RISC-V MCU

Board Support Package Module Using Software Integration System

Summary

The Renesas board support package module (r_bsp) forms the foundation of any project that uses Software Integration System. 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

RISC-V MCU

Supported Compilers

- LLVM C/C++ Compiler for Renesas RISC-V
- IAR C/C++ Compiler for Renesas RISC-V
- SEGGER Compiler

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

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 be followed to configure the above items. The r_bsp is provided 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.

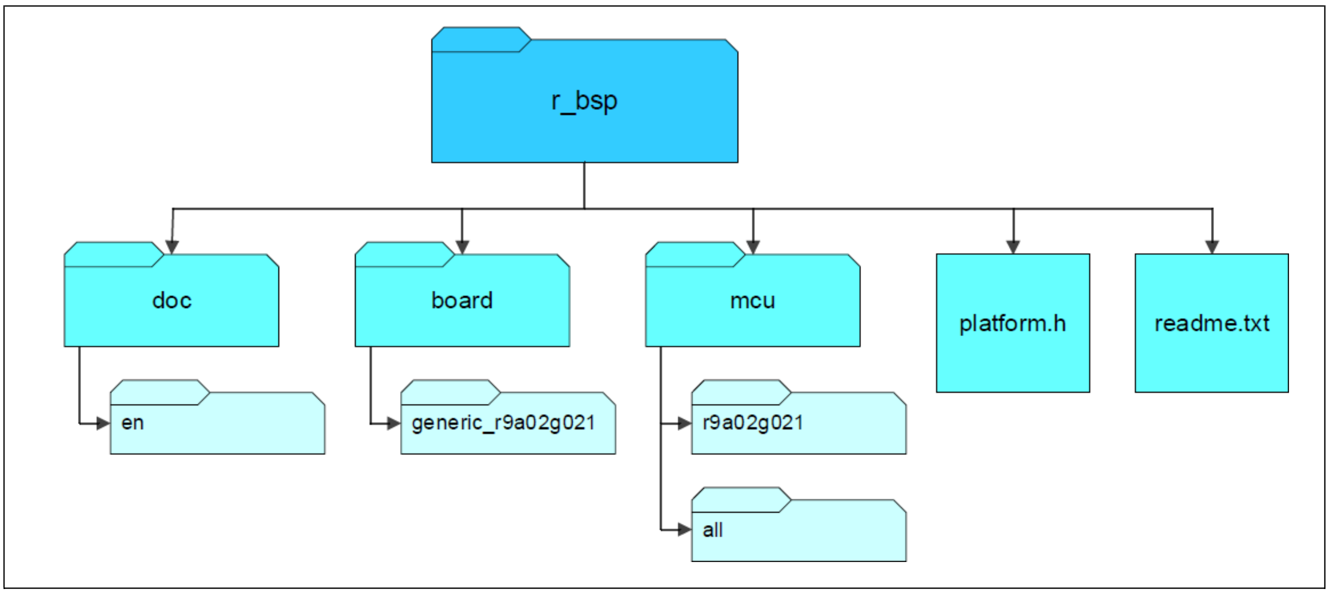
The *board* folder contains the *generic* folders.

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

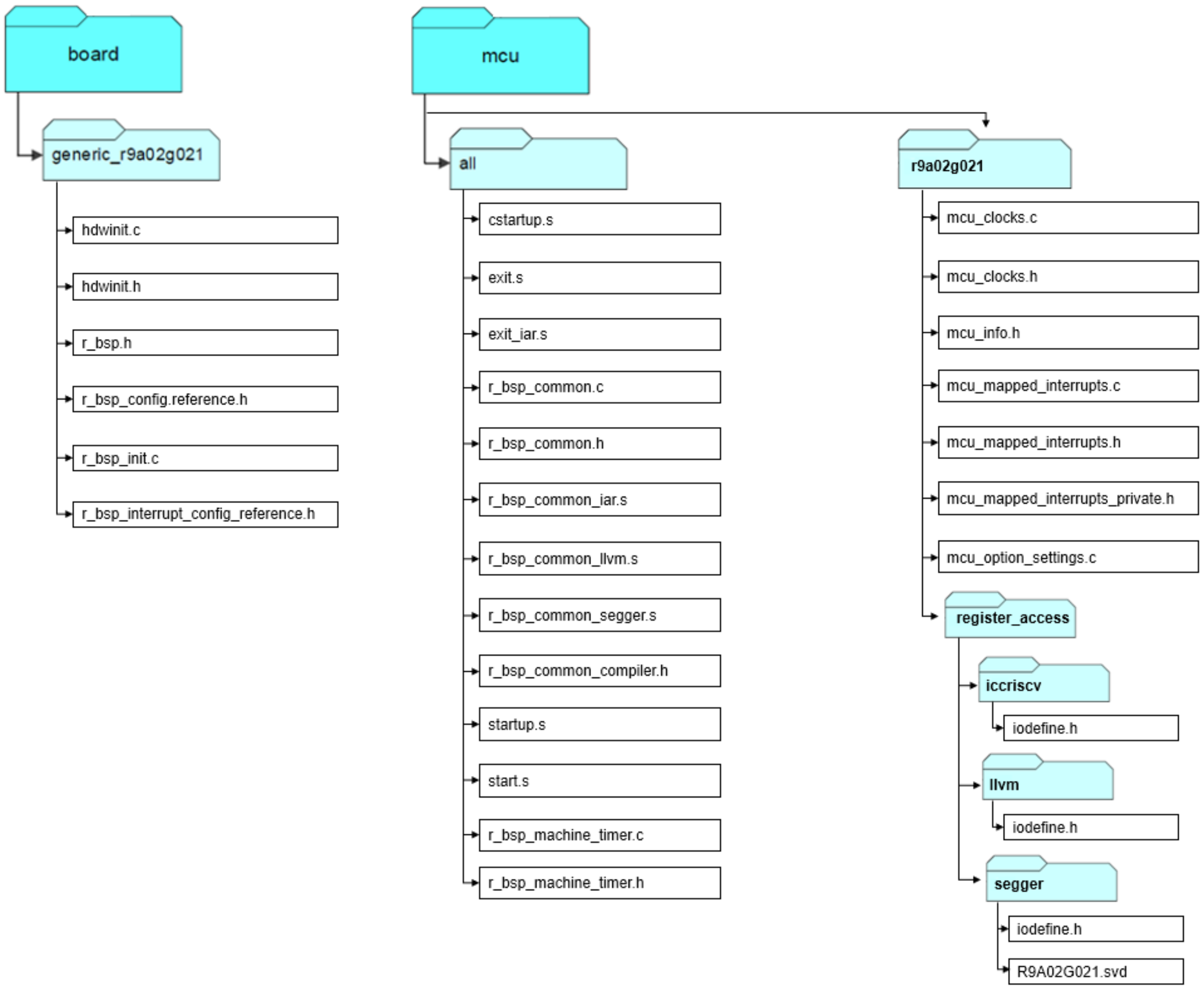


Figure 1.2 Structure of Generic Folder

2. Functionality

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

2.1 MCU Information

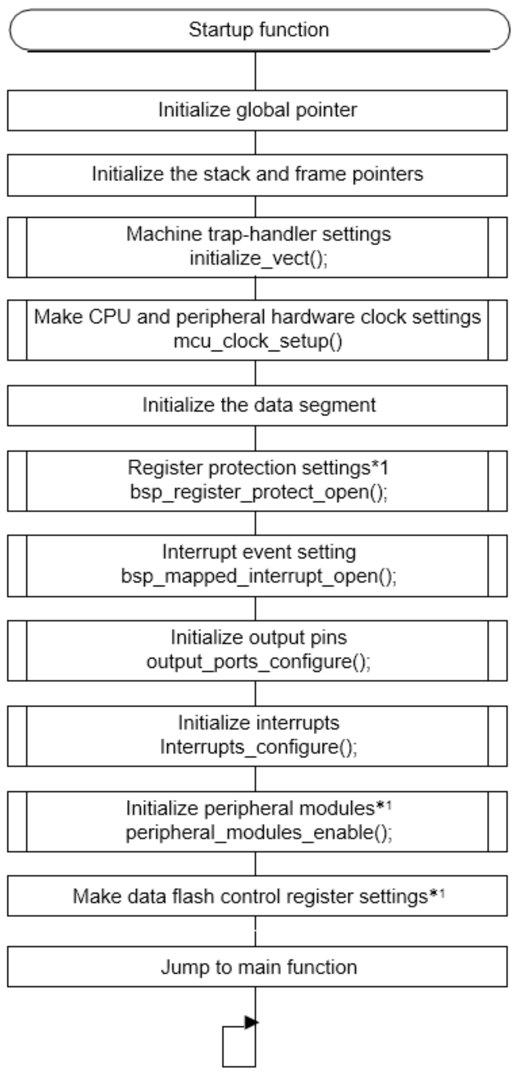
One of the main benefits of the r_bsp is that it lets you define the global system settings only once, in a single place in the project, and those settings are then shared throughout. This information is defined in the r_bsp and can then be used by the SIS modules and user code. SIS modules use this 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_RISCV_MCU	Which MCU Family this MCU belongs to.
BSP_MCU_GROUP_G021	Which MCU group this MCU belongs to.
BSP_LOCO_HZ BSP_SOSC_HZ BSP_MOCO_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

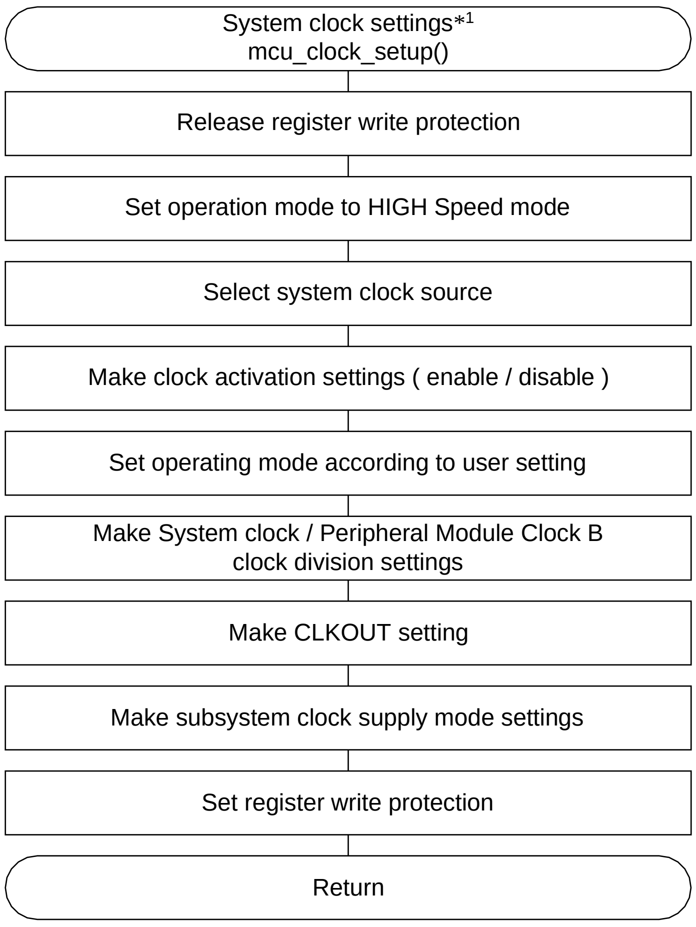
The _PowerON_Reset function is set as the reset vector when using the LLVM compiler or SEGGER compiler. The_iar_program_start function is set as the reset vector for the MCU when using the IAR compiler. The _PowerON_Reset function (the startup function when using LLVM or SEGGER compiler), or function_ 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.


Startup function
Initialize global pointer
Initialize the stack and frame pointers
Machine trap-handler settings
initialize_vect();
Make CPU and peripheral hardware clock settings
mcu_clock_setup()
Initialize the data segment
Register protection settings*1
bsp_register_protect_open();
Interrupt event setting
bsp_mapped_interrupt_open();
Initialize output pins
output_ports_configure();
Initialize interrupts
Interrupts_configure();
Initialize peripheral modules*1
peripheral_modules_enable();
Make data flash control register settings*1
Jump to main function

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

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

Figure 2.1 Flowchart of Startup Function


System clock settings*1
mcu_clock_setup()
Release register write protection
Set operation mode to HIGH Speed mode
Select system clock source
Make clock activation settings (enable / disable)
Set operating mode according to user setting
Make System clock / Peripheral Module Clock B clock division settings
Make CLKOUT setting
Make subsystem clock supply mode settings
Set register write protection
Return

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_CFG_INTERRUPT_SETTING_API_FUNCTIONS function to specifies the interrupt vector for each interrupt event.

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.

2.6 ID Code

RISC-V MCUs have an 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 *r_bsp_config.h* in the LLVM environment. In the IAR and the SEGGER environment they are specified in *mcu_option_settings.c*. For details of ID code options, refer to the Option-Setting Memory and chapters is mentioned on-chip debug mode in your MCU’s hardware manual.

2.7 Option-Setting Memory

The Option-Setting Memory are located in the flash memory of RISC-V MCUs. The Option-Setting Memory are referenced automatically after power-on or a reset, and the specified function settings are applied. Option-Setting Memory can be used to specify settings for the watchdog timer or voltage detection circuit, for example. Option-Setting Memory setting values (macro) are specified in *r_bsp_config.h,* its value is depended on setting of user in Smart Configurator. Option-Setting Memory is set in *mcu_option_settings.c* using macros that is defined in *r_bsp_config.h*.

2.8 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.

2.9 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:

- LLVM compiler: start.s
- IAR compiler: cstartup.s
- SEGGER compiler: startup.s

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

2.9.1 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.

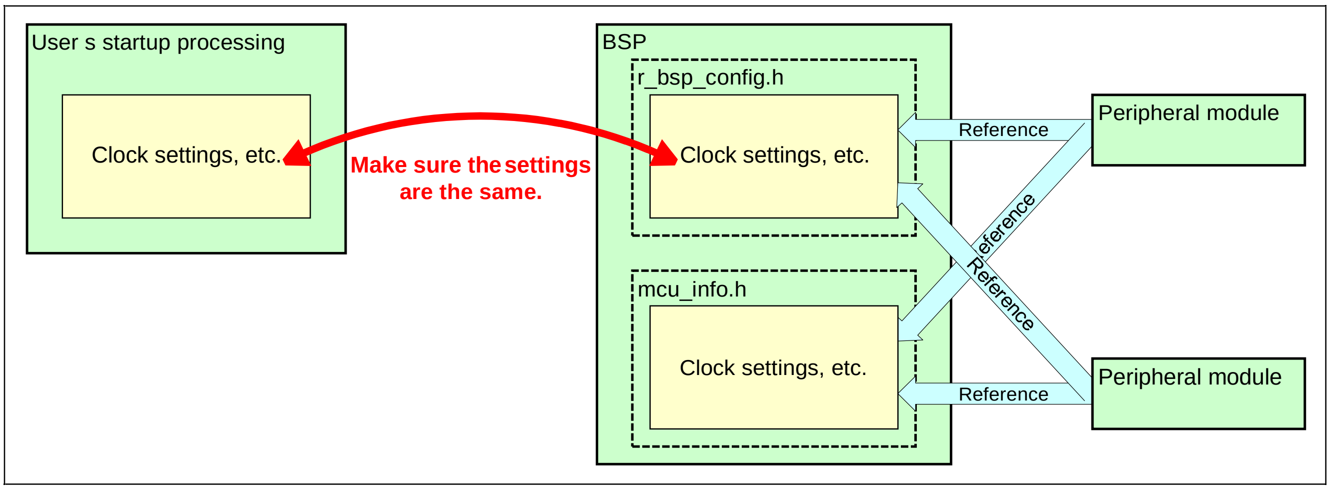


Figure 2.3 Configuration File Settings

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_TYPE	See comments above #define in *r_bsp_config.h*.	Defines the device type.
BSP_CFG_MCU_PART_PACKAGE		Defines the package type.

3.2.2 Data Flash Access Restriction

RISC-V 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 3.2 Data Flash Access Restriction Definitions

Definition	Value	Description
BSP_CFG_DATA_FLASH_ACCESS_ENABLE	0 : Access to the data flash memory area is disabled. 1 : Access to the data flash memory area is enabled.	Data flash memory area access control Data flash control register(DFLCTL) DFLEN

3.2.3 Prefetch Buffer Enable Register

RISC-V MCUs are provided with functionality to enable or disable access to the prefetch buffer. Flash memory provides an instruction prefetch function to accelerate code execution. The prefetch function can be used by enabling the prefetch buffer.

Table 3.3 Prefetch buffer enable Definitions

Definition	Value	Description
BSP_CFG_PFB_ENABLE	0 : Prefetch buffer is disabled. 1 : Prefetch buffer is enabled.	Prefetch buffer enable register (PFBER)

3.2.4 Machine Timer Operation

The machine timer includes 64-bit counter, comparator, and software interrupt registers. It generates machine timer interrupts (MTIP) and machine software interrupts (MSIP) for the Core Local Interrupt Controller (CLIC). When the counter matches the comparator, an MTIP is triggered.

Table 3.4 Machine Timer Definitions

Definition	Value	Description
BSP_CFG_MTIME_CLOCK_SOURCE	0 : Machine timer clock. 1 : CPU clock.	Clock source select
BSP_CFG_MACHINE_TIMER	0 : Machine timer is disable. 1 : Machine timer is enable.	Machine Timer enable register
BSP_CFG_MTIME_COUNTER_HIGHERBITS BSP_CFG_MTIME_COUNTER_LOWERBITS	Counter value higher or lower in machine timer	Specifies the setting value of the counter.
BSP_CFG_MTIME_COMPARE_HIGHERBITS BSP_CFG_MTIME_COMPARE_LOWERBITS	Compare value higher or lower in machine timer	Specifies the setting value of the compare.
BSP_CFG_MTIP_PRIORITY BSP_CFG_MSIP_PRIORITY	0 : 0xFF: Level 0 (high) 1 : 0xEF: Level 1 2 : 0xDF: Level 2 3 : 0xCF: Level 3 4 : 0xBF: Level 4 5 : 0xAF: Level 5 6 : 0x9F: Level 6 7 : 0x8F: Level 7 8 : 0x7F: Level 8 9 : 0x6F: Level 9 10 : 0x5F: Level 10 11 : 0x4F: Level 11 12 : 0x3F: Level 12 13 : 0x2F: Level 13 14 : 0x1F: Level 14 15 : 0x0F: Level 15 (low)	Machine timer interrupt priority (Level 0 to Level15)
BSP_CFG_SOFTWARE_INTERRUPT	0 : Disable. 1 : Enable.	Enable machine software interrupt

3.2.5 Clock Settings

The available clocks vary among RISC-V 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 3.5 Clock Setting Definitions

Definition	Value	Description
BSP_CFG_ICLK_DIV	0 : x 1/1 1 : x 1/2 2 : x 1/4 3 : x 1/8 4 : x 1/16 5 : x 1/32 6 : x 1/64	System Clock Division Control Register (SCKDIVCR) System Clock (ICLK) Select (ICK[2:0])
BSP_CFG_PCLKB_DIV	0 : x 1/1 1 : x 1/2 2 : x 1/4 3 : x 1/8 4 : x 1/16 5 : x 1/32 6 : x 1/64	System Clock Division Control Register (SCKDIVCR) Peripheral Module Clock B (PCLKB) Select (PCKB[2:0])
BSP_CFG_CLOCK_SOURCE_SEL	0 : HOCO 1 : MOCO 2 : LOCO 3 : External clock input (EXTAL) 4 : Sub-clock oscillator (SOSC)	System Clock Source Control Register (SCKSCR) Clock Source Select (CKSEL[2:0])
BSP_CFG_EXTCLK_OPERATION	0 : Stop 1 : Operate	External Clock Input Control Register (MOSCCR)
BSP_CFG_SUBCLK_OPERATION	0 : Stop 1 : Operate	Sub-Clock Oscillator Control Register (SOSCCR).
BSP_CFG_SUBCLK_MODE	0 : Normal Mode 1 : Low Power Mode 1 2 : Low Power Mode 2 3 : Low Power Mode 3	Sub-Clock Oscillator Mode Control Register (SOMCR)
BSP_CFG_SUBCLK_MARGIN	0 : Normal Current 1 : Lower Margin check 2 : Upper Margin check	Sub-Clock Oscillator Margin Check Register (SOMRG)
BSP_CFG_LOCO_OPERATION	0 : Stop 1 : Operate	Low-Speed On-Chip Oscillator Control Register (LOCOCR)
BSP_CFG_HOCO_OPERATION	0 : Stop 1 : Operate	High-Speed On-Chip Oscillator Control Register (HOCOCR)
BSP_CFG_MOCO_OPERATION	0 : Stop 1 : Operate	Middle-Speed On-Chip Oscillator Control Register (MOCOCR)
BSP_CFG_CLKOUT_SEL	0 : HOCO 1 : MOCO 2 : LOCO 3 : External clock input (EXTAL) 4 : Sub-clock oscillator (SOSC)	Clock Out Control Register (CKOCR) Clock Out Source Select (CKOSEL[2:0])

Definition	Value	Description
BSP_CFG_CLKOUT_DIV	0 : x 1/1 1 : x 1/2 2 : x 1/4 3 : x 1/8 4 : x 1/16 5 : x 1/32 6 : x 1/64 7 : x 1/128	Clock Out Control Register (CKOCR) Clock Output Frequency Division Ratio (CKODIV[2:0])
BSP_CFG_CLKOUT_ENABLE	0 : Disable 1 : Enable	Clock Out Control Register (CKOCR) Clock Out Enable (CKOEN)
BSP_CFG_SUBCLK_SEL	0 : Subsystem clock (SOSC) 1 : Low-speed on-chip oscillator clock (LOCO)	Subsystem Clock Supply Mode Control Register (OSMCR) Selection of the operating clock for the realtime clock, 32-bit interval timer,serial interfaces UARTA0 and UARTA1, remote control signal receiver (WUTMMCK0)
BSP_CFG_EXTAL_HZ	Frequency (Hz)	Input clock frequency in Hz (EXTAL).
BSP_CFG_EXTCLK_INPUT_JTAG_HZ	Frequency (Hz)	Input clock frequency in Hz (JTAG).
BSP_CFG_OPERATION_MODE	0: High-speed mode 1: Middle-speed mode 2: Subosc-speed mode 3: Low-speed mode	Operation Power Mode Select (OPCCR register and SOPCCR register).

3.2.6 Option-Setting Memory

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

Option-Setting Memory setting values (macro) are specified in *r_bsp_config.h,* its value is depended on setting of user in Smart Configurator. Option-Setting Memory is set in *mcu_option_settings.c* using macros that is defined in *r_bsp_config.h.*

Table 3.6 Option-Setting Memory Definitions

Definition	Value	Description
BSP_CFG_OFS0_REG_VALUE BSP_CFG_OFS1_REG_VALUE	Option-Setting Memory value	Specifies the setting value of the corresponding Option-Setting Memory.

3.2.7 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(macro) are defined *r_bsp_config.h,* its value is depended on setting of user in Smart Configurator. The Security ID Codes for On-Chip Debugging is set in *mcu_option_settings.c* using macros that is defined in *r_bsp_config.h.*

Table 3.7 Security ID Codes for On-Chip Debugging Definitions

Definition	Value	Description
BSP_CFG_OCD_SERIAL_ID_VALUE_0 BSP_CFG_OCD_SERIAL_ID_VALUE_1 BSP_CFG_OCD_SERIAL_ID_VALUE_2 BSP_CFG_OCD_SERIAL_ID_VALUE_3	ID Codes for On-Chip Debugging / Serial programing value	Specifies the setting value of the corresponding Security ID Codes for On-Chip Debugging or serial programming.

3.2.8 Startup API Functions

Table 3.8 Startup macro Definitions

Definition	Value	Description
BSP_CFG_STARTUP_FUNCTIONS	0 : Enable BSP startup program. 1 : Disable BSP startup program.(e.g. Using user startup program.)	Start up select.

3.2.9 Smart Configurator

Table 3.9 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.

3.2.10 API Functions disable Usage

Table 3.10 API Functions disable Usage Definitions

Definition	Value	Description
BSP_CFG_CLOCK_OPERATION_API_FUNCTIONS	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 = 1, cannot use API Functions, but can reduce the memory size.
BSP_CFG_CHANGE_CLOCK_SETTING_API_FUNCTIONS		Defines whether API Functions(R_BSP_ChangeClockSetting) is disabled. When BSP_CFG_CHANGE_CLOCK_SETTING_API_FUNCTIONS = 1, cannot use API Functions, but can reduce the memory size.
BSP_CFG_INTERRUPT_SETTING_API_FUNCTIONS		Defines whether API Functions(bsp_mapped_interrupt_open) is disabled. When BSP_CFG_INTERRUPT_SETTING_API_FUNCTIONS = 1, cannot use API Functions, but can reduce the memory size.

3.2.11 Parameter check Usage

Table 3.11 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 System clock source.

3.2.12 Callback Function at Warm Start

Table 3.12 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.

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 Supported Toolchains

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

4.5 Interrupt Vectors Used

This SIS module does not use interrupt vectors.

4.6 Header Files

All API calls are included by incorporating the file *platform.h*, which is supplied with the 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 API Typedef

4.8.1 Clock Resource

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

Available resources vary from device to device.

See the user's manual or *r_bsp_common.h*.

```
/* clock mode */
typedef enum
{
    HOCO,       // High-speed on-chip oscillator
    MOCO,       // Middle-speed on-chip oscillator
    LOCO,       // Low-speed on-chip oscillator
    EXTAL,      // External clock
    SOSC,       // Subsystem clock
} e_clock_mode_t;
```

4.8.2 Clock Divider

This typedef defines commands that can be used with the R_BSP_ChangeICLKClockSetting() functions.

Available setting of divider varies from device to device.

See the user's manual or *r_bsp_common.h*.

```
typedef enum
{
   DIV_BY_1,
   DIV_BY_2,
   DIV_BY_4,
   DIV_BY_8,
   DIV_BY_16,
   DIV_BY_32,
   DIV_BY_64,
   DIV_BY_128,
} e_clock_div_t;
```

4.8.3 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 = 1,              /* Requested delay amount is in seconds. */
   BSP_DELAY_MILLISECS = 1000,      /* Requested delay amount is in milliseconds. */
   BSP_DELAY_MICROSECS = 1000000    /* Requested delay amount is in microseconds. */
} e_bsp_delay_units_t;
```

4.8.4 Register Write Protection Unit

This typedef defines the types of registers that can be used by the R_BSP_RegisterProtectEnable() and R_BSP_RegisterProtectDisable() functions.typedef enum.

```
typedef enum
{
   /* PRC0
      Enables writing to the registers related to the clock generation circuit:
SCKDIVCR, SCKSCR, HOCOCR, MOCOCR, CKOCR, HOCOUTCR, LOCOCR, LPOPT, OSMCR, MOSCCR,
SOSCCR, SOMCR, SOMRG, MEMWAIT, LOCOUTCR, MOCOUTCR */
   BSP_REG_PROTECT_CGC = 0,

   /* PRC1
      Enables writing to the registers related to low power mode: SBYCR, OPCCR,
SYOCDCR, PSMCR, SNZCR, SNZEDCR0, SNZEDCR1, SNZREQCR0, SOPCCR, SYOCDCR, PSMCR */
   BSP_REG_PROTECT_LPM,

   /* PRC3
      Enables writing to the registers related to the LVD: LVD1CR1, LVD1SR, LVD2CR1,
LVD2SR, LVCMPCR, LVDLVLR, LVD1CR0, LVD2CR0 */
   BSP_REG_PROTECT_LVD,

   /* SRAM.SRAMPRCR
      Enables writing to the PARIOAD register.   */
   BSP_REG_PROTECT_SRAM,

   /* SRAM.ECCPRCR
      Enables writing to the ECCMODE, ECC1STSEN, and ECCOAD registers.   */
   BSP_REG_PROTECT_ECC,

   /* SRAM.ECCPRCR2
      Enables writing to the ECCETST register.   */
   BSP_REG_PROTECT_ECC2,

   /* PWPR
      Enable write to the PmnPFS register */
   BSP_REG_PROTECT_PMNPFS,

   /* 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
} e_bsp_reg_protect_t;
```

4.9 Return Values

4.9.1 Error Codes

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

```
typedef enum
{
   BSP_OK,
   BSP_ARG_ERROR,
   BSP_ERROR1,
   BSP_ERROR2,
   BSP_ERROR3
} 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.

4.10 Code Size

The sizes od ROM, RAM and maximum stack usage associated with this module are listed below.

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 v1.10
Compiler version:	LLVM C/C++ Compiler for RISC-V (17.0.0.202310)
	IAR C/C++ Compiler for RISC-V (V3.30.1)
	SEGGER Compiler (V8.10)

Configuration options: Default settings

ROM, RAM, and Stack Code Sizes (R9A02G021)

Compiler	API function *1	Clock setting *2	ROM	RAM	STACK
LLVM compiler *3	Disable	Default	2450	0	208
		All enable	2646	0	208
	Enable	Default	4924	0	208
		All enable	5140	0	208
IAR compiler	Disable	Default	1604	16	112
		All enable	1752	16	112
	Enable	Default	3146	16	112
		All enable	3294	16	112
SEGGER compiler	Disable	Default	1706	14	160
		All enable	1782	14	160
	Enable	Default	2436	14	160
		All enable	2498	14	160

Note 1:

Use macro definition BSP_CFG_XXXX_API_FUNCTIONS 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.

4.11 “for,” and “while,” Statements

This module uses “for” 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:
    /* WAIT_LOOP */
    for (w_count = 0U; w_count < 2U; w_count++)
    {
        R_BSP_NOP();
    }
```

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_GetIClkFreqHz	Returns the system clock frequency.
R_BSP_SoftwareDelay	Delays the specified duration.
R_BSP_ChangeICLKClockSetting	Switch CPU/peripheral hardware clock (ICLK) clock source.
R_BSP_GetVersion	Get the current version of the r_bsp.
R_BSP_RegisterProtectEnable	Enables write protection for selected registers.
R_BSP_RegisterProtectDisable	Disables write protection for selected registers.
R_BSP_DelayCycle	Delay the specified duration in CPU cycle.
machine_timer_start	Enable machine timer interrupt.
machine_timer_stop	Disable machine timer interrupt.
trigger_software_interrupt	Trigger software interrupt.
clear_software_interrupt	Clear software interrupt

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	/* The specified clock is started. */
BSP_ARG_ERROR	/* The specified clock is incorrect. */

Properties

Prototyped in *r_bsp_common.h*.

Description

This function starts oscillation of the specified clock.

To use the oscillated clock as the system clock, the CSKSCR register must be changed by separately calling "5.5 R_BSP_ChangeICLKClockSetting".

Example

```
e_bsp_err_t err;

/* Disable register protection */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_CGC);

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

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

/* Enable register protection */
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_CGC);
```

Special Notes:

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

5.3 API for Machine Timer

This function describe how to use machine timer. The functions user can use:

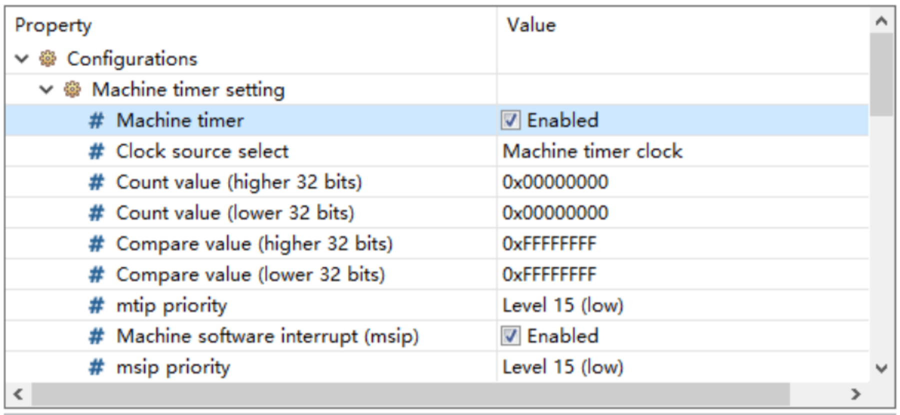
machine_timer_start(); machine_timer_stop(); trigger_software_interrupt(); clear_software_interrupt();

Format

```
void machine_timer_start (void);
void machine_timer_stop (void);
void trigger_software_interrupt (void);
void clear_software_interrupt (void);
```

GUI

Need to enable machine timer



Property	Value
Configurations	
Machine timer setting	
Machine timer	✓ Enabled
Clock source select	Machine timer clock
Count value (higher 32 bits)	0x00000000
Count value (lower 32 bits)	0x00000000
Compare value (higher 32 bits)	0xFFFFFFFF
Compare value (lower 32 bits)	0xFFFFFFFF
mtip priority	Level 15 (low)
Machine software interrupt (msip)	✓ Enabled
msip priority	Level 15 (low)

Description

The functions trigger_software_interrupt() and machine_timer_start() like enable interrupt, while machine_timer_stop() and clear_software_interrupt() disable them.

To trigger the 2nd software interrupt, just call trigger_software_interrupt() again.

Example

```
<main.c>:
#include "r_smc_entry.h"
volatile int mtip_flag = 0;
volatile int msip_flag = 0;
void main(void)
{
        trigger_software_interrupt();
        machine_timer_start();
        while(1);
}
```

When use machine interrupt

```
<r_cg_inthandler.c>
#include "r_smc_entry.h"
extern volatile int msip_flag;

void INT_ACLINT_MSIP(void)
{
    /* Start user code for INT_ACLINT_MSIP. Do not edit comment generated here */
        msip_flag ++;
    /* End user code. Do not edit comment generated here */
}
```

When use software interrupt

```
<r_cg_inthandler.c>
#include "r_smc_entry.h"
/* Start user code */
extern volatile int msip_flag;
/* End user code */

void INT_ACLINT_MSIP(void)
{
    /* Start user code for INT_ACLINT_MSIP. Do not edit comment generated here */
        msip_flag ++;
    /* End user code. Do not edit comment generated here */
}
```

5.4 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	/* The specified clock is stopped. */
BSP_ERROR2	/* The specified clock can not be stopped because it is ICLK clock's source. */
BSP_ARG_ERROR	/* The specified clock is incorrect. */

Properties

Prototyped in *r_bsp_common.h*.

Description

This function stops oscillation of the specified clock.

Example

```
e_bsp_err_t err;

/* Disable register protection */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_CGC);

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

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

/* Enable register protection */
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_CGC);
```

Special Notes:

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

5.5 R_BSP_GetIClkFreqHz()

This function returns the system clock frequency.

Format

```
    uint32 t R BSP GetIClkFreqHz(void);
```

Parameters

None

Return Values

System clock frequency specified by the r_bsp

Properties

Prototyped in *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'.

Example

```
uint32_t fclk_freq;

fclk_freq = R_BSP_GetFclkFreqHz();
```

5.6 R_BSP_ChangeICLKClockSetting ()

This function changes the ICLK clock value by changing its clock source and division. This function is also used to change PCLKB clock value.

Format

```
    e_bsp_err_t R_BSP_ChangeICLKClockSetting(e_clock_mode_t mode, e_clock_div_t
iclkdiv, e_clock_div_t pclkdiv);
```

Parameters

mode

Specifies clock resources supplied to the system clock (see 4.9.1)

iclkdiv, pclkdiv

The division ratio for the clock source is specified by the following constants defined in the e_clock_div_t structure.

- DIV_BY_1 : 1/1.
- DIV_BY_2 : 1/2
- DIV_BY_4 : 1/4
- DIV_BY_8 : 1/8
- DIV_BY_16 : 1/16
- DIV_BY_32 : 1/32
- DIV_BY_64 : 1/64
- DIV_BY_128 : 1/128

Return Values

BSP_OK	when changing setting is done.
BSP_ERROR1	The specified clock is not oscillating.
BSP_ERROR3	An unsupported state transition was specified. Refer to the user's manual.
BSP_ARG_ERROR	An invalid argument was input.

Properties

Prototyped in *r_bsp_common.h*.

Description

This function changes the clock source of the system clock to the specified clock and division value.

Example

```
e_bsp_err_t err;

/* Disable register protection */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_CGC);

/* Start clock operation(HOCO) with division */
    err = R_BSP_ChangeICLKClockSetting(HOCO, DIV_BY_2, DIV_BY_8);

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

/* Enable register protection */
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_CGC);
```

Special Note:

This function is available only when the macro definition (BSP_CFG_CHANGE_CLOCK_SETTING_API_FUNCTIONS) is set to 0.

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 Section4.8.3.

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 */
}
```

5.8 R_BSP_DelayCycle ()

This function is an assembly language wait loop.

Format

```
void R_BSP_DelayCycle(uint32_t wait_cycle);
```

Parameters

wait_cycle

The number of CPU cycle to delay.

Return Values

None

Properties

Prototype declared in r_bsp_common.h.

Description

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

Example

```
R_BSP_DelayCycle(100);
```

5.9 R_BSP_GetVersion ()

This function gets the version of the BSP.

Format

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

Parameters

None

Return Values

32-bit integer representing the BSP version

(((uint32_t)R_BSP_VERSION_MAJOR) << 16) | ((uint32_t)R_BSP_VERSION_MINOR)

Properties

Prototype declared in r_bsp_common.h.

Description

This function can get the compiler's current BSP version information as an integer value.

Example

```
uint32_t ver_num;
ver_num = R_BSP_GetVersion();
```

5.10 R_BSP_RegisterProtectEnable ()

This function sets write protection to the specified register.

Format

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

Parameters

regs_to_protect

Register type to set write protection (see 4.8.4)

Return Values

None

Properties

Prototype declared in r_bsp_common.h.

Description

This function allows the user to set write protection for a specific register. It is limited to specific registers that can be specified. (See 4.8.4)

Example

```
bsp_reg_protect_t regs_to_protect = BSP_REG_PROTECT_ECC;

/* set BSP_REG_PROTECT_ECC registers (ECCMODE, ECC1STSEN, and ECCOAD) to disable
writing */
 R_BSP_RegisterProtectEnable (regs_to_protect);
```

5.11 R_BSP_RegisterProtectDisable ()

This function disables write protection for selected registers.

Format

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

Parameters

regs_to_unprotect

Register type to release write protection.

Return Values

None

Properties

Prototype declared in r_bsp_common.h.

Description

This function can remove write protection to a specific register. Only certain registers can be specified.

Please check bsp_reg_protect_t (enum structure) in r_bsp_common.h to see which registers can be specified.

Example

```
bsp_reg_protect_t regs_to_unprotect = BSP_REG_PROTECT_ECC;

/* set BSP_REG_PROTECT_ECC registers (ECCMODE, ECC1STSEN, and ECCOAD) to disable
writing */
 R_BSP_RegisterProtectDisable (regs_to_unprotect);
```

6. Project setup

This section explains how to add r_bsp to a project.

6.1 How to add the BSP

This module must be added for each project in which it is used. Renesas recommends using the Smart Configurator.

(1) Adding the BSP using the Smart Configurator on e2 studio
Use the Smart Configurator on e2 studio to automatically add BSP to user projects. For details, refer to the RISC-V Smart Configurator User Guide: e2 studio (R20AN0730) for details.

(2) Adding the BSP using Smart Configurator in IAREW
You can add the BSP to your project automatically by using the standalone version of Smart Configurator. Refer to the application note RISC-V Smart Configurator User's Guide: IAREW, SEGGER Embedded Studio (R20AN0731) for details.

(3) Adding the BSP using Smart Configurator in SEGGER
You can add the BSP to your project automatically by using the standalone version of Smart Configurator. Refer to the application note RISC-V Smart Configurator User's Guide: IAREW, SEGGER Embedded Studio (R20AN0731) for details.

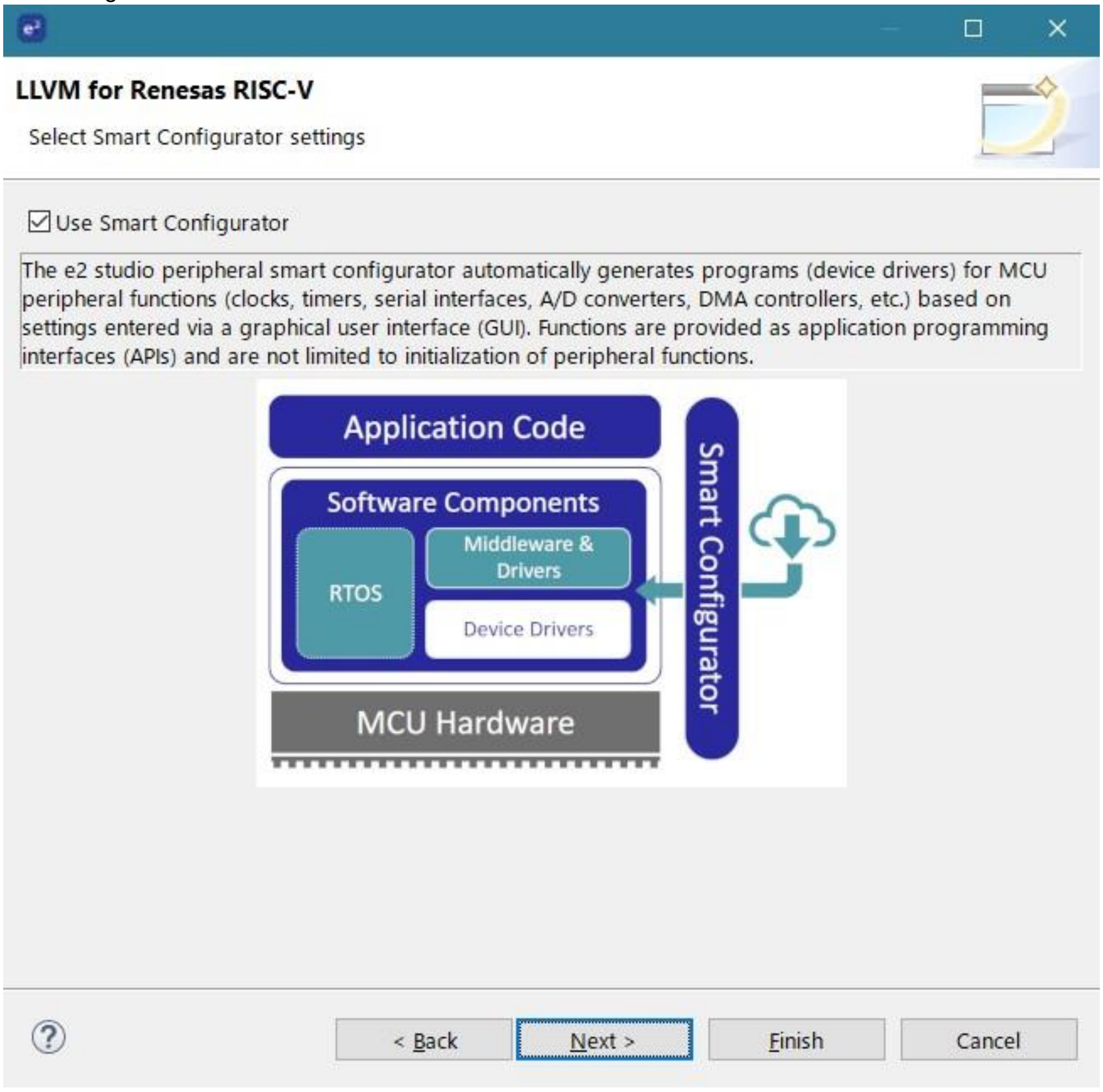
6.2 Adding the BSP to a Project in e^2 studio

How to add the BSP to a project in e^2 studio is described below.

6.2.1 Adding the BSP Using Smart Configurator in e^2 studio

This explanation uses e^2 studio (2024-01).

1. Create a new project in e^2 studio.
   When creating your project, check the box next to "Use Smart Configurator" to launch Smart Configurator.



LLVM for Renesas RISC-V

Select Smart Configurator settings

☑ Use Smart Configurator

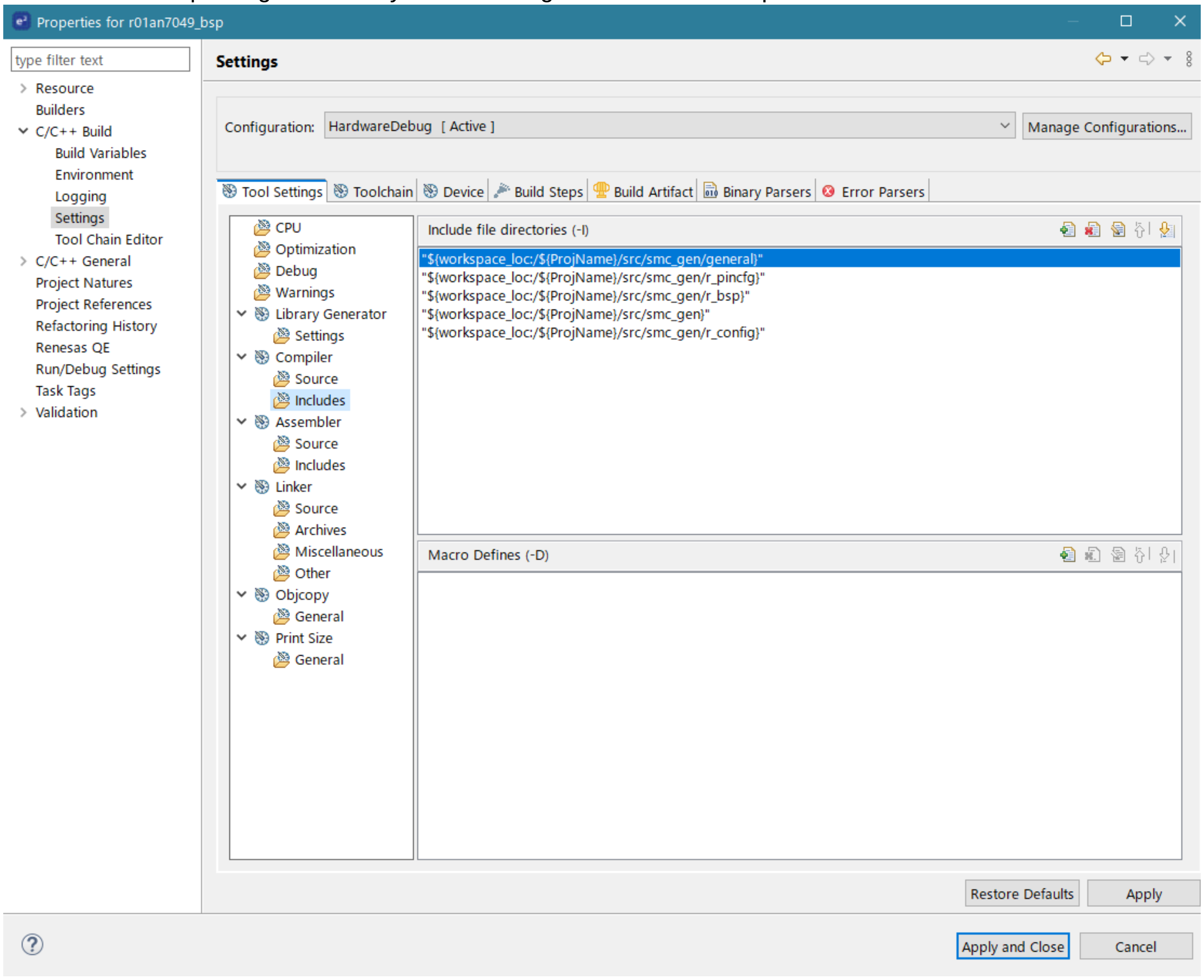
The e2 studio peripheral smart configurator automatically generates programs (device drivers) for MCU peripheral functions (clocks, timers, serial interfaces, A/D converters, DMA controllers, etc.) based on settings entered via a graphical user interface (GUI). Functions are provided as application programming interfaces (APIs) and are not limited to initialization of peripheral functions.

2. Follow the procedure described in 6.1, How to add the BSP, to add the BSP to your project in e^2 studio.

3. Right-click the project and click "Properties."

4. On the Tool Settings tab, select Compiler → Includes.

5. BSP include paths generated by Smart Configurator have been specified.



Properties for r01an7049_bsp

Settings

Configuration: HardwareDebug [Active]

Include file directories (-I)

```
"${workspace_loc:/${ProjName}/src/smc_gen/general}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_pincfg}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_bsp}"
"${workspace_loc:/${ProjName}/src/smc_gen}"
"${workspace_loc:/${ProjName}/src/smc_gen/r_config}"
```

Macro Defines (-D)

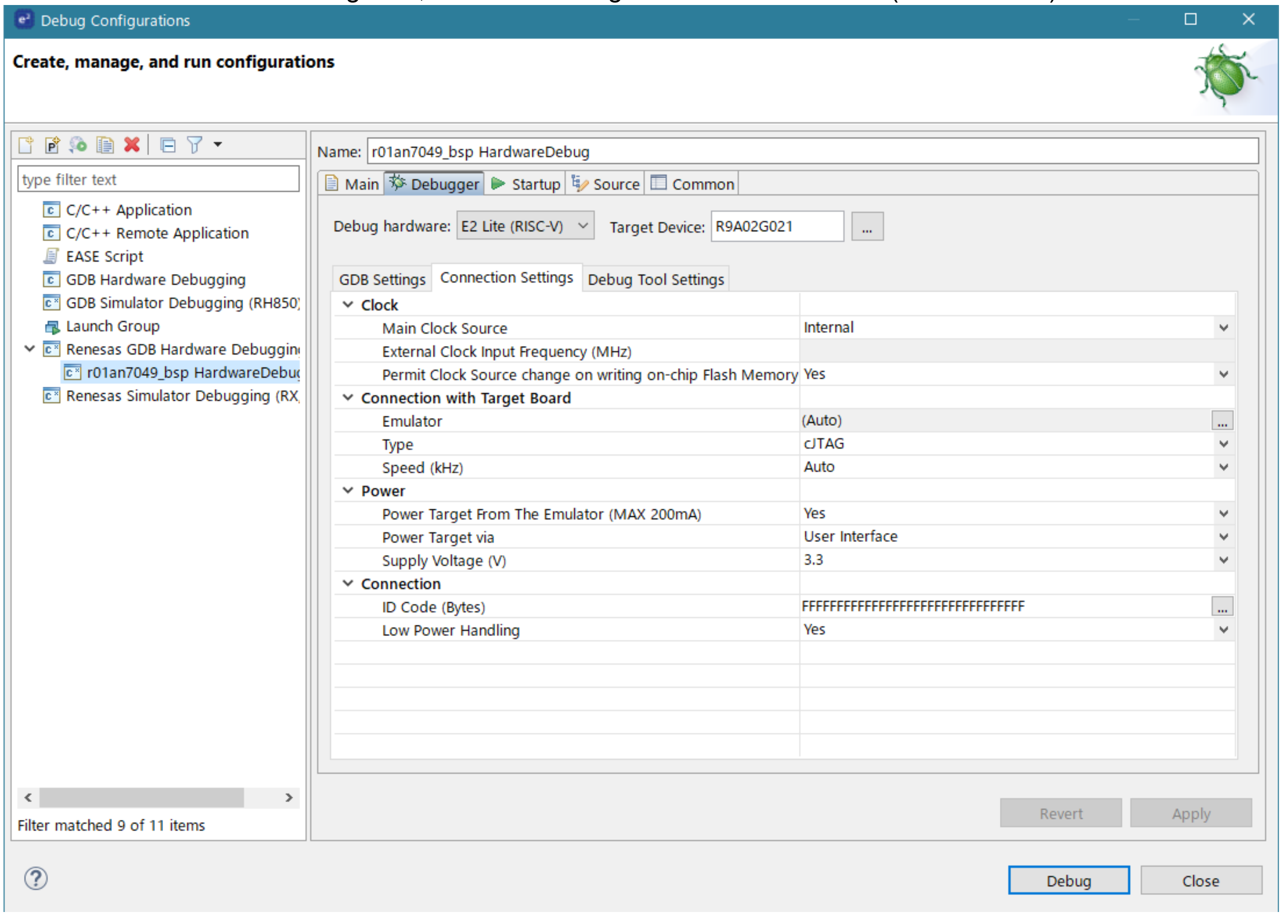
8. Right-click the project and click “Build Project.”

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 (RISC-V).”

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



Name: r01an7049_bsp HardwareDebug

Debug hardware: E2 Lite (RISC-V) Target Device: R9A02G021

Clock	
Main Clock Source	Internal
External Clock Input Frequency (MHz)	
Permit Clock Source change on writing on-chip Flash Memory	Yes
Connection with Target Board	
Emulator	(Auto)
Type	cJTAG
Speed (kHz)	Auto
Power	
Power Target From The Emulator (MAX 200mA)	Yes
Power Target via	User Interface
Supply Voltage (V)	3.3
Connection	
ID Code (Bytes)	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Low Power Handling	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 (2024-01) IAR Systems IAR Embedded Workbench for Renesas (V3.30.1)
C compiler	LLVM for RISC-V 17.0.0.202310
Module revision	Rev.1.00
Board used	FPB-R9A02G021 WS (Product type: RTK9FPG021S000W0BJ)

Table 7.2 Confirmed Operating Environment (Rev. 1.10)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio (2024-04) IAR Systems IAR Embedded Workbench for Renesas (V3.30.1) SEGGER Embedded Studio 8.10
C compiler	LLVM for RISC-V 17.0.2.202401
Module revision	Rev.1.10
Board used	FPB-R9A02G021 (Product type: RTK9FPG021S00001BJ)

Table 7.3 Confirmed Operating Environment (Rev. 1.20)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio (2024-07) IAR Systems IAR Embedded Workbench for Renesas (V3.30.1) SEGGER Embedded Studio 8.10b
C compiler	LLVM for RISC-V 17.0.2.202403
Module revision	Rev.1.20
Board used	FPB-R9A02G021 (Product type: RTK9FPG021S00001BJ)

7.2 Note on the updated macro define name from Rev1.10 to Rev1.20

The Macro definition name change from BSP_MCU_SERIES_ASSPEASY to BSP_MCU_SERIES_GPMCU. in mcu_info.h.
The Macro definition name change from BSP_CFG_RTC_ALM_VECT to BSP_CFG_RTC_ALM_OR_PRD_VECT in mcu_mapped_interrupts.c.
The Macro definition name change from ICU_EVENT_RTC_ALM to ICU_EVENT_RTC_ALM_OR_PRD in mcu_mapped_interrupts.h.

7.3 Note on the protect for user bricks the device from Rev1.10 to Rev1.20

In v1.20 and later versions, the [startup select] setting doesn't affect option settings (mcu_option_settings.c file). Add the lock for OCD/Serial Programmer ID Setting, Access Window Block Address Setting and User ID Setting.

7.4 Note on the Prefetch buffer enable register (PFBER) from Rev1.10 to Rev1.20

In v1.20 and later versions, add prefetch buffer function and the BSP_CFG_PFB_ENABLE default value is 1(Enable).

Revision History

Rev.	Date	Description	
		Page	**Summary**
1.00	Nov.27.23	—	Initial release
1.10	Mar.29.24	1,8,10,11, 22,36,40	Added support for SEGGER and IAR compiler.
1.20	June.28.24	6,7,10,13,15 ,16,22,25,26 ,29	Support machine timer and machine software interrupt
		_	- Add Enable OCD/Serial Programmer ID Setting to allow GUI setting of OCD/Serial Programmer ID Setting Register value. - Add Enable Access Window Block Address Setting to GUI setting of Access Window Star/End Block Address. - Add Enable User ID setting to allow GUI setting of User ID Setting Register n value.
		-	- In start.s initialize_vect called before bsp_init_systems
		6	Update structure of generate folder: - Remove file generate vecttbl.h. - Change the name vecttbl.c to mcu_option_settings.c. - Add new file r_bsp_machine_timer.c, r_bsp_machine_timer.h support machine timer.
		17	Change all API endwith DISABLE to FUNTIONS: - BSP_CFG_FLRPROTAC_REG_DISABLE - BSP_CFG_STARTUP_DISABLE - BSP_CFG_CLOCK_OPERATION_API_FUNCTIONS_DISABLE - BSP_CFG_CHANGE_CLOCK_SETTING_API_FUNCTIONS_DISABLE - BSP_CFG_INTERRUPT_SETTING_DISABLE

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.

Corporate Headquarters

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

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/.

Trademarks