RE01 1500KB,256KB Group

Getting Started Guide to Development Using CMSIS Package

Summary

This application note describes the procedure for developing software using the CMSIS Driver Package for RE01 1500KB and RE01 256KB Group. Refer to this document to gain an understanding of the basic setting workflow for using drivers and how to implement code for peripheral functions not supported by the drivers included in this package.

Target Device

- RE01 1500KB Group
- RE01 256KB Group

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

Summary

- ✓ The CMSIS Driver Package includes startup code and drivers for target device.
- ✓ Chapter 2 describes how to run a project for this package.
- ✓ Chapter 3 explains features of the components of this package.
- ✓ Chapters 4 and 5 summarize drivers in this package.
- ✓ Chapter 6 explains how to use basic functions such as interrupts and pins.
- ✓ Chapter 7 explains how to create a user program
- ✓ Chapter 8 explains how to create a project.
- ✓ Chapter 9 explains debugging.

■List of Terms

Term	Description
CMSIS	A software interface standard specified by ARM®. CMSIS stands for “Cortex Microcontroller Software Interface Standard.”
CMSIS-Driver	A peripheral function driver for a software interface conforming to CMSIS
Device HAL	A driver with specifications exclusive to an MCU vendor (Renesas in the case of this document)
CMSIS-CORE	An MCU startup routine conforming to CMSIS
R_CORE	Specifications exclusive to Renesas added to CMSIS-CORE
R_SYS	Drivers exclusive to Renesas for clock switching, interrupt control, etc.
Startup	Activation processing from when the MCU is reset to the execution of the Main function
Address mapping	Indicates the regions in the MCU internal space (addresses) to which programs and data are allocated
Callback function	A function, created by the user, that is called by a driver when a specific event occurs
ICU	RE01 interrupt controller unit
NVIC	The nested vectored interrupt controller of processors such as the ARM Cortex®-M0+. NVIC stands for “Nested Vectored Interrupt Controller.”
IRQ number	NVIC external interrupt input number
User’s manual	The MCU manual, available from Renesas
EWARM	The integrated development environment (IDE) supplied by IAR. EWARM stands for IAR Embedded Workbench® for Arm®.
e^2 studio	The integrated development environment (IDE) supplied by Renesas

Table of Contents

1. Package Summary

1.1 About CMSIS

Included with this package are the components appearing with red frames in Figure 1-1 CMSIS Overview.

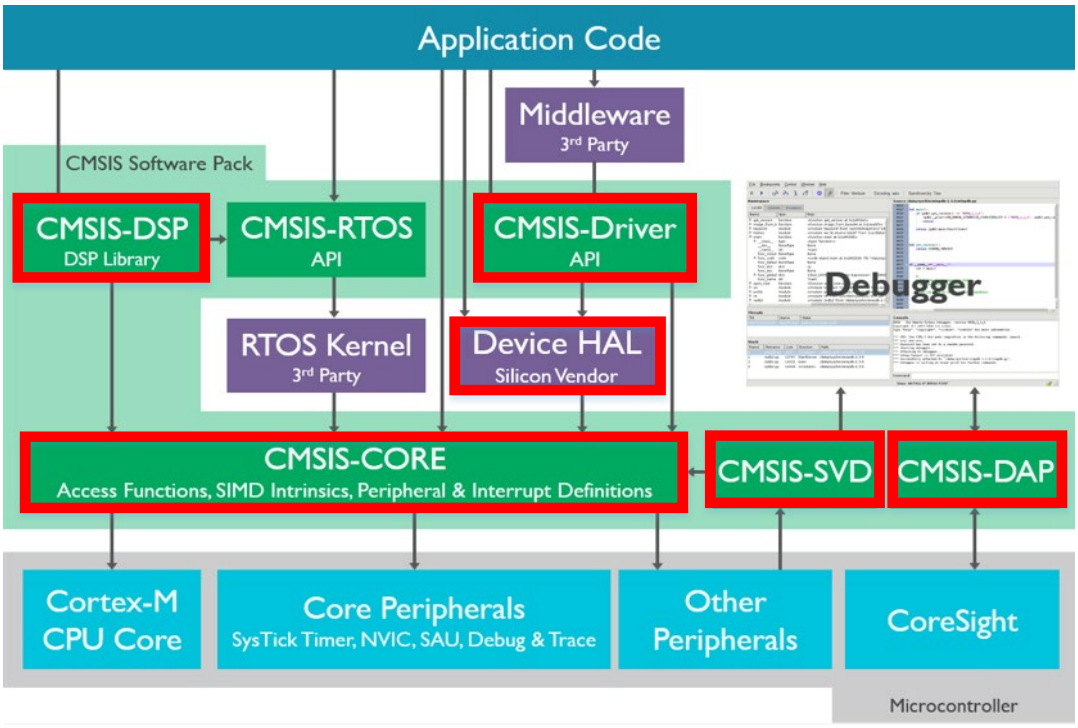


Figure 1-1 CMSIS Overview

The relationship of drivers to the components included in this package is indicated in Figure 1-2.

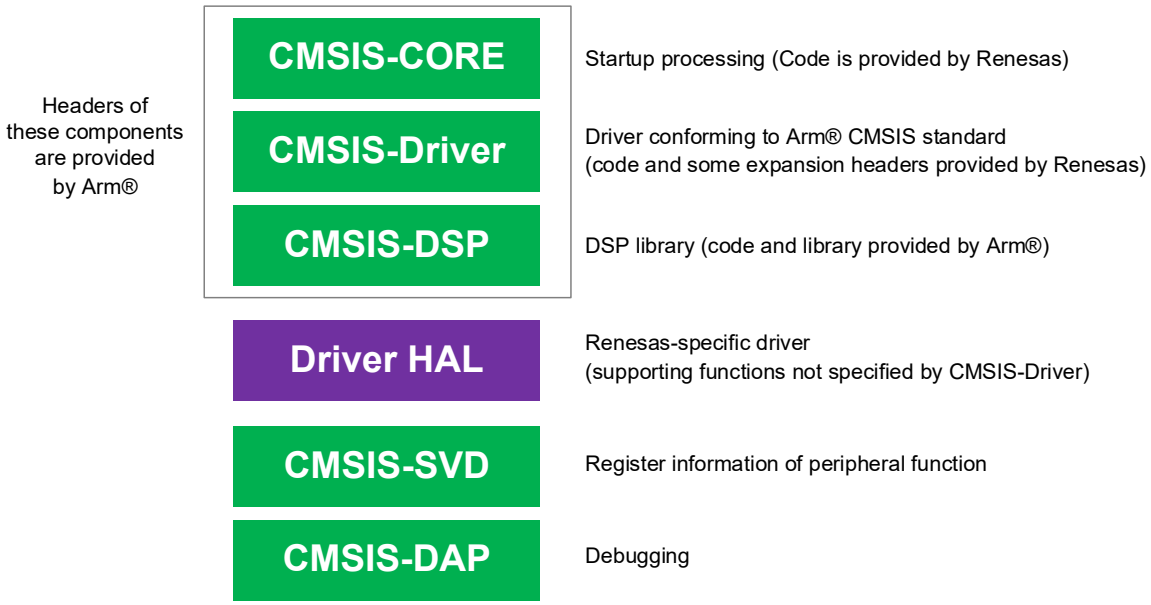


Figure 1-2 Relationship between Components and Drivers

1.2 Folder Structure

The folder structure for this package is indicated in Figure 1-3.

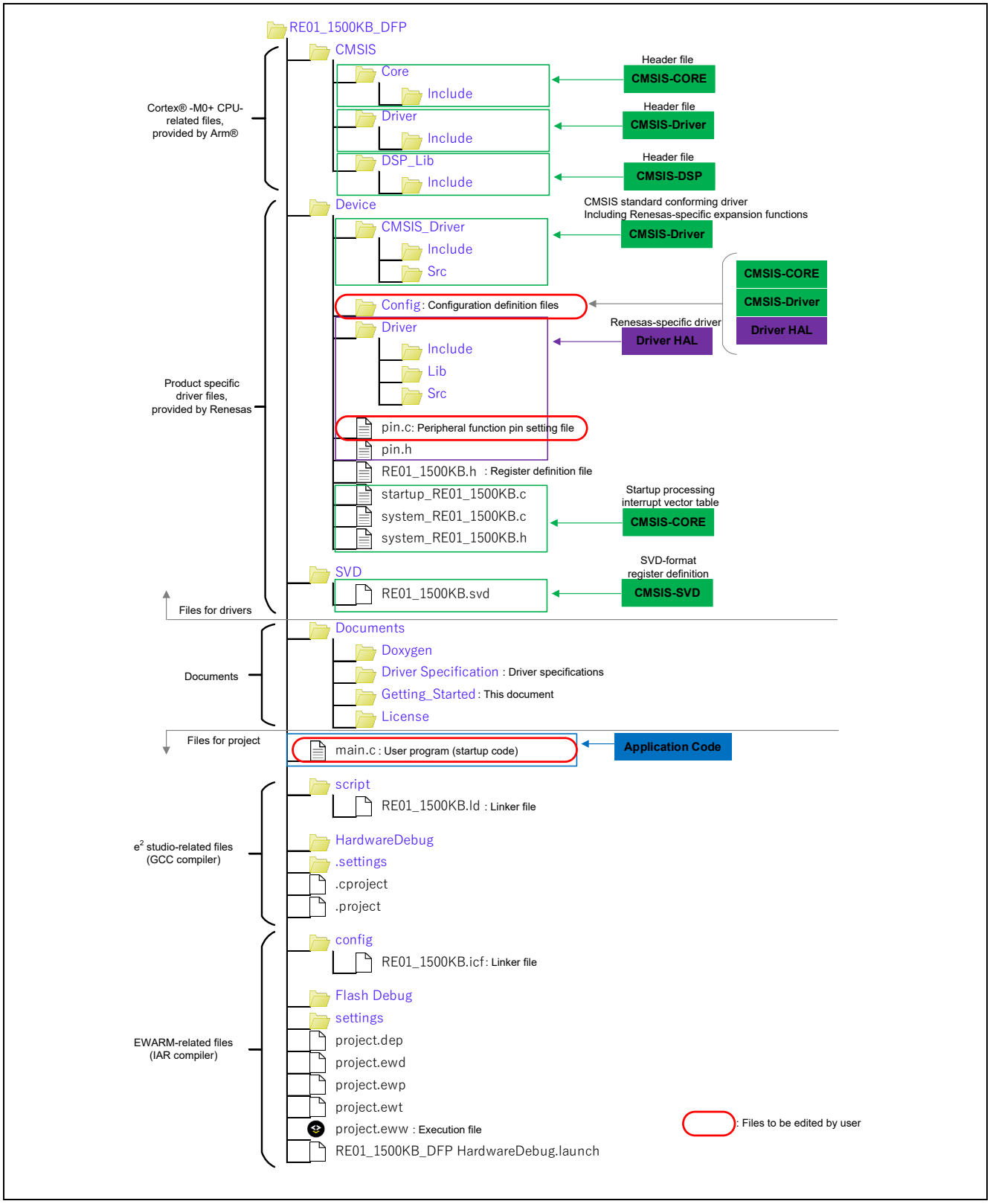


Figure 1-3 CMSIS Driver Package Folder Structure for RE01 1500KB Group

1.3 Supported Functions

Functions supported by this package are explained. For driver specifications, refer to the driver specifications presented in chapter 4. For details on each function, refer to the User's Manual: Hardware of the relevant device.

CMSIS-CORE

Table 1-1 CMSIS-CORE Supported Functions

Driver Name	Support Function	Related Hardware Function
R_CORE	Interrupt vector table	Interrupt controller unit (ICU)
	Startup processing	Clock generation circuit Power consumption reduction function

CMSIS-Driver

Table 1-2 CMSIS-Driver Supported Functions

Driver Name	Support Function	Related Hardware Function
R_SPI	SPI serial communication	Serial peripheral interface (SPI)
R_I2C	I2C serial communication	I2C bus interface (RIIC)
R_USART	UART serial communication	Serial communication interface (SCIg, SCIi)

Driver HAL

Table 1-3 Driver HAL Supported Functions

Driver Name	Support Function	Related Hardware Function
R_SYS	Clock setting	Clock generation circuit
	Power control mode	Power consumption reduction function
	Option setting	Option setting memory
	Interrupt control	Interrupt controller unit (ICU) NVIC
	Register write protection	Register write protection
	RAM expansion of programs	(handled in software)
	Hardware resource locking	(handled in software)
R_LPM	I/O power supply domain control	Power consumption reduction function
	Power consumption reduction functions • Module stopping • Internal flash memory power shutoff etc.	
R_PIN	Pin setting	Multifunction pin controller
R_ADC	14-bit A/D conversion	14-bit A/D converter
R_DMAC	DMA transfer	DMA controller (DMAC)
R_DTC	DTC transfer	Data transfer controller (DTC)
R_FLASH	On-chip flash memory	Flash memory
R_GDT	2D graphic data	2D graphic data conversion circuit (GDT)
R_SMIP	Serial MIP LCD	(handled by software using the following functions) • R_SPI driver • Asynchronous general-purpose timer (AGT)
R_PMIP	Parallel MIP LCD	MIP LCD controller (MLCD)
R_USB	USB2.0 FS Host/Function	USB2.0 FS Host/Function(USB)

[Note] Refer to the chapter on NVIC in the Cortex™-M0+ Technical Reference Manual (ARM DDI 0484C).

1.4 Package Features

This section describes the main features of this package.

- Startup processing
  This package provides startup processing from reset cancellation until execution of the main function.
  For details, see section 6.1, Startup Processing.

- Function to suppress propagation of I/O power supply domain undefined values
  This device has multiple I/O power supply domains. After reset cancellation, an undefined value propagation suppression function is enabled for nearly all of the I/O power supply domains. The undefined value propagation suppression function must be disabled for those I/O power supply domains to which power is being supplied.
  For details, see section 6.2, Control of Undefined Value Propagation Suppression in I/O Power Supply Domains.

- Preparations prior to user program creation
  When using this package, the user must not only create a program, but must also undertake preparations such as creation of specific functions and editing of driver configuration definition headers and driver functions.
  For details, see sections 6.3, Interrupt Control, 6.4, Clock Settings, 6.5, Pin Settings, and 7.1, Preparation for User Program Creation.

- Program placement in RAM
  Power consumption of this device can be reduced by shutting off the supply of power to internal flash memory. When running a program expanded in RAM after shutting off the supply of power to internal flash memory, it is necessary to place a desired program in RAM.
  For details, see section 6.6, RAM Placement of Programs.

1.5 Environments in which Operation is Confirmed

The environments in which operation of this package has been confirmed are indicated below.

Table 1-4 IAR Compiler Environment

Item	Description	Vendor
MCU	R7F0E015D2CFB (144 pins)	Renesas
Target Board	Evaluation Kit RE01 1500KB (Model name: RTK70E015DSxxxxxBE)	Renesas
Integrated development environment (IDE))	EWARM V8.3 or later (AR Embedded Workbench® for Arm®)	IAR Systems
Compiler	IAR V8.32 or later	IAR Systems
Debugger	I-Jet	IAR Systems
	J-Link OB (implemented on the target board)	SEGGER

Table 1-5 GCC Compiler Environment

Item	Description	Vendor
MCU	R7F0E015D2CFB (144 pins)	Renesas
Target Board	Evaluation Kit RE01 1500KB (Model name: RTK70E015DSxxxxxBE)	Renesas
Integrated development environment (IDE))	e^2 studio V.7 or later	Renesas
Compiler	GCC V.6 GNU 6-2017-q2-update	—
Debugger	J-Link OB (implemented on the target board)	SEGGER

2. Running a Project

This chapter describes the flow of operation of the program included with this package.

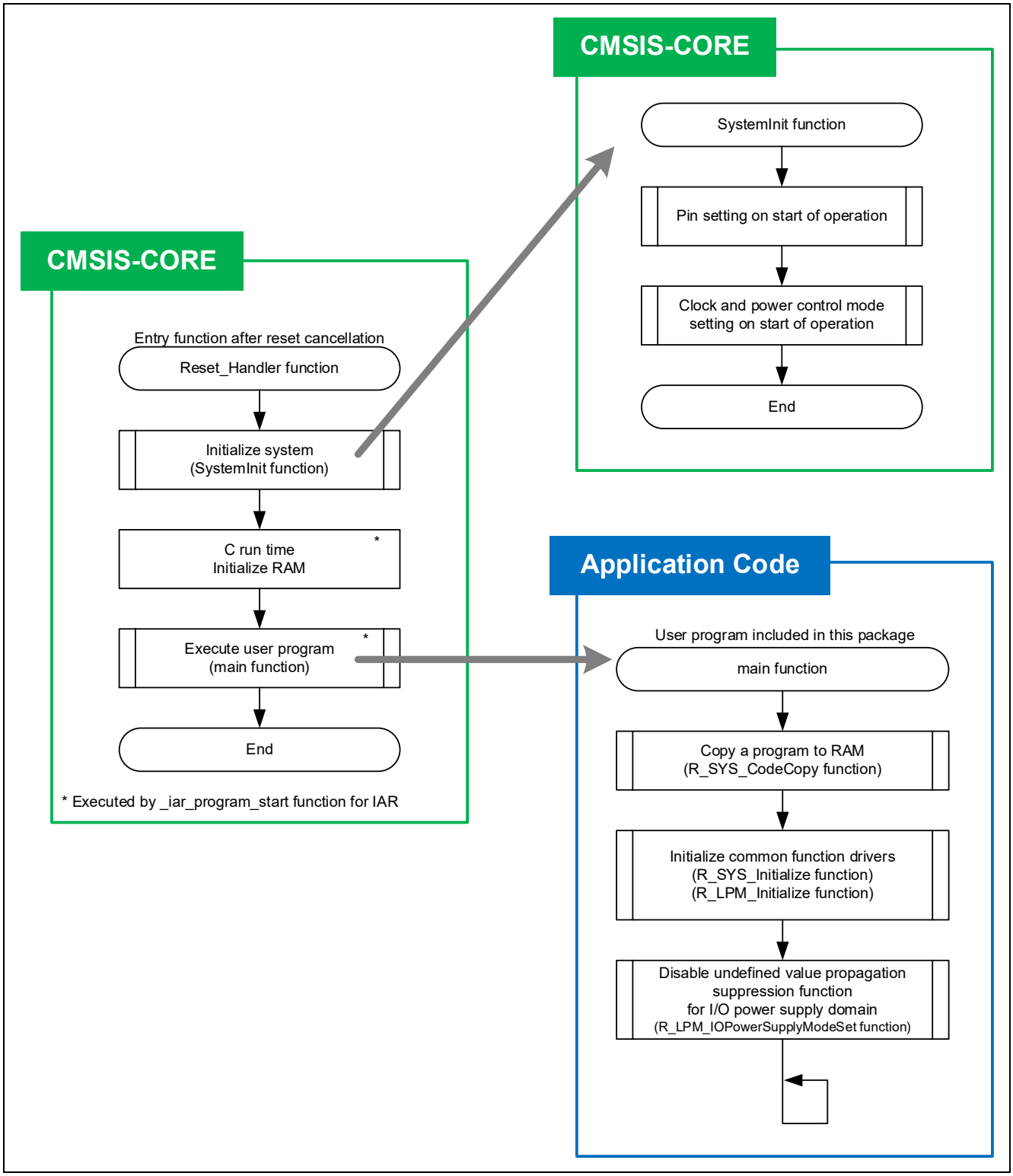


Figure 2-1 Program Operation Flow

2.1 EWARM Version

This section describes the method for running a project included with this package.

The following is an example of settings to be used with the Evaluation Kit board.

(1) Start up the project.

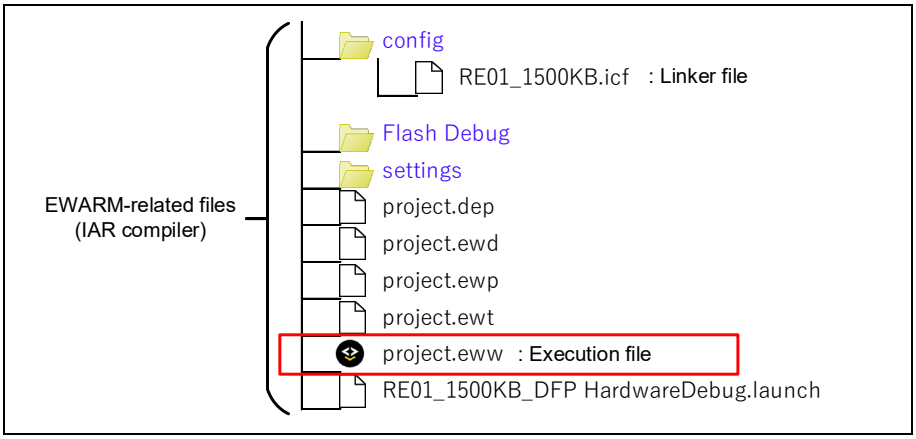


Figure 2-2 Project Startup File

(2) Perform compiling.

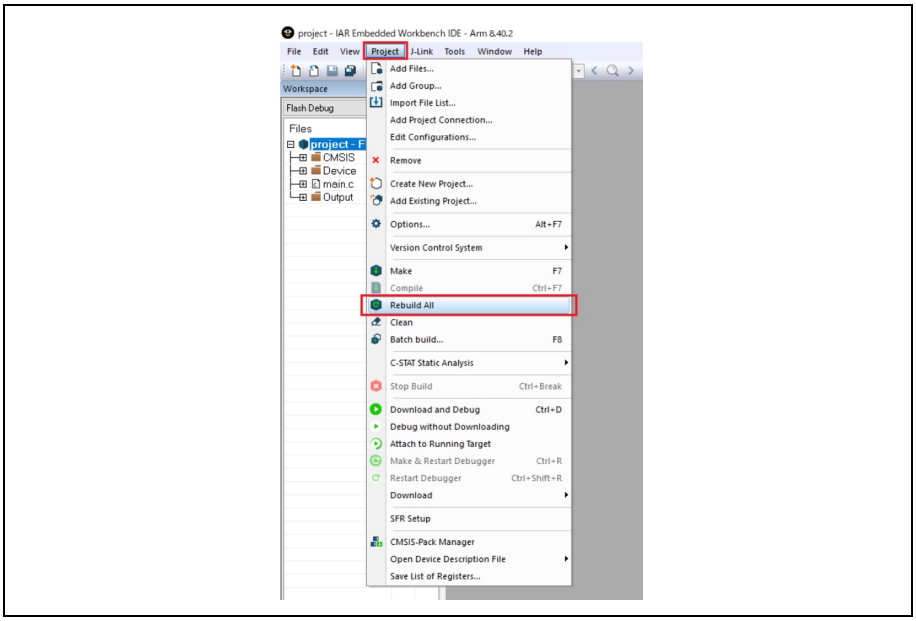


Figure 2-3 Compile Menu

(3) Select J-Link.

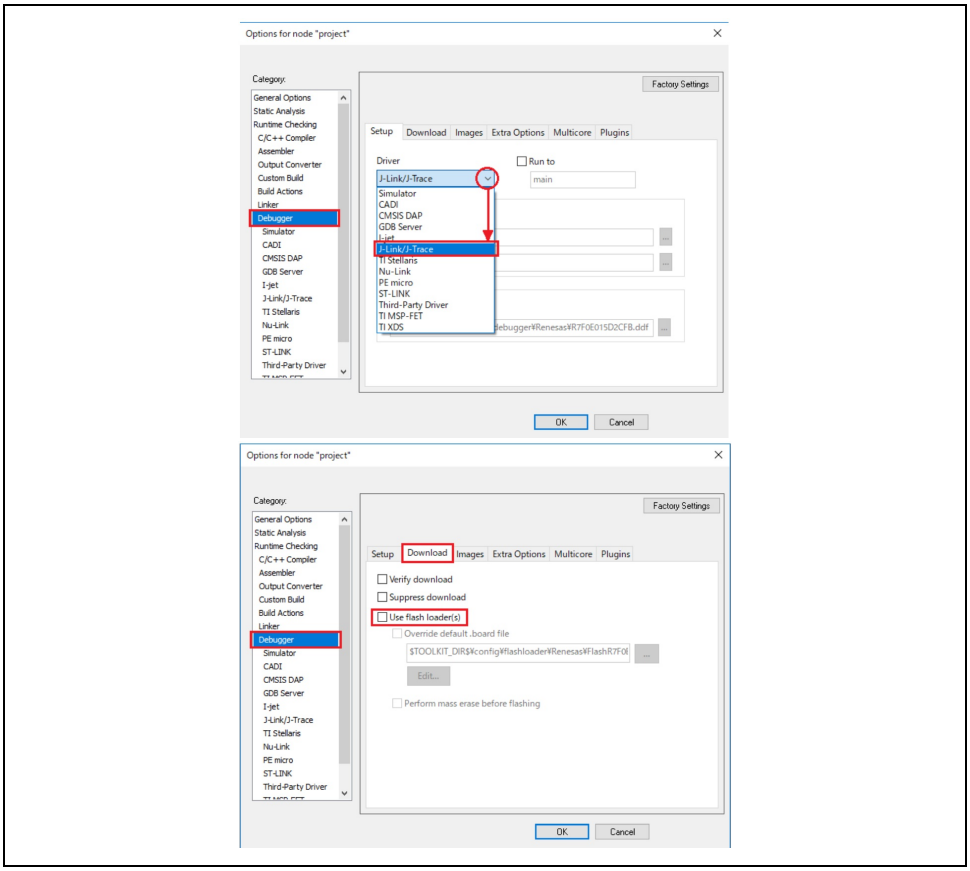


Figure 2-4 Procedure for J-Link Selection

(4) Set J-Link.

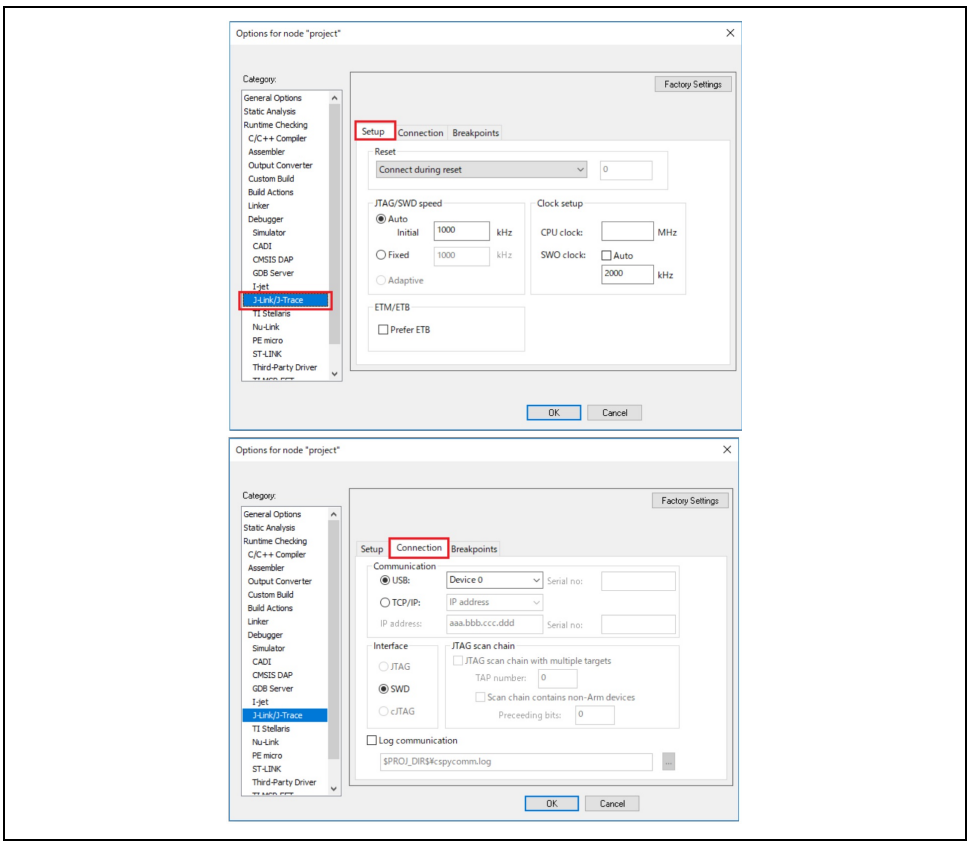


Figure 2-5 Procedure for J-Link Settings

(5) Connection with the board

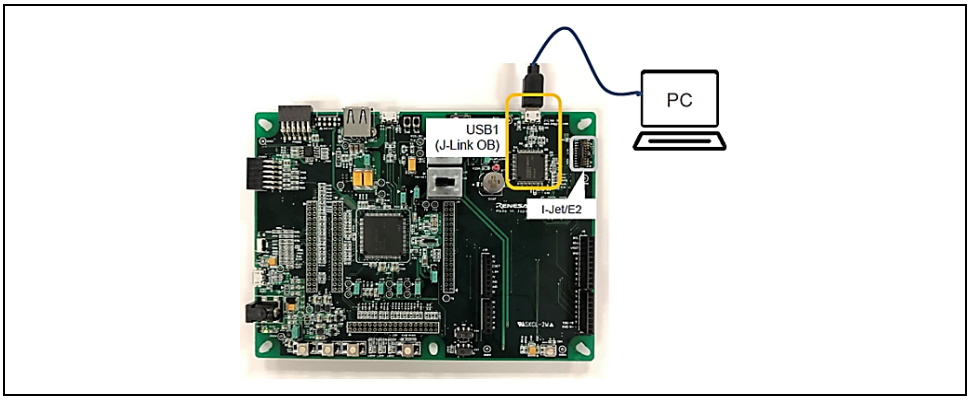


Figure 2-6 Board Connection Example

(6) Download and start up the debugger.

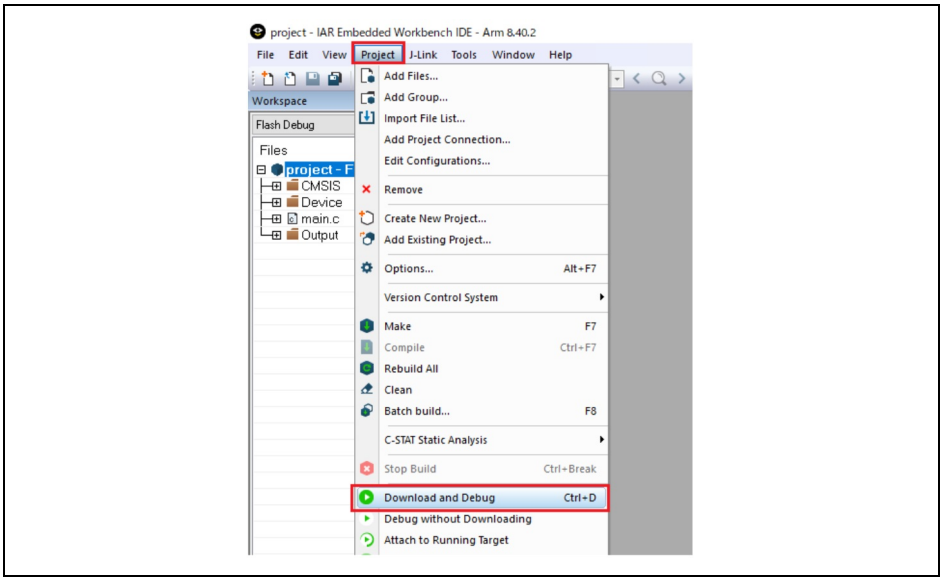


Figure 2-7 Debugger Startup Menu

2.2 e² studio Version

This section describes the method for running a project included with this package.

The following is an example of settings when using the Evaluation Kit board.

(1) Start up the e² studio

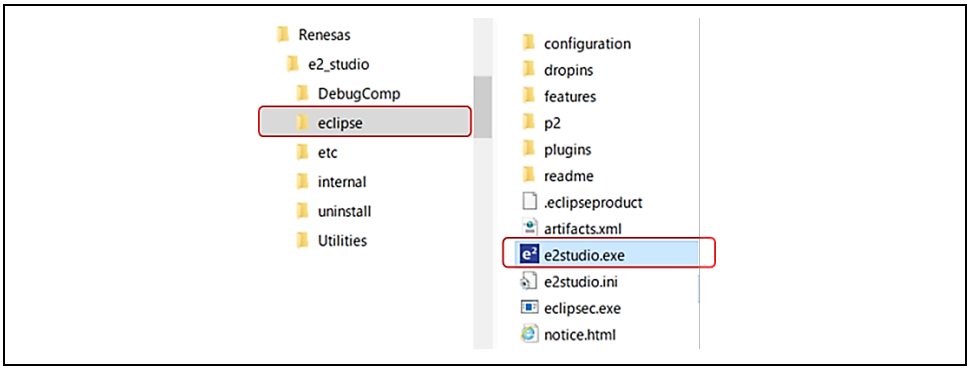


Figure 2-8 e2 studio Startup File

(2) Perform import of a project.

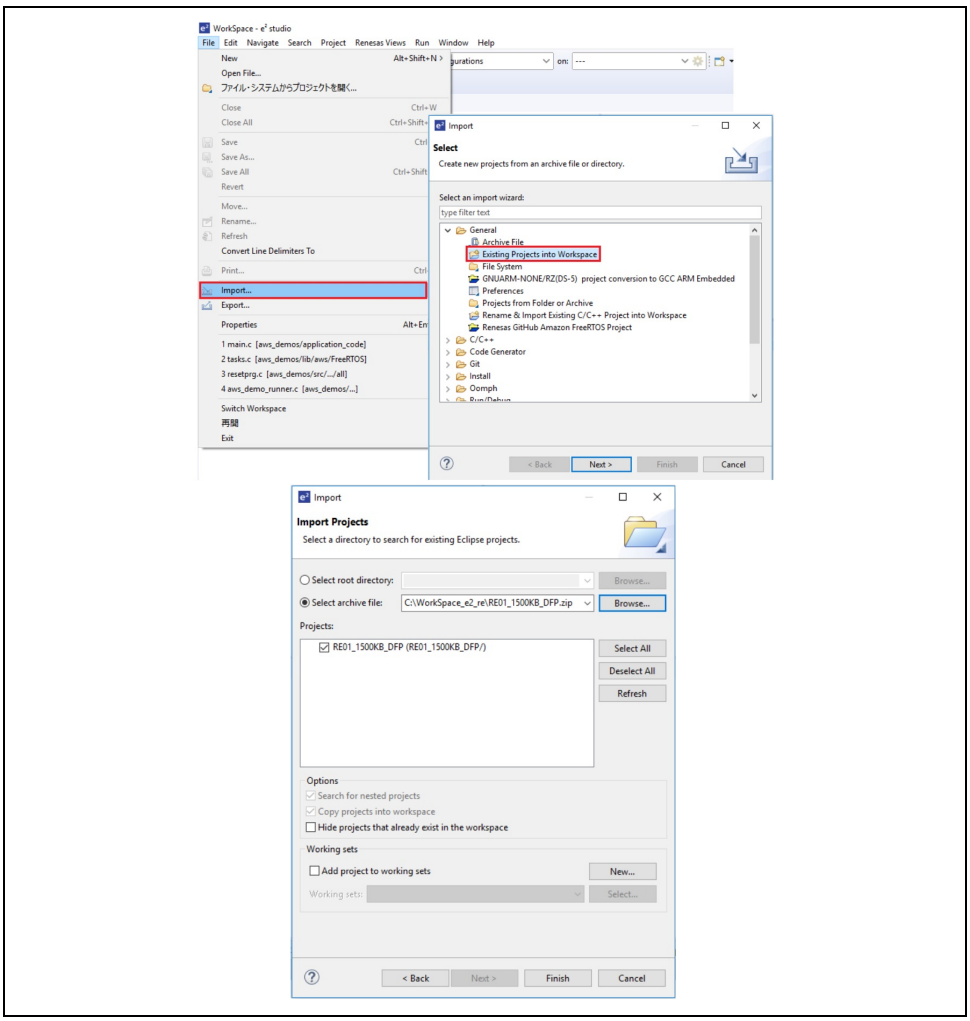


Figure 2-9 Project Import Procedure

(3) Perform compiling.

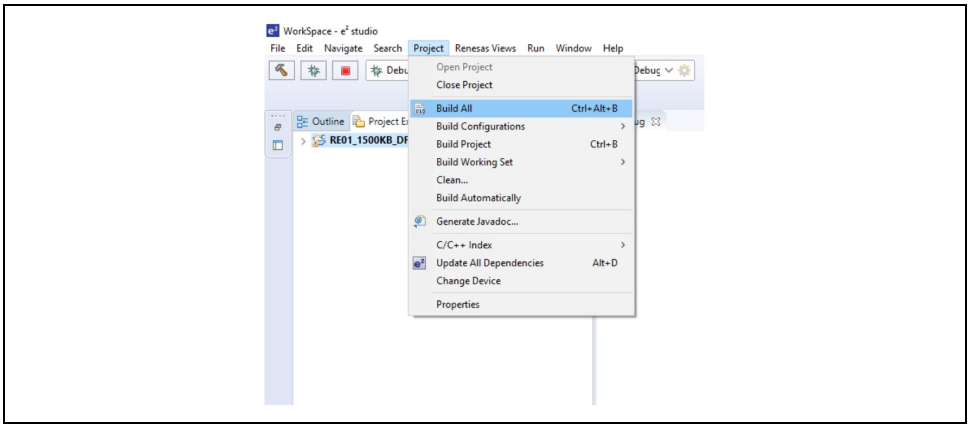


Figure 2-10 Compiler Menu

(4) Set J-Link.

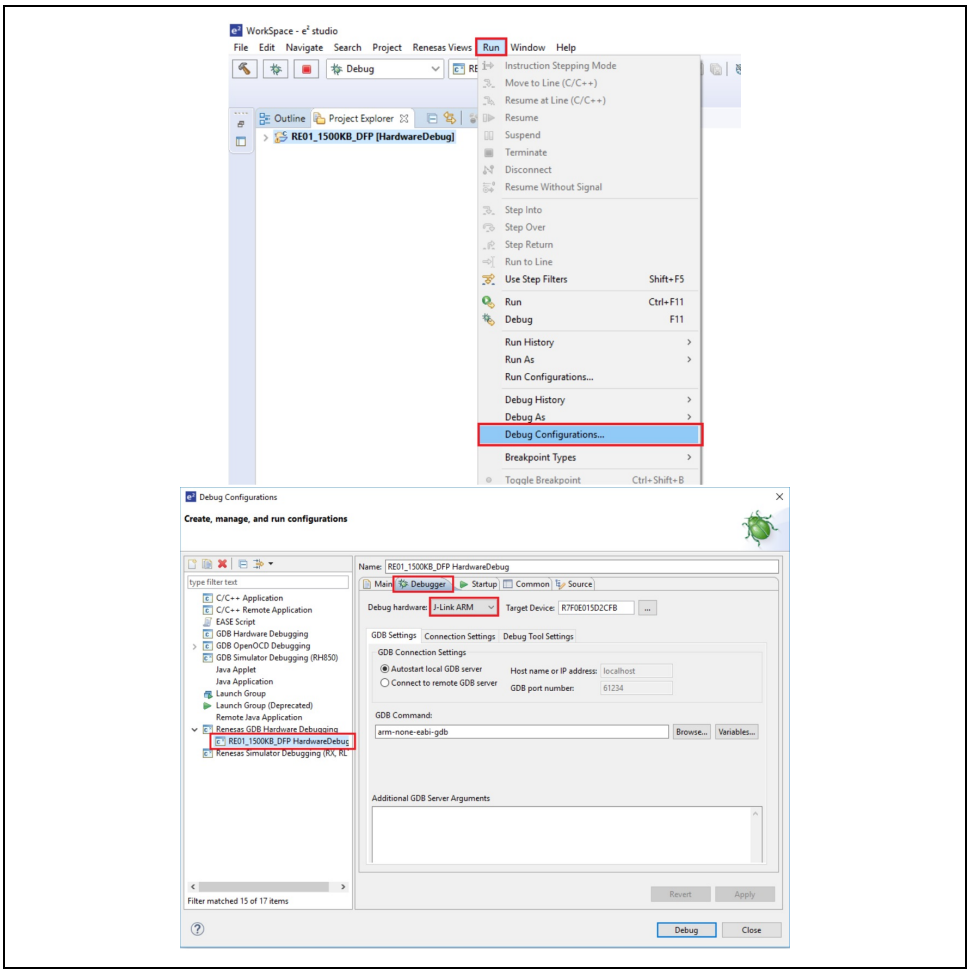


Figure 2-11 Procedure for J-Link Settings

(5) Connection with the board

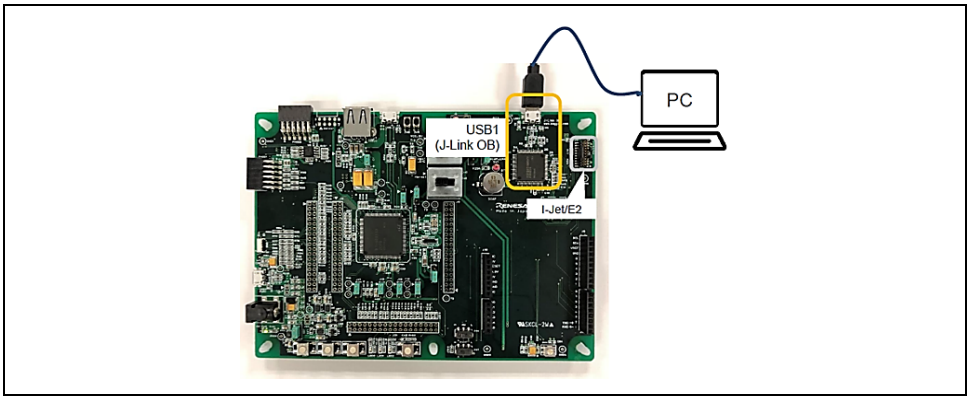


Figure 2-12 Board Connection Example

(6) Download and start up the debugger

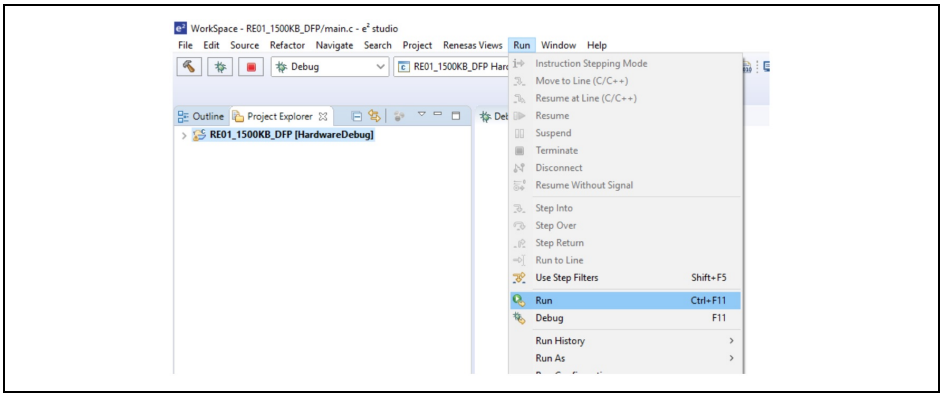


Figure 2-13 Debugger Startup Menu

3. Components

This chapter describes the components constituted from drivers included in this package.

3.1 CMSIS-CORE

The CMSIS-CORE in this package is a component made up of a driver group that uses headers provided by Arm® and code provided by Renesas.

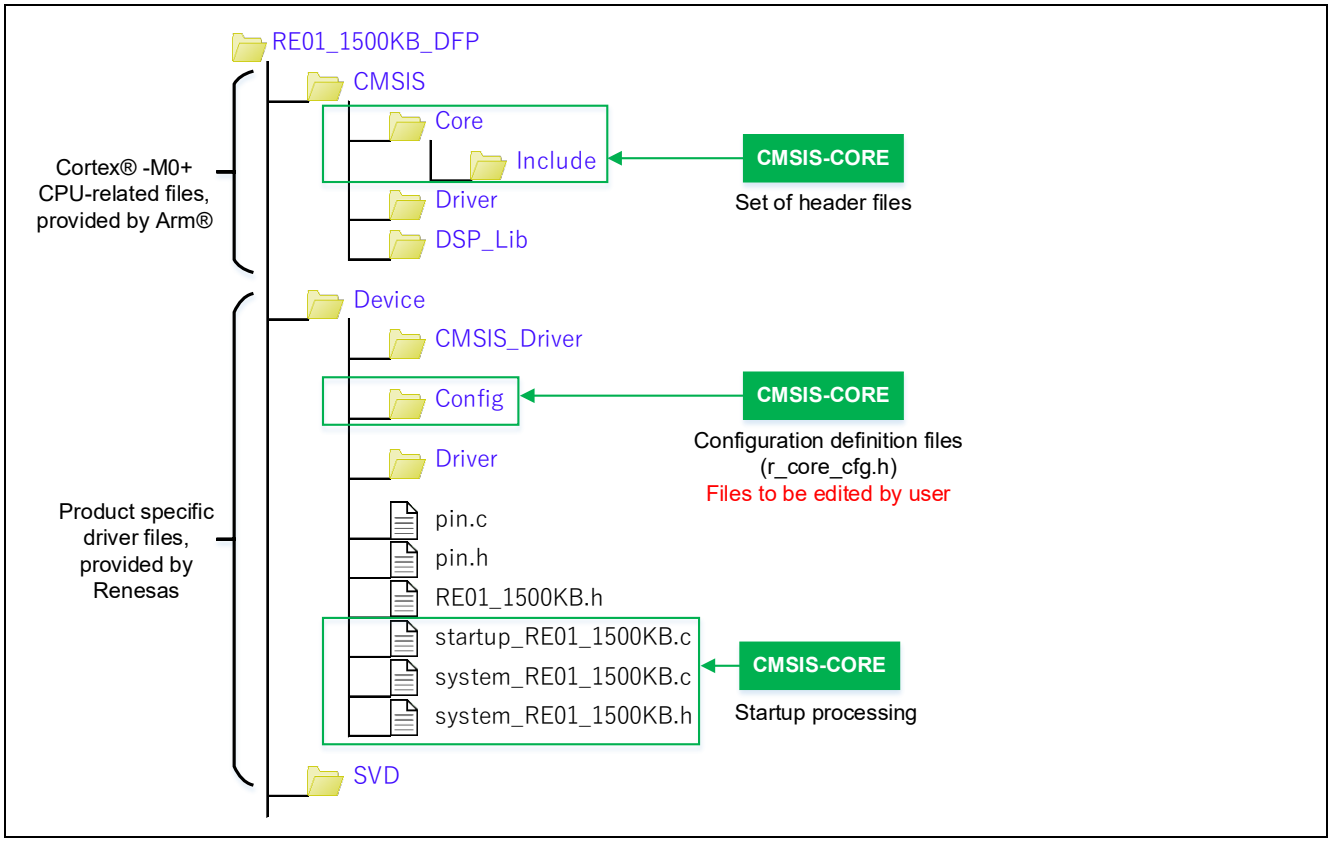


Figure 3-1 CMSIS-CORE Related Files

3.1.1 Supported Drivers

The CMSIS-CORE in this package includes R_CORE drivers.

R_CORE drivers have configuration definition headers, and the user can edit definition values in accordance with the operating environment.

3.1.2 Main Functions of R_CORE Drivers

The main functions of R_CORE drivers are explained.

Table 3-1 Main Functions of R_CORE Drivers

Function	Description
Interrupt vector table	An interrupt vector table is present that manages entry function addresses when reset cancellation or an IRQ or other interrupt occurs. For details, see section, 7.2.4.1, Interrupt Control.
Startup processing	An entry function after reset cancellation that performs startup processing prior to execution of the main function. In this package, in addition to startup defined in CMSIS-CORE, initial settings for the operation clock and power control mode are made according to the settings in r_core_cfg.h. For details, see section 6.1, Startup Processing.

3.2 CMSIS-Driver

The CMSIS-Driver in this package is a component made up of a driver group that uses headers provided by Arm® and code and some extended headers provided by Renesas.

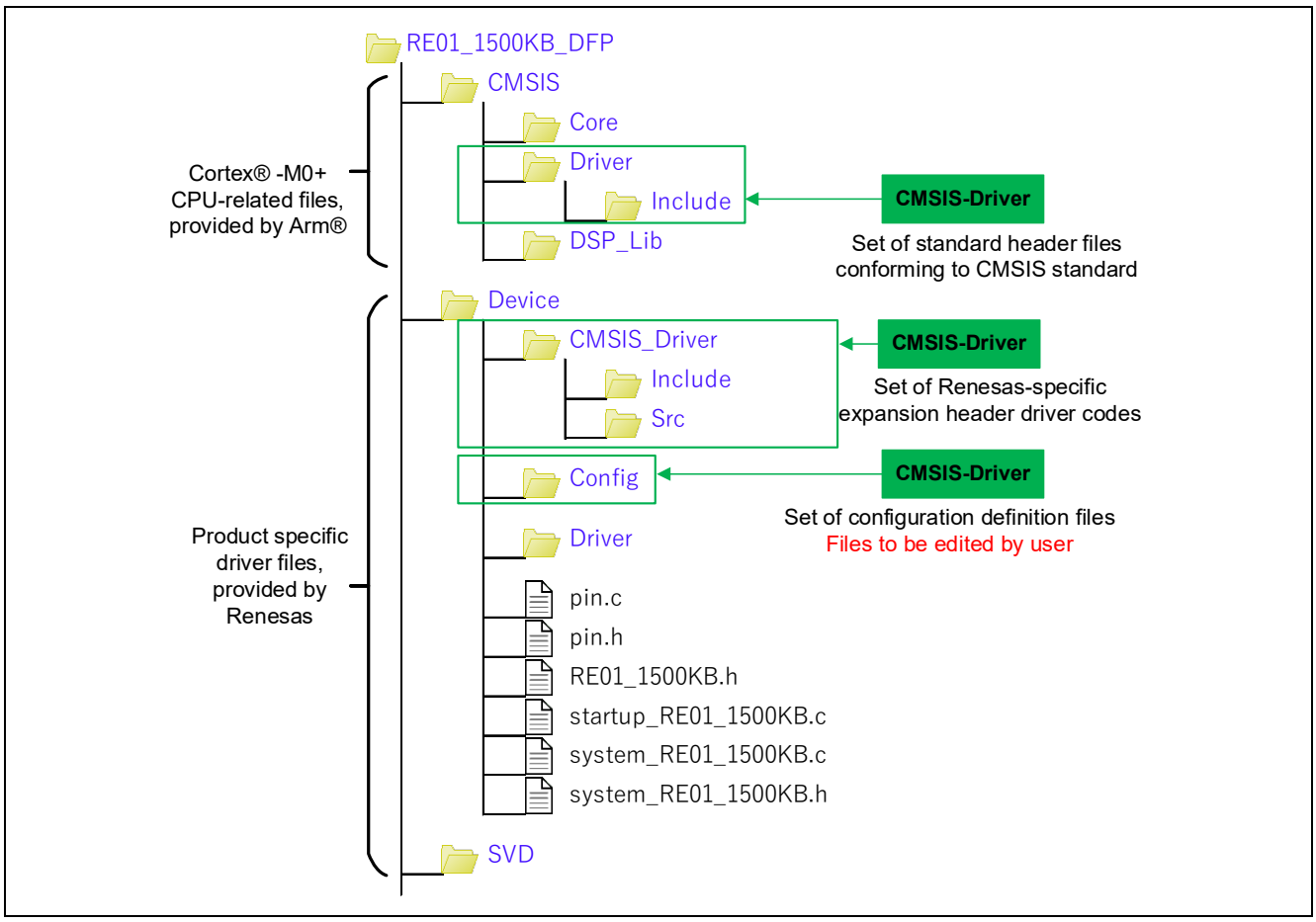


Figure 3-2 CMSIS-Driver Related Files

3.2.1 Supported Drivers

The CMSIS-Driver in this package includes peripheral function drivers. The CMSIS-Driver drivers supported by this package appear in Table 3-2.

Each driver has a configuration definition header in the Config folder. The user can edit the definition values according to the operating environment.

Table 3-2 CMSIS-Driver Supported Drivers

Driver Name	Renesas -Specific Expended Function
R_SPI	No
R_I2C	Yes
R_USART	Yes

3.2.2 Extended Functions

Some of the drivers in the CMSIS-Driver component of this package have extended functions.

When using a driver that has a Renesas-specific extended function, the extended header file should be included. The standard header file is included in the extended header file, so that inclusion of the standard header file is unnecessary.

3.3 CMSIS-DSP

The CMSIS-DSP in this package is a component consisting only of headers provided by Arm®.

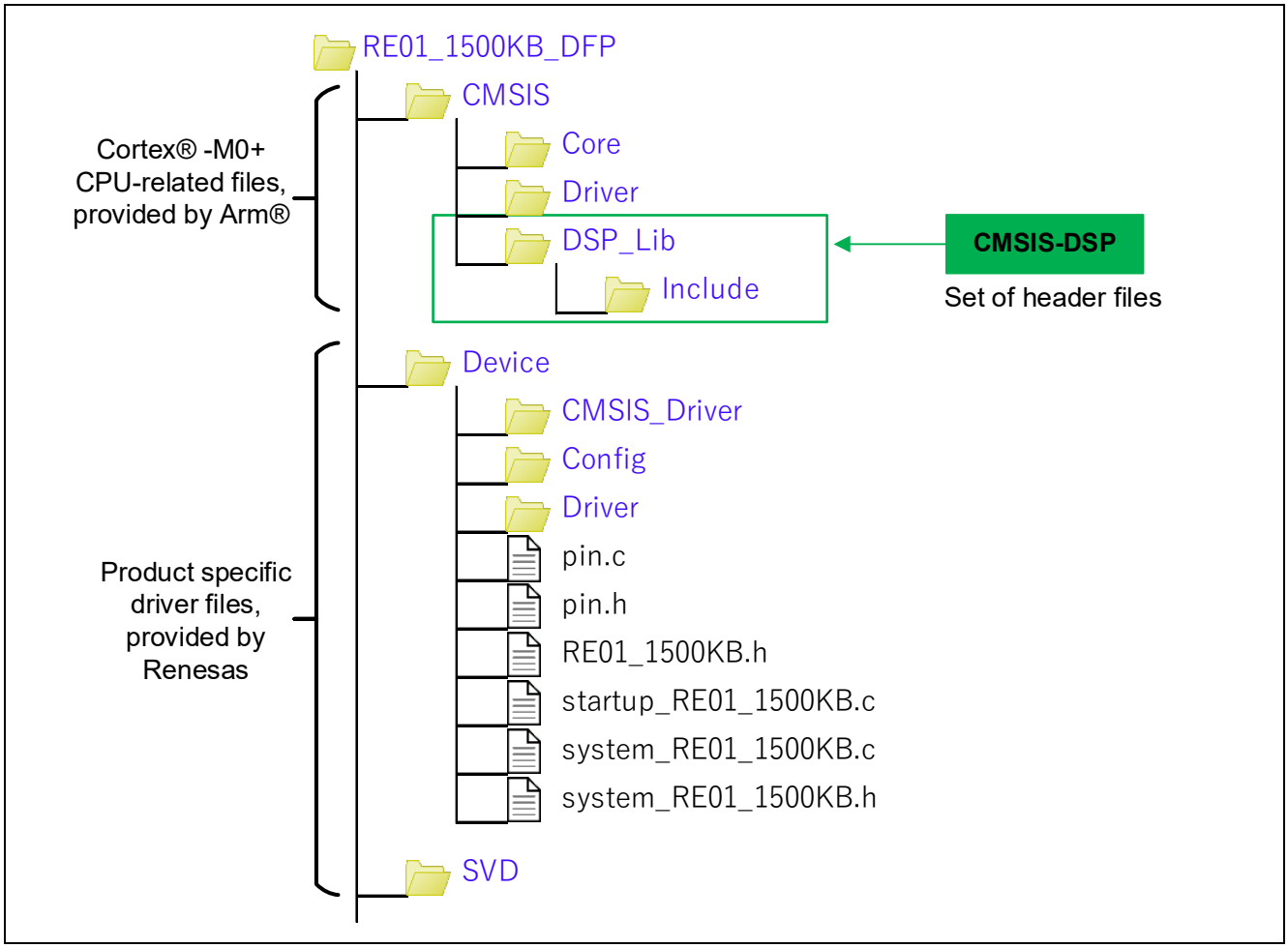


Figure 3-3 CMSIS-DSP Related Files

3.3.1 DSP Library Copying Procedure

This section describes the procedure for copying the DSP library within the CMSIS package provided by Arm® into the project.

3.3.1.1 EWARM version

The DSP library provided by Arm® includes some functions with large code size, and there is the problem that compiling is not possible when using the EWARM free evaluation version license (a version which limits the code size).

This problem can be resolved by using a library generated by compiling DSP source code within the CMSIS package.

Here the procedure for generating a library from DSP source code and copying the library into the project is explained.

See Figure 3-4 for the locations of the files. In Figure 3-4, the CMSIS package provided by Arm® is on the left, and on the right is the project of this package (RE01_1500KB_DFP).

The version of the CMSIS package is set as 5.4.0, but this should be changed appropriately according to the version being used.

(1) Double-click on "arm_cortexM_math.eww" in the CMSIS package provided by Arm® to start up the project.
(2) Confirm that "Cortex-M0" is selected as the processor in the project options and execute compiling. The DSP library "iar_cortexM0l_math.a" is generated.
(3) Copy the DSP header files "arm_common_tables.h," "arm_const_structs.h" and "arm_math.h" and the generated DSP library "iar_cortexM0l_math.a" into the project of this package.

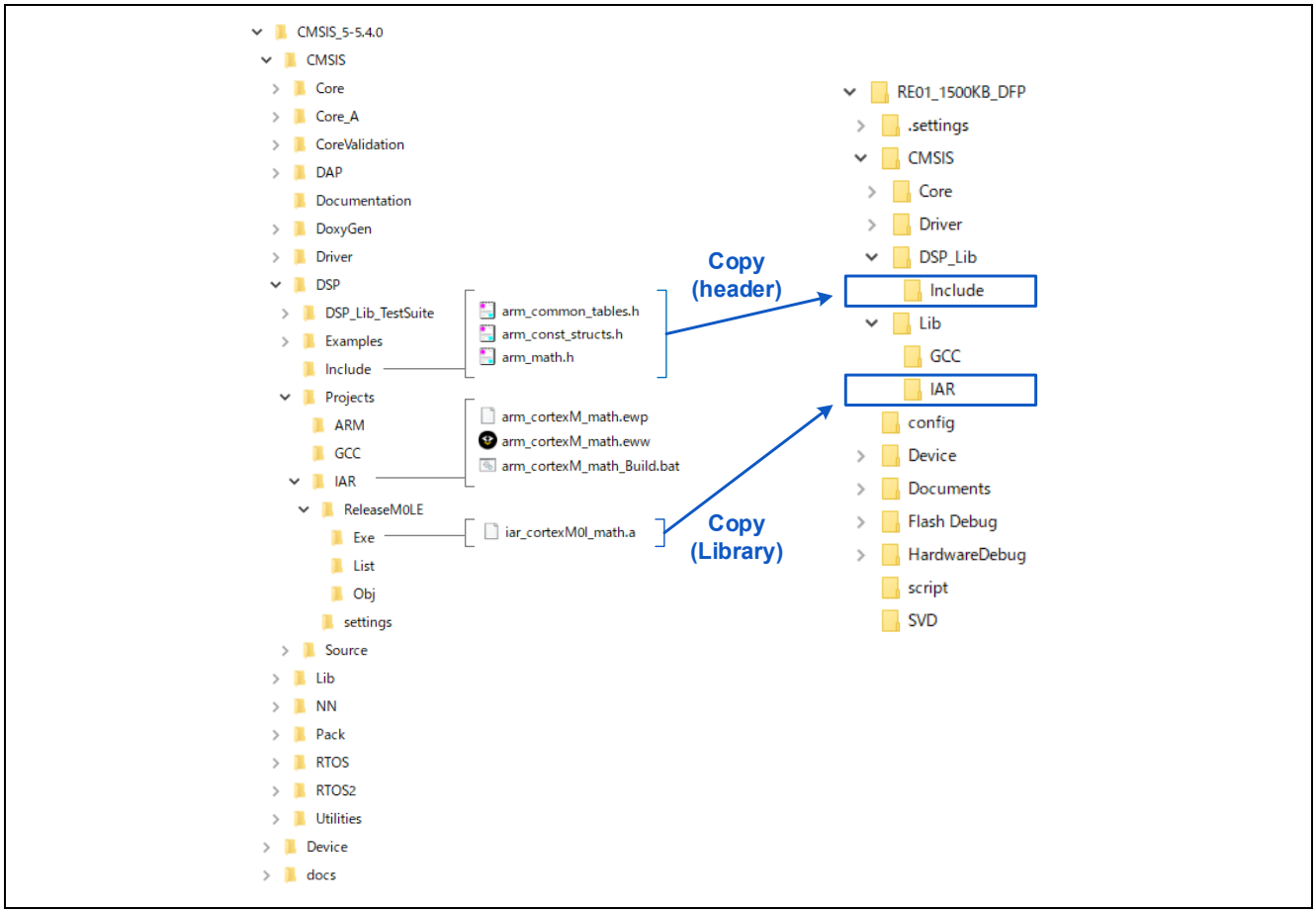


Figure 3-4 DSP Library Copying (for IAR Compiler)

(4) Add the DSP library as a build file to the project of this package.

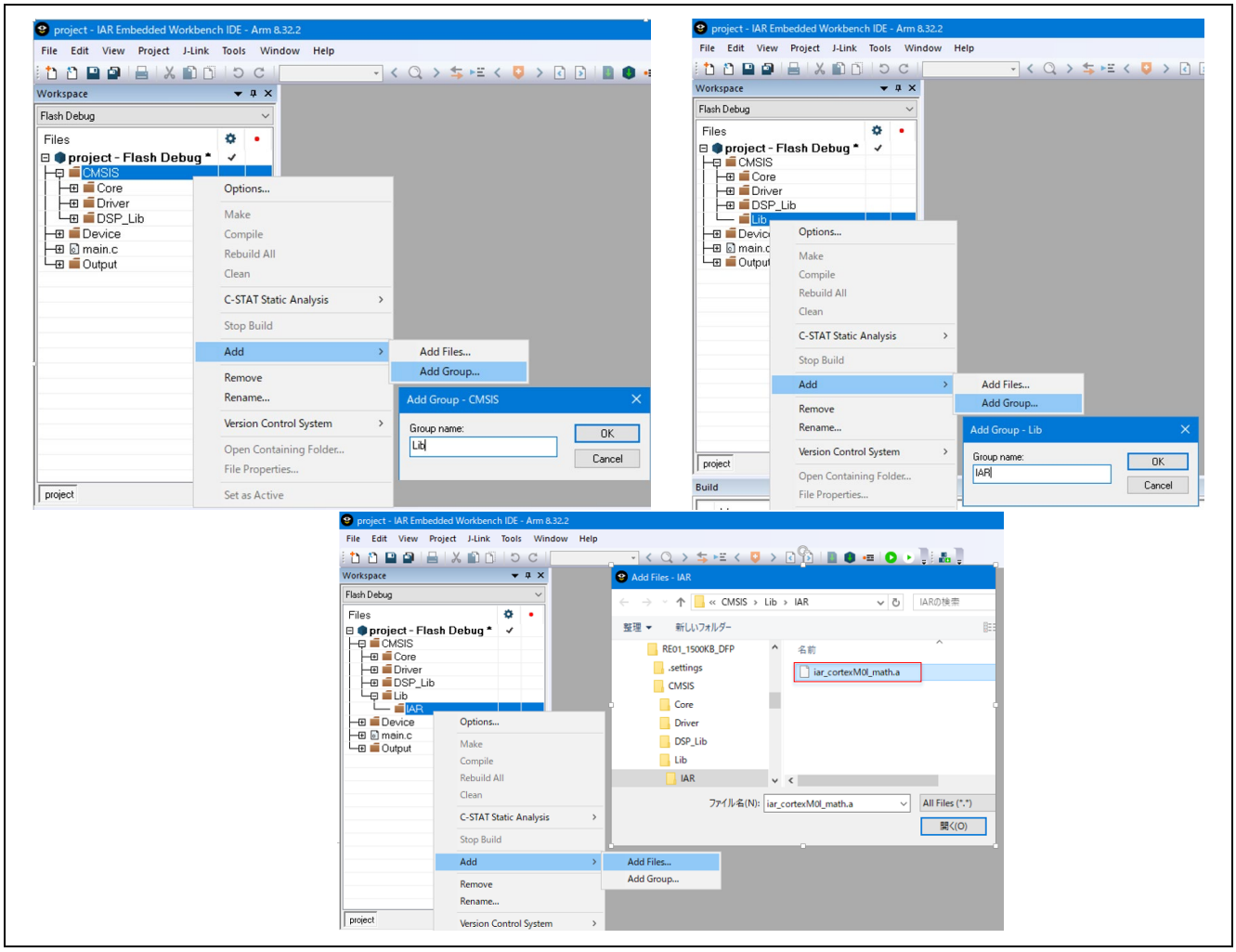


Figure 3-5 Addition of Build File in EWARM

(5) The following settings are made in project options.

- The folder path in which DSP header files are stored is added to the include paths.
- "ARM_MATH_CM0PLUS" is added to the preprocessor.

See Figure 3-6 for option settings in EWARM.

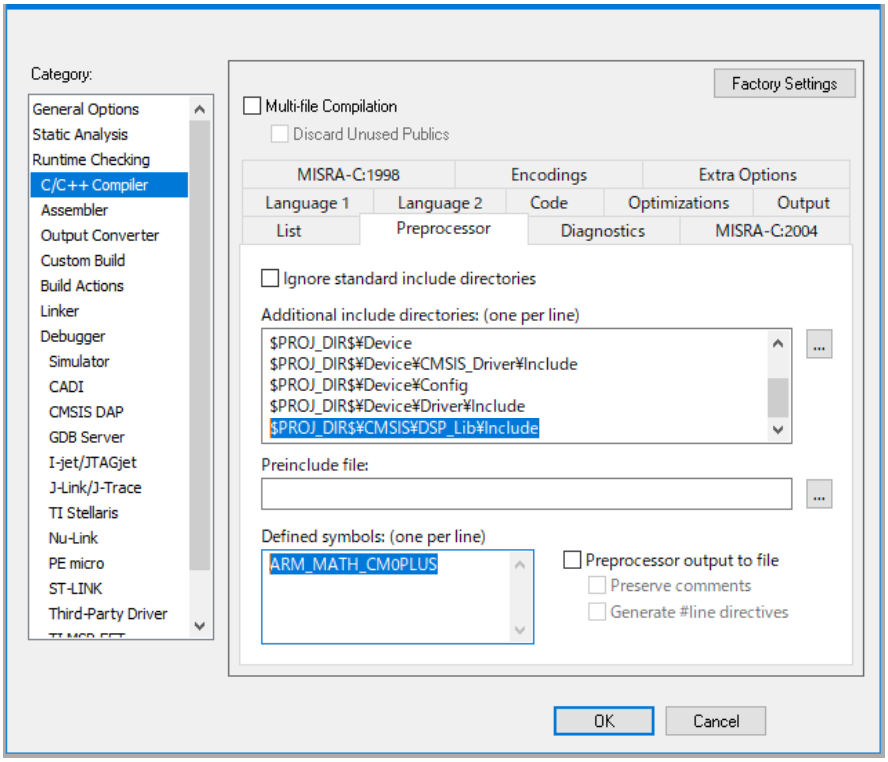


Figure 3-6 Include Path and Preprocessor Settings in EWARM

3.3.1.2 e² studio version

See Figure 3-7 for the locations of the files. In Figure 3-7, the CMSIS package provided by Arm® is on the left, and on the right is a project in this package (RE01_1500KB_DFP).

The version of the CMSIS package is set as 5.4.0, but this should be changed appropriately according to the version being used.

(1) Copy the DSP header files "arm_common_tables.h," "arm_const_structs.h" and "arm_math.h" and the DSP library "iar_cortexM0l_math.a" into the project of this package.

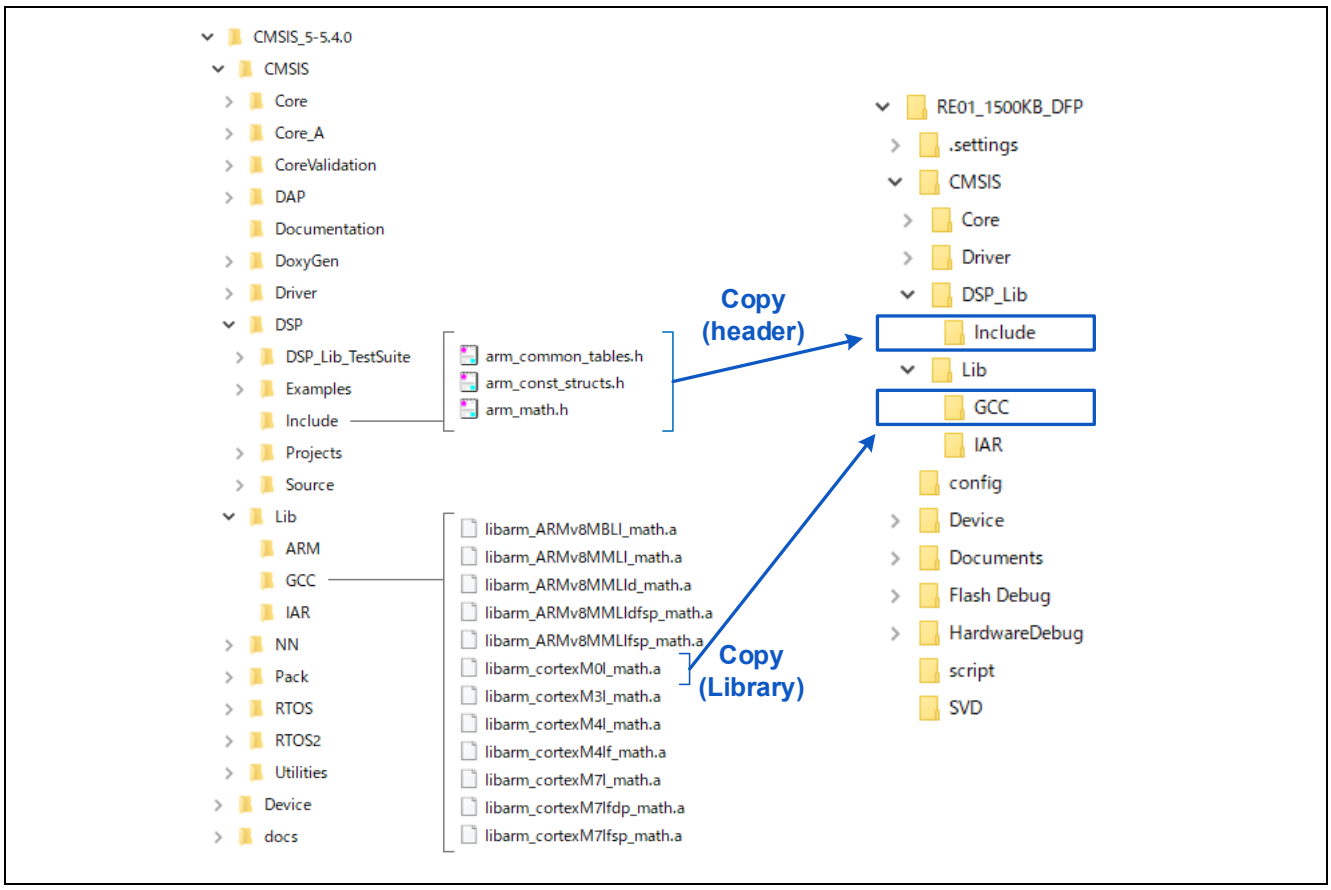


Figure 3-7 DSP Library Copying (for GCC Compiler)

(2) Make the following settings in project properties.
Add the folder path in which the DSP header files are stored to the include paths.

- Add "ARM_MATH_CM0PLUS" to the preprocessor.
- Add the DSP filename and the folder path in which the DSP library files are stored to the library.

For property settings in e^2 studio, see Figure 3-8 to Figure 3-10.

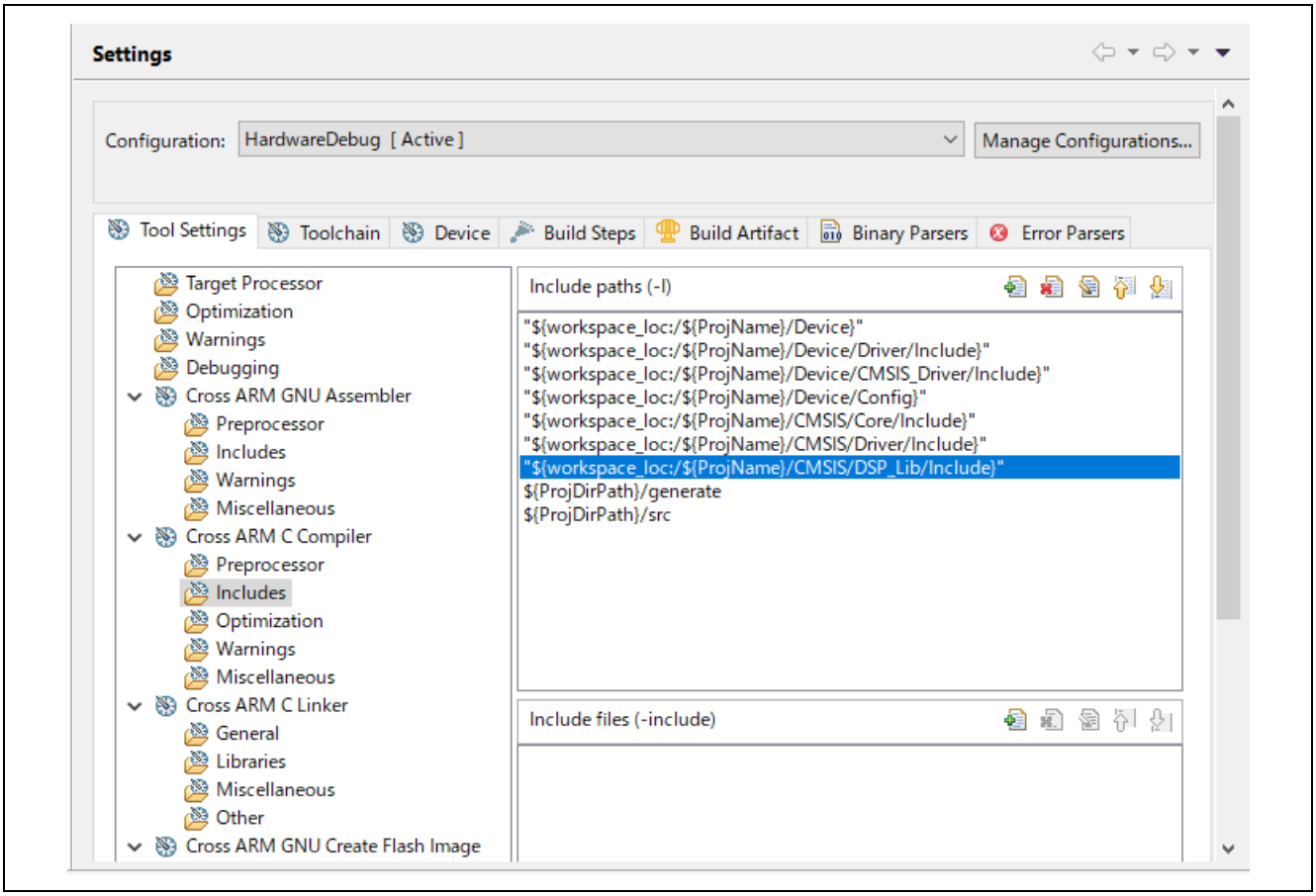


Figure 3-8 e^2 studio Include Path Settings

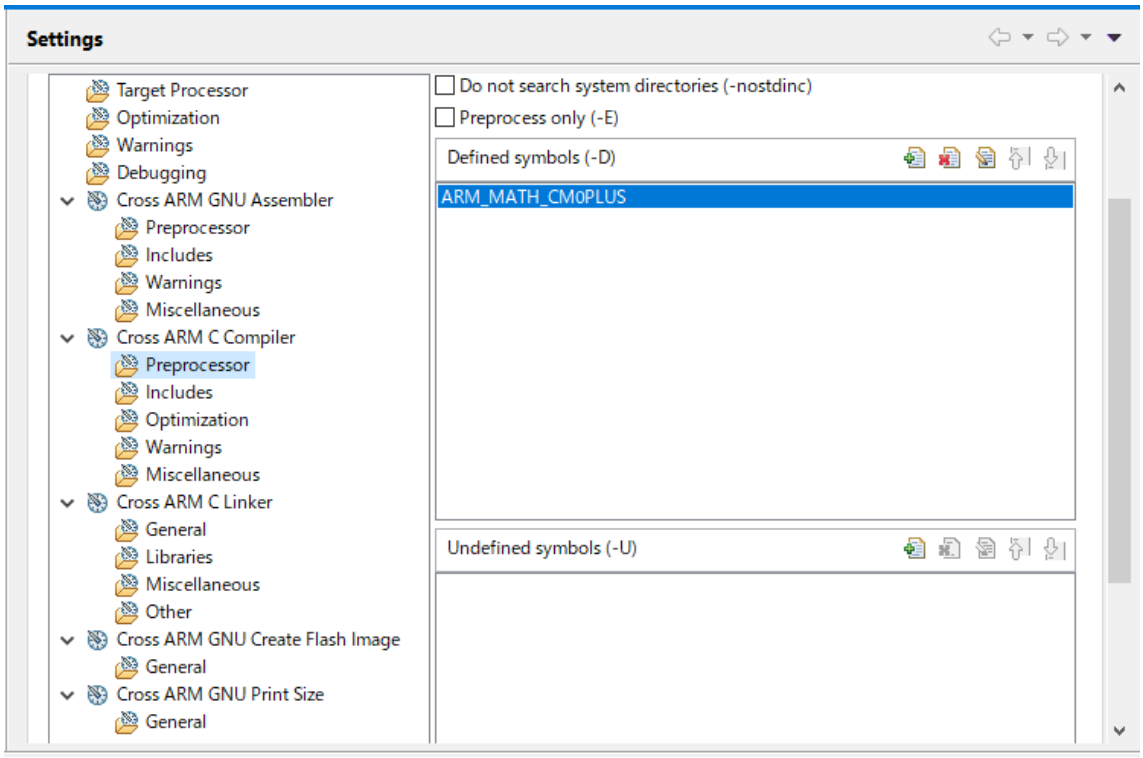


Figure 3-9 Preprocessor Setting in e^2 studio

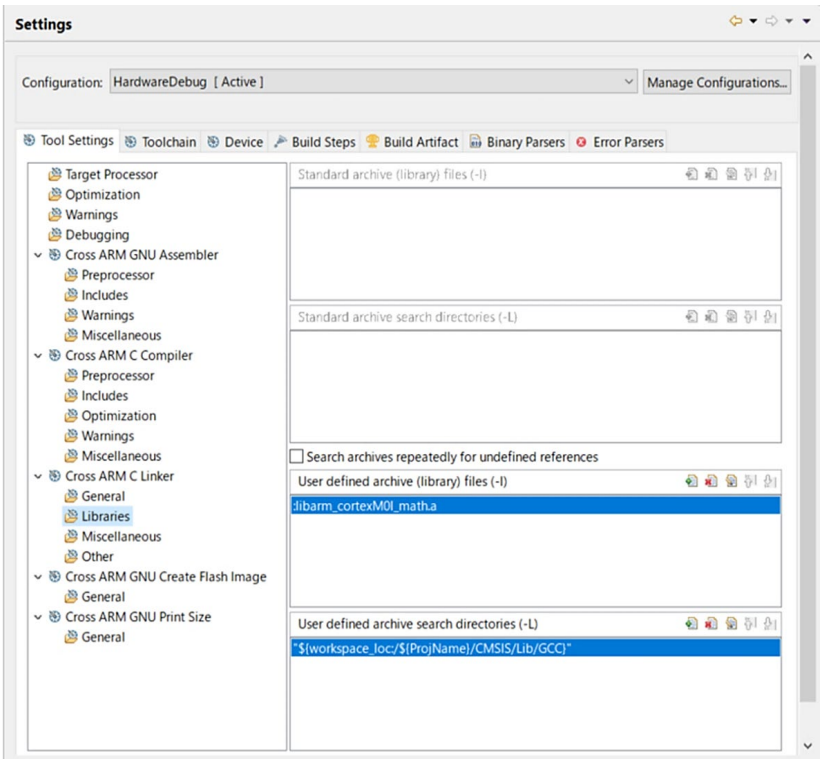


Figure 3-10 Library Setting in e^2 studio

3.4 HAL-Driver

The HAL-Driver of this package is a component consisting of drivers the headers and code for which are both provided by Renesas.

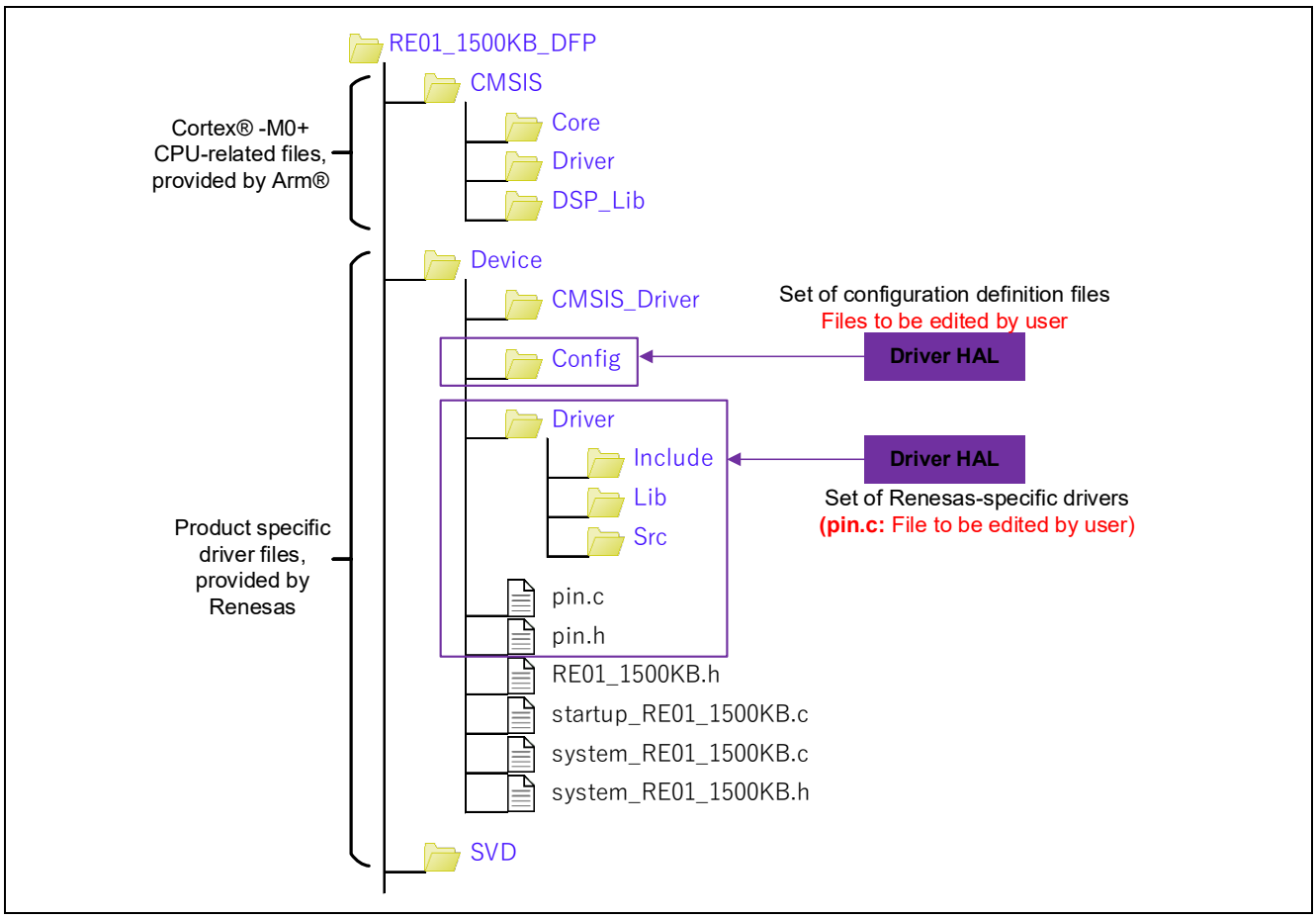


Figure 3-11 HAL-Driver Related Files

3.4.1 Supported Drivers

The Driver HAL of this package includes common function drivers and peripheral function drivers. The Driver-HAL drivers supported in this package are shown in Table 3-3.

Each driver has a configuration definition header in the Config folder; the user can edit the definition values according to the operating environment.

Table 3-3 HAL-Driver Supported Drivers

Driver Name	Category
R_SYS	Common function drivers
R_LPM	
R_PIN	
R_ADC	Peripheral function drivers
R_DMAC	
R_DTC	
R_FLASH	
R_GDT	
R_SMIP	
R_PMIP	

3.4.2 Common Function Drivers

Common function drivers have common functions that are used by user programs and drivers.

3.4.2.1 Main Functions of R_SYS Drivers

Table 3-4 shows the main functions of R_SYS drivers.

Table 3-4 Main Functions of R_SYS Drivers

Function	Description
Clock setting	A function to set the clock is provided. For details, see section 6.4, Clock Settings.
Interrupt setting	A function to control interrupts and a definition file are provided. For details, see section 6.3, Interrupt Control.
Program expansion in RAM	A function is provided that copies to a RAM area a program that has been placed in a section for RAM placement. For placing a program in RAM, see section 6.6, RAM Placement of Programs..

3.4.2.2 Main Functions of R_LPM Driver

Table 3-5 shows the main functions of R_LPM drivers.

Table 3-5 Main Functions of R_LPM Drivers

Function	Description
Sets the undefined value propagation suppression function for the I/O power supply domain	A function is provided for suppressing undefined value propagation in an I/O power supply domain. For details, see section 6.2, Control of Undefined Value Propagation Suppression in I/O Power Supply Domains.
Sets the low power consumption mode	A function is provided for controlling the low power consumption mode.

3.4.2.3 Main Functions of R_PIN Drivers

Table 3-6 shows the main functions of R_PIN drivers.

Table 3-6 Main Functions of R_PIN Drivers

Function	Description
Pin settings	A function is provided for controlling pins used by peripheral functions. For details, see section 6.5, Pin Settings.

3.4.3 Peripheral Function Drivers

Peripheral function drivers have peripheral functions that are used by user programs.

4. Driver Specifications

This package includes specifications for the drivers. The driver specification locations are shown below.

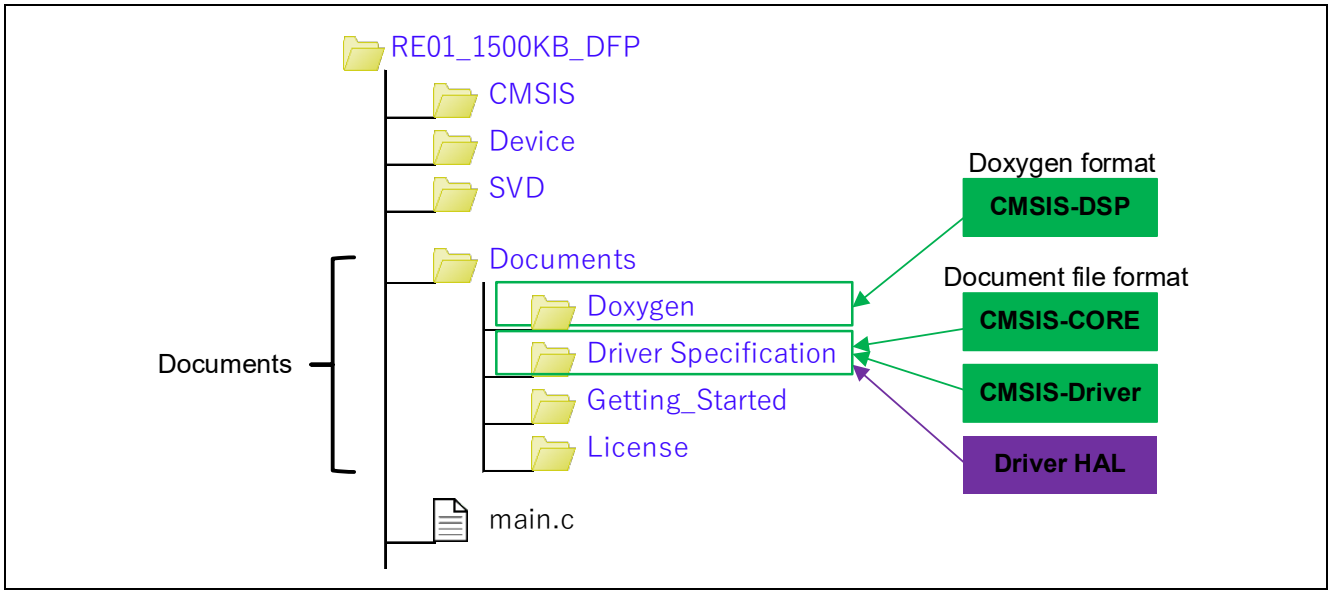


Figure 4-1 Files Related to Driver Specifications

Documentation for the DSP library supplied by Arm® is provided using Doxygen. In this package, Doxygen-related files are compressed.
The procedure for displaying the DSP library specifications is shown in Figure 4-2.

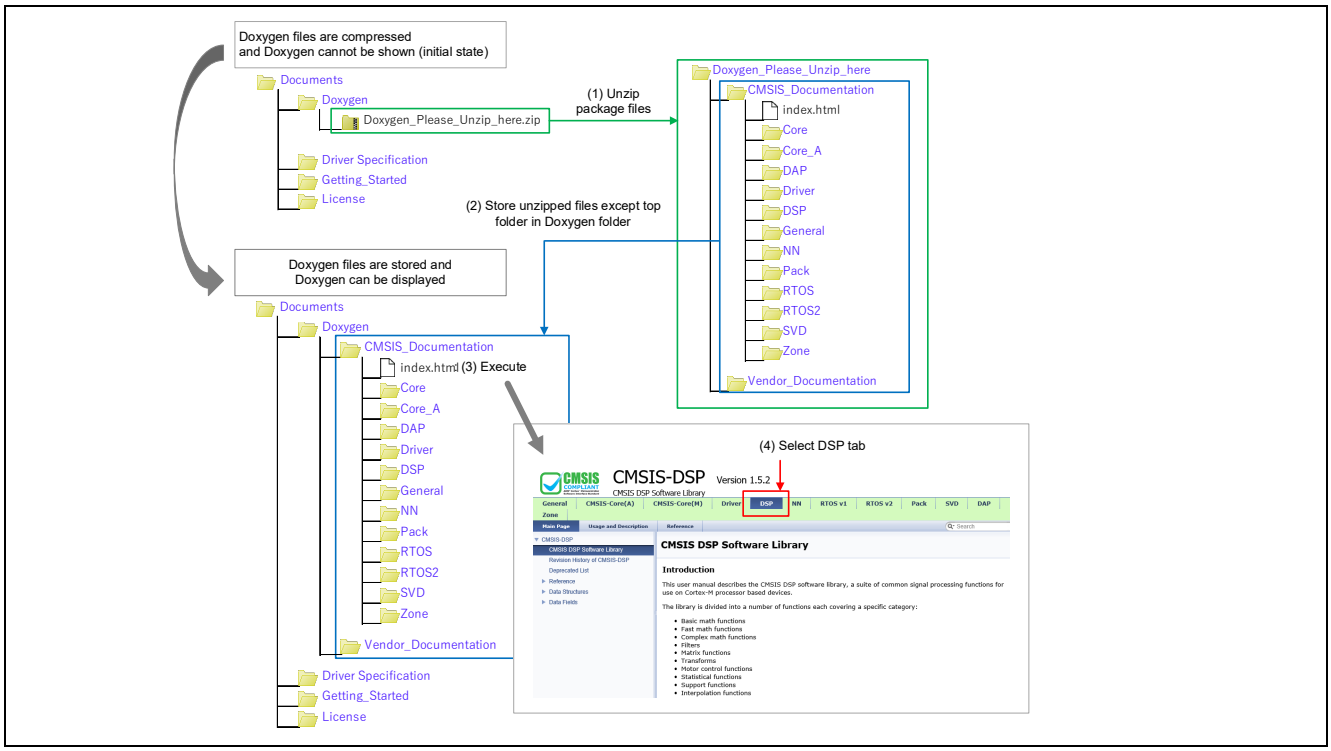


Figure 4-2 Method for Displaying CMSIS-DSP Specifications Using Doxygen

5. Driver Basic Concepts

5.1 Common Function Drivers and Peripheral Function Drivers

Drivers included in this package are of two kinds, common function drivers and peripheral function drivers.

Table 5-1 Driver Categories

Category	Description
Common function drivers	Drivers having common functions used by user programs and drivers These drivers include: • R_CORE driver • R_SYS driver • R_PIN driver • R_LPM driver
Peripheral function drivers	Drivers having peripheral functions used by user programs These drivers include: • All drivers other than the common function drivers described above

5.2 Driver Configuration

Each driver (with some exceptions) is configured from three types of file.

Table 5-2 Files Constituting Drivers

Filename	Description
r_***_api.c	Body of code for the driver
r_***_api.h	Include header that makes necessary definitions when using the driver The user must include this file when using the driver.
r_***_cfg.h	Configuration definition header defining operating conditions for the driver The user can edit the definition values in this file according to the operating environment. The user need not include this file. The configuration definition header for the R_CORE driver is used in startup processing and by the R_SYS driver.

【Note】In filenames, "***" represents the driver function name.

5.3 Common Function Settings

In peripheral function drivers, functions of common function drivers are executed internally to set common functions.

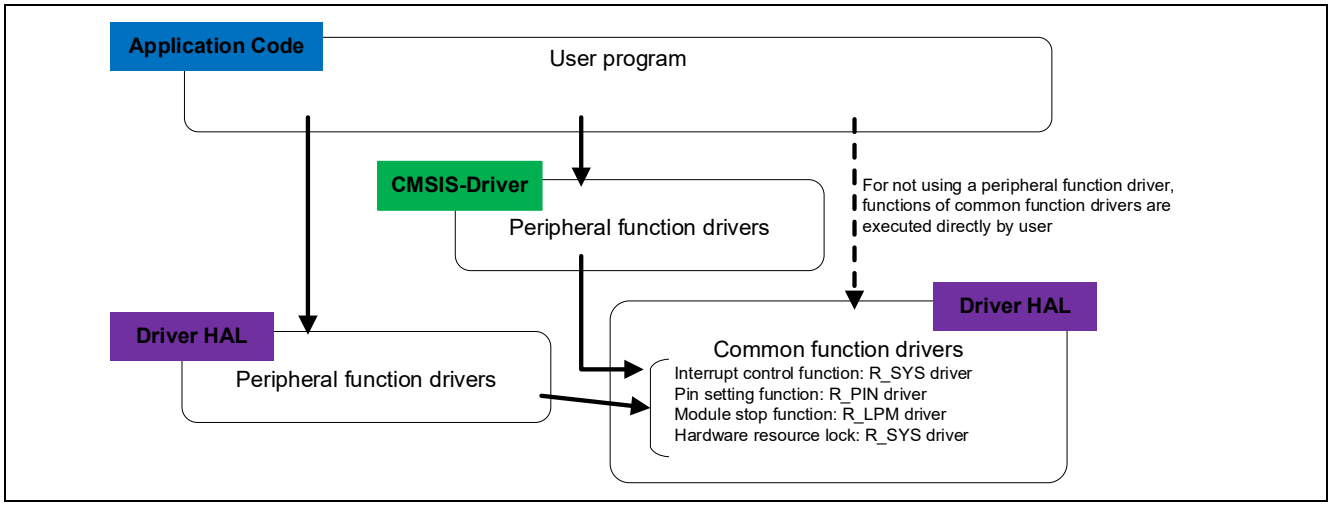


Figure 5-1 Setting Common Functions

6. Basic Functions

6.1 Startup Processing

When using this package, a Reset_Handler function, which is registered as an entry function after reset cancellation, is called. The Reset_Handler function executes startup processing before execution of the main function. The flow of startup processing is shown in Figure 6-1.

In startup processing, the following processing is mainly performed.

- Pin settings upon start of operation
- Setting of the clock and power control mode upon start of operation

Startup processing is performed prior to execution of the main function. At the time when the main function is executed, there are cases in which hardware register values are changed from those after reset, so caution must be exercised.

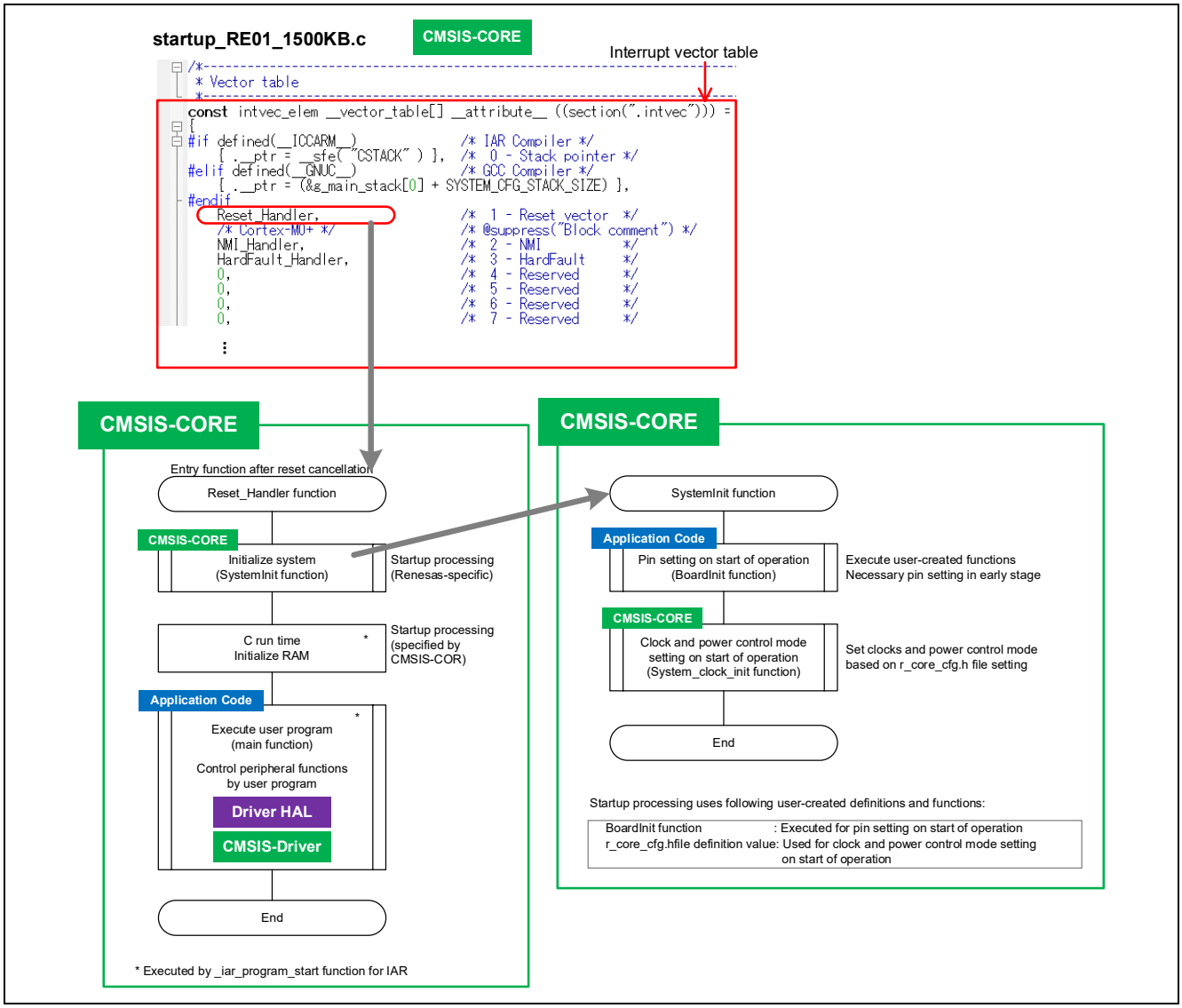


Figure 6-1 Flow of Startup Processing

6.1.1 Pin Settings Upon Start of Operation

In startup processing, a Boardinit function created by the user is executed. When pin settings must be made at an early stage after reset cancellation, the Boardinit function should be created and pin processing performed.

BoardInit function

- Function format: void BoardInit(void)
- Description: User-created function executed in the startup processing before execution of the main function
- Features: Because a weak BoardInit function is provided in the R_CORE driver, a compile error does not occur even if the user does not create this function.
- Setting example: An example of creation of this function appears in Figure 6-2.

User program **Application Code**

```
/*****************************************************************************************************************
* Function Name: BoardInit
* Description  : Configure board initial setting.
*                This function is called by SystemInit() function in system_RE01_1500KB.c file.
*                This is reference to perform BoardInit process. Sample code of target is E015 SDK board(RTK70E015D)
*                 on Renesas. Please modify this function to suit your board.
* Arguments    : none
* Return Value : none
*****************************************************************************************************************/
void BoardInit(void)
{
    /*****   This function performs at beginning of start-up after released reset pin *****/
    /***** Please set pins here if your board is needed pins setting at the device start-up. ****/
    /*****   This function is suiting RE01_1500KB SDK board. Please change to your board pin setting *****/

    /* Handling of Unused ports (IOVCC domain) */
    /* PORT4 Setting: RE01_1500KB SDK has DCDCs. Those are connect P404 and P405. */
    /* This perform to disable those output.
    /* Those are needed to enable when using EHC start up of this
    /* Set P404 and P405 not to be used as DCDC_EN (output low: D

    /* PODR - Port Output Data
       b15-b0   PODR15 - PORD00     - Output Low Level */
    PORT4->PODR = 0x0000;

    /* PDR - Port Direction
       b15-b6   PDR15 - PRD06       - Input
       b5 -b4   PDR05 - PRD04       - Output
       b3 -b0   PDR03 - PRD00       - Input */
    PORT4->PDR  = 0x0030;

    /* Handling of Unused pors (AVCC1 domain) */

    /* PORT0 Setting */
    /* Set P009, P008 and P007 as LEDs (output high) */

    /* PODR - Port Output Data
       b15-b10  PODR15 - PORD10     - Output Low Level
       b9 -b7   PODR09 - PORD07     - Output High Level
       b6 -b0   PODR06 - PORD00     - Output Low Level */
    PORT0->PODR = 0x0380;

    /* PDR - Port Direction
       b15-b10  PDR15 - PRD10       - Input PORT
       b9 -b7   PDR09 - PRD07       - Output PORT
       b6 -b0   PDR06 - PRD00       - Input PORT */
    PORT0->PDR = 0x0380;
} /* End of function BoardInit */
```

Set P404 and P405 connected to DCDC on the board to LOW output of general port

Set P007 and P008 connected to LED on the board to HIGH output of general port

Figure 6-2 Example of Creation of Pin Setting Function on Start of Operation

6.1.2 Setting Clock/Power Control Modes on Start of Operation

In startup processing, initial settings for the clock and power control modes are made according to the settings in r_core_cfg.h. The user should edit definition values in r_core_cfg.h according to the operating environment.

r_core_cfg.h: Setting the clock/power control modes on start of operation

- Description: Sets the clock/power control modes on start of operation
- Definition values: For the definition names and values, see section 6.4.1, Clock Definitions. Power control modes that can be selected as the mode on start of operation, and the clock/power control mode state selected by an initial value, appear in Figure 6-3.
- Setting example: Shown in Figure 6-4.

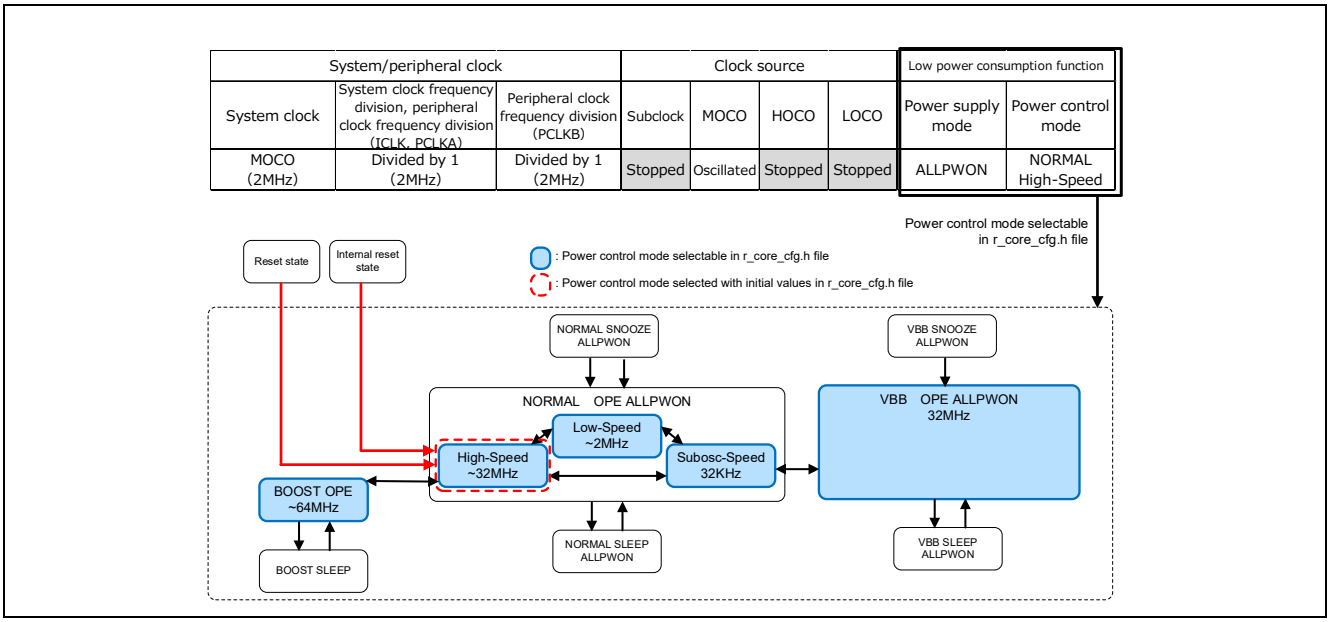


Figure 6-3 Clock and Power Control Modes that can be Selected in r_core_cfg.h

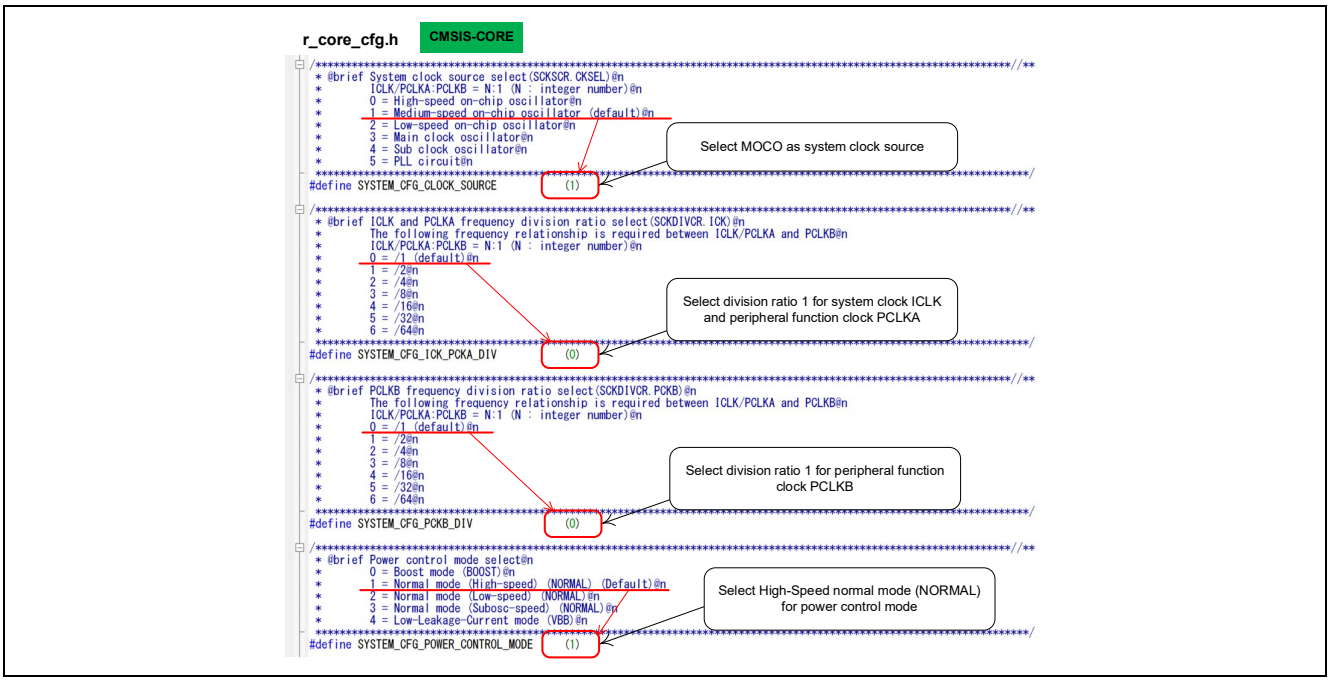


Figure 6-4 Example of Settings of Clock and Power Control Mode on Start of Operation

6.2 Control of Undefined Value Propagation Suppression in I/O Power Supply Domains

This device has multiple I/O power supply domains, and power to each domain can be supplied or shut off. In addition, there is an undefined value propagation suppression function that suppresses the propagation of undefined values from domains to which there is no supply of power.

After reset cancellation, the undefined value propagation suppression function is enabled for all I/O power supply domains other than the IOVCC domain, and pins cannot be used even if power is supplied to an I/O power supply domain. This function must be controlled according to the connection status of the power supplies.

- Domains to which power is being supplied: Undefined value propagation suppression function should be disabled
- Domains to which power is not being supplied: Undefined value propagation suppression function should be enabled

The Evaluation Kit allows the following selections using jumper settings.

- Power supplied to all power supply domains other than IOVCC (for normal startup)
- No power supply to all power supply domains other than IOVCC until external DC/DC is enabled (for energy harvesting startup)

6.2.1 Applicable Power Supply Domains

Hardware functions and pins disposed in different I/O power supply domains are shown in Table 6-1. For power supply domains applicable to the pins, refer to the List of Pins and Pin Functions in the User's Manual: Hardware of the relevant device.

Table 6-1 Hardware Functions and Pins Disposed in I/O Power Supply Domains

I/O Power Supply Domain	Hardware Function/Pin
IOVCC	All hardware functions/pins other than those below
AVCC0	14-bit A/D converter (S14AD) Temperature sensor circuit (TEMPS) Reference voltage generation circuit (VREF)
AVCC1	12-bit D/A converter (R12DA) Analog comparator (ACMP)
IOVCC0	I/O functions allocated to port 8
IOVCC1	I/O functions allocated to ports 3, 6, 7, and P202 to 204
IOVCC2	I/O functions allocated to port 1
IOVCC3	I/O functions allocate to ports P010 to P015, and 5
USB	USB2.0FS host/function module (USB)
VPM	Motor driver control circuit (MTDV)

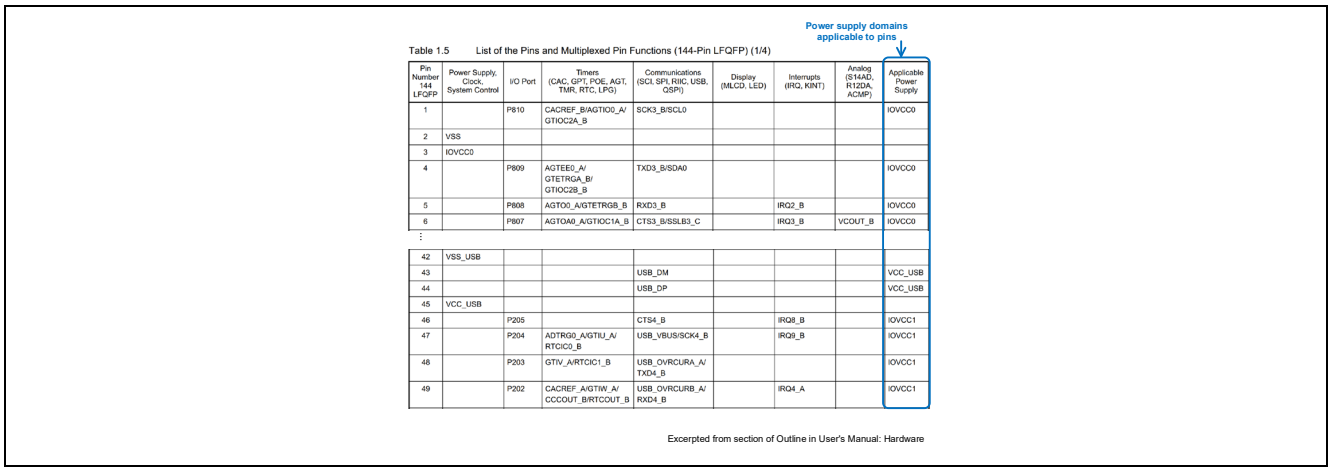


Figure 6-5 Confirmation of Power Supply Domains Applicable to Pins

6.2.2 Driver Functions

The R_LPM driver provides an undefined value propagation suppression control function for I/O power supply domains.

R_LPM_IOPowerSupplyModeSet function

- Description: Enables the undefined value propagation suppression function for a specified domain
- Argument: Specifies the I/O power supply domain for which the undefined value propagation suppression function is enabled
  0x00: Disables the undefined value propagation suppression function for all power supply domains other than IOVCC
- Setting example: Figure 6-6 shows an example of a case in which, because power is being supplied only to IOVCC and IOVCC0, the undefined value propagation suppression function is set to “disabled” for the IOVCC0 domain.

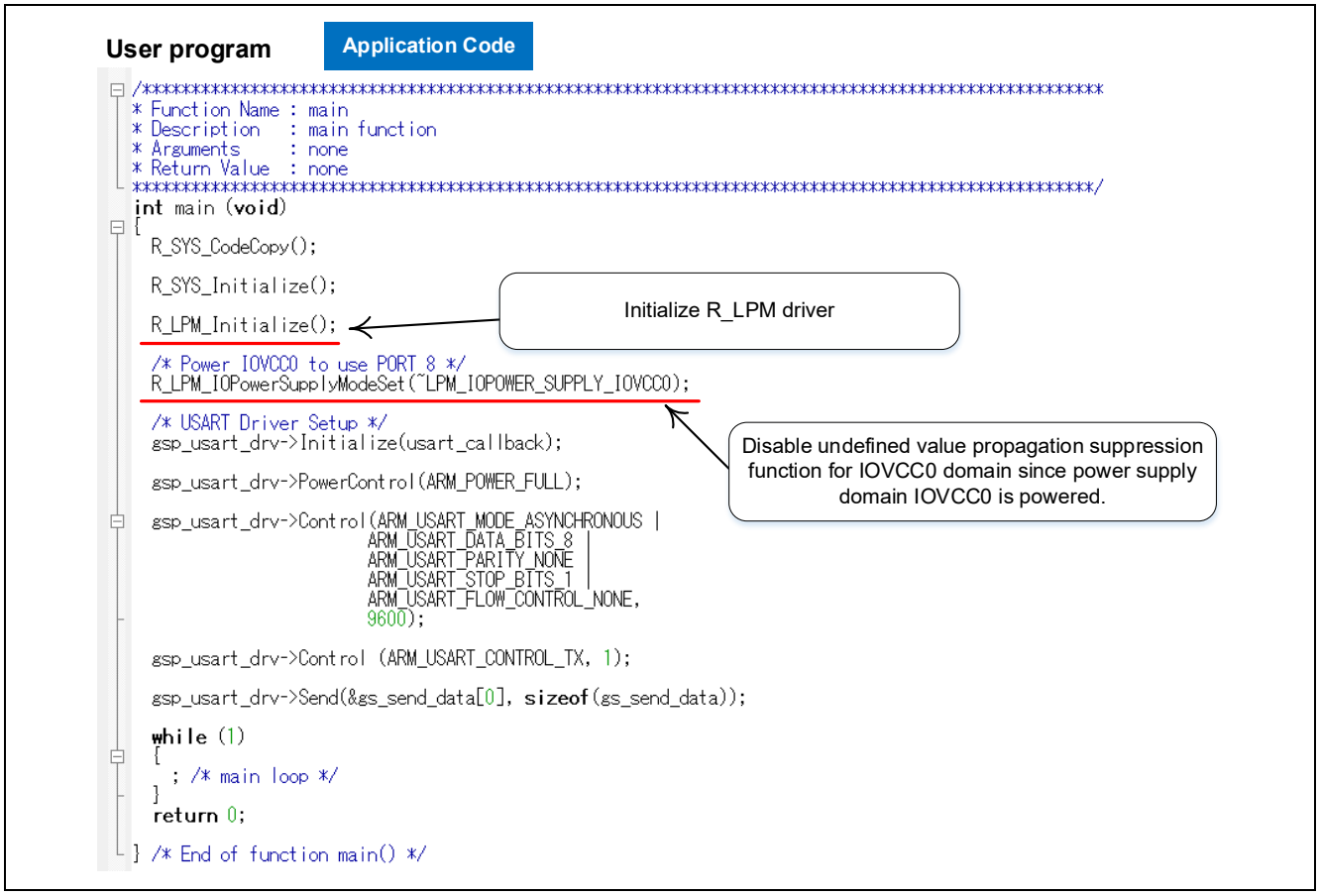


Figure 6-6 Example of Control of Undefined Value Propagation Suppression Function for I/O Power Supply Domain

6.3 Interrupt Control

6.3.1 Interrupt Vector Table and Entry Functions

In this package, multiple drivers perform interrupt control. A vector table that defines entry functions when an interrupt occurs is provided by the R_CORE driver. Entry functions upon interrupt occurrence are provided by the R_CORE driver or the R_SYS driver, depending on the type of interrupt.

The relationship between the interrupt vector table and entry functions is indicated in Figure 6-7.

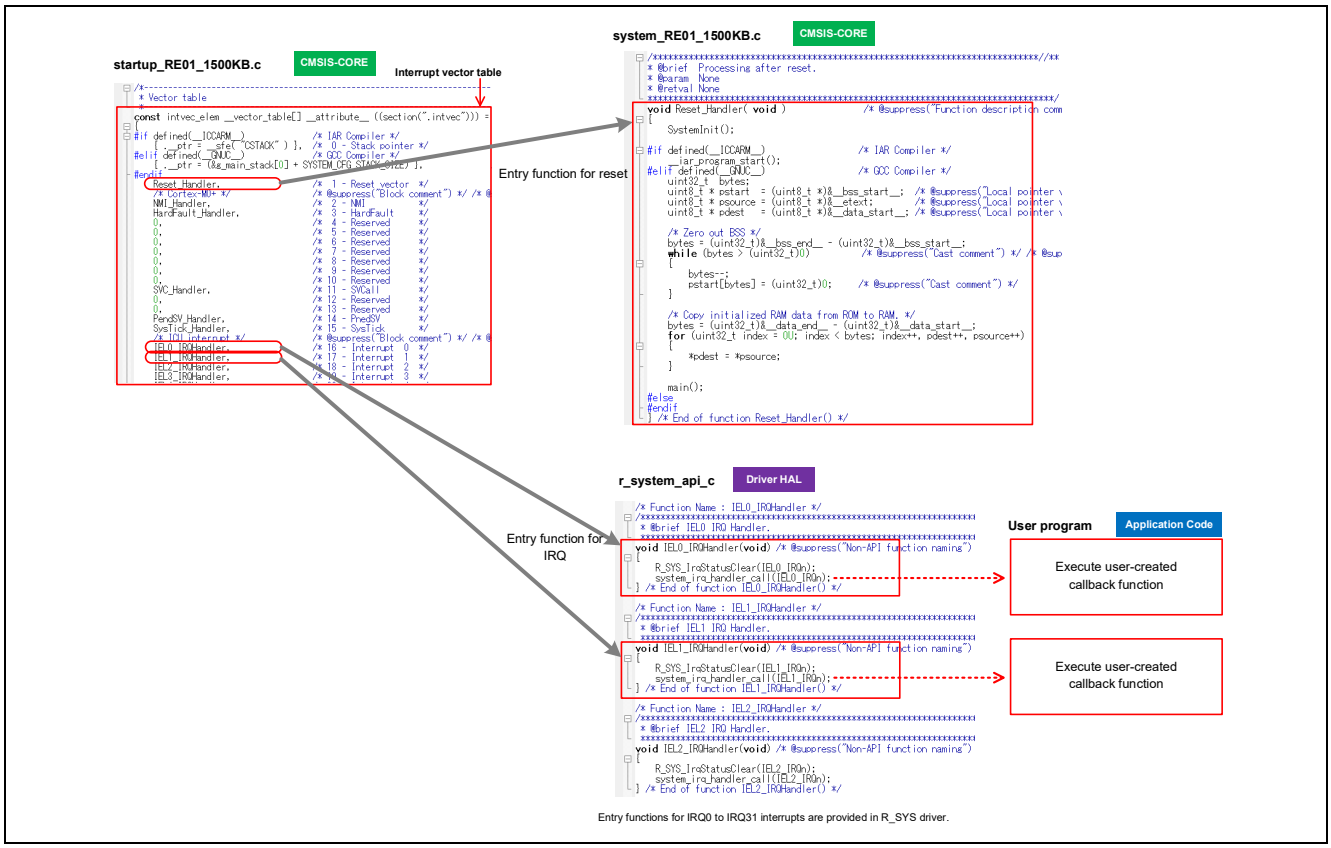


Figure 6-7 Interrupt Vector Table and Entry Functions

6.3.2 IRQ Number Allocation

When using interrupts in this device, interrupts from peripheral functions must be allocated to IRQ numbers. Using as an example a case in which interrupts from AGT0 (AGT0_AGTI) are used, peripheral function interrupts and connections with the Cortex®-M0+ processor are illustrated in Figure 6-8.

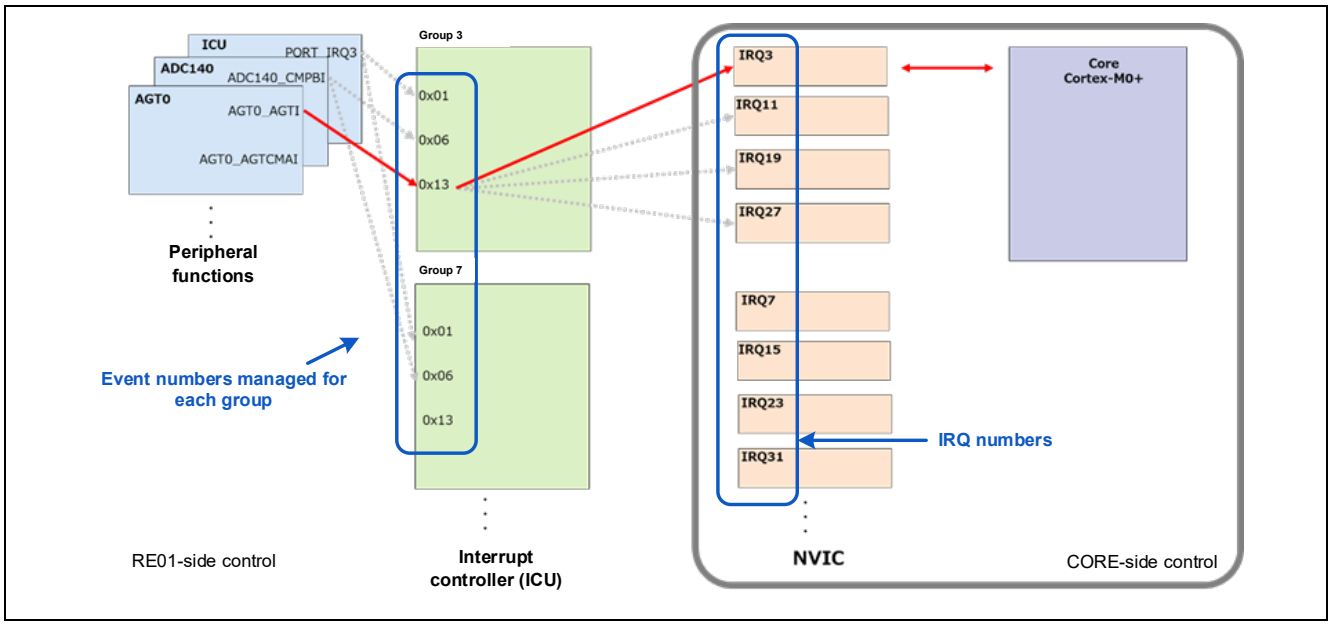


Figure 6-8 Peripheral Function Interrupts and Connections to the Cortex®-M0+

For the IRQ numbers that can be connected to interrupts from peripheral functions, refer to the Interrupt Controller Unit (ICU) section in the User's Manual: Hardware of the relevant device.

Taking AGT1_AGTI as an example, Figure 6-9 indicates the method for confirming IRQ numbers that can be connected to peripheral function interrupts.

Table 16.7 Register Set Values of Each Event Selection (1/5)

Name	IELSRn.IELS[4:0]								DELSRn. DELS[7:0] (n = 0 to 3)	SELSR0. SELS[7:0]
	Group 0 (n = 0/8/ 16/24)	Group 1 (n = 1/9/ 17/25)	Group 2 (n = 2/10/ 18/26)	Group 3 (n = 3/11/ 19/27)	Group 4 (n = 4/12/ 20/28)	Group 5 (n = 5/13/ 21/29)	Group 6 (n = 6/14/ 22/30)	Group 7 (n = 7/15/ 23/31)		
PORT_IRQ0	01h				01h				01h	
⋮	(1)				(2)					
AGT0_AGTI				13h					18h	
AGT0_AGTCMBI		13h							1Ah	
AGT1_AGTI	06h				06h				1Bh	
AGT1_AGTCMAI		05h				05h			1Ch	
AGT0_AGTCMAI			04h				04h		1Dh	

Excerpted from section of Interrupt Controller Units (ICU) in User's Manual: Hardware

The above table indicates that:

(1) The AGT1_AGTI interrupt belongs to ICU group 0 and AGT1_AGTI interrupt event number in group 0 is 0x06
(interrupts in group 0 can be connected to IRQ0, IRQ8, IRQ16, and IRQ24)

(2) The AGT1_AGTI interrupt also belongs to ICU group 4 and AGT1_AGTI interrupt event number in group 4 is 0x06
(interrupts in group 4 can be connected to IRQ4, IRQ12, IRQ20, and IRQ28)

Figure 6-9 IRQ Number Confirmation Method

6.3.3 r_system_cfg.h Editing

The R_SYS driver provides, in r_system_cfg.h, definitions relating the peripheral function interrupts to IRQ numbers. In the initial state, all of the peripheral function interrupts are set to "no IRQ allocation". When using interrupts, the definition values in r_system_cfg.h should be edited according to the operating environment.

r_system_cfg.h: IRQ number definitions

- Description: Associates IRQ numbers with peripheral function interrupts
- Definition names: SYSTEM_CFG_EVENT_NUMBER_XXX
  where "XXX" is the peripheral function interrupt name
- Definition value:
  SYSTEM_IRQ_EVENT_NUMBER_NOT_USED: Interrupt not allocated to IRQ (initial value)
  SYSTEM_IRQ_EVENT_NUMBERn: Interrupt allocated to IRQn (n = 0 to 31)
  The same IRQ number cannot be allocated to multiple interrupts.
- Setting example: A setting example in which the AGT1 interrupt, AGT1_AGTI, is allocated to IRQ0 is shown in Figure 6-10.

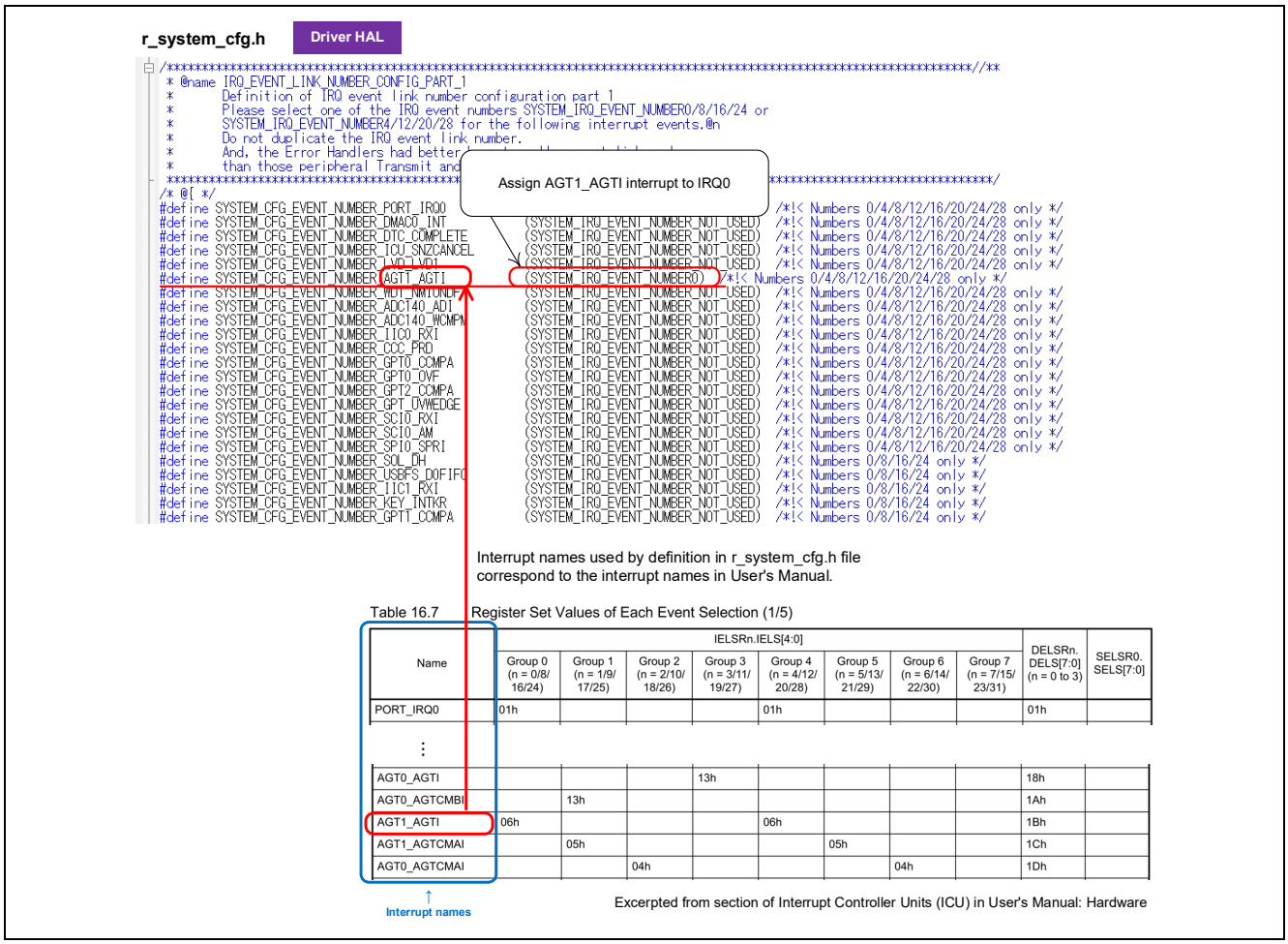


Figure 6-10 Interrupt Name Confirmation Method and Example of IRQ Number Allocation

6.3.4 Driver Functions

The R_SYS driver provides interrupt control functions. The main functions are here explained.

R_SYS_IrqEventLinkSet function: Connects a peripheral function interrupt and an IRQn, and registers a callback function
- First argument: IRQ number (IRQn)
- Second argument: Event number of the peripheral function interrupt
- Third argument: Callback function address

R_SYS_IrqStatusClear function: Clears the IRQn status flag
- First argument: IRQ number (IRQn)

R_NVIC_EnableIRQ function: Permits IRQn interrupts
- First argument: IRQ number (IRQn)

R_NVIC_SetPriority function: Sets the IRQn interrupt priority level
- First argument: IRQ number (IRQn)
- Second argument: Priority (0: high to 3: low)

R_NVIC_ClearPendingIRQ function: Clears the IRQn interrupt pending state
- First argument: IRQ number (IRQn)

Functions beginning with "R_NVIC" perform processing similar to NVIC control functions provided by Arm®. These functions are redefined as forced inline functions that enable use of interrupt control functions even after shutting off power to internal flash memory when in a low power consumption mode. Functions beginning with "R_NVIC" can be used even in modes other than a low power consumption mode.

Cases in which a peripheral function driver is used to control interrupts:

In a user program, there is no need to execute the interrupt control functions of the R_SYS driver. The interrupt control functions of the R_SYS driver are executed within peripheral function drivers according to the IRQ numbers set in r_system_cfg.h. The definition values in r_system_cfg.h should be edited according to the operating environment.

For interrupts used by each driver, refer to the driver specifications that are presented in chapter 4. As one example, a list of interrupts used by drivers is shown in Figure 10-1 of chapter 10, Appendix.

Cases in which interrupts are controlled without using peripheral function drivers:

In a user program, interrupt control functions of the R_SYS driver should be executed. By editing the definition values in r_system_cfg.h according to the operating environment and using IRQ number settings when executing interrupt control functions, unified management of IRQ numbers is possible.

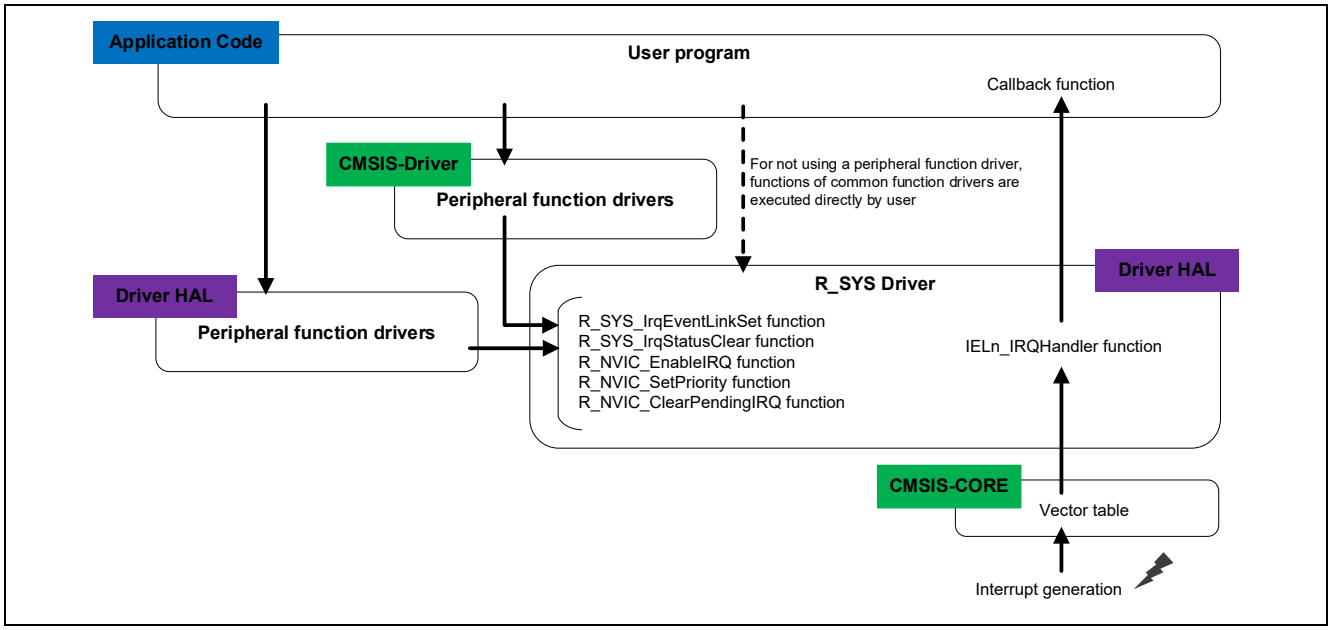


Figure 6-11 Interrupt Control

6.4 Clock Settings

6.4.1 Clock Definitions

In the R_CORE driver, clock definitions are prepared in r_core_cfg.h. The definition values in r_core_cfg.h should be edited according to the operating environment.

Clock-related definitions in r_core_cfg.h are shown in Figure 6-12.

Table 4-1 List of Initial Settings of R_CORE Configurations

Section	Configuration	Description	Initial Value
4.1.1	SYSTEM_CFG_MOSC_ENABLE	Specifies whether to operate or stop the main clock (MOSC).	0
4.1.2	SYSTEM_CFG_MOSC_FREQUENCY_HZ	Specifies the operating frequency of the main clock (MOSC).	32000000
4.1.3	SYSTEM_CFG_MOSC_DRIVE	Specifies the drive capability of the main clock oscillator.	7
4.1.4	SYSTEM_CFG_MOSC_CLOCK_SOURCE	Specifies the oscillation source of the main clock oscillator.	0
4.1.5	SYSTEM_CFG_MOSC_LOW_POWER_ENABLE	Specifies the low consumption oscillation function of the main clock oscillator.	0
4.1.6	SYSTEM_CFG_MOSC_WAIT_TIME	Specifies the stabilization wait time for the main clock oscillator.	5
4.1.7	SYSTEM_CFG_HOCO_ENABLE	Specifies whether to operate or stop the high-speed on-chip oscillator (HOCO).	0
4.1.8	SYSTEM_CFG_HOCO_FREQUENCY	Specifies the oscillation frequency of the high-speed on-chip oscillator (HOCO).	0
4.1.9	SYSTEM_CFG_MOCO_ENABLE	Specifies whether to operate or stop the middle-speed on-chip oscillator (MOCO).	1
4.1.10	SYSTEM_CFG_LOCO_ENABLE	Specifies whether to operate or stop the low-speed on-chip oscillator (LOCO).	1
4.1.11	SYSTEM_CFG_SOSC_ENABLE	Specifies whether to operate or stop the sub-clock oscillator (SOSC).	0
4.1.12	SYSTEM_CFG_SOSC_DRIVE	Specifies the drive capability of the sub-clock oscillator.	0
4.1.13	SYSTEM_CFG_SOSC_NF_STOP	Specifies whether to operate or stop the noise filter for the sub-clock oscillator.	0
4.1.14	SYSTEM_CFG_PLL_ENABLE	Specifies whether to operate or stop the PLL circuit.	0
4.1.15	SYSTEM_CFG_PLL_DIV	Specifies the frequency division ratio of PLL clock source.	1
4.1.16	SYSTEM_CFG_PLL_MUL	Specifies the frequency multiplication factor of the PLL.	1
4.1.18	SYSTEM_CFG_CLCOK_SOURCE	Specifies the source of the system clock.	1
4.1.19	SYSTEM_CFG_ICK_PCKA_DIV	Specifies the frequency division of the system clock (ICLK) and peripheral module clock A (PCLKA).	0
4.1.20	SYSTEM_CFG_PCKB_DIV	Specifies the frequency division of peripheral module clock B (PCLKB).	0
4.1.21	SYSTEM_CFG_POWER_CONTROL_MODE	Specifies the power control mode.	1

Set main clock state on start of operation
Settings referenced on using main clock
Set HOCO state on start of operation
Setting referenced on using HOCO
Set MOCO on start operation
Set LOCO on start of operation
Set subclock state on start of operation
Settings referenced on using subclock
Set PLL circuit state on start of operation
Settings referenced on using PLL circuit
Set system clock on start of operation
Set division ratio for ICLK and PCLKA on start of operation
Set division ratio for PCLKB on start of operation
Set power control mode on start of operation

Excerpted from section of Configuration in R_CORE Detailed Specification

R_CORE driver startup processing: All definitions are referenced.
R_SYS driver clock setting function: Only the black letter definitions are referenced.
Blue-letter definitions are only referenced in startup processing.

Figure 6-12 Clock-Related Definitions in r_core_cfg.h

6.4.2 Driver Functions

The R_SYS driver provides clock control functions. The clock control functions of the R_SYS driver control the clock according to the settings in r_core_cfg.h.

Some of the clock control functions of the R_SYS driver are shown below.

- R_SYS_MainOscSpeedClockStart function: Starts oscillation of the main clock
- R_SYS_MainOscSpeedClockStop function: Stops the main clock
- R_SYS_SystemClockMOSCSet function: Sets the main clock to the system clock

There are many other functions besides these. (For details, refer to the R_SYS driver specifications that are presented in chapter 4.)

6.5 Pin Settings

This device enables selection of pins to be used by peripheral functions, interrupts, and general I/O ports from among multiple pins. The R_PIN driver provides functions in pin.c to set the pins used by peripheral functions. After reset cancellation, these are set for general input ports with the exception of some pins, and so function processing in pin.c should be edited according to the operating environment.

6.5.1 Driver Functions

The R_PIN driver provides pin setting functions that are used by peripheral functions in the pin.c file. In this package, objects in the pin.c file are placed in RAM, and therefore pin setting functions can be used even after power to internal flash memory has been shut off in low power consumption mode.

For the RAM placement of objects, see section 6.6, RAM Placement of Programs.

R_XXX_Pinset_YYY function: Sets pins used by peripheral functions
- XXX: Peripheral function name
- YYY: Channel/function name

R_XXX_Pinclr_YYY function: Sets pins that are no longer used by peripheral functions to general I/O ports
- XXX: Peripheral function name
- YYY: Channel/function name

Pins used by peripheral functions are indicated in Table 6-2. For details on pin assignment, refer to the “Functions assigned to each multiplexed pin” in the Multi-Function Pin Controller (MPC) chapter in the User’s Manual: Hardware.

Table 6-2 Pins Used by Hardware Functions

Hardware Function	Corresponding Driver	Peripheral Function Name	Channel/ Function Name	Pin Function
Pin interrupt	—	ICU	NMI	NMI
			CHn (n=0 to 9)	IRQn (n=0 to 9)
General PWM Timer (GPT)	—	GPT	CHn (n=0 to 5)	GTIOCnA, GTIOCnB (n=0 to 5)
			COM[Note] (Common)	GTETRGA, GTETRGB
			OPS[Note]	GTIU, GTOULO, GTOUUP, GTIV, GTOVLO, GTOVUP, GTIW, GTOWLO, GTOWUP
Asynchronous general-purpose timer (AGT)	—	AGT	CHn (n=0, 1)	AGTEEn, AGTIOn, AGTOn, AGTOAn, AGTOBn（n=0, 1）
Serial communication interface (SCIg, SCIi)	R_USART	SCI	CHn (n=0 to 9)	CTSn, RXDn, SCKn, TXDn (n=0 to 9)）
Serial peripheral interface (SPI)	R_SPI	RSPI	CH0	MISOA, MOSIA, RSPCKA, SSLAn (n=0 to 3)
			CH1	MISOB, MOSIB, RSPCKB, SSLBn (n=0 to 3)
Quad serial peripheral interface (QSPI)	—	QSPI	—	QSPCLK, QSSL, QIOn (n=0 to 3)
I2C bus interface (RIIC)	R_I2C	RIIC	CHn (n=0, 1)	SCLn, SDAn (n=0, 1)
Clock frequency accuracy measurement circuit (CAC)	—	CAC	—	CACREF
14-bit A/D converter (S14AD)	R_ADC	S14AD	—	ADTRG0, AN0nn (nn=00 to 06, 16, 17, 20 to 28)
12-bit D/A converter (R12DA)	—	R12DA	—	DA0
Analog converter (ACMP)	—	ACMP	—	CMPIN, CMPREF, VCOUT
MIP LCD controller (MLCD)	R_PMIP	MLCD	—	MLCD_DEN, MLCD_ENBG, MLCD_ENBS, MLCD_SCLK, MLCD_SIn（n=0 to 7）, MLCD_VCOM, MLCD_XRST
Key interrupt function	—	KINT	—	KRMnn (nn=00 to 07)
USB2.0FS host/function module	—	USB	—	USB_EXICEN, USB_ID, USB_VBUS, USB_VBUSEN USB_OVRCURA, USB_OVRCURB
Real-time clock	—	RTC	—	RTCICn (n=0 to 2), RTCOUT
Clock correction circuit	—	CCC	—	CCCOUT
Low-speed pulse generator	—	LPG	—	LPGOUT
8-bit timer	—	TMR	—	TMCIn, TMOn, TMRIn (n=0, 1)
Clock generation circuit	—	CLKOUT	—	CLKOUT, CLKOUT32
LED driver	—	LED	—	LEDn (n=1 to 3)

Note: The "peripheral function name" and "channel/function name" of function names are opposites.
R_COM_Pinset_GPT, R_COM_Pinclr_GPT
R_OPS_Pinset_GPT, R_OPS_Pinclr_GPT

When a peripheral function driver is used to make pin settings:

Because R_PIN pin control functions are being executed within peripheral function drivers, in a user program there is no need to execute R_PIN driver pin setting functions. The function processing in pin.c should be edited according to the operating environment.

When pin settings are made without using a peripheral function driver:

Pin control should be performed by a user program. By editing the function processing in pin.c according to the operating environment and using the functions for pin control, unified management of the information of used pins can be performed.

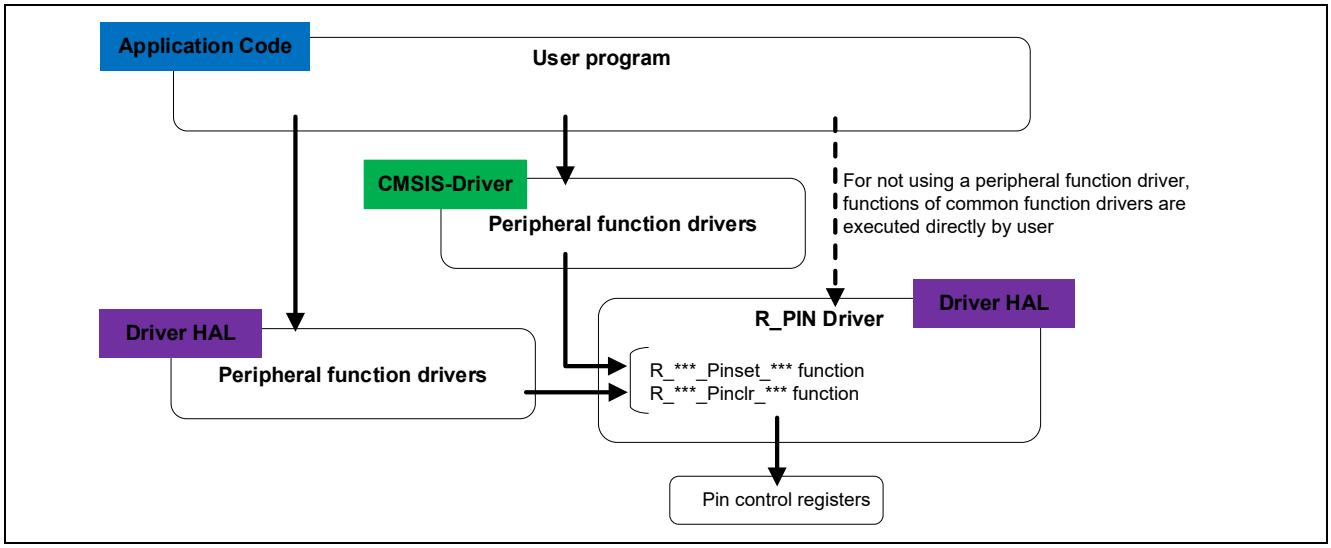


Figure 6-13 Calling Pin Control Functions

6.5.2 Editing Driver Functions

R_PIN driver functions are functions that the user edits and uses according to the operating environment. Each function describes the settings of pins that can be used by peripheral functions.

An example of editing of R_PIN driver functions when transmit/receive data pins of a serial communication interface SCI4 are allocated to ports as indicated below and used is shown in Figure 6-14.

P812: TXD4
P813: RXD4

pin.c Driver HAL

```
/***********************************************************************************************//**
 * @brief This function sets Pin of SCI4.
 **********************************************************************************************/
/* Function Name : R_SCI_Pinset_CH4 */
void R_SCI_Pinset_CH4(void)  // @suppress("API function naming") @suppress("Function length")
{
    /* Disable protection for PFS function (Set to PWPR register) */
    R_SYS_RegisterProtectDisable(SYSTEM_REG_PROTECT_MPC);

    /* CTS4 : P111 */
//    PFS->P111PFS_b.ASEL = 0U;  /* 0: Do not use as an analog pin, 1: Use as an analog pin. */
//    PFS->P111PFS_b.ISEL = 0U;  /* 0: Do not use as an IRQn input pin,  1: Use as an IRQn input pin. */
//    PFS->P111PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P111PFS_b.PMR  = 1U;  /* 0: Use the pin as a general I/O port, 1: Use the pin as a peripheral module. */
            :

    /* When using SCI in I2C mode, set the pin to NMOS Open drain. */
////    PFS->P203PFS_b.NCODR = 1U;  /* 0: CMOS output, 1: NMOS open-drain output. */
////    PFS->P203PFS_b.PCODR = 0U;  /* 0: CMOS output, 1: PMOS open-drain output. */
//    PFS->P203PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P203PFS_b.PMR  = 1U;  /* 0: Use the pin as a general I/O port, 1: Use the pin as a peripheral module. */

    /* TXD4 : P812 */
    PFS->P812PFS_b.ASEL = 0U;  /* 0: Do not use as an analog pin, 1: Use as an analog pin. */
    PFS->P812PFS_b.ISEL = 0U;  /* 0: Do not use as an IRQn input pin,  1: Use as an IRQn input pin. */

    /* When using SCI in I2C mode, set the pin to NMOS Open drain. */
////    PFS->P812PFS_b.NCODR = 1U;  /* 0: CMOS output, 1: NMOS open-drain output. */
////    PFS->P812PFS_b.PCODR = 0U;  /* 0: CMOS output, 1: PMOS open-drain output. */
    PFS->P812PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
    PFS->P812PFS_b.PMR  = 1U;  /* 0: Use the pin as a general I/O port, 1: Use the pin as a peripheral module. */

    /* RXD4 : P112 */
//    PFS->P112PFS_b.ASEL = 0U;  /* 0: Do not use as an analog pin, 1: Use as an analog pin. */
//    PFS->P112PFS_b.ISEL = 0U;  /* 0: Do not use as an IRQn input pin,  1: Use as an IRQn input pin. */
            :

    /* When using SCI in I2C mode, set the pin to NMOS Open drain. */
////    PFS->P202PFS_b.NCODR = 1U;  /* 0: CMOS output, 1: NMOS open-drain output. */
////    PFS->P202PFS_b.PCODR = 0U;  /* 0: CMOS output, 1: PMOS open-drain output. */
//    PFS->P202PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P202PFS_b.PMR  = 1U;  /* 0: Use the pin as a general I/O port, 1: Use the pin as a peripheral module. */

    /* RXD4 : P813 */
    PFS->P813PFS_b.ASEL = 0U;  /* 0: Do not use as an analog pin, 1: Use as an analog pin. */
    PFS->P813PFS_b.ISEL = 0U;  /* 0: Do not use as an IRQn input pin,  1: Use as an IRQn input pin. */

    /* When using SCI in I2C mode, set the pin to NMOS Open drain. */
////    PFS->P813PFS_b.NCODR = 1U;  /* 0: CMOS output, 1: NMOS open-drain output. */
////    PFS->P813PFS_b.PCODR = 0U;  /* 0: CMOS output, 1: PMOS open-drain output. */
    PFS->P813PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
    PFS->P813PFS_b.PMR  = 1U;  /* 0: Use the pin as a general I/O port, 1: Use the pin as a peripheral module. */

    /* SCK4 : P108 */
//    PFS->P108PFS_b.ASEL = 0U;  /* 0: Do not use as an analog pin, 1: Use as an analog pin. */
//    PFS->P108PFS_b.ISEL = 0U;  /* 0: Do not use as an IRQn input pin,  1: Use as an IRQn input pin. */
//    PFS->P108PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P108PFS_b.PMR  = 1U;  /* 0: Use the pin as a general I/O port, 1: Use the pin as a peripheral module. */

    /* SCK4 : P204 */
//    PFS->P204PFS_b.ASEL = 0U;  /* 0: Do not use as an analog pin, 1: Use as an analog pin. */
//    PFS->P204PFS_b.ISEL = 0U;  /* 0: Do not use as an IRQn input pin,  1: Use as an IRQn input pin. */
//    PFS->P204PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P204PFS_b.PMR  = 1U;  /* 0: Use the pin as a general I/O port, 1: Use the pin as a peripheral module. */

    /* SCK4 : P814 */
//    PFS->P814PFS_b.ASEL = 0U;  /* 0: Do not use as an analog pin, 1: Use as an analog pin. */
//    PFS->P814PFS_b.ISEL = 0U;  /* 0: Do not use as an IRQn input pin,  1: Use as an IRQn input pin. */
//    PFS->P814PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
//    PFS->P814PFS_b.PMR  = 1U;  /* 0: Use the pin as a general I/O port, 1: Use the pin as a peripheral module. */


    /* Enable protection for PFS function (Set to PWPR register) */
    R_SYS_RegisterProtectEnable(SYSTEM_REG_PROTECT_MPC);

}/* End of function R_SCI_Pinset_CH4() */
```

Set P812 as TXD4 pin.

Set P813 as RXD4 pin.

Figure 6-14 Example of Editing Pin Setting Function

Table 24.1 Functions Assigned to Each Multiplexed Pin (4/9)

Module/Function	Channel	Pin Function	Allocation Port	Package		
				156-pin WLBGA	144 pins	100 pins
Serial communications interface (SCIg, SCIi)	SCI0	CTS0_A	P107	✓	✓	✓
		CTS0_B	P500	✓	✓	✓
		CTS0_C	P704	✓	✓	✓
⋮						
	SCI4	CTS4_A	P111	✓	✓	✓
		CTS4_B	P205	✓	✓	✓
		CTS4_C	P815	✓	✓	×
⋮						
Serial communications interface (SCIg, SCIi)	SCI4	RXD4_A	P112	✓	✓	✓
		RXD4_B	P202	✓	✓	✓
		RXD4_C	P813	✓	✓	×
		SCK4_A	P108	✓	✓	✓
		SCK4_B	P204	✓	✓	✓
		SCK4_C	P814	✓	✓	×
		TXD4_A	P113	✓	✓	✓
		TXD4_B	P203	✓	✓	✓
		TXD4_C	P812	✓	✓	×
	SCI5	CTS5_A	P109	✓	✓	✓

Excerpted from section of Multi-Function Pin Controller (MPC) in User's Manual: Hardware

The above table indicates that:

Pins used by SCI4 can be selected from:

- CTS pins: P111, P205, P815
- RXD pins: P112, P202, P813
- SCK pins: P108, P204, P814
- TXD pins: P113, P203, P812

Figure 6-15 Method of Confirming Pins that Can be Used by Peripheral Functions

6.6 RAM Placement of Programs

In this device, power consumption can be reduced by shutting off power to internal flash memory.

When a program expanded in RAM is run after having shut off power to internal flash memory, it is necessary to place the desired program in RAM.

Below, methods for placing a desired program in RAM that are used in this package are introduced.

1. RAM placement method using a RAM placement section
   This is the method performed by each driver; placement locations are set in function units according to RAM placement definitions in the configuration definition header.
   Only the R_PIN driver sets placement locations in object units.
   This method can be used even for standard functions.

2. Method of RAM placement by forced inline expansion
   This method is performed by interrupt control functions in the R_SYS driver beginning with "R_NVIC".
   When execution is from programs placed in RAM, inline expansion to RAM is performed, but when execution is from programs placed in internal flash memory, inline expansion to internal flash memory is performed.

6.6.1 RAM Placement Method Using RAM Placement Section

The procedures for setting RAM placement using a RAM placement section are explained.

Step 1. Definition of RAM placement sections in a linker file
(Sections defined in this package are shown in Figure 6-16.)
Step 2. A desired program is allocated to the RAM placement section
Step 3. After reset cancellation, the program allocated to the RAM placement section is expanded in RAM

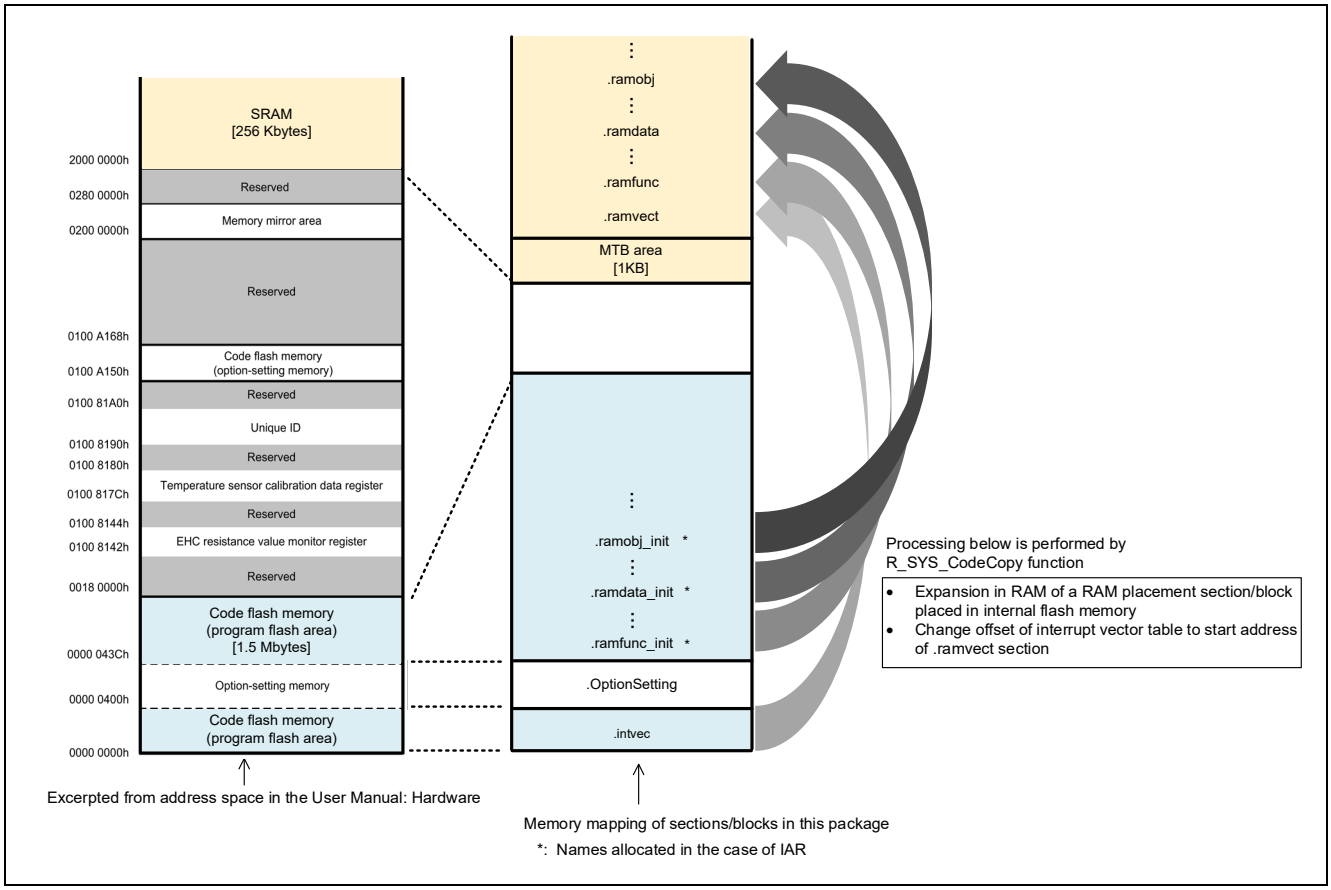


Figure 6-16 Memory Mapping When Using RAM Placement Sections

(1) Definition of RAM Placement sections in a linker file

Table 6-3 indicates the RAM placement sections defined in a linker file that is included with this package.

Table 6-3 RAM Placement Sections

Section Name	Description	Allocation Circumstances
.ramobj	Section for RAM placement of objects	R_PIN driver
.ramdata	Section for RAM placement of variables	Variables used by driver functions
.ramfunc	Section for RAM placement of functions	Driver functions
.ramvect	Section for RAM placement of vector table	Vector table

The R_PIN driver allocates the object file pin.o to .ramobj in the linker file. Because pin.o is specified in the linker file, the filename of pin.c should not be changed.

The settings in the linker file included with this package are shown in Figure 6-17 and Figure 6-18.

RE01_1500KB.icf

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$¥config¥ide¥IcfEditor¥cortex_v1_1.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x00000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_IROM1_start__ = 0x00000000;
define symbol __ICFEDIT_region_IROM1_end__   = 0x000003ff;
define symbol __ICFEDIT_region_IROM2_start__ = 0x0000043C;
define symbol __ICFEDIT_region_IROM2_end__   = 0x0017FFFF;
define symbol __ICFEDIT_region_EROM1_start__ = 0x0;
define symbol __ICFEDIT_region_EROM1_end__   = 0x0;
define symbol __ICFEDIT_region_EROM2_start__ = 0x0;
define symbol __ICFEDIT_region_EROM2_end__   = 0x0;
define symbol __ICFEDIT_region_EROM3_start__ = 0x0;
define symbol __ICFEDIT_region_EROM3_end__   = 0x0;
define symbol __ICFEDIT_region_IRAM1_start__ = 0x20000400;  // Change the starting address of RAM for the MTB.
define symbol __ICFEDIT_region_IRAM1_end__   = 0x2003FFFF;
define symbol __ICFEDIT_region_IRAM2_start__ = 0x00000000;
define symbol __ICFEDIT_region_IRAM2_end__   = 0x00000000;
define symbol __ICFEDIT_region_ERAM1_start__ = 0x0;
    :
define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size =   ICFEDIT_size_heap     { };
define block .ramvect  with alignment = 256, size = 0x400                 { };
define block .ramobj_init
{
    /* If you need RAM allocation per file, add "XXXX.o" to this definition. */
    ro code section *_init object pin.o,
    ro data section *_init object pin.o,
    ro code section *_init object I32DivModFast.o,
    ro data section *_init object I32DivModFast.o,
}
    :
define block .ramobj
{
    /* If you need RAM allocation per file, add "XXXX.o" to this definition. */
    rw code object pin.o,
    rw data object pin.o,
    rw code object I32DivModFast.o,
    rw data object I32DivModFast.o,
}
    :
// for sram copy
initialize manually with packing = none { section .ramfunc };
initialize manually with packing = none { section .ramdata };
initialize manually with packing = none
{
    /* If you need RAM allocation per file, add "XXXX.o" to this definition. */
    ro object pin.o,
    rw object pin.o,
    ro object I32DivModFast.o,
    rw object I32DivModFast.o,
}
    :
initialize manually with packing = none { section .ehc_ramfunc };
/* Place ".intvec" section at ROM address __ICFEDIT_intvec_start__(=0x00000000). */
place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

/* Place ".ramfunc_init" section and ".ramobj_init" section in ROM */
place in IROM_region  { readonly, section .ramfunc_init, block .ramobj_init, section .ramdata_init, section .ehc_ramfunc_init };
place in EROM_region  { readonly section application_specific_ro };
/* Place ".ramfunc" section and ".ramobj" section in RAM */
place in IRAM_region  { first block .ramvect, readwrite, block CSTACK, block HEAP, section .ramfunc, block .ramobj, section .ehc_ramfunc };
/* Place ".ramdata" section in RAM */
place in IRAM_region  { section .ramdata };
place in ERAM_region  { readwrite section application_specific_rw };
place in OptionSetting_region{section .OptionSetting};
place in OptionSettingID_region{section .OptionSettingID};
place in OptionSettingAWS_region{section .OptionSettingAWS};
```

Move beginning of RAM section back 1 KB to secure 1 KB for MTB at beginning of RAM

Place .ramvect in block

Place .ramobj_init, including initial value data of objects, in block

Place .ramobj, including objects, in block

Place .ramfunc, .ramdata in section with initial value data uncompressed

Initial value data of objects uncompressed

Specify .ramfunc_init, .ramobj_init, .ramdata_init in internal flash memory region

Specify .ramvect block at beginning (+1 KB) of RAM region
Specify .ramfunc section in RAM region
Specify .ramobj block in RAM region

Specify .ramdata section in RAM region

Figure 6-17 Example of Linker File Settings (IAR Compiler)

RE01_1500KB.ld

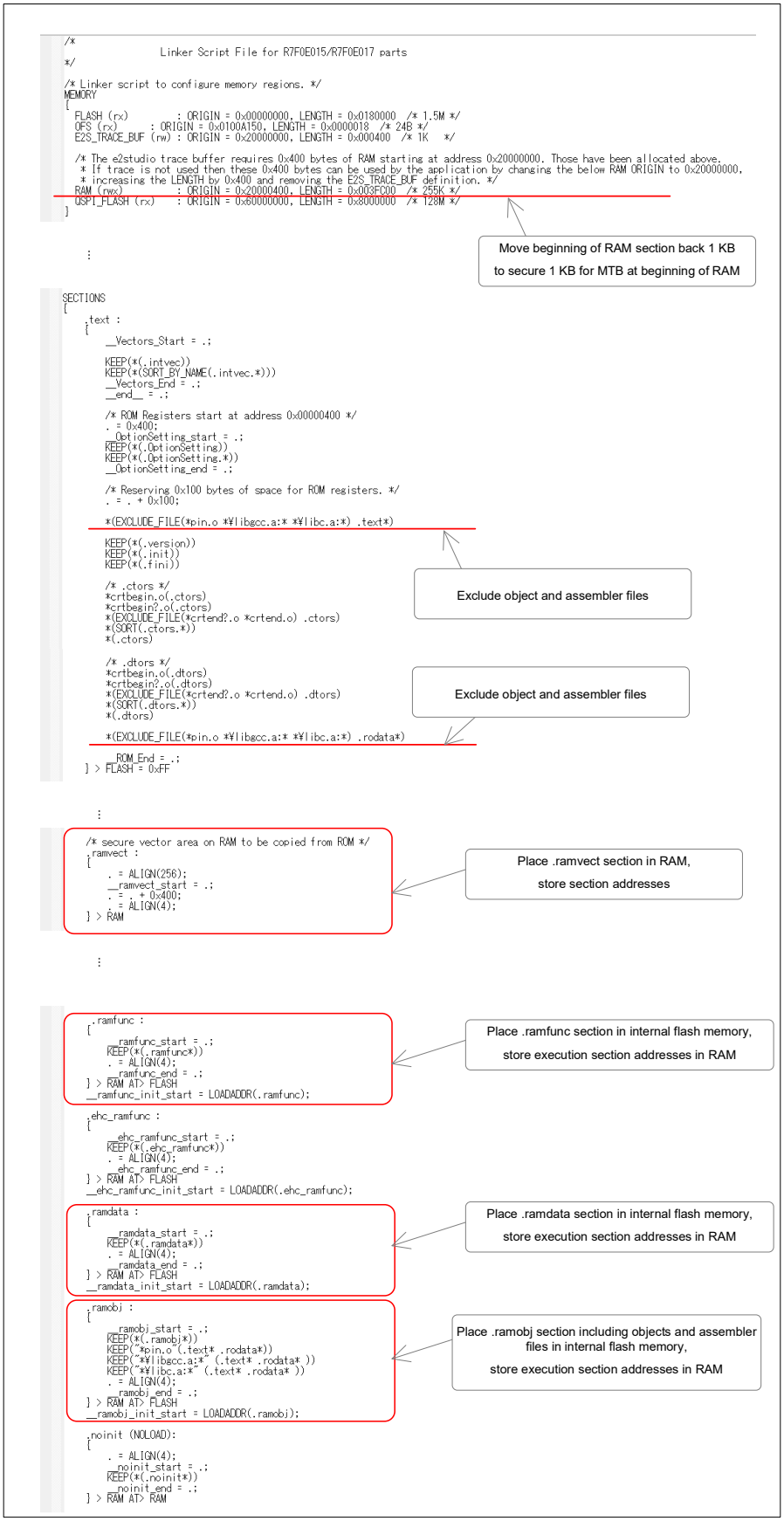


Figure 6-18 Example of Linker File Settings (GCC Compiler)

(2) A desired program is allocated to a RAM placement section

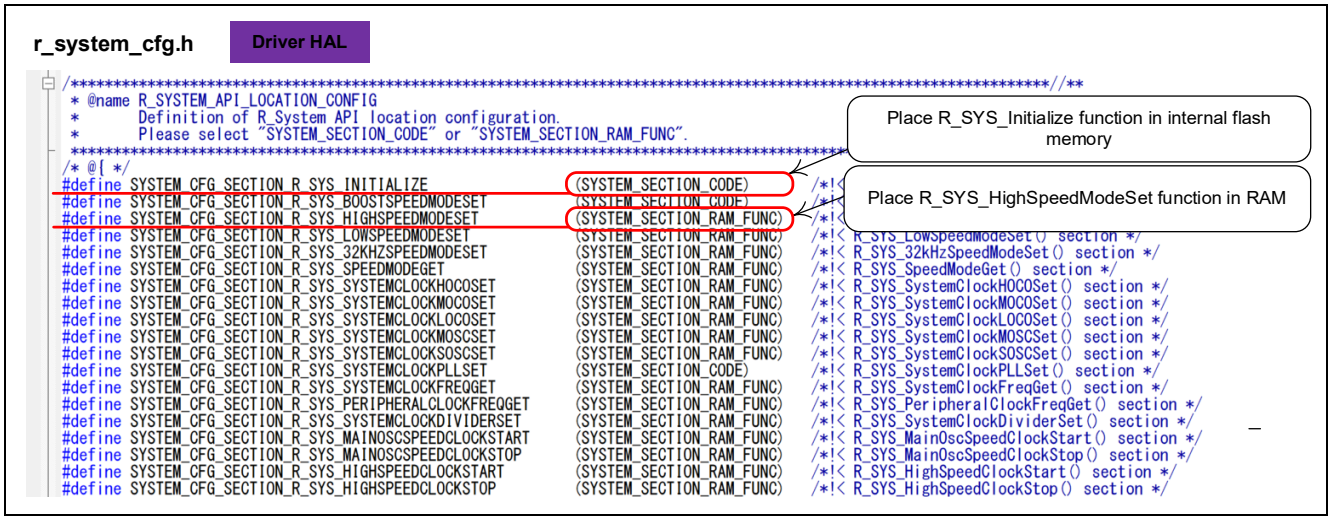
In order to place a program to RAM, the desired program is specified in a section for RAM placement. Drivers included with this package are devised to facilitate specification of driver functions in sections for RAM placement.

➢ Method of setting a driver in a section for RAM placement:

For each driver, the desired program is specified in the section for RAM placement according to definition values in the configuration definition header.

Definition for program RAM placement:

- Description: Sets whether a driver function is placed to RAM
- Definition name: XXX__CFG_SECTION_YYY
  XXX: Driver name
  YYY: Function name (all uppercase)
- Definition values:
  SYSTEM_SECTION_CODE: Function allocation to a section for internal flash memory code
  SYSTEM_SECTION_RAM_FUNC: Function allocation to a section for RAM placement
- Setting example: A setting example for an R_SYS driver function appears in Figure 6-19.




```
/***********************************************************************************************************************//**
 * @name R_SYSTEM_API_LOCATION_CONFIG
 *       Definition of R_System API location configuration.
 *       Please select "SYSTEM_SECTION_CODE" or "SYSTEM_SECTION_RAM_FUNC".
 **********************************************************************************************************************/
/* @{ */
#define SYSTEM_CFG_SECTION_R_SYS_INITIALIZE              (SYSTEM_SECTION_CODE)       /*!< R_SYS_Initialize() section */
#define SYSTEM_CFG_SECTION_R_SYS_BOOSTSPEEDMODESET       (SYSTEM_SECTION_CODE)       /*!< R_SYS_BoostSpeedModeSet() section */
#define SYSTEM_CFG_SECTION_R_SYS_HIGHSPEEDMODESET        (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_HighSpeedModeSet() section */
#define SYSTEM_CFG_SECTION_R_SYS_LOWSPEEDMODESET         (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_LowSpeedModeSet() section */
#define SYSTEM_CFG_SECTION_R_SYS_32KHZSPEEDMODESET       (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_32kHzSpeedModeSet() section */
#define SYSTEM_CFG_SECTION_R_SYS_SPEEDMODEGET            (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_SpeedModeGet() section */
#define SYSTEM_CFG_SECTION_R_SYS_SYSTEMCLOCKHOCOSET      (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_SystemClockHOCOSet() section */
#define SYSTEM_CFG_SECTION_R_SYS_SYSTEMCLOCKMOCOSET      (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_SystemClockMOCOSet() section */
#define SYSTEM_CFG_SECTION_R_SYS_SYSTEMCLOCKLOCOSET      (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_SystemClockLOCOSet() section */
#define SYSTEM_CFG_SECTION_R_SYS_SYSTEMCLOCKMOSCSET      (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_SystemClockMOSCSet() section */
#define SYSTEM_CFG_SECTION_R_SYS_SYSTEMCLOCKSOSCSET      (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_SystemClockSOSCSet() section */
#define SYSTEM_CFG_SECTION_R_SYS_SYSTEMCLOCKPLLSET       (SYSTEM_SECTION_CODE)       /*!< R_SYS_SystemClockPLLSet() section */
#define SYSTEM_CFG_SECTION_R_SYS_SYSTEMCLOCKFREQGET      (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_SystemClockFreqGet() section */
#define SYSTEM_CFG_SECTION_R_SYS_PERIPHERALCLOCKFREQGET  (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_PeripheralClockFreqGet() section */
#define SYSTEM_CFG_SECTION_R_SYS_SYSTEMCLOCKDIVIDERSET   (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_SystemClockDividerSet() section */
#define SYSTEM_CFG_SECTION_R_SYS_MAINOSCSPEEDCLOCKSTART  (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_MainOscSpeedClockStart() section */
#define SYSTEM_CFG_SECTION_R_SYS_MAINOSCSPEEDCLOCKSTOP   (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_MainOscSpeedClockStop() section */
#define SYSTEM_CFG_SECTION_R_SYS_HIGHSPEEDCLOCKSTART     (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_HighSpeedClockStart() section */
#define SYSTEM_CFG_SECTION_R_SYS_HIGHSPEEDCLOCKSTOP      (SYSTEM_SECTION_RAM_FUNC)   /*!< R_SYS_HighSpeedClockStop() section */
```

Figure 6-19 Setting Example for Section to Place Driver Function

➢ Method of setting a user program in a section for RAM placement:

The prototype declaration "__attribute__ ((section(“xxxx”)))" for functions and variables can be used to specify a desired program in a section for RAM placement.

Below are setting examples for placing variables and functions in a desired section.
(These can be used with both the IAR compiler and the GCC compiler.)

- Variable sample_data: Variable placement in section ".ramdata" for RAM placement
- Function sample_func: Function placement in section ".ramfunc" for RAM placement

✓ Setting example in which __attribute__((section(“xxx”))) is used to specify a section

```
static const int32_t sample_data  __attribute__ ((section(".ramdata")));
static void sample_function(void) __attribute__ ((section(".ramfunc")));
```

✓ Setting example in which __attribute__ ((noinline)) is used to suppress function inline specification

Depending on the compiler optimization, there are cases in which a function is inline-expanded and is not placed in RAM as expected. For functions inline expansion of which must be prohibited, "__attribute__ ((noinline))" can be used to suppress inline expansion.

```
static void sample_function(void) __attribute__ ((section(".ramfunc"")))
__attribute__ ((noinline));
```

➢ Method of specifying a standard function in a section for RAM placement:

Commands that the CPU cannot support as standard, such as the standard functions of stdio.h, division, floating-point operations and the like, are executed using standard functions included with the compiler. The method for specifying these standard functions in a section for RAM placement is explained.

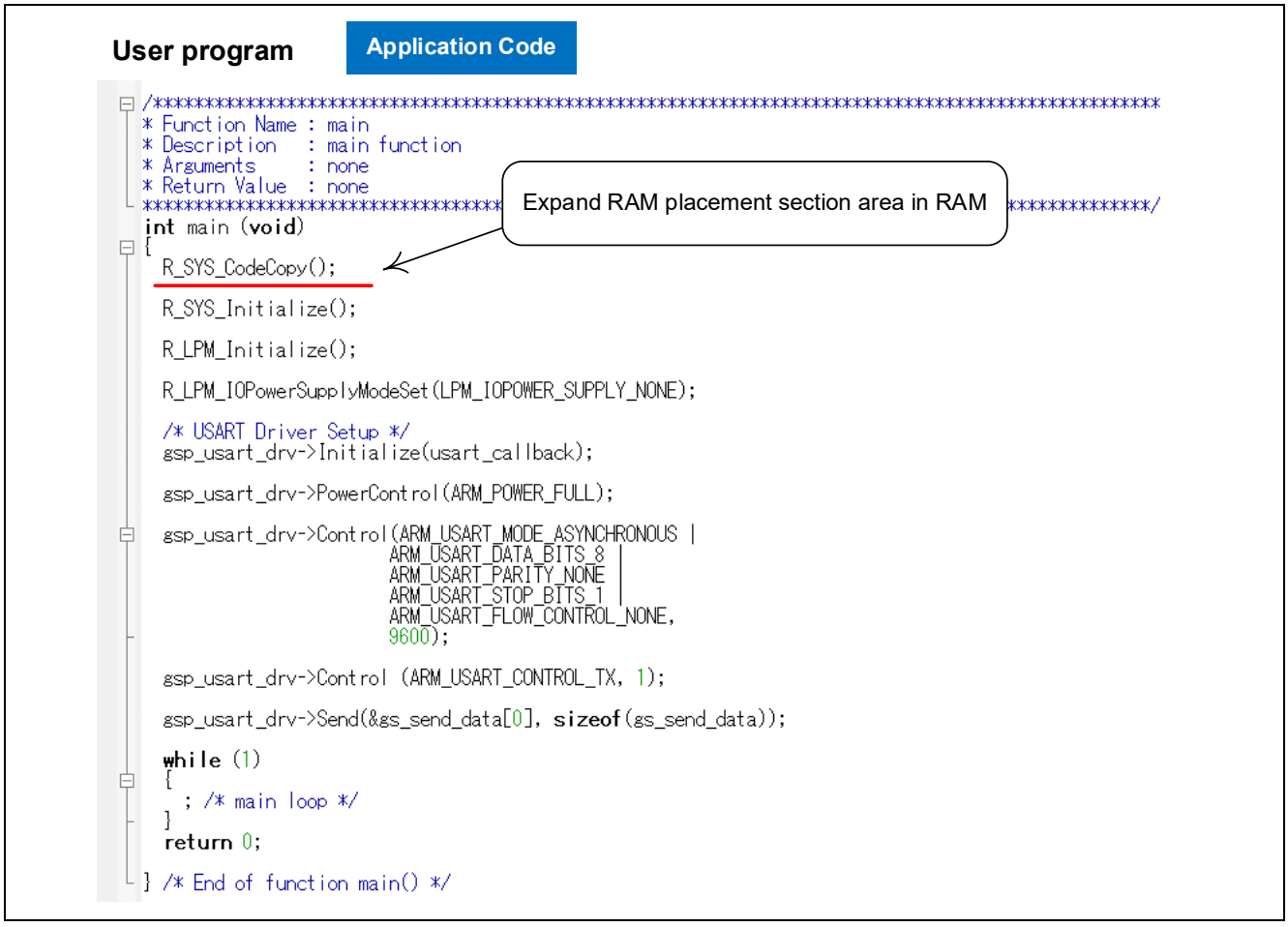
Standard functions are generated as object files (*.o). These object files are allocated to the section for RAM placement of objects (.ramobj) in a linker file. A section for adding objects is provided in advance in the linker script of this package. Adding should be performed referring to Figure 6-17 and Figure 6-18. A method similar to that for the pin.o file is used for allocation to the section for RAM placement, and therefore an object file to be allocated to a section for RAM placement should be added, referring to pin.o.

(3) After reset cancellation, the program allocated to the section for RAM placement is expanded in RAM

After reset cancellation, the R_SYS_CodeCopy function of the R_SYS driver is executed to expand the desired program in RAM.

R_SYS_CodeCopy function

- Description: Function to expand a function allocated to a section for RAM placement from internal flash memory to RAM
- Execution timing:
  The R_SYS_CodeCopy function should be executed only once after reset cancellation and before executing the function allocated to the section for RAM placement. (Calling the function multiple times is unnecessary.)
  This function should be executed before executing the initialization function of the R_SYS driver R_SYS_Initialize.
- Setting example: An example of use of this function is shown in Figure 6-20.




```
/*****************************************************************************************************************
* Function Name : main
* Description   : main function
* Arguments     : none
* Return Value  : none
*****************************************************************************************************************/
int main (void)
{
  R_SYS_CodeCopy();

  R_SYS_Initialize();

  R_LPM_Initialize();

  R_LPM_IOPowerSupplyModeSet(LPM_IOPOWER_SUPPLY_NONE);

  /* USART Driver Setup */
  gsp_usart_drv->Initialize(usart_callback);

  gsp_usart_drv->PowerControl(ARM_POWER_FULL);

  gsp_usart_drv->Control(ARM_USART_MODE_ASYNCHRONOUS |
                         ARM_USART_DATA_BITS_8 |
                         ARM_USART_PARITY_NONE |
                         ARM_USART_STOP_BITS_1 |
                         ARM_USART_FLOW_CONTROL_NONE,
                         9600);

  gsp_usart_drv->Control (ARM_USART_CONTROL_TX, 1);

  gsp_usart_drv->Send(&gs_send_data[0], sizeof(gs_send_data));

  while (1)
  {
    ; /* main loop */
  }
  return 0;
} /* End of function main() */
```

Figure 6-20 Example of Use of Program RAM Expansion Function

6.6.2 Method of RAM Placement by Forced Inline Expansion

Because inline functions are sometimes not inline-expanded due to compiler optimizations and the like, a forced inline expansion method is used.

When executed from a program placed in RAM, by using forced inline expansion, a program can be reliably expanded in RAM.

This method is used for functions of the R_SYS driver beginning with "R_NVIC".

There are similar NVIC functions among the CMSIS standard functions as well, but when a program must be placed in RAM, a Renesas-specific function beginning with "R_NVIC" should be used.

When using forced inline expansion, please refer to the R_SYS driver settings.

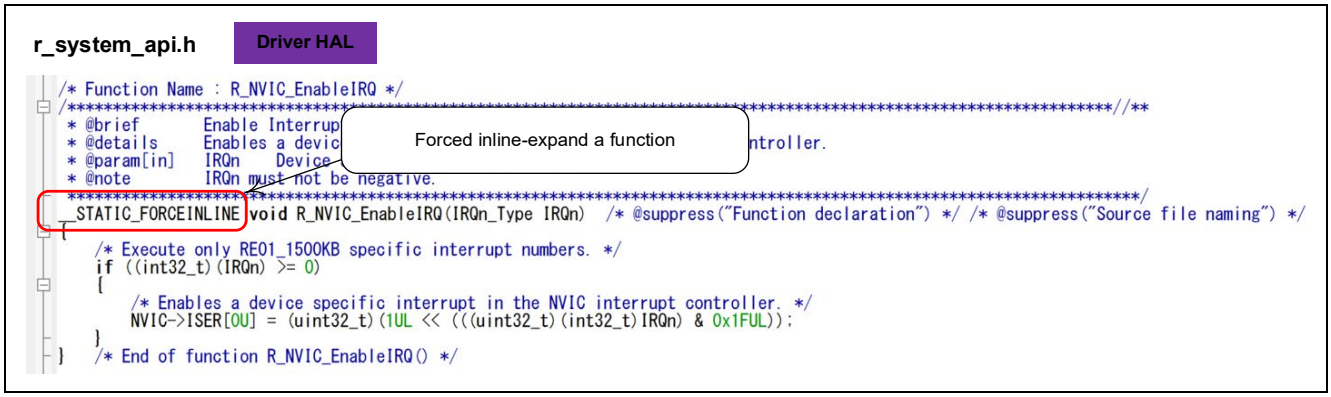


Figure 6-21 Setting Example for Forced Inline Expansion of Function

7. Creating a User Program

7.1 Preparation for User Program Creation

When creating a user program, preparation such as creating specific functions and editing the configuration definition headers of drivers is necessary. Here the preparation required prior to program execution is explained.

The processing may be performed in any order. Prior to program execution, preparation should be performed according to the operating environment.

Table 7-1 Preparation for User Program Creation

When using peripheral function drivers	When not using peripheral function drivers
Preparation for startup processing ① Pin settings upon operation start (User-created function: BoardInit function) ② Definition of clock/power control mode upon operation start	
Preparation of common function drivers ③ IRQ number definitions ④ Setting of pins used by peripheral functions ⑤ Definitions of operating conditions of common function drivers ⑥ Definition of clock used by R_SYS driver	
Preparation of peripheral function drivers	
⑦ Definitions of operating conditions of peripheral function drivers	—

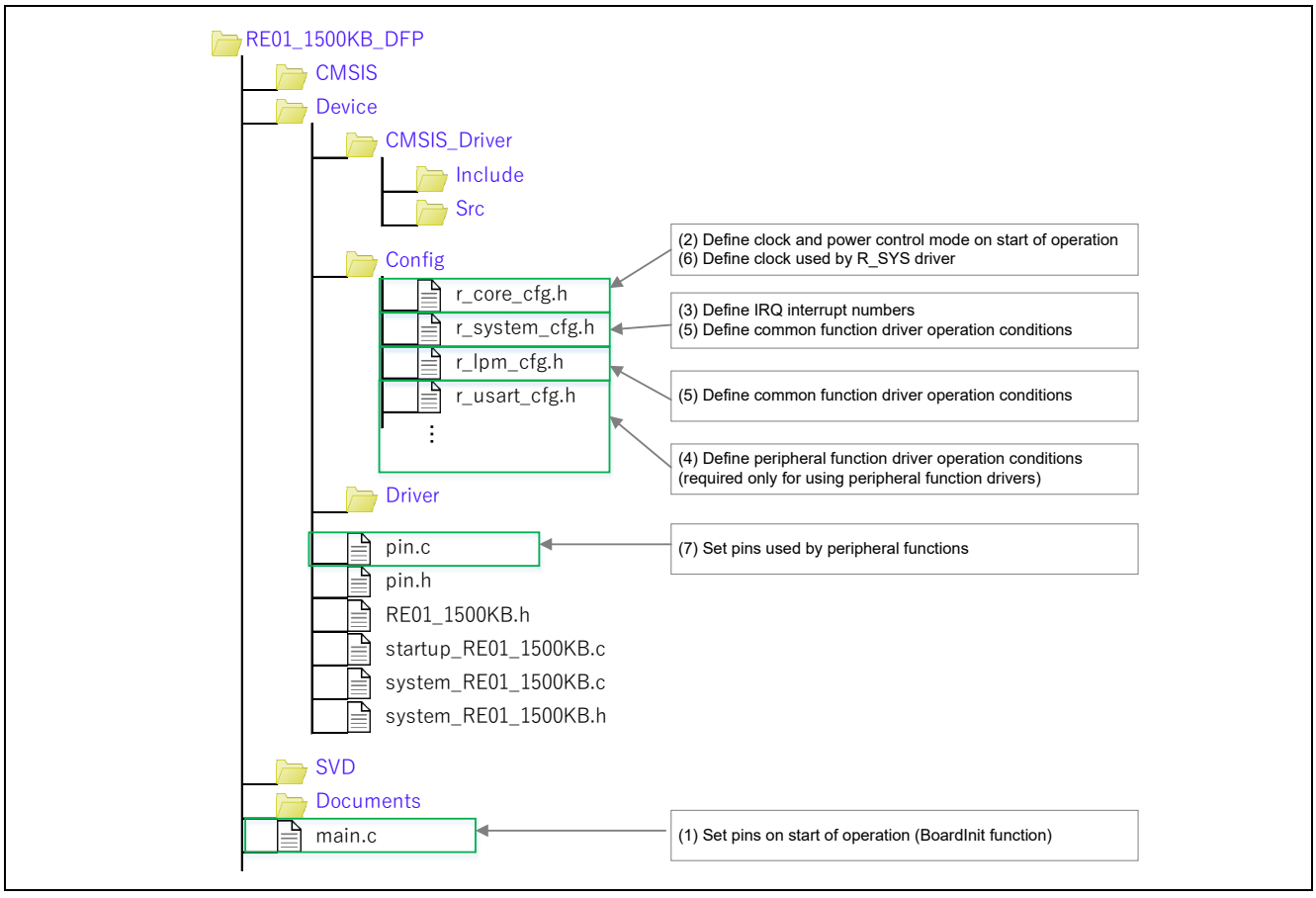


Figure 7-1 Files for Editing in Preparation for User Program Creation

7.1.1 Preparation for Startup Processing

After reset cancellation, in startup processing performed before execution of the main function, any desired definitions and functions are used. These should be edited or created according to the operating environment.

7.1.1.1 Pin settings upon operation start

In startup processing, the user-created function BoardInit is executed. After reset cancellation, when pin settings must be made at an early stage, the BoardInit function should be created and pin processing performed. For initial pin settings, see section 6.1.1, Pin Settings Upon Start of Operation.

A setting example with the BoardInit function is shown in Figure 7-2.

User program **Application Code**

```
/*****************************************************************************************************************
* Function Name: BoardInit
* Description  : Configure board initial setting.
*                This function is called by SystemInit() function in system_RE01_1500KB.c file.
*                This is reference to perform BoardInit process. Sample code of target is E015 SDK board(RTK70E015D)
*                 on Renesas. Please modify this function to suit your board.
* Arguments    : none
* Return Value : none
*****************************************************************************************************************/
void BoardInit(void)
{
    /*****   This function performs at beginning of start-up after released reset pin *****/
    /***** Please set pins here if your board is needed pins setting at the device start-up. ****/
    /*****   This function is suiting RE01_1500KB SDK board. Please change to your board pin setting *****/

    /* Handling of Unused ports (IOVCC domain) */
    /* PORT4 Setting: RE01_1500KB SDK has DCDCs. Those are connect P404 and P405. */
    /* This perform to disable those output.                                      */
    /* Those are needed to enable when using EHC start up of this
    /* Set P404 and P405 not to be used as DCDC_EN (output low: D

    /* PODR - Port Output Data
       b15-b0   PODR15 - PORD00     - Output Low Level */
    PORT4->PODR = 0x0000;

    /* PDR - Port Direction
       b15-b6   PDR15 - PRD06       - Input
       b5 -b4   PDR05 - PRD04       - Output
       b3 -b0   PDR03 - PRD00       - Input */
    PORT4->PDR  = 0x0030;

    /* Handling of Unused pors (AVCC1 domain) */

    /* PORT0 Setting */
    /* Set P009, P008 and P007 as LEDs (output high) */

    /* PODR - Port Output Data
       b15-b10  PODR15 - PORD10     - Output Low Level
       b9 -b7   PODR09 - PORD07     - Output High Level
       b6 -b0   PODR06 - PORD00     - Output Low Level */
    PORT0->PODR = 0x0380;

    /* PDR - Port Direction
       b15-b10  PDR15 - PRD10       - Input PORT
       b9 -b7   PDR09 - PRD07       - Output PORT
       b6 -b0   PDR06 - PRD00       - Input PORT */
    PORT0->PDR = 0x0380;
} /* End of function BoardInit */
```

Set P404 and P405 connected to DCDC on board to LOW output of general port

Set P007, P008, and P009 connected to LED on board to HIGH output of general port

Figure 7-2 Pin Setting Example Upon Operation Start

7.1.1.2 Definition of clock and power control mode upon operation start

In startup processing, initial settings for the clock and power control mode are made according to the settings in r_core_cfg.h. The definition values in r_core_cfg.h should be edited according to the operating environment. For settings in r_core_cfg.h, see section 6.4.1, Clock Definitions.

An example of settings of the clock and power control mode upon operation start is shown in Figure 7-3.

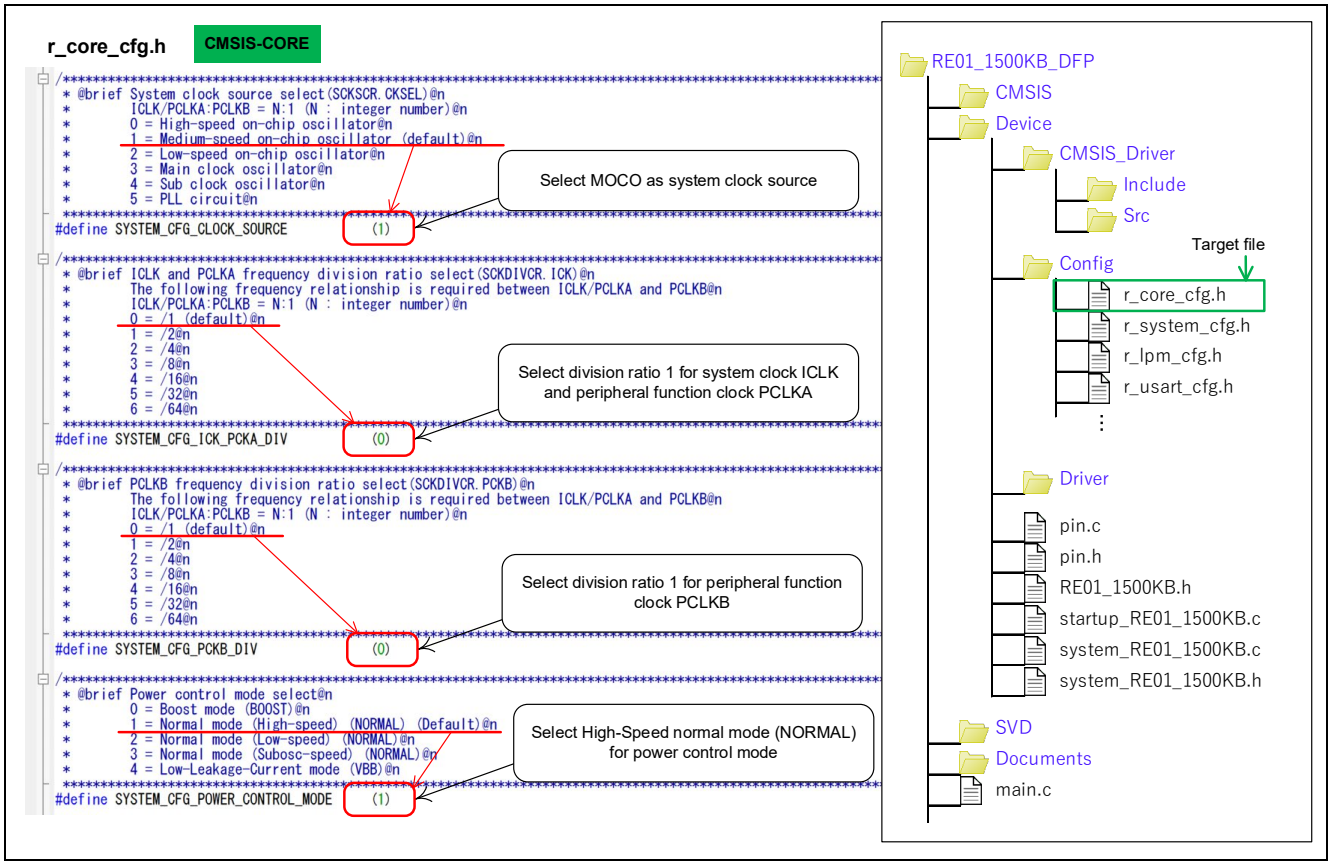


Figure 7-3 Example of Settings of Clock and Power Control Mode Upon Operation Start

7.1.2 Preparation of Common Function Drivers

Regarding interrupts, pin settings and other common functions, driver functions and configuration definition headers should be edited according to the operating environment.

7.1.2.1 IRQ number definitions

When using interrupts, interrupts from peripheral functions must be allocated to IRQ numbers.

The definition values of the IRQ numbers for each interrupt in r_system_cfg.h should be edited according to the operation environment. For settings of IRQ numbers, see section 6.3.2, IRQ Number Allocation.

An example of IRQ number settings is shown in Figure 7-4.

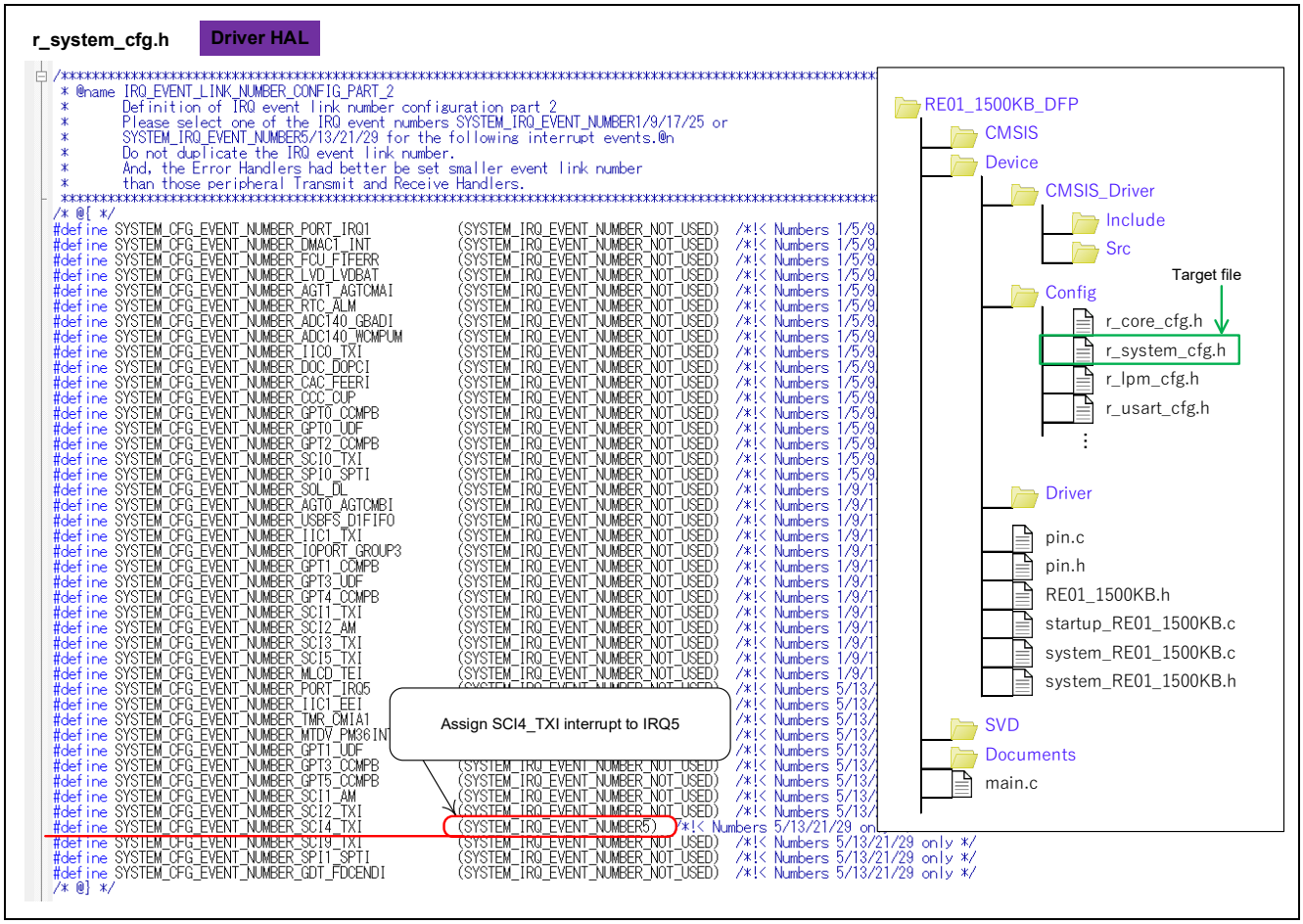


Figure 7-4 Example of IRQ Number Settings

7.1.2.2 Pin settings used by peripheral functions

The processing of pin setting functions in pin.c should be edited according to the operating environment. For editing the pin setting functions, see section 6.5.2, Editing Driver Functions.

An example of setting pins used by peripheral functions is shown in Figure 7-5.

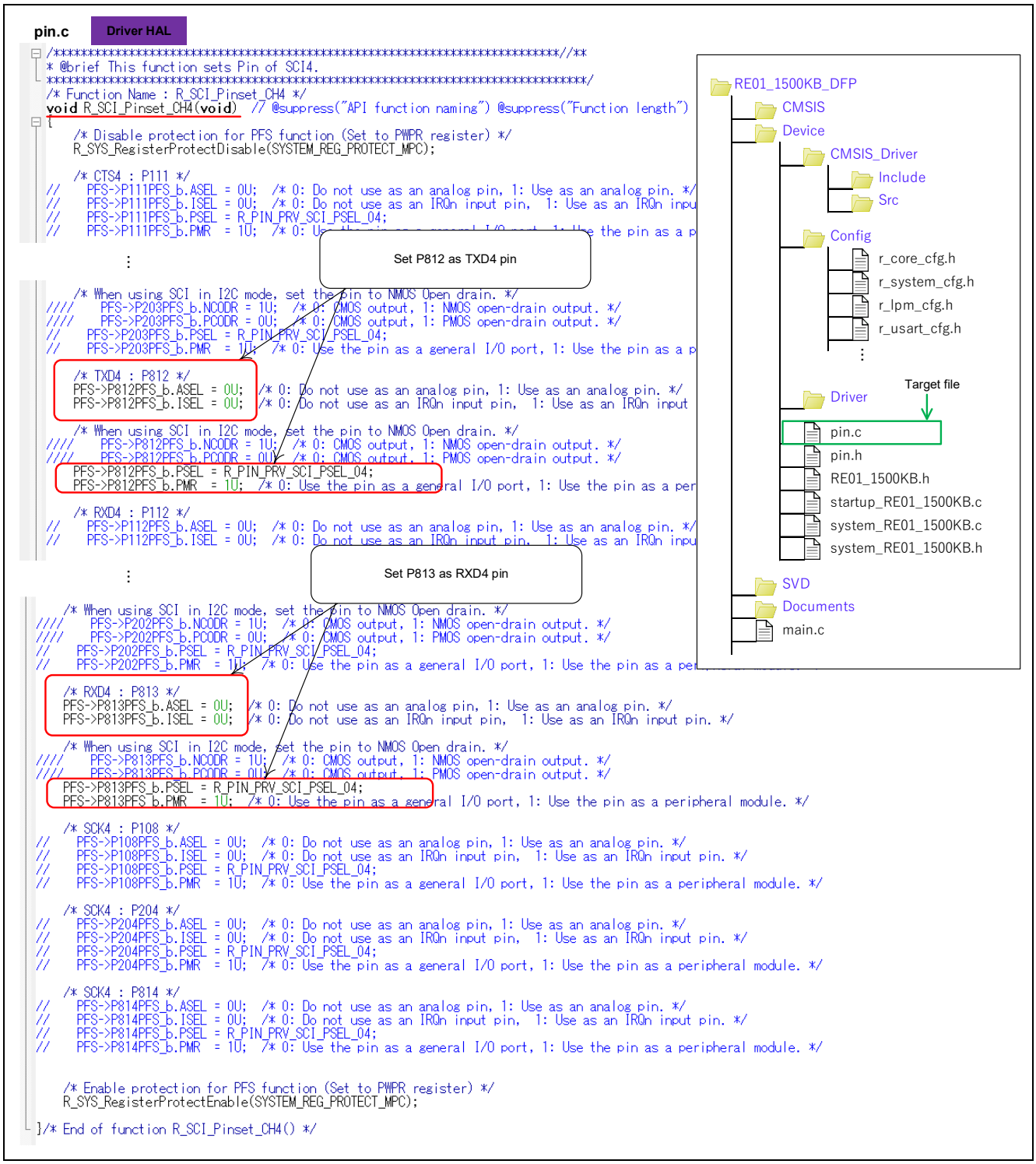


Figure 7-5 Example of Setting Pins Used by Peripheral Functions

7.1.2.3 Definitions of common function driver operating conditions

The R_SYS driver and R_LPM driver have the configuration definition headers r_system_cfg.h and r_lpm_cfg.h which define operating conditions. The definition values in the configuration definition header should be edited according to the operating environment.

An example of settings of common function driver operating conditions is shown in Figure 7-6.

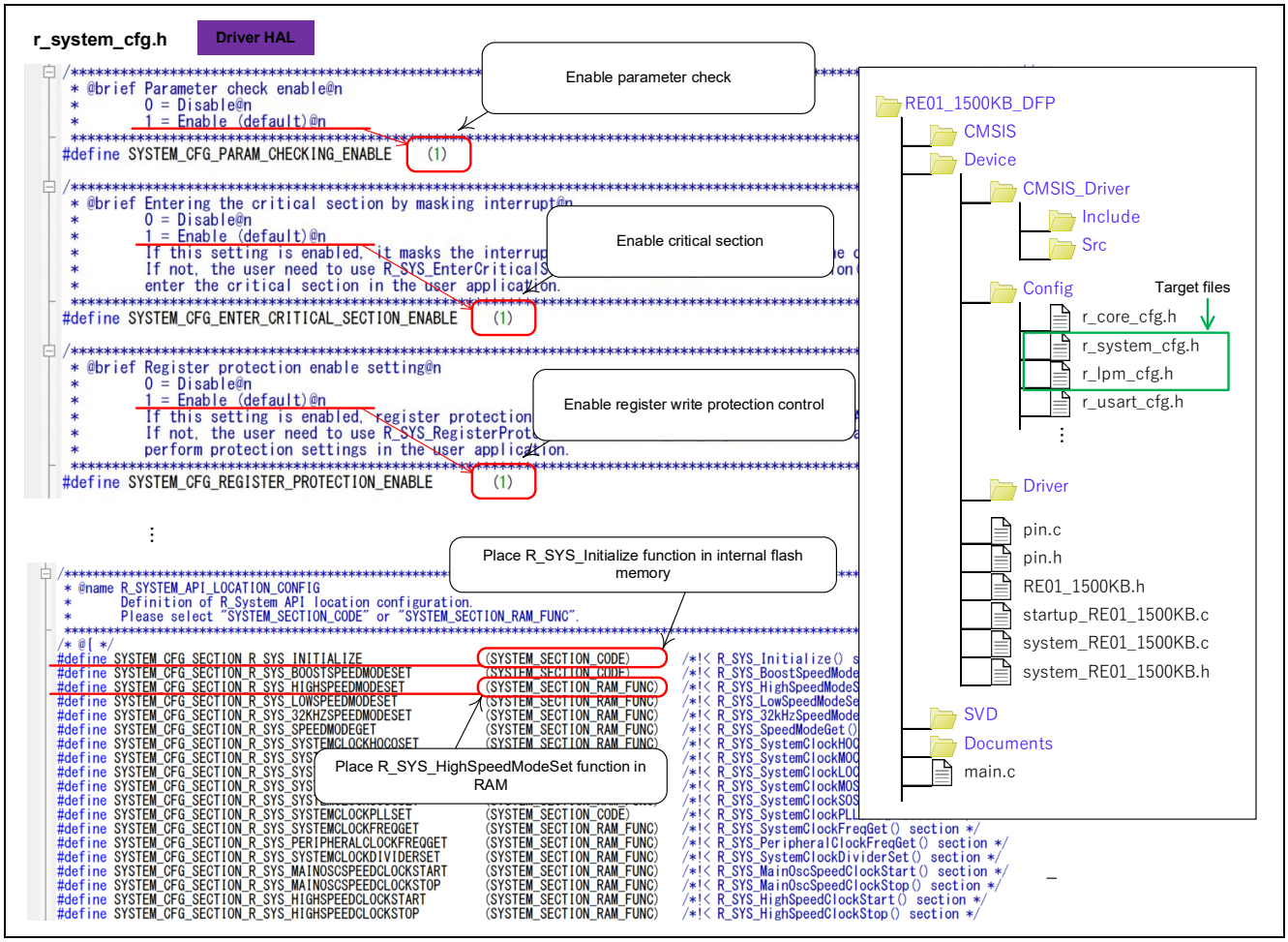


Figure 7-6 Example of Settings of Common Function Driver Operating Conditions

7.1.2.4 Definitions of clock used by R_SYS driver

A clock control function of the R_SYS driver sets the clock according to the settings in r_core_cfg.h. The definition values in r_core_cfg.h should be edited according to the operating environment. For settings in r_core_cfg.h, see section 6.4.1, Clock Definitions. The clock definitions in r_core_cfg.h are also referenced in startup processing.

An example of clock settings used by the R_SYS driver is shown in Figure 7-7.

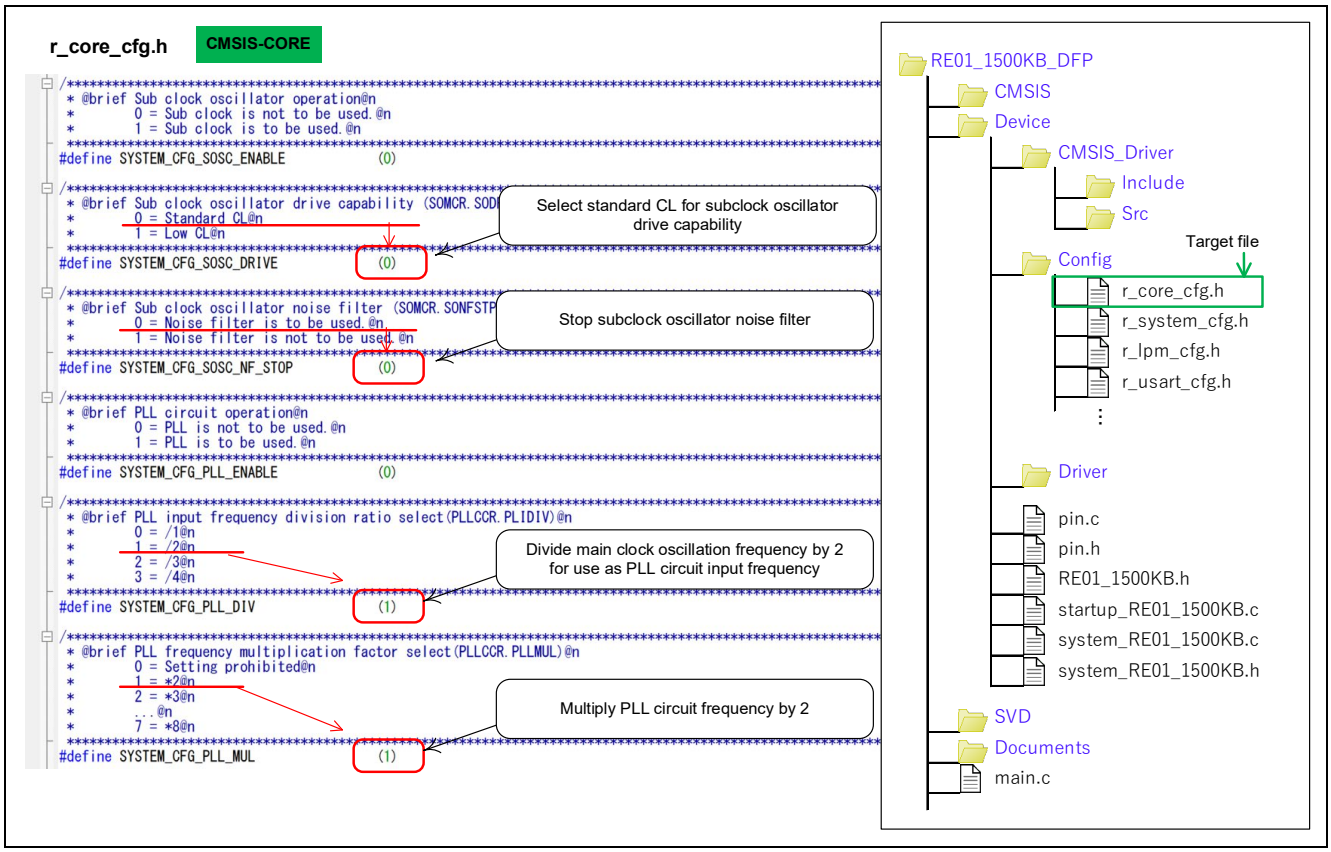


Figure 7-7 Example of Clock Settings Used by R_SYS Driver

7.1.3 Preparation of Peripheral Function Drivers

7.1.3.1 Definitions of operating conditions for peripheral function drivers

Each driver has a configuration definition header r_xxx_cfg.h (where "xxx" is the driver function name) that defines operating conditions. The definition values in r_xxx_cfg.h file should be edited according to the operating environment.

For a setting example, see section 7.1.2.3, Definitions of common function driver operating conditions.

7.2 User Program Creation

When creating a user program, there are settings that are necessary in order to use peripheral functions.

In this package, when using peripheral functions not supported by drivers, a program with the relevant functions must be created. Functions that use drivers and functions that do not use drivers may coexist in the same project without causing problems.

When using peripheral functions, the required processing differs when using peripheral function drivers and when not using peripheral function drivers.

Table 7-2 User Program Initial Settings and Required Control when Using Peripheral Functions

When using peripheral function drivers	When not using peripheral function drivers	Related functions and files
Initial settings		
Include common function driver headers		See Figure 7-8 ① for relevant files
RAM placement of desired programs		R_SYS_CodeCopy function
Power supply to I/O power supply domains		R_LPM_IOPowerSupplyModeSet function
Control of peripheral functions		
Include peripheral function driver headers	—	See Figure 7-8 ② for relevant files
Declaration of peripheral function driver instances	—	—
—	Include register definition headers	See Figure 7-8 ③ for relevant files
—	Module stop control	R_LPM_ModuleStart function R_LPM_ModuleStop function
—	Register write protect control	R_SYS_RegisterProtectDisable function R_SYS_RegisterProtectEnable function
—	Resource lock control	R_SYS_ResourceLock function
Control of pins		
—	Include R_PIN driver headers	See Figure 7-8 ④ for relevant files
—	Pin control	R_***_Pinset_*** functions R_***_Pinclr_*** functions
Control of interrupts		
—	Interrupt control	R_NVIC_EnableIRQ function R_NVIC_SetPriority function R_NVIC_ClearPendingIRQ function R_SYS_IrqEventLinkSet function R_SYS_IrqStatusClear function etc.

*: "***" differs depending on the driver and function used.

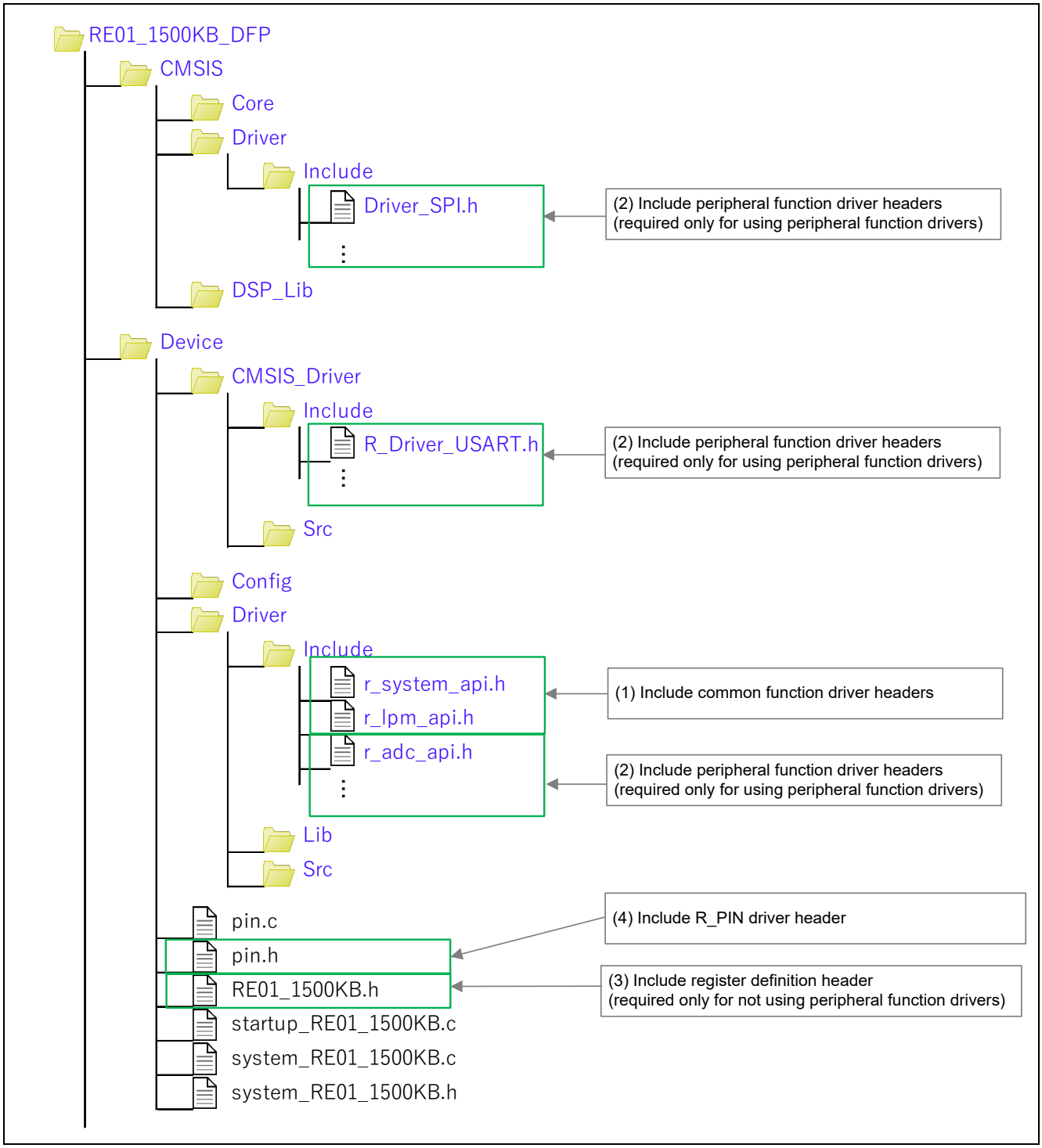


Figure 7-8 Files to Include when Creating User Program

7.2.1 Initial Settings

In the initial settings, R_SYS driver and R_LPM driver functions are used. A user program should include the common function driver headers for the R_SYS driver and R_LPM driver and should make initial settings.

7.2.1.1 Include common function driver headers

The file r_system_api.h should be included so that the R_SYS driver can be used.

The file r_lpm_api.h should be included so that the R_LPM driver can be used.

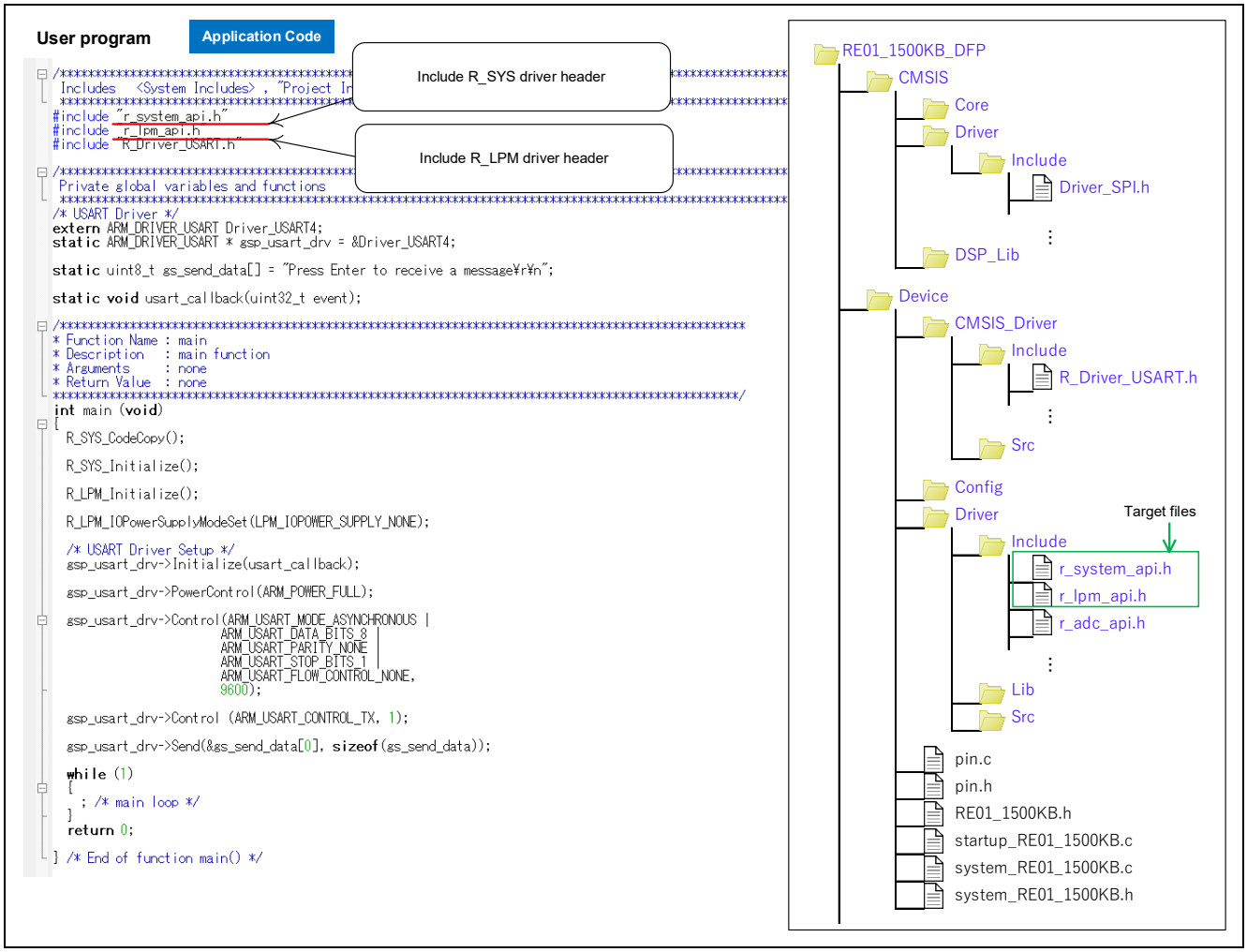
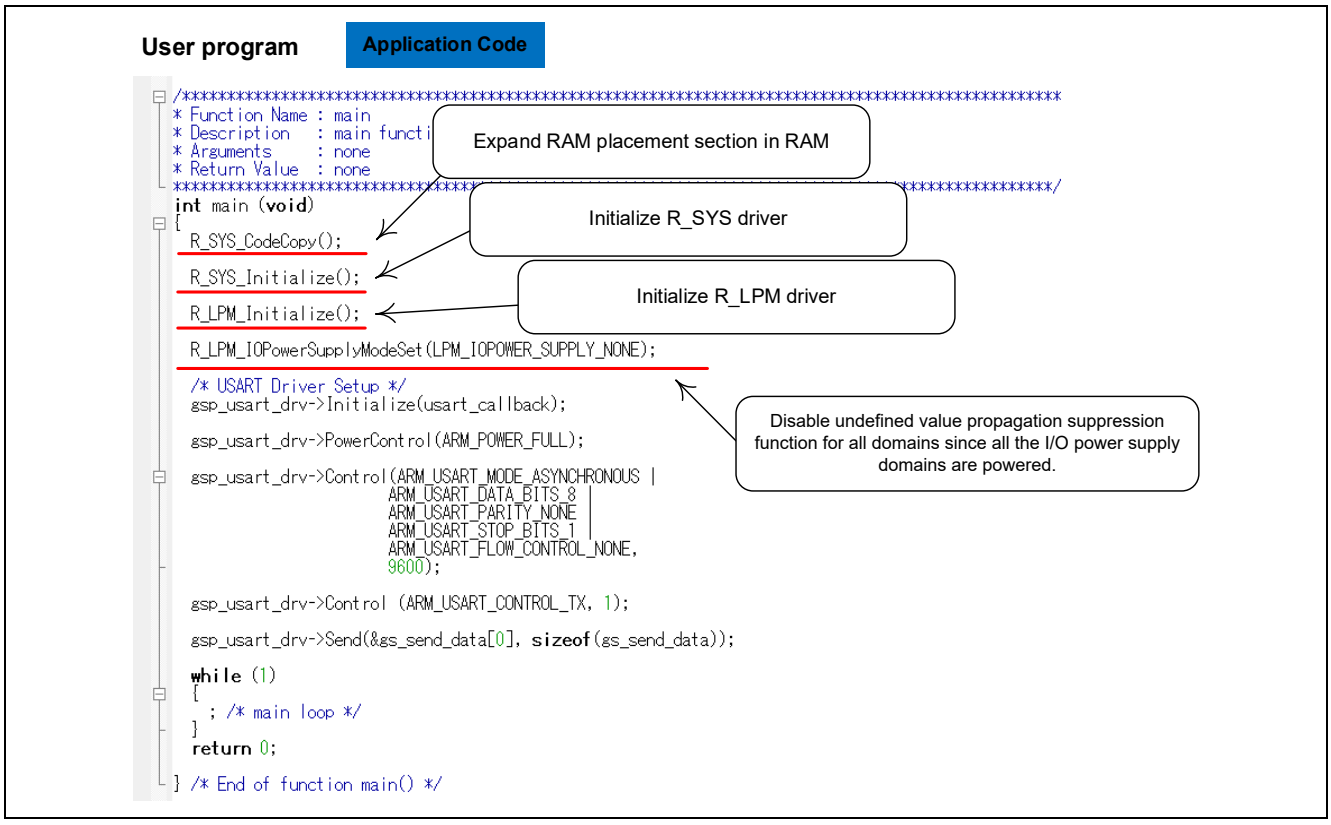


Figure 7-9 Including of Common Function Driver Headers

7.2.1.2 RAM expansion of desired programs

In this device, power consumption can be reduced by shutting off power to internal flash memory. When running a program expanded in RAM after having shut off power to internal flash memory, the R_SYS_CodeCopy function should be executed at the beginning of the user program to expand a desired program in RAM.

RAM placement of a program cannot be accomplished by this processing alone. For details, see section 6.6, RAM Placement of Programs.




```
/*****************************************************************************************************************
* Function Name : main
* Description   : main functi
* Arguments     : none
* Return Value  : none
*****************************************************************************************************************/
int main (void)
{
  R_SYS_CodeCopy();

  R_SYS_Initialize();

  R_LPM_Initialize();

  R_LPM_IOPowerSupplyModeSet(LPM_IOPOWER_SUPPLY_NONE);

  /* USART Driver Setup */
  gsp_usart_drv->Initialize(usart_callback);

  gsp_usart_drv->PowerControl(ARM_POWER_FULL);

  gsp_usart_drv->Control(ARM_USART_MODE_ASYNCHRONOUS |
                         ARM_USART_DATA_BITS_8 |
                         ARM_USART_PARITY_NONE |
                         ARM_USART_STOP_BITS_1 |
                         ARM_USART_FLOW_CONTROL_NONE,
                         9600);

  gsp_usart_drv->Control (ARM_USART_CONTROL_TX, 1);

  gsp_usart_drv->Send(&gs_send_data[0], sizeof(gs_send_data));

  while (1)
  {
    ; /* main loop */
  }
  return 0;
} /* End of function main() */
```

Figure 7-10 Example of Settings for RAM Expansion of a Program and Power Supply to an I/O Power Supply Domain

7.2.1.3 Control of the I/O power supply domain undefined value propagation suppression function

After reset cancellation, the undefined value propagation suppression function is enabled for all I/O power supply domains other than IOVCC. The R_LPM_IOPowerSupplyModeSet function should be executed according to the state of power supply to power supply domains in the operating environment, to disable the undefined value propagation suppression function for any desired I/O power supply domains.

For I/O power supply domain, see section 6.2, Control of Undefined Value Propagation Suppression in I/O Power Supply Domains.

This setting cannot be made by a peripheral function driver. It should be performed by the user program.

7.2.2 Control of Peripheral Functions

7.2.2.1 Including peripheral function driver headers

Each driver has a header file that makes necessary definitions in order to use the driver. The header file of a peripheral function driver to be used should be included.

Some of the drivers of CMSIS-Driver support Renesas-specific expansion functions. When using the following drivers of CMSIS-Driver, the extended header should be included. When the extended header is included, it is not necessary to include the standard header.

Table 7-3 Drivers Supporting Extended Functions

Driver name	Extended header file to include
R_I2C	R_Driver_I2C.h
R_USART	R_Driver_USART.h

This processing is necessary only when using peripheral function drivers. It is not required if peripheral function drivers are not used.

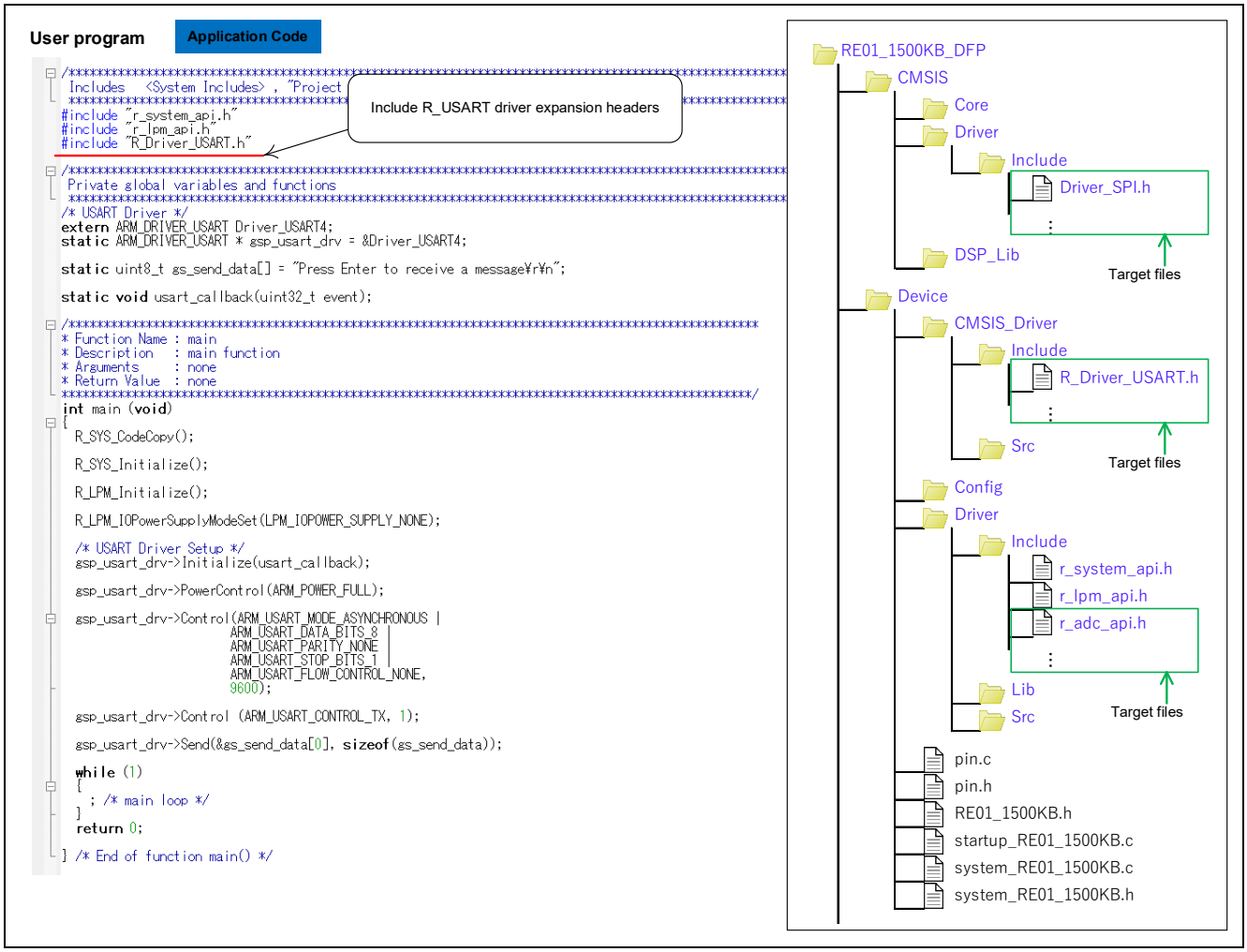


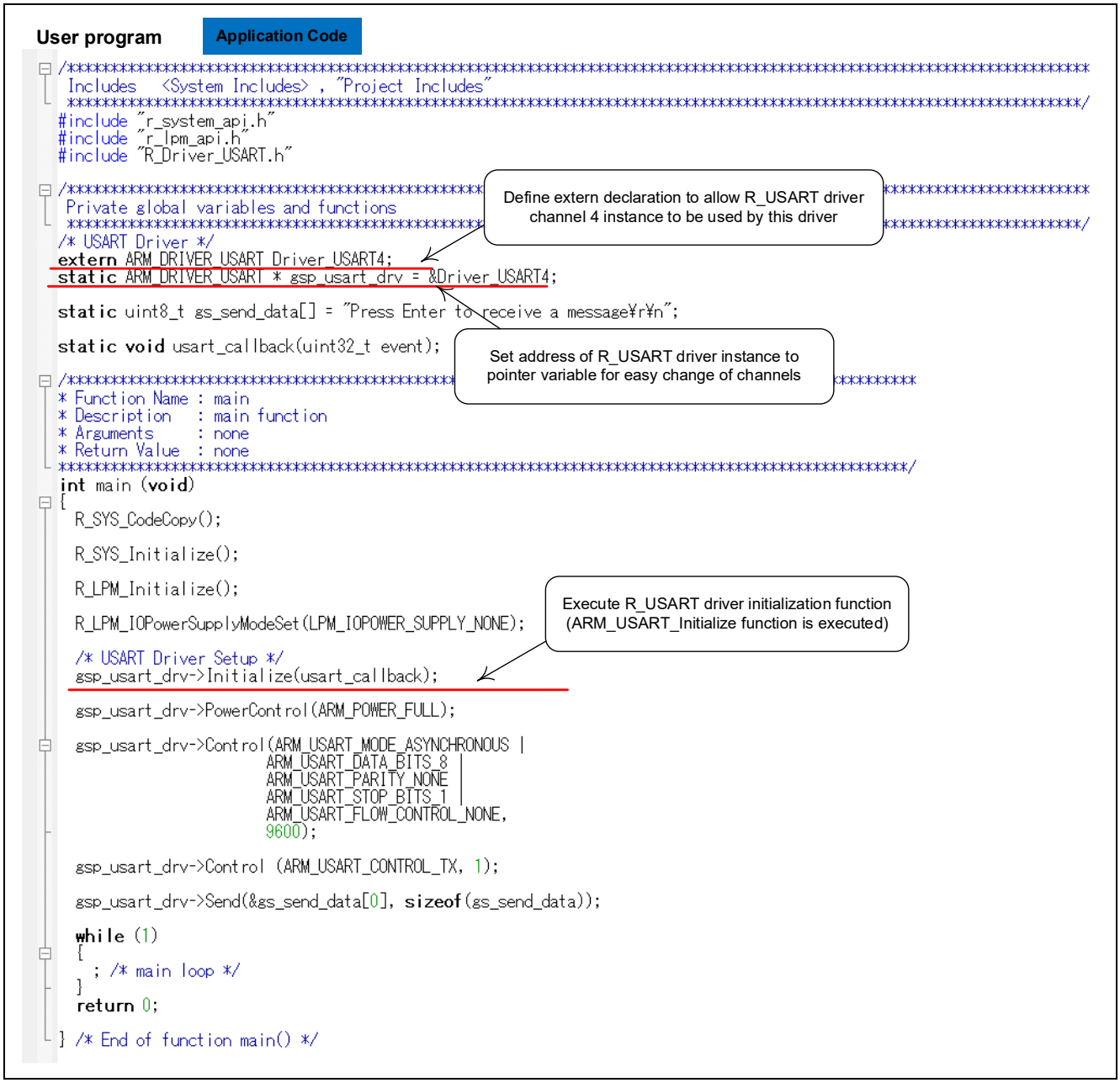
Figure 7-11 Example of Including Peripheral Function Driver Header

7.2.2.2 Instance declaration for peripheral function drivers

In some drivers, instances are provided by channel.

When using these drivers, function pointers within an instance can be used to execute each driver function. For the method for executing each driver function, refer to the driver specifications that are presented in chapter 4.

This processing is necessary only when using peripheral function drivers. It is not required if peripheral function drivers are not used.




```
/**********************************************************************************************************************
 Includes   <System Includes> , "Project Includes"
 **********************************************************************************************************************/
#include "r_system_api.h"
#include "r_lpm_api.h"
#include "R_Driver_USART.h"

/**********************************************************************************************************************
 Private global variables and functions
 **********************************************************************************************************************/
/* USART Driver */
extern ARM_DRIVER_USART Driver_USART4;
static ARM_DRIVER_USART * gsp_usart_drv = &Driver_USART4;

static uint8_t gs_send_data[] = "Press Enter to receive a message¥r¥n";

static void usart_callback(uint32_t event);

/**********************************************************************************************************************
* Function Name : main
* Description   : main function
* Arguments     : none
* Return Value  : none
**********************************************************************************************************************/
int main (void)
{
  R_SYS_CodeCopy();

  R_SYS_Initialize();

  R_LPM_Initialize();

  R_LPM_IOPowerSupplyModeSet(LPM_IOPOWER_SUPPLY_NONE);

  /* USART Driver Setup */
  gsp_usart_drv->Initialize(usart_callback);

  gsp_usart_drv->PowerControl(ARM_POWER_FULL);

  gsp_usart_drv->Control(ARM_USART_MODE_ASYNCHRONOUS |
                         ARM_USART_DATA_BITS_8 |
                         ARM_USART_PARITY_NONE |
                         ARM_USART_STOP_BITS_1 |
                         ARM_USART_FLOW_CONTROL_NONE,
                         9600);

  gsp_usart_drv->Control (ARM_USART_CONTROL_TX, 1);

  gsp_usart_drv->Send(&gs_send_data[0], sizeof(gs_send_data));

  while (1)
  {
    ; /* main loop */
  }
  return 0;
} /* End of function main() */
```

Figure 7-12 Example of Use of Instance Declaration of Peripheral Function Driver

7.2.2.3 Including register definition headers

By using register definition headers, hardware register write/read operations can be performed using the register name rather than an address. When using a register definition header, RE01_1500KB.h should be included.

This processing is necessary only when not using peripheral function drivers. It is not required if peripheral function drivers are used.

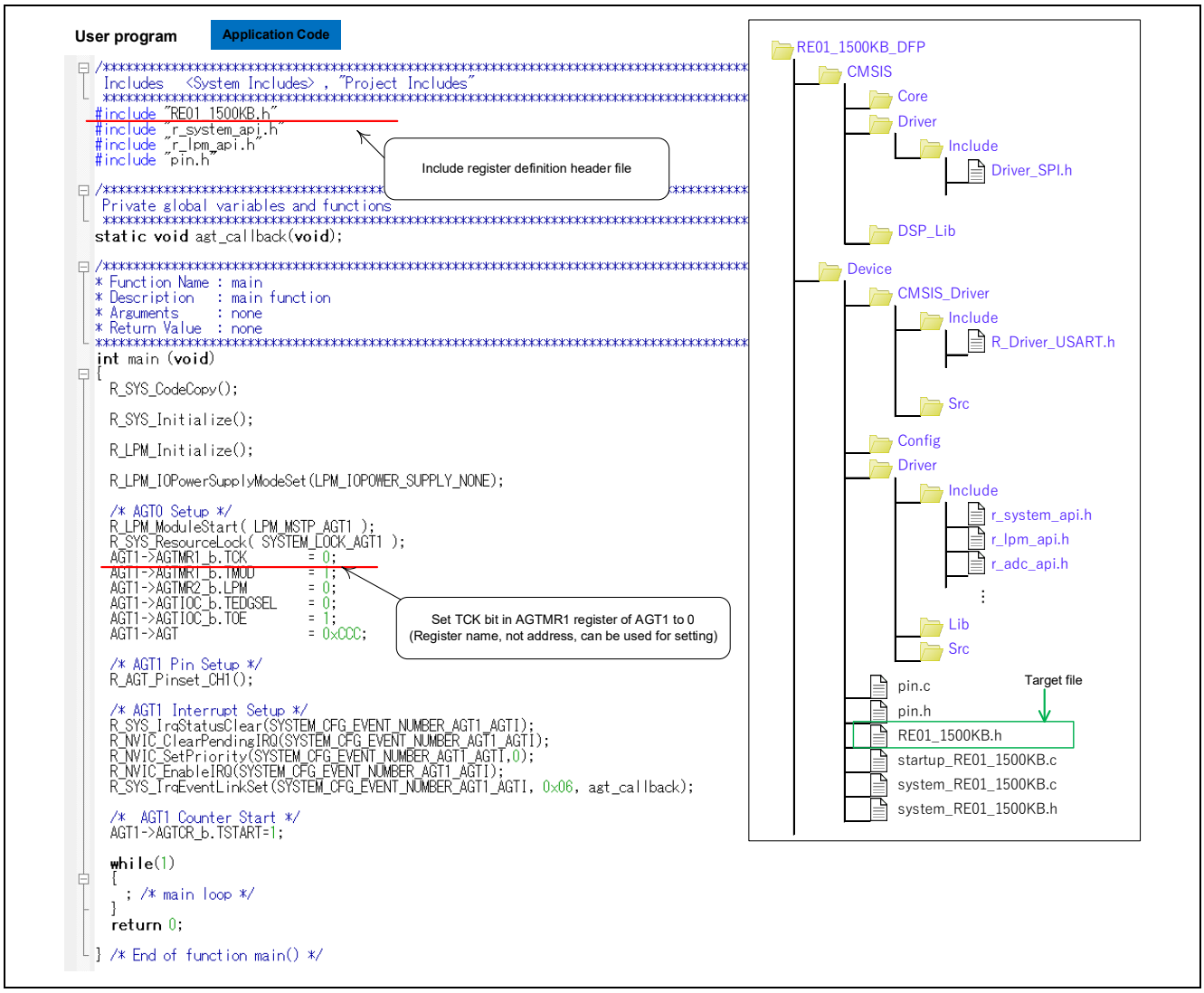


Figure 7-13 Example of Use of Register Definition File

7.2.2.4 Module stop control

Except for some peripheral functions, this device has module stop functions.

After reset cancellation, all module stop functions other than DMA/DTC are enabled (peripheral functions are stopped). When using a relevant peripheral function in a user program, prior to accessing the peripheral function register, the R_LPM_ModuleStart function should be used to disable the module stop function (peripheral function operates).

This processing is necessary only when not using peripheral function drivers. It is not required if peripheral function drivers are used.

- R_LPM_ModuleStart function: Module operates
- R_LPM_ModuleStop function: Module stopped ←initial state (except for DMA/DTC)

User program **Application Code**

```
/***********************************************************************************************************************
 Includes   <System Includes> , "Project Includes"
 ***********************************************************************************************************************/
#include "RE01_1500KB.h"
#include "r_system_api.h"
#include "r_lpm_api.h"
#include "pin.h"

/***********************************************************************************************************************
 Private global variables and functions
 ***********************************************************************************************************************/
static void agt_callback(void);

/*****************************************************************************************************************
* Function Name : main
* Description   : main function
* Arguments     : none
* Return Value  : none
*****************************************************************************************************************/
int main (void)
{
  R_SYS_CodeCopy();

  R_SYS_Initialize();

  R_LPM_Initialize();

  R_LPM_IOPowerSupplyModeSet(LPM_IOPOWER_SUPPLY_NONE);

  /* AGT0 Setup */
  R_LPM_ModuleStart( LPM_MSTP_AGT1 );
  R_SYS_ResourceLock( SYSTEM_LOCK_AGT1 );
  AGT1->AGTMR1_b.TCK        = 0;
  AGT1->AGTMR1_b.TMOD       = 1;
  AGT1->AGTMR2_b.LPM        = 0;
  AGT1->AGTIOC_b.TEDGSEL    = 0;
  AGT1->AGTIOC_b.TOE        = 1;
  AGT1->AGT                 = 0xCCC;

  /* AGT1 Pin Setup */
  R_AGT_Pinset_CH1();

  /* AGT1 Interrupt Setup */
  R_SYS_IrqStatusClear(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_NVIC_ClearPendingIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI,0);
  R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI, 0x06, agt_callback);

  /*  AGT1 Counter Start */
  AGT1->AGTCR_b.TSTART=1;

  while(1)
  {
    ; /* main loop */
  }
  return 0;
} /* End of function main() */
```

Cancel AGT1 module stop function and start AGT1 operation

Figure 7-14 Example of Module Stop Control

7.2.2.5 Register write protect control

Some of the registers of this device have a write protect function. After reset cancellation, all registers have the write protect function enabled (are protected from writing to the registers). Hence when a user program writes to a register, before performing the register write operation, the R_SYS_RegisterProtectDisable function should be used to disable protection.

- R_SYS_RegisterProtectDisable function: Disables protection
- R_SYS_RegisterProtectEnable function: Enables protection (initial state)

This processing is necessary only when not using peripheral function drivers. It is not required if peripheral function drivers are used.

When a peripheral function driver is used, this processing is performed within the driver. When register write protect control is disabled in the configuration definition header for the driver, this processing is not performed within the driver, and so processing should be performed in the user program.

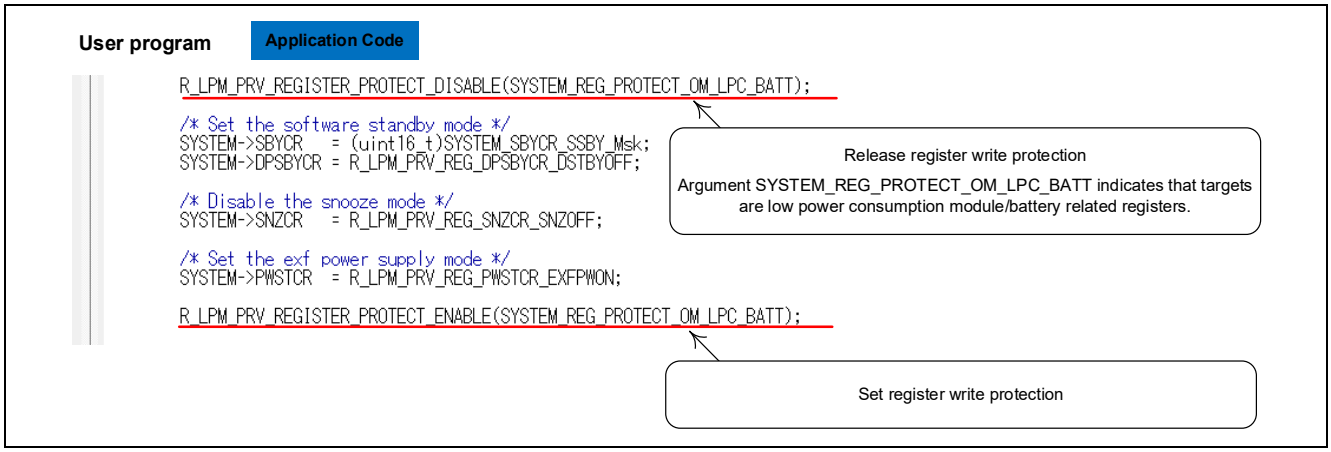


Figure 7-15 Example of Register Write Protect Control

7.2.2.6 Resource lock control

The R_SYS driver provides a resource lock function to detect conflict of hardware functions.

When using a peripheral function, the R_SYS_ResourceLock function should be used to lock resources.

- R_SYS_ResourceLock function: Locks a resource
- R_SYS_ResourceUnlock function: Unlocks a resource (initial state)

This processing is necessary only when not using peripheral function drivers. It is not required if peripheral function drivers are used. When a peripheral function driver is used, this processing is performed within the driver.

User program Application Code

```
/***********************************************************************************************************************
 Includes   <System Includes> , "Project Includes"
 ***********************************************************************************************************************/
#include "RE01_1500KB.h"
#include "r_system_api.h"
#include "r_lpm_api.h"
#include "pin.h"

/***********************************************************************************************************************
 Private global variables and functions
 ***********************************************************************************************************************/
static void agt_callback(void);

/***********************************************************************************************************************
* Function Name : main
* Description   : main function
* Arguments     : none
* Return Value  : none
***********************************************************************************************************************/
int main (void)
{
  R_SYS_CodeCopy();

  R_SYS_Initialize();

  R_LPM_Initialize();

  R_LPM_IOPowerSupplyModeSet(LPM_IOPOWER_SUPPLY_NONE);

  /* AGT0 Setup */
  R_LPM_ModuleStart( LPM_MSTP_AGT1 );
  R_SYS_ResourceLock( SYSTEM_LOCK_AGT1 );
  AGT1->AGTMR1_b.TCK        = 0;
  AGT1->AGTMR1_b.TMOD       = 1;
  AGT1->AGTMR2_b.LPM        = 0;
  AGT1->AGTIOC_b.TEDGSEL    = 0;
  AGT1->AGTIOC_b.TOE        = 1;
  AGT1->AGT                 = 0xCCC;

  /* AGT1 Pin Setup */
  R_AGT_Pinset_CH1();

  /* AGT1 Interrupt Setup */
  R_SYS_IrqStatusClear(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_NVIC_ClearPendingIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI,0);
  R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI, 0x06, agt_callback);

  /*  AGT1 Counter Start */
  AGT1->AGTCR_b.TSTART=1;

  while(1)
  {
    ; /* main loop */
  }
  return 0;

} /* End of function main() */
```

Lock AGT1 hardware resources

Figure 7-16 Example of Hardware Resource Lock Control

7.2.3 Control of Pins

The R_PIN driver provides pin setting functions that are used by peripheral functions. A user program can use the pin setting functions of the R_PIN driver to set pins.

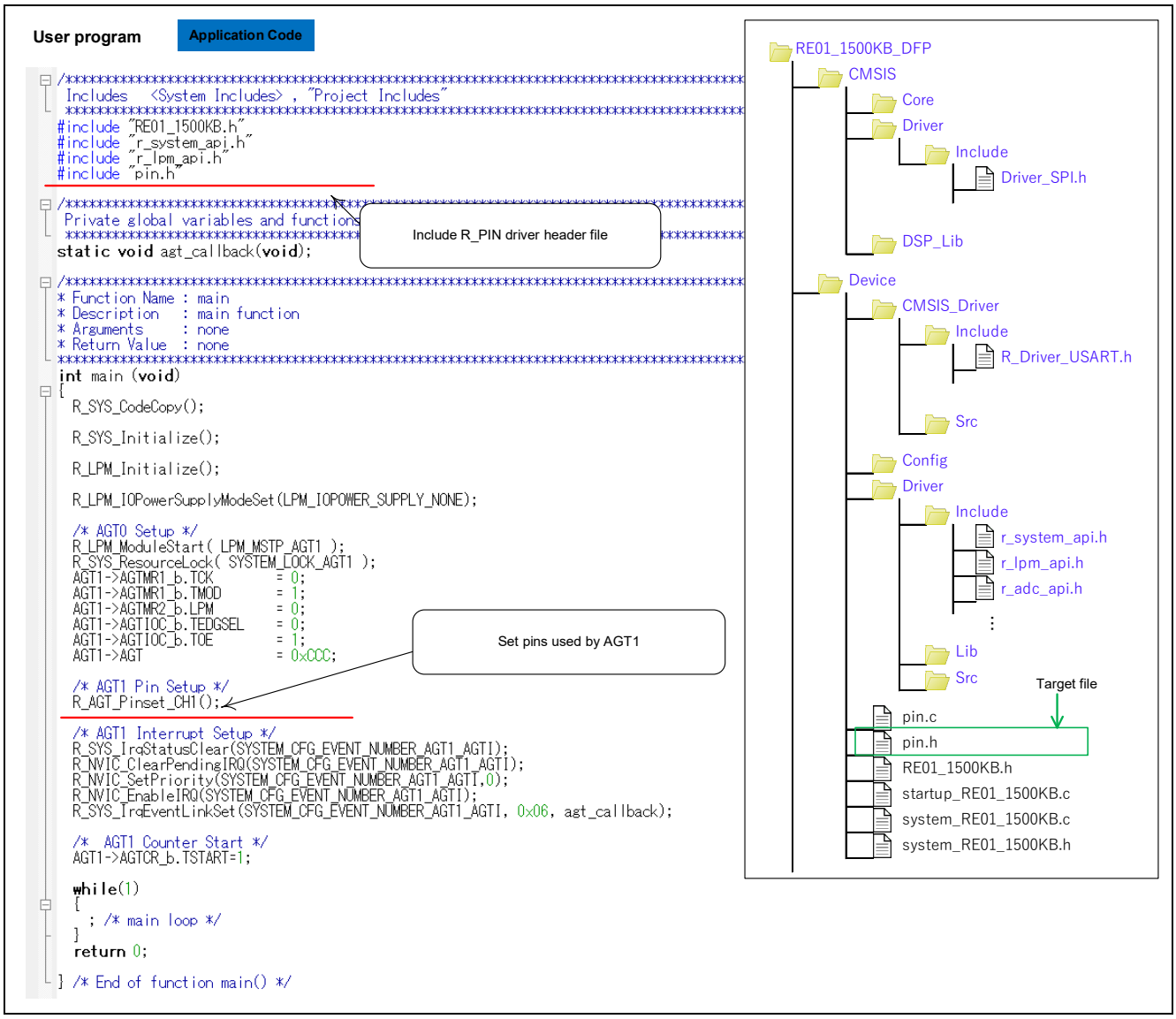
7.2.3.1 Including R_PIN driver header

The R_PIN driver has a header file that makes the necessary definitions to use the driver. When using the R_PIN driver to make pin settings, the pin.h file should be included.

7.2.3.2 Pin control

The R_PIN driver functions can be used to make pin settings.

When a peripheral function driver is used, this processing is performed within the driver.




```
/***********************************************************************************************************************
 Includes   <System Includes> , "Project Includes"
 ***********************************************************************************************************************/
#include "RE01_1500KB.h"
#include "r_system_api.h"
#include "r_lpm_api.h"
#include "pin.h"

/***********************************************************************************************************************
 Private global variables and functions
 ***********************************************************************************************************************/
static void agt_callback(void);

/***********************************************************************************************************************
* Function Name : main
* Description   : main function
* Arguments     : none
* Return Value  : none
***********************************************************************************************************************/
int main (void)
{
  R_SYS_CodeCopy();

  R_SYS_Initialize();

  R_LPM_Initialize();

  R_LPM_IOPowerSupplyModeSet(LPM_IOPOWER_SUPPLY_NONE);

  /* AGT0 Setup */
  R_LPM_ModuleStart( LPM_MSTP_AGT1 );
  R_SYS_ResourceLock( SYSTEM_LOCK_AGT1 );
  AGT1->AGTMR1_b.TCK        = 0;
  AGT1->AGTMR1_b.TMOD       = 1;
  AGT1->AGTMR2_b.LPM        = 0;
  AGT1->AGTIOC_b.TEDGSEL    = 0;
  AGT1->AGTIOC_b.TOE        = 1;
  AGT1->AGT                 = 0xCCC;

  /* AGT1 Pin Setup */
  R_AGT_Pinset_CH1();

  /* AGT1 Interrupt Setup */
  R_SYS_IrqStatusClear(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_NVIC_ClearPendingIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI,0);
  R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI, 0x06, agt_callback);

  /*  AGT1 Counter Start */
  AGT1->AGTCR_b.TSTART=1;

  while(1)
  {
    ; /* main loop */
  }
  return 0;
} /* End of function main() */
```

Figure 7-17 Example of Pin Control

7.2.4 Controlling Interrupts

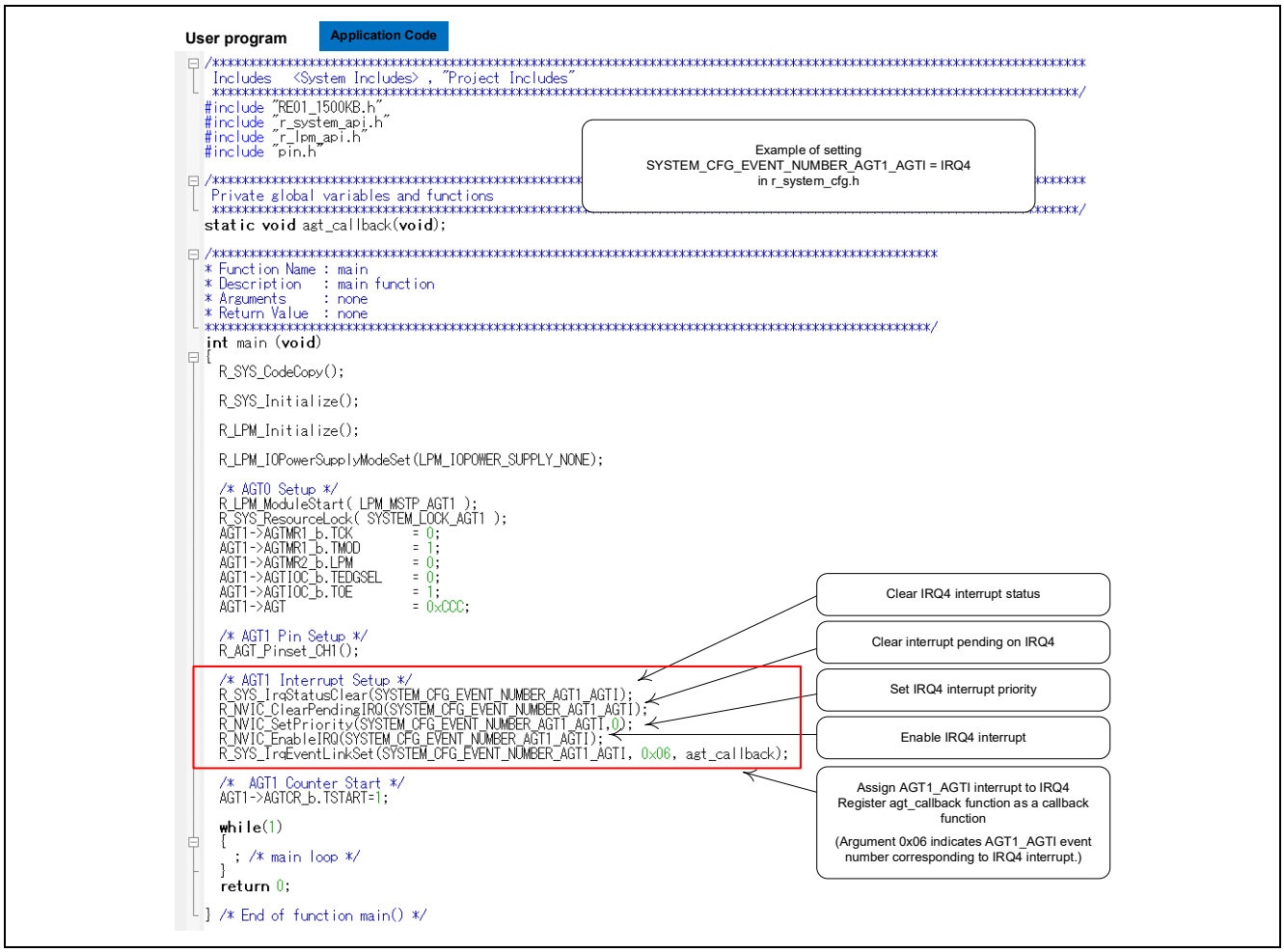
When using interrupts, an interrupt from a peripheral function must be allocated to an IRQ number. The r_system_cfg.h file, which defines IRQ numbers, is included in r_system_api.h.

7.2.4.1 Interrupt control

When using interrupts, settings must be made for each of peripheral functions, ICU, and NVIC.

This section shows examples of interrupt settings for ICU and NVIC. Interrupt settings for peripheral functions should be set in the registers of the peripheral functions. For details on interrupt settings, see section 6.3, Interrupt Control.

When a peripheral function driver is used, this processing is performed within the driver.




```
/*****************************************************************************************************************
 Includes   <System Includes> , "Project Includes"
 *****************************************************************************************************************/
#include "RE01_1500KB.h"
#include "r_system_api.h"
#include "r_lpm_api.h"
#include "pin.h"

/*****************************************************************************************************************
 Private global variables and functions
 *****************************************************************************************************************/
static void agt_callback(void);

/*****************************************************************************************************************
* Function Name : main
* Description   : main function
* Arguments     : none
* Return Value  : none
*****************************************************************************************************************/
int main (void)
{
  R_SYS_CodeCopy();

  R_SYS_Initialize();

  R_LPM_Initialize();

  R_LPM_IOPowerSupplyModeSet(LPM_IOPOWER_SUPPLY_NONE);

  /* AGT0 Setup */
  R_LPM_ModuleStart( LPM_MSTP_AGT1 );
  R_SYS_ResourceLock( SYSTEM_LOCK_AGT1 );
  AGT1->AGTMR1_b.TCK        = 0;
  AGT1->AGTMR1_b.TMOD       = 1;
  AGT1->AGTMR2_b.LPM        = 0;
  AGT1->AGTIOC_b.TEDGSEL    = 0;
  AGT1->AGTIOC_b.TOE        = 1;
  AGT1->AGT                 = 0xCCC;

  /* AGT1 Pin Setup */
  R_AGT_Pinset_CH1();

  /* AGT1 Interrupt Setup */
  R_SYS_IrqStatusClear(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_NVIC_ClearPendingIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI,0);
  R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI, 0x06, agt_callback);

  /*  AGT1 Counter Start */
  AGT1->AGTCR_b.TSTART=1;

  while(1)
  {
    ; /* main loop */
  }
  return 0;
} /* End of function main() */
```

Figure 7-18 Example of Interrupt Control

7.3 User Program Creation Example

7.3.1 Example of Use of Peripheral Function Driver (UART Communication)

main.c file

```
#include "r_system_api.h"
#include "r_lpm_api.h"
#include "R_Driver_USART.h"

/* USART Driver */
extern ARM_DRIVER_USART Driver_USART4;
static ARM_DRIVER_USART * gsp_usart_drv = &Driver_USART4;

static uint8_t gs_send_data[] = "Press Enter to receive a message¥r¥n";
static void usart_callback(uint32_t event);

/* main function */
int main (void)
{
  R_SYS_CodeCopy();
  R_SYS_Initialize();
  R_LPM_Initialize();
  R_LPM_IOPowerSupplyModeSet(LPM_IOPOWER_SUPPLY_NONE);

  /* USART Driver Setup */
  gsp_usart_drv->Initialize(usart_callback);
  gsp_usart_drv->PowerControl(ARM_POWER_FULL);
  gsp_usart_drv->Control(ARM_USART_MODE_ASYNCHRONOUS |
            ARM_USART_DATA_BITS_8 |
            ARM_USART_PARITY_NONE |
            ARM_USART_STOP_BITS_1 |
            ARM_USART_FLOW_CONTROL_NONE,
            9600);
  gsp_usart_drv->Control (ARM_USART_CONTROL_TX, 1);
  gsp_usart_drv->Send(&gs_send_data[0], sizeof(gs_send_data));

  while (1); /* main loop */
  return 0;
} /* End of function main() */

/* callback function */
static void usart_callback (uint32_t event)
{
  switch (event)
  {
    case ARM_USART_EVENT_SEND_COMPLETE:
      ; /* Success */
    break;
    default:
      ; /*  */
    break;
  }
  return ;
} /* End of function usart_callback() */
```

Include header of common function driver R_USART

Preparation to use instance of R_USART driver channel 4

Initial settings

Control using R_USART driver

Callback function

pin.c file

```
/***********************************************************************//**
* @brief This function sets Pin of SCI4.
***********************************************************************/
/* Function Name : R_SCI_Pinset_CH4 */
void R_SCI_Pinset_CH4(void)
{
    R_SYS_RegisterProtectDisable(SYSTEM_REG_PROTECT_MPC);

    /* TXD4 : P812 */
    PFS->P812PFS_b.ASEL = 0U;
    PFS->P812PFS_b.ISEL = 0U;
    PFS->P812PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
    PFS->P812PFS_b.PMR  = 1U;

    /* RXD4 : P813 */
    PFS->P813PFS_b.ASEL = 0U;
    PFS->P813PFS_b.ISEL = 0U;
    PFS->P813PFS_b.PSEL = R_PIN_PRV_SCI_PSEL_04;
    PFS->P813PFS_b.PMR  = 1U;

    R_SYS_RegisterProtectEnable(SYSTEM_REG_PROTECT_MPC);

}/* End of function R_SCI_Pinset_CH4() */
```

Set P812 to TXD4

Set P813 to RXD4

r_system_api_cfg.h file

```
/***********************************************************************//**
 * @name IRQ_EVENT_LINK_NUMBER_CONFIG_PART_2
 *    Definition of IRQ event link number configuration part 2
 *    Please select one of the IRQ event numbers SYSTEM_IRQ_EVENT_NUMBER1/9/17/25 or
 *    SYSTEM_IRQ_EVENT_NUMBER5/13/21/29 for the following interrupt events.@n
 *    Do not duplicate the IRQ event link number.
 *    And, the Error Handlers had better be set smaller event link number
 *    than those peripheral Transmit and Receive Handlers.
 ***********************************************************************/
/* @{ */

// 省略
#define SYSTEM_CFG_EVENT_NUMBER_GPT1_UDF   (SYSTEM
#define SYSTEM_CFG_EVENT_NUMBER_GPT3_CCMPB (SYSTEM
#define SYSTEM_CFG_EVENT_NUMBER_GPT5_CCMPB (SYSTEM_IRQ_EVENT_NUMB
#define SYSTEM_CFG_EVENT_NUMBER_SCI1_AM    (SYSTEM_IRQ_EVENT_NUMBE
#define SYSTEM_CFG_EVENT_NUMBER_SCI2_TXI   (SYSTEM_IRQ_EVENT_NUMBE
#define SYSTEM_CFG_EVENT_NUMBER_SCI4_TXI   (SYSTEM_IRQ_EVENT_NUMBER5)
#define SYSTEM_CFG_EVENT_NUMBER_SCI9_TXI   (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)
#define SYSTEM_CFG_EVENT_NUMBER_SPI1_SPTI  (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)
#define SYSTEM_CFG_EVENT_NUMBER_GDT_FDCENDI (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)
 /* @} */
```

Allocate SCI4_TXI interrupt to IRQ5

7.3.2 Example in which Peripheral Function Driver Is Not Used (Pulse Output)

main.c file

```
#include "RE01_1500KB.h"
#include "r_system_api.h"
#include "r_lpm_api.h"
#include "pin.h"

static void agt_callback(void);

/* main function */
int main (void)
{
  R_SYS_CodeCopy();
  R_SYS_Initialize();
  R_LPM_Initialize();
  R_LPM_IOPowerSupplyModeSet(LPM_IOPOWER_SUPPLY_NONE);

  /* AGT1 Setup */
  R_LPM_ModuleStart( LPM_MSTP_AGT1 );
  R_SYS_ResourceLock( SYSTEM_LOCK_AGT1 );
  AGT1->AGTMR1_b.TCK        = 0;
  AGT1->AGTMR1_b.TMOD    = 1;
  AGT1->AGTMR2_b.LPM     = 0;
  AGT1->AGTIOC_b.TEDGSEL    = 0;
  AGT1->AGTIOC_b.TOE     = 1;
  AGT1->AGT                 = 0xCCC;

  /* AGT1 Pin Setup */
  R_AGT_Pinset_CH1();

  /* AGT1 Interrupt Setup */
  R_SYS_IrqStatusClear(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_NVIC_ClearPendingIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI,0);
  R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI);
  R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI, 0x06, agt_callback);

  AGT1->AGTCR_b.TSTART=1;  // AGT1 Counter Start

  while (1); /* main loop */
  return 0;
} /* End of function main() */

/* callback function */
static void agt_callback (void)
{
  /* Clear underflow flag */
  AGT1->AGTCR_b.TUNDF = 0;

  return ;
} /* End of function agt_callback() */
```

Include register definition header
Include common function driver header
Include R_PIN driver header

Initial settings

Disable AGT1 module stop function
Lock AGT1 hardware resource

Use register definition header to write to register

Set pin used by AGT1

Set AGT1_AGTI interrupts

Callback function

pin.c file

```
/***********************************************************************//**
* @brief This function sets Pin of AGT1.
***********************************************************************/
/* Function Name : R_AGT_Pinset_CH1 */
void R_AGT_Pinset_CH1(void)
{
    R_SYS_RegisterProtectDisable(SYSTEM_REG_PROTECT_MPC);

    /* AGTIO1 : P309 */
    PFS->P309PFS_b.ASEL = 0U;
    PFS->P309PFS_b.ISEL = 0U;
    PFS->P309PFS_b.PSEL = R_PIN_PRV_AGT_PSEL;
    PFS->P309PFS_b.PMR  = 1U;

    /* AGTO1 : P308 */
    PFS->P308PFS_b.ASEL = 0U;
    PFS->P308PFS_b.ISEL = 0U;
    PFS->P308PFS_b.PSEL = R_PIN_PRV_AGT_PSEL;
    PFS->P308PFS_b.PMR  = 1U;

    R_SYS_RegisterProtectEnable(SYSTEM_REG_PROTECT_MPC);

}/* End of function R_AGT_Pinset_CH1 () */
```

Set P309 to AGTIO1

Set P308 to AGTO1

r_system_api_cfg.h file

```
/***********************************************************************//**
 * @name IRQ_EVENT_LINK_NUMBER_CONFIG_PART_1
 *    Definition of IRQ event link number configuration part 1
 *    Please select one of the IRQ event numbers SYSTEM_IRQ_EVENT_NUMBER0/8/16/24 or
 *    SYSTEM_IRQ_EVENT_NUMBER4/12/20/28 for the following interrupt events.@n
 *    Do not duplicate the IRQ event link number.
 *    And, the Error Handlers had better be set smaller event link number
 *    than those peripheral Transmit and Receive Handlers.
 ***********************************************************************/
/* @{ */
#define SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0     (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)
#define SYSTEM_CFG_EVENT_NUMBER_DMAC0_INT     (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)
#define SYSTEM_CFG_EVENT_NUMBER_DTC_COMPLETE  (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)
#define SYSTEM_CFG_EVENT_NUMBER_ICU_SNZCANCEL (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)
#define SYSTEM_CFG_EVENT_NUMBER_LVD_LVD1      (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)
#define SYSTEM_CFG_EVENT_NUMBER_AGT1_AGTI     (SYSTEM_IRQ_EVENT_NUMBER0)
#define SYSTEM_CFG_EVENT_NUMBER_WDT_NMIUNDF   (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)
#define SYSTEM_CFG_EVENT_NUMBER_ADC140_ADI    (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)
#define SYSTEM_CFG_EVENT_NUMBER_ADC140_WCMPM  (SYSTEM_

// omitted
/* @} */
```

Allocate AGT1_AGTI interrupt to IRQ0

8. Creating a Project

This chapter describes the method for changing settings according to the operating environment after starting the project included with this package.

8.1 EWARM Version

Double-click on the executable file included with this package to start the project using EWARM.

Executable file: project.eww

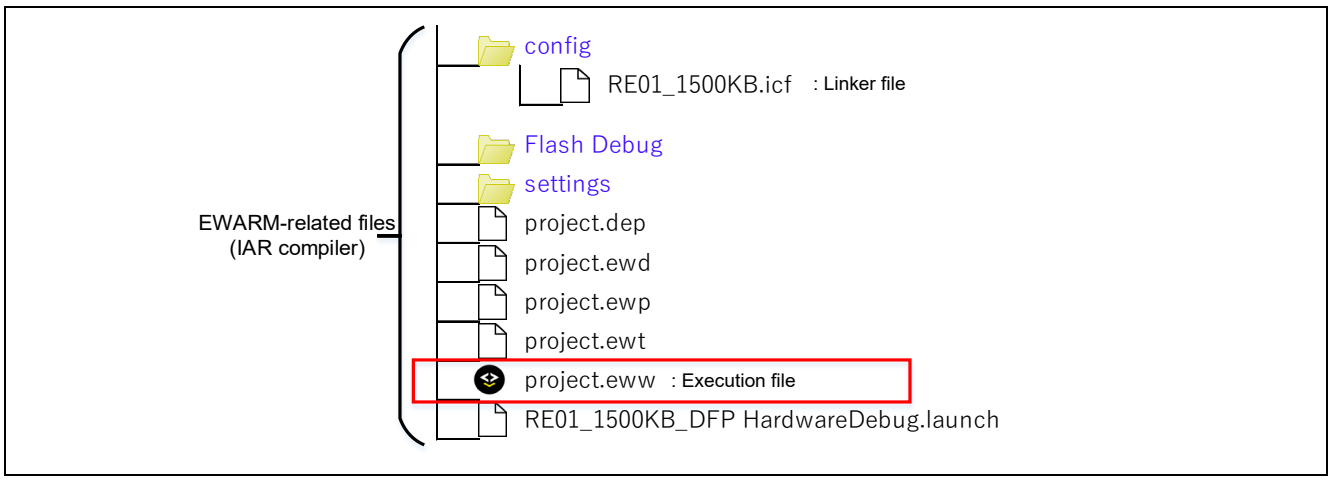


Figure 8-1 Project Executable File (EWARM)

The project settings are set on the project option screen. See Figure 8-2 to display the option screen.

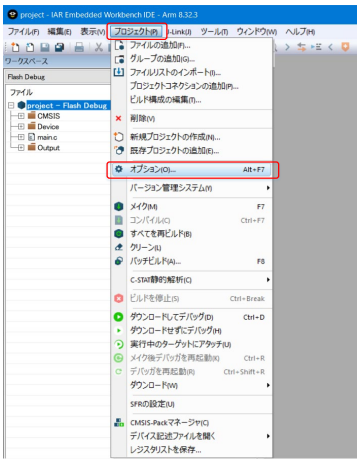


Figure 8-2 Displaying Option Screen (EWARM)

8.1.1 Setting Target Processor

In this package, the target processor is set to "RE01 1500KB R7F0E015D2CFB". When the user is to set the processor, the device to be used should be selected referring to Figure 8-3.

Device: Renesas R7F0E015D2CFB

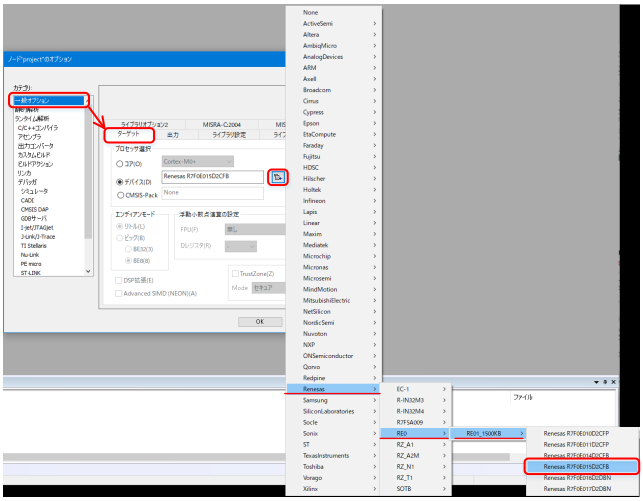


Figure 8-3 Example of Target Processor Setting

8.1.2 Linker File Setting

In this package, the linker file "RE01_1500KB.icf", with section definitions for RAM placement and MTB regions added, is set. When the user sets a linker file, the linker file to be used should be set referring to Figure 8-4.

Linker setting file: $PROJ_DIR$¥config¥RE01_1500KB.icf

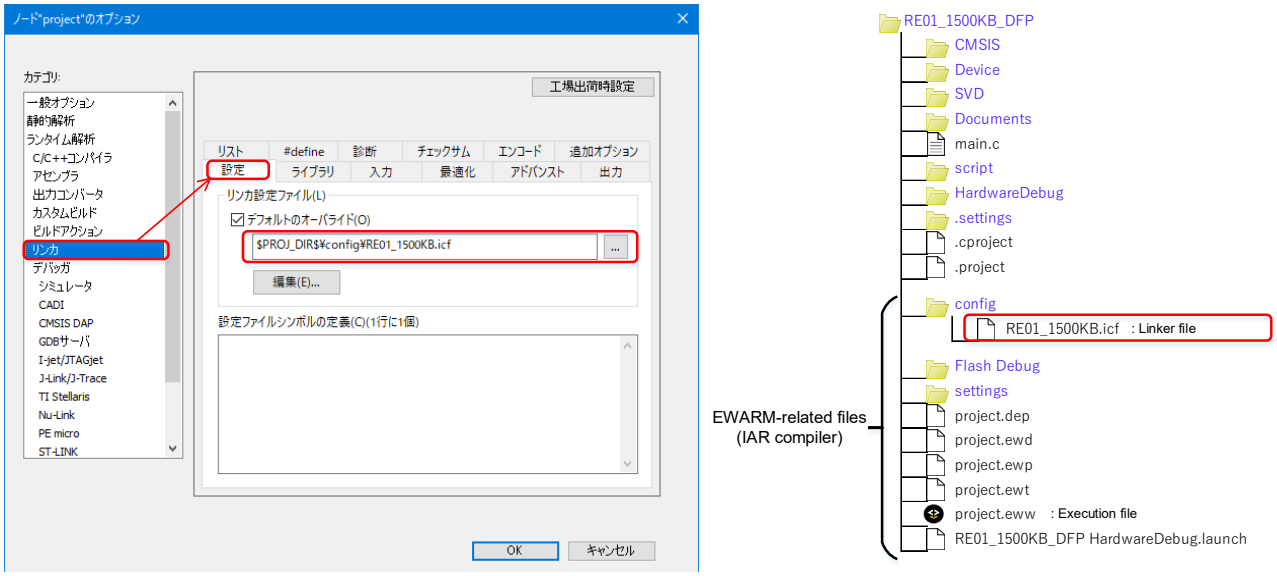


Figure 8-4 Example of Linker File Settings (EWARM)

For RAM placement, see section 6.6.1, RAM Placement Method Using RAM Placement Section.

For MTB region, see section 9.1, Securing MTB Region for Debugger.

8.1.3 Include Directory Settings

The drivers in this package include header files using filenames only, and therefore the location of the include files must be specified at compile time. In this package, the include directory necessary for using drivers is set.

When the user sets the include directory, the necessary path should be set referring to Figure 8-5.

Include path setting examples

- $PROJ_DIR$¥CMSIS¥
- $PROJ_DIR$¥CMSIS¥Core¥Include
- $PROJ_DIR$¥CMSIS¥Driver¥Include
- $PROJ_DIR$¥Device
- $PROJ_DIR$¥Device¥CMSIS_Driver¥Include
- $PROJ_DIR$¥Device¥Config
- $PROJ_DIR$¥Device¥Driver¥Include
- $PROJ_DIR$¥Device¥DSP_Lib¥Include

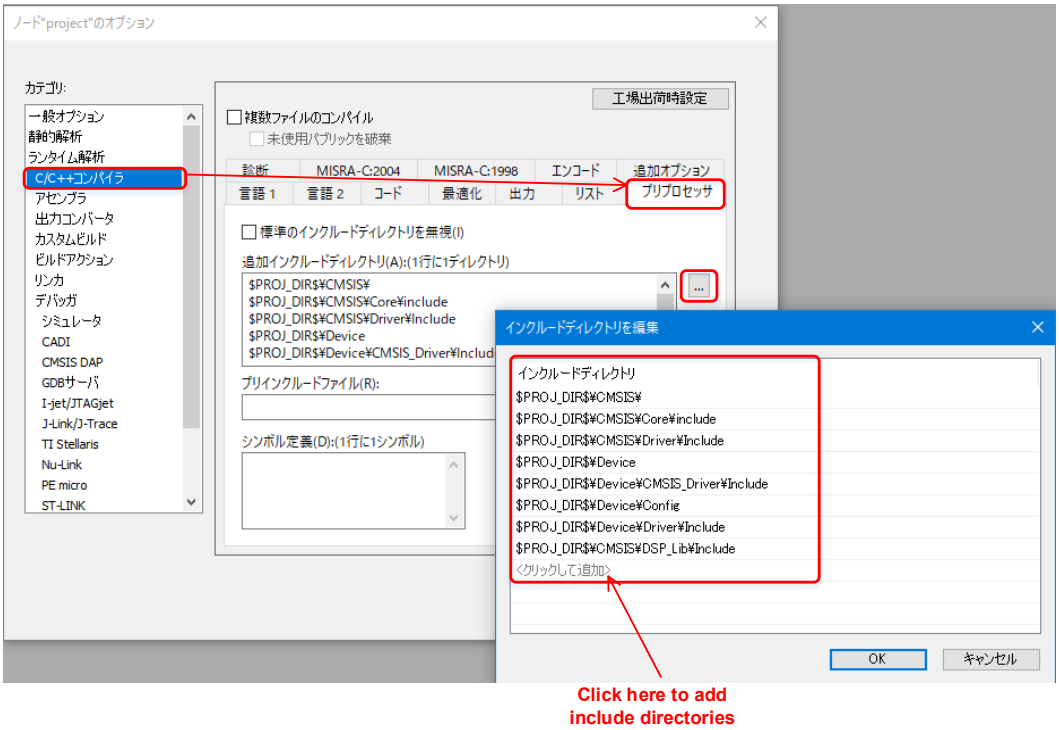


Figure 8-5 Example of Include Directory Setting (EWARM)

8.2 e² studio Version

The executable file stored in the e² studio install directory should be double-clicked to start e² studio. For the procedure from startup of e² studio up to import of this package, refer to 2.2 e² studio Version.

Project settings are set on the project Properties screen. Please refer to Figure 8-6 to display the Properties screen.

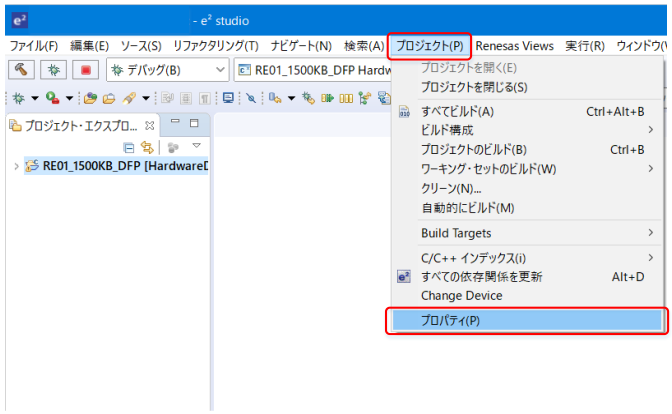


Figure 8-6 Displaying the Properties Screen (e² studio)

8.2.1 Setting Toolchain

In this package, the RE series is set as the toolchain. When the user sets the toolchain, it should be selected referring to Figure 8-7.

If "RE" is not displayed as the toolchain, the toolchain must be installed and registered. For how to use e² studio, refer to the Development Environment User's Manual.

Current toolchain: RE

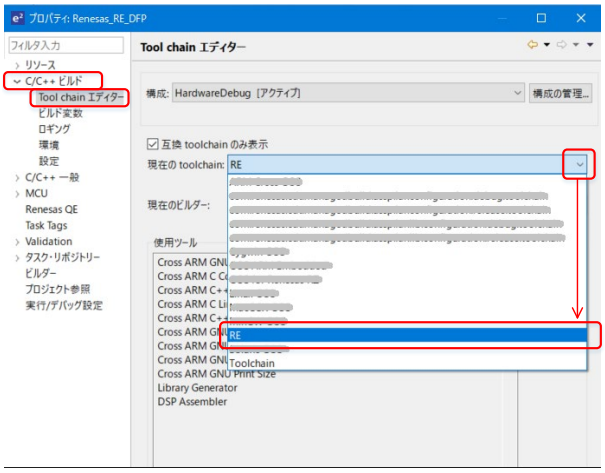


Figure 8-7 Toolchain Setting Example (e² studio)

8.2.2 Linker File Settings

In this package, the linker file "RE01_1500KB.ld", with RAM placement section definitions and MTB regions added, is set. When the user sets a linker file, the linker file to be used should be set referring to Figure 8-8.

Script files: "${workspace_loc:/${ProjName}/script/RE01_1500KB.ld}"

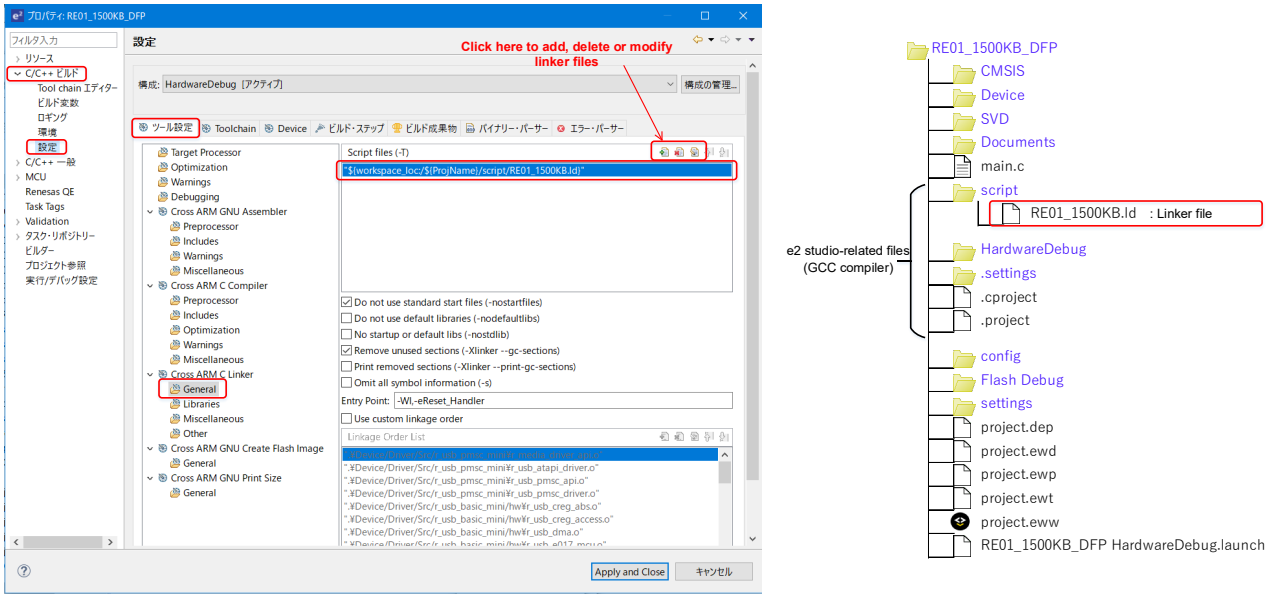


Figure 8-8 Linker File Setting Example (e^2 studio)

For RAM placement, see section 6.6.1, RAM Placement Method Using RAM Placement Section.

For MTB region, see section 9.1, Securing MTB Region for Debugger.

8.2.3 Include Path Settings

Drivers in this package include header files using only filenames, and so the locations of include files must be specified at compile time. In this package, the include path necessary for using drivers is set.

When the user sets the include path, the necessary path should be set referring to Figure 8-9.

Include path setting examples

- "${workspace_loc:/${ProjName}/Device}"
- "${workspace_loc:/${ProjName}/Device/Driver/Include}"
- "${workspace_loc:/${ProjName}/Device/CMSIS_Driver/Include}"
- "${workspace_loc:/${ProjName}/Device/Config}"
- "${workspace_loc:/${ProjName}/CMSIS/Core/Include}"
- "${workspace_loc:/${ProjName}/CMSIS/Driver/Include}"
- "${workspace_loc:/${ProjName}/CMSIS/DSP_Lib/Include}"
- ${ProjDirPath}/generate
- ${ProjDirPath}/src

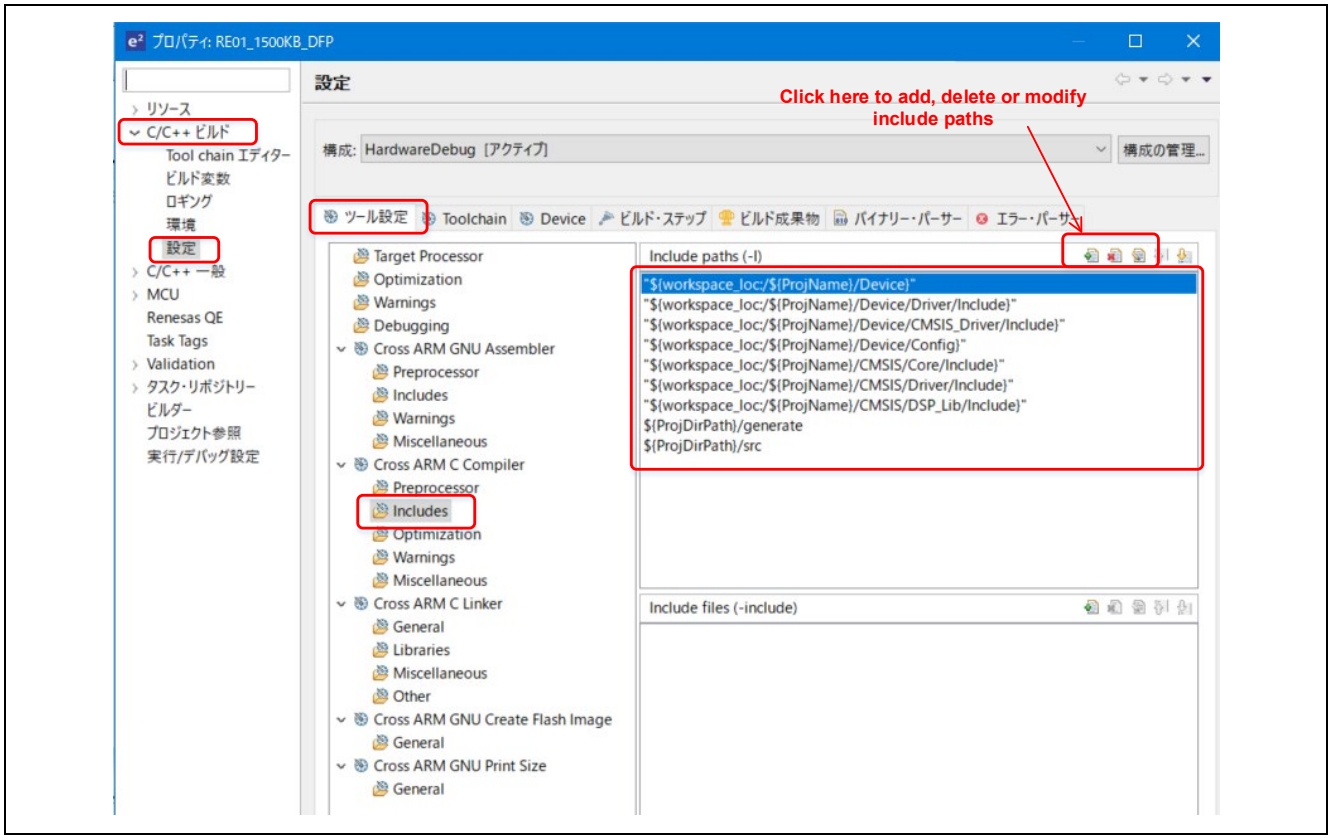


Figure 8-9 Include Path Setting Example (e^2 studio)

9. Using Debugger for Downloading

9.1 Securing MTB Region for Debugger

In this device, the leading 1 KB of RAM must be kept vacant as an MTB region for the debugger. In the linker file used by this package, the leading 1 KB of RAM is kept vacant for MTB. RAM memory mapping in this package is shown in Figure 9-1.

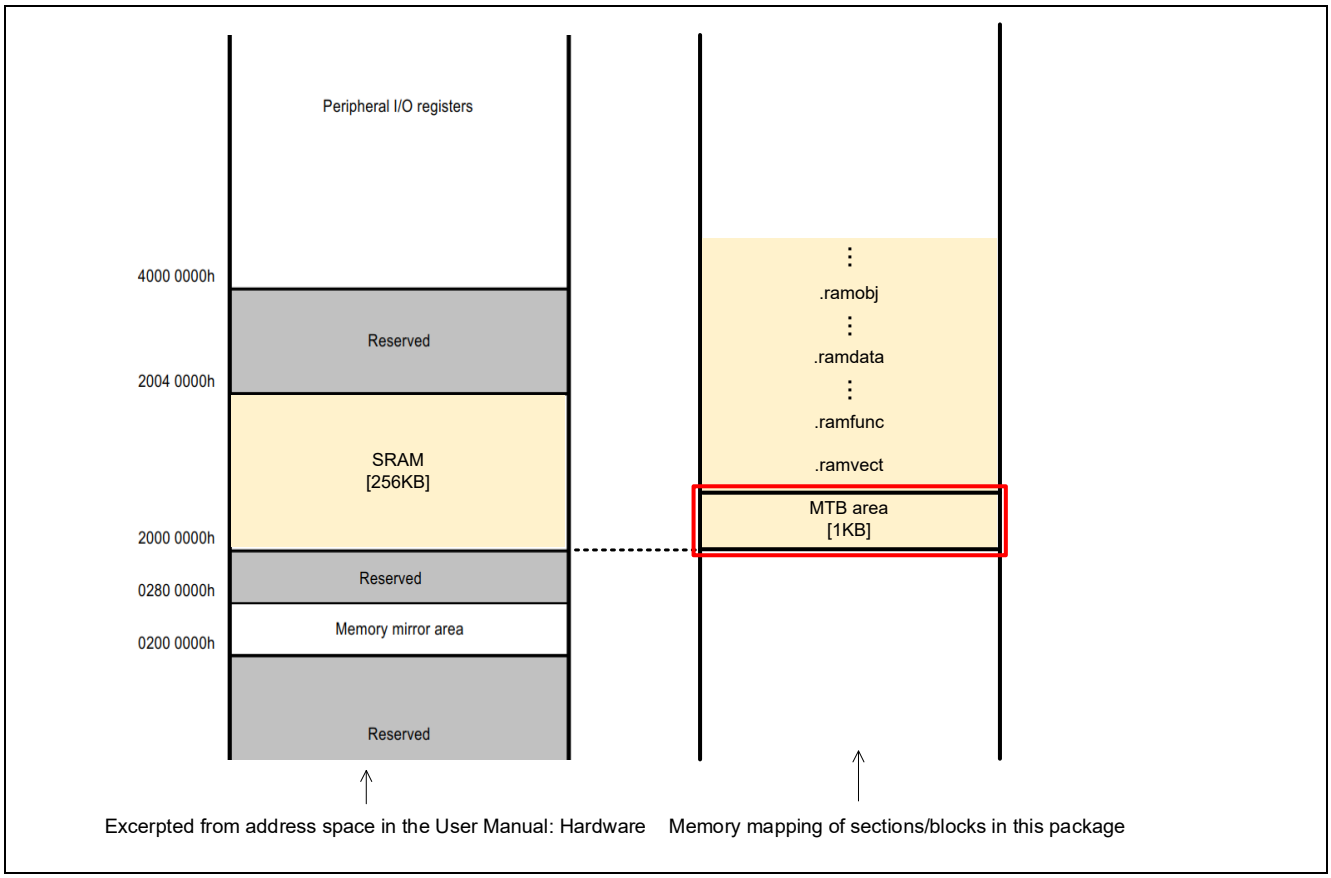


Figure 9-1 Example of Securing MTB Region

9.2 Usage Notes on Low Power Consumption Mode

9.2.1 Conditions in which Software Breaks Cannot be Set

This device has low power consumption modes such as deep software standby mode and software standby mode. During a low power consumption mode, the on-chip debugger (OCD) cannot read out values from the CPU registers or the like, and the OCD is disconnected.

Moreover, this device has modes (conditions) in which internal flash memory cannot be overwritten.

A software break should not be set in an internal flash memory region for debugging an application corresponding to any of these conditions.

Conditions under which a software break cannot be set in internal flash memory are the following.

- A device power control mode other than High-Speed mode
- When the system clock is lower than 1 MHz

For debugger connection while in a low power consumption mode, and for the method for setting breakpoints, see the explanations in sections 9.3 and 9.4.

9.3 EWARM Version

9.3.1 J-Link

This section describes use of the J-Link OB mounted on the target board.

9.3.1.1 Debugger selection

In this package, “J-Link” is set as the debugger. When the user sets a debugger for use, Figure 9-2 should be used for reference. Items enclosed in red lines in Figure 9-2 show ones to be checked when J-Link has been selected.

Driver: J-Link/J-Trac

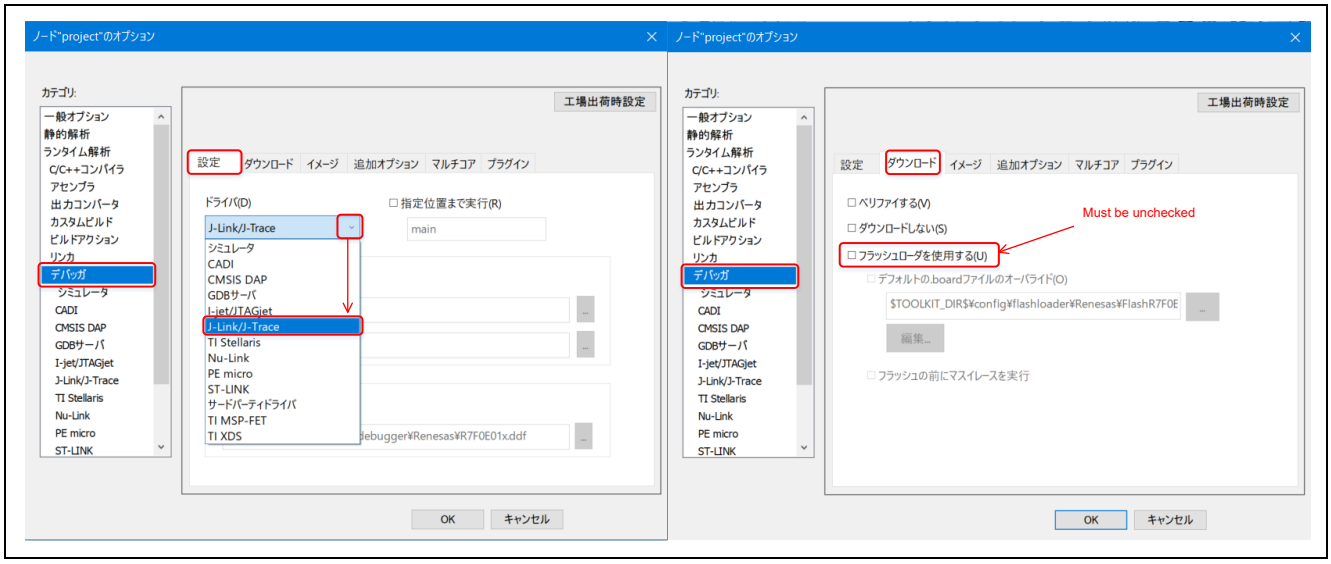


Figure 9-2 Example of J-Link Selection as Debugger (EWARM)

9.3.1.2 J-Link settings

In this package, initial settings when "J-Link" is selected as the debugger are made. When the user sets a debugger for use, Figure 9-3 should be used for reference. Items enclosed in red in Figure 9-3 show ones to be checked when J-Link is selected.

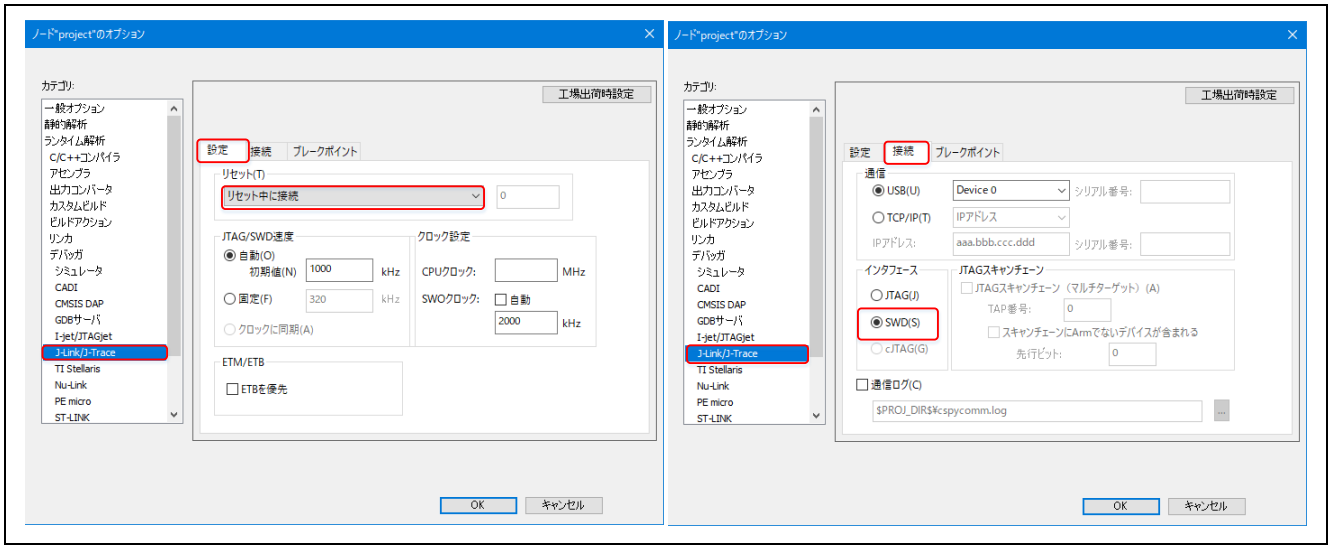


Figure 9-3 J-Link Setting Example (EWARM)

In this package, settings are not made for cases in which a program that uses a low power consumption mode is debugged. Here an example of settings when debugging a program that uses a low power consumption mode is shown.

- Use EWARM to open the project, and edit the file [project name]_Flash Debug.jlink that is generated upon first starting up J-Link OB, referring to Figure 9-4.

File to modify: [project name]_Flash Debug.jlink

After modification: LowPowerHandlingMode = 1

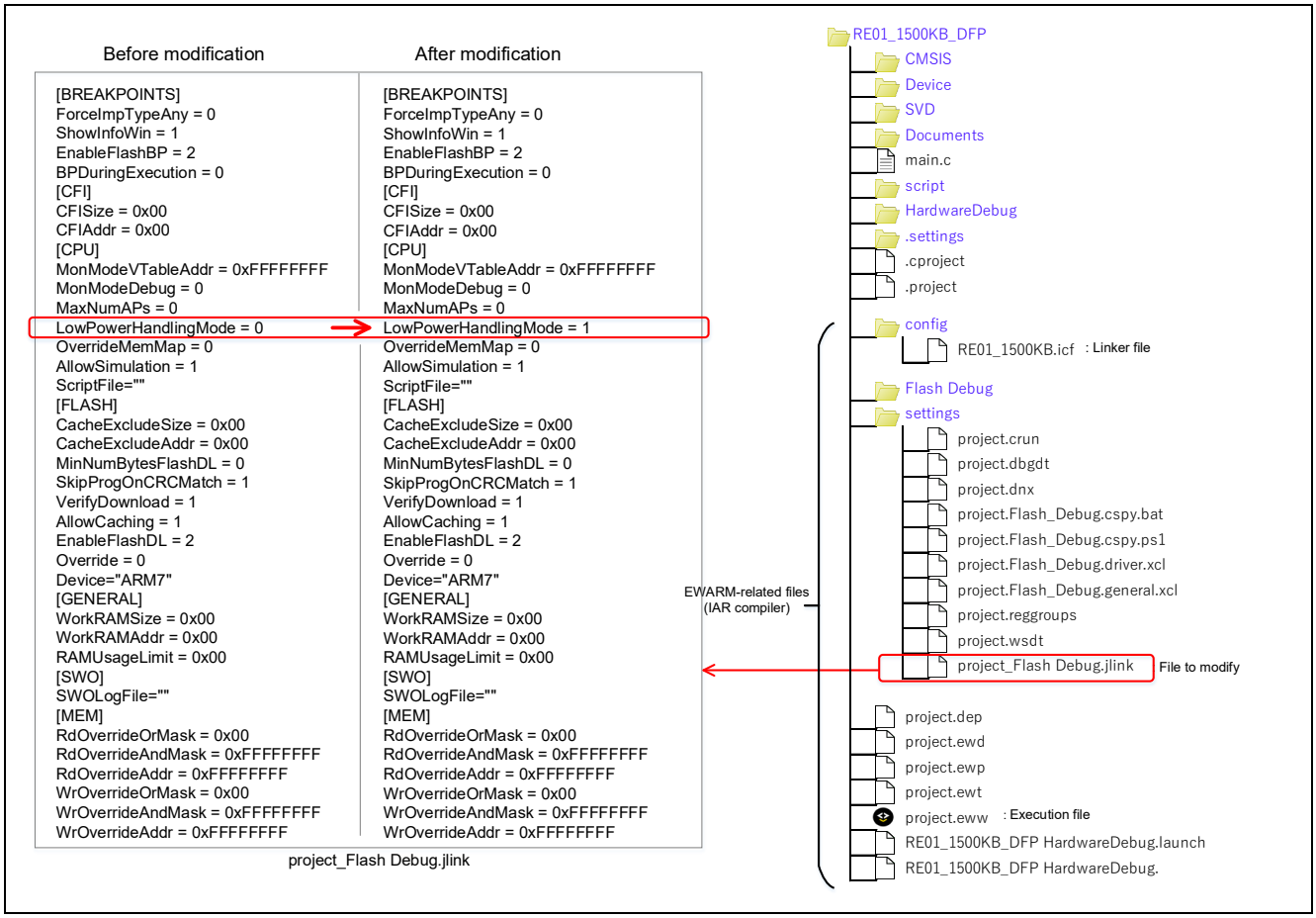


Figure 9-4 Example of Debugging Settings for Program Using Low Power Consumption Mode (EWARM)

9.3.1.3 Prohibition of the setting of software breaks in internal flash memory regions

In this package, prohibition of the setting of software breaks in an internal flash memory region is not set. The following is an example of settings by which the user prohibits setting of software breaks in an internal flash memory region.

- Open the project using EWARM, edit the file [project name]_Flash Debug.jlink that is generated upon initially starting J-Link OB referring to Figure 9-5, and newly create a script file with a desired filename.

File to modify: [project name]_Flash Debug.jlink

After modification: ScriptFile="project.JLinkScript" (according to the path and name of the newly created file)

Newly created file: project.JLinkScript (using any desired filename)

Contents of newly created file:

```
int ConfigTargetSettings(void) {
    JLINK_ExecCommand("SetWorkRAM 0x20008000-0x20017FFF");
    JLINK_ExecCommand("DisableFlashBPs");
    return 0;
}
```

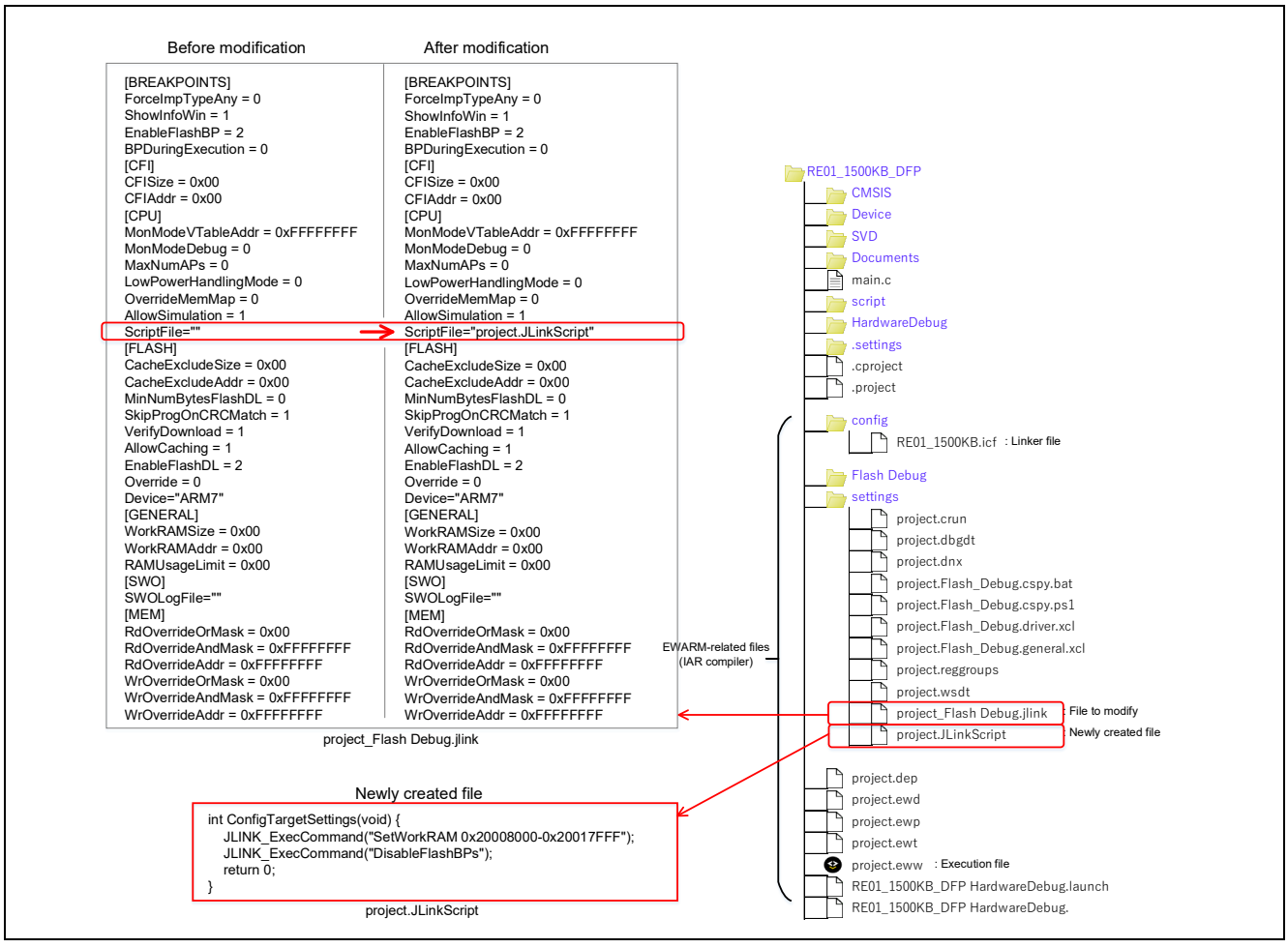


Figure 9-5 Example of Prohibition of Setting Software Breakpoints in Internal Flash Memory Regions (EWARM)

9.3.2 I-Jet

9.3.2.1 Debugger Selection

In this package, “J-Link” is set as the debugger. When using I-Jet as the debugger, select it referring to Figure 9-6. Items enclosed in red lines in Figure 9-6 show ones to be checked when I-Jet has been selected.

Driver: I-jet/JTAGjet

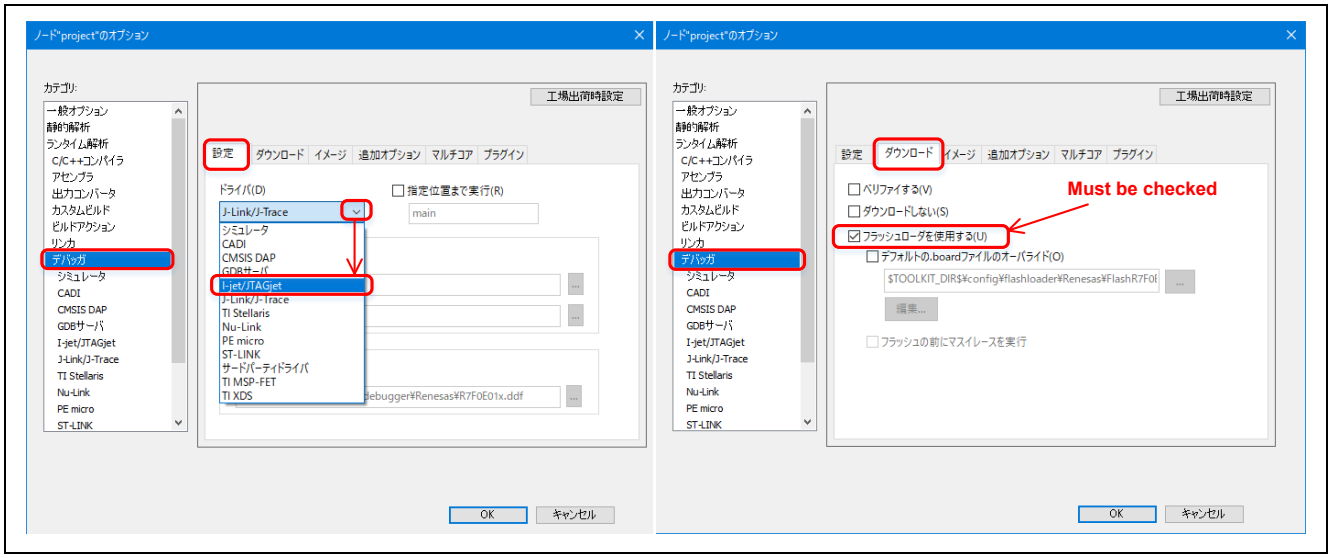


Figure 9-6 Example of I-Jet Selection as Debugger (EWARM)

9.3.2.2 I-Jet settings

When using I-Jet as the debugger, settings should be made referring to Figure 9-7. Items enclosed in red lines in Figure 9-7 show ones to be checked when I-Jet has been selected.

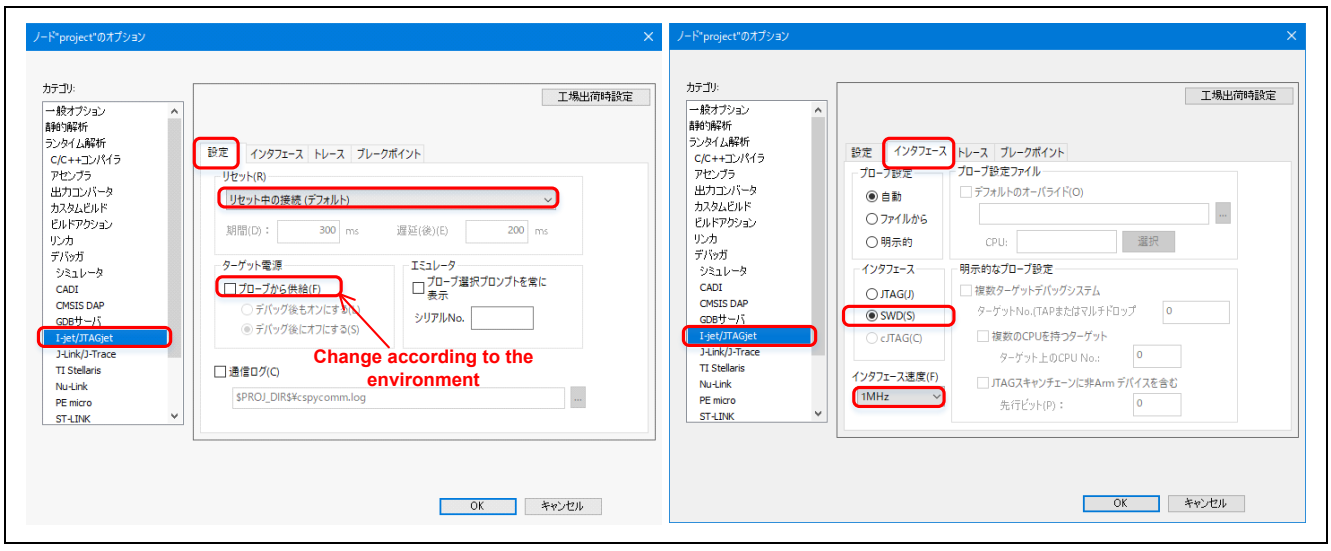


Figure 9-7 Example of I-Jet Settings (EWARM)

9.4 e² studio Version

9.4.1 J-Link

This section describes use of J-Link OB mounted on the target board.

9.4.1.1 Debugger selection

In this package, “J-Link” is set as the debugger. When the user sets a debugger for use, Figure 9-8 should be used for reference. Items enclosed in red lines in Figure 9-8 show ones to be checked when J-Link has been selected.

Debug hardware: J-Link ARM

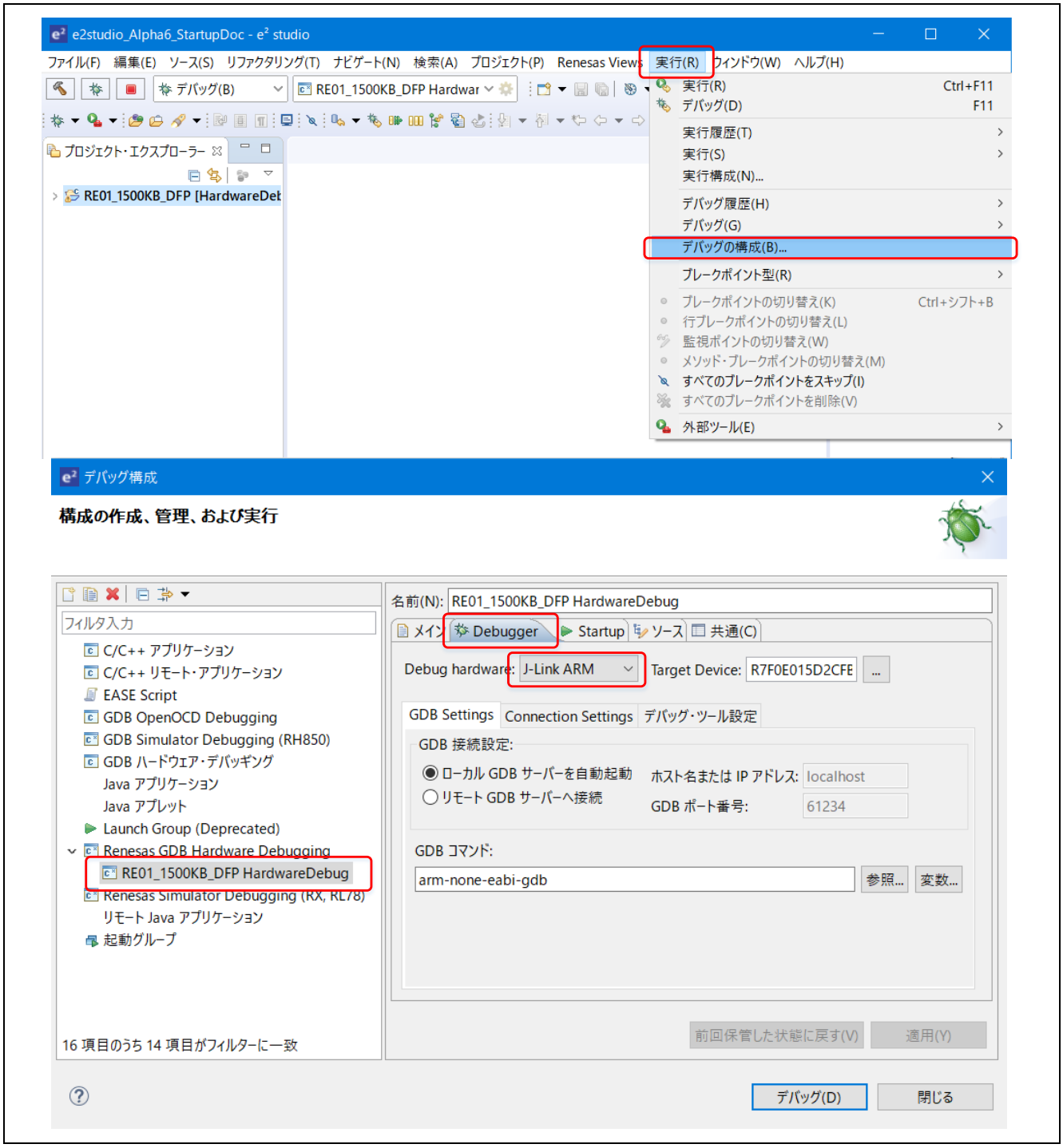


Figure 9-8 Example of Selection of J-Link as Debugger (e² studio)

9.4.1.2 J-Link settings

In this package, initial settings when "J-Link" is selected as the debugger are made. When the user sets a debugger for use, Figure 9-9 should be used for reference. Items enclosed in red lines in Figure 9-9 show ones to be checked when J-Link has been selected.

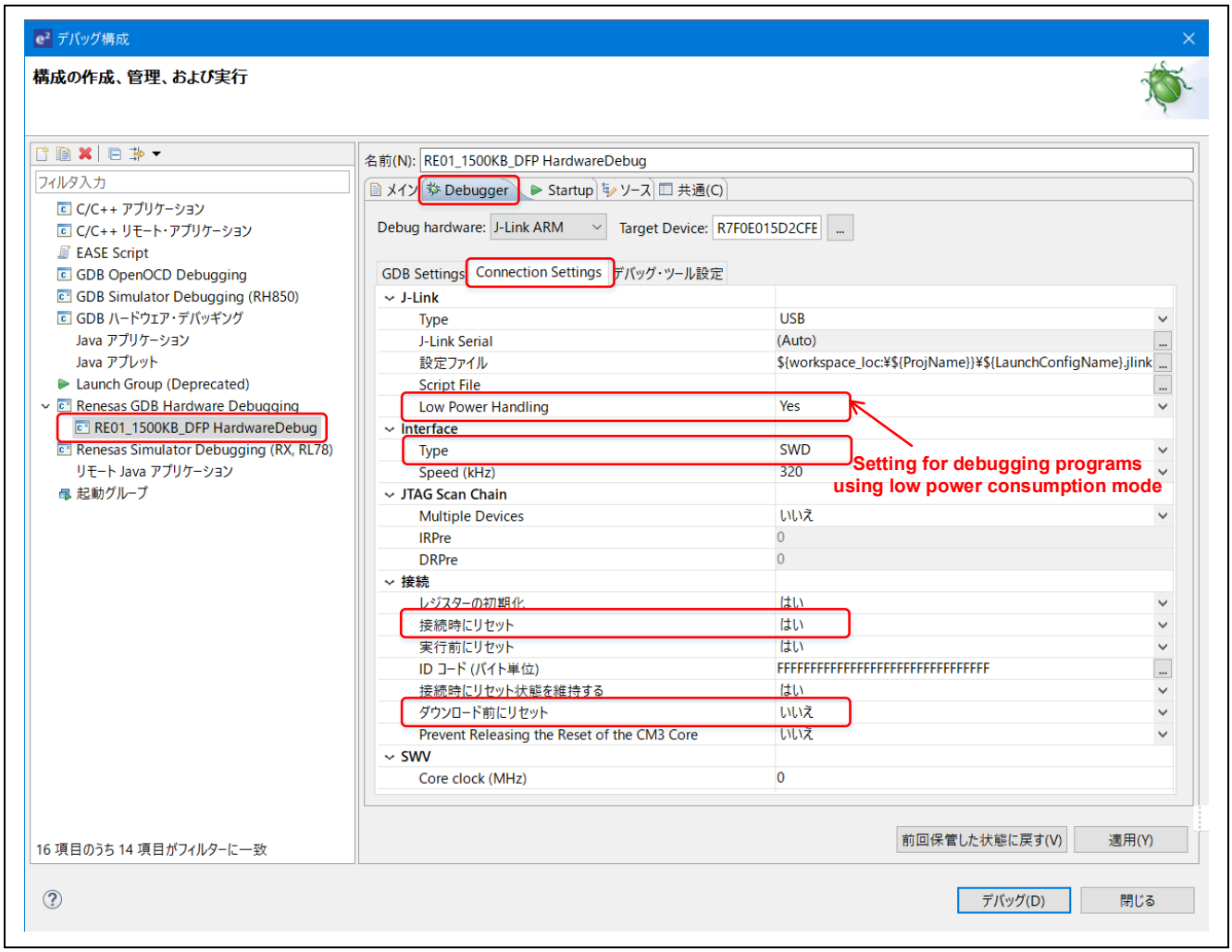


Figure 9-9 J-Link Settings Example (e² studio)

9.4.1.3 Prohibition of Setting of Software Breaks in Internal Flash Memory Regions

In this package, when "J-Link" has been selected as the debugger, prohibition of setting of software breaks in internal flash memory regions is set. When the user sets the prohibition, the setting should be made referring to Figure 9-10. Items enclosed in red lines in Figure 9-10 show ones to be checked when prohibition of setting of software breaks in internal flash memory regions has been set.

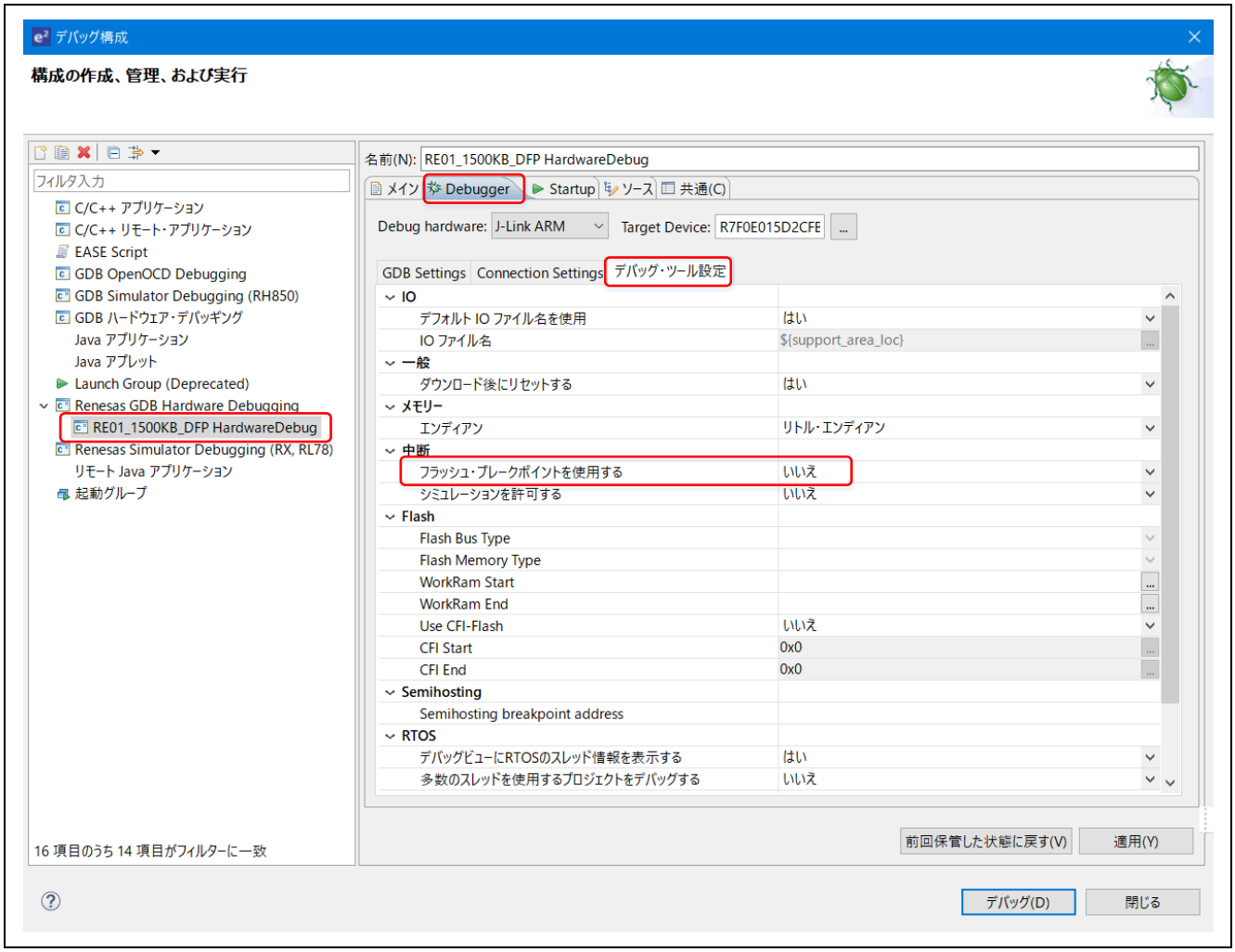


Figure 9-10 Example of Prohibition of Setting of Software Breaks in Internal Flash Memory Regions

9.4.1.4 Setting Breakpoints in RAM Regions

Hardware breakpoints cannot be set in RAM regions. An example of use that switches between hardware breakpoints and software breakpoints is explained.

1. Setting the initial breakpoint settings to hardware breakpoints
   Under specific conditions, software breakpoints cannot be set in internal flash memory, and so normally hardware breakpoints are set.
   The initial settings should be set to hardware breakpoints, referring to Figure 9-11.

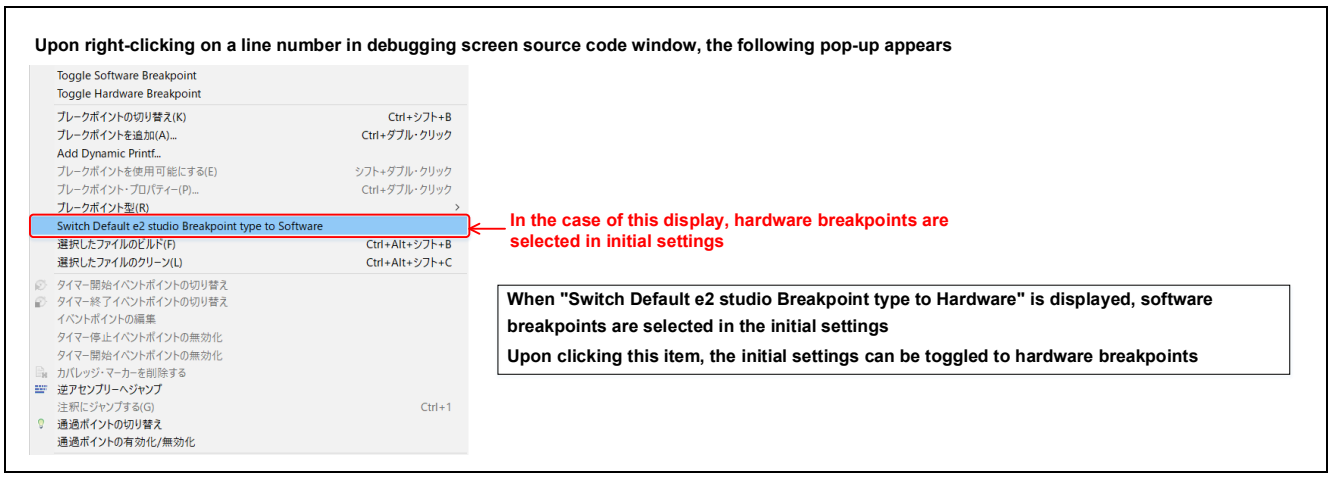


Figure 9-11 Example of Confirmation of Initial Breakpoint Settings (e^2 studio)

2. Setting software breakpoints in RAM regions
   Because hardware breakpoints cannot be set in RAM regions, a setting is made to switch to software breakpoints in RAM regions.

A setting should be made to switch the breakpoint type, referring to Figure 9-12.

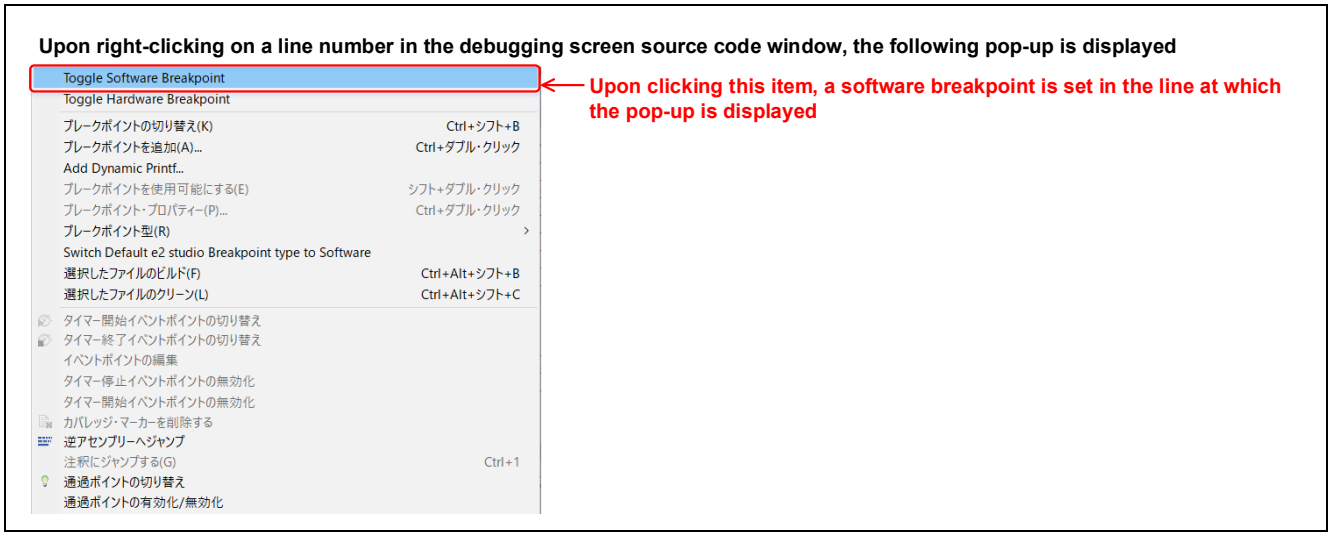


Figure 9-12 Example of Software Breakpoint Setting (e^2 studio)

10. Appendix

Module/Function	Corresponding driver	Peripheral function name	Interrupt name	Master		Slave	
				Transmit	Transmit/Receive	Transmit	Transmit/Receive
Serial peripheral interface	R_SPI	SPI	SPIn_SPTI	✓	✓	✓	✓
			SPIn_SPRI		✓		✓
			SPIn_SPII	✓	✓		
			SPIn_SPEI		✓	✓	✓
			SPIn_SPTEND	✓		✓	

Module/Function	Corresponding driver	Peripheral function name	Interrupt name
I2C bus interface	R_I2C	RIIC	IICn_RXI
			IICn_TXI
			IICn_TEI
			IICn_EEI

Module/Function	Corresponding driver	Peripheral function name	Interrupt name	DTC/DMAC	DMAC used	DTC used
I14-bit A/D converter	R_ADC	S14AD	ADC140_ADI	✓		✓
			ADC140_GBADI	✓		✓
			ADC140_GCADI	✓		✓
			ADC140_CMPAI	✓		
			ADC140_CMPBI	✓		
			ADC140_WCMPM	✓		✓
			ADC140_WCMPUM	✓		✓
			DMACn_INT		✓	

Module/Function	Corresponding driver	Peripheral function name	Interrupt name
DMA controller	R_DMAC	DMAC	DMACn_INT

Module/Function	Corresponding driver	Peripheral function name	Interrupt name
Data transfer controller	R_DTC	DTC	DTC_COMPLETE

Module/Function	Corresponding driver	Peripheral function name	Interrupt name
2D graphic data conversion circuit	R_GDT	GDT	GDT_DATII
			GDT_DATOI
			DMACn_INT (n = 0 to 3)

Interrupt names for peripheral functions

Figure 10-1 List of Interrupts Used by Drivers

11. Troubleshooting

11.1 Q: A build error occurs

11.1.1 A-1: Is an include directory set?

When including files, the compiler searches the location specified as the include directory. The location of include files should be specified as the include directory.

- When using EWARM, see section 8.1.3, Include Directory Settings.
- When using e^2 studio, see section 8.2.3, Include Path Settings.

11.1.2 A-2: Are the required files to be compiled set?

- When using EWARM

The files in the workspace window of EWARM are the files to be compiled. If files are missing, right-click anywhere in the EWARM workspace window and select files to be added. Select "Add Group" for folders, and "Add File" for files.

- When using e^2 studio

Files marked by diagonal lines in the e^2 studio project explorer are files not to be compiled. Right-click on a file and select "Resource Configurations" → "Exclude from Build" and uncheck the checkbox.

11.1.3 A-3: Is the compiler for the device selected?

At compile time, the compiler for the device should be selected.

- When using EWARM, see section 8.1.1, Setting Target Processor.
- When using e^2 studio, see section 8.2.1, Setting Toolchain.

11.2 Q: When a driver function is executed, a HardFault Error occurs

11.2.1 A-1: Is the program placed in a section for RAM placement expanded in RAM?

Driver functions in this package is used to expand a desired program in RAM using a RAM placement section. At the beginning of a user program, the R_SYS_CodeCopy function should be executed to expand the program in RAM.

For placing a program in RAM, see section 6.6, RAM Placement of Programs.

11.2.2 A-2: Is internal flash memory being accessed while internal flash memory is shut off?

When operation continues after internal flash memory has been shut off, the following causes are conceivable.

- When a program placed in internal flash memory, such as an NVIC function, standard function or the like, has been executed.
- When a program that should have been placed in RAM could not be placed in RAM as expected.
- When a function executed from a program that has been placed in internal flash memory is inline-expanded, and the program could not be placed in RAM as expected.

For placing a program in RAM, see section 6.6, RAM Placement of Programs.

11.3 Q: A driver function is being executed, but the peripheral function does not run

11.3.1 A-1: Is there a problem with the driver function settings?

It is possible that an error in the driver function is being detected.

Check the return value of the driver function to see if an error has occurred.

In particular, errors are often caused by problems related to insufficient interrupt settings.

For interrupt settings when the peripheral function driver is used, see section 6.3.4, Driver Functions.

11.4 Q: The return value of a driver function is normal, but the expected input or output from a peripheral function pin cannot be confirmed

11.4.1 A-1: Was the pin setting function of the R_PIN driver edited?

A peripheral function driver function executes an R_PIN driver function and sets the pin to be used by the peripheral function. The R_PIN driver function processing should be edited such that the desired pin is used by the peripheral function.

For pin settings, see section 6.5.2, Editing Driver Functions.

11.4.2 A-2: Is power being fed to the applicable I/O power supply domain?

This device has multiple I/O power supply domains, and control can be executed to supply or shut off power by domain. Power should be supplied to I/O power supply domains in which pins used by peripheral functions are disposed.

For applicable I/O power supply domains, see section 6.2.1, Applicable Power Supply Domains.

11.4.3 A-3: Is the undefined value propagation suppression function enabled for the I/O power supply domain?

When the expected input and output cannot be confirmed even though power is being supplied to the applicable I/O power supply domain, it is conceivable that the undefined value propagation suppression function is enabled. After reset cancellation, the undefined value propagation suppression function is enabled for all I/O power supply domains other than IOVCC, and so the undefined value propagation suppression function should be disabled for domains to which power is supplied.

For the driver function that controls the undefined value propagation suppression function for I/O power supply domains, see section 6.2.2, Driver Functions.

11.5 Q: Write to clock and power consumption reduction function related registers was performed, but does not take effect

11.5.1 A-1: Is register write protection enabled?

After reset cancellation, the register write protect function is enabled, so that write to registers cannot be performed. When performing wrie to clock and power consumption reduction function related registers, the register write protect function should be disabled.

For information on using the register write protect function, see section 7.2.2.5, Register write protect control.

11.6 Q: Write to a peripheral function related register was performed, but does not take effect

11.6.1 A: Is the module stop function enabled?

After reset cancellation, except for some functions, the module stop function is enabled. Before using a peripheral function, the module stop function should be disabled.

For information on using the module stop function, see section 7.2.2.4, Module stop control.

11.7 Q: The debugger cannot be used to download to the target board

11.7.1 A-1: Check the debugger settings

- When using J-Link with EWARM, see section 9.3.1, J-Link.
- When using I-Jet with EWARM, see section 9.3.2, I-Jet.
- When using J-Link with e^2 studio, see section 9.4.1, J-Link.

11.7.2 A-2: Is power being supplied correctly?

When using I-Jet, power supply to the target board can be selected from the debugger or from an external source. The debugger settings should be made according to the operating environment.

For information on I-Jet settings, see section 9.3.2.2, I-Jet settings.

11.8 Q: The debugger was connected, but does not run

11.8.1 A-1: Check the debugger settings

- When using J-Link with EWARM, see section 9.3.1, J-Link.
- When using I-Jet with EWARM, see section 9.3.2, I-Jet.
- When using J-Link with e^2 studio, see section 9.4.1, J-Link.

12. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

https://www.renesas.com/products/software-tools/software-os-middleware-driver/software-package/re-software-development-kit.html

13. Reference Documents

User's Manual: Hardware

RE01 Group 1500KB User's Manual: Hardware

RE01 Group 256KB User's Manual: Hardware

(The latest version can be downloaded from the Renesas Electronics website.)

https://www.renesas.com/products/microcontrollers-microprocessors/re.html

Technical Update/Technical News
(The latest information can be downloaded from the Renesas Electronics website.)

User's Manual: Development Tools
(The latest version can be downloaded from the Renesas Electronics website.)

J-Link/J-Trace User Guide: UM08001_JLink.pdf

<J-Link install folder>/Doc/Manuals/ UM08001_JLink.pdf

Website and Support

Renesas Electronics Website
http://japan.renesas.com

Inquiries
http://japan.renesas.com/contact/

Revision History

Rev.	Date	Description	
		Page	Summary
0.10	June 22, 2018	—	First edition issued
0.70	Jan. 28, 2019		Conformed to RE01 1.5MB WS2
0.71	Feb. 15, 2019		Supplemental description of interrupts in section 2.4.1 added
			Description in section 2.4 improved
1.00	July 1, 2019		Structure of all the chapters changed substantially: Chapter 1: Details on CMSIS PACK added Chapter 5: Method to include user code added
1.01	Sept. 25, 2019		Overall style optimized according to the Style Guide. Technical terms unified (including those in figures) Erroneous descriptions corrected (including those in figures) Product family and group names such as SOTB corrected (examples: RE family, RE01_1500KB) Folder Structure updated to the latest version 1.4.4: Correspondence between functions in pin.c and drivers added 6.3 (a): Linker file path setting in EWARM added to the RAM mapping method 7: Notes on J-link added 9.6: Troubleshooting items added End of this document: “Notice” replaced with the latest version Overall chapter structure changed “2. Running a Project” added
1.02	June.16,2020		Change the title to add “RE01 256KB product”.

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

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

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

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

Trademarks

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/.