# RX Family

## Flash Module Using Firmware Integration Technology

R01AN2184EU0110
Rev.1.10
Nov.13, 2014

## Introduction

The Renesas Flash FIT Module has been created to allow users of RX devices to easily integrate reprogramming abilities into their applications using User Mode programming. User Mode programming is the term used to describe a Renesas MCU's ability to reprogram its own internal flash memory while running in its normal operational mode. This application note focuses on using that API and integrating it with your application program.

The API is different from the Simple Flash API that supports the RX600 and the RX200 series of MCUs.

## Target Device

The following is a list of devices able to use this API:

- **RX110, RX111, RX113 Groups**
- **RX610 Group**
- **RX621, RX62N, RX62T, RX62G Groups**
- **RX630, RX631, RX63N, RX63T Groups**
- **RX210 Group**
- **RX21A Group**
- **RX220 Group**
- **RX64M Group**

## Related Documents

- Firmware Integration Technology User's Manual (R01AN1833EU)
- Board Support Package Firmware Integration Technology Module (R01AN1685EU)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)
- Adding Firmware Integration Technology Modules to CubeSuite+ Projects (R01AN1826EJ)
- Using the Simple Flash API for RX without the r_bsp Module (R01AN1890EU)

## Contents

# 1. Overview

This 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 API in its simplest form can be used to perform blocking erase and program operations. The term 'blocking' means that when a program or erase function is called, the function does not return until the operation has finished. When a code flash operation is on-going, that code flash area cannot be accessed by the user. If an attempt to access the code flash area is made, the flash control unit will transition into an error state. For this reason 'blocking' operations are preferred by some users to prevent the possibility of a flash error The Data Flash area can be accessed at any time.

## 1.1 Features

Below is a list of the features supported by the Flash API.

- Blocking and non-blocking erasing, programming, and blank checking of both code and data flash
- Access window control allowing only specified areas of code flash to be erased or written (RX100 Series)
- Swap block functionality is available (in products with a 32-Kbyte or larger ROM) which allows safe re-writing of the startup program without first erasing it.

## 1.2 BSP

This Flash API module is written as a Firmware Integration Technology (FIT) module and is intended to be used in conjunction with a Renesas Board Support Package (r_bsp). Though not recommended, some users will wish to use the Flash API for RX without the r_bsp. Those users will need to create a u_bsp module which is detailed in the application note: Using the Simple Flash API for RX without the r_bsp Module (R01AN1890EU).

# 2. API Information

This driver follows the Renesas API naming standards.

## 2.1 Hardware Requirements

This driver requires an RX100, RX200, or RX600 Series MCU.

## 2.2 Software Requirements

This software is intended to be used with a Renesas Board Support Package, although as described earlier it is possible to use it without one. Currently the driver is dependent on the following board support package:

- Renesas Board Support Package (r_bsp) v2.70 or higher.

## 2.3 Supported Toolchains

This driver is tested and working with the following toolchains:

- Renesas RX Toolchain v2.01.00

## 2.4 Header Files

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

## 2.5 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*.

## 2.6 Configuration Overview

Configuring this middleware 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.

| Configuration Options in *r_flash_rx_config.h* | | |
|---|---|---|
| **Equate** | **Default Value** | **Description** |
| **FLASH_CFG_PARAM_CHECKING_ENABLE** | **0** | Setting to 1 includes parameter checking. Setting to 0 compiles out parameter checking.<br>Setting to BSP_CFG_PARAM_CHECKING_ENABLE utilizes the system default setting. |
| **FLASH_CFG_CODE_FLASH_ENABLE** | **0** | Setting to 1 configures the API such that code is included to program the User ROM program area of Flash. Since this code must be executed from within RAM, the sections 'PFRAM'(ROM) and 'RPFRAM'(RAM) must be added to the linker settings. Also the linker option:<br>'-rom=PFRAM=RPFRAM' must be added.<br>If this macro is set to 0 then the user does not need to setup and initialize the PFRAM and RPFRAM sections.<br>If you are only using Data Flash set this to 0. |
| **FLASH_CFG_DATA_FLASH_BGO** | **0** | Setting this to 0 will cause Data Flash operations to block and not return until the operation is complete.<br>Setting to 1 places the API in 'BGO' mode. In this mode the Flash API routines that deal with Data Flash will return after the operation has been started<br>instead of blocking until it is complete. For MCU devices that support a Flash interrupt (e.g. RX63N), notification of the operation completion will be done via the interrupt. For devices that do not support a Flash interrupt (e.g.. RX111), the completion must be polled for using a Control command. Set to 1 to operate in polled or BGO mode. |
| **FLASH_CFG_CODE_FLASH_BGO** | **0** | Setting this to 0 will cause Code Flash (ROM) operations to block and not return until the operation is complete. Setting to 1 places the API in 'BGO' mode. In this mode the Flash API routines<br>that deal with ROM will return after the operation has been started instead of blocking until it is complete. For MCU devices that support a Flash<br>interrupt (e.g. RX63N), notification of the operation completion will be done via the interrupt. For devices that do not support a Flash interrupt |

| | | |
|---|---|---|
| | | (e.g. RX111), the completion must be polled for using a Control command. |
| **FLASH_CFG_FLASH_READY_IPL** | **5** | For MCU devices that support a Flash interrupt (e.g. RX63n), this defines the interrupt priority level for that interrupt. |
| **FLASH_CFG_IGNORE_LOCK_BITS** | **1** | This applies only to MCU's that support lock bits (e.g. RX63N), and only to ROM as Data Flash does not support lock bits. Each erasure block has a corresponding lock bit that can be used to protect that block from being programmed/erased after the lock bit is set. The use of lock bits can be used or ignored.<br>Setting this to 1 will cause lock bits to be ignored and programs/erases to a block will not be limited.<br>Setting this to 0 will cause lock bits to be used as the user configures through the Control command. |

**Table 1 : Flash API Configuration Items**

## 2.7 Code Size

The code size is based on optimization level for 2 and optimization type for size for the toolchain used. The maximum and minimum values are determined by the build-time configuration options set in the module configuration header file.

| RX111/RX113 ROM and RAM code size | |
|---|---|
| `FLASH_CFG_PARAM_CHECKING_ENABLE = 1`<br>`FLASH_CFG_CODE_FLASH_ENABLE = 1`<br>`FLASH_CFG_CODE_FLASH_BGO = 1` | ROM: 2882 bytes |
| | RAM: 40 bytes |
| `FLASH_CFG_PARAM_CHECKING_ENABLE = 0`<br>`FLASH_CFG_CODE_FLASH_ENABLE = 1`<br>`FLASH_CFG_CODE_FLASH_BGO = 1` | ROM: 2811 bytes |
| | RAM: 36 bytes |
| `FLASH_CFG_PARAM_CHECKING_ENABLE = 0`<br>`FLASH_CFG_CODE_FLASH_ENABLE = 1`<br>`FLASH_CFG_CODE_FLASH_BGO = 0` | ROM: 2948 bytes |
| | RAM: 36 bytes |
| `FLASH_CFG_PARAM_CHECKING_ENABLE = 0`<br>`FLASH_CFG_CODE_FLASH_ENABLE = 0`<br>`FLASH_CFG_CODE_FLASH_BGO = 0` | ROM: 1928 bytes |
| | RAM: 24 bytes |

| RX110 ROM and RAM code size | |
|---|---|
| `FLASH_CFG_PARAM_CHECKING_ENABLE = 1`<br>`FLASH_CFG_CODE_FLASH_BGO = 1` | ROM: 1965 bytes |
| | RAM: 16 bytes |

| | |
|---|---|
| **FLASH_CFG_PARAM_CHECKING_ENABLE = 0**<br>**FLASH_CFG_CODE_FLASH_BGO = 1** | ROM: 1894 bytes |
| | RAM: 16 bytes |
| **FLASH_CFG_PARAM_CHECKING_ENABLE = 0**<br>**FLASH_CFG_CODE_FLASH_BGO = 0** | ROM: 2003 bytes |
| | RAM: 16 bytes |

## 2.8 Return Values

This shows the different values API functions can return. These definitions are all defined in the enumeration *flash_err_t* in *r_flash_rx_if.h*..

```
/* Flash API error codes */
typedef enum _flash_err
{
    FLASH_SUCCESS = 0,
    FLASH_ERR_BUSY,             // Peripheral Busy
    FLASH_ERR_ACCESSW,          // Access window error
    FLASH_ERR_FAILURE,          // Operation failure, programming or erasing or
                                // error due to something other than lock bit
    FLASH_ERR_CMD_LOCKED,       // RX64M - Peripheral in command locked state
    FLASH_ERR_LOCKBIT_SET,      // RX64M - Pgm/Erase error due to lock bit.
    FLASH_ERR_FREQUENCY,        // RX64M - Illegal Frequency value attempted (4-60Mhz)
    FLASH_ERR_ALIGNED,          // RX600/RX200 - The address that was supplied was not
                                // on aligned correctly for code flash or data flash
    FLASH_ERR_BOUNDARY,         // RX600/RX200 - Writes cannot cross the 1MB boundary
                                // on some parts
    FLASH_ERR_OVERFLOW,         // RX600/RX200 - Address + number of bytes' for this
                                // operation went past the end of this memory area.
    FLASH_ERR_BYTES,            // Invalid number of bytes passed
    FLASH_ERR_ADDRESS,          // Invalid address or address not on a programming
                                // boundary.
    FLASH_ERR_BLOCKS,           // The "number of blocks" argument is invalid
    FLASH_ERR_PARAM,            // Illegal parameter
    FLASH_ERR_NULL_PTR,         // received null ptr; missing required argument
    FLASH_ERR_TIMEOUT,          // Timeout Condition
 } flash_err_t;
```

## 2.9 Adding Middleware to Your Project

The driver must be added to an existing e2studio project. It is best to use the e2studio FIT plugin to add the driver to your project as that will automatically update the include file paths for you. Alternatively, the driver can be imported from the archive that accompanies this application note and manually added by following these steps:
Follow the steps below to add the middleware's code to your project.

1. This application note is distributed with a zip file package that includes the FLASH FIT module in its own folder r_flash_rx.
2. Unzip the package into the location of your choice.
3. In a file browser window, browse to the directory where you unzipped the distribution package and locate the r_flash_rx folder.
4. Open your e2studio workspace.
5. In the e2studio project explorer window, select the project that you want to add the FLASH module to.
6. Drag and drop the r_flash_rx folder from the browser window (or copy/paste) into your e2studio project at the top level of the project.
7. Update the source search/include paths for your project by adding the paths to the module files:
    a. Navigate to the "Add directory path" control:

i. 'project name'->properties->C/C++ Build->Settings->Compiler->Source -Add (green + icon)

b. Add the following paths:

i. "${workspace_loc:/${ProjName}/r_flash_rx}"
ii. "${workspace_loc:/${ProjName}/r_flash_rx/src}"
iii. "${workspace_loc:/${ProjName}/r_flash_rx/src/targets}"
iv. "${workspace_loc:/${ProjName}/r_flash_rx/src/non_fcu/rx100}" (e.g.RX111)
v. "${workspace_loc:/${ProjName}/r_flash_rx/src/fcu}" (e.g. RX63N)

It is necessary to configure the module for your application before it can be used.

8. Locate the r_flash_rx_config_reference.h file in the r_flash_rx/ref/ folder in your project and copy it to your project's r_config folder.
9. Change the name of the copy in the r_config folder to r_flash_rx_config.h
10. Make the required configuration settings by editing the copied r_flash_rx_config.h file. See Configuration Overview.
11.

The FLASH module uses the r_bsp package for certain MCU information and support functions. The r_bsp package is easily configured through the platform.h header file which is located in the r_bsp folder. To configure the r_bsp package, open up platform.h and uncomment the #include for the board you are using. To run the demo on a RSKRX11X board, the user would uncomment the #include for ‘./board/rskRX11x/r_bsp.h’ macro and make sure all other board #includes are commented out.

The following steps are only required if you are programming or erasing ROM. If you are only operating on data flash, then these steps can be ignored. These steps are discussed with more detail in Section 2.11.

1. Make a ROM section named ‘PFRAM’.
2. Make a RAM section named ‘RPFRAM’.
3. Configure your linker such that code allocated in the ‘FRAM’ section will actually be executed in RAM.

## 2.10 Limitations

1. This code is not re-entrant but does protect against multiple concurrent function calls.
2. During ROM operations ROM cannot be accessed. If using ROM and BGO/polling then make sure application code runs from RAM.

## 2.11 Putting Flash API Code in RAM

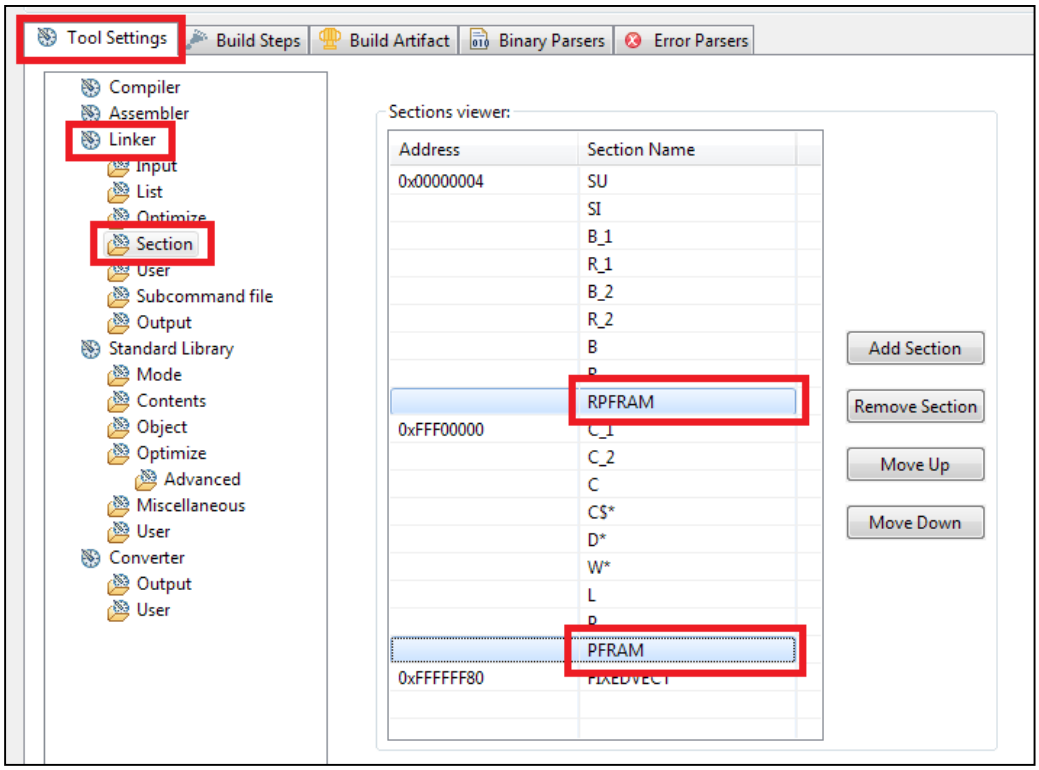
The MCUs require that sections in RAM and ROM be created to hold the API functions for reprogramming ROM. This is required because the flash control circuit cannot program or erase ROM while executing or reading from ROM. Also, the RAM section will need to be initialized after reset.

In order to enable ROM operations, configure the FLASH_CFG_CODE_FLASH_ENABLE to “1” in the file *r_flash_rx_config.h*. Note that this is only for ROM programming. Please follow the steps below if you are programming or erasing ROM:

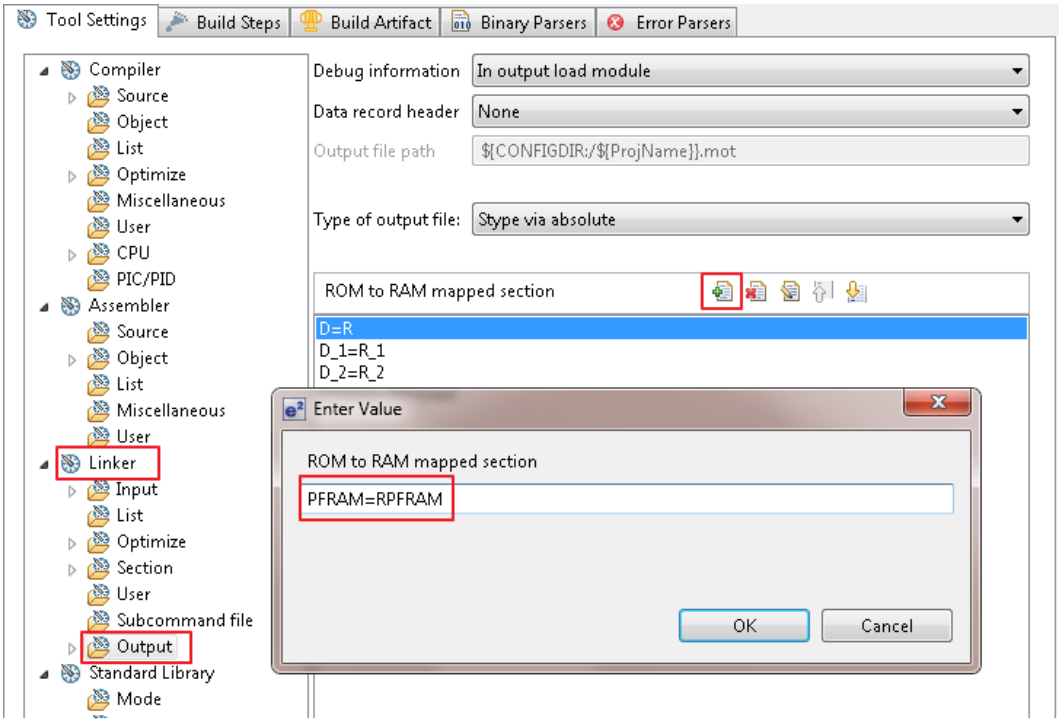
**In e2studio:**

The process of setting up the linker sections and mapping ROM to RAM needs to be done in e2studio as listed below.

1. Add a new section titled ‘**RPFRAM**’ in a RAM area.
2. Add a new section titled ‘**PFRAM**’ in a ROM area.




3. Add the linker option to map the ROM section (PFRAM) address to RAM section address (RPFRAM) by adding 'PFRAM=RPFRAM' to the Output options as seen below. This is done using the Linker >> Output section of the Tool Settings in E2Studio.




4. The linker is now setup to correctly allocate the appropriate Flash API code to RAM. Now we need to make sure that the code gets copied from ROM to RAM after reset. This is done automatically on calling the R_FLASH_Open() function. If this is not done before a Flash API function is called then the MCU will jump to uninitialized RAM.

## 2.12 Using Non-Blocking Background Operations

When operating in polling/BGO mode, API function calls will not block and will return before the flash operation has finished. The user should take care in these instances that they do not try to access the flash area that is being operated on until the operation has finished. If the area is accessed during an operation then the flash circuit will go into an error state and the operation will fail.

In order to determine when an operation has completed, the status for that operation must be periodically polled for using the by calling the R_FLASH_Control() function and issuing a FLASH_CMD_STATUS_GET command.

The Flash driver will reset the flash control circuit when an error is detected and call the callback with the appropriate event to alert the user that the flash operation did not complete successfully.

# 3. Usage Notes

## 3.1 Data Flash BGO Precautions

When using data flash in BGO/polling mode, the User ROM, RAM, and external memory can still be accessed. This means that care should be taken to make sure that the data flash is not accessed during a data flash operation. This includes interrupts that may access the data flash.

## 3.2 Code Flash BGO Precautions

When using code flash, external memory and RAM can still be accessed. Since most users will put their code in ROM, extra care should be taken compared to performing BGO/polling data flash operations. Since the API code will return before the ROM operation has finished, the code that calls the API function will need to be in RAM, and the code will need to poll for completion before issuing another Flash command. Note that this includes reading the Unique ID with another FIT Module (R01AN2191EJ).

Another important issue to be aware of is the relocatable vector table. The vector table by default resides in the User ROM. If an interrupt occurs during the ROM operation then ROM 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 service routines that may occur outside of ROM. The user will also need to change the relocatable vector table's pointer register (INTB). Examples of this are shown in the example workspace that comes with this application note.

## 3.3 Interrupts

Code flash or data flash areas cannot be accessed while a flash operation is on-going for that particular memory area. Therefore, care should be taken to ensure that no interrupts of any kind occur during code flash operations (no code executed in Flash). These precautions apply whether the user is using BGO/polling operations or not.

# 4. API Functions

## 4.1 Summary

The following functions are included in this API:

| Function | Description |
|---|---|
| **R_FLASH_Open()** | Initializes the Flash API. |
| **R_FLASH_Erase()** | Erases an entire flash block. |
| **R_FLASH_Write()** | Write data to ROM or data flash. |
| **R_FLASH_BlankCheck()** | Check if a data flash address (or block) is erased. |
| **R_FLASH_Control()** | Configure lock-bits, configuration data, flash clock, suspend/resume etc |
| **R_FLASH_GetVersion()** | Get the current version of this API. |

## 4.2 R_FLASH_Open

The function initializes the Flash API. This function **must** be called before using the rest of the API.

### Format

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

### Parameters

*none*

### Return Values

```
FLASH_SUCCESS:         API Initialized successfully
FLASH_ERR_BUSY:        Other flash operation in progress, try again later
FLASH_ERROR_FAILURE:   Initialization failed. A RESET was performed on the flash
                       control circuit to rectify any internal errors
```

### Properties

Prototyped in file “r_flash_rx_if.h”

### Description

This function initializes the API, and if FLASH_CFG_CODE_FLASH_ENABLE == 1, copies the flash control API functions necessary for Code flash functionality into RAM.

### Reentrant

No, but is protected by lock to prevent errors from concurrent function calls.

### Example

```
flash_err_t err;

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

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

## 4.3 R_FLASH_Erase

This function implements flash block erase functionality and supports both Code Flash and 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 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 starting with the block number specified in the preceding argument.

### Return Values

| | |
|---|---|
| *FLASH_SUCCESS:* | *Operation successful (if BGO is enabled this means the operations was started successfully)* |
| *FLASH_ERR_BLOCKS:* | *Invalid number of blocks specified* |
| *FLASH_ERR_ADDRESS:* | *Invalid address specified* |
| *FLASH_ERR_BUSY:* | *Other flash op in progress, or API not initialized* |
| *FLASH_ERR_FAILURE:* | *Other flash op failure. Flash circuit has been reset.* |

### Properties

Prototyped in file "r_flash_rx_if.h"

### Description

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

The size of a code or data block varies between MCU types.
For example, on the RX111, both code and data flash block sizes are 1K.

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.

### Reentrant

No, but is protected by lock to prevent errors from concurrent function calls.

### 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, there must not be a lockbit set for it (e.g. RX63N), or the access window must be inactive, or configured to allow erase/write access to the specific block(s) (e.g. RX111). Refer to 4.6 R_FLASH_Control for more details.

## 4.4 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 a pointer to the buffer containing the data to write to Flash.

*dest_address*

This is a pointer to the Code Flash or Data Flash area to write. The address must be on a programming line boundary. See *Description* below for important restrictions regarding this parameter.

*num_bytes*

The number of bytes contained in the *src_address* buffer. This number must be a multiple of the programming size for memory area you are writing to. See *Special Notes* below for important restrictions regarding this parameter.

### Return Values

| | |
|---|---|
| *FLASH_SUCCESS:* | *Operation successful (if BGO/polling is enabled this means the operation was started successfully)* |
| *FLASH_ERR_FAILURE:* | *Operation failed. Possibly dest address under access window control.* |
| *FLASH_ERR_BUSY:* | *Other flash operation in progress or API not initialized* |
| *FLASH_ERR_BYTES:* | *Number of bytes provided was not a multiple of the programming size or exceed the maximum range* |
| *FLASH_ERR_ADDRESS:* | *Invalid address was input or address not on programming boundary* |

### Properties

Prototyped in file “r_flash_rx_if.h”

### Description

Writes data to flash memory.

When performing a write the user must make sure to start the write on a programming boundary and the number of bytes to write must be a multiple of the programming size. The boundaries and programming sizes differ depending on what MCU package is being used and whether the ROM or data flash is being written to. Programming boundaries start at the beginning of the flash area and then each boundary is a multiple of the programming size.

For the RX100 Series, in order to program any part of a code flash block, the access window must be inactive (default), or configured such that the write area specified has not been protected a write access control command. Refer to 4.6 R_FLASH_Control for more details.

### Reentrant

No, but is protected by lock to prevent errors from concurrent function calls.

### Example

```
flash_err_t ret;
uint8_t write_buffer[PROGRAM_SIZE] = "Hello World...";

/* Write data to internal memory. */
ret = R_FLASH_Write((uint32_t)write_buffer, address, PROGRAM_SIZE);

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

**Special Notes:**

- For RX 100 MCU's the minimum programming size is 4 for Code Flash and 1 for Data Flash.

## 4.5 R_FLASH_BlankCheck

This function is used to determine if an area in either the Code or Data Flash area is blank or not.

### Format

```
uint8_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 specifies the number of bytes that need to be checked. This value varies quite a bit depending on the MCU type.

For RX200 and RX600 Series MCUs, only Data Flash may be blank checked.

RX630/631/63N support block and 2 byte checks

RX610 & RX62x support block and 8 byte checks

For RX100 Series MCU's data flash, this must be a multiple of 1 and < 8192. For code flash it must be a multiple of 4 and < 256K. In addition, for parts whose Code flash exceeds 384 Kbytes, you may not specify an address and size combination that will cross a 256-Kbyte boundary.

**blank_check_result*

Pointer that will be populated by the API with the results of the blank check operation in non-BGO (blocking) mode

### Return Values

| | |
|---|---|
| *FLASH_SUCCESS:* | *Operation completed successfully (blocking) or was started successfully (BGO/polling mode).* |
| *FLASH_ERR_FAILURE:* | *Operation Failed for some other reason* |
| *FLASH_ERR_BUSY:* | *Another flash operation is in progress* |
| *FLASH_ERR_ADDRESS:* | *Invalid data flash address was input* |
| *FLASH_ERR_BYTES:* | *'num_bytes' was either too large or not aligned for the CF(4)/DF(1) boundary size.* |
| *FLASH_ERR_BUSY:* | *Other flash operation in progress or API not initialized* |

### Properties

Prototyped in file "r_flash_rx_if.h"

### Description

Before you can write to any flash area in an MCU, the area must already be blank.

The results of the blank-check operation are populated by the API into the *blank_check_result* variable. This variable is of type flash_res_t which is defined in r_flash_rx_if.h. If the API is used in BGO/polling mode, then the operation completion is determined by Control calls with a FLASH_CMD_STATUS_GET command. When the command no longer returns busy, then the blank-check status is returned.

### Reentrant

No, but is protected by lock to prevent errors from concurrent function calls.

### Example

```
flash_err_t ret;
flash_res_t result;

/* Blank check an entire data flash block. */
ret = R_FLASH_BlankCheck((uint32_t)FLASH_DF_BLOCK_0, 64, &result);

/* Check result. */
if (FLASH_RES_NOT_BLANK == result)
{
    /* Block is not blank. */
    . . .
}
else if (FLASH_RES_BLANK == ret)
{
    /* Block is blank. */
    . . .
}
```

## 4.6 R_FLASH_Control

This function implements all non-core functionality of the flash control circuit.

### 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 (if BGO is enabled this means the operations was started successfully)* |
| *FLASH_ERR_BYTES:* | *Number of blocks exceeds max range* |
| *FLASH_ERR_ ADDRESS:* | *Address is an invalid Code/Data Flash block start address* |
| *FLASH_ERR_ NULL_PTR:* | *Other flash was NULL for a command that expects a configuration structure* |
| *FLASH_ERR_BUSY:* | *Other flash operation in progress or API not initialized* |
| *FLASH_ERR_LOCKED:* | *The flash control circuit was in a command locked state and has was reset* |

### Properties

Prototyped in file “r_flash_rx_if.h”

### Description

This function is an expansion function that implements non-core flash control circuit functionality. Depending upon the command type, a different argument type has to be passed.

The table below lists the currently supported commands and required arguments:

| Command | Argument | Operation |
|---|---|---|
| FLASH_CMD_RESET | NULL | Kill any ongoing operation and reset the flash control circuit. |
| FLASH_CMD_STATUS_GET | NULL | Return the Status of the API (Busy or Idle) |
| FLASH_CMD_ACCESSWINDOW_SET | flash_access_window_config_t | RX100 - Sets the Access Window boundaries for Code Flash. |
| FLASH_CMD_ACCESSWINDOW_GET | flash_access_window_config_t | RX100 - Returns the Access Window boundaries for Code Flash. |
| FLASH_CMD_SWAPFLAG_TOGGLE | NULL | RX100 - Inverts the start-up program swap flag |
| FLASH_CMD_LOCKBIT_WRITE | flash_lockbit_config_t * | |
| FLASH_CMD_LOCKBIT_READ | flash_lockbit_config_t * | |
| FLASH_CMD_LOCKBIT_ENABLE | NULL | |
| FLASH_CMD_LOCKBIT_DISABLE | NULL | |
| FLASH_CMD_SET_BGO_CALLBACK | uint32_t * | |
| FLASH_CMD_CONFIG_CLOCK | uint32_t * | |
| FLASH_CMD_LOCKBIT_PROTECTION | flash_lockbit_enable_t * | |
| FLASH_CMD_LOCKBIT_PROGRAM | flash_program_lockbit_config_t * | |

### Reentrant

No, but is protected by lock to prevent errors from concurrent function calls, except for the FLASH_CMD_RESET command which can be executed at any time.

**Examples:**

1. The example below shows how to check for a previously started operation for completion when running in BGO/polling mode.

```
flash_err_t g_DF_err;

        // Begin a flash write operation
        g_DF_err = R_FLASH_Write((uint32_t)g_prog_buff, FLASH_DF_BLOCK_0,
                                 sizeof(g_prog_buff));

        // Check for status through control function and move operation to
        // completion.
        do
        {
            // This will move the write sequence along...
            g_DF_err = R_FLASH_Control(FLASH_CMD_STATUS_GET, NULL);
        }
        while (FLASH_ERR_BUSY == g_DF_err);

        if (FLASH_SUCCESS != g_DF_err)
        {
            printf("BGO Polling mode Write failure.");
        }
```

2. The access window is used to prevent unauthorized programming or erasure of code flash blocks. The following example makes only block 3 writeable, and by default everything else not writeable.

```
uint8_t err;
flash_access_window_config_t  access_info;

// Allow write to Code Flash block 3, and prevent writes to everywhere else
access_info.start_addr =  (uint32_t)FLASH_CF_BLOCK_3;
access_info.end_addr =  (uint32_t)(FLASH_CF_BLOCK_2);

R_FLASH_Control(FLASH_CMD_ACCESSWINDOW_SET, (void *)&access_info);
```

3. The following example shows how to toggle the active flash swap area:

```
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.");
    }
```

4. The following example shows how to read the currently active swap area.

```
uint32_t  g_swapFlag;
    err = R_FLASH_Control(FLASH_CMD_SWAPFLAG_GET, (void *)&g_swapFlag);
    if (FLASH_SUCCESS != err)
    {
        printf("Control FLASH_CMD_SWAPFLAG_GET command failure.");
    }
```

5. The example below shows how to read the currently configured 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 ACCESSWINDOW_GET command failure.");
    }
```

## 4.7 R_FLASH_GetVersion

Returns the current version of the Flash API.

### Format

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

### Parameters

None.

### Return Values

Version of Flash API.

### Properties

Prototyped in file “r_flash_rx_if.h”

### Description

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

### Reentrant

Yes.

### Example

```
uint32_t cur_version;

/* Get version of installed Flash API. */
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 API version is not new enough and does not have XXX feature
       that is needed by this application. Alert user. */
    ...
}
```

### Special Notes:

- This function is specified to be an inline function in *r_flash_rx.c.*

# Website and Support

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

Inquiries
http://www.renesas.com/inquiry

# Revision Record

| | | Description | |
|---|---|---|---|
| **Rev.** | **Date** | **Page** | **Summary** |
| 1.00 | July.24.14 | — | First edition issued |
| 1.10 | Nov.13.14 | — | Added RX113 support. |
| | | 7 | Updated “ROM to RAM” image. |

## General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU 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. Handling of Unused Pins
   Handle unused pins in accord 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.
5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different type number, confirm that the change will not lead to problems.
   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended 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; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate 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. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may 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 should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be 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, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

**SALES OFFICES** Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.


**Renesas Electronics America Inc.**
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

**Renesas Electronics Canada Limited**
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

**Renesas Electronics Europe Limited**
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

**Renesas Electronics Europe GmbH**
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

**Renesas Electronics (China) Co., Ltd.**
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

**Renesas Electronics (Shanghai) Co., Ltd.**
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

**Renesas Electronics Hong Kong Limited**
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

**Renesas Electronics Taiwan Co., Ltd.**
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

**Renesas Electronics Singapore Pte. Ltd.**
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

**Renesas Electronics Malaysia Sdn.Bhd.**
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

**Renesas Electronics Korea Co., Ltd.**
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141



Colophon 4.0