint32_t vector)
```

Parameters

vector

Interrupt vector number.

Return Values

None.

Properties

Prototyped in file “r_bsp_interrupts.h”.

Description

Disable the specified interrupt request. Calculate the corresponding IER [m].IEN [j] from the vector number of the argument, and clear “0” to that bit.

The macro defined in iodefine.h can be used to the setting of the argument "vector". A description example is shown in Example.

Reentrant

Yes.

Example

```
void main(void)
{
   /* Disable interrupt of CMT0. */
   R_BSP_InterruptRequestDisable(VECT(CMT0, CMI0));
}
```

Special Notes:

When setting an immediate value for an argument “vector”, the argument must be 0 to 255.

Don't set the vector number of the reserved interrupt source to the argument.

5.22 R_BSP_ConfigClockSetting ()

This function is available only on the RX23W. This function is used by Bluetooth® Low Energy Protocol Stack Basic Package.

Format

```
void R_BSP_ConfigClockSetting (void)
```

Parameters

None.

Return Values

None.

Properties

Prototyped in file “r_bsp_clock.h”.

Description

Under certain conditions, Bluetooth® Low Energy Protocol Stack Basic Package uses this function to set the clock.

For details, refer to Bluetooth® Low Energy Protocol Stack Basic Package User’s Manual (R01UW0205)

Reentrant

No.

Special Notes:

None.

6. Intrinsic Functions

In this module, common macros are defined so that intrinsic functions can be used without relying on the compiler. The common macros determine the compiler to be used and replace it with the intrinsic function of each compiler. The common macros are defined in r_rx_intrinsic_functions.h. The common macros available in this module are shown in Table 6.1 to Table 6.7.

The argument and return value type of intrinsic functions may differ depending on the compiler, and the common macros cast the argument and return type according to the CCRX.

Example

```
#include “platform.h”     /* r_rx_intrinsic_functions.h is included */

void main (void)
{
    /* The argument and return value type are declared according to the CCRX
intrinsic function*/
    unsigned long  args = 0x12345678;
    unsigned long  ret;

    ret = R_BSP_REVW(args);
}
```

Intrinsic functions may not be supported by some compilers. Those functions are substituted by the API function of BSP. If the Category in the table is “○”, the common macro is replaced with an intrinsic function. If the Category in the table is “BSP API”, the common macro is replaced with the API function of BSP.

For specifications and usage of intrinsic functions, refer to the manual of their respective compilers.

Table 6.1 Common Macros of Intrinsic Functions (1/7)

Common Macros	Compiler	Functions	Category
R_BSP_MAX(x, y)	ccrx	signed long max(signed long data1, signed long data2)	○
	gnuc	signed long R_BSP_Max(signed long data1, signed long data2)	BSP API
	iccrx	signed long __MAX(signed long, signed long)	○
R_BSP_MIN(x, y)	ccrx	signed long min(signed long data1, signed long data2)	○
	gnuc	signed long R_BSP_Min(signed long data1, signed long data2)	BSP API
	iccrx	signed long __MIN(signed long, signed long)	○
R_BSP_REVL(x)	ccrx	unsigned long revl(unsigned long data)	○
	gnuc	uint32_t __builtin_bswap32(uint32_t x)	○
	iccrx	unsigned long __REVL(unsigned long)	○
R_BSP_REVW(x)	ccrx	unsigned long revw(unsigned long data)	○
	gnuc	int __builtin_rx_revw(int)	○
	iccrx	unsigned long __REVW(unsigned long)	○

Table 6.2 Common Macros of Intrinsic Functions (2/7)

Common Macros	Compiler	Functions	Category
R_BSP_EXCHANGE(x, y)	ccrx	void xchg(signed long *data1, signed long *data2)	○
	gnuc	void __builtin_rx_xchg (int *, int *)	○
	iccrx	void _builtin_xchg(signed long *, signed long *)	○
R_BSP_RMPAB(w, x, y, z)	ccrx	long rmpab(long long init, unsigned long count, signed char *addr1, signed char *addr2)	○
	gnuc	long R_BSP_MulAndAccOperation_B(long long init, unsigned long count, signed char *addr1, signed char *addr2)	BSP API
	iccrx	long rmpab(long long init, unsigned long count, signed char *addr1, signed char *addr2)	○
R_BSP_RMPAW(w, x, y, z)	ccrx	long rmpaw(long long init, unsigned long count, short *addr1, short *addr2)	○
	gnuc	long R_BSP_MulAndAccOperation_W(long long init, unsigned long count, short *addr1, short *addr2)	BSP API
	iccrx	long rmpaw(long long init, unsigned long count, short *addr1, short *addr2)	○
R_BSP_RMPAL(w, x, y, z)	ccrx	long rmpal(long long init, unsigned long count, long *addr1, long *addr2)	○
	gnuc	long R_BSP_MulAndAccOperation_L(long long init, unsigned long count, long *addr1, long *addr2)	BSP API
	iccrx	long rmpal(long long init, unsigned long count, long *addr1, long *addr2)	○
R_BSP_ROLC(x)	ccrx	unsigned long rolc(unsigned long data)	○
	gnuc	unsigned long R_BSP_RotateLeftWithCarry(unsigned long data)	BSP API
	iccrx	unsigned long __ROLC(unsigned long)	○
R_BSP_RORC(x)	ccrx	unsigned long rorc(unsigned long data)	○
	gnuc	unsigned long R_BSP_RotateRightWithCarry(unsigned long data)	BSP API
	iccrx	unsigned long __RORC(unsigned long)	○
R_BSP_ROTL(x, y)	ccrx	unsigned long rotl(unsigned long data, unsigned long num)	○
	gnuc	unsigned long R_BSP_RotateLeft(unsigned long data, unsigned long num)	BSP API
	iccrx	unsigned long __ROTL(unsigned long, unsigned long)	○

Table 6.3 Common Macros of Intrinsic Functions (3/7)

Common Macros	Compiler	Functions	Category
R_BSP_ROTR(x, y)	ccrx	unsigned long rotr (unsigned long data, unsigned long num)	○
	gnuc	unsigned long R_BSP_RotateRight(unsigned long data, unsigned long num)	BSP API
	iccrx	unsigned long __ROTR(unsigned long, unsigned long)	○
R_BSP_BRK()	ccrx	void brk(void)	○
	gnuc	void __builtin_rx_brk (void)	○
	iccrx	void __break(void)	○
R_BSP_INT(x)	ccrx	void int_exception(signed long num)	○
	gnuc	void __builtin_rx_int (int)	○
	iccrx	void __software_interrupt(unsigned char)	○
R_BSP_WAIT()	ccrx	void wait(void)	○
	gnuc	void __builtin_rx_wait (void)	○
	iccrx	void __wait_for_interrupt(void)	○
R_BSP_NOP()	ccrx	void nop(void)	○
	gnuc	__asm("nop")	○
	iccrx	void __no_operation(void)	○
R_BSP_SET_IPL(x)	ccrx	void set_ipl(signed long level)	○
	gnuc	void __builtin_rx_mvtipl (int)	○
	iccrx	void __set_interrupt_level(__ilevel_t)	○
R_BSP_GET_IPL()	ccrx	unsigned char get_ipl(void)	○
	gnuc	uint32_t R_BSP_CpuInterruptLevelRead (void)	BSP API
	iccrx	__ilevel_t __get_interrupt_level(void)	○
R_BSP_SET_PSW(x)	ccrx	void set_psw(unsigned long data)	○
	gnuc	void __builtin_rx_mvtc (int reg, int val)	○
	iccrx	void __set_PSW_register(unsigned long)	○
R_BSP_GET_PSW()	ccrx	unsigned long get_psw(void)	○
	gnuc	int __builtin_rx_mvfc (int)	○
	iccrx	unsigned long __get_PSW_register(void)	○
R_BSP_SET_FPSW(x)	ccrx	void set_fpsw(unsigned long data)	○
	gnuc	void __builtin_rx_mvtc (int reg, int val)	○
	iccrx	void __set_FPSW_register(unsigned long)	○
R_BSP_GET_FPSW()	ccrx	unsigned long get_fpsw(void)	○
	gnuc	int __builtin_rx_mvfc (int)	○
	iccrx	unsigned long __get_FPSW_register(void)	○

Table 6.4 Common Macros of Intrinsic Functions (4/7)

Common Macros	Compiler	Functions	Category
R_BSP_SET_USP(x)	ccrx	void set_usp(void *data)	○
	gnuc	void __builtin_rx_mvtc (int reg, int val)	○
	iccrx	void __set_USP_register(unsigned long)	○
R_BSP_GET_USP()	ccrx	void *get_usp(void)	○
	gnuc	int __builtin_rx_mvfc (int)	○
	iccrx	unsigned long __get_USP_register(void)	○
R_BSP_SET_ISP(x)	ccrx	void set_isp(void *data)	○
	gnuc	void __builtin_rx_mvtc (int reg, int val)	○
	iccrx	void __set_ISP_register(unsigned long)	○
R_BSP_GET_ISP()	ccrx	void *get_isp(void)	○
	gnuc	int __builtin_rx_mvfc (int)	○
	iccrx	unsigned long __get_ISP_register(void)	○
R_BSP_SET_INTB(x)	ccrx	void set_intb (void *data)	○
	gnuc	void __builtin_rx_mvtc (int reg, int val)	○
	iccrx	void __set_interrupt_table(unsigned long address)	○
R_BSP_GET_INTB()	ccrx	void *get_intb(void)	○
	gnuc	int __builtin_rx_mvfc (int)	○
	iccrx	unsigned long __get_interrupt_table(void)	○
R_BSP_SET_BPSW(x)	ccrx	void set_bpsw(unsigned long data)	○
	gnuc	void __builtin_rx_mvtc (int reg, int val)	○
	iccrx	void R_BSP_SetBPSW(uint32_t data)	BSP API
R_BSP_GET_BPSW()	ccrx	unsigned long get_bpsw(void)	○
	gnuc	int __builtin_rx_mvfc (int)	○
	iccrx	uint32_t R_BSP_GetBPSW(void)	BSP API
R_BSP_SET_BPC(x)	ccrx	void set_bpc(void *data)	○
	gnuc	void __builtin_rx_mvtc (int reg, int val)	○
	iccrx	void R_BSP_SetBPC(void *data)	BSP API
R_BSP_GET_BPC()	ccrx	void *get_bpc(void)	○
	gnuc	int __builtin_rx_mvfc (int)	○
	iccrx	void *R_BSP_GetBPC(void)	BSP API
R_BSP_SET_FINTV(x)	ccrx	void set_fintv(void *data)	○
	gnuc	void __builtin_rx_mvtc (int reg, int val)	○
	iccrx	void __set_FINTV_register(__fast_int_f)	○

Table 6.5 Common Macros of Intrinsic Functions (5/7)

Common Macros	Compiler	Functions	Category
R_BSP_GET_FINTV()	ccrx	void *get_fintv(void)	○
	gnuc	int __builtin_rx_mvfc (int)	○
	iccrx	__fast_int_f __get_FINTV_register(void)	○
R_BSP_EMUL(x, y)	ccrx	signed long long emul(signed long data1, signed long data2)	○
	gnuc	signed long long R_BSP_SignedMultiplication(signed long data1, signed long data2)	BSP API
	iccrx	signed long long R_BSP_SignedMultiplication(signed long data1, signed long data2)	BSP API
R_BSP_EMULU(x, y)	ccrx	unsigned long long emulu(unsigned long data1, unsigned long data2)	○
	gnuc	unsigned long long R_BSP_UnsignedMultiplication(unsigned long data1, unsigned long data2)	BSP API
	iccrx	unsigned long long R_BSP_UnsignedMultiplication(unsigned long data1, unsigned long data2)	BSP API
R_BSP_CHG_PMUSR()	ccrx	void chg_pmusr(void)	○
	gnuc	void R_BSP_ChangeToUserMode(void)	BSP API
	iccrx	void R_BSP_ChangeToUserMode(void)	BSP API
R_BSP_SET_ACC(x)	ccrx	void set_acc(signed long long data)	○
	gnuc	void R_BSP_SetACC(signed long long data)	BSP API
	iccrx	void R_BSP_SetACC(signed long long data)	BSP API
R_BSP_GET_ACC()	ccrx	signed long long get_acc(void)	○
	gnuc	signed long longR_BSP_GetACC(void)	BSP API
	iccrx	signed long longR_BSP_GetACC(void)	BSP API
R_BSP_SETPSW_I()	ccrx	void setpsw_i(void)	○
	gnuc	void __builtin_rx_setpsw (int)	○
	iccrx	void __enable_interrupt(void)	○
R_BSP_CLRPSW_I()	ccrx	void clrpsw_i(void)	○
	gnuc	void __builtin_rx_clrpsw (int)	○
	iccrx	void __disable_interrupt(void)	○

Table 6.6 Common Macros of Intrinsic Functions (6/7)

Common Macros	Compiler	Functions	Category
R_BSP_MACL(x, y, z)	ccrx	long macl(short *data1, short *data2, unsigned long count)	○
	gnuc	long R_BSP_MulAndAccOperation_2byte(short *data1, short *data2, unsigned long count)	BSP API
	iccrx	long __macl(short * data1, short * data2, unsigned long count)	○
R_BSP_MACW1(x, y, z)	ccrx	short macw1(short *data1, short *data2, unsigned long count)	○
	gnuc	short R_BSP_MulAndAccOperation_FixedPoint1(short *data1, short *data2, unsigned long count)	BSP API
	iccrx	short __macw1(short * data1, short * data2, unsigned long count)	○
R_BSP_MACW2(x, y, z)	ccrx	short macw2(short *data1, short *data2, unsigned long count)	○
	gnuc	short R_BSP_MulAndAccOperation_FixedPoint2(short *data1, short *data2, unsigned long count)	BSP API
	iccrx	short __macw2(short * data1, short * data2, unsigned long count)	○
R_BSP_SET_EXTB(x)	ccrx	void set_extb(void *data)	○
	gnuc	void __builtin_rx_mvtc (int reg, int val)	○
	iccrx	void R_BSP_SetEXTB(void *value)	BSP API
R_BSP_GET_EXTB()	ccrx	void * get_extb(void)	○
	gnuc	int __builtin_rx_mvfc (int) `0xD extb'	○
	iccrx	void *R_BSP_GetEXTB(void)	BSP API
R_BSP_BIT_CLEAR(x,y)	ccrx	void __bclr(unsigned char *data, unsigned long bit)	○
	gnuc	void R_BSP_BitClear(uint8_t *data, uint32_t bit)	BSP API
	iccrx	void R_BSP_BitClear(uint8_t *data, uint32_t bit)	BSP API
R_BSP_BIT_SET(x,y)	ccrx	void __bset(unsigned char *data, unsigned long bit)	○
	gnuc	void R_BSP_BitSet(uint8_t *data, uint32_t bit)	BSP API
	iccrx	void R_BSP_BitSet(uint8_t *data, uint32_t bit)	BSP API
R_BSP_BIT_REVERSE(x,y)	ccrx	void __bnot(unsigned char *data, unsigned long bit)	○
	gnuc	void R_BSP_BitReverse(uint8_t *data, uint32_t bit)	BSP API
	iccrx	void R_BSP_BitReverse(uint8_t *data, uint32_t bit)	BSP API

Table 6.7 Common Macros of Intrinsic Functions (7/7)

Common Macros	Compiler	Functions	Category
R_BSP_SET_DPSW(x)	ccrx	void __set_dpsw(unsigned long data)	○
	gnuc	void R_BSP_SET_DPSW(uint32_t data)	BSP API
	iccrx	void R_BSP_SET_DPSW(uint32_t data)	BSP API
R_BSP_GET_DPSW()	ccrx	unsigned long __get_dpsw(void)	○
	gnuc	uint32_t R_BSP_GET_DPSW(void)	BSP API
	iccrx	uint32_t R_BSP_GET_DPSW(void)	BSP API
R_BSP_SET_DECNT(x)	ccrx	void __set_decnt(unsigned long data)	○
	gnuc	void R_BSP_SET_DECNT(uint32_t data)	BSP API
	iccrx	void R_BSP_SET_DECNT(uint32_t data)	BSP API
R_BSP_GET_DECNT()	ccrx	unsigned long __get_decnt(void)	○
	gnuc	uint32_t R_BSP_GET_DECNT(void)	BSP API
	iccrx	uint32_t R_BSP_GET_DECNT(void)	BSP API
R_BSP_GET_DEPC()	ccrx	void *__get_depc(void)	○
	gnuc	void *R_BSP_GET_DEPC(void)	BSP API
	iccrx	void *R_BSP_GET_DEPC(void)	BSP API
R_BSP_INIT_TFU()	ccrx	void __init_tfu(void)	○
	gnuc	void R_BSP_InitTFU(void)	BSP API
	iccrx	void R_BSP_InitTFU(void)	BSP API

7. Project Setup

This section details adding the r_bsp to your project.

7.1 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to "RX Smart Configurator User Guide: e^2 studio (R20AN0451)" for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio
By using the FIT Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to "Adding Firmware Integration Technology Modules to Projects (R01AN1723)" for details.

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to "RX Smart Configurator User Guide: CS+ (R20AN0470)" for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)" for details.

7.2 Adding FIT Modules to the IAR Project

This section describes how to add FIT modules to IAR projects. You can add a FIT module to an IAR project by converting a CCRX project to which the FIT module has been added to an IAR project. Use the following procedure to create an IAR project.

In this explanation, IAR Embedded Workbench for Renesas RX 4.10.1 is used.

(1) Open a workspace for e2 studio.

(2) Create a CCRX project by following the procedure in “7.1 Adding the FIT Module to Your Project”.

(3) Start IAR Embedded Workbench for Renesas RX.

(4) Click “Tools >> Convert To IAR for RX...”.

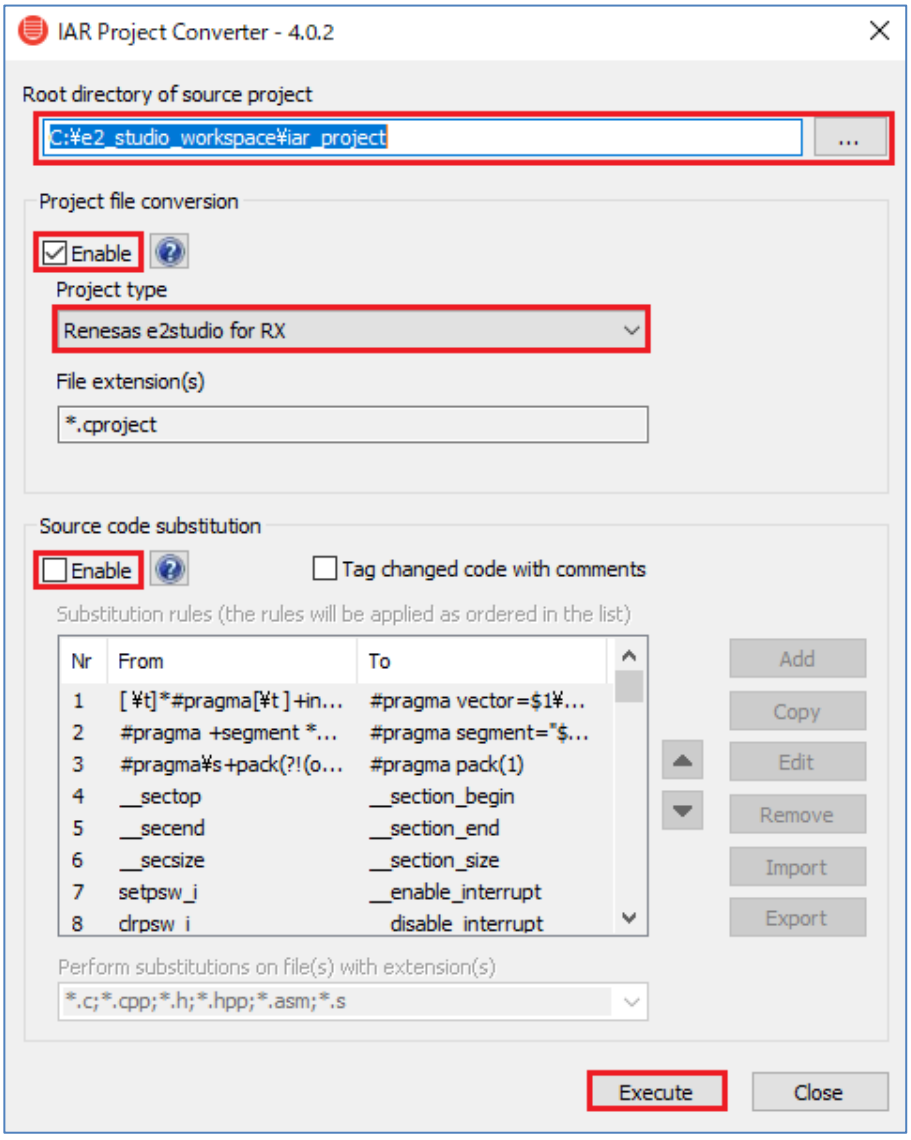
(5) For the Root directory of source project, select “CCRX project”.

(6) For Project file conversion, tick the “Enable” check box.

(7) For Project type, select “Renesas e2studio for RX”.

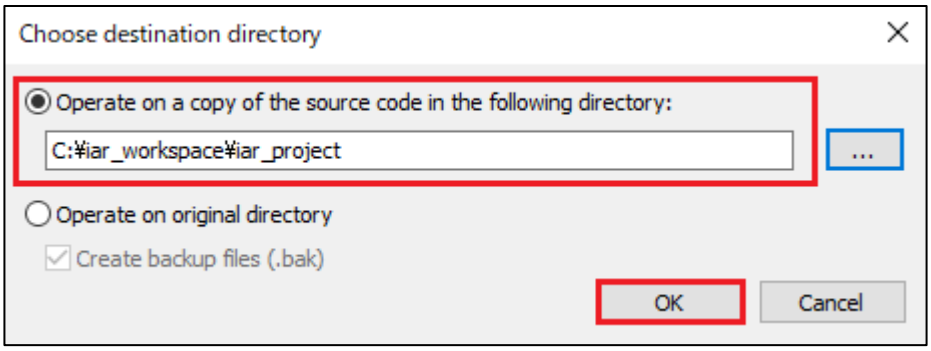
(8) For Source code substitution, de-select the “Enable” check box.

(9) Click “Execute”.

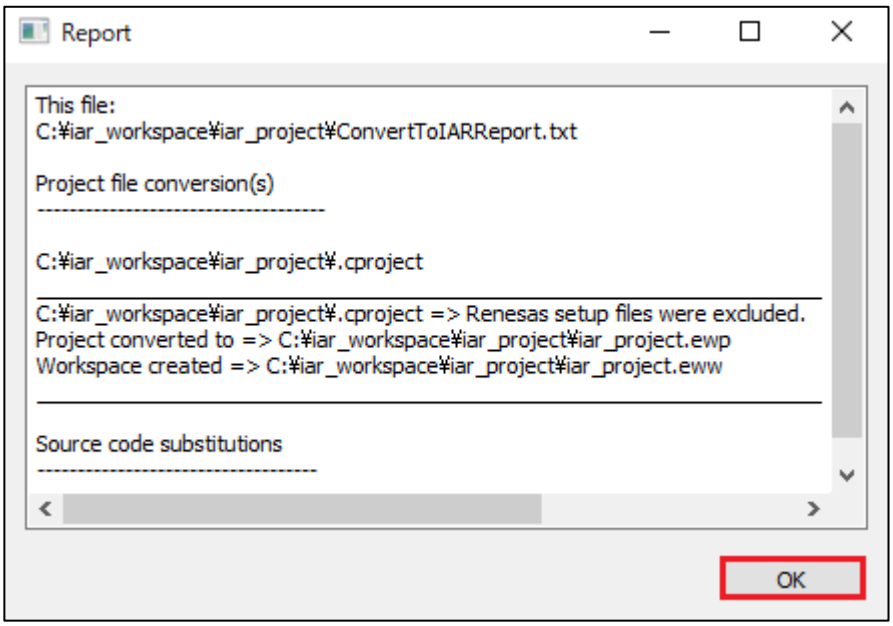


(10) On the Choose destination directory window, tick the “Operate on a copy of the source code in the following directory” check box.

(11) Select “IAR project” and click “OK”.

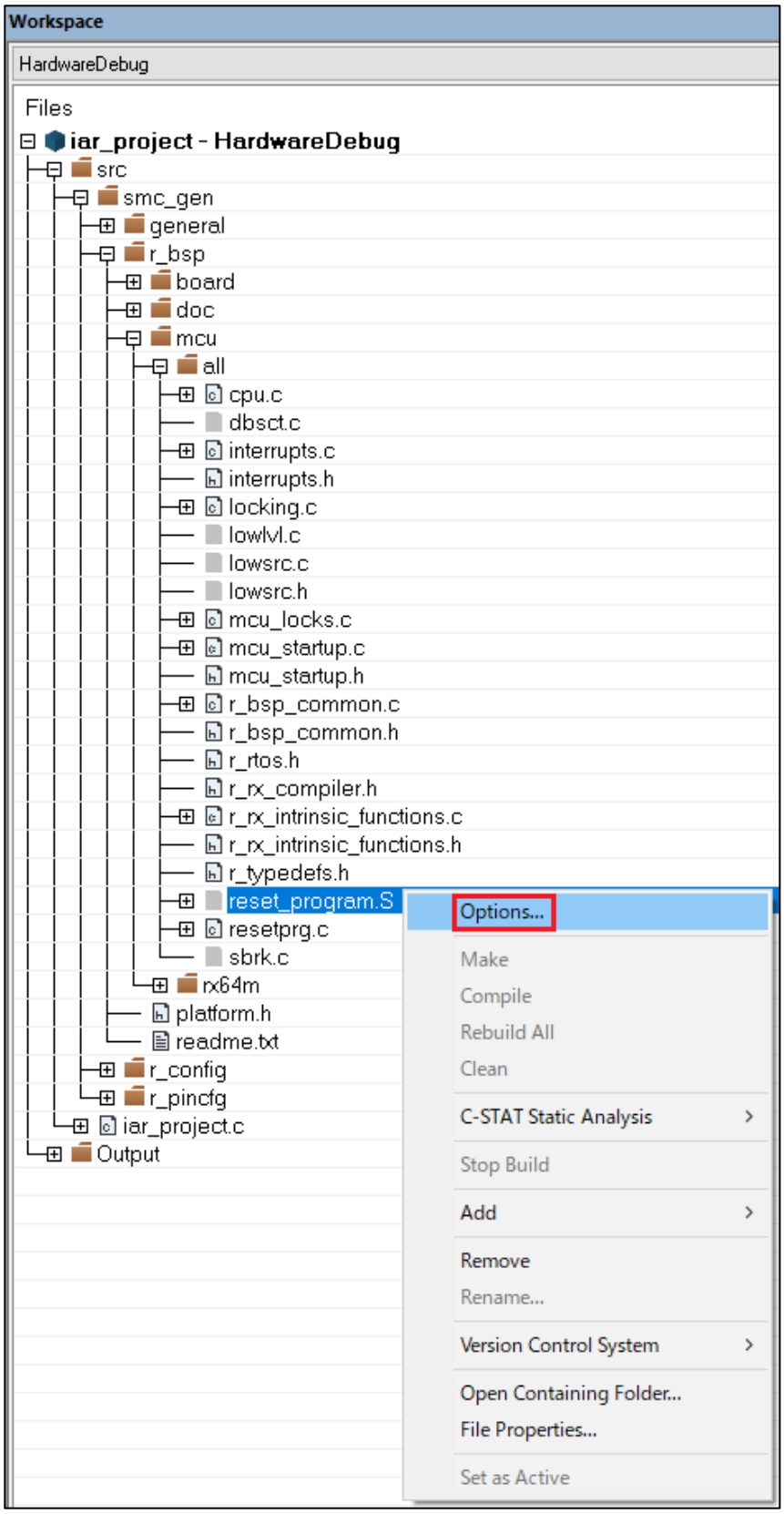


(12) On the Report window, click “OK”.

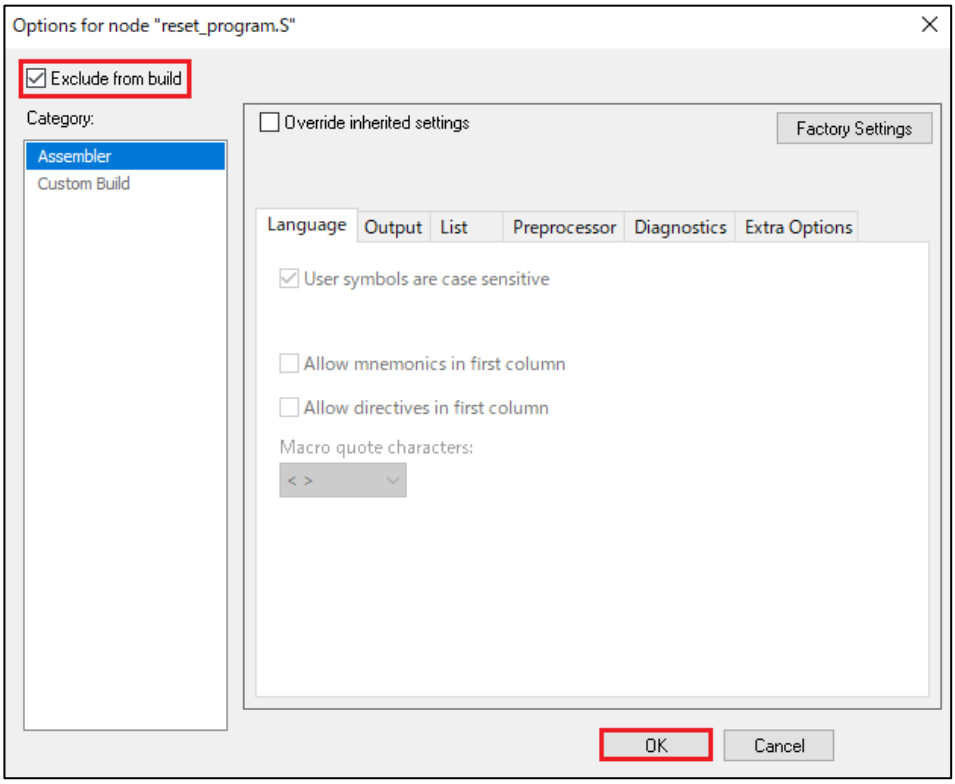


(13) Click “Project >> Add Existing Project…” and open .ewp file of the IAR project.

(14) Right-click on reset_program.s and select “Options…”.



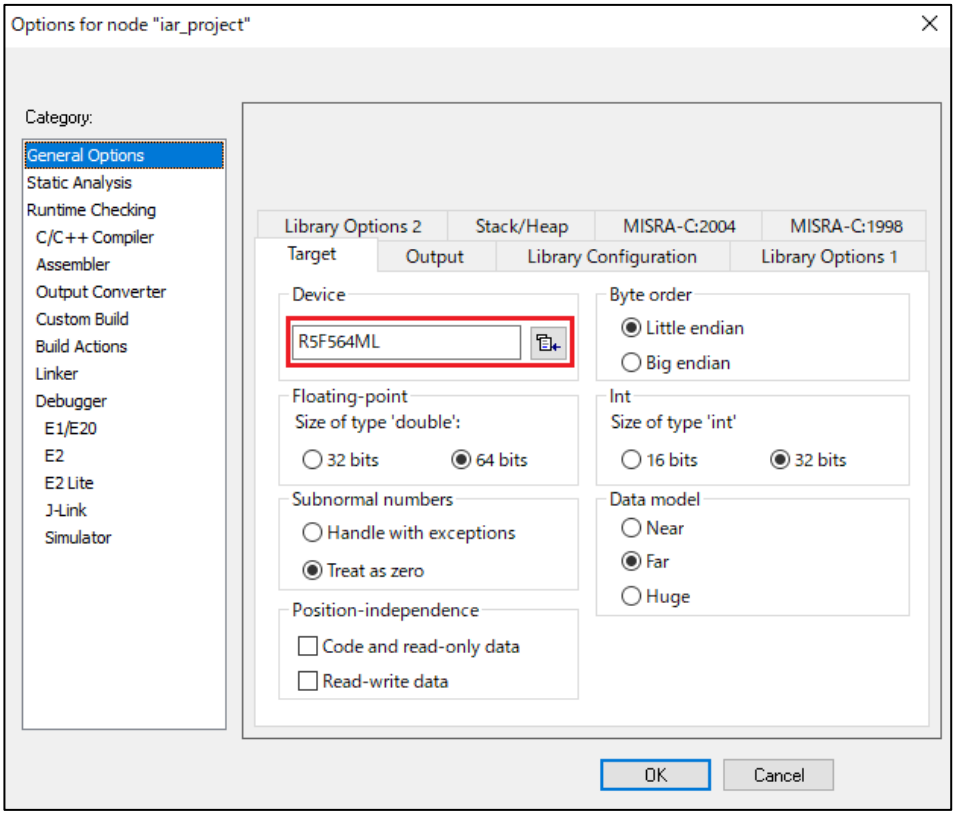
(15) On the Option window, tick the “Exclude from build” check box.



(16) Right-click on the project and click “Options…”.

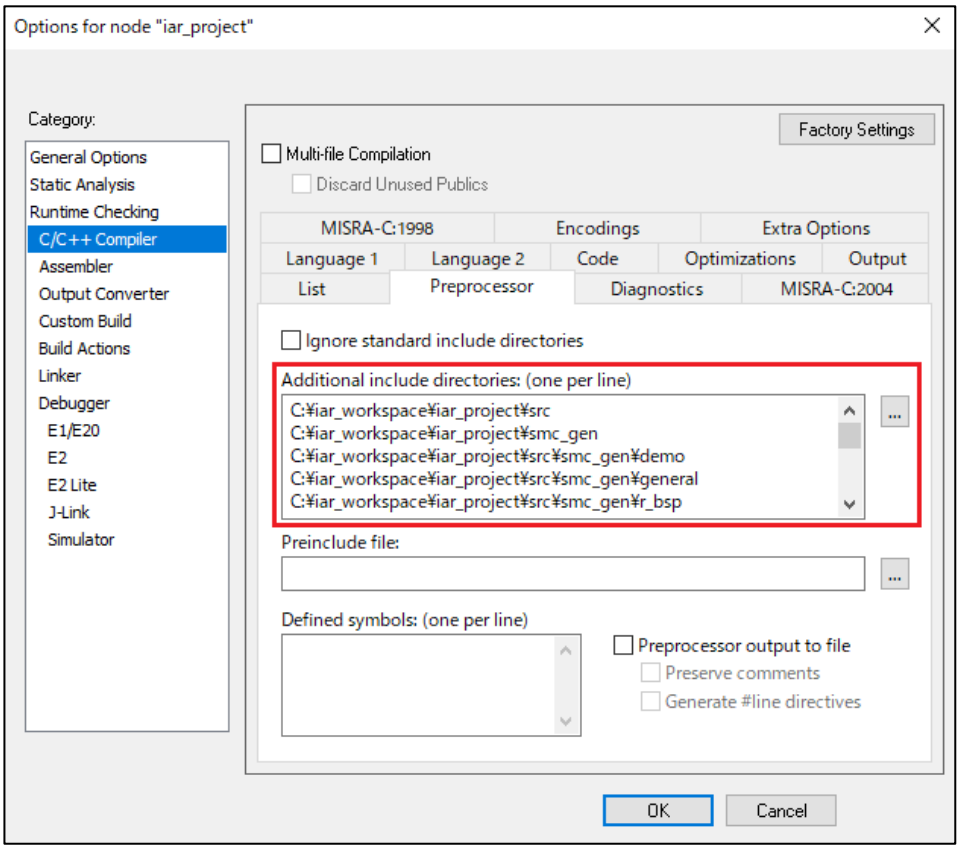
(17) Select “Target” on the General Options tab.

(18) For “Device”, select a device to use.



(19) Select “Preprocessor” on the C/C++ Compiler tab.

(20) Set the same include path as the CCRX project.

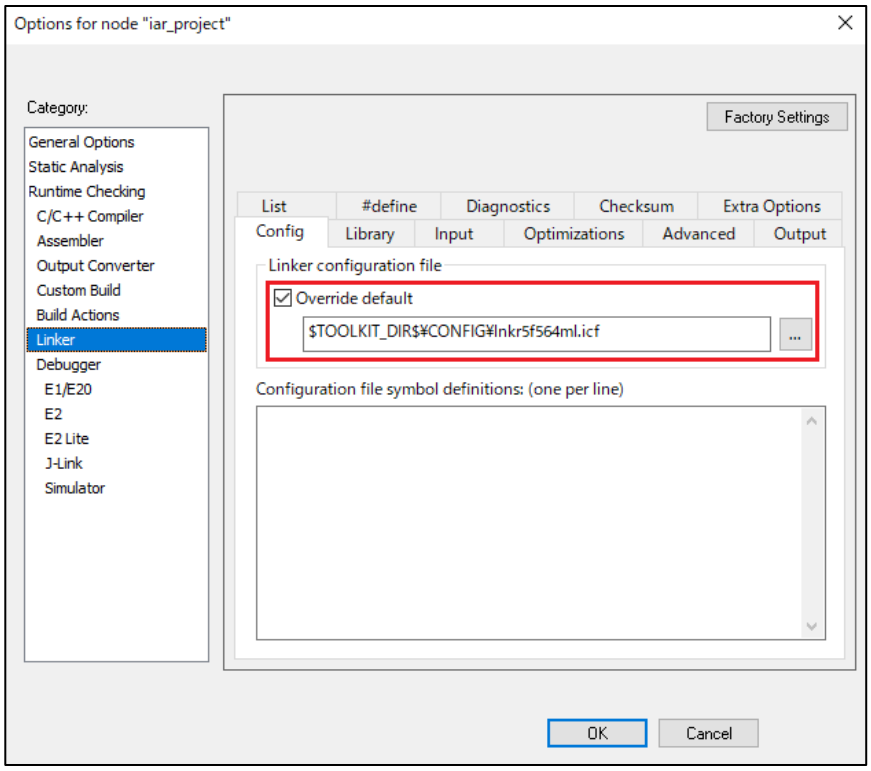


The following are setup examples of include paths that are required to use rx64m.

C:\iar_workspace\iar_project\src
C:\iar_workspace\iar_project\src\smc_gen
C:\iar_workspace\iar_project\src\smc_gen\general
C:\iar_workspace\iar_project\src\smc_gen\r_bsp
C:\iar_workspace\iar_project\src\smc_gen\r_bsp\board
C:\iar_workspace\iar_project\src\smc_gen\r_bsp\board\generic_rx64m
C:\iar_workspace\iar_project\src\smc_gen\r_bsp\mcu
C:\iar_workspace\iar_project\src\smc_gen\r_bsp\mcu\all
C:\iar_workspace\iar_project\src\smc_gen\r_bsp\mcu\rx64m
C:\iar_workspace\iar_project\src\smc_gen\r_bsp\mcu\rx64m\register_access
C:\iar_workspace\iar_project\src\smc_gen\r_config
C:\iar_workspace\iar_project\src\smc_gen\r_pincfg

(21) Select “Config” on the Linker tab.

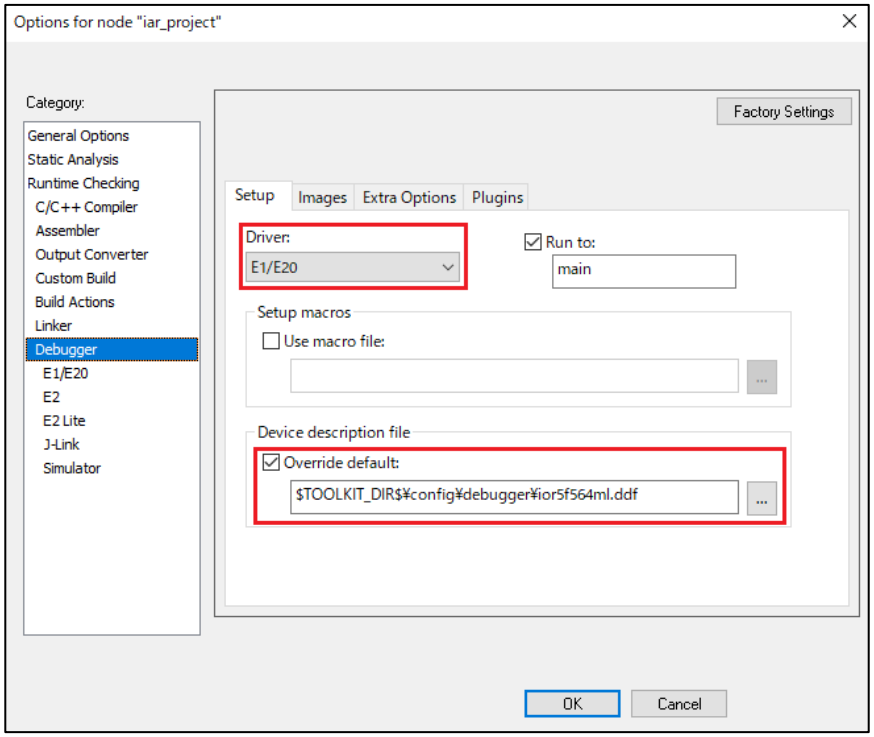
(22) For the linker configuration file, tick the “Override default” check box. Then, select “the target device.icf file”.



(23) Select “Setup” on the Debugger tab.

(24) For the driver, select ”Emulator”.

(25) For the device description file, tick the “Override default” check box, and then select “the target device.dff file”.



(26) Click “Project >> Rebuild All”.

(27) Click “E1/E20 Emulator >> Hardware Setup…”.

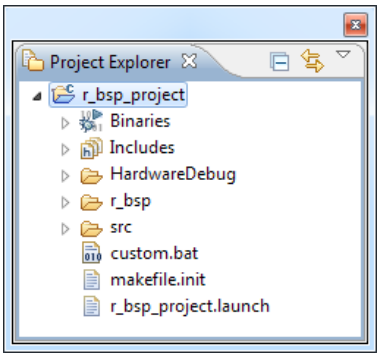
(28) On the hardware setup window, set ”Debug Configurations” and press OK.

(29) Click “Project >> Download and Debug”.

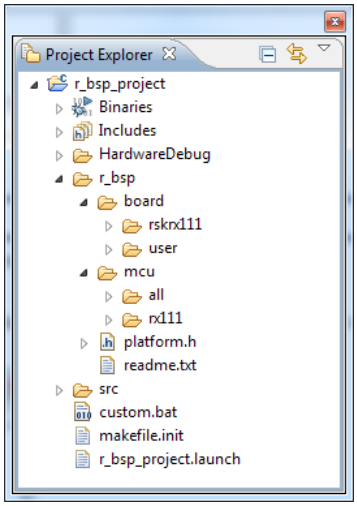
8. Adding r_bsp manually

This section gives instruction on how to add an r_bsp to an e^2 studio project manually (without use of the FIT Plug-in).

1. Copy the r_bsp folder to your e^2 studio project's root. Once clicking Copy in Windows you can right-click on your project in e^2 studio and click Paste.

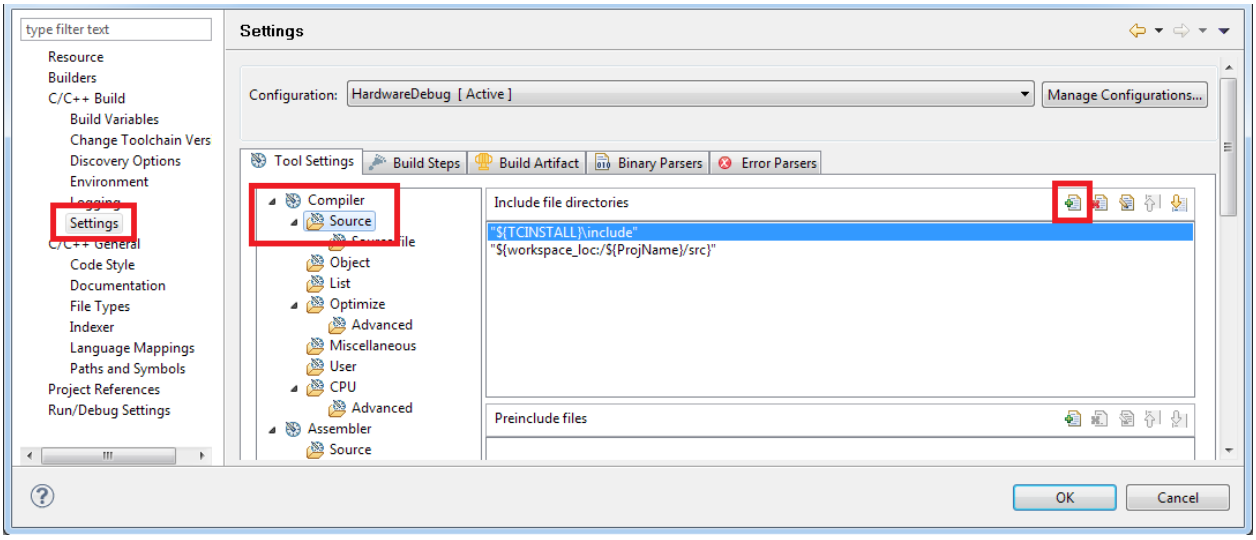


2. Expand the *r_bsp* >> *board* folder and delete all folders except the one for the board you are using. You can leave the 'user' directory if you wish to have a directory to start off with when you create your own BSP.
3. Expand the *r_bsp* >> *mcu* folder and delete all folders except the one for your MCU group and the one named *all*.

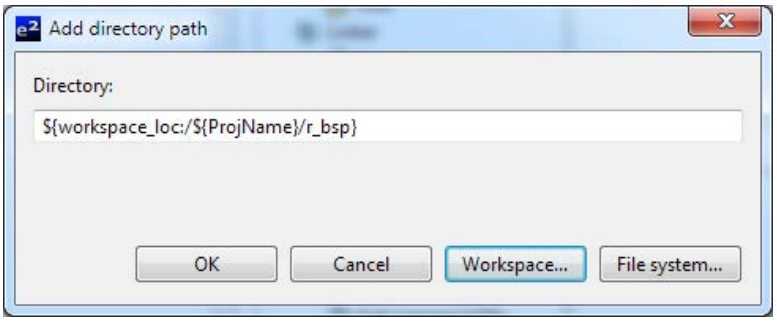


4. It is recommended to create a directory to store all FIT configuration files. Having one place for configuration files make them easy to find and easy to backup. The default name for this folder is *r_config*. If an *r_config* folder was not included in your r_bsp zip file then we will create one here. Create an *r_config* folder for your project by right-clicking on your project and choosing New >> Folder. In the window that pops up enter 'r_config' for the folder name and click Finish.
5. We will now setup include paths for the r_bsp and r_config folders. Right-click on your project and click Properties.
6. Under 'Tool Settings' select Compiler >> Source.

7. In the ‘Include file directories’ box click the ‘Add’ button.



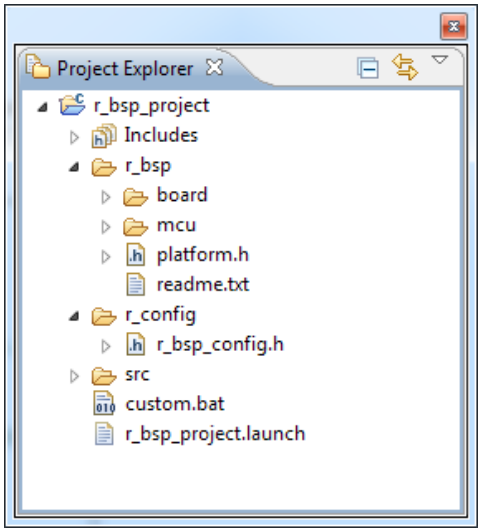
8. The ‘Add directory path’ window will pop up; click the Workspace button.
9. In the ‘Folder selection’ window choose the r_bsp folder and click OK.



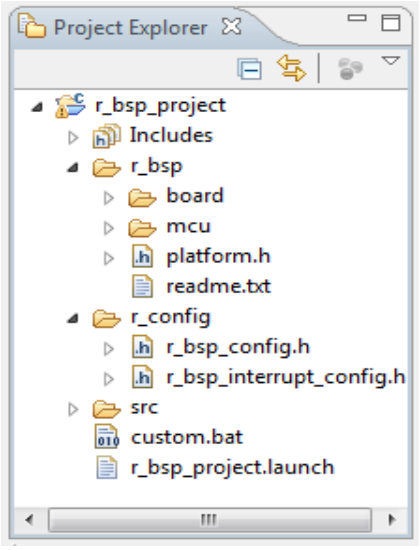
10. Verify that your window looks like the one above and click OK.
11. Back in the main Properties window verify that you now have an include path for the r_bsp.
12. Follow the same steps to add an include path for the *r_config* folder.
13. Back in the main Properties window verify that you now have an include path for the *r_bsp* and *r_config* folders and click Apply. Click OK to return to your project.
14. Which board is being used needs to be selected in the *platform.h* header file. Open up *platform.h* and uncomment the #include for the board you are using. In this example the RSKRX111 is being used so the #include for “./board/rskrx111/r_bsp.h” is uncommented.

```
86 /* RSKRX63N */
87 //#include "./board/rskrx63n/r_bsp.h"
88
89 /* RSKRX63T_64PIN */
90 //#include "./board/rskrx63t_64pin/r_bsp.h"
91
92 /* RSKRX63T_144PIN */
93 //#include "./board/rskrx63t_144pin/r_bsp.h"
94
95 /* RDKRX63N */
96 //#include "./board/rdkrx63n/r_bsp.h"
97
98 /* RSKRX210 */
99 //#include "./board/rskrx210/r_bsp.h"
100
101 /* RSKRX111 */
102 #include "./board/rskrx111/r_bsp.h"
```

15. In order to configure the r_bsp the user needs to create an *r_bsp_config.h* file. Copy the *r_bsp_config_reference.h* file from your *board* folder and paste it into the *r_config* folder. Right-click on the file in the *r_config* folder and click Rename. Rename the file to *r_bsp_config.h*. If the MCU has an *r_bsp_interrupt_config_reference.h* file, copy that file as well and rename it to *r_bsp_interrupt_config.h*.



16. Configure the r_bsp for your board by going through and modifying the *r_bsp_config.h* file as needed.
17. For RX64M, RX65N and RX71M MCU's configuring the bsp requires that the user also create an *r_bsp_interrupt_config.h* file. Copy the *r_bsp_interrupt_config_reference.h* file from your *board* folder and paste it into the *r_config* folder. Right-click on the file in the *r_config* folder and click Rename. Rename the file to *r_bsp_interrupt_config.h*.



18. Configure the software configurable interrupts for your RX64M/RX65N/RX71M board by going through and modifying the *r_bsp_interrupt_config.h* file as needed.
19. Build the project.

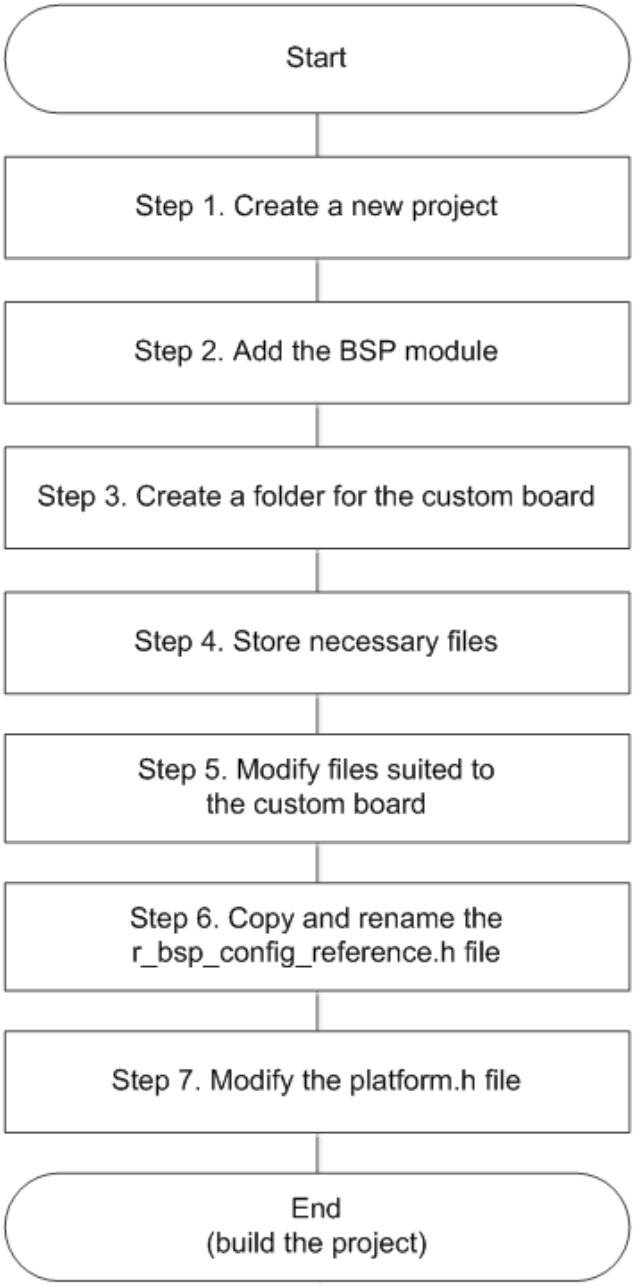
8.1 Creating a BSP Module for a Custom Board

This section describes how to create a custom BSP.

When there is a *generic* folder for the MCU used, create a project selecting the Generic board (refer to the procedure in 7. Project Setup).

When there is no *generic* folder for the MCU used, create a project following the procedure below. This section describes an example procedure using the RX111 MCU.

The figure below shows the procedure for creating a bsp for a custom board.



Step 1. Create a New Project (Mandatory)

To create a new project, refer to "Creating Empty Project" in the "Board Support Package Module Using Firmware Integration Technology" application note (R01AN1685).

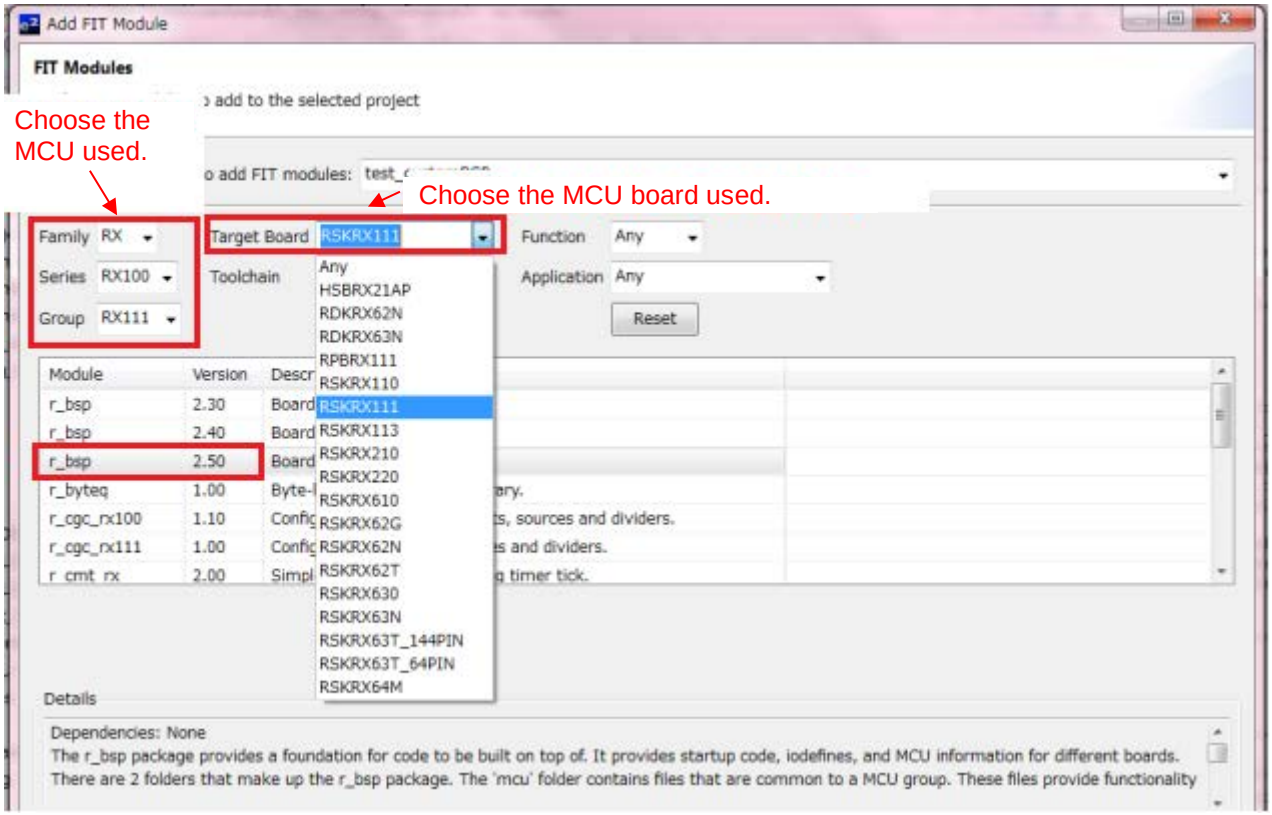
Step 2. Add the BSP Module (Mandatory)

To add the BSP module to the new project (user project) created in step 1, refer to "Adding r_bsp with e^2 studio FIT Plug-in" in the "Board Support Package Module Using Firmware Integration Technology" application note (R01AN1685).

Choose the following options when adding the BSP module on the FIT plug-in.

- Family, Series, Group: MCU used.
- Target Board: MCU board used.

For example, when using the RX111 to create the user board, choose "RSKRX111" or "RSKRX64M". By choosing the appropriate options here, the board folder for the custom board can be created easily.



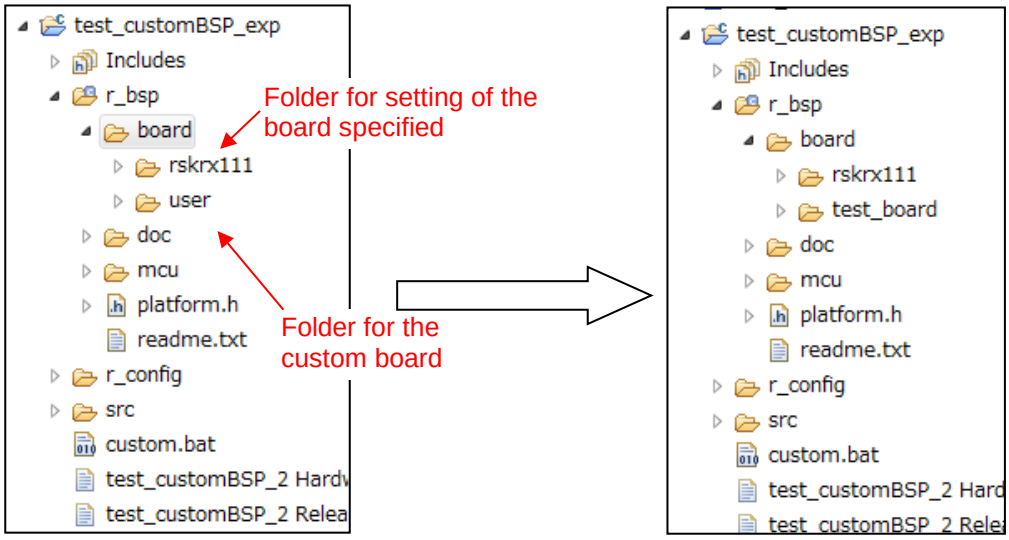
Step 3. Create a Folder for the Custom Board

The r_bsp folder should now be present in the user project. Below, the board folder under the r_bsp folder is modified to create the custom BSP. The code in the mcu folder does not require modification.

1) Confirm that the board folder (rskrx111 here) specified in step 2 and the user folder are generated in the board folder under the r_bsp folder.

2) Use the user folder as the folder for the custom board (optional).

   Rename the folder name (optional). The folder name does not have to be changed.



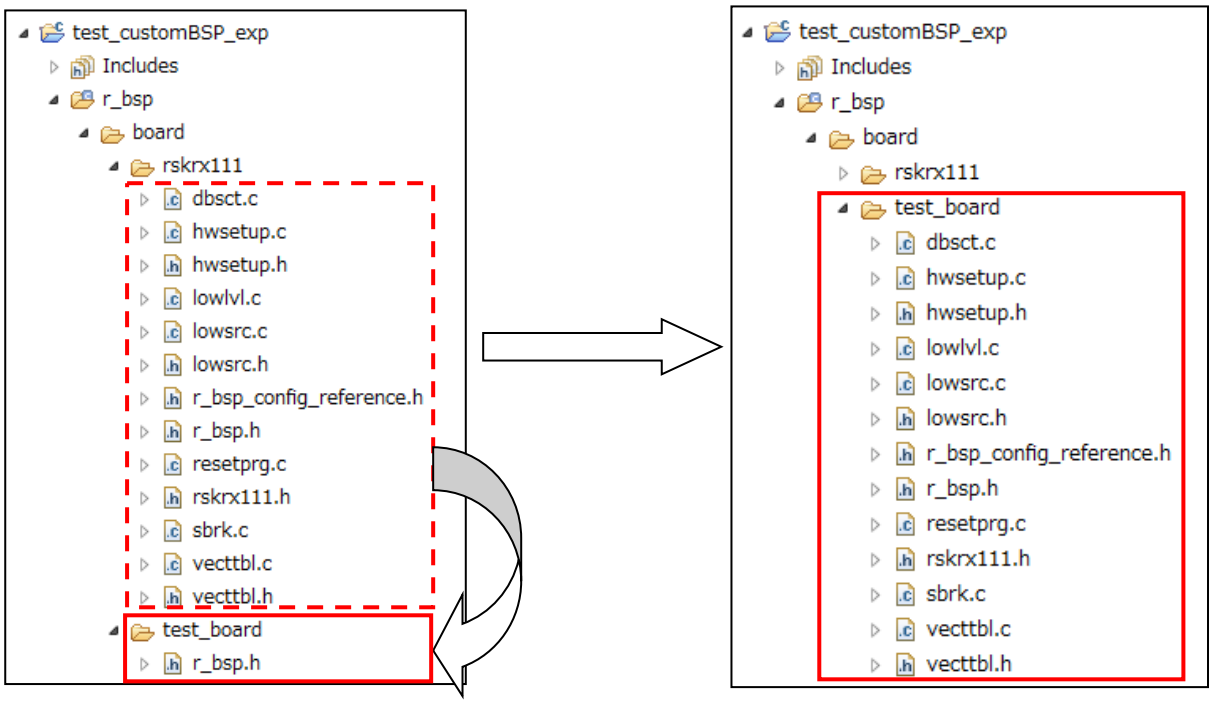
Folder structure after the BSP module is added.

Example when the folder name is changed for the custom board

Step 4. Store Necessary Files (Mandatory)

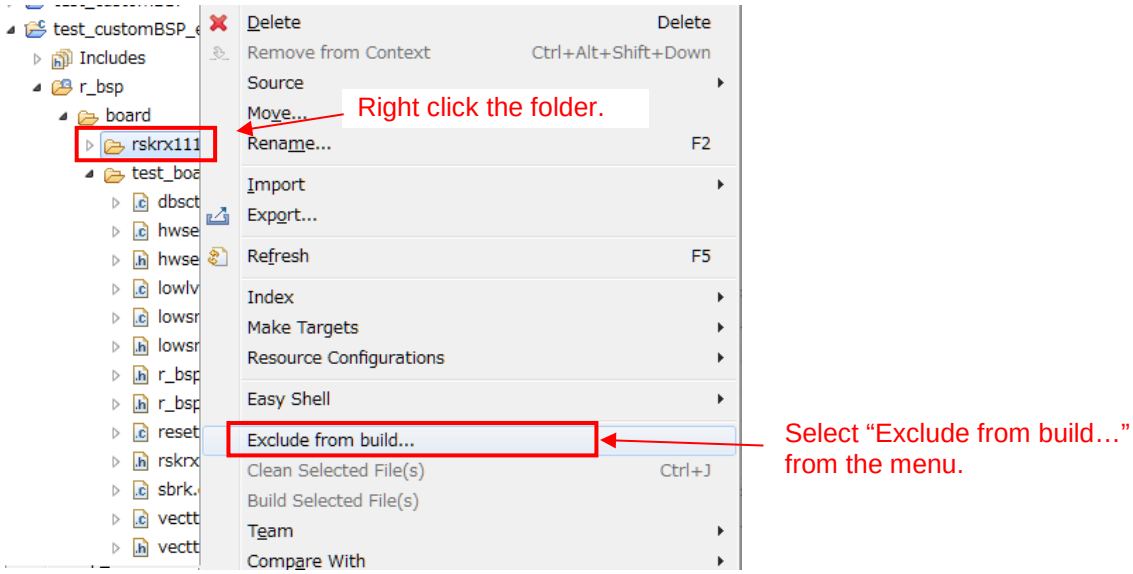
Store necessary files in the folder created in step 3.

1) Copy all files in the rskrx111 folder and paste them in the folder for the custom board. Then overwrite the r_bsp.h file.



2) Exclude the rskrx111 folder from build.

(The folder can be deleted if it is not necessary after the folder for the custom board is created.)



Step 5. Modify Files Suited to the Custom Board (Mandatory)

Modify the following four files suited to the custom board.

1. hwsetup.c

This file executes the following four functions.

- **Function: output_ports_configure**

This function initializes ports used for LEDs, switches, SCI, and ADC.

Ports need to be configured with either of procedures below according to the board used.

If not configuring pins in this function:

1) Comment out or delete the function declaration of the output_ports_configure function.
2) Delete the output_ports_configure function which is called in the hardware_setup function.
3) Comment out or delete the output_ports_configure function.

Then configure settings described in "2. *board_specific_defines*.h" as well.

If configuring pins in this function:

1) Comment out or delete the source code in the output_ports_configure function.
2) Configure pins according to the board used.

- **Function: bsp_non_existent_port_init**

This function initializes nonexistent ports. No additional processing is required for this function.

- **Function: interrupts_configure**

This function configures interrupt settings which are performed prior to the main function.

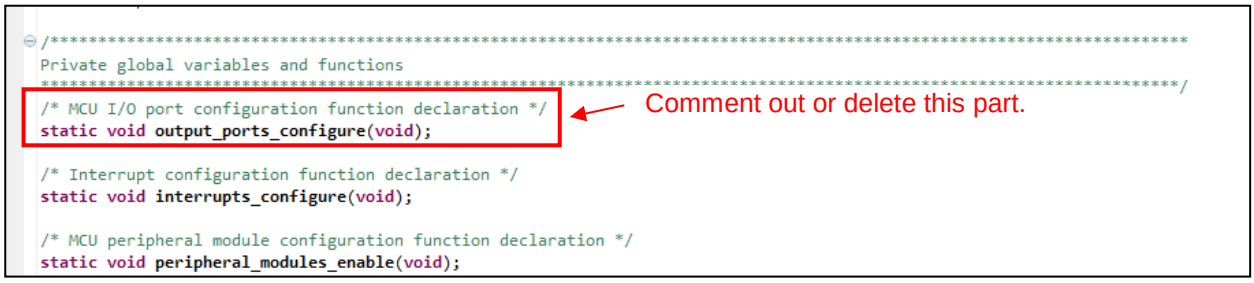
When such settings are required, add the settings in this function.

- **Function: peripheral_modules_enable**

This function configures settings for peripheral functions which are performed prior to the main function.

When such settings are required, add the settings in this function.

Examples of processing are shown below when not configuring pins in the output_ports_configure function.



```
/***********************************************************************************************************************
* Function name: hardware_setup
* Description  : Contains setup functions called at device restart
* Arguments    : none
* Return value : none
***********************************************************************************************************************/
void hardware_setup(void)
{
    output_ports_configure();
    interrupts_configure();
    peripheral_modules_enable();
    bsp_non_existent_port_init();
}
```

Comment out or delete this line.

```
static void output_ports_configure(void)
{
    /* Enable LEDs. */
    /* Start with LEDs off. */
    LED0 = LED_OFF;
    LED1 = LED_OFF;
    LED2 = LED_OFF;
    LED3 = LED_OFF;

    /* Set LED pins as outputs. */
    LED0_PDR = 1;
    LED1_PDR = 1;
    LED2_PDR = 1;
    LED3_PDR = 1;

    /* Enable switches. */
    /* Set pins as inputs. */
    SW1_PDR = 0;
    SW2_PDR = 0;
    SW3_PDR = 0;

    /* Set port mode registers for switches. */
    SW1_PMR = 0;
    SW2_PMR = 0;
    SW3_PMR = 0;

    /* Unlock MPC registers to enable writing to them. */
    R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);

    /* TXD1 is output. */
    PORT1.PMR.BIT.B6 = 0;
    MPC.P16PFS.BYTE  = 0x0A;
    PORT1.PDR.BIT.B6 = 1;
    PORT1.PMR.BIT.B6 = 1;
    /* RXD1 is input. */
    PORT1.PMR.BIT.B5 = 0;
    MPC.P15PFS.BYTE  = 0x0A;
    PORT1.PDR.BIT.B5 = 0;
    PORT1.PMR.BIT.B5 = 1;

    /* Configure the pin connected to the ADC Pot as an analog input */
#if   (BSP_CFG_BOARD_REVISION == 0)
    PORT4.PMR.BIT.B4 = 0;
    MPC.P44PFS.BYTE = 0x80;     //Set ASEL bit and clear the rest
    PORT4.PDR.BIT.B4 = 0;
#elif (BSP_CFG_BOARD_REVISION == 1)
    PORT4.PMR.BIT.B0 = 0;
    MPC.P40PFS.BYTE = 0x80;     //Set ASEL bit and clear the rest
    PORT4.PDR.BIT.B0 = 0;
#endif
```

Comment out or delete this part.

2. *board_specific_defines*.h

The board used becomes the name of this file (e.g. rskrx111.h). This file has definitions of pins used for switches, LEDs, and so on, and their settings vary depending on the board used.

However this file is not necessary when using a custom board. Perform the following steps.

1) Delete the *board_specific_defines*.h file from the folder for the custom board.

2) Delete the following line in the r_bsp.h file.

```
#include     "board/rskrx111/rskrx111.h"
```

3. r_bsp.h

This header file is included in platform.h and has all #includes required for the board and the MCU. The include paths associated with the board need to be modified.

1) Modify the include paths which start with "board/" as follows:

Change the path to "board/*name of the folder for the custom board*/file name".

Example:

Before modification: `#include "board/rskrx111/rskrx111.h"`

After modification: `#include "board/test_board/rskrx111.h"`

```
/***********************************************************************************************************************
INCLUDE APPROPRIATE MCU AND BOARD FILES
***********************************************************************************************************************/
#include    "mcu/all/r_bsp_common.h"
#include    "r_bsp_config.h"
#include    "mcu/rx111/register_access/iodefine.h"
#include    "mcu/rx111/mcu_info.h"
#include    "mcu/rx111/mcu_locks.h"
#include    "mcu/rx111/locking.h"
#include    "mcu/rx111/cpu.h"
#include    "mcu/rx111/mcu_init.h"
#include    "mcu/rx111/mcu_interrupts.h"
#include    "board/test_board/rskrx111.h"
#include    "board/test_board/hwsetup.h"
#include    "board/test_board/lowsrc.h"
#include    "board/test_board/vecttbl.h"

#endif /* BSP_BOARD_RSKRX111 */
```

Change this part to the folder name for the custom board.

4. r_bsp_config_reference.h

This header file has settings to provide default options of the board. Macro definitions that are included in this file and need to be modified according to the custom board are listed in the table below. Change the settings as required.

For example, when the setting in the copied board folder uses the PLL as the system clock while the user system uses the HOCO, change the clock setting for BSP_CFG_CLOCK_SOURCE from PLL to HOCO.

Also confirm usage conditions for macros not in the table below and modify them as required.

Table 8.1 Macros to be modified to reflect the Custom Board

Macro	Description
BSP_CFG_CLOCK_SOURCE	Selects a crystal on the board and a clock source.
BSP_CFG_XTAL_HZ	Specifies a value according to the crystal on the board (default value: RSK setting).
BSP_CFG_PLL_DIV	When using the PLL: Specifies an available setting value using the crystal on the board.
BSP_CFG_PLL_MUL	When using the PLL: Specifies an available setting value using the crystal on the board.
BSP_CFG_ICK_DIV	Specifies an available setting value using the crystal on the board.
BSP_CFG_PCKB_DIV	Specifies an available setting value using the crystal on the board.
BSP_CFG_PCKD_DIV	Specifies an available setting value using the crystal on the board.
BSP_CFG_FCK_DIV	Specifies an available setting value using the crystal on the board.

Step 6. Copy and Rename the r_bsp_config_reference.h File (Mandatory)

After step 5, copy the r_bsp_config_reference.h file, paste it in the r_config folder, and rename the copied file to "r_bsp_config.h".

Step 7. Modify the platform.h File (Mandatory)

This header file needs to be modified to specify the r_bsp.h file in the newly created folder for the custom board. Follow the steps below for the modification.

1) Uncomment the line under the comment "/* User Board - Define your own board here. */ ".

2) Change the folder name after "board/" to the folder name for the custom board.

Before modification:

```
/* User Board - Define your own board here. */
//#include "./board/user/r_bsp.h"
```

After modification:

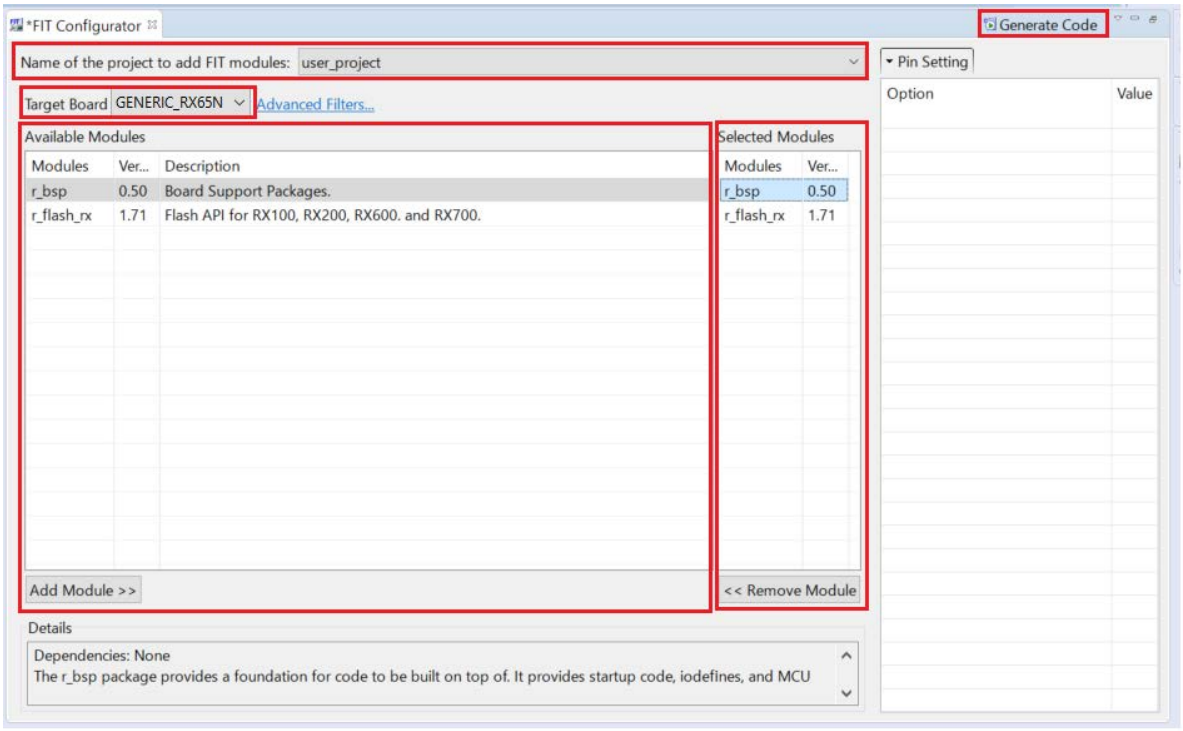
```
/* User Board - Define your own board here. */
#include "./board/test_board/r_bsp.h"
```

9. Adding FIT Modules to the User Project

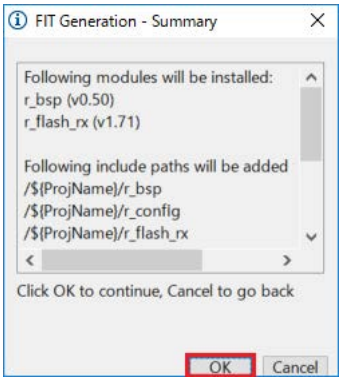
This section describes how to add a FIT module to the user project. The procedure to add the peripheral FIT module to the existing user project without creating a new project is described. The FIT configurator in the e^2 studio is used to add the FIT module.

Step 1. Adding the FIT module using the FIT configurator

1. Click Renesas Views >> e2 solution toolkit >> FIT Configurator to open the FIT configurator.
2. Select the user created project from the list in the ‘Name of the project to add FIT modules’ field.
3. Select a GENERIC board from the list in the ‘Target Board’ field.
4. Select the *r_bsp* and the peripheral FIT module from the ‘Available Modules’ pane and click the Add Module button.
5. Confirm that the *r_bsp* and the peripheral FIT module selected are displayed in the ‘Selected Modules’ pane, and click the Generate Code button.

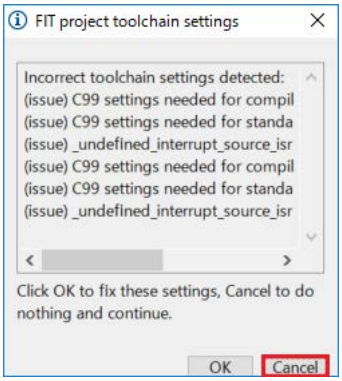


6. Check the contents for code generation in the ‘FIT Generation – Summary’ window and click OK.



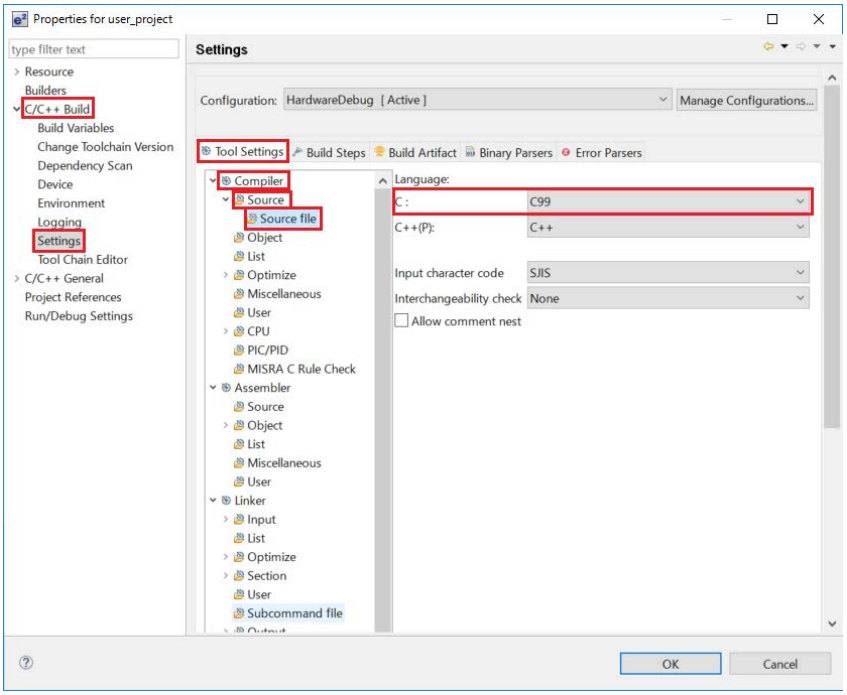
7. When the ‘FIT project toolchain settings’ window appears, click **Cancel**.

This window appears when the necessary settings to use the BSP and FIT modules have not been done. Settings for the compiler option and the Standard library option are described in “Step 2. Setting the Project Environment”.

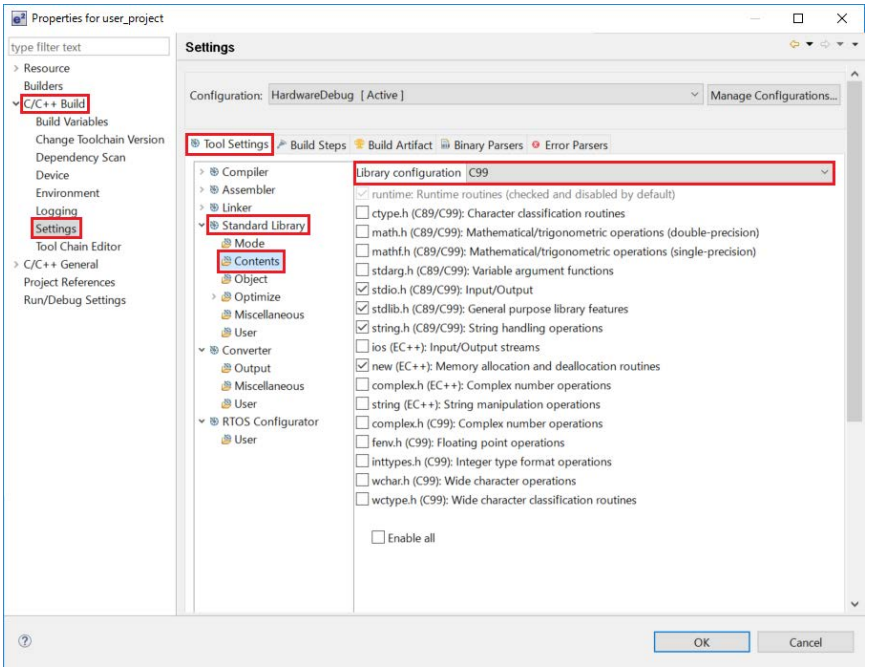


Step 2. Setting the Project Environment

1. Select Renesas Tool Setting (click the e^2 icon on the menu bar) to open the 'Properties for <*project name*>' window and select
   C/C++ Build >> Settings >> Tool Settings (tab) >> Compiler >> Source >> Source file, and then specify 'C99' in the 'C:' field. The FIT module assumes 'C99' to be specified for the C language setting.



2. Select Renesas Tool Setting (click the e^2 icon on the menu bar) to open the 'Properties for <*project name*>' window and select
   C/C++ Build >> Settings >> Tool Settings >> Standard Library >> Contents, and then specify 'C99' in the 'Library configuration' field. The FIT module assumes 'C99' to be specified for the library setting for C language.



3. Specify sections for the FIT module.

When the FIT module project is generated in the e^2 studio, sections for the FIT module will be specified. FIT module assumes these sections are used for the project.

Table 9.1 lists the Sections for the FIT module.

Table 9.1 Sections for the FIT module

Address	Section Name
0x00000004	SU
	SI
	B_1
	R_1
	B_2
	R_2
	B
	R
0xFFxxxxxx *1	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	P*
0xFFFFFF80	EXCEPTVECT / FIXEDVECT *2
0xFFFFFFFC *3	RESETVECT *3

Note 1. The address varies depending on the device selected when generating the project.

Note 2. Section names are different for each CPU. The section names are EXCEPTVECT for RXv2 and RXv3, and FIXEDVECT for RXv1.

Note 3. This is only specified when RXv2 core and RXv3 are selected.

For the device CPU, refer to the Features section in the User's Manual: Hardware.

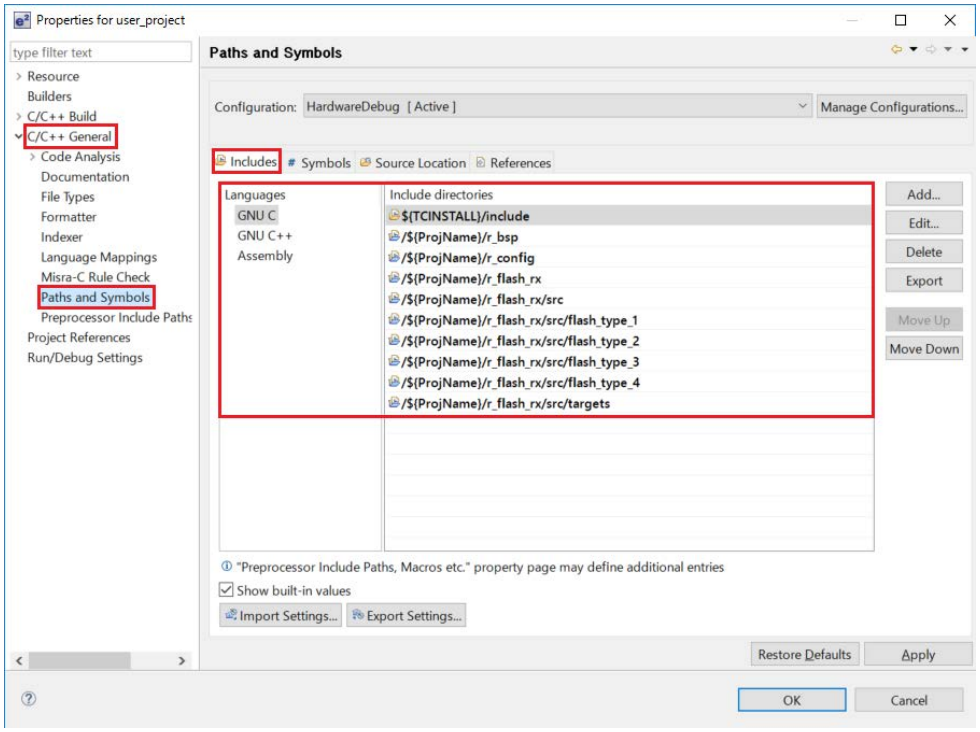
Step 3. Startup disable

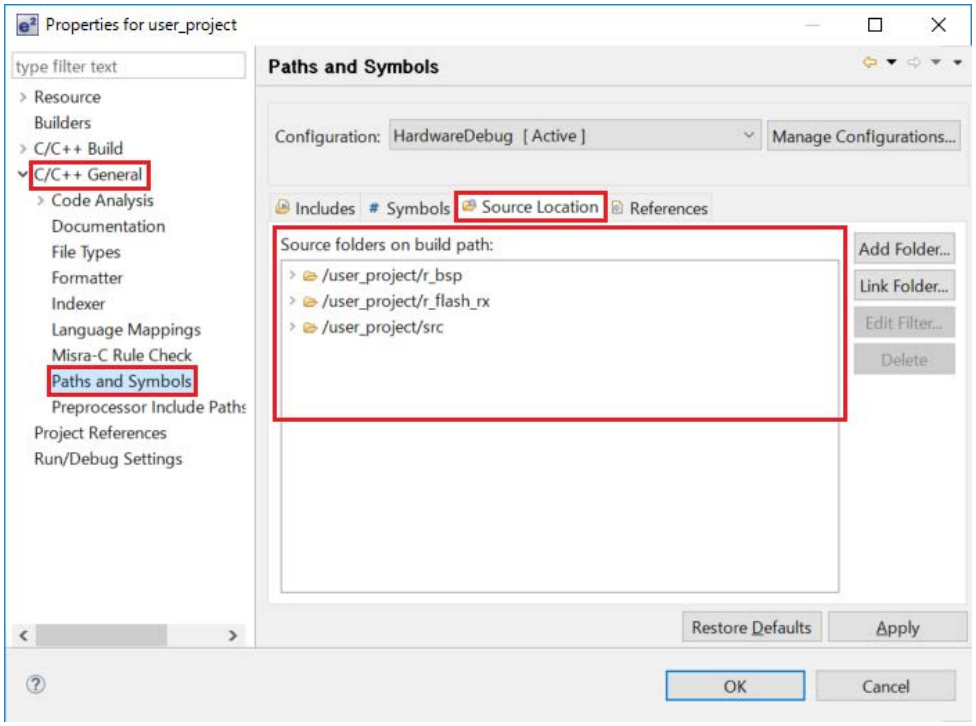
1. Disable the BSP startup. See Section 2.22.1 Setting the Startup Disable Function for details.

Notes

1. When the code is generated with the FIT configurator, include paths necessary for using the FIT module are automatically added.

   To check include paths added, select Renesas Tool Setting (click the e^2 icon on the menu bar) to open the 'Properties for <*project name*>' window and select C/C++ General >> Path and Symbols, and then check the paths in the 'Includes' and the 'Source and Location' tabs.





10. Appendices

10.1 Confirmed Operation Environment

This section describes confirmed operation environment for this module.

Table 10.1 Confirmed Operation Environment (Rev.3.10)

Item	Details
Integrated development environment	Renesas Electronics e^2 studio Version 4.1.0.018
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.03.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev.3.10
Board used	Renesas Starter Kit for RX130 (product No.: RTK5005130SxxxxxBE)

Table 10.2 Confirmed Operation Environment (Rev.3.20)

Item	Details
Integrated development environment	Renesas Electronics e^2 studio Version 4.1.0.018
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.03.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev.3.20
Board used	Renesas Starter Kit for RX24T (product No.: RTK500524TSxxxxxBE)

Table 10.3 Confirmed Operation Environment (Rev.3.30)

Item	Details
Integrated development environment	Renesas Electronics e^2 studio Version 4.2.0.012
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.03.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev.3.30
Board used	Renesas Starter Kit for RX231 (product No.: R0K505231SxxxBE)

Table 10.4 Confirmed Operation Environment (Rev.3.31)

Item	Details
Integrated development environment	Renesas Electronics e^2 studio Version 4.3.0.007
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.03.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev.3.31
Board used	Renesas Starter Kit for RX23T (product No.: RTK500523TSxxxxxBE)

Table 10.5 Confirmed Operation Environment (Rev.3.40)

Item	Details
Integrated development environment	Renesas Electronics e^2 studio Version 5.0.1.005
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.05.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev.3.40
Board used	Renesas Starter Kit+ for RX65N (product No.: RTK500565NSxxxxxBE)

Table 10.6 Confirmed Operation Environment (Rev.3.50)

Item	Details
Integrated development environment	Renesas Electronics e^2 studio Version 5.2.0.020
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.06.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev.3.50
Board used	Renesas Starter Kit for RX24T (product No.: RTK500524TSxxxxxBE) Renesas Starter Kit for RX24U (product No.: RTK500524USxxxxxBE)

Table 10.7 Confirmed Operation Environment (Rev.3.60)

Item	Details
Integrated development environment	Renesas Electronics e² studio Version 5.4.0.015 (RX130) Renesas Electronics e² studio Version 6.0.0.001 (RX65N)
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev3.60
Board used	Renesas Starter Kit for RX130 (product No.: RTK5005130SxxxxxBE) Renesas Starter Kit for RX130-512KB (product No.: RTK5051308SxxxxxBE) Renesas Starter Kit+ for RX65N (product No.: RTK500565NSxxxxxBE) Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2SxxxxxBE)

Table 10.8 Confirmed Operation Environment (Rev.3.70)

Item	Details
Integrated development environment	Renesas Electronics e² studio Version 6.1.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev3.70
Board used	Renesas Starter Kit for RX111 (product No.: R0K505111SxxxBE) Renesas Starter Kit for RX113 (product No.: R0K505113SxxxBE) Renesas Starter Kit for RX130 (product No.: RTK5005130SxxxxxBE) Renesas Starter Kit for RX130-512KB (product No.: RTK5051308SxxxxxBE) Renesas Starter Kit for RX231 (product No.: R0K505231SxxxBE) Renesas Starter Kit for RX23T (product No.: RTK500523TSxxxxxBE) Renesas Starter Kit for RX24T (product No.: RTK500524TSxxxxxBE) Renesas Starter Kit for RX24U (product No.: RTK500524USxxxxxBE) Renesas Starter Kit+ for RX64M (product No.: R0K50564MSxxxBE) Renesas Starter Kit+ for RX71M (product No.: R0K50571MSxxxBE) RX65N Envision Kit (product No.: RTK5RX65N2CxxxxxBR)

Table 10.9 Confirmed Operation Environment (Rev.3.71)

Item	Details
Integrated development environment	Renesas Electronics e² studio Version 6.1.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev3.71
Board used	Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2SxxxxxBE) RX65N Envision Kit (product No.: RTK5RX65N2CxxxxxBR)

Table 10.10 Confirmed Operation Environment (Rev.3.80)

Item	Details
Integrated development environment	Renesas Electronics e^2 studio Version 7.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.08.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev3.80
Board used	Renesas Starter Kit for RX111 (product No.: R0K505111SxxxBE) Renesas Starter Kit for RX113 (product No.: R0K505113SxxxBE) Renesas Starter Kit for RX130 (product No.: RTK5005130SxxxxxBE) Renesas Starter Kit for RX130-512KB (product No.: RTK5051308SxxxxxBE) Renesas Starter Kit for RX210 (B Mask) (product No.: R0K505210SxxxBE) Renesas Starter Kit for RX231 (product No.: R0K505231SxxxBE) Renesas Starter Kit for RX23T (product No.: RTK500523TSxxxxxBE) Renesas Starter Kit for RX63T (64-pin) (product No.: R0K50563TSxxxBE) Renesas Starter Kit for RX63T (144-pin) (product No.: R0K5563THSxxxBE) Renesas Starter Kit+ for RX64M (product No.: R0K50564MSxxxBE) Renesas Starter Kit+ for RX65N (product No.: RTK500565NSxxxxxBE) Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit+ for RX71M (product No.: R0K50571MSxxxBE) Target Board for RX130 (product No.: RTK5RX1300CxxxxxBR) Target Board for RX231 (product No.: RTK5RX2310CxxxxxBR) Target Board for RX65N (product No.: RTK5RX65N0CxxxxxBR) RX65N Envision Kit (product No.: RTK5RX65N2CxxxxxBR)

Table 10.11 Confirmed Operation Environment (Rev.3.90)

Item	Details
Integrated development environment	Renesas Electronics e^2 studio Version 7.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev3.90
Board used	Renesas Starter Kit for RX66T (product No.: RTK50566T0SxxxxxBE)

Table 10.12 Confirmed Operation Environment (Rev.3.91)

Item	Details
Integrated development environment	Renesas Electronics e² studio Version 7.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev3.91
Board used	Renesas Starter Kit for RX66T (product No.: RTK50566T0SxxxxxBE)

Table 10.13 Confirmed Operation Environment (Rev.4.00)

Item	Details
Integrated development environment	Renesas Electronics e² studio Version 7.1.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev4.00

Table 10.14 Confirmed Operation Environment (Rev.4.01)

Item	Details
Integrated development environment	Renesas Electronics e² studio Version 7.2.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev4.01
Board used	Renesas Starter Kit for RX72T (product No.: RTK5572Txxxxxxxxxx)

Table 10.15 Confirmed Operation Environment (Rev.5.00)

Item	Details
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0 IAR Embedded Workbench for Renesas RX 4.11.1 IAR Embedded Workbench for Renesas RX 4.12.1 (RX66T and RX72T only)
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201902 Compiler option: The following option is aded to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.11.1 IAR C/C++ Compiler for Renesas RX version 4.12.1 (RX66t and RX72T only) Compiler option: The default settings of the integrated development environment. (RX110 is excluded)
Endian	Big endian/little endian
Revision of the module	Rev.5.00
Board used	Renesas Starter Kit for RX110 (Part number: R0K505110xxxxxx) Renesas Starter Kit for RX111 (Part number: R0K505111xxxxxx) Renesas Starter Kit for RX113 (Part number: R0K505113xxxxxx) Renesas Starter Kit for RX130-512KB (Part number: RTK505130xxxxxxxxx) Renesas Starter Kit for RX231 (Part number: R0K505231xxxxxx) Renesas Starter Kit for RX23T (Part number: RTK500523Txxxxxxxx) Renesas Starter Kit for RX24T (Part number: RTK500524Txxxxxxxx) Renesas Starter Kit for RX24U (Part number: RTK500524Uxxxxxxxx) Renesas Starter Kit+ for RX63N (Part number: R0K50563Nxxxxxx) Renesas Starter Kit+ for RX64M (Part number: R0K50564Mxxxxxx) Renesas Starter Kit+ for RX65N-2MB (Part number: RTK50565Nxxxxxxxxx) Renesas Starter Kit+ for RX71M (Part number: R0K50571Mxxxxxx) Renesas Starter Kit for RX66T (Part number: RTK50566Txxxxxxxxx) Renesas Starter Kit for RX72T (Part number: RTK5572Txxxxxxxxxx)

Table 10.16 Confirmed Operation Environment (Rev.5.10)

Item	Details
Integrated development environment	Renesas Electronics e^2 studio Version 7.1.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev5.10
Board used	Renesas Solution Starter Kit for RX23W (product No.: RTK5523Wxxxxxxxxx)

Table 10.17 Confirmed Operation Environment (Rev.5.20)

Item	Details
Integrated development environment	Renesas Electronics e^2 studio Version 7.4.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment. (R_BSP_CHG_PMUSR function and R_BSP_ChangeToUserMode function are excluded.)
Endian	Big endian/little endian
Revision of the module	Rev.5.20
Board used	Renesas Starter Kit+ for RX72M (Part number: RTK5572Mxxxxxxxxxx)

Table 10.18 Confirmed Operation Environment (Rev.5.21)

Item	Details
Integrated development environment	Renesas Electronics e^2 studio Version 7.5.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment. (R_BSP_CHG_PMUSR function and R_BSP_ChangeToUserMode function are excluded.)
Endian	Big endian/little endian
Revision of the module	Rev.5.21

10.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the following error: Could not open source file “platform.h”.

A: The FIT module may not be added to the project properly. Using the following documents, check if the method for adding FIT modules is correct with the following documents:

- When using CS+:

  Application note “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)”
- When using e^2 studio:

  Application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)”

When using a FIT module, the board support package FIT module (BSP module) must also be added to the project. For this, refer to the application note “Board Support Package Module Using Firmware Integration Technology (R01AN1685)”.

(2) Q: I have added the FIT module to the project and built it. Then I got the following error: ERROR - Valid clock source must be chosen in r_bsp_config.h using BSP_CFG_CLOCK_SOURCE macro.

A: The setting in the file “r_bsp_config.h” may be wrong. Check the file “r_bsp_config.h”. If there is a wrong setting, set the correct value for that. Refer to 3 Configuration.

Technical Update Information

The following technical update applies to this module.

- TN-RX*-A021A
- TN-RX*-A138A
- TN-RX*-A153A
- TN-RX*-A164A
- TN-RX*-A169A

Revision Record

Rev.	Date	Description	
		Page	Summary
2.30	Nov.15.13	—	First Release.
2.40	Feb.18.14	—	Added support for RX21A, RX220, RX110. Expanded the 'MCU Information' subsection.
2.50	Mar.13.14	—	Added support for RX64M.
2.60	Jul.15.14	—	Added section for Creating a BSP Module for a Custom Board.
2.70	Aug.05.14	—	Added support for RX113.
2.80	Jan.21.15	—	Added support for RX71M.
2.81	Mar.31.15	—	Supported 240 MHz of the operating frequency (default) for RX71M.
2.90	Jun.30.15	—	Added support for RX231.
3.00	Sep.30.15	—	Added support for RX23T.
3.01	Sep.30.15	Program	Modified BSP FIT module due to the software issues. Modification Regarding Clocks [Description] For processing to switch a clock immediately after a reset, there is an error in determination of the condition in processing for switching to high-speed operating mode when exceeding the allowable frequency range of middle-speed operating mode. This may cause middle-speed operating mode to be set with a frequency out of the allowable frequency range. [Conditions] When the following three conditions are all met: - RX231 or RX23T is used with the BSP FIT module rev. 3.00 or earlier. - The initial definition of the highest clock frequency is as follows: 12 MHz < the highest clock frequency ≤ 32 MHz (RX231). - The initial definition of the ICLK is as follows: 12 MHz < ICLK ≤ 32 MHz (RX23T). [Workaround] Use rev. 3.01 or a later version of the BSP FIT module.
			Modification Regarding Stacks [Description] The large stack size defined by the BSP FIT module may cause a lack of the RAM area used for other than stack or heap. [Conditions] When the following two conditions are met: - RX23T is used with the BSP FIT module rev. 3.00. - BSP_CFG_USER_STACK_ENABLE = 1 [Workaround] Use rev. 3.01 or a later version of the BSP FIT module.

Rev.	Date	Description	
		Page	Summary
3.01	Sep.30.15	Program	Modified the BSP FIT module due to software issues. Modification Regarding Locks [Description] For the lock function, predefined indexes according to hardware functions do not exactly correspond to actual hardware functions supported. Thus the lock function may not be used for some hardware functions. [Conditions] When the following three conditions are all met: - RX231 or RX23T is used with the BSP FIT module rev. 3.00 or earlier. - The function R_BSP_HardwareLock or R_BSP_HardwareUnlock is used. - BSP_CFG_USER_LOCKING_ENABLED = 0 [Workaround] Use rev. 3.01 or a later version of the BSP FIT module. This modification includes the following changes in definitions. - Definitions added (RX23T) BSP_LOCK_CMPC0, CMPC1, CMPC2, BSP_LOCK_SMCI1, SMCI5 - Definitions added (RX231) BSP_LOCK_CMPB0, CMPB1, CMPB2, CMPB3, BSP_LOCK_LPT - Definitions deleted (RX231) BSP_LOCK_CMPB, BSP_LOCK_SMCI2, SMCI3, SMCI4, SMCI7, SMCI10, SMCI11
3.10	Dec.01.15	—	Added support for RX130.
		1, 6, 8	Modified descriptions in the following sections: Target Device, 2.6 Clock Setup, 2.14 Trusted Memory
		62	Added the following section: Technical Update Information
3.20	Feb.01.16	—	Added support for RX24T.
		13, 14	Added the following macro definitions in section 3.2.6 Clock Setup: - BSP_CFG_MAIN_CLOCK_SOURCE - BSP_CFG_MOSC_WAIT_TIME - BSP_CFG_ROM_CACHE_ENABLE
		Program	Modified the PCLKA to satisfy the clock restriction (ICLK=PCLKA) of Ethernet Controller (ETHERC). (RX63N)
3.30	Feb.29.16	—	Added support for RX230.
		—	Update RX113 iodefine.h to V1.0A.
		43	5.15 R_BSP_SoftwareDelay, Description changed

Rev.	Date	Description	
		Page	Summary
3.30	Feb.29.16	Program	Modified the BSP FIT module. Modification Regarding API Functions [Description] Since subtraction of overhead in the R_BSP_SoftwareDelay function is more than necessary, it may not be possible to secure the specified duration. [Workaround] Change the following definition (the overhead cycles). - OVERHEAD_CYCLES - OVERHEAD_CYCLES_64 [Note] This modification is compared with the BSP FIT module Rev.3.20 or earlier, the processing time of R_BSP_SoftwareDelay function is longer.
3.31	May.19.16	—	Updated RX230 and RX231 iodefine.h to V1.0F.
		—	Changed RX23T iodefine.h to V1.1.
		—	Changed RX24T iodefine.h to V1.0A.
		—	Changed RX64M iodefine.h to V1.0.
		14	3.2.6 Clock Setup Amended the following macro definition: - BSP_CFG_MOSC_WAIT_TIME Added the following macro definitions: - BSP_CFG_HOCO_WAIT_TIME - BSP_CFG_SOSC_WAIT_TIME
		Program	Modification Regarding Memory Changed the setting values in the following macro definition to match the increased RAM capacity (RX23T): - BSP_RAM_SIZE_BYTES
			Modification Regarding Clocks The following items are now supported. Made partial changes to the program code (RX23T, RX64M, and RX71M). [Description] • Added HOCO as a selectable clock source for the system clock (RX23T only). • The oscillation source of the main clock oscillator is selectable. • The wait time of the main clock oscillator is selectable. The oscillation source of the sub-clock oscillator is selectable (RX64M and RX71M only). [Note] With these changes, the wait time default values for the main clock oscillator and sub-clock oscillator of the RX64M and RX71M are set to the values after a reset listed in the user's manual. Note that the new default values differ from the BSP FIT module default values listed in rev. 3.30 and earlier.

Rev.	Date	Description	
		Page	Summary
3.31	May.19.16	Program	Modification Regarding Clocks Amended the HOCO oscillation setting because the setting value was not appropriate when HOCO oscillation is enabled in option function select register 1 (OFS1.HOCOEN = 1) (RX64M and RX71M). [Description] • Made changes so that when HOCO oscillation is enabled in option function select register 1 (OFS1.HOCOEN = 1) and HOCO is selected as the clock source of the system clock, HOCO oscillation does not stop. • Set the HOCO power supply to OFF when HOCO oscillation is disabled in option function select register 1 (OFS1.HOCOEN = 0) and HOCO is not selected as the clock source of the system clock.
			Modification Regarding interrupts Made changes to the bsp_interrupt_group_enable_disable function in the program code to conform to the IPR setting procedure in the user's manual (RX64M and RX71M). [Description] Changed the program code so that writing to the IPRr register occurs when the value of the corresponding IERm.IENj bit is 0.
			Modification Regarding STDIO & Debug console Improved the following (RX23T, RX64M, and RX71M). [Description] The module did not operate properly when BSP_CFG_USER_CHARGET_ENABLED or BSP_CFG_USER_CHARPUT_ENABLED was set to "enabled" (1), so the program code was modified to ensure proper operation.
			Modification Regarding API Functions Deleted unnecessary enumerated constants from the R_BSP_RegisterProtectEnable and R_BSP_RegisterProtectDisable functions, and added HOCO enumerated constant (RX23T). [Description] Deleted the BSP_REG_PROTECT_VRCR constant of the bsp_reg_protect_t enumerated argument of the R_BSP_RegisterProtectEnable and R_BSP_RegisterProtectDisable functions. Added BSP_REG_PROTECT_HOCOWTCR.
3.40	Oct.01.16	—	Added support for RX65N.
		17	3.2.7 Registers in ROM & External Memory Access Protection Added the following macro definitions: • BSP_CFG_FAW_REG_VALUE • BSP_CFG_ROMCODE_REG_VALUE

Rev.	Date	Description	
		Page	Summary
3.40	Oct.01.16	Program	Modification Regarding Clocks (1) Changed the default value of the following definition in *r_bsp_config_reference.h*, because it becomes the cause of compile errors in the LPT module (RX130). • BSP_CFG_LPT_CLOCK_SOURCE Changed the default value from 2 to 0. (2) Fixed the error of the following definitions in *mcu_info.h* (RX230, RX231). Case of the “BSP_CFG_LPT_CLOCK_SOURCE = 1”. • BSP_LPTSRCCLK_HZ Changed the default value from "15360" to "15000". Case of the “BSP_CFG_LPT_CLOCK_SOURCE = 2”. • Deleted the definition. (3) Added the following macro definition in *mcu_info.h* (RX130). • BSP_LPTSRCCLK_HZ
3.50	Mar.15.17	—	Added support for RX24U
		—	Changed RX24T iodefine.h to V1.0H.
		15	3.2.6 Clock Setup Amended the following macro definition contests: - BSP_CFG_USE_CGC_MODULE
		16	3.2.7 Registers in ROM & External Memory Access Protection Amended the following macro definition contests: - BSP_CFG_OFS1_REG_VALUE
		19	4.5 Supported Toolchains Amended the contents
		62	Added 8, Appendix
		Program	Modification Regarding Memory Changed the setting values in the following macro definition to match the increased ROM and RAM capacity (RX24T): • BSP_ROM_SIZE_BYTES • BSP_RAM_SIZE_BYTES
			Modification Regarding Package Added the following macro definition to match the increased 64 Pin Packages (RX24T): • BSP_PACKAGE_LFQFP64 • BSP_PACKAGE_PINS
			Modification Regarding Clocks The following items are now supported. Made partial changes to the program code (RX24T). [Description] - Added HOCO as a selectable clock source for the system clock. - Added HOCO as a selectable input clock source for the PLL circuit.

Rev.	Date	Description	
		Page	Summary
3.50	Mar.15.17	Program	Modification Regarding STDIO & Debug console Improved the following (RX24T). [Description] The module did not operate properly when BSP_CFG_USER_CHARGET_ENABLED or BSP_CFG_USER_CHARPUT_ENABLED was set to “enabled” (1), so the program code was modified to ensure proper operation.
			Modification Regarding API Functions Deleted unnecessary enumerated constants from the R_BSP_RegisterProtectEnable and R_BSP_RegisterProtectDisable functions, and added HOCO enumerated constant (RX24T). [Description] Deleted the BSP_REG_PROTECT_VRCR constant of the bsp_reg_protect_t enumerated argument of the R_BSP_RegisterProtectEnable and R_BSP_RegisterProtectDisable functions. Added BSP_REG_PROTECT_HOCOWTCR.
3.60	May.15.17	—	Added support for RX130-512KB. Added support for RX65N-2MB. Added support for GENERIC-RX65N.
		—	Updated RX110 iodefine.h to V1.0B. Updated RX111 iodefine.h to V1.1A. Updated RX113 iodefine.h to V1.0C. Updated RX130 iodefine.h to V2.0. Updated RX210 iodefine.h to V1.5. Updated RX21A iodefine.h to V1.1C. Updated RX220 iodefine.h to V1.1A. Updated RX230 iodefine.h to V1.0I. Updated RX231 iodefine.h to V1.0I. Updated RX23T iodefine.h to V1.1C. Updated RX62N iodefine.h to V1.4. Updated RX62T iodefine.h to V2.0. Updated RX62G iodefine.h to V2.0. Updated RX630 iodefine.h to V1.6A. Updated RX63N/RX631 iodefine.h to V1.8A. Updated RX63T iodefine.h to V2.1C. Updated RX64M iodefine.h to V1.0A. Updated RX65N iodefine.h to V2.0. Updated RX71M iodefine.h to V1.0A.
		—	Applied the following technical update: - TN-RX*-A138A - TN-RX*-A164A - TN-RX*-A169A
		4	Modified the description in 1.2 File Structure.
			Revised Figure 1-1: r_bsp File Structure.

Rev.	Date	Description	
		Page	Summary
3.60	May.15.17	5	Added Figure 1-2: Structures of Evaluation Board Folder and generic Folder.
		7	Modified the description in 2.2 Initialization. Revised Figure 2-1: PowerON_Reset_PC() Flowchart.
		8	Added Figure 2-2: Flowchart of System Clock Setting
		11	Modified the descriptions in 2.14 Trusted Memory. Added 2.15 Bank Mode. Moved the section “Group Interrupts” from 4.10.7 to 2.20 and modified the description. Moved the section “Software Configurable Interrupts” from 4.10.8 to 2.21.
		13	Added 2.23 Startup Disable.
		19	Modified the descriptions in 3.2.4 CPU Modes & Boot Modes including Table 3-4.
		20	Corrected and added the following definitions in 3.2.6 Clock Setup: - Addition: BSP_CFG_RTC_ENABLE - Addition: BSP_CFG_SOSC_DRV_CAP - Correction: BSP_CFG_PLL_SOURCE -> BSP_CFG_PLL_SCR
		22-23	Modified and added the following definitions in 3.2.7 Registers in ROM & External Memory Access Protection: - Modification: BSP_CFG_OFS1_REG_VALUE - Addition: BSP_CFG_CODE_FLASH_BANK_MODE - Addition: BSP_CFG_CODE_FLASH_START_BANK
		24	Added 3.2.11 Startup Disable.
		26	Added 4.9.3 Interrupt Control Parameter.
		28	Modified 4.10.3 Interrupt Error Codes. Modified 4.10.4 Interrupt Control Commands.
		29	Added 4.10.7 Unit for Software Delay. Changed 4.12 Adding Driver to Your Project.
		30	Added 4.13 Code size.
		31	Modified 5. API Functions.
		50	Added 5.15 R_BSP_InterruptControl().
		52	Added 5.17 R_BSP_GetIClkFreqHz().
		53	Added 5.18 R_BSP_StartupOpen().
		55	Added 6.1 Creating a FIT Project.
		59	Added 6.2 Adding FIT Module with e^2 studio FIT Configurator.
		72	Added 8. Adding FIT Modules to the User Project.
		79	Added Table 9.7 Operation Confirmation Environment (Rev.3.60). Added 9.2 Creating a Project with FIT Plug-in. Moved the section “Creating an Empty Project” from 6.1 to 9.2.1 and modified some descriptions.
		85	Moved the section “Adding r_bsp with e^2 studio FIT Plug-in” from 6.2 to 9.2.2.
		88	Added 9.3 Troubleshooting.

Rev.	Date	Description	
		Page	Summary
3.60	May.15.17	Program	Changes associated with functions: Deleted unnecessary transition to User Mode. (RX130) - Description: Deleted the following function. PSW_PM_to_UserMode function. Added the startup disable function. (RX65N) - Description: Added the macro definition, BSP_CFG_STARTUP_DISABLE. Added the bank function. (RX65N) - Description: Added processing for setting the bank function in vecttbl.c. If a package with the ROM of 1 Mbytes or less is selected, this processing will be disabled. Modified the procedure for initializing the ADSAM register. (RX65N) - Description: Modified the procedure to hold the setting of module-stop state before the ADSAM register is initialized so that the setting can be restored after the initialization.
			Changes associated with packages: Added new package specifications. (RX130) [Description] (1) Added the following macro definitions for new packages. - BSP_MCU_RX130_512KB - BSP_PACKAGE_LFQFP100 (2) Added setting values of the following macro definitions regarding new packages. - BSP_CFG_MCU_PART_PACKAGE: Values: FP = 0x5 = LFQFP/100/0.50 - BSP_CFG_MCU_PART_MEMORY_SIZE: Values: 6 = 0x6 = 128KB/32KB/8KB 7 = 0x7 = 384KB/48KB/8KB 8 = 0x8 = 512KB/48KB/8KB Added new package specifications. (RX65N) [Description] (1) Added the following macro definitions for new packages. - BSP_CFG_CODE_FLASH_BANK_MODE - BSP_CFG_CODE_FLASH_START_BANK - BSP_MCU_RX65N_2MB - BSP_PACKAGE_LFQFP176 - BSP_PACKAGE_LFBGA176 - BSP_PACKAGE_TFLGA177 - BSP_PRV_PORTG_NE_PIN_MASK

Rev.	Date	Description	
		Page	Summary
3.60	May.15.17	Program	(2) Added setting values of the following macro definitions regarding new packages. - BSP_CFG_MCU_PART_PACKAGE: Values: FC = 0x0 = LFQFP/176/0.50 BG = 0x1 = LFBGA/176/0.80 LC = 0x2 = TFLGA/177/0.50 - BSP_CFG_MCU_PART_ENCRYPTION_INCLUDED: Values: D = false =Encryption module not included, SDHI/SDSI module included, dual-bank structure H = true = Encryption module included, SDHI/SDSI module included, dual-bank structure. - BSP_CFG_MCU_PART_MEMORY_SIZE: Values: C = 0xC = 1.5MB/640KB/32KB E = 0xE = 2MB/640KB/32KB Changed the macro definitions for the RX231 package. (1) Added the following macro definition. - BSP_PACKAGE_WFLGA64 (2) Deleted the following macro definition. - BSP_PACKAGE_LQFP64 (3) Added setting values of the following macro definitions: - BSP_CFG_MCU_PART_PACKAGE: Values: LF = 0x1 = WFLGA/64/0.50 - BSP_CFG_MCU_PART_VERSION: Values: C = 0xC = Chip version C = Security function not included, SDHI module not included, CAN module not included. (4) Deleted the setting values for the following macro definition: - BSP_CFG_MCU_PART_PACKAGE: Values: FK = 0x3 = LQFP/64/0.80 LJ = 0xA = TFLGA//100/0.65 - BSP_CFG_MCU_PART_MEMORY_SIZE: Values: 3 = 0x3 = 64KB/12KB/8KB (5) Changed the setting values for the following macro definitions: - BSP_CFG_MCU_PART_MEMORY_SIZE: Value: 5 = 0x5 = 128KB/20KB/8KB -> 5 = 0x5 = 128KB/32KB/8KB

Rev.	Date	Description	
		Page	Summary
3.60	May.15.17	Program	Changed the macro definition for the RX63N/RX631 package. [Description] (1) Changed the default value of the following macro definition: - BSP_CFG_MCU_PART_MEMORY_SIZE: (RSK only) Value: (0xB) -> (0xF) - BSP_CFG_MCU_PART_GROUP: (RX631 only) Value: (0x2) -> (0x1) (2) Added the following macro definition. - BSP_PACKAGE_TFLGA64 (3) Deleted the following macro definition. - BSP_PACKAGE_LQFP80 (4) Added setting values of the following macro definitions: - BSP_CFG_MCU_PART_PACKAGE: Values: LJ = 0xA = TFLGA/100/0.65 LH = 0xB = TFLGA/64/0.65 - BSP_CFG_MCU_PART_ENCRYPTION_INCLUDED: Values: H = true = CAN included/DEU included/PDC not included. G = false = CAN not included/DEU included/PDC not included. S = true = CAN included/DEU not included/PDC included. F (only 64-pin TFLGA) = true = CAN included/DEU not included/PDC not included. - BSP_CFG_MCU_PART_MEMORY_SIZE: Values: F = 0xF = 2MB/256KB/32KB G = 0x10 = 1.5MB/192KB/32KB J = 0x13 = 1.5MB/256KB/32KB K = 0x14 = 2MB/192KB/32KB M = 0x16 = 256KB/64KB/32KB N = 0x17 = 384KB/64KB/32KB P = 0x19 = 512KB/64KB/32KB W = 0x20 = 1MB/192KB/32KB Y = 0x22 = 1MB/256KB/32KB (5) Deleted the setting values for the following macro definition: - BSP_CFG_MCU_PART_PACKAGE: Values: LA = 0x6 = TFLGA/100/0.65 FN = 0x7 = LQFP/80/0.50 - BSP_CFG_MCU_PART_CAN_INCLUDED: Values: E = = 3V included (RX63T). Ignore. - BSP_CFG_MCU_PART_MEMORY_SIZE: Values: 4 = 0x4 = 32KB/8KB/8KB 5 = 0x5 = 48KB/8KB/8KB

Rev.	Date	Description	
		Page	Summary
3.60	May.15.17	Program	(6) Changed the setting values for the following macro definitions: - BSP_CFG_MCU_PART_MEMORY_SIZE: Value: 6 = 0x6 = 64KB/8KB/8KB -> 6 = 0x6 = 256KB/128KB/32KB Value: 7 = 0x7 = 384KB/64KB/32KB -> 7 = 0x7 = 384KB/128KB/32KB Value: 8 = 0x8 = 512KB/64KB/32KB -> 8 = 0x8 = 512KB/128KB/32KB Changed the macro definitions for the RX64M package. [Description] (1) Corrected typo for the following macro definitions: - BSP_PACKAGE_LQFP176 -> BSP_PACKAGE_LFQFP176 - BSP_PACKAGE_LQFP144 -> BSP_PACKAGE_LFQFP144 - BSP_PACKAGE_LQFP100 -> BSP_PACKAGE_LFQFP100 (2) Added the setting value for the following macro definition: - BSP_CFG_MCU_PART_PACKAGE: Value: LJ = 0xA = TFLGA/100/0.65 (3) Deleted the setting values for the following macro definition: - BSP_CFG_MCU_PART_PACKAGE: Values: LA = 0x6 = TFLGA/100/0.50 JA = 0x7 = TFLGA/100/0.65 Changed the macro definition for the RX65N package. [Description] (1) Corrected typo for the following macro definitions: - BSP_PACKAGE_LQFP144 -> BSP_PACKAGE_LFQFP144 - BSP_PACKAGE_LQFP100 -> BSP_PACKAGE_LFQFP100 (2) Changed the setting values for the following macro definitions: - BSP_CFG_MCU_PART_PACKAGE: Value: LJ = 0x6 = TFLGA/100/0.65 -> LJ = 0xA = TFLGA/100/0.65 - BSP_CFG_MCU_PART_GROUP: Value: 5N = 0x0 = RX65N Group -> 5N/51 = 0x0 = RX65N Group/RX651 Group (3) Deleted the setting value for the following macro definition: - BSP_CFG_MCU_PART_GROUP: Value: 51 = 0x1 = RX65N Group

Rev.	Date	Description	
		Page	Summary
3.60	May.15.17	Program	Changed the macro definition for the RX71M package. [Description] (1) Corrected typo for the following macro definitions: - BSP_PACKAGE_LQFP176 -> BSP_PACKAGE_LFQFP176 - BSP_PACKAGE_LQFP144 -> BSP_PACKAGE_LFQFP144 - BSP_PACKAGE_LQFP100 -> BSP_PACKAGE_LFQFP100 (2) Changed the setting value for the following macro definition: - BSP_CFG_MCU_PART_PACKAGE: Value: LJ = 0x6 = TFLGA/100/0.65 -> LJ = 0xA = TFLGA/100/0.65
			<u>Changes associated with clocks:</u> The following items are now supported. Made partial changes to the program code. (RX130) [Description] (1) Added definition of the following clock setting: - The oscillation source of the main clock oscillator is selectable. - The wait time of the main clock oscillator is selectable. (2) Added the following macro definition in mcu_info.h. - BSP_ILOCO_HZ Made add to the lpt_clock_source_select function in the program code to conform to the notes on LPT in the user's manual. (RX130) [Description] When the IWDT-dedicated on-chip oscillator is used as the clock source for the low-power timer, changed the program code to write the IWDTCSTPR.SLCSTP bit to 0. Improved the following. (RX130) [Description] (1) When restarting the sub-clock oscillator after it has been stopped, allow at least five cycles of the sub-clock as an interval over which it is still stopped. Deleted the following unnecessary branch condition in the lpt_clock_source_select function. - BSP_CFG_LPT_CLOCK_SOURCE == 2

Rev.	Date	Description	
		Page	Summary
3.60	May.15.17	Program	Modified the sub-clock oscillation settings. (RX64M, RX65N, RX71M) [Description] (1) Modified for the sub-clock oscillation settings to be specified according to the settings in r_bsp_config.h. (2) Added processing at warm start. (3) Added the following macro definitions regarding changes in the sub-clock oscillation settings. - BSP_CFG_RTC_ENABLE - BSP_CFG_SOSC_DRV_CAP
			Changes associated with interrupts: Made add to the bsp_interrupt_enable_disable function in the program code. (RX130) [Description] Added timeout detection enable bit (BSC.BEREN.BIT.TOEN) settings. Modified the following items for software configurable interrupts. [Description] (1) Corrected the macro definitions due to typos in names of the software configurable interrupt sources. (RX64M, RX65N, RX71M) The corrected interrupt sources are as follows: - TPU0_TGI0V -> TPU0_TCI0V - TPU1_TGI1V -> TPU1_TCI1V - TPU1_TGI1U -> TPU1_TCI1U - TPU2_TGI2V -> TPU2_TCI2V - TPU2_TGI2U -> TPU2_TCI2U - TPU3_TGI3V -> TPU3_TCI3V - TPU4_TGI4V -> TPU4_TCI4V - TPU4_TGI4U -> TPU4_TCI4U - TPU5_TGI5V -> TPU5_TCI5V - TPU5_TGI5U -> TPU5_TCI5U - MTU0_TGIV0 -> MTU0_TCIV0 - MTU1_TGIV1 -> MTU1_TCIV1 - MTU1_TGIU1 -> MTU1_TCIU1 - MTU2_TGIV2 -> MTU2_TCIV2 - MTU2_TGIU2 -> MTU2_TCIU2 - MTU3_TGIV3 -> MTU3_TCIV3 - MTU4_TGIV4 -> MTU4_TCIV4 - MTU6_TGIV6 -> MTU6_TCIV6 - MTU7_TGIV7 -> MTU7_TCIV7 - MTU8_TGIV8 -> MTU8_TCIV8

Rev.	Date	Description	
		Page	Summary
3.60	May.15.17	Program	(2) Deleted the macro definition, MTU8_TGI8U since the corresponding interrupt source does not exist. (RX64M, RX71M) (3) Added interrupt sources for software configurable interrupts regarding new packages. (RX65N) - BSP_MAPPED_INT_CFG_B_VECT_TSIP_PROC_BUSY - BSP_MAPPED_INT_CFG_B_VECT_TSIP_ROMOK - BSP_MAPPED_INT_CFG_B_VECT_TSIP_LONG_PLG - BSP_MAPPED_INT_CFG_B_VECT_TSIP_TEST_BUSY - BSP_MAPPED_INT_CFG_B_VECT_TSIP_WRRDY0 - BSP_MAPPED_INT_CFG_B_VECT_TSIP_WRRDY1 - BSP_MAPPED_INT_CFG_B_VECT_TSIP_WRRDY4 - BSP_MAPPED_INT_CFG_B_VECT_TSIP_RDRDY0 - BSP_MAPPED_INT_CFG_B_VECT_TSIP_RDRDY1 - BSP_MAPPED_INT_CFG_B_VECT_TSIP_INTEGRATE_WRRDY - BSP_MAPPED_INT_CFG_B_VECT_TSIP_INTEGRATE_RDRDY Modified the following items for group interrupts. [Description] (1) Modified the sequence to call callback functions for group interrupts. (RX64M, RX65N, RX71M) Peripherals influenced by the sequence change are as follows: - SCI0 to SCI7, SCI12 - SCI8 to SCI11 (RX65N only) - PDC - SCIFA8 to SCIFA11 (RX64M and RX71M only) - RSPI0 - RSPI1 (RX71M and RX65N only) - RSPI2 (RX65N only) (2) Added the following enum definitions regarding new packages. (RX65N) bsp_int_src_t - BSP_INT_SRC_BL1_RIIC1_TEI1 - BSP_INT_SRC_BL1_RIIC1_EEI1 - BSP_INT_SRC_AL1_GLCDC_VPOS - BSP_INT_SRC_AL1_GLCDC_GR1UF - BSP_INT_SRC_AL1_GLCDC_GR2UF - BSP_INT_SRC_AL1_DRW2D_DRW_IRQ

Rev.	Date	Description	
		Page	Summary
3.60	May.15.17	Program	(3) Corrected the macro definitions due to typos in names of the group interrupt sources. (RX64M, RX71M) The corrected interrupt sources are as follows: - BSP_INT_SRC_BL0_CAC_FERRF -> BSP_INT_SRC_BL0_CAC_FERRI - BSP_INT_SRC_BL0_CAC_ MENDF -> BSP_INT_SRC_BL0_CAC_ MENDI - BSP_INT_SRC_BL0_CAC_OVFF -> BSP_INT_SRC_BL0_CAC_OVFI - BSP_INT_SRC_BL0_DOC_DOPCF -> BSP_INT_SRC_BL0_DOC_DOPCI Changed the following items regarding non-maskable interrupts. [Description] (1) Added the enum definition, BSP_INT_SRC_EXRAM to bsp_int_src_t regarding new packages. (RX65N) (2) Added interrupt processing for EXRAM in vecttbl.c regarding new packages. (RX65N) If a package with ROM of 1 Mbytes or less is selected, this processing will be disabled.
			Change associated with API functions: Modified the branch condition for the number of loop cycles of the R_BSP_SoftwareDelay function. [Description] Before: #if defined(BSP_MCU_RX231) || defined(BSP_MCU_RX64M) || defined(BSP_MCU_RX71M) || ... #define CPU_CYCLES_PER_LOOP 4 #else #define CPU_CYCLES_PER_LOOP 5 #endif After: #ifdef __RXV1 #define CPU_CYCLES_PER_LOOP (5) #else #define CPU_CYCLES_PER_LOOP (4) #endif

Rev.	Date	Description	
		Page	Summary
3.60	May.15.17	Program	Change associated with the lock function: Modified the lock function. (RX130) [Description] Added the following enums regarding new packages: mcu_lock_t - BSP_LOCK_REMC0 - BSP_LOCK_REMC1 - BSP_LOCK_REMCOM - BSP_LOCK_SCI0 - BSP_LOCK_SCI8 - BSP_LOCK_SCI9 - BSP_LOCK_SMCI0 - BSP_LOCK_SMCI8 - BSP_LOCK_SMCI9 - BSP_LOCK_TEMPS Modified the lock function. (RX65N) [Description] Added the following enums regarding new packages: mcu_lock_t - BSP_LOCK_RIIC1 - BSP_LOCK_GLCDC - BSP_LOCK_DRW2D
			Modification Regarding STDIO & Debug console Improved the following. (RX130) [Description] The module did not operate properly when BSP_CFG_USER_CHARGET_ENABLED or BSP_CFG_USER_CHARPUT_ENABLED was set to “enabled” (1), so the program code was modified to ensure proper operation.
			Modification Regarding Pin Function Made changes to the output_ports_configure function in the program code to conform to the notes on MPC in the user’s manual. (RX130) [Description] When setting the given bits of the PMR register to 0, the PDR register to 0, and the PCR register to 0, changed the program code to write the PmnPFS.ASEL bit to 1. Improved the following. (RX111) [Description] (1) PORTH does not exist. Therefore, deleted the port setting. (2) Deleted the following macro definition. - BSP_PRV_PORTH_NE_PIN_MASK

Rev.	Date	Description	
		Page	Summary
3.70	Nov.01.17	—	Added support for GENERIC-RX110. Added support for GENERIC-RX111. Added support for GENERIC-RX113. Added support for GENERIC-RX130. Added support for GENERIC-RX230. Added support for GENERIC-RX231. Added support for GENERIC-RX23T. Added support for GENERIC-RX24T. Added support for GENERIC-RX24U. Added support for GENERIC-RX64M. Added support for GENERIC-RX71M. Added support for Envision Kit for RX65N-2MB
		20	3.2.6 Clock Setup For BSP_CFG_LPT_CLOCK_SOURCE, “2 = LPT not used” has been added as the setting value in the Value and “The default value is 2 (LPT not used)” has been added to the description in the Meaning.
		Program	Changes associated with functions: Added the startup disable function for RX110, RX111, RX113, RX130, RX230, RX231, RX23T, RX24T, RX24U, RX64M, and RX71M. [Description] Added the macro definition, BSP_CFG_STARTUP_DISABLE.
			Changes associated with the low power timer: Modified the following items for RX230 and RX231: [Description] (1) To follow the description for the ILCSTP bit in the User’s Manual, processing to wait for the ILOCO oscillation stabilization time has been added in the usb_lpc_clock_source_select function. (2) To follow the note on the LPT in the User's Manual, the code has been modified to write 0 to the IWDTCSTPR.SLCSTP bit when the IWDT-dedicated on-chip oscillator is used as the clock source of the low power timer. (3) The usb_lpc_clock_source_select function included processing to stop the ILOCO. This processing has been removed since the ILOCO cannot be stopped by the program once it starts oscillation. (4) Added the definition of the IWDT-dedicated on-chip oscillator “BSP_ILOCO_HZ” for RX230 and RX231.

Rev.	Date	Description	
		Page	Summary
3.70	Nov.01.17	Program	Modified the following items for RX130, RX230, and RX231: [Description] (1) Added the following definition for when the LPT module is not used: “BSP_CFG_LPT_CLOCK_SOURCE = 2” (2) Changed the default value of the following definition: BSP_CFG_LPT_CLOCK_SOURCE (0) → (2) (3) Added a branch to processing not to oscillate the sub-clock and the ILOCO when the LPT is not used.
3.71	Dec.20.17	24	3.2.10 Extended Bus Master Priority Setting
		80	Corrected typo for the ‘Board used’ in the Table 9.8 Operation Confirmation Environment.
		108	Corrected typo in Revision Record. (Rev3.70)
		Program	Changes associated with functions: Added the Extended Bus Master Priority Setting function for RX65N-2MB. [Description] Added the following macro definition: - BSP_CFG_EBMAPCR_1ST_PRIORITY - BSP_CFG_EBMAPCR_2ND_PRIORITY - BSP_CFG_EBMAPCR_3RD_PRIORITY - BSP_CFG_EBMAPCR_4TH_PRIORITY - BSP_CFG_EBMAPCR_5TH_PRIORITY
3.80	Jul.01.18	—	Added support for Target Board for RX130 Added support for Target Board for RX231 Added support for Target Board for RX65N Added support for the 384 KB and 256 KB ROM size for RX111.
		—	Changed the name of Section 9.
		—	Updated RX113 iodefine.h to V1.1. Updated RX65N iodefine.h to V2.0A.
		24, 25	Corrected some table names.
		25	Added 3.2.13 Using Smart Configurator.
		32	Corrected a typo for Function name for the "5.1 Summary" table.
		81	R_BSP_StartupOpen -> R_BSP_StartupOpen Added Table 9.10 Confirmed Operation Environment (Rev.3.80).
		Program	Changes associated with functions: Added support setting function of configuration option Using GUI on Smart Configurator for only Generic of RX110, RX111,RX113, RX230, RX231, RX64M, RX65N, and RX71M. [Description] Added a setting file to support configuration option setting function by GUI. Supports peripheral function initialization processing by smart configurator. [Description] Added the following macro definition: -BSP_CFG_CONFIGURATOR_SELECT

Rev.	Date	Description	
		Page	Summary
3.80	Jul.01.18	Program	Processing was added after writing the ROMWT register of RX65N. Processing was added after writing the MEMWAIT register of RX71M. [Description] Added processing to check that the value written to the ROMWT or MEMWAIT register was reflected after the value was written to the ROMWT or MEMWAIT register. Supported tool news number R20TS0302. (RX113, RX210 and RX63T) [Description] Corrected a problem that caused a build error when selecting and building a specific package. For more information on this problem please reference the tool news (R20TS0302). Deleted unnecessary processing. (RX230, RX231 and RX23T) [Description] Deleted the processing for user boot function from RX230, RX231 and RX23T.
3.90	Jul.27.18	—	Added support for RX66T.
		1	Related Documents: Added the following documents: “RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)” “RX Family Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)”
		19	3.2.3 STDIO & Debug Console Changed the chapter headings. Added descriptions regarding definitions. - Addition: BSP_CFG_USER_CHARGET_ENABLED - Addition: BSP_CFG_USER_CHARGET_FUNCTION - Addition: BSP_CFG_USER_CHARPUT_ENABLED - Addition: BSP_CFG_USER_CHARPUT_FUNCTION
		22	3.2.6. Clock Setup Deleted BSP_CFG_ROM_CACHE_ENABLE because it was moved to 3.2.15 ROM Cache Function.
		27	3.2.11. MCU Voltage Added descriptions and definitions. - Addition: BSP_CFG_MCU_AVCC_MV
		28, 29	Added 3.2.14 Negative Voltage Input Settings for AD Pins.
		30, 31	Added 3.2.15 ROM Cache Function.
		32	Added 3.2.16 Callback function at warm start
			Added 3.2.17 Board Revision.
		33	Added 3.2.18 Interrupt Priority Level When FIT Module Interrupts Are Disabled.
		40	Added 4.14 “for”, “while” and “do while” statements.
		41	5.1 Summary Added R_BSP_VoltageLevelSetting function.
		47	5.7 R_BSP_RegisterProtectEnable() Changed the contents of the “Description” section.

Rev.	Date	Description	
		Page	Summary
3.90	Jul.27.18	49	5.8 R_BSP_RegisterProtectDisable() Changed the contents of the “Description” section.
		64, 65	Added 5.19 R_BSP_VoltageLevelSetting()
		66	6. Project Setup 6.1 Adding the FIT Module to Your Project Changed the chapter title and description.
		—	Deleted 6.2 Adding FIT Module with e^2 studio FIT Configurator
		87	9.1 Confirmed Operation Environment Added Table 9.11 Confirmed Operation Environment (Rev. 3.90).
		—	Deleted 9.2 Creating a Project with FIT Plug-in
		Program	Changes associated with functions: Changed the board folder of devices supporting Smart Configurator (RX110, RX111, RX113, RX130, RX230, RX231, RX64M, RX65N, and RX71M). [Description] Deleted all folders other than the generic folder, since other boards can all be substituted with GENERIC_RXxxx. Corresponds to cautionary note in Tool News (R20TS0302) regarding port initialization processing in “bsp_non_existent_port_init” function (RX113, RX210, RX231, RX610, RX62G, RX62N, RX62T, RX631, and RX63N) [Description] Revised port initialization settings. For details, see Tool News (R20TS0302). Added the macro definition of the ID code for RX64M, RX65N, and RX71M. [Description] (1)Added the following macro definition: - BSP_CFG_ID_CODE_LONG_1 - BSP_CFG_ID_CODE_LONG_2 - BSP_CFG_ID_CODE_LONG_3 - BSP_CFG_ID_CODE_LONG_4 (2)Added settings related to macro definitions to the settings file for the GUI-based configuration option setting functionality.
			Changes associated with packages: Changed the macro definitions for the RX23T package. [Description] (1) Deleted the following macro definition. - BSP_CFG_MCU_PART_VERSION Changed the macro definitions for the RX220 package. [Description] (1) Added setting values of the following macro definitions regarding packages. - BSP_CFG_MCU_PART_PACKAGE: Values: FK = 0x3 = LQFP/64/0.80

Rev.	Date	Description	
		Page	Summary
3.90	Jul.27.18	Program	Changed the macro definitions for the RX62T package. [Description] (1) Added setting values of the following macro definitions regarding packages. - BSP_CFG_MCU_PART_PACKAGE: Values: FK = 0x4 = LQFP/64/0.80
			Changes associated with the lock function: Modified the lock function. (RX113) [Description] (1) Added the following enums. mcu_lock_t - BSP_LOCK_TEMPS Modified the lock function. (RX65N) [Description] (1) Added the following enums. mcu_lock_t - BSP_LOCK_SMCI10 - BSP_LOCK_SMCI11
			Modification Regarding Pin Function: Changed the program in the output_ports_configure function, since the timing of the pin settings differs depending on the peripheral function FIT module specifications. (RX210, RX220, RX23T, RX24T, RX24U, RX62G, RX62N, RX630, RX63N, and RX631) [Description] Deleted pin setting processing related to peripheral functions other than those for LEDs and switches.
3.91	Aug.31.18	Program	Changes associated with functions: Modified the following item for RX66T. (1) Corrected execution timing for bsp_volsr_initial_configure function. (2) Added the process for saving and restoring general-purpose registers to the stack in the R_BSP_VoltageLevelSetting function.
4.00	Oct.31.18	—	Added support for RX651 with 64 pin package.
		—	Updated RX65N iodefine.h to V2.2.
		20	3.2.5 RTOS Added BSP_CFG_RTOS_SYSTEM_TIMER in Table 3-5 RTOS Defines.
		87,88	9.1 Confirmed Operation Environment Corrected board used in Table 9.10 Confirmed Operation Environment (Rev. 3.90) and Table 9.11 Confirmed Operation Environment (Rev. 3.91).
		88	9.1 Confirmed Operation Environment Added Table 9.13 Confirmed Operation Environment (Rev. 4.00).

Rev.	Date	Description	
		Page	Summary
4.00	Oct.31.18	Program	Changes associated with functions: Added settings related to macro definitions to the settings file for the GUI-based configuration option setting functionality for only Generic of RX23T, RX24T, and RX24U. [Description] Added a setting file to support configuration option setting function by GUI. Changed the board folder of devices supporting Smart Configurator (RX23T, RX24T, and RX24U). [Description] Deleted all folders other than the generic folder, since other boards can all be substituted with GENERIC_RXxxx. Added support for RTOS of RX64M, RX65N and RX71M. [Description] Added RTOS processing Added the following macro definition: - BSP_CFG_RTOS_SYSTEM_TIMER
			Changes associated with packages: Changed the macro definitions for the RX65N package. [Description] (1) Added setting values of the following macro definitions regarding packages. - BSP_CFG_MCU_PART_PACKAGE: Values: FM = 0x8 = LFQFP/64/0.50 Values: BP = 0xC = TFBGA/64/0.50 (2) Added the following macro definitions: - BSP_PACKAGE_LFQFP64 - BSP_PACKAGE_TFBGA64
4.01	Jan.11.19	—	Added support for RX72T.
		41	5.1 Summary Added notes about RX72T.
		64	5.19 R_BSP_VoltageLevelSetting Added descriptions about RX72T.
		88	9.1 Confirmed Operation Environment Added Table 9.14 Confirmed Operation Environment (Rev. 4.01).
5.00	Mar.15.19	—	Supported the following compilers. - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		—	Performed the following technical update. - TN-RX*-A153A

Rev.	Date	Description	
		Page	Summary
5.00	Mar.15.19	—	Excluded the following devices from operation confirmed devices. - RX210 group - RX21A group - RX220 group - RX610 group - RX62N group - RX62T, RX62G group - RX630 group - RX63T group
		1	Updated operation confirmed devices. Added target compilers. Deleted related documents.
		3	Updated Overview.
		4, 5	Updated 1.2 File Structure.
		6	Updated 2.1 MCU Information.
		7, 8	Updated 2.2 Initialization.
		9	Updated 2.3 Global Interrupts. Updated 2.4 Interrupt Callbacks.
		10	Updated 2.6 Clock Setup. Updated 2.7 STDIO & Debug Console. Updated 2.8 Stacks Area and Heap Area.
		11	Updated 2.10 ID Code. Updated 2.12 Endian. Updated 2.13 Option Function Select Registers. Deleted 2.16 Definition for Each Board.
		12	Updated 2.18 Register Protection.
		14-17	Updated 2.22 Startup Disable.
		18	Updated 3.1 Choosing a Platform.
		19-20	Updated 3.2.1 MCU Product Part Number Information. Updated 3.2.2 Stack & Heap Sizes.
		21	Updated 3.2.5 RTOS.
		22	Updated 3.2.6 Clock Setup.
		24	Updated 3.2.7 Registers in ROM & External Memory Access Protection.
		28	Updated 3.2.12 Startup Disable.
		37	Updated 4.10.2 Hardware Resource Locks.
		40	Updated 4.13 Code Size.
		42	Updated 5.1 Summary.
		43	Updated 5.2 R_BSP_GetVersion().
		63, 64	Updated 5.18 R_BSP_StartupOpen().
		67	Added 5.20 R_BSP_InterruptRequestEnable().
		68	Added 5.21 R_BSP_InterruptRequestDisable().
		69-75	Updated 6. Intrinsic Functions.
		76	Updated 7.1 Adding FIT Module to Your Project.
		77-83	Added 7.2 Adding FIT Modules to the IAR Project.
		95-98	Updated 9. Adding FIT Modules to the User Project.

Rev.	Date	Description	
		Page	Summary
5.00	Mar.15.19	105	10.1 Confirmed Operation Environment Added Table 10.15 Confirmed Operation Environment (Rev.5.00).
		107	Updated Technical Update Information. Deleted web page and support contact.
		Program	**Folder Structure** Changed the folder structure. [Description] (1) Added the following files. - r_bsp_interrupt.c - r_bsp_interrupt.h - linker_script_rvectors.inc - r_rx_compiler.h - r_rx_intrinsic_functions.c - r_rx_intrinsic_functions.h - r_rots.h - reset_program.s - mcu_clocks.h (2) Eliminated device dependence from the following files in the board folder and moved them to the all folder. - dbsct.c - lowlvl.c - lowsrc.c - lowsrc.h - resetprg.c - sbrk.c (3) Eliminated device dependence from the following files in the mcu folder and moved them to the all folder. - cpu.c - locking.c - mcu_locks.c - mcu_startup.c - mcu_startup.h - resetprg.c - sbrk.c (4) Eliminated board dependence from the following files in the board folder and moved them to the mcu folder. - vecttbl.c - vecttbl.h (5) Added the following folders to the register_access folder. - ccrx - gnuc - iccrx

Rev.	Date	Description	
		Page	Summary
5.00	Mar.15.19	Program	(6) Moved the following processing. - Moved the ROM cache settings from resetprg.c to hwsetup.c. - Moved the clock settings from resetprg.c to mcu_clocks.c. - Moved the include settings of RTOS from r_bsp.h to rots.h. - Moved intrinsic related API functions from mcu_interrupt.c to r_bsp_interrupts.c. - Moved exception interrupt functions from vecttbl.c to r_bsp_interrupts.c. (7) Changed the name of the following files. - cpu.c -> r_bsp_cpu.c - cpu.h -> r_bsp_cpu.h - locking.c -> r_bsp_locking.c - locking.h -> r_bsp_locking.h - mcu_startup.c -> r_bsp_mcu_startup.c - mcu_startup.h -> r_bsp_mcu_startup.h
			Clock related Corrected the clock setting procedure. (RX110, RX111, RX113, RX130, RX230, RX231) [Description] (1) Corrected the clock setting processing of HOCO, main clock, sub clock, and PLL. (2) Added the following macro definitions. - BSP_CFG_MAIN_CLOCK_SOURCE - BSP_CFG_MOSC_WAIT_TIME - BSP_CFG_RTC_ENABLE - BSP_CFG_SOSC_DRV_CAP - BSP_CFG_SOSC_WAIT_TIME (3) Deleted the following macro definitions. (RX110, RX111, RX113) - BSP_CFG_USE_CGC_MODULE Change Main Clock Oscillator Wait Time to initial value of register. [Description] (1) Changed the default value of the following macro definitions. - BSP_CFG_MOSC_WAIT_TIME (0x06) ⇒ (0x04) Supported low power timers. (RX113) [Description] (1) Added processing that oscillates the clock source of the low power timer when the low power timer is used. (2) Added the following macro definition. - BSP_CFG_LPT_CLOCK_SOURCE

Rev.	Date	Description	
		Page	Summary
5.00	Mar.15.19	Program	Changed the default value of the clocks (ICLK, PCLKB, PCLKD, FCLK) from 24 MHz to 32 MHz. (RX113) [Description] (1) Changed the default value of the following macro definitions. - BSP_CFG_PLL_DIV (2) ⇒ (4) - BSP_CFG_PLL_MUL (6) ⇒ (8) - BSP_CFG_ICK_DIV (2) ⇒ (1) - BSP_CFG_PCKB_DIV (2) ⇒ (1) - BSP_CFG_PCKD_DIV (2) ⇒ (1) - BSP_CFG_FCK_DIV (2) ⇒ (1) Added settings in case the LCD module is not used. (RX113) [Description] (1) Added the following definition in case the LCD module is not used. - BSP_CFG_LCD_CLOCK_SOURCE = 5 (2) Changed the default value of the following definition. - BSP_CFG_LCD_CLOCK_SOURCE (2) -> (5) Deleted processing related to the CGC FIT module. (RX110, RX111, RX113) [Description] Deleted all processing related to the FIT module of CGC. Modified the following items for RX113 and RX231: [Description] Changed the default value of the following definition: BSP_CFG_USB_CLOCK_SOURCE (0) -> (1)
			Lock related Changed the content related to the lock function. (RX100, RX200, RX600 (RX631, RX63N, RX64M are excluded), Operation confirmed device supporting RX700 (RX71M is excluded)) [Description] (1) Deleted the following enum definition. - BSP_LOCK_SMCIx (x is any value from 0 to 12) Changed the content of the lock function. (RX64M, RX71M) [Description] (1) Changed the following enum definitions. - BSP_LOCK_EPTPC0 - BSP_LOCK_EPTPC1 - BSP_LOCK_PTPEDMAC
			STDIO/debug console related Corrected the following content. (RX110, RX113, RX230, RX231) [Description] Even though BSP_CFG_USER_CHARGET_ENABLED or BSP_CFG_USER_CHARPUT_ENABLED was set to enabled (“1”), correct operation was not performed, therefore, it was corrected so that normal operation is performed.

Rev.	Date	Description	
		Page	Summary
5.00	Mar.15.19	Program	**Function related** Supported the extended language specifications of CCRX by multiple compilers. [Description] Added #pragma, key word, and the macro definition of intrinsic functions and section address operators. (For details, refer to r_rx_compiler.h, r_rx_intrinsic_functions.c, r_rx_intrinsic_functions.h.) Added initialization processing of variables to resetprg.c. [Description] Added processing that initializes variables that have not been initialized after reset release. Added initialization of double precision floating point function to resetprg.c. [Description] Added processing that initializes DPSW after reset release. Added initialization of trigonometric function calculator to resetprg.c. [Description] Added processing that initializes TFU after reset release. Added the macro definition of the MCU function to mcu_info.h. [Description] Added the macro definition for judging the function implemented for each device. Added support Group BE0 intterrupts. (RX64M, RX65N, RX66T, RX71M, and RX72T) [Description] (1) Added the group_be0_handler_isr function. (2) Added the following enums. bsp_int_src_t - BSP_INT_SRC_BE0_CAN0_ERS0 - BSP_INT_SRC_BE0_CAN1_ERS1 (except RX66T, and RX72T) - BSP_INT_SRC_BE0_CAN2_ERS2 (except RX66T, and RX72T)
5.10	Mar.29.19	—	Added support for RX23W.
		24	Added the following macro definitions in section 3.2.6 Clock Setup: - BSP_CFG_CLKOUT_RF_MAIN
		42	5.1 Summary Added R_BSP_ConfigClockSetting function.
		69	Added 5.22 R_BSP_ConfigClockSetting()
		106	10.1 Confirmed Operation Environment Added Table 10.16 Confirmed Operation Environment (Rev.5.10).
5.20	Apr.08.19	—	Added support for RX72M.
		22	3.2.6 Clock Setup Revised the value of BSP_CFG_USB_CLOCK_SOURCE.

Rev.	Date	Description	
		Page	Summary
5.20	Apr.08.19	24	3.2.6 Clock Setup Added the following definition: - BSP_CFG_PPLL_DIV - BSP_CFG_PPLL_MUL - BSP_CFG_PHY_CLOCK_SOURCE - BSP_CFG_ESC_CLOCK_SOURCE - BSP_CFG_CLKOUT_SOURCE - BSP_CFG_CLKOUT_DIV - BSP_CFG_CLKOUT_OUTPUT
		108	10.1 Confirmed Operation Environment Added Table 10.17 Confirmed Operation Environment (Rev.5.20).
5.21	Jul.23.19	108	10.1 Confirmed Operation Environment Added Table 10.18 Confirmed Operation Environment (Rev.5.21).
		Program	Changes associated with functions: Added changes for RTOS support of RX110, RX111, RX113, RX130, RX230, RX231, RX23T, RX23W, RX24T, RX24U, RX63N RX66T, and RX72T. [Description] Added the following macro definition: - BSP_CFG_RTOS_SYSTEM_TIMER

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 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. 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.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

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/.