RX600 & RX200 Series

R01AN2066EJ0100
Rev. 1.00
Sep. 22, 2014

Reprogramming the On-Chip Flash Memory Using Simple Flash API for RX

Abstract

The latest Simple Flash API includes definitions associated with the Firmware Integration Technology (FIT). When using the Simple Flash API with a non-FIT project, many of those definitions are not necessary.

This application note describes the method to implement the Simple Flash API without unnecessary definitions for a non-FIT project.

Products

- RX610 Group
- RX62N, RX621, RX62T, and RX62G Groups
- RX630, RX63N, RX631, and RX63T Groups
- RX210, RX21A, and RX220 Groups

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

Contents

1. Specifications

In this application note, the Simple Flash API is modified to have only necessary definitions for a non-FIT project to implement the definitions in the user non-FIT project. The RX210 Group Initial Setting application note is used as an example of a non-FIT project here.

Figure 1.1 shows the Implementing the API.

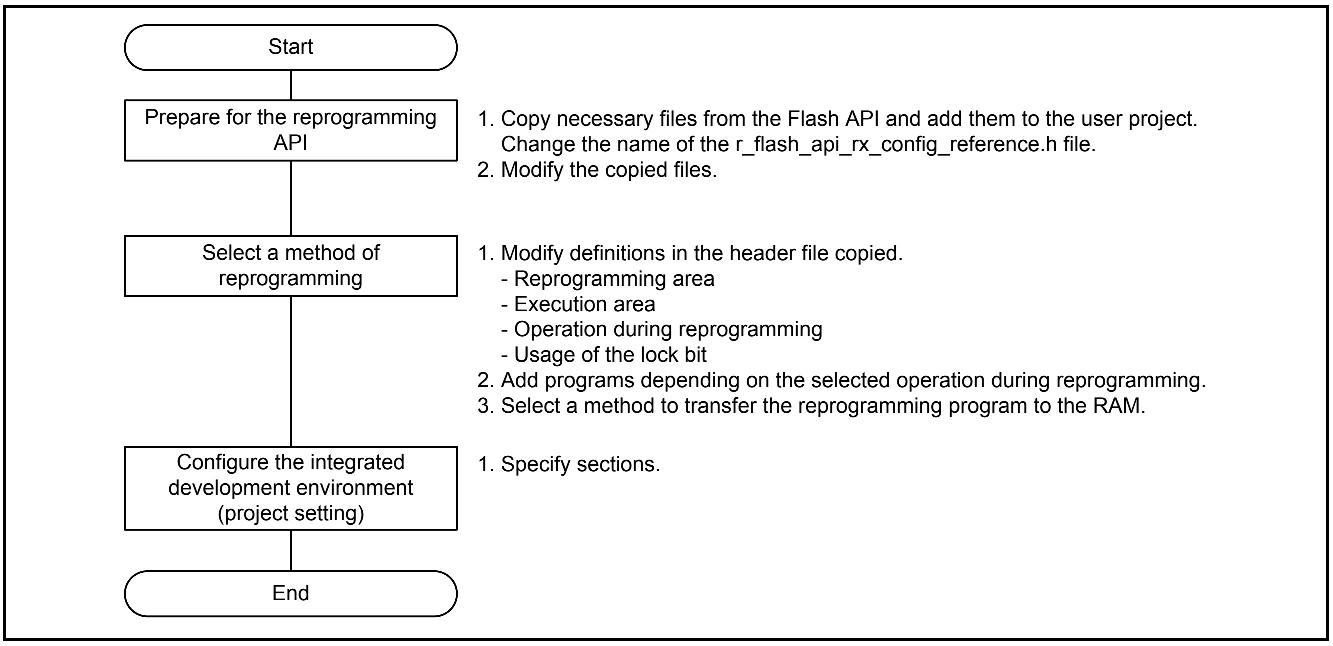
Start
Prepare for the reprogramming API
1. Copy necessary files from the Flash API and add them to the user project. Change the name of the r_flash_api_rx_config_reference.h file.
2. Modify the copied files.
Select a method of reprogramming
1. Modify definitions in the header file copied.
- Reprogramming area
- Execution area
- Operation during reprogramming
- Usage of the lock bit
2. Add programs depending on the selected operation during reprogramming.
3. Select a method to transfer the reprogramming program to the RAM.
Configure the integrated development environment (project setting)
1. Specify sections.
End

Figure 1.1 Implementing the API

The sample code accompanying this application note includes the sample code from the RX210 Group Initial Setting as well as the sample code for the reprogramming program (flash_api function).

With the sample code for the reprogramming program, when reprogramming of the E2 DataFlash (flash memory for storing data) is enabled, erasing, programming, and verification of the E2 DataFlash are executed. Subsequently, when reprogramming of the ROM (flash memory for storing code) is enabled, erasing, programming, and verification of the ROM are executed.

2. Operation Confirmation Conditions

The sample code accompanying this application note has been run and confirmed under the conditions below.

Table 2.1 Operation Confirmation Conditions

Item	Contents
MCU used	R5F5210BBDFP (RX210 Group)
Operating frequencies	- Main clock: 20 MHz - PLL: 100 MHz (main clock divided by 2 and multiplied by 10) - System clock (ICLK): 50 MHz (PLL divided by 2) - Peripheral module clock B (PCLKB): 25 MHz (PLL divided by 4) - FlashIF clock (FCLK): 25 MHz (PLL divided by 4)
Operating voltage	5.0 V
Integrated development environment	Renesas Electronics Corporation High-performance Embedded Workshop Version 4.09.01
C compiler	Renesas Electronics Corporation C/C++ Compiler Package for RX Family V.1.02 Release 01
	Compile options -cpu=rx200 -output=obj="$(CONFIGDIR)\$(FILELEAF).obj" -debug -nologo (The default setting is used in the integrated development environment.)
iodefine.h version	Version 1.4
Endian	Little endian
Operating mode	Single-chip mode
Processor mode	Supervisor mode
Sample code version	Version 1.00
Board used	Renesas Starter Kit for RX210 (product part no.: R0K505210C002BE)

3. Reference Application Notes

For additional information associated with this document, refer to the following application notes.

- RX210 Group Initial Setting Rev. 2.20 (R01AN1002EJ) (1)
- RX600 & RX200 Series Simple Flash API for RX Rev.2.50 (R01AN0544EU)

Note:

1. The initial setting functions in the RX210 Group Initial Setting application note above are used in the sample code in this application note. The revision number of the RX210 Group Initial Setting application note is current as of when this application note was made. However the latest version is always recommended. Visit the Renesas Electronics Corporation website to check and download the latest version.

4. Preparing for the Reprogramming Program

4.1 Adding Necessary Files from the Flash API

Copy the following files from the Simple Flash API project to add them to the user project.

The locations in the brackets below indicate original locations in the Simple Flash API project.

- r_flash_api_rx_if.h (location: \r_flash_api_rx)
- r_flash_api_rx210.h (location: \r_flash_api_rx\src\targets\rx210)
- r_flash_api_rx.c (location: \r_flash_api_rx\src)
- r_flash_api_rx_config_reference.h (location: \r_flash_api_rx\ref)

Rename the r_flash_api_rx_config_reference.h file to r_flash_api_rx_config.h when adding to the user project.

4.2 Modifying r_flash_api_rx_if.h

The r_flash_api_rx_if.h file needs to be modified as follows:

- Delete platform.h.

```
// /* Used to get which MCU is currently being used. */
// #include "platform.h"
```

- Delete read processing in r_flash_api_rxXXX.h.

```
/* Memory specifics for the each MCU group */
// #if defined(BSP_MCU_RX610)
//          #include "./src/targets/rx610/r_flash_api_rx610.h"
// #elif defined(BSP_MCU_RX621) || defined(BSP_MCU_RX62N)
//          #include "./src/targets/rx62n/r_flash_api_rx62n.h"
// #elif defined(BSP_MCU_RX62T)
//          #include "./src/targets/rx62t/r_flash_api_rx62t.h"
// #elif defined(BSP_MCU_RX62G)
//          #include "./src/targets/rx62g/r_flash_api_rx62g.h"
// #elif defined(BSP_MCU_RX630)
//          #include "./src/targets/rx630/r_flash_api_rx630.h"
// #elif defined(BSP_MCU_RX631) || defined(BSP_MCU_RX63N)
//          #include "./src/targets/rx63n/r_flash_api_rx63n.h"
// #elif defined(BSP_MCU_RX63T)
//          #include "./src/targets/rx63t/r_flash_api_rx63t.h"
// #elif defined(BSP_MCU_RX210)
//          #include "./src/targets/rx210/r_flash_api_rx210.h"
// #elif defined(BSP_MCU_RX21A)
//          #include "./src/targets/rx21a/r_flash_api_rx21a.h"
// #elif defined(BSP_MCU_RX220)
//          #include "./src/targets/rx220/r_flash_api_rx220.h"
// #else
//          #error "!!! No 'targets' folder for this MCU Group !!!"
// #endif
```

4.3 Modifying r_flash_api_rx210.h

The r_flash_api_rx210.h file needs to be modified as follows:

- Add definitions.

```
#define BSP_ROM_SIZE_BYTES (524288)
#define BSP_RAM_SIZE_BYTES (65536)
#define BSP_DATA_FLASH_SIZE_BYTES (8192)

#define ROM_PE_ADDR ((0x100000000-BSP_ROM_SIZE_BYTES)&(0x00FFFFFF))

#define BSP_ICLK_HZ (50000000)
#define BSP_FCLK_HZ (25000000)

/* FCU-RAM address define */
/* FCU F/W Store Address */
#define FCU_PRG_TOP (0xFEFFE000)
/* FCU RAM Address */
#define FCU_RAM_TOP (0x007F8000)
/* FCU RAM Size */
#define FCU_RAM_SIZE (0x2000)
```

Note: • Modify the values of these definitions according to the MCU used and the user system.

- Add the hardware lock mcu_lock_t.

```
typedef enum
{
        BSP_LOCK_FLASH = 0,
        BSP_NUM_LOCKS //This entry is not a valid lock. It is used for sizing g_bsp_Locks[] array below.
                               Do not touch!
} mcu_lock_t;
```

4.4 Modifying r_flash_api_rx.c

The r_flash_api_rx.c file needs to be modified as follows:

- Add #include for stdint.h and stdbool.h.

```
#include <stdint.h>
#include <stdbool.h>
```

- Replace #include "mcu_info.h" with "r_flash_api_rx210.h".

```
// #include "mcu_info.h"
#include "r_flash_api_rx210.h"
```

- Delete #include for "platform.h" and "r_flash_api_rx_private.h".

```
// #include "platform.h"
// #include "r_flash_api_rx_private.h"
```

- Modify the directory level of iodefine.h according to the user project.
  The following shows an example when the directory of iodefine.h is one level above the directory of r_flash_api_rx.c.

```
// #include "iodefine.h"
  #include "..¥iodefine.h"
```

- Add functions R_BSP_HardwareLock and R_BSP_HardwareUnlock

```
bool R_BSP_HardwareLock(mcu_lock_t const hw_index);
bool R_BSP_HardwareUnlock(mcu_lock_t const hw_index);
```

```
/******************************************************************************************************************
* Function Name: R_BSP_HardwareLock
* Description: Attempt to acquire the lock that has been sent in. This function takes in a peripheral index into
*                  the array that holds hardware locks.
* Arguments: hw_index -
*                      Index in locks array to the hardware resource to lock.
* Return Value: true -
*                       Lock was acquired.
*                      false -
*                       Lock was not acquired.
******************************************************************************************************************/
bool R_BSP_HardwareLock (mcu_lock_t const hw_index)
{
            return true;
} /* End of function R_BSP_HardwareLock() */
/******************************************************************************************************************
* Function Name: R_BSP_HardwareUnlock
* Description: Release hold on lock.
* Arguments: hw_index -
*                      Index in locks array to the hardware resource to unlock.
* Return Value: true -
*                       Lock was released.
*                      false -
*                       Lock was not released.
******************************************************************************************************************/
bool R_BSP_HardwareUnlock (mcu_lock_t const hw_index)
{
            return true;
} /* End of function R_BSP_HardwareUnlock() */
```

5. Selecting a Method of Reprogramming

This section describes methods of reprogramming with selection of a reprogramming area, an execution area, an operation during reprogramming, and a RAM transfer method.

5.1 Specifying a Method of Reprogramming

Operations and control methods vary depending on the selected reprogramming area, execution area, and operation during reprogramming. Select one of the methods listed in Table 5.1 and configure r_flash_api_rx_config.h accordingly.

Table 5.2 to Table 5.12 list settings in r_flash_api_rx_config.h. Numbers in table titles correspond to numbers in Table 5.1.

Table 5.1 Methods of Reprogramming

No.	Reprogramm-ing Area	Program Execution Area	Operation During Reprogramming	Lock Bit Protection (ROM Only)
1	ROM, E2 DataFlash	RAM	Subsequent processing not executed (1)	Disabled by the API
2			Subsequent processing executed (2)	Disabled by the API
3			Subsequent processing not executed (1)	Disabled by the user program
4			ROM: Subsequent processing not executed (1) E2 DataFlash: Subsequent processing executed (2)	Disabled by the user program (3)
5	ROM	RAM	Subsequent processing not executed (1)	Disabled by the API
6			Subsequent processing executed (2)	Disabled by the API
7			Subsequent processing not executed (1)	Disabled by the user program
8	E2 DataFlash	RAM	Subsequent processing not executed (1)	—
9			Subsequent processing executed (2)	
10		ROM	Subsequent processing not executed (1)	
11			Subsequent processing executed (2)	

Notes:
1. Waits for completion of reprogramming processing, and the subsequent processing is not executed.
2. Subsequent processing of reprogramming processing is executed.
3. When the lock bit protection is disabled by the user program, the operation 'subsequent processing executed' cannot be selected with the ROM. Always choose the operation 'subsequent processing not executed' with the ROM.

Table 5.2 No. 1: Settings in r_flash_api_rx_config.h

Changed Contents	Code after the Change
The ROM is reprogrammed (program executed in the RAM).	#define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
Programming data is not stored in the RAM.	// #define FLASH_API_RX_CFG_FLASH_TO_FLASH
Wait for completion of reprogramming processing for the E2 DataFlash and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_DATA_FLASH_BGO
Wait for completion of reprogramming processing for the ROM and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_ROM_BGO
The lock bit protection is disabled automatically by the API.	#define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

Table 5.3 No. 2: Settings in r_flash_api_rx_config.h

Changed Contents	Code after the Change
The ROM is reprogrammed (program executed in the RAM).	#define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
Programming data is stored in the RAM.	#define FLASH_API_RX_CFG_FLASH_TO_FLASH
Subsequent processing is executed during reprogramming the E2 DataFlash.	#define FLASH_API_RX_CFG_DATA_FLASH_BGO
Subsequent processing is executed during reprogramming the ROM.	#define FLASH_API_RX_CFG_ROM_BGO
The lock bit protection is disabled automatically by the API.	#define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

Table 5.4 No. 3: Settings in r_flash_api_rx_config.h

Changed Contents	Code after the Change
The ROM is reprogrammed (program executed in the RAM).	#define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
Programming data is not stored in the RAM.	// #define FLASH_API_RX_CFG_FLASH_TO_FLASH
Wait for completion of reprogramming processing for the E2 DataFlash and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_DATA_FLASH_BGO
Wait for completion of reprogramming processing for the ROM and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_ROM_BGO
The lock bit protection is disabled by the user program.	// #define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

Table 5.5 No. 4: Settings in r_flash_api_rx_config.h

Changed Contents	Code after the Change
The ROM is reprogrammed (program executed in the RAM).	#define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
Programming data is stored in the RAM.	#define FLASH_API_RX_CFG_FLASH_TO_FLASH
Subsequent processing is executed during reprogramming the E2 DataFlash.	#define FLASH_API_RX_CFG_DATA_FLASH_BGO
Wait for completion of reprogramming processing for the ROM and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_ROM_BGO
The lock bit protection is disabled by the user program.	// #define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

Table 5.6 No. 5: Settings in r_flash_api_rx_config.h

Changed Contents	Code after the Change
The ROM is reprogrammed (program executed in the RAM).	#define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
Programming data is not stored in the RAM.	// #define FLASH_API_RX_CFG_FLASH_TO_FLASH
Wait for completion of reprogramming processing for the E2 DataFlash and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_DATA_FLASH_BGO
Wait for completion of reprogramming processing for the ROM and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_ROM_BGO
The lock bit program is disabled automatically by the API.	#define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

Table 5.7 No. 6: Settings in r_flash_api_rx_config.h

Changed Contents	Code after the Change
The ROM is reprogrammed (program executed in the RAM).	#define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
Programming data is stored in the RAM.	#define FLASH_API_RX_CFG_FLASH_TO_FLASH
Wait for completion of reprogramming processing for the E2 DataFlash and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_DATA_FLASH_BGO
Subsequent processing is executed during reprogramming the ROM.	#define FLASH_API_RX_CFG_ROM_BGO
The lock bit program is disabled automatically by the API.	#define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

Table 5.8 No. 7: Settings in r_flash_api_rx_config.h

Changed Contents	Code after the Change
The ROM is reprogrammed (program executed in the RAM).	#define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
Programming data is not stored in the RAM.	// #define FLASH_API_RX_CFG_FLASH_TO_FLASH
Wait for completion of reprogramming processing for the E2 DataFlash and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_DATA_FLASH_BGO
Wait for completion of reprogramming processing for the ROM and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_ROM_BGO
The lock bit protection is disabled by the user program.	// #define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

Table 5.9 No. 8: Settings in r_flash_api_rx_config.h

Changed Contents	Code after the Change
The ROM is reprogrammed (program executed in the RAM).	#define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
Programming data is not stored in the RAM.	// #define FLASH_API_RX_CFG_FLASH_TO_FLASH
Wait for completion of reprogramming processing for the E2 DataFlash and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_DATA_FLASH_BGO
Wait for completion of reprogramming processing for the ROM and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_ROM_BGO
The lock bit program is disabled automatically by the API (default).	#define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

Table 5.10 No. 9: Settings in r_flash_api_rx_config.h

Changed Contents	Code after the Change
The ROM is reprogrammed (program executed in the RAM).	#define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
Programming data is stored in the RAM.	#define FLASH_API_RX_CFG_FLASH_TO_FLASH
Subsequent processing is executed during reprogramming the E2 DataFlash.	#define FLASH_API_RX_CFG_DATA_FLASH_BGO
Wait for completion of reprogramming processing for the ROM and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_ROM_BGO
The lock bit program is disabled automatically by the API (default).	#define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

Table 5.11 No. 10: Settings in r_flash_api_rx_config.h

Changed Contents	Code after the Change
The ROM is not reprogrammed.	// define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
Programming data is not stored in the RAM.	// #define FLASH_API_RX_CFG_FLASH_TO_FLASH
Wait for completion of reprogramming processing for the E2 DataFlash and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_DATA_FLASH_BGO
Wait for completion of reprogramming processing for the ROM and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_ROM_BGO
The lock bit program is disabled automatically by the API (default).	#define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

Table 5.12 No. 11: Settings in r_flash_api_rx_config.h

Changed Contents	Code after the Change
The ROM is not reprogrammed.	// define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
Programming data is stored in the RAM.	#define FLASH_API_RX_CFG_FLASH_TO_FLASH
Subsequent processing is executed during reprogramming the E2 DataFlash.	#define FLASH_API_RX_CFG_DATA_FLASH_BGO
Wait for completion of reprogramming processing for the ROM and the subsequent processing is not executed.	// #define FLASH_API_RX_CFG_ROM_BGO
The lock bit program is disabled automatically by the API (default).	#define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

5.2 Selecting an Operation During Reprogramming

When the operation 'subsequent processing executed' is selected as an operation during reprogramming, the flash ready interrupt generation is enabled and completion of reprogramming is confirmed in the interrupt handling.

The interrupt priority level of the flash ready interrupt is shown in Table 5.13.

If the flash ready interrupt and another interrupt occur simultaneously, a higher priority interrupt is executed first.

Table 5.13 Interrupt Priority Level of the Flash Ready Interrupt

Setting	Code after the Level is Set
The priority level of the flash ready interrupt is set to 5.	#define FLASH_API_RX_CFG_FLASH_READY_IPL 5

The following four functions must be written in the reprogramming program. Use the functions adding required processing.

Refer to the section "Using Non-Blocking Background Operations" in the Simple Flash API for RX application note.

- void FlashEraseDone(void)
- void FlashWriteDone(void)
- void FlashBlankCheckDone(uint8_t result)
- void FlashError(void)

When the operation 'subsequent processing executed' is selected, make sure that registers which are write disabled must not be rewritten within the loop of the processing during ROM/E2 DataFlash P/E mode. When in ROM/E2 DataFlash P/E mode, processing in the loop including additional processing are executed in P/E mode.

5.3 Method of Transferring the Reprogramming API to the RAM

When APIs used for reprogramming are executed in the RAM, the APIs need to be transferred from the ROM to the RAM. The transfer method can be selected from the following two methods.

- Transfer is performed at a given timing.
- Transfer is performed immediately after a reset.

Table 5.14 and Table 5.15 list definitions in r_flash_api_rx_config.h for each method.

Table 5.14 Settings in r_flash_api_rx_config.h (Transfer at a Given Timing)

Setting	Code after Setting
The ROM is reprogrammed (program executed in the RAM).	#define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
The R_FlashCodeCopy function is used to transfer programs.	#define FLASH_API_RX_CFG_COPY_CODE_BY_API

Table 5.15 Settings in r_flash_api_rx_config.h (Transfer Immediately After a Reset)

Setting	Code after Setting
The ROM is reprogrammed (program executed in the RAM).	#define FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING
The R_FlashCodeCopy function is not used to transfer programs.	// #define FLASH_API_RX_CFG_COPY_CODE_BY_API

Note: • When selecting the method to transfer immediately after a reset, the dbsct.c file which is made when generating the project needs to be modified.

When the reprogramming program is executed in the RAM, write codes for transferring the program from the ROM to the RAM after a reset.

- When using the R_FlashCodeCopy function for ROM to RAM transfer, add processing to call the R_FlashCodeCopy function.

- When not using the R_FlashCodeCopy function for ROM to RAM transfer, add a code to the dbsct.c file as shown in red below.

```
{ __sectop("D"), __secend("D"), __sectop("R") },
{ __sectop("D_2"), __secend("D_2"), __sectop("R_2") },
{ __sectop("D_1"), __secend("D_1"), __sectop("R_1") },
{ __sectop("PFRAM"), __secend("PFRAM"), __sectop("RPFRAM") }
```

6. Additional Setting in the Integrated Development Environment

The following settings are required to specify sections in the integrated development environment.

For details on the settings, refer to the section "Putting Flash API Code in RAM" in the Simple Flash API for RX application note.

- Define the PFRAM section in the ROM area.
- Define the RPFRAM section in the RAM area.
- Specify the linker setting so that code in the FRAM section is executed in the RAM.

7. Software

7.1 Operation Overview

This section describes the sample code for the reprogramming program.

The ROM and the E2 DataFlash are reprogrammed using APIs from the Simple Flash API.

In the reprogramming areas, all areas in the E2 DataFlash and blocks 255 to 16 in the ROM are erased, and the content in the start address of each block is reprogrammed to the following:
"hello world0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ".

Figure 7.1 shows the Memory Map.

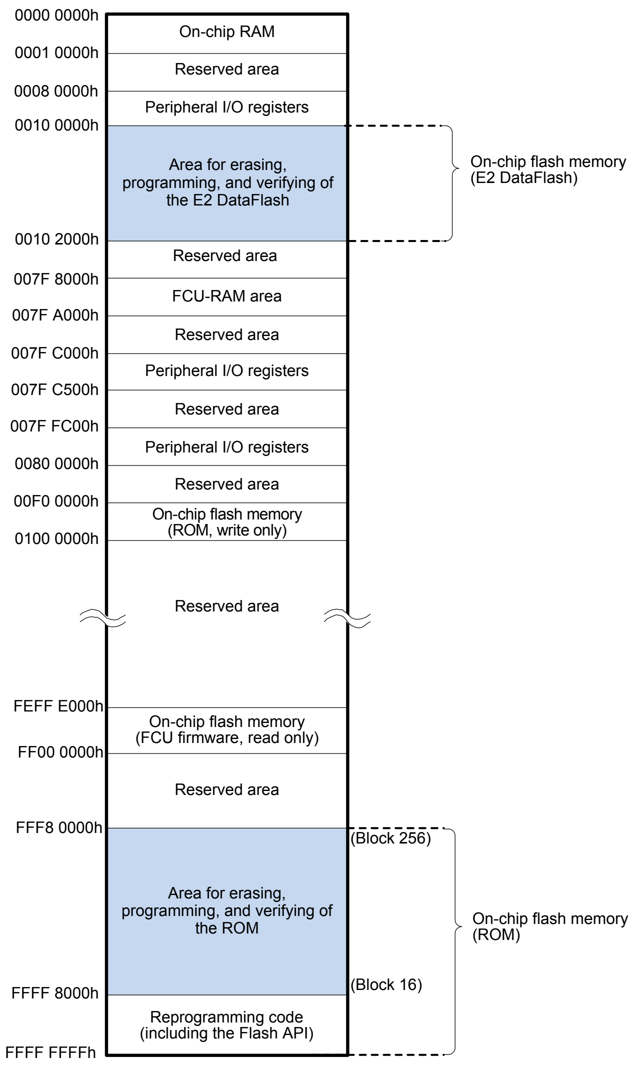

Figure 7.1 Memory Map

7.2 File Composition

Table 7.1 lists the Files Used in the Sample Code, Table 7.2 lists the Standard Include File, Table 7.3 lists the Functions and Their Settings in the RX210 Group Initial Setting Application Note, and Table 7.4 lists the File in the Simple Flash API. Files generated by the integrated development environment are not included in this table.

Table 7.1 Files Used in the Sample Code

File Name	Outline	Remarks
main.c	Main processing	
flash_write_main.c	Sample processing for reprogramming the ROM and the E2 DataFlash	
flash_api_rx.c	Flash memory reprogramming program (Simple Flash API)	r_flash_api_rx.c modified as described in 4.4
flash_api_rx_if.h	Header file for flash_api_rx.c	r_flash_api_rx_if.h modified as described in 4.2
flash_api_rx210.h	Header file for flash_api_rx.c	r_flash_api_rx210.h modified as described in 4.3

Table 7.2 Standard Include File

File Name	Outline
stdbool.h	Defines macros associated with Boolean and its value.
stdint.h	Defines macros declaring the integer type with the specified width.
machine.h	Defines types of intrinsic functions for the RX Family.

Table 7.3 Functions and Their Settings in the RX210 Group Initial Setting Application Note

File Name	Function	Contents
r_init_stop_module.c	R_INIT_StopModule()	—
r_init_stop_module.h	—	Module-stop state is canceled for DMAC/DTC, EXDMAC, RAM0, and RAM1.
r_init_non_existent_port.c	R_INIT_NonExistentPort()	—
r_init_non_existent_port.h	—	100-pin package is specified.
r_init_clock.c	R_INIT_Clock()	—
r_init_clock.h	—	Clock setting No. 1 is specified.

Table 7.4 File in the Simple Flash API
(RX600 & RX200 Series Simple Flash API for RX application note)

File Name	Function	Contents
r_flash_api_rx_config.h	—	- Reprograms the ROM and the E2 DataFlash. - Executes reprogramming processing in the RAM, waits for the completion of the reprogramming. The subsequent processing is not executed. - Lock bit protection is disabled by the API.

7.3 Option-Setting Memory

Table 7.5 lists the Option-Setting Memory Configured in the Sample Code. When necessary, set a value suited to the user system.

Table 7.5 Option-Setting Memory Configured in the Sample Code

Symbol	Address	Setting Value	Contents
OFS0	FFFF FF8Fh to FFFF FF8Ch	FFFF FFFFh	The IWDT is stopped after a reset. The WDT is stopped after a reset.
OFS1	FFFF FF8Bh to FFFF FF88h	FFFF FFFFh	The voltage monitor 0 reset is disabled after a reset. HOCO oscillation is disabled after a reset.
MDES	FFFF FF83h to FFFF FF80h	FFFF FFFFh	Little endian
MDEB	FF7F FFF8h to FF7F FFFBh	FFFF FFFFh	Little endian

7.4 Constants

Table 7.6 lists the Constants Used in the Sample Code.

Table 7.6 Constants Used in the Sample Code

Constant Name	Setting Value	Contents
E2FLASH_WRITE_START	0	Reprogramming of the E2 DataFlash started
E2FLASH_WRITE_ERASE	1	Erasing the E2 DataFlash
E2FLASH_WRITE_BLANK	2	Blank checking the E2 DataFlash
E2FLASH_WRITE_PROGRAM	3	Programming the E2 DataFlash
E2FLASH_WRITE_FIN	4	E2 DataFlash reprogramming completed
FLASH_WRITE_START	0	Reprogramming of the ROM started
FLASH_WRITE_ERASE	1	Erasing the ROM
FLASH_WRITE_PROGRAM	2	Programming the ROM
FLASH_WRITE_FIN	3	ROM reprogramming completed
ROM_RESERVED_BYTES	32768	Storage size for the reprogramming program
E2FLASH_WRITE_ENABLE	Enabled	Reprogramming of the E2 DataFlash enabled
FLASH_WRITE_ENABLE	Enabled	Reprogramming of the ROM enabled

7.5 Variables

Table 7.7 lists the Global Variables.

Table 7.7 Global Variables

Type	Variable Name	Contents	Function Used
uint8_t	E2Flash_WriteStatus	Status of E2 DataFlash reprogramming	main
uint8_t	Flash_WriteStatus	Status of ROM reprogramming	main

7.6 Functions

Table 7.8 lists the Functions.

Table 7.8 Functions

Function Name	Outline
main	Main processing
flash_write	Reprogramming flash memory
E2Flash_Write_start	Peprogramming E2 DataFlash (subsequent processing is executed during programming)
E2Flash_Write	Reprogramming E2 DataFlash
Flash_Write_start	Reprogramming ROM (subsequent processing is executed during programming)
Flash_Write	Reprogramming ROM
rom_bgo_init	Transferring relocatable vector to RAM
lock_bit_tests	Disabling ROM lock bit protection

7.7 Function Specifications

The following tables list the sample code function specifications.

main	
Outline	Main processing
Header	None
Declaration	void main(void);
Description	After initialization, the function for reprogramming the flash memory is called.
Arguments	None
Return Value	None

flash_api	
Outline	Reprogramming flash memory
Header	None
Declaration	void flash_api (void)
Description	The function for reprogramming the flash memory is executed. The function to be executed is selected according to the enable/disable setting of reprogramming and the selected operation during reprogramming.
Arguments	None
Return Value	None

E2Flash_Write_start	
Outline	Reprogramming E2 DataFlash (subsequent processing is executed during programming)
Header	None
Declaration	static void E2Flash_Write_start (void)
Description	Reprogramming of the E2 DataFlash is executed. If the FCU command is issued successfully, the subsequent processing is executed until the command execution is completed.
Arguments	None
Return Value	None

E2Flash_Write	
Outline	Reprogramming E2 DataFlash
Header	None
Declaration	static void E2Flash_Write (void)
Description	Reprogramming of the E2 DataFlash is executed. If the FCU command is issued, a loop is performed within the processing until the command execution is completed.
Arguments	None
Return Value	None

Flash_Write_start	
Outline	Reprogramming ROM (subsequent processing is executed during programming)
Header	None
Declaration	static void Flash_Write_start (void)
Description	Reprogramming of the ROM is executed. If the FCU command is issued successfully, the subsequent processing is executed until the command execution is completed.
Arguments	None
Return Value	None

Flash_Write	
Outline	Reprogramming ROM
Header	None
Declaration	static void Flash_Write (void)
Description	Reprogramming of the ROM is executed. After issuing the FCU command, a loop is performed within the processing until the command execution is completed.
Arguments	None
Return Value	None

rom_bgo_init	
Outline	Transferring relocatable vector to RAM
Header	None
Declaration	static void rom_bgo_init(void)
Description	The relocatable vector table is transferred to the RAM to use the flash ready interrupt.
Arguments	None
Return Value	None

lock_bit_tests	
Outline	Disabling ROM lock bit protection
Header	None
Declaration	static void lock_bit_tests (void)
Description	Lock bit protection is disabled for the ROM. If the result of the issued read lock bit status command is "lock bit enabled", the lock bit protection is disabled and block erasing is executed.
Arguments	None
Return Value	None

7.8 Flowcharts

7.8.1 Main Processing

Figure 7.2 shows the Main Processing.

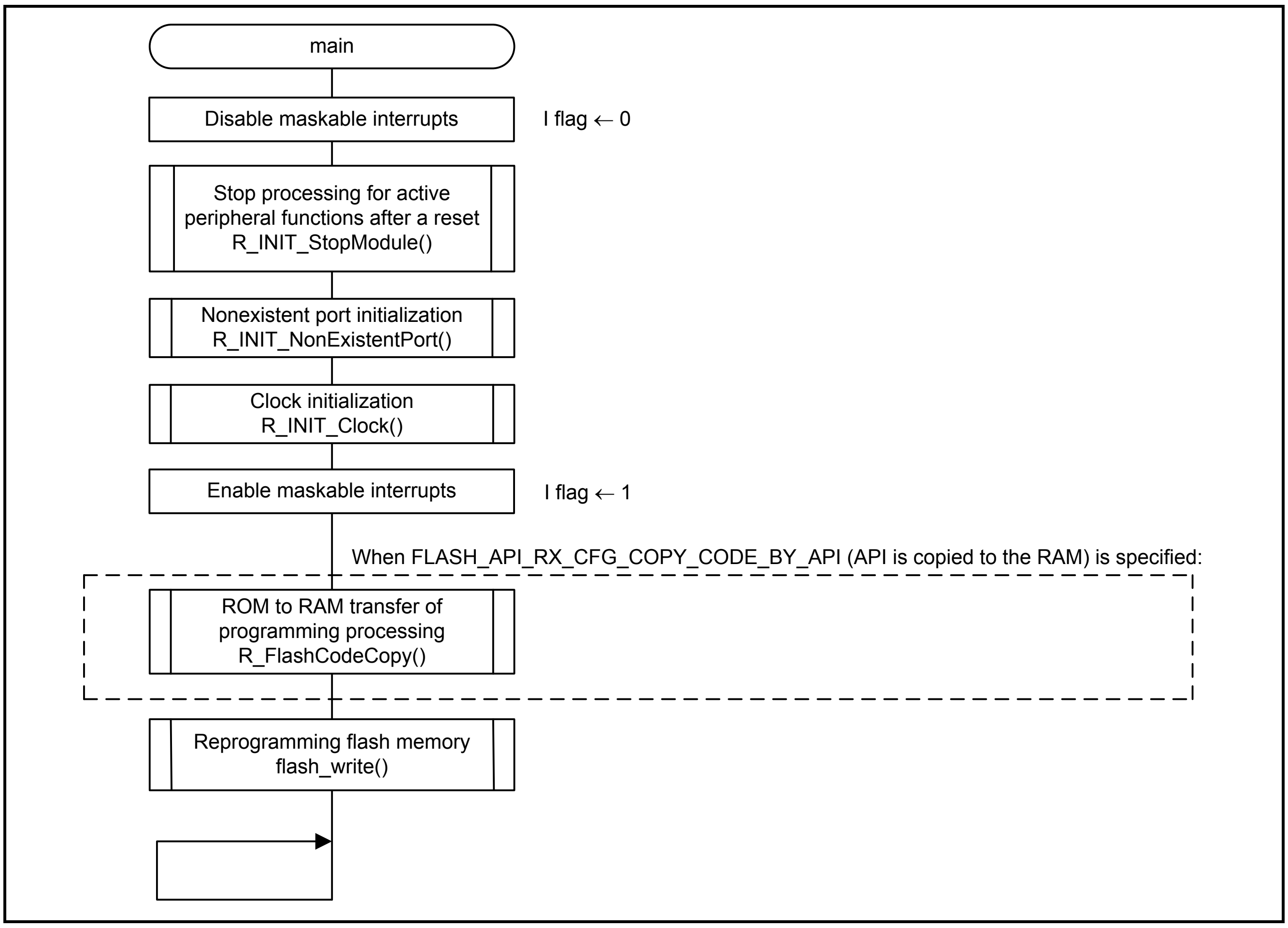
main
Disable maskable interrupts
I flag ← 0
Stop processing for active peripheral functions after a reset
R_INIT_StopModule()
Nonexistent port initialization
R_INIT_NonExistentPort()
Clock initialization
R_INIT_Clock()
Enable maskable interrupts
I flag ← 1
When FLASH_API_RX_CFG_COPY_CODE_BY_API (API is copied to the RAM) is specified:
ROM to RAM transfer of programming processing
R_FlashCodeCopy()
Reprogramming flash memory
flash_write()

Figure 7.2 Main Processing

7.8.2 Reprogramming Flash Memory

Figure 7.3 shows the Reprogramming Flash Memory.

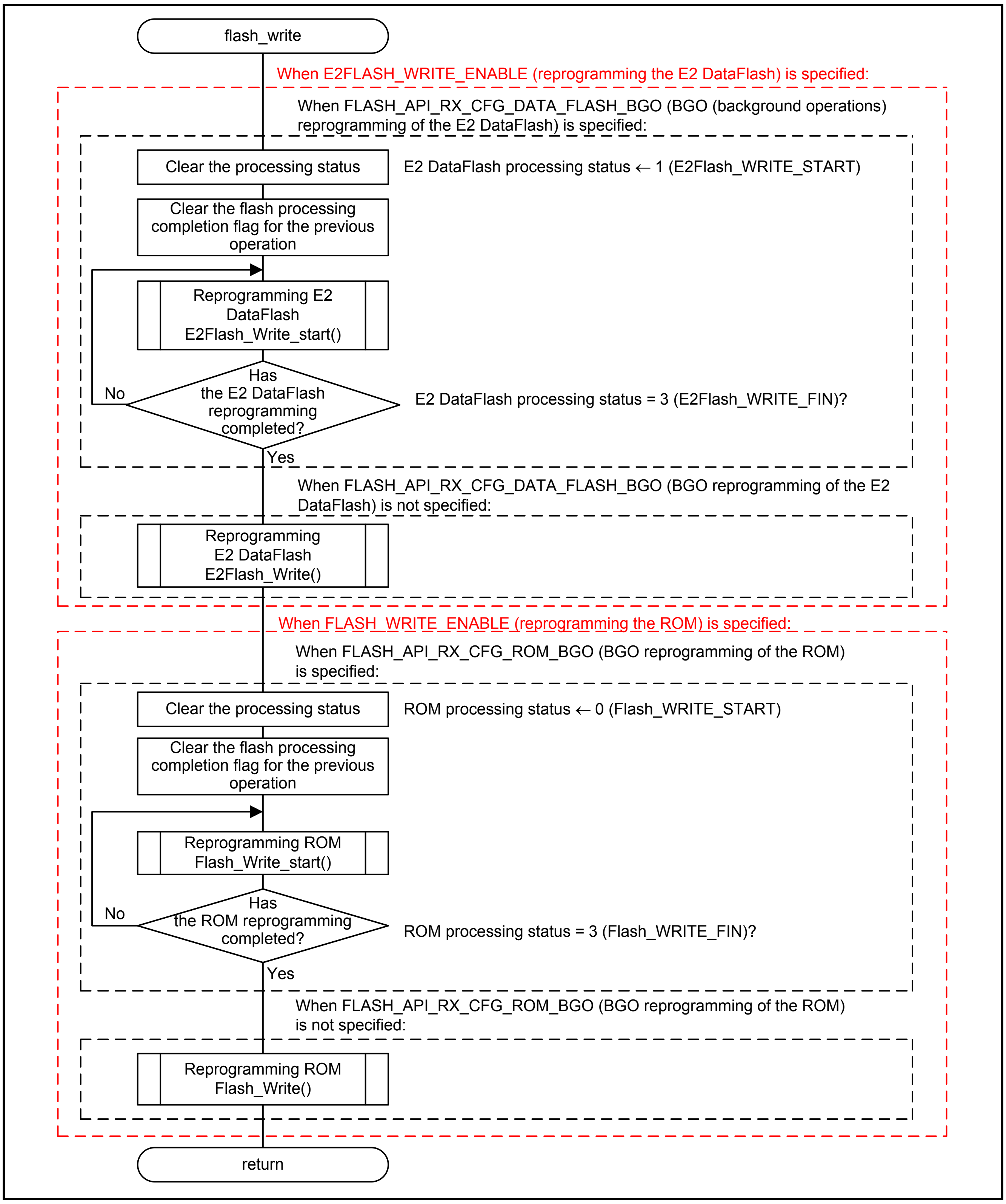
flash_write
When E2FLASH_WRITE_ENABLE (reprogramming the E2 DataFlash) is specified:
When FLASH_API_RX_CFG_DATA_FLASH_BGO (BGO (background operations) reprogramming of the E2 DataFlash) is specified:
Clear the processing status
E2 DataFlash processing status ← 1 (E2Flash_WRITE_START)
Clear the flash processing completion flag for the previous operation
Reprogramming E2 DataFlash E2Flash_Write_start()
Has the E2 DataFlash reprogramming completed?
No
Yes
E2 DataFlash processing status = 3 (E2Flash_WRITE_FIN)?
When FLASH_API_RX_CFG_DATA_FLASH_BGO (BGO reprogramming of the E2 DataFlash) is not specified:
Reprogramming E2 DataFlash E2Flash_Write()
When FLASH_WRITE_ENABLE (reprogramming the ROM) is specified:
When FLASH_API_RX_CFG_ROM_BGO (BGO reprogramming of the ROM) is specified:
Clear the processing status
ROM processing status ← 0 (Flash_WRITE_START)
Clear the flash processing completion flag for the previous operation
Reprogramming ROM Flash_Write_start()
Has the ROM reprogramming completed?
No
Yes
ROM processing status = 3 (Flash_WRITE_FIN)?
When FLASH_API_RX_CFG_ROM_BGO (BGO reprogramming of the ROM) is not specified:
Reprogramming ROM Flash_Write()
return

Figure 7.3 Reprogramming Flash Memory

7.8.3 Reprogramming E2 DataFlash (Subsequent Processing is Executed During Programming)

Figure 7.4 and Figure 7.5 show the Reprogramming E2 DataFlash (Subsequent Processing is Executed During Programming).

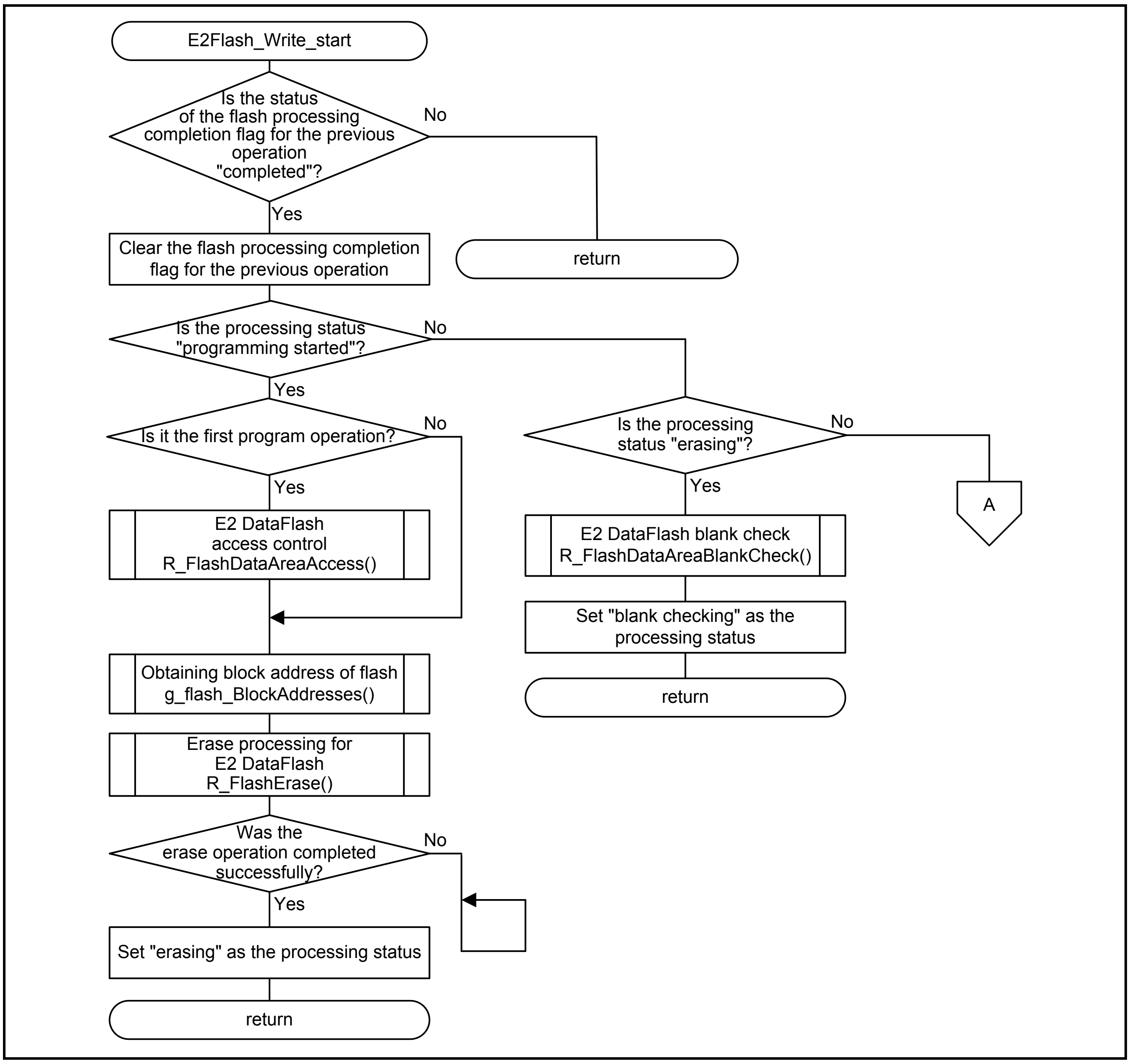
E2Flash_Write_start
Is the status of the flash processing completion flag for the previous operation "completed"?
No
Yes
Clear the flash processing completion flag for the previous operation
return
Is the processing status "programming started"?
No
Yes
Is it the first program operation?
No
Yes
E2 DataFlash access control R_FlashDataAreaAccess()
Obtaining block address of flash g_flash_BlockAddresses()
Erase processing for E2 DataFlash R_FlashErase()
Was the erase operation completed successfully?
No
Yes
Set "erasing" as the processing status
return
Is the processing status "erasing"?
No
Yes
A
E2 DataFlash blank check R_FlashDataAreaBlankCheck()
Set "blank checking" as the processing status
return

Figure 7.4 Reprogramming E2 DataFlash (Subsequent Processing is Executed During Programming) (1/2)

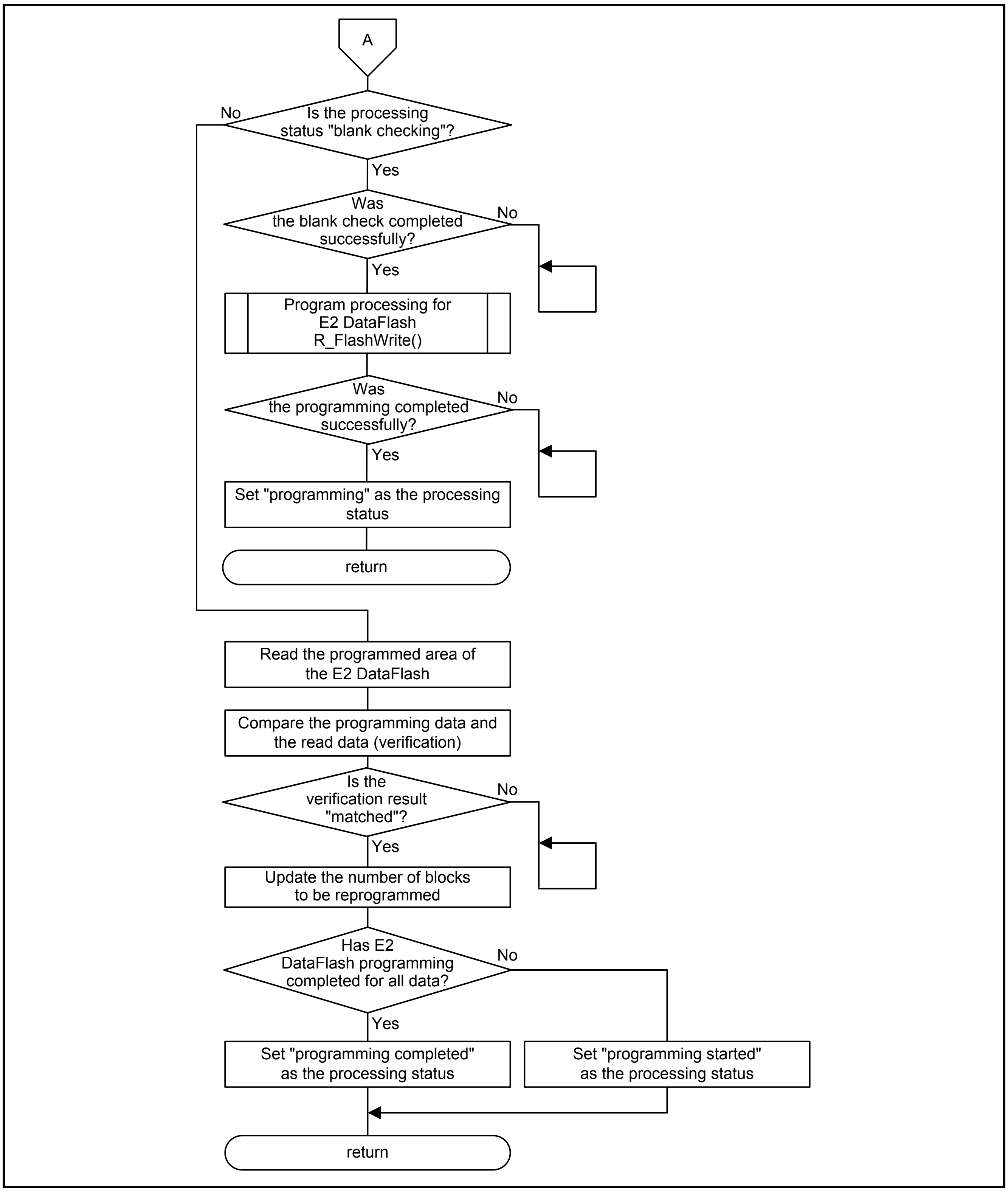
A
Is the processing status "blank checking"?
No
Yes
Was the blank check completed successfully?
No
Yes
Program processing for E2 DataFlash R_FlashWrite()
Was the programming completed successfully?
No
Yes
Set "programming" as the processing status
return
Read the programmed area of the E2 DataFlash
Compare the programming data and the read data (verification)
Is the verification result "matched"?
No
Yes
Update the number of blocks to be reprogrammed
Has E2 DataFlash programming completed for all data?
No
Yes
Set "programming completed" as the processing status
Set "programming started" as the processing status
return

Figure 7.5 Reprogramming E2 DataFlash (Subsequent Processing is Executed During Programming) (2/2)

7.8.4 Reprogramming E2 DataFlash

Figure 7.6 shows the Reprogramming E2 DataFlash.

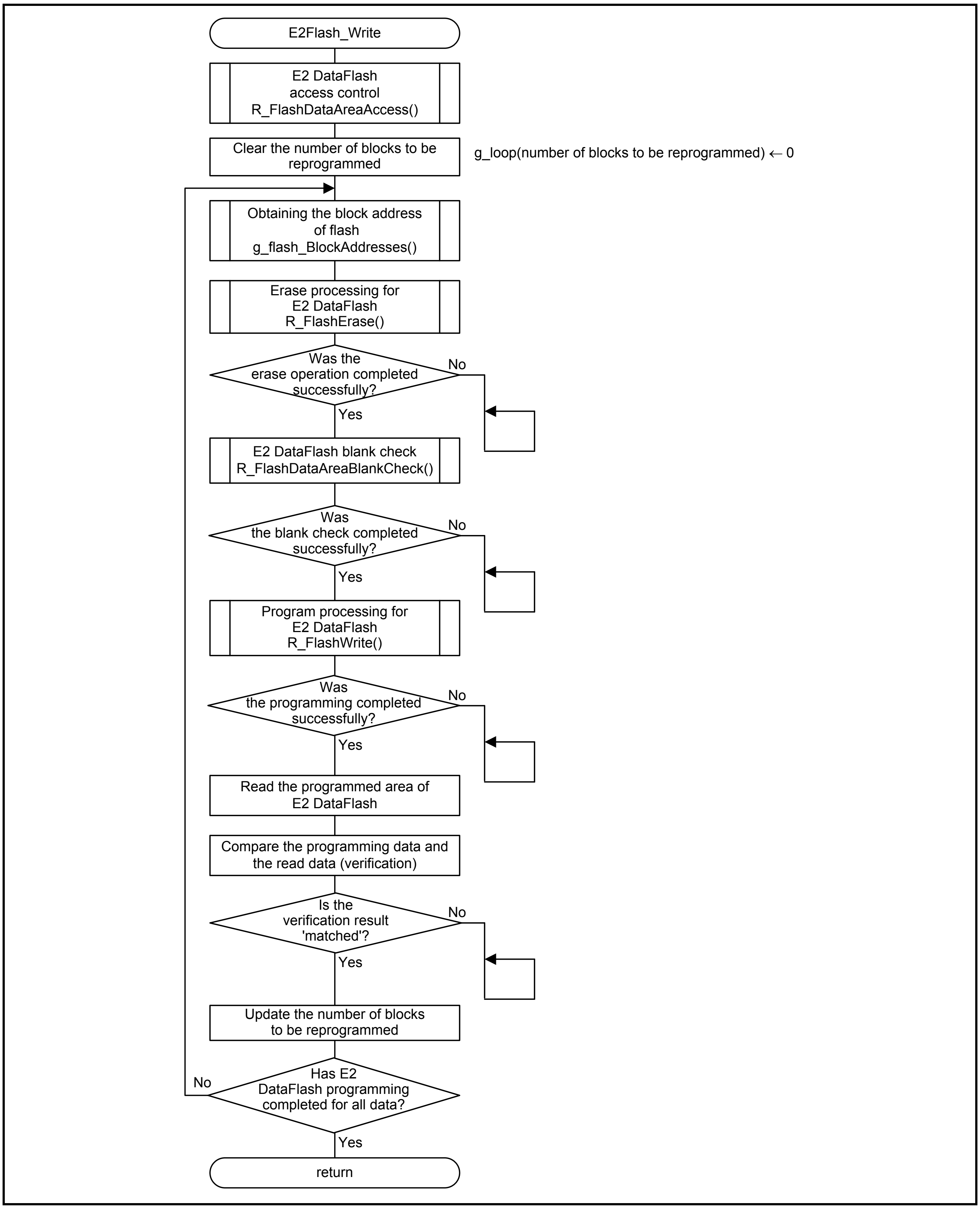
E2Flash_Write
E2 DataFlash access control R_FlashDataAreaAccess()
Clear the number of blocks to be reprogrammed
g_loop(number of blocks to be reprogrammed) ← 0
Obtaining the block address of flash g_flash_BlockAddresses()
Erase processing for E2 DataFlash R_FlashErase()
Was the erase operation completed successfully?
No
Yes
E2 DataFlash blank check R_FlashDataAreaBlankCheck()
Was the blank check completed successfully?
No
Yes
Program processing for E2 DataFlash R_FlashWrite()
Was the programming completed successfully?
No
Yes
Read the programmed area of E2 DataFlash
Compare the programming data and the read data (verification)
Is the verification result 'matched'?
No
Yes
Update the number of blocks to be reprogrammed
Has E2 DataFlash programming completed for all data?
No
Yes
return

Figure 7.6 Reprogramming E2 DataFlash

7.8.5 Reprogramming ROM (Subsequent Processing is Executed During Programming)

Figure 7.7 and Figure 7.8 show the Reprogramming ROM (Subsequent Processing is Executed During Programming).

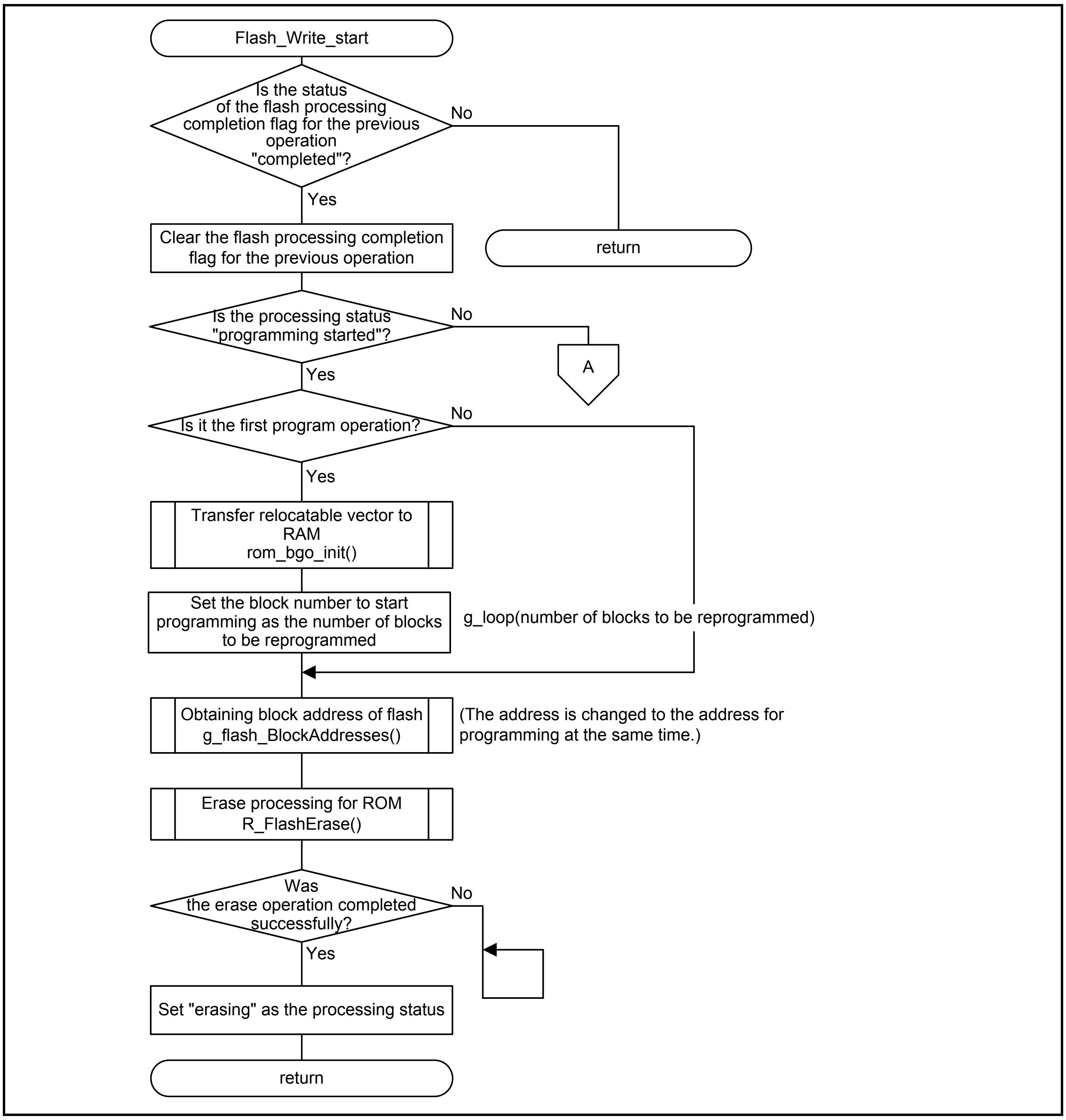
Flash_Write_start
Is the status of the flash processing completion flag for the previous operation "completed"?
No
Yes
return
Clear the flash processing completion flag for the previous operation
Is the processing status "programming started"?
No
A
Yes
Is it the first program operation?
No
Yes
Transfer relocatable vector to RAM rom_bgo_init()
Set the block number to start programming as the number of blocks to be reprogrammed
g_loop(number of blocks to be reprogrammed)
Obtaining block address of flash g_flash_BlockAddresses()
(The address is changed to the address for programming at the same time.)
Erase processing for ROM R_FlashErase()
Was the erase operation completed successfully?
No
Yes
Set "erasing" as the processing status
return

Figure 7.7 Reprogramming ROM (Subsequent Processing is Executed During Programming) (1/2)

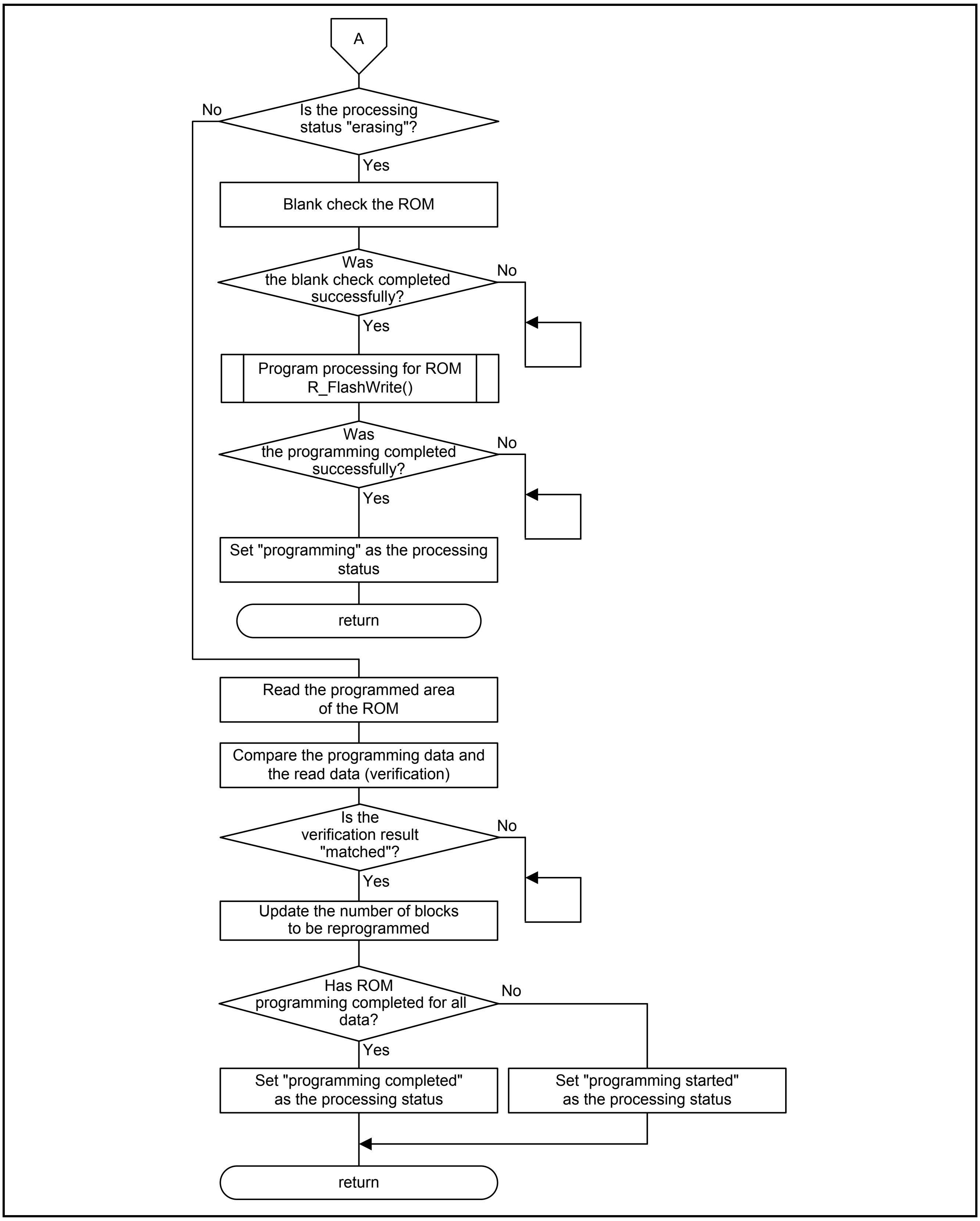
A
Is the processing status "erasing"?
No
Yes
Blank check the ROM
Was the blank check completed successfully?
No
Yes
Program processing for ROM R_FlashWrite()
Was the programming completed successfully?
No
Yes
Set "programming" as the processing status
return
Read the programmed area of the ROM
Compare the programming data and the read data (verification)
Is the verification result "matched"?
No
Yes
Update the number of blocks to be reprogrammed
Has ROM programming completed for all data?
No
Yes
Set "programming completed" as the processing status
Set "programming started" as the processing status
return

Figure 7.8 Reprogramming ROM (Subsequent Processing is Executed During Programming) (2/2)

7.8.6 Reprogramming ROM

Figure 7.9 shows the Reprogramming ROM.

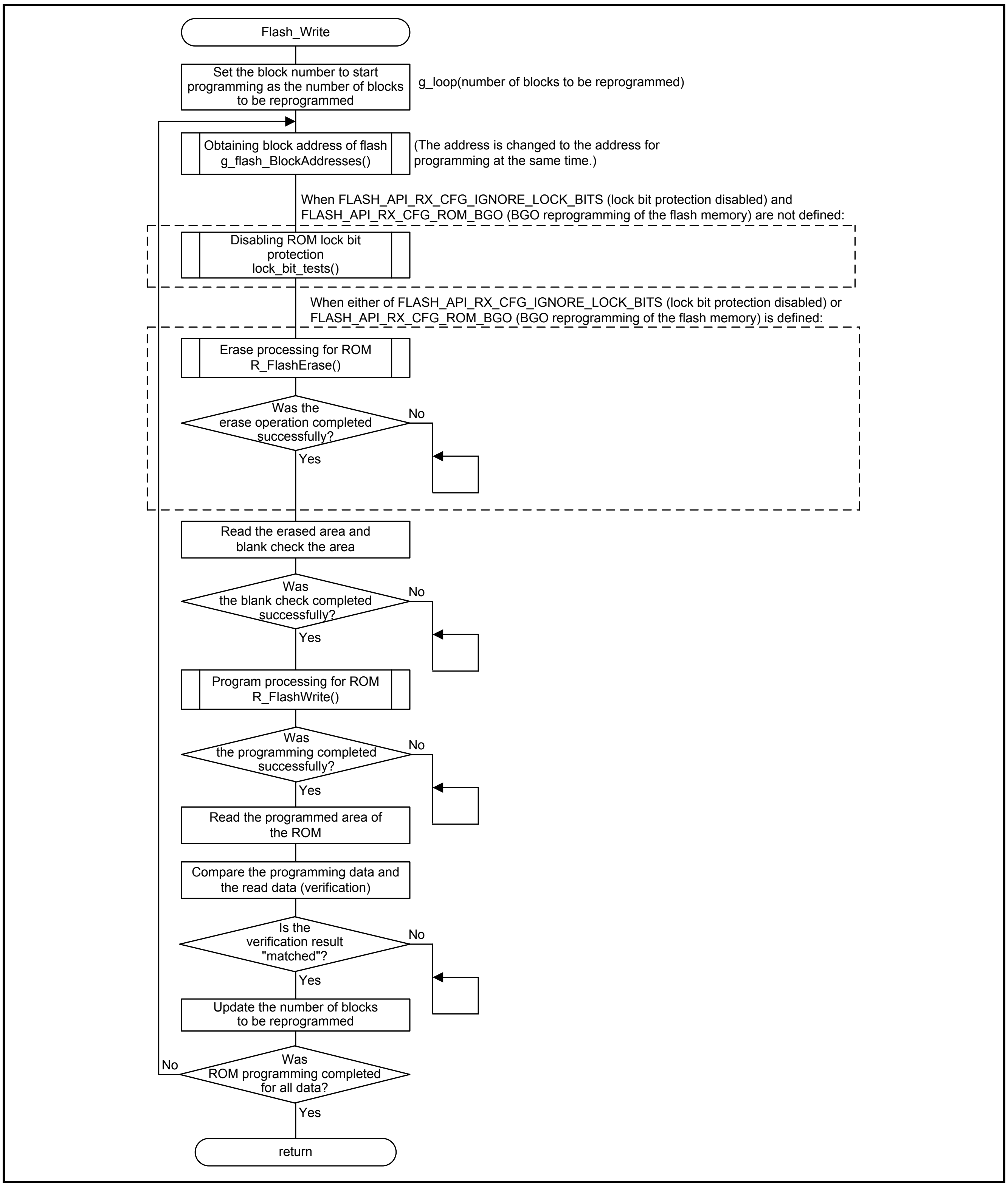
Flash_Write
Set the block number to start programming as the number of blocks to be reprogrammed
g_loop(number of blocks to be reprogrammed)
Obtaining block address of flash g_flash_BlockAddresses()
(The address is changed to the address for programming at the same time.)
When FLASH_API_RX_CFG_IGNORE_LOCK_BITS (lock bit protection disabled) and FLASH_API_RX_CFG_ROM_BGO (BGO reprogramming of the flash memory) are not defined:
Disabling ROM lock bit protection lock_bit_tests()
When either of FLASH_API_RX_CFG_IGNORE_LOCK_BITS (lock bit protection disabled) or FLASH_API_RX_CFG_ROM_BGO (BGO reprogramming of the flash memory) is defined:
Erase processing for ROM R_FlashErase()
Was the erase operation completed successfully?
No
Yes
Read the erased area and blank check the area
Was the blank check completed successfully?
No
Yes
Program processing for ROM R_FlashWrite()
Was the programming completed successfully?
No
Yes
Read the programmed area of the ROM
Compare the programming data and the read data (verification)
Is the verification result "matched"?
No
Yes
Update the number of blocks to be reprogrammed
Was ROM programming completed for all data?
No
Yes
return

Figure 7.9 Reprogramming ROM

7.8.7 Transferring Relocatable Vector to RAM

Figure 7.10 shows the Transferring Relocatable Vector to RAM.

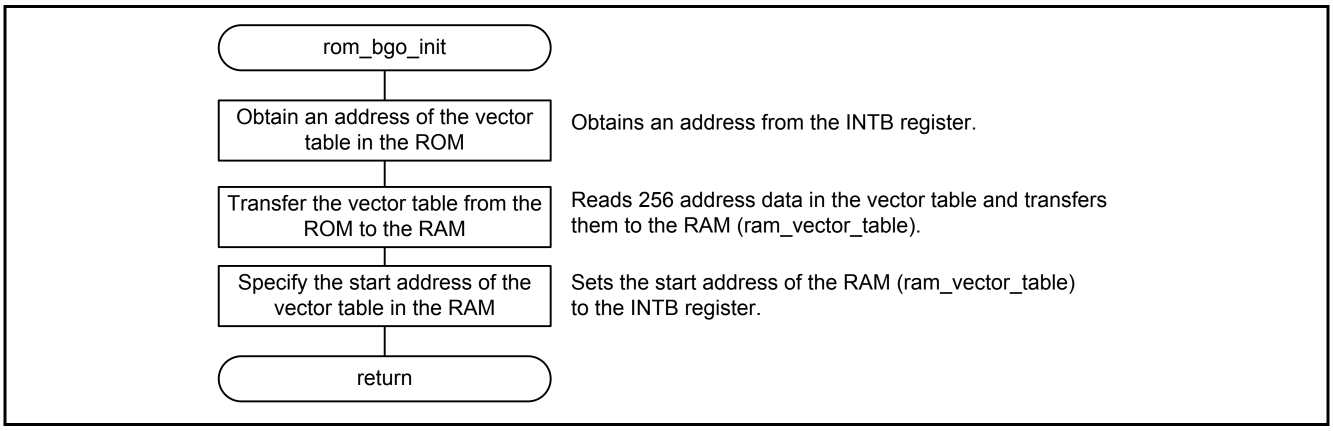
rom_bgo_init
Obtain an address of the vector table in the ROM
Obtains an address from the INTB register.
Transfer the vector table from the ROM to the RAM
Reads 256 address data in the vector table and transfers them to the RAM (ram_vector_table).
Specify the start address of the vector table in the RAM
Sets the start address of the RAM (ram_vector_table) to the INTB register.
return

Figure 7.10 Transferring Relocatable Vector to RAM

7.8.8 Disabling ROM Lock Bit Protection

Figure 7.11 shows the Disabling ROM Lock Bit Protection.

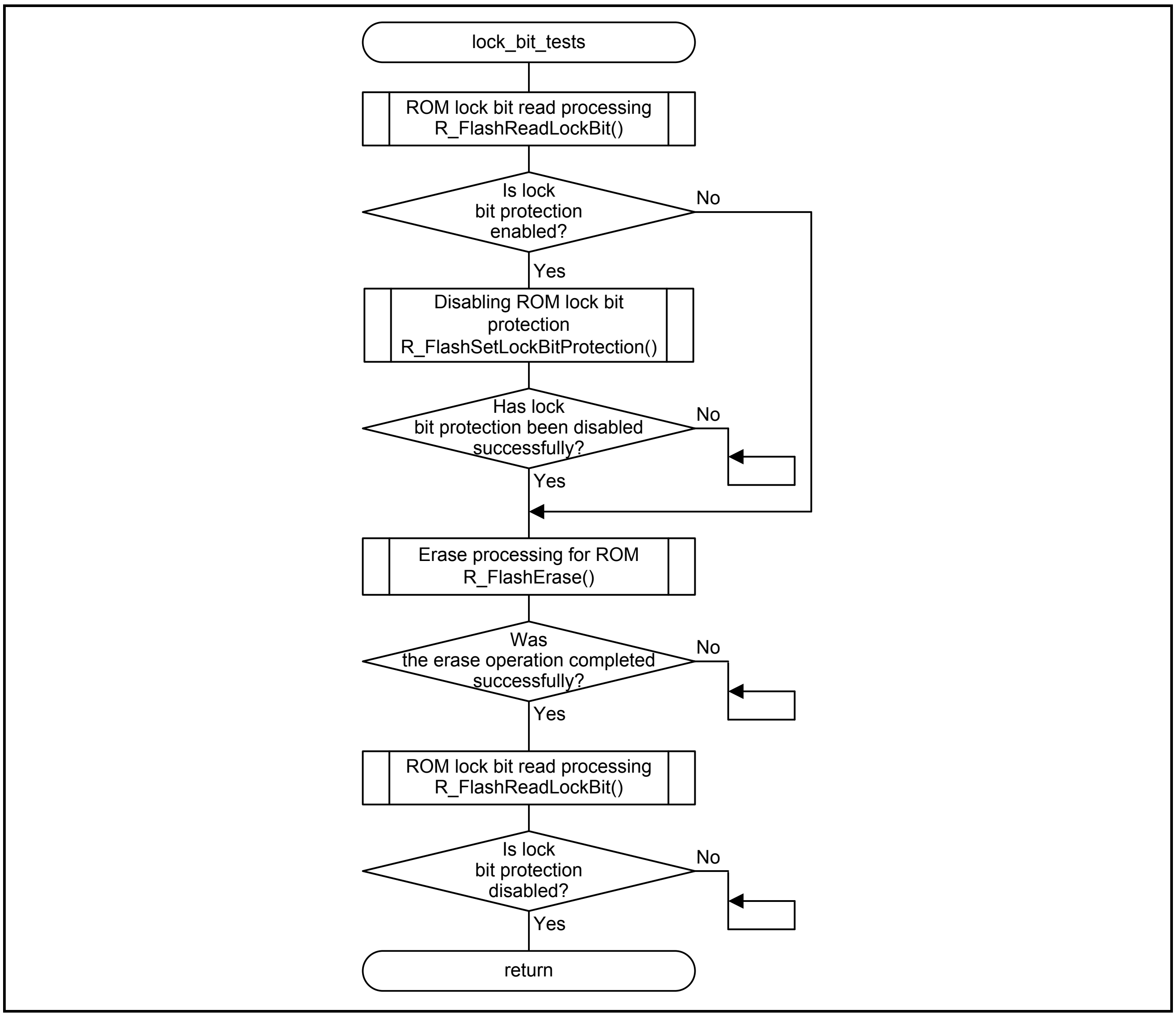
lock_bit_tests
ROM lock bit read processing R_FlashReadLockBit()
Is lock bit protection enabled?
No
Yes
Disabling ROM lock bit protection R_FlashSetLockBitProtection()
Has lock bit protection been disabled successfully?
No
Yes
Erase processing for ROM R_FlashErase()
Was the erase operation completed successfully?
No
Yes
ROM lock bit read processing R_FlashReadLockBit()
Is lock bit protection disabled?
No
Yes
return

Figure 7.11 Disabling ROM Lock Bit Protection

8. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

9. Reference Documents

User's Manual: Hardware

RX210 Group User's Manual: Hardware Rev.1.50 (R01UH0037EJ)
The latest version 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 Package V.1.01 User's Manual Rev.1.00 (R20UT0570EJ)
The latest version can be downloaded from the Renesas Electronics website.

Website and Support

Renesas Electronics website

http://www.renesas.com

Inquiries

http://www.renesas.com/contact/

REVISION HISTORY	RX600 & RX200 Series Application Note Reprogramming the On-Chip Flash Memory Using Simple Flash API for RX

Rev.	Date	Description	
		Page	Summary
1.00	Sep. 22, 2014	—	First edition issued

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 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 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 part 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.

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