APPLICATION NOTE

RX63N Group, RX631 Group

Flash Bootloader with the USB Peripheral CDC

R01AN1710EJ0100
Rev. 1.00
Sep. 16, 2014

Abstract

This application note describes a method to reprogram the on-chip flash memory via USB memory (flash bootloader via USB) using the USB 2.0 host/function module as a function controller.

The features of the flash bootloader via USB are described below.

- Controls the target device by transmitting commands from a PC.
  With commands from a PC, erasing, programming, blank checking, or target program execution are performed for the flash memory on the target device.
- Programs in the Motorola S format can be programmed.
- Supports USB 2.0 full-speed transfer.
- Compliant with the Abstract Control Model in the Universal Serial Bus Class Definitions for Communications Devices.

Products

- RX63N Group 177-pin and 176-pin packages with a ROM size between 768 KB and 2 MB

- RX63N Group 145-pin and 144-pin packages with a ROM size between 768 KB and 2 MB

- RX63N Group 100-pin package with a ROM size between 768 KB and 2 MB

- RX631 Group 177-pin and 176-pin packages with a ROM size between 256 KB and 2 MB

- RX631 Group 145-pin and 144-pin packages with a ROM size between 256 KB and 2 MB

- RX631 Group 100-pin package with a ROM size between 256 KB and 2 MB

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

Contents

1. Specifications

With the flash bootloader via USB, commands are transmitted from the terminal application on the host PC to the MCU to reprogram the flash memory on the MCU.

Table 1.1 lists the Peripheral Functions and Their Applications and Figure 1.1 shows a Usage Example.

Table 1.1 Peripheral Functions and Their Applications

Peripheral Function	Application
ROM (flash memory for code storage)	Reprogramming the on-chip flash memory in ROM P/E mode.
USB 2.0 host/function module	Communication with the host PC.

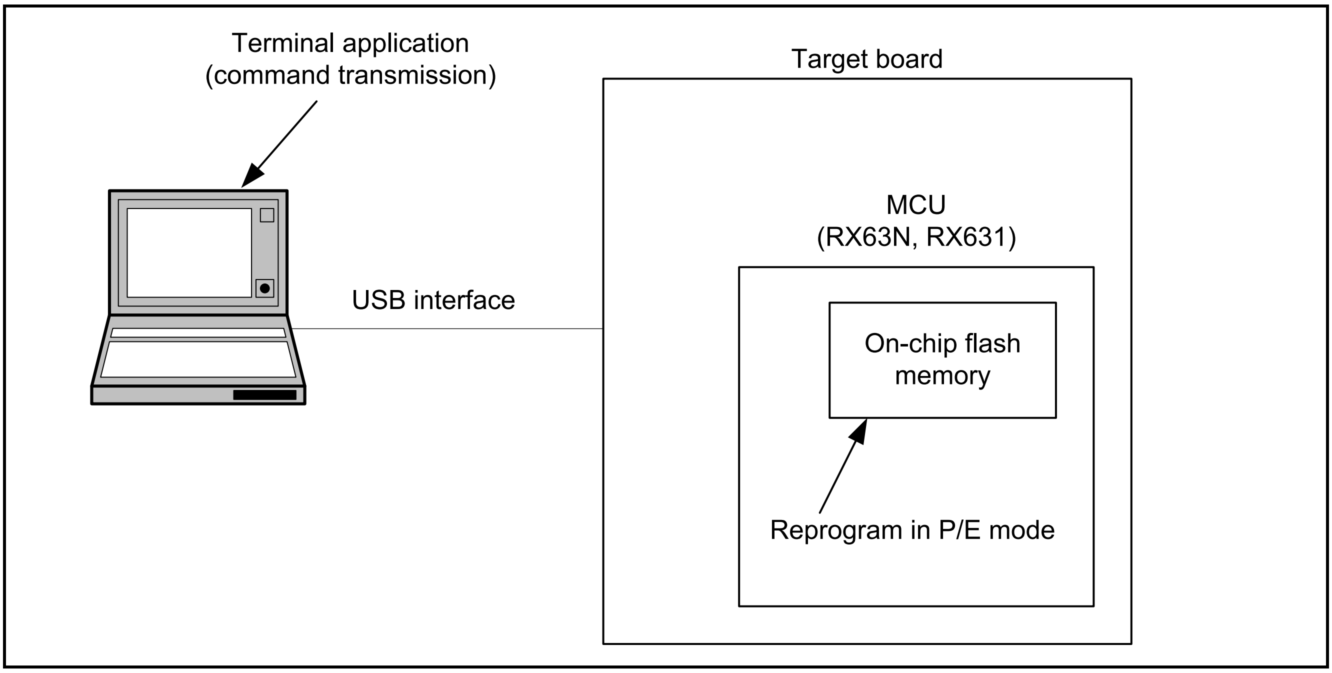

Figure 1.1 Usage Example

2. Operation Confirmation Conditions

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

Table 2.1 Operation Confirmation Conditions

Item	Contents
MCU used	R5F563NBDDFC (RX63N Group)
Operating frequencies	- Main clock: 12 MHz - PLL: 192 MHz (main clock divided by 1 and multiplied by 16) - System clock (ICLK): 96 MHz (PLL divided by 2) - FlashIF clock (FCLK): 48 MHz (PLL divided by 4) - External bus clock (BCLK): 24 MHz (PLL divided by 8) - Peripheral module clock B (PCLKB): 48 MHz (PLL divided by 4) - USB clock (UCLK): 48 MHz (PLL divided by 4)
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics Corporation High-performance Embedded Workshop Version 4.09.01
C compiler	Renesas Electronics Corporation C/C++ Compiler Package for RX Family V.1.02 Release 01
	Compile options --cpu=rx600 -bit_order=left -include="$(WORKSPDIR)\WorkSpace\ANSI" -include="$(WORKSPDIR)\WorkSpace\CDCFW\include" -include="$(WORKSPDIR)\WorkSpace\HwResourceForUSB\inc" -include="$(WORKSPDIR)\WorkSpace\HwResourceForUSB\USBHW" -include="$(WORKSPDIR)\WorkSpace\HwResourceForUSB\USBHW\DEF" -include="$(WORKSPDIR)\WorkSpace\HwResourceForUSB\USBHW\REG" -include="$(WORKSPDIR)\WorkSpace\HwResourceForUSB\USRCFG" -include="$(WORKSPDIR)\WorkSpace\SmplMain\APL" -include="$(WORKSPDIR)\WorkSpace\USBSTDFW\include" -include="$(WORKSPDIR)\WorkSpace\FLASH" -include="$(WORKSPDIR)\WorkSpace\FLASH\src" -include="$(WORKSPDIR)\WorkSpace\r_bsp" -define=USB_FW_PP=USB_FW_NONOS_PP,USBC_DEBUGLCD_PP -output=obj="$(CONFIGDIR)\$(FILELEAF).obj" -debug -nostuff -nologo
iodefine.h version	Version 1.6A
Endian	Little endian
Operating mode	Single-chip mode
Processor mode	Supervisor mode
Sample code version	Version 1.00
Board used	Renesas Starter Kit+ for RX63N (product part no.: R0K50563NC000BE)

3. Reference Application Notes

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

- RX600 & RX200 Series Simple Flash API for RX Rev.2.40 (R01AN0544EU)
- Renesas USB MCU and USB ASSP USB Basic Host and Peripheral firmware Rev.2.10 (R01AN0512EJ)
- Renesas USB MCU and USB ASSP USB Peripheral Communications Device Class Driver (PCDC) Rev.2.10 (R01AN0273EJ)
- RX63N Group, RX631 Group Initial Setting Rev. 1.10 (R01AN1245EJ)

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

4. Hardware

4.1 Pins Used

Table 4.1 lists the Pins Used and Their Functions.

Table 4.1 Pins Used and Their Functions

Pin Name	I/O	Function
USB1_DP	I/O	D+ I/O pin of the USB on-chip transceiver Connected to the D+ pin of the USB bus.
USB1_DM	I/O	D- I/O pin of the USB on-chip transceiver Connected to the D- pin of the USB bus.
USB1_VBUS	Input	USB cable connection monitor pin Connected to VBUS of the USB bus. The state of the VBUS connection (connected/disconnected) can be detected.
USB1_DPUPE	Output	1.5-kΩ pull-up resistor control signal for USB D+ signal

Note: • For settings of the other pins used in the USB Peripheral Communication Device Class Driver (USB PCDC driver) such as switch or SCI, refer to the USB Peripheral Communications Device Class Driver (PCDC) application note.

5. Software

5.1 Operation Overview

The sample code receives command data from the host PC and performs the operation (command display, blank check, erase, program, or target program execution) according to the command. Commands from the host PC are transmitted by the terminal application. The sample code can reprogram only the specific part (target area) of the user area. Addresses FFFF0000h to FFFFFFFFh are used by the sample code itself and cannot be reprogrammed.

Figure 5.1 shows the Memory Allocation.

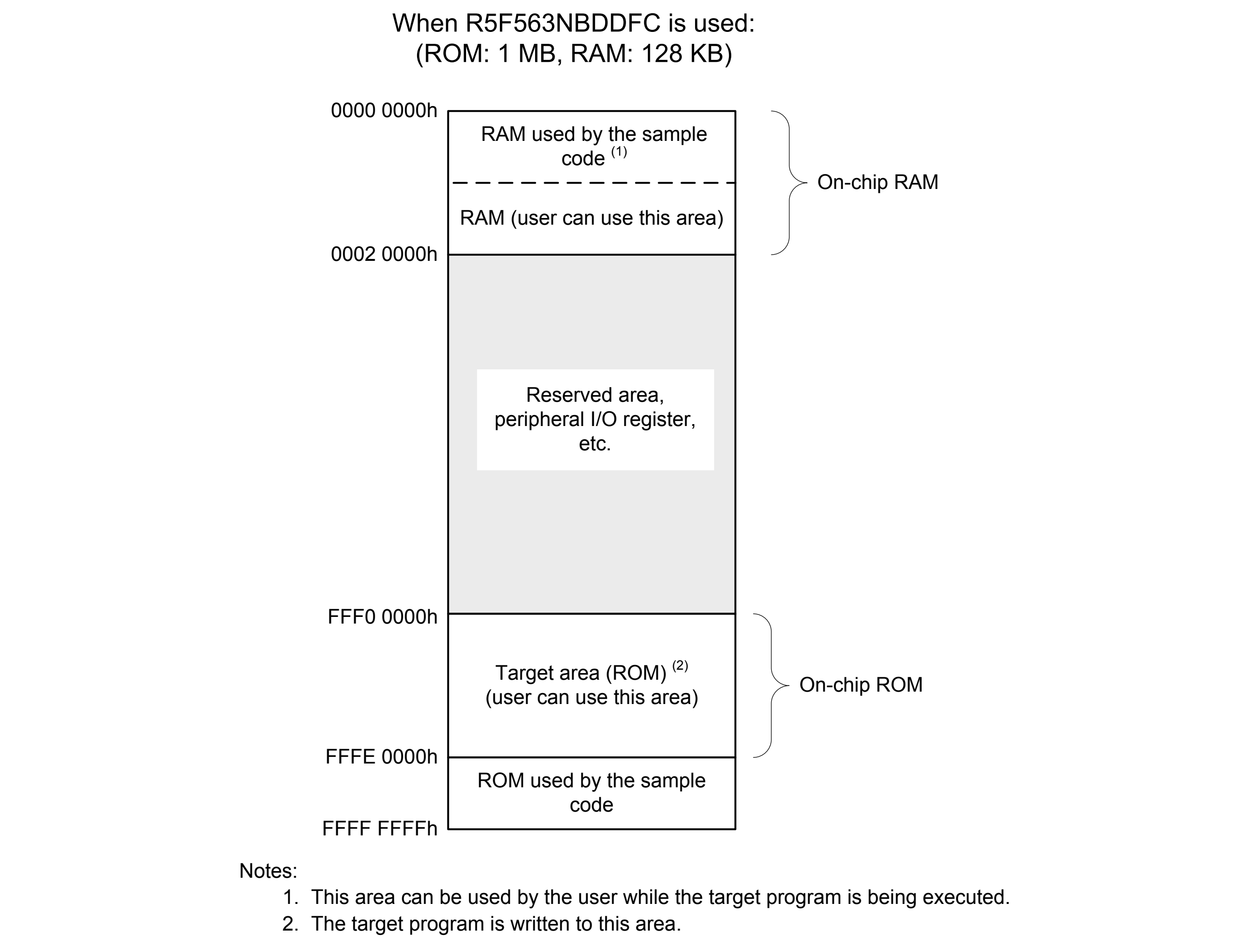

Figure 5.1 Memory Allocation

In the sample code, the flash memory is reprogrammed in the following steps.

(1) After a reset, if the MCU is not connected to a PC and the value in the target reset vector (address FFFD FFFCh) is not FFFF FFFFh, the target program is executed. If connected to the PC, the MCU transmits the message "Press any key" to the PC at a certain intervals until the MCU receives a data (until the terminal application is started and a key is pressed on the keyboard). See Figure 5.2 below.

(2) When the MCU receives data from the PC, it transmits the command list to the PC and waits for receiving a command data. See Figure 5.3 below.

(3) When a command is received, the MCU performs the operation (command display, blank check, erase, program, or target program execution) according to the command.

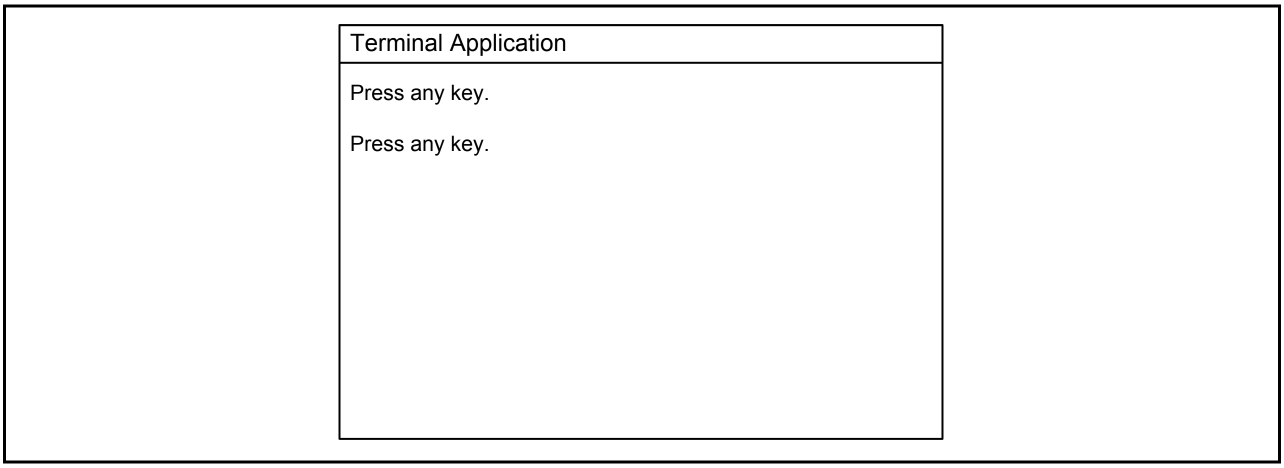

Figure 5.2 Screen Example When Waiting for Key Input

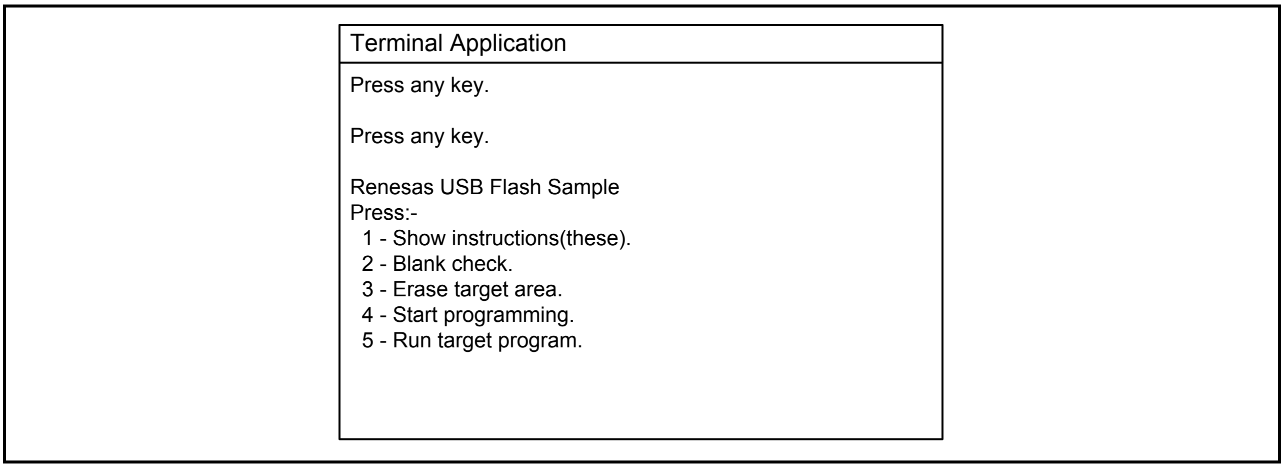

Figure 5.3 Screen Example for Inputting a Command

5.2 Commands

Table 5.1 lists the Commands. The commands are listed on the terminal application by the show instructions command.

Table 5.1 Commands

Key Entered on the PC	ASCII Code	Command
1	31h	Show instructions(these)
2	32h	Blank check
3	33h	Erase target area.
4	34h	Start programming
5	35h	Run the target program

5.3 Start Address of the Target Program

In the sample code, if the USB is not connected after a reset, or if the target program execution command is selected, the target program is executed. The target program is executed from the address written in address FFFD FFFCh (target reset vector). This means that the reset vector of the target program should be FFFD FFFCh. Thus the start address of the target program needs to be stored in the target reset vector in advance.

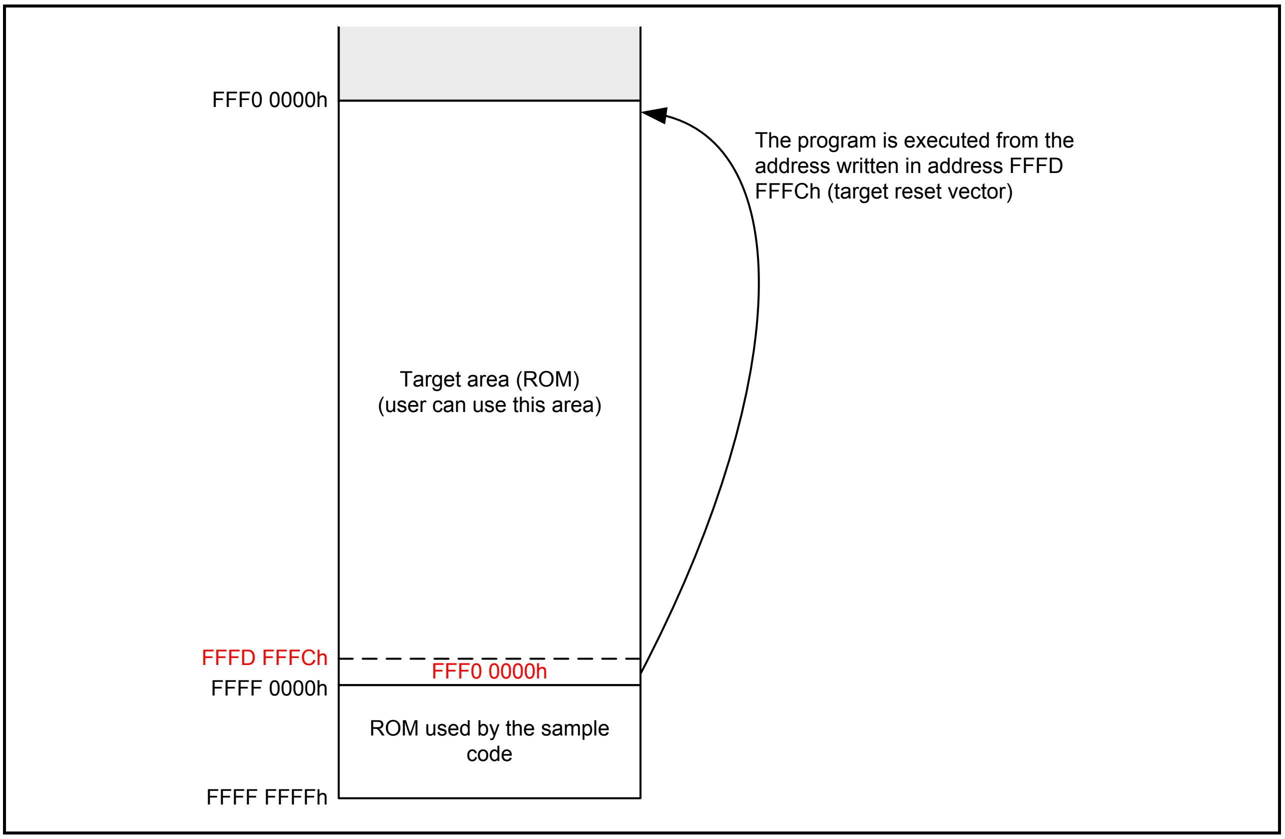

Figure 5.4 Target Reset Vector

5.4 Mode Transitions

Figure 5.5 shows the Mode Transitions in the Sample Codes.

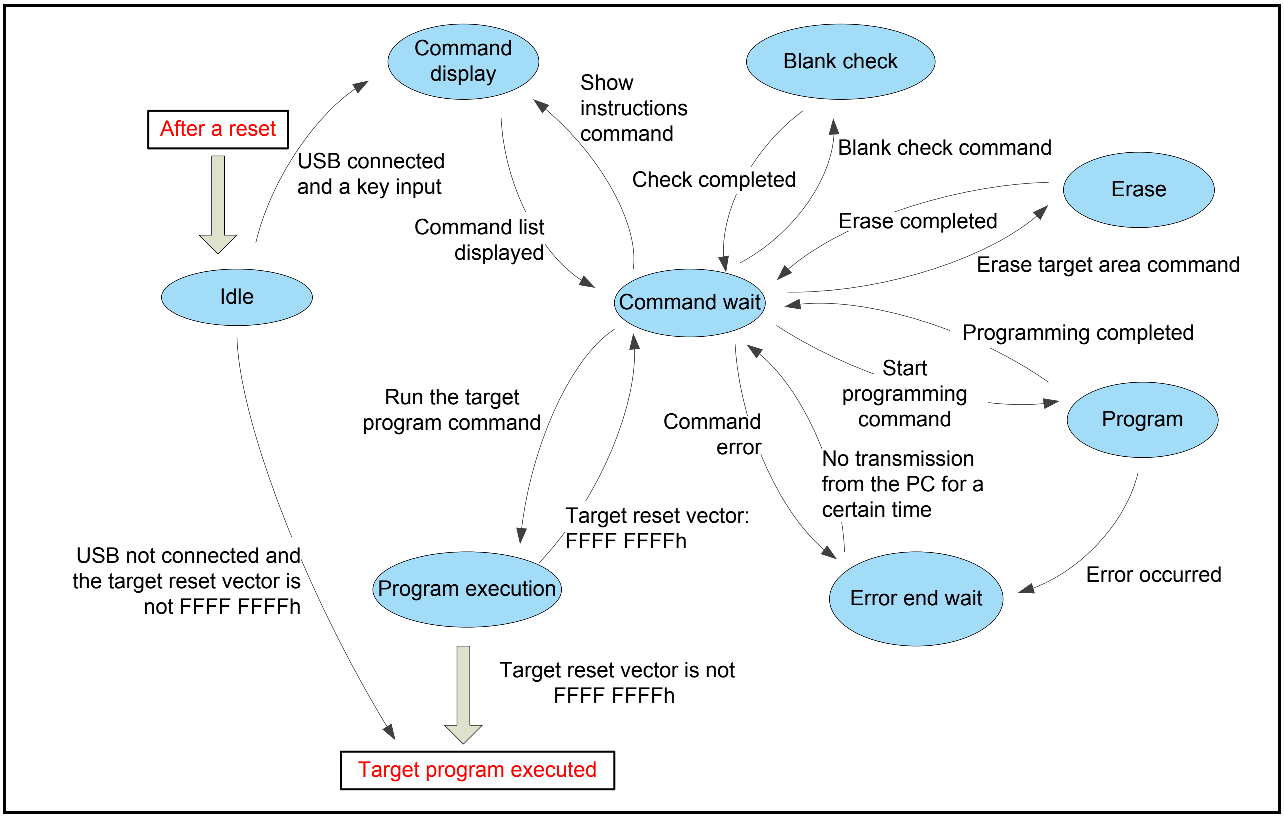

Figure 5.5 Mode Transitions in the Sample Codes

5.4.1 Idle Mode

After a reset, the MCU enters idle mode. If the USB is not connected and the target reset vector has a value other than FFFF FFFFh, the target program is executed. If the USB is connected, the MCU transmits the message to prompt for key input at a certain intervals. When a key is pressed on the PC (i.e., when the MCU receives a given data), the MCU enters command display mode.

5.4.2 Command display Mode

The command list is displayed. After the list is displayed, the MCU enters command wait mode.

5.4.3 Command Wait Mode

The MCU waits for a command from the PC. When a command is received, the MCU enters a mode according to the command received.

5.4.4 Blank Check Mode

The blank check is performed for the target area. When the blank check is completed, the blank check result is displayed and the MCU enters command wait mode.

5.4.5 Erase Mode

The target area is erased using the Simple Flash API. When the erasing is completed, the erase result is displayed and the MCU enters command wait mode.

5.4.6 Program Mode

The target area is programmed using the Simple Flash API. After the MCU receives the start programming command, it waits for receiving a mot file. Send a mot file in ASCII from the PC. Operations after the MCU receives the command are as follows:

(1) The MCU waits for receiving 'S' (in ASCII) that is the first data of the Motorola S format. Any data other than 'S' is discarded.

(2) After receiving 'S', the MCU receives data in Motorola S format and programs the received data. Once 'S' is received, if the subsequent received data is not in Motorola S format, the MCU enters error end wait mode. It also enters error end wait mode if the other error occurs such as checksum. Refer to 5.10 “Message Table List” for details on errors and displayed messages.

(3) When the program operation is completed, a message to inform of the result is displayed. Then the MCU enters command wait mode.

Note: • Once data is determined as Motorola S format data, data through to checksum is recognized as Motorola S format in the sample code. The sample code discards data after checksum, which is received simultaneously. If the data which is supposed to be discarded is received after the actual discard operation, the MCU receives the data as a command. In this case the message for command error may be displayed.

5.4.7 Error End Wait Mode

If the MCU receives a command not listed in 5.2 Commands, or if an error occurs in program mode, the MCU enters error end wait mode. If no data is received from the PC for a certain period during error end wait mode, the MCU enters command wait mode.

When there is an error in a received mot file, the MCU cancels the program operation while the terminal application may continue to transmit data. Then if the MCU enters command wait mode immediately after an error occurred, such transmit data may be received as a command. Therefore the MCU enters error end wait mode once and waits until no data is transmitted from the PC. Note that any data received from the PC during error end wait mode will be discarded.

5.4.8 Program Execution Mode

The USB is stopped and the target program is executed. If the value of the target reset vector is FFFF FFFFh, an appropriate error message is displayed and the MCU enters command wait mode.

5.5 Data Flow when Programming

Figure 5.6 shows the flow of data within the MCU when programming the target area on the flash memory.

When receiving:

(1) Data received from the PC is transferred to the receive ring buffer.

(2) One record of Motorola S format data is copied to the MotS buffer (ASCII).

(3) The header part of the Motorola S format data is analyzed, and at the same time, ASCII data is converted to binary data and stored in the MotS buffer (binary).

(4) Data is stored in the programming buffer.
Data is programmed in the user area of the MCU in 256 bytes. Steps (1) to (4) are repeated until the size of the stored programming data becomes 256 bytes. If the size exceeds 256 bytes, the excess data is temporarily stored and used when programming the next 256 bytes.

(5) The prepared data (256 bytes) is written to the flash memory using the Simple Flash API.

When transmitting:

(6) The result of the programming operation is determined by the return value from the Simple Flash API.

(7) The message corresponding to the result of the programming operation is stored in the transmit ring buffer.

(8) The message is transmitted to the PC using the USB PCDC driver.

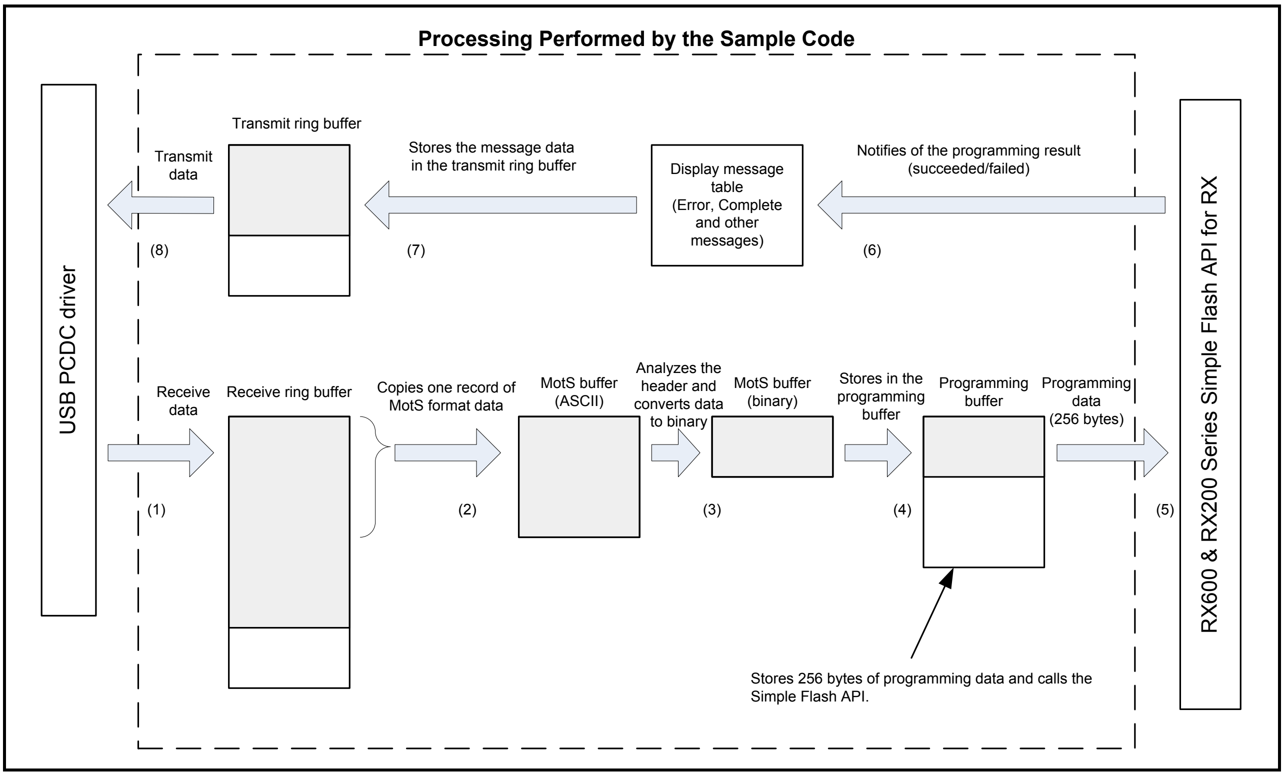

Figure 5.6 Data Flow when Programming

Figure 5.7 shows the Data Structures when Programming.

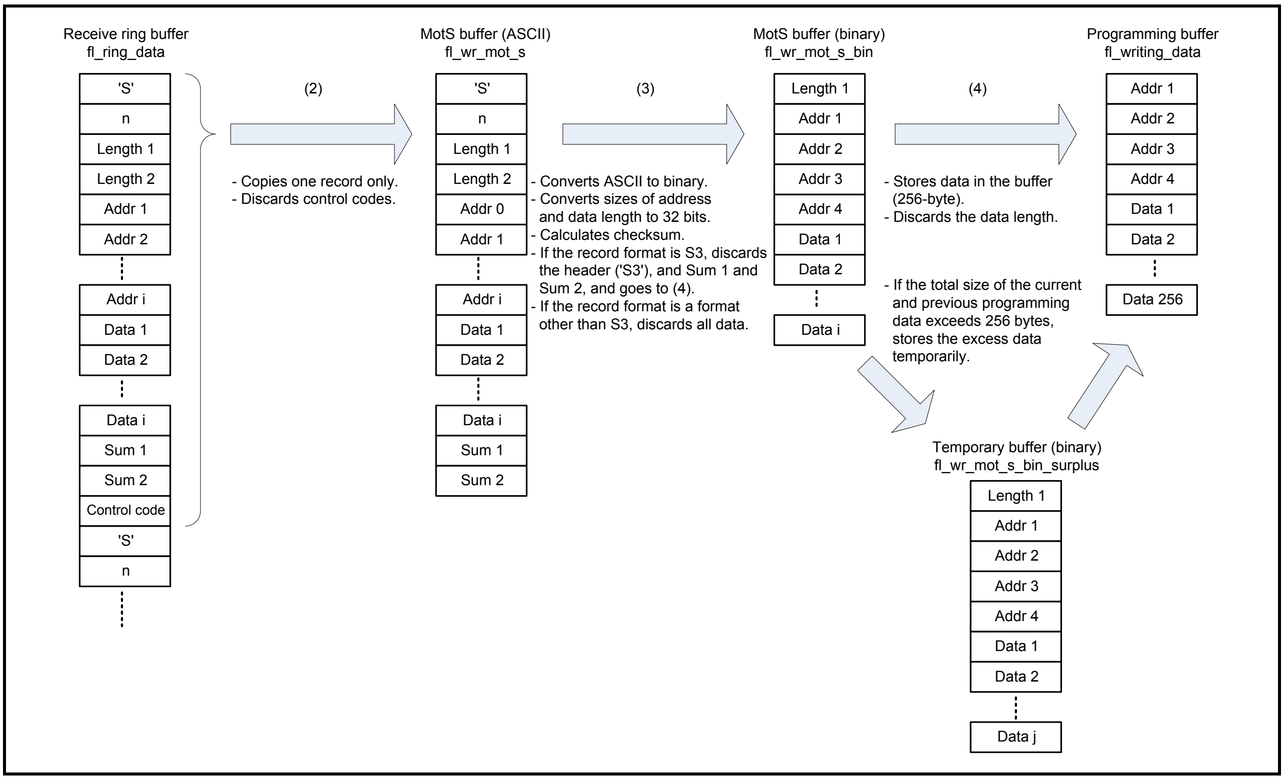

Figure 5.7 Data Structures when Programming

5.6 ROM Capacity

The sample code assumes 1 MB ROM is used. When an MCU with ROM capacity other than 1 MB is used, change the "FL_END_BLOCK_NUM" definition in the r_Flash_main.h file appropriate to the MCU used.

Table 5.2 lists the ROM Capacities of the Target Area.

Table 5.2 ROM Capacities of the Target Area

ROM Capacity	ROM Capacity of the Target Area	Start Address of the Target Area	Block Number of the Target Area
2 MB	1920 KB	FFE0 0000h	EB14 to EB69
1.5 MB	1408 KB	FFE8 0000h	EB14 to EB61
1 MB	896 KB	FFF0 0000h	EB14 to EB53
768 KB	640 KB	FFF4 0000h	EB14 to EB45
512 KB	384 KB	FFF8 0000h	EB14 to EB37
384 KB	256 KB	FFFA 0000h	EB14 to EB29
256 KB	128 KB	FFFC 0000h	EB14 to EB21

5.7 File Composition

Table 5.3 lists the Files Used in the Sample Code.

Table 5.3 Files Used in the Sample Code

File Name	Outline	Remarks
r_flash_api_rx.c	RX600 & RX200 Series Simple Flash API program for RX	Refer to the RX600 & RX200 Series Simple Flash API for RX application note for details.
r_flash_api_rx63n.h	External reference header file for RX600 & RX200 Series Simple Flash API program for RX	
r_flash_api_rx_private.h	Header file for RX600 & RX200 Series Simple Flash API for RX.	
r_flash_api_rx_config.h	External reference header file for RX600 & RX200 Series Simple Flash API program for RX.	
r_flash_api_rx_if.h	External reference header file for RX600 & RX200 Series Simple Flash API program for RX.	
r_init_stop_module.c	Stop processing for active peripheral functions after a reset	
r_init_stop_module.h	Header file for r_init_stop_module.c	
r_init_non_existent_port.c	Nonexistent port initialization	
r_init_non_existent_port.h	Header file for r_init_non_existent_port.c	
r_init_clock.c	Clock initialization	
r_init_clock.h	Header file for r_init_clock.c	
r_Flash_main.c	Flash reprogramming data processing	
r_Flash_main.h	External reference header file for flash reprogramming data processing	
r_Flash_buff.c	Processing associated with buffers used for USB transmission/reception	
r_Flash_buff.h	External reference header file for processing associated with buffers used for USB transmission/reception	
TrgtPrgDmmy.c	Dummy program for allocating space for the target program	
Files in the r_bsp folder	Programs in the r_bsp package used for RX600 & RX200 Series Simple Flash API program for RX	
Other files	Programs of the USB PCDC driver	Refer to the “USB Basic Host and Peripheral firmware” and “USB Peripheral Communications Device Class Driver” application notes for details.

5.8 Constants

Table 5.4 lists the Constants Used in the Sample Code. Note that constants used in the USB PCDC driver and Simple Flash API are not included here.

Table 5.4 Constants Used in the Sample Code

Constant Name	Setting Value	Contents
FL_CMD_DATA_SHOW_INST	31h	Command value that is 1 in ASCII
FL_CMD_DATA_BLNK_CHECK	32h	Command value that is 2 in ASCII
FL_CMD_DATA_ERASE_TRGT_AREA	33h	Command value that is 3 in ASCII
FL_CMD_DATA_PRG_TRGT_AREA	34h	Command value that is 4 in ASCII
FL_CMD_DATA_RUN_TRGT_AREA	35h	Command value that is 5 in ASCII
FL_RINGBUFF_SIZE	1024	Ring buffer size for receiving data from the USB
FL_RINGBUFF2_SIZE	256	Ring buffer size for transmitting data to the USB
FL_USB_RCV_BLANK_SIZE	64	The data size that the USB can transmit at a time.
FL_SEND_END_CODE	00h	End code of the message table
FL_MOTS_ADDR_SIZE	4	Buffer size for the address of Motorola S format data
FL_MOTS_SUM_SIZE	1	Buffer size for the checksum of Motorola S format data
FL_START_BLOCK_NUM	14	First block of the target area
FL_END_BLOCK_NUM	53	Last block of the target area
FL_TARGET_REST_VECT_ADDR	FFFD FFFCh	Target reset vector
FL_USB_UNCONNECT_WAIT_PERIOD	10000h	Wait time until the target program is executed with the USB disconnected
FL_IDLE_MESSAGE_OUTPUT_PERIOD	10000h	Interval of time to display the message when in idle mode
FL_ERROR_WAIT_PERIOD	10000h	Wait time for completion of an error (the wait processing is terminated if nothing is received from the USB during counting this value.)

5.9 Structure/Union List

Figure 5.8 shows the Structure/Union Used in the Sample Code. Note that structure/union used in the USB PCDC driver and Simple Flash API are not included here.

```
/* buffer for mot S format data */
typedef struct {
    uint8_t type[2]; /* "S0", "S1" and so on */
    uint8_t len[2]; /* "0-255" */
    uint8_t addr_data_sum[512];
} Fl_prg_mot_s_t;

/* buffer for write data
    (this data is the converted data from mot S format data) */
typedef struct {
    uint8_t len;
    uint32_t addr;
    uint8_t data[256];
} Fl_prg_mot_s_binary_t;

/* buffer for writing flash */
typedef struct {
    uint32_t addr;
    uint8_t data[256];
} Fl_prg_writing_data_t;
```

Figure 5.8 Structure/Union Used in the Sample Code

5.10 Message Table List

Table 5.5 lists messages used in the sample code. The new line character is omitted from each message in the table.

Table 5.5 Messages Used in the Sample Code

Message	Description
Press any key.	Displayed periodically in idle mode.
Renesas USB Flash Sample Press:- 1 - Show instructions(these). 2 - Blank check. 3 - Erase target area. 4 - Start programming. 5 - Run target program.	Command list
Target area is blank.	After a blank check, displayed when the target area is blank.
Target area is NOT blank.	After a blank check, displayed when the target area is not blank.
Erase target area...	Displayed when an erase operation is in progress.
Erase complete.	Displayed when an erase operation is completed.
ERROR!!! - Erase is failed.	Displayed when an erase operation is failed.
Please send a mot file.	Displayed when a program operation is started. Send a mot file after this message is displayed.
Program complete.	Displayed when a program operation is completed.
Program failed.	Displayed when a program operation is failed.
ERROR!!! - Verify error.	Displayed when a verify error occurred.
Run target program.	Displayed when the target program is executed.
ERROR!!! - Target reset vector(0xFFFDFFFC) is 0xFFFFFFFF.	Displayed when the target vector is FFFF FFFFh in attempting to execute the target program.
ERROR!!! - Command error. Please press a number from 1 to 5.	Displayed when an incorrect command is entered.
ERROR!!! - Mot file format is NOT correct.	Displayed when an incorrect target program is transmitted.
ERROR!!! - Check sum error.	Displayed when a checksum error occurred in Motorola S format in the target program.
Please wait for instructions to be shown.	Displayed when an error occurred and data continues to be transmitted. The MCU enters command wait mode when data is not transmitted for a certain period. If data continues to be sent after the error occurrence, a period (.) is displayed in regular intervals.
Period (.)	Displayed in regular intervals after an error occurred.
Newline code (\r\n)	Used for a newline following each message.

5.11 Functions

Table 5.6 lists the Functions. Note that functions used in the USB PCDC driver and Simple Flash API are not included here.

Table 5.6 Functions

Function Name	Outline
R_INIT_StopModule	Stop processing for active peripheral functions after a reset
R_INIT_NonExistentPort	Nonexistent port initialization
R_INIT_Clock	Clock initialization
R_FI_Rewrite_process	Reprogramming main processing
R_FI_Idle	Processing during idle mode (until a key is pressed after a reset)
R_FI_AnalyzeCMD	Command analysis
R_FI_EraseTrgtArea	Erase processing
R_FI_PrgTrgtArea	Program processing
R_FI_RunTrgtPrg	Executing the target program
R_FI_ErrorWait	Wait for completion of an error processing
R_FI_cmd_ShowInst	Displaying commands
R_FI_cmd_BlankCheckStart	Start processing for blank check
R_FI_cmd_EraseStart	Start processing for erasing
R_FI_cmd_PrgStart	Start processing for programming
R_FI_cmd_RunTrgtPrgStart	Executing the target program
R_FI_Blnk_BlankCheck	Blank check
R_FI_Ers_EraseFlash	Erasing the target area
R_FI_Prg_StoreMotS	Storing Motorola S format data
R_FI_Prg_ProcessForMotS_data	Motorola S format header analysis and binary data conversion
R_FI_Prg_MotS_AsciiToBinary	ASCII to binary conversion of Motorola S format data
R_FI_Prg_MakeWriteData	Making programming data
R_FI_Prg_WriteData	Programming the target area
R_FI_Prg_ClearMotSVariables	Clearing variables for Motorola S format data
R_FI_Run_StopUSB	Stopping USB
R_FI_RcvDataString	Storing USB receive data
R_FI_SetSendData	Storing USB transmit data
R_FI_SetDisplayMsgData	Setting message data to be displayed (dot display)
R_FI_RingCheckBlank	Checking receive ring buffer space
R_FI_Ring2CheckData	Checking data in the transmit ring buffer
R_FI_USB_NonConnect_Run	Executing the target program with the USB disconnected
R_FI_RingEnQueue	Storing data in the receive ring buffer
R_FI_RingDeQueue	Reading the receive ring buffer
R_FI_RingClear	Clearing the receive ring buffer
R_FI_RingCheck	Checking the number of data in the receive ring buffer
R_FI_Ring2EnQueue	Storing data in the transmit ring buffer
R_FI_Ring2DeQueue	Reading the transmit ring buffer
R_FI_Ring2Clear	Clearing the transmit ring buffer
R_FI_Ring2Check	Checking the number of data in the transmit ring buffer
R_FI_AsciiToHexByte	ASCII to Binary conversion

5.12 Function Specifications

The following tables list the sample code function specifications.

R_INIT_StopModule	
Outline	Stop processing for active peripheral functions after a reset
Header	r_init_stop_module.h
Declaration	void R_INIT_StopModule(void)
Description	Configures the setting to enter the module-stop state.
Arguments	None
Return Value	None
Remarks	Transition to the module-stop state is not performed in the sample code. Refer to the RX63N Group, RX631 Group Initial Setting Rev. 1.10 application note for details on this function.

R_INIT_NonExistentPort	
Outline	Nonexistent port initialization
Header	r_init_non_existent_port.h
Declaration	void R_INIT_NonExistentPort(void)
Description	Initializes port direction registers for ports that do not exist in products with less than 176 pins.
Arguments	None
Return Value	None
Remarks	The number of pins in the sample code is set for the 176-pin package (PIN_SIZE=176). After this function is called, when writing in byte units to the PDR registers or PODR registers which have nonexistent ports, set the corresponding bits for nonexistent ports as follows: set the I/O select bits in the PDR registers to 1 and set the output data store bits in the PODR registers to 0. Refer to the RX63N Group, RX631 Group Initial Setting Rev. 1.10 application note for details on this function.

R_INIT_Clock	
Outline	Clock initialization
Header	r_init_clock.h
Declaration	void R_INIT_Clock(void)
Description	Initializes the clock.
Arguments	None
Return Value	None
Remarks	The sample code selects processing which uses PLL as the system clock without using the sub-clock. The following settings have been changed from the original settings. 1. BCLK division ratio: Changed from divide-by-4 to divide-by-8. 2. USB clock: Changed “not used” to divide-by-4. 3. BCLK pin output: Changed “no division” to divide-by-2. Refer to the RX63N Group, RX631 Group Initial Setting Rev. 1.10 application note for details on this function.

R_Fl_Rewrite_process	
Outline	Reprogramming main processing
Header	r_Flash_main.h
Declaration	void R_Fl_Rewrite_process(void)
Description	Calls functions for processing according to programming mode.
Arguments	None
Return Value	None

R_Fl_Idle	
Outline	Processing during idle mode (until a key is pressed after a reset)
Header	None
Declaration	static void R_Fl_Idle(void)
Description	Displays “Press any key” in a regular intervals until the USB receives any data (until any key is pressed from PC).
Arguments	None
Return Value	None

R_Fl_AnalyzeCMD	
Outline	Command analysis
Header	None
Declaration	static Fl_SMPL_command_t R_Fl_AnalyzeCMD(void)
Description	When data is present in the receive ring buffer, the first byte of the data is analyzed as a command. Then the rest of data is discarded.
Arguments	None
Return Value	- Data not received: FL_CMD_NONE - Show instruction command received: FL_CMD_SHOW_INST - Blank check command received: FL_CMD_BLNK_CHECK - Erase target area command received: FL_CMD_ERASE_TRGT_AREA - Start programming command received: FL_CMD_PRG_TRGT_AREA - Run the target program command received: FL_CMD_RUN_TRGT_AREA - Anything other than above received: FL_CMD_ERROR

R_Fl_EraseTrgtArea	
Outline	Erase processing
Header	r_Flash_main.h
Declaration	static void R_Fl_EraseTrgtArea(void)
Description	Calls the function to erase the target area and displays the message for the erase result.
Arguments	None
Return Value	None

R_Fl_PrgTrgtArea	
Outline	Program processing
Header	None
Declaration	static void R_Fl_PrgTrgtArea(void)
Description	When data is present in the receive ring buffer, calls the function to store the data in Motorola S format. When one record of Motorola S format data is received, calls the function to analyze the header and convert the data to binary data. When the conversion to binary data is completed, calls the function to store the converted data in the programming buffer.
Arguments	None
Return Value	None

R_Fl_RunTrgtPrg	
Outline	Executing the target program
Header	None
Declaration	static void R_Fl_RunTrgtPrg(void)
Description	If the target reset vector is a value other than FFFF FFFFh, stops the USB and then executes the target program. If the target reset vector is FFFF FFFFh, displays the message for target vector error.
Arguments	None
Return Value	None

R_Fl_ErrorWait	
Outline	Error end wait mode processing
Header	None
Declaration	static void R_Fl_ErrorWait(void)
Description	If a certain period of time elapsed while the receive ring buffer is empty, the MCU enters command wait mode. If there is any data in the receive ring buffer, the data is discarded and then a wait processing for a certain period is performed again.
Arguments	None
Return Value	None

R_Fl_cmd_ShowInst	
Outline	Displaying commands
Header	None
Declaration	static void R_Fl_cmd_ShowInst(void)
Description	Displays commands.
Arguments	None
Return Value	None

R_Fl_cmd_BlankCheckStart	
Outline	Start processing for blank check
Header	None
Declaration	static void R_Fl_cmd_BlankCheckStart(void)
Description	Calls the function to blank check for the target area and displays the message for the result.
Arguments	None
Return Value	None

R_FI_cmd_EraseStart	
Outline	Start processing for erasing
Header	None
Declaration	static void R_FI_cmd_ EraseStart (void)
Description	Starts erasing the target area and displays the message for starting an erase operation.
Arguments	None
Return Value	None

R_FI_cmd_PrgStart	
Outline	Start processing for programming
Header	None
Declaration	static void R_FI_cmd_PrgStart(void)
Description	Starts programming the target area and displays the message for starting a program operation.
Arguments	None
Return Value	None

R_FI_cmd_RunTrgtPrgStart	
Outline	Executing the target program
Header	None
Declaration	static void R_FI_cmd_RunTrgtPrgStart(void)
Description	If the target vector is FFFF FFFFh, the message for the target vector error is transmitted. If the target vector is not FFFF FFFFh, the message for target program execution is displayed and the MCU enters program execution mode.
Arguments	None
Return Value	None

R_FI_Blnk_BlankCheck	
Outline	Blank check
Header	None
Declaration	FI_API_SMPL_rtn_t R_FI_Blnk_BlankCheck(void)
Description	Performs a blank check for the target area.
Arguments	None
Return Value	- When the target area is blank: FLASH_API_SAMPLE_OK - When the target area is not blank: FLASH_API_SAMPLE_NG

R_FI_Ers_EraseFlash	
Outline	Erasing the target area
Header	None
Declaration	FI_API_SMPL_rtn_t R_FI_Ers_EraseFlash(void)
Description	Erases the target area.
Arguments	None
Return Value	- When the erase is completed successfully: FLASH_API_SAMPLE_OK - When the erase is failed: FLASH_API_SAMPLE_NG
Remarks	To avoid a ROM access by an interrupt during the erase operation, the processor interrupt priority level (IPL) for the processor status word (PSW) is changed.

R_Fl_Prg_StoreMotS	
Outline	Storing Motorola S format data
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_Prg_StoreMotS(uint8_t)
Description	Stores the data, which is received with the argument, in Motorola S format in bytes. Any data is discarded until 'S' in ASCII is received.
Arguments	mot_data: Motorola S format data
Return Value	- When one record of Motorola S format data (from 'S' to checksum) is stored: FLASH_API_SAMPLE_OK - When the stored Motorola S format data is not enough for one record: FLASH_API_SAMPLE_NG
Remarks	- To use this function, one byte of Motorola S format data is passed repeatedly with the argument. - Checksum is not calculated.

R_Fl_Prg_ProcessForMotS_data	
Outline	Motorola S format header analysis and binary data conversion
Header	None
Declaration	static void R_Fl_Prg_ProcessForMotS_data(void)
Description	Analyzes the Motorola S format header and calls the function to convert the data to binary. If there is non-Motorola S format data, the error message is displayed.
Arguments	None
Return Value	None

R_Fl_Prg_MotS_AsciiToBinary	
Outline	ASCII to binary conversion of Motorola S format data
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_Prg_MotS_AsciiToBinary (Fl_prg_mot_s_t *, Fl_prg_mot_s_binary_t *)
Description	Converts Motorola S format in ASCII to binary data and verifies the checksum of the converted binary data. If there is non-Motorola S format data, the error message is displayed. If a checksum error occurs, the error message is displayed.
Arguments	First argument: *tmp_mot_s: Pointer to Motorola S format data in ASCII Second argument: *tmp_mot_s_binary: Pointer to the variable for storing data which is converted to binary
Return Value	- When conversion is completed: FLASH_API_SAMPLE_OK - When conversion is not completed: FLASH_API_SAMPLE_NG

R_Fl_Prg_MakeWriteData	
Outline	Making programming data
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_Prg_MakeWriteData(void)
Description	Makes data separated by 256 bytes.
Arguments	None
Return Value	- When 256 bytes of programming data is made successfully: FLASH_API_SAMPLE_OK - When 256 bytes of programming data is not made: FLASH_API_SAMPLE_NG

R_Fl_Prg_WriteData	
Outline	Programming the target area
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_Prg_WriteData(void)
Description	Programming is performed for the target area and verifies the programmed data. If the programming is failed, the error message is displayed. If the verify error occurs, the error message is displayed.
Arguments	None
Return Value	- When programming is completed successfully: FLASH_API_SAMPLE_OK - When programming is failed: FLASH_API_SAMPLE_NG
Remarks	To avoid a ROM access by an interrupt during the programming operation, the processor interrupt priority level (IPL) for the processor status word (PSW) is changed.

R_Fl_Prg_ClearMotSVariables	
Outline	Clearing variables for Motorola S format data
Header	None
Declaration	static void R_Fl_Prg_ClearMotSVariables(void)
Description	Clears variables for Motorola S format data.
Arguments	None
Return Value	None

R_Fl_Run_StopUSB	
Outline	Stopping USB
Header	iodefine.h, r_usb_usrconfig.h
Declaration	static void R_Fl_Run_StopUSB(void)
Description	Stops the USB.
Arguments	None
Return Value	None

R_Fl_RcvDataString		
Outline	Storing USB receive data	
Header	R_Flash_main.h	
Declaration	Fl_API_SMPL_rtn_t R_Fl_RcvDataString(void *, uint16_t)	
Description	Stores data received by the USB in the receive ring buffer.	
Arguments	First argument: *tranadr:	Pointer to the buffer for storing data received by the USB
	Second argument: length:	The number of data received by the USB
Return Value	- When the received data is stored successfully: FLASH_API_SAMPLE_OK - When the receive ring buffer was full: FLASH_API_SAMPLE_NG	

R_Fl_SetSendData		
Outline	Storing USB transmit data	
Header	R_Flash_main.h	
Declaration	uint16_t R_Fl_SetSendData(void *, uint16_t)	
Description	Stores data to be transmitted by the USB in the transmit ring buffer.	
Arguments	First argument: *tranadr:	Pointer to the transmit buffer
	Second argument: length_lim:	Limit number of data stored in the transmit buffer
Return Value	The number of data stored in the transmit buffer	

R_FI_SetDisplayMsgData	
Outline	Setting message data to be displayed (dot display)
Header	None
Declaration	static FI_API_SMPL_rtn_t R_FI_SetDisplayMsgData(FI_disp_tbl_num_t)
Description	Stores the specified message in the transmit ring buffer.
Arguments	table_num: Message number to be displayed
Return Value	- When the transmit data is stored successfully: FLASH_API_SAMPLE_OK - When the transmit ring buffer was full: FLASH_API_SAMPLE_NG

R_FI_RingCheckBlank	
Outline	Checking receive ring buffer space
Header	R_Flash_main.h
Declaration	FI_API_SMPL_rtn_t R_FI_RingCheckBlank(void)
Description	Checks whether the receive ring buffer has enough space for a USB reception (64 bytes).
Arguments	None
Return Value	- When the buffer has enough space: FLASH_API_SAMPLE_OK - When the buffer does not have enough space: FLASH_API_SAMPLE_NG

R_FI_Ring2CheckData	
Outline	Checking data in the transmit ring buffer
Header	R_Flash_main.h
Declaration	FI_API_SMPL_rtn_t R_FI_Ring2CheckData(void)
Description	Checks whether data is present in the transmit ring buffer.
Arguments	None
Return Value	- When data is present: FLASH_API_SAMPLE_OK - When no data is present: FLASH_API_SAMPLE_NG

R_FI_USB_NonConnect_Run	
Outline	Executing the target program with the USB disconnected
Header	R_Flash_main.h
Declaration	void R_FI_USB_NonConnect_Run(void)
Description	Stops the USB and executes the target program.
Arguments	None
Return Value	None
Remarks	This function is called when the USB is disconnected.

R_FI_RingEnQueue	
Outline	Storing data in the receive ring buffer
Header	r_Flash_buff.h
Declaration	FI_API_SMPL_rtn_t R_FI_RingEnQueue(uint8_t)
Description	Stores data in the receive ring buffer.
Arguments	enq_data: Data to be stored
Return Value	- When data is stored successfully: FLASH_API_SAMPLE_OK - When the buffer was full: FLASH_API_SAMPLE_NG

R_Fl_RingDeQueue	
Outline	Reading the receive ring buffer
Header	r_Flash_buff.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_RingDeQueue(uint8_t *)
Description	Reads data from the receive ring buffer.
Arguments	*deq_data: Pointer to the buffer for storing the read data
Return Value	- When data is read successfully: FLASH_API_SAMPLE_OK - When no data to be read is present: FLASH_API_SAMPLE_NG

R_Fl_RingClear	
Outline	Clearing the receive ring buffer
Header	r_Flash_buff.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_RingClear(void)
Description	Clears the receive ring buffer.
Arguments	None
Return Value	FLASH_API_SAMPLE_OK is always returned.

R_Fl_RingCheck	
Outline	Checking the number of data in the receive ring buffer
Header	r_Flash_buff.h
Declaration	uint32_t R_Fl_RingCheck(void)
Description	Checks the number of data in the receive ring buffer.
Arguments	None
Return Value	The number of received data is returned.

R_Fl_Ring2EnQueue	
Outline	Storing data in the transmit ring buffer
Header	r_Flash_buff.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_Ring2EnQueue(uint8_t)
Description	Stores data in the transmit ring buffer.
Arguments	enq_data: data to be stored
Return Value	- When data is stored successfully: FLASH_API_SAMPLE_OK - When the buffer was full: FLASH_API_SAMPLE_NG

R_Fl_Ring2DeQueue	
Outline	Reading the transmit ring buffer
Header	r_Flash_buff.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_Ring2DeQueue(uint8_t *)
Description	Reads data in the transmit ring buffer.
Arguments	*deq_data: Pointer to the buffer for storing the read data
Return Value	- When data is read successfully: FLASH_API_SAMPLE_OK - When no data to be read is present: FLASH_API_SAMPLE_NG

R_Fl_Ring2Clear	
Outline	Clearing the transmit ring buffer
Header	r_Flash_buff.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_Ring2Clear(void)
Description	Clears the transmit ring buffer.
Arguments	None
Return Value	FLASH_API_SAMPLE_OK is always returned.

R_Fl_Ring2Check	
Outline	Checking the number of data in the transmit ring buffer
Header	r_Flash_buff.h
Declaration	uint32_t R_Fl_Ring2Check(void)
Description	Checking the number of data in the transmit ring buffer.
Arguments	None
Return Value	The number of transmit data is returned.

R_Fl_AsciiToHexByte		
Outline	ASCII to Binary conversion	
Header	r_Flash_buff.h	
Declaration	uint8_t R_Fl_AsciiToHexByte(uint8_t, uint8_t)	
Description	Converts 2-byte ASCII data to 1-byte binary data.	
Arguments	First argument: in_upper:	Upper byte of ASCII data
	Second argument: in_lower:	Lower byte of ASCII data
Return Value	Converted binary data is returned.	

5.13 Flowcharts

5.13.1 USB Main Processing

Figure 5.9 and Figure 5.10 show the USB Main Processing.

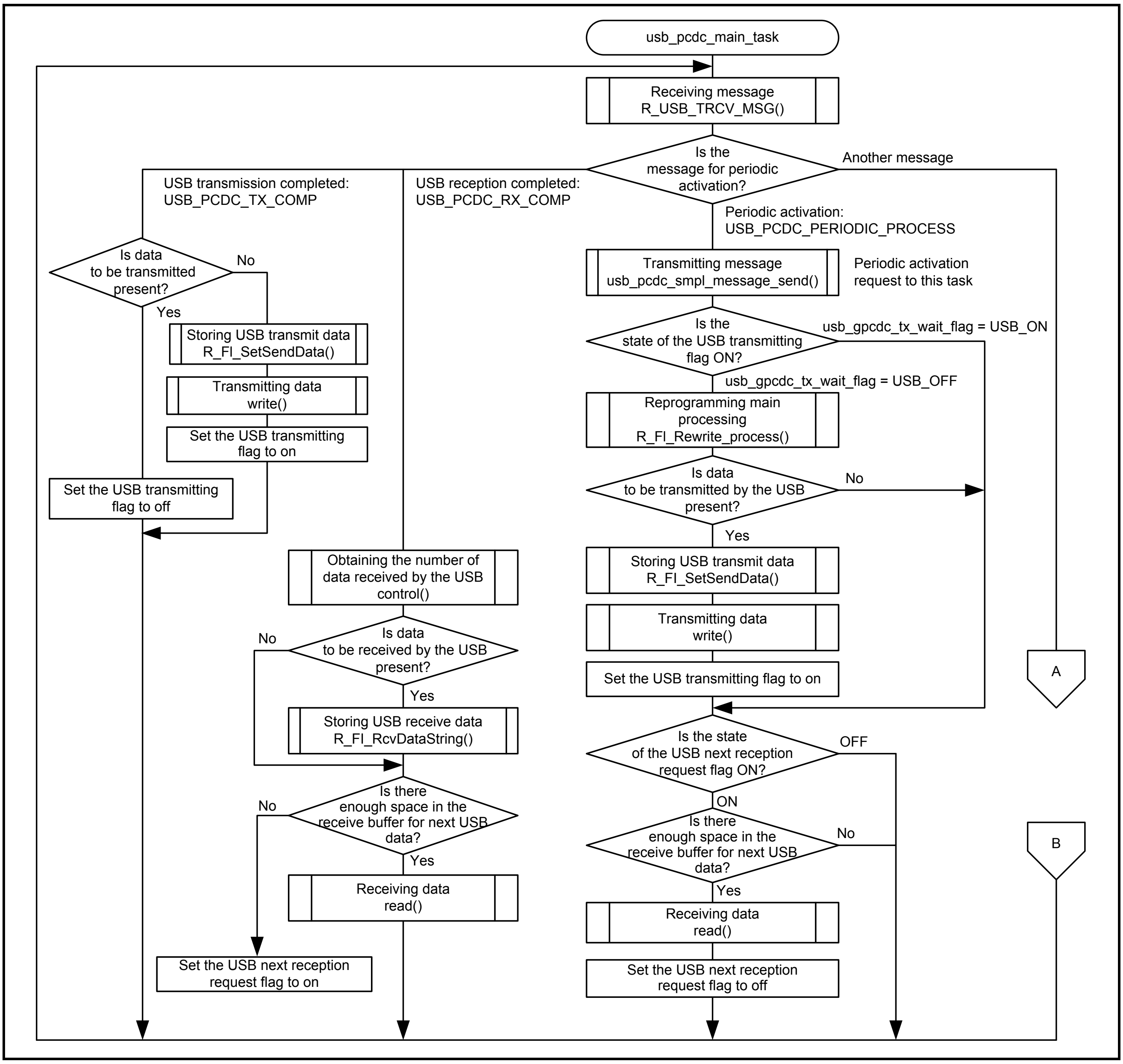

Figure 5.9 USB Main Processing (1/2)

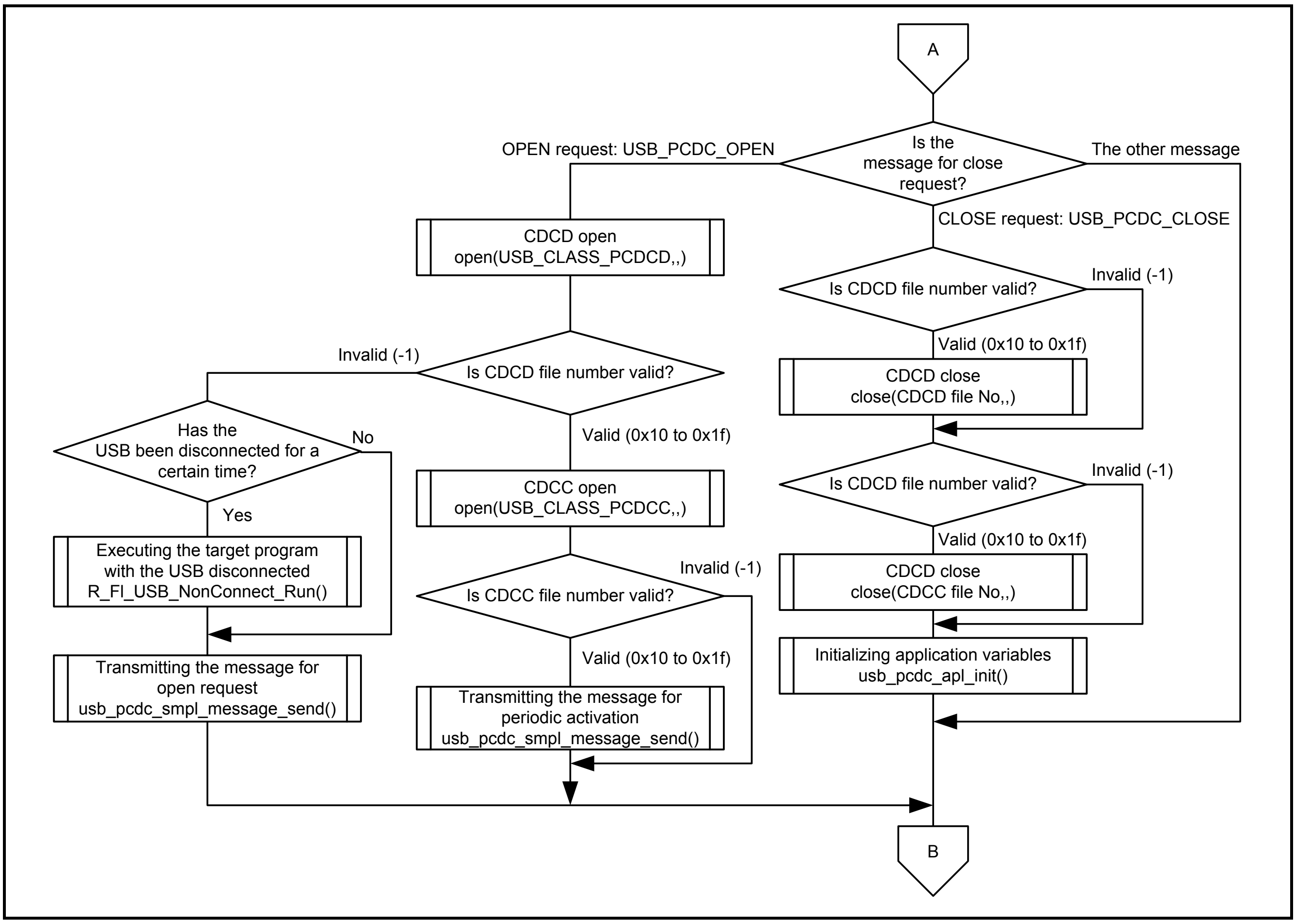

Figure 5.10 USB Main Processing (2/2)

5.13.2 Reprogramming Main Processing

Figure 5.11 shows the Reprogramming Main Processing.

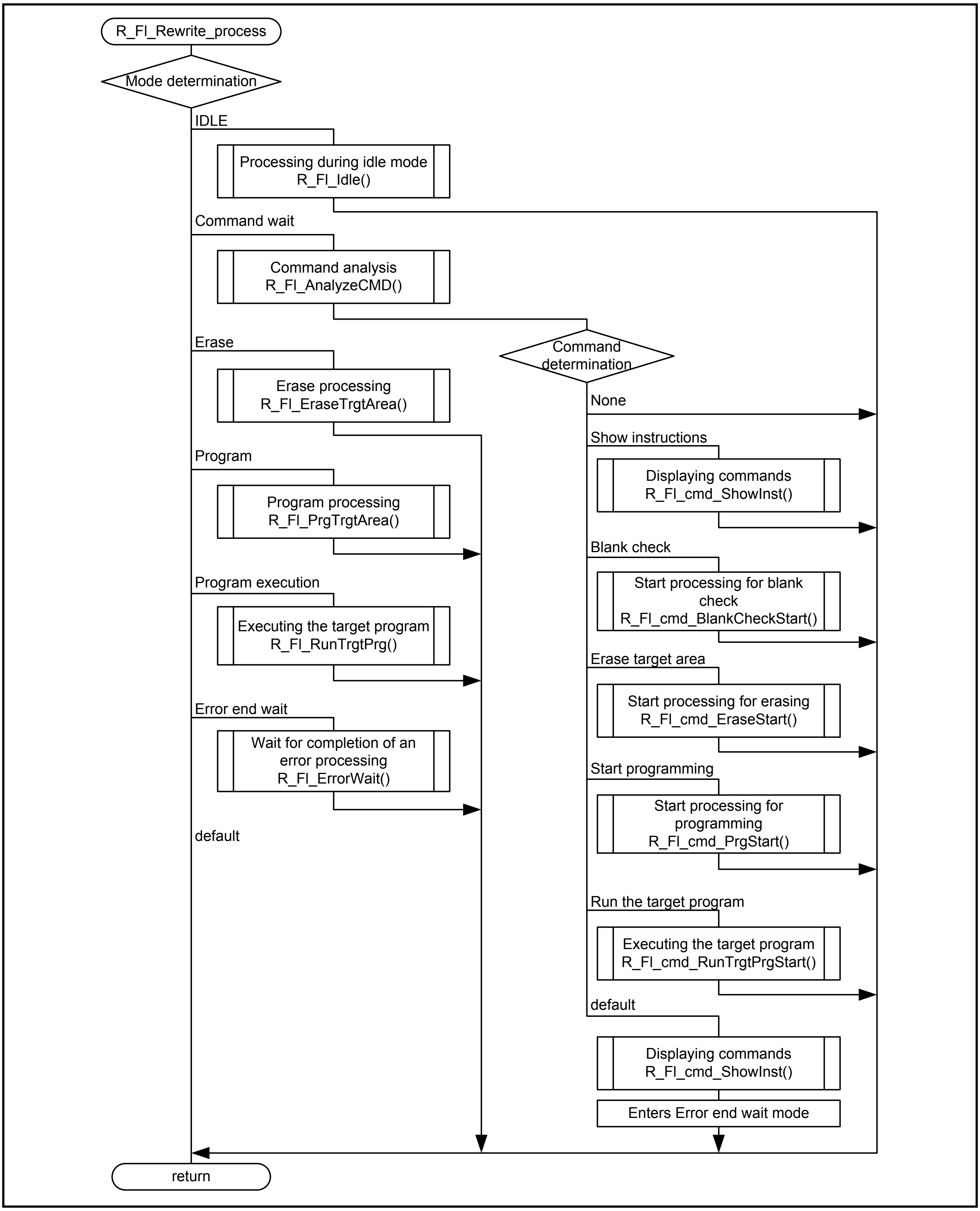

Figure 5.11 Reprogramming Main Processing

5.13.3 Processing During Idle Mode

Figure 5.12 shows the Processing During Idle Mode.

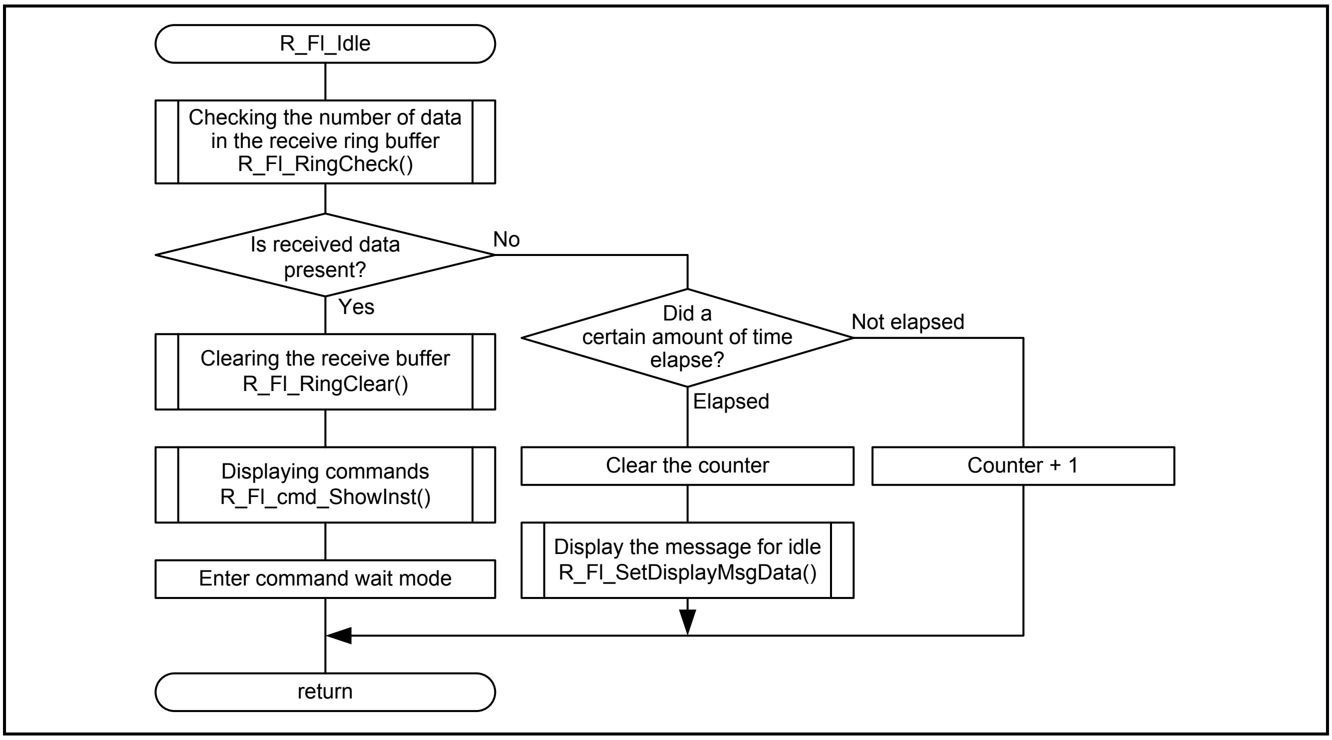

Figure 5.12 Processing During Idle Mode

5.13.4 Command Analysis

Figure 5.13 shows the Command Analysis.

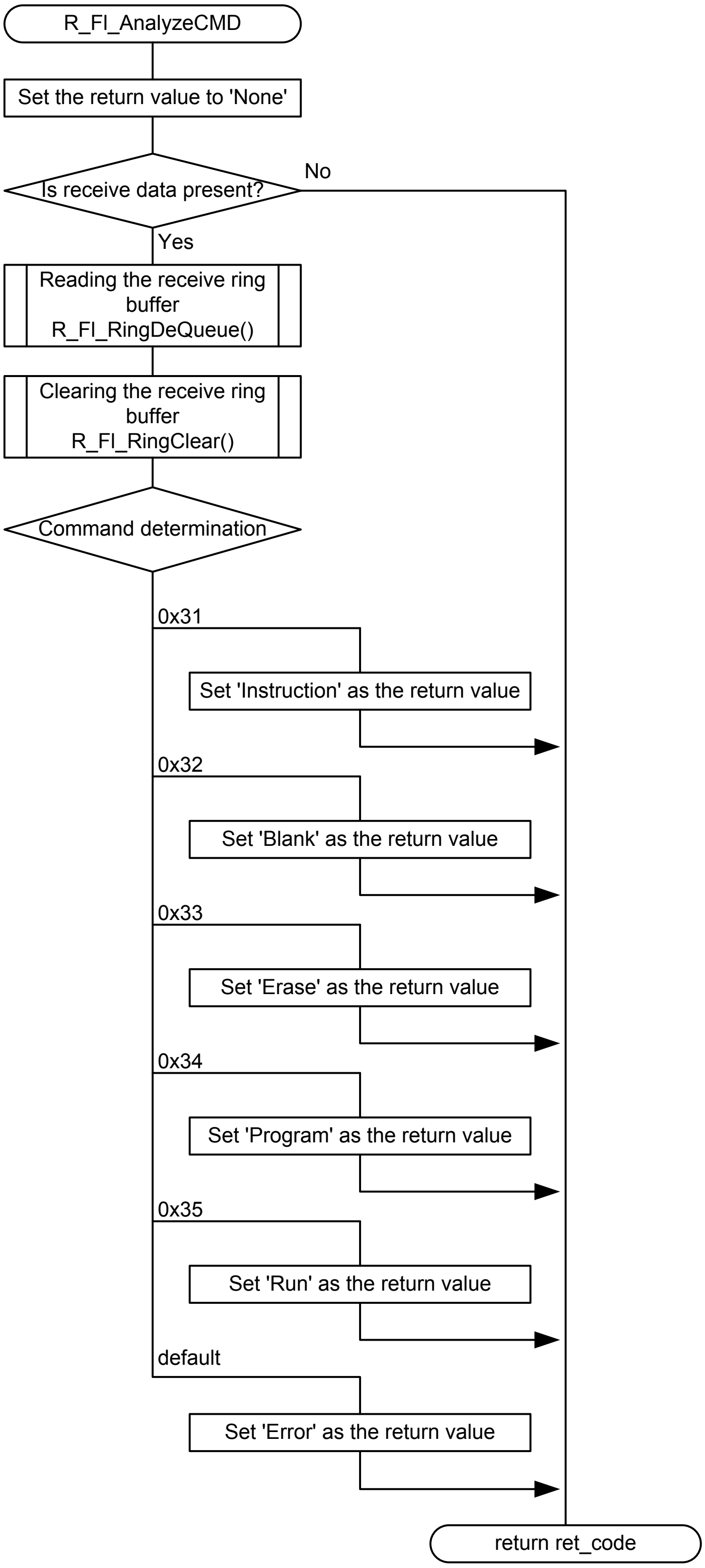

Figure 5.13 Command Analysis

5.13.5 Erase Processing

Figure 5.14 shows the Erase Processing.

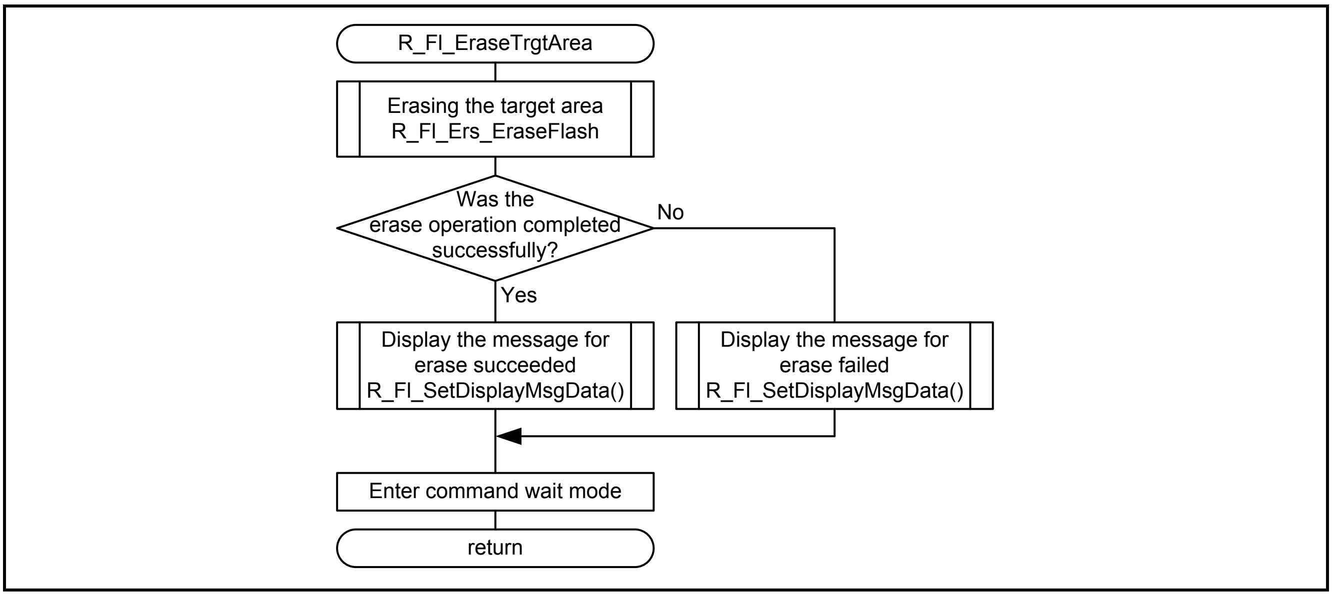

Figure 5.14 Erase Processing

5.13.6 Program Processing

Figure 5.15 shows the Program Processing.

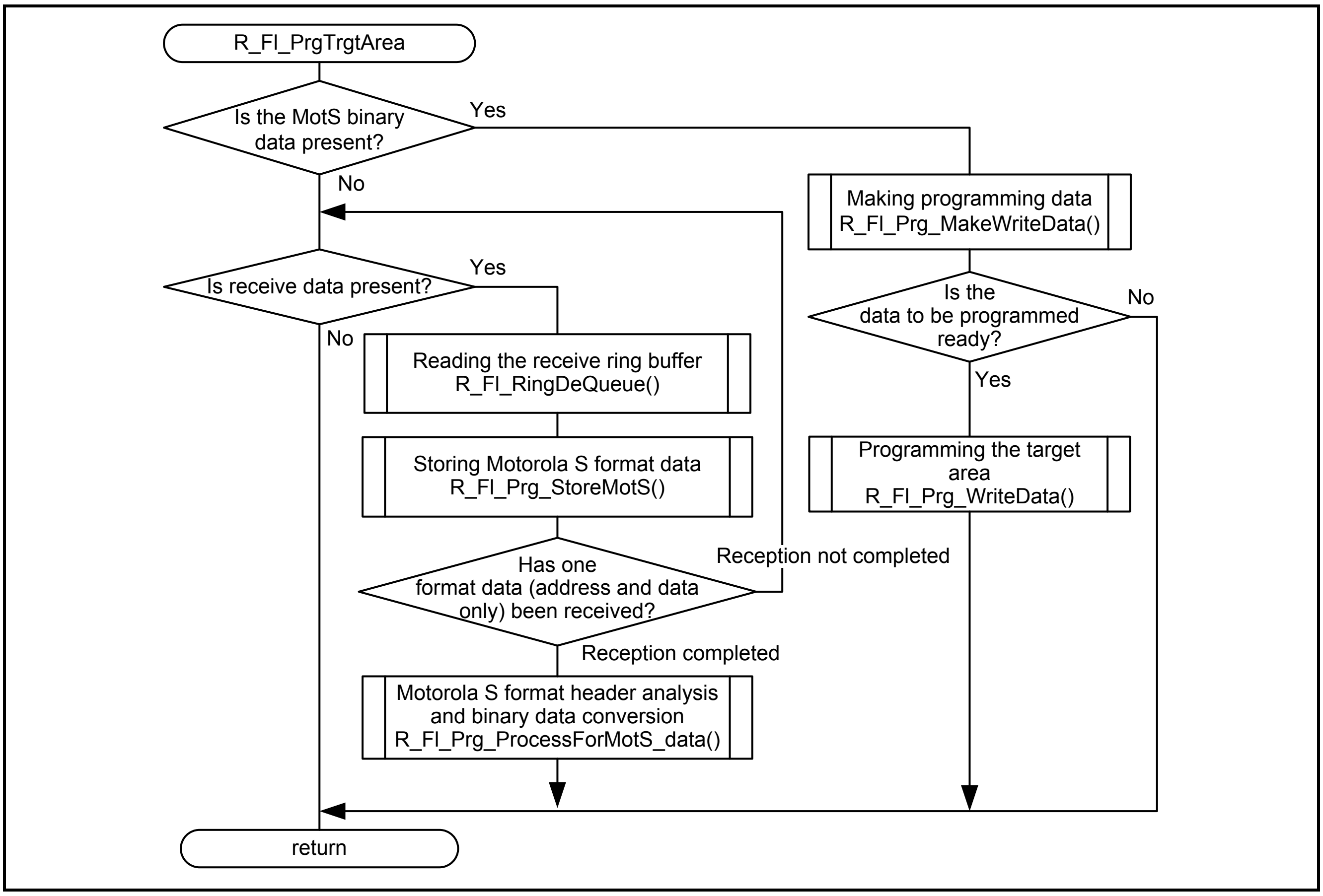

Figure 5.15 Program Processing

5.13.7 Executing the Target Program

Figure 5.16 shows the Executing the Target Program.

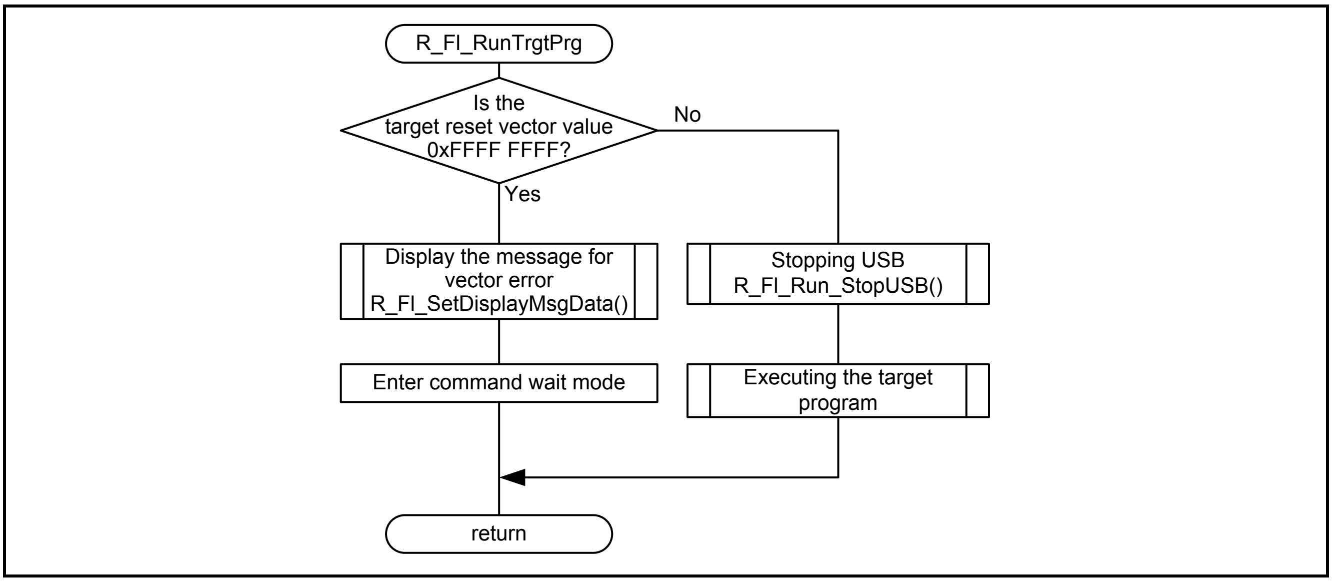

Figure 5.16 Executing the Target Program

5.13.8 Wait for Completion of an Error Processing

Figure 5.17 shows the Wait for Completion of an Error Processing.

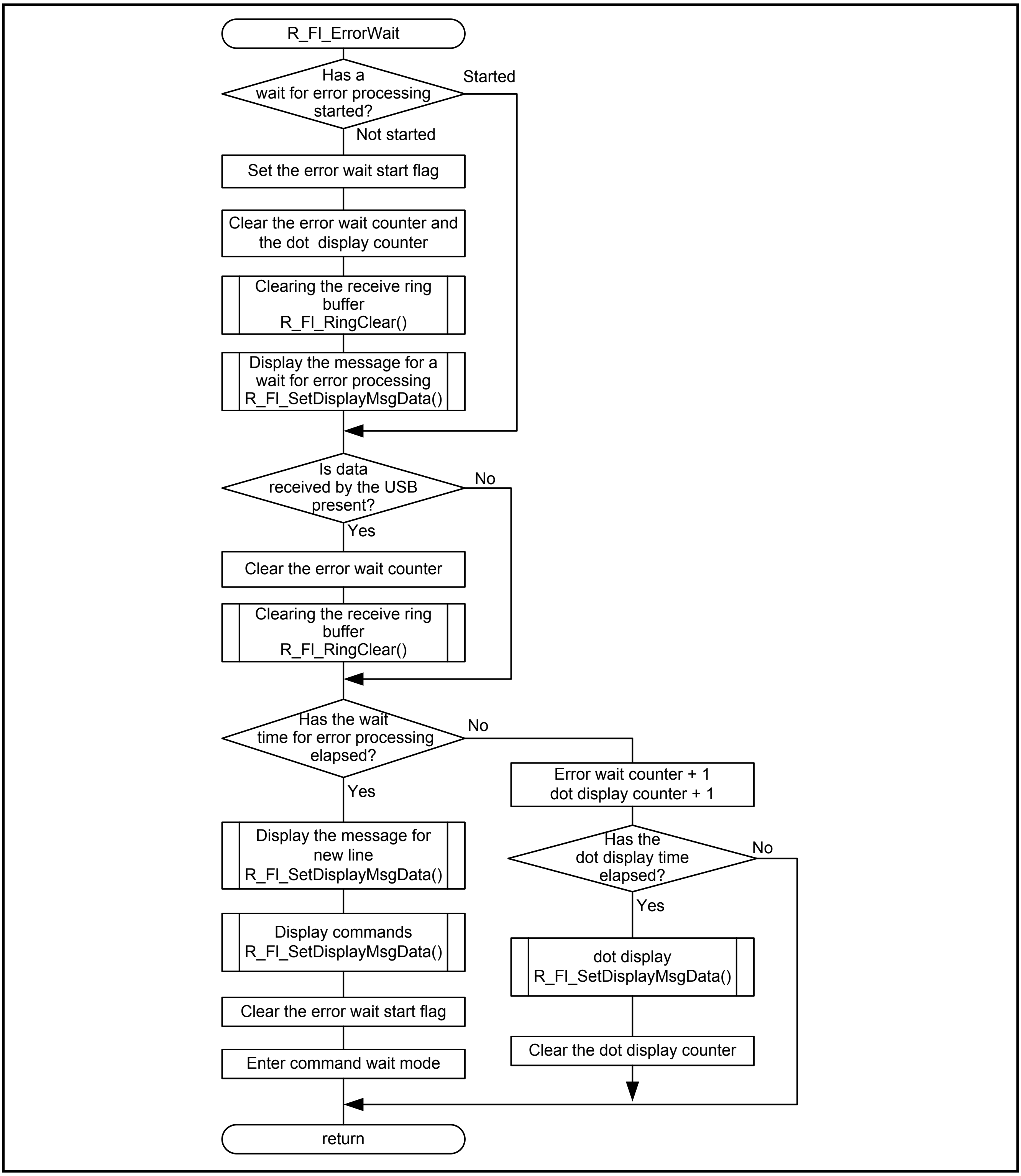

Figure 5.17 Wait for Completion of an Error Processing

5.13.9 Displaying Commands

Figure 5.18 shows the Displaying Commands.

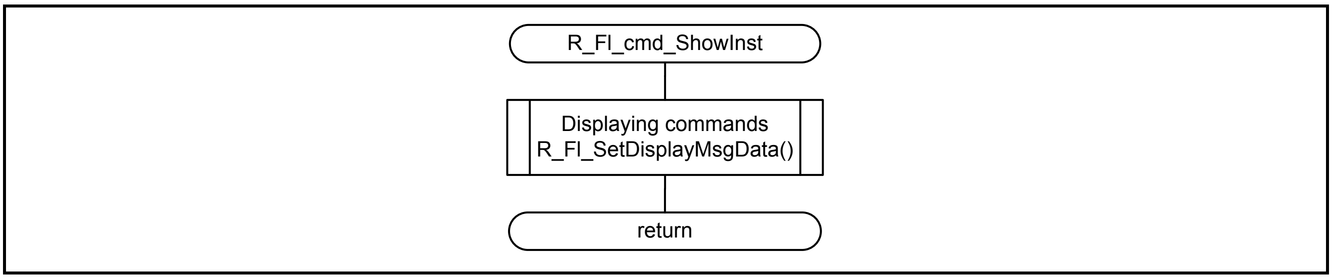

Figure 5.18 Displaying Commands

5.13.10 Start Processing for Blank Check

Figure 5.19 shows the Start Processing for Blank Check.

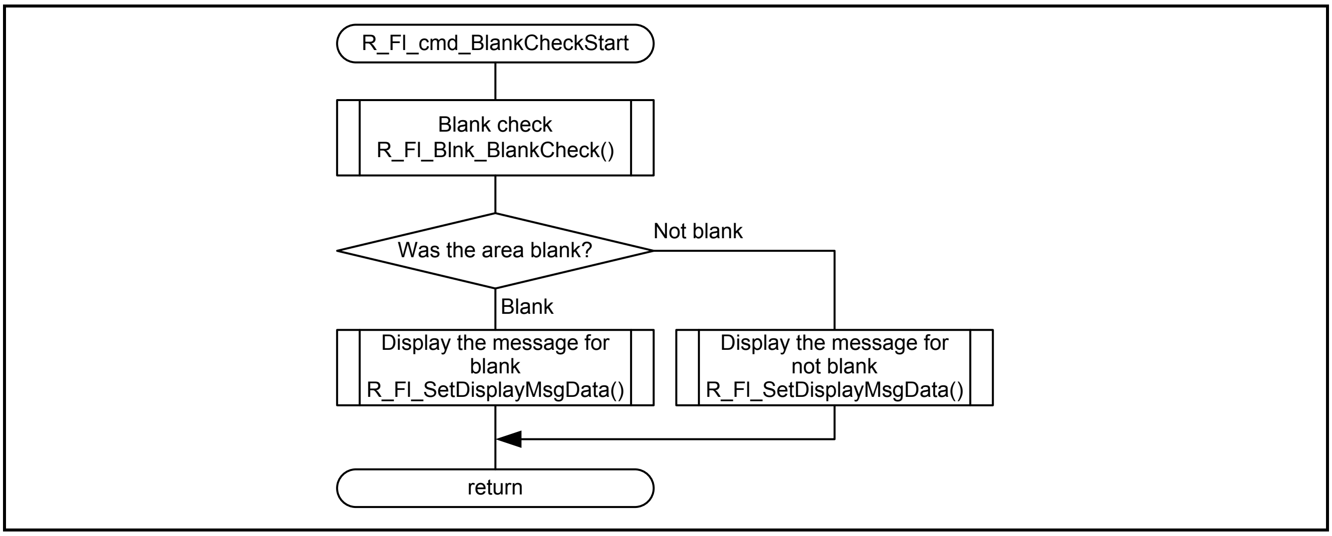

Figure 5.19 Start Processing for Blank Check

5.13.11 Start Processing for Erasing

Figure 5.20 shows the Start Processing for Erasing.

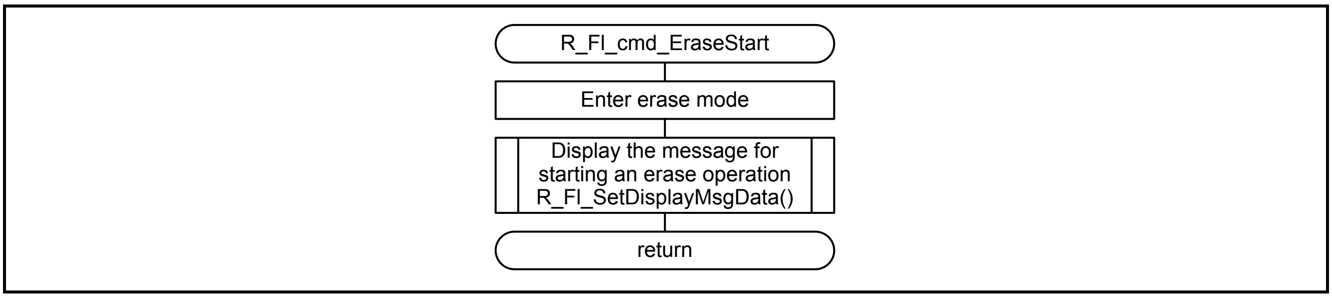

Figure 5.20 Start Processing for Erasing

5.13.12 Start Processing for Programming

Figure 5.21 shows the Start Processing for Programming.

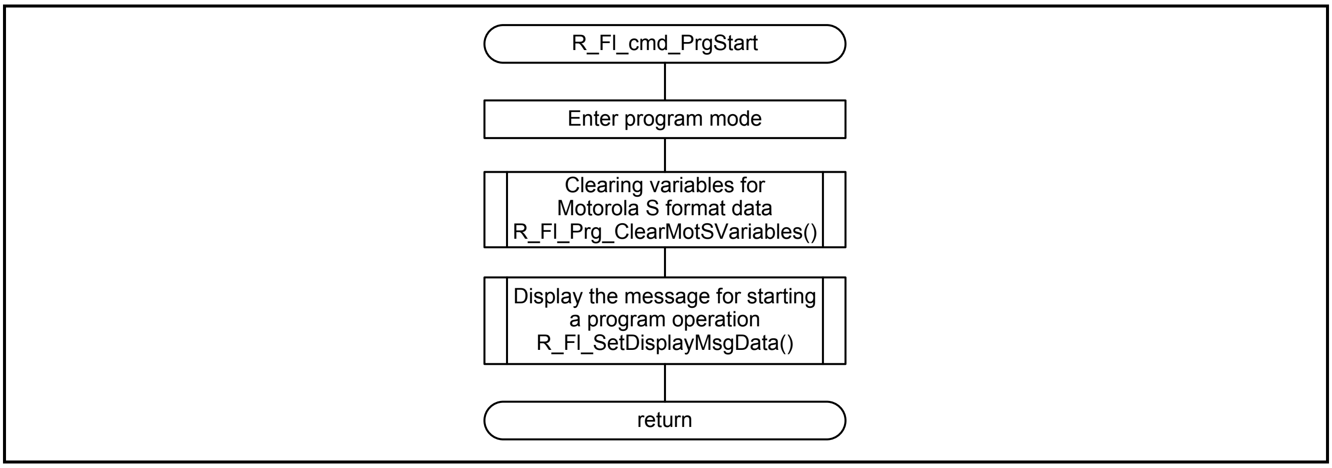

Figure 5.21 Start Processing for Programming

5.13.13 Executing the Target Program

Figure 5.22 shows the Executing the Target Program.

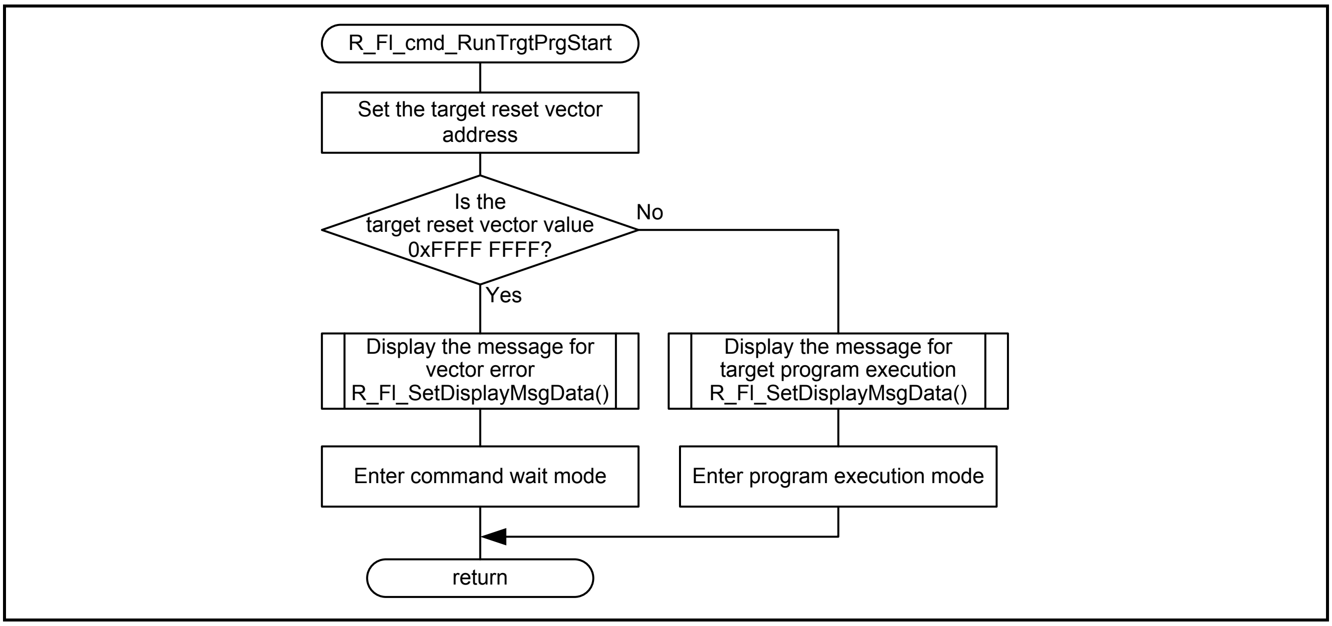

Figure 5.22 Executing the Target Program

5.13.14 Blank Check

Figure 5.23 shows the Blank Check.

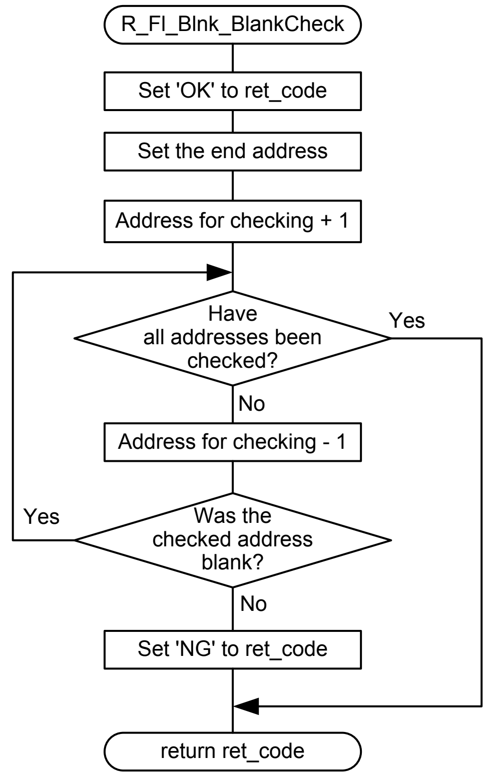

Figure 5.23 Blank Check

5.13.15 Erasing the Target Area

Figure 5.24 shows the Erasing the Target Area.

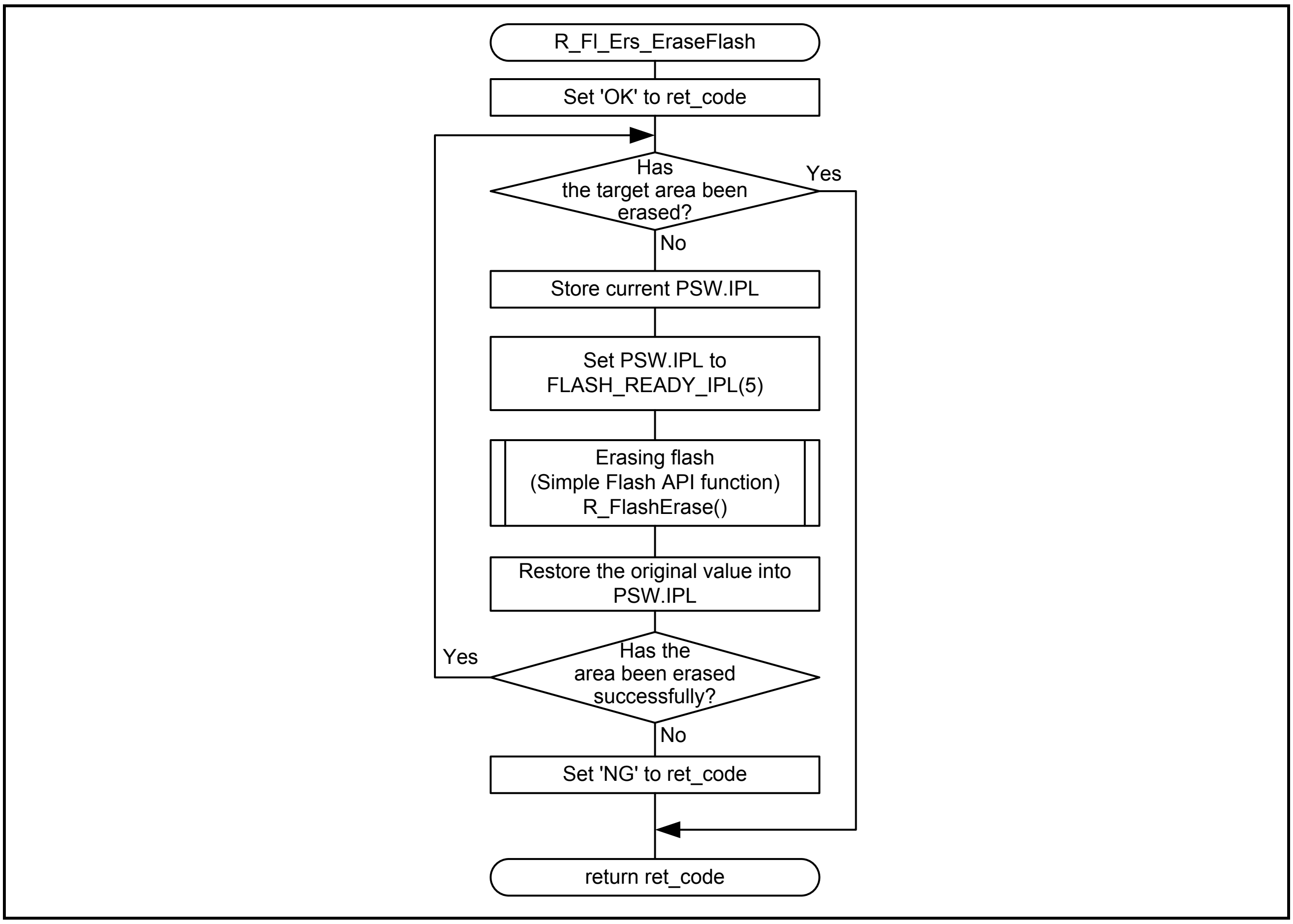

Figure 5.24 Erasing the Target Area

5.13.16 Storing Motorola S Format Data

Figure 5.25 shows the Storing Motorola S Format Data.

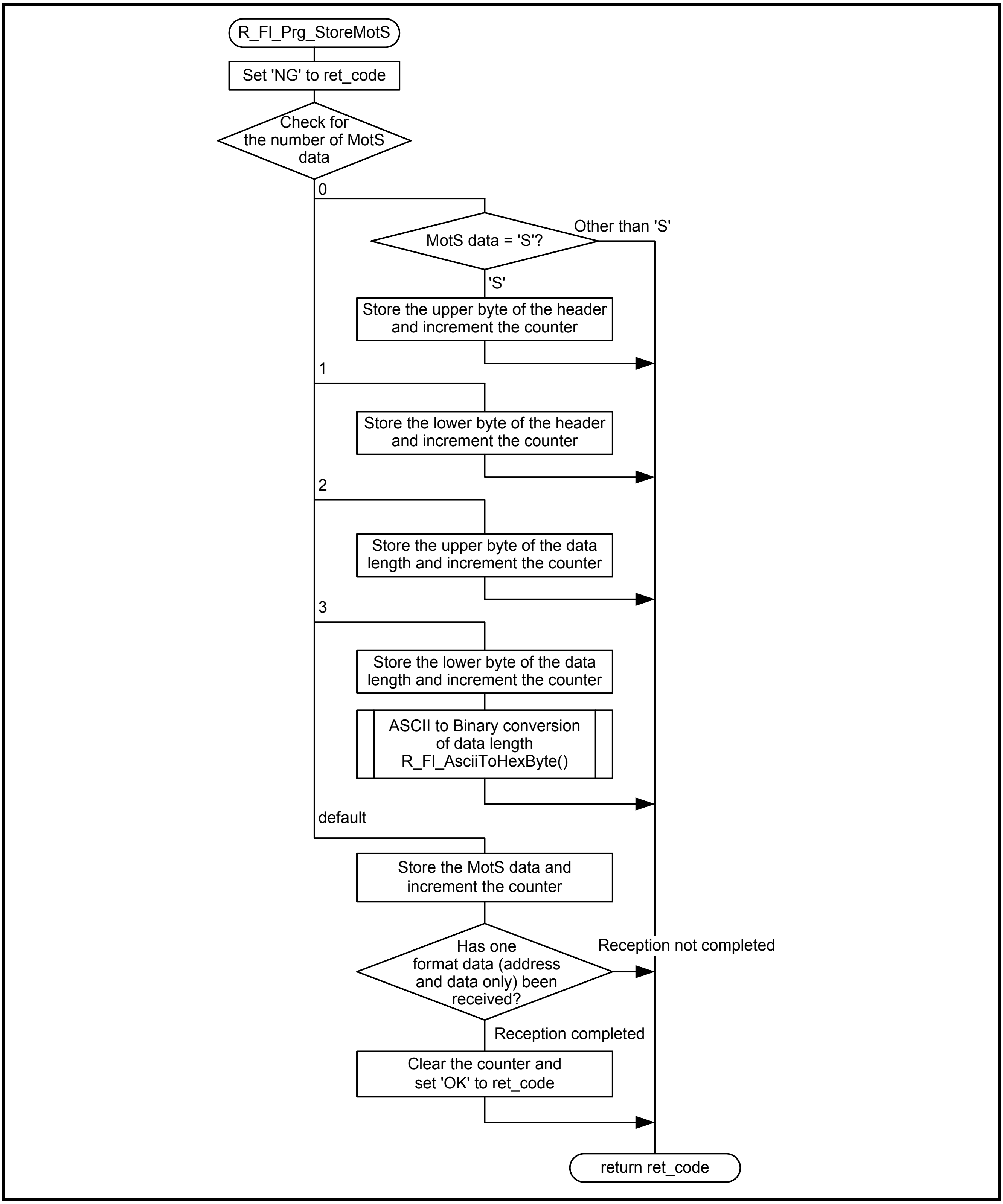

Figure 5.25 Storing Motorola S Format Data

5.13.17 Motorola S Format Header Analysis and Binary Data Conversion

Figure 5.26 and Figure 5.27 show the Motorola S Format Header Analysis and Binary Data Conversion.

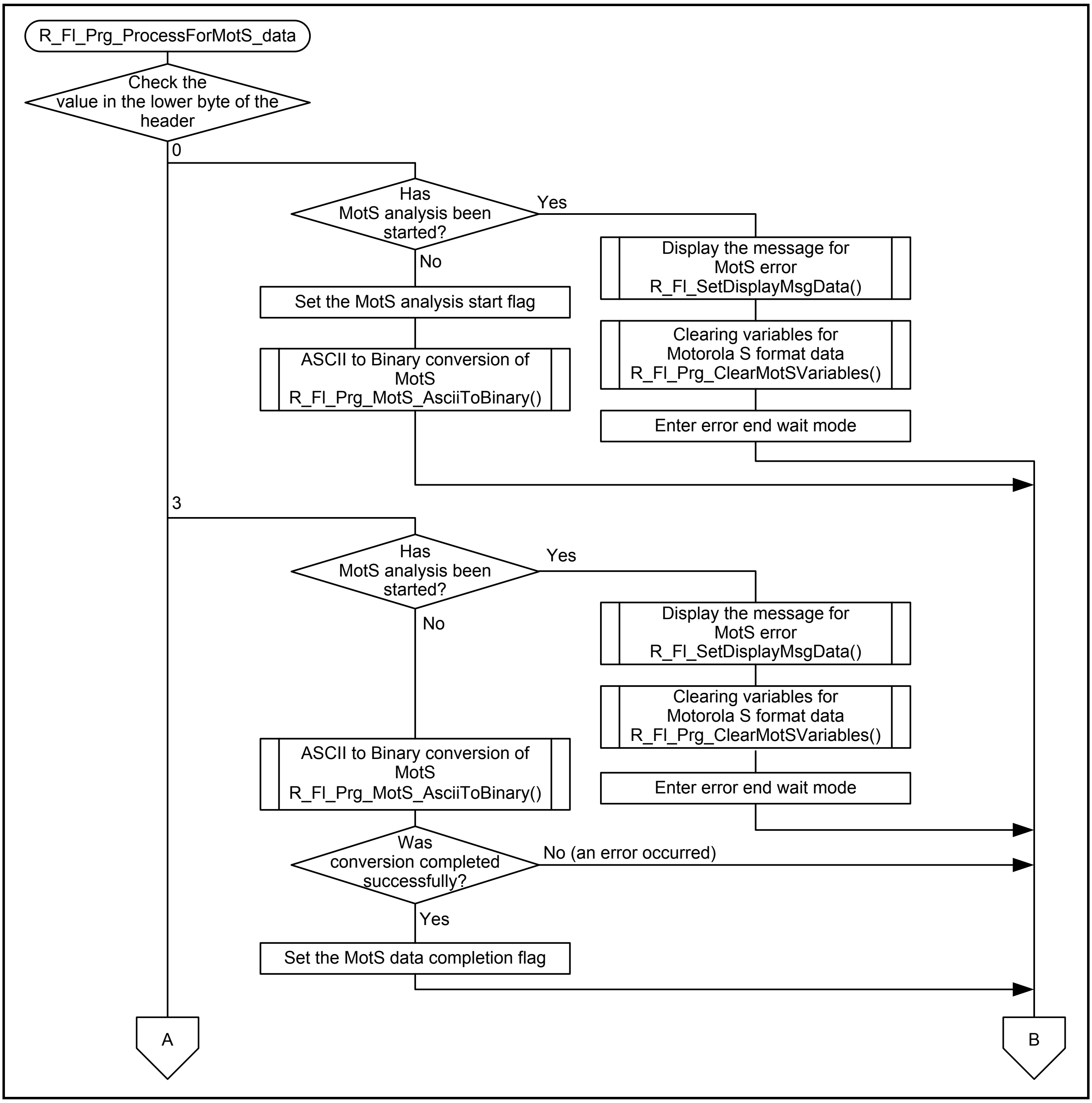

Figure 5.26 Motorola S Format Header Analysis and Binary Data Conversion (1/2)

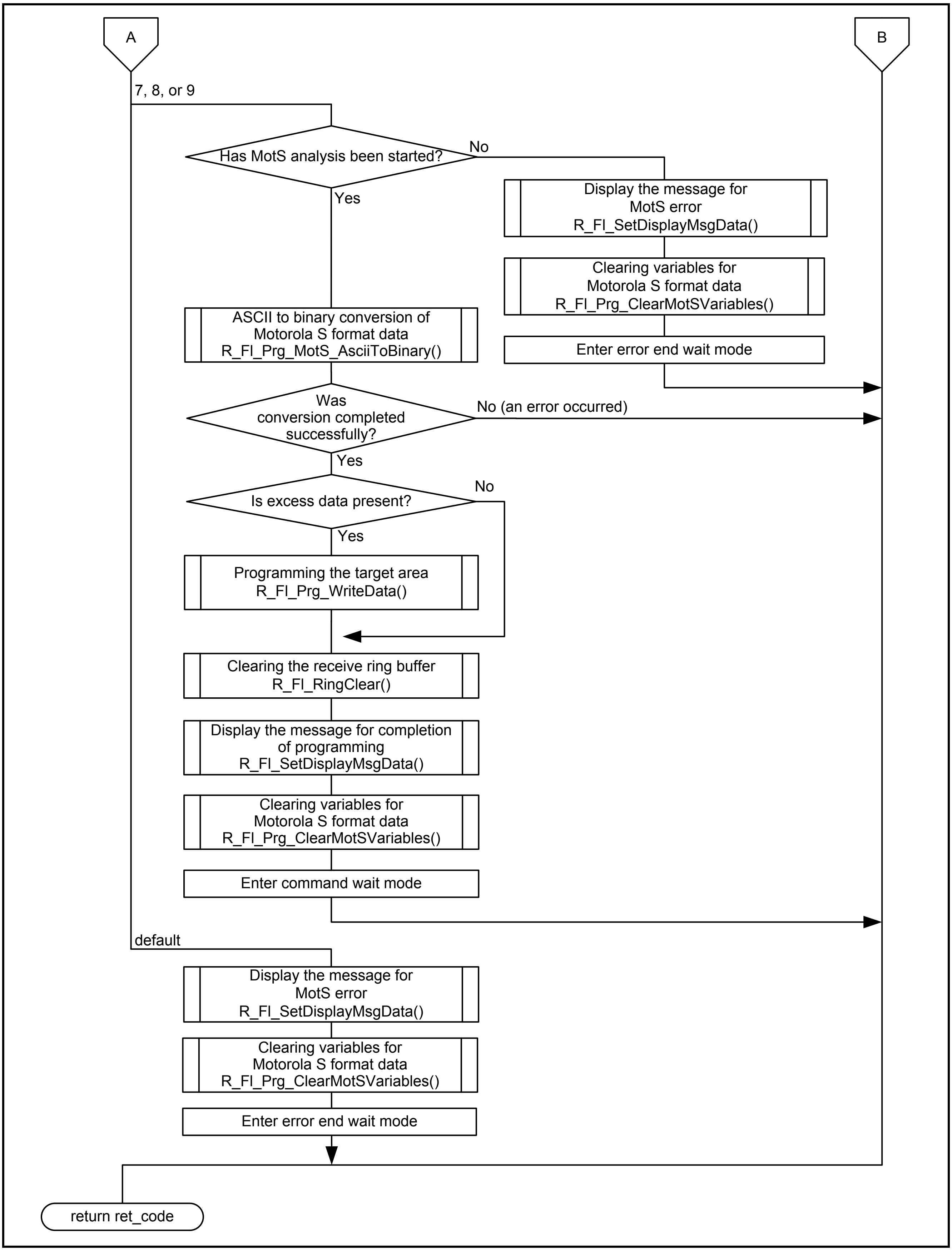

Figure 5.27 Motorola S Format Header Analysis and Binary Data Conversion (2/2)

5.13.18 ASCII to Binary Conversion of Motorola S Format Data

Figure 5.28 shows the ASCII to Binary Conversion of Motorola S Format Data.

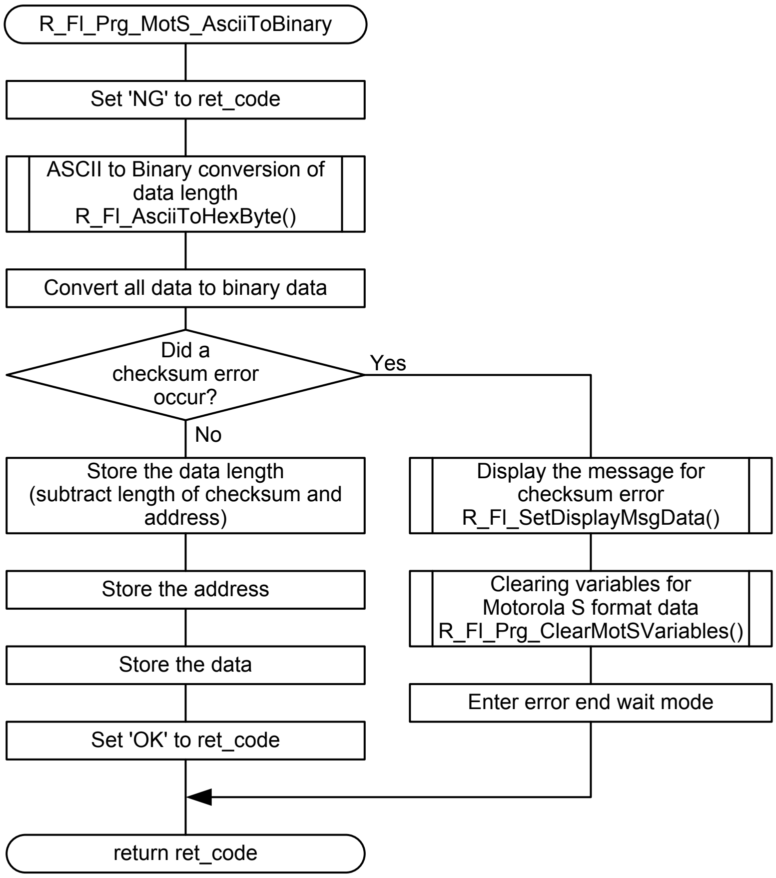

Figure 5.28 ASCII to Binary Conversion of Motorola S Format Data

5.13.19 Making Programming Data

Figure 5.29 shows the Making Programming Data.

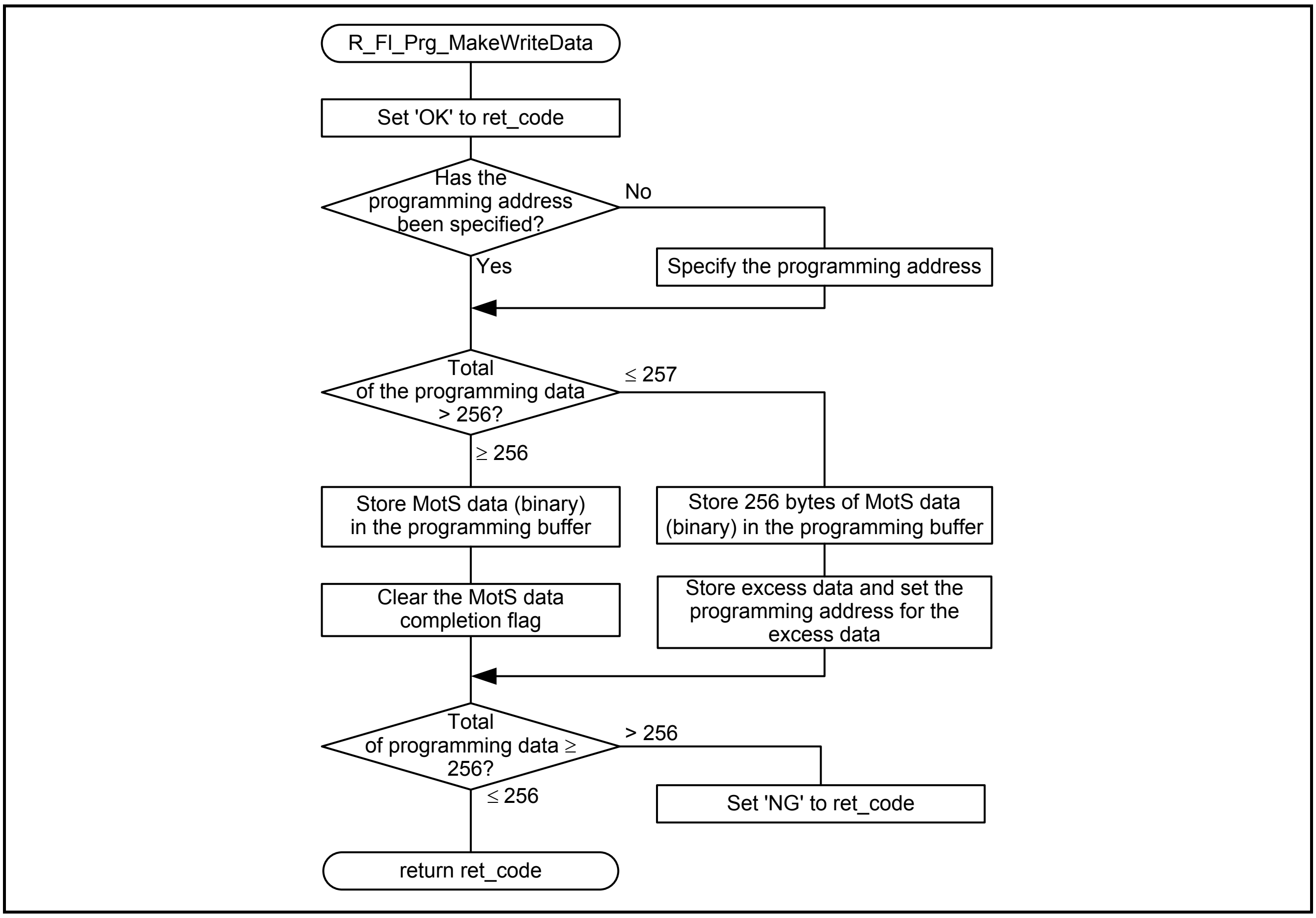

Figure 5.29 Making Programming Data

5.13.20 Programming the Target Area

Figure 5.30 show the Programming the Target Area.

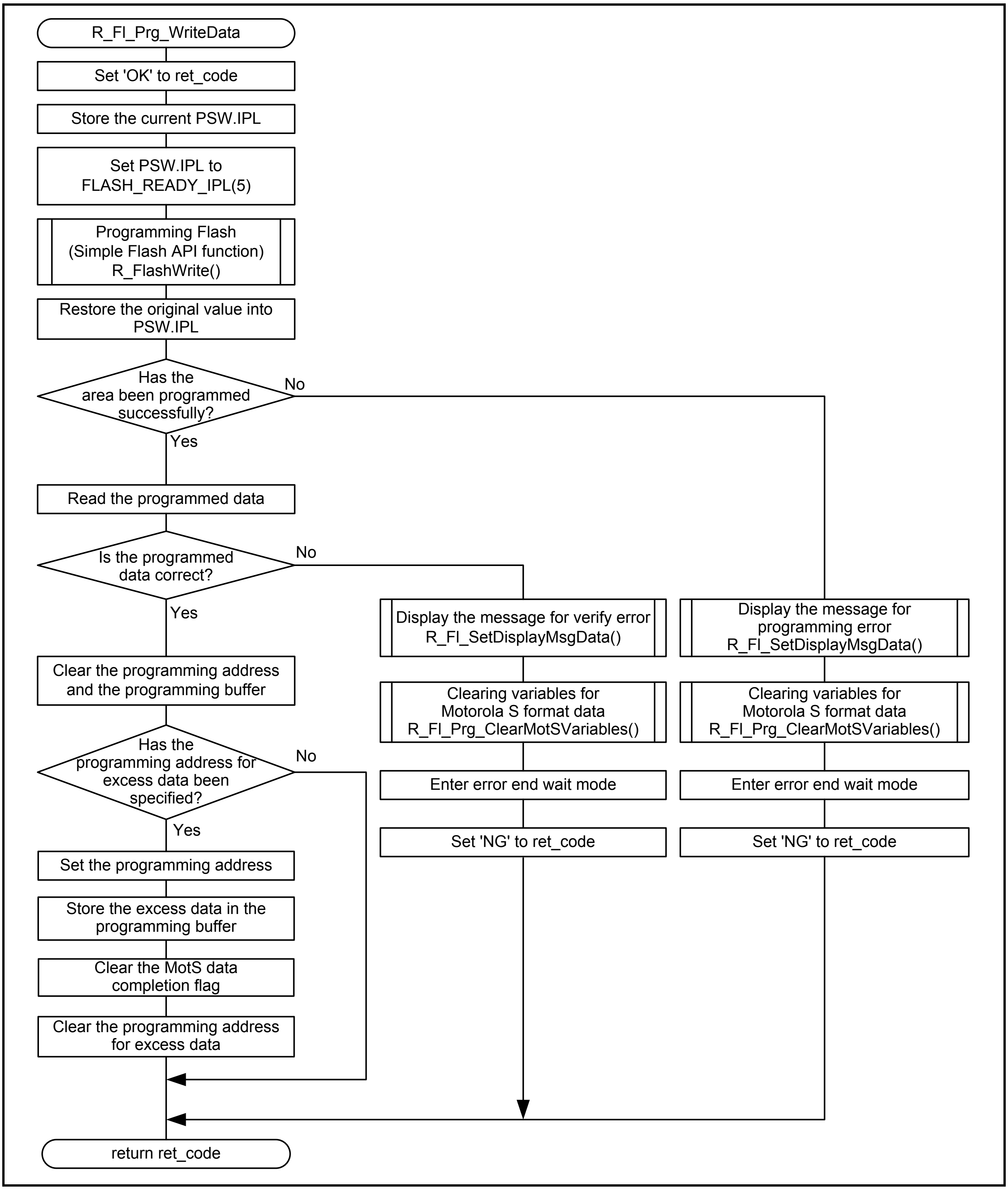

Figure 5.30 Programming the Target Area

5.13.21 Clearing Variables for Motorola S Format Data

Figure 5.31 shows the Clearing Variables for Motorola S Format Data.

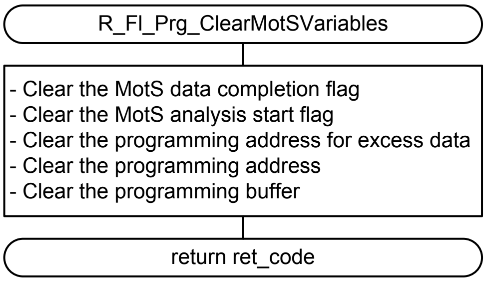

Figure 5.31 Clearing Variables for Motorola S Format Data

5.13.22 Stopping USB

Figure 5.32 shows the Stopping USB.

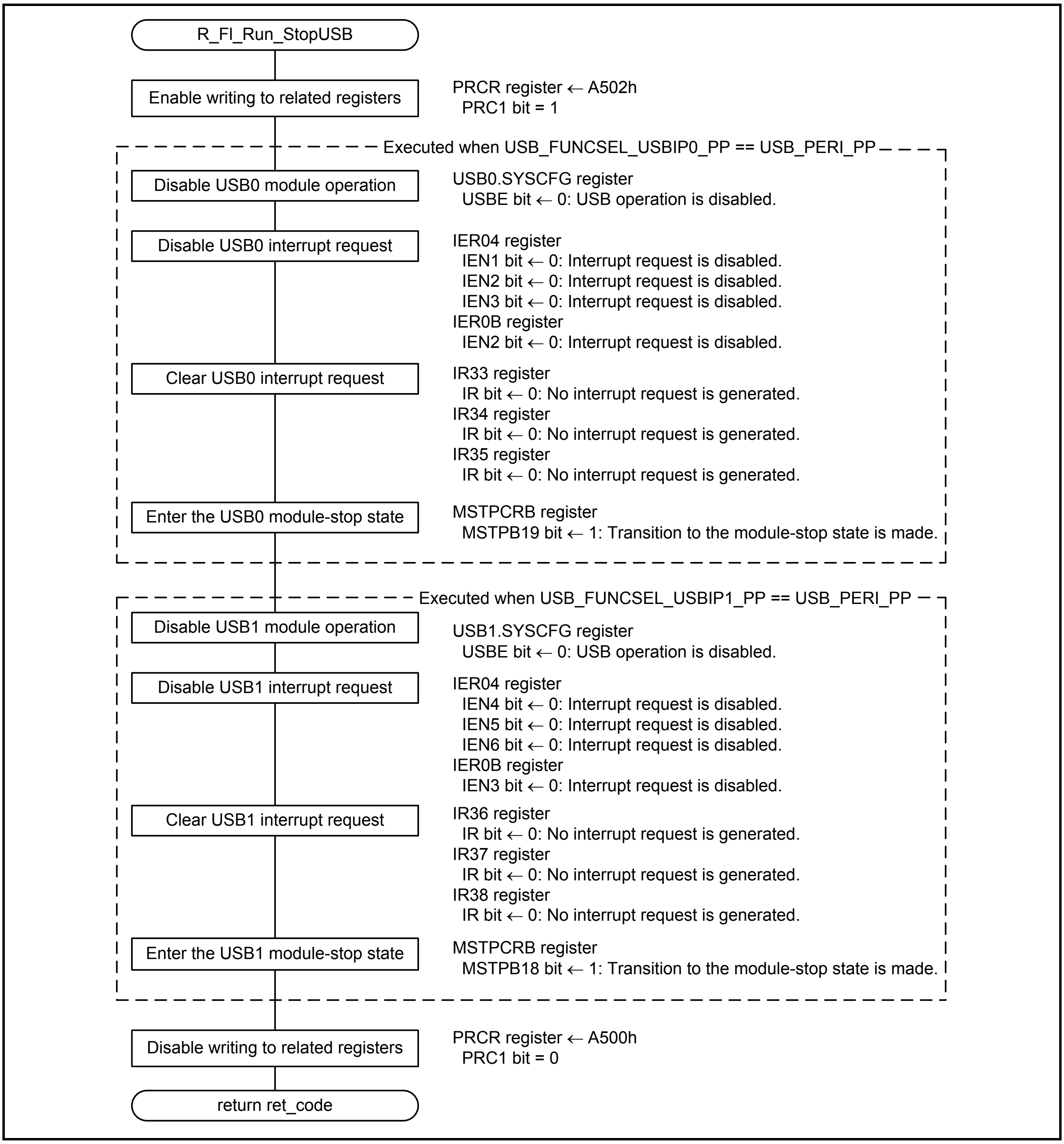

Figure 5.32 Stopping USB

5.13.23 Storing USB Receive Data

Figure 5.33 shows the Storing USB Receive Data.

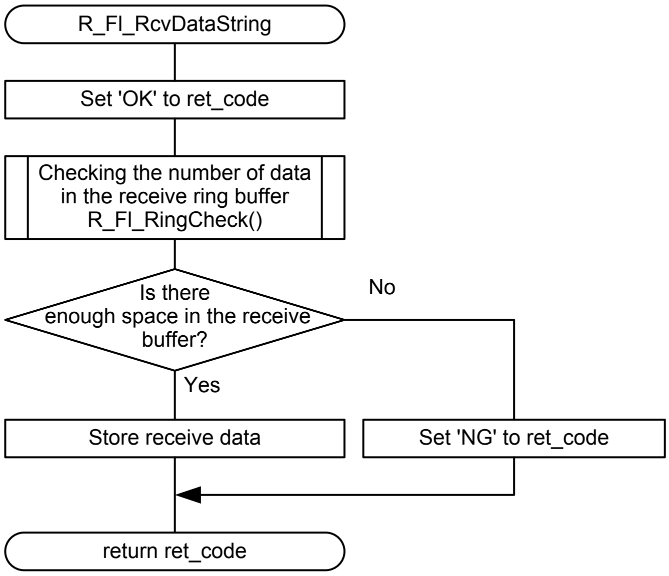

Figure 5.33 Storing USB Receive Data

5.13.24 Storing USB Transmit Data

Figure 5.34 shows the Storing USB Transmit Data.

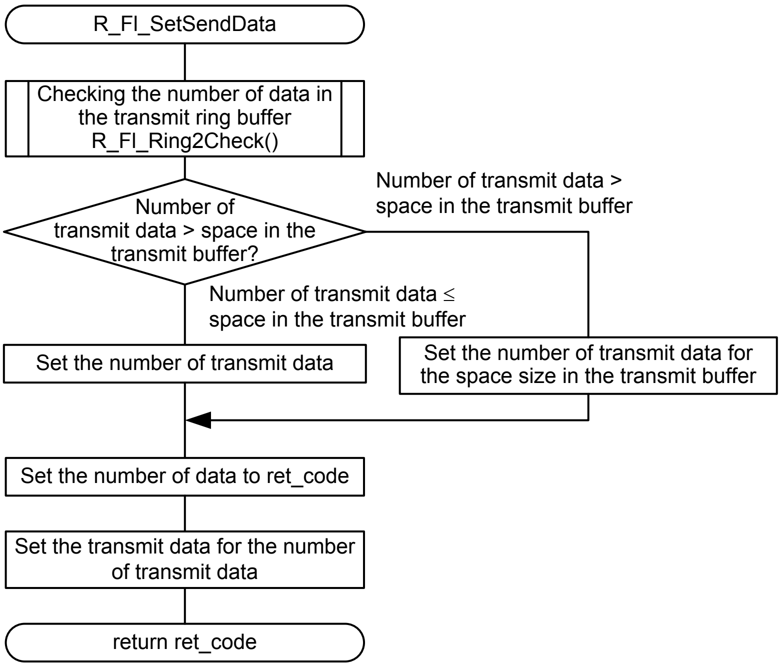

Figure 5.34 Storing USB Transmit Data

5.13.25 Setting Message Data to be Displayed

Figure 5.35 shows the Setting Message Data to be Displayed.

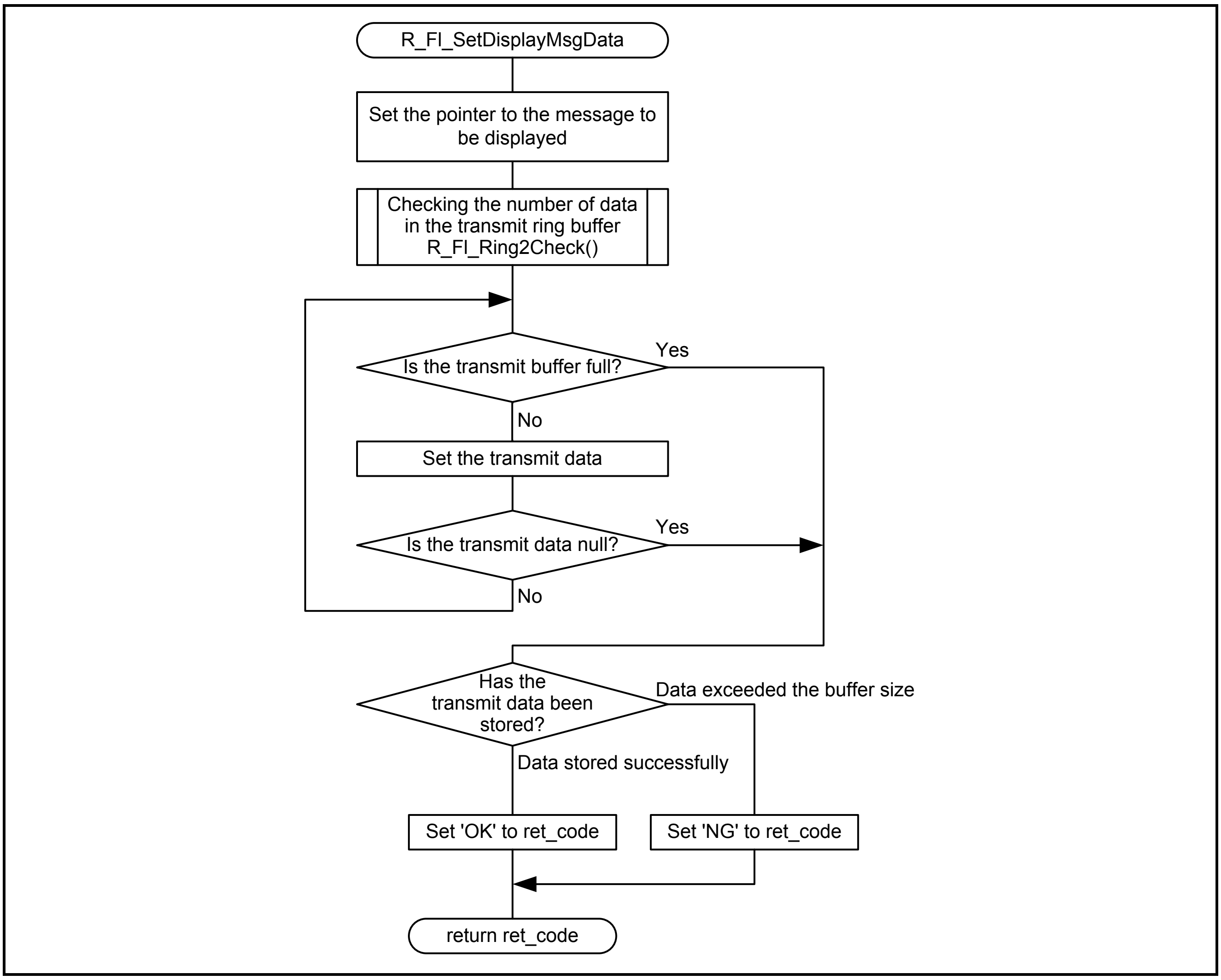

Figure 5.35 Setting Message Data to be Displayed

5.13.26 Checking Receive Ring Buffer Space

Figure 5.36 shows the Checking Receive Ring Buffer Space.

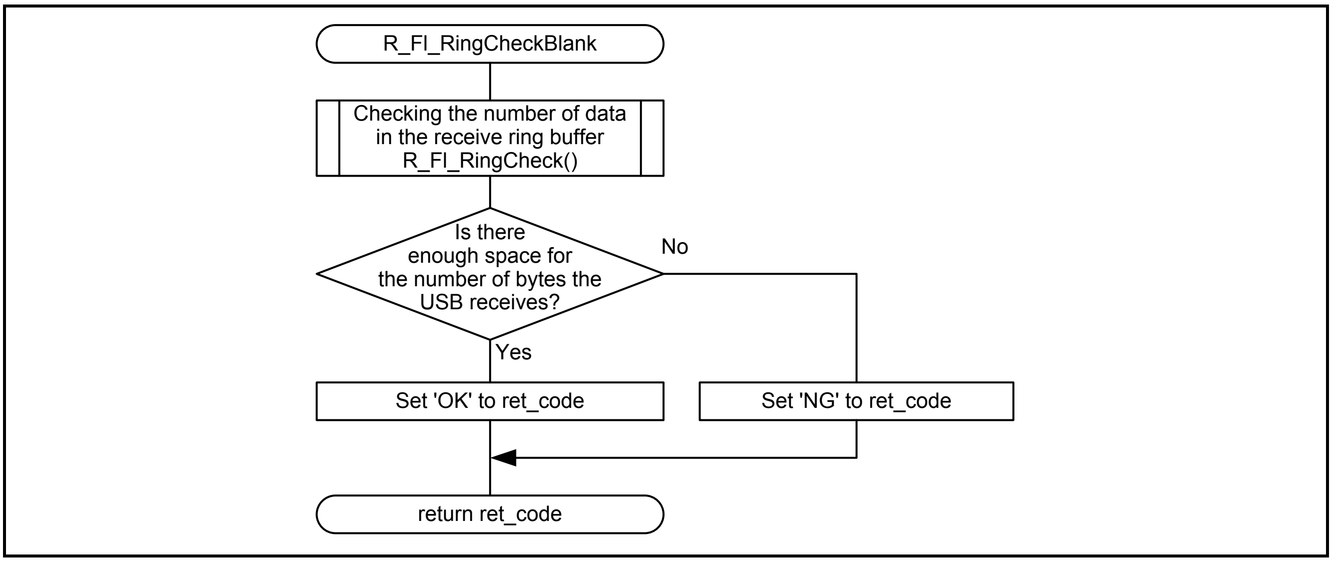

Figure 5.36 Checking Receive Ring Buffer Space

5.13.27 Checking Data in the Transmit Ring Buffer

Figure 5.37 shows the Checking Data in the Transmit Ring Buffer.

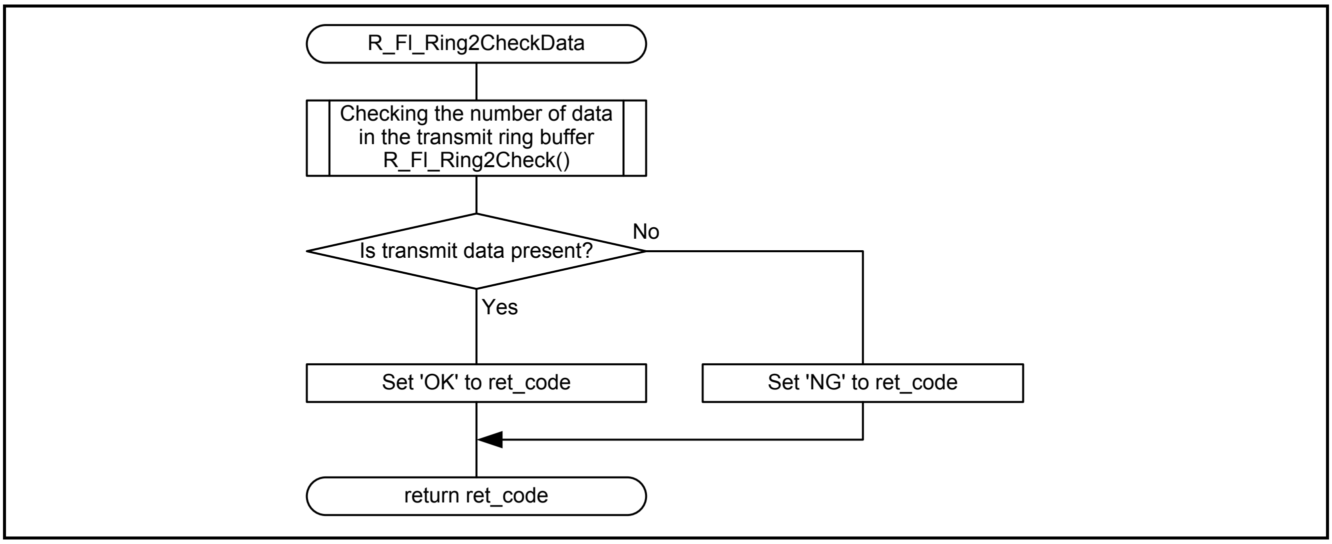

Figure 5.37 Checking Data in the Transmit Ring Buffer

5.13.28 Executing the Target Program with the USB Disconnected

Figure 5.38 shows the Executing the Target Program with the USB Disconnected.

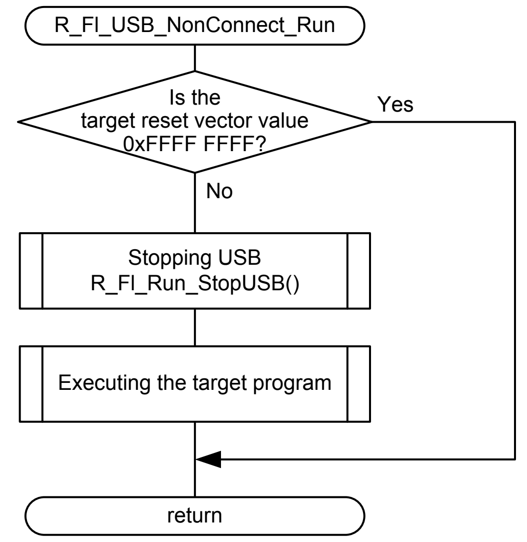

Figure 5.38 Executing the Target Program with the USB Disconnected

5.13.29 Storing Data in the Receive Ring Buffer

Figure 5.39 shows the Storing Data in the Receive Ring Buffer.

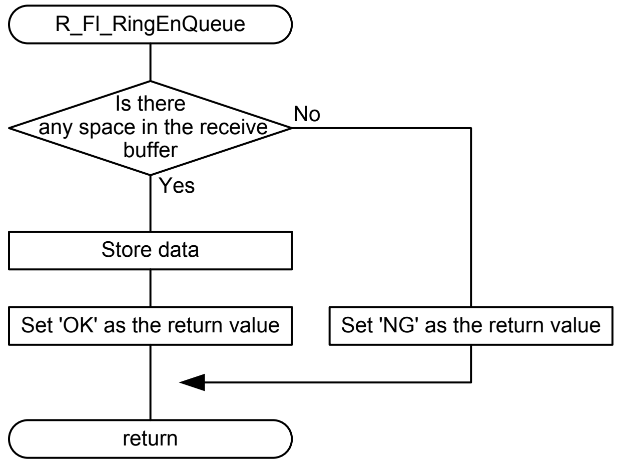

Figure 5.39 Storing Data in the Receive Ring Buffer

5.13.30 Reading the Receive Ring Buffer

Figure 5.40 shows the Reading the Receive Ring Buffer.

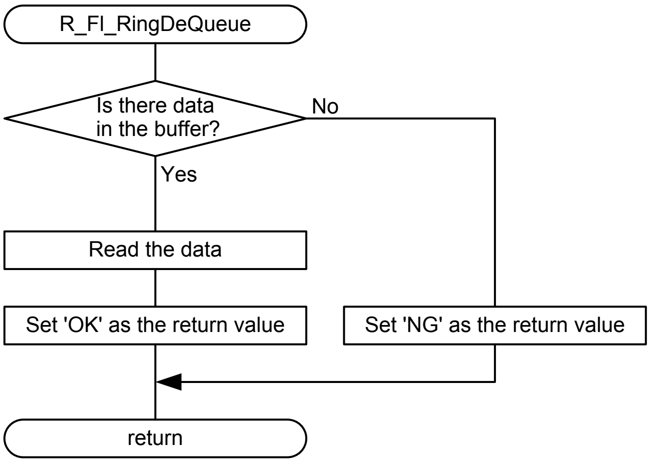

Figure 5.40 Reading the Receive Ring Buffer

5.13.31 Clearing the Receive Ring Buffer

Figure 5.41 shows the Clearing the Receive Ring Buffer.

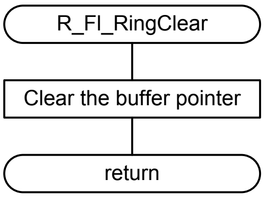

Figure 5.41 Clearing the Receive Ring Buffer

5.13.32 Checking the Number of Data in the Receive Ring Buffer

Figure 5.42 shows the Checking the Number of Data in the Receive Ring Buffer.

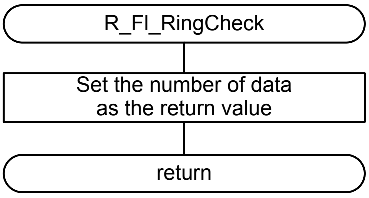

Figure 5.42 Checking the Number of Data in the Receive Ring Buffer

5.13.33 Storing Data in the Transmit Ring Buffer

Figure 5.43 shows the Storing Data in the Transmit Ring Buffer.

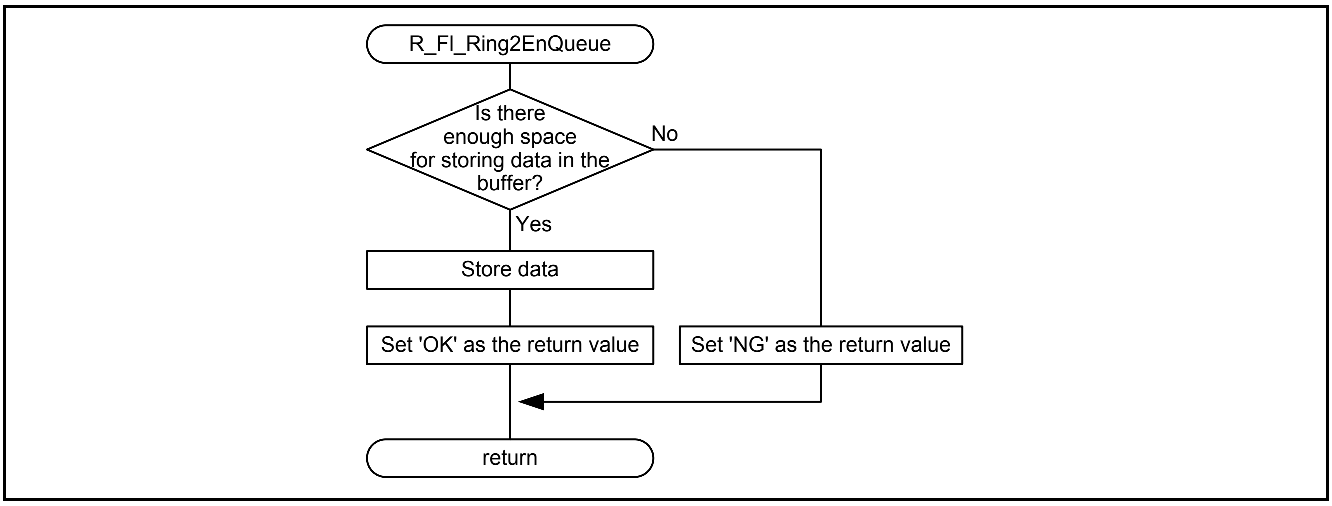

Figure 5.43 Storing Data in the Transmit Ring Buffer

5.13.34 Reading the Transmit Ring Buffer

Figure 5.44 shows the Reading the Transmit Ring Buffer.

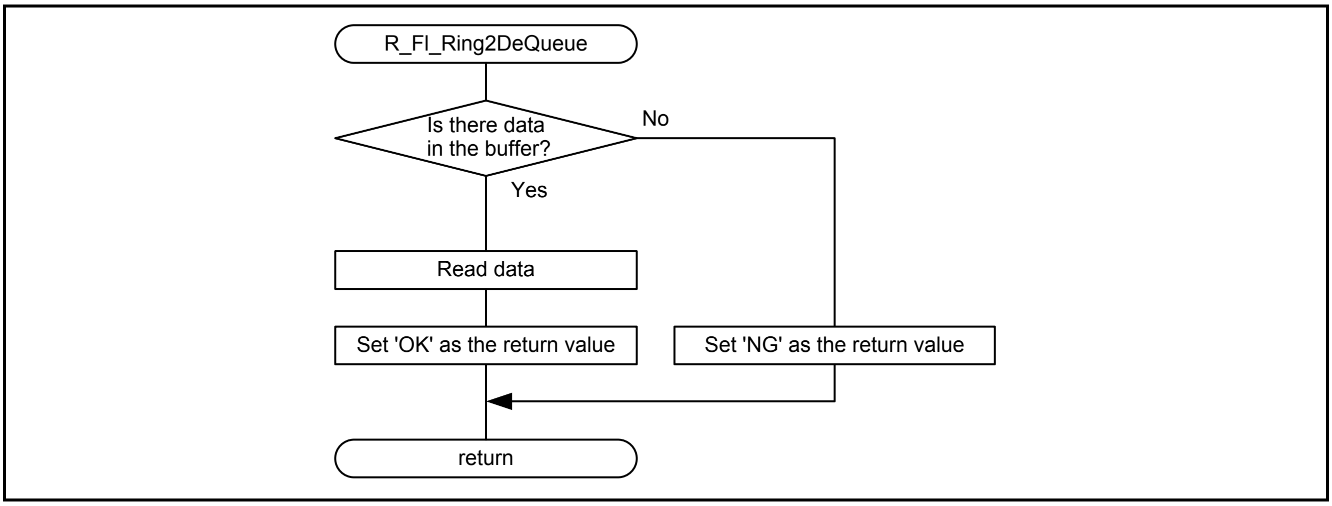

Figure 5.44 Reading the Transmit Ring Buffer

5.13.35 Clearing the Transmit Ring Buffer

Figure 5.45 shows the Clearing the Transmit Ring Buffer.

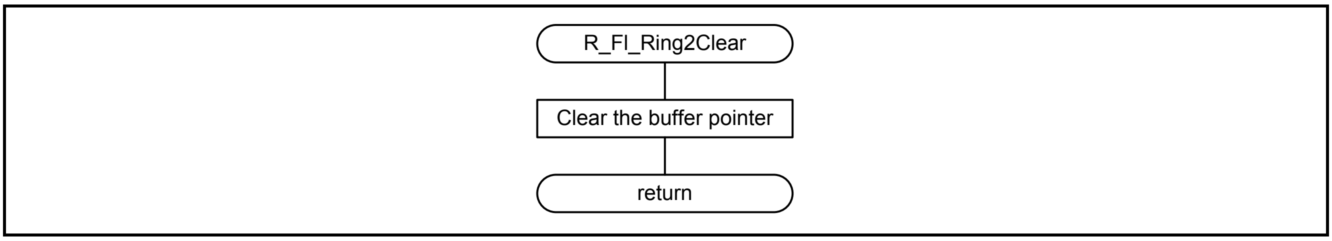

Figure 5.45 Clearing the Transmit Ring Buffer

5.13.36 Checking the Number of Data in the Transmit Ring Buffer

Figure 5.46 shows the Checking the Number of Data in the Transmit Ring Buffer.

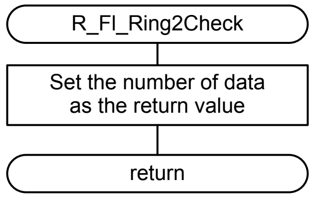

Figure 5.46 Checking the Number of Data in the Transmit Ring Buffer

5.13.37 ASCII to Binary Conversion

Figure 5.47 shows the ASCII to Binary Conversion.

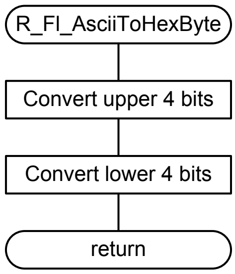

Figure 5.47 ASCII to Binary Conversion

6. Using the Sample Code

(1) Change the vendor ID and product ID for the sample code.
Change USB_VENDORID and USB_PRODUCTID in the r_usb_pcdc_descriptor.c file appropriate to your environment. Refer to the USB Peripheral Communication Device Class Driver and USB Basic Firmware application notes for details.

```
/******************************************************************************
User define macro definitions
******************************************************************************/
#define USB_VENDORID    0x0000
#define USB_PRODUCTID   0x0002
```

Figure 6.1 Vendor ID and Product ID in r_usb_pcdc_descriptor.c for the Sample Code

(2) Change the vendor ID and product ID for the USB driver.
Change VID_0000 and PID_0002 in the CDC_Demo.inf or CDC_Demo_Win7.inf file in the usb_driver folder. The changed values must be the same as the vendor ID and product ID for the sample code set in step (1) above. Refer to the USB Peripheral Communication Device Class Driver and USB Basic Firmware application notes for details.

```
[Manufacturer]
%STRING_MAUNUFACTURER%=Model

[Model]
%STRING_MODEL%=CDC, USB\VID_0000&PID_0002
```

Figure 6.2 Vendor ID and Product ID in CDC_Demo.inf for the USB Driver

```
[DeviceList]
%DESCRIPTION%=DriverInstall, USB\VID_0000&PID_0002

[DeviceList.NTamd64]
%DESCRIPTION%=DriverInstall, USB\VID_0000&PID_0002
```

Figure 6.3 Vendor ID and Product ID in CDC_Demo_Win7.inf for the USB Driver

(3) Build all codes of the sample code and program the codes into the MCU.

(4) After the MCU is powered on, connect the PC and MCU via the USB. (1)

(5) In the Windows installation screen for the USB driver, select the CDC_Demo.inf or CDC_Demo_Win7.inf file in the usb_driver folder to install the driver. If the USB driver is already installed, this step can be skipped.

(6) Start the terminal application to start communication with the MCU.

(7) After step (6), follow the message displayed in the terminal application to continue the operation.

Note:

1. If a value other than FFFF FFFFh is written to the target reset vector, and the USB interface remains disconnected for a certain time, the target program will be executed. Note that the USB will be stopped when the target program is executed. Write FFFF FFFFh to the target reset vector when the USB driver is installed since it may take some time to connect the USB interface.

7. Sample Target Program

A sample target program (UsrPrgSample.zip) is provided with this application note. In the sample target program, LEDs on the board listed on "2. Operation Confirmation Conditions" are turned on in order. Use this program as a reference when setting the target reset vector and sections. This program assumes an operation with 1 MB ROM.

8. Notes on Using This Application Note

8.1 Programming Speed

Programming speed may be extremely slow depending on the terminal application used. If this is the case, try another terminal application.

8.2 USB Disconnection During Programming or Erasing

Do not disconnect or reconnect the USB interface during programming or erasing the target area.

8.3 HEW Configuration

When reprogramming the flash, the sample code on the ROM is transferred to the RAM and executed. For details on settings regarding the sample code operations, refer to 2.10 “Adding Middleware to Your Project” and 2.12 “Putting Flash API Code in RAM” in the RX600 & RX200 Series Simple Flash API for RX application note.

8.4 Vender ID and Product ID for the USB

When using the sample code, the vender ID and product ID for the USB need to be changed. Refer to 6. “Using the Sample Code” in this application note, and the USB Peripheral Communication Device Class Driver and USB Basic Firmware application notes.

8.5 Interrupts in the Fixed Vector Table

The sample code only defines the reset vector from interrupts allocated to the fixed vector table. If the other interrupts in the fixed vector table are necessary, change the sample code appropriate to the user program.

8.6 Reset Vector of the Target Program

The start address of the target program, which is programmed by the sample code, is specified with the value in the target reset vector (FFFD FFFCh). Therefore the reset vector of the target program needs to be FFFD FFFCh. Refer to 5.3 “Start Address of the Target Program” for details on the start address, and refer to 7. “Sample Target Program” for details on the target program.

8.7 Motorola S Format

The sample code only supports S0, S3, and S7 of the Motorola S formats. Also the order of addresses must be in ascending order. Do not transmit a mot file with addresses that are in descending order or mixed order.

8.8 Processing with the while(1) Statement

In the sample code, when the transmit ring buffer overflows, the while(1) statement is used for deadlock.

8.9 Stop of the Program During USB Communication

If the MCU is reset while the MCU is connected to the terminal application on the PC, and if the MCU is restarted, communication may not be performed correctly. In this case, exit the terminal application and then connect the MCU to the PC again.

8.10 Endian

This sample code only supports the little endian.

8.11 Changes in the Simple Flash API for RX

The sample code uses the Simple Flash API program with some changes. Refer to the RX600 & RX200 Series Simple Flash API for RX application note for details on the specifications of the Simple Flash API.

Files r_flash_api_rx600_config.h and r_bsp_config.h are changed in the Simple Flash API for this application note.

- Changes in r_flash_api_rx600_config.h:
  1. The processor interrupt priority level (IPL) of the processor status word (PSW) is changed to the value specified with the macro definition shown below to prevent ROM access due to interrupts during programming and erasing the flash. The value is set to 5 in the application note.

     Macro definition: #define FLASH_READY_IPL 5
  2. Settings for the Simple Flash API are changed as follows:

     Before: //#define FLASH_API_RX_CFG_FLASH_TO_FLASH

     #define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

     #define FLASH_API_RX_CFG_COPY_CODE_BY_API

     After: #define FLASH_API_RX_CFG_FLASH_TO_FLASH

     //#define FLASH_API_RX_CFG_IGNORE_LOCK_BITS

     //#define FLASH_API_RX_CFG_COPY_CODE_BY_API
- Changes in r_bsp_config.h:
  1. Files stored in the r_bsp/board/rskrx63n of the Simple Flash API are used.
  2. Settings for the Simple Flash API are changed as follows:

     Before: #define BSP_CFG_PCKA_DIV (4)

     #define BSP_CFG_IEBCK_DIV (8)

     #define BSP_CFG_BCLK_OUTPUT (0)

     After: #define BSP_CFG_PCKA_DIV (2)

     #define BSP_CFG_IEBCK_DIV (2)

     #define BSP_CFG_BCLK_OUTPUT (2)

8.12 Changes in the USB PCDC Driver

The sample code uses programs that use the ANSI interface in the USB PCDC driver with some changes. For details on the specifications of the USB PCDC driver, refer to the USB PCDC Driver and USB Basic Firmware application notes.

8.12.1 Changed Items

Files r_usb_pcdc_apl.c, dbsct_pcdc.c, and rx_mcu.c are changed in the USB PCDC driver for this application note.

- Changes in r_usb_pcdc_apl.c:
  1. Include files have been added.

     Added: #include "r_Flash_main.h"

     #include " r_Flash_buff.h"
  2. Changes other than above are included in the section of #ifdef R_FLASH_USB.

- Changes in dbsct_pcdc.c:

  The changes are indicated with the comment "// Flash table".

- Changes in rx_mcu.c:
  1. Include files have been added.

     Added: #include "r_init_clock.h"

     #include "r_init_non_existent_port.h"

     #include "r_init_stop_module.h"
  2. The CPU initialization (usb_cpu_McuInitialize function) is changed to use the R_INIT_StopModule, R_INIT_NonExistentPort, and R_INIT_Clock functions in the RX63N Group, RX631 Group Initial Setting application note.

8.12.2 Additional Files

For details on files added to the USB PCDC driver, refer to 5.7 “File Composition”.

8.12.3 Additional Sections

Table 8.1 lists the Additional Sections.

Table 8.1 Additional Sections

Section Name	Description
B_flash_api_sec	Section for variables used in the flash reprogramming codes which operate in the RAM
R_flash_api_sec	
RPFRAM	Section for the flash reprogramming codes which operate in the RAM
TRGT_DMMY_FIXEDVECT	Section for the fixed vector of the target program

8.12.4 Include File Directory

Include file directories “WorkSpace\FLASH” and “WorkSpace\r_bsp” have been added.

8.12.5 Linker Settings

Settings for ROM to RAM mapping have been added to the Linker.

- ROM PFRAM is mapped to RPFRAM.

- ROM D_flash_api_sec is mapped to R_flash_api_sec.

8.13 Changes in the RX63N Group, RX631 Group Initial Setting

The sample code uses programs in the RX63N Group, RX631 Group initial setting with some changes. For details on the specifications of the initial setting, refer to the RX63N Group, RX631 Group Initial Setting application note.

The r_init_clock.c file is changed in the initial setting.

1. The BCLK division ratio is changed from divide-by-4 to divide-by-8.

   Before: SYSTEM.SCKCR.LONG = 0x21C21211;

   while (0x21C21211 != SYSTEM.SCKCR.LONG)

   After: SYSTEM.SCKCR.LONG = 0x21C31211;

   while (0x21C31211 != SYSTEM.SCKCR.LONG)

2. The USB clock setting is changed from ‘not used’ to divide-by-4.

   Before: SYSTEM.SCKCR2.WORD = 0x0012;

   After: SYSTEM.SCKCR2.WORD = 0x0032;

3. The BCLK pin output is changed from ‘no division’ to divide-by-2.

   Before: SYSTEM.BCKCR.BYTE = 0x00;

   while (0x00 != SYSTEM.BCKCR.BYTE)

   After: SYSTEM.BCKCR.BYTE = 0x01;

   while (0x01 != SYSTEM.BCKCR.BYTE)

9. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

10. Reference Documents

User's Manual: Hardware

RX63N Group, RX631 Group User's Manual: Hardware Rev.1.70 (R01UH0041EJ)
The latest version can be downloaded from the Renesas Electronics website.

Technical Update/Technical News

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

User's Manual: Development Tools

RX Family C/C++ Compiler Package V.1.01 User's Manual Rev.1.00 (R20UT0570EJ)
The latest version can be downloaded from the Renesas Electronics website.

Website and Support

Renesas Electronics website

http://www.renesas.com

Inquiries

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

REVISION HISTORY	RX63N Group, RX631 Group Application Note Flash Bootloader with the USB Peripheral CDC

Rev.	**Date**	**Description**	
		Page	**Summary**
1.00	Sep. 16, 2014	—	First edition issued

General Precautions in the Handling of MPU/MCU Products

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

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

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

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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

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

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

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

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

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

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

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

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

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

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

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

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

Colophon 4.0