RL78 Family

Board Support Package Module Using Software Integration System

Summary

The Renesas board support package SIS module (r_bsp) forms the foundation of any project that uses Software Integration System (SIS) modules. The r_bsp is easily configurable and provides all the code needed to get the MCU and the board from reset to the main() function. This document describes r_bsp conventions and explains how to use it, configure it, and create a BSP for your own board.

Device on Which Operation Confirmed

RL78/G23 Group

RL78/F23, F24 Group

RL78/G15 Group

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

Supported Compilers

- Renesas Electronics C/C++ Compiler Package for RL78 Family
- IAR C/C++ Compiler for Renesas RL78
- LLVM C/C++ Compiler for Renesas RL78

For details of the confirmed operation of each compiler, refer to 7.1, Confirmed Operating Environment.

Limitations apply to some functions. Refer to 4.4, Limitations.

Contents

1. Overview

Before running a user application there are a series of operations that must be performed to get the MCU set up properly. These operations, and their number, will vary depending on the MCU being used. Common examples include: setting up stack(s), initializing memory, configuring the CPU and peripheral hardware clock, and setting up port pins. The steps described in this document must to be followed in order to configure the above items. The r_bsp is provided in order to make configuration easier.

The r_bsp provides all the elements needed to get the MCU from reset to the start of the user application's main() function. The r_bsp also provides common functionality that is needed by many applications. Examples of this include functions to start and stop the clocks and to get the frequency of the CPU and peripheral hardware clock.

The necessary steps after a reset are the same for every application, but this does not mean that the settings will be the same. For example, stack sizes and the clocks used will vary depending on the application. The r_bsp configuration options are contained in the config header file for easy access.

1.1 Terminology

Term	Description
Platform	The user's development board. Used interchangeably with "board."
BSP	Abbreviation of "board support package."

1.2 File Structure

The r_bsp file structure is shown below in Figure 1.1. The *r_bsp* folder contains three folders and two files.

The *doc* folder contains r_bsp documentation.

The *board* folder contains the *generic* folders.

There is a *generic* folder for each supported MCU.

Figure 1.2 shows the contents of the *generic* folder.

The *mcu* folder contains one folder for each supported MCU. The *mcu* folder also contains the *all* folder, which contains source code common to all MCUs supported by the r_bsp.

The *platform.h* file allows you to choose your current development platform. It is used to select all the header files from the *board* and *mcu* folders required for your project. This is discussed in more detail in later sections.

The *readme.txt* file provides a summary of information about the r_bsp.

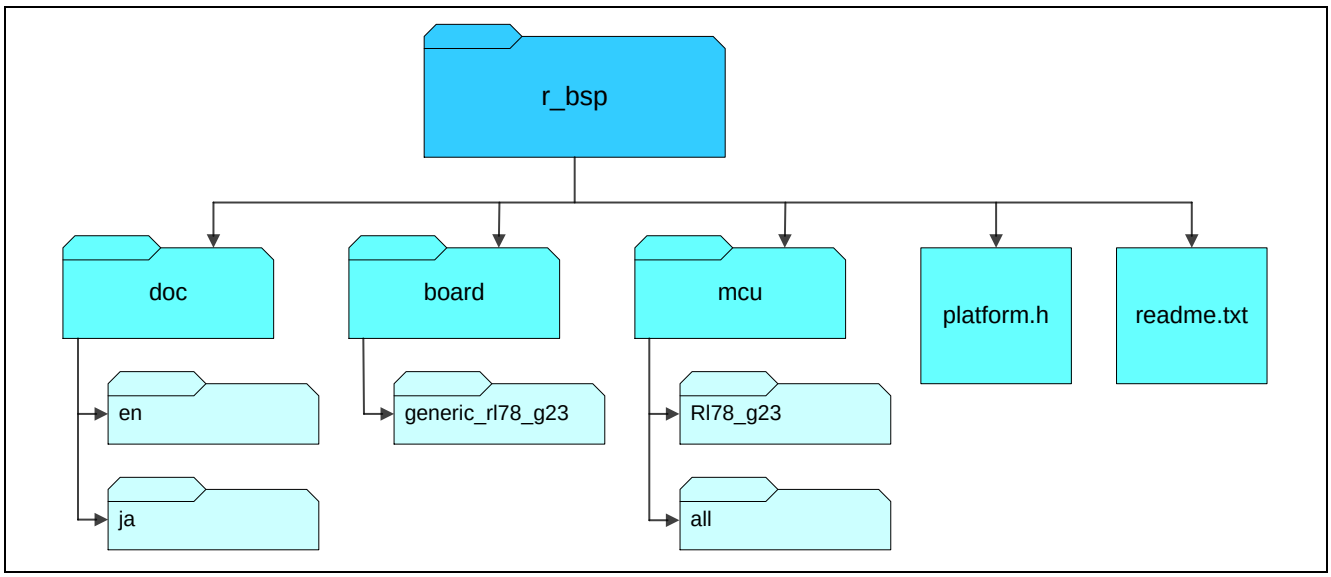


Figure 1.1 r_bsp File Structure

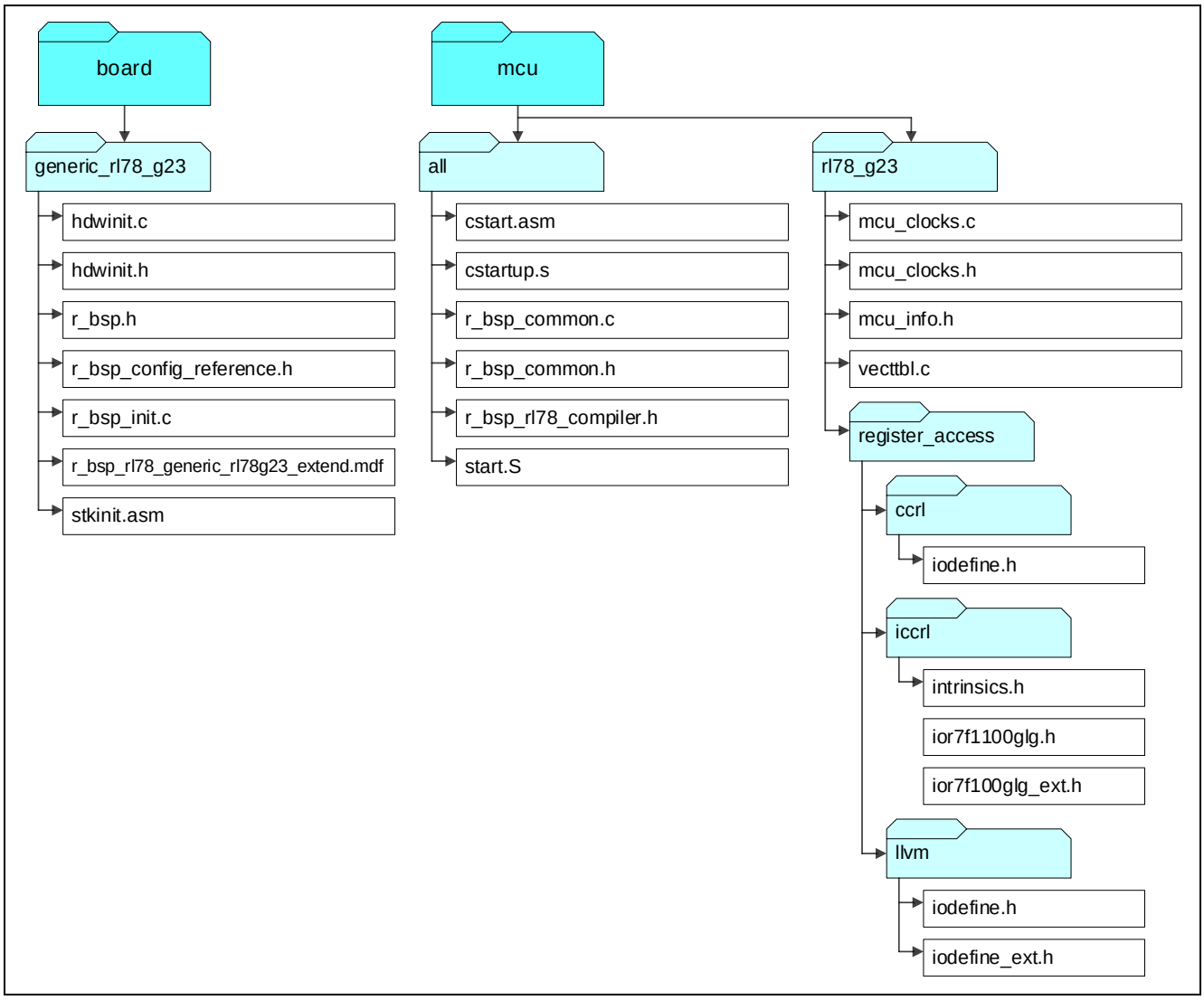


Figure 1.2 Structure of Generic Folder

2. Functionality

This section describes in detail the functionality provided by the r_bsp.

2.1 MCU Information

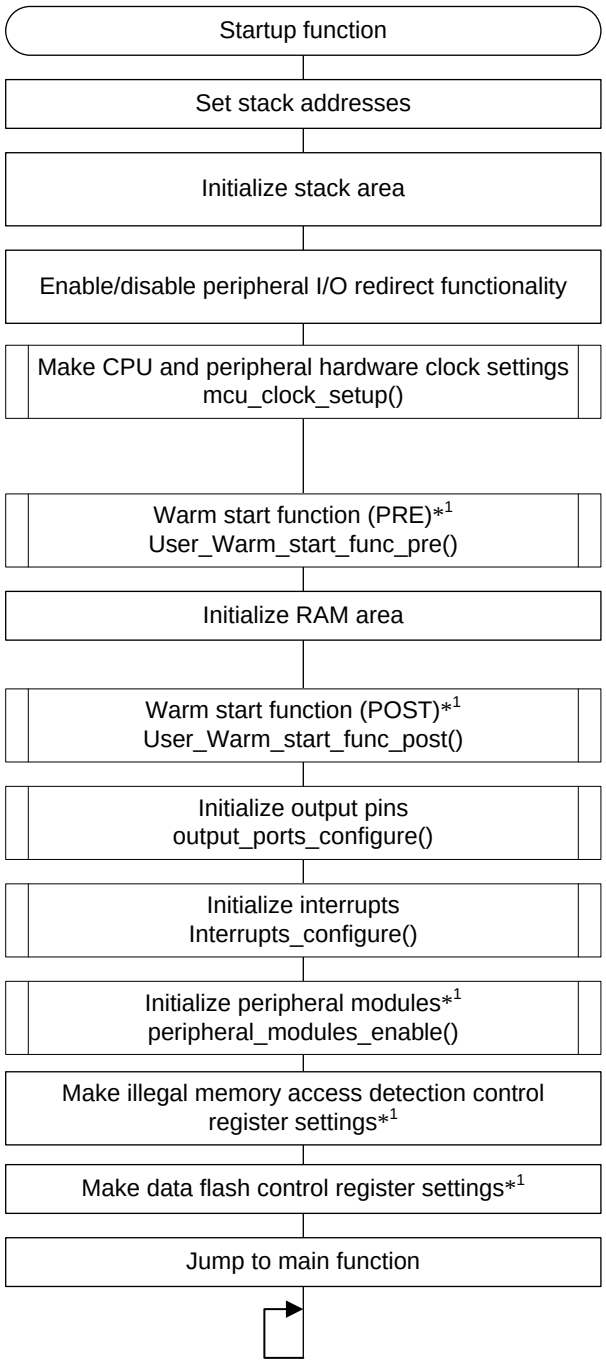
One of the main benefits of the r_bsp is that it lets you define the global system settings only once, in a single place in the project, and those settings are then shared throughout. This information is defined in the r_bsp and can then be used by the SIS modules and user code. SIS modules use this information to automatically configure their code to match your system configuration. If the r_bsp did not provide this information, you would have to specify system information to each SIS 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*. An example of an MCU-specific macro in *mcu_info.h* is shown below.

Definition	Description
BSP_MCU_FAMILY_<MCU_FAMILY>	Which MCU Family this MCU belongs to. Example: BSP_MCU_FAMILY_RL78 would be defined if the MCU was an RL78/G23.
BSP_MCU_SERIES_<MCU_SERIES>	Which MCU Series this MCU belongs to. Example: BSP_CMU_SERIES_RL78G2X would be defined if the MCU was RL78/G23.
BSP_MCU_GROUP_<MCU_GROUP>	Which MCU group this MCU belongs to. Example: BSP_MCU_GROUP_RL78G23 would be defined if the MCU was RL78/G23.
BSP_<CLOCK>_HZ	Each of these macros corresponds to one of the MCU's clocks. Each macro defines the corresponding clock's frequency in hertz (Hz). For example, BSP_LOCO_HZ defines the LOCO frequency in Hz, and BSP_SUB_CLOCK_HZ defines the subsystem clock frequency in Hz.

2.2 Initial Settings

The _start function is set as the reset vector for the MCU when using the Renesas compiler, and the PowerON_Reset function is set as the reset vector when using the LLVM compiler. The __iar_program_start function is set as the reset vector for the MCU when using the IAR compiler. The _start function, PowerON_Reset_PC function, or function __iar_program_start function (the startup function) performs various types of initialization processing to get the MCU ready to use the user application. The flowcharts below show startup function operations and CPU and peripheral hardware clock settings.

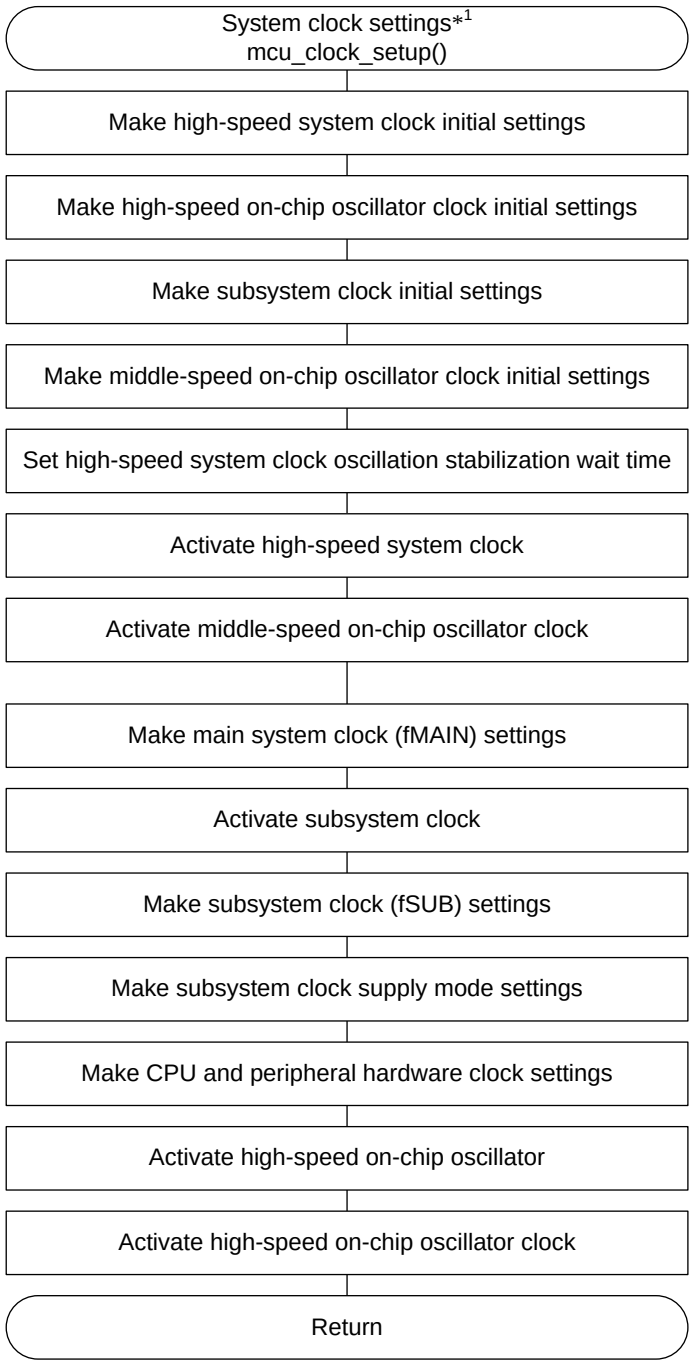


- Sets the clock division ratio and multiplication factor.
- Stops clocks that are not used as the clock source.
- Waits for stabilization of the clock used as the clock source.
- Transitions to the selected clock.

- Sets uninitialized data areas to zero.
- Copies data to the data area to be initialized.

Note: 1. The operation differs according to the settings in *r_bsp_config.h*.

Figure 2.1 Flowchart of Startup Function



Note: 1. The operation differs according to the settings in *r_bsp_config.h*.

Figure 2.2 Flowchart of CPU and Peripheral Hardware Clock Settings

2.3 Global Interrupts

Interrupts are disabled after a reset. Enable interrupts as needed. Use the BSP_ENABLE_INTERRUPT function to enable interrupts and the BSP_DISABLE_INTERRUPT function to disable them. For details, refer to 5.1, Overview.

RL78 devices have a fixed vector table. The fixed vector table is located at a static location at the top of the memory map.

When using the Renesas compiler or LLVM compiler the fixed vector table is defined in *iodefine.h*, and when using the IAR compiler it is defined in *iorxxx.h*.*[1]

Note: 1. The characters represented by xxx differ depending on the device.

2.4 Clock Settings

CPU and peripheral hardware clock settings are made during r_bsp initialization. Clocks are configured based upon the user's settings in the *r_bsp_config.h* file (see 3.2.5). Clock settings are applied before the C runtime environment is initialized. When a clock is selected, the code in the r_bsp implements the required delays to allow the selected clock to stabilize.

2.5 Stack Area

The stacks are configured and initialized by the startup function after a reset. When using the IAR compiler it is possible to specify the stack size using a GUI.

2.6 ID Code

RL78 MCUs have a 10-byte ID code stored 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. ID code resides in the on-chip debug security ID setting memory. The value of the security ID is specified in the compile options of the Renesas compiler environment. In the IAR or LLVM environment it is specified in *r_bsp_config.h*. For details of ID code options, refer to the Option Bytes and On-Chip Debug Function chapters in your MCU's hardware manual.

2.7 Option Bytes

The option bytes are located in the flash memory of RL78 MCUs. The option bytes are referenced automatically after power-on or a reset, and the specified function settings are applied. Option bytes can be used to specify settings for the watchdog timer or voltage detection circuit, for example. Option byte setting values are specified in the compile options of the Renesas compiler or LLVM environment. In the IAR environment they are specified in *r_bsp_config.h* (see 3.2.6).

2.8 RAM/SFR Guard Functionality

RL78 MCUs are provided with an illegal memory access detection control register that protects the data in the specified RAM space as well as the data in the control registers of the port, interrupt, clock control, voltage detection circuit, and RAM parity error detection functions. The setting values can be specified in *r_bsp_config.h*.

2.9 CPU Functionality

API functions are provided for making settings related to CPU functionality such as enabling and disabling interrupts. Refer to Section 5 for details.

2.10 Disabling Startup

To disable startup, manually delete the startup assembler code. The names of the files containing the startup assembler code for each environment are as follows:

- Renesas compiler: cstart.asm
- LLVM compiler: start.S
- IAR compiler: cstartup.s

Additionally, you will need to add your own startup code.

2.10.1 Settings to Disable Startup

Make settings as described below to disable BSP startup processing.

(1) Configuration File Settings

Specify your own startup processing in *r_bsp_config.h*. Some BSP API functions and peripheral SIS modules reference the contents of *r_bsp_config.h*. Note that some SIS modules may not function correctly if there are discrepancies between the details of the startup processing you created and the contents of *r_bsp_config.h*.

The BSP information referenced by the peripheral SIS modules is generated based on *r_bsp_config.h*, so it is necessary to ensure that the details of the startup processing you created and the contents of *r_bsp_config.h* match.

Figure 2.3 illustrates configuration file settings.

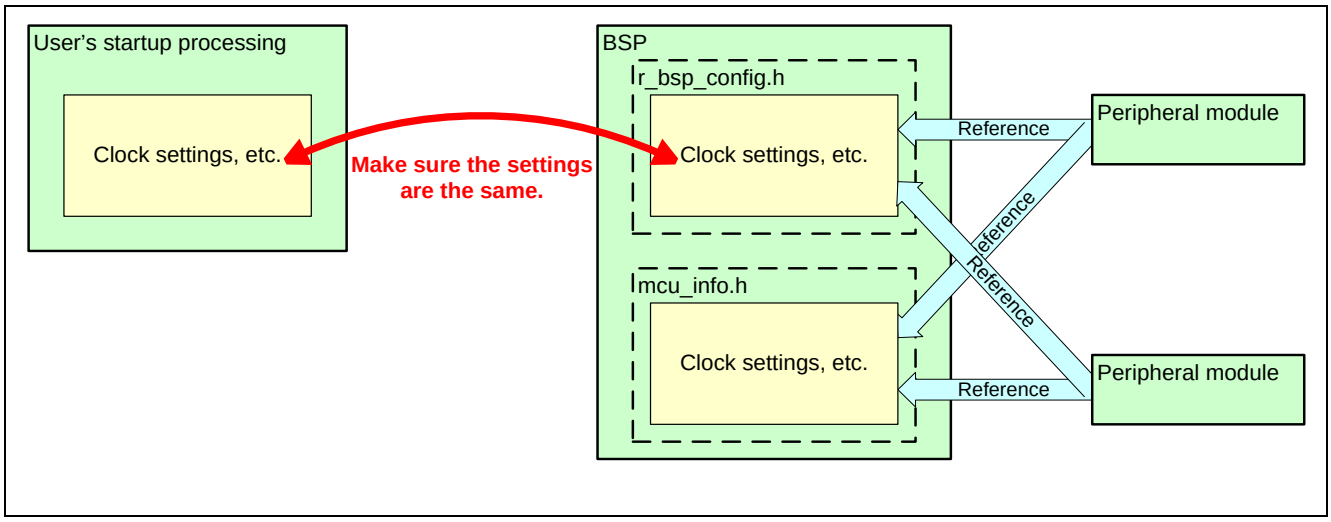


Figure 2.3 Configuration File Settings

3. Configuration

Two header files are used to configure the r_bsp. One is used to choose the platform, and the other to configure the chosen platform.

3.1 Choosing a Platform

The r_bsp provides board support packages for a variety of MCUs. Choosing the platform to be used is accomplished by modifying the *platform.h* file located in the *r_bsp* folder.

3.2 Platform Configuration

After selecting a platform, you must configure it. The file *r_bsp_config.h* contains the platform settings. Each platform has a configuration file called *r_bsp_config_reference.h*, which is located in the platform's *board* folder.

The contents of each *r_bsp_config.h* file differs according to the MCU associated with it, but many of the options are the same. The following sections 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 identify.

When using Smart Configurator, the configuration options can be set on the software component configuration screen. Setting values are automatically reflected in *r_bsp_config.h* when adding modules to a user project.

3.2.1 MCU Product Part Number Information

The MCU's product part number information makes it possible to provide a variety of information about the MCU along with the r_bsp. Information related to the MCU's product part number is defined at the beginning of the configuration file. All of these macros start with "BSP_CFG_MCU_PART." Some MCUs have more product part number–related information than others, but the standard definitions are listed below.

Table 3.1 Product Part Number Definitions

Definition	Value	Description
BSP_CFG_MCU_PART_ROM_SIZE	See comments above #define in *r_bsp_config.h*.	Defines the ROM size.
BSP_CFG_MCU_PART_PIN_NUM		Defines the pin count.
BSP_CFG_MCU_PART_HAS_DATA_FLASH		Defines whether or not the device incorporates flash memory.
BSP_CFG_MCU_PART_ROM_TYPE		Defines the device type.

3.2.2 Peripheral I/O Redirection Register

RL78 MCUs provide functionality to switch the ports assigned to alternate functions. After a reset the r_bsp makes MCU pin assignment settings using the pin assignment configuration macros in *r_bsp_config.h*.

Table 3.2 Peripheral I/O Redirection Register Definitions

Definition	Corresponding Device	Value	Description
BSP_CFG_PIORx (x=0–5)	RL78/G23	See comments above #define in *r_bsp_config.h*.	Defines ports to which alternate functions are assigned. The value of x varies from device to device. Refer to r_bsp_config.h for the details of each definition.
BSP_CFG_PIORyy (yy=00-99)	RL78/F23, RL78/F24, RL78/G15	See comments above #define in *r_bsp_config.h*.	Defines ports to which alternate functions are assigned. The value of yy varies from device to device. Refer to r_bsp_config.h for the details of each definition.

3.2.3 RAM/SFR Guard Functionality

RL78 MCUs are provided with functionality to protect the data in the specified RAM space as well as the data in the control registers of the port, interrupt, clock control, voltage detection circuit, and RAM parity error detection functions. After a reset the r_bsp makes MCU guard area settings using the guard functionality configuration macros in *r_bsp_config.h*.

Table 3.3 RAM/SFR Guard Functionality Definitions

Definition	Value	Description
BSP_CFG_INVALID_MEMORY_ACCESS_DETECTION_ENABLE	See comments above #define in *r_bsp_config.h*.	Defines whether or not illegal memory access detection is performed.
BSP_CFG_RAM_GUARD_FUNC		Defines the size of the RAM guard space.
BSP_CFG_PORT_FUNCTION_GUARD		Defines whether or not guarding is applied to port function control registers.
BSP_CFG_INT_FUNCTION_GUARD		Defines whether or not guarding is applied to interrupt function registers.
BSP_CFG_CHIP_STATE_CTRL_GUARD		Defines whether or not guarding is applied to clock control, voltage detection circuit, and RAM parity error detection function control registers.

3.2.4 RAM start address

RL78 MCUs has a function that can change the start address of RAM.

After reset, r_bsp sets the RAM start address using the RAMSAR address definition and RAM start address definition in r_bsp_config.inc.

Table 3.4 RAM start address setting register Definitions

Definition	Value	Description
BSP_CFG_ASM_RAMSAR_ADDRESS	Set in r_bsp_config.inc.	Defines the address of the RAMSAR register. No setting is required in the CC-RL environment.
BSP_CFG_ASM_RAM_GUARD_START_ADDRESS		Defines the RAM start address. Do not define for devices that do not have a RAMSAR register.

3.2.5 Data Flash Access Restriction

RL78 MCUs are provided with functionality to enable or disable access to the data flash. After a reset the r_bsp makes data flash access settings using the data flash access restriction functionality configuration macros in *r_bsp_config.h*.

Table 3.5 Data Flash Access Restriction Definitions

Definition	Value	Description
BSP_CFG_DATA_FLASH_ACCESS_ENABLE	See comments above #define in *r_bsp_config.h*.	Defines whether access to the data flash is enabled or disabled.

3.2.6 RTOS(r_bsp_config.h)

Defines if a RTOS is being used in the current application. After a reset the r_bsp makes RTOS settings using the RTOS functionality configuration macros in r_bsp_config.h.

Table 3.6 RTOS(r_bsp_config.h) Definitions

Definition	Value	Description
BSP_CFG_RTOS_USED	0=RTOS is not used. 1=Reserved. 2=Reserved. 3=Reserved. 4=Renesas ITRON is used.	Defines whether access to the data flash is enabled or disabled. Set the same value as BSP_CFG_ASM_RTOS_USED in r_bsp_config.inc.

3.2.7 RTOS(r_bsp_config.inc)

Defines if a RTOS is being used in the current application. After a reset the r_bsp makes RTOS settings using the RTOS functionality configuration macros in r_bsp_config.inc.

Table 3.7 RTOS(r_bsp_config.inc) Definitions

Definition	Value	Description
BSP_CFG_ASM_RTOS_USED	0=RTOS is not used. 1=Reserved. 2=Reserved. 3=Reserved. 4=Renesas ITRON is used.	Defines whether access to the data flash is enabled or disabled. Set the same value as BSP_CFG_RTOS_USED in r_bsp_config.h.

3.2.8 Clock Settings

The available clocks vary among RL78 MCUs, but the same basic concepts apply to all. After a reset the r_bsp initializes the MCU clocks using the clock configuration macros in *r_bsp_config.h*.

Table 3.8 Clock Setting Definitions

Definition	Value	Description
BSP_CFG_HISYSCLK_SOURCE	0 = Port 1 = Connected crystal/ceramic oscillator 2 = External clock input	Defines the oscillation source of the high-speed system clock.
BSP_CFG_HISYSCLK_OPERATION	(X1 oscillation mode) 0 = X1 oscillator operating 1 = X1 oscillator stopped (External clock input mode) 0 = External clock from EXCLK pin is valid 1 = External clock from EXCLK pin is invalid (Port mode) 0 = I/O port 1 = I/O port	Defines high-speed system clock operation control.
BSP_CFG_SUBCLK_SOURCE	0 = Input port 1 = Connected crystal oscillator 2 = External clock input	Defines the oscillation source of the subsystem clock.
BSP_CFG_SUBCLK_OPERATION	(XT1 oscillation mode) 0 = XT1 oscillator operating 1 = XT1 oscillator stopped (External clock input mode) 0 = External clock from EXCLKS pin is valid 1 = External clock from EXCLKS pin is invalid (Port mode) 0 = Input port 1 = Input port	Defines subsystem clock operation control.
BSP_CFG_MOCO_SOURCE	0 = Middle-speed on-chip oscillator stopped 1 = Middle-speed on-chip oscillator operating	Defines whether the middle-speed on-chip oscillator clock operates or is stopped.
BSP_CFG_OCOCLK_SOURCE	0 = High-speed on-chip oscillator clock 1 = Middle-speed on-chip oscillator clock	Defines the clock source used as the main on-chip oscillator clock (f_{OCO}).
BSP_CFG_MAINCLK_SOURCE	0 = Main on-chip oscillator clock (f_{OCO}) 1 = High-speed system clock (f_{MX})	Defines the clock source used as the main system clock (f_{MAIN}).
BSP_CFG_SUBSYSCLK_SOURCE	0 = Subclock 1 = Low-speed on-chip oscillator clock	Defines the clock source used as the subsystem clock.

Definition	Value	Description
BSP_CFG_FCLK_SOURCE	0 = Main system clock (f_{MAIN}) 1 = Subsystem clock (f_{SUB})	Defines the clock source used as the CPU and peripheral hardware clock (f_{CLK}).
BSP_CFG_XT1_OSCMODE	0 = Low-power oscillation 1 (default) 1 = Normal oscillation 2 = Low-power oscillation 2 3 = Low-power oscillation 3	Defines the oscillation mode of the XT1 oscillator circuit.
BSP_CFG_FMX_HZ	High-speed system clock frequency (unit: Hz)	Defines the frequency of the high-speed system clock.
BSP_CFG_X1_WAIT_TIME_SEL	0 = $2^8/f_X$ 1 = $2^9/f_X$ 2 = $2^{10}/f_X$ 3 = $2^{11}/f_X$ 4 = $2^{13}/f_X$ 5 = $2^{15}/f_X$ 6 = $2^{17}/f_X$ 7 = $2^{18}/f_X$	Defines the oscillation stabilization time of the X1 clock.
BSP_CFG_ALLOW_FSUB_IN_STOPHALT	0 = Supply of subsystem clock to peripheral functions enabled 1 = Supply of subsystem clock to peripheral functions other than realtime clock stopped	Defines supply of the subsystem clock in STOP mode and in HALT mode when the CPU is operating on the subsystem clock.
BSP_CFG_ALLOW_FSL_IN_STOPHALT	0 = Enables supply of subsystem/low-speed on-chip oscillator select clock to peripheral functions. 1 = Stops supply of subsystem/low-speed on-chip oscillator select clock to peripheral functions.	Defines setting in STOP mode or HALT mode while subsystem/low-speed on-chip oscillator select clock is selected as CPU clock.
BSP_CFG_FIL_OPERATION	0 = Low-speed on-chip oscillator stopped. 1 = Low-speed on-chip oscillator operating.	Defines selection of CPU/peripheral hardware clock(fCLK).
BSP_CFG_RTC_OUT_CLK_SOURCE	0 = Subsystem clock 1 = Low-speed on-chip oscillator clock	Defines the operating clock of the realtime clock, 32-bit interval timer, UART0 and UART1 serial interfaces, remote control signal reception function, and clock output/buzzer output control circuit.
BSP_CFG_HOCO_DIVIDE	(When FRQSEL3 = 0) 0 = f_{IH}: 24 MHz 1 = f_{IH}: 12 MHz 2 = f_{IH}: 6 MHz 3 = f_{IH}: 3 MHz (When FRQSEL3 = 1) 0 = f_{IH}: 32 MHz 1 = f_{IH}: 16 MHz 2 = f_{IH}: 8 MHz 3 = f_{IH}: 4 MHz 4 = f_{IH}: 2 MHz	Defines the frequency of the high-speed on-chip oscillator. Use an option byte (000C2H) to specify the setting of FRQSEL3. See 2.7 for the setting procedure.

Definition	Value	Description
	5 = f_{IH}: 1 MHz	
BSP_CFG_WAKEUP_MODE	0 = Normal activation 1 = Fast activation	Defines the high-speed on-chip oscillator activation setting when STOP mode is canceled and when transitioning to SNOOZE mode.
BSP_CFG_MOSC_DIVIDE	0 = f_{MX} 1 = f_{MX}/2 2 = f_{MX}/4 3 = f_{MX}/8 4 = f_{MX}/16	Defines the frequency dividing ratio of the high-speed system clock.
BSP_CFG_MOCO_DIVIDE	0 = 4 MHz 1 = 2 MHz 2 = 1 MHz	Defines the frequency of the middle-speed on-chip oscillator.
BSP_CFG_FMP_DIVIDE	0 = Selects fMP 1 = Selects fMP/2 2 = Selects fMP/4 3 = Selects fMP/8 4 = Selects fMP/16 5 = Selects fMP/32 6 = Selects fMP/64	Defines fMP clock division control.
BSP_CFG_PLL_DIVIDE	0 = No division 1 = Divides the clock frequency by 2 2 = Divides the clock frequency by 4	Defines control of PLL frequency division selection.
BSP_CFG_PLL_OPERATION	0 = Stops PLL operation 1 = Starts PLL operation	Defines control of PLL operation.
BSP_CFG_FMAIN_DIVIDE	0 = No division 1 = Divided by 2 (fMAIN = 16 MHz input only) 2 = Divided by 4 (fMAIN = 20 MHz input only)	Defines control of PLL input clock (fPLLI) division selection.
BSP_CFG_PLL_MULTI	0 = Multiples the clock frequency by 12. 1 = Multiples the clock frequency by 16. 2 = Multiples the clock frequency by 10. 3 = Multiples the clock frequency by 20.	Defines control of PLL multiplication selection.
BSP_CFG_PLL_MODE	0 = Clock through mode (fMAIN 1 = PLL-clock-selected mode (fPLL)	Defines control of clock mode selection.
BSP_CFG_FPLL_HZ	PLL clock frequency (unit: Hz)	Defines the frequency of the PLL clock.
BSP_CFG_LOCKUP_WAIT_COUNT_SEL	0 = Selects 128/fMAIN 1 = Selects 256/fMAIN 2 = Selects 512/fMAIN 3 = Selects 1024/fMAIN	Defines control of setting lock-up wait counter.
BSP_CFG_CAN_CLOCK_OPERATION	0 = Stops CAN X1 clock (fX) supply.	Defines control of supplying or stopping CAN X1 clock (fX).

Definition	Value	Description
	1 = Enables CAN X1 clock (fX) supply.	
BSP_CFG_LIN1_CLOCK_SOURCE	0 = Selects the fCLK clock 1 = Selects the fMX clock	Defines control of selecting LIN1 communication clock source.
BSP_CFG_LIN1_CLOCK_OPERATION	0 = Stops LIN1 communication clock source supply 1 = Enables LIN1 communication clock source supply	Defines control of supplying or stopping LIN1 communication clock source.
BSP_CFG_LIN0_CLOCK_SOURCE	0 = Selects the fCLK clock 1 = Selects the fMX clock	Defines control of selecting LIN0 communication clock source.
BSP_CFG_LIN0_CLOCK_OPERATION	0 = Stops LIN0 communication clock source supply 1 = Enables LIN0 communication clock source supply	Defines control of supplying or stopping LIN0 communication clock source.
BSP_CFG_TRD_CLOCK_SOURCE	0 = Selects fCLK or fMP 1 = Selects fSL	Defines control of TRDe clock selection.
BSP_CFG_ADC_ENABLE	0 = Stops input clock supply. - SFR used by the A/D converter cannot be written. - The A/D converter is in the reset status. 1 = Enables input clock supply. - SFR used by the A/D converter can be read and written.	Defines control over supply of the input clock for the 12-bit A/D converter
BSP_CFG_ADCLK_DIVIDE	0 = Non divided (fCLK) 1 = Divided by 2 (fCLK/2) 2 = Divided by 4 (fCLK/4) 3 = Divided by 8 (fCLK/8)	Defines selection A/D conversion clock.
BSP_CFG_SUBWAITTIME	Loop count (unit: number of times)	Defines the subsystem clock oscillation stabilization wait time. Defined as the loop count using the main system clock.*1
BSP _CFG_FIHWAITTIME	Loop count (unit: number of times)	Defines the high-speed on-chip oscillator clock oscillation stabilization wait time. Defined as the loop count using the main system clock.*1
BSP_CFG_FIMWAITTIME	Loop count (unit: number of times)	Defines the middle-speed on-chip oscillator clock oscillation stabilization wait time. Defined as the loop count using the main system clock.*1
BSP_CFG_FILWAITTIME	Loop count (unit: number of times)	Defines the low-speed on-chip oscillator clock oscillation stabilization wait time. Defined as the loop count using the main system clock.*1

Definition	Value	Description
BSP_CFG_PLLWAITTIME	Loop count (unit: number of times)	Defines the stable wait time for the PLL multiplication setting. Defined as the loop count using the main system clock.*1
BSP_CFG_FIH_START_ON_STARTUP	0 = High-speed on-chip oscillator clock stops 1 = High-speed on-chip oscillator clock starts	Defines the operation of the high-speed on-chip oscillator clock at initialization.

Note: 1. The loop count refers to a loop consisting of a “for” statement that executes a single NOP instruction.
The actual source code is as follows:

```
/* WAIT_LOOP */
for (w_count = 0U; w_count <= BSP_CFG_SUBWAITTIME; w_count++)
{
    BSP_NOP();
}
```

However, since the actual number of cycles will differ according to factors such as the optimization option, you will need to specify a setting that matches your environment.

3.2.9 Option Bytes

You can select the behavior after a reset by setting option bytes. For example, you can specify settings for the watchdog timer and voltage detection circuit.

The option byte setting values are defined *r_bsp_config.h* when using the IAR environment. When using another environment, specify these settings in the project properties.

Table 3.9 Option Byte Definitions

Definition	Value	Description
BSP_CFG_OPTBYTE0_VALUE BSP_CFG_OPTBYTE1_VALUE BSP_CFG_OPTBYTE2_VALUE BSP_CFG_OPTBYTE3_VALUE BSP_CFG_OPTBYTE4_VALUE	Option byte value	Specifies the setting value of the corresponding option byte. These macro definitions are used by the IAR environment only. For the Renesas compiler or LLVM environment, specify these settings in the compile options.

3.2.10 Security ID Codes for On-Chip Debugging

You can protect against third parties reading the contents memory by setting Security ID Codes for On-Chip Debugging.

The Security ID Codes for On-Chip Debugging setting values are defined *r_bsp_config.h* when using the IAR environment. When using another environment, specify these settings in the project properties.

Table 3.10 Security ID Codes for On-Chip Debugging Definitions

Definition	Value	Description
BSP_CFG_SECUID0_VALUE BSP_CFG_SECUID1_VALUE BSP_CFG_SECUID2_VALUE BSP_CFG_SECUID3_VALUE BSP_CFG_SECUID4_VALUE BSP_CFG_SECUID5_VALUE BSP_CFG_SECUID6_VALUE BSP_CFG_SECUID7_VALUE BSP_CFG_SECUID8_VALUE BSP_CFG_SECUID9_VALUE BSP_CFG_SECUIDA_VALUE BSP_CFG_SECUIDB_VALUE BSP_CFG_SECUIDC_VALUE BSP_CFG_SECUIDD_VALUE BSP_CFG_SECUIDE_VALUE BSP_CFG_SECUIDF_VALUE	Security ID Codes for On-Chip Debugging value	Specifies the setting value of the corresponding Security ID Codes for On-Chip Debugging. These macro definitions are used by the IAR environment only. For the Renesas compiler or LLVM environment, specify these settings in the compile options.

3.2.11 Startup Disable

Table 3.11 Startup Disable Definitions

Definition	Value	Description
BSP_CFG_STARTUP_DISABLE	0 = BSP startup enabled 1 = BSP startup disabled	Defines whether initial clock setting processing is enabled or disabled. When “disabled” is selected, initial clock setting processing is disabled. To disable startup entirely, manually delete the startup assembler code and add your own startup processing.

3.2.12 Smart Configurator

Table 3.12 Smart Configurator Definitions

Definition	Value	Description
BSP_CFG_CONFIGURATOR_SELECT	0 = Smart Configurator not used 1 = Smart Configurator used	Defines whether or not Smart Configurator is used in the current project. When BSP_CFG_CONFIGURATOR_SELECT = 1, the Smart Configurator initialization function is called.
BSP_CFG_CONFIGURATOR_VERSION	See comments above #define in *r_bsp_config.h*.	Defines the version of Smart Configurator you are using.

3.2.13 API Functions disable Usage

Table 3.13 API Functions disable Usage Definitions

Definition	Value	Description
BSP_CFG_CLOCK_OPERATION_API_FUNCTIONS_DISABLE	0 = API Functions enable 1 = API Functions disable	Defines whether API Functions(R_BSP_StartClock, R_BSP_StopClock) is disabled. When BSP_CFG_CLOCK_OPERATION_API_FUNCTIONS_DISAB LE = 1, cannot use API Functions, but can reduce the memory size.
BSP_CFG_GET_FREQ_API_FUNCTIONS_DISABLE		Defines whether API Functions(R_BSP_GetFclkFreqHz) is disabled. When BSP_CFG_GET_FREQ_API_FUNCTIONS_DISAB LE = 1, cannot use API Functions, but can reduce the memory size.
BSP_CFG_SET_CLOCK_SOURCE_API_FUNCTIONS_DISABLE		Defines whether API Functions(R_BSP_SetClockSource) is disabled. When BSP_CFG_SET_CLOCK_SOURCE_API_FUNCTIONS_DISAB LE = 1, cannot use API Functions, but can reduce the memory size.
BSP_CFG_CHANGE_CLOCK_SETTING_API_FUNCTIONS_DISABLE		Defines whether API Functions(R_BSP_ChangeClockSetting) is disabled. When BSP_CFG_CHANGE_CLOCK_SETTING_API_FUNCTIONS_DISAB LE = 1, cannot use API Functions, but can reduce the memory size.
BSP_CFG_SOFTWARE_DELAY_API_FUNCTIONS_DISABLE		Defines whether API Functions(R_BSP_SoftwareDelay) is disabled. When BSP_CFG_SOFTWARE_DELAY_API_FUNCTIONS_DISAB LE = 1, cannot use API Functions, but can reduce the memory size.

3.2.14 Parameter check Usage

Table 3.14 Parameter check Usage Definitions

Definition	Value	Description
BSP_CFG_PARAM_CHECKING_ENABLE	0 = Parameter check is invalid 1 = Parameter check is valid	Defines whether parameter check is enabled. Returns an error for incorrect setting when switching fCLK source.

3.2.15 Callback Function at Warm Start

Table 3.15 Warm Start Callback Function Definitions

Definition	Value	Description
BSP_CFG_USER_WARM_START_CALLBACK_PRE_INITC_ENABLED	0 = User function is not called before C runtime environment is initialized 1 = User function is called before C runtime environment is initialized	Defines whether or not a user function is called before the C runtime environment is initialized.
BSP_CFG_USER_WARM_START_PRE_C_FUNCTION	Function called before C runtime environment is initialized	Defines the user function called before the C runtime environment is initialized.
BSP_CFG_USER_WARM_START_CALLBACK_POST_INITC_ENABLED	0 = User function is not called after C runtime environment is initialized 1 = User function is called after C runtime environment is initialized	Defines whether or not a user function is called after the C runtime environment is initialized.
BSP_CFG_USER_WARM_START_POST_C_FUNCTION	Function called after C runtime environment is initialized	Defines the user function called after the C runtime environment is initialized.

3.2.16 Watchdog timer refresh

Table 3.16 Watchdog timer refresh Definitions

Definition	Value	Description
BSP_CFG_WDT_REFRESH_ENABLE	0 = WDT operation disabled. 1 = WDT operation enabled. Window Open Period of Watchdog timer is 100% 2 = WDT operation enabled. Window Open Period of Watchdog timer is 50%. 3 = WDT operation enabled. Window Open Period of Watchdog timer is 75%.	Defines how to use the watchdog timer. Please also set this config as the same setting in Watchdog Timer config.
BSP_CFG_USER_WDT_REFRESH_INIT_FUNCTION	Function to set the interval interrupt of the watchdog timer.	Defines the function to be called when calling the user function before setting the clock.
BSP_CFG_USER_WDT_REFRESH_SETTING_FUNCTION	Function to set the refresh permission flag of the watchdog timer.	Defines a function that sets a flag that allows the watchdog timer to refresh while waiting for clock oscillation to stabilize.

4. API Information

The driver API conforms to Renesas API naming conventions.

4.1 Hardware Requirements

Not applicable.

4.2 Hardware Resource Requirements

Not applicable.

4.3 Software Requirements

None

4.4 Limitations

4.4.1 IAR Compiler Limitations

When using the IAR compiler, use *r_bsp_config.h* to make option byte settings.

4.4.2 Watchdog Timer Refresh Limitations

When the window open period of the watchdog timer is set to 50% or 75%, the refresh timing assumes an interval interrupt.
Do not refresh at any timing other than interval interrupts.

4.5 Supported Toolchains

The operation of this SIS module has been confirmed with the toolchains listed in 7.1, Confirmed Operating Environment.

4.6 Interrupt Vectors Used

This SIS module does not use interrupt vectors.

4.7 Header Files

All API calls are included by incorporating the file *platform.h*, which is supplied with the driver's project code.

4.8 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*.

4.9 API Typedef

4.9.1 Clock Resource

This typedef defines commands that can be used with the R_BSP_StartClock(), R_BSP_StopClock(),R_BSP_SetClockSource() and R_BSP_ChangeClockSetting() functions.

Available resources vary from device to device.

See the user's manual or r_bsp_common.h.

```
/* clock mode */
typedef enum
{
    HIOCLK,     // High-speed on-chip oscillator
    SYSCLK,     // High-speed system clock
    SXCLK,      // Subsystem clock
    MIOCLK,     // Middle-speed on-chip oscillator
    LOCLK,      // Low-speed on-chip oscillator
    PLLCLK,     // PLL clock
    ADCLK       // A/D conversion clock
} e_clock_mode_t;
```

4.9.2 Unit of Software Delay

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

```
/* Available delay units. */
typedef enum
{
    BSP_DELAY_SECS = 0,     /* Requested delay amount is in seconds. */
    BSP_DELAY_MILLISECS,    /* Requested delay amount is in milliseconds. */
    BSP_DELAY_MICROSECS     /* Requested delay amount is in microseconds. */
} e_bsp_delay_units_t;
```

4.10 Return Values

4.10.1 Error Codes

This typedef defines the error codes that can be returned by the R_BSP_StartClock(), R_BSP_StopClock(), R_BSP_SetClockSource() and R_BSP_ChangeClockSetting() functions.

```
/* Error identification */
typedef enum
{
    /* Refer to table below for members. */
} e_bsp_err_t;
```

Member	Description
BSP_OK	Success.
BSP_ARG_ERROR	An invalid argument was input.
BSP_ERROR1	The specified clock is not oscillating or stopping. The error occurrence conditions differ depending on the function.
BSP_ERROR2	When switching between clock resources, a clock resource that is not oscillating may have been switched to.
BSP_ERROR3	An unsupported state transition was specified. Refer to the user's manual.

4.11 Code Size

The sizes od ROM, RAM and maximum stack usage associated with this module are listed below. Information is listed for a single representative device of the RL78/G2x Series, RL78/F2x Series respectively.

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

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

Module revision: r_bsp v1.30

Compiler version: Renesas Electronics C Compiler Package for RL78 Family V1.11.00
LLVM C/C++ Compiler for Renesas RL78 10.0.0.202203
IAR C/C++ Compiler for Renesas RL78 version 4.21.3

Configuration options: Default settings

ROM, RAM, and Stack Code Sizes (RL78/G23)

Compiler	API function *1	Clock setting *2	ROM	RAM	STACK
Renesas compiler	Disable	Default	291	0	16
		All enable	363	0	16
	Enable	Default	1568	0	62
		All enable	1670	0	62
LLVM compiler *3	Disable	Default	495	0	30
		All enable	774	0	74
	Enable	Default	4142	0	286
		All enable	4531	0	286
IAR compiler	Disable	Default	233	0	32
		All enable	319	0	32
	Enable	Default	1684	0	66
		All enable	1793	0	66

ROM, RAM, and Stack Code Sizes (RL78/F24)

Compiler	API function *1	Clock setting *2	ROM	RAM	STACK
Renesas compiler	Disable	Default	319	0	12
		All enable	462	0	12
	Enable	Default	2093	0	62
		All enable	2335	0	62
LLVM compiler *3	Disable	Default	446	0	6
		All enable	916	0	72
	Enable	Default	5186	0	286
		All enable	5917	0	286
IAR compiler	Disable	Default	254	0	32
		All enable	435	0	32
	Enable	Default	2344	0	66
		All enable	2631	0	66

Note 1:
Use macro definition BSP_CFG_XXXX_API_FUNCTIONS_DISABLE in r_bsp_config.h to enable / disable. The above measurement results are the values when all macro definitions are enabled or disabled.

Note 2:
The default is the initial value of Smart Configurator.
Only valid for high-speed on-chip oscillator clock.
Note 3:
If measure the stack size using the LLVM compiler, add “-fstack-size-section” to the Compiler options.

4.12 “for,” “while,” and “do while” Statements

This module uses “for” and “do while” statements (loop processing) for wait processing to allow register values to take effect, for example. These instances of loop processing are indicated by the comment keyword “WAIT_LOOP.” Therefore, if you wish to incorporate fail-safe processing into the instances of loop processing, you can locate them in the source code by searching for the keyword “WAIT_LOOP.”

A code sample is shown below:

```
for statement:
HIOSTOP = 0;
/* WAIT_LOOP */
for (w_count = 0U; w_count <= BSP_CFG_FIHWAITTIME; w_count++)
{
    BSP_NOP();
}

do while statement:
MSTOP = 0;
/* WAIT_LOOP */
do{
    tmp_stab_wait = OSTC;
    tmp_stab_wait &= STAB_WAIT;
}while(tmp_stab_wait != STAB_WAIT);
```

5. API Functions

5.1 Overview

The module uses the following functions:

Function	Description
R_BSP_StartClock	Starts oscillation of the specified clock.
R_BSP_StopClock	Stops oscillation of the specified clock.
R_BSP_GetFclkFreqHz	Returns the CPU and peripheral hardware clock frequency.
R_BSP_SetClockSource	Changes the clock source of the CPU and peripheral hardware clock to the specified clock.
R_BSP_ChangeClockSetting	Changes the specified clock setting.
R_BSP_SoftwareDelay	Delays the specified duration.
BSP_DISABLE_INTERRUPT	Disables acceptance of all maskable interrupts. This is a macro function.
BSP_ENABLE_INTERRUPT	Enables acceptance of all maskable interrupts. This is a macro function.
BSP_NOP	Executes a NOP instruction. This is a macro function.

5.2 R_BSP_StartClock()

This function starts oscillation of the specified clock.

Format

```
e_bsp_err_t R_BSP_StartClock(e_clock_mode_t mode);
```

Parameters

mode

Specifies the clock on which oscillation will start (see 4.9.1).

Return Values

BSP_OK	/* Specified clock is oscillating correctly. */
BSP_ARG_ERROR	/* An invalid argument was input. */

Properties

Prototyped in *r_bsp_common.h*.

Description

This function starts oscillation of the specified clock.

In order to use this function to start oscillation on the high-speed system clock or subsystem clock, it is necessary to make the correct settings in the clock operating mode control register (CMC).

For example, even if the high-speed system clock is entered as an argument for this function, the high-speed system clock will not oscillate if EXCLK/OSCSEL is specified as the port.

The CMC register can only be read once after a reset, so make sure to enable it in the initial settings if you plan to use the high-speed system clock or subsystem clock.

Example

```
e_bsp_err_t err;

/* Start High-speed on-chip oscillator */
err = R_BSP_StartClock(HIOCLK);

if (err != BSP_OK)
{
    /* NG processing */
}
```

Special Note:

This function is only available if the macro definition (BSP_CFG_CLOCK_OPERATION_API_FUNCTIONS_DISABLE) is set to 0.

5.3 R_BSP_StopClock()

This function stops oscillation of the specified clock. However, operation cannot be guaranteed if oscillation of a clock used as the CPU and peripheral hardware clock is stopped.

Format

```
e_bsp_err_t R_BSP_StopClock(e_clock_mode_t mode);
```

Parameters

mode

Specifies the clock on which oscillation will stop (see 4.9.1).

Return Values

BSP_OK	/* Oscillation-stop processing performed for specified clock. */
BSP_ARG_ERROR	/* An invalid argument was input. */

Properties

Prototyped in *r_bsp_common.h*.

Description

This function stops oscillation of the specified clock.

The function does not do error checking for the specified clock, so operation cannot be guaranteed if oscillation of a clock used as the CPU and peripheral hardware clock is stopped.

Example

```
e_bsp_err_t err;

/* Stop High-speed on-chip oscillator */
err = R_BSP_StopClock(HIOCLK);

if (err != BSP_OK)
{
    /* NG processing */
}
```

Special Note:

This function is only available if the macro definition (BSP_CFG_CLOCK_OPERATION_API_FUNCTIONS_DISABLE) is set to 0.

5.4 R_BSP_SetClockSource()

This function changes the clock resource supplied to the CPU and peripheral hardware clock.

In order to change the clock resource to the high-speed system clock or subsystem clock, the same clock must be enabled in the initial settings.

The clock operating mode control register (CMC), which controls the same clock, can only be read once after a reset.

As a result, it cannot be enabled during operation if it was disabled in the initial settings.

Format

```
e_bsp_err_t R_BSP_SetClockSource(e_clock_mode_t mode);
```

Parameters

mode

Specifies the clock resource to be supplied to the CPU and peripheral hardware clock (see 4.9.1).

Return Values

BSP_OK	/* The CPU and peripheral hardware clock was switched to the specified clock. */
BSP_ERROR1	/* The specified clock is not oscillating. */
BSP_ERROR2	/* A state transition was specified in which, when switching the resource of the CPU and peripheral hardware clock, a clock resource that is not oscillating may have been switched to. */
BSP_ERROR3	/* An unsupported state transition was specified. */
BSP_ARG_ERROR	/* An invalid argument was input. */

Properties

Prototyped in *r_bsp_common.h*.

Description

This function changes the clock resource supplied to the CPU and peripheral hardware clock.

Example

```
e_bsp_err_t err;

/* Start clock operation (HIOCLK) */
err = R_BSP_StartClock(HIOCLK);

if(err != BSP_OK)
{
    /* NG processing */
}
/* Change clock source */
err = R_BSP_SetClockSource(HIOCLK);

if (err != BSP_OK)
{
    /* NG processing */
}
```

Special Note:
This function is only available if the macro definition (BSP_CFG_SET_CLOCK_SOURCE_API_FUNCTIONS_DISABLE) is set to 0.

When switching the clock, check the precautions in the user's manual before using.

5.5 R_BSP_GetFclkFreqHz()

This function returns the frequency of the CPU and peripheral hardware clock.

Format
uint32_t R_BSP_GetFclkFreqHz(void);

Parameters
None

Return Values
Frequency of CPU and peripheral hardware clock

Properties
Prototyped in *r_bsp_common.h*.

Description
This function returns the frequency of the CPU and peripheral hardware clock. For example, there might be a setting in *r_bsp_config.h* to specify 20 MHz as the frequency of the CPU and peripheral hardware clock. In this case, if you changed the frequency of the CPU and peripheral hardware clock to 5 MHz after the r_bsp had finished making clock settings, the function's return value would be "5000000."

Example
```
uint32_t fclk_freq;

fclk_freq = R_BSP_GetFclkFreqHz();
```

Special Note:
This function is only available if the macro definition (BSP_CFG_GET_FREQ_API_FUNCTIONS_DISABLE) is set to 0.

5.6 R_BSP_ChangeClockSetting()

This function changes the clock setting.

The setting value is specified by setting the array pointer as an argument.

Since the setting values stored in the array differ depending on the device and clock resource, set them referring to the following parameters.

Format

e_bsp_err_t R_BSP_ChangeClockSetting(e_clock_mode_t mode, uint8_t * set_values);

Parameters

mode

Specify the clock resource for which change the setting (see 4.9.1).

set_values

Specify the setting value to be changed (see below).

(RL78/G23)

When HIOCLK is specified for mode

set_values[0] : See BSP_CFG_HOCO_DIVIDE comments.

When MIOCLK is specified for mode

set_values[0] : See BSP_CFG_MOCO_DIVIDE comments.

When SYSCLK is specified for mode

set_values[0] : See BSP_CFG_MOSC_DIVIDE comments.

(RL78/F23, RL78/F24)

When HIOCLK is specified for mode

set_values[0] : See BSP_CFG_HOCO_DIVIDE comments.

set_values[1] : See BSP_CFG_FMP_DIVIDE comments.

When SYSCLK is specified for mode

set_values[0] : See BSP_CFG_FMP_DIVIDE comments.

When PLLCLK is specified for mode

set_values[0] : 0 When the frequency of fPLL is 40 MHz or less.

1 When the frequency of fPLL is faster than 40 MHz.

set_values[1] : See BSP_CFG_LOCKUP_WAIT_COUNT_SEL comments.

set_values[2] : See BSP_CFG_FMAIN_DIVIDE comments.

set_values[3] : See BSP_CFG_PLL_DIVIDE comments.

set_values[4] : See BSP_CFG_PLL_MULTI comments.

set_values[5] : BSP_CFG_PLLWAITTIME comments.

set_values[6] : BSP_CFG_FMP_DIVIDE comments.

When ADCLK is specified for mode

set_values[0] : See BSP_CFG_ADCLK_DIVIDE comments.

(RL78/G23)

When HIOCLK is specified for mode

set_values[0] : See BSP_CFG_HOCO_DIVIDE comments.

Return Values

BSP_OK	/* The specified clock setting was changed. */
BSP_ERROR1	/* The specified clock is oscillating. */
BSP_ARG_ERROR	/* An invalid argument was input. */

Properties

Prototyped in *r_bsp_common.h*.

Description

This function changes the clock setting.

Example

```
e_bsp_err_t err;
uint8_t     set_values[2];

set_values[0] = 2U;
set_values[1] = 3U;

/* Stop clock(HIOCLK) */
err = R_BSP_StopClock(HIOCLK);

/* Change clock setting(HIOCLK) */
err = R_BSP_ChangeClockSetting(HIOCLK, set_values);

if (err != BSP_OK)
{
    /* NG processing */
}

/* Start clock(HIOCLK) */
err = R_BSP_StartClock(HIOCLK);
```

Special Note:

This function is only available if the macro definition (BSP_CFG_CHANGE_CLOCK_SETTING_API_FUNCTIONS_DISABLE) is set to 0.

When changing the clock setting, check the precautions in the user's manual before using.

5.7 R_BSP_SoftwareDelay()

Delay the specified duration in units and return.

Format

```
e_bsp_err_t R_BSP_SoftwareDelay(uint32_t delay, e_bsp_delay_units_t units);
```

Parameters

delay

The number of ‘units’ to delay.

units

The ‘base’ for the units specified. See Section4.9.2.

Return Values

BSP_OK	/* BSP_OK if delay executed. */
BSP_ ERROR1	/* BSP_ERROR1 if delay/units combination resulted in overflow/underflow. */

Properties

Prototyped in *r_bsp_common.h*.

Description

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

The actual delay time will take overhead into account. 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.

Example

```
e_bsp_err_t ret;

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

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

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

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

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

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

Special Note:

This function is only available if the macro definition (BSP_CFG_SOFTWARE_DELAY_API_FUNCTIONS_DISABLE, BSP_CFG_GET_GREQ_API_FUNCTIONS_DISABLE) is set to 0.

Do not use this function with a combination of RL78/G15 device and LLVM compiler.

6. Project Setup

This section describes how to add the r_bsp to your project.

6.1 Adding the SIS Module

This module must be added to each project in which it is used. Renesas recommends the method using Smart Configurator described in (1) or (3) below.

(1) Adding the SIS module using Smart Configurator in e^2 studio
You can add the SIS module to your project automatically by using Smart Configurator in e^2 studio. Refer to the application note RL78 Smart Configurator User's Guide: e^2 studio (R20AN0579) for details.

(2) Adding the SIS module using Smart Configurator in CS+
You can add the SIS module to your project automatically by using the standalone version of Smart Configurator in CS+. Refer to the application note RL78 Smart Configurator User's Guide: CS+ (R20AN0580) for details.

(3) Adding the SIS module using Smart Configurator in IAREW
You can add the SIS module to your project automatically by using the standalone version of Smart Configurator. Refer to the application note RL78 Smart Configurator User's Guide: IAREW (R20AN0581) for details.

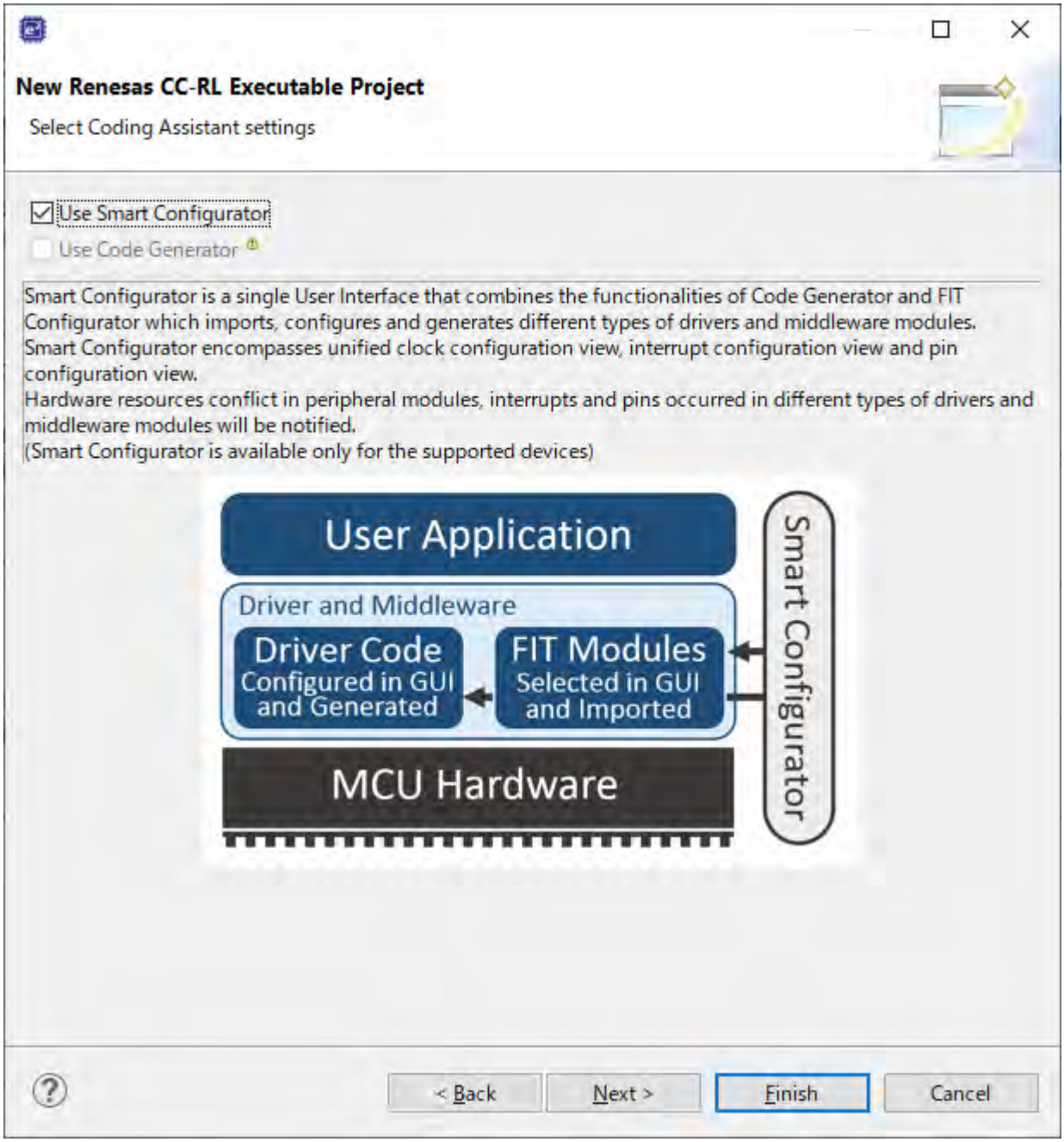
6.2 Adding the SIS Module to a Project in e² studio

How to add a the SIS module to a project in e² studio is described below.

6.2.1 Adding the SIS Module Using Smart Configurator in e² studio

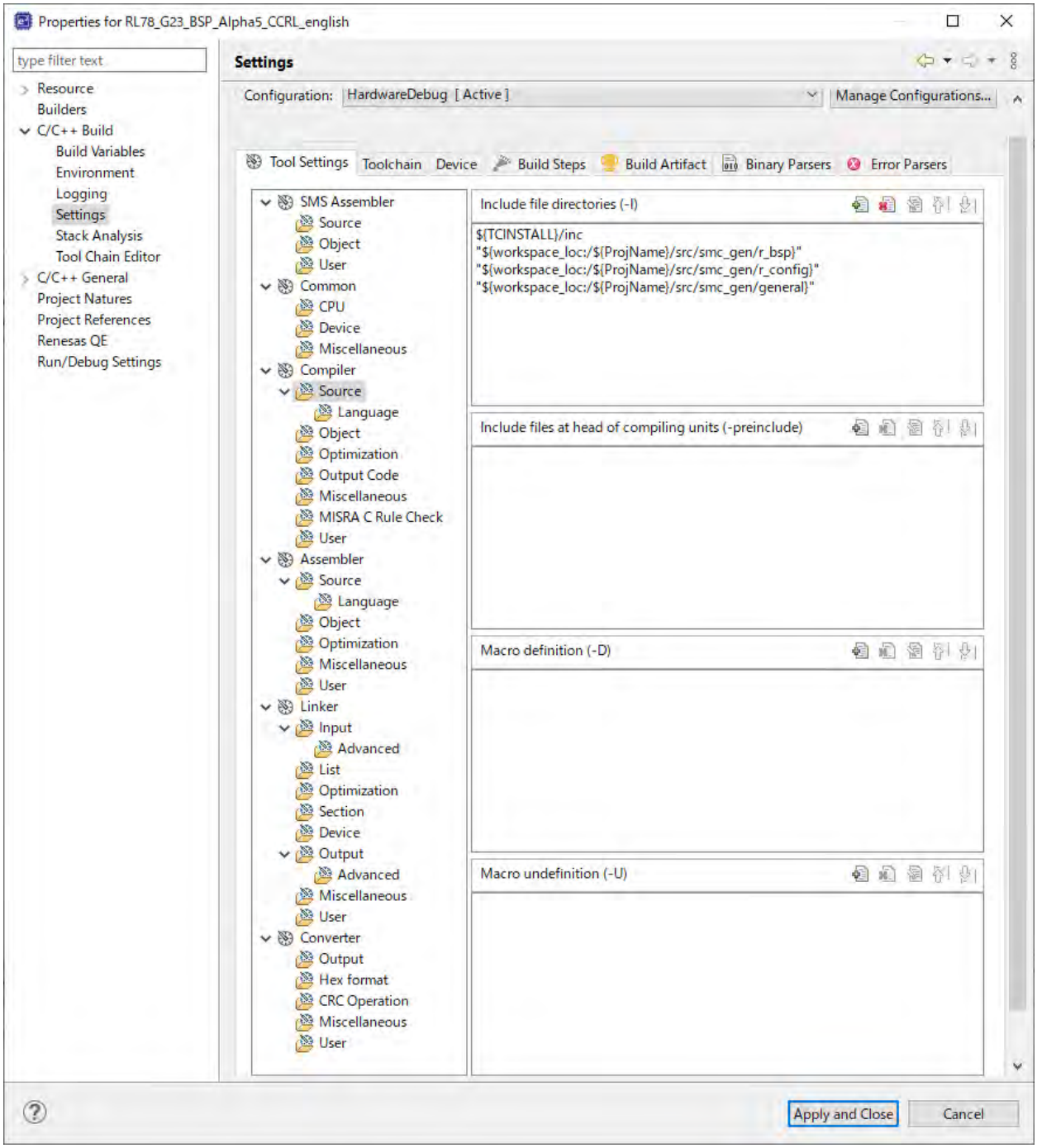
This explanation uses e² studio (2021-01).

1. Create a new project in e² studio.
   When creating your project, check the box next to “Use Smart Configurator” to launch Smart Configurator.



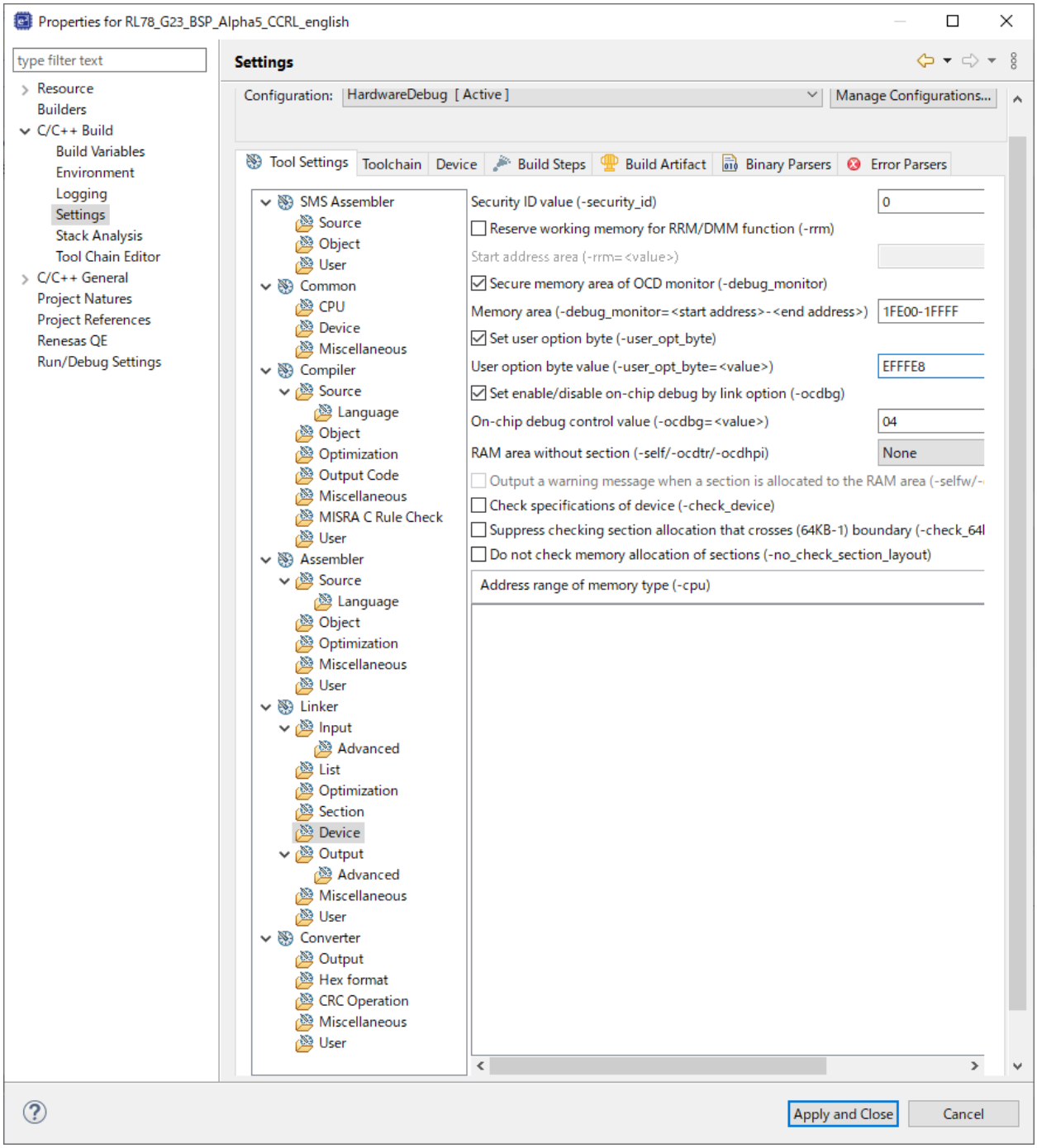
2. Follow the procedure described in 6.1, Adding the SIS Module, to add the SIS module to your project in e² studio.
3. Right-click the project and click “Properties.”
4. On the Tool Settings tab, select Compiler → Source.

5. SIS module include paths generated by Smart Configurator have been specified.



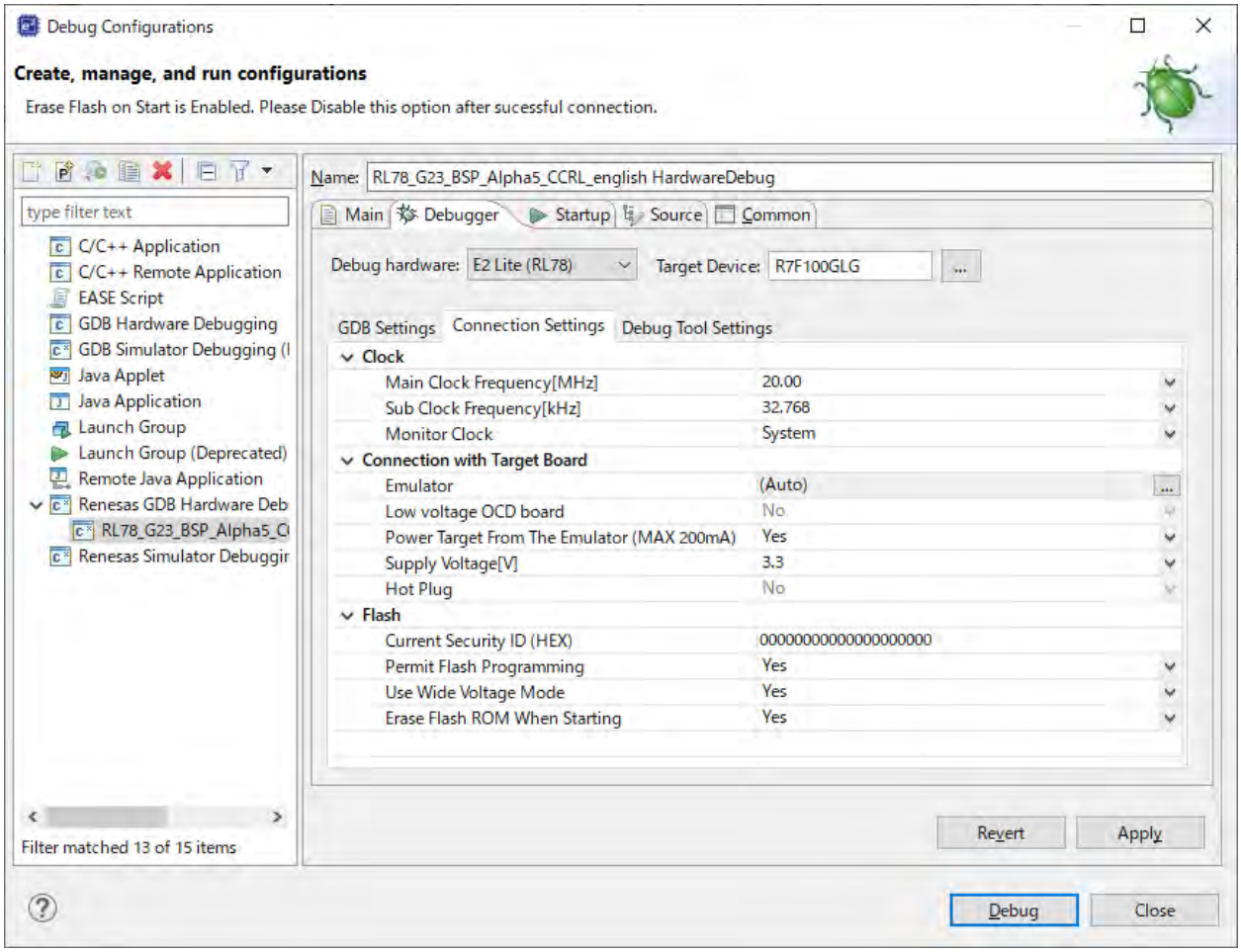
6. On the Tool Settings tab, select Linker → Device.

7. Enter settings for the option bytes area.



8. Right-click the project and click “Build Project.”
9. Right-click the project and click “Debug” → “Configure Debugger.”
10. Click “Renesas GDB Hardware Debugging” → “Project Name Hardware Debug.”
11. On the Debugger tab, set “Debug hardware:” to “E2 Lite (RL78).”
12. On the Tool Connection Setting tab, set the main clock frequency and subclock frequency.

13. On the Connection Settings tab, set “Power Target From The Emulator (MAX 200mA)” to “Yes.”



7. Appendix

7.1 Confirmed Operating Environment

The environment in which the operation of the module has been confirmed is shown below.

Table 7.1 Confirmed Operating Environment (Rev. 1.00)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio (2021-01) IAR Systems IAR Embedded Workbench for Renesas RL78 4.20.1
C compiler	Renesas Electronics C compiler for R78 Family V.1.09.0 LLVM for Renesas RL78 Build Support 0.1.0.v20200629-1555
Module revision	Rev.1.00
Board used	RL78/G23-64p Fast Prototyping Board (Product type: RTK7RLG230CLG000BJ)

Table 7.2 Confirmed Operating Environment (Rev. 1.10)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio (2021-04) IAR Systems IAR Embedded Workbench for Renesas RL78 4.20.1
C compiler	Renesas Electronics C compiler for R78 Family V.1.10.0 GCC & LLVM for Renesas RL78 Build Support 21.4.0.v20210325-1643
Module revision	Rev.1.10
Board used	RL78/G23-64p Fast Prototyping Board (Product type: RTK7RLG230CLG000BJ)

Table 7.3 Confirmed Operating Environment (Rev. 1.11)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio (2021-04) IAR Systems IAR Embedded Workbench for Renesas RL78 4.20.1
C compiler	Renesas Electronics C compiler for R78 Family V.1.10.0 GCC & LLVM for Renesas RL78 Build Support 21.4.0.v20210325-1643
Module revision	Rev.1.11
Board used	RL78/G23-64p Fast Prototyping Board (Product type: RTK7RLG230CLG000BJ)

Table 7.4 Confirmed Operating Environment (Rev. 1.12)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio (2021-07) IAR Systems IAR Embedded Workbench for Renesas RL78 4.21.1
C compiler	Renesas Electronics C compiler for R78 Family V.1.10.0 GCC & LLVM for Renesas RL78 Build Support 21.7.0.v20210630-0826
Module revision	Rev.1.12
Board used	RL78/G23-64p Fast Prototyping Board (Product type: RTK7RLG230CLG000BJ)

Table 7.5 Confirmed Operating Environment (Rev. 1.13)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio (2021-10) IAR Systems IAR Embedded Workbench for Renesas RL78 4.21.1
C compiler	Renesas Electronics C compiler for R78 Family V.1.10.0 GCC & LLVM for Renesas RL78 Build Support 21.7.0.v20210630-0826
Module revision	Rev.1.13
Board used	RL78/G23-64p Fast Prototyping Board (Product type: RTK7RLG230CLG000BJ)

Table 7.6 Confirmed Operating Environment (Rev. 1.20)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio (2022-01) IAR Systems IAR Embedded Workbench for Renesas RL78 4.21.3
C compiler	Renesas Electronics C/C++ compiler for R78 Family V.1.11.0 GCC & LLVM for Renesas RL78 Build Support 21.7.0.v20210630-0826
Module revision	Rev.1.20
Board used	RL78/F24 Target Board (Product type: RTK7F124FPC0 1000BJ)

Table 7.7 Confirmed Operating Environment (Rev. 1.30)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio (2022-07) IAR Systems IAR Embedded Workbench for Renesas RL78 4.21.3
C compiler	Renesas Electronics C/C++ compiler for R78 Family V.1.11.0 GCC & LLVM for Renesas RL78 Build Support 22.7.0.v20220419-1309
Module revision	Rev.1.30
Board used	RL78/G15 Target Board (Product type: RTK5RLG150C00WS1BJ)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Mar. 08, 2021	—	First edition issued
1.10	Apr. 05, 2021	—	Added support for RTOS.
		15	Added RTOS macro definition. - BSP_CFG_RTOS_USED
		19	Added Security ID Codes for On-Chip Debugging definition. - BSP_CFG_SECUID0_VALUE - BSP_CFG_SECUID1_VALUE - BSP_CFG_SECUID2_VALUE - BSP_CFG_SECUID3_VALUE - BSP_CFG_SECUID4_VALUE - BSP_CFG_SECUID5_VALUE - BSP_CFG_SECUID6_VALUE - BSP_CFG_SECUID7_VALUE - BSP_CFG_SECUID8_VALUE - BSP_CFG_SECUID9_VALUE
		30	Renamed application notes referenced when adding SIS modules - RL78 Smart Configurator User's Guide: e^2 studio (R20AN0579) - RL78 Smart Configurator User's Guide: CS+ (R20AN0580) - RL78 Smart Configurator User's Guide: IAREW (R20AN0581)
		35	Added Table 7.2 confirmed operating environment (Rev. 1.10)
1.11	May.25.21	—	Review macro definition.
		12	Removed description about platform selection by version.
		18,24	Renamed following macro definitions of oscillation stabilization wait time. - BSP_CFG_SUBWAITTIME - BSP_CFG_FIHWAITTIME - BSP_CFG_FIMWAITTIME - BSP_CFG_FILWAITTIME
		19	Renamed following macro definitions of Option Bytes. - BSP_CFG_OPTBYTE0_VALUE - BSP_CFG_OPTBYTE1_VALUE - BSP_CFG_OPTBYTE2_VALUE - BSP_CFG_OPTBYTE3_VALUE
		35	Added Table 7.3 confirmed operating environment (Rev. 1.11)
1.12	Aug.04.21	—	Added a process to call user functions in start.S Updated RL78/G23 iodefine.h(LLVM)
		35	Added Table 7.4 confirmed operating environment (Rev. 1.12)

Rev.	Date	Description	
		Page	Summary
1.13	Oct.29.21	—	Added support for FSP. Renamed iodefine.h. Added macro definition showing MCU attributes. Added version check of smart configurator. Removed C++ from Renesas Electronics compiler.
		7	Added macro definition showing MCU attributes. - BSP_CFG_FAMILY_<MCU_FAMILY> - BSP_CFG_SERIES_<MCU_SERIES> - BSP_CFG_GROUP_<MCU_GROUP>
		20	Added macro definition showing version of Smart Configurator. - BSP_CFG_CONFIGURATOR_VERSION
		36	Added Table 7.5 confirmed operating environment (Rev. 1.13)
1.20	Feb.28.22	—	Added support for RL78/F23 and RL78/F24. Added R_BSP_ChangeClockSetting function. Added RAMSAR register setting function. Supports enabling / disabling for each API function.
		13	3.2.2 Peripheral I/O redirection Register Added the following macro definition. - BSP_CFG_PIORyy (yy=00-99)
		14	3.2.4 RAM start address Added chapter and added the following macro definition. - BSP_CFG_ASM_RAMSAR_ADDRESS - BSP_CFG_ASM_RAM_GUARD_START_ADDRESS
		16-20	3.2.8 Clock Setting Added the following macro definition. - BSP_CFG_ALLOW_FSL_IN_STOPHALT - BSP_CFG_FIL_OPERATION - BSP_CFG_PLL_DIVIDE - BSP_CFG_FMAIN_DIVIDE - BSP_CFG_CAN_CLOCK_OPERATION - BSP_CFG_LIN1_CLOCK_SOURCE - BSP_CFG_LIN1_CLOCK_OPERATION - BSP_CFG_LIN0_CLOCK_SOURCE - BSP_CFG_LIN0_CLOCK_OPERATION - BSP_CFG_TRD_CLOCK_SOURCE - BSP_CFG_LOCKUP_WAIT_COUNT_SEL - BSP_CFG_PLL_MULTI - BSP_CFG_PLL_MODE - BSP_CFG_PLL_OPERATION - BSP_CFG_FPLL_HZ - BSP_CFG_FMP_DIVIDE - BSP_CFG_ADC_ENABLE - BSP_CFG_ADCLK_DIVIDE - BSP_CFG_PLLWAITTIME
		21	3.2.9 Option Bytes Added the following macro definition. - BSP_CFG_OPTBYTE4_VALUE

		21	3.2.10 Security ID Codes for On-Chip Debugging. Added the following macro definition. - BSP_CFG_SECUIDA_VALUE - BSP_ CFG_SECUIDB_VALUE - BSP_ CFG_SECUIDC_VALUE - BSP_ CFG_SECUIDD_VALUE - BSP_ CFG_SECUIDE_VALUE - BSP_ CFG_SECUIDF_VALUE
		23	3.2.13 API Functions disable Usage Added the following macro definition. - BSP_CFG_CLOCK_OPERATION_API_FUNCTIONS_ DISABLE - BSP_CFG_GET_FREQ_API_FUNCTIONS_DISABLE - BSP_CFG_SET_CLOCK_SOURCE_API_FUNCTIONS_ DISABLE - BSP_CFG_CHANGE_CLOCK_SETTING_API_ FUNCTIONS_DISABE
		26	4.9.1 Clock Resource Added R_BSP_ChangeClockSetting function to functions that use clock resource.
		27	4.11 Code Size The measurement result of the code size has been update.
		28	5.1 Overview Added R_BSP_ChangeClockSetting function.
		29,30,32	Added conditions to use in Special Note for API functions.
		33,34	Added chapter 5.6 R_BSP_ChangeClockSetting.
		41	Added Table 7.6 confirmed operating environment (Rev. 1.20)
1.30	May.31.22	—	Added support for RL78/G15. Added R_BSP_SoftwareDelay function.
		23	3.2.13 API Functions disable Usage Added the following macro definition. - BSP_CFG_SOFTWARE_DEALY_API_FUNCTIONS_ DISABLE
		26	4.9.2 Unit of Software Delay Added definition for R_BSP_SoftwareDelay function.
		29	5.1 Overview Added R_BSP_SoftwareDelay function.
		35	5.6 R_BSP_ChangeClockSetting() Added a setting that can be changed with RL78/G15 in the Parameter column.
		36,37	Added chapter 5.7 R_BSP_SoftwareDelay.
		44	Added Table 7.7 confirmed operating environment (Rev. 1.30)

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

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

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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