RL78/G23

Updating Firmware by Using UART Communication and Boot Swapping

Introduction

This application note describes how to update firmware in code flash memory by using an update program that remains in the code flash memory.

In this method, the code flash memory is divided into two areas: the Execute area and the Temporary area.

Renesas Flash Driver RL78 Type01 is used to reprogram the flash memory and perform boot swapping.

Target Device

RL78/G23

When applying the sample program covered in this application note to another microcomputer, modify the program according to the specifications for the target microcomputer and conduct an extensive evaluation of the modified program.

Contents

1. Specifications

1.1 Overview of Specifications

The sample program covered in this application note updates the firmware in the code flash memory.

The boot area is reprogrammed by using the boot swapping function. The other areas are reprogrammed by using temporary areas in which the reprogramming data is temporarily saved. This method allows the firmware to be updated while the user program (application) is running.

The firmware is updated via UART communication by using four commands: START, WRITE_BOOT1, WRITE_TEMP, and END.

The execution status of the application and commands is indicated by LEDs.

Two sample projects are included in this application note, each can be replaced by firmware updates.

If you use a product with ROM size other than 128 KB or 768 KB, please refer to “1.2.8 Copy Flag” and “4.3 Setting the ROM Size Specification Constant” and modify the sample programs.

Table 1-1 Directory of Sample Project

workspace				**Description**
\workspace				
	\CS+ \e2studio \IAR			
		\128KB		Project for 128KB products
			\LED1	Sample project 1 (Blinks LED1)
			\LED8	Sample project 2 (Blinks LED8)
		\768KB		Project for 768KB products
			\LED1	Sample project 1 (Blinks LED1)
			\LED8	Sample project 2 (Blinks LED8)

LED output port assign differ between the project for 128KB and the project for 768KB. In this application note, in the case of using the project for 128 KB is explained as an example. When using the project for 768 KB, please read the port numbers as shown in the table below.

Table 1-2 Assigned port for LED output

LED no.	Project for 128KB products	Project for 768KB products
LED1	P03	P33
LED2	P02	P34
LED3	P43	P145
LED4	P42	P106
LED5	P77	P105
LED6	P41	P104
LED7	P31	P103
LED8	P76	P46

Table 1-3 Peripheral Function and Use

Peripheral Function	Use
Serial Array Unit UART0	Data communication
P03, P02, P43, P42, P77, P41, P31, P76	Digital output controlling LED1 to LED8

Table 1-4 Application operating state and indication on LED1 to LED8. (Updating sample project 1 to 2)

Application operating state	Indication on LED1 to LED8	Operating firmware
Application before updated is running	LED1 blinks	Sample project 1
START command received	LED2 lights up	
WRITE_BOOT1 command received	LED3 lights up	
WRITE_TEMP command received	LED4 lights up	
END command received	LED5 lights up	
Temporary area being copied	LED6 lights up	
Error termination	Only LED7 lights up	
Application after updated is running	LED8 blinks	Sample project 2

1.1.1 Overview of Renesas Flash Driver RL78 Type01

Renesas Flash Driver RL78 Type01 is software that reprograms the firmware in the code flash memory installed on an RL78 microcontroller.

The content of the code flash memory can be reprogrammed by calling Renesas Flash Driver RL78 Type01 from the user program.

To perform flash memory self-programming, the user program needs to perform the necessary initialization processing and run the functions that correspond to the necessary operations in C or assembly language.

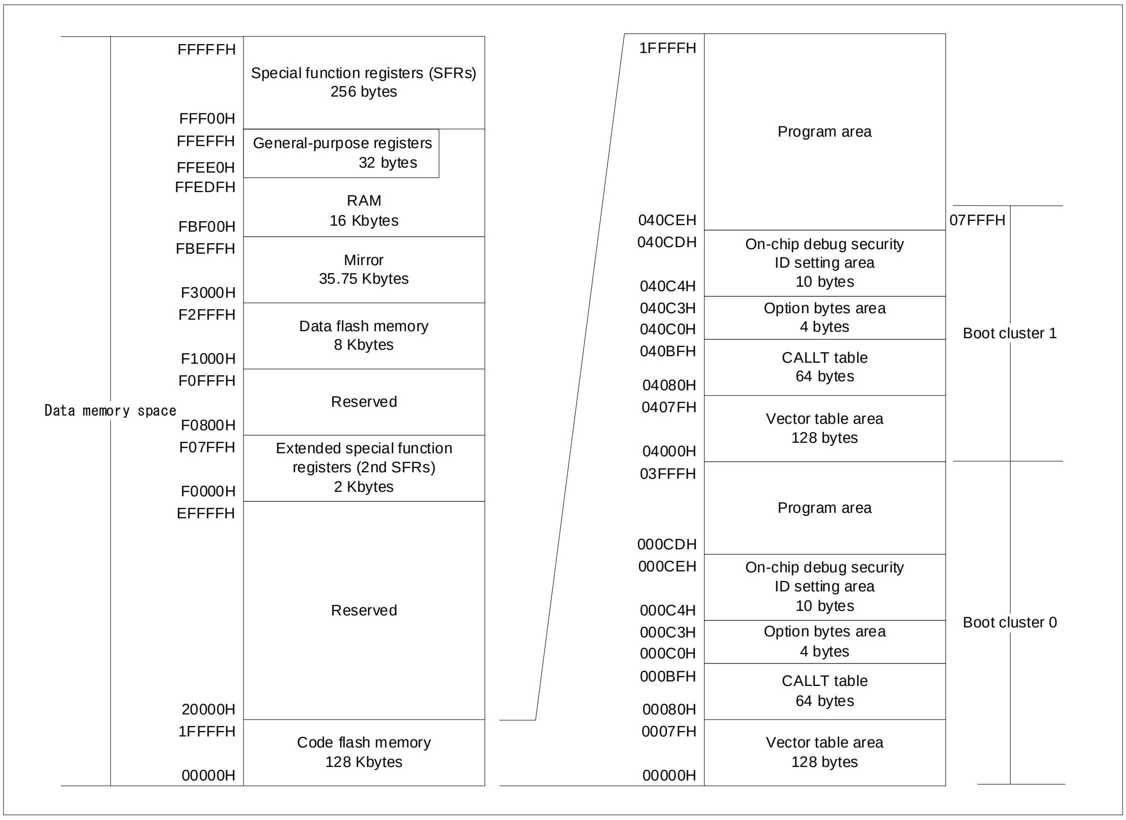
1.1.2 Code Flash Memory

The program area (from address 08000H to the last address) is divided into two areas and the sample program covered in this application note uses these two areas. The first area (from address 08000H to the boundary) is called the Execute area and the second area (from the boundary to the last address) is called the Temporary area. The address of the boundary and the last address differ depending on the size of the ROM. The update program is written in boot cluster 1 and the Temporary area. Therefore, if you write a user program, make sure that it is stored within boot cluster 0 and the Execute area.

Table 1-5 Start and End Addresses of the Two Areas According to ROM Size

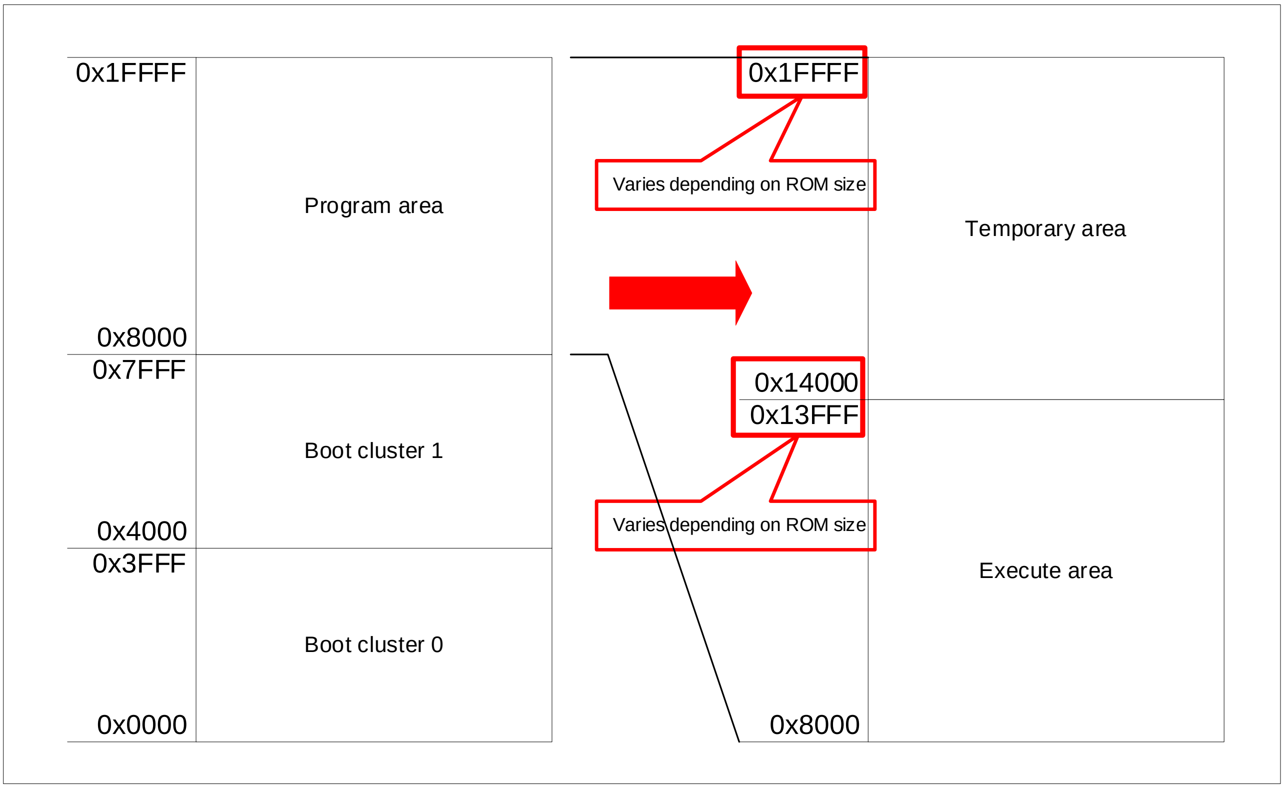
ROM size	Execute area	Temporary area
96KB	08000H to FFFFH	10000H to 17FFFH
128KB	08000H to 13FFFH	14000H to 1FFFFH
192KB	08000H to 1BFFFH	1C000H to 2FFFFH
256KB	08000H to 23FFFH	24000H to 3FFFFH
384KB	08000H to 33FFFH	34000H to 5FFFFH
512KB	08000H to 43FFFH	44000H to 7FFFFH
768KB	08000H to 63FFFH	64000H to BFFFFH

Figure 1-1 Memory Map



Caution: If you use the boot swap function, make sure that the same value that is set in the option byte area in boot cluster 0 (000C0H to 000C3H) is also set in the option byte area in boot cluster 1 (010C0H to 010C3H) because these areas are swapped by the function.

Figure 1-2 Code Flash Memory Map



The following table summarizes the features of the code flash memory of the RL78/G23 microcontroller.

Table 1-6 Features of the Code Flash Memory

Item	Description
Minimum unit of erasure	1 block (2,048 bytes)
Minimum unit of writing	1 word (4 bytes)
Minimum unit of verification	1 byte
Security functions	The functions for protection against erasure of blocks, writing to blocks, and reprogramming of the boot area are provided. (All these functions are disabled in the factory settings.)
	The flash shield window is provided, which can protect all area except the specified window range from write and erasure operations during flash memory self-programming only.
	Renesas Flash Driver RL78 Type01 can be used to change the security settings.

Caution: The security functions that are available during flash memory self-programming are only protection against reprogramming of the boot area and the flash shield window.

1.1.3 Flash Memory Self-Programming

The RL78/G23 microcontroller is provided with a library required for performing flash memory self-programming. Flash memory self-programming can be performed by calling functions of Renesas Flash Driver RL78 Type01 from the reprogramming program.

The RL78/G23 microcontroller has a sequencer, which is a circuit that only controls the flash memory. The flash memory self-programming in the RL78/G23 microcontroller uses the sequencer to control the reprogramming of the flash memory. Note that the code flash memory cannot be read while it is being controlled by the sequencer. However, the user program may need to operate while the sequencer is controlling the code flash memory. In such a case, when erasure and write operations are performed and security flags are set for the code flash memory, certain Renesas Flash Driver RL78 Type01 segments or the reprogramming program must be relocated to the RAM. If the user program does not need to run while the sequencer is controlling the code flash memory, Renesas Flash Driver RL78 Type01 and the reprogramming program located on the ROM (code flash memory) can run without relocation.

1.1.4 Boot Swap Function

If the reprogramming of the area in which any of following items are located fails for reasons such as a temporary blackout or reset due an external factor, the data being reprogrammed is corrupted: vector table data, basic program functions, and Renesas Flash Driver RL78 Type01. If data corruption occurs, the user program can no longer be restarted or reloaded by performing a reset. This problem can be prevented by using the boot swap function.

The boot swap function swaps the boot program area (boot cluster 0) with the swap area (boot cluster 1). Before reprogramming starts, the boot swap function writes a new boot program boot cluster 1. The function then swaps boot cluster 0 with boot cluster 1, causing boot cluster 1 to become the boot program area. This ensures that the boot program can normally be started when a reset is performed the next time even if a temporary blackout occurs while the boot program area is being reprogrammed because boot cluster 1 is used to boot the program.

1.1.5 Updating the Firmware

The following shows an overview of how a program is rewritten by flash memory self-programming. The program that performs flash memory self-programming is deployed in boot cluster 0.

The sample program covered in this application note is designed to reprogram the boot area and program area.

Figure 1-3 Rewriting operation image (1/2)

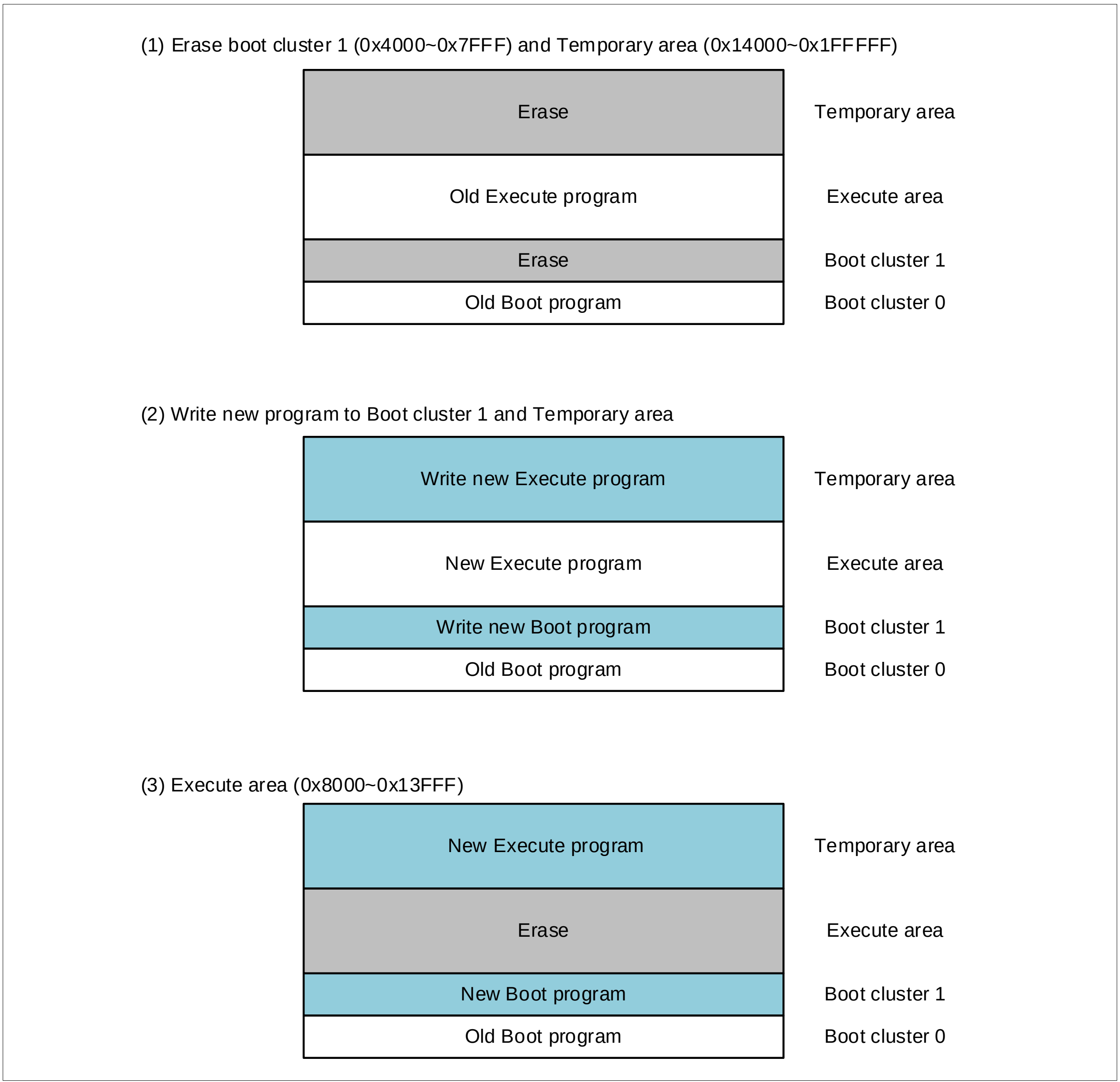
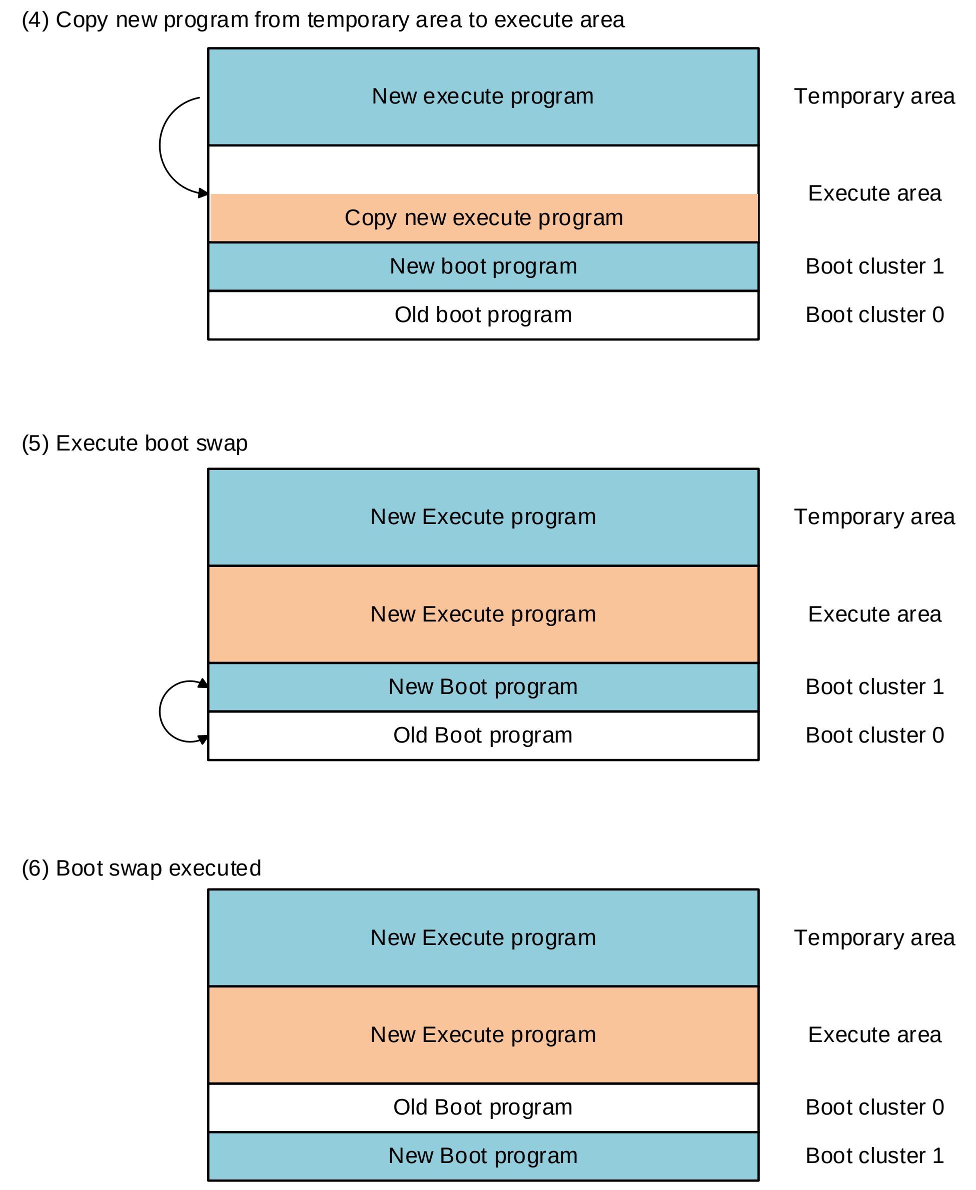


Figure 1-4 Rewriting operation image (2/2)

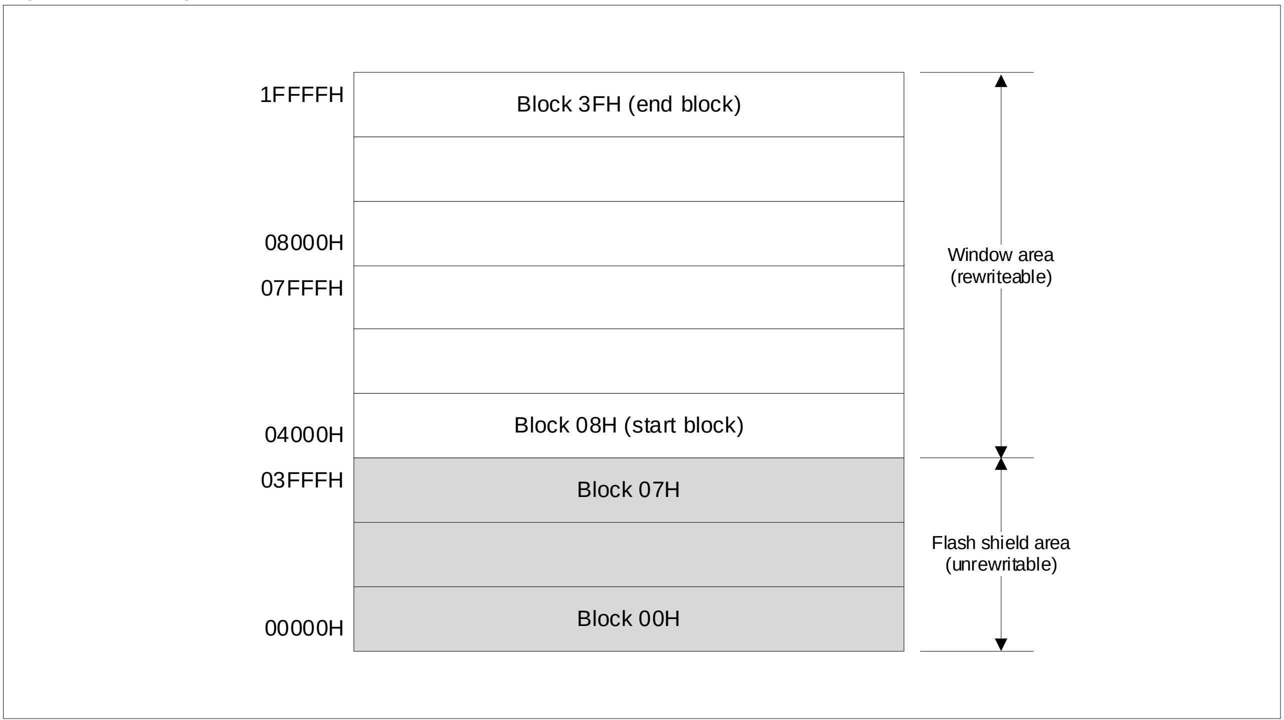


1.1.6 Flash Shield Window

The flash shield window is a security function available during flash memory self-programming. This function protects all areas except the specified window range from the write and erase operations during flash memory self-programming only.

The following figure is an overview of the flash shield window when the start block is 08H and the end block is 1FH.

Figure 1-5 Image of a flush shield window



1.1.7 Obtaining Renesas Flash Driver RL78 Type01

Before you compile the sample program, download the latest version of flash memory self-programming code (Renesas Flash Driver RL78 Type01), and then copy it to the RFD folder.

workspace				**Description**
r01an6255jj0100-rl78g23-flash				
	\src			
		\RFD		
			\include	Place the downloaded Renesas Flash Driver RL78 Type01
			\source	
			\userown	

You can obtain the Renesas Flash Driver RL78 Type01 from the following URL:

https://www.renesas.com/jp/ja/document/scd/renesas-flash-driver-rl78-type-01-rl78g23

1.2 Overview of Operation

(1) Perform initial setup for pins.

・Set the P03, P02, P43, P42, P77, P41, P31, and P76 pins to output mode.

(2) Perform initial setup for the serial array unit.

- Use the UART0 serial array unit (set TXD0 for P12 and RXD0 for P11).
- Set CK00 for the operation clock and fCLK/2 for the clock source.
- Set the clock source for the transfer mode settings.
- Set 8 bits for the data bit length settings.
- Set LSB for the data transfer direction settings.
- Set "no parity" for the parity settings.
- Set 1 bit for the stop bit length settings.
- Set "standard" for the send data level settings.
- Set 115,200 bps for the baud rate settings.

(3) Use command communication to reprogram the data in boot cluster 1 and the program area, and then perform boot swapping.

1.2.1 Communication Specifications

The sample program covered in this application note receives the reprogramming data via UART and performs flash memory self-programming. The sample program then receives the START, WRITE_BOOT1, WRITE_TEMP, or END command. The sample program then performs the processing according to the received command. If the processing terminates normally, the sample program returns "01H" (normal) to the command sender. If the processing terminates abnormally, the sample program turns on the LED that indicates abnormal termination (without returning a response) and performs no subsequent processing. The following shows the UART communication settings and the specifications of the commands.

Table 1-7 UART Communication Settings

Data bit length (bits)	8
Data transfer direction	LSB first
Parity setting	No parity
Transfer rate (bps)	115,200

1.2.2 START Command

When the sample program receives the START command, it performs initial setup for flash memory self-programming and erases the Temporary area in boot cluster 1. If the processing terminates normally, the sample program returns "01H" (normal). If the processing terminates abnormally, the sample program turns on the LED that indicates abnormal termination (without returning a response) and performs no subsequent processing.

START code (01H)	Data length (0002H)	Command (02H)	Data (empty)	Checksum (1 byte)

1.2.3 WRITE_BOOT1 Command

When the sample program receives the WRITE_BOOT1 command, it writes the received data to the boot cluster 1 area (4000H to 7FFFH) while verifying the written data for each 256 bytes. If the processing terminates normally, the sample program increments the write destination address by 256 bytes and returns "01H" (normal) to the command sender. If the processing terminates abnormally, the sample program turns on the LED that indicates abnormal termination (without returning a response) and performs no subsequent processing.

START code (01H)	Data length (0102H)	Command (03H)	Data (256 bytes)	Checksum (1 byte)

1.2.4 WRITE_TEMP Command

When the sample program receives the WRITE_TEMP command, it writes the received data to the Temporary area while verifying the written data for each 256 bytes. If the processing terminates normally, the sample program increments the write destination address by 256 bytes and returns "01H" (normal) to the command sender. If the processing terminates abnormally, the sample program turns on the LED that indicates abnormal termination (without returning a response) and performs no subsequent processing.

(The write destination address of the Temporary area differs depending on the product used.)

START code (01H)	Data length (0102H)	Command (04H)	Data (256 bytes)	Checksum (1 byte)

1.2.5 END Command

When the sample program receives the END command, it erases the Execute area. If erasure terminates normally, the sample program copies data from the Temporary area to the Execute area. If copy terminates normally, the sample program returns "01H" (normal) to the command sender. The sample program then reverses the boot flag to cause a reset to occur and performs boot swapping.

START code (01H)	Data length (0002H)	Command (05H)	Data (empty)	Checksum (1 byte)

1.2.6 Checksum Calculation Method

For checksum calculation, the 32-bit addition method is used. This method uses as a checksum the last 8 bits of the result of adding a 1-byte value from address 00000000H for the command or data.

1.2.7 Operation of the Sample Program

The following shows the operation of this sample program:

(1) Set up the input and output ports.

(2) Perform initial setup for SAU0 channel 0.

(3) Wait for data to be sent from the command sender.

(4) Upon receiving the START command, perform initial setup for self-programming.

(5) Set the P02 pin for high-level output to turn on LED2, which indicates that the START command was received.

(6) Call the r_CF_EraseBlock function to erase boot cluster 1.

(7) Call the r_CF_EraseBlock function to erase the data in the Temporary area.

(8) Send "01H" (normal) to the command sender.

(9) Set the P02 pin for low-level output to turn off LED2, which indicates that the START command was received.

(10) Receive the WRITE_BOOT1 command (03H) and write data (256 bytes).

(11) Set the P43 pin for high-level output to turn on LED3, which indicates that the WRITE_BOOT1 command was received.

(12) Call the r_CF_WriteData function to write the received data to the write destination address (local variable for writing to boot cluster 1). The initial value of the local variable for writing to boot cluster 1 is the start address of boot cluster 1.

(13) Call the r_CF_VerifyData function to verify the written data against the received data.

(14) Add a 256-byte checksum to the write destination address (local variable for writing to boot cluster 1).

(15) Send "01H" (normal) to the command sender.

(16) Set the P43 pin for low-level output to turn off LED3, which indicates that the WRITE_BOOT1 command was received.

(17) Repeat steps (11) to (17) until receiving the WRITE_TEMP command (04H).

(18) Receive the WRITE_TEMP command (04H) and write data (256 bytes).

(19) Set the P42 pin for high-level output to turn on LED4, which indicates that the WRITE_BOOT1 command was received.

(20) Call the r_CF_WriteData function to write the received data to the write destination address (local variable for writing to the Temporary area). The initial value of the local variable for writing to the Temporary area is the start address of the Temporary area.

(21) Call the r_CF_VerifyData function to verify the written data against the received data.

(22) Add a 256-byte checksum to the write destination address (local variable for writing to the Temporary area).

(23) Send "01H" (normal) to the command sender.

(24) Set the P42 pin for low-level output to turn off LED4, which indicates that the WRITE_TEMP command was received.

(25) Repeat steps (19) to (25) until receiving the END command (05H).

(26) Perform the following processing if receiving the END command:

(27) Set the P77 pin for high-level output to turn on LED5, which indicates that the END command was received.

(28) Call the r_CF_EraseBlock function to erase the data in the Execute area.

(29) Set the P41 pin for high-level output to turn on LED6, which indicates that copy to the Temporary area is in progress.

(30) Call the r_temp_copy function to copy data from the Temporary area to the Execute area.*

(31) Set the P41 pin for low-level output to turn off LED6, which indicates that copy to the Temporary area is in progress.

(32) Send "01H" (normal) to the command sender.

(33) Call the r_RequestBootSwap function to reverse the value of the boot flag so that boot clusters 0 and 1 are swapped when a reset occurs. Cause an internal reset to occur.

* If a reset occurs (due to a temporary blackout, for example) while data is being copied from the Temporary area to the Execute area, the r_temp_copy function is called again. The r_RequestBootSwap function is called after the copy is complete.

Caution: If the sample program receives the END command (05H) in steps (10) to (17), the sample program copies data from the Temporary area to the Execute area unless there is an error. Then, the sample program sends "01H" (normal), calls the r_RequestBootSwap function, and performs boot swapping. If the sample program receives the END command (05H) while boot cluster 1 is being reprogrammed, the sample program performs boot swapping before the reprogramming ends normally. In this case, the sample program can no longer start after the boot area is swapped.

Caution: If flash memory self-programming does not end normally, the sample program only turns on LED6 and performs no subsequent processing.

1.2.8 Copy Flag

The sample program covered in this application note uses a 4-byte area at the end of the Execute area as the copy flag section in which to set a copy flag.

If a program is normally written, this copy flag is set to AAAA5555H. The copy flag is initialized when the Execute area is erased immediately before data is copied from the Temporary area to the Execute area. If a reset occurs (due to a temporary blackout) while data is being written, the copy flag is set to a value other than AAAA5555H because the write processing does not terminate normally.

When the sample program starts, it checks the copy flag. If the value of the copy flag is not AAAA5555H, the sample program writes data and then performs swapping.

The following table shows the start and end addresses of the Execute area and the address of the copy flag section according to the ROM size.

Table 1-8 Location of the Copy Flag Section According to the ROM Size

ROM Size	Execute Area	Address of the Copy Flag Section
96KB	08000H to FFFFH	FFFCH
128KB	08000H to 13FFFH	13FFCH
192KB	08000H to 1BFFFH	1BFFCH
256KB	08000H to 23FFFH	23FFCH
384KB	08000H to 33FFFH	33FFCH
512KB	08000H to 43FFFH	43FFCH
768KB	08000H to 63FFFH	63FFCH

2. Operation Confirmation Conditions

The operation of the sample code provided with this application note has been tested under the following conditions.

Table 2-1 Operation Confirmation Conditions

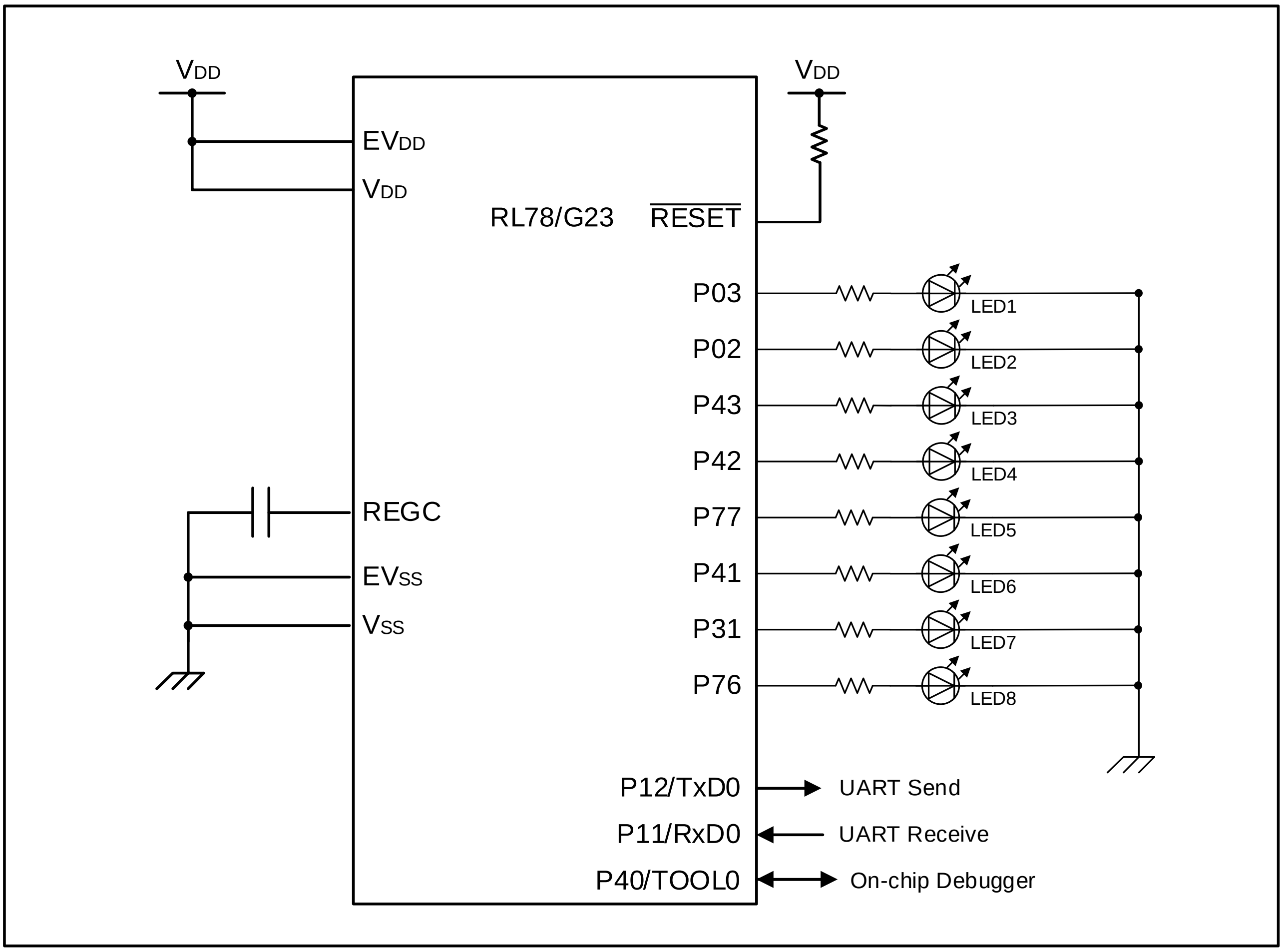
Item	Description
MCU used	RL78/G23 (R7F100GLG)
Board used	[Project for 128KB products] RL78/G23-64p Fast Prototyping Board (RTK7RLG230CLG000BJ) [Project for 768KB products] RL78/G23-128p Fast Prototyping Board (RTK7RLG230CSN000BJ)
Operation frequency	High-speed on-chip oscillator clock (fIH): 32MHz
Operating voltage	3.3V (can be operated at 3.1V to 3.5V) LVD operation (V_{LVD}): Reset mode At rising edge TYP. 1.90 V (1.84 V to 1.95 V) At falling edge TYP. 1.86 V (1.80 V to 1.91 V)
Integrated development environment (CS+)	CS+ for CC V8.06.00 from Renesas Electronics Corp.
C compiler (CS+)	CC-RL V1.10.00 from Renesas Electronics Corp.
Integrated development environment (e2studio)	e2studio V2021-10 from Renesas Electronics Corp.
C compiler (e2studio)	CC-RL V1.10.00 from Renesas Electronics Corp.
Integrated development environment (IAR)	IAR Embedded Workbench for Renesas RL78 V4.21.1 from IAR Systems Corp.
C compiler (IAR)	IAR C/C++ Compiler for Renesas RL78 V 4.21.1.2409 from IAR Systems Corp.

3. Hardware Descriptions

3.1 Example of Hardware Configuration

Figure 3-1 shows an example of the hardware configuration used in the application note.

Figure 3-1 Hardware Configuration



Cautions:1. The purpose of this circuit is only to provide the connection outline and the circuit is simplified accordingly. When designing and implementing an actual circuit, provide proper pin treatment and make sure that the hardware's electrical specifications are met (connect the input-only ports separately to V_{DD} or V_{SS} via a resistor).

2. Connect any pins whose name begins with EV_{SS} to V_{SS} and any pins whose name begins with EV_{DD} to V_{DD}, respectively.

3. V_{DD} must be held at not lower than the reset release voltage (V_{LVD}) that is specified as LVD.

3.2 List of Pins to be Used

Table 3.1 lists the pins to be used and their functions.

Table 3-1 Pins to be Used and their Functions

Pin	Input/Output	Description
P12//TxD0	Output	UART serial data transmit pin
P11/ RxD0	Input	UART serial data receive pin
P03, P02, P43, P42, P77, P41, P31, P76	Output	LED1-LED8 control pins

Caution: In this application note, only the used pins are processed. When actually designing your circuit, make sure the design includes sufficient pin processing and meets electrical characteristic requirements.

4. Software Explanation

4.1 Setting of Option Byte

Table 4-1 shows the option byte settings.

Table 4-1 Option Byte Settings

Address	Setting Value	Description
000C0H/040C0H	11101111B	Disables the watchdog timer. (Counting stopped after reset)
000C1H/040C1H	11111110B	LVD operation (V_{LVD}): Reset mode At rising edge TYP. 1.90 V (1.84 V to 1.95 V) At falling edge TYP. 1.86 V (1.80 V to 1.91 V)
000C2H/040C2H	11101000B	HS mode, High-speed on-chip oscillator clock (f_{IH}): 32 MHz
000C3H/040C3H	10000101B	Enables on-chip debugging

The option bytes of the RL78/G23 comprise the user option bytes (000C0H to 000C2H) and on-chip debug option byte (000C3H).

The option bytes are automatically referenced and the specified settings are configured at power-on time or the reset is released. When using the boot swap function for self-programming, it is necessary to set the same values that are set in 000C0H to 000C3H also in 040C0H to 040C3H because the bytes in 000C0H to 000C3H are swapped with the bytes in 040C0H to 040C3H.

4.2 Setting Up the Startup Routine

4.2.1 Defining the Stack Area Section (.stack_bss)

Define the stack area section (.stack_bss).

In the startup routine configuration file (cstart.asm), change the settings as follows:

;$IF (__RENESAS_VERSION__ < 0x01010000)	Comment out the line by prefixing a semicolon (;).
;-- ; ;-- ; !!! [CAUTION] !!! ; Set up stack size suitable for a project. .SECTION .stack_bss, BSS _stackend:	
.DS 0x800	Specify any stack size of your choice by using a hexadecimal number.
_stacktop:	
;$ENDIF	Comment out the line by prefixing a semicolon (;).
;-- ; setting the stack pointer ;-- $IF (__RENESAS_VERSION__ >= 0x01010000)	
;MOVW SP,#LOWW(__STACK_ADDR_START)	Comment out the line by prefixing a semicolon (;).
;$ELSE ; for CC-RL V1.00	Comment out the line by prefixing a semicolon (;).
MOVW SP,#LOWW(_stacktop) $ENDIF	
;-- ; initializing stack area ;-- $IF (__RENESAS_VERSION__ >= 0x01010000)	
;MOVW AX,#LOWW(__STACK_ADDR_END)	Comment out the line by prefixing a semicolon (;).
;$ELSE ; for CC-RL V1.00	Comment out the line by prefixing a semicolon (;).
MOVW AX,#LOWW(_stackend) $ENDIF CALL !!_stkinit	

4.2.2 Deploying the Reprogramming Program in the RAM Area

Confirm the programs that are used to reprogram the firmware and deploy them to the RAM area.

Table4-2 shows the sections where the programs used to reprogram the firmware exist and the sections in which the programs are to be deployed.

Table4-2 Section Information

Section	Destination Section	Description
RFD_CMN_f	RFD_CMN_fR	Program section for the API functions that control the common flash memory
RFD_CF_f	RFD_CF_fR	Program section for the API functions that control the code flash memory
RFD_EX_f	RFD_EX_fR	Program section for the API functions that control the extra area
SMP_CMN_f	SMP_CMN_fR	Program section for the sample functions that control the common flash memory
SMP_CF_f	SMP_CF_fR	Program section for the sample functions that control the code flash memory

To deploy the preceding sections in the RAM area, you must add the necessary processing to the cstart.asm file.

In the cstart.asm file, locate the following lines, and then add the necessary processing after these lines:

```
;--------------------------------------------------
; ROM data copy
;--------------------------------------------------
```

The following are the details to be added.

```
    ; copy .text to RAM (section-name)
    MOV     C,#HIGHW(STARTOF(section-name))
    MOVW    HL,#LOWW(STARTOF(section-name))
    MOVW    DE,#LOWW(STARTOF(destination-section-name))
    BR      $.Lm2_TEXT
.Lm1_TEXT:
    MOV     A,C
    MOV     ES,A
    MOV     A,ES:[HL]
    MOV     [DE],A
    INCW    DE
    INCW    HL
    CLRW    AX
    CMPW    AX,HL
    SKNZ
    INC
.Lm2_TEXT:
    MOVW    AX,HL
    CMPW    AX,#LOWW(STARTOF(section-name) + SIZEOF(section-name))
    BNZ     $.Lm1_TEXT
```

Note 1. For ***section-name***, specify the name of the section to be deployed.

Note 2. Add the preceding set of entries for each section to be deployed.

Note 3. For m, specify any numeric value of your choice. Make sure that you specify a different value for each section.

4.3 Setting the ROM Size Specification Constant

Conditional compilation allows this sample program to support several ROM sizes for the RL78/G23 microcontroller.

The following table lists the constants that correspond to the supported ROM sizes. In the r_cg_userdefine.h file, these constants are commented out. Enable the constant for the installed ROM by uncommenting it.

Table4-3 Constants for the Supported ROM Sizes

Constant Name	Supported ROM Size
ROM_SIZE_96KB	96-KB product
ROM_SIZE_128KB	128-KB product
ROM_SIZE_192KB	192-KB product
ROM_SIZE_256KB	256-KB product
ROM_SIZE_384KB	384-KB product
ROM_SIZE_512KB	512-KB product
ROM_SIZE_768KB	768-KB product

4.4 On-chip Debug Security ID

The RL78/G23 microcontroller provides the on-chip debug security ID area at addresses 000C4H to 000CDH in the flash memory so that the memory content is not read by third parties.

If boot swapping is performed during self-programming, the area at addresses 000C4H to 000CDH and the area at addresses 010C4H to 010CDH are swapped. Therefore, the same value that is set in the area at 000C4H to 000CDH must also be set in the area at 040C4H to 040CDH.

4.5 Resources Used by the Sample Program

4.5.1 List of Sections in the ROM Area

Table4-4 lists the sections that the sample program uses in the ROM area.

Table4-4 List of the Sections in the ROM Area

Section Name	Description
RFD_DATA_n	Data section for RFD RL78 Type01
RFD_CMN_f	Program section for the API functions that control the common flash memory
RFD_CF_f	Program section for the API functions that control the code flash memory
RFD_EX_f	Program section for the API functions that control the extra area
RFD_DF_f	Program section for the API functions that control the data flash memory
SMP_CMN_f	Program section for the sample functions that control the common flash memory
SMP_CF_f	Program section for the sample functions that control the code flash memory
BOOT_AREA1	Program section for boot cluster 1
USER_APPLICATION	Program section for the user application
COPY_FLAG_f	Program section for storing the copy completion flag
TEMPORARY_AREA	Program section for storing the receive data

4.5.2 List of the Sections in the RAM Area

Table4-5 lists the sections that the sample program uses in the RAM area.

Table4-5 List of the Sections in the RAM Area

Section Name	Description
RFD_DATA_nR	Data section for RFD RL78 Type01
RFD_CMN_fR	Program section for the API functions that control the common flash memory
RFD_CF_fR	Program section for the API functions that control the code flash memory
RFD_EX_fR	Program section for the API functions that control the extra area
SMP_CMN_fR	Program section for the sample functions that control the common flash memory
SMP_CF_fR	Program section for the sample functions that control the code flash memory

4.6 List of Constants

Table4-6 and Table4-7 list the constants that are used in the sample program.

Table4-6 List of Constants (1/2)

Constant Name	Value Set By This Constant	Description
ROM_SIZE_96KB	01H	Value that sets the ROM size to 96 KB
ROM_SIZE_128KB	01H	Value that sets the ROM size to 128 KB
ROM_SIZE_192KB	01H	Value that sets the ROM size to 192 KB
ROM_SIZE_256KB	01H	Value that sets the ROM size to 256 KB
ROM_SIZE_384KB	01H	Value that sets the ROM size to 384 KB
ROM_SIZE_512KB	01H	Value that sets the ROM size to 512 KB
ROM_SIZE_768KB	01H	Value that sets the ROM size to 768 KB
LED_ON	01H	LED ON
LED_OFF	00H	LED OFF
WRITE_DATA_SIZE	0100H	Size of data written to the code flash memory (256 bytes)
CF_BLOCK_SIZE	0800H	Block size of the code flash memory (2,048 bytes)
BT1_START_ADDRESS	00004000H	Start address of boot cluster 1
BT1_END_ADDRESS	00007FFFH	End address of boot cluster 1
EXECUTE_START_ADDRESS	00008000H	Start address of the Execute area
EXECUTE_END_ADDRESS[Note]	00013FFFH	End address of the Execute area
TEMPORARY_START_ADDRESS[Note]	00014000H	Start address of the Temporary area
TEMPORARY_END_ADDRESS[Note]	0001FFFFH	End address of the Temporary area
CPU_FREQUENCY	32	CPU operating frequency
COMMAND_START	02H	Command code for the START command
COMMAND_WRITE_BOOT1	03H	Command code for the WRITE_BOOT1 command
COMMAND_WRITE_TEMP	04H	Command code for the WRITE_TEMP command
COMMAND_END	05H	Command code for the END command
VALUE_U08_MASK1_FSQ_STATUS_ERR_ERASE	01H	Error status mask value for the execution results of the flash memory sequencer Bit 0: Erase command error
VALUE_U08_MASK1_FSQ_STATUS_ERR_WRITE	02H	Error status mask value for the execution results of the flash memory sequencer Bit 1: Write command error
VALUE_U08_MASK1_FSQ_STATUS_ERR_BLANKCHECK	08H	Error status mask value for the execution results of the flash memory sequencer Bit 3: Blank check command error
VALUE_U08_MASK1_FSQ_STATUS_ERR_CFDF_SEQUENCER	10H	Error status mask value for the execution results of the flash memory sequencer Bit 4: Code/data flash area sequencer error
VALUE_U08_MASK1_FSQ_STATUS_ERR_EXTRA_SEQUENCER	20H	Error status mask value for the execution results of the flash memory sequencer Bit 5: Extra area sequencer error

Note: The address differs depending on the product used.

Table4-7 List of Constants (2/2)

VALUE_U08_SHIFT_ADDR_TO_BLOCK_CF	11	Constant for bit shifting performed for calculating the block number of the code flash memory
VALUE_U01_MASK0_1BIT	0	Constant for arithmetic operation (0)

VALUE_U01_MASK1_1BIT	1	Constant for arithmetic operation (1)
VALUE_U08_MASK0_8BIT	00H	Constant for arithmetic operation (00H)
VALUE_U08_MASK1_8BIT	FFH	Constant for arithmetic operation (FFH)
COPY_FLAG_USUAL	AAAA5555H	Value set in the copy flag section

4.7 Enumeration Type

Table4-8 defines the enumeration-type variable used by the sample program.

Table4-8 enum e_ret (Enumeration Variable Name: e_ret_t)

Symbol Name	Value	Description
ENUM_RET_STS_OK	00H	Normal status
ENUM_RET_STS_RECEIVING	01H	Waiting for a command to be sent, or receiving a command
ENUM_RET_ERR_CFDF_SEQUENCER	02H	Code/data flash area sequencer error
ENUM_RET_ERR_EXTRA_SEQUENCER	03H	Extra area sequencer error
ENUM_RET_ERR_ERASE	04H	Erase error
ENUM_RET_ERR_WRITE	05H	Write error
ENUM_RET_ERR_BLANKCHECK	06H	Blank error
ENUM_RET_ERR_CHECK_WRITE_DATA	07H	Error in comparison between the written data against the read value
ENUM_RET_ERR_MODE_MISMATCHED	08H	Mode mismatch error
ENUM_RET_ERR_PARAMETER	09H	Parameter error
ENUM_RET_ERR_CONFIGURATION	0AH	Device configuration error
ENUM_RET_ERR_PACKET	0BH	Packet reception error

4.8 List of Variables

Table4-9 lists the global variables that are used in the sample program.

Table4-9 List of Global Variables

Type	Variable Name	Description	Function Supporting the Variable
uint8_t	f_UART0_sendend	Flag indicating that data sending by the UART0 was completed	r_Send_nByte r_Config_UART0_callback_sendend
uint32_t	g_copy_end	Flag indicating that data copy was ended normally	main
uint8_t	g_recv_data [261]	Receive data buffer	R_Config_UART0_Receive r_AsyncRecvPacketData
uint8_t	g_soft_recv_overrun	Flag indicating that data larger than the receive data buffer was received	r_Config_UART0_callback_softwareoverrun r_ClearUARTRecvBuff r_AsyncRecvPacketData

4.9 List of Functions

Table4-10 lists the functions that are used in the sample program.

Table4-10 List of Functions

Function Name	Summary
r_rfd_initialize	Initialization processing for RFD RL78 Type01
r_cmd_start	START command processing
r_cmd_end	END command processing
r_CF_RangeErase	Range erase processing for the code flash memory
r_CF_EraseBlock	Block erase processing for the code flash memory
r_CF_WriteVerifySequence	Write-and-verify processing for the code flash memory
r_CF_WriteData	Write processing for the code flash memory
r_CF_VerifyData	Verify processing for the code flash memory
r_CheckCFDFSequencerEnd	Sequence end processing for the code flash memory
r_CheckExtraSequencerEnd	Sequence end processing for the extra area
r_RequestBootSwap	Boot swapping execution processing
r_Config_UART0_callback_sendend	Callback processing at a sending completion interrupt for UART0
r_Send_nByte	Data sending processing by UART0
r_SendACK	Normal response sending processing by UART0
r_CF_TempCopy	Processing to copy data from the Temporary area
r_CF_MemoryWrite	Processing to reprogram the code flash memory
r_AsyncRecvPacketData	Processing to receive asynchronous command packets
r_GetUARTRecvSize	Processing to obtain the size of the receive data
r_ClearUARTRecvBuff	Processing to clear the receive buffer
userApplicationLoop	Function to implement user application
updateLoop	Processing to receive and run the firmware update command
errorLedOn	Processing to turn on the error LED

4.10 Specifications of Functions

This section describes the specifications of the functions used in the sample code.

r_rfd_initialize	
Summary	Initialization processing for RFD RL78 Type01
Header	r_rfd_common_api.h, r_rfd_code_flash_api.h, r_cg_userdefine.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_rfd_initialize(void);
Explanation	This function performs the processing to initialize RFD RL78 Type01.
Arguments	None
Return values	ENUM_RET_STS_OK: Normal end ENUM_RET_ERR_CONFIGURATION: Clock configuration error ENUM_RET_ERR_PARAMETER: Frequency setting error

r_cmd_start	
Summary	START command processing
Header	r_rfd_common_api.h, r_cg_userdefine.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_cmd_start(void);
Explanation	This function performs processing in response to reception of the START command.
Arguments	None
Return values	ENUM_RET_STS_OK: Normal end ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_ERASE: Erase error

r_cmd_end	
Summary	END command processing
Header	r_rfd_common_api.h, r_cg_userdefine.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_cmd_end(void);
Explanation	This function performs processing in response to reception of the END command.
Arguments	None
Return values	ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error

r_CF_RangeErase	
Summary	Range erase processing for the code flash memory
Header	r_rfd_common_api.h, r_rfd_code_flash_api.h, r_cg_userdefine.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_CF_RangeErase(uint32_t start_addr, uint32_t end_addr);
Explanation	This function erases data in the code flash memory. Data is erased in blocks. The blocks in the range of addresses specified for arguments will be erased.
Arguments	uint32_t start_addr: Erase start address uint32_t end_addr: Erase end address
Return values	ENUM_RET_STS_OK: Normal end ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_ERASE: Erase error

r_CF_EraseBlock

Summary	Block erase processing for the code flash memory
Header	r_rfd_common_api.h, r_rfd_code_flash_api.h, r_cg_userdefine.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_CF_EraseBlock(uint32_t start_addr);
Explanation	This function erases data in the code flash memory. A block of data is erased. The block that includes the address specified for an argument will be erased.
Arguments	uint32_t start_addr: Erase start address
Return values	ENUM_RET_STS_OK: Normal end ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_ERASE: Erase error

r_CF_WriteVerifySequence

Summary	Write-and-verify processing for the code flash memory
Header	r_rfd_common_api.h, r_rfd_code_flash_api.h, r_cg_userdefine.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_CF_WriteVerifySequence(uint32_t write_start_addr, uint16_t write_data_length, uint8_t __near *write_data);
Explanation	This function writes data to the code flash memory and verifies the written data.
Arguments	uint32_t start_addr,: Write start address uint16_t write_data_length: Size of the data to be written uint8_t __near *write_data: Data to be written
Return values	ENUM_RET_STS_OK: Normal end ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_ERASE: Erase error

r_CF_WriteData

Summary	Write processing for the code flash memory
Header	r_rfd_common_api.h, r_rfd_code_flash_api.h, r_cg_userdefine.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_CF_WriteData(uint32_t start_addr, uint16_t write_data_length, uint8_t __near *write_data);
Explanation	This function writes data to the code flash memory.
Arguments	uint32_t start_addr,: Write start address uint16_t write_data_length: Size of the data to be written uint8_t __near *write_data: Data to be written
Return values	ENUM_RET_STS_OK: Normal end ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_WRITE: Write error

r_CF_VerifyData	
Summary	Verify processing for the code flash memory
Header	r_cg_userdefine.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_CF_VerifyData(uint32_t start_addr, uint16_t data_length, uint8_t __near * write_data);
Explanation	This function verifies the data written to the code flash memory.
Arguments	uint32_t start_addr: Verify start address uint16_t data_length: Data size uint8_t __near * write_data: Data to be compared with
Return values	ENUM_RET_STS_OK: Normal end (matched) ENUM_RET_ERR_CHECK_WRITE_DATA: Error in comparison between the written data and the read value (mismatched)

r_CheckCFDFSequencerEnd	
Summary	Sequence end processing for the code flash memory
Header	r_rfd_common_api.h, r_cg_userdefine.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_CheckCFDFSequencerEnd(void);
Explanation	This function confirms that the code flash memory sequence has terminated.
Arguments	None
Return values	ENUM_RET_STS_OK: Normal end ENUM_RET_ERR_CFDF_SEQUENCER: Code/data flash memory sequencer error ENUM_RET_ERR_ERASE: Erase error ENUM_RET_ERR_WRITE: Write error ENUM_RET_ERR_BLANKCHECK: Blank error

r_CheckExtraSequencerEnd	
Summary	Sequence end processing for the extra area
Header	r_rfd_common_api.h, r_cg_userdefine.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_CheckExtraSequencerEnd (void);
Explanation	This function confirms that the extra area sequence has terminated.
Arguments	None
Return values	ENUM_RET_STS_OK: Normal end ENUM_RET_ERR_EXTRA_SEQUENCER: Extra area sequencer error ENUM_RET_ERR_ERASE: Erase error ENUM_RET_ERR_WRITE: Write error ENUM_RET_ERR_BLANKCHECK: Blank error

r_RequestBootSwap	
Summary	Boot swapping execution processing
Header	r_rfd_common_api.h, r_rfd_extra_area_api.h , r_cg_userdefine.h
Declaration	e_ret_t r_RequestBootSwap(void);
Explanation	After a reset is performed, this function enables the boot swapping settings, and then generates an internal reset to restart the CPU.
Arguments	None
Return values	ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error

r_Config_UART0_callback_sendend()	
Summary	Callback processing at a sending completion interrupt for UART0
Header	r_cg_macrodriver.h
Declaration	static void r_Config_UART0_callback_sendend(void);
Explanation	This is a callback function that is called at a sending completion interrupt for UART0.
Arguments	None
Return values	None

r_Send_nByte	
Summary	Data sending processing by UART0
Header	Config_UART0.h, Config_WDT.h
Declaration	MD_STATUS r_Send_nByte(uint8_t *tx_buff, const uint16_t tx_num);
Explanation	This function performs sending processing by UART0. This function waits until sending of the number of characters specified for an argument is completed.
Arguments	uint8_t *rx_buff: Pointer to the send data storage buffer const uint16_t rx_num: Number of characters to be sent
Return values	MD_OK: Normal end (sending completed) MD_ARGERROR: Parameter error

r_SendACK	
Summary	Normal response sending processing by UART0
Header	Config_UART0.h, Config_WDT.h
Declaration	MD_STATUS r_SendACK (void);
Explanation	This function uses UART0 to perform sending processing for normal response (01H).
Arguments	None
Return values	MD_OK: Normal end (sending completed) MD_ARGERROR: Parameter error

r_CF_TempCopy	
Summary	Processing to copy data from the Temporary area
Header	r_cg_userdefine.h, string.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_CF_TempCopy(void);
Explanation	This function copies data from the Temporary area.
Arguments	None
Return values	ENUM_RET_STS_OK: Normal end

r_CF_MemoryWrite	
Summary	Processing to reprogram the code flash memory
Header	r_cg_userdefine.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_CF_MemoryWrite(uint32_t* write_start_addr, uint32_t write_end_addr, uint8_t __near * write_data);
Explanation	This function writes data to memory.
Arguments	uint32_t* write_start_addr: Write start address uint32_t write_end_addr: Write end address uint8_t __near * write_data: Data to be written
Return values	ENUM_RET_STS_OK: Normal end ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_ERASE: Erase error

r_AsyncRecvPacketData	
Summary	Processing to receive asynchronous command packets
Header	r_cg_userdefine.h
Declaration	__far uint8_t r_AsyncRecvPacketData(uint8_t *p_cmd_type, uint8_t rdata[]);
Explanation	This function analyzes asynchronously received data and returns the status.
Arguments	uint8_t *p_cmd_type: Command information uint8_t rdata[]: Receive data buffer
Return values	ENUM_PACKET_STATUS_OK: Normal end ENUM_PACKET_STATUS_ERROR: Packet reception error ENUM_PACKET_STATUS_RECEIVING: Now receiving packets

r_GetUARTRecvSize	
Summary	Processing to obtain the size of the receive data
Header	r_cg_userdefine.h Config_UART0.h
Declaration	uint16_t r_GetUARTRecvSize(void);
Explanation	This function returns the length of the received data.
Arguments	None
Return values	Size: Length of the received data

r_ClearUARTRecvBuff	
Summary	Processing to clear the receive buffer
Header	r_cg_userdefine.h Config_UART0.h
Declaration	void r_ClearUARTRecvBuff(void);
Explanation	This function clears the buffer that stores received data.
Arguments	None
Return values	None

userApplicationLoop	
Summary	Function to implement user application
Header	r_cg_userdefine.h
Declaration	void userApplicationLoop(void);
Explanation	Sample application implemented that blinks LED1/LED8.
Arguments	None
Return values	None

updateLoop	
Summary	Processing to receive and run the firmware update command
Header	r_cg_userdefine.h
Declaration	e_ret_t updateLoop(void);
Explanation	This function receives and runs the firmware update command.
Arguments	None
Return values	ENUM_RET_STS_OK: Normal end ENUM_RET_ERR_CONFIGURATION: Clock configuration error ENUM_RET_ERR_PARAMETER: Frequency setting error ENUM_PACKET_STATUS_ERROR: Packet reception error ENUM_PACKET_STATUS_RECEIVING: Receiving packets ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_ERASE: Erase error

errorLedOn	
Summary	Processing to turn on the error LED
Header	r_cg_userdefine.h
Declaration	void errorLedOn(void);
Explanation	This function turns on LED7 and turns off the other LEDs if an error occurs.
Arguments	None
Return values	None

4.11 Flowcharts

4.11.1 Main Processing

Figure 4-1, Figure 4-2 shows the flowchart of the main processing.

Figure 4-1 Main Processing (1/2)

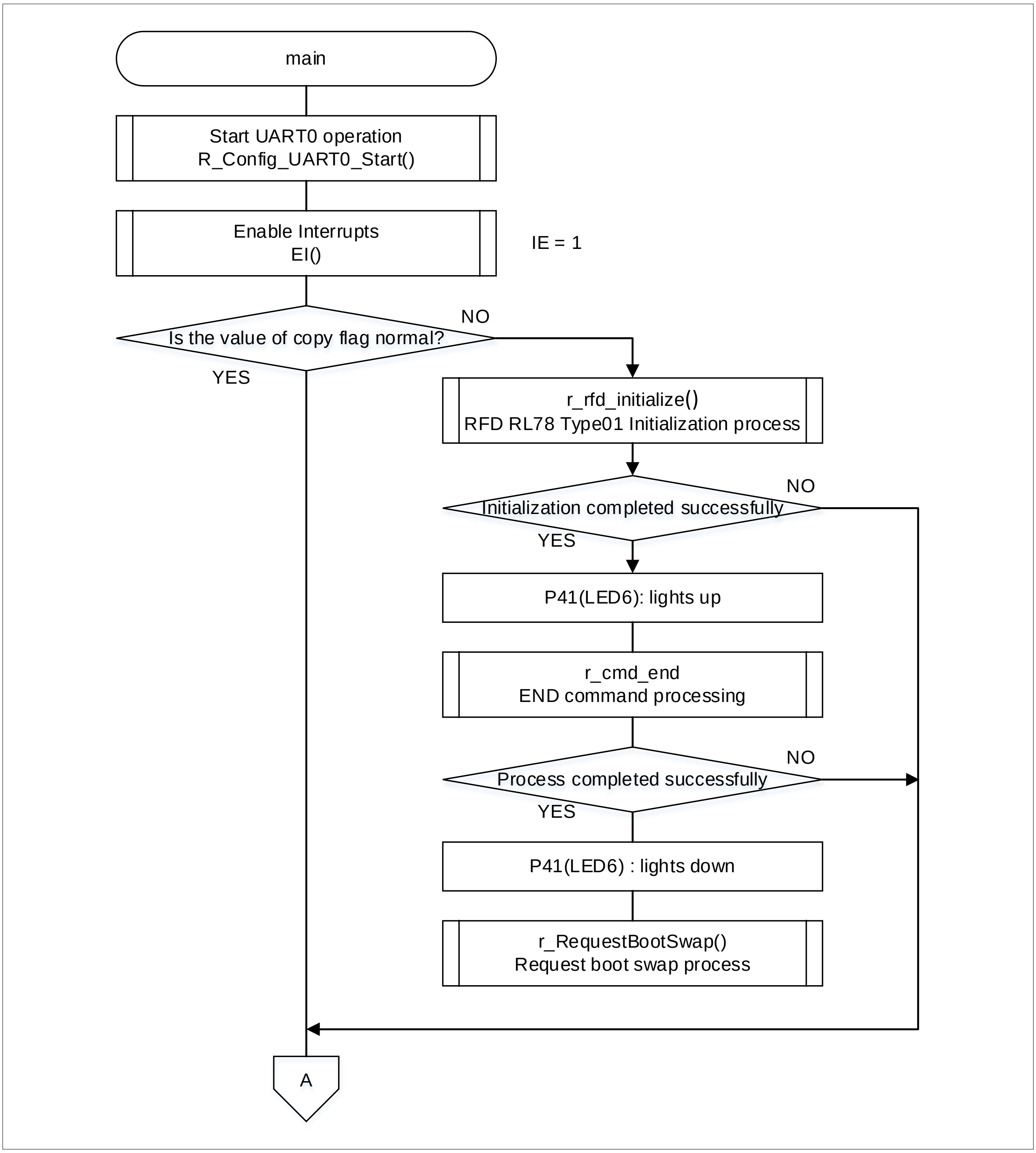
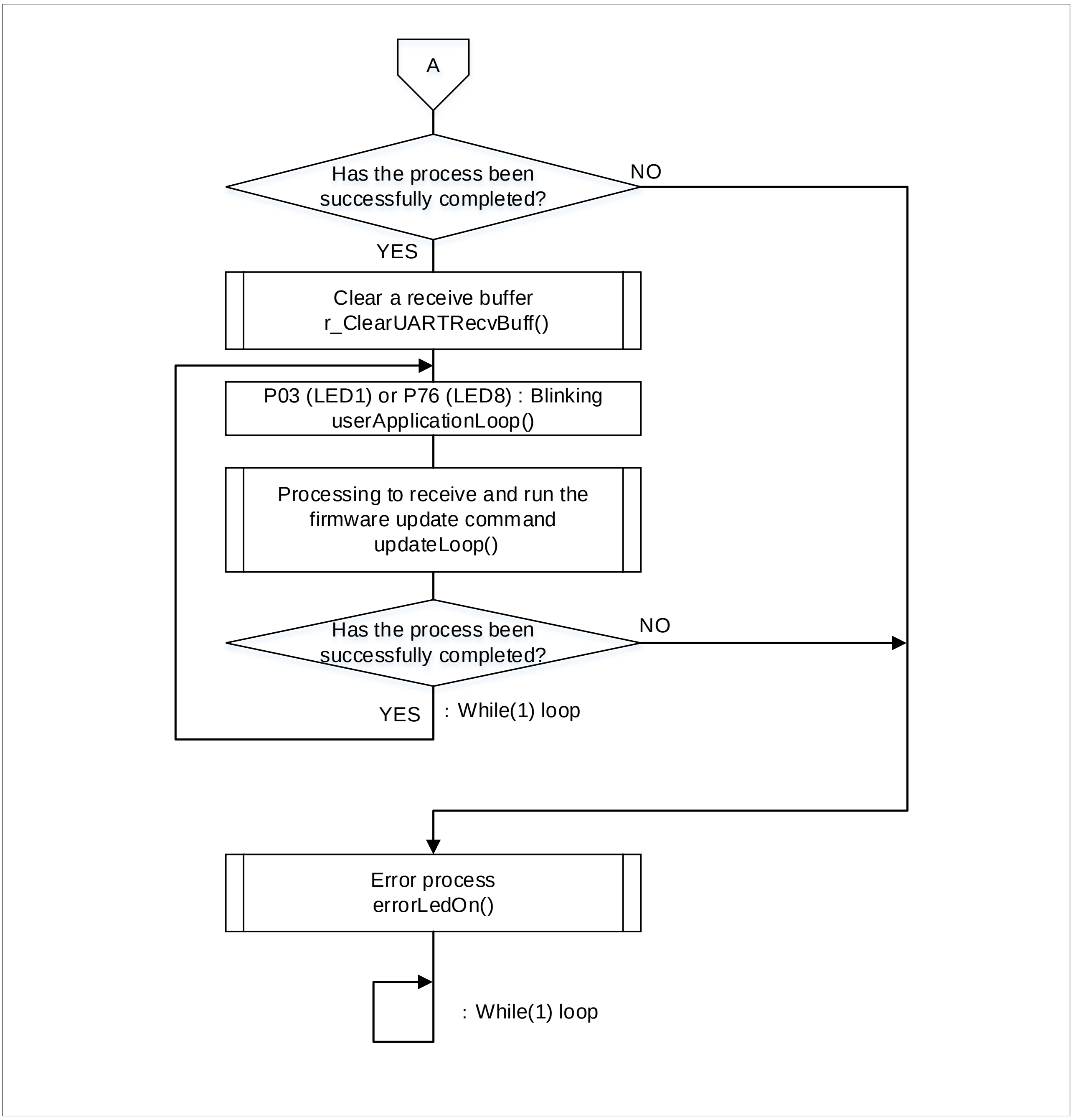


Figure 4-2 Main Processing (2/2)



4.11.2 Processing to receive and run the firmware update command

Figure 4-3, Figure 4-4, and Figure 4-5 shows the flowchart of processing to receive and run the firmware update command

Figure 4-3 Processing to receive and run the firmware update command (1/3)

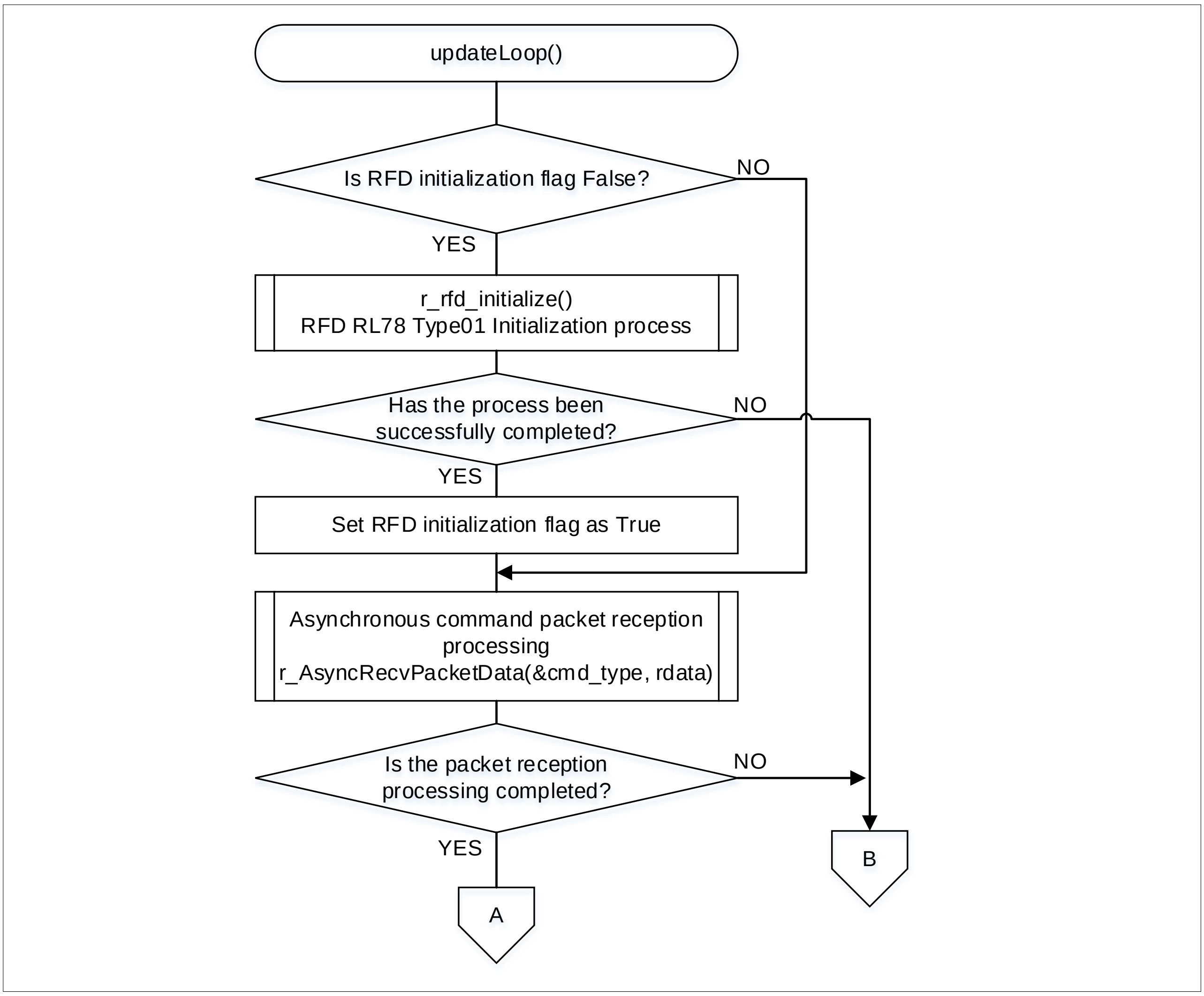


Figure 4-4 Processing to receive and run the firmware update command (2/3)

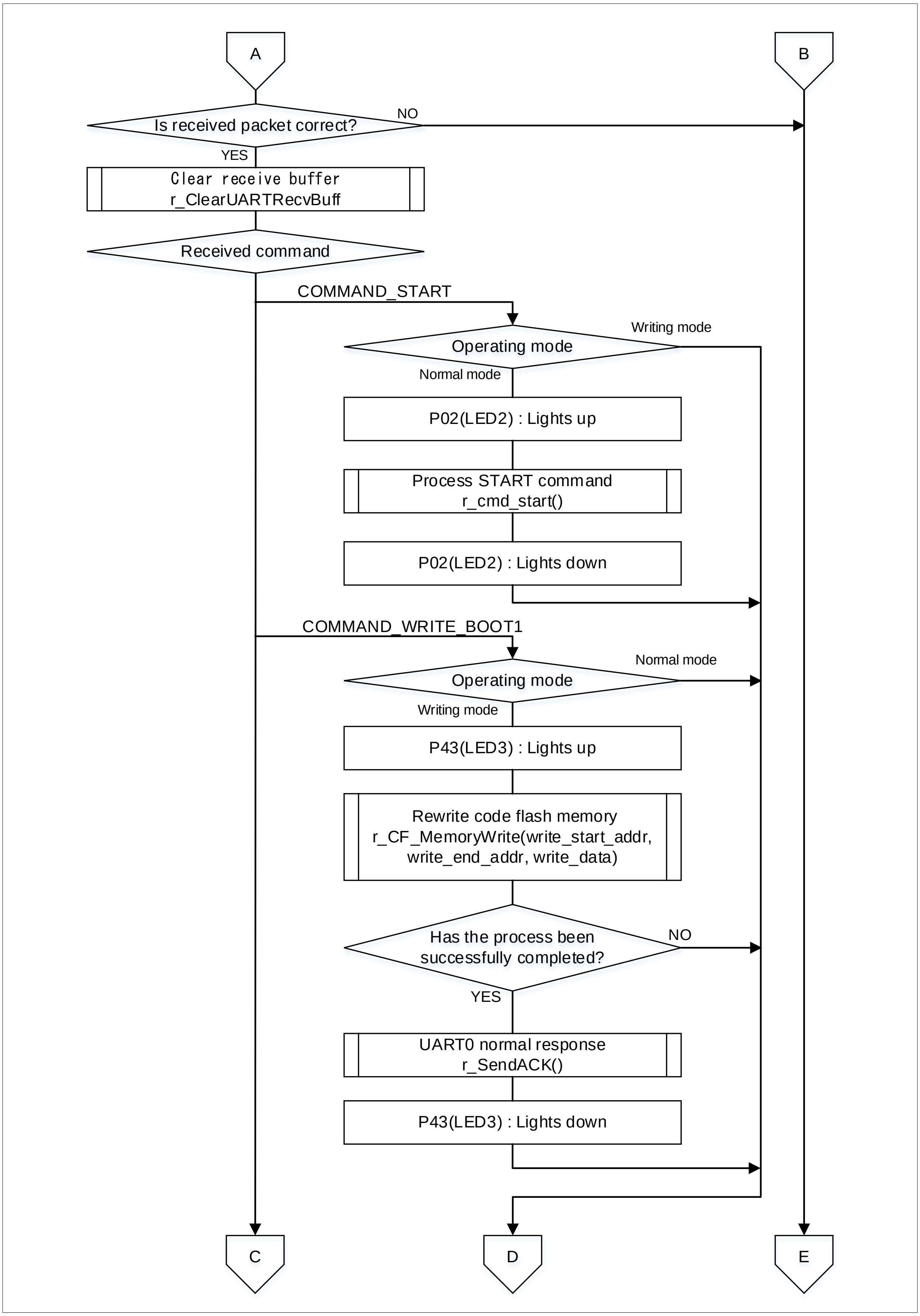
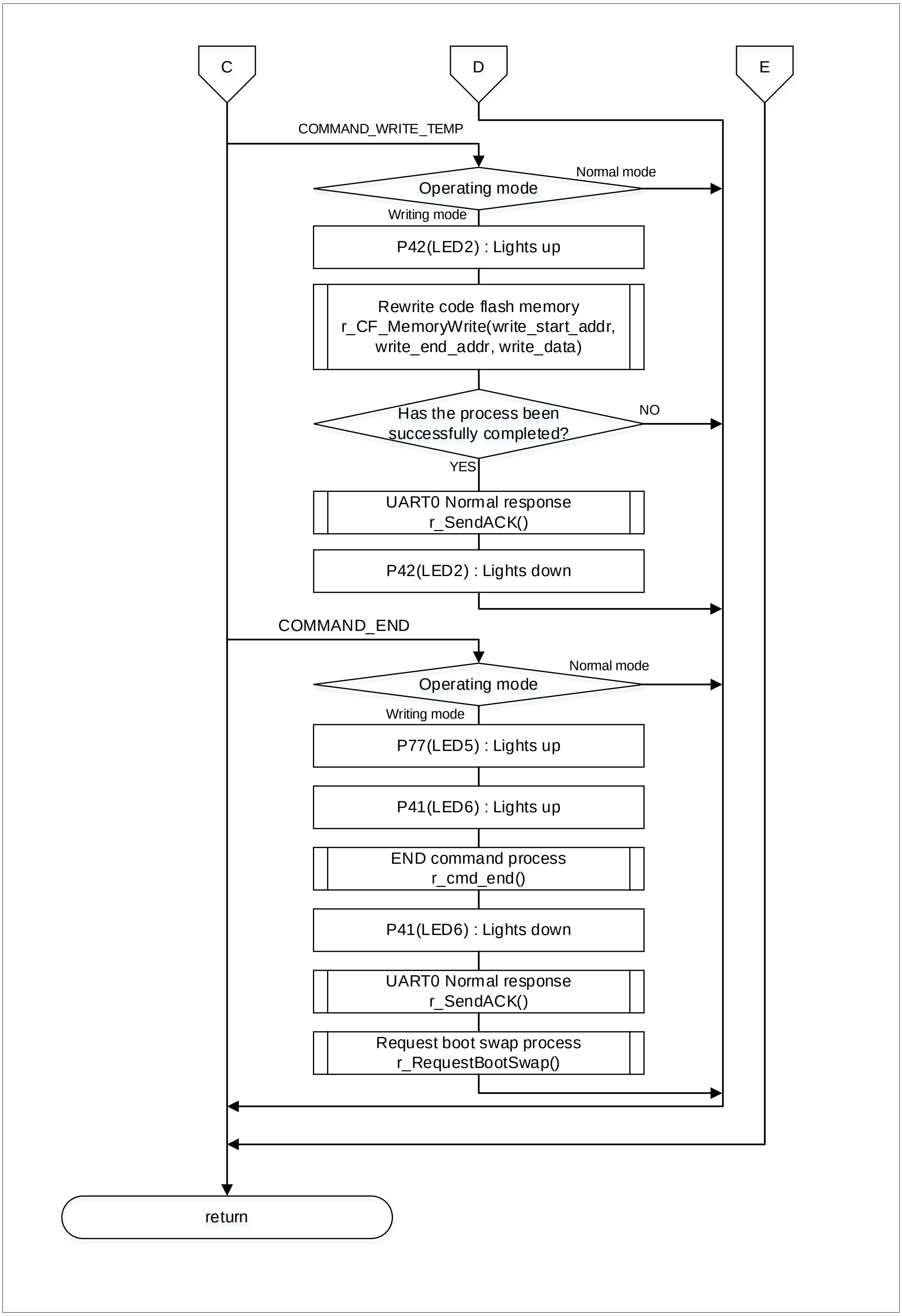


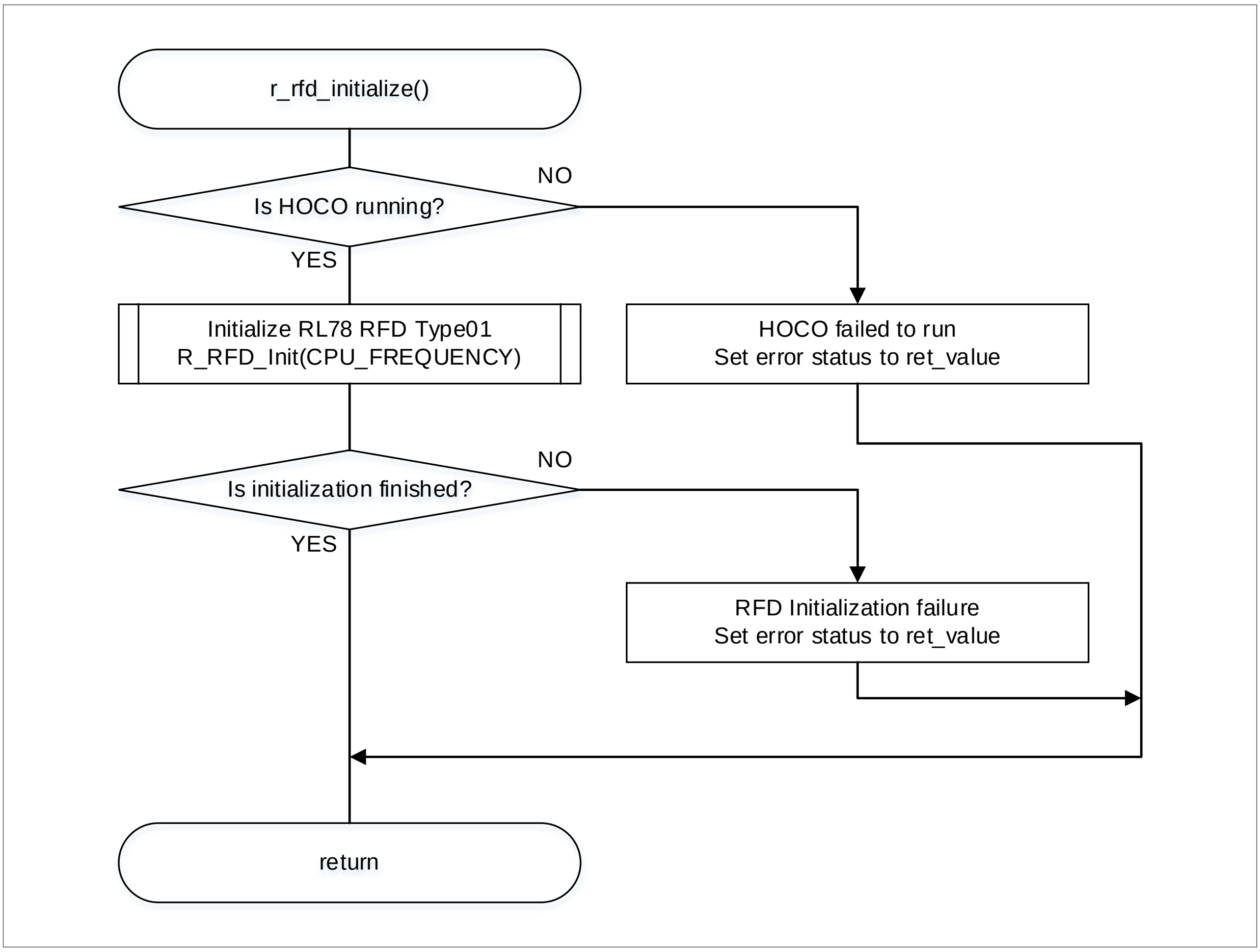
Figure 4-5 Processing to receive and run the firmware update command (3/3)



4.11.3 Initialization processing for RFD RL78 Type01

Figure 4-6 shows the flowchart of Initialization processing for RFD RL78 Type01.

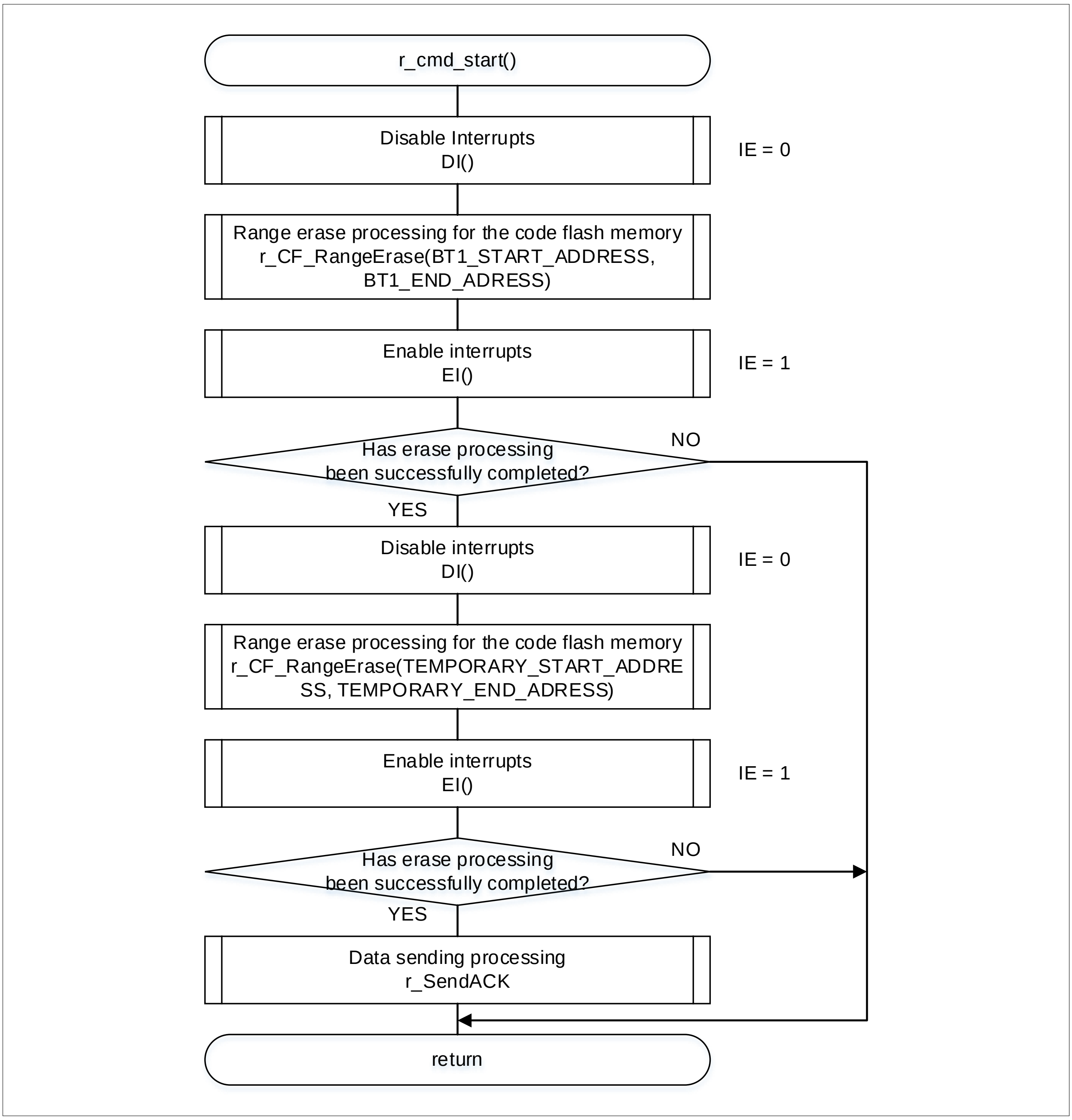
Figure 4-6 Initialization processing for RFD RL78 Type01



4.11.4 START command processing

Figure 4-7 shows the flowchart of START command processing

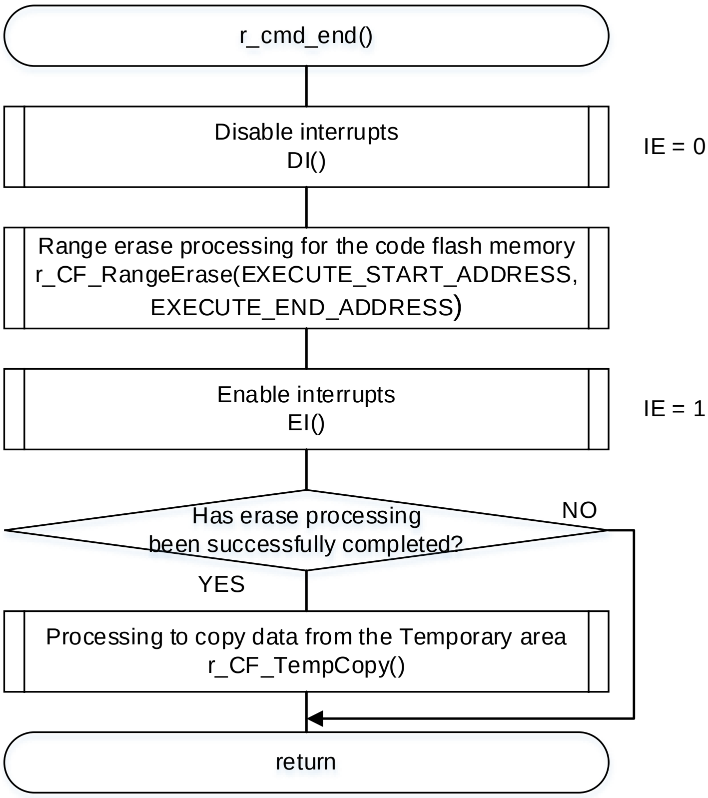
Figure 4-7 START command processing



4.11.5 END command processing

Figure 4-8 shows the flowchart of END command processing

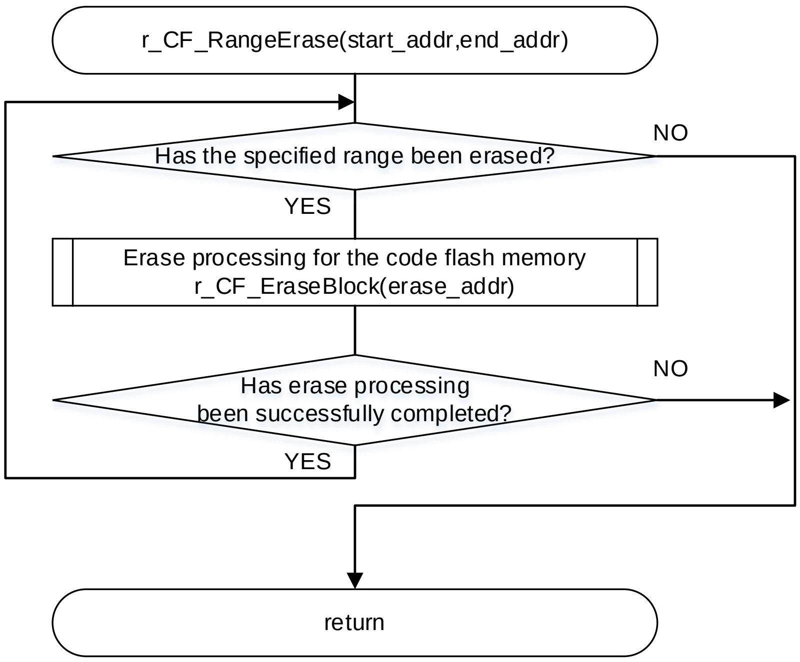
Figure 4-8 END command processing



4.11.6 Range erase processing for the code flash memory

Figure 4-9 shows the flowchart of range erase processing for the code flash memory

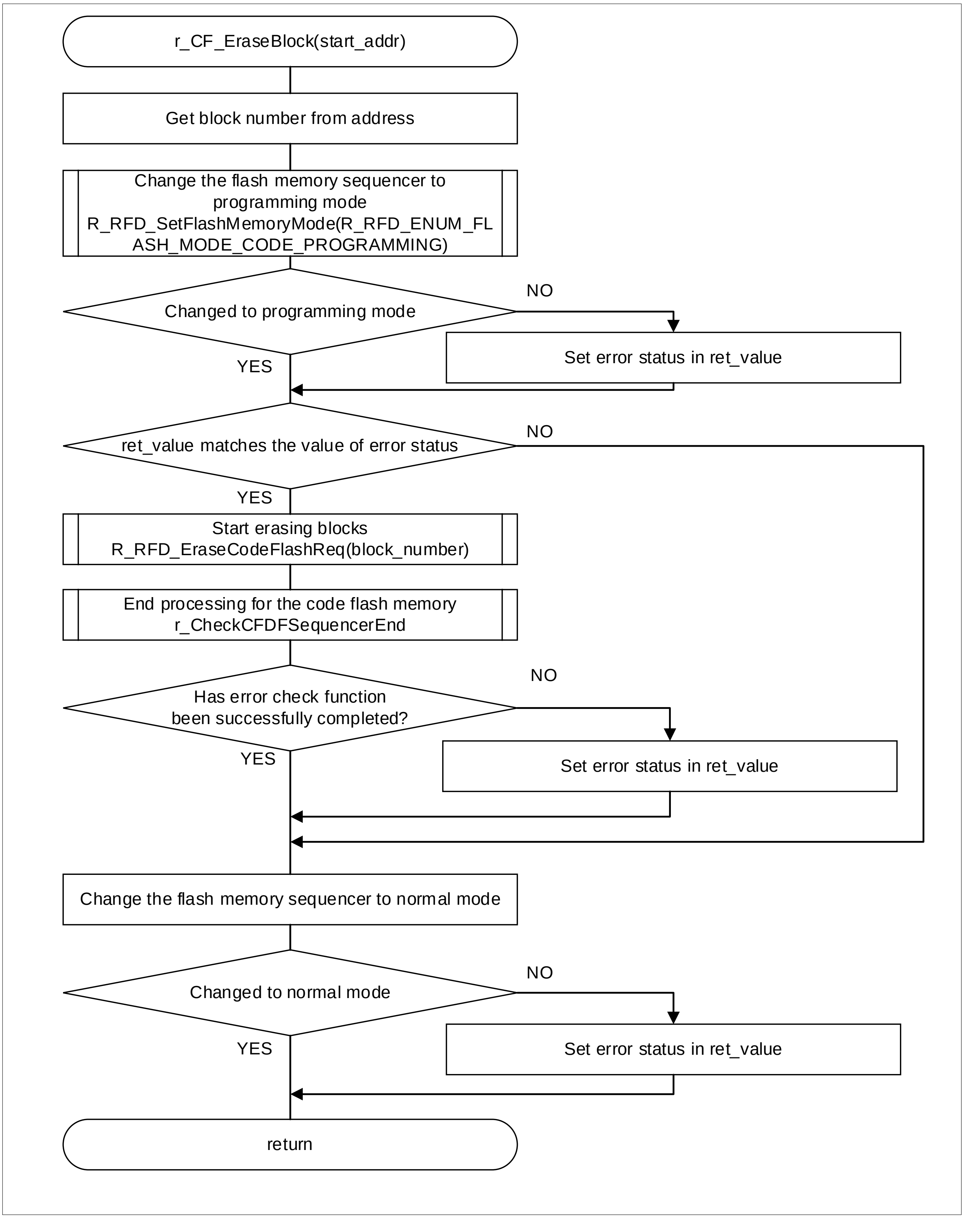
Figure 4-9 Range erase processing for the code flash memory



4.11.7 Block erase processing for the code flash memory

Figure 4-10 shows the flowchart of block erase processing for the code flash memory

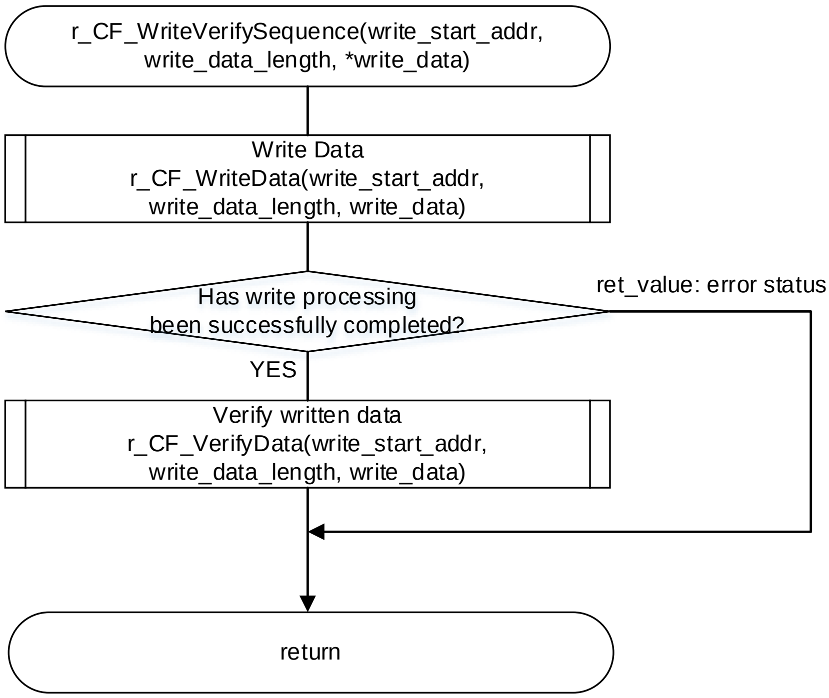
Figure 4-10 Block erase processing for the code flash memory



4.11.8 Write-and-verify processing for the code flash memory

Figure 4-11 shows the flowchart of write-and-verify processing for the code flash memory

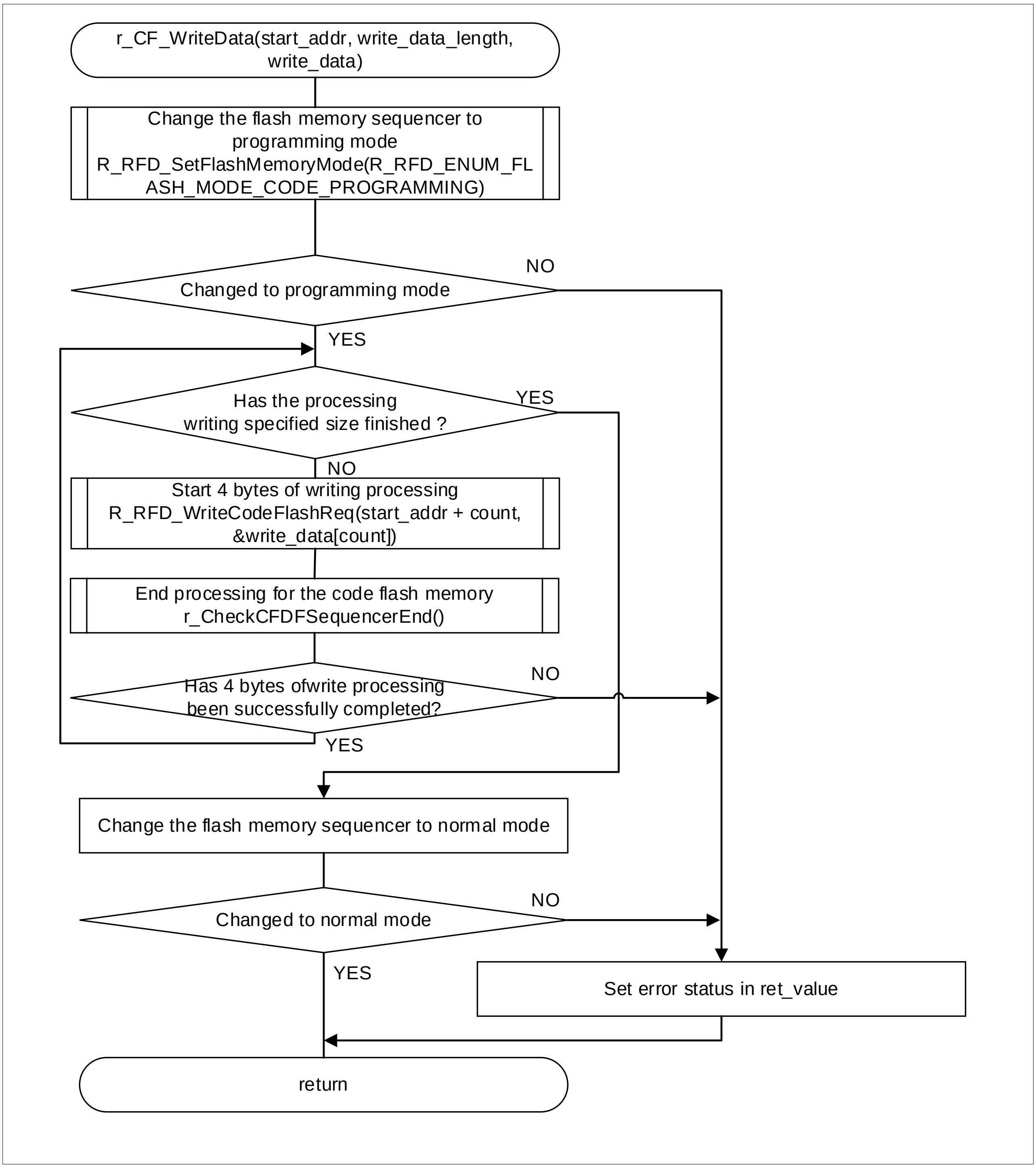
Figure 4-11 Write-and-verify processing for the code flash memory



4.11.9 Write processing for the code flash memory

Figure 4-12 shows the flowchart of write processing for the code flash memory

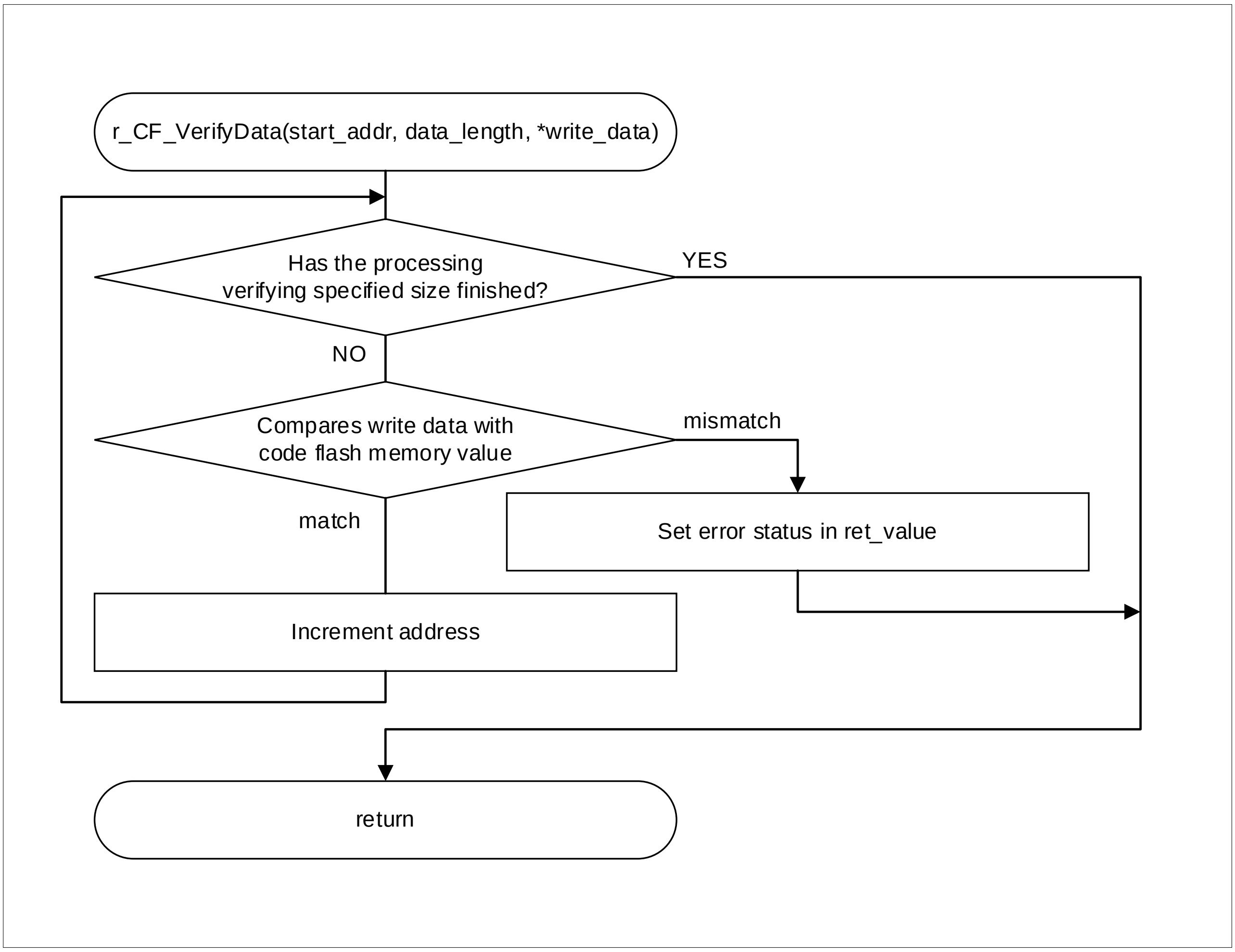
Figure 4-12 Write processing for the code flash memory



4.11.10 Verify processing for the code flash memory

Figure 4-13 shows the flowchart of verify processing for the code flash memory

Figure 4-13 Verify processing for the code flash memory



4.11.11 Sequence end processing for the code flash memory

Figure 4-14 and Figure 4-15 shows the flowchart of sequence end processing for the code flash memory

Figure 4-14 Sequence end processing for the code flash memory (1/2)

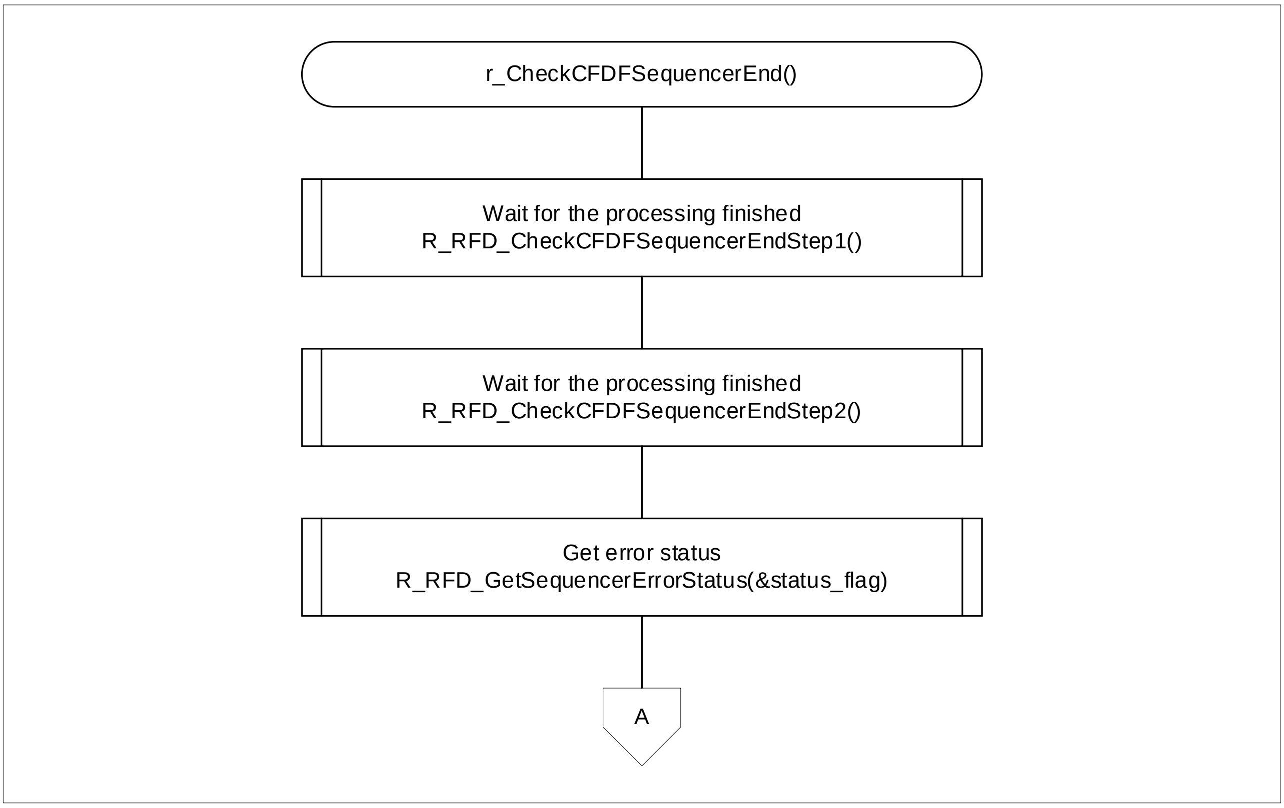
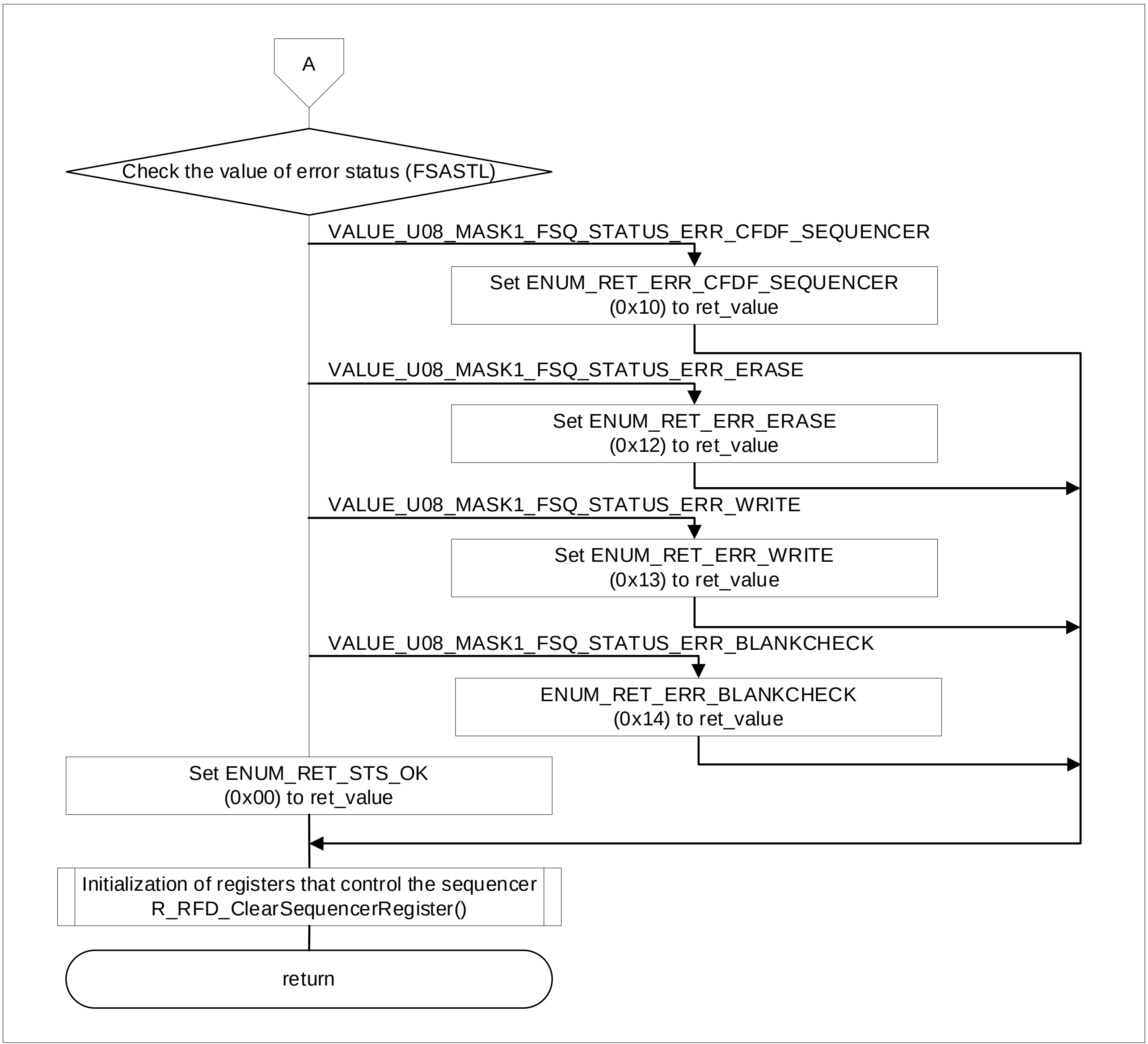


Figure 4-15 Sequence end processing for the code flash memory (2/2)



4.11.12 Sequence end processing for the extra area

Figure 4-16 and Figure 4-17 shows the flowchart of sequence end processing for the extra area

Figure 4-16 Sequence end processing for the extra area (1/2)

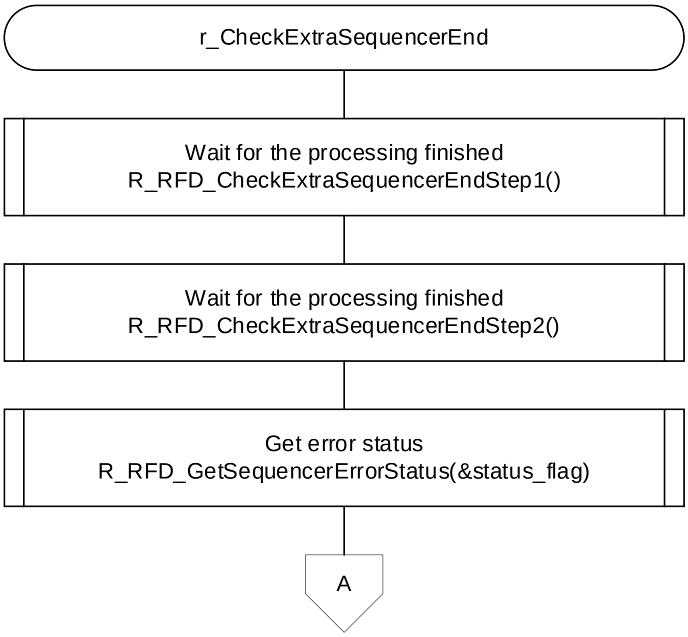
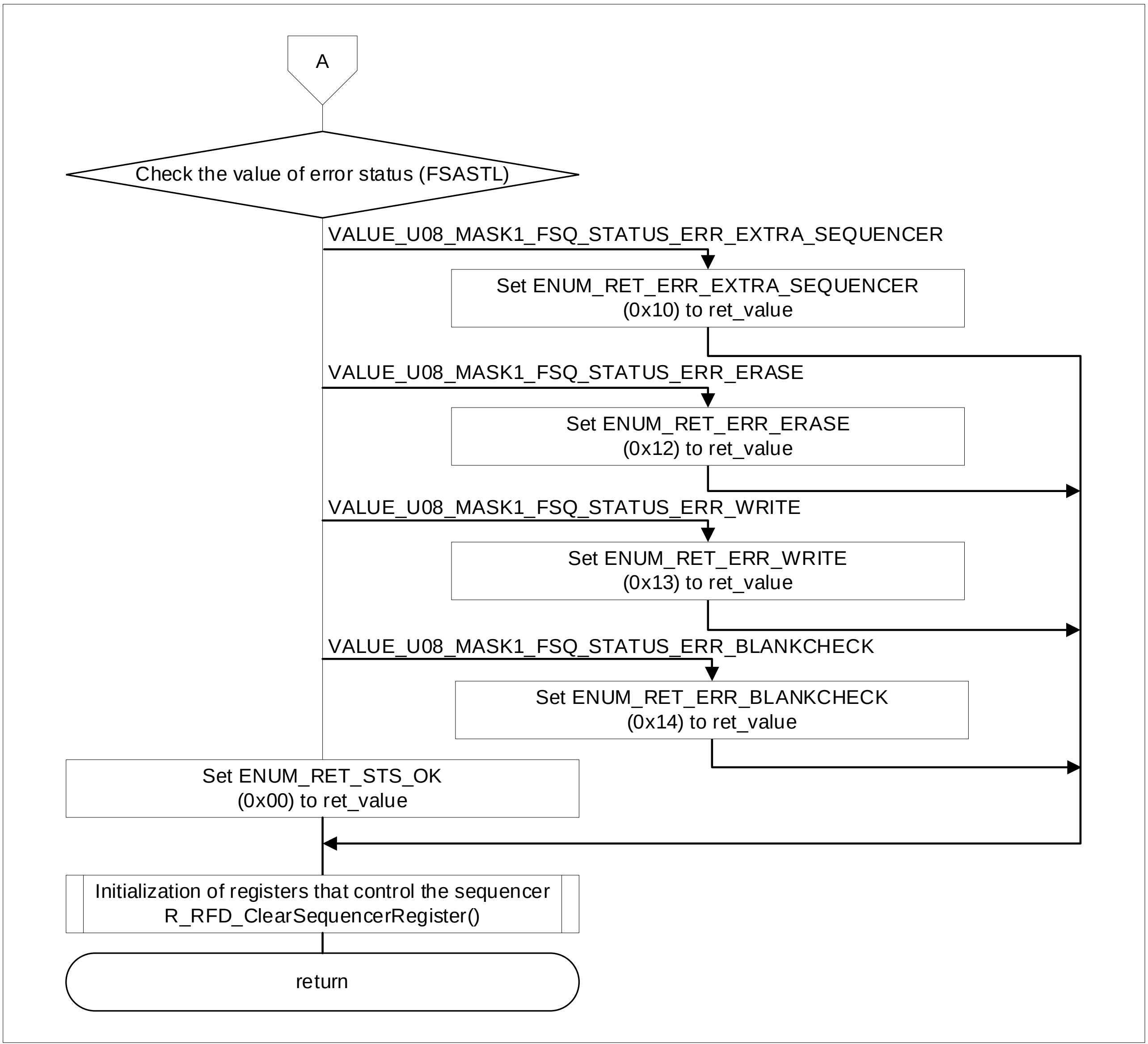


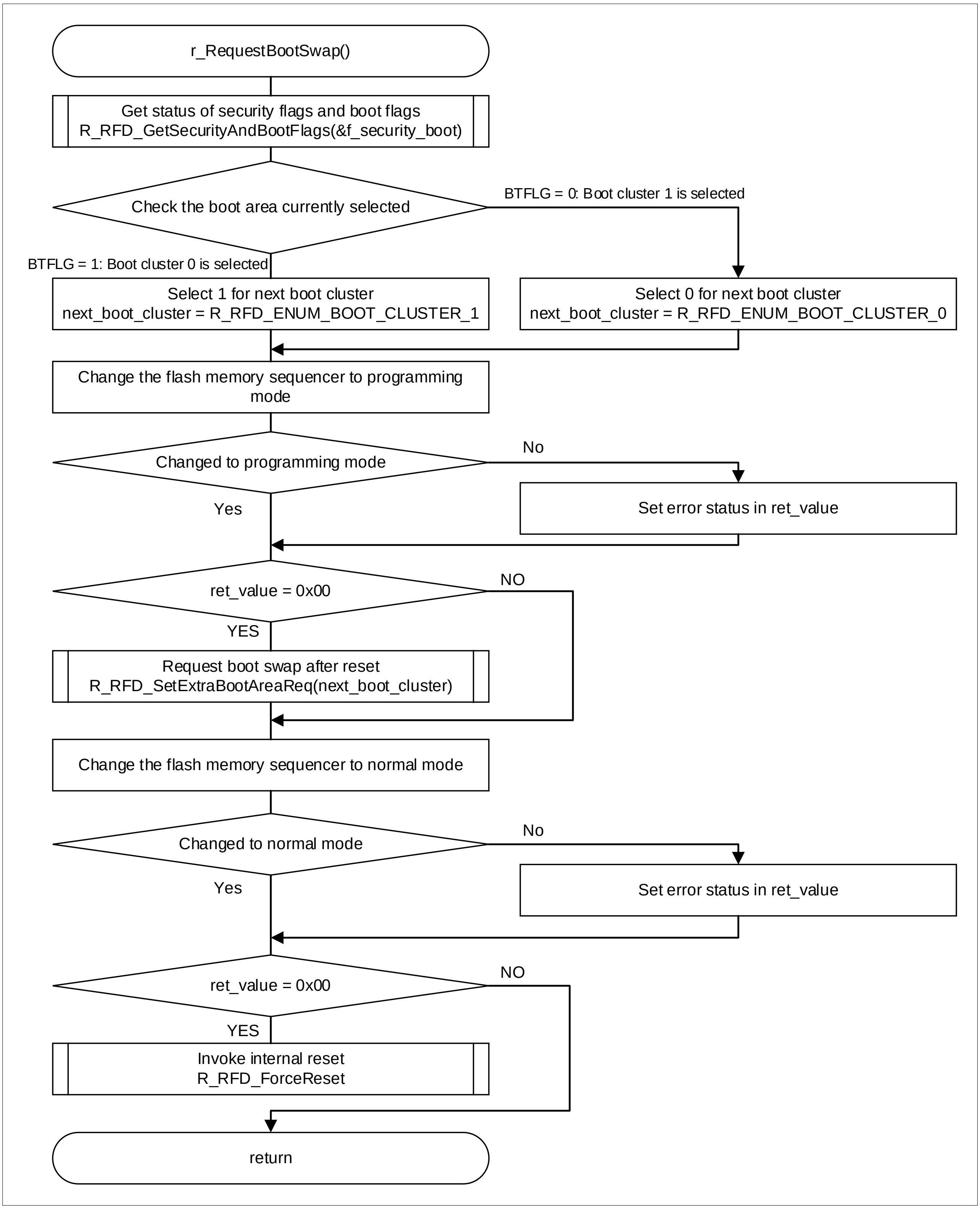
Figure 4-17 Sequence end processing for the extra area (2/2)



4.11.13 Boot swapping execution processing

Figure 4-18 shows the flowchart of boot swapping execution processing

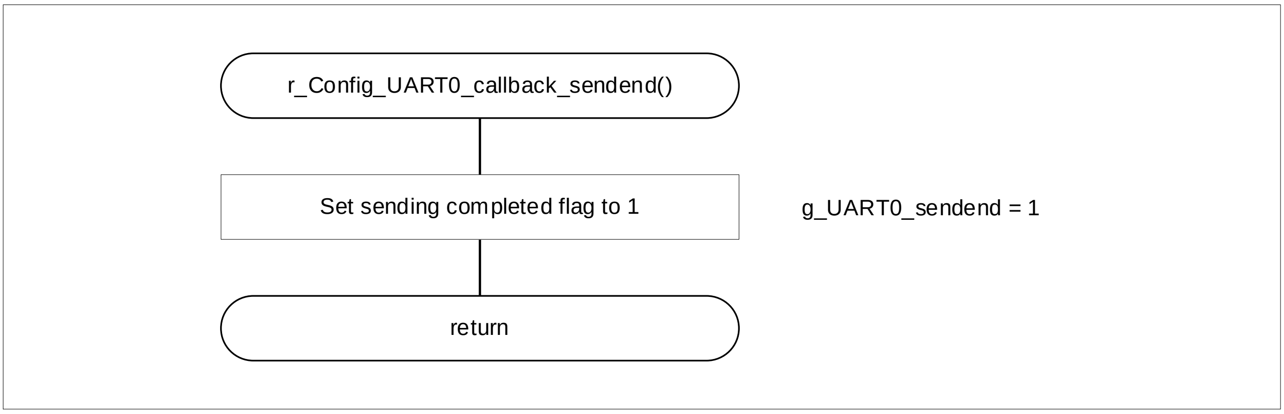
Figure 4-18 Boot swapping execution processing



4.11.14 Callback processing at a sending completion interrupt for UART0

Figure 4-19 shows the flowchart of callback processing at a sending completion interrupt for UART0

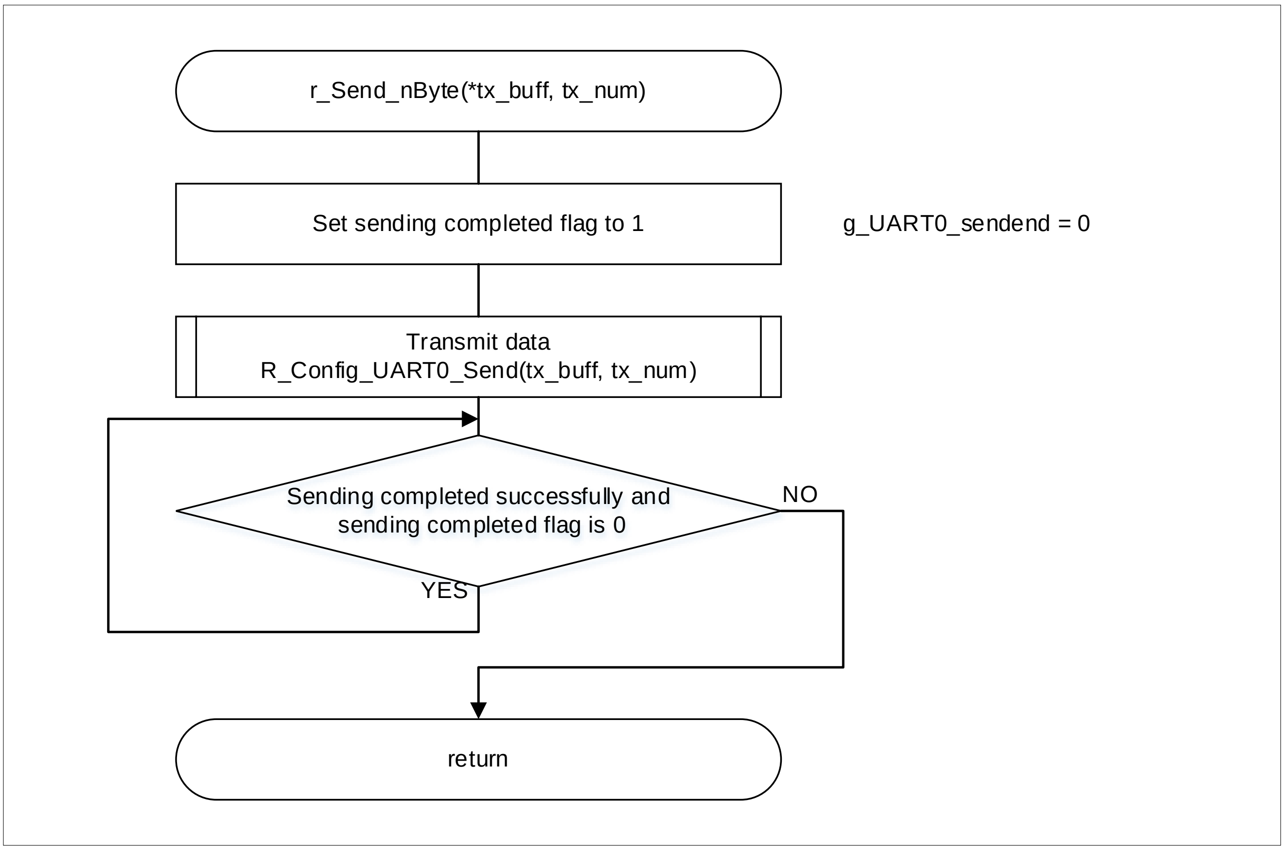
Figure 4-19 Callback processing at a sending completion interrupt for UART0



4.11.15 Data sending processing by UART0

Figure 4-20 shows the flowchart of Data sending processing by UART0

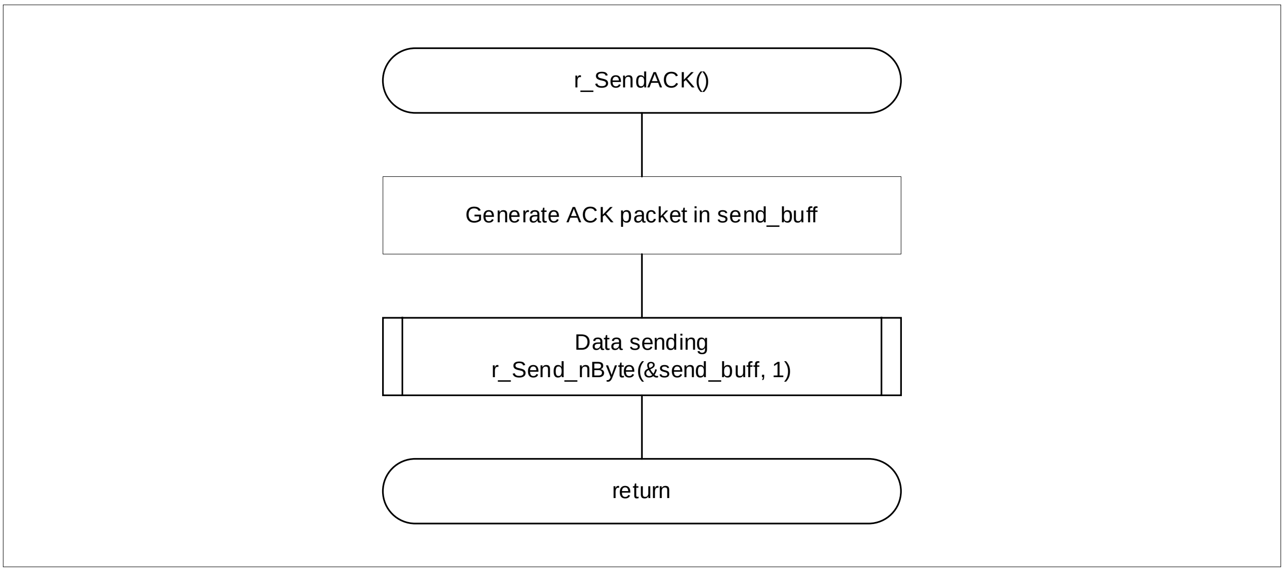
Figure 4-20 Data sending processing by UART0



4.11.16 Normal response sending processing by UART0

Figure 4-21 shows the flowchart of normal response sending processing by UART0

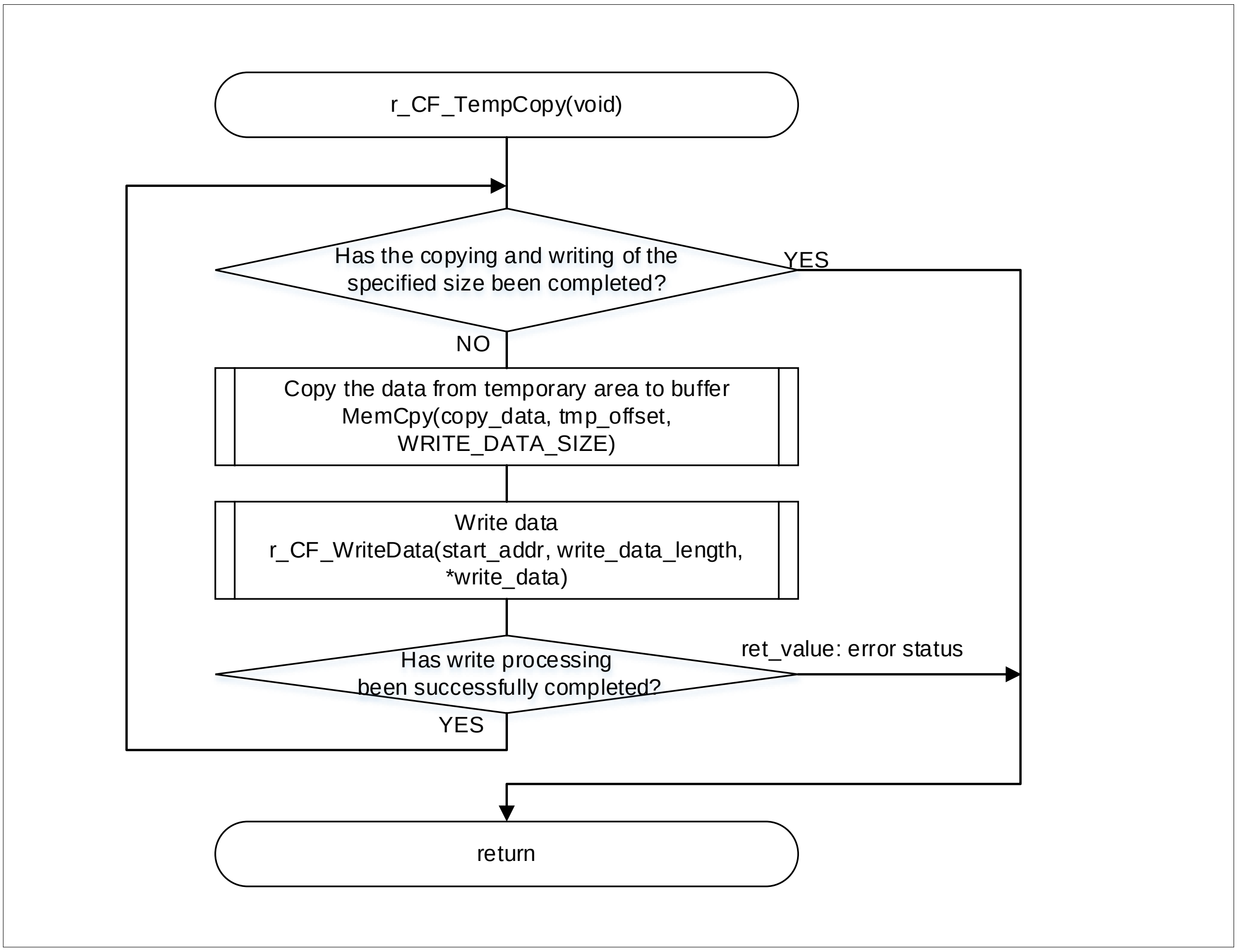
Figure 4-21 Normal response sending processing by UART0



4.11.17 Processing to copy data from the Temporary area

Figure 4-22 shows the flowchart of processing to copy data from the Temporary area

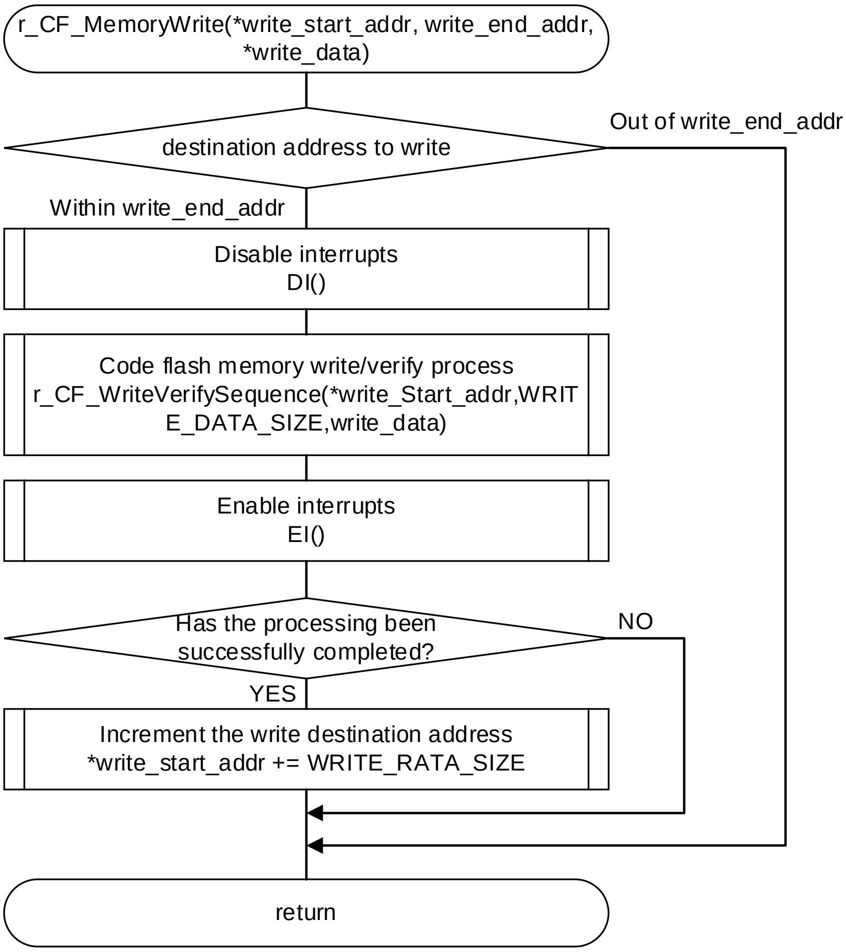
Figure 4-22 Processing to copy data from the Temporary area



4.11.18 Processing to reprogram the code flash memory

Figure 4-23 shows the flowchart of processing to reprogram the code flash memory

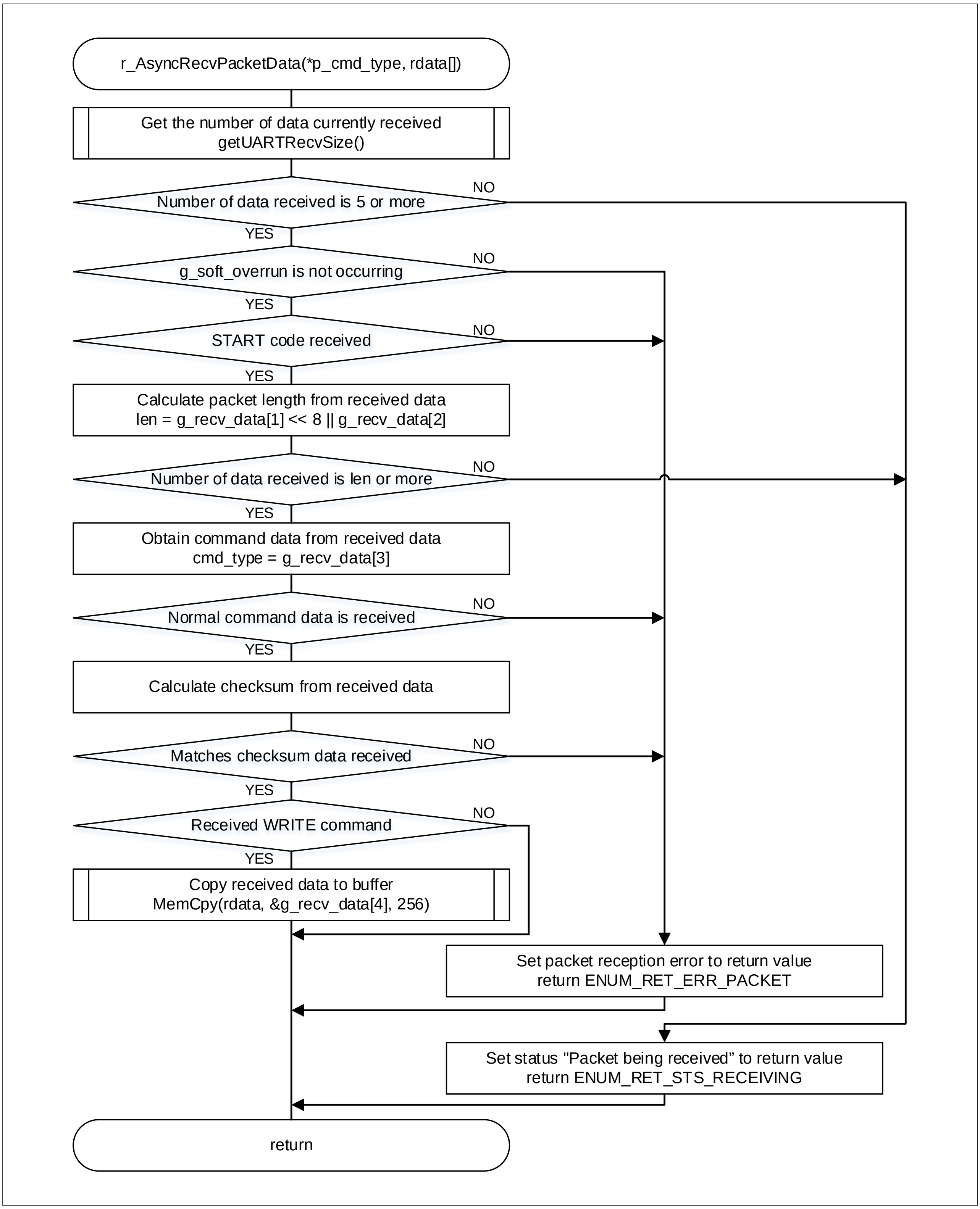
Figure 4-23 Processing to reprogram the code flash memory



4.11.19 Processing to receive asynchronous command packets

Figure 4-24 shows the flowchart of processing to receive asynchronous command packets

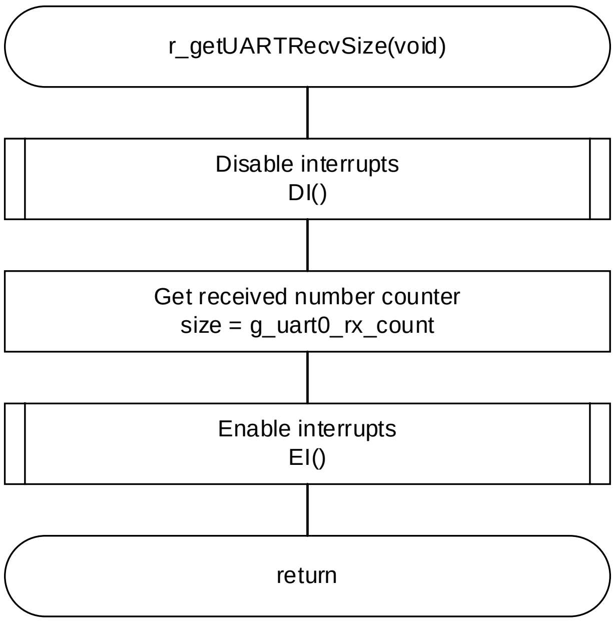
Figure 4-24 Processing to receive asynchronous command packets



4.11.20 Processing to obtain the size of the receive data

Figure 4-25 shows the flowchart of processing to obtain the size of the receive data

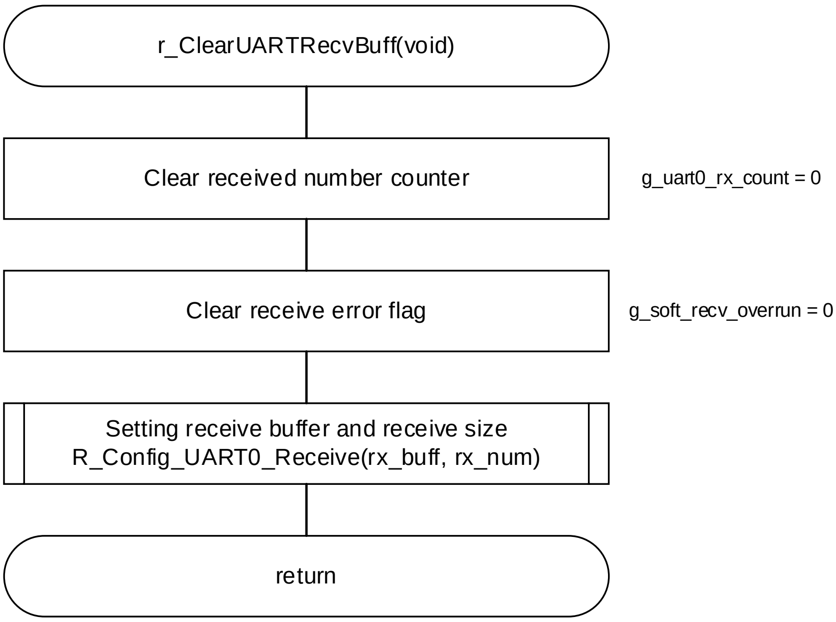
Figure 4-25 Processing to obtain the size of the receive data



4.11.21 Processing to clear the receive buffer

Figure 4-26 shows the flowchart of processing to clear the receive buffer

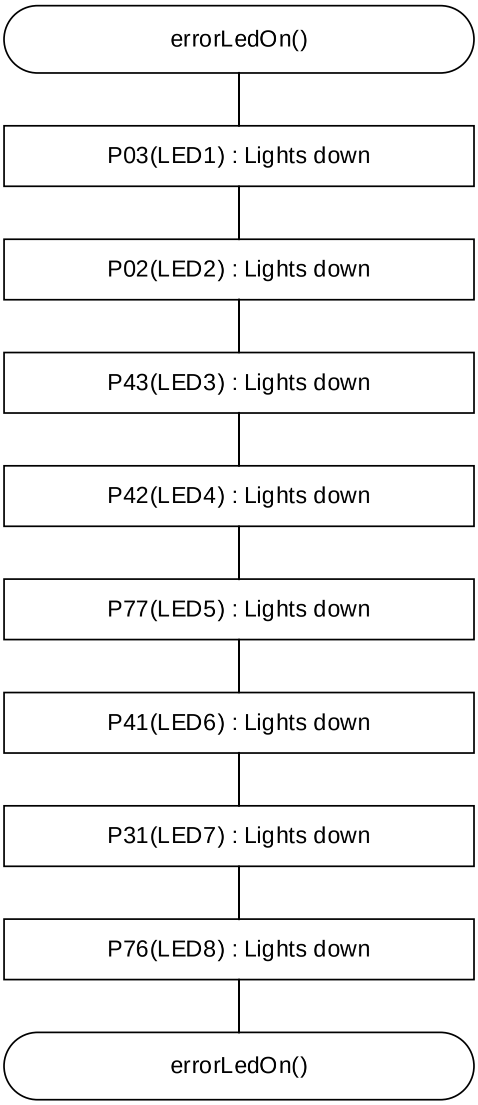
Figure 4-26 Processing to clear the receive buffer



4.11.22 Processing to turn on the error LED

Figure 4-27 shows the flowchart of processing to turn on the error LED

Figure 4-27 Processing to turn on the error LED



5. GUI-Based Tool for Writing Data

This chapter describes the GUI-based tool for writing data to the target device simply by running an executable file (.exe). Select the binary file (.bin) that contains the data to be written. To perform a write again, restart the tool.

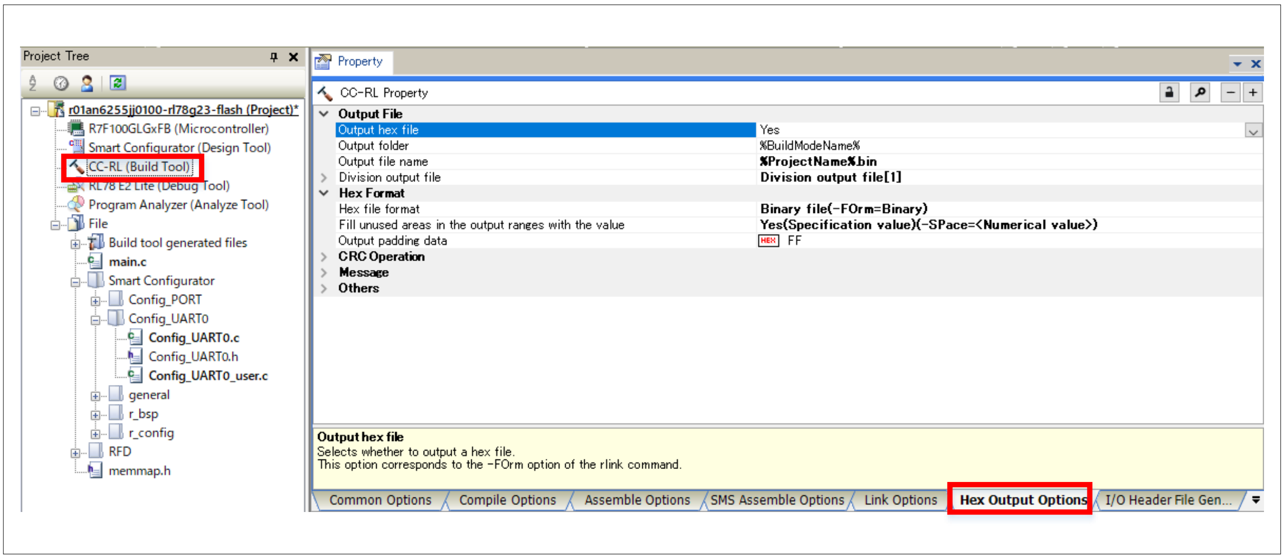
5.1 Generating a File Required to Write Data

Before you can use the GUI-based tool, generate a binary file (.bin) that will be written. For details about how to generate a binary file, see the following sections.

5.1.1 Using CS+ to Generate a Binary File

In the [Project Tree], select [CC-RL (Build Tool)], and then open the [Hex Output Options] tab.

Figure 5-1 Generate a binary file in CS+ (1/6)



In the [Hex Output Options] tab, under [Output File], set [Yes] for [Output hex file].

Select [Division output file], and then, in the dialog box that appears, enter a character string in the following pattern:

`XXXXXXXXXX.bin=0-YYYYYYYYY`

For *XXXXXXXXXX*, specify the project name. For *YYYYYYYYY*, specify the last address of the code flash memory of the device to be used.

Figure 5-2 Generate a binary file in CS+ (2/6)

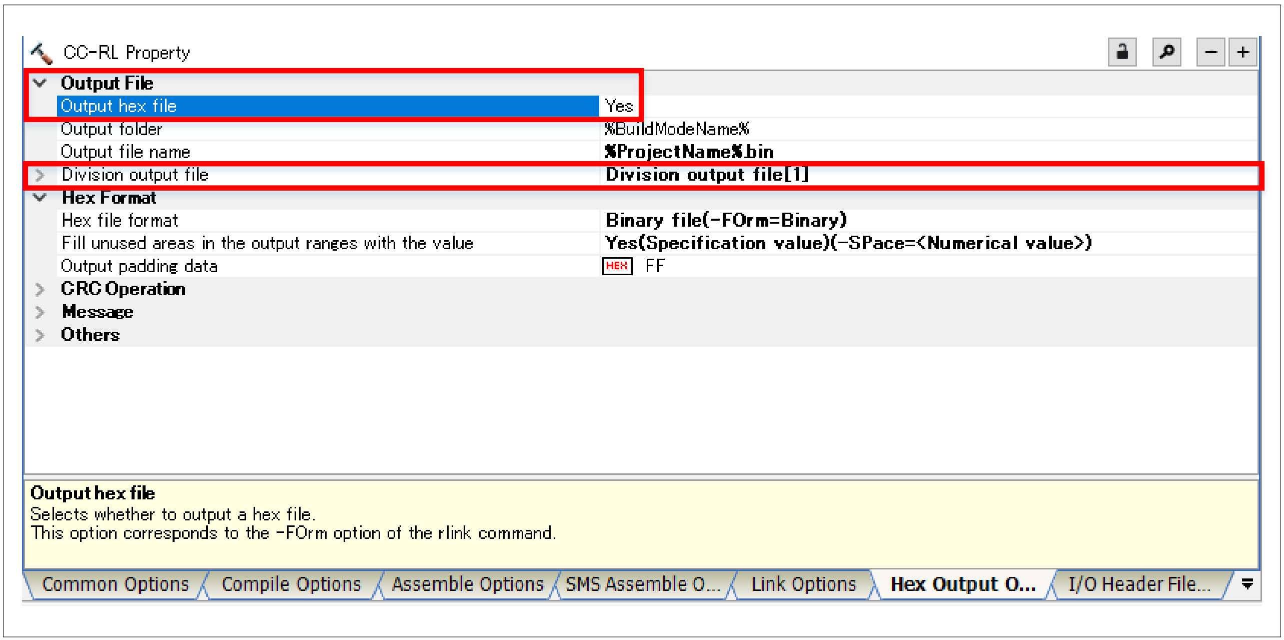
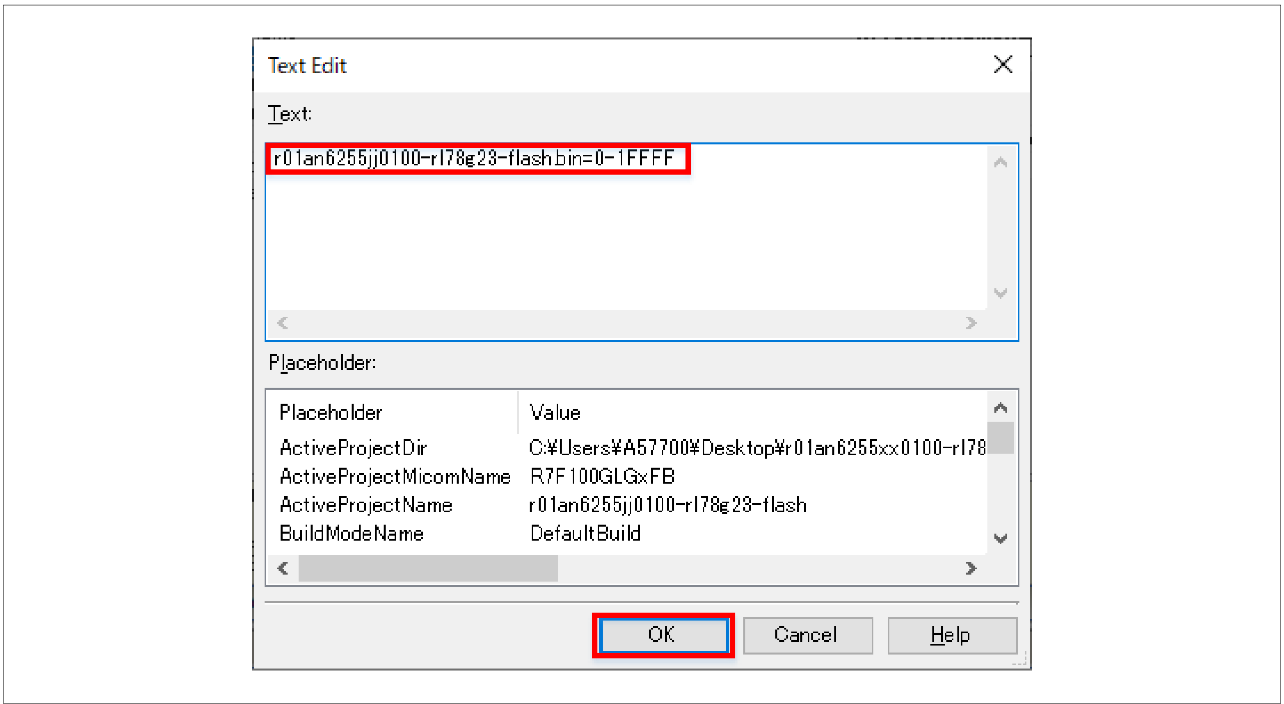
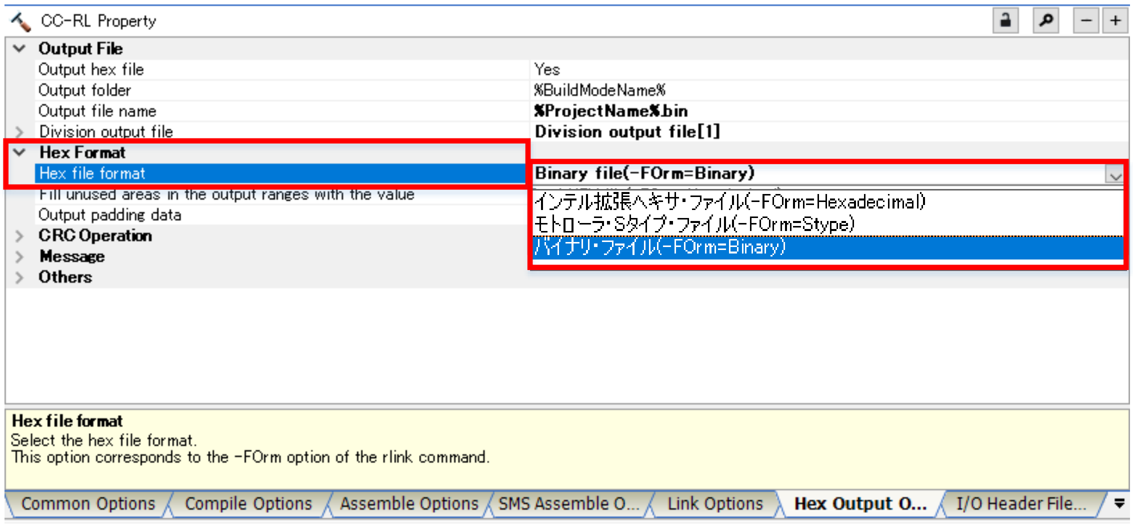


Figure 5-3 Generate a binary file in CS+ (3/6)



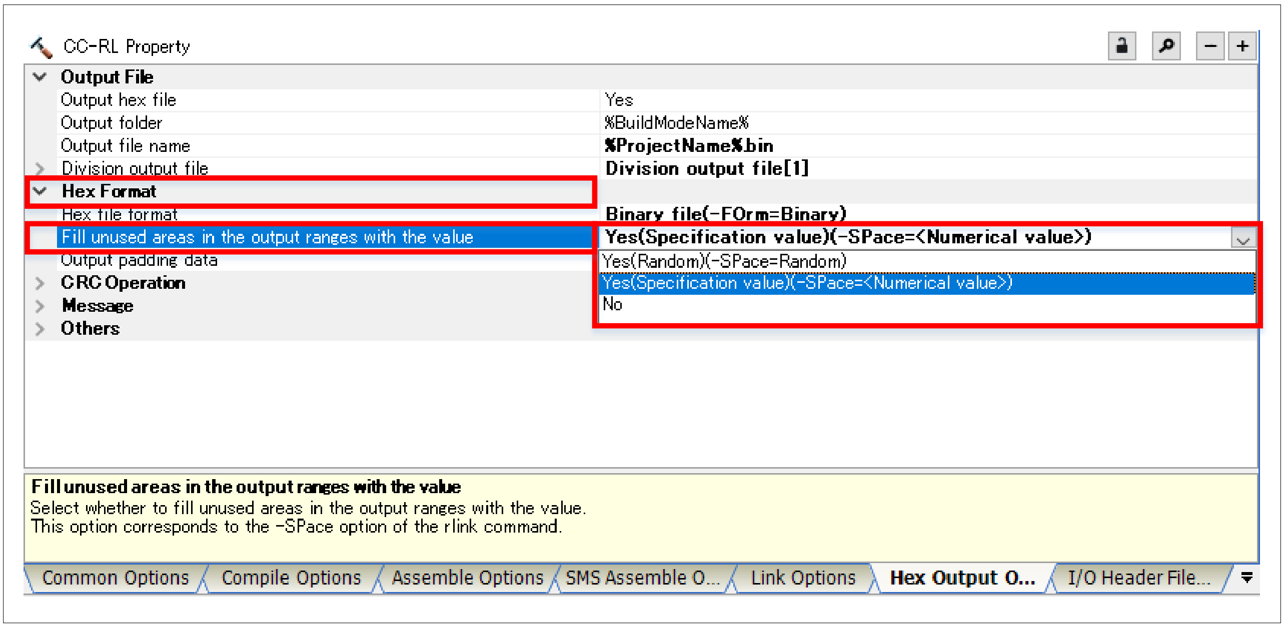
In the [Hex Output Options: tab, under [Hex Format], set [Hex file format] to [Binary file (-FOrm=Binary)].

Figure 5-4 Generate a binary file in CS+ (4/6)



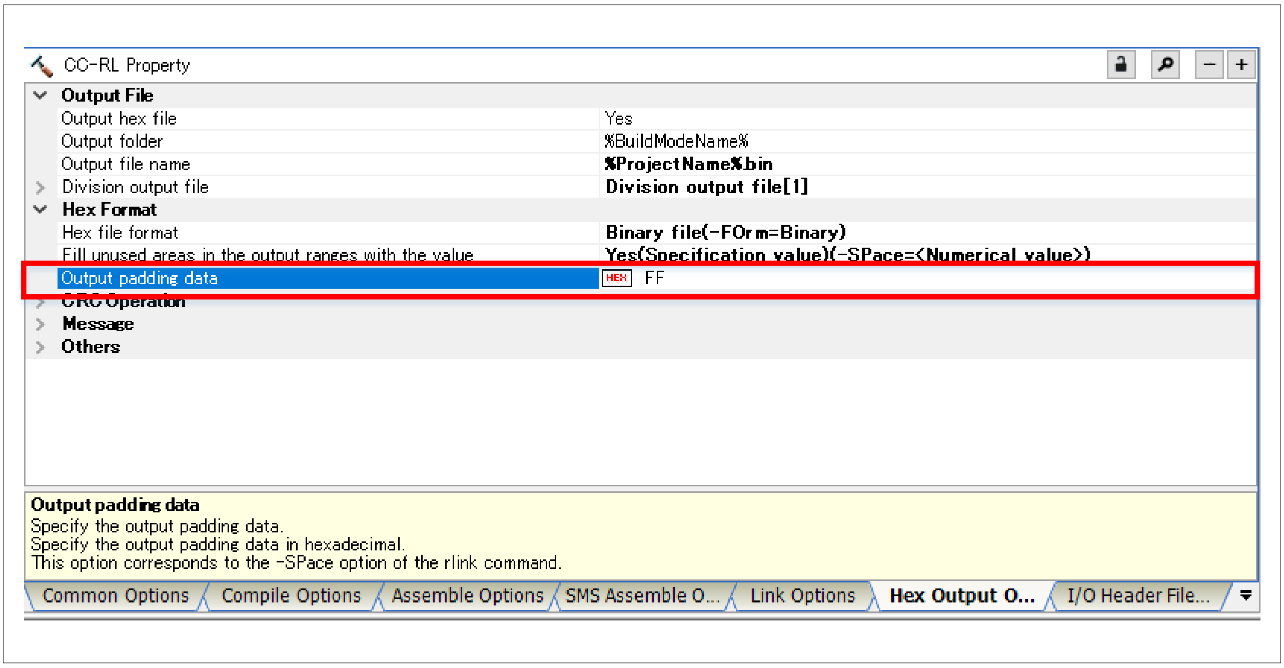
In the [Hex Output Options] tab, under [Hex Format], set [Fill unused areas in the output ranges with the value] to [Yes (Specification value) (-SPace=<Numerical value>)].

Figure 5-5 Generate a binary file in CS+ (5/6)



In the [Hex Output Options] tab, under [Hex Format], set [Output padding data] to [FF].

Figure 5-6 Generate a binary file in CS+ (6/6)



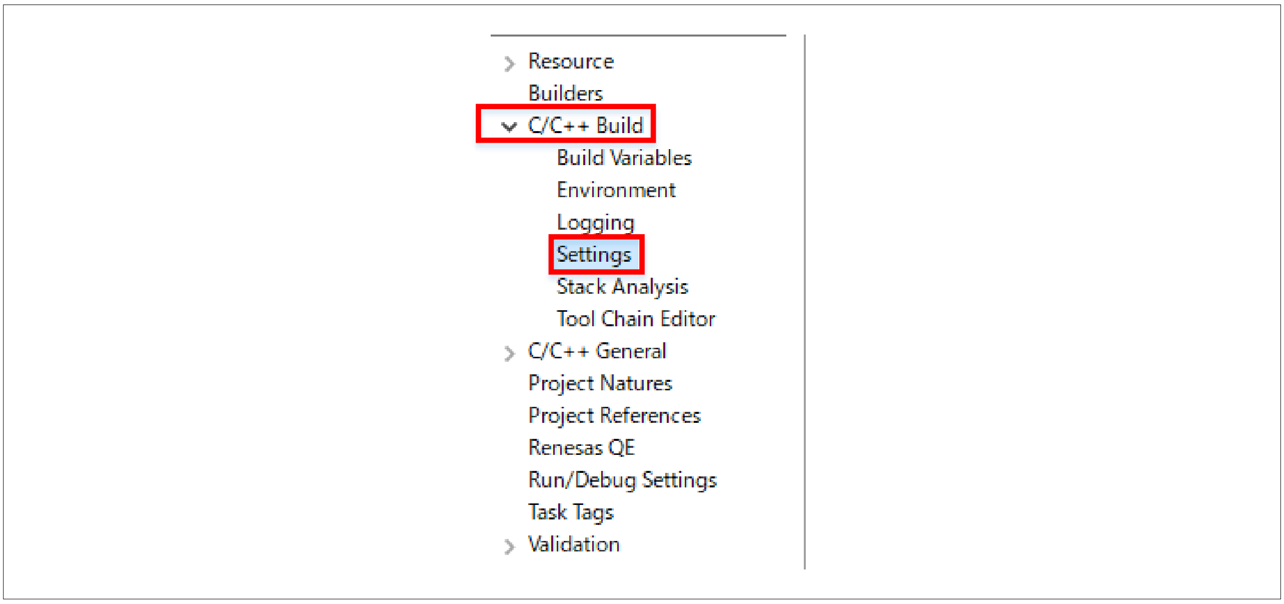
A binary file is generated when you build a project.

5.1.2 Using e2studio to Generate a Binary File

In the [Project] tab, select [Properties].

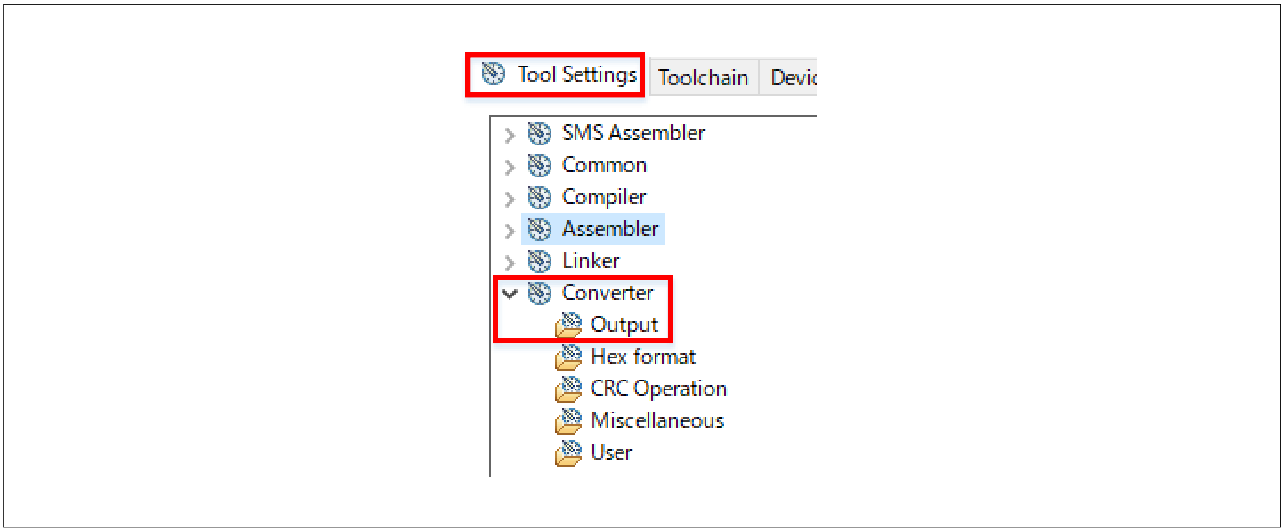
Under [C/C++ Build], select [Settings].

Figure 5-7 Generate a binary file in e2 studio (1/3)



Select [Converter] and [Output] in the [Tool Settings] tab.

Figure 5-8 Generate a binary file in e2 studio (2/3)



Select the [Run the load module converter] check box.

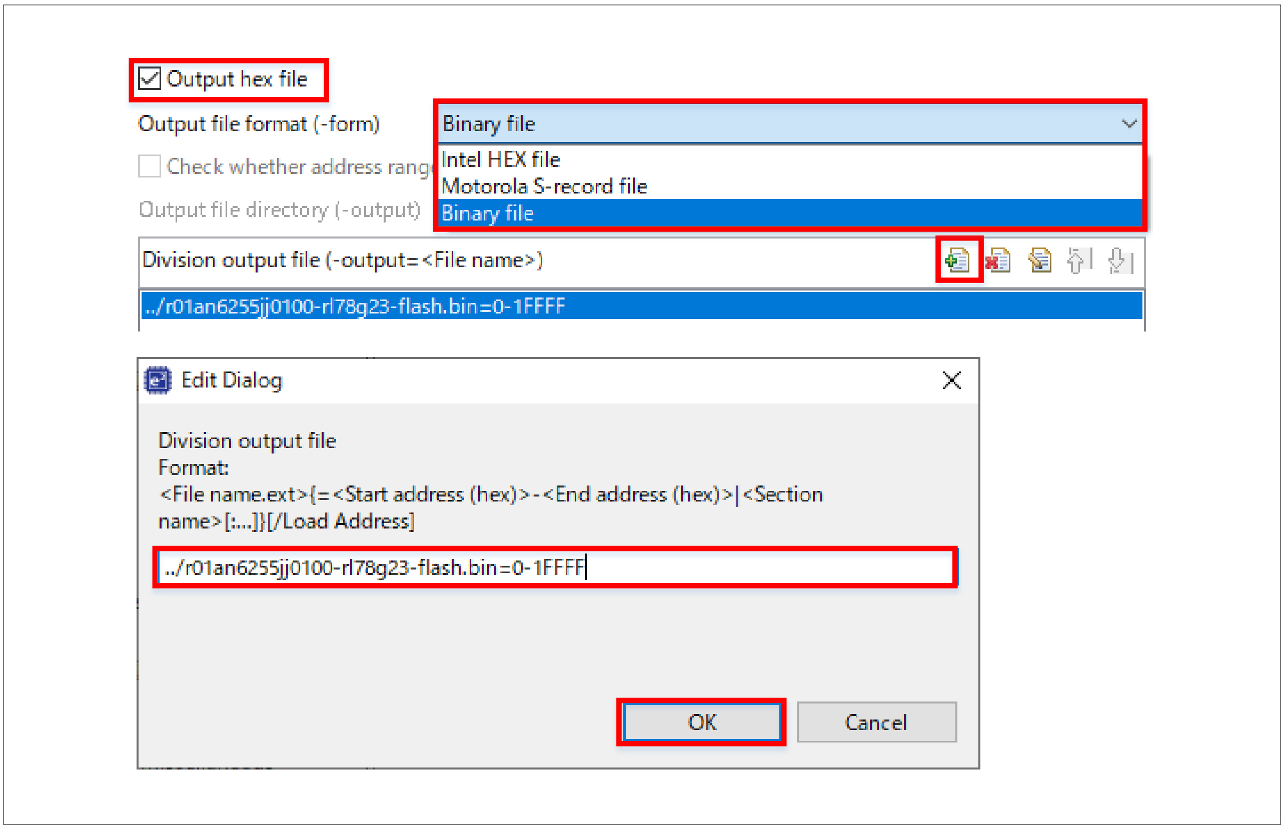
From the [Output file format] drop-down list, select [Output a binary file].

Click the [Add] button, and then enter a character string in the following pattern:

`../XXXXXX.bin=0-YYYYYY`

For *XXXXXX*, specify the project name. For *YYYYYY*, specify the last address of the code flash memory of the device to be used.

Figure 5-9 Generate a binary file in e2 studio (3/3)



A binary file is generated when you build a project.

5.1.3 Using IAR EW to Generate a Binary File

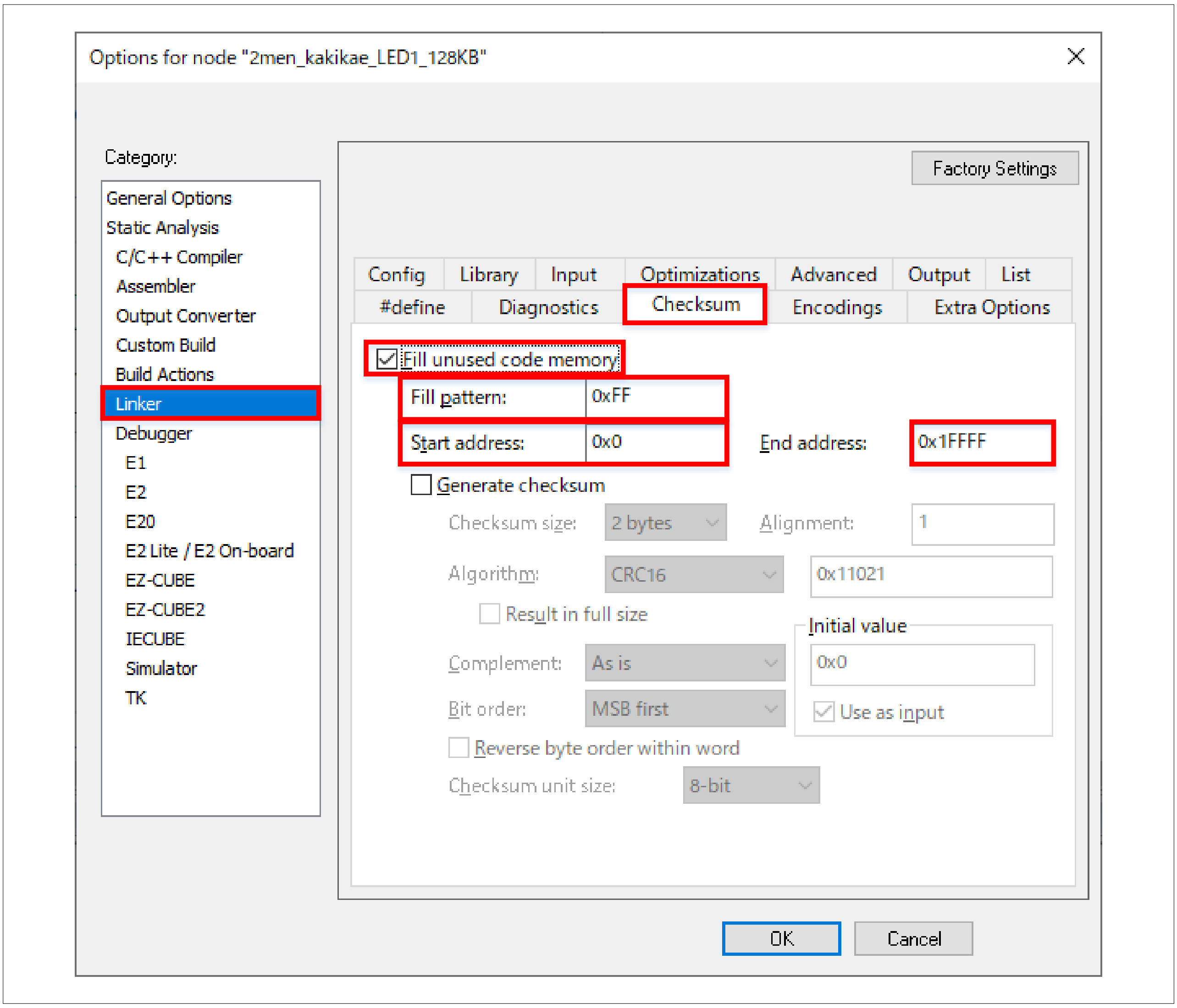
In the [Project] tab, select [Options].

In the [Category] list box, select [Linker], and then select the [Checksum] tab.

Select the [Fill unused code memory] check box.

For [Fill pattern], specify 0xFF. For [Start address], specify 0x0. For [End address], specify the last address of the code flash memory of the device to be used with a hexadecimal number prefixed by "0x".

Figure 5-10 Generate a binary file in IAR EW (1/2)

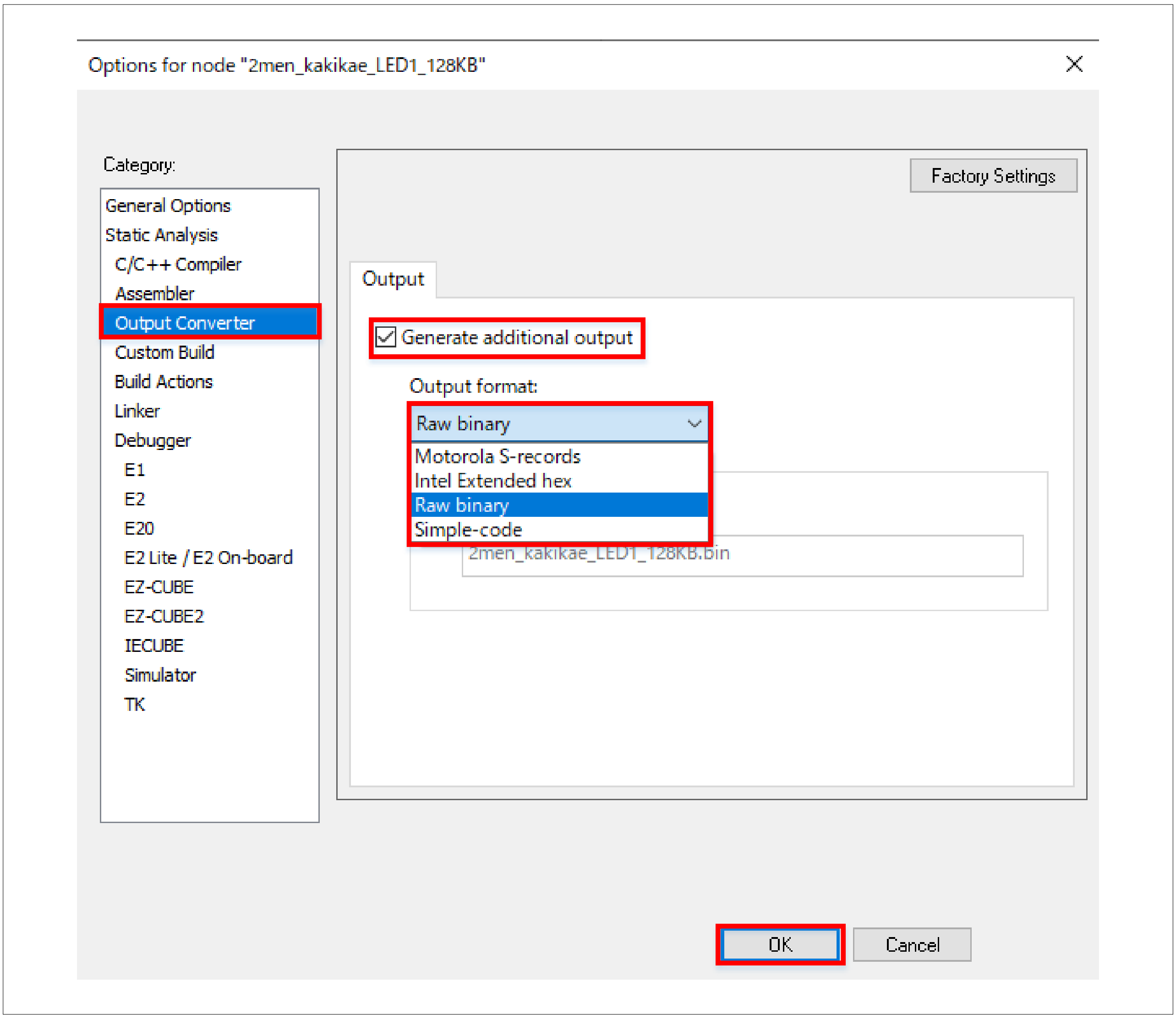


In the [Output Converter] tab, select the [Generate additional output] check box.

From the [Output format] drop-down list, select [Raw binary].

Then, click [OK]. A binary file is generated when you build a project.

Figure 5-11 Generate a binary file in IAR EW (2/2)

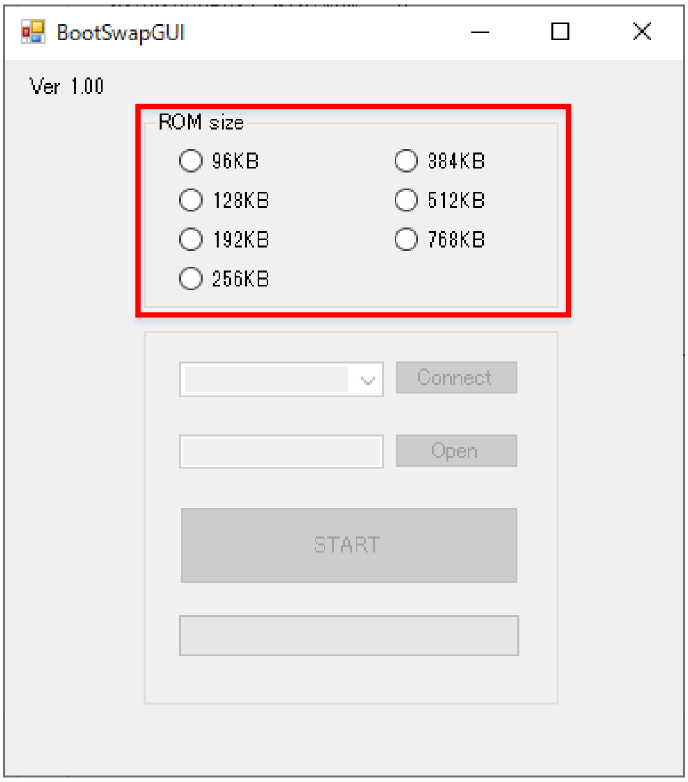


5.2 Using GUI-Based Tool

Run BootSwapGUI.exe.

Select the radio button for the ROM size of the device to be used.

Figure 5-12 Description of GUI (1/2)



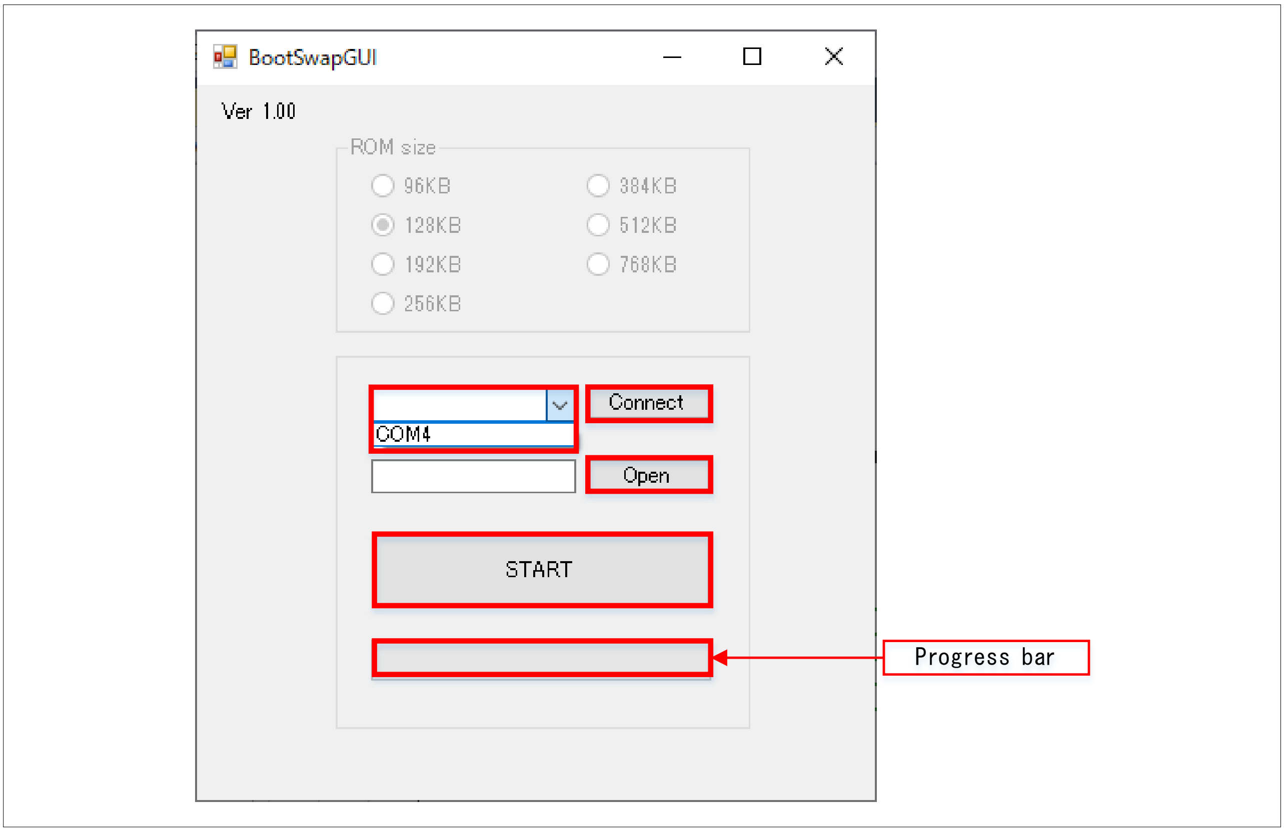
From the drop-down list, select an available COM port, and then click [Connect] to connect to the target device.

Click [Open], and then select the program (.bin) to be written.

Click [START] to start writing the program.

The progress bar shows the progress of write processing.

Figure 5-13 Description of GUI (2/2)



After writing is complete, exit BootSwapGUI.exe.

6. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

7. Reference Documents

RL78/G23 User's Manual: Hardware (R01UH0896)
RL78 family user's manual software (R01US0015)
The latest versions can be downloaded from the Renesas Electronics website.

Technical update
The latest versions 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

Rev.	Date	Description	
		Page	**Summary**
1.00	Aug.04.22	—	First Edition

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Corporate Headquarters

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

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.

Trademarks