

RX Family

Flash Module Using Firmware Integration Technology

Introduction

The Flash Module Using Firmware Integration Technology (FIT) has been developed to allow users of supported RX devices to easily integrate reprogramming abilities into their applications using self-programming. Self-programming is the feature to reprogram the on-chip flash memory while running in single-chip mode. This application note focuses on using the Flash FIT module and integrating it with your application program.

The Flash FIT module is different from "RX600 & RX200 Series Simple Flash API for RX (R01AN0544)".

Target Device

- RX110 Group
- RX111 Group
- RX113 Group
- RX130 Group
- RX13T Group
- RX230, RX231 Groups
- RX23E-A Group
- RX23T Group
- RX23W Group
- RX24T Group
- RX24U Group
- RX64M Group
- RX65N, RX651 Groups
- RX66N Group
- RX66T Group
- RX71M Group
- RX72M Group
- RX72N Group
- RX72T Group

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

Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

For details of the confirmed operation contents of each compiler, refer to "5.1 Confirmed Operation Environment".

Related Documents

- Firmware Integration Technology User's Manual (R01AN1833)
- Board Support Package Firmware Integration Technology Module (R01AN1685)

Contents

1. Overview.....	4
1.1 Features	4
2. API Information	5
2.1 Hardware Requirements	5
2.2 Software Requirements	5
2.3 Limitations	5
2.4 Supported Toolchains	5
2.5 Interrupt Vector	5
2.6 Header Files	6
2.7 Integer Types.....	6
2.8 Flash Types and Features.....	6
2.9 Configuration Overview	7
2.10 Code Size	9
2.11 Parameters	14
2.12 Return Values.....	14
2.13 Blocking Mode and Non-blocking Mode.....	15
2.13.1 Using in Blocking Mode.....	15
2.13.2 Using in Non-blocking Mode	15
2.14 Adding the FIT Flash Module to Your Project	16
2.15 Usage Combined with Existing User Projects.....	16
2.16 Programming Code Flash from RAM	16
2.17 Programming Code Flash from Code Flash.....	17
2.18 Dual Bank Operation	17
2.19 Usage Notes.....	18
2.19.1 Data Flash Operations in Non-blocking Mode	18
2.19.2 Code Flash Operations in Non-blocking Mode	18
2.19.3 Code Flash Operations and General Interrupts	18
2.19.4 Emulator Debug Configuration	19
3. API Functions	20
3.1 Summary	20
3.2 R_FLASH_Open()	21
3.3 R_FLASH_Close().....	22
3.4 R_FLASH_Erase()	23
3.5 R_FLASH_BlankCheck().....	25
3.6 R_FLASH_Write()	27
3.7 R_FLASH_Control()	29
3.8 R_FLASH_GetVersion().....	37
4. Demo Projects	38

4.1	flash_demo_rskrx113	38
4.2	flash_demo_rskrx231	38
4.3	flash_demo_rskrx23t	39
4.4	flash_demo_rskrx130	39
4.5	flash_demo_rskrx24t	39
4.6	flash_demo_rskrx65n	40
4.7	flash_demo_rskrx24u	40
4.8	flash_demo_rskrx65n2mb_bank0_bootapp / _bank1_otherapp	40
4.9	flash_demo_rskrx64m	41
4.10	flash_demo_rskrx64m_runrom	41
4.11	flash_demo_rskrx66t	41
4.12	flash_demo_rskrx72t	42
4.13	flash_demo_rskrx72m_bank0_bootapp / _bank1_otherapp	42
4.14	Adding a Demo to a Workspace	42
4.15	Downloading Demo Projects	42
5.	Appendices	43
5.1	Confirmed Operation Environment	43
5.2	Troubleshooting	46
5.3	Compiler-Dependent Settings	48
5.3.1	Using Renesas Electronics C/C++ Compiler Package for RX Family	48
5.3.1.1	Programming Code Flash from RAM	49
5.3.1.2	Programming Code Flash Using the Dual Bank Function	51
5.3.2	Using GCC for Renesas RX	54
5.3.2.1	Programming Code Flash from RAM	54
5.3.2.2	Programming Code Flash Using the Dual Bank Function	56
5.3.3	Using IAR C/C++ Compiler for Renesas RX	59
5.3.3.1	Programming Code Flash from RAM	59
5.3.3.2	Programming Code Flash Using the Dual Bank Function	64
6.	Reference Documents	66
	Revision History	67

1. Overview

The Flash FIT module is provided to customers to make the process of programming and erasing on-chip flash areas easier. Both code flash and data flash areas are supported. The module can be used to perform program, erase, and blank check operations in blocking or non-blocking mode. When a program, erase or blank check function is called in blocking mode, these API functions do not return until the operation has finished. In non-blocking mode, the API functions return immediately after the operation has begun. When a code flash operation is on-going, that code flash area cannot be accessed by the user. If an attempt is made to access the code flash area, the sequencer will transition into an error state. In non-blocking mode, whether operating on code flash or data flash, the user must poll for operation completion or provide a flash interrupt callback.

1.1 Features

Below is a list of the features supported by the Flash FIT module.

- Erasing, programming, and blank checking for code flash and data flash in blocking mode or non-blocking mode.
- Area protection via access windows or lockbits.
- Start-up program protection; this function is used to safely rewrite block 0 to block 7 in code flash

2. API Information

This FIT module has been confirmed to operate under the following conditions.

2.1 Hardware Requirements

This driver requires that your MCU supports the following peripheral(s):

- Flash

2.2 Software Requirements

This driver is dependent upon the following FIT packages:

- Renesas Board Support Package (r_bsp) v5.00 or later

2.3 Limitations

- This code protects against multiple concurrent API function calls (not including some R_FLASH_Control() commands).
- During code flash reprogramming, code flash cannot be accessed. When reprogramming code flash, make sure application code runs from RAM.

- RAM Location Limitations

In FIT, if a value equivalent to NULL is set as the pointer argument of an API function, error might be returned due to parameter check. Therefore, do not pass a NULL equivalent value as pointer argument to an API function.

The NULL value is defined as 0 because of the library function specifications. Therefore, the above phenomenon would occur when the variable or function passed to the API function pointer argument is located at the start address of RAM (address 0x0). In this case, change the section settings or prepare a dummy variable at the top of the RAM so that the variable or function passed to the API function pointer argument is not located at address 0x0.

In the case of CCRX project (e2 studio V7.5.0), the RAM start address is set as 0x4 to prevent the variable from being located at address 0x0. In the case of GCC project (e2 studio V7.5.0) and IAR project (EWRX V4.12.1), the start address of RAM is 0x0, so the above measures are necessary.

The default settings of the section may be changed due to IDE version upgrade. Please check the section settings when using the latest IDE.

2.4 Supported Toolchains

This driver has been confirmed to work with the toolchain listed in 5.1 Confirmed Operation Environment.

2.5 Interrupt Vector

When the FLASH_CFG_DATA_FLASH_BGO or FLASH_CFG_CODE_FLASH_BGO configuration option (see section 2.9) is 1, the interrupts shown in Table 2.1 below are enabled.

Table 2.1 Interrupt Vector Used in the Flash FIT Module

Device	Interrupt Vector
RX110, RX111, RX113, RX130, RX13T RX230, RX231, RX23E-A, RX23T, RX23W, RX24T, RX24U	FRDYI interrupt (vector no.: 23)
RX64M, RX66T, RX71M, RX72T RX651, RX65N, RX66N, RX72M, RX72N	FIFERR interrupt (vector no.:21) FRDYI interrupt (vector no.: 23)

2.6 Header Files

All API calls and their supporting interface definitions are located in “r_flash_rx_if.h”. This file should be included by all files which utilize the Flash FIT.

The configuration options that can be set at build time are defined in the “r_flash_rx_config.h” file.

2.7 Integer Types

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

2.8 Flash Types and Features

The flash driver is divided into three separate types based upon the technology and sequencer used. The compiled flash driver size is based upon the flash type (see section 2.10).

FLASH TYPE 1

RX110*, RX111, RX113, RX130, RX13T
RX230, RX231, RX23E-A, RX23T*, RX23W, RX24T, RX24U
**has no data flash*

FLASH TYPE 3

RX64M, RX66T, RX71M, RX72T

FLASH TYPE 4

RX651*, RX65N*, RX66N, RX72M, RX72N
**no data flash on parts with less than or equal to 1M code flash*

Because of the different flash types, not all flash commands or features are available on all MCUs. The file r_flash_rx_if.h identifies which features are available on each MCU using #defines.

2.9 Configuration Overview

Configuring this module is done through the supplied `r_flash_rx_config.h` header file. Each configuration item is represented by a macro definition in this file. Each configurable item is detailed in the table below.

Table 2 Flash general configuration settings

<i>Configuration options in <code>r_flash_rx_config.h</code></i>		
Equate	Default Value	Description
FLASH_CFG_PARAM_CHECKING_ENABLE	1	Setting to 1 includes parameter checking. Setting to 0 omits parameter checking.
FLASH_CFG_CODE_FLASH_ENABLE	0	If you are only using data flash, set this to 0. Setting to 1 includes code to program the code flash area. When programming code flash, code must be executed from RAM. See section 2.16 for details on how to set up code and the linker to execute code from RAM. Codes can be executed from code flash in Flash Types 3 and 4 with limitations.
FLASH_CFG_DATA_FLASH_BGO	0	Setting this to 0 forces data flash API function to block until completed. Setting to 1 places the module in non-blocking mode. In non-blocking mode, data flash operations return immediately after the operation has been started. Notification of the operation completion is done via the callback function. When FLASH_CFG_CODE_FLASH_ENABLE is set to 1, make the same setting as FLASH_CFG_CODE_FLASH_BGO.
FLASH_CFG_CODE_FLASH_BGO	0	Setting this to 0 forces code flash API function to block until completed. Setting to 1 places the module in non-blocking mode. In non-blocking mode, code flash operations return immediately after the operation has been started. Notification of the operation completion is done via the callback function. When reprogramming code flash, the relocatable vector table and corresponding interrupt routines must be relocated to an area other than code flash in advance. See sections 2.19 Usage Notes. When FLASH_CFG_CODE_FLASH_ENABLE is set to 1, make the same setting as FLASH_CFG_CODE_FLASH_BGO.

Configuration options in <i>r_flash_rx_config.h</i>		
Equate	Default Value	Description
FLASH_CFG_CODE_FLASH_RUN_FROM_ROM	0	<p>For FLASH_TYPE_3, and FLASH_TYPE_4 with more than or equal to 1.5M code flash. Valid only when FLASH_CFG_CODE_FLASH_ENABLE is set to 1.</p> <p>Set this to 0 when programming code flash while executing in RAM.</p> <p>Set this to 1 when programming code flash while executing from another segment in code flash (see section 2.17).</p>

2.10 Code Size

The ROM size, RAM size, and the maximum stack size of this module are described in the following table. One device is listed at a time as the representative of each flash type.

The code size is based on optimization level 2 and optimization type for size for the RXC toolchain in Section 2.4. The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options set in the module configuration header file.

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

Module Revision:	r_flash_rx Rev.4.30
Compiler Version:	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 (The option of “-lang = c99” is added to the default settings of the integrated development environment.) GCC for Renesas RX 4.08.04.201902 (The option of “-std = gnu99” is added to the default settings of the integrated development environment.) IAR C/C++ Compiler for Renesas RX version 4.12.1 (The default settings of the integrated development environment.)
Configuration Options:	The setting of configuration options that are different is described in each table. Other configuration options are default settings.

Flash Type 1: ROM, RAM and Stack Code Sizes (Maximum Size)							
Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX130	ROM	3492 bytes	3167 bytes	7340 bytes	6776 bytes	5222 bytes	4800 bytes
	RAM	2843 bytes		5692 bytes		4187 bytes	
	STACK	112 bytes		-		100 bytes	
Configuration options: FLASH_CFG_PARAM_CHECKING_ENABLE 0: Without parameter check, 1: With parameter check FLASH_CFG_CODE_FLASH_ENABLE 1 FLASH_CFG_DATA_FLASH_BGO 1 FLASH_CFG_CODE_FLASH_BGO 1							

Flash Type 1: ROM, RAM and Stack Code Sizes (Minimum Size)							
Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX130	ROM	1803 bytes	1688 bytes	3792 bytes	3624 bytes	2496 bytes	2360 bytes
	RAM	73 bytes		76 bytes		53 bytes	
	STACK	44 bytes		-		48 bytes	
Configuration options: FLASH_CFG_PARAM_CHECKING_ENABLE 0: Without parameter check, 1: With parameter check FLASH_CFG_CODE_FLASH_ENABLE 0 FLASH_CFG_DATA_FLASH_BGO 0 FLASH_CFG_CODE_FLASH_BGO 0							

Flash Type 1: ROM, RAM and Stack Code Sizes (Maximum Size)							
Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX23T*	ROM	2810 bytes	2545 bytes	5564 bytes	5068 bytes	4000 bytes	3656 bytes
	RAM	2566 bytes		5160 bytes		3733 bytes	
	STACK	108 bytes		-		100 bytes	
Configuration options: FLASH_CFG_PARAM_CHECKING_ENABLE 0: Without parameter check, 1: With parameter check FLASH_CFG_CODE_FLASH_ENABLE 1 FLASH_CFG_DATA_FLASH_BGO 1 FLASH_CFG_CODE_FLASH_BGO 1							

*Device without data flash

Flash Type 1: ROM, RAM and Stack Code Sizes (Minimum Size)							
Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX23T*	ROM	2561 bytes	2312 bytes	5076 bytes	4604 bytes	3544 bytes	3208 bytes
	RAM	2319 bytes		4672 bytes		3271 bytes	
	STACK	48 bytes		-		48 bytes	
Configuration options: FLASH_CFG_PARAM_CHECKING_ENABLE 0: Without parameter check, 1: With parameter check FLASH_CFG_CODE_FLASH_ENABLE 1 FLASH_CFG_DATA_FLASH_BGO 0 FLASH_CFG_CODE_FLASH_BGO 0							

*Device without data flash

Flash Type 3: ROM, RAM and Stack Code Sizes (Maximum Size)							
Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX64M	ROM	3536 bytes	3102 bytes	7228 bytes	6476 bytes	5448 bytes	4920 bytes
	RAM	3091 bytes		6416 bytes		4827 bytes	
	STACK	224 bytes		-		180 bytes	
Configuration options: FLASH_CFG_PARAM_CHECKING_ENABLE 0: Without parameter check, 1: With parameter check FLASH_CFG_CODE_FLASH_ENABLE 1 FLASH_CFG_DATA_FLASH_BGO 1 FLASH_CFG_CODE_FLASH_BGO 1							

Flash Type 3: ROM, RAM and Stack Code Sizes (Minimum Size)							
Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX64M	ROM	2110 bytes	1967 bytes	4444 bytes	4192 bytes	3068 bytes	2898 bytes
	RAM	65 bytes		68 bytes		48 bytes	
	STACK	76 bytes		-		56 bytes	
Configuration options: FLASH_CFG_PARAM_CHECKING_ENABLE 0: Without parameter check, 1: With parameter check FLASH_CFG_CODE_FLASH_ENABLE 0 FLASH_CFG_DATA_FLASH_BGO 0 FLASH_CFG_CODE_FLASH_BGO 0							

Flash Type 4: ROM, RAM and Stack Code Sizes (Maximum Size)							
Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX65N	ROM	3524 bytes	3077 bytes	7068 bytes	6284 bytes	5284 bytes	4736 bytes
	RAM	3177 bytes		5960 bytes		4542 bytes	
	STACK	208 bytes		-		176 bytes	
Configuration options: FLASH_CFG_PARAM_CHECKING_ENABLE 0: Without parameter check, 1: With parameter check FLASH_CFG_CODE_FLASH_ENABLE 1 FLASH_CFG_DATA_FLASH_BGO 1 FLASH_CFG_CODE_FLASH_BGO 1							

Flash Type 4: ROM, RAM and Stack Code Sizes (Minimum Size)							
Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX65N	ROM	1941 bytes	1798 bytes	3968 bytes	3752 bytes	2827 bytes	2657 bytes
	RAM	61 bytes		92 bytes		47 bytes	
	STACK	72 bytes		-		52 bytes	
Configuration options: FLASH_CFG_PARAM_CHECKING_ENABLE 0: Without parameter check, 1: With parameter check FLASH_CFG_CODE_FLASH_ENABLE 0 FLASH_CFG_DATA_FLASH_BGO 0 FLASH_CFG_CODE_FLASH_BGO 0							

2.11 Parameters

This section describes the structures used as arguments to API functions. These structures are included in the file `r_flash_rx_if.h` along with the API function prototype declarations. API functions are explained in Section 3.

2.12 Return Values

This shows the different values API functions can return. This return type is defined in "`r_flash_rx_if.h`".

```
/* Flash FIT error codes */
typedef enum _flash_err
{
    FLASH_SUCCESS = 0,
    FLASH_ERR_BUSY,          /* Flash module busy */
    FLASH_ERR_ACCESSW,       /* Access window error */
    FLASH_ERR_FAILURE,       /* Flash operation failure; programming error,
                             erasing error, blank check error, etc. */

    FLASH_ERR_CMD_LOCKED,    /* Peripheral in command locked state */
    FLASH_ERR_LOCKBIT_SET,    /* Type3 - Program/Erase error due to lock bit. */
    FLASH_ERR_FREQUENCY,     /* Illegal Frequency value attempted */
    FLASH_ERR_BYTES,         /* Invalid number of bytes passed */
    FLASH_ERR_ADDRESS,       /* Invalid address */
    FLASH_ERR_BLOCKS,        /* The "number of blocks" argument is invalid. */
    FLASH_ERR_PARAM,         /* Illegal parameter */
    FLASH_ERR_NULL_PTR,      /* Missing required argument */
    FLASH_ERR_UNSUPPORTED,   /* Command not supported for this flash type */
    FLASH_ERR_SECURITY,      /* Type4 - Pgm/Erase err due to part locked (FAW.FSPR) */
    FLASH_ERR_TIMEOUT,       /* Timeout condition */
    FLASH_ERR_ALREADY_OPEN   /* Open() called twice without intermediate Close() */
} flash_err_t;
```

2.13 Blocking Mode and Non-blocking Mode

This module supports both blocking mode and non-blocking mode.

2.13.1 Using in Blocking Mode

When using this module in blocking mode, set configuration options as shown below.

- FLASH_CFG_DATA_FLASH_BGO: 0
- FLASH_CFG_CODE_FLASH_BGO: 0

In blocking mode, the API functions of this module do not return until the operation has finished.

2.13.2 Using in Non-blocking Mode

When using this module in non-blocking mode, set configuration options as shown below.

- FLASH_CFG_DATA_FLASH_BGO: 1
- FLASH_CFG_CODE_FLASH_BGO: 1

In non-blocking mode, the API functions of this module return immediately after the flash process starts. Users should not access the respective area until the operation in the flash area finishes. Accessing it causes an error on the sequencer and the operation does not finish normally.

Results of flash process are provided via the callback function.

Register the callback function in advance by executing R_FLASH_Open() and specifying the FLASH_CMD_SET_BGO_CALLBACK command for the argument of R_FLASH_Control(). (See section 3.7 for details.)

A FRDYI interrupt occurs when the flash process completes. The callback function registered from the process in the FRDYI interrupt is called. Events indicating the completion status are passed to the callback function. These events are defined in "r_flash_rx_if.h" as shown below.

```
typedef enum _flash_interrupt_event
{
    FLASH_INT_EVENT_INITIALIZED,
    FLASH_INT_EVENT_ERASE_COMPLETE,
    FLASH_INT_EVENT_WRITE_COMPLETE,
    FLASH_INT_EVENT_BLANK,
    FLASH_INT_EVENT_NOT_BLANK,
    FLASH_INT_EVENT_TOGGLE_STARTUPAREA,
    FLASH_INT_EVENT_SET_ACCESSWINDOW,
    FLASH_INT_EVENT_LOCKBIT_WRITTEN,
    FLASH_INT_EVENT_LOCKBIT_PROTECTED,
    FLASH_INT_EVENT_LOCKBIT_NON_PROTECTED,
    FLASH_INT_EVENT_ERR_DF_ACCESS,
    FLASH_INT_EVENT_ERR_CF_ACCESS,
    FLASH_INT_EVENT_ERR_SECURITY,
    FLASH_INT_EVENT_ERR_CMD_LOCKED,
    FLASH_INT_EVENT_ERR_LOCKBIT_SET,
    FLASH_INT_EVENT_ERR_FAILURE,
    FLASH_INT_EVENT_TOGGLE_BANK,
    FLASH_INT_EVENT_END_ENUM
} flash_interrupt_event_t;
```

2.14 Adding the FIT Flash Module to Your Project

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

(1) Adding the FIT module to your project using the Smart Configurator in e² studio

By using the Smart Configurator in e² studio, the FIT module is automatically added to your project. Refer to "Renesas e² studio Smart Configurator User Guide (R20AN0451)" for details.

(2) Adding the FIT module to your project using the FIT Configurator in e² studio

By using the FIT Configurator in e² studio, the FIT module is automatically added to your project. Refer to "Adding Firmware Integration Technology Modules to Projects (R01AN1723)" for details.

(3) Adding the FIT module to your project using the Smart Configurator in CS+

By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to "Renesas e² studio Smart Configurator User Guide (R20AN0451)" for details.

(4) Adding the FIT module to your project in CS+

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

2.15 Usage Combined with Existing User Projects

Using the BSP startup disable function, this module can be used in combination with existing user projects.

The BSP startup disable function is a function to add and use this module and other peripheral FIT modules to an existing user project without creating a new project.

BSP and this module (if necessary, other peripheral FIT modules) are incorporated into the existing user project. Even though it is necessary to incorporate BSP, since all startup processing performed by the BSP become disabled, this module and other peripheral FIT modules can be used in combination with startup processing of the existing user project.

There are some settings and notes for using the BSP startup disable function. For details, refer to the application note "RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)".

2.16 Programming Code Flash from RAM

MCUs require that sections in RAM and code flash be created to hold the API functions for reprogramming code flash. This is required because the sequencer (with some exceptions in Type 3) cannot program or erase code flash while executing from code flash. The RAM section will need to be initialized after reset.

In order to enable code flash reprogramming, configure the `FLASH_CFG_CODE_FLASH_ENABLE` to 1 in the `r_flash_rx_config.h` file. Note that this is only for code flash programming.

This module of Rev. 4.00 or later supports multiple compilers. To use this module, different settings are required for each compiler. For details of the settings appropriate for the compiler to be used, refer to section 5.3.

2.17 Programming Code Flash from Code Flash

For Flash Type 3, and Flash Type 4 with more than or equal to 1.5M code flash, with certain limitations code flash can be programmed while running from code flash. Code flash is broken into multiple regions. Program and erase operations can be performed from the region in which a code is being executed to a different region. The size of these regions vary based upon the amount of code flash on the MCU. Refer to the Hardware Manuals for boundary details. Only MCUs with large code flash areas support this feature. For Flash Type 4 with more than or equal to 1.5M code flash, the boundaries correspond to bank boundaries.

When this method is used, set `FLASH_CFG_CODE_FLASH_ENABLE` and `FLASH_CFG_CODE_FLASH_RUN_FROM_ROM` to 1 in the `r_flash_rx_config.h` file. `FLASH_CFG_CODE_FLASH_BGO` (functions do not block/wait for completion) may be set to 0 or 1, but must match with `FLASH_CFG_DATA_FLASH_BGO`.

Be sure not set up the linker as just described in section 2.16, but do guarantee that the region the code is running from is not the region being operated on!

2.18 Dual Bank Operation

Flash Type 4 with more than or equal to 1.5M code flash can operate in two modes- linear and dual bank. Dual bank mode is a mode in which the code flash memory to be used is divided into two areas (banks). In dual bank mode, the start bank can be swapped using the command `R_FLASH_Control(FLASH_CMD_BANK_TOGGLE)`. Note that the swap does not take effect until the next MCU reset.

To operate in dual bank mode, it is necessary to change the constant defined in the configuration file (`r_bsp_config.h`) of BSP as follows.

- `BSP_CFG_CODE_FLASH_BANK_MODE`: 1 → 0
The default setting is "1". To operate in dual bank mode, set this constant to "0".

In addition, for the setting related to the start bank, the following constant is used. Set the constant as required.

- `BSP_CFG_CODE_FLASH_START_BANK`: 0 or 1
The default start bank is "0". If you wish the start bank to be "1", change the constant to "1".

This module of Rev. 4.00 or later supports multiple compilers. To use this module, different settings are required for each compiler. For details of the settings appropriate for the compiler to be used, refer to section 5.3.

2.19 Usage Notes

2.19.1 Data Flash Operations in Non-blocking Mode

When reprogramming data flash in non-blocking mode, code flash, RAM, and external memory can still be accessed. Care should be taken to make sure that the data flash is not accessed during data flash operations. This includes interrupts that may access the data flash.

2.19.2 Code Flash Operations in Non-blocking Mode

When reprogramming code flash in non-blocking mode, external memory and RAM can still be accessed. Since the flash FIT module API functions will return before the code flash operation has finished, the code that calls the API function will need to be in RAM, and the code will need to check for completion before issuing another Flash command. Note that this includes setting the code flash access window, swapping boot blocks/toggling startup area flag, erasing code flash, writing code flash, as well as reading the Unique ID with another FIT Module (R01AN2191).

2.19.3 Code Flash Operations and General Interrupts

Code flash or data flash areas cannot be accessed while a flash operation is on-going for that particular memory area. This means that the relocatable vector table will need to be taken care of when allowing interrupts to occur during flash operations.

The vector table is placed in code flash by default. If an interrupt occurs during code flash operation, then code flash will be accessed to fetch the interrupt's starting address and an error will occur. To fix this situation the user will need to relocate the vector table and any interrupt handlers that may occur outside of code flash. The user will also need to change the interrupt table register (INTB).

The module does not include the function to relocate the vector table and the interrupt handler. Please consider an appropriate method to relocate them according to the user system.

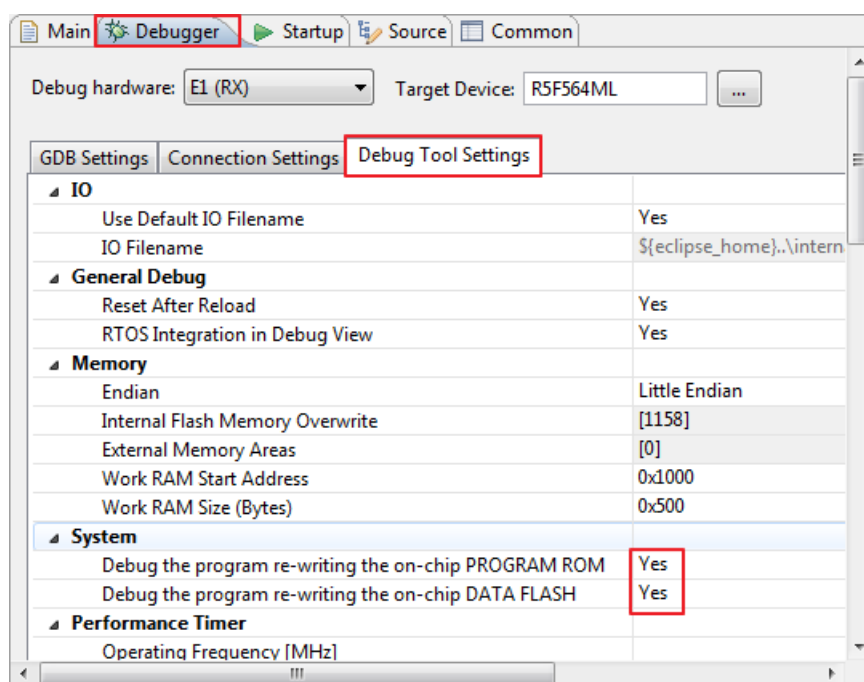
2.19.4 Emulator Debug Configuration

To confirm the data written to code flash and data flash during debug, change the Debug Tool Settings of the debug configuration as follows.

1. In Project Explorer, click the project you want to debug.
2. Click Execute > Debug Configuration to open the Debug Configuration window.
3. On the Debug Configuration window, expand the display of the “Renesas GDB Hardware Debugging” debug configuration and click the debug configuration you want to debug.
4. Switch to the “Debugger” tab, click the “Debug Tool Settings” in the “Debugger” tab and make the following settings.

— System

- Debug the program re-writing the on-chip PROGRAM ROM = “Yes”
- Debug the program re-writing the on-chip DATA FLASH = “Yes”



3. API Functions

3.1 Summary

The following functions are included in this design:

Function	Description
R_FLASH_Open()	Initializes the Flash FIT module.
R_FLASH_Close()	Closes the Flash FIT module.
R_FLASH_Erase()	Erases the specified block of code flash or data flash.
R_FLASH_BlankCheck()	Checks if the specified data flash or code flash area is blank.
R_FLASH_Write()	Write data to code flash or data flash.
R_FLASH_Control()	Configures settings for the status check, area protection, and switching areas for start-up program protection.
R_FLASH_GetVersion()	Returns the current version of this FIT module.

3.2 R_FLASH_Open()

The function initializes the Flash FIT module. This function must be called before calling any other API functions.

Format

```
flash_err_t R_FLASH_Open(void);
```

Parameters

None

Return Values

<i>FLASH_SUCCESS:</i>	<i>Flash FIT module initialized successfully</i>
<i>FLASH_ERR_BUSY:</i>	<i>A different flash process is being executed, try again later</i>
<i>FLASH_ERR_ALREADY_OPEN:</i>	<i>Open() called twice without an intermediate Close()</i>

Properties

Prototyped in file "r_flash_rx_if.h"

Description

This function initializes the Flash FIT module, and if FLASH_CFG_CODE_FLASH_ENABLE is 1, copies the API functions necessary for code flash erasing/reprogramming into RAM (not including vector table). Note that this function must be called before any other API function.

Example

```
flash_err_t err;

/* Initialize the API. */
err = R_FLASH_Open();

/* Check for errors. */
if (FLASH_SUCCESS != err)
{
    . . .
}
```

Special Notes:

None

3.3 R_FLASH_Close()

The function closes the Flash FIT module.

Format

```
flash_err_t R_FLASH_Close(void);
```

Parameters

None

Return Values

<i>FLASH_SUCCESS:</i>	<i>Flash FIT module closed successfully</i>
<i>FLASH_ERR_BUSY:</i>	<i>A different flash process is being executed, try again later</i>

Properties

Prototyped in file "r_flash_rx_if.h"

Description

This function closes the Flash FIT module. It disables the flash interrupts (if enabled) and sets the driver to an uninitialized state.

Example

```
flash_err_t err;

/* Close the driver */
err = R_FLASH_Close();

/* Check for error */
if (FLASH_SUCCESS != err)
{
    . . .
}
```

Special Notes:

None

3.4 R_FLASH_Erase()

This function is used to erase the specified block in code flash or data flash.

Format

```
flash_err_t R_FLASH_Erase(flash_block_address_t block_start_address,
                          uint32_t num_blocks);
```

Parameters

block_start_address

Specifies the start address of block to erase. The enum *flash_block_address_t* is defined in the corresponding MCU's *r_flash_rx/src/targets/mcu/r_flash_mcu.h* file. The blocks are labeled in the same fashion as they are in the device's Hardware Manual. For example, the block located at address 0xFFFFC000 is called Block 7 in the RX113 hardware manual, therefore "FLASH_CF_BLOCK_7" should be passed for this parameter. Similarly, to erase Data Flash Block 0 which is located at address 0x00100000, this argument should be FLASH_DF_BLOCK_0.

num_blocks

Specifies the number of blocks to be erased. For type 1 parts, *address + num_blocks* cannot cross a 256K boundary.

Return Values

<i>FLASH_SUCCESS:</i>	<i>Operation successful (if non-blocking mode is enabled, this means the operation was started successfully)</i>
<i>FLASH_ERR_BLOCKS:</i>	<i>Invalid number of blocks specified</i>
<i>FLASH_ERR_ADDRESS:</i>	<i>Invalid address specified</i>
<i>FLASH_ERR_BUSY:</i>	<i>A different flash process is being executed, or the module is not initialized</i>
<i>FLASH_ERR_FAILURE:</i>	<i>Erasing failure. Sequencer has been reset. Or callback function not registered (in non-blocking mode)</i>

Properties

Prototyped in file "r_flash_rx_if.h"

Description

Erases a contiguous number of code flash or data flash memory blocks.

The block size varies depending on MCU types. For example, on the RX111 both code and data flash block sizes are 1Kbytes. On the RX231 and RX23T the block size for code flash is 2 Kbytes and for data flash is 1Kbyte (no data flash on the RX23T). The equates FLASH_CF_BLOCK_SIZE for code flash and FLASH_DF_BLOCK_SIZE for data flash are provided for these values.

The enum *flash_block_address_t* is configured at compile time based on the memory configuration of the MCU device specified in the *r_bsp* module.

When the API is used in non-blocking mode, the FRDYI interrupt occurs after blocks for the specified number are erased, and then the callback function is called.

Example

```
flash_err_t err;

/* Erase Data Flash blocks 0 and 1 */
err = R_FLASH_Erase(FLASH_DF_BLOCK_0, 2);

/* Check for errors. */
if (FLASH_SUCCESS != err)
{
    . . .
}
```

Special Notes:

In order to erase a code flash block, the area to be erased needs to be in a rewritable area. FLASH_TYPE_1 uses access windows to identify this. The other flash types use lock bits which must be off for erasing.

3.5 R_FLASH_BlankCheck()

This function is used to determine if the specified area in either code flash or data flash is blank or not.

Format

```
flash_err_t R_FLASH_BlankCheck(uint32_t address,
                               uint32_t num_bytes,
                               flash_res_t *blank_check_result);
```

Parameters

address

The address of the area to blank check.

num_bytes

This is the number of bytes to be checked. The number of bytes specified must be a multiple of FLASH_DF_MIN_PGM_SIZE for a data flash address or FLASH_CF_MIN_PGM_SIZE for a code flash address. These equates are defined in r_flash_rx\src\targets\<mcu>\r_flash_<mcu>.h and are MCU specific. For type 1 parts, *address* + *num_bytes* cannot cross a 256K boundary.

**blank_check_result*

In blocking mode, specify the memory address to which the blank check results will be stored. In non-blocking mode, specify any value since this parameter is not used.

Return Values

<i>FLASH_SUCCESS:</i>	<i>Operation successful (in non-blocking mode, this means the operation was started successfully)</i>
<i>FLASH_ERR_FAILURE:</i>	<i>Blank check failed. Sequencer has been reset, or callback function not registered (in non-blocking mode)</i>
<i>FLASH_ERR_BUSY:</i>	<i>A different flash process is being executed or the module is not initialized</i>
<i>FLASH_ERR_BYTES:</i>	<i>num_bytes was either too large or not a multiple of the minimum programming size or exceed the maximum range</i>
<i>FLASH_ERR_ADDRESS:</i>	<i>Invalid address was input or address not divisible by the minimum programming size</i>
<i>FLASH_ERR_NULL_PTR:</i>	<i>blank_check_result for storing blank check results was NULL</i>

Properties

Prototyped in file "r_flash_rx_if.h"

Description

The flash area to write the program into must be blank.

The result of the blank check operation is placed into "blank_check_result" when operation is in blocking mode. This variable is of type flash_res_t which is defined in r_flash_rx_if.h. If the API is used in non-blocking mode, the result of the blank check is passed as the argument of the callback function after the blank check is complete.

Example:

Second argument is number of bytes to check (must be multiple of FLASH_DF_MIN_PGM_SIZE).

```
flash_err_t err;
flash_res_t result;

/* Blank check first 64 bytes of data flash block 0 */
err = R_FLASH_BlankCheck((uint32_t)FLASH_DF_BLOCK_0, 64, &result);
if (err != FLASH_SUCCESS)
{
    /* handle error */
}
else
{
    /* Check result. */
    if (FLASH_RES_NOT_BLANK == result)
    {
        /* Block is not blank. */
        . . .
    }
    else if (FLASH_RES_BLANK == ret)
    {
        /* Block is blank. */
        . . .
    }
}
```

Special Notes:

None

3.6 R_FLASH_Write()

This function is used to write data to code flash or data flash.

Format

```
flash_err_t R_FLASH_Write(uint32_t src_address,
                           uint32_t dest_address,
                           uint32_t num_bytes);
```

Parameters

src_address

This is the first address of the buffer containing the data to write to Flash.

dest_address

This is the first address of the code flash or data flash area to rewrite data. The address specified must be divisible by the minimum programming size. See *Description* below for important restrictions regarding this parameter.

num_bytes

The number of bytes contained in the buffer specified with *src_address*. This number must be a multiple of the minimum programming size for memory area you are writing to.

Return Values

<i>FLASH_SUCCESS:</i>	<i>Operation successful (in non-blocking mode, this means the operation was started successfully)</i>
<i>FLASH_ERR_FAILURE:</i>	<i>Programming failed. Possibly the destination address under access window or lockbit control; or callback function not present(in non-blocking mode)</i>
<i>FLASH_ERR_BUSY:</i>	<i>A different flash process is being executed or the module is not initialized</i>
<i>FLASH_ERR_BYTES:</i>	<i>Number of bytes provided was not a multiple of the minimum programming size or exceed the maximum range</i>
<i>FLASH_ERR_ADDRESS:</i>	<i>Invalid address was input or address not divisible by the minimum programming size.</i>

Properties

Prototyped in file "r_flash_rx_if.h"

Description

Writes data to flash memory. Before writing to any flash area, the area must already be erased.

When performing a write the user must make sure to start the write on an address divisible by the minimum programming size and make the number of bytes to write be a multiple of the minimum programming size. The minimum programming size differs depending on what MCU package is being used and whether the code flash or data flash is being written to.

An area to write data to code flash must be rewritable area (access window or lockbit allowed).

When the API is used in non-blocking mode, the callback function is called when all write operations are complete.

Example

```
flash_err_t err;
uint8_t write_buffer[16] = "Hello World...";

/* Write data to internal memory. */
err = R_FLASH_Write((uint32_t)write_buffer, dst_addr, sizeof(write_buffer));

/* Check for errors. */
if (FLASH_SUCCESS != err)
{
    . . .
}
```

Special Notes:

FLASH_DF_MIN_PGM_SIZE defines the minimum data flash program size.

FLASH_CF_MIN_PGM_SIZE defines the minimum code flash program size.

3.7 R_FLASH_Control()

This function implements all functionality except for programming, erasing, and black check.

Format

```
flash_err_t R_FLASH_Control(flash_cmd_t cmd
                           void *pcfg);
```

Parameters

cmd

Command to execute.

**pcfg*

Configuration parameters required by the specific command. This maybe NULL if the command does not require it.

Return Values

FLASH_SUCCESS: Operation successful (in non-blocking mode, this means the operations was started successfully)

FLASH_ERR_ADDRESS: Address is an invalid Code/Data Flash block start address

FLASH_ERR_NULL_PTR: *pcfg* was NULL for a command that expects a configuration structure

FLASH_ERR_BUSY: A different flash process is being executed or the module is not initialized

FLASH_ERR_CMD_LOCKED: The flash control circuit was in a command locked state and was reset

FLASH_ERR_ACCESSW: Access window error: Incorrect area specified

FLASH_ERR_PARAM: Invalid command

Properties

Prototyped in file "r_flash_rx_if.h"

Description

This function is an expansion function that implements non-core functionality of the sequencer. Depending on the command type a different argument type has to be passed.

Command	Argument	Operation
FLASH_CMD_RESET (Flash type 1, 3, 4)	NULL	Resets the flash sequencer. This may or may not wait for the current flash operation to complete (operation dependent). This command can be used even during flash operation.
FLASH_CMD_STATUS_GET (Flash type 1, 3, 4)	NULL	Returns the status of the API (Busy or Idle). This command can be used even during flash operation.
FLASH_CMD_SET_BGO_CALLBACK (Flash type 1, 3, 4)	flash_interrupt_config_t *	Registers the callback function.
FLASH_CMD_ACCESSWINDOW_GET (Flash type 1, 4)	flash_access_window_config_t *	Returns the access window boundaries for code flash.

Command	Argument	Operation
FLASH_CMD_ACCESSWINDOW_SET (Flash type 1, 4)	flash_access_window_config_t *	Specifies the access window boundaries for code flash. When using in non-blocking mode, FRDYI interrupt occurs after setting the access window and the callback function is called.**
FLASH_CMD_SWAPFLAG_GET (Flash type 1, 4)	uint32_t *	Loads the flag indicating the designated start-up area (SASMF type 1, BTFLG type 4).
FLASH_CMD_SWAPFLAG_TOGGLE (Flash type 1, 4)	NULL	Toggles the startup program area. Switches the area by the function placed in the RAM. The setting is enabled at next reset. When using in non-blocking mode, FRDYI interrupt occurs after switching the area and the callback function is called.**
FLASH_CMD_SWAPSTATE_GET (Flash type 1, 4)	uint8_t *	Reads the current startup-area select bit value (SAS value).
FLASH_CMD_SWAPSTATE_SET (Flash type 1, 4)	uint8_t *	Defines the startup-area select bits (FISR.SAS) by r_flash_rx_if.h and sets it. #define (value) FLASH_SAS_EXTRA (0) FLASH_SAS_DEFAULT (2) FLASH_SAS_ALTERNATE (3) FLASH_SAS_SWITCH_AREA (4) When FLASH_SAS_EXTRA, FLASH_SAS_DEFAULT, or FLASH_SAS_ALTERNATE is set, the value is directly set to FISR.SAS and the area is switched according to the value. When FLASH_SAS_SWITCH_AREA is set, the area is switched immediately. The function placed in the RAM is used for switching. The area specified in FLASH_SAS_EXTRA will be used after a reset.

Command	Argument	Operation
FLASH_CMD_LOCKBIT_READ (Flash type 3)	flash_lockbit_config_t *	Lockbit information of the specified block (FLASH_RES_LOCKBIT_STATE_PROTECTED or FLASH_RES_LOCKBIT_STATE_NON_PROTECTED) is set to the argument.
FLASH_CMD_LOCKBIT_WRITE (Flash type 3)	flash_lockbit_config_t *	Sets the lockbit for the number of blocks specified starting with the block address provided.
FLASH_CMD_LOCKBIT_ENABLE (Flash type 3)	NULL	Prohibits erasing/writing of blocks with lockbit set.
FLASH_CMD_LOCKBIT_DISABLE (Flash type 3)	NULL	Allows erasing/writing of blocks with lockbit set. NOTE: Erasing a block clears the lockbit.
FLASH_CMD_CONFIG_CLOCK (Flash type 3, 4)	uint32_t *	Speed in Hz that FCLK is running at. Only needs to be called if clock speed changes at run time.
FLASH_CMD_ROM_CACHE_ENABLE (RX24T, RX24U, RX66T, RX72T, flash type 4)	NULL	Enables caching of code flash (invalidates cache first).
FLASH_CMD_ROM_CACHE_DISABLE (RX24T, RX24U, RX66T, RX72T, flash type 4)	NULL	Disables caching of code flash. Call before rewriting code flash.
FLASH_CMD_ROM_CACHE_STATUS (RX24T, RX24U, RX66T, RX72T, flash type 4)	uint8_t *	Sets the value to 1 if caching is enabled; 0 if disabled.
FLASH_CMD_SET_NON_CACHED_RANGE0 (RX66T, RX66N, RX72T, RX72M, RX72N)	flash_non_cached_t *	Sets the range of RANGE0 to disable the code flash cache. Executing this command while cache is enabled temporarily prohibits cache.
FLASH_CMD_SET_NON_CACHED_RANGE1 (RX66T, RX66N, RX72T, RX72M, RX72N)	flash_non_cached_t *	Sets the range of RANGE1 to disable the code flash cache. Executing this command while cache is enabled temporarily prohibits cache.
FLASH_CMD_GET_NON_CACHED_RANGE0 (RX66T, RX66N, RX72T, RX72M, RX72N)	flash_non_cached_t *	Retrieves non-cached settings for RANGE0.
FLASH_CMD_GET_NON_CACHED_RANGE1 (RX66T, RX66N, RX72T, RX72M, RX72N)	flash_non_cached_t *	Retrieves non-cached settings for RANGE1.

Command	Argument	Operation
FLASH_CMD_BANK_TOGGLE (Flash type 4 with more than or equal to 1.5M code flash)	NULL	Swaps startup bank. Becomes effective at next reset. When using in non-blocking mode, FRDYI interrupt occurs after setting the bank selection register and the callback function is called.**
FLASH_CMD_BANK_GET (Flash type 4 with more than or equal to 1.5M code flash)	flash_bank_t *	Loads the current BANKSEL value (bank and address effective at next reset).

Note: ** These commands will block until completed even when in non-blocking mode. This is necessary while flash reconfigures itself. The callback function will still be called upon completion in non-blocking mode.

Example 1: Setting to non-blocking mode

Non-blocking mode is enabled when FLASH_CFG_DATA_FLASH_BGO equals 1 or FLASH_CFG_CODE_FLASH_BGO equals 1. To execute a code from the RAM and reprogram code flash, relocate the relocatable vector table to the RAM. Also, the callback function must be registered prior to write/erase/blank check calls.

```
void func(void)
{
    flash_err_t          err;
    flash_interrupt_config_t cb_func_info;
    uint32_t             *pvect_table;

    /* Relocate the Relocatable Vector Table in RAM */

    /* It is also possible to set the address of the flash ready interrupt
       function directly to ram_vect_table[23]. Please consider the method
       according to the user's system.*/
    pvect_table = (uint32_t *)__sectop("C$VECT");
    ram_vect_table[23] = pvect_table[23]; /* FRDYI Interrupt function Copy */
    set_intb((void *)ram_vect_table);

    /* Initialize the API. */
    err = R_FLASH_Open();
    /* Check for errors. */
    if (FLASH_SUCCESS != err)
    {
        ... (omission)
    }

    /* Set callback function and interrupt priority */
    cb_func_info.pcallback = u_cb_function;
    cb_func_info.int_priority = 1;
    err = R_FLASH_Control(FLASH_CMD_SET_BGO_CALLBACK, (void *)&cb_func_info);
    if (FLASH_SUCCESS != err)
    {
        printf("Control FLASH_CMD_SET_BGO_CALLBACK command failure.");
    }

    /* Perform operations on code flash */
    do_rom_operations();

    ... (omission)
}

#pragma section FRAM

void u_cb_function(void *event) /* callback function */
{
    flash_int_cb_args_t *ready_event = event;

    /* Perform ISR callback functionality here */
}

void do_rom_operations(void)
{
    /* Set cf access window, toggle startup area flag/swap boot blocks,
       erase, blank check, or write code flash here */

    ... (omission)
}

#pragma section
```

Example 2: Checking the API status

This section shows an example of R_FLASH_Erase() in non-blocking mode.

```
flash_err_t err;

/* erase all of data flash */
R_FLASH_Erase(FLASH_DF_BLOCK_0, FLASH_NUM_BLOCKS_DF);

/* check the API status */
while (R_FLASH_Control(FLASH_CMD_STATUS_GET, NULL) == FLASH_ERR_BUSY)
{
    /* execute any process */
}
```

Example 3: Get range of current access window

```
flash_err_t err;
flash_access_window_config_t access_info;

err = R_FLASH_Control(FLASH_CMD_ACCESSWINDOW_GET, (void *)&access_info);
if (FLASH_SUCCESS != err)
{
    printf("Control FLASH_CMD_ACCESSWINDOW_GET command failure.");
}
```

Example 4: Set access window code flash (flash types 1 and 4)

The area protection is used to prevent unauthorized programming or erasure of code flash blocks. The following example makes only block 3 writeable (and everything else is not writeable).

```
flash_err_t err;
flash_access_window_config_t access_info;

/* Allow write to Code Flash block 3 */

access_info.start_addr = (uint32_t) FLASH_CF_BLOCK_3;
access_info.end_addr = (uint32_t) FLASH_CF_BLOCK_2;
err = R_FLASH_Control(FLASH_CMD_ACCESSWINDOW_SET, (void *)&access_info);
if (FLASH_SUCCESS != err)
{
    printf("Control FLASH_CMD_ACCESSWINDOW_SET command failure.");
}
```

Example 5: Get value of active startup area

The following example shows how to read the value of the start-up area setting monitor flag (FSCMR.SSMF).

```
uint32_t swap_flag;
flash_err_t err;

err = R_FLASH_Control(FLASH_CMD_SWAPFLAG_GET, (void *)&swap_flag);
if (FLASH_SUCCESS != err)
{
    printf("Control FLASH_CMD_SWAPFLAG_GET command failure.");
}
```

Example 6: Swap active startup area

The following example shows how to toggle the active start-up program area. Swap the area with the function placed in RAM.

```
flash_err_t err;

/* Swap the active area from Default to Alternate or vice versa. */

err = R_FLASH_Control(FLASH_CMD_SWAPFLAG_TOGGLE, FIT_NO_PTR);
if(FLASH_SUCCESS != err)
{
    printf("Control FLASH_CMD_SWAPFLAG_TOGGLE command failure.");
}
```

Example 7: Get value of startup area select bit

The example below shows how to read the current value in the start-up area select bit (FISR.SAS).

```
uint8_t      swap_area;
flash_err_t err;

err = R_FLASH_Control(FLASH_CMD_SWAPSTATE_GET, (void *)&swap_area);
if (FLASH_SUCCESS != err)
{
    printf("Control FLASH_CMD_SWAPSTATE_GET command failure.");
}
```

Example 8: Set value of startup area select bit

The example below shows how to set the value to the start-up area select bit (FISR.SAS) for the start-up program area. Swap the area with the function placed in RAM. After a reset, the area will be the one specified with FLASH_SAS_EXTRA.

```
uint8_t      swap_area;
flash_err_t err;

swap_area = FLASH_SAS_SWITCH_AREA;
err = R_FLASH_Control(FLASH_CMD_SWAPSTATE_SET, (void *)&swap_area);
if (FLASH_SUCCESS != err)
{
    printf("Control FLASH_CMD_SWAPSTATE_SET command failure.");
}
```

Example 9: Configure cache setting during code flash rewriting

The example below shows cache command usage when rewriting code flash.

```
uint8_t    status;

/* Enable caching towards beginning of application */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_ENABLE, NULL);

/* Put main code here; optionally verify that flash is enabled */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_STATUS, &status);
if (status != 1)
{
    // should never happen
}

/* Prepare to rewrite code flash */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_DISABLE, NULL);

/* Erase, write, and verify new code here */

/* Re-enable caching */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_ENABLE, NULL);
```

Example 10: Limit code flash caching over a specific range.

The example below shows non-cached range command usage. It is legal to have non-cached ranges overlap.

```
flash_non_cached_t range;
uint8_t    status;

/* Do not cache fast-instruction fetching or operand access by the CPU
 * for the first 1K of code flash in flash block 10. Code flash caching will be
 * temporarily
 * disabled by the Control() command if was already enabled.
 */
range.start_addr = (uint32_t)FLASH_CF_BLOCK_10;
range.size = FLASH_NON_CACHED_1_KBYTE;
range.type_mask = FLASH_NON_CACHED_MASK_IF | FLASH_NON_CACHED_MASK_OA;

R_FLASH_Control(FLASH_CMD_SET_NON_CACHED_RANGE0, &range);

/* Enable caching */
R_FLASH_Control(FLASH_CMD_ROM_CACHE_ENABLE, NULL);

/* Retrieve non-cached settings for RANGE0 */
R_FLASH_Control(FLASH_CMD_GET_NON_CACHED_RANGE0, &range);
```

Special Notes:

None

3.8 R_FLASH_GetVersion()

Returns the current version of the Flash FIT module.

Format

```
uint32_t R_FLASH_GetVersion(void);
```

Parameters

None.

Return Values

Version of the Flash FIT module.

Properties

Prototyped in file "r_flash_rx_if.h"

Description

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

Example

```
uint32_t cur_version;

/* Get version of installed Flash FIT. */
cur_version = R_FLASH_GetVersion();

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

Special Notes:

None

4. Demo Projects

Demo projects are complete stand-alone programs. They include function main() that utilizes the module and its dependent modules (e.g. r_bsp). The standard naming convention for the demo project is <module>_demo_<board> where <module> is the peripheral acronym (e.g. s12ad, cmt, sci) and the <board> is the standard RSK (e.g. rskrx113). For example, s12ad FIT module demo project for RSKRX113 will be named as s12ad_demo_rskrx113. Similarly the exported .zip file will be <module>_demo_<board>.zip. For the same example, the zipped export/import file will be named as s12ad_demo_rskrx113.zip

Note that demo projects do not support a compiler other than Renesas Electronics C/C++ Compiler Package for RX Family.

4.1 flash_demo_rskrx113

This is a simple demo for the RSKRX113 starter kit. The demo uses blocking mode to execute flash erasing, blank check, and programming. Each write function is verified with a read-back of data. Note the “pragma section FRAM” for writing to code flash and the corresponding section definitions in the linker (see project Properties->C/C++ Build ->Settings ->Tool Settings (tab) ->Linker ->Section and ->Output).

Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.

Boards Supported

RSKRX113

Evaluation Environment

Version used: BSP Rev. 5.30, FLASH FIT Rev. 4.30

4.2 flash_demo_rskrx231

This is a simple demo for the RSKRX231 starter kit. The demo uses blocking mode to execute flash erasing, blank check, and programming. Each write function is verified with a read-back of data. Note the “pragma section FRAM” for writing to code flash and the corresponding section definitions in the linker (see project Properties->C/C++ Build ->Settings ->Tool Settings (tab) ->Linker ->Section and ->Output).

Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.

Boards Supported

RSKRX231

Evaluation Environment

Version used: BSP Rev. 5.30, FLASH FIT Rev. 4.30

4.3 flash_demo_rskrx23t

This is a simple demo for the RSKRX23T starter kit. The demo uses blocking mode to execute flash erasing, blank check, and programming. Each write function is verified with a read-back of data. Note the “pragma section FRAM” for writing to code flash and the corresponding section definitions in the linker (see project Properties->C/C++ Build ->Settings ->Tool Settings (tab) ->Linker ->Section and ->Output).

Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.

Boards Supported

RSKRX23T

Evaluation Environment

Version used: BSP Rev. 5.30, FLASH FIT Rev. 4.30

4.4 flash_demo_rskrx130

This is a simple demo for the RSKRX130 starter kit. The demo uses blocking mode to execute flash erasing, blank check, and programming. Each write function is verified with a read-back of data. Note the “pragma section FRAM” for writing to code flash and the corresponding section definitions in the linker (see project Properties->C/C++ Build ->Settings ->Tool Settings (tab) ->Linker ->Section and ->Output).

Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.

Boards Supported

RSKRX130

Evaluation Environment

Version used: BSP Rev. 5.30, FLASH FIT Rev. 4.30

4.5 flash_demo_rskrx24t

This is a simple demo for the RSKRX24T starter kit. The demo uses blocking mode to execute flash erasing, blank check, and programming. Each write function is verified with a read-back of data. Note the “pragma section FRAM” for writing to code flash and the corresponding section definitions in the linker (see project Properties->C/C++ Build ->Settings ->Tool Settings (tab) ->Linker ->Section and ->Output).

Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.

Boards Supported

RSKRX24T

Evaluation Environment

Version used: BSP Rev. 5.30, FLASH FIT Rev. 4.30

4.6 flash_demo_rskrx65n

This is a simple demo for the RSKRX65N starter kit. The demo uses blocking mode to execute flash erasing, blank check, and programming. Each write function is verified with a read-back of data. Note the “pragma section FRAM” for writing to code flash and the corresponding section definitions in the linker (see project Properties->C/C++ Build ->Settings ->Tool Settings (tab) ->Linker ->Section and ->Output).

Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.

Boards Supported

RSKRX65N-1

Evaluation Environment

Version used: BSP Rev. 5.30, FLASH FIT Rev. 4.30

4.7 flash_demo_rskrx24u

This is a simple demo for the RSKRX24U starter kit. The demo uses blocking mode to execute flash erasing and programming. Each write function is verified with a read-back of data. Note the “pragma section FRAM” for writing to code flash and the corresponding section definitions in the linker (see project Properties->C/C++ Build ->Settings ->Tool Settings (tab) ->Linker ->Section and ->Output).

Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.

Boards Supported

RSKRX24U

Evaluation Environment

Version used: BSP Rev. 5.30, FLASH FIT Rev. 4.30

4.8 flash_demo_rskrx65n2mb_bank0_bootapp / _bank1_otherapp

This is a simple demo for the dual bank operation of the RX65N-2MB demo board. The demo uses blocking mode and the banks are swapped according to the BANKSEL register value at next reset. The bank 0 application flashes LED0 when it is running. The bank 1 application flashes LED1 when it is running.

Setup and Execution

1. Build flash_demo_rskrx65n2mb_bank0_bootapp, and build flash_demo_rskrx65n2mb_bank1_otherapp.
2. Download (HardwareDebug) flash_demo_rskrx65n2mb_bank0_bootapp (its debug configuration also downloads the other app).
3. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.
4. Notice LED0 is flashing. Press the reset switch on the board. Notice LED1 is flashing (banks have swapped and the other application is now running). Continue this reset process if desired.

Boards Supported

RSKRX65N-2MB

Evaluation Environment

Version used: BSP Rev. 5.30, FLASH FIT Rev. 4.30

4.9 flash_demo_rskrx64m

This is a simple demo for the RSKRX64M starter kit. The demo uses blocking mode to execute flash erasing and programming. Each write function is verified with a read-back of data. Note the “pragma section FRAM” for writing to code flash and the corresponding section definitions in the linker (see project Properties->C/C++ Build ->Settings ->Tool Settings (tab) ->Linker ->Section and ->Output).

Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.

Boards Supported

RSKRX64M

Evaluation Environment

Version used: BSP Rev. 5.30, FLASH FIT Rev. 4.30

4.10 flash_demo_rskrx64m_runrom

This is a simple demo for the RSKRX64M starter kit. What sets this apart from other demos is that this makes use of the RX64M feature which allows an application to run from one region of code flash while erasing/writing to another. (Most other MCUs require code that could execute during a code flash erase/write to be located in RAM.) The demo uses blocking mode to execute flash erasing and programming. Each write function is verified with a read-back of data. Notice that the typical Linker set up for supporting code flash erase/write (RAM locating) is not necessary in this demo, and that FLASH_CFG_CODE_FLASH_RUN_FROM_ROM is set to 1 in “r_flash_rx_config.h”.

Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.

Boards Supported

RSKRX64M

Evaluation Environment

Version used: BSP Rev. 5.30, FLASH FIT Rev. 4.30

4.11 flash_demo_rskrx66t

This is a simple demo for the RSKRX66T starter kit. The demo uses blocking mode to execute flash erasing and programming. Each write function is verified with a read-back of data. Note the “pragma section FRAM” for writing to code flash and the corresponding section definitions in the linker (see project Properties->C/C++ Build ->Settings ->Tool Settings (tab) ->Linker ->Section and ->Symbol file).

Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.

Boards Supported

RSKRX66T

Evaluation Environment

Version used: BSP Rev. 5.30, FLASH FIT Rev. 4.30

4.12 flash_demo_rskrx72t

This is a simple demo for the RSKRX72T starter kit. The demo uses blocking mode to execute flash erasing and programming. Each write function is verified with a read-back of data. Note the “pragma section FRAM” for writing to code flash and the corresponding section definitions in the linker (see project Properties->C/C++ Build ->Settings ->Tool Settings (tab) ->Linker ->Section and ->Symbol file).

Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.

Boards Supported

RSKRX72T

Evaluation Environment

Version used: BSP Rev. 5.30, FLASH FIT Rev. 4.30

4.13 flash_demo_rskrx72m_bank0_bootapp / _bank1_otherapp

This is a simple demo for the dual bank operation of the RSKRX72M demo board. The demo uses blocking mode and the banks are swapped according to the BANKSEL register value at next reset. The bank 0 application flashes LED0 when it is running. The bank 1 application flashes LED1 when it is running.

Setup and Execution

1. Build flash_demo_rskrx72m_bank0_bootapp, and build flash_demo_rskrx72m_bank1_otherapp.
2. Download (HardwareDebug) flash_demo_rskrx72m_bank0_bootapp (its debug configuration also downloads the other app).
3. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.
4. Notice LED0 is flashing. Press the reset switch on the board. Notice LED1 is flashing (banks have swapped and the other application is now running). Continue this reset process if desired.

Boards Supported

RSKRX72M

Evaluation Environment

Version used: BSP Rev.5.30, FLASH FIT Rev.4.30

4.14 Adding a Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the distribution file for this application note. To add a demo project to a workspace, select *File >> Import >> General >> Existing Projects into Workspace*, then click “Next”. From the Import Projects dialog, choose the “Select archive file” radio button. “Browse” to the FITDemos subdirectory, select the desired demo zip file, then click “Finish”.

4.15 Downloading Demo Projects

Demo projects are not included in the RX Driver Package. When using the demo project, the FIT module needs to be downloaded. To download the FIT module, right click on this application note and select “Sample Code (download)” from the context menu in the *Smart Brower >> Application Notes* tab.

5. Appendices

5.1 Confirmed Operation Environment

This section describes confirmed operation environment for the Flash FIT module.

Table 5.1 Confirmed Operation Environment (Rev. 4.00)

Item	Contents
Integrated development environment	Renesas Electronics e ² studio Version 7.3.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.4.00
Board used	Renesas Starter Kit for RX113 (product No.: R0K505113xxxxxx) Renesas Starter Kit for RX130 (product No.: RTK5005130xxxxxxxxx) Renesas Starter Kit for RX231 (product No.: R0K505231xxxxxx) Renesas Starter Kit for RX23T (product No.: RTK500523Txxxxxxxxx) Renesas Starter Kit for RX24T (product No.: RTK500524Txxxxxxxxx) Renesas Starter Kit for RX24U (product No.: RTK500524Uxxxxxxxxx) Renesas Starter Kit+ for RX64M (product No.: R0K50564Mxxxxxx) Renesas Starter Kit for RX66T (product No.: RTK50566Txxxxxxxxx) Renesas Starter Kit for RX72T (product No.: RTK5572Txxxxxxxxxx) Renesas Starter Kit+ for RX65N (product No.: RTK500565Nxxxxxxxxx) Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565Nxxxxxxxxxx)

Table 5.2 Confirmed Operation Environment (Rev. 4.10)

Item	Contents
Integrated development environment	Renesas Electronics e ² studio Version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.4.10
Board used	Renesas Solution Starter Kit for RX23W (product No.: RTK5523Wxxxxxxxxxx)

Table 5.3 Confirmed Operation Environment (Rev. 4.20)

Item	Contents
Integrated development environment	Renesas Electronics e ² studio Version 7.3.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.4.20
Board used	Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx)

Table 5.4 Confirmed Operation Environment (Rev. 4.30)

Item	Contents
Integrated development environment	Renesas Electronics e ² studio Version 7.4.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.4.30
Board used	RX13T CPU Card (product No.: RTK0EMXA10xxxxxxxxxx)

Table 5.5 Confirmed Operation Environment (Rev. 4.40)

Item	Contents
Integrated development environment	Renesas Electronics e ² studio Version 7.5.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment.-lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.4.40
Board used	Renesas Solution Starter Kit for RX23E-A (product No.: RTK0ESXB10xxxxxxx)

Table 5.6 Confirmed Operation Environment (Rev. 4.50)

Item	Contents
Integrated development environment	Renesas Electronics e ² studio Version 7.5.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.4.50
Board used	Renesas Starter Kit+ for RX72N (product No.: RTK5572Nxxxxxxxxxx)

5.2 Troubleshooting

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

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

- Using CS+:
Application note "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)"
- Using e² studio:
Application note "Adding Firmware Integration Technology Modules to Projects (R01AN1723)"

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

(2) Q: I have added the FIT module to the project, changed the compiler option and built it. Then a ROM access violation is detected.

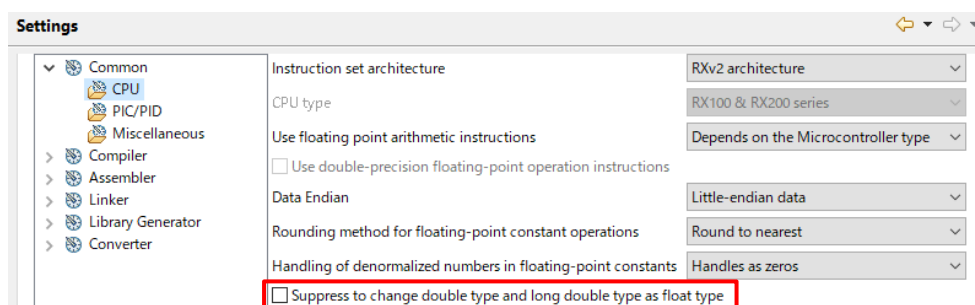
A: To use this FIT module to run codes from RAM to rewrite the code flash memory, all codes used need to be loaded to the RAM.

Depending on the compiler option setting, the loaded destination may be ROM or RAM.

If the compiler option needs to be changed, confirm by outputting to a list file the fact that the codes may not be loaded to the ROM as a result of the change of the compiler option.

The following shows an example of a ROM access violation due to a change in the compiler option.

A-1: Default compiler option settings



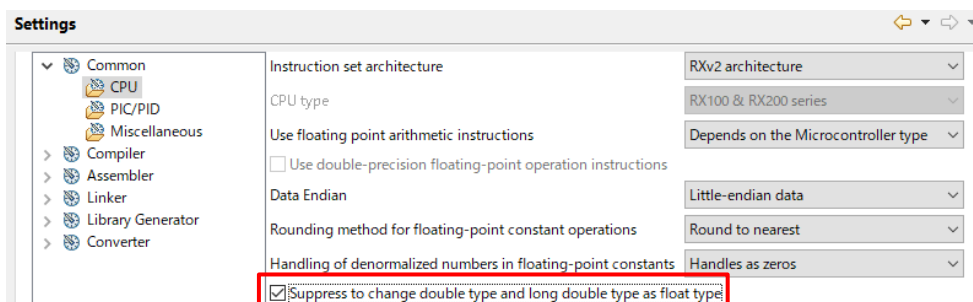
Output result of the list file of the default compiler option settings

```

000003AC FC472E      ^ ^      ITOF R2, R14J
000003AF FD723E005CC149  ^ ^      FMUL #49C15C00H, R14J
000003B6          ^ ^      L127: ^ ; if_break_bb33J
000003B6 92C5          ^ ^      MOV.W R5, 14H[R4]J
000003B8 FCA7E1          ^ ^      FTOU R14, R1J
000003BB A1C1          ^ ^      MOV.L R1, 18H[R4]J
000003BD 6601          ^ ^      MOV.L #00000000H, R1J

```

A-2: Compiler option change



Output result of the list file after the change in the compiler option

```

000003C1 EF21      ^ ^   MOV.L R2, R1J
000003C3 05rrrrrr      A ^ ^   BSR COM_CONV32udJ
000003C7 6603      ^ ^ ^   MOV.L #00000000H, R3J
000003C9 FB4280283841      ^ ^ ^   MOV.L #41382880H, R4J
000003CF 05rrrrrr      A ^ ^   BSR COM_MULdJ
000003D3 754740      ^ ^ ^   MOV.L #00000040H, R7J
000003D6      L127: ^ ^   ; if break bb33J
000003D6 05rrrrrr      A ^ ^   BSR COM_CONVd32uJ
000003DA A1E1      ^ ^ ^   MOV.L R1, 18H[R6]J
000003DC 6601      ^ ^ ^   MOV.L #00000000H, R1J
000003DE 92E7      ^ ^ ^   MOV.W R7, 14H[R6]J

```

A-1 shows a list file of the default compiler option settings, and A2 shows a list file after the change in the compiler option.

The difference between the A1 and A2 compiler option results in the difference between list file output results.

The red frame parts shown in the list file of A-2 indicate that they have been replaced with runtime library functions.

These runtime library functions are positioned in the "P" section by default and are not loaded to RAM. For that reason, a ROM access violation will occur during program execution.

5.3 Compiler-Dependent Settings

This module of Rev. 4.00 or later supports multiple compilers. To use this module, different settings are required for each compiler as shown below.

5.3.1 Using Renesas Electronics C/C++ Compiler Package for RX Family

This section describes how to use Renesas Electronics C/C++ Compiler Package for RX Family as the compiler.

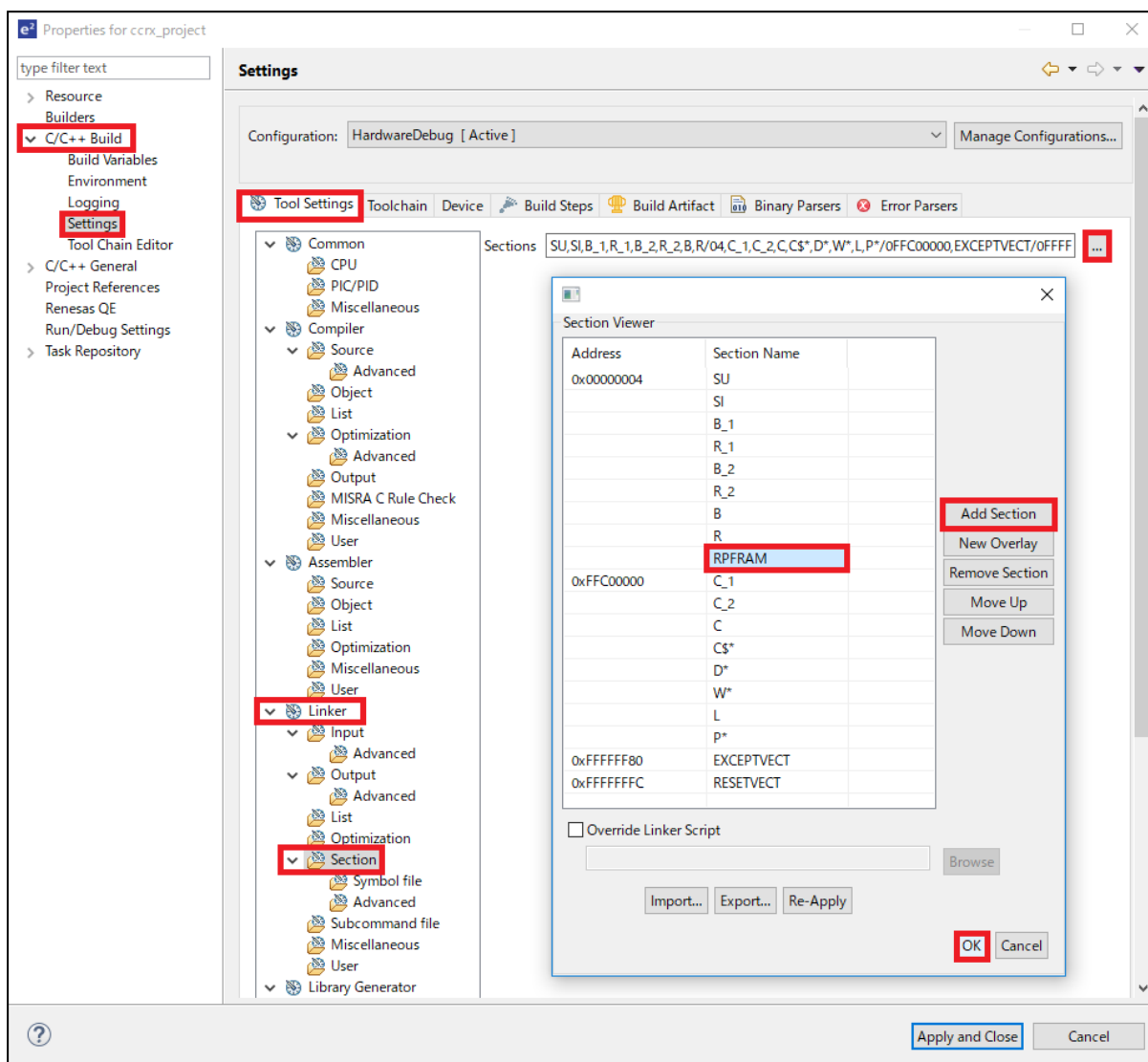
The process of setting up the linker sections and mapping from code flash to RAM need to be done in e² studio.

5.3.1.1 Programming Code Flash from RAM

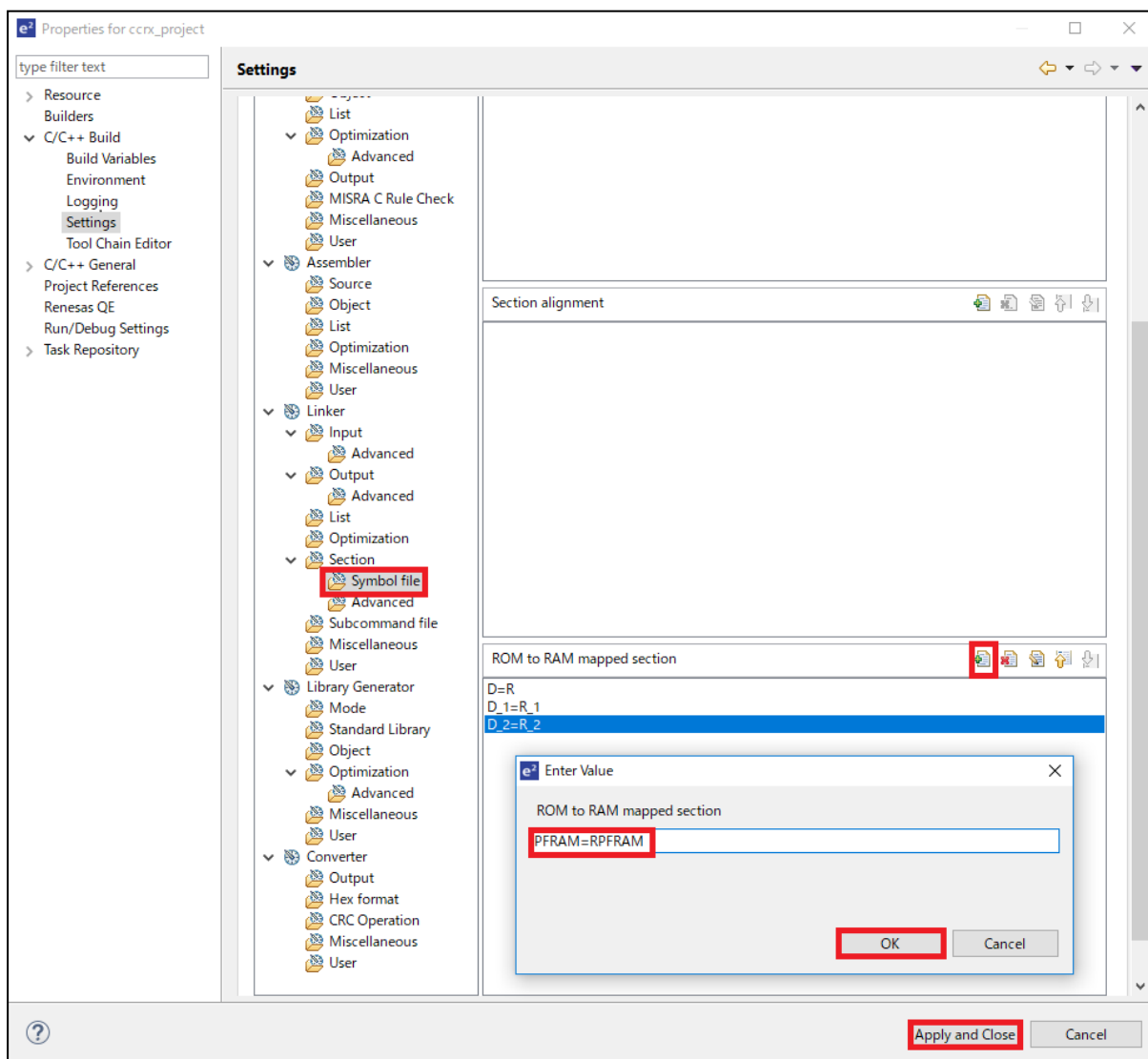
This section describes addition of sections, mapping from code flash to RAM, and placement of programs that operate during code flash re-writing.

1. Add a 'RPFRAM' section in a RAM area.

- (1) In Project Explorer, click the project you want to debug.
- (2) Click File > Properties to open the Properties window.
- (3) On the Properties window, click C/C++ Build > Settings.
- (4) Select the "Tool Settings" tab, click Linker > Section, and click the [...] button to display the Section Viewer window.
- (5) On the Section Viewer window, click the [Add Section] button to add a 'RPFRAM' section in a RAM area, and then click the [OK] button.



2. Map the code flash section (PFRAM) address to the RAM section (RPFRAM) address.
 - (1) After clicking “Symbol file”, click the “Add” icon of the section to be mapped from ROM to RAM.
 - (2) On the Enter Value window, enter ‘PFRAM=RPFRAM’ and then click the [OK] button.
 - (3) Click the [Apply and Close] button.



3. Programs that operate during code flash re-writing such as interrupt callback function, etc. need to be placed in the FRAM section.

```
#pragma section FRAM
/* Function that operates during code flash re-writing */
void func(void){...}

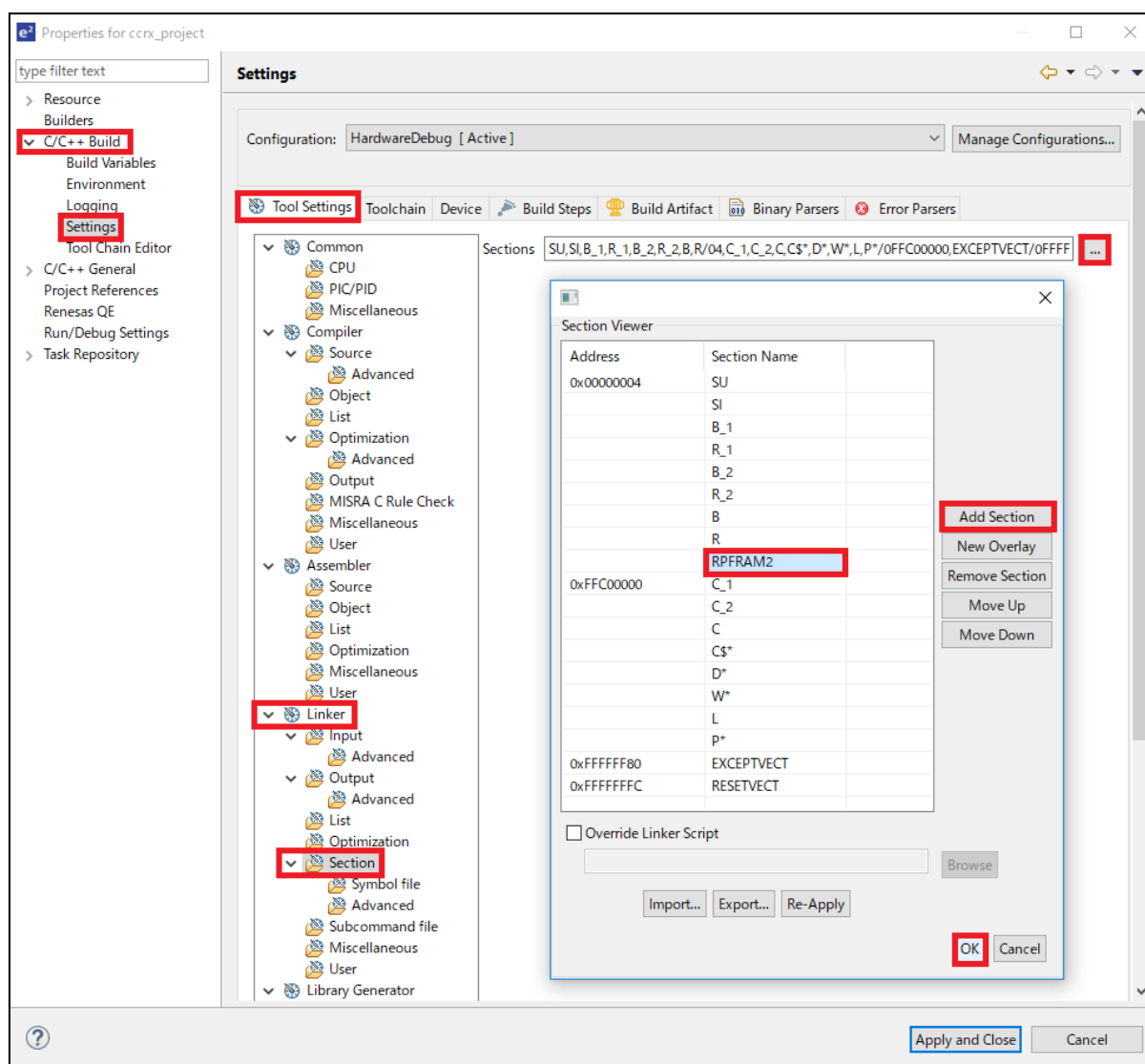
/* Callback function that operates during code flash re-writing */
void cb_func(void){...}
#pragma section
```

5.3.1.2 Programming Code Flash Using the Dual Bank Function

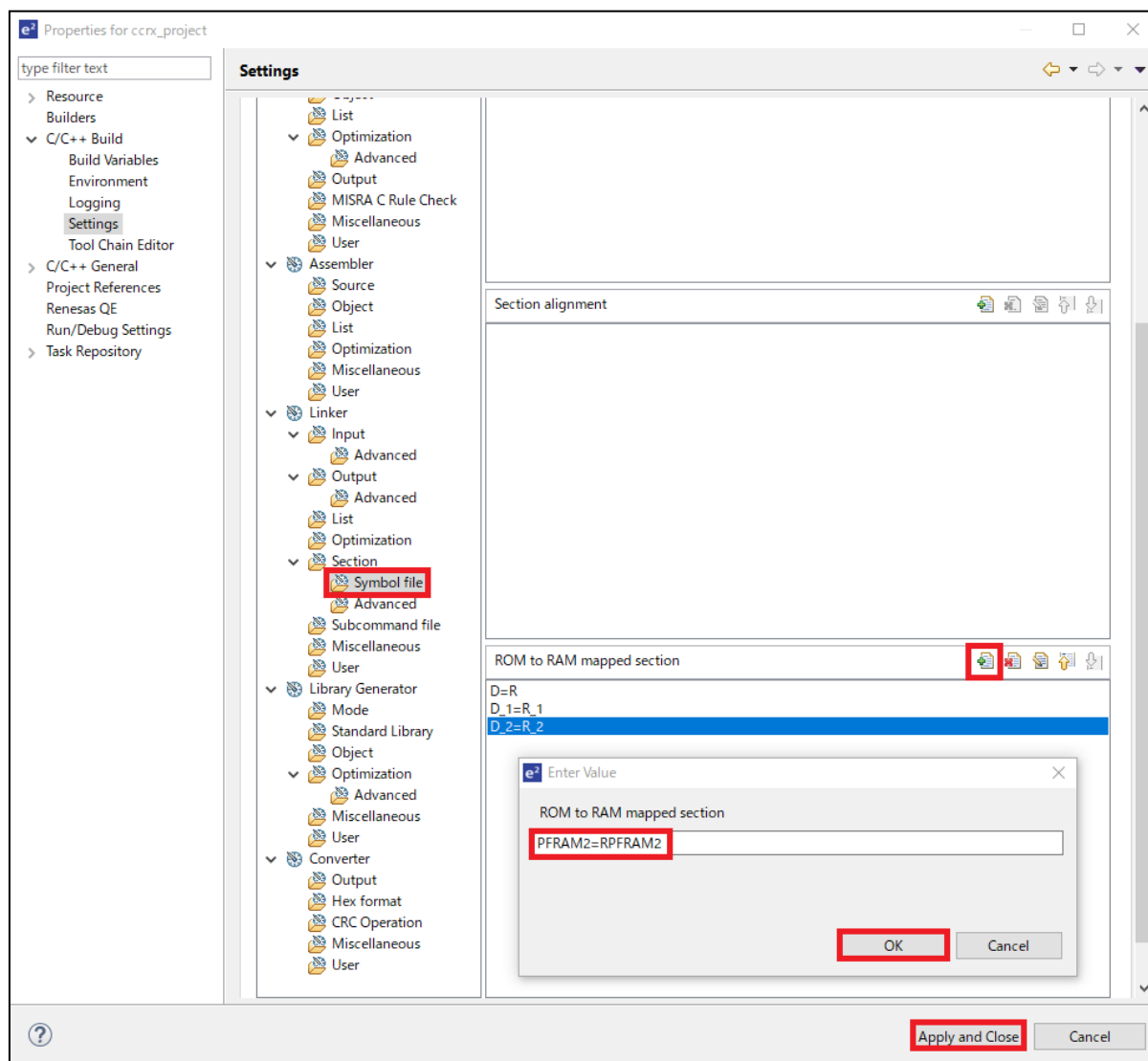
This section describes addition of sections, mapping from code flash to RAM, and debugging with the dual bank function.

1. Add a 'RPFRAM2' section in a RAM area.

- (1) In Project Explorer, click the project you want to debug.
- (2) Click File > Properties to open the Properties window.
- (3) On the Properties window, click C/C++ Build > Settings.
- (4) Select the "Tool Settings" tab, click Linker > Section, and click the [...] button to display the Section Viewer window.
- (5) On the Section Viewer window, click the [Add Section] button to add a 'RPFRAM2' section in a RAM area, and then click the [OK] button.



2. Map the code flash section (PFRAM2) address to the RAM section (RPFRAM2) address.
 - (1) After clicking “Symbol file”, click the “Add” icon of the section to be mapped from ROM to RAM.
 - (2) On the Enter Value window, enter ‘PFRAM2=RPFRAM2’ and then click the [OK] button.
 - (3) Click the [Apply and Close] button.



3. The following describes how to load objects for two banks when connecting an application related to the dual bank function to a target device and perform debugging. Perform this procedure as required.

- (1) In Project Explorer, click the project you want to debug.
- (2) Click Execute > Debug Configuration to open the Debug Configuration window.
- (3) On the Debug Configuration window, expand the display of the “Renesas GDB Hardware Debugging” debug configuration and click the debug configuration you want to debug.
- (4) Switch to the “Startup” tab, and click the [Add...] button of “Load image and symbols” on the “Startup” tab.
- (5) On the Edit Download Module window, specify an object for the other startup bank and then click the [OK] button.
- (6) Select a “Load type”.

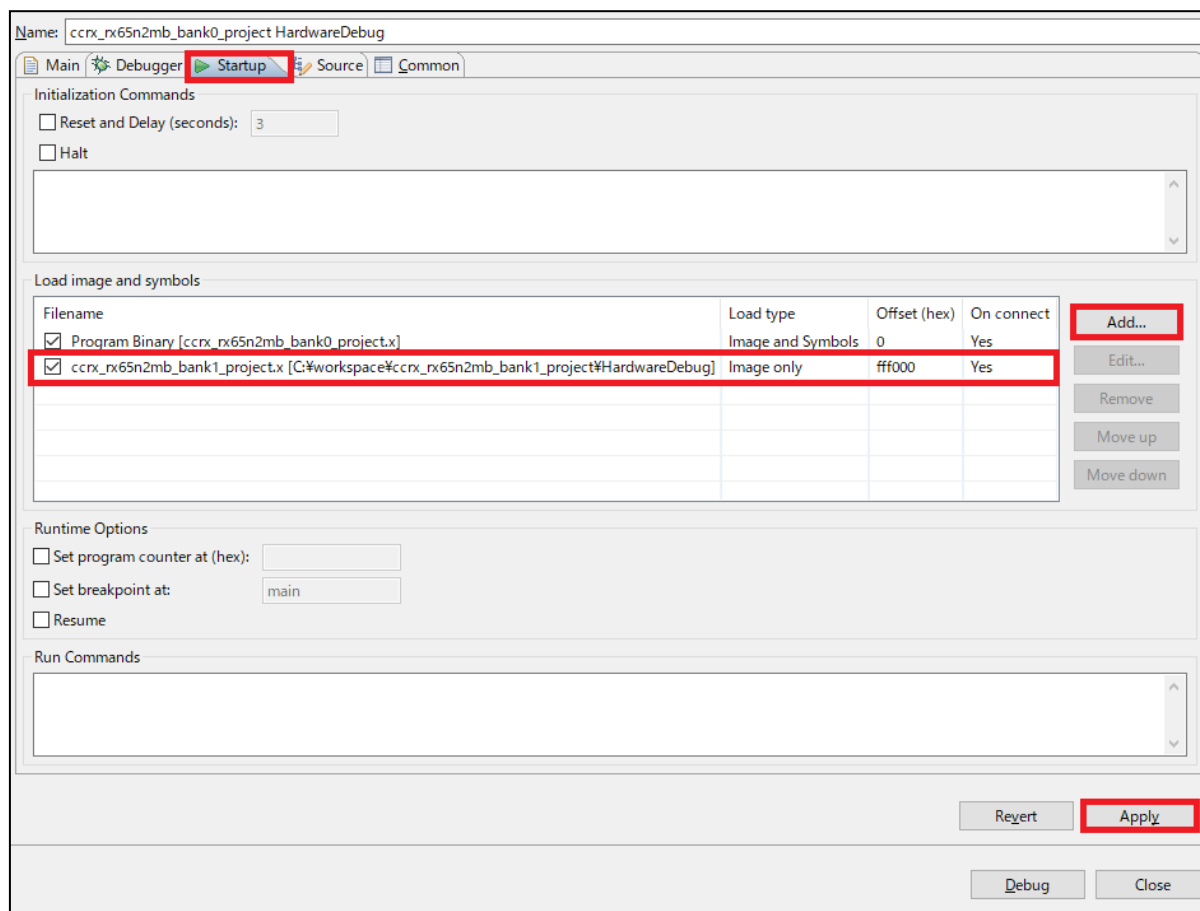
e2studio can only maintain one debug symbol table at a time. You can set the load type of either of the applications to “Image and Symbols”.

- (7) Specify “Offset”.

The “Offset” specification differs depending on the capacity of the code flash memory. For details, refer to the User's Manual for each device used.

The following shows an example when the target device is RX65N and the code flash capacity is 2MB. For “Offset”, enter fff00000, for which -1MB is two's complement. By doing so, the application to be assigned to the other startup bank will be loaded into memory 1Mb lower than the values shown in the linker or map file.

- (8) Click the [Apply] button.



5.3.2 Using GCC for Renesas RX

This section describes how to use GCC for Renesas RX as the compiler.

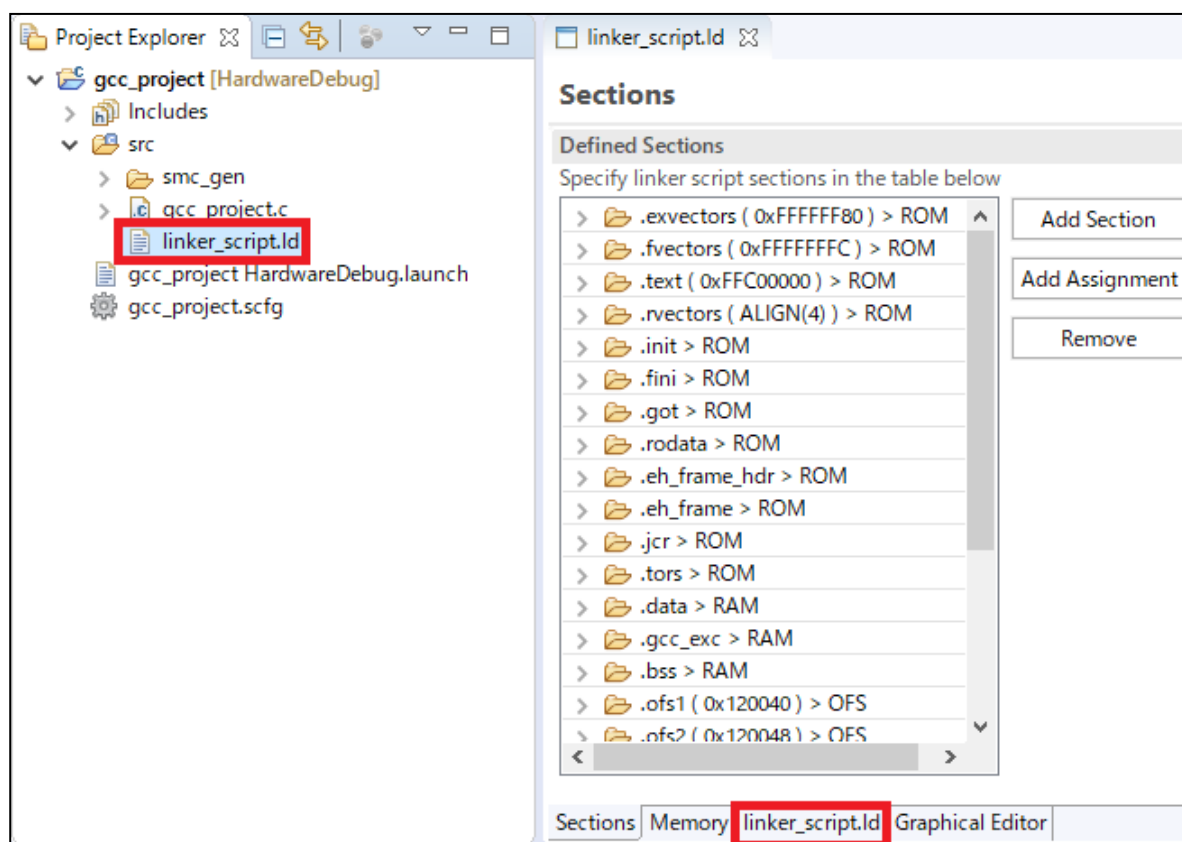
For the linker setting, it is necessary to edit the linker settings file generated by e² studio.

5.3.2.1 Programming Code Flash from RAM

This section describes addition of linker settings and placement of programs that operate during code flash re-writing.

1. Add a setting in the linker settings file (linker_script.ld).

- (1) From Project Explorer, right-click the linker settings file (linker_script.ld), and select "Open".
- (2) On the linker_script.id window, click the "linker_script_id" tab.



(3) Add the following (a) to (c) in the linker settings file (linker_script.ld).

- (a) `. += _edata - _data;`
- (b) `.pfram ALIGN(4):`

```
{
    _PFRAM_start = .;
    . += _RPFRAM_end - _RPFRAM_start;
    _PFRAM_end = .;
} > ROM
```
- (c) `.rpfram ALIGN(4): AT(_PFRAM_start)`

```
{
    _RPFRAM_start = .;
    *(PFRAM)
    . = ALIGN(4);
    _RPFRAM_end = .;
} > RAM
```

```

77     .ctors :
78     {
79         __CTOR_LIST__ = .;
80         . = ALIGN(2);
81         __ctors = .;
82         *(.ctors)
83         __ctors_end = .;
84         __CTOR_END__ = .;
85         __DTOR_LIST__ = .;
86         __dtors = .;
87         *(.dtors)
88         __dtors_end = .;
89         __DTOR_END__ = .;
90         . = ALIGN(2);
91         __mdata = .;
92         . += _edata - _data;
93     } > ROM
94     .pfram ALIGN(4):
95     {
96         __PFRAM_start = .;
97         . += _RPFRAM_end - _RPFRAM_start;
98         __PFRAM_end = .;
99     } > ROM
100    .data : AT(__mdata)
101    {
102        __data = .;
103        *(.data)
104        *(.data.*)
105        *(D)
106        *(D_1)
107        *(D_2)
108        __edata = .;
109    } > RAM
110    .rpfram ALIGN(4): AT(__PFRAM_start)
111    {
112        __RPFRAM_start = .;
113        *(PFRAM)
114        . = ALIGN(4);
115        __RPFRAM_end = .;
116    } > RAM
117    .gcc_exc :
118    {
119        *(.gcc_exc)
120    } > RAM

```

The screenshot shows a linker script editor with the file `linker_script.ld` open. The script defines several sections: `.ctors` (ROM), `.pfram` (ROM), `.data` (RAM), `.rpfram` (RAM), and `.gcc_exc` (RAM). Three sections are highlighted with red boxes: `.ctors` (lines 77-93), `.pfram` (lines 94-99), and `.rpfram` (lines 110-116). The `.pfram` and `.rpfram` sections are marked as `> ROM` and `> RAM` respectively, indicating they are placed in non-volatile memory for code flash re-writing.

2. Programs that operate during code flash re-writing such as interrupt callback function, etc. need to be placed in a FRAM section by specifying the FRAM section for each function.

```

__attribute__((section("PFRAM")))
/* Function that operates during code flash re-writing */
void func(void){...}

__attribute__((section("PFRAM")))
/* Callback function that operates during code flash re-writing */
void cb_func(void){...}

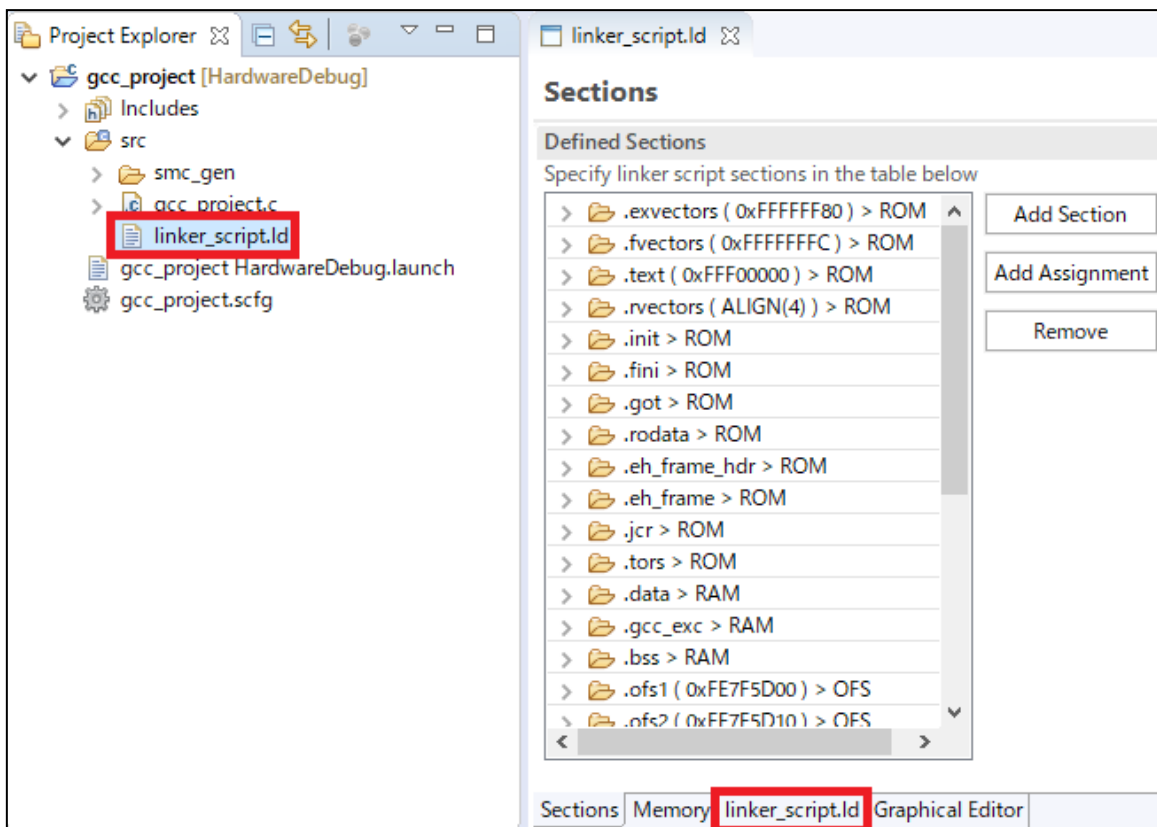
```

5.3.2.2 Programming Code Flash Using the Dual Bank Function

This section describes addition of linker settings and debugging with the dual bank function.

1. Add a setting in the linker settings file (linker_script.ld).

- (1) From Project Explorer, right-click the linker settings file (linker_script.ld), and select "Open".
- (2) On the linker_script.id window, click the "linker_script_id" tab.

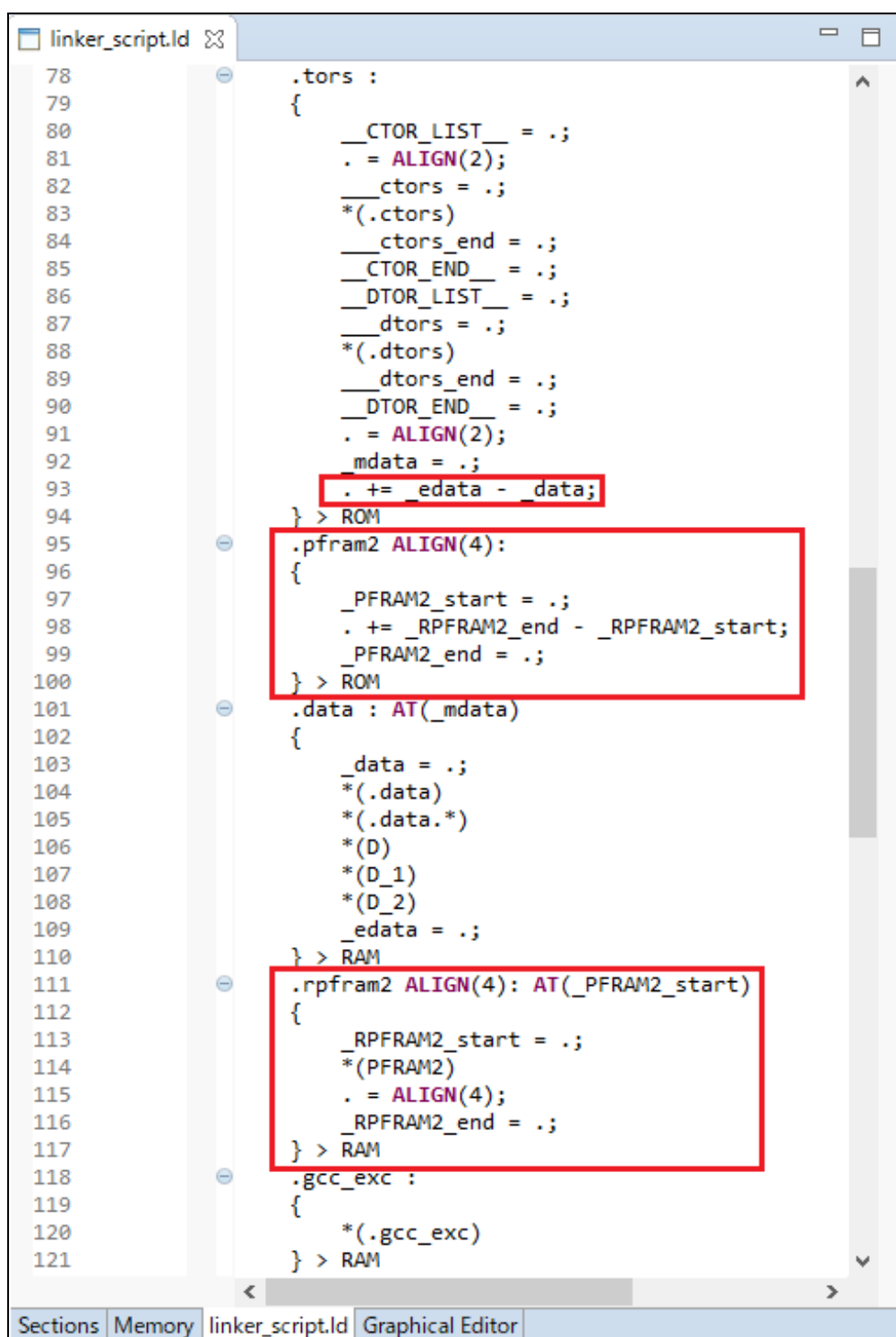


(3) Add the following (a) to (c) in the linker settings file (linker_script.ld).

- (a) `. += _edata - _data;`
- (b) `.pfram2 ALIGN(4):`

```
{
    _PFRAM2_start = .;
    . += _RPFRAM2_end - _RPFRAM2_start;
    _PFRAM2_end = .;
} > ROM
```
- (c) `.rpfram2 ALIGN(4): AT(_PFRAM2_start)`

```
{
    _RPFRAM2_start = .;
    *(PFRAM2)
    . = ALIGN(4);
    _RPFRAM2_end = .;
} > RAM
```

```
linker_script.ld
78      .ctors :
79      {
80          __CTOR_LIST__ = .;
81          . = ALIGN(2);
82          __ctors = .;
83          *(.ctors)
84          __ctors_end = .;
85          __CTOR_END__ = .;
86          __DTOR_LIST__ = .;
87          __dtors = .;
88          *(.dtors)
89          __dtors_end = .;
90          __DTOR_END__ = .;
91          . = ALIGN(2);
92          __mdata = .;
93          . += _edata - _data;
94      } > ROM
95      .pfram2 ALIGN(4):
96      {
97          __PFRAM2_start = .;
98          . += _RPFRAM2_end - _RPFRAM2_start;
99          __PFRAM2_end = .;
100     } > ROM
101     .data : AT(__mdata)
102     {
103         __data = .;
104         *(.data)
105         *(.data.*)
106         *(D)
107         *(D_1)
108         *(D_2)
109         __edata = .;
110     } > RAM
111     .rpfram2 ALIGN(4): AT(__PFRAM2_start)
112     {
113         __RPFRAM2_start = .;
114         *(PFRAM2)
115         . = ALIGN(4);
116         __RPFRAM2_end = .;
117     } > RAM
118     .gcc_exc :
119     {
120         *(.gcc_exc)
121     } > RAM
```

The screenshot shows a linker script editor with the file `linker_script.ld` open. The script contains several sections, with three specific blocks highlighted by red rectangles:

- Block 1 (Lines 92-93):** `. += _edata - _data;`
- Block 2 (Lines 95-100):** `.pfram2 ALIGN(4):` section, including `__PFRAM2_start`, `__PFRAM2_end`, and alignment logic.
- Block 3 (Lines 111-116):** `.rpfram2 ALIGN(4): AT(__PFRAM2_start)` section, including `__RPFRAM2_start`, `__RPFRAM2_end`, and alignment logic.

The editor interface includes a left sidebar with a tree view, a main text area with line numbers, and a bottom tab bar with tabs for `Sections`, `Memory`, `linker_script.ld`, and `Graphical Editor`.

2. The following describes how to load objects for two banks when connecting an application related to the dual bank function to a target device and perform debugging. Perform this procedure as required.

- (1) In Project Explorer, click the project you want to debug.
- (2) Click Execute > Debug Configuration to open the Debug Configuration window.
- (3) On the Debug Configuration window, expand the display of the “Renesas GDB Hardware Debugging” debug configuration and click the debug configuration you want to debug.
- (4) Switch to the “Startup” tab, and click the [Add...] button of “Load image and symbols” on the “Startup” tab.
- (5) On the Edit Download Module window, specify an object for the other startup bank and then click the [OK] button.
- (6) Select a “Load type”.

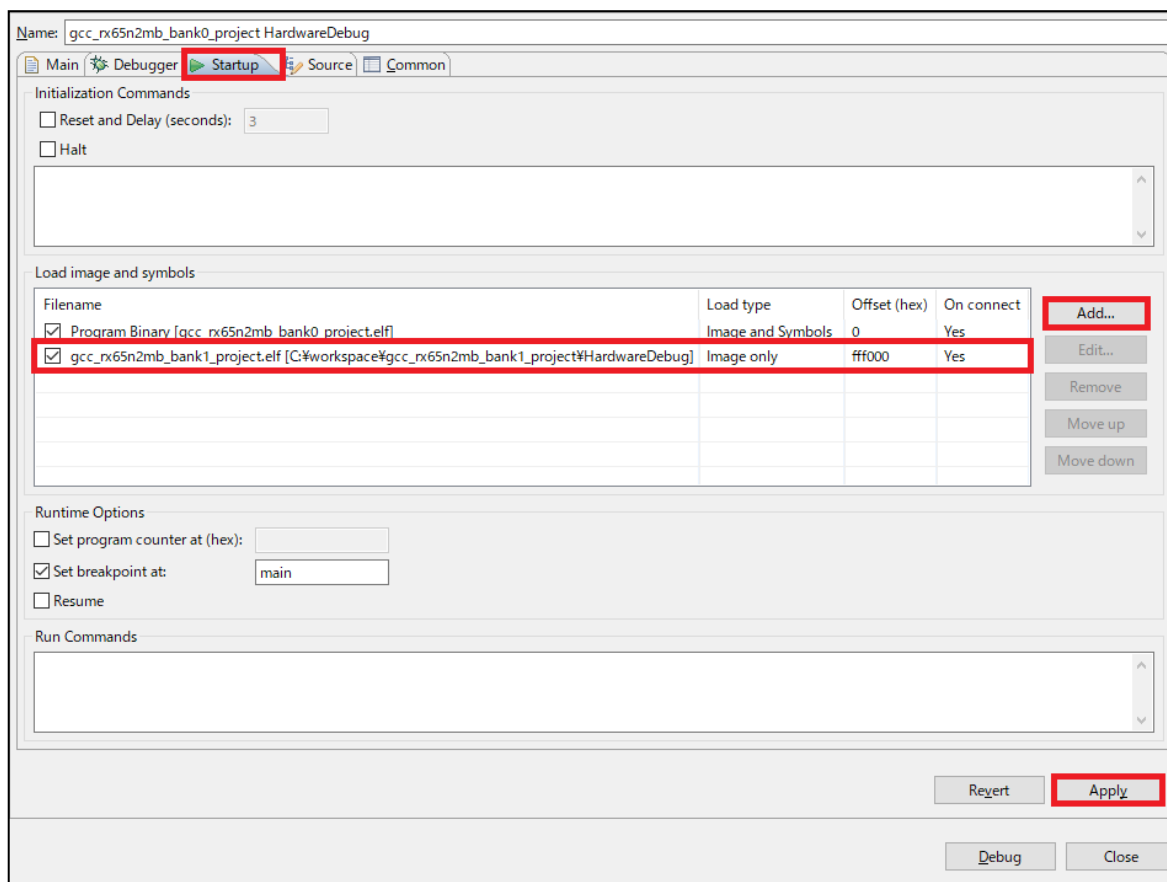
e²studio can only maintain one debug symbol table at a time. You can set the load type of either of the applications to “Image and Symbols”.

- (7) Specify “Offset”.

The “Offset” specification differs depending on the capacity of the code flash memory. For details, refer to the User's Manual for each device used.

The following shows an example when the target device is RX65N and the code flash capacity is 2MB. For “Offset”, enter fff00000, for which -1MB is two's complement. By doing so, the application to be assigned to the other startup bank will be loaded into memory 1Mb lower than the values shown in the linker or map file.

- (8) Click the [Apply] button.



5.3.3 Using IAR C/C++ Compiler for Renesas RX

This section describes how to use IAR C/C++ Compiler for Renesas RX as the compiler.

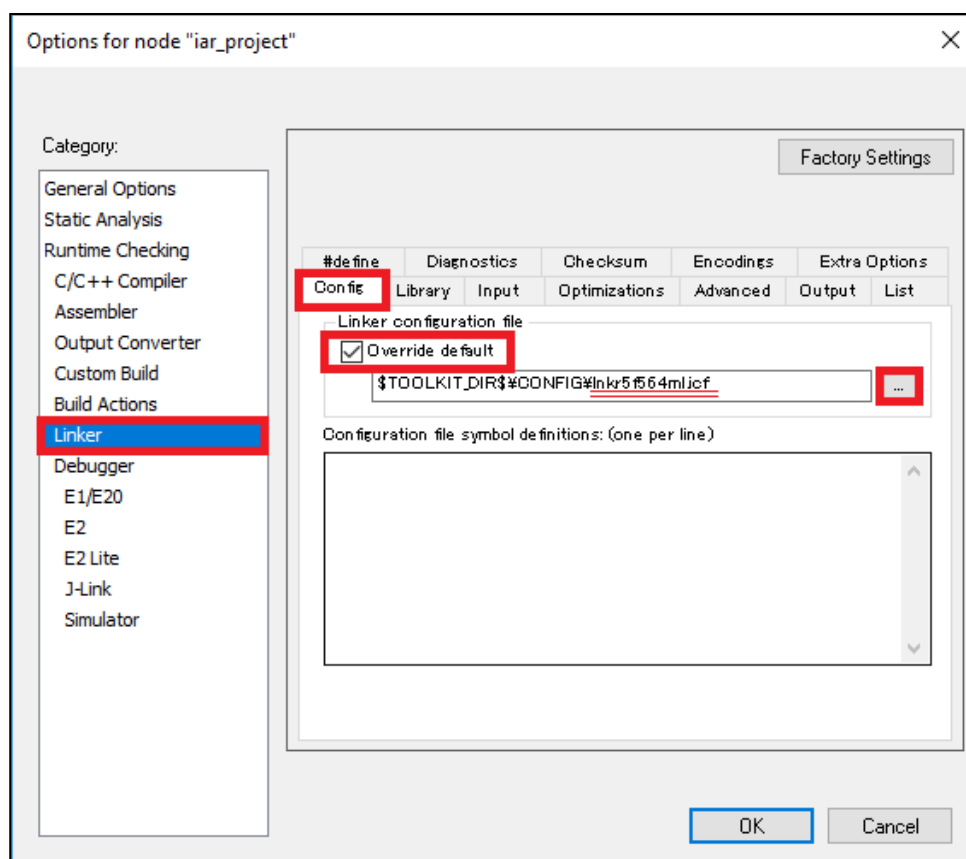
- Using the FIT Module Importer of IAR Embedded Workbench
FIT Module Importer of IAR Embedded Workbench is used to generate and use a project for IAR to which this FIT module or BSP is added. For details of FIT Module Importer, refer to the latest information on the IAR website.
- Using the IAR Project Converter of IAR Embedded Workbench
IAR Project Converter of IAR Embedded Workbench is used to convert a project created for CCRX to a project for IAR before the project is used. For details of IAR Project Converter, refer to the latest information on the IAR website. In addition, how to convert a project created for CCRX to a project for IAR is also described in the application note "RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)".

To use this FIT module for a project for IAR, the following settings are required.

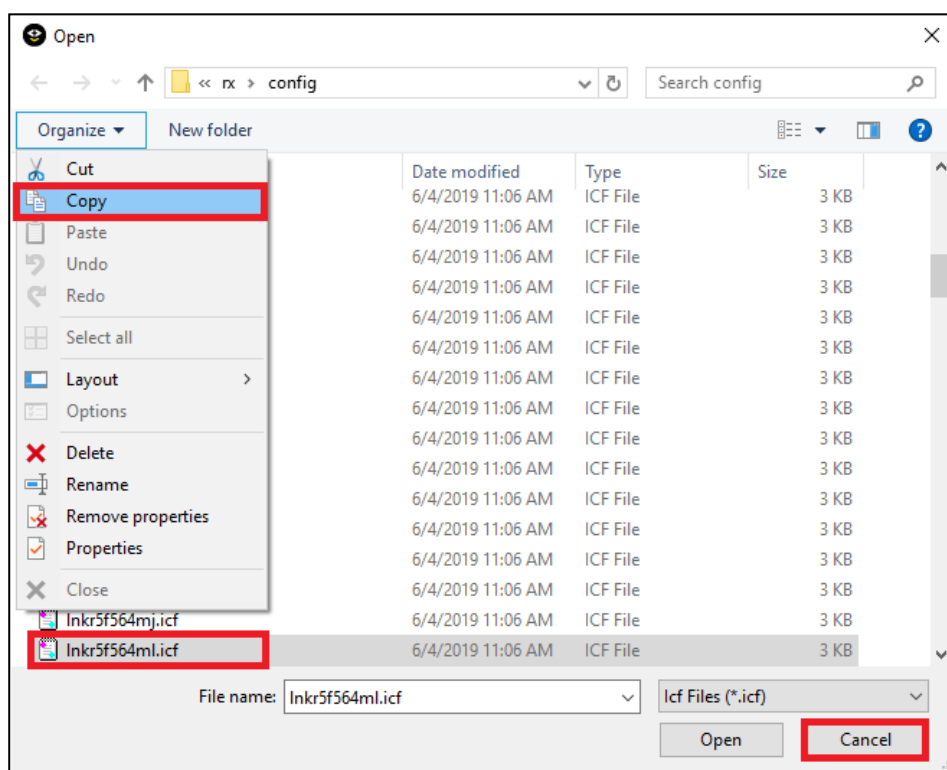
5.3.3.1 Programming Code Flash from RAM

This section describes addition of linker settings and placement of programs that operate during code flash re-writing.

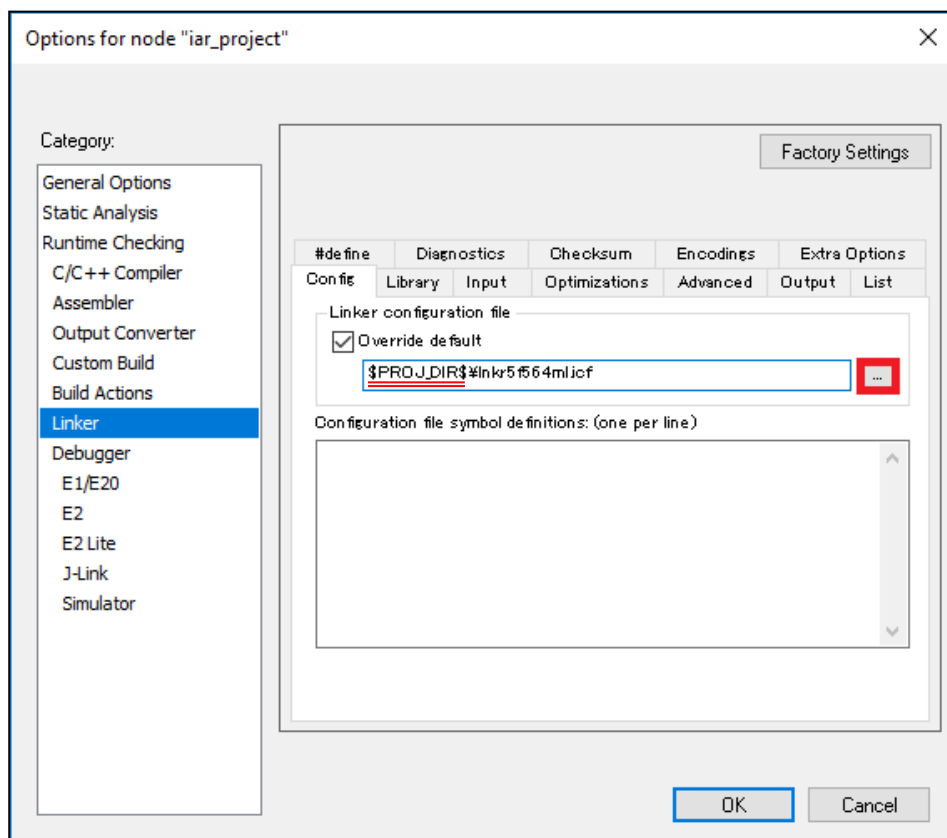
1. Open the Options window of the project for IAR, select "Linker" under "Category:", and select the "Config" tab. Then, after confirming that the "Override default" check box has been selected, click the [...] button.



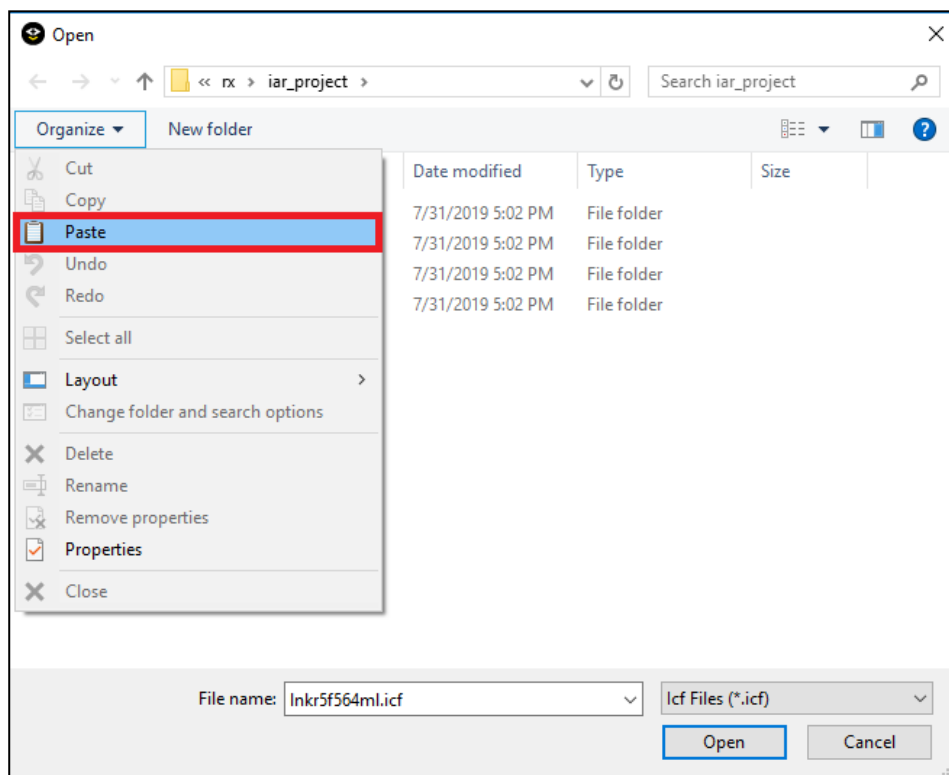
2. On the Open window, copy the .icf file of the target device (the double underlined part of the text box of the linker settings file in the Options window of step 1.), and click the [Cancel] button.



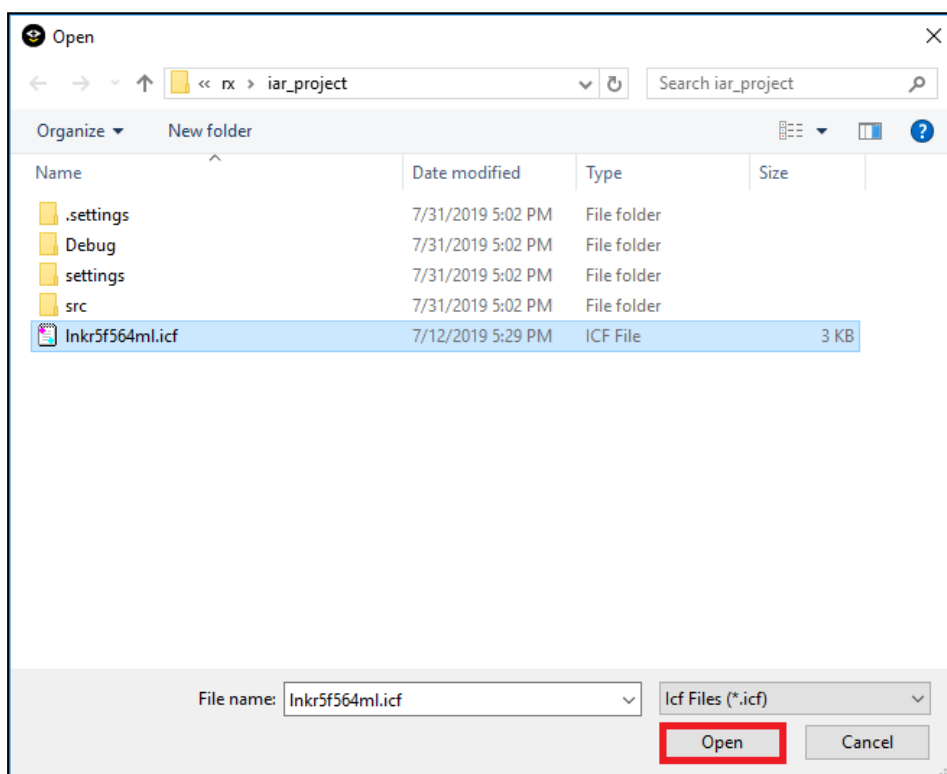
3. Re-write the path to the linker settings file on the Options window to the desired location. (In this example, "\$PROJ_DIR\$" is used as the path variable to place the file directly under the project folder.) After re-writing, click the [...] button again.



4. On the Open window, paste the .icf file of the target device copied in 2 above. (In this example, it is pasted directly under the project folder)



Click the [Open] button.



Now, the default linker settings file was copied and it is ready to edit the copied linker settings file.

5. Copy the following (a) to (d) to add to the replaced linker settings file.

- (a) `initialize manually { rw section .textrw, section PFRAM };`
- (b) `define block PFRAM with alignment = 4 { section PFRAM };`
`define block PFRAM_init with alignment = 4 { section PFRAM_init };`
- (c) `"ROM32":place in ROM_region32 { ro,`
`block PFRAM_init };`
- (d) `"RAM32":place in RAM_region32 { rw,`
`ro section D,`
`ro section D_1,`
`ro section D_2,`
`block PFRAM,`
`block HEAP };`

```

Inkr5f564ml.icf x
//-----
// Linker configuration file template for the Renesas RX microcontroller R5F564ML
//-----
// Compatibility check
define exported symbol __link_file_version_4 = 1;

define memory mem with size = 4G;

define region RAM_region16 = mem:[from 0x00000004 to 0x00007FFF];
define region RAM_region24 = mem:[from 0x00000004 to 0x0007FFFF];
define region RAM_region32 = mem:[from 0x00000004 to 0x0007FFFF];

define region ROM_region16 = mem:[from 0xFFFF8000 to 0xFFFFFFFF];
define region ROM_region24 = mem:[from 0xFFC00000 to 0xFFFFFFFF];
define region ROM_region32 = mem:[from 0xFFC00000 to 0xFFFFFFFF];

define region DATA_FLASH = mem:[from 0x00100000 to 0x0010FFFF];

initialize manually { rw section .textrw, section PFRAM };
initialize by copy { rw, ro section D, ro section D_1, ro section D_2 };
initialize by copy with packing = none { section __DLIB_PERTHREAD };
do not initialize { section .*.noinit };

define block HEAP with alignment = 4, size = _HEAP_SIZE { };
define block USTACK with alignment = 4, size = _USTACK_SIZE { };
define block ISTACK with alignment = 4, size = _ISTACK_SIZE { };

define block PFRAM with alignment = 4 { section PFRAM };
define block PFRAM_init with alignment = 4 { section PFRAM_init };

define block STACKS with fixed order { block USTACK,
                                         block ISTACK };

place at address mem:0x00120040 { ro section .option_mem };

place at address mem:0xFFFFFFFFFC { ro section .resetvect };
place at address mem:0xFFFFFFFF80 { ro section .exceptvect };

"ROM16":place in ROM_region16 { ro section .code16*,
                                ro section .data16* };
"RAM16":place in RAM_region16 { rw section .data16*,
                                rw section __DLIB_PERTHREAD };
"ROM24":place in ROM_region24 { ro section .code24*,
                                ro section .data24* };
"RAM24":place in RAM_region24 { rw section .data24* };
"ROM32":place in ROM_region32 { ro,
                                block PFRAM_init };
"RAM32":place in RAM_region32 { rw,
                                ro section D,
                                ro section D_1,
                                ro section D_2,
                                block PFRAM,
                                block HEAP };

"STACKS":place at end of RAM_region32 { block STACKS };

```

6. Programs that operate during code flash re-writing such as interrupt callback function, etc. need to be placed in a FRAM section by specifying the FRAM section for each function.

```
#pragma location="PFRAM"  
/* Function that operates during code flash re-writing */  
void func(void){...}  
  
#pragma location="PFRAM"  
/* Callback function that operates during code flash re-writing  
void cb_func(void){...}
```

5.3.3.2 Programming Code Flash Using the Dual Bank Function

This section describes addition of linker settings.

After performing items 1. to 4. in section 5.3.3.1, perform the following settings.

1. Copy the following (a) to (e) to change and add to the replaced linker settings file.

- (a) Changes to the first address of bank 0 of dual mode.
 define region ROM_region24 = mem:[from 0xFF00000 to 0xFFFFFFFF];
 define region ROM_region32 = mem:[from 0xFF00000 to 0xFFFFFFFF];
- (b) initialize manually { rw section .textrw, section PFRAM2 };
- (c) define block PFRAM2 with alignment = 4 { section PFRAM2 };
 define block PFRAM2_init with alignment = 4 { section PFRAM2_init };
- (d) "ROM32":place in ROM_region32 { ro,
 block PFRAM2_init };
- (e) "RAM32":place in RAM_region32 { rw,
 ro section D,
 ro section D_1,
 ro section D_2,
 block PFRAM2,
 block HEAP };

```

Inkr5f565ne_dual.icf x
//-----
// Linker configuration file template for the Renesas RX microcontroller R5F565NE_DUAL
//-----
// Compatibility check
define exported symbol __link_file_version_4 = 1;

define memory mem with size = 4G;

define region RAM_region1 = mem:[from 0x00000004 to 0x0003FFFF];
define region RAM_region2 = mem:[from 0x00800000 to 0x0085FFFF];

define region RAM_region16 = mem:[from 0x00000004 to 0x00007FFF];
define region RAM_region24 = RAM_region1 | RAM_region2;
define region RAM_region32 = RAM_region1 | RAM_region2;

define region STANDBY_RAM = mem:[from 0x000A4000 to 0x000A5FFF];

define region ROM_region16 = mem:[from 0xFFFF8000 to 0xFFFFFFFF];
define region ROM_region24 = mem:[from 0xFF000000 to 0xFFFFFFFF];
define region ROM_region32 = mem:[from 0xFF000000 to 0xFFFFFFFF];

define region DATA_FLASH = mem:[from 0x00100000 to 0x00107FFF];

initialize manually { rw section .textrw, section PFRAM2 };
initialize by copy { rw, ro section D, ro section D_1, ro section D_2 };
initialize by copy with packing = none { section __DLIB_PERTHREAD };
do not initialize { section .*.noinit };

define block HEAP with alignment = 4, size = _HEAP_SIZE { };
define block USTACK with alignment = 4, size = _USTACK_SIZE { };
define block ISTACK with alignment = 4, size = _ISTACK_SIZE { };

define block PFRAM2 with alignment = 4 { section PFRAM2 };
define block PFRAM2_init with alignment = 4 { section PFRAM2_init };

define block STACKS with fixed order { block USTACK,
block ISTACK };

place at address mem:0xFE7F5D00 { ro section .option_mem };
place at address mem:0xFFFFF0FC { ro section .resetvect };
place at address mem:0xFFFFF080 { ro section .exceptvect };

"ROM16":place in ROM_region16 { ro section .code16*,
ro section .data16* };
"RAM16":place in RAM_region16 { rw section .data16*,
rw section __DLIB_PERTHREAD };
"ROM24":place in ROM_region24 { ro section .code24*,
ro section .data24* };
"RAM24":place in RAM_region24 { rw section .data24* };
"ROM32":place in ROM_region32 { ro,
block PFRAM2_init };
"RAM32":place in RAM_region32 { rw,
ro section D,
ro section D_1,
ro section D_2,
block PFRAM2,
block HEAP };

"STACKS":place at end of RAM_region1 { block STACKS };

```

6. Reference Documents

User's Manual: Hardware

The latest versions can be downloaded from the Renesas Electronics website.

Technical Update/Technical News

The latest information can be downloaded from the Renesas Electronics website.

User's Manual: Development Tools

RX Family C/C++ Compiler CC-RX User's Manual (R20UT3248)

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

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	July.24.14	—	First edition issued
1.10	Nov.13.14	1, 4 7	Added RX113 support. Updated “ROM to RAM” image.
1.11	Dec.11.14	—	Added RX64M to xml support file.
1.20	Dec.22.14	1, 4	Added RX71M support.
1.30	Aug.28.15	All 5, 10	Updated template. Added RX231 support Added flash type 3 code flash run-from-rom info. Fixed RX64M/71M erase boundary issue.
1.40	Sep.03.15	1, 4	Added RX23T support Fixed Big Endian bug in R_DF_Write_Operation() for Flash Type 1. Fixed FLASH_xF_BLOCK_INVALID values for Flash Type 3.
1.50	Nov.11.15	1, 4	Added RX130 support
1.51	Nov.11.15	—	Repackaged demo with BSP v3.10
1.60	Nov.17.15	1, 5 22, 25	Added RX24T support Added ROM cache support Fixed incorrect FLASH_CF_BLOCK_INVALID for RX210/21A/62N/630/63N/63T in code (Flash Type 2).
1.61	May.20.16	10, 11	Added erase/write/blankcheck BGO support for RX64M/71M Fixed lockbit enable/disable commands.
1.62	May.25.16	—	Added lockbit write/read BGO support for RX64M/71M
1.63	Jun.13.16	—	Fixed bug where large flash writes returned success when actually failed (improper timeout handling) on RX64M/71M
1.64	Aug.11.16	—	Fixed RX64M/71M bug where R_FLASH_Control (FLASH_CMD_STATUS_GET, NULL) always returned BUSY. Added #if to exclude ISR code when not in BGO mode.
1.70	Aug.11.16	1, 4-6, 8 —	Added RX651/RX65N support (Flash Type 4) Fixed bug in Flash Type 2 that caused erroneous blankcheck results.
2.00	Aug.17.16	1, 3, 4, 6-9	Added RX230 and RX24T support (Flash Type 1) Added configuration option for operation without FIT BSP. Inserted document sections 2.12.2 thru 2.12.4. Modified values for FLASH_CF_LOWEST_VALID_BLOCK and FLASH_CF_BLOCK_INVALID for Flash TYPE 1.
2.10	Dec.20.16	1, 5-7, 11, 13, 17, 19, 21, 23-26, 31-32	Added RX24U and RX24T-512 support (Flash Type 1) Fixed several minor bugs in all flash types and added more parameter checking. See History in r_flash_rx_if.h for complete list of changes.
3.00	Dec.21.16	8, 9	Merged code common to types 1, 3, and 4 and restructured high level code for cleaner operation. Modified ROM/RAM size tables.

Rev.	Date	Description	
		Page	Summary
3.10	Feb.17.17	5-7, 13-17, 26-28, 35	Added RX65N-2M support. Added sections 2.16 and 2.17.4. Added commands FLASH_CMD_BANK_xxx. Fixed potential "BUSY" return from Flash Type 1 API calls (potential bug with very slow flash). Added clearing of ECC flag during initialization of Flash Type 3.
3.20	Aug.11.17	1, 5, 10-14, 16, 36	Added RX130-512KB support. Added e ² studio v6.0.0 differences. Modified driver so mcu_config.h only necessary when not using BSP. Fixed bug in RX65N-2M dual mode operation where sometimes when running in bank 0, performing a bank swap caused application execution to fail.
3.30	Nov.1.17	10, 20, 19, 21, 32, 25	Added FLASH_ERR_ALREADY_OPEN. Added R_FLASH_Close(). Added Flash Type 2 set access window example Added Flash Type 2 blankcheck example.
3.40	Mar.8.18	1, 5, 6, 14, 14-15, 39-40	Added support for RX66T. Added support for new 256K and 384K RX111 and RX24T variants. Updated table numbers in Section 2.14. Added interrupt event enumeration in Section 2.15 Added demos for RDKRX63N, RSKRX66T, and two for RSKRX64M.
3.41	Nov.8.18	6, 31, 36	Added NON_CACHED Control() commands. Added document number of the application note accompanying the sample program of the FIT module to xml file.
3.42	Feb.12.19	38-41	Modified typos in sections 4.1 to 4.12.
3.50	Feb.26.19	1, 5, 6, 31, 41	Added support for RX72T. Added demo for RX72T. Fixed write failure bug in RX210 768K and 1M variants.

Rev.	Date	Description	
		Page	Summary
4.00	Apr.19.19	—	Added support for GCC/IAR compiler.
		1, 6	Deleted the following flash type 2 devices from the target device. RX210, RX21A, RX220, RX610, RX621, RX62N, RX62T, RX62G, RX630, RX631, RX63N, RX63T
		1	Deleted the following documents from Related Documents Adding Firmware Integration Technology Modules to e ² studio Adding Firmware Integration Technology Modules to CS+ Projects Renesas e ² studio Smart Configurator User Guide
		6	Deleted FLASH_CFG_USE_FIT_BSP. Deleted FLASH_CFG_FLASH_READY_IPL. Deleted FLASH_CFG_IGNORE_LOCK_BITS. Added the explanation of FLASH_CFG_DATA_FLASH_BGO. Added the explanation of FLASH_CFG_CODE_FLASH_BGO.
		7-10	Updated “2.9 Code Size” section.
		11	Deleted the following return values, which are no longer necessary, from “2.11 Return Values” section. FLASH_ERR_ALIGNED FLASH_ERR_BOUNDARY FLASH_ERR_OVERFLOW
		12	Updated “2.12 Adding the FIT FLASH Module to Your Project” section. Added “2.13 Usage Combined with Existing User Projects” section.
		13	Revised and updated as follows the structure of “2.14 Programming Code Flash from RAM” section. “2.14.1 Using Renesas Electronics C/C++ Compiler Package for RX Family”, “2.14.2 Using GCC for Renesas RX”, “2.14.3 Using IAR C/C++ Compiler for Renesas RX”
		22	Added “2.18.4 Emulator Debug Configuration” section.

Rev.	Date	Description	
		Page	Summary
4.00	Apr.19.19	Program	<p>Changed as a result of supporting the GCC/IAR compiler.</p> <p>Changed as a result of deletion of FLASH_CFG_USE_FIT_BSP.</p> <p>Changed as a result of deletion of FLASH_CFG_FLASH_READY_IPL.</p> <p>Changed as a result of deletion of FLASH_CFG_IGNORE_LOCK_BITS.</p> <p>Deleted flash type 2 device from target device.</p> <p>Deleted FLASH_ERR_ALIGNED.</p> <p>Deleted FLASH_ERR_BOUNDARY.</p> <p>Deleted FLASH_ERR_OVERFLOW.</p> <p>Added the process to output error when BSP is earlier than Rev.5.00.</p>
4.10	Jun.07.19	1, 5 7-11 17-18 48 49-50	<p>Added support for RX23W.</p> <p>Updated "2.9 Code Size" section.</p> <p>Updated "2.14.2 Using GCC for Renesas" section.</p> <p>Added "5. Appendices" section.</p> <p>Added "5.1 Confirmed Operation Environment" section.</p> <p>Added "5.2 Troubleshooting" section.</p>
		Program	<p>Added support for RX23W.</p> <p>Modified FEARL and FSARL register settings.</p> <p>Updated the demo project environment.</p>
4.20	Jul.19.19	1, 5 47 50	<p>Added support for RX72M.</p> <p>Added "4.13 flash_demo_rskrx72m_bank0_bootapp / _bank1_otherapp" section.</p> <p>Updated "5.1 Confirmed Operation Environment" section.</p>
		Program	<p>Added support for RX72M.</p> <p>Added the RX72M demo project.</p> <p>Updated the demo project environment.</p> <p>Deleted the warning.</p> <p>Deleted definitions and include, which are no longer used.</p> <p>Granted the volatile declaration to global variables.</p> <p>Modified the section related to dual mode and linear mode.</p> <p>Modified part of Flash Type 4 timeout processing.</p>

Rev.	Date	Description	
		Page	Summary
4.30	Sep.09.19	1, 6 5 7-11 14	Added support for RX13T. Added "2.5 Interrupt Vectors" section. Updated "2.10 Code Size" section. Modified the following descriptions and moved to "5.3 Compiler-Dependent Settings". "2.14.1 Using Renesas Electronics C/C++ Compiler Package for RX Family" "2.14.2 Using GCC for Renesas RX" "2.14.3 Using IAR C/C++ Compiler for Renesas RX"
		15	Modified "2.18 Dual Bank Operation" section and moved the content that depends on the compiler to "5.3 Compiler-Dependent Settings".
		42 45-62	Updated "5.1 Confirmed Operation Environment" section. Added "5.3 Compiler-Dependent Settings" section.
		Program	Added support for RX13T. Modified part of the flash type 1 error processing. Modified the copy method of R_FlashCodeCopy() when using IAR. Modified the implementation of r_flash_control() to the if_then method.
4.40	Sep.27.19	1, 5, 6 23	Added support for RX23E-A. Added FLASH_ERR_NULL_PTR to Return Values in "3.5 R_FLASH_BlankCheck()" section.
		42	Updated "5.1 Confirmed Operation Environment" section.
		Program	Added support for RX23E-A. Added the NULL check of the 3rd argument of r_flash_blankcheck().
4.50	Nov.18.19	1, 5, 6 5 15 16 21-37	Added support for RX66N and RX72N. Added limitations to "2.3 Limitations" section. Added "2.13 Blocking Mode and Non-blocking Mode" section. Deleted "2.17 Operations in BGO Mode" section. Deleted description of Reentrant from "3.2 R_FLASH_Open()", "3.3 R_FLASH_Close()", "3.4 R_FLASH_Erase()", "3.5 R_FLASH_BlankCheck()", "3.6 R_FLASH_Write()", "3.7 R_FLASH_Control()", and "3.8 R_FLASH_GetVersion()" sections.
		29-32	Modified the content of Description in "3.7 R_FLASH_Control()" section.
		45	Updated "5.1 Confirmed Operation Environment" section.
		Program	Added support for RX66N and RX72N. Supported Doxygen. Modified enabling and disabling IEN to use R_BSP_InterruptRequestEnable() and R_BSP_InterruptRequestDisable().

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/.