RL78/G23

Self-Programming Using Boot Swapping via UART communications

Introduction

This application note gives the outline of self-programming via UART communications.

This application note explains how the flash self-programming code (Renesas Flash Driver RL78 Type01) is used to rewrite the boot area in 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

Outline

First, the sample program displays the current program version on the LCD module. Then, when it receives the START command via UART communications, it turns LED1 on (flash memory is being accessed) and enters the code flash programming mode. After that, the sample program erases the data that has been written to the code flash memory's boot cluster 1 (04000H to 07FFFH) and waits for the WRITE command.

When the sample program receives the WRITE command together with the rewrite data, it rewrites the contents of boot cluster 1. When the sample program completes rewriting and receives the END command, it turns LED1 off. If all processing performed before this point in time has terminated normally, the sample program generates an internal reset and performs boot swapping. After the sample program restarts, it displays the version of the new (rewritten) program on the LCD module.

Table 1-1 Peripheral Functions to be Used and their Uses

Peripheral Function	Use
Serial array unit UART0	Obtain of rewrite data
Serial interface IICA0	Communication with LCD module

1.1.1 Outline of the Flash Self-Programming Code (Renesas Flash Driver RL78 Type01)

The flash self-programming code is software for rewriting the data in the code flash memory installed on the sample program.

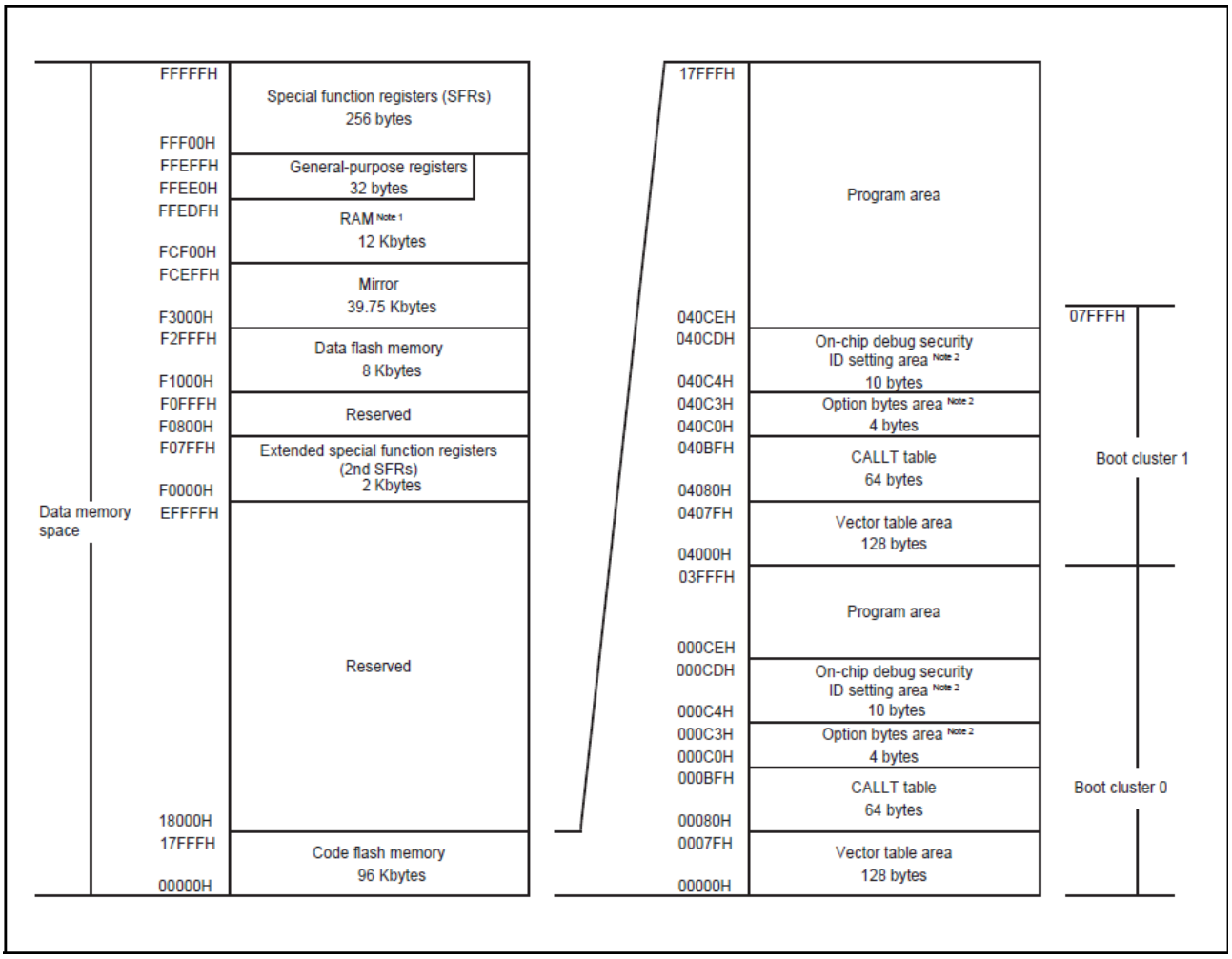
The contents of the code flash memory can be rewritten by calling the flash self-programming code from a user program.

To perform self-programming, the C or assembly language function corresponding to the self-programming initialization processing or function to be used must be run from a user program.

1.1.2 Code Flash Memory

The configuration of the RL78/G23 (R7F100GLG) code flash memory is shown below.

Figure 1-1 Code Flash Memory Configuration



Caution: When the boot swap function is used, the option byte area (000C0H to 000C3H) in boot cluster 0 is swapped with the option byte area (040C0H to 040C3H) in boot cluster 1. Accordingly, place the same values in the area (040C0H to 040C3H) as those in the area (000C0H to 000C3H) when using the boot swap function.

The features of the RL78/G23 code flash memory are summarized below.

Table 1-2 Features of the Code Flash Memory

Item	Description
Minimum unit of erasure and verification	1 block (2048 bytes)
Minimum unit of programming	1 word (4 bytes)
Security functions	Block erasure, programming, and boot cluster 0 reprogramming protection are supported. (They are enabled at shipment)
	It is possible to disable reprogramming and erasure outside the specified window only at flash memory self-programming time using the flash shield window.
	Security settings programmable using the flash self-programming code (Renesas Flash Driver RL78 Type01)

Caution: The boot cluster 0 reprogramming protection setting and the security settings for outside the flash shield window are disabled during flash memory self-programming.

1.1.3 Flash Memory Self-Programming

The RL78/G23 is provided with the flash self-programming code for flash memory self-programming. Flash memory self-programming is accomplished by calling functions of the flash self-programming code from the reprogramming program.

The flash self-programming code for the RL78/G23 controls flash memory reprogramming using a sequencer (a dedicated circuit for controlling flash memory). The code flash memory cannot be referenced while control by the sequencer is in progress. When the user program needs to be run while the sequencer control is in progress, therefore, it is necessary to relocate part of the segments for the flash self-programming code and the reprogramming program in RAM when erasing or reprogramming the code flash memory or making settings for the security flags. If there is no need to run the user program while the sequencer control is in progress, it is possible to keep the flash self-programming code and reprogramming program on ROM (code flash memory) for execution.

1.1.4 Boot Swap Function

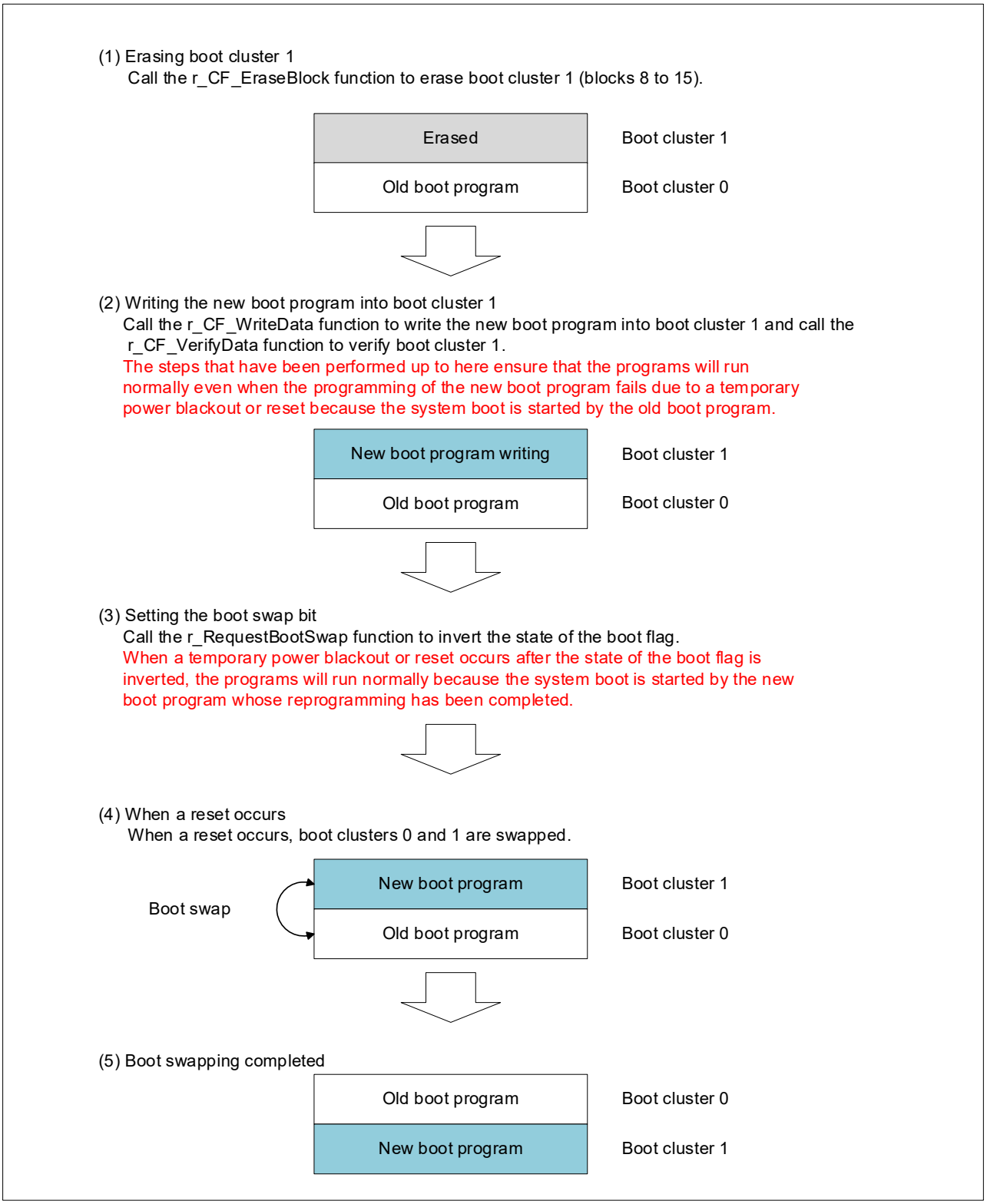
When reprogramming of the area where vector table data, the basic functions of the program, and flash self-programming code are allocated fails due to a temporary power blackout or a reset caused by an external factor, the data that is being reprogrammed will be corrupted, as the result of which the restarting of the user program or reprogramming cannot be accomplished when a reset is subsequently performed. This problem is be avoided by the introduction of the boot swap function.

The boot swap function swaps between boot cluster 0 which is the boot program area and boot cluster 1 which is the target of boot swapping. A new program is written into boot cluster 1 before reprogramming is attempted. This boot cluster 1 is swapped with boot cluster 0 and boot cluster 1 is designated as the boot program area. In this configuration, even when a temporary power blackout occurs while the boot program area is being reprogrammed, the system boot will start at boot cluster 1 on the next reset start, thus ensuring the normal execution of the programs.

The outline image of boot swapping is shown in the figure below.

Below is an image of boot swapping.

Figure 1-3 Outline of Boot Swapping

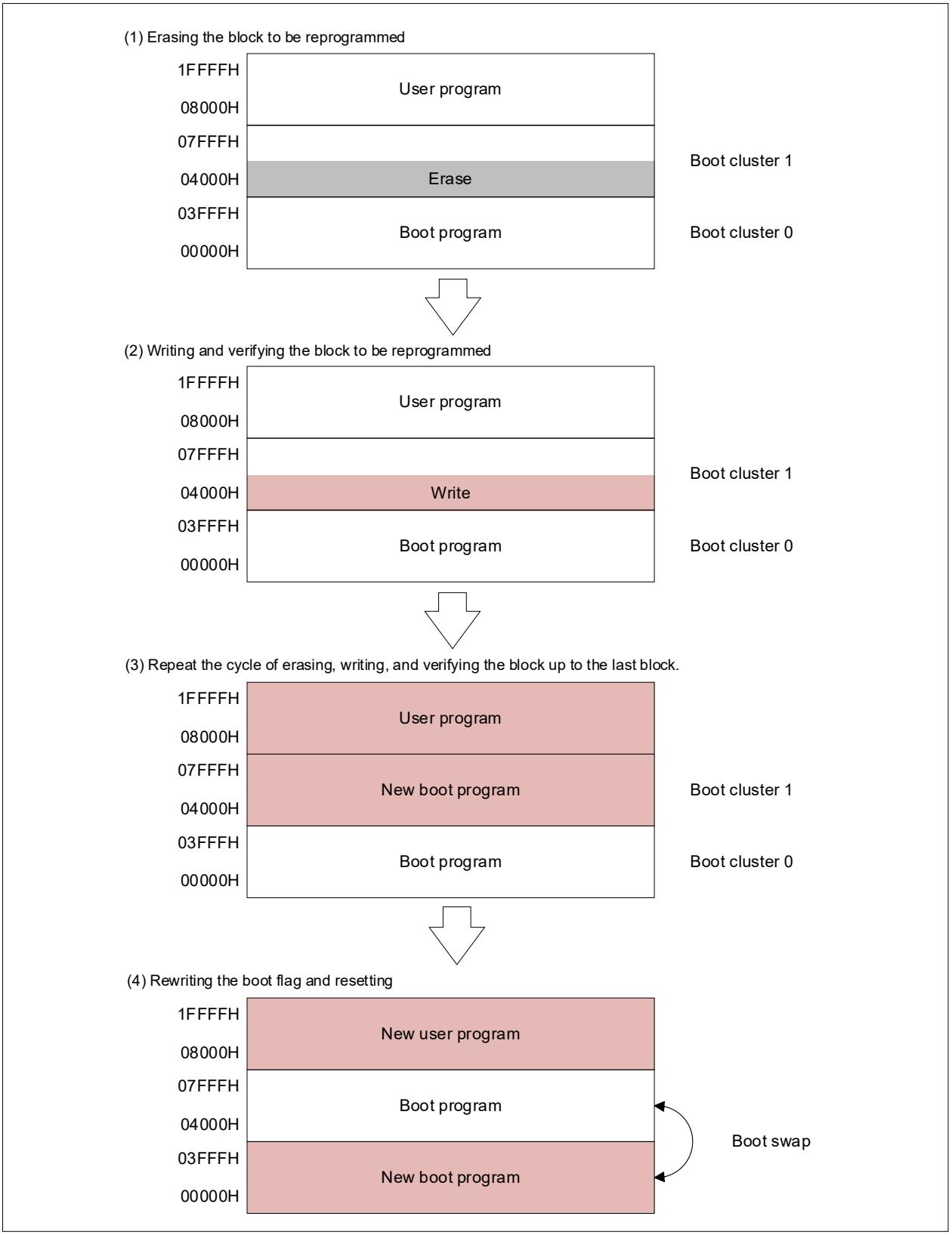


1.1.5 Flash Memory Reprogramming

This subsection describes the outline image of reprogramming using the flash memory self-programming technique. The flash memory self-programming program is located in boot cluster 0.

In this application note, the rewrite target is limited to the boot area.

Figure 1-4 Outline of Flash Memory Reprogramming

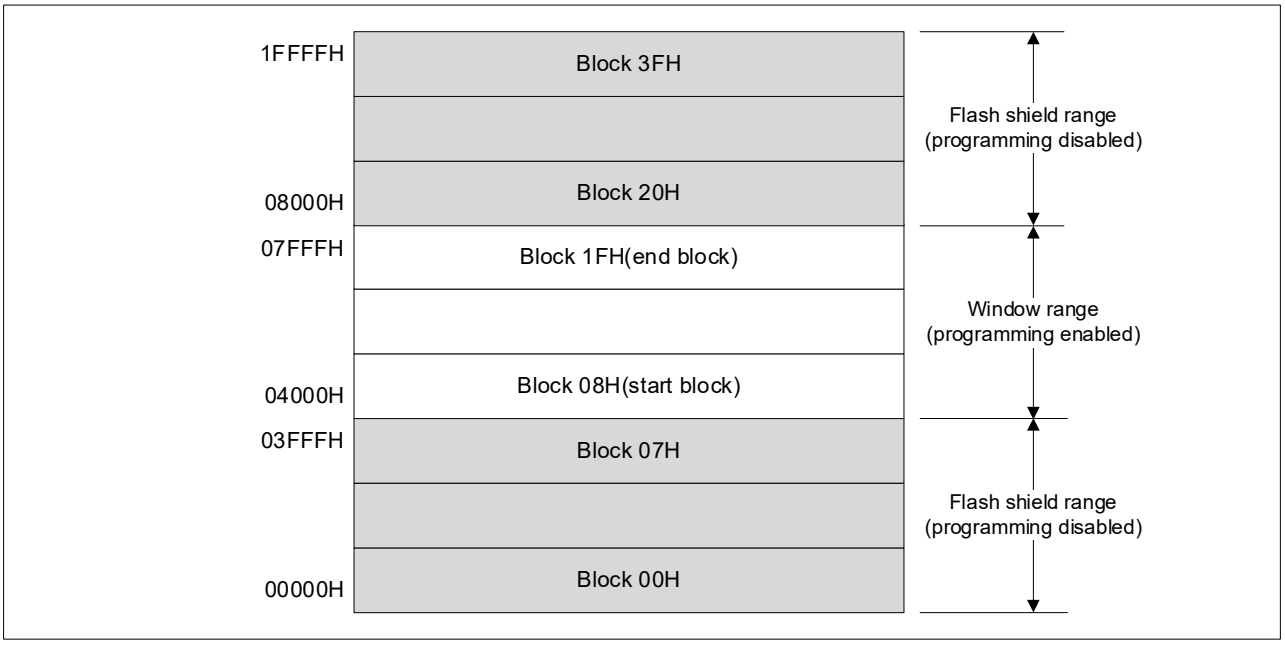


1.1.6 Flash Shield window

The flash shield window is one of security mechanisms used for flash memory self-programming. It disables the write and erase operations on the areas outside the designated window only during flash memory self-programming.

The figure below shows the outline image of the flash shield window on the area of which the start block is 08H and the end block is 0FH.

Figure 1-5 Outline of the Flash Shield Window



1.1.7 Communication Specifications

This application note explains how to perform self-programming via UART communications. The sample program performs the processing corresponding to the received command (START, WRITE, or END). If the processing terminates normally, the sample program sends 01H, which indicates normal response. If the processing terminates abnormally, the sample program displays "ERROR!" on the LCD module and terminates processing without sending data. The following shows the UART communication settings and the specifications of each command.

Table 1-2 UART Communication Settings

Data bit length [bit]	8
Data transfer direction	LSB first
Parity setting	No parity
Transfer rate [bps]	115200

- START command

When the sample program receives the START command, it initializes the self-programming settings. If the processing terminates normally, the sample program sends 01H, which indicates normal response. If the processing terminates abnormally, the sample program does not send data.

START code (01H)	Date length (0002H)	Command (02H)	Date (None)	Checksum (1 byte)

- WRITE command

When the sample program receives the WRITE command, it writes the received data to the flash memory. At this time, the sample program verifies the written data every 256 bytes. If the processing terminates normally, the sample program sends 01H, which indicates normal response. If the processing terminates normally, the sample program does not send data.

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

- END command

When the sample program receives the END command, it sends 01H as notification of response. The sample program then reverses the boot flag. If the processing terminates normally, the sample program generates a reset and performs boot swapping. If the processing terminates abnormally, it does not perform boot swapping.

START code (01H)	Date length (0002H)	Command (04H)	Date (None)	Checksum (1 byte)

- Abnormal termination

The sample program displays "ERROR!" on the LCD module and terminates processing.

- Checksum calculation method

The checksum is calculated by using the "32-bit addition calculation method".

The low-order 8 bits of the results of sequentially adding a value by one byte from 00000000H is used as the checksum for the command or data.

1.1.8 How to obtain the flash self-programming code

Before starting compilation, download the latest version of the flash self-programming code (Renesas Flash Driver RL78 Type01) and copy the file to the RFD_RL78_TYPE1 folder.

You can obtain the flash self-programming code from the following URL:

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

Operation Outline

This application note explains how to perform self-programming via UART communications.

(1) Initial settings

Initial port settings

- Set P53 as the output port (initial value: high level, LED1 turned off).

Initial settings of the serial array unit:

- Use channels 0 and 1 for a UART.
- Use the P12/TxD0 pin for data output. Use the P11/RxD0 pin for data input.
- Set the operation clock for CK00. Set the clock source for fCLK/2.
- Set an interrupt source for the transfer completion interrupt.
- Specify the following settings: No parity bit, transfer order = LSB first, stop bit length = 1 bit, data length = 8 bits
- Set non-reverse (standard) sending.
- Set the baud rate to 115,200 bps.

Initial settings of the IICA serial interface:

- Use the IICA0 (P60/SCLA0 and P61/SDAA0 pins).
- Set the operation clock of the IICA0 for fCLK/2.
- Set the local address for 10H.
- Set the operation mode to "standard".
- Set the transfer click to 80,000 bps.
- Permit the INTIICA0 interrupt.

Initial settings of the LCD module and display of the current program version:

- Display the string of the LCD_STRING constant on the LCD module.

Initialization of Renesas Flash Driver RL78 Type01

(2) Processing of the START command

- Set the P53 pin to low output level and turn LED1 (flash memory being accessed) on.
- Use the r_CF_EraseBlock function to erase the data of boot cluster 1 (04000H to 07FFFH).
  If the processing terminates normally, the sample program sends 01H, which indicates normal response.
  If the processing terminates abnormally, the sample program does not send data.

(3) Processing of the WRITE command

- Receive the data to be written (256 bytes).
- Use the r_CF_WriteData function to write the received data to the write-destination address.

  Increase the write-destination address by the size of written data.
- Use the r_CF_VerifyData function to verify the written data against the received data every 256 bytes.
- If the processing terminates normally, the sample program sends 01H, which indicates normal response.
  If the processing terminates abnormally, the sample program does not send data.

(4) Processing of the END command

- Set the P53 pin to high level output and turn LED1 (flash memory being accessed) off.
- Send 01H, which indicates normal response.
- Use the r_RequestBootSwap function to reverse the value of the boot flag.
  If ret_value is normal, the sample program generates an internal reset.
  The generated internal reset will exchange boot clusters 0 and 1.
  If the processing terminates normally, the sample program reverses the boot flag to generate a reset and performs boot swapping. If the processing terminates abnormally, the sample program does not perform boot swapping.

(5) Handling of abnormal termination

- The sample program displays "ERROR!" on the LCD module and terminates processing.

Note 1: If data has already been completely written up to the last address (07FFFH) of boot cluster 1, the sample program writes no more data even when a new WRITE command is received.

Note 2: When the sample program receives the END command (04H), it always sends 01H, which indicates normal response, and sets the P52 to high level output (LED1 turned off). The r_RequestBootSwap function is run to perform boot swapping.

Note 3: If self-programming does not terminate normally, the sample program displays "ERROR!" on the LCD module and performs no subsequent processing.

2. Operation Check Conditions

The sample code described in this application note has been checked under the conditions listed in the table below.

Table 2-1 Operation Check Conditions

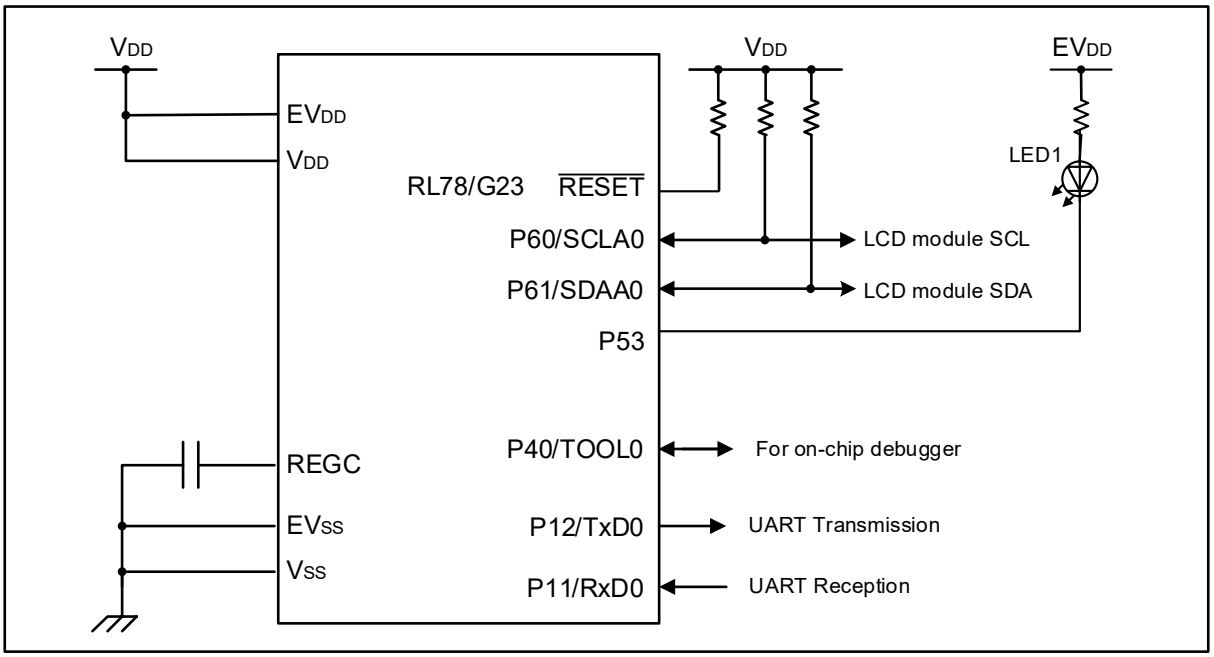
Item	Description
Microcontroller used	RL78/G23 (R7F100GLG)
Operating frequency	High-speed on-chip oscillator (fIH): 32MHz
Operating voltage	3.3 V (can be operated at 3.1 V to 5.5 V) LVD operations (V_{LVD}): Reset mode At rising edge TYP. 1.90 V At falling edge TYP. 1.86 V
Integrated development environment (CS+)	CS+ for CC V8.05.00 from Renesas Electronics Corp.
C compiler (CS+)	CC-RL V1.10.00 from Renesas Electronics Corp.
Integrated development environment (e^2 studio)	e2studio V2021-04(21.4.0) from Renesas Electronics Corp.
C compiler (e^2 studio)	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 V4.21.1 from IAR Systems Corp.
Board support package (BSP)	V1.0.1 from Renesas Electronics Corp.
Board to be used	RL78/G23-64p Fast Prototyping Board, RTK7RLG230CLG000BJ

3. Description of the Hardware

Hardware Configuration Example

Figure 3-1 shows an example of the hardware configuration used for this 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. V_{DD} must be held at not lower than the reset release voltage (V_{LVD0}) that is specified as LVD0.

List of Pins to be Used

Table 3-1 lists pins to be used and their functions.

Table 3-1 Pins to be Used and their Functions

Pin name	I/O	Description
P12//TxD0	Output	Pin for sending UART serial data
P11/ RxD0	Input	Pin for receiving UART serial data
P53	Output	Pin used to turn on or off LED1, which indicates the access status of flash memory
P60/SCLA0、P61/SDAA0	Input/Output	Pin used for I2C communication with the LCD module

Caution In this application note, only the pins used are processed. When actually creating a circuit, perform pin processing appropriately and design it so that it satisfies the electrical characteristics.

4. Software Explanation

List of Option Byte Settings

Table 4-1 summarizes the settings of the option bytes.

Table 4-1 Option Byte Settings

Address	Setting	Description
000C0H/040C0H	11101111B	Disables the watchdog timer. (Stops counting after the release from the reset status.)
000C1H/040C1H	11111110B	LVD operations (V_{LVD}): Reset mode At rising edge TYP. 1.90 V At falling edge TYP. 1.86 V
000C2H/040C2H	11101000B	HS mode High-speed on-chip oscillator clock: 32MHz
000C3H/040C3H	10000101B	Enables the on-chip debugger

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.

Startup routine settings

4.2.1 Definition of the section for the stack area (.stack_bss)

In the sample program, the data to be written to boot cluster 1 is saved in a local variable. Because local variables are placed in stack areas, you need to modify "cstart.asm" so that any stack area of your choice is secured and the stack area is initialized.

```
;$IF (__RENESAS_VERSION__ < 0x01010000)                Add ';' to the first line and comment out
;-------------------------------------------------------------------------------
;     stack area
;-------------------------------------------------------------------------------
; !!! [CAUTION] !!!
; Set up stack size suitable for a project.
.SECTION .stack_bss, BSS
_stackend:
      .DS       0x200
_stacktop:
;$ENDIF                                                Add ';' to the first line and comment out
    .
    .
    .
    .
      ;------------------------------------------------
      ; setting the stack pointer
      ;------------------------------------------------
;$IF (__RENESAS_VERSION__ >= 0x01010000)               Add ';' to the first line and comment out
;      MOVW      SP,#LOWW(__STACK_ADDR_START)          Add ';' to the first line and comment out
;$ELSE     ; for CC-RL V1.00                           Add ';' to the first line and comment out
      MOVW      SP,#LOWW(_stacktop)
;$ENDIF                                                Add ';' to the first line and comment out

      ;------------------------------------------------
      ; initializing stack area
      ;------------------------------------------------
;$IF (__RENESAS_VERSION__ >= 0x01010000)               Add ';' to the first line and comment out
;      MOVW      AX,#LOWW(__STACK_ADDR_END)            Add ';' to the first line and comment out
;$ELSE     ; for CC-RL V1.00                           Add ';' to the first line and comment out
      MOVW      AX,#LOWW(_stackend)
;$ENDIF                                                Add ';' to the first line and comment out
      CALL      !!_stkinit
```

4.2.2 Deploying the Rewrite Programs in the RAM Area

Deploy the programs that will be used to rewrite boot cluster 1 in the RAM area. These programs are deployed in the sections listed in Table 4-2.

Table 4-2 Section Information

Section Name	Deployment-destination section name	Item to Be Deployed
RFD_CMN_f	RFD_CMN_fR	Program section for the common flash memory control API function
RFD_CF_f	RFD_CF_fR	Program section for the code flash memory API function
RFD_EX_f	RFD_EX_fR	Program section for the extra area control API function
SMP_CMN_f	SMP_CMN_fR	Program section for the common flash memory control sample function
SMP_CF_f	SMP_CF_fR	Program section for the code flash memory control sample function

To deploy the preceding sections in the RAM area, you need to add processing to "cstart.asm".

In "cstart.asm", add code for the processing after the following lines:

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

The code to be added is as follows:

```
    ; copy .text to RAM (section-name)
    MOV     C,#HIGHW(STARTOF(section-name))
    MOVW    HL,#LOWW(STARTOF(section-name))
    MOVW    DE,#LOWW(STARTOF(Placement section name))
    BR      $.L12_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     $.L11_TEXT
```

- In **section-name**, specify the name of the section to be deployed.
- Add the preceding code for each section that needs to be deployed.
- For m, set any number of your choice. Specify a different number for each section.

On-chip Debug Security ID

The RL78/G23 has the on-chip debug security ID area allocated to addresses 000C4H to 000CDH of flash memory to preclude the memory contents from being sneaked by the unauthorized third party.

When using the boot swap function for self-programming, it is necessary to set the same values that are set in 000C4H to 000CDH also in 040C4H to 040CDH because bytes in 000C4H to 000CDH are swapped with the bytes in 040C4H to 040CDH.

Resources Used by the Sample Program

4.4.1 List of Sections in the ROM Area

Table 4-3 lists the sections that are deployed in the ROM area and used by the sample program.

Table 4-3 List of Sections in the ROM Area

Section Name	Item to Be Deployed
RFD_CMN_f	Program section for the common flash memory control API function
RFD_CF_f	Program section for the code flash memory control API function
RFD_EX_f	Program section for the extra area control API function
RFD_DF_f	Program section for the data flash memory control API function
SMP_CMN_f	Program section for the common flash memory control sample function
SMP_CF_f	Program section for the code flash memory control sample function

4.4.2 List of Sections in the RAM Area

Table 4-4 lists the sections that are deployed in the RAM area and used by the sample program.

Table 4-4 List of Sections in the RAM Area

Section Name	Items to Be Deployed
RFD_DATA_n	Data section for RFD RL78 Type01
RFD_CMN_fR	Program section for the common flash memory control API function
RFD_CF_fR	Program section for the code flash memory control API function
RFD_EX_fR	Program section for the extra area control API function
SMP_CMN_fR	Program section for the common flash memory control sample function
SMP_CF_fR	Program section for the code flash memory control sample function

List of Constants

Table 4-5 lists the constants for the sample program.

Table 4-5 Constants for the Sample Program

Constant	Setting	Description
LED_ON	00H	LED ON
LED_OFF	01H	LED OFF
START_WRITE_ADDRESS	00004000H	Write start address
END_WRITE_ADDRESS	00007FFFH	Write end address
WRITE_DATA_SIZE	0100H	Size of data to be 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
CPU_FREQUENCY	32	CPU operating frequency
COMMAND_START	02H	Command code: START
COMMAND_WRITE	03H	Command code: WRITE
COMMAND_END	04H	Command code: END
COMMAND_ERROR	FFH	Command code: ERROR
VALUE_U08_MASK1_FSQ_STATUS_ERR_ERASE	01H	Error status mask value for the execution result of the flash memory sequencer bit0: Erase command error
VALUE_U08_MASK1_FSQ_STATUS_ERR_WRITE	02H	Error status mask value for the execution result of the flash memory sequencer bit1: Write command error
VALUE_U08_MASK1_FSQ_STATUS_ERR_BLANKCHECK	08H	Error status mask value for the execution result of the flash memory sequencer bit3: Blank check command error
VALUE_U08_MASK1_FSQ_STATUS_ERR_CFDF_SEQUENCER	10H	Error status mask value for the execution result of the flash memory sequencer bit4: Code/data flash area sequencer error
VALUE_U08_MASK1_FSQ_STATUS_ERR_EXTRA_SEQUENCER	20H	Error status mask value for the execution result of the flash memory sequencer bit5: Extra area sequencer error
VALUE_U08_SHIFT_ADDR_TO_BLOCK_CF	11	Constant used for bit shifting performed to calculate the block number of Code Flash
VALUE_U08_SHIFT_ADDR_TO_BLOCK_DF	8	Constant used for bit shifting performed to calculate the block number of Data Flash
VALUE_U01_MASK0_1BIT	0	1-bit mask value
VALUE_U01_MASK1_1BIT	1	1-bit mask value
VALUE_U08_MASK0_8BIT	00H	8-bit mask value
VALUE_U08_MASK1_8BIT	FFH	8-bit mask value

Enumerated type

Table 4-6 shows the definition of the enumeration used in the sample program.

Table 4-6 enum e_ret (Enumerated variable name: e_ret_t)

Symbol Name	Setting	Description
ENUM_RET_STS_OK	00H	Normal status
ENUM_RET_ERR_CFDF_SEQUENCER	10H	Code/data flash area sequencer error
ENUM_RET_ERR_EXTRA_SEQUENCER	11H	Extra area sequencer error
ENUM_RET_ERR_ERASE	12H	Erase error
ENUM_RET_ERR_WRITE	13H	Write error
ENUM_RET_ERR_BLANKCHECK	14H	Blank error
ENUM_RET_ERR_CHECK_WRITE_DATA	15H	Error in comparison between the written and read values
ENUM_RET_ERR_MODE_MISMATCHED	16H	Mode mismatch error
ENUM_RET_ERR_PARAMETER	17H	Parameter error
ENUM_RET_ERR_CONFIGURATION	18H	Device configuration error

List of Variables

Table 4-7 shows the definition of the global variables used in the sample program.

Table 4-7 Global Variables

Type	Variable Name	Description	Function Used
uint8_t	f_UART0_sendend	Flag indicating that data sending by the UART0 was completed	r_Send_nByte r_Config_UART0_callback_sendend
uint8_t	f_UART0_receiveend	Flag indicating that data reception by the UART0 was completed	r_Receive_nByte r_Config_UART0_callback_receiveend

List of Functions

Table 4-8 and Table 4-9 lists the functions that are used in this sample program.

Table 4-8 List of Functions (1/2)

Function Name	Outline
r_rfd_initialize	Initialization processing for RFD RL78 Type01
r_cmd_start	START command processing
r_cmd_write	WRITE 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_Config_UART0_callback_receiveend	Callback processing at a reception completion interrupt for UART0
r_RecvPacket	Command reception processing by UART0
r_ReceivePacketAnalyze	Command analysis processing by UART0
r_Receive_nByte	Data reception processing by UART0
r_Send_nByte	Data sending processing by UART0
r_SendACK	Normal response sending processing by UART0
r_Config_IICA0_callback_master_sendend	Callback processing at a sending completion interrupt for IICA0
r_Config_IICA0_callback_master_error	Callback processing at a sending error interrupt for IICA0
r_LCM_init	Processing to initialize the LCD module
r_LCM_clear	Processing to clear display for the LCD module
r_LCM_send_string	Processing to send strings to the LCD module
r_LCM_send_command	Command sending processing for the LCD module
r_LCM_send_data	Processing to send data to the LCD module
r_LCM_turn_sendend_on	Communication end flag setting for the LCD module
r_LCM_wait_sendend	Communication end wait processing for the LCD module

Table 4-9 List of Functions (2/2)

R_RFD_Init[Note]	Initialization processing for RFD RL78 Type01
R_RFD_SetFlashMemoryMode[Note]	Flash memory control mode change processing
R_RFD_EraseCodeFlashReq[Note]	Code flash memory erase processing
R_RFD_WriteCodeFlashReq[Note]	Code flash memory write processing
R_RFD_CheckCFDFSeqEndStep1[Note]	Processing to check whether the code/data flash area sequencer has terminated
R_RFD_CheckCFDFSeqEndStep2[Note]	Processing to check whether the command was terminated by clearing the flash memory sequencer control register
R_RFD_GetSeqErrorStatus[Note]	Processing to obtain error information generated by the code/data flash area sequencer command or extra area sequencer command
R_RFD_ClearSeqRegister[Note]	Processing to clear the register that controls the code/data flash area sequencer or extra area sequencer
R_RFD_CheckExtraSeqEndStep1[Note]	Processing to confirm that the extra area sequencer has terminated
R_RFD_CheckExtraSeqEndStep2[Note]	Processing to check whether the command was terminated by clearing the extra area sequencer control register
R_RFD_GetSecurityAndBootFlags[Note]	Processing to obtain the security flag and boot area switching flag
R_RFD_SetDataFlashAccessMode[Note]	Processing to set whether to permit or prohibit access to the data flash memory
R_RFD_SetExtraBootAreaReq[Note]	Boot area switching flag write processing
R_RFD_ForceReset[Note]	Internal CPU reset request

Note: This is an API function defined for the flash self-programming code. For details about the API function, see the "RL78 Family Renesas Flash Driver RL78 Type01 User's Manual".

Function Specifications

This section describes the specifications for the functions that are used in the sample program.

r_rfd_initialize	
Synopsis	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 initializes RFD RL78 Type01.
Arguments	None
Return value	ENUM_RET_STS_OK: Normal status ENUM_RET_ERR_CONFIGURATION: Device configuration error ENUM_RET_ERR_PARAMETER: Parameter error

r_cmd_start	
Synopsis	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 required when the START command is received.
Arguments	None
Return value	ENUM_RET_STS_OK: Normal status ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_ERASE: Erase error

r_cmd_write	
Synopsis	WRITE command processing
Header	r_rfd_common_api.h、r_cg_userdefine.h
Declaration	R_RFD_FAR_FUNC e_ret_t r_cmd_write(uint32_t* write_start_addr, uint8_t __near * write_data);
Explanation	This function performs processing required when the WRITE command is received.
Arguments	uint32_t i_u32_start_addr: Write start address uint8_t __near * inp_u08_write_data: Write data
Return value	ENUM_RET_STS_OK: Normal status ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_ERASE: Erase error

r_cmd_end	
Synopsis	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 required when the END command is received. If this function terminates normally, an internal reset occurs and the CPU is restarted.
Arguments	None
Return value	ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error

r_CF_RangeErase	
Synopsis	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 value	ENUM_RET_STS_OK: Normal status ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_ERASE: Erase error

r_CF_EraseBlock	
Synopsis	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 value	ENUM_RET_STS_OK: Normal status ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_ERASE: Erase error

r_CF_WriteVerifySequence	
Synopsis	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 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 i_u32_start_addr: Write start address uint16_t i_u16_write_data_length: Write size uint8_t __near * inp_u08_write_data: Write data
Return value	ENUM_RET_STS_OK: Normal status ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_WRITE: Write error ENUM_RET_ERR_CHECK_WRITE_DATA: Error in comparison between the written and read values

r_CF_WriteData	
Synopsis	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 i_u32_start_addr, uint16_t i_u16_write_data_length, uint8_t __near * inp_u08_write_data);
Explanation	This function writes data to the code flash memory.
Arguments	uint32_t i_u32_start_addr: Write start address uint16_t i_u16_write_data_length: Write size uint8_t __near * inp_u08_write_data: Write data
Return value	ENUM_RET_STS_OK: Normal status ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error ENUM_RET_ERR_WRITE: Write error

r_CF_VerifyData	
Synopsis	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: Comparison data
Return value	ENUM_RET_STS_OK: Normal status (match) ENUM_RET_ERR_CHECK_WRITE_DATA: Error in comparison between the written and read values (Mismatch)

r_CheckCFDFSequencerEnd	
Synopsis	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 value	ENUM_RET_STS_OK: Normal status ENUM_RET_ERR_CFDF_SEQUENCER: Code/data flash area sequencer error ENUM_RET_ERR_ERASE: Erase error ENUM_RET_ERR_WRITE: Write error ENUM_RET_ERR_BLANKCHECK: Blank error

r_CheckExtraSequencerEnd	
Synopsis	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 memory sequence has terminated.
Arguments	None
Return value	ENUM_RET_STS_OK: Normal status ENUM_RET_ERR_EXTRA_SEQUENCER: Code/data flash area sequencer error ENUM_RET_ERR_ERASE: Erase error ENUM_RET_ERR_WRITE: Write error ENUM_RET_ERR_BLANKCHECK: Blank error

r_RequestBootSwap	
Synopsis	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 value	ENUM_RET_ERR_MODE_MISMATCHED: Mode mismatch error

r_Config_UART0_callback_sendend()	
Synopsis	Callback processing at a sending completion interrupt for UART0
Header	r_cg_macrodriver.h、Config_IICA0.h、LCM_driver.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 value	None

r_Config_UART0_callback_receiveend	
Synopsis	Callback processing at a reception completion interrupt for UART0
Header	r_cg_macrodriver.h、Config_IICA0.h、LCM_driver.h
Declaration	static void r_Config_UART0_callback_receiveend(void);
Explanation	This is a callback function that is called at a reception completion interrupt for UART0.
Arguments	MD_STATUS flag: Error type
Return value	None

r_RecvPacket	
Synopsis	Command reception processing by UART0
Header	r_cg_macrodriver.h、r_cg_userdefine.h
Declaration	MD_STATUS r_RecvPacket(uint8_t *data, uint16_t *length);
Explanation	This function uses UART0 to perform command reception processing. This function waits until reception of a packet of data is completed.
Arguments	uint8_t *data: Address of receive data buffer uint16_t *length: Address of area storing receive data length
Return value	MD_OK: Normal status [reception completion] COMMAND_ERROR: Parameter error

r_ReceivePacketAnalyze	
Synopsis	Command analysis processing by UART0
Header	r_cg_userdefine.h
Declaration	uint8_t r_ReceivePacketAnalyze(uint8_t *rxbuf, uint16_t rxlength);
Explanation	This function checks the checksum of received data. If the checksum matches, the function obtains the command code in the received data.
Arguments	uint8_t *rxbuf: Address of receive data buffer uint16_t rxlength: Address of area storing receive data length
Return value	COMMAND_START: Receive START command COMMAND_WRITE: Receive WRITE command COMMAND_END: Receive END command COMMAND_ERROR: Checksum error or command code error

r_Receive_nByte	
Synopsis	Data reception processing by UART0
Header	Config_UART0.h、Config_WDT.h
Declaration	MD_STATUS r_Receive_nByte(uint8_t *rx_buff, const uint16_t rx_num);
Explanation	This function performs reception processing by UART0. This function waits until reception of the number of characters specified for an argument is completed.
Arguments	uint8_t *rx_buff: Address of receive data buffer const uint16_t rx_num: Number of characters received
Return value	MD_OK: Normal status [reception completion] MD_ARGERROR: Parameter error

r_Send_nByte	
Synopsis	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 *tx_buff: Address of send data buffer const uint16_t tx_num: Number of characters to send
Return value	MD_OK: Normal status [reception completion] MD_ARGERROR: Parameter error

r_SendACK	
Synopsis	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 value	MD_OK: Normal status [sending completion] MD_ARGERROR: Parameter error

r_Config_IICA0_callback_master_sendend

Synopsis	Callback processing at a sending completion interrupt for IICA0
Header	r_cg_macrodriver.h、Config_IICA0.h、LCM_driver.h
Declaration	static void r_Config_IICA0_callback_master_receiveend(void);
Explanation	This is a callback function that is called at a sending completion interrupt for IICA0.
Arguments	None
Return value	None

r_Config_IICA0_callback_master_error

Synopsis	Callback processing at a sending error interrupt for IICA0
Header	r_cg_macrodriver.h、Config_IICA0.h、LCM_driver.h
Declaration	static void r_Config_IICA0_callback_master_error(MD_STATUS flag);
Explanation	This is a callback function that is called at a sending error interrupt for IICA0.
Arguments	MD_STATUS flag: Error type
Return value	None

r_LCM_init

Synopsis	Processing to initialize the LCD module
Header	LCM_driver.h、Config_IICA0.h
Declaration	void r_LCM_init(void);
Explanation	This function initializes the LCD module.
Arguments	None
Return value	None

r_LCM_clear

Synopsis	Processing to clear display for the LCD module
Header	LCM_driver.h、Config_IICA0.h
Declaration	void r_LCM_clear(void);
Explanation	This function sends the Clear Display command to the LCD module.
Arguments	None
Return value	None

r_LCM_send_string	
Synopsis	Processing to send strings to the LCD module
Header	LCM_driver.h、Config_IICA0.h
Declaration	void r_LCM_send_string(uint8_t * const str, lcm_position_t pos);
Explanation	This function displays the character string passed by using the "str" argument on the LCD module. A line can also be displayed by using the "pos" argument.
Arguments	uint8_t * const str: Character string to be displayed lcm_position_t pos: Displayed at the top with LCM_POSITION_TOP Displayed at the bottom with LCM_POSITION_BOTTOM.
Return value	None

r_LCM_send_command	
Synopsis	Command sending processing for the LCD module
Header	LCM_driver.h、Config_IICA0.h
Declaration	void r_LCM_send_command(uint8_t command);
Explanation	This function sends the command passed by using the "command" argument to the LCD module.
Arguments	uint8_t command: Command to send to LCD module
Return value	None

r_LCM_send_data	
Synopsis	Processing to send data to the LCD module
Header	LCM_driver.h、Config_IICA0.h
Declaration	void r_LCM_send_data(uint8_t data);
Explanation	This function sends the data passed by using the "data" argument to the LCD module.
Arguments	uint8_t data: Data to be sent to the LCD module
Return value	None

r_LCM_turn_sendend_on	
Synopsis	Communication end flag setting for the LCD module
Header	LCM_driver.h、Config_IICA0.h
Declaration	void r_LCM_turn_sendend_on(void);
Explanation	This function sets (for g_LCM_is_sendend) the flag that indicates the end of IIC communication with the LCD module.
Arguments	None
Return value	None

r_LCM_wait_sendend	
Synopsis	Communication end wait processing for the LCD module
Header	LCM_driver.h、Config_IICA0.h
Declaration	static void r_LCM_wait_sendend(void);
Explanation	This function waits until IIC communication with the LCD module ends, and then waits for the command execution wait time (5 ms).
Arguments	None
Return value	None

Flowcharts

4.10.1 Main Processing

Figure 4-1 to Figure 4-2 shows the flowchart for main processing.

Figure 4-1 Main Processing (1/2)

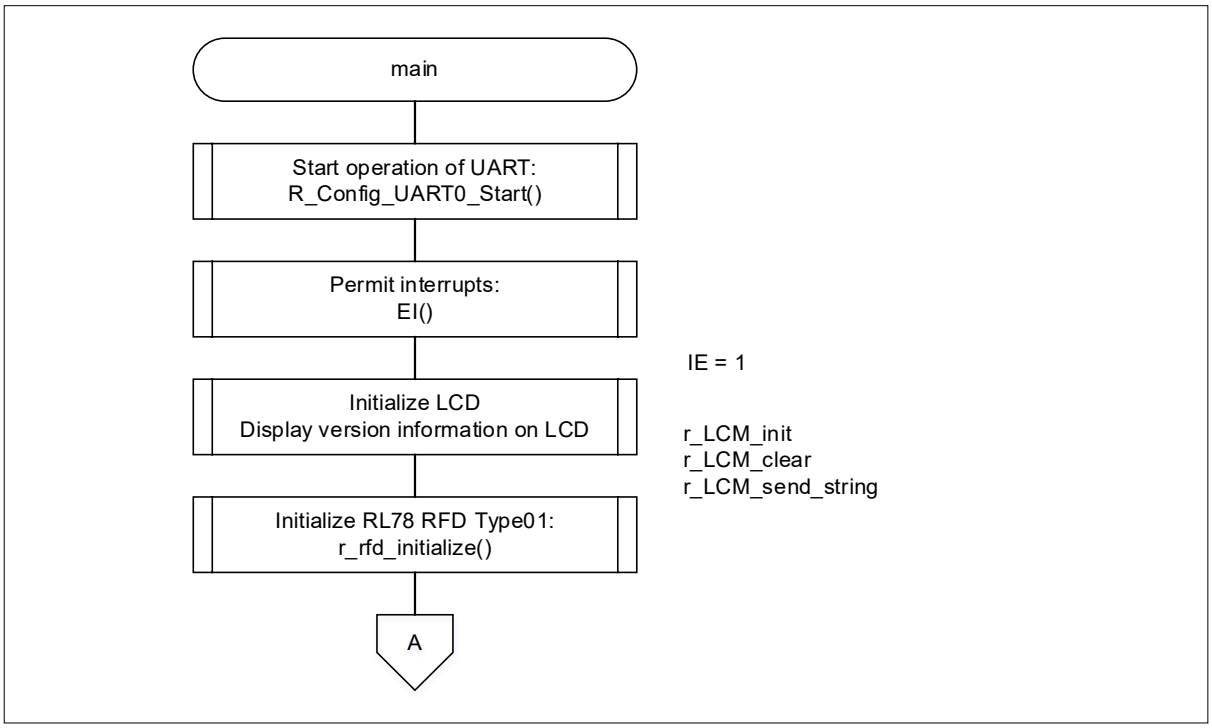
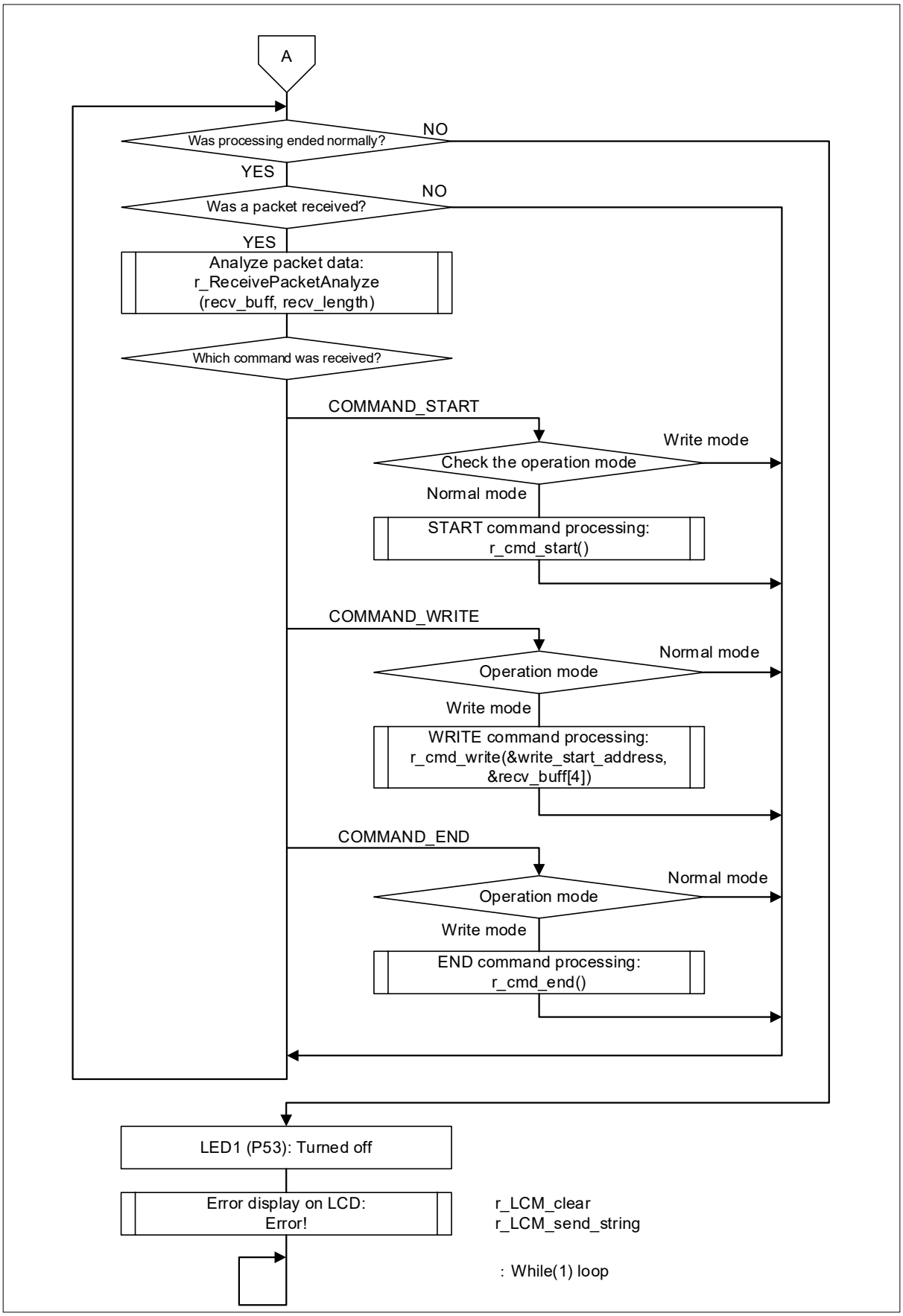


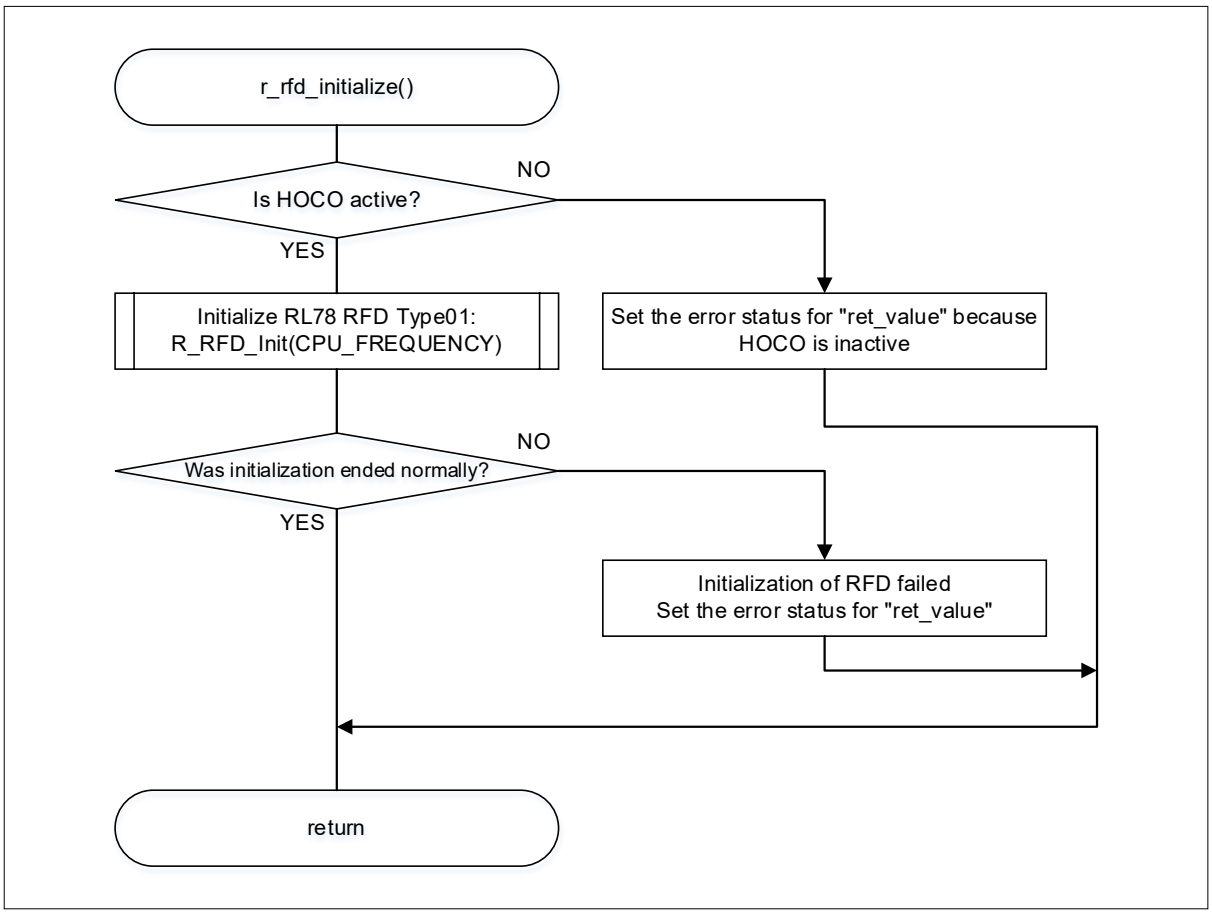
Figure 4-2 Main Processing (2/2)



4.10.2 Initialization Processing for RFD RL78 Type01

Figure 4-3 shows the flowchart for initialization processing for RFD RL78 Type01.

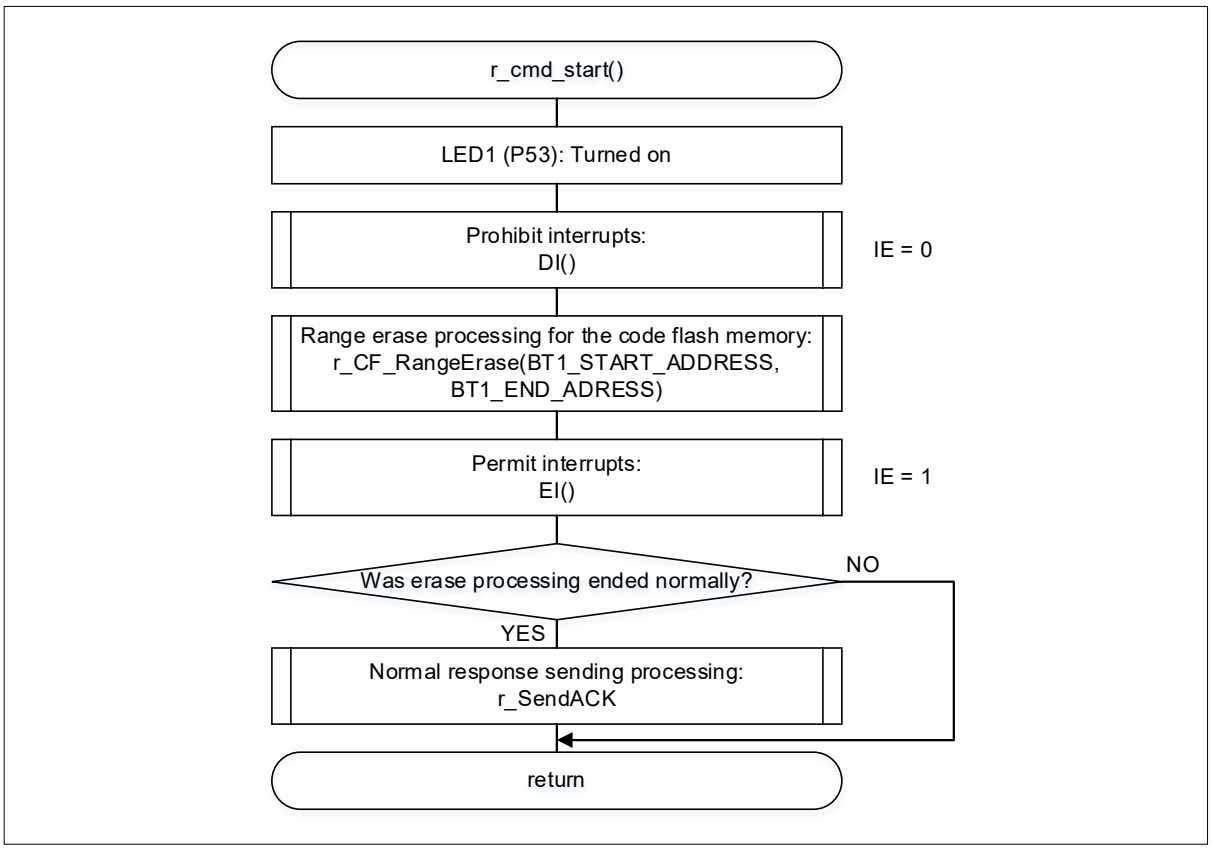
Figure 4-3 Initialization Processing for RFD RL78 Type01



4.10.3 START Command Processing

Figure 4-4 shows the flowchart for START command processing.

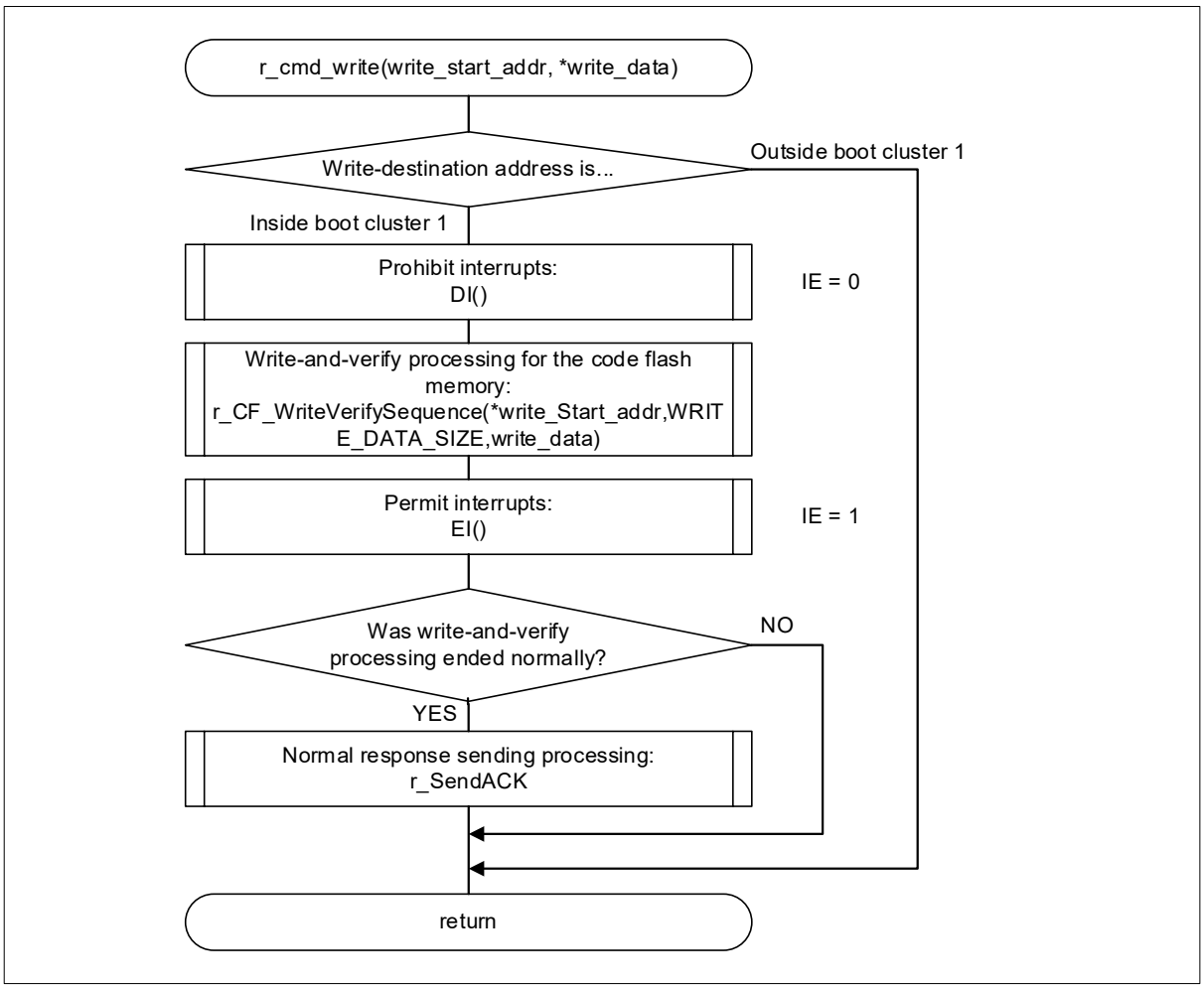
Figure 4-4 START Command Processing



4.10.4 WRITE Command Processing

Figure 4-5 shows the flowchart for WRITE command processing.

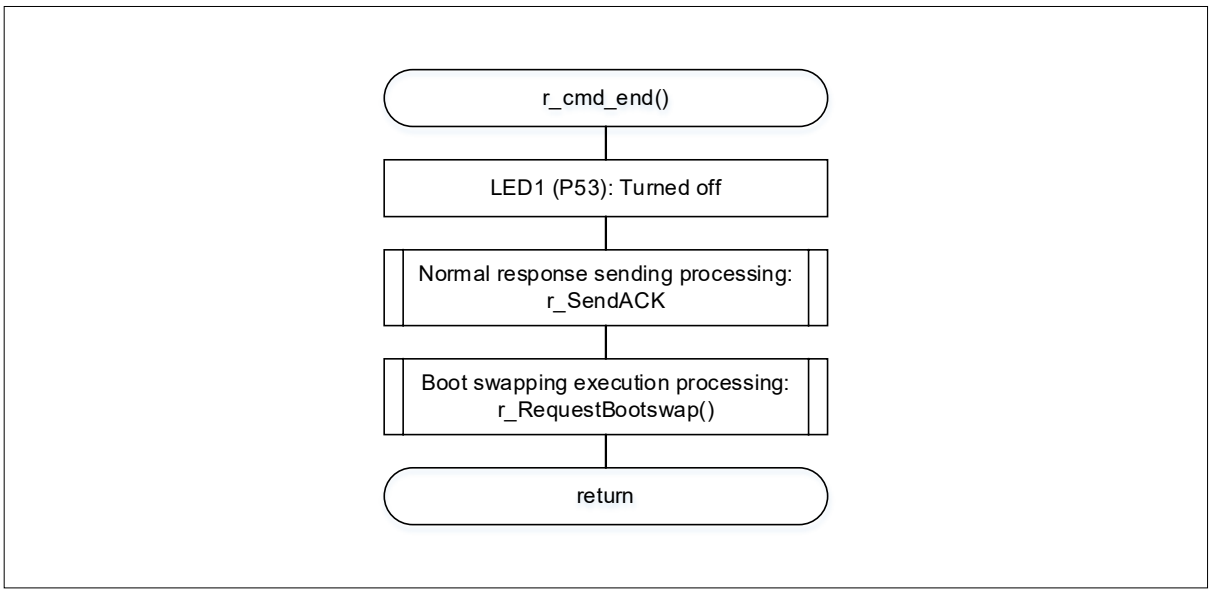
Figure 4-5 WRITE Command Processing



4.10.5 END Command Processing

Figure 4-6 shows the flowchart for END command processing.

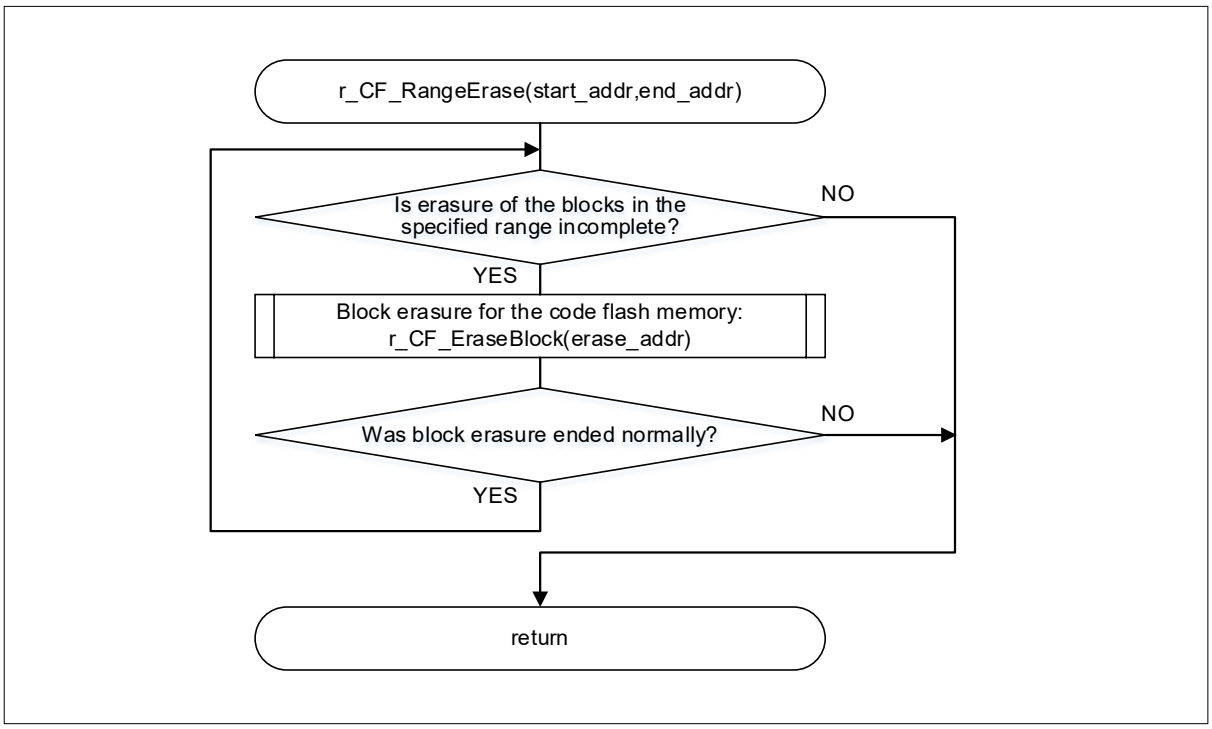
Figure 4-6 END Command Processing



4.10.6 Range Erase Processing for the Code Flash Memory

Figure 4-7 shows the flowchart for range erase processing for the code flash memory.

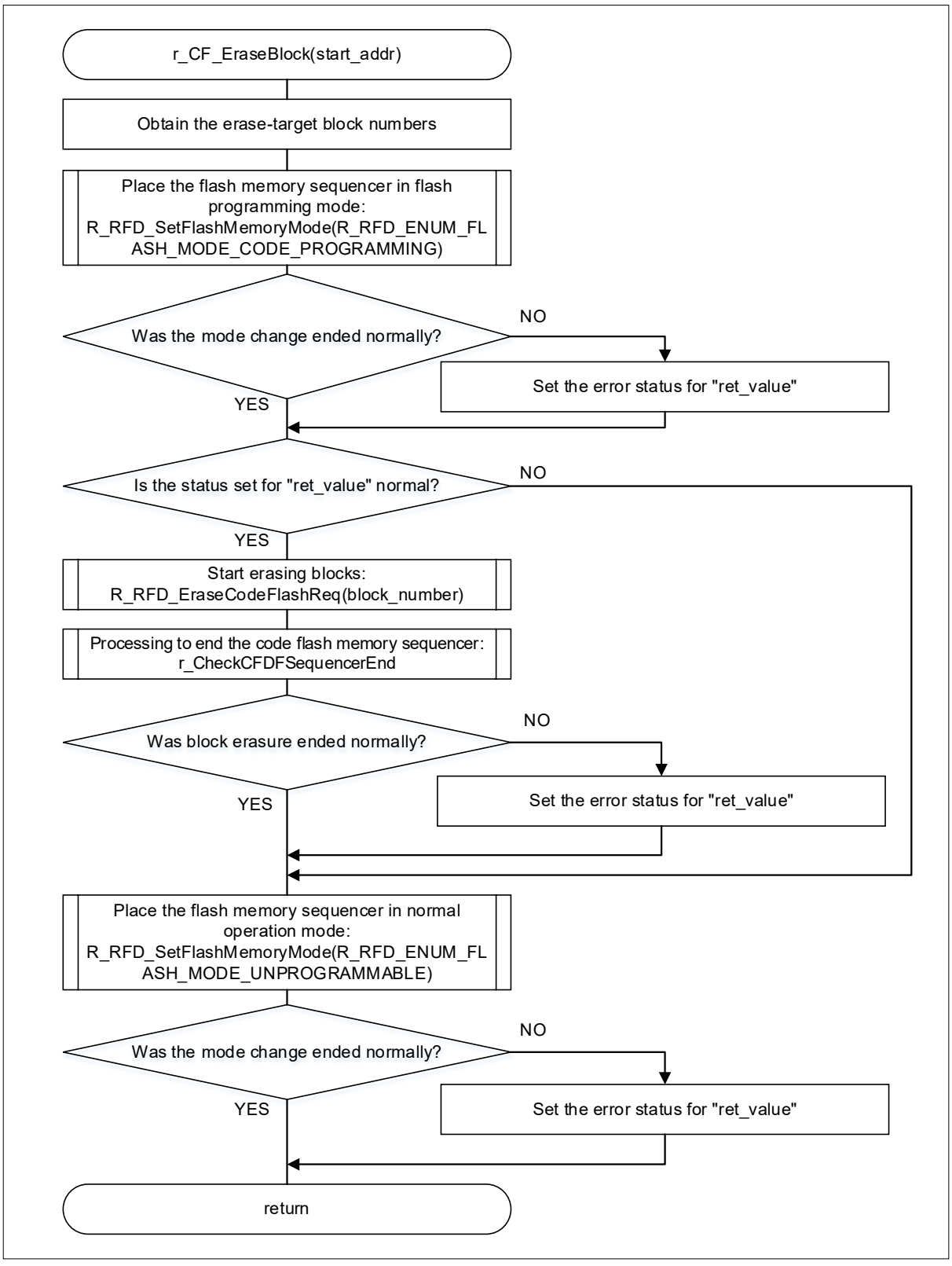
Figure 4-7 Range Erase Processing for the Code Flash Memory



4.10.7 Block Erase Processing for the Code Flash Memory

Figure 4-8 shows the flowchart for block erase processing for the code flash memory.

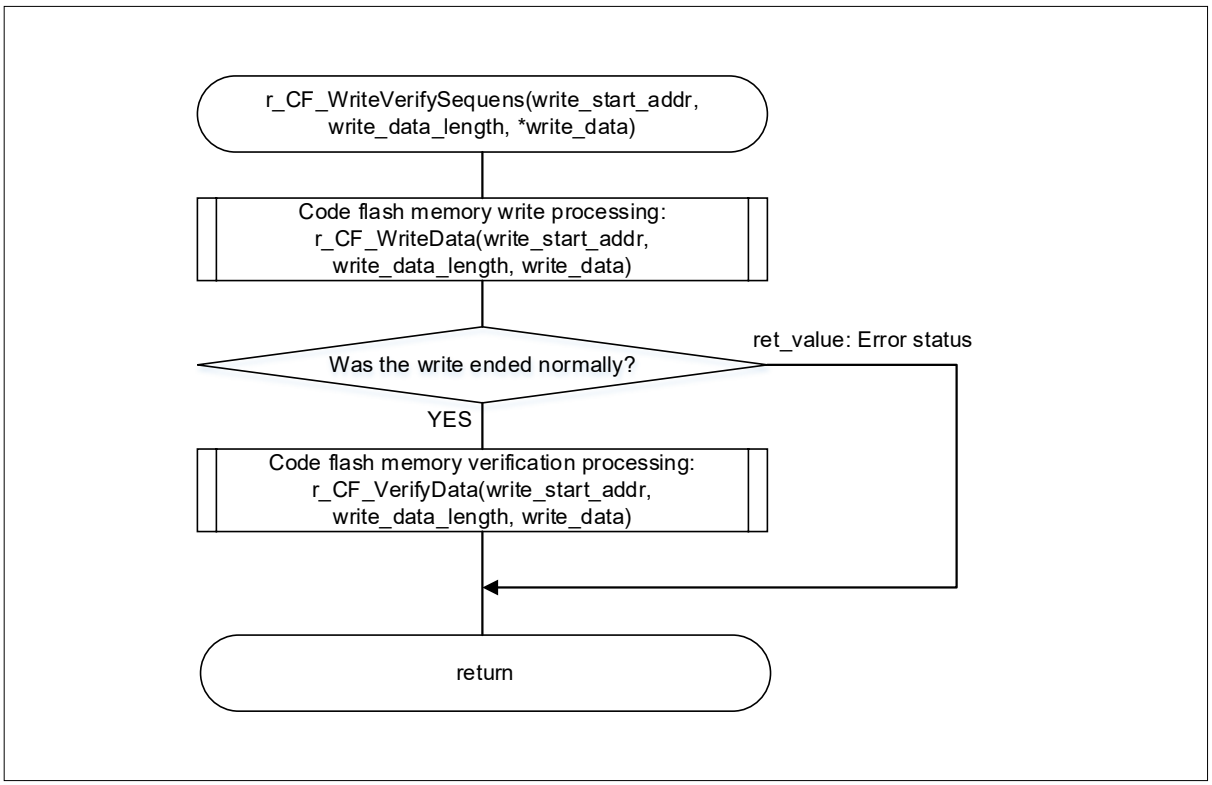
Figure 4-8 Block Erase Processing for the Code Flash Memory



4.10.8 Write-and-verify Processing for the Code Flash Memory

Figure 4-9 shows the flowchart for write-and-verify processing for the code flash memory.

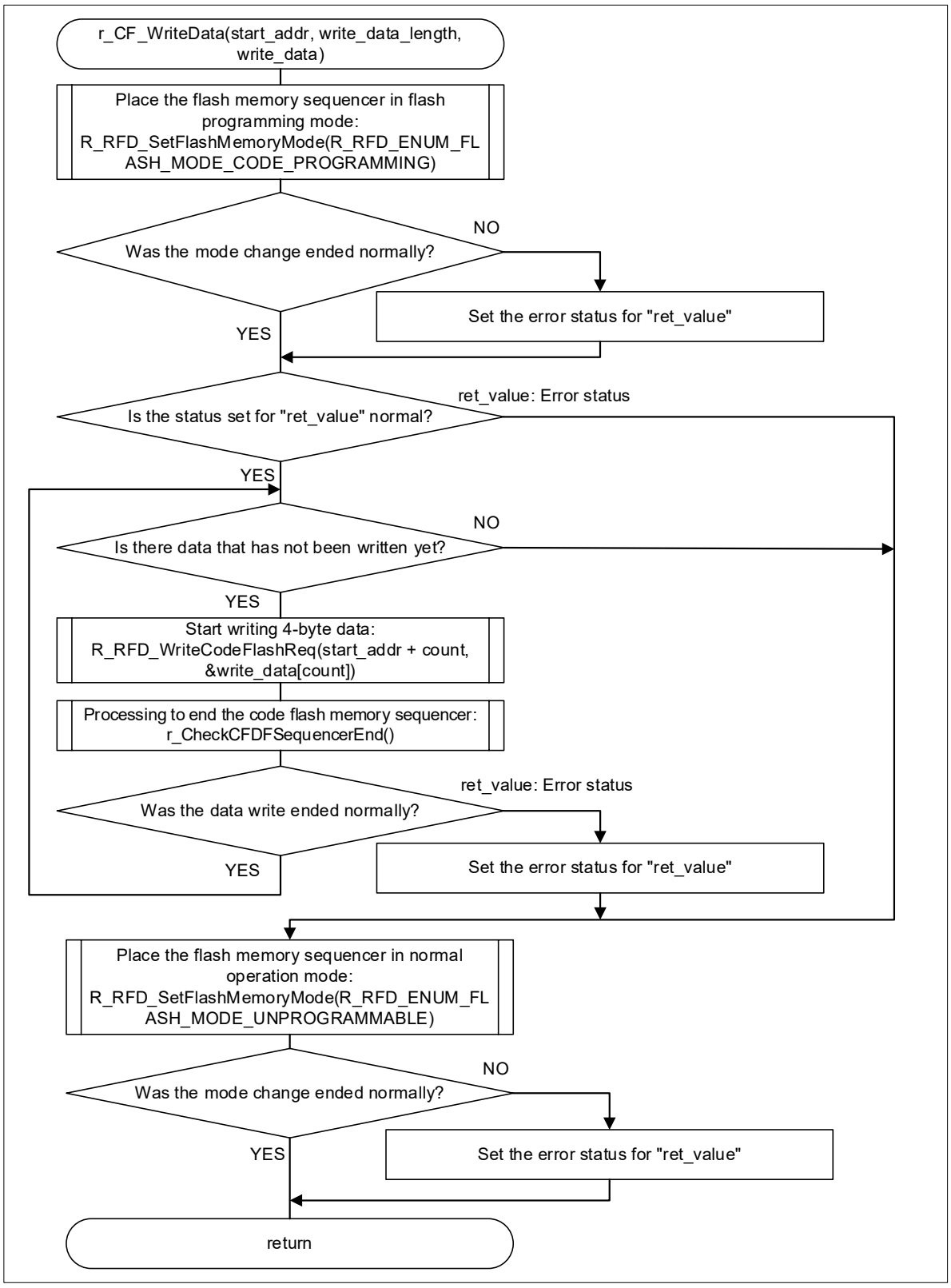
Figure 4-9 Write-and-verify Processing for the Code Flash Memory



4.10.9 Write Processing for the Code Flash Memory

Figure 4-10 shows the flowchart for write processing for the code flash memory.

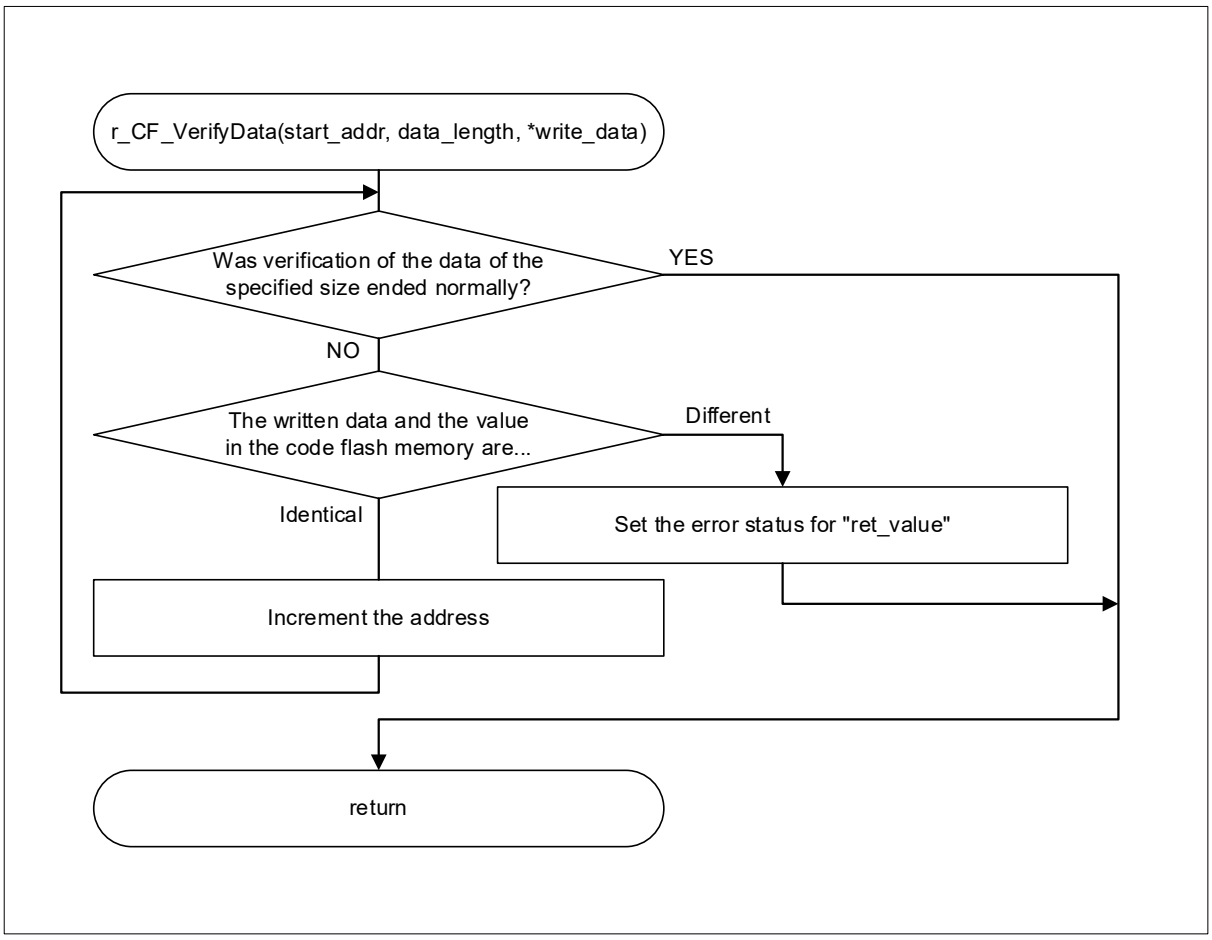
Figure 4-10 Write processing for the Code Flash Memory



4.10.10 Verify Processing for the Code Flash Memory

Figure 4-11 shows the flowchart for verify processing for the code flash memory.

Figure 4-11 Verify Processing for the Code Flash Memory



4.10.11 Sequence End Processing for the Code Flash Memory

Figure 4-12 to Figure 4-13 shows the flowchart for sequence end processing for the code flash memory.

Figure 4-12 Sequence End Processing for the Code Flash Memory (1/2)

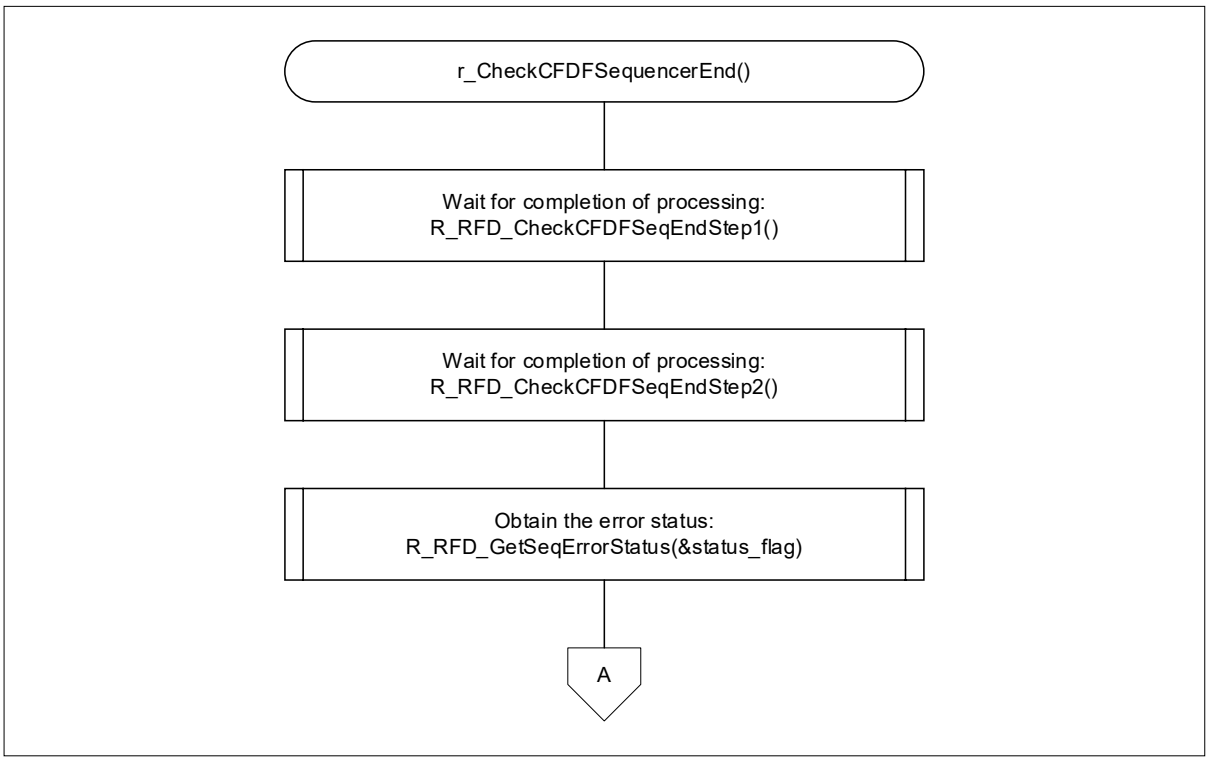
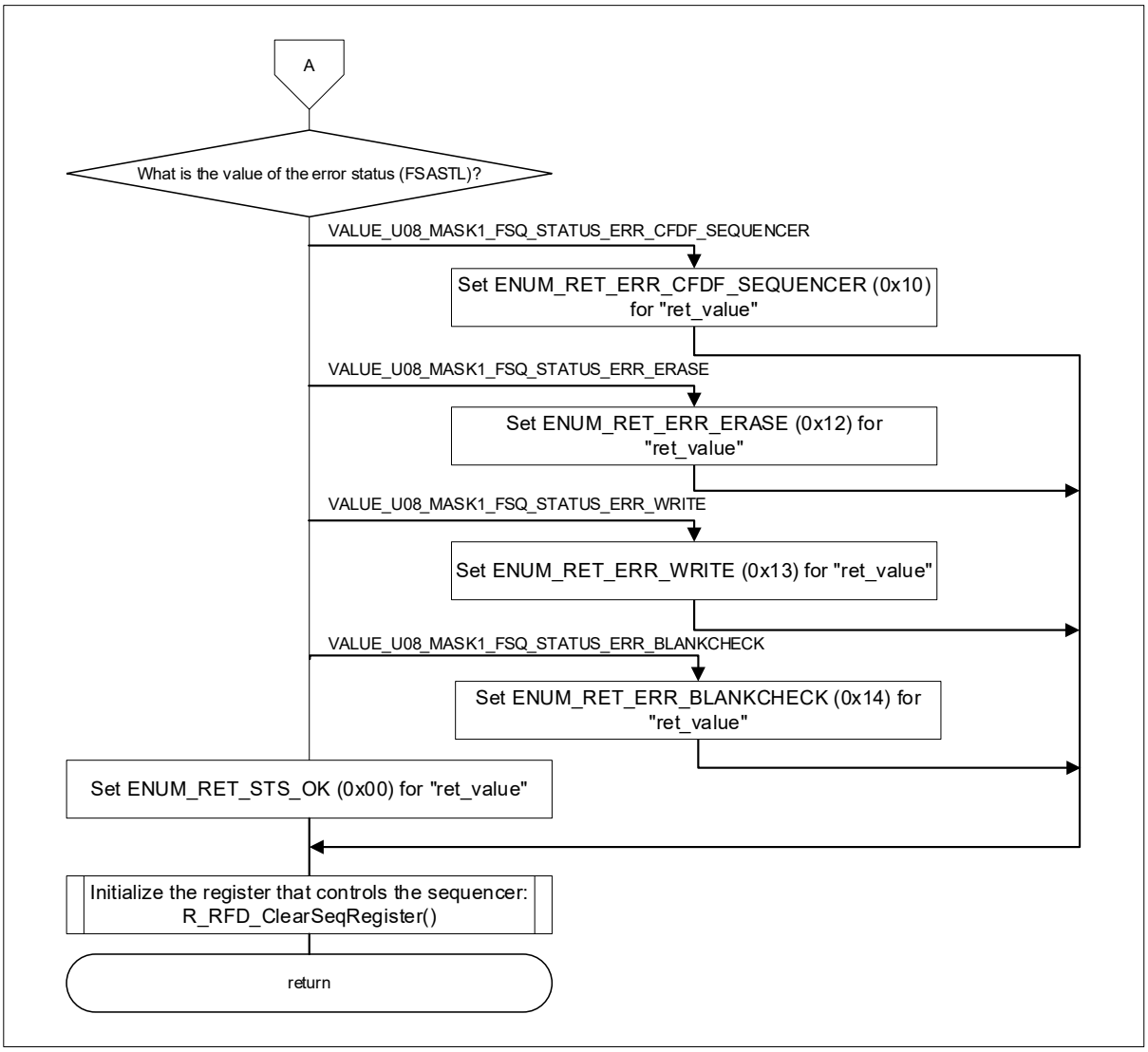


Figure 4-13 Sequence End Processing for the Code Flash Memory (2/2)



4.10.12 Sequence End Processing for the Extra Area

Figure 4-14 to Figure 4-15 shows the flowchart for sequence end processing for the extra area.

Figure 4-14 Sequence End Processing for the Extra Area (1/2)

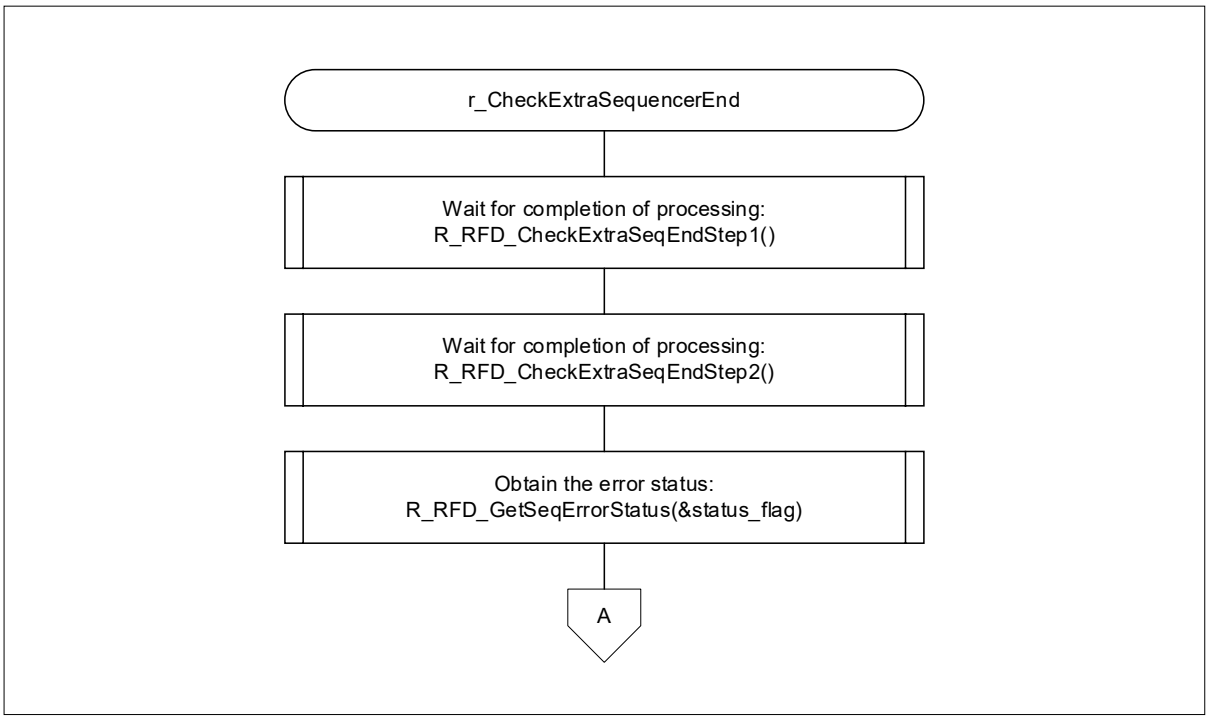
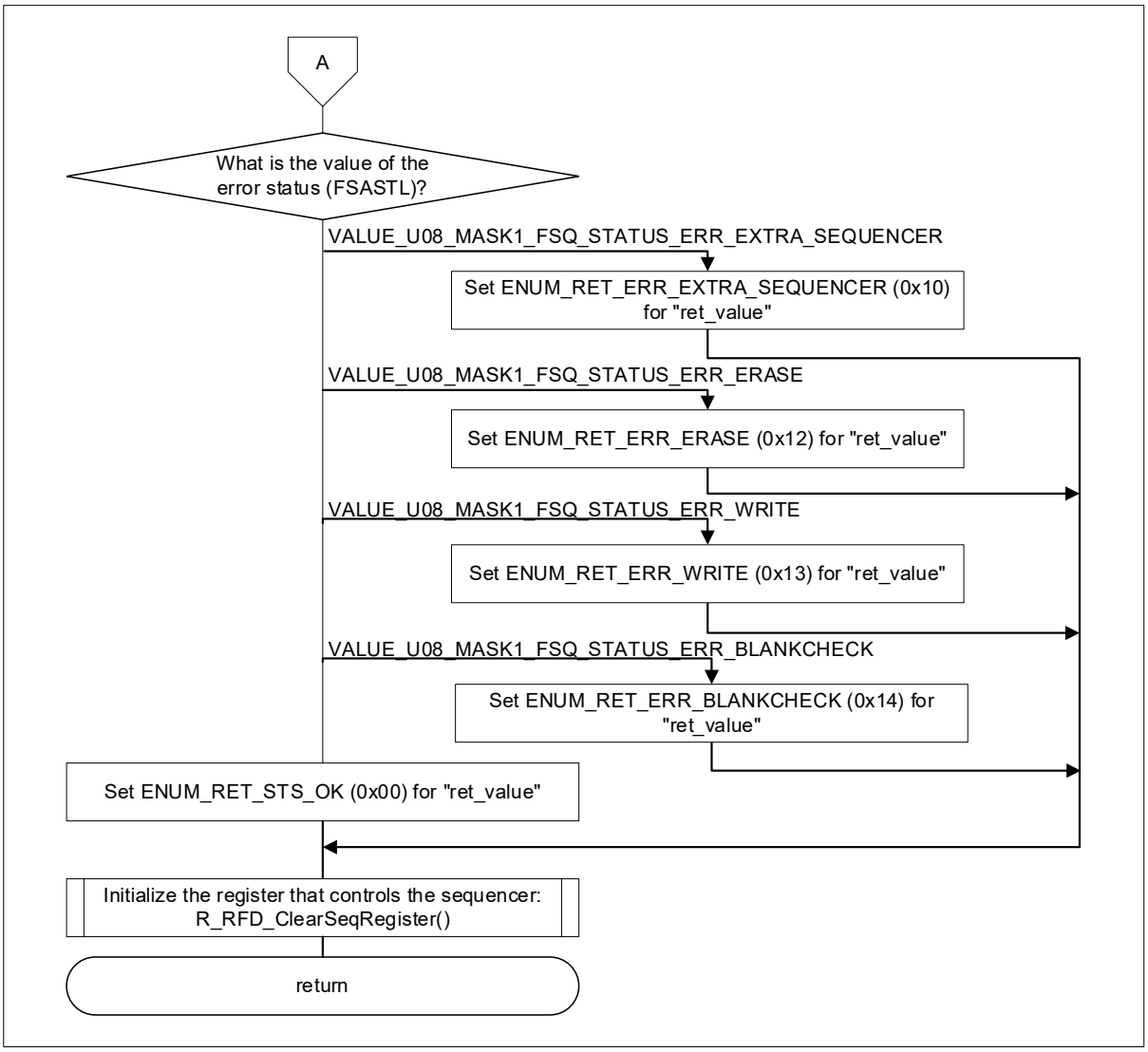


Figure 4-15 Sequence End Processing for the Extra Area (2/2)



4.10.13 Boot Swapping Execution Processing

Figure 4-16 to Figure 4-17 shows the flowchart for boot swapping execution processing.

Figure 4-16 Boot Swapping Execution Processing (1/2)

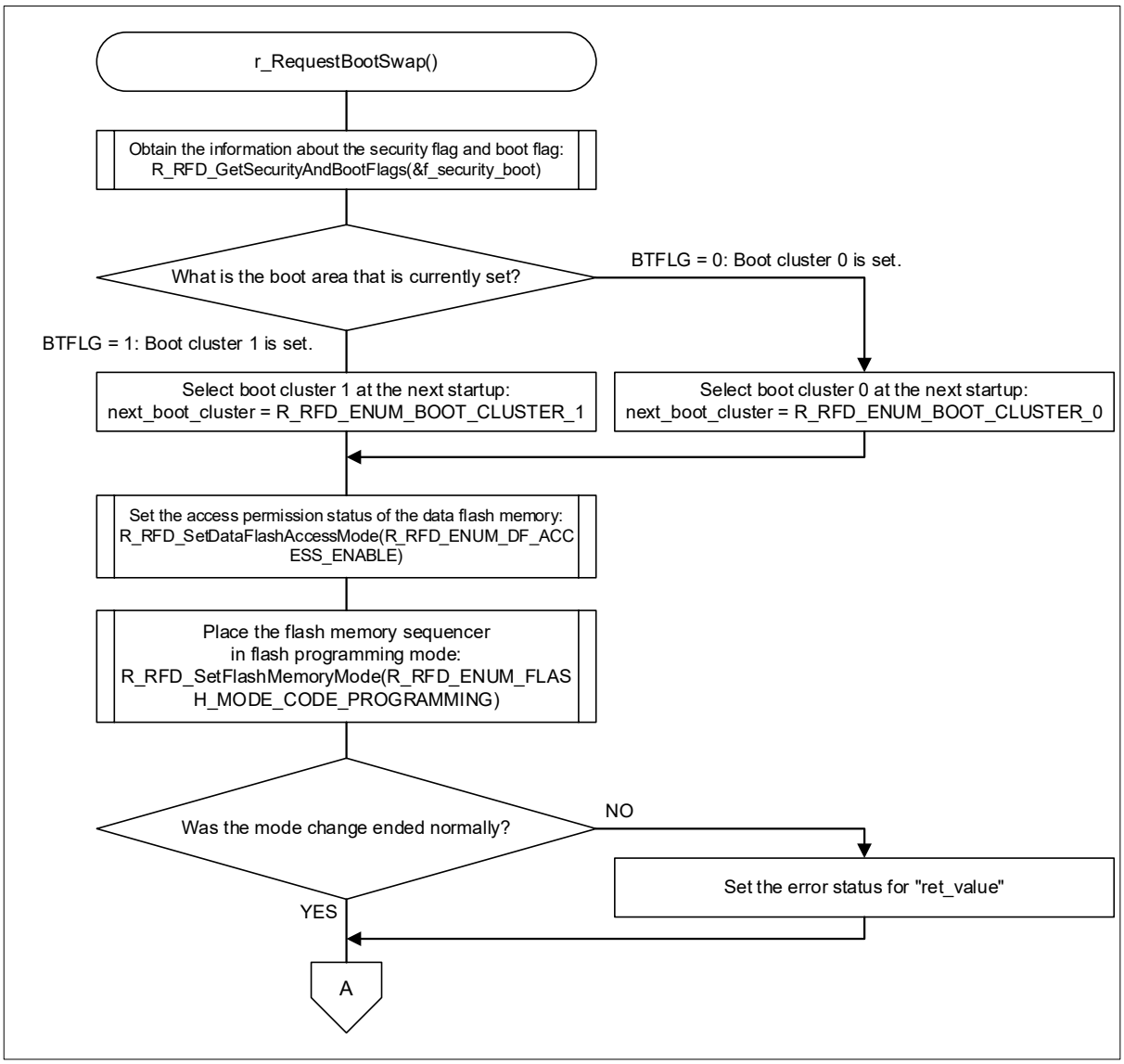
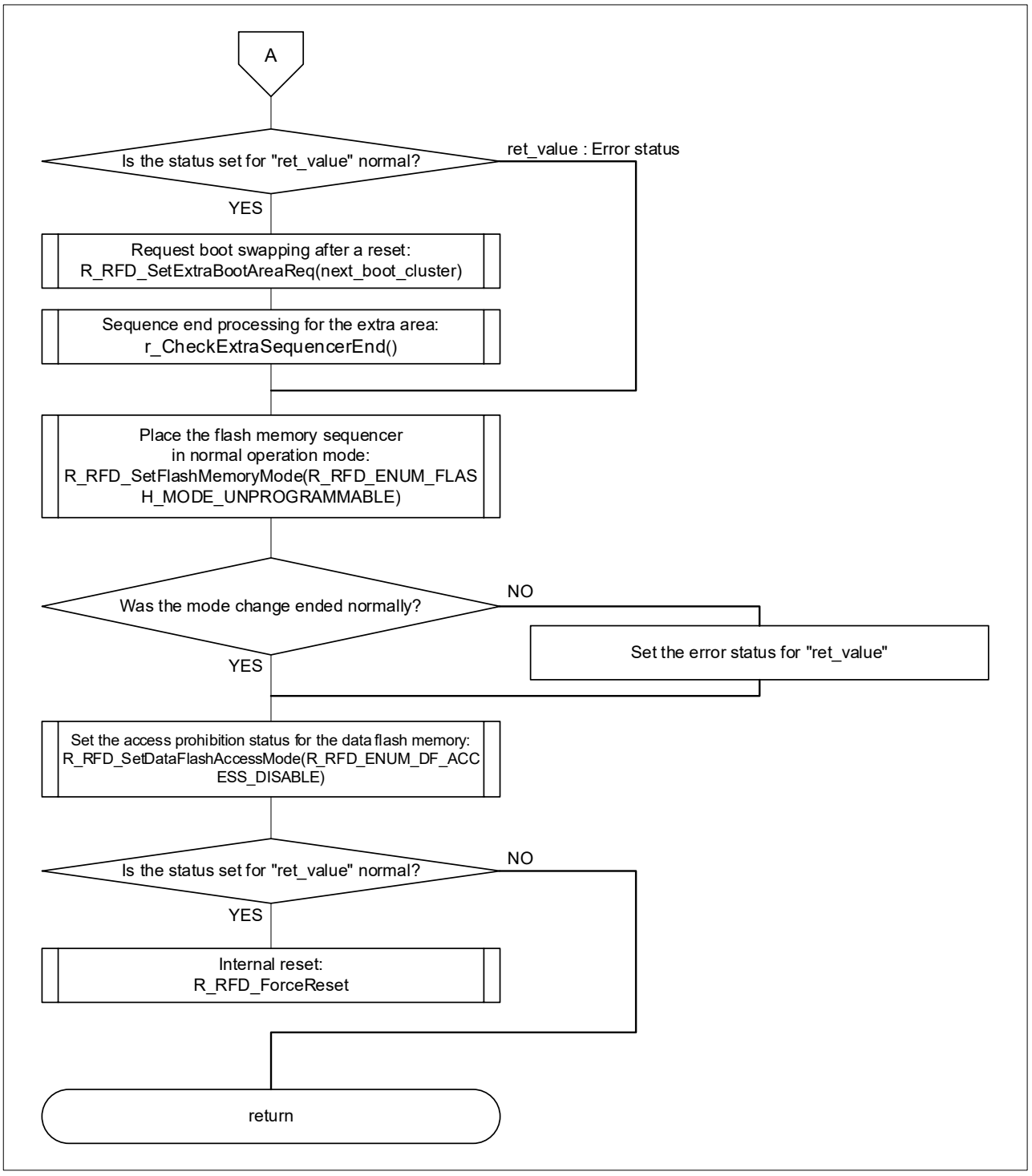


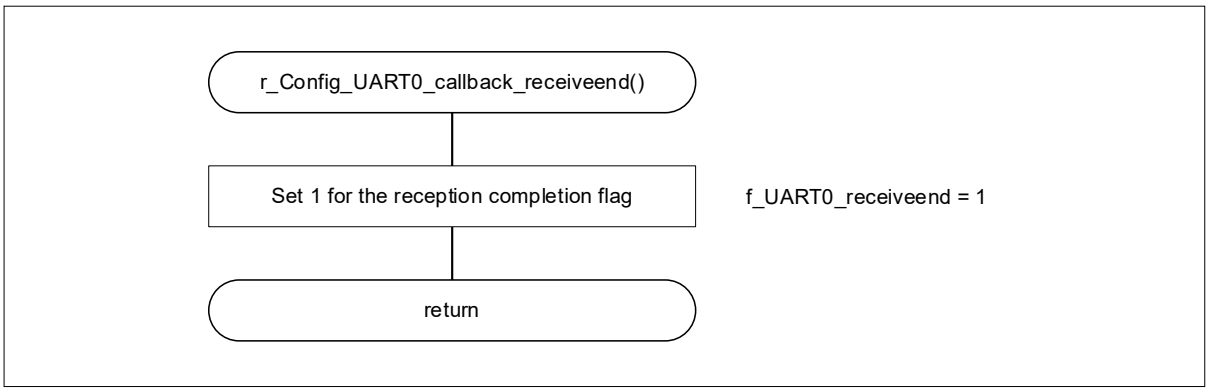
Figure 4-17 Boot Swapping Execution Processing (2/2)



4.10.14 Callback Processing at a Reception Completion Interrupt for UART0

Figure 4-18 shows the flowchart for callback processing at a reception completion interrupt for UART0.

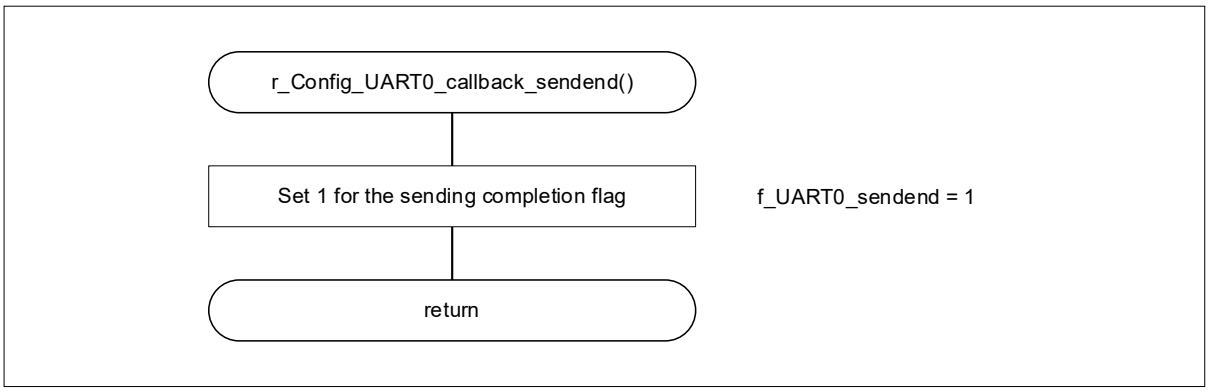
Figure 4-18 Callback Processing at a Reception Completion Interrupt for UART0



4.10.15 Callback Processing at a Sending Completion Interrupt for UART0

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

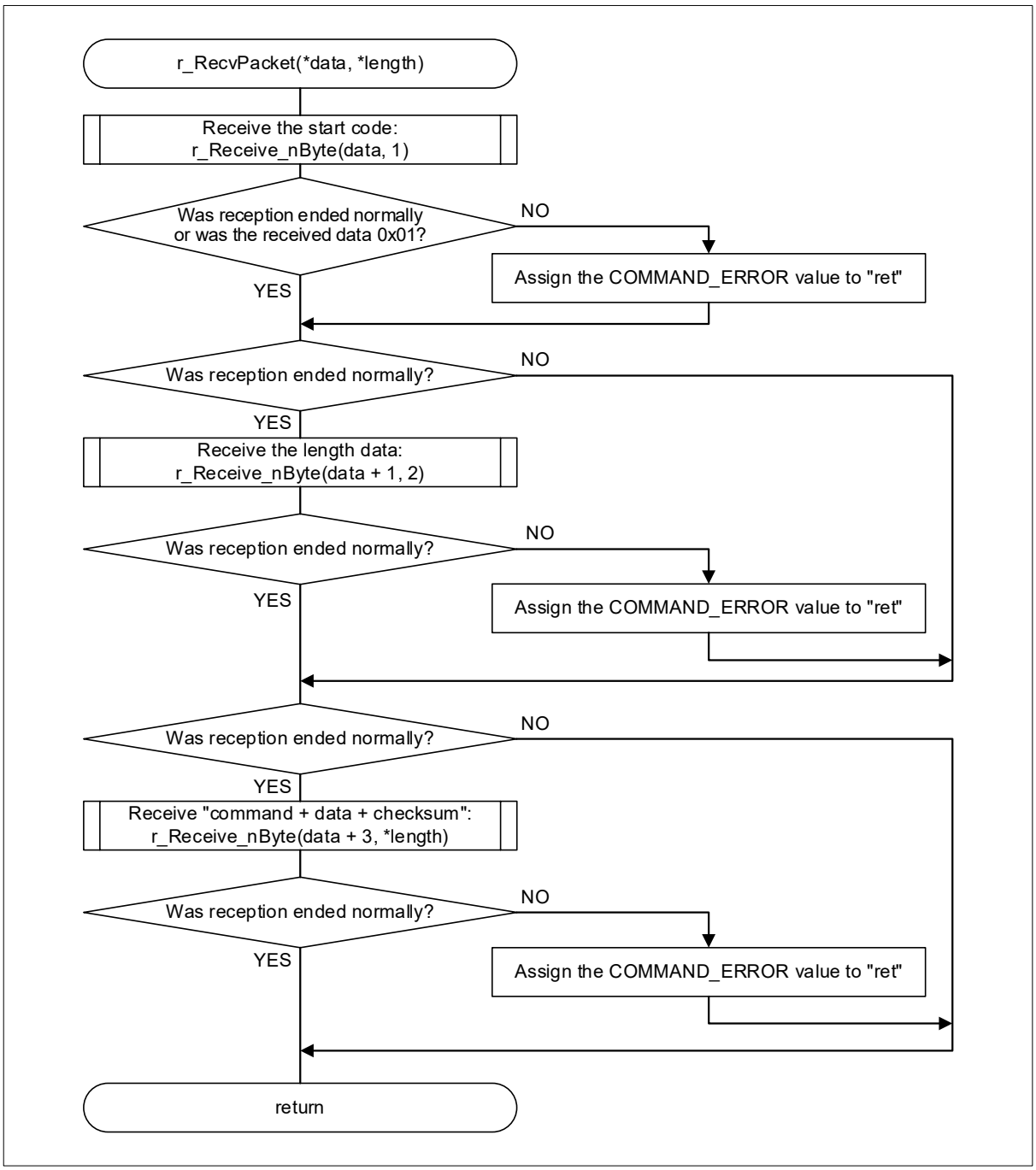
Figure 4-19 Callback Processing at a Sending Completion Interrupt for UART0



4.10.16 Command Reception Processing by UART0

Figure 4-20 shows the flowchart for command reception processing by UART0.

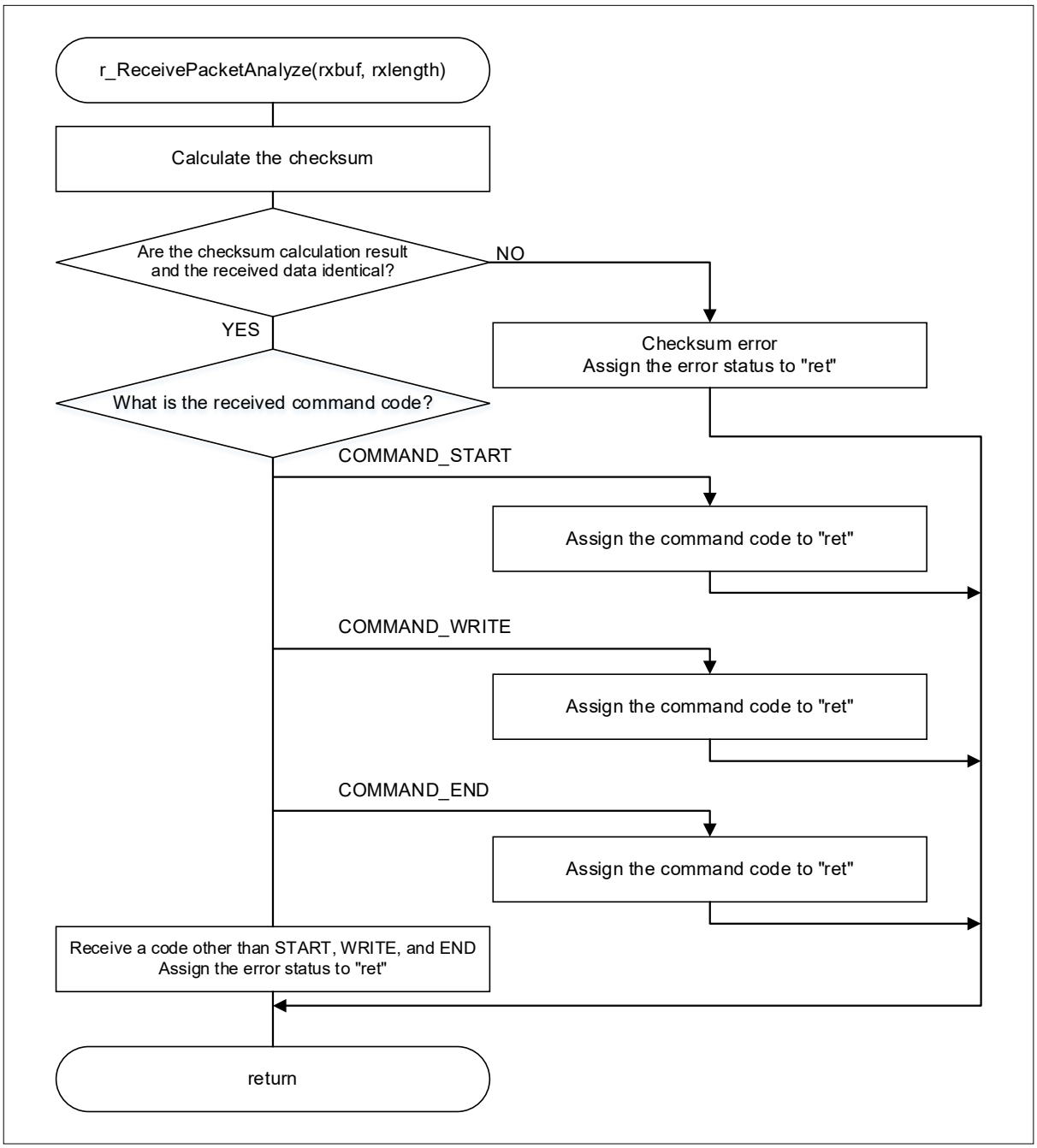
Figure 4-20 Command Reception Processing by UART0



4.10.17 Command Analysis Processing by UART0

Figure 4-21 shows the flowchart for command analysis processing by UART0.

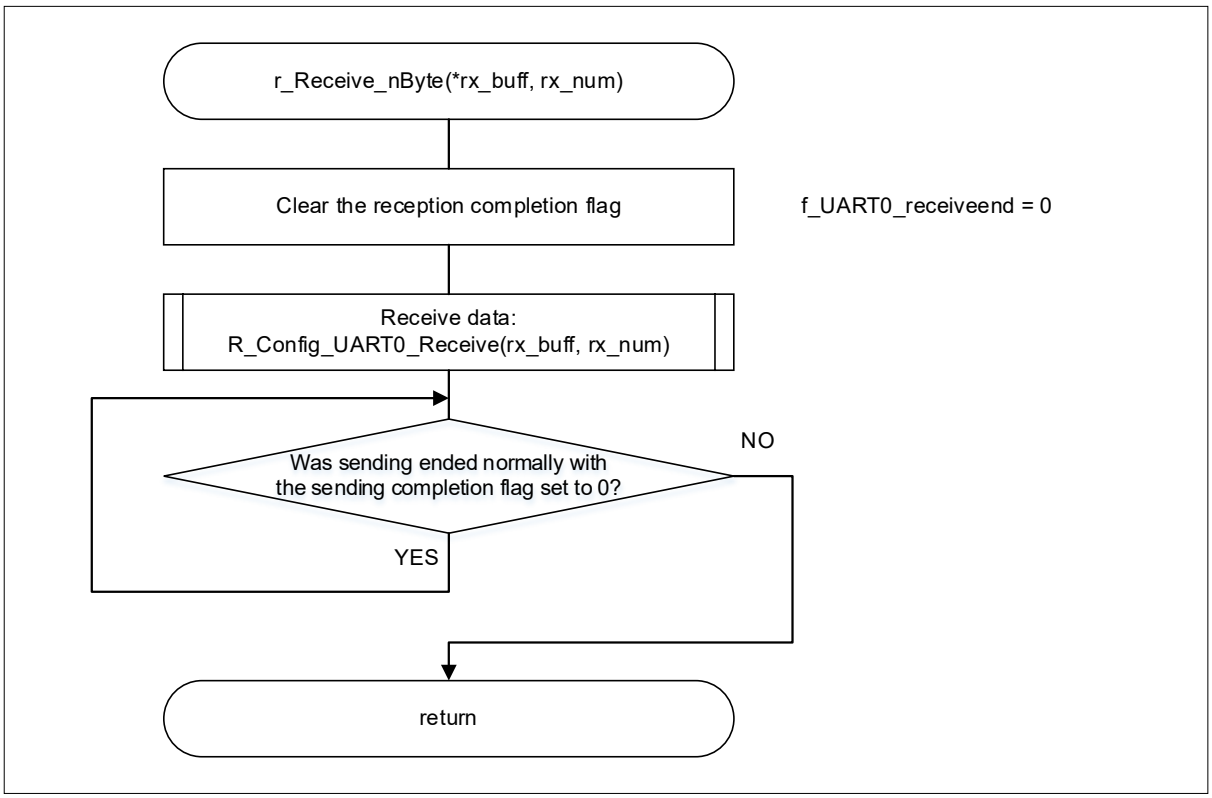
Figure 4-21 Command Analysis Processing by UART0



4.10.18 Data Reception Processing by UART0

Figure 4-22 shows the flowchart for data reception processing by UART0.

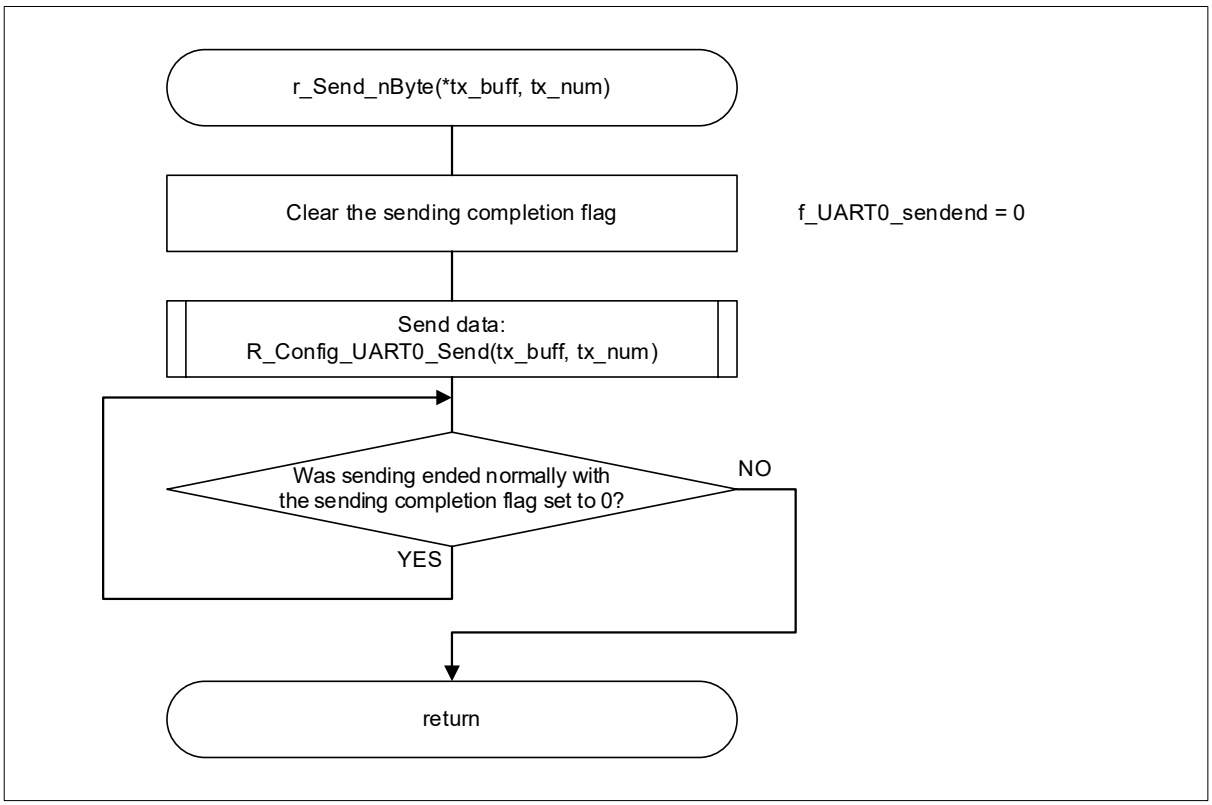
Figure 4-22 Data Reception Processing by UART0



4.10.19 Data Sending Processing by UART0

Figure 4-23 shows the flowchart for data sending processing by UART0.

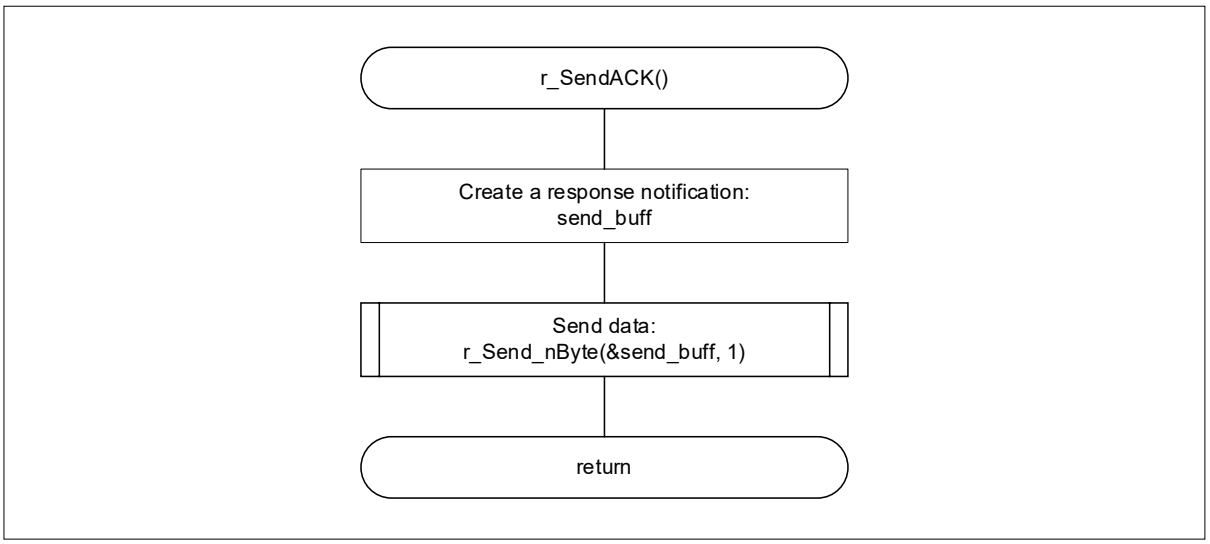
Figure 4-23 Data Sending Processing by UART0



4.10.20 Normal Response Sending Processing by UART0

Figure 4-23 shows the flowchart for normal response sending processing by UART0.

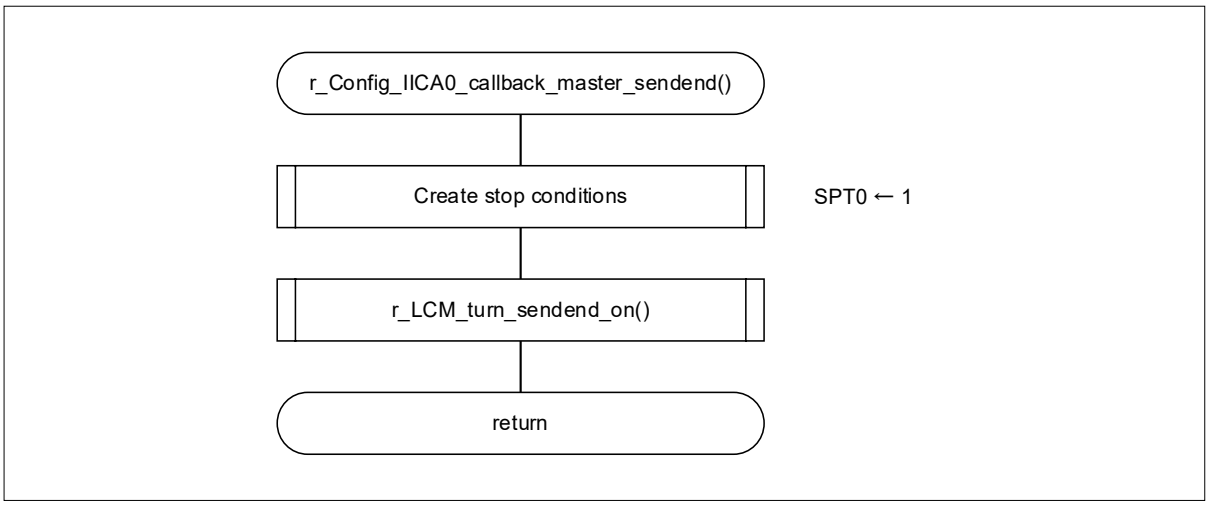
Figure 4-24 Normal Response Sending Processing by UART0



4.10.21 Callback Processing at a Sending Completion Interrupt for IICA0

Figure 4-25 shows the flowchart for callback processing at a sending completion interrupt for IICA0.

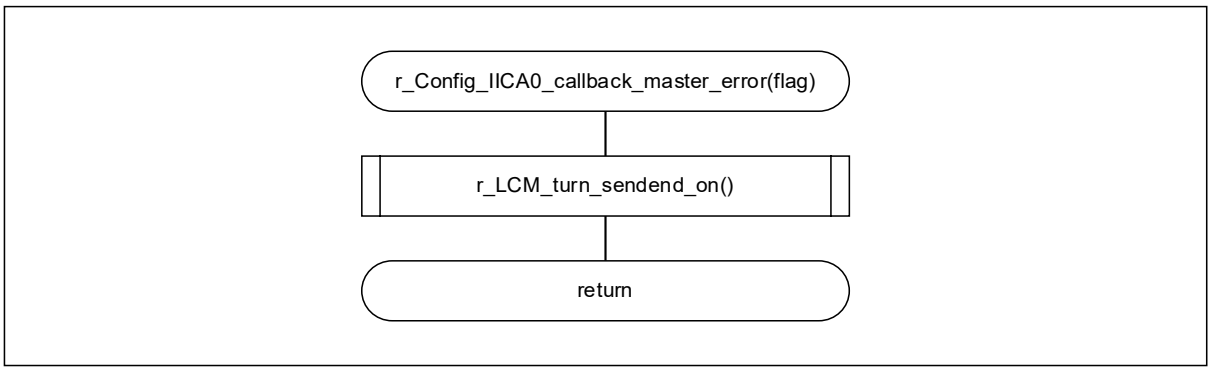
Figure 4-25 Callback Processing at a Sending Completion Interrupt for IICA0



4.10.22 Callback Processing at a Sending Error Interrupt for IICA0

Figure 4-26 shows the flowchart for callback processing at a sending error interrupt for IICA0.

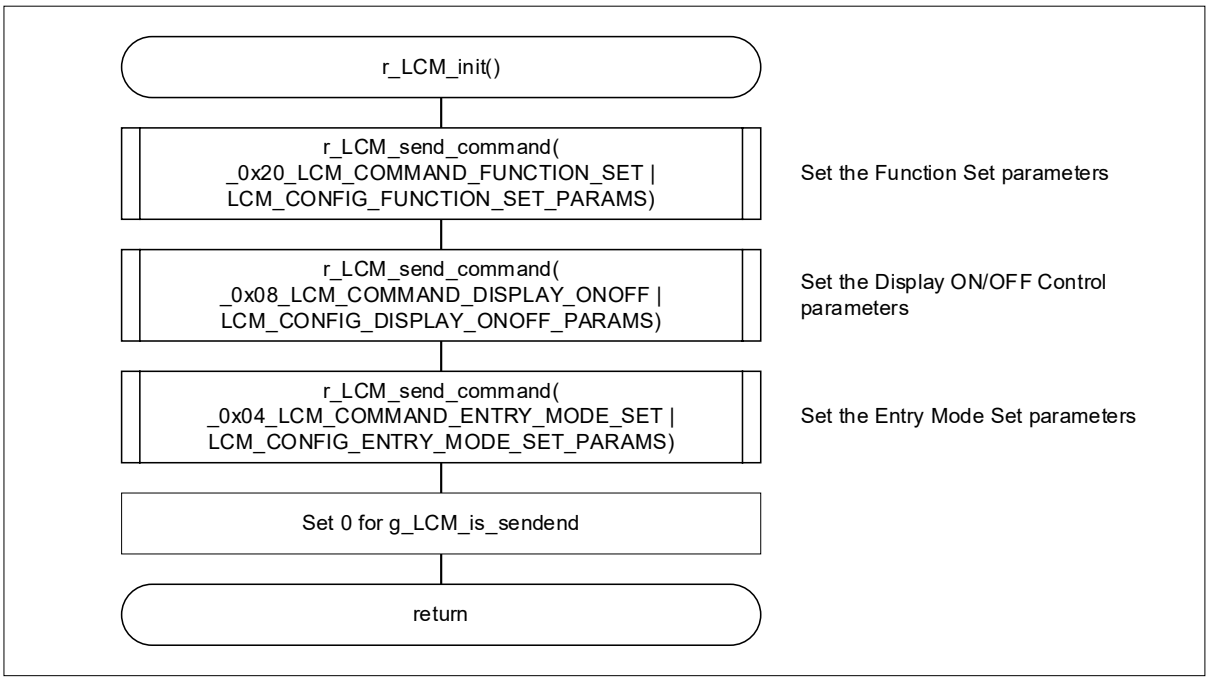
Figure 4-26 Callback Processing at a Sending Error Interrupt for IICA0



4.10.23 Processing to Initialize the LCD Module

Figure 4-27 shows the flowchart for processing to initialize the LCD module.

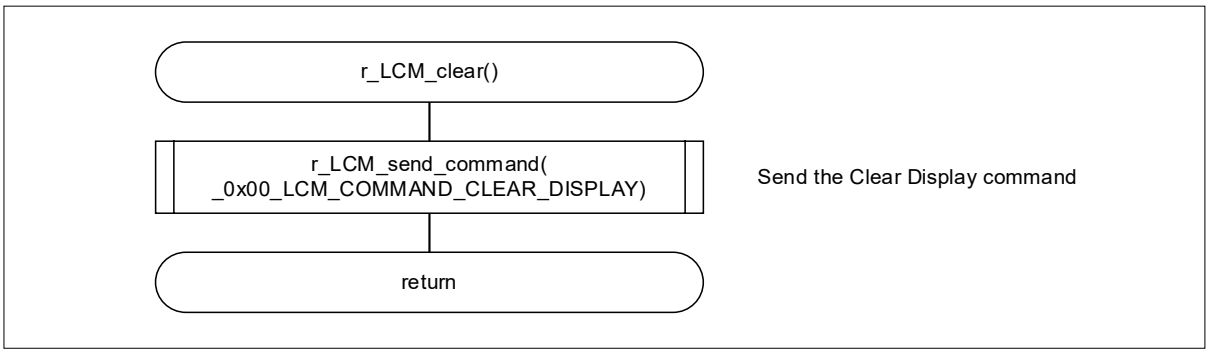
Figure 4-27 Processing to Initialize the LCD Module



4.10.24 Processing to Clear Display for the LCD Module

Figure 4-28 shows the flowchart for processing to clear display for the LCD module.

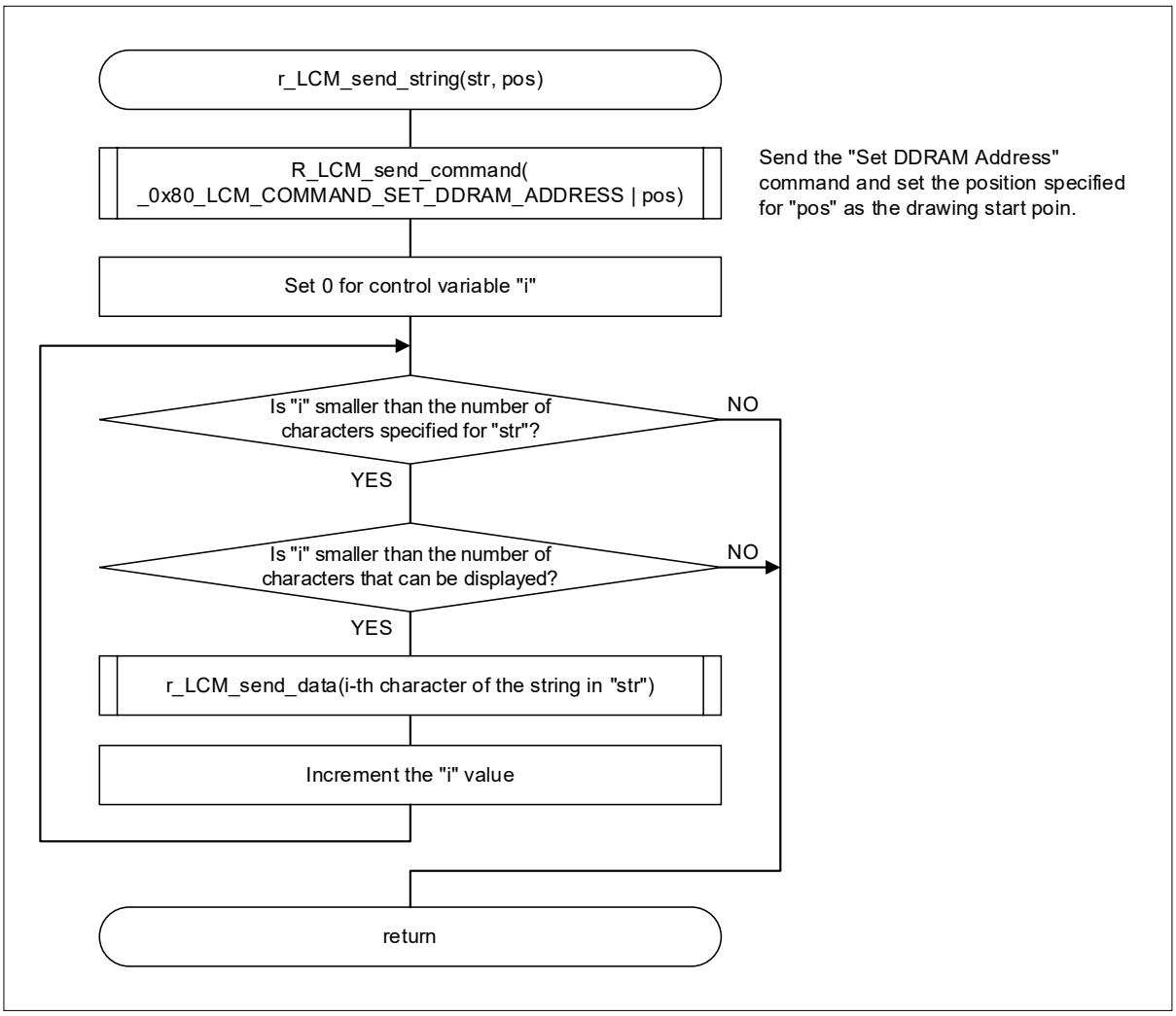
Figure 4-28 Processing to Clear Display for the LCD Module



4.10.25 Processing to Send Strings to the LCD Module

Figure 4-29 shows the flowchart for processing to send strings to the LCD module.

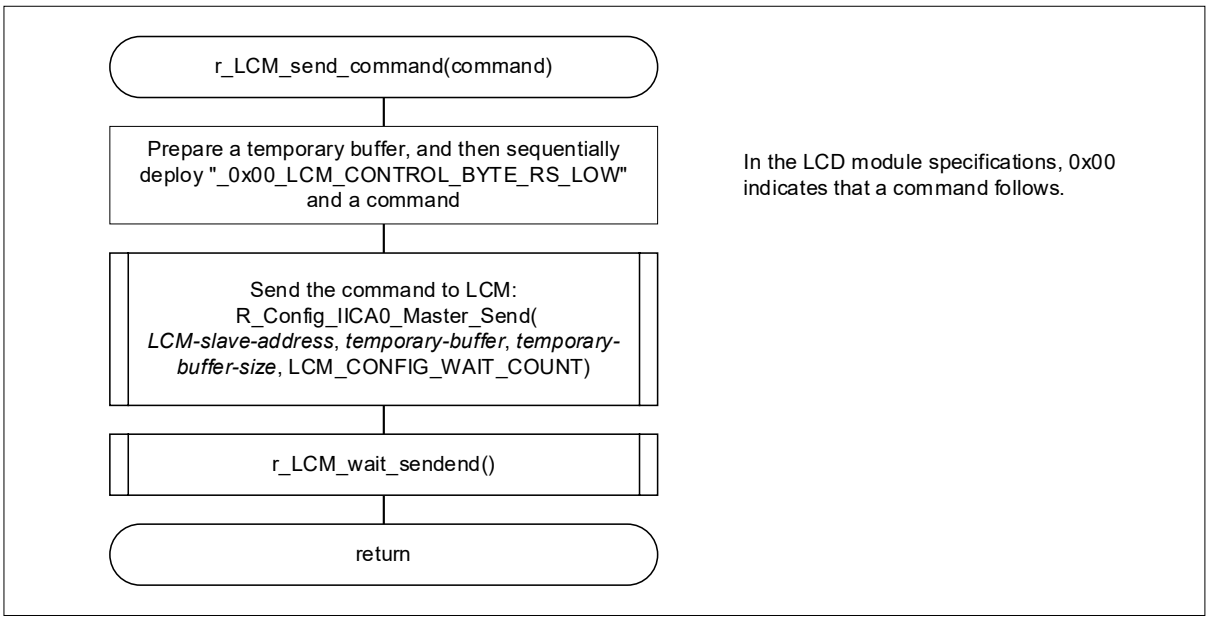
Figure 4-29 Processing to Send Strings to the LCD Module



4.10.26 Command Sending Processing for the LCD Module

Figure 4-30 shows the flowchart for command sending processing for the LCD module.

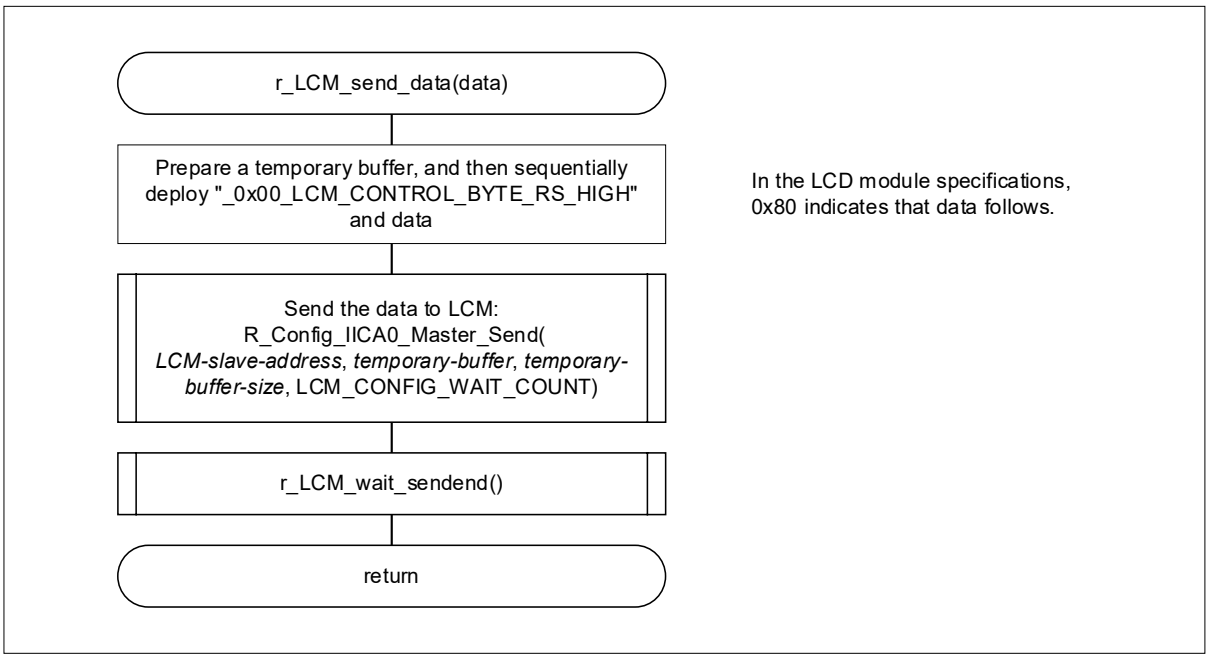
Figure 4-30 Command Sending Processing for the LCD Module



4.10.27 Processing to Send Data to the LCD Module

Figure 4-31 shows the flowchart for processing to send data to the LCD module.

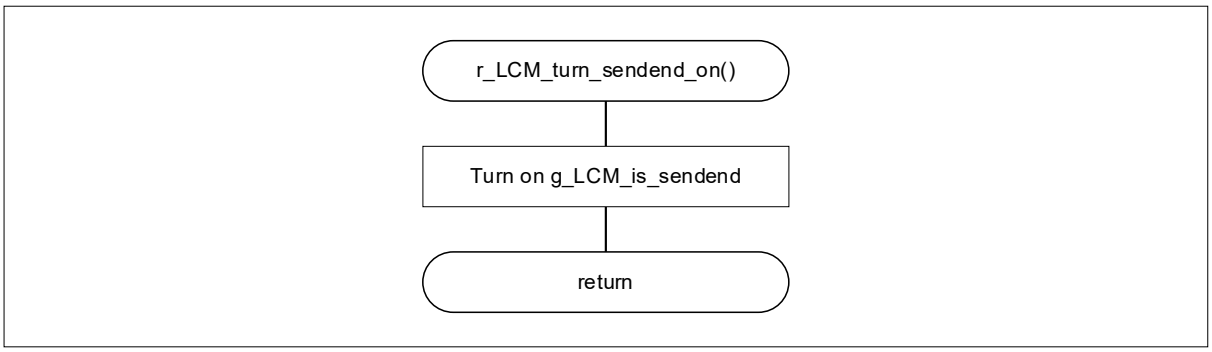
Figure 4-31 Processing to Send Data to the LCD Module



4.10.28 Communication End Flag Setting for the LCD Module

Figure 4-32 shows the flowchart for communication end flag setting for the LCD module.

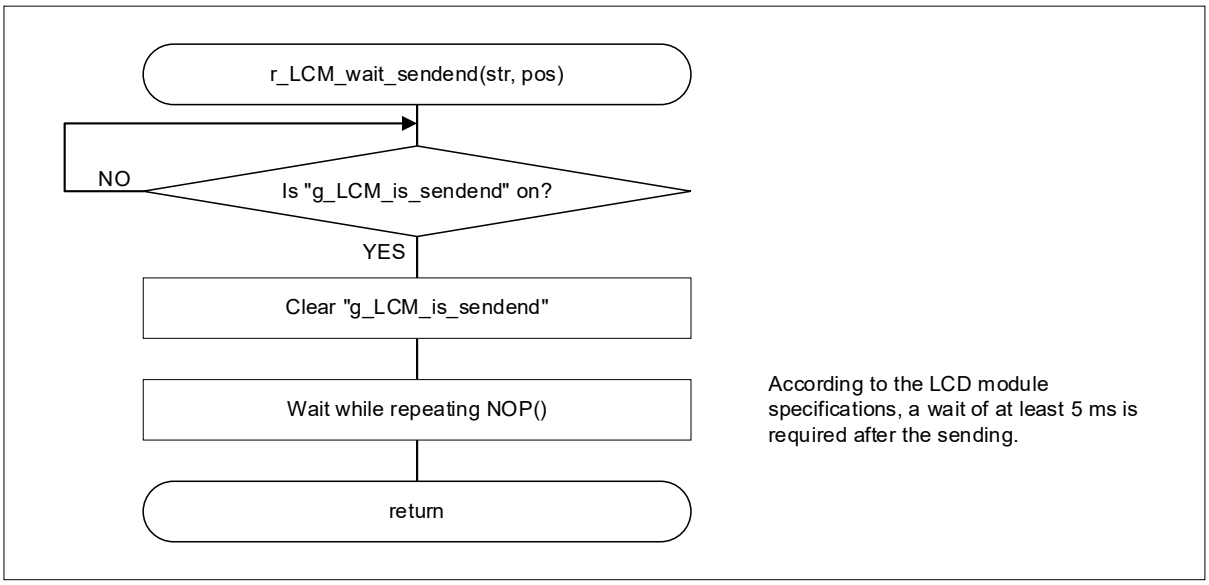
Figure 4-32 Communication End Flag Setting for the LCD Module



4.10.29 Communication End Wait Processing for the LCD Module

Figure 4-33 shows the flowchart for communication end wait processing for the LCD module.

Figure 4-33 Communication End Wait Processing for the LCD Module



5. Sample code

Sample code can be downloaded from the Renesas Electronics website.

6. Reference Documents

RL78/G23 User's Manual: Hardware (R01UH0896J)
RL78 family user's manual software (R01US0015J)
The latest versions can be downloaded from the Renesas Electronics website.

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

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jun. 18. 21	—	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.

Notice

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

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.5.0-1 October 2020)

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

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.