RX62N Group, RX621 Group

USB Peripheral CDC Flash Boot Loader

R01AN0632EJ0101
Rev.1.01
Mar 07, 2013

Introduction

This application note describes a USB flash downloader (flash memory programmer) that uses the RX62N and RX621 Group microcontroller USB 2.0 host/function module, operates the USB function in single-chip mode, and programs the on-chip flash memory over a USB connection.

Note that this application note uses the sample code and drivers from the following application notes.

On-chip flash memory erase and write:

RX600 Series Simple Flash API for RX600 Rev. 2.20 (R01AN0544EU0220)

USB communication:

Renesas USB Device USB Peripheral Communication Device Class Driver Rev.1.10 (R01AN0273EJ0110)

Renesas USB Device USB Basic Firmware Rev.1.10 (R01AN0512EJ0110)

This USB flash boot loader has the following features.

- The target device can be manipulated by commands transferred from a PC.
  Flash memory erase, program, and blank check operations, as well as running a target program, can be performed by commands issued by a PC.
- Programs in the Motorola S format can be loaded.
- The USB 2.0 function module in the target device is used.
  USB 2.0 standard full speed transfers are supported.
- Conforms to the Abstract Control Model in the USB communication device class specifications.

Target Devices

RX62N Group and RX621 Group

If this application note's sample programs are used with other microcontrollers, they must be modified according to the specifications of the microcontroller used and tested thoroughly.

Contents

1. Specifications

When using this USB flash boot loader, commands issued from a terminal emulator running on a host PC are transmitted to the microcontroller to program the flash memory in the microcontroller.

Table 1.1 lists the peripheral functions used by the sample program and their applications, and figure 1.1 shows a usage overview.

Table 1.1 Peripheral Functions and Their Applications

Peripheral Function	Application
ROM (program code storage flash memory)	Internal flash memory programming using ROM P/E mode
USB 2.0 host/function module	For communication with the host PC

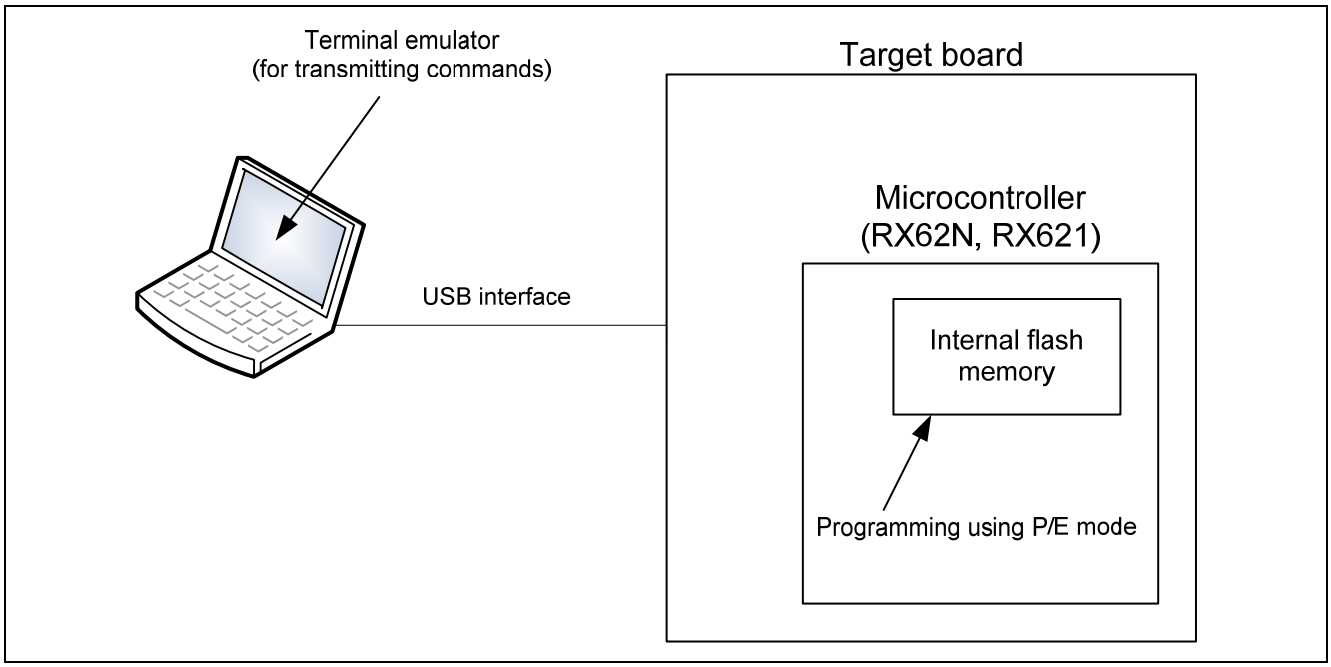


Figure 1.1 Usage Overview

2. Operation Verification Environment

Operation of the sample code provided in this application note has been verified in the following environment.

Table 2.1 Operation Verification Environment

Item	Description
Microcontroller Used	RX62N Group (R5F562N8BDBG)
Operating frequency	EXTAL: 12 MHz ICLK: 48 MHz BCLK: 48 MHz PCLK: 24 MHz UCLK: 48 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics High-performance Embedded Workshop Version 4.09.00.007
C Compiler	Renesas Electronics RX Standard Toolchain Version 1.1.0.0
	-cpu=rx600 -bit_order=left -include="$(WORKSPDIR)\WorkSpace\USBCSTDFW\include", "$(WORKSPDIR)\WorkSpace\USBSTDFW\include", "$(WORKSPDIR)\WorkSpace\USB2STDFW\include", "$(WORKSPDIR)\WorkSpace\SmplMain\APL", "$(WORKSPDIR)\WorkSpace\HwResourceForUSB", "$(WORKSPDIR)\WorkSpace\CDCCFW\include", "$(WORKSPDIR)\WorkSpace\CDCFW\include", "$(WORKSPDIR)\WorkSpace\FLASH" -define=USB_FUNCSEL_PP=USBC_PERI_PP, USBC_FW_PP=USBC_FW_NONOS_PP -output=obj="$(CONFIGDIR)\$(FILELEAF).obj" -debug -nostuff -optimize=0 -nologo
Operating mode	Single-chip mode
Sample code version	Version 1.00
Board used	The RSK + RX62N provided with the Renesas Development Tools (Catalog number: R0K5562N0S000BE)
Tools used	Terminal emulator

3. Related Application Notes

Related application notes are listed below. These related application notes should be used in conjunction with this application note.

- Renesas USB Device USB Basic Firmware Rev.1.10 (R01AN0512EJ)
- Renesas USB Device USB Peripheral Communication Device Class Driver Rev.1.10 (R01AN0273EJ)
- RX600 Series: Simple Flash API for RX600 Rev.2.20 (R01AN0544EU)

4. Hardware

4.1 Pins Used

Table 4.1 lists the pins used.

Table 4.1 Pins and Functions

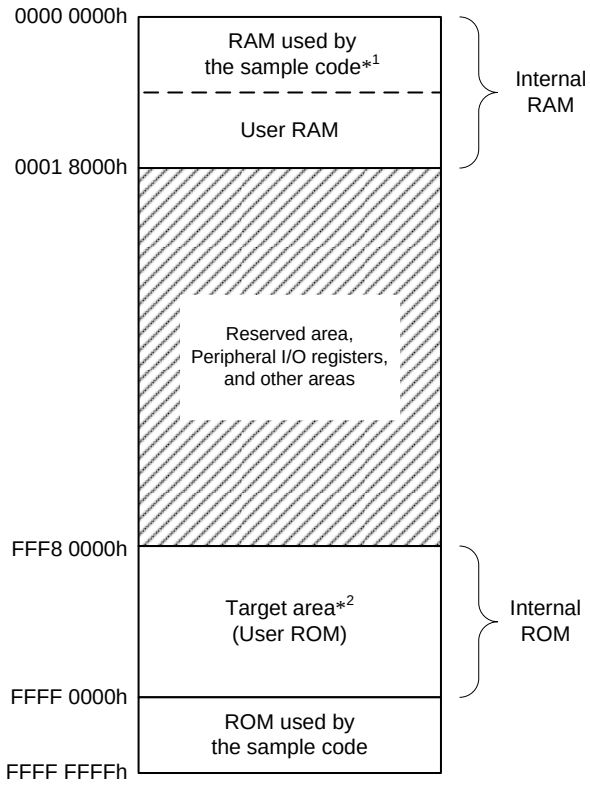
Pin	I/O	Use
USB0_DP	I/O	Port 0 USB internal transceiver D+ I/O pin Connected to the USB bus D+ pin.
USB0_DM	I/O	Port 0 USB internal transceiver D- I/O pin Connected to the USB bus D- pin.
USB0_VBUS	Input	Port 0 USB cable connection monitor pin Connected to the USB bus VBUS. During function operation, this pin can be used to detect the VBUS connected/disconnected state.
USB0_DPUPE	Output	Port 0 SUB D+ signal 1.5 kΩ pull-up resistor control signal used during function operation

Note: See the USB Peripheral Communication Device Class Driver application note for details on the other pin settings (switch, SCI, and other functions) used by the USB peripheral communication device class driver.

5. Software

5.1 Operation Overview

The sample code receives command data from the host PC and performs the operation (menu display, blank check, erase, program, or executing a target program) corresponding to the command. The terminal emulator performs the command transmission from the host PC. The target that this sample code can program is limited to one part (the target area) of the user MAT. The area (FFFF0000h to FFFFFFFFh) used by the sample code itself is not programmed. Figure 5.1 shows the memory allocation.



Notes: 1. This area can be used by the user when the target program is running.
2. Area to which the target program is written.

Figure 5.1 Memory Allocation

The sample code programs the flash memory in the sequence (1) to (3) shown below. Figures 5.2 and 5.3 show examples of the screen display in steps (1) and (2) of that sequence.

(1) After a reset is cleared, if the microcontroller is not connected to a PC, the target program is executed.
If a PC is connected, the microcontroller sends the message "Press any key" to the PC at fixed intervals until data is sent (until the terminal emulator is started and a key on the keyboard is pressed). (Figure 5.2)
(2) When an arbitrary data item is sent from the PC, the microcontroller sends the menu to the PC and waits for command data. (Figure 5.3)
(3) When a command is received, the microcontroller performs the operation (menu display, blank check, erase, program, or executing the target program).

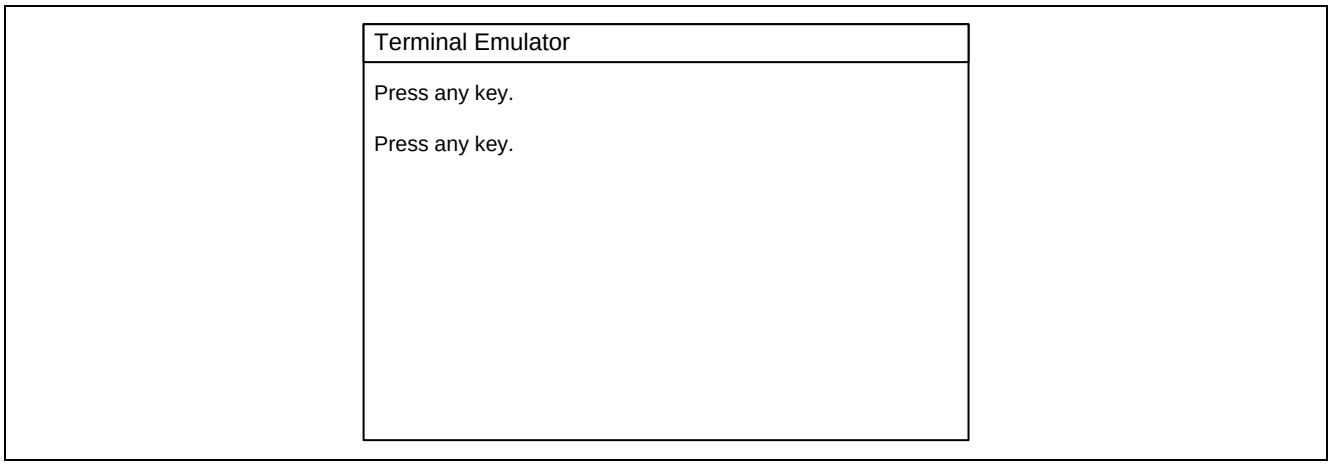


Figure 5.2 Key Input Wait Screen

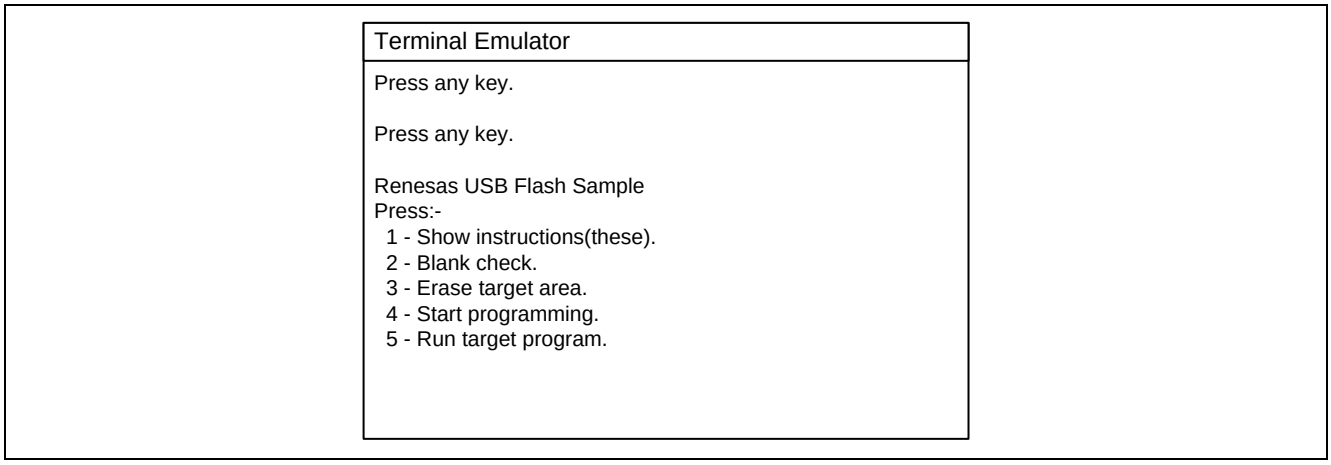


Figure 5.3 Command Input Wait Screen

5.2 Commands

Table 5.1 lists the commands. This command list is displayed on the terminal emulator by executing the menu display command. (Figure 5.3)

Table 5.1 Commands

Key Entered at the PC	ASCII Code	Command
1	31h	Display menu
2	32h	Blank check
3	33h	Erase
4	34h	Program
5	35h	Run target program

5.3 Target Program Execution Start Position

If the USB is not connected after a microcontroller reset is cleared or if execute target program is selected as the command, the sample code runs the target program. Here, the sample code starts execution from the address stored at location FFFE FFFCh. That is, for the target program, location FFFE FFFCh is the reset vector (target reset vector).

The target program must be coded so that the start address is stored in advance in the target reset vector.

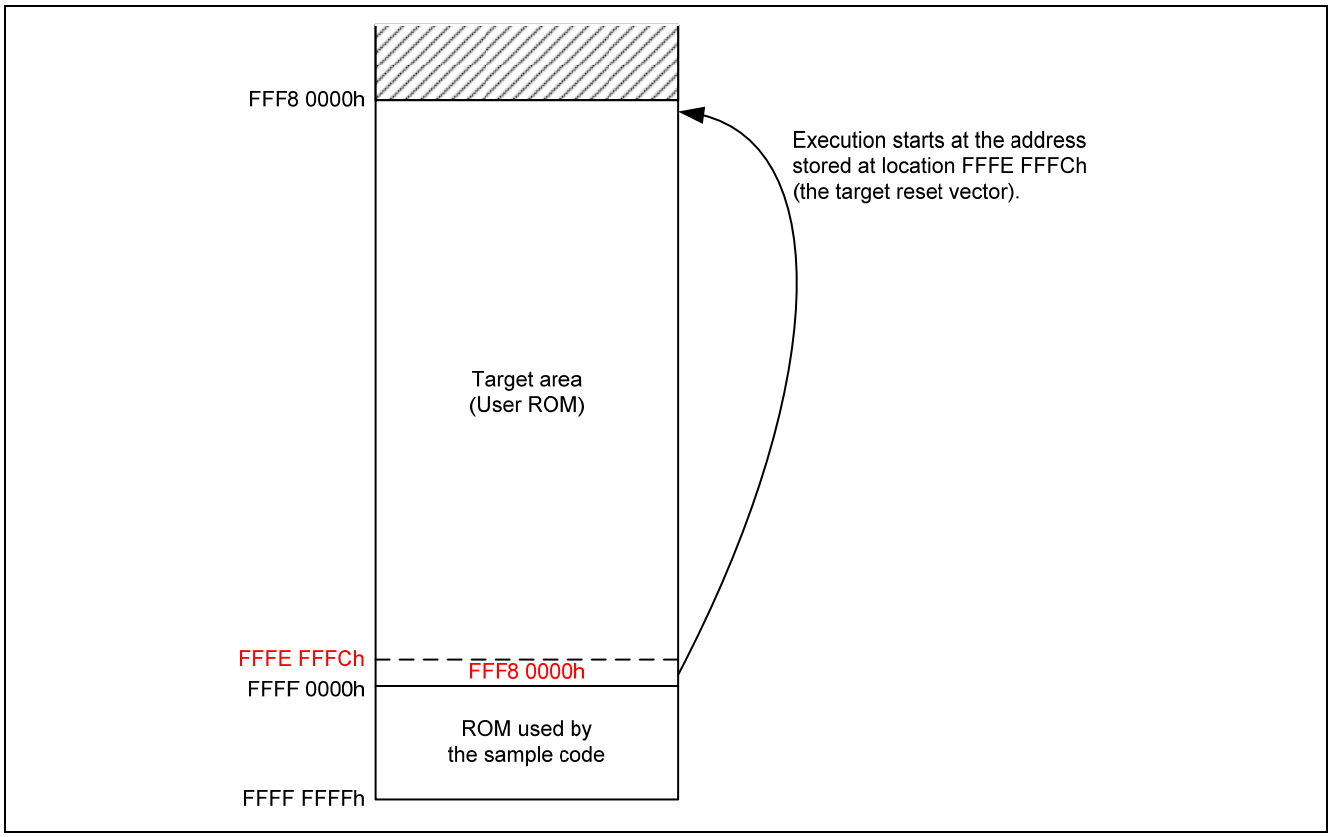


Figure 5.4 Target Reset Vector

5.4 Sample Code Modes

Figure 5.5 shows the mode transitions for the sample code.

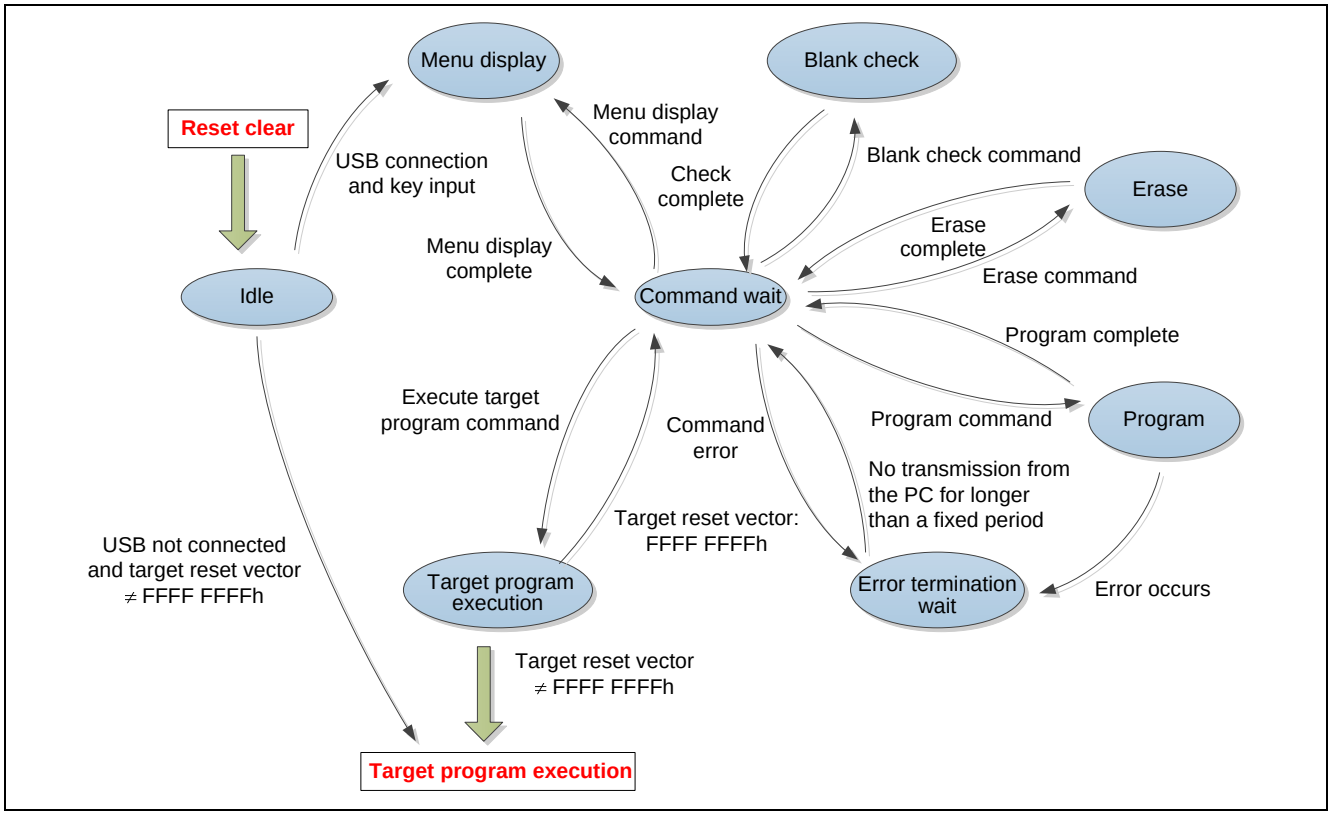


Figure 5.5 Mode Transition Diagram

5.4.1 Idle Mode

After a reset is cleared, the sample code transitions to idle mode. If the USB interface is not connected and furthermore the target reset vector has a value other than FFFF FFFFh, the target program is executed. If the USB interface is connected, the sample code issues a request for key input prompt at fixed intervals. If a key is pressed on the PC (that is, if the microcontroller receives an arbitrary data value), the sample code transitions to menu display mode.

5.4.2 Menu Display Mode

In this mode, the sample code displays a menu. When display has completed, it transitions to command wait mode.

5.4.3 Command Wait Mode

Here, the sample code waits for a command from the PC. When it receives a command, it transitions to the mode corresponding to that command.

5.4.4 Blank Check Mode

The sample code performs a blank check of the target area. When the blank check completes, it displays the result as a message and then transitions to command wait mode.

5.4.5 Erase Mode

The sample code erases the target area using the Simple Flash API. When the erase completes, it displays the result as a message and then transitions to command wait mode.

5.4.6 Program Mode

In this mode, the sample code writes to the target area using the Simple Flash API. After the microcontroller receives this command, it waits for the transmission of an mot file. Therefore, the PC should send a mot file in ASCII. The microcontroller operates as follows after receiving a program command.

(1) The microcontroller waits for an ASCII 'S', which is the first data in a Motorola S format file. At this time, all data other than ASCII 'S' will be discarded.*
(2) After receiving an 'S', the microcontroller receives data in the Motorola S format and writes that data.
After receiving and 'S', if any following data does not conform to the Motorola S format, the microcontroller transitions to the error termination wait mode. It also transitions to error termination wait mode if a checksum or other error occurs. See section 5.11, Message Table, for the messages displayed for each error.
(3) If the program operation completes, the microcontroller displays the result of that operation and transition to command wait mode.

Note: While newline codes are appended to the end of Motorola S format files, the sample code here handles the data through the checksum as the Motorola S format. Therefore, these newline codes are all discarded by the determination in step (1) above. Furthermore, the sample code discards any and all excess data received when the program operation completes. However, if the newline codes are received with a timing that follows this discard operation, the microcontroller will receive that data as commands. Therefore, command error messages may be displayed in this case.

5.4.7 Error Termination Wait Mode

If the microcontroller receives a command not listed in section 5.2, Commands, or if an error occurs in program mode, it transitions to error termination wait mode. If no data is received from the PC for a fixed period in error termination wait mode, the microcontroller transitions to command wait mode.

This is because if an error occurs in the mot file being received, although the microcontroller has cancelled the program operation, the terminal emulator may continue to transmit data. If the microcontroller transitioned to command wait mode immediately after an error occurred, this data would be interpreted as commands. Therefore the microcontroller transitions to error termination wait mode temporarily and waits until data is not being sent by the PC. Note that all data received from the PC in error termination wait mode is discarded.

5.4.8 Target Program Execution Mode

In this mode, the microcontroller stops the USB interface and executes the target program. Note, however, that if the value of the target reset vector is FFFF FFFFh, the microcontroller displays an error message and transitions to command wait mode.

5.5 Data Flow During Flash Programming

Figure 5.6 shows the flow of data within the microcontroller when writing a target program to flash memory.

During reception:

(1) The USB interface transfers the data received from the PC to the receive ring buffer.
(2) A single Motorola S format record is copied to a MotS buffer (ASCII).
(3) At the same time as analyzing the header of the Motorola S format record, the microcontroller converts the ASCII codes to binary data and stores it in a MotS buffer (binary).
(4) Data is stored in a write buffer.
In the RX62N and RX621 Group microcontrollers, the data unit for writing to the user MAT is 256 bytes. Therefore if there is less than 256 bytes of data, steps (1) to (4) are repeated until a full 256 bytes of data is acquired. Also, if the total amount of write data exceeds 256 bytes, the excess data is temporarily stored and used when writing the next 256 bytes.
(5) The prepared write data (256 bytes) is written to flash memory using the Simple Flash API.

During transmission:

(6) The result of the write operation is determined from the return value from the Simple Flash API call.
(7) A message corresponding to the result of the write operation is stored in a transmission ring buffer.
(8) The message is sent to the PC using the USB peripheral communication device class driver.

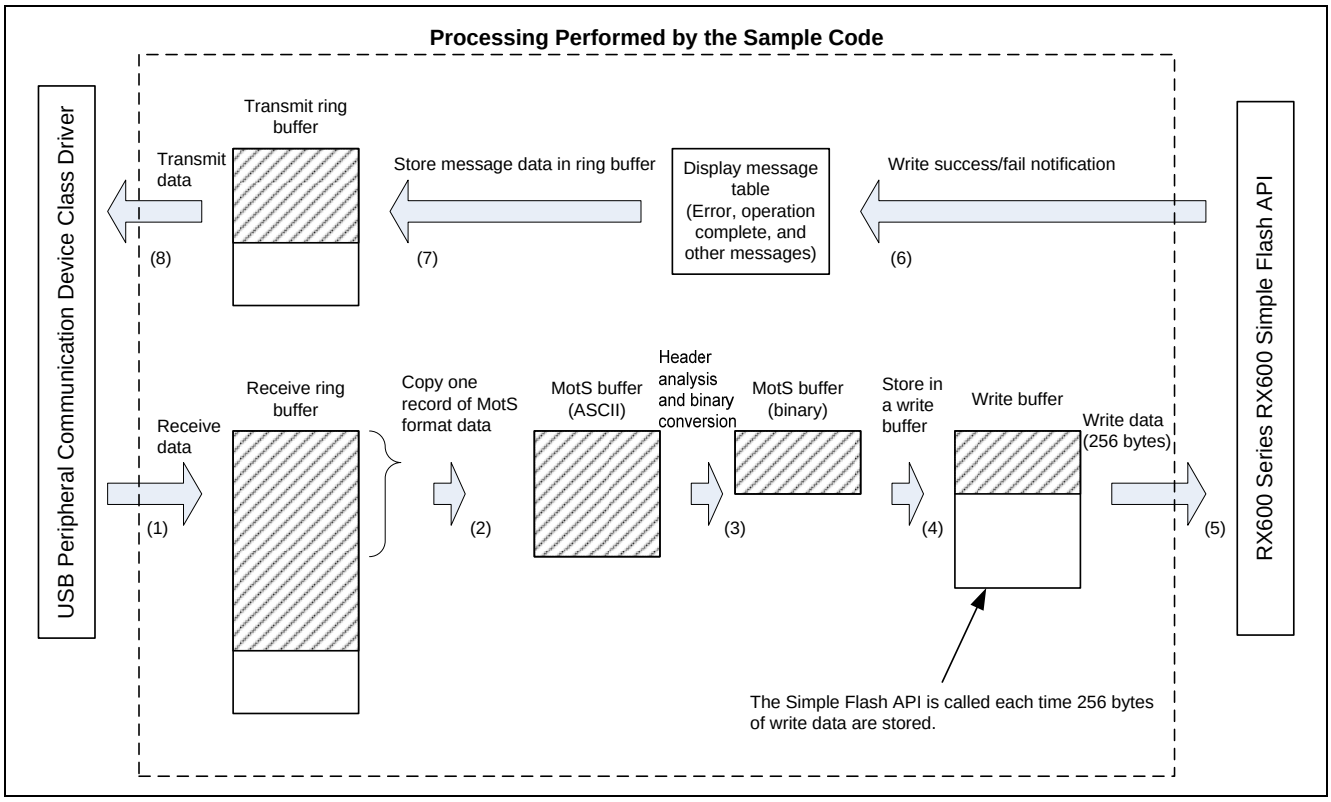


Figure 5.6 Data Flow During Flash Programming

Figure 5.7 shows the data structures used in programming.

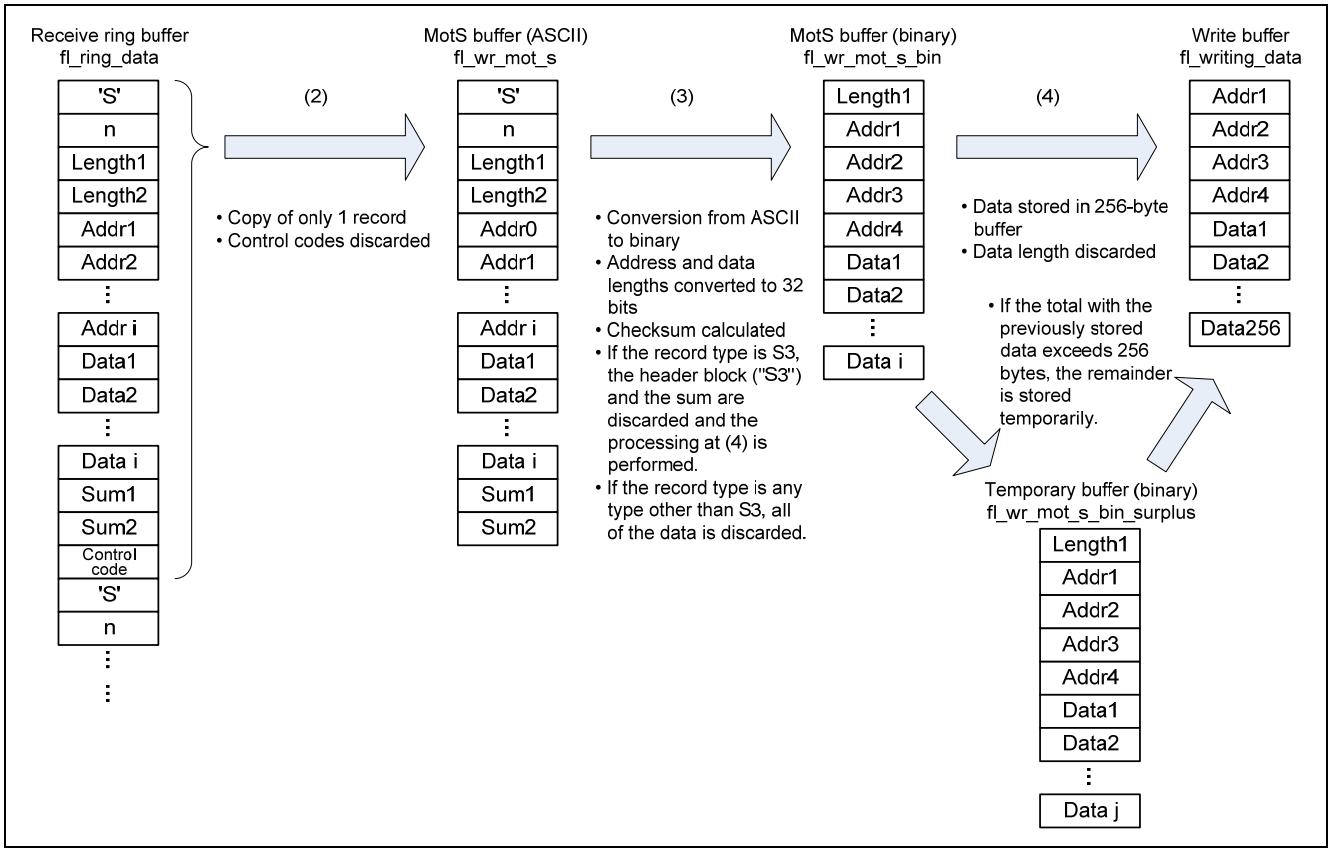


Figure 5.7 Data Structures for Flash Programming

5.6 Required Memory Capacity

Table 5.2 lists the memory capacity required.

Table 5.2 Required Memory Capacity

Memory Type	Size	Notes
ROM	60,286 bytes	Since the sample code is allocated at FFFF 0000h, the amount of ROM that can be programmed (the target area capacity) is the total ROM capacity minus 65,535 bytes.
RAM	17,070 bytes	The target program can use this area freely when it runs.

Note: The required memory capacity differs depending on the C compiler version used and the compiler options.

5.7 Changes in the ROM Capacity

The sample code assumes that a 512 KB ROM will be used. If a microcontroller with a 384 KB or 256 KB ROM capacity is used, the FL_END_BLOCK_NUM constant defined in the file r_Flash_main.y must be changed to match the actual capacity.

Table 5.3 lists the target area ROM capacities.

Table 5.3 Target Area ROM Capacities

Product	ROM Capacity	Target Area ROM Capacity	Target Area Start Address	Target Area Block Numbers
R5F562x8	512 K	448 K	FFF8 0000h	EB10 to EB37
R5F562x7	384 K	320 K	FFFA 0000h	EB10 to EB29
R5F562x6	256 K	192 K	FFFC 0000h	EB10 to EB21

5.8 File Structure

Table 5.4 lists the sample code files.

Table 5.4 File Structure

File Name	Overview	Notes
r_flash_api_rx600.c	RX600 Series RX600 Simple Flash API program	See the application note that describes the RX600 Series RX600 Simple Flash API for details.
r_flash_api_rx600.h	External reference include header for the RX600 Series RX600 Simple Flash API program	See the application note that describes the RX600 Series RX600 Simple Flash API for details.
r_flash_api_rx600_private.h	External reference include header for the RX600 Series RX600 Simple Flash API program	See the application note that describes the RX600 Series RX600 Simple Flash API for details.
r_flash_api_rx600_config.h	External reference include header for the RX600 Series RX600 Simple Flash API program	See the application note that describes the RX600 Series RX600 Simple Flash API for details.
mcu_info.h	External reference include header for the RX600 Series RX600 Simple Flash API program	See the application note that describes the RX600 Series RX600 Simple Flash API for details.
r_Flash_main.c	Flash programming data processing	
r_Flash_main.h	External reference include header for the flash programming data processing file	
r_Flash_buff.c	USB send/receive buffer related processing	
r_Flash_buff.h	External reference include header for the USB send/receive buffer related processing	
TrgtPrgDmmy.c	Dummy program for allocating space for the target program	
Other files	USB peripheral communication device class driver programs	See the application notes that describe the Renesas USB device USB peripheral communication device class driver and USB basic firmware for details.

5.9 Constants

Table 5.5 lists the constants used in the sample code.

Table 5.5 Sample Code Constants

Constant	Set Value	Description
FL_CMD_DATA_SHOW_INST	31h	Command value. ASCII "1"
FL_CMD_DATA_BLNK_CHECK	32h	Command value. ASCII "2"
FL_CMD_DATA_ERASE_TRGT_AREA	33h	Command value. ASCII "3"
FL_CMD_DATA_PRG_TRGT_AREA	34h	Command value. ASCII "4"
FL_CMD_DATA_RUN_TRGT_AREA	35h	Command value. ASCII "5"
FL_RINGBUFF_SIZE	1024	USB data reception ring buffer size
FL_RINGBUFF2_SIZE	256	USB data transmission ring buffer size
FL_USB_RCV_BLANK_SIZE	64	Number of data items that USB can transmit at a time
FL_SEND_END_CODE	0	Message table end code
FL_MOTS_ADDR_SIZE	4	Motorola S format address buffer size
FL_MOTS_SUM_SIZE	1	Motorola S format checksum buffer size
FL_START_BLOCK_NUM	10	First target area block
FL_END_BLOCK_NUM	37	Last target area block
FL_TARGET_REST_VECT_ADDR	FFFE FFFCh	Target reset vector
FL_USB_UNCONNECT_WAIT_PERIOD	10000h	Wait time from USB disconnect until target program execution
FL_IDLE_MESSAGE_OUTPUT_PERIOD	10000h	Message display interval in idle mode
FL_ERROR_WAIT_PERIOD	10000h	The error termination wait time (The error termination wait state is terminated if nothing is received from the USB while counting this value.)

5.10 Structures and Unions

Figure 5.8 shows the structures and unions used in the sample code. Note, however, that structures and unions used by USB peripheral communication device class driver and the Simple Flash API are not included.

```
/* 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 Sample Code Structures and Unions

5.11 Message Table

Table 5.6 lists the messages output by the sample code. Note that the newline codes in the messages have been omitted.

Table 5.6 Sample Code Messages

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.	The menu
Target area is blank.	After a blank check, indicates that the target area was blank.
Target area is NOT blank.	After a blank check, indicates that the target area was not blank.
Erase target area...	Displayed when an erase operation is in progress.
Erase complete.	Displayed when erase operation has completed.
ERROR!!! - Erase is failed.	Displayed when an erase operation has failed.
Please send a mot file.	Displayed when a write (program) operation is started. The user should send a mot file after this message is displayed.
Program complete.	Displayed when a write (program) operation has completed.
Program failed.	Displayed when a write (program) operation has failed.
ERROR!!! - Verify error.	Displayed when a verify operation has failed.
Run target program.	Displayed when the target program is run.
ERROR!!! - Target reset vector (0xFFFEFFFC) is 0xFFFFFFFF.	Displayed when the target vector is FFFF FFFFh on an attempt to run 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 was sent.
ERROR!!! - Check sum error.	Displayed when a checksum error occurred for the target program (in the Motorola S format).
Please wait for instructions to be shown.	Displayed in the state where an error has occurred and data continues to be sent. The sample code transitions to the command wait state when data is not sent for a certain period. If data continues to be sent after this error occurs, a period will be displayed at fixed intervals.
Period: "."	Displayed at fixed intervals after an error has occurred.
Newline code: "\r\n"	Used for the newline following each message.

5.12 Functions

Table 5.7 lists the functions in the sample code. Note, however, that the functions in the USB peripheral communication device class driver and the Simple Flash API are not included.

Table 5.7 Functions

Function	Overview
R_Fl_Rewrite_process	The main flash memory write (program) function
R_Fl_Idle	The processing between the point a reset has been cleared and a key is pressed
R_Fl_AnalyzeCMD	Command analyzer
R_Fl_EraseTrgtArea	Erase
R_Fl_PrgTrgtArea	Programs flash memory
R_Fl_RunTrgtPrg	Runs the target program
R_Fl_ErrorWait	Waits for error termination
R_Fl_cmd_ShowInst	Displays the menu
R_Fl_cmd_BlankCheck	Performs a blank check
R_Fl_cmd_EraseStart	Starts an erase operation
R_Fl_cmd_PrgStart	Starts a program operation
R_Fl_cmd_RunTrgtPrgStart	Starts program execution
R_Fl_Blnk_BlankCheck	Blank check (processing according to the Simple Flash API example)
R_Fl_Ers_EraseFlash	Erase (processing according to the Simple Flash API example)
R_Fl_Prg_StoreMotS	Stores Motorola S format data
R_Fl_Prg_ProcessForMotS_data	Analyzes Motorola S format data and converts it to binary
R_Fl_Prg_ConvertAsciiToBinary	Analyzes Motorola S format data and converts it to binary
R_Fl_Prg_MakeWriteData	Creates the write data
R_Fl_Prg_WriteData	Writes a target program (processing according to the Simple Flash API example)
R_Fl_Prg_ClearMotSVariables	Clears variables related to the Motorola S format data
R_Fl_Run_StopUSB	Stops the USB interface
R_Fl_RcvDataString	Stores USB receive data
R_Fl_SetSendData	Stores USB transmit data
R_Fl_SetDisplayMsgData	Stores display message data
R_Fl_RingCheckBlank	Checks that a buffer used to store USB receive data is blank
R_Fl_Ring2CheckData	Checks that a buffer used to store USB transmit data is blank
R_Fl_USB_NonConnect_Run	Runs the target program when the USB is disconnected
R_Fl_RingEnQueue	Writes to a USB receive data buffer
R_Fl_RingDeQueue	Reads from a USB receive data buffer
R_Fl_RingClear	Clears a USB receive data buffer
R_Fl_RingCheck	Checks the number of items in a USB receive data buffer
R_Fl_Ring2EnQueue	Writes to a USB transmit data buffer
R_Fl_Ring2DeQueue	Reads from a USB transmit data buffer
R_Fl_Ring2Clear	Clears a USB transmit data buffer
R_Fl_Ring2Check	Checks the number of items in a USB transmit data buffer
R_Fl_AsciiToHexByte	Converts ASCII codes to binary data

5.13 Function Specifications

This section presents the specifications of the functions in the sample code.

R_Fl_Rewrite_process	
Overview	The main flash memory write (program) function
Header	r_Flash_main.h
Declaration	void R_Fl_Rewrite_process(void)
Arguments	Calls the processing function for the mode, according to the write processing mode.
Return values	None
Description	None
Notes	

R_Fl_Idle	
Overview	Idle mode processing
Header	None
Declaration	static void R_Fl_Idle(void)
Description	Displays the "Press any key" message at fixed intervals until data is received from the USB interface, that is, until there is key input from the PC.
Arguments	None
Return values	None
Notes	

R_Fl_AnalyzeCMD	
Overview	Command analysis
Header	None
Declaration	static Fl_SMPL_command_t R_Fl_AnalyzeCMD(void)
Description	• If there is data in the receive ring buffer, analyzes the first byte of that data as a command. • Discards all of the data other than the first byte.
Arguments	None
Return values	• No data received: FL_CMD_NONE • Menu display command received: FL_CMD_SHOW_INST • Blank check command received: FL_CMD_BLNK_CHECK • Erase command received: FL_CMD_ERASE_TRGT_AREA • Write command received: FL_CMD_PRG_TRGT_AREA • Execute target area command received: FL_CMD_RUN_TRGT_AREA • Data other than one of the above received: FL_CMD_ERROR
Notes	

R_Fl_EraseTrgtArea	
Overview	Erase mode processing
Header	r_Flash_main.h
Declaration	static void R_Fl_EraseTrgtArea(void)
Description	• Calls the function that erases the target area. • Displays the erase result as a message.
Arguments	None
Return values	None
Notes	

R_Fl_PrgTrgtArea	
Overview	Write mode processing
Header	None
Declaration	static void R_Fl_PrgTrgtArea(void)
Description	• If there is data in the receive ring buffer, calls the function that stores that data as Motorola S format data. • If one Motorola S format data record has been received, analyzes the header and calls the function that converts that data to binary. • If conversion to binary has completed, calls the function that stores that data in a write buffer.
Arguments	None
Return values	None
Notes	

R_Fl_RunTrgtPrg	
Overview	Execute mode processing
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 interface and then executes the target program. • If the target reset vector is FFFF FFFFh, displays the target vector error message.
Arguments	None
Return values	None
Notes	

R_Fl_ErrorWait	
Overview	Error termination wait mode processing
Header	None
Declaration	static void R_Fl_ErrorWait(void)
Description	• If a period longer than a certain fixed period has elapsed in the state where the receive ring buffer is empty, transitions to command wait mode. • If there is data in the receive ring buffer, discards that data and then starts waiting for the fixed period to elapse.
Arguments	None
Return values	None
Notes	

R_Fl_cmd_ShowInst	
Overview	Menu display
Header	None
Declaration	static void R_Fl_cmd_ShowInst(void)
Description	Displays the menu.
Arguments	None
Return values	None
Notes	

R_Fl_cmd_BlankCheckStart	
Overview	Start blank check
Header	None
Declaration	static void R_Fl_cmd_BlankCheckStart(void)
Description	• Calls the function that performs a blank check for the target area. • Displays the result of the blank check as a message.
Arguments	None
Return values	
Notes	

R_Fl_cmd_EraseStart	
Overview	Start target area erase
Header	None
Declaration	static void R_Fl_cmd_EraseStart(void)
Description	• Starts an erase of the target area. • Displays the erase start message.
Arguments	None
Return values	None
Notes	

R_Fl_cmd_PrgStart	
Overview	Start target area write
Header	None
Declaration	static void R_Fl_cmd_PrgStart(void)
Description	• Starts the write of the target area. • Displays the write start message.
Arguments	None
Return values	None
Notes	

R_Fl_cmd_RunTrgtPrgStart	
Overview	Start target program execution
Header	None
Declaration	static void R_Fl_cmd_RunTrgtPrgStart(void)
Description	• If the target vector is FFFF FFFFh, issues the target vector error message. • If the target vector is not FFFF FFFFh, displays the executing target program message and then transitions to target program execution mode.
Arguments	None
Return values	None
Notes	

R_Fl_Blnk_BlankCheck	
Overview	Blank check
Header	None
Declaration	Fl_API_SMPL_rtn_t R_Fl_Blnk_BlankCheck(void)
Description	Performs a blank check of the target area.
Arguments	None
Return values	• If the target area was blank: FLASH_API_SAMPLE_OK • If the target area was not blank: FLASH_API_SAMPLE_NG
Notes	

R_Fl_Ers_EraseFlash	
Overview	Erase target area
Header	None
Declaration	Fl_API_SMPL_rtn_t R_Fl_Ers_EraseFlash(void)
Description	Erases the target area.
Arguments	None
Return values	• If the erase completed normally: FLASH_API_SAMPLE_OK • If the erase did not complete normally: FLASH_API_SAMPLE_NG
Notes	The processor status word (PSW) processor interrupt priority level (IPL) is modified to prevent ROM access due to interrupts during erase operations.

R_Fl_Prg_StoreMotS	
Overview	Store Motorola S format data
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_Prg_StoreMotS(uint8_t)
Description	• Stores the data acquired as an argument 1 byte at a time as Motorola S format data. • Discards all data until the first 'S' (ASCII code) is received.
Arguments	First argument: mot_data : Motorola S format data
Return values	• If one unit of Motorola S format data (from the 'S' to the checksum) was stored: FLASH_API_SAMPLE_OK • If one unit of Motorola S format data was not stored: FLASH_API_SAMPLE_NG
Notes	• This function is used by passing the Motorola S format data one byte at a time. • It does not check the checksum.

R_Fl_Prg_ProcessForMotS_data	
Overview	Motorola S format header analysis and binary conversion
Header	None
Declaration	static void R_Fl_Prg_ProcessForMotS_data(void)
Description	• Analyzes the Motorola S format header and calls the function that converts to binary. • If the data differs from the Motorola S format, displays an error message.
Arguments	None
Return values	None
Notes	

R_Fl_Prg_MotS_AsciiToBinary	
Overview	Motorola S format data ASCII to binary conversion
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 ASCII codes in the Motorola S format data to binary. • Verifies the checksum of the converted binary data • If the data differs from the Motorola S format, displays an error message. • If a checksum error occurs, displays an error message.
Arguments	First argument: *tmp_mot_s : Pointer to Motorola S format data consisting of ASCII codes
	Second argument: *tmp_mot_s_binary : Pointer to a variable to hold the binary converted data
Return values	• If conversion completed normally: FLASH_API_SAMPLE_OK • If conversion did not complete normally: FLASH_API_SAMPLE_NG
Notes	

R_Fl_Prg_MakeWriteData	
Overview	Target area write data creation
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_Prg_MakeWriteData(void)
Description	Creates data divided into 256-byte units.
Arguments	None
Return values	• If creation of 256 bytes of write data completed normally: FLASH_API_SAMPLE_OK • If creation of 256 bytes of write data did not complete normally: FLASH_API_SAMPLE_NG
Notes	

R_Fl_Prg_WriteData	
Overview	Target area write
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_Prg_WriteData(void)
Description	• Writes data to the target area. • Verifies the written data. • If the write failed, displays an error message. • If a verify error occurs, displays an error message.
Arguments	None
Return values	• If the write completed normally: FLASH_API_SAMPLE_OK • If the write did not complete normally: FLASH_API_SAMPLE_NG
Notes	The processor status word (PSW) processor interrupt priority level (IPL) is modified to prevent ROM access due to interrupts during write operations.

R_Fl_Prg_ClearMotSVariables	
Overview	Clear Motorola S format variables
Header	None
Declaration	static void R_Fl_Prg_ClearMotSVariables(void)
Description	Clears the variables used for the Motorola S format data.
Arguments	None
Return values	None
Notes	

R_Fl_Run_StopUSB	
Overview	USB stop
Header	iodefine.h
Declaration	static void R_Fl_Run_StopUSB(void)
Description	Stops the USB interface.
Arguments	None
Return values	None
Notes	

R_Fl_RcvDataString		
Overview	Store USB receive data	
Header	R_Flash_main.h	
Declaration	Fl_API_SMPL_rtn_t R_Fl_RcvDataString(void *, uint16_t)	
Description	Stores data received over the USB bus in a ring buffer.	
Arguments	First argument: *tranadr	: Pointer to buffer to store the data received over the USB bus.
	Second argument: length	: Data count for the receive data.
Return values	• If the store completes: FLASH_API_SAMPLE_OK • If the receive ring buffer becomes full: FLASH_API_SAMPLE_NG	
Notes		

R_Fl_SetSendData		
Overview	Store USC transmit data	
Header	R_Flash_main.h	
Declaration	uint16_t R_Fl_SetSendData(void *, uint16_t)	
Description	Stores the data to be sent over the USB interface in a transmit buffer.	
Arguments	First argument: *tranadr	: Pointer to transmit buffer
	Second argument: length_lim	: Limit on length of data to store in transmit buffer
Return values	Count of data items stored in transmit buffer	
Notes		

R_Fl_SetDisplayMsgData		
Overview	Message display	
Header	None	
Declaration	static Fl_API_SMPL_rtn_t R_Fl_SetDisplayMsgData(Fl_disp_tbl_num_t)	
Description	Stores the specified message in the transmit ring buffer	
Arguments	First argument: table_num	: Number of the message to display
Return values	• If the store completes: FLASH_API_SAMPLE_OK • If the transmit ring buffer becomes full: FLASH_API_SAMPLE_NG	
Notes		

R_FI_RingCheckBlank	
Overview	Check the receive ring buffer for remaining space
Header	R_Flash_main.h
Declaration	FI_API_SMPL_rtn_t R_FI_RingCheckBlank(void)
Description	Checks whether or not there is space remaining in the receive ring buffer for a single USB reception (64 bytes).
Arguments	None
Return values	• If there is space: FLASH_API_SAMPLE_OK • If there isn't space: FLASH_API_SAMPLE_NG
Notes	

R_FI_Ring2CheckData	
Overview	Check for data in the transmit ring buffer
Header	R_Flash_main.h
Declaration	FI_API_SMPL_rtn_t R_FI_Ring2CheckData(void)
Description	Checks whether or not there is data in the transmit data buffer.
Arguments	None
Return values	• If there is data present: FLASH_API_SAMPLE_OK • If there is no data present: FLASH_API_SAMPLE_NG
Notes	

R_FI_USB_NonConnect_Run	
Overview	Run the target program when the USB is disconnected
Header	R_Flash_main.h
Declaration	void R_FI_USB_NonConnect_Run(void)
Description	Stops the USB interface and runs the target program.
Arguments	None
Return values	None
Notes	This function is only called when the USB is not connected.

R_FI_RingEnQueue	
Overview	Store 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	First argument: *enq_data : Data to store
Return values	• If the store completes: FLASH_API_SAMPLE_OK • If the buffer becomes full: FLASH_API_SAMPLE_NG
Notes	

R_FI_RingDeQueue	
Overview	Read data from the receive ring buffer
Header	r_Flash_buff.h
Declaration	FI_API_SMPL_rtn_t R_FI_RingDeQueue(uint8_t *)
Description	Reads data from the receive ring buffer.
Arguments	First argument: *deq _data : Pointer to buffer to hold the data read.
Return values	• If data is read out normally: FLASH_API_SAMPLE_OK • If there was no data to read: FLASH_API_SAMPLE_NG
Notes	

R_Fl_RingClear	
Overview	Clear 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 values	Always returns FLASH_API_SAMPLE_OK.
Notes	

R_Fl_RingCheck	
Overview	Check number of data items in the receive ring buffer
Header	r_Flash_buff.h
Declaration	uint32_t R_Fl_RingCheck(void)
Description	Checks the number of data items in the receive ring buffer
Arguments	None
Return values	Returns the number of data items.
Notes	

R_Fl_Ring2EnQueue	
Overview	Store 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	First argument: enq_data : Data to store
Return values	• If the store completes: FLASH_API_SAMPLE_OK • If the buffer becomes full: FLASH_API_SAMPLE_NG
Notes	

R_Fl_Ring2DeQueue	
Overview	Read data from the transmit ring buffer
Header	r_Flash_buff.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_Ring2DeQueue(uint8_t *)
Description	Reads data from the transmit ring buffer.
Arguments	First argument: *deq_data : Pointer to buffer to hold the data read.
Return values	• If data is read out normally: FLASH_API_SAMPLE_OK • If there was no data to read: FLASH_API_SAMPLE_NG
Notes	

R_Fl_Ring2Clear	
Overview	Clear 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 values	Always returns FLASH_API_SAMPLE_OK.
Notes	

R_Fl_Ring2Check	
Overview	Check number of data items in the transmit ring buffer
Header	r_Flash_buff.h
Declaration	uint32_t R_Fl_Ring2Check(void)
Description	Checks the number of data items in the transmit ring buffer
Arguments	None
Return values	Returns the number of data items.
Notes	

R_Fl_AsciiToHexByte		
Overview	Convert from ASCII code to binary data	
Header	r_Flash_buff.h	
Declaration	uint8_t R_Fl_AsciiToHexByte(uint8_t, uint8_t)	
Description	Converts a 2-byte ASCII code to 1 byte of binary data.	
Arguments	First argument: in_upper	: ASCII code data (upper byte)
	Second argument: in_lower	: ASCII code data (lower byte)
Return values	Returns the data converted to binary	
Notes		

5.14 Flowcharts

5.14.1 Main USB Processing

Figure 5.9 shows the flowchart for the main USB processing

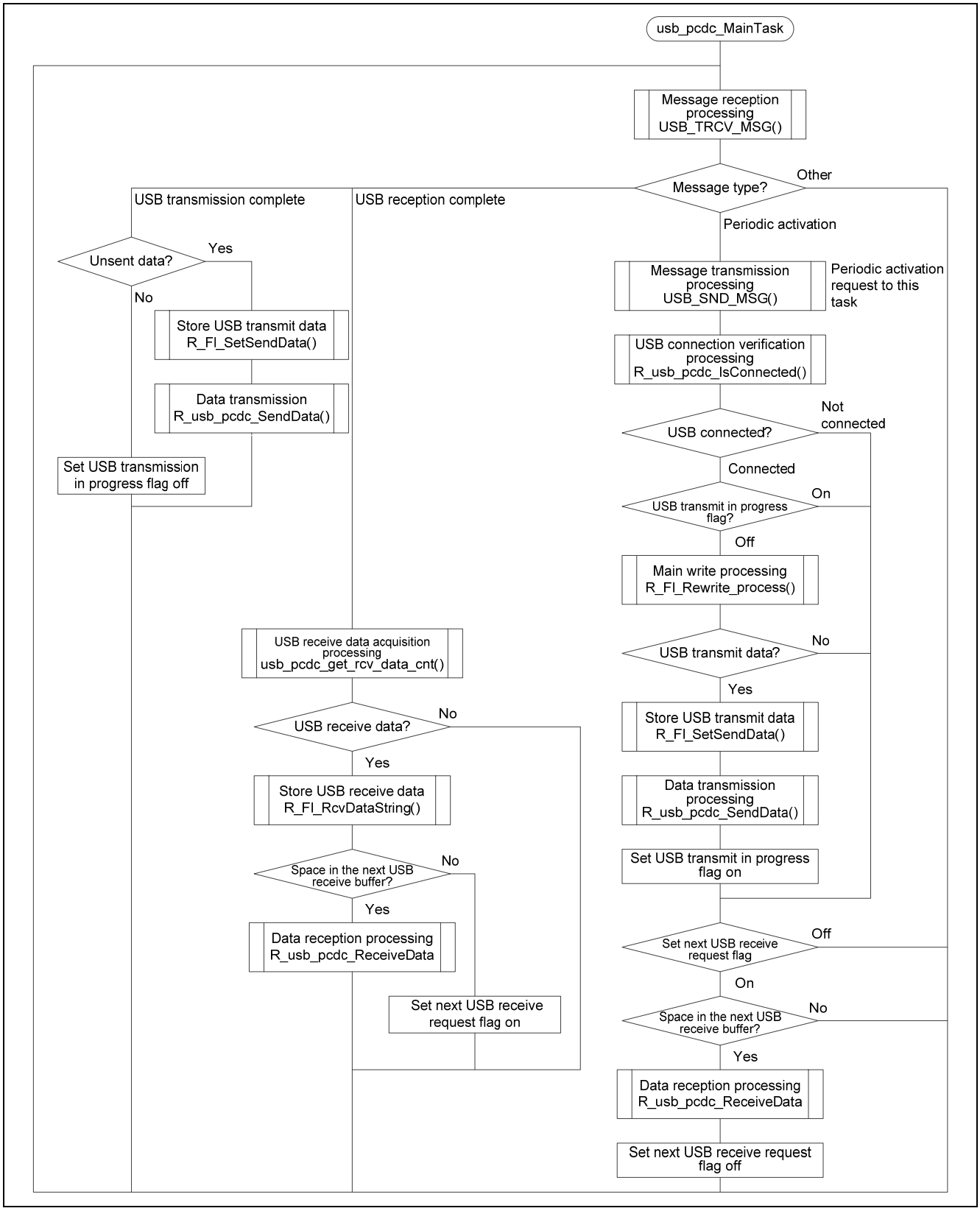


Figure 5.9 Main USB Processing

5.14.2 Main Write Processing

Figure 5.10 shows the flowchart for the main write processing.

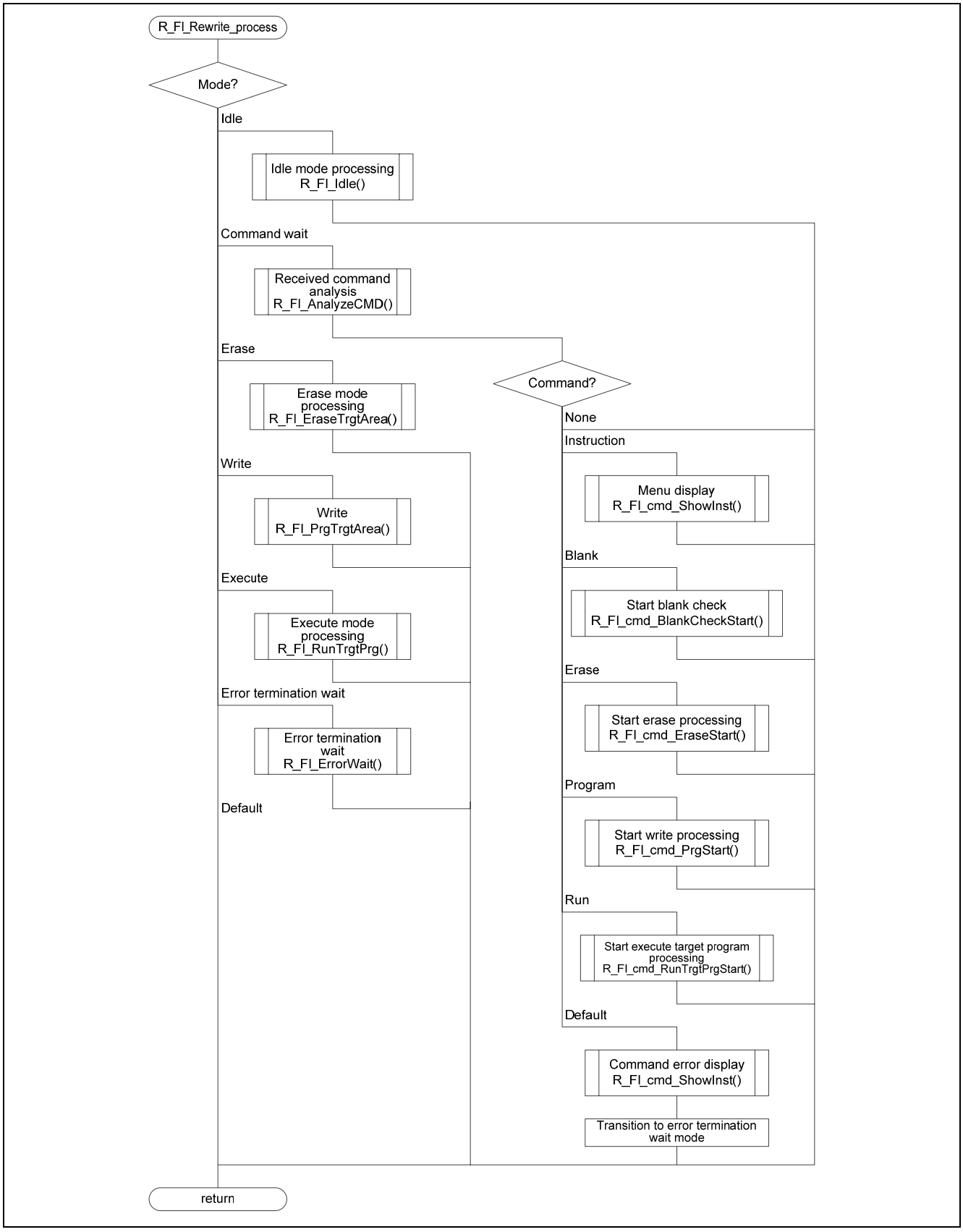


Figure 5.10 Main Write Processing

5.14.3 Idle Mode Processing

Figure 5.11 shows the flowchart for the idle mode processing.

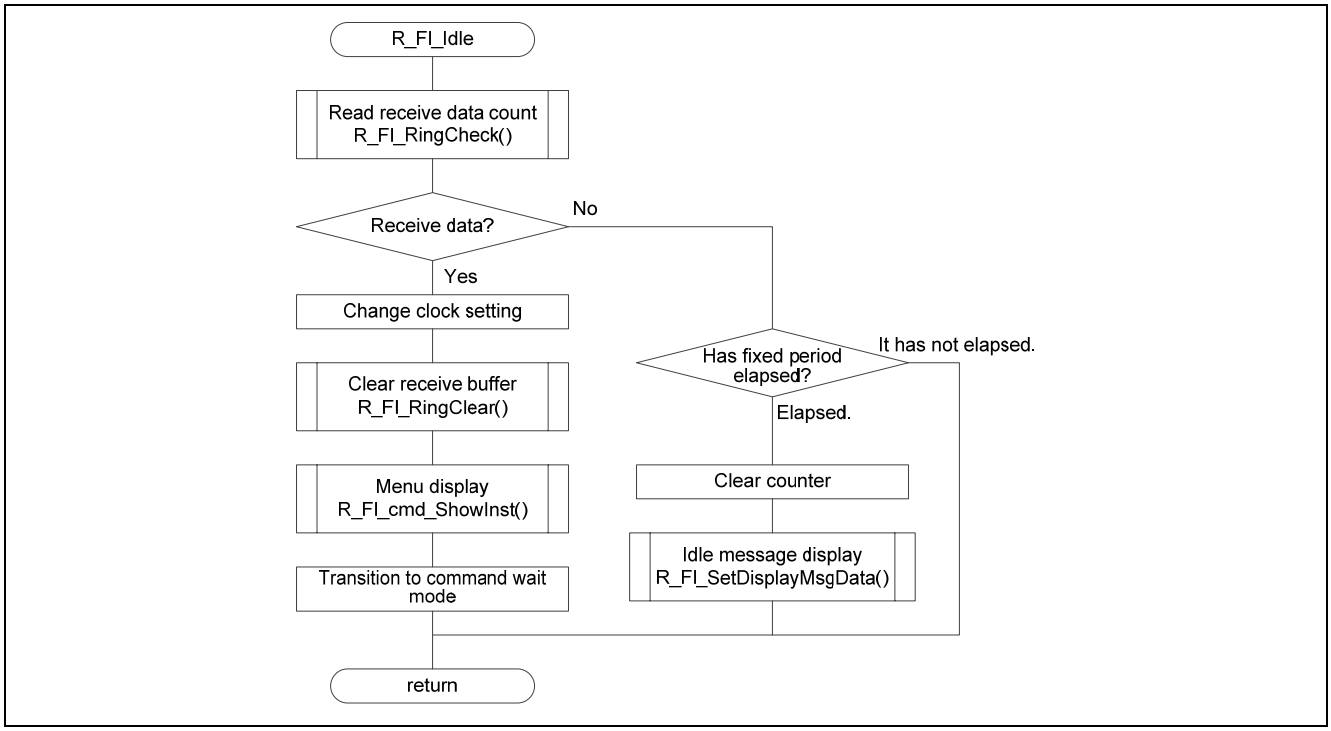


Figure 5.11 Idle Mode Processing

5.14.4 Command Analysis

Figure 5.12 shows the flowchart for the command analysis.

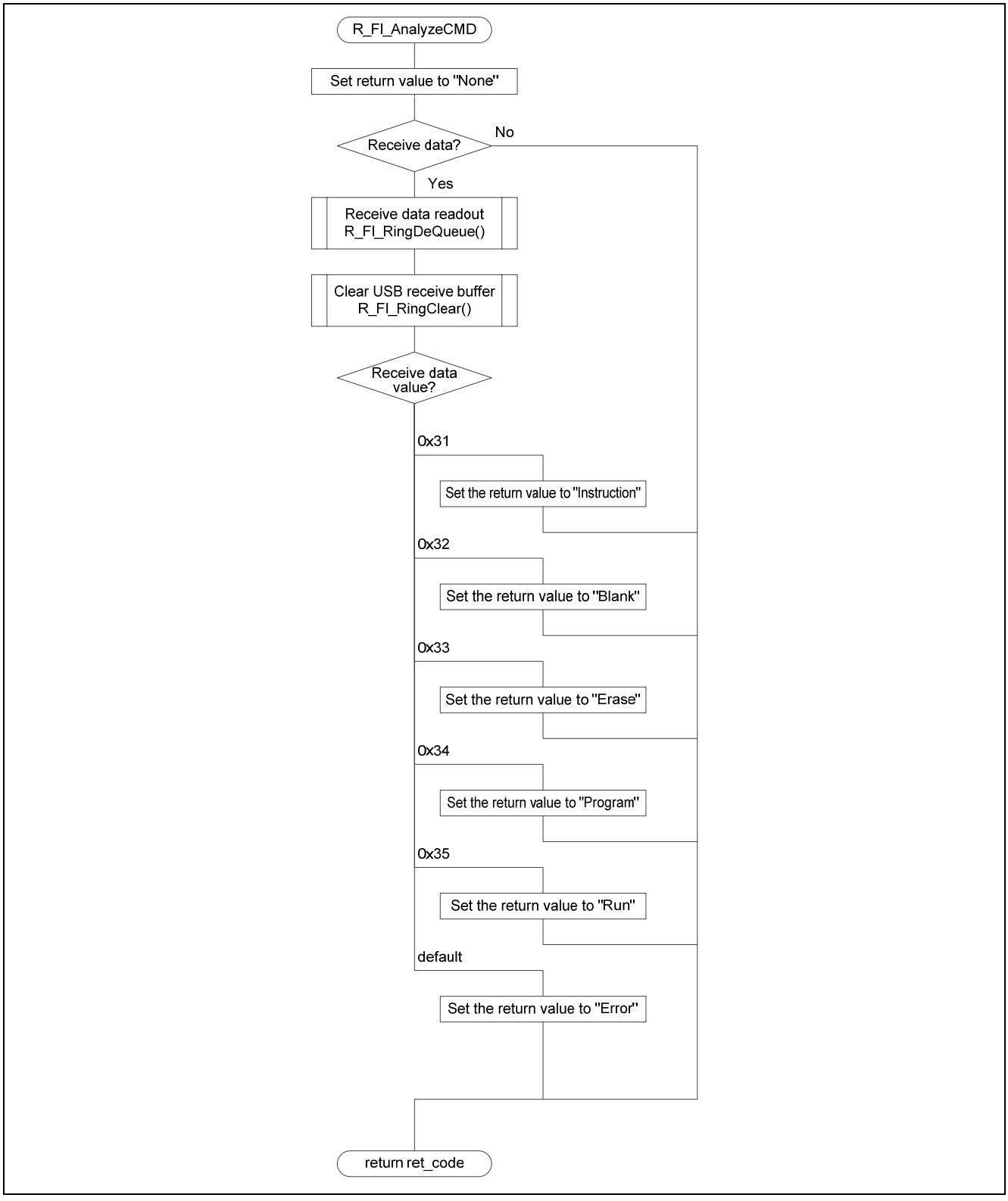


Figure 5.12 Command Analysis

5.14.5 Erase Mode Processing

Figure 5.13 shows the flowchart for the erase mode processing.

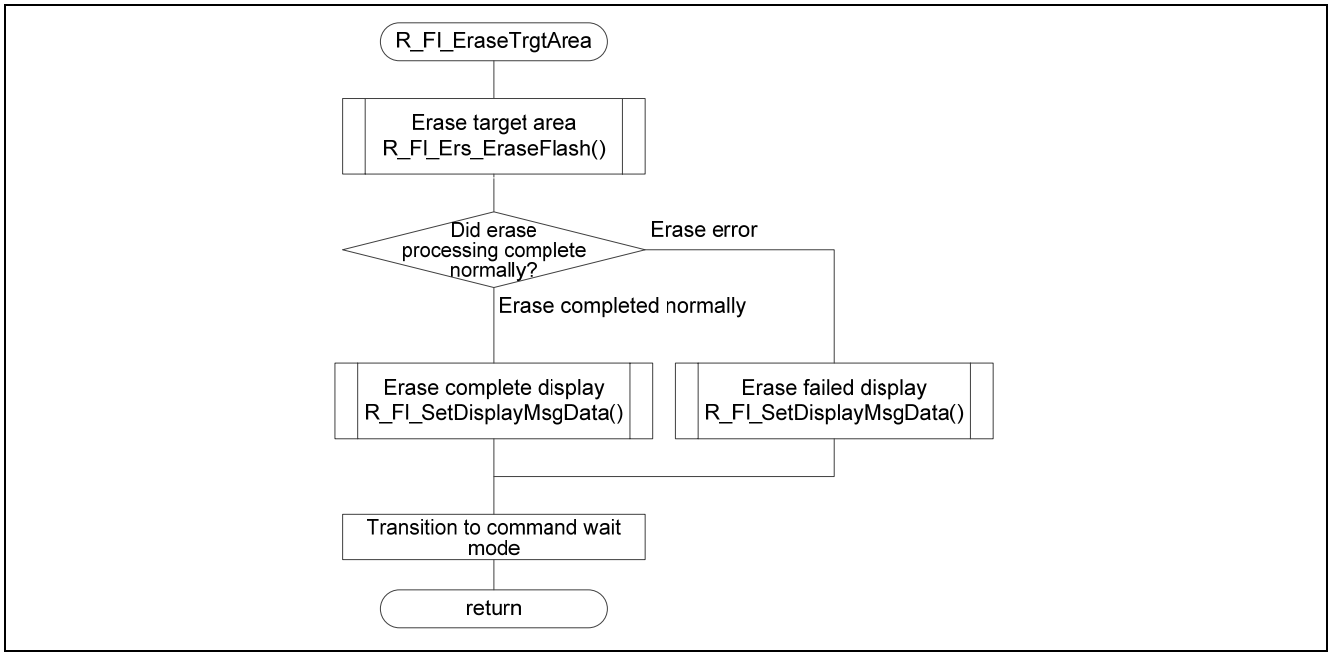


Figure 5.13 Erase Mode Processing

5.14.6 Write Mode Processing

Figure 5.14 shows the flowchart for the write mode processing.

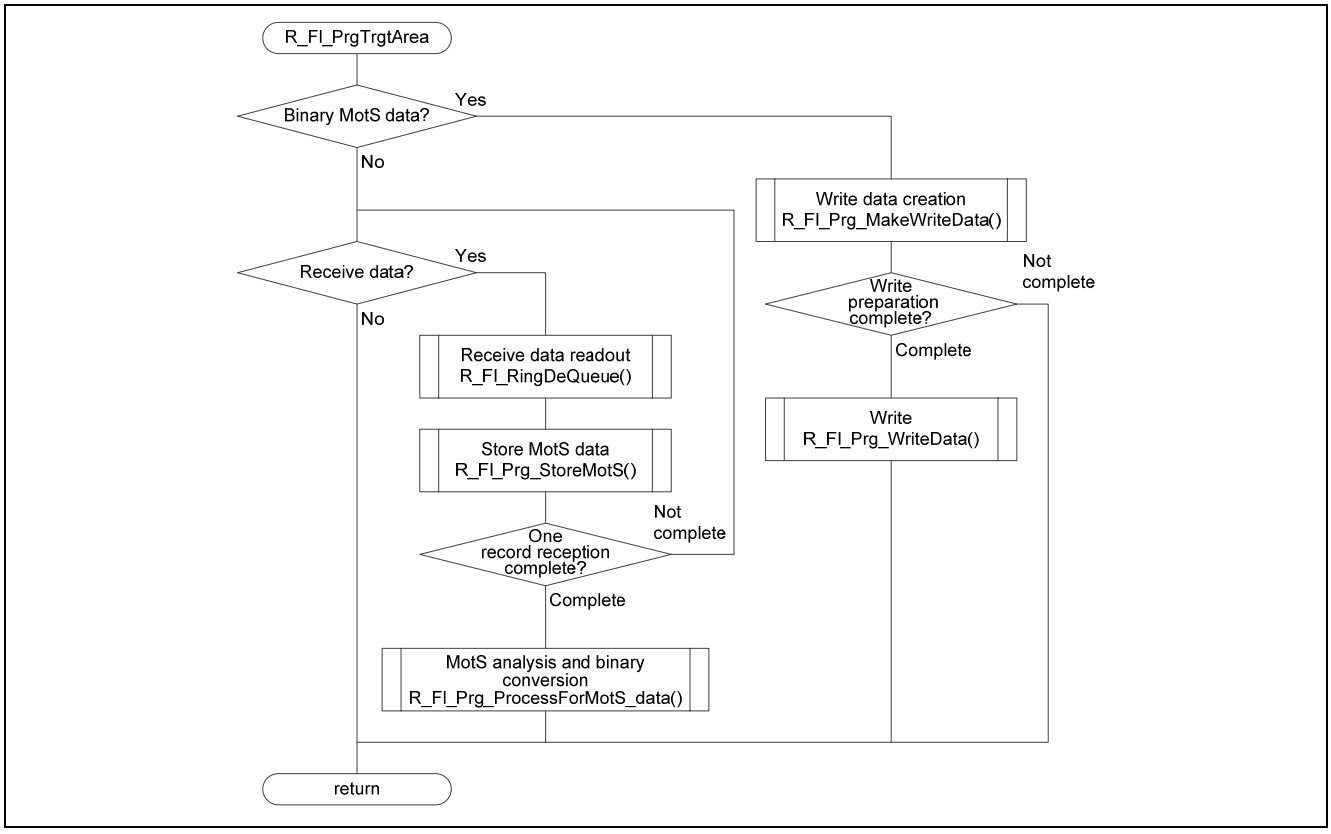


Figure 5.14 Write Mode Processing

5.14.7 Execute Mode Processing

Figure 5.15 shows the flowchart for the execute mode processing.

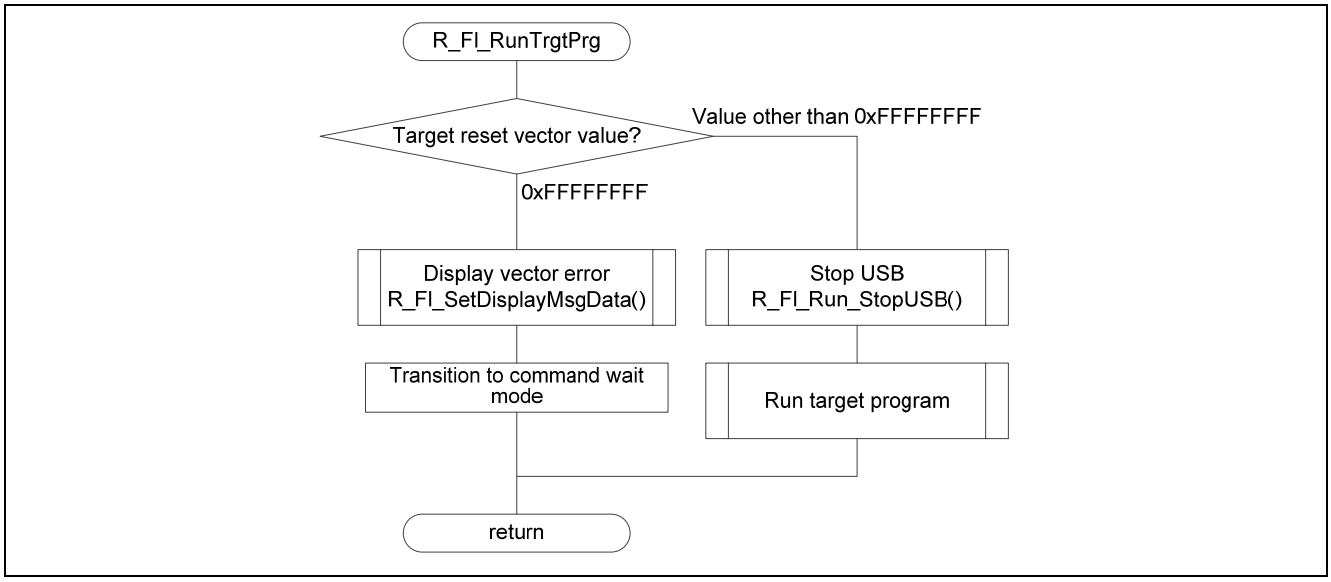


Figure 5.15 Execute Mode Processing

5.14.8 Error Termination Wait Mode Processing

Figure 5.16 shows the flowchart for the error termination wait mode processing.

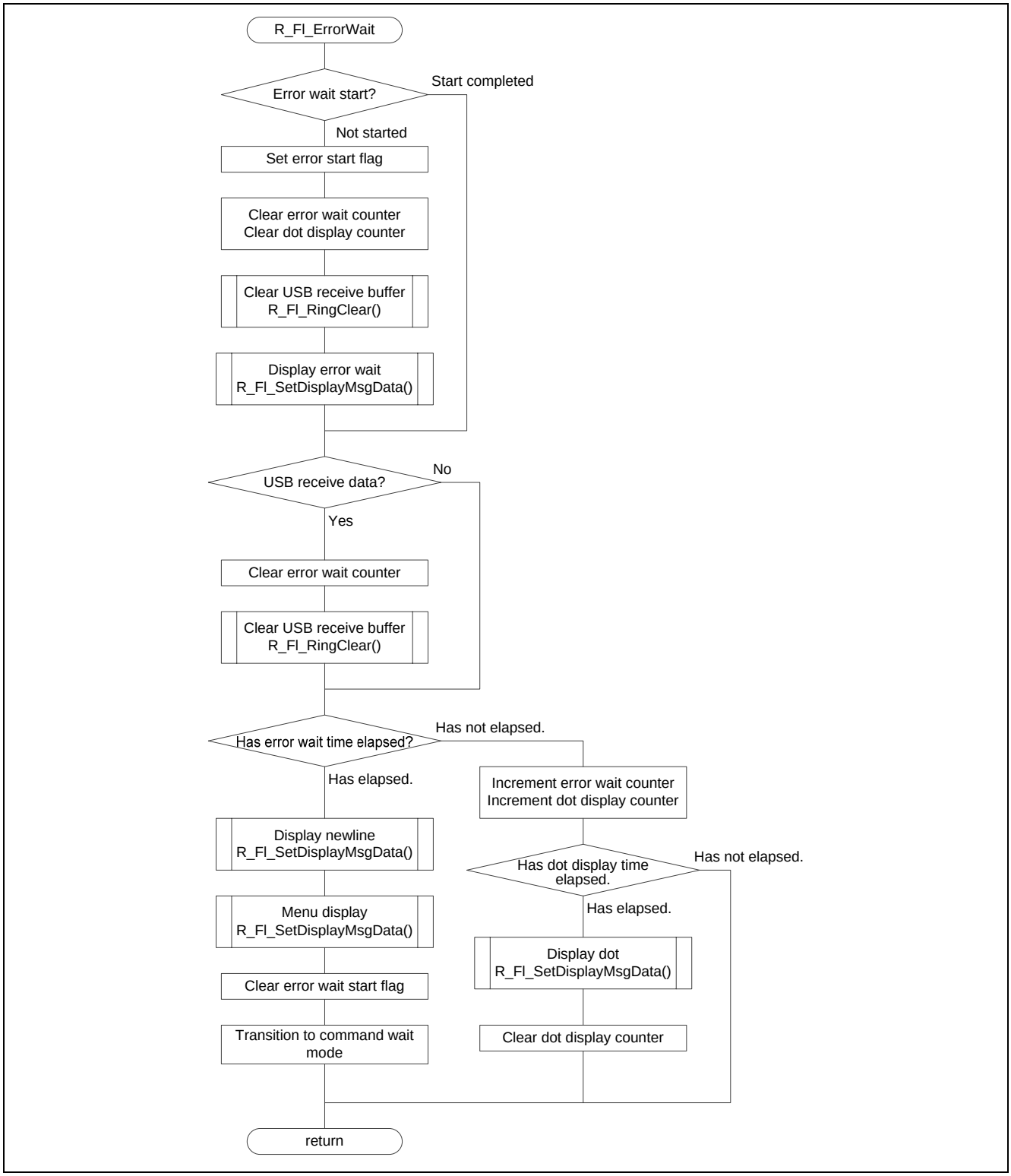


Figure 5.16 Error Termination Wait Mode Processing

5.14.9 Menu Display

Figure 5.17 shows the flowchart for the menu display.

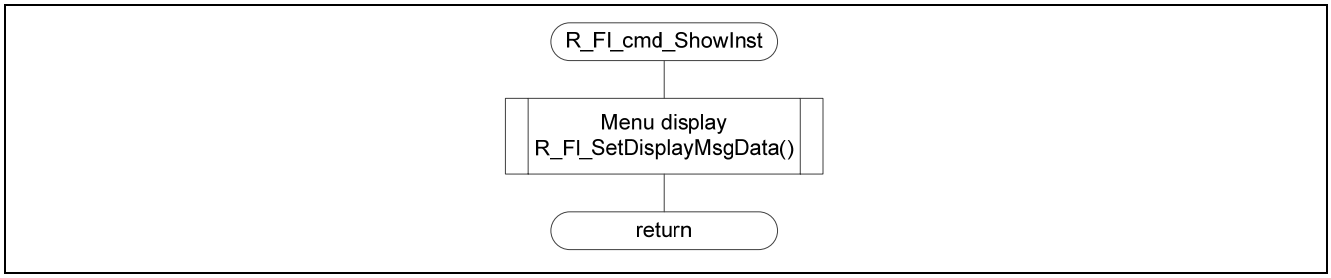


Figure 5.17 Menu Display

5.14.10 Start Blank Check

Figure 5.18 shows the flowchart for the start blank check.

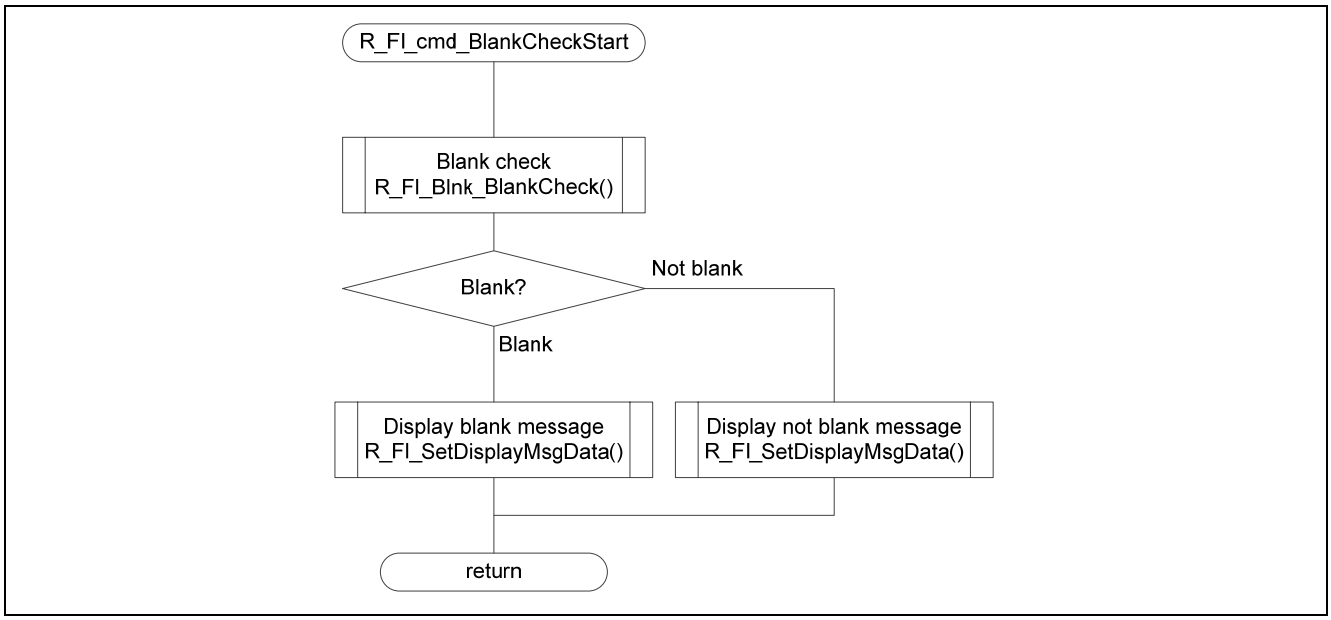


Figure 5.18 Start Blank Check

5.14.11 Start Erase

Figure 5.19 shows the flowchart for the start erase.

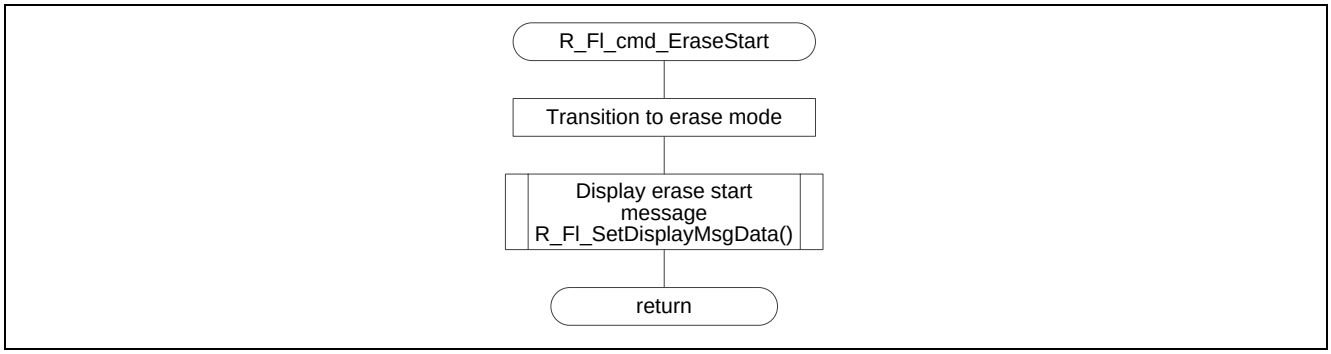


Figure 5.19 Start Erase

5.14.12 Start Write

Figure 5.20 shows the flowchart for the start write.

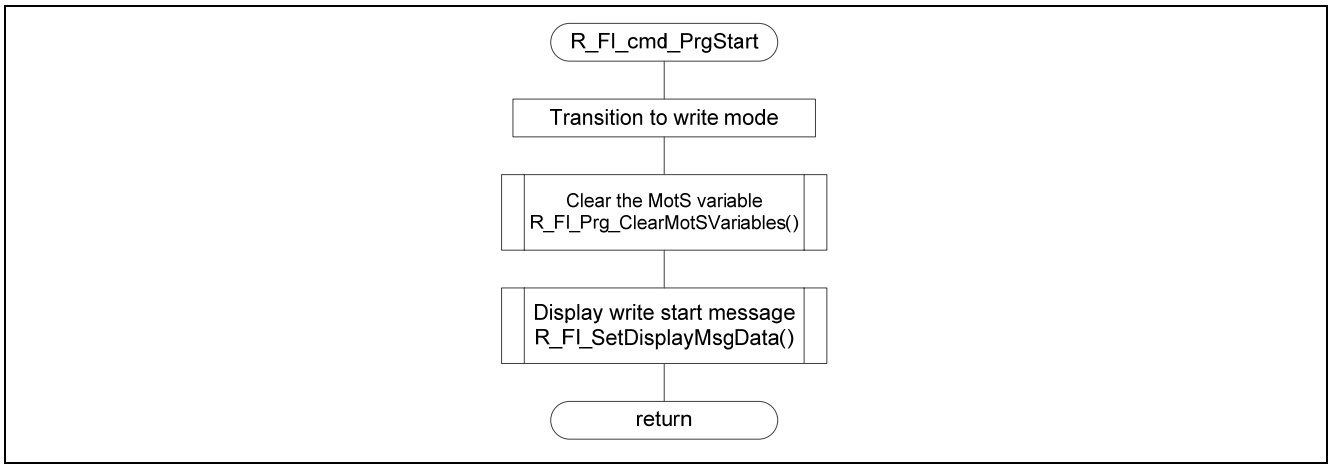


Figure 5.20 Start Write

5.14.13 Start Target Program Execution

Figure 5.21 shows the flowchart for the start target program execution.

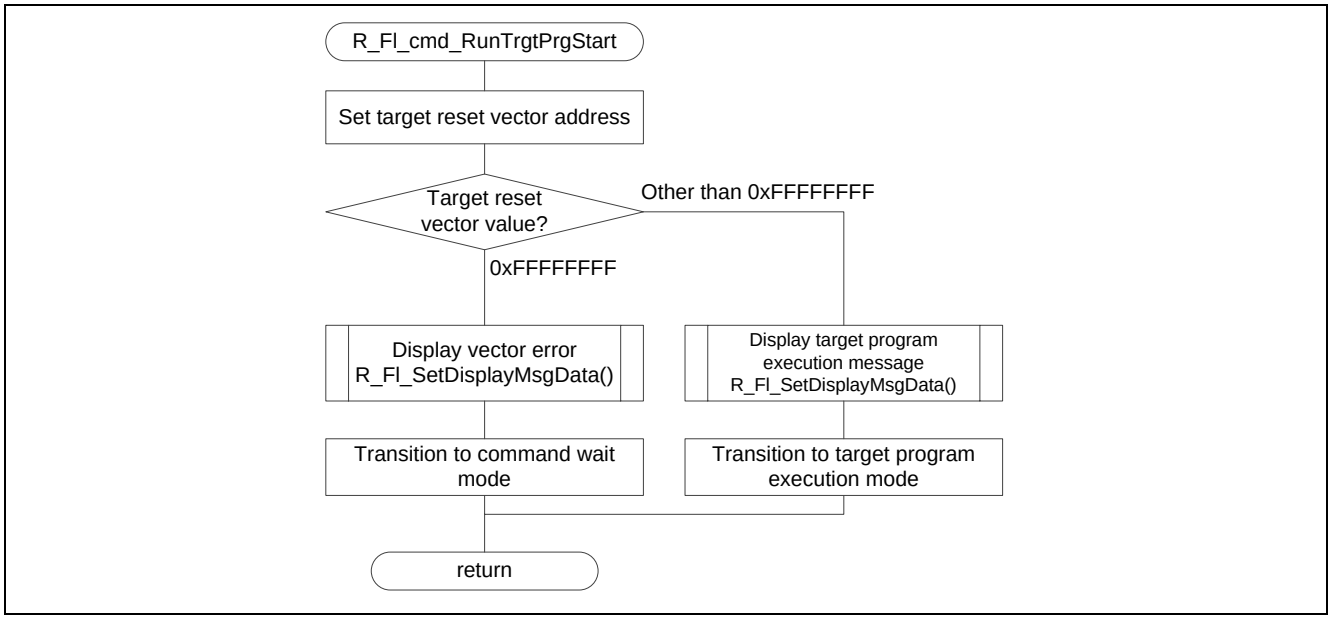


Figure 5.21 Start Target Program Execution

5.14.14 Blank Check

Figure 5.22 shows the flowchart for the blank check.

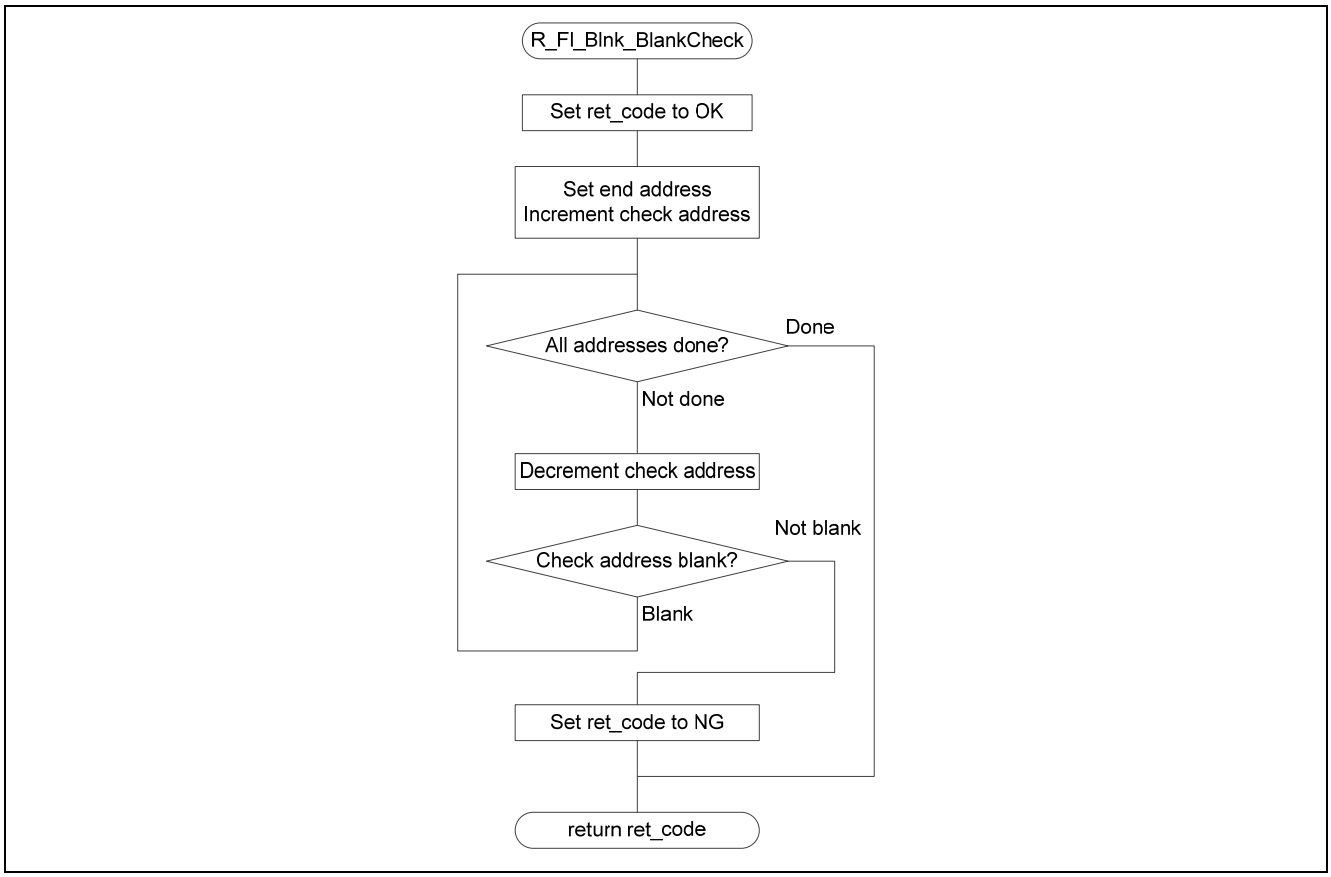


Figure 5.22 Blank Check

5.14.15 Target Area Erase

Figure 5.23 shows the flowchart for the target area erase.

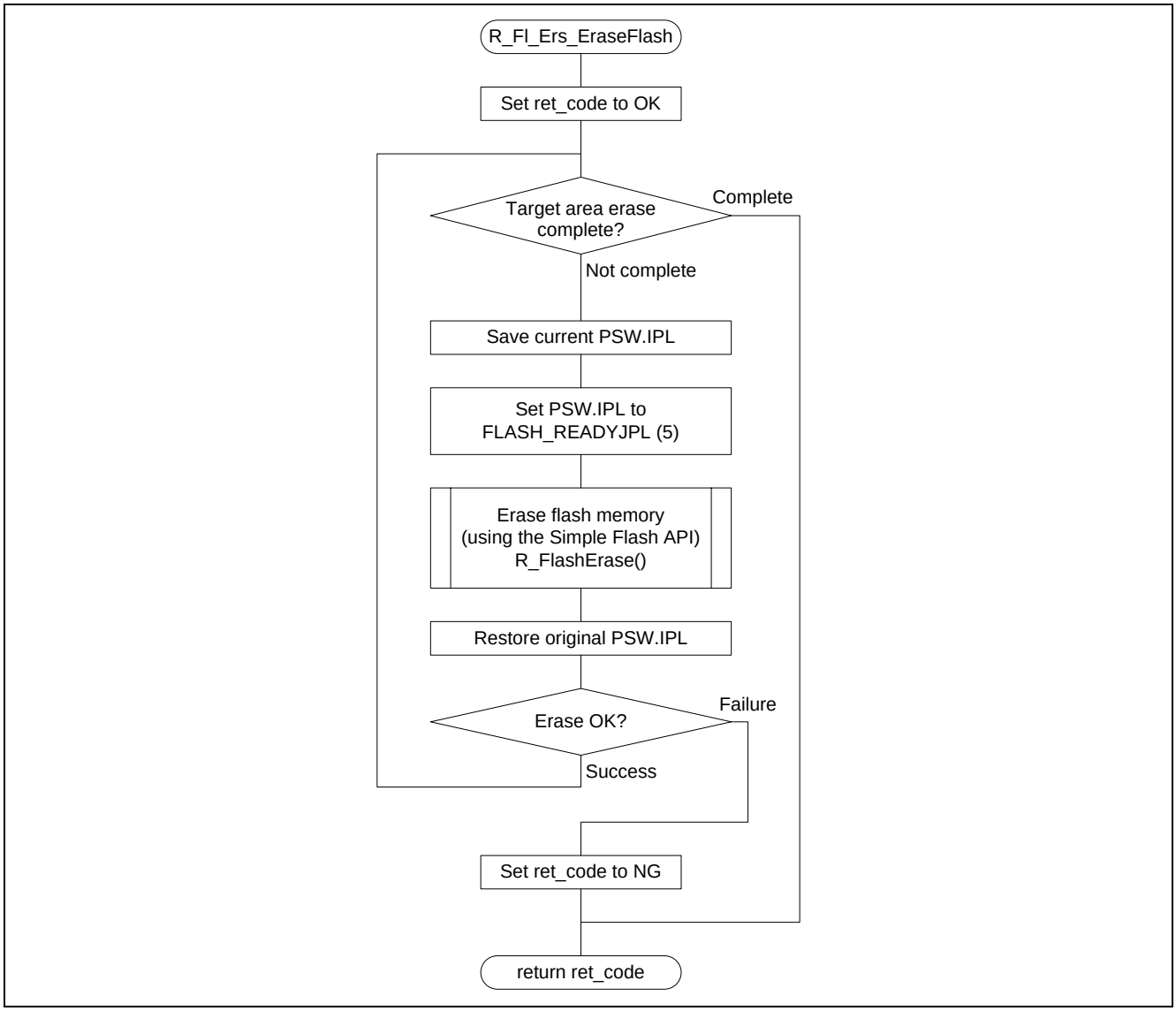


Figure 5.23 Target Area Erase

5.14.16 Store Motorola S Format Data

Figure 5.24 shows the flowchart for the store Motorola S format data.

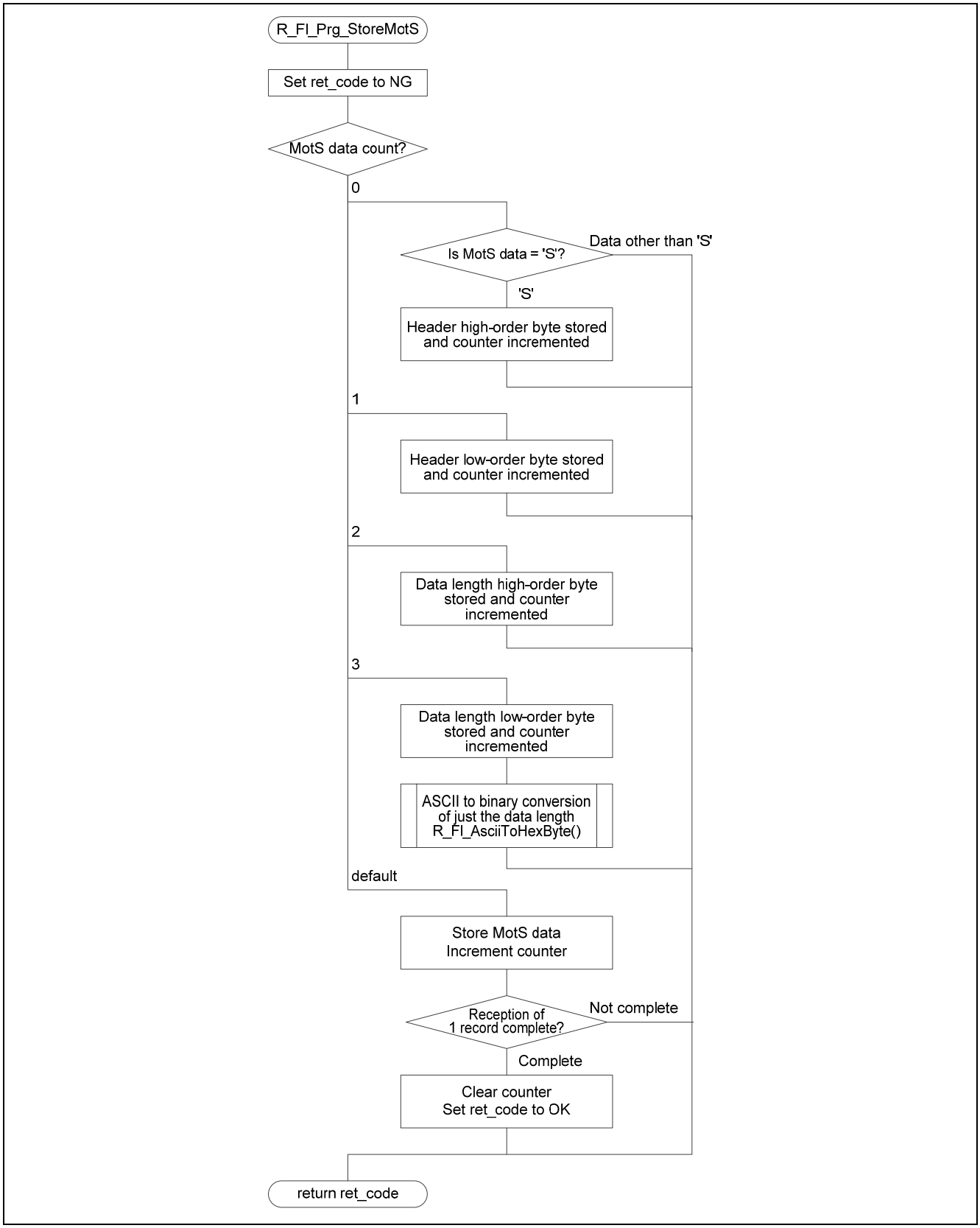


Figure 5.24 Store Motorola S Format Data

5.14.17 Motorola S Format Header Analysis and Binary Conversion

Figures 5.25 and 5.26 show the flowcharts for the Motorola S format header analysis and binary conversion.

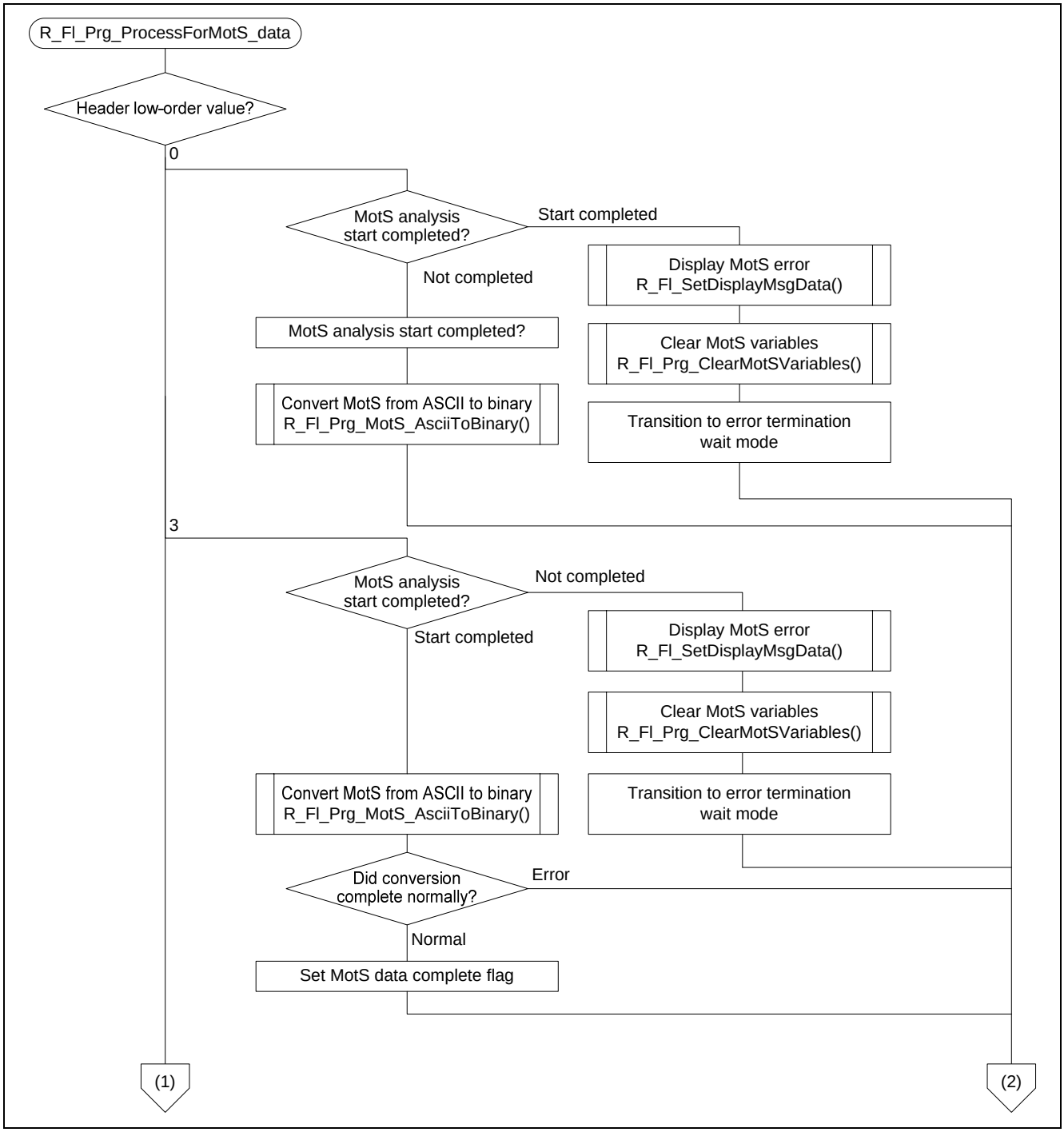


Figure 5.25 Motorola S Format Header Analysis and Binary Conversion (1)

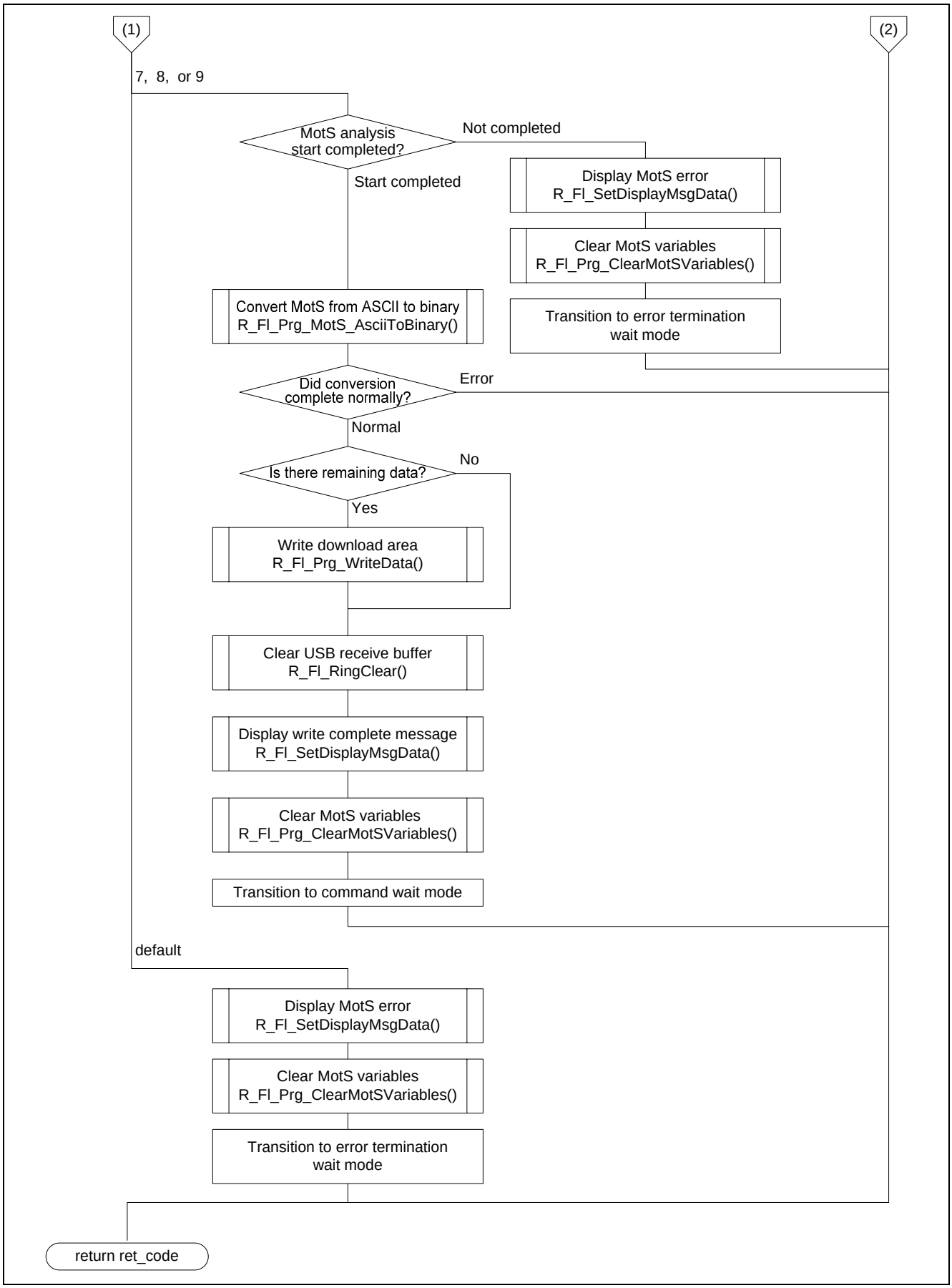


Figure 5.26 Motorola S Format Header Analysis and Binary Conversion (2)

5.14.18 Motorola S Format ASCII to Binary Conversion

Figure 5.27 shows the flowchart for the Motorola S format ASCII to binary conversion.

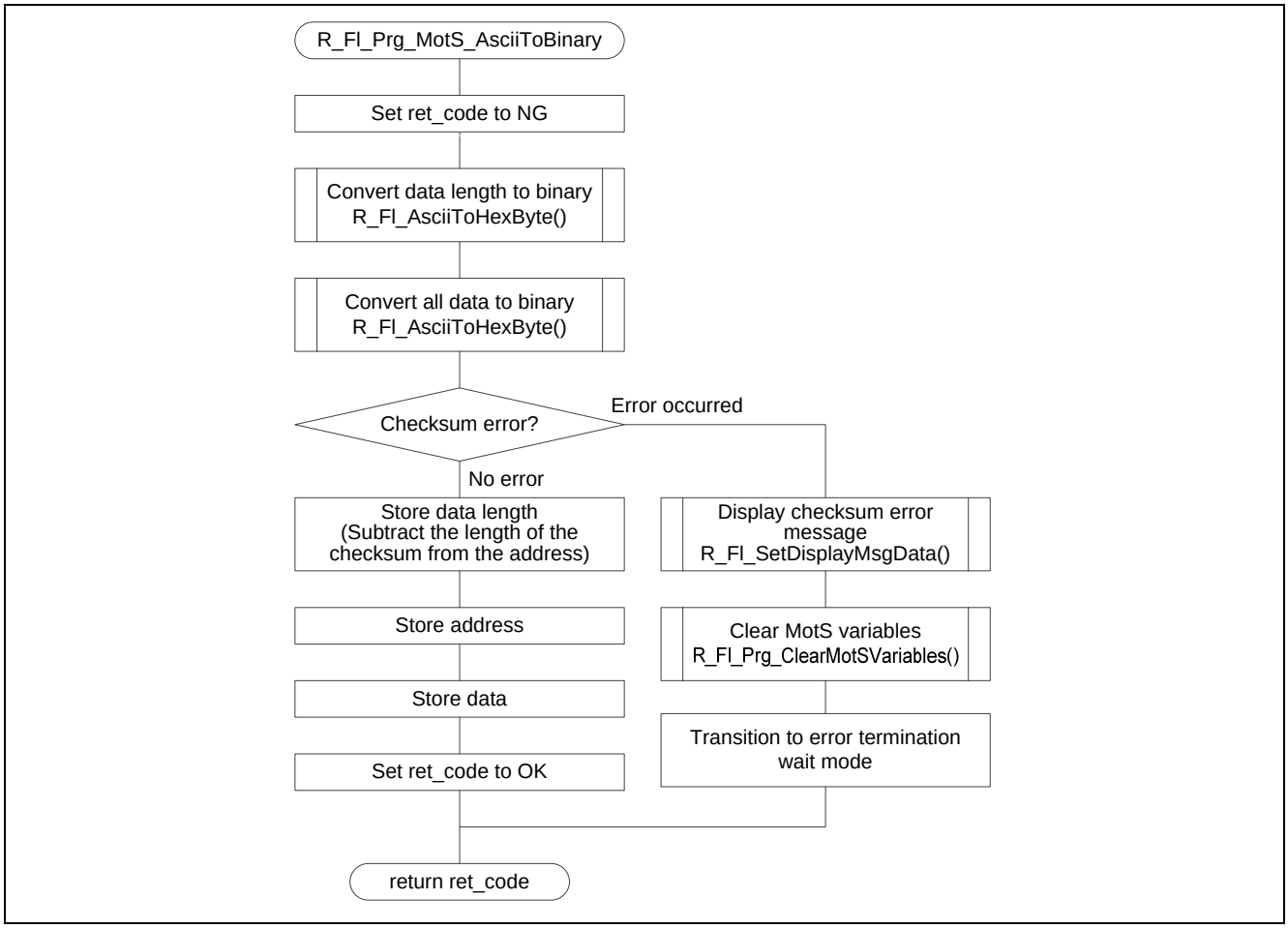


Figure 5.27 Motorola S Format ASCII to Binary Conversion

5.14.19 Create Target Area Write Data

Figure 5.28 shows the flowchart for the create target area write data.

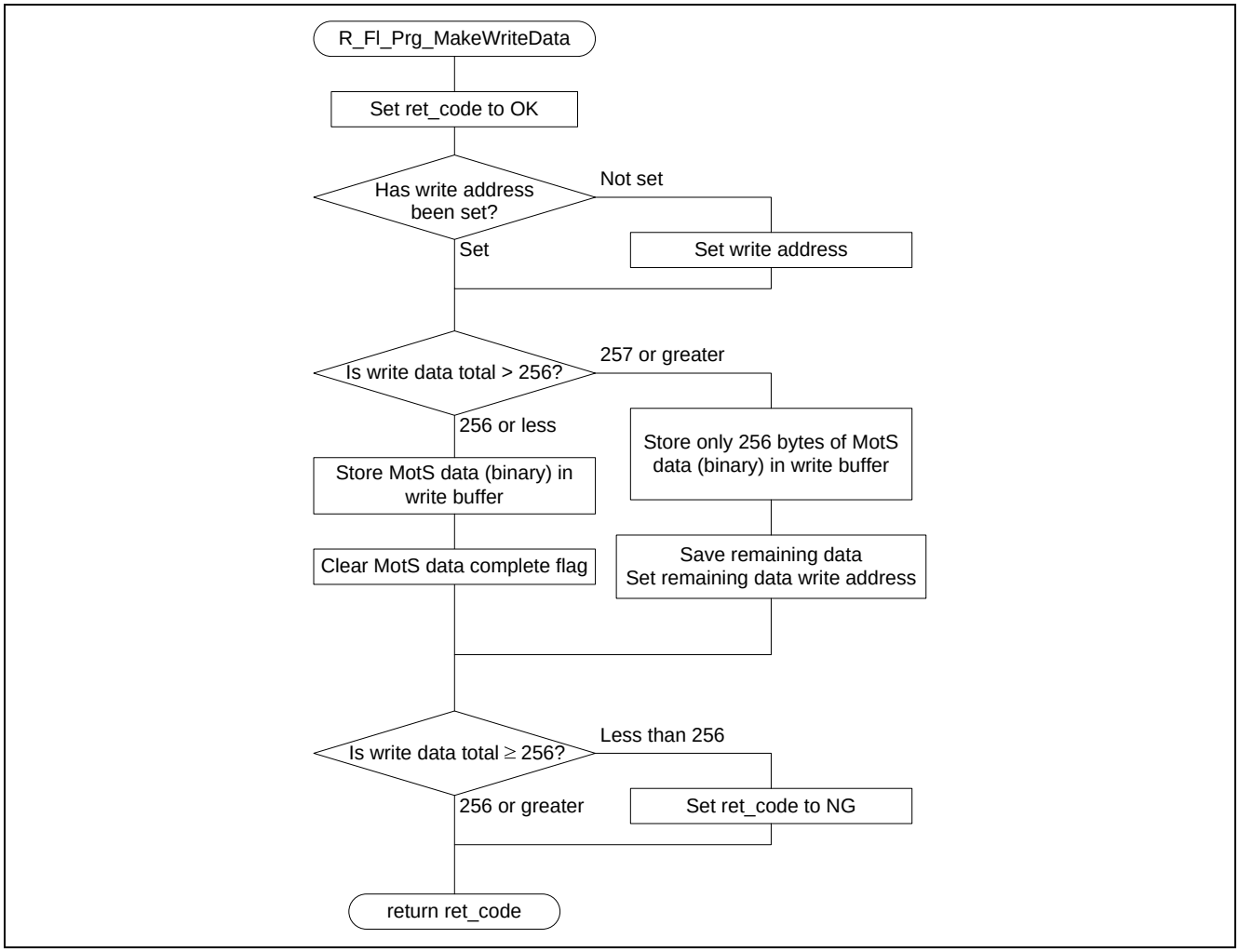


Figure 5.28 Create Target Area Write Data

5.14.20 Target Area Write

Figure 5.29 shows the flowchart for the target area write.

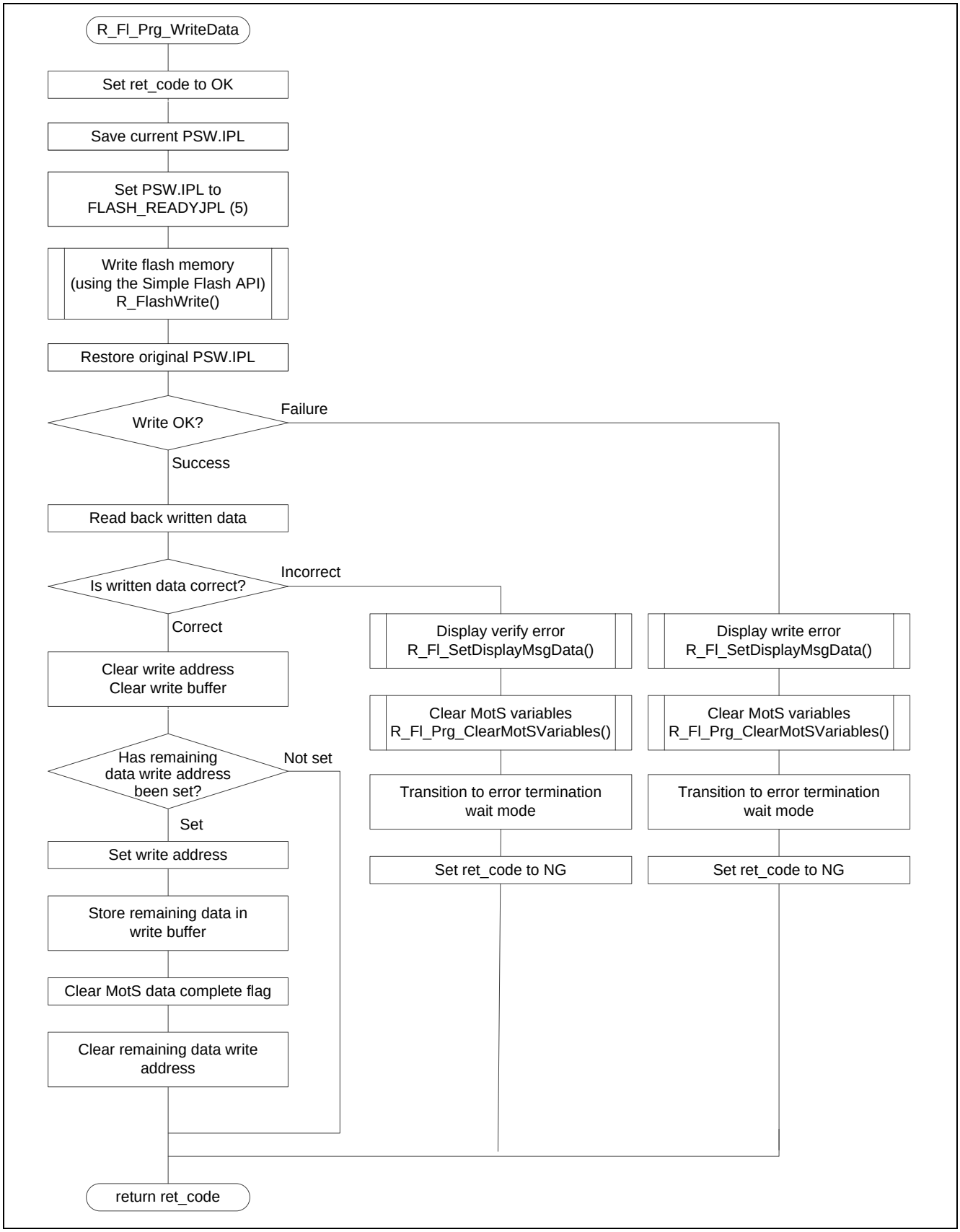


Figure 5.29 Target Area Write

5.14.21 Clear Motorola S Format Variables

Figure 5.30 shows the flowchart for the clear Motorola S format variables.

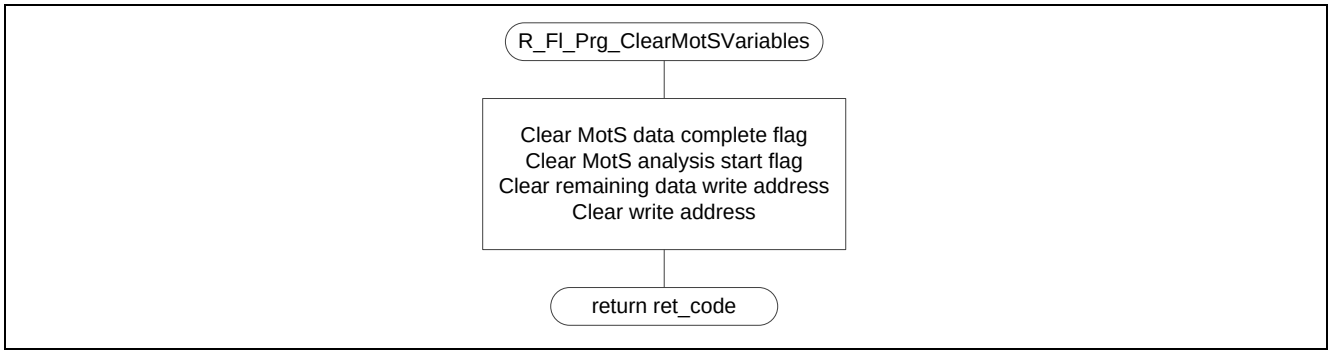


Figure 5.30 Clear Motorola S Format Variables

5.14.22 Stop USB

Figure 5.31 shows the flowchart for the stop USB.

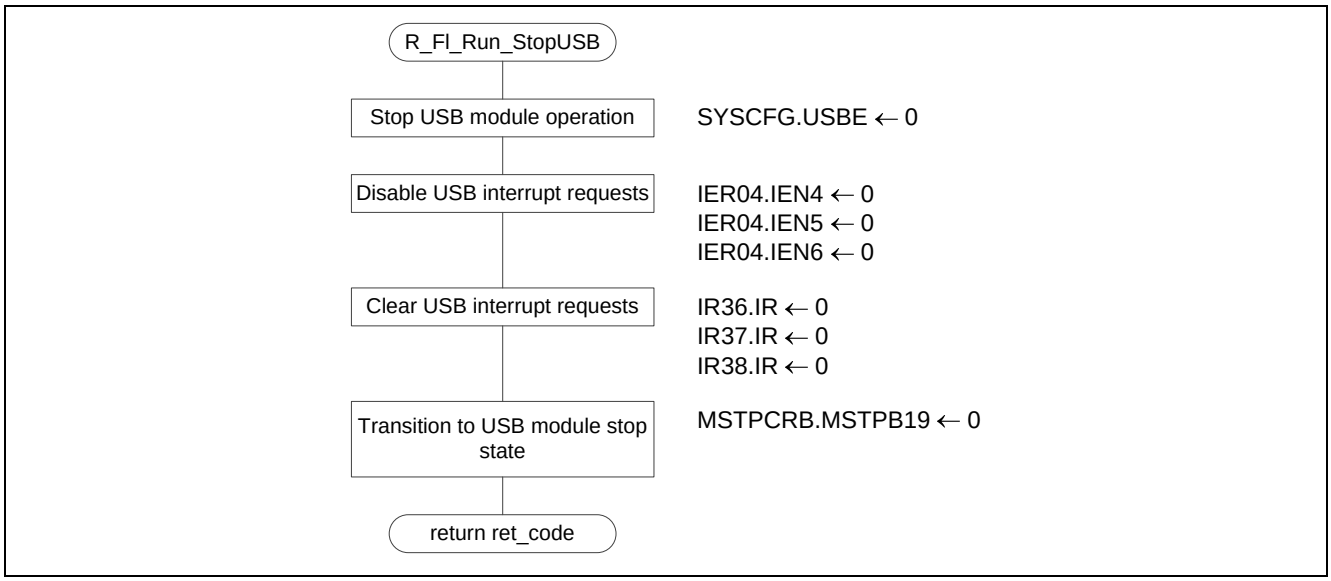


Figure 5.31 Stop USB

5.14.23 Store USB Receive Data

Figure 5.32 shows the flowchart for the store USB receive data.

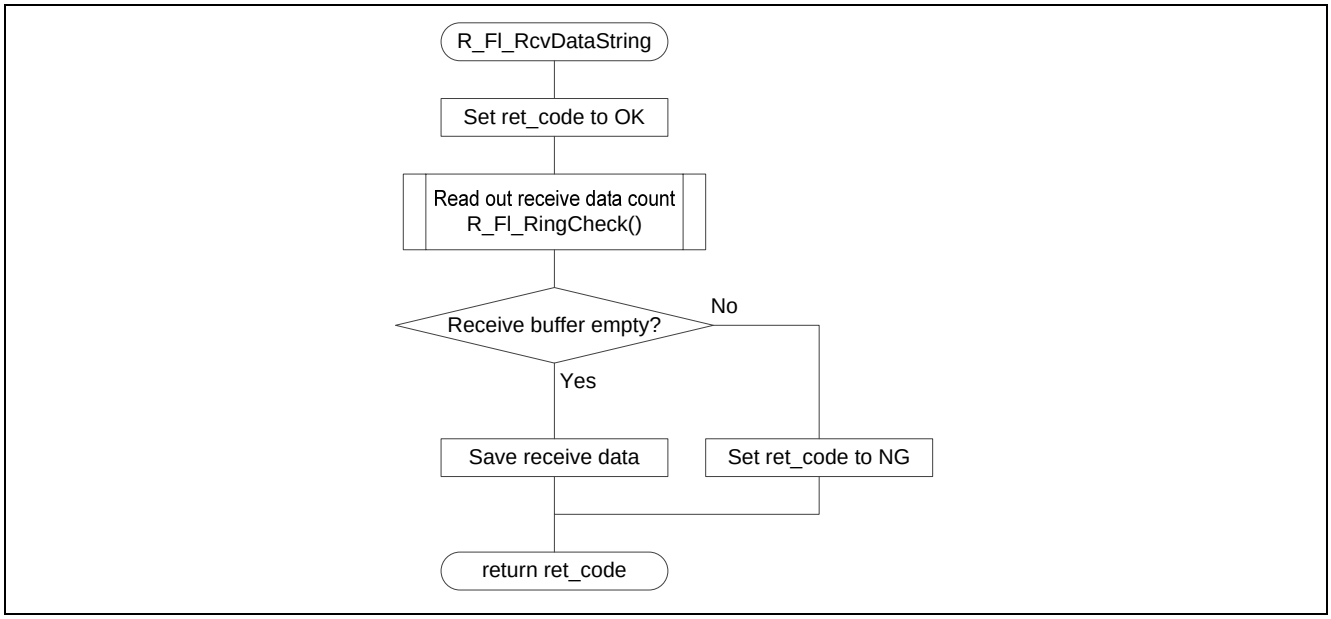


Figure 5.32 Store USB Receive Data

5.14.24 Store USB Transmit Data

Figure 5.33 shows the flowchart for the store USB transmit data.

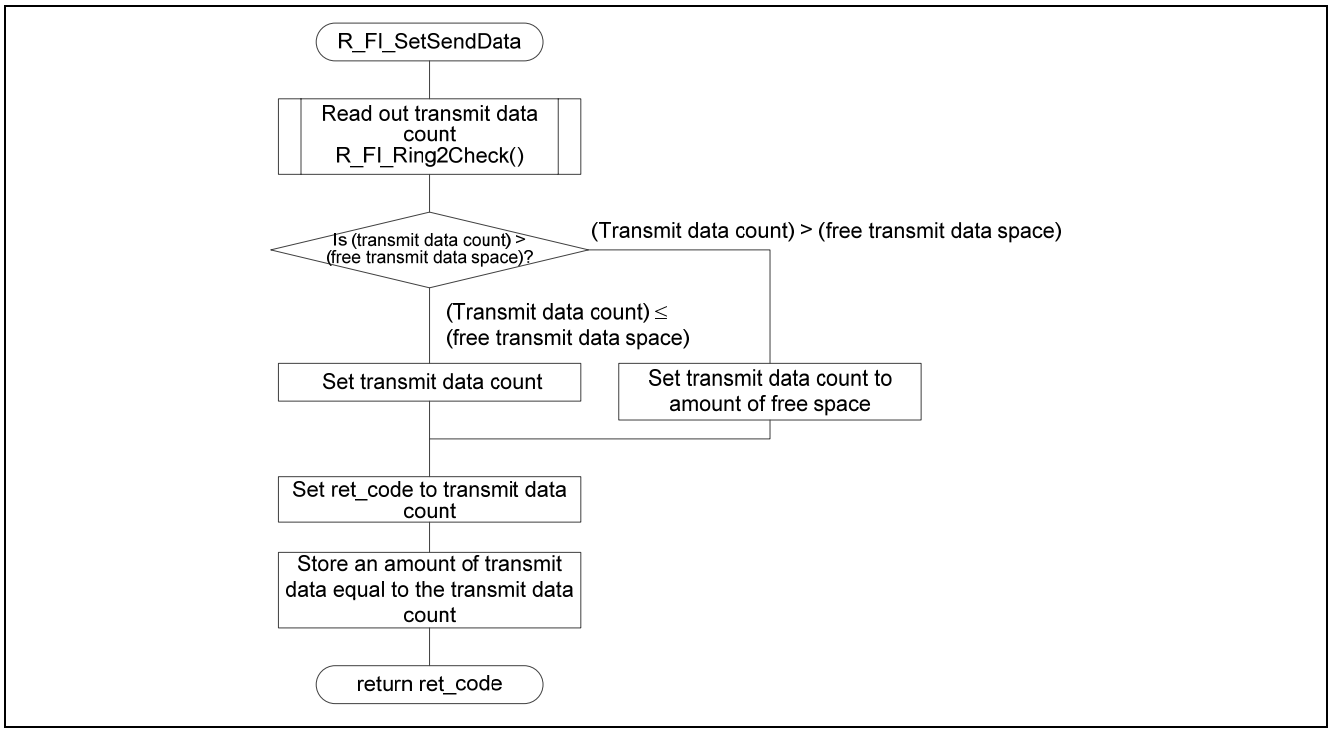


Figure 5.33 Store USB Transmit Data

5.14.25 Message Display

Figure 5.34 shows the flowchart for the message display.

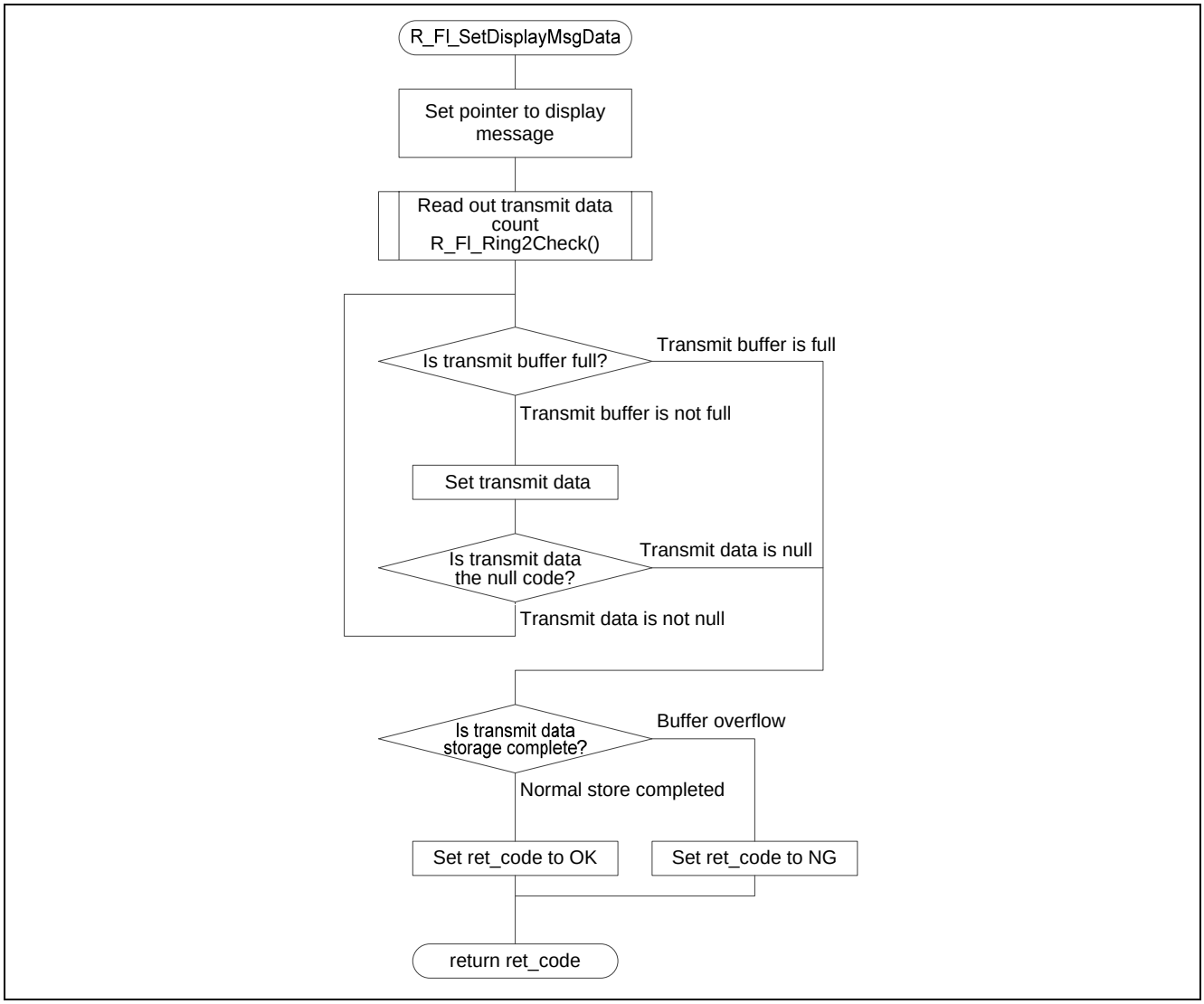


Figure 5.34 Message Display

5.14.26 Check Receive Ring Buffer Free Space

Figure 5.35 shows the flowchart for the check receive ring buffer free space.

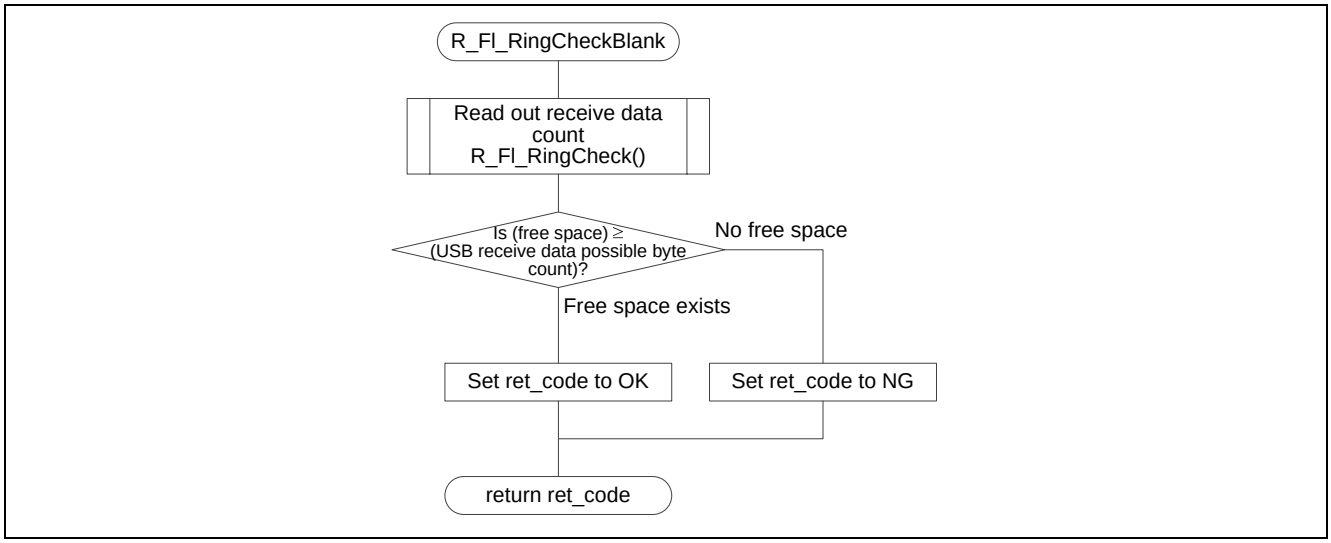


Figure 5.35 Check Receive Ring Buffer Free Space

5.14.27 Transmit Ring Buffer Data Check

Figure 5.36 shows the flowchart for the transmit ring buffer data check.

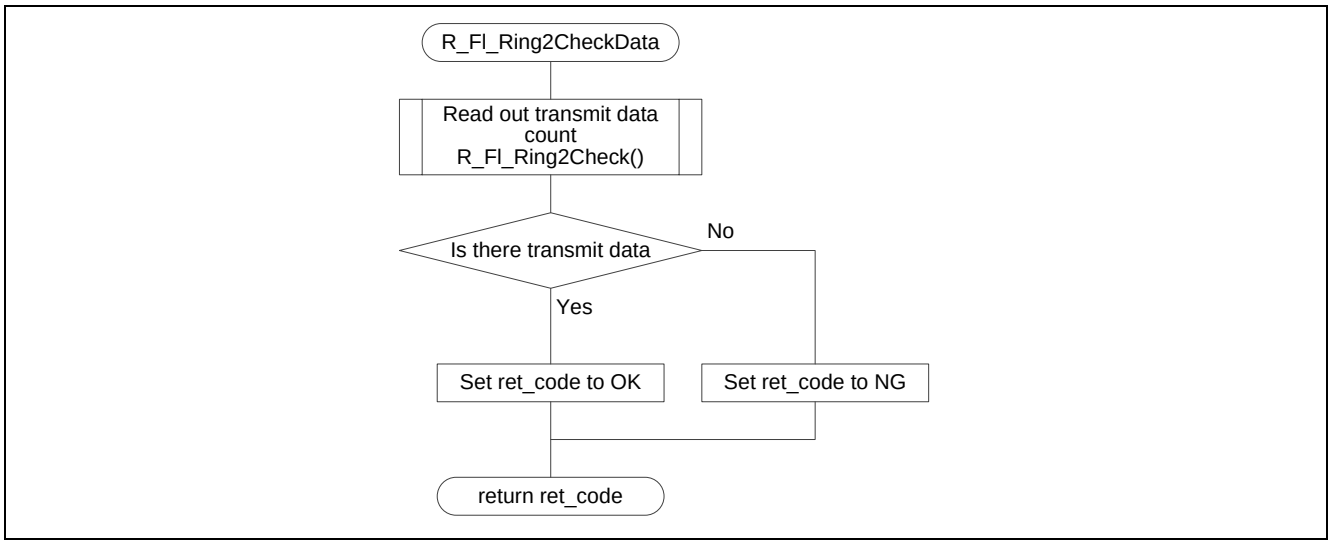


Figure 5.36 Transmit Ring Buffer Data Check

5.14.28 Run Target Program when USB Disconnected

Figure 5.37 shows the flowchart for the run target program when USB disconnected.

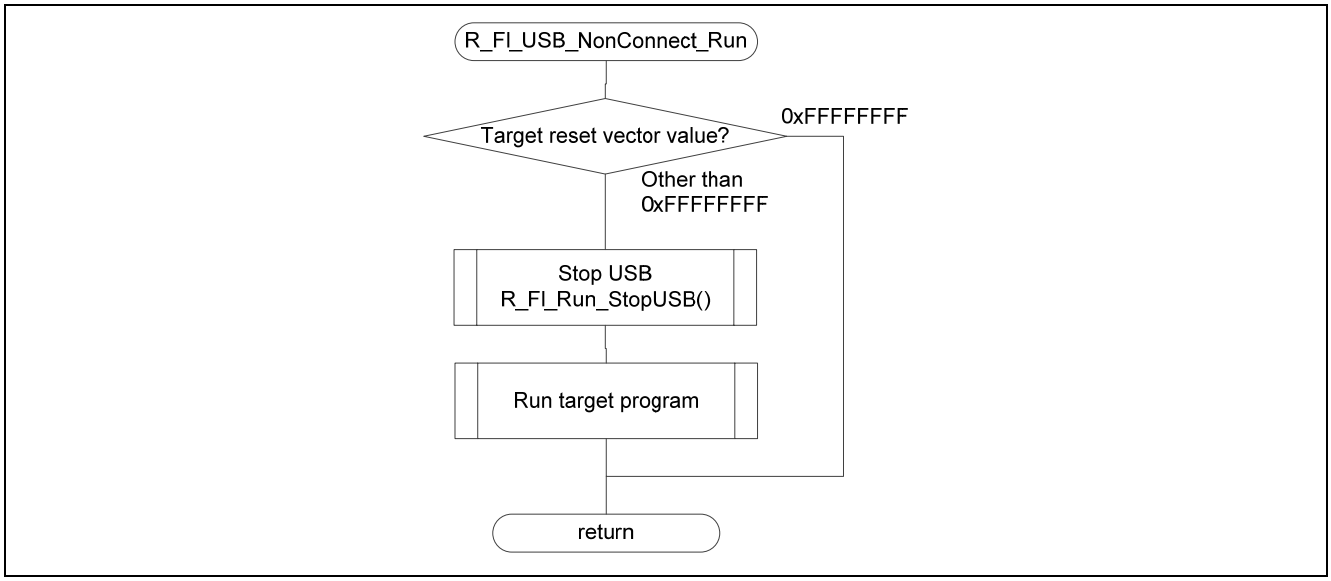


Figure 5.37 Run Target Program when USB Disconnected

5.14.29 Store Data in Receive Ring Buffer

Figure 5.38 shows the flowchart for the store data in receive ring buffer.

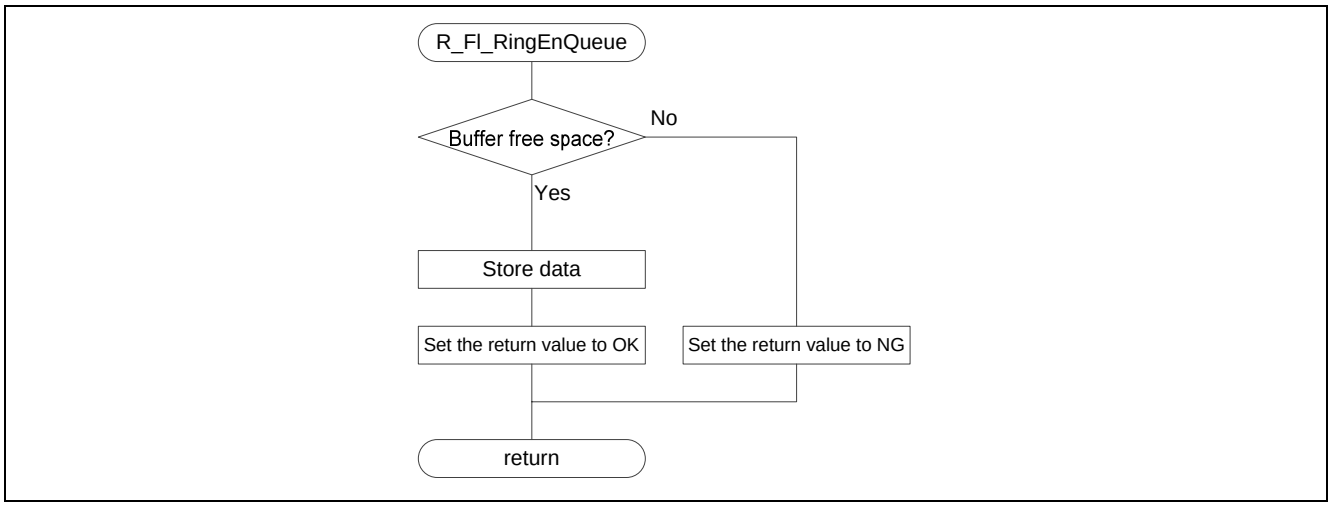


Figure 5.38 Store Data in Receive Ring Buffer

5.14.30 Read Data from Receive Ring Buffer

Figure 5.39 shows the flowchart for the read data from receive ring buffer.

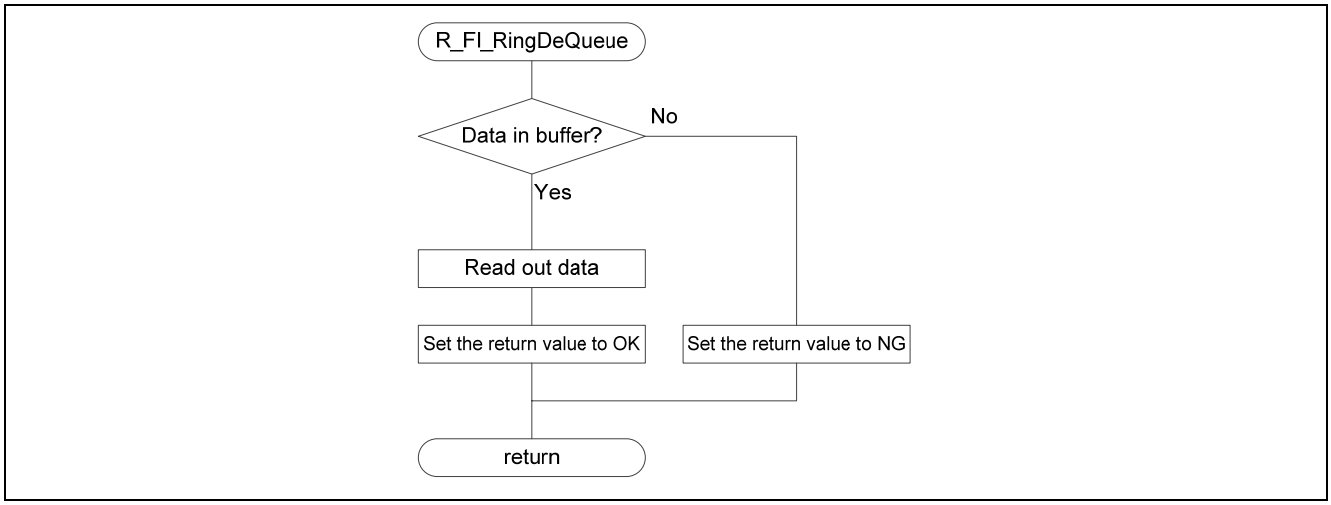


Figure 5.39 Read Data from Receive Ring Buffer

5.14.31 Clear Receive Ring Buffer

Figure 5.40 shows the flowchart for the clear receive ring buffer.

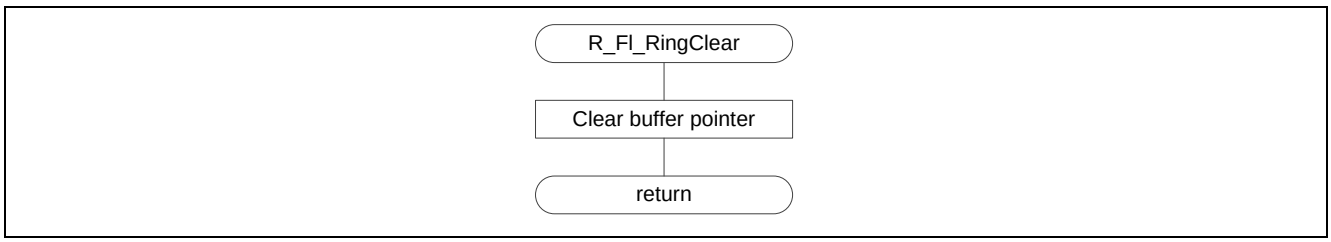


Figure 5.40 Clear Receive Ring Buffer

5.14.32 Check Data Count in Receive Ring Buffer

Figure 5.41 shows the flowchart for the check data count in receive ring buffer.

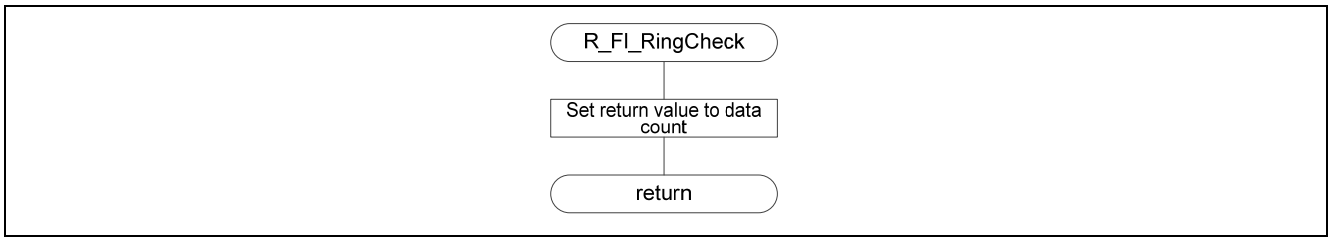


Figure 5.41 Check Data Count in Receive Ring Buffer

5.14.33 Store Data in Transmit Ring Buffer

Figure 5.42 shows the flowchart for the store data in transmit ring buffer.

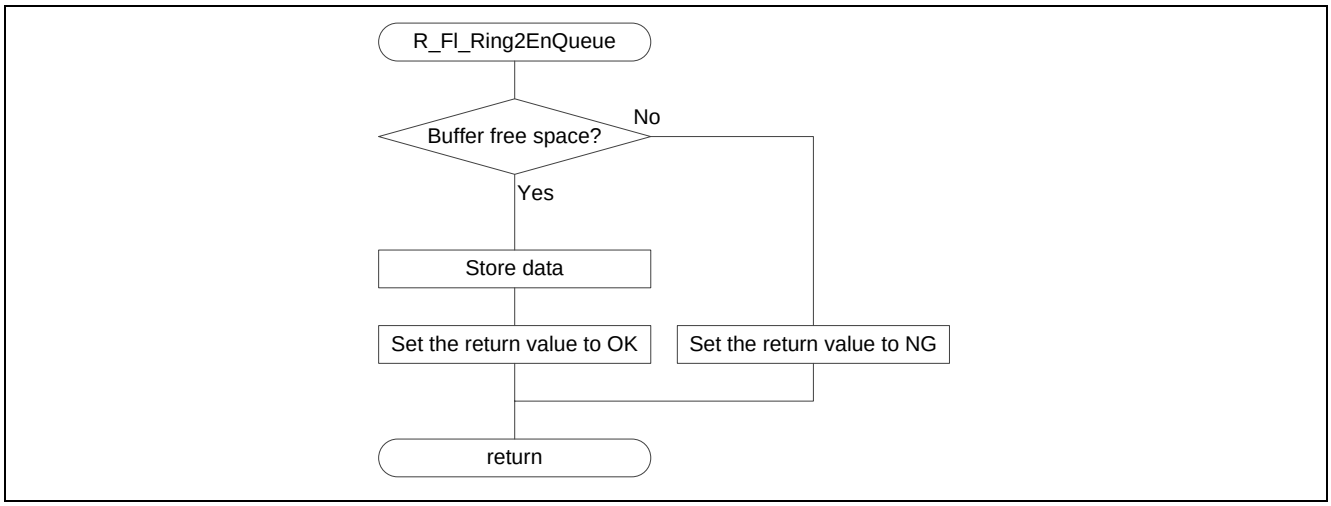


Figure 5.42 Store Data in Transmit Ring Buffer

5.14.34 Read Data from Transmit Ring Buffer

Figure 5.43 shows the flowchart for the read data from transmit ring buffer.

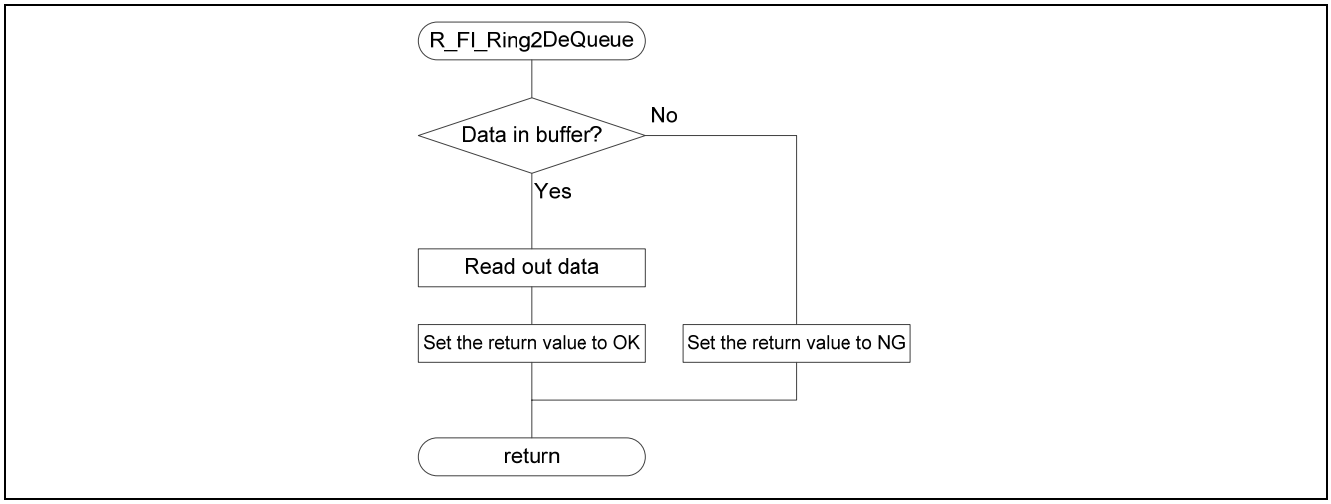


Figure 5.43 Read Data from Transmit Ring Buffer

5.14.35 Clear Transmit Ring Buffer

Figure 5.44 shows the flowchart for the clear transmit ring buffer.

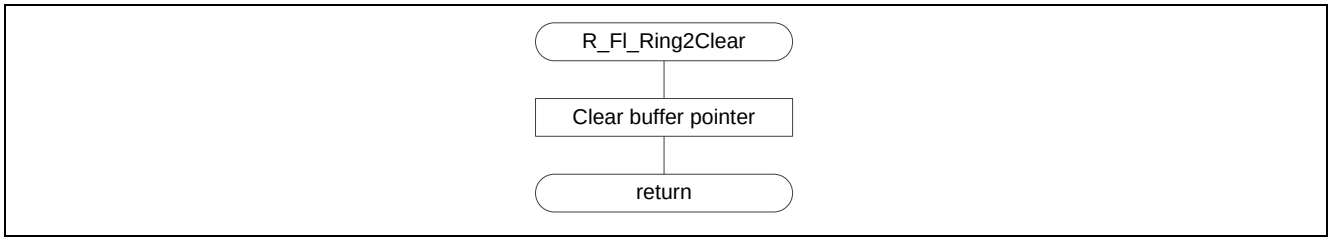


Figure 5.44 Clear Transmit Ring Buffer

5.14.36 Check Data Count in Transmit Ring Buffer

Figure 5.45 shows the flowchart for the check data count in transmit ring buffer.

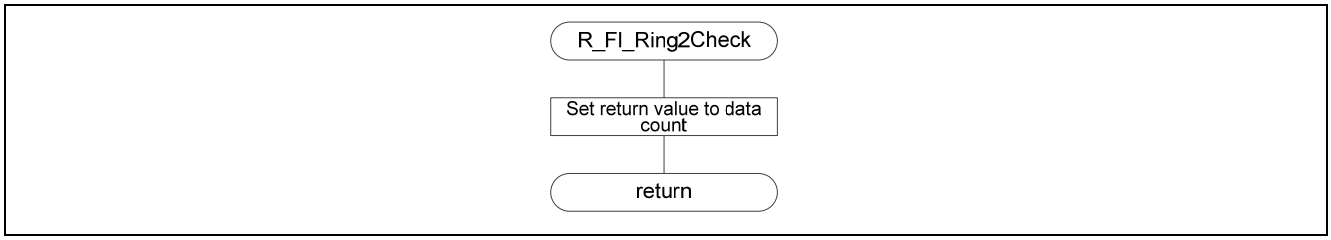


Figure 5.45 Check Data Count in Transmit Ring Buffer

5.14.37 Convert from ASCII Code to Binary Data

Figure 5.46 shows the flowchart for the convert from ASCII code to binary data.

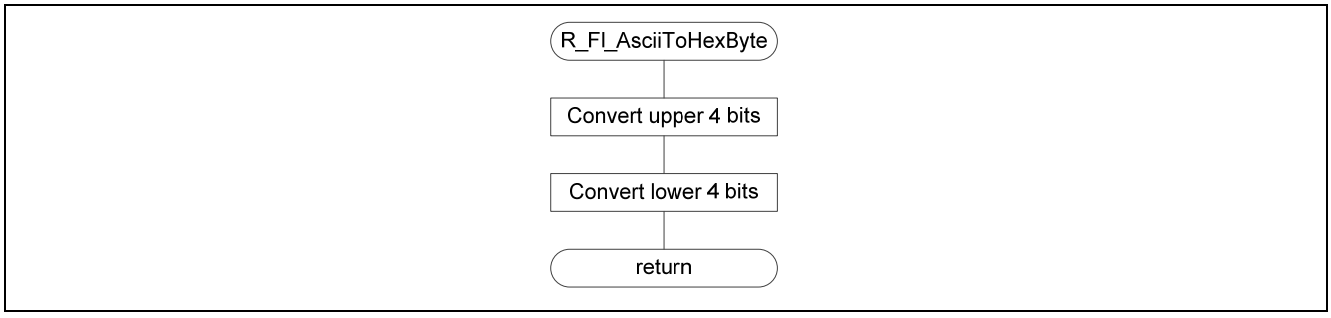


Figure 5.46 Convert from ASCII Code to Binary Data

6. Usage

(1) Change the sample code vendor ID and product ID.
Change USB_VENDORID and USB_PRODUCTID in the r_usb_cDefUsr.h file to match the environment being used. See the USB peripheral communication device class driver and USB basic firmware application notes for details on these changes.

```
/***********************************************************************
Macro definitions (User define PERIPHERAL)
***********************************************************************/

#define  USB_VENDORID     0x0000u     /* Vendor  ID */
#define  USB_PRODUCTID    0x0000u     /* Product ID */

#define  USB_BCDNUM       0x0200u     /* bcdUSB */
#define  USB_RELEASE      0x0100u     /* Release Number */
#define  USB_DCPMAXP      64u         /* DCP max packet size */
#define  USB_CONFIGNUM    1u          /* Configuration number */
#define  USB_STRINGNUM    7u          /* Max of string descriptor */
```

Figure 6.1 Sample Code Vendor ID and Product ID (r_usb_cDefUsr.h)

(2) Change the USB driver vendor ID and product ID.
Change VID_000 and PID_000 in the CDC_Demo.inf file in the usb_driver folder. The changed values must be the same as those used for the sample code vendor ID and product ID as set in item (1) above. See the USB peripheral communication device class driver and USB basic firmware application notes for details on these changes.

```
[Manufacturer]
%STRING_MAUNUFACTURER%=Model

[Model]
%STRING_MODEL%=CDC, USB\VID_0000&PID_0000
```

Figure 6.2 USB Driver Vendor ID and Product ID (CDC_Demo.inf)

(3) Build all the sample code and download it to the microcontroller.
(4) After powering up the microcontroller, connect the PC and the microcontroller with the USB bus. (See note 1.)
(5) At the Windows USB driver install screen, select CDC_Demo.inf in the usb_driver folder to install the driver. This step is not required if the USB driver has already been installed. (See note 1.)
(6) Run the terminal emulator and connect the microcontroller.
Note that the USB peripheral communication device class driver will be displayed a new "COM" in the terminal emulator.
(7) After this, follow the instructions displayed as messages on the terminal emulator.

Note 1: If a value other than FFFF FFFFh is written as the target reset vector, and the USB remains in the disconnected state for longer than a certain fixed time, the target program will be executed. Note that the USB will be stopped when the target program runs. Since it may take some amount of time for USB connection when the USB driver is installed, FFFF FFFFh should be written to the target reset vector.

7. Sample Target Program

A sample target program (UsrPrgSample.zip) is included with this application note. It is a program that lights the LEDs on the board in order as shown in section 2, Operation Check Conditions. Use this program as reference material regarding setting the target reset vector and setting up sections. Note that this target program assumes that 512 KB of the ROM capacity is used.

8. Notes

8.1 Write Speed

With certain terminal emulators, the write speed can be extremely slow. This is because some terminal emulators take 1 ms to transfer each byte from the PC to the microcontroller. If you find that the write speed is extremely slow, try using a different terminal emulator.

8.2 USB Disconnection During Write or Erase

Do not disconnect or reconnect the USB cable during write to or erase of the target area.

8.3 HEW Settings

The sample code in ROM is copied to RAM during flash memory programming. See the RX600 Series RX600 Simple Flash API application note for details on these settings.

8.4 USB Vendor ID and Product ID

The USB vendor ID and product ID must be modified to use the sample code. See section 6, Usage, and the USB peripheral communication device class driver and USB basic firmware application notes for details.

8.5 Fixed Vector Table Interrupts

Of the interrupts allocated to the fixed vector table, the sample code only uses the reset interrupt. If any other fixed vector table interrupts are used, the sample code must be modified to allow that usage.

8.6 Target Program Reset Vector

The execution start position of the target program downloaded by the sample code is determined by the value of the target reset vector (FFFF FFFCh). Therefore the target program must be set up so that the target vector is allocated at location FFFF FFFCh. See section 5.3, Target Program Execution Start Position, for details.

See section 7, Sample Target Program, for an example of a target program.

8.7 Motorola S Format

The Motorola S formats supported by the sample code is limited to the S0, S3, and S7 Motorola S formats only. Also, only increasing addresses are supported. Do not transmit mot files with formats that include decreasing addresses or out-of-order addresses.

8.8 "while(1)" Statements Processing

Note that if a transmit buffer overflow occurs for the USB interface, the sample code deadlocks with a "while(1)" statement.

8.9 Program Stops During USB Communication

If the microcontroller is reset and execution restarted with the terminal emulator on the PC connected to the microcontroller, it may not be possible to resume normal communication. In this case, exit from the terminal emulator and then reconnect the microcontroller to the PC.

8.10 Endian Order

This sample code only supports the little endian order.

8.11 Changes to the RX600 Simple Flash API

This flash boot loader uses the RX600 Simple Flash API. See the RX600 Simple Flash API application note for the Simple Flash API specifications. Below, we note the places that have been changed for use in this application note.

The following RX600 Simple Flash API files have been changed: r_flash_api_rx600_config.h and mcu_info.h.

- Changes to r_flash_api_rx600_config.h
  (1) The processor status word (PSW) processor interrupt priority level (IPL) is changed to the value specified with the macro definition shown below to prevent ROM access due to interrupts during write and erase operations. It is set to the value 5 in this application note.
  Macro definition: #define FLASH_READY_IPL 5
  (2) Certain Simple Flash API settings are changed.
  Before modification: #define IGNORE_LOCK_BITS
  #define COPY_CODE_BY_API
  #define FLASH_API_USE_R_BSP
  After modification: //#define IGNORE_LOCK_BITS
  //#define COPY_CODE_BY_API
  //#define FLASH_API_USE_R_BSP

- Changes to mcu_info.h
  (1) The files stored in the folder r_bsp/board/rskrx62n in the Simple Flash API directory are used.
  (2) Certain Simple Flash API settings are changed.
  Before modification: #define ICLK_HZ (96000000)
  #define PCLK_HZ (48000000)
  #define BCLK_HZ (12000000)
  After modification: #define ICLK_HZ (48000000)
  #define PCLK_HZ (24000000)
  #define BCLK_HZ (48000000)

8.12 Chances to the USB Peripheral Communication Device Class Driver

This sample code reuses the code in the USB peripheral communication device class driver. See the USB peripheral communication device class driver and USB basic firmware application notes for details on and specifications of the USB peripheral communication device class driver.

8.12.1 Changes

Three files, r_usb_PCDC_apl.c, dbsct_pcdc.c, and resetprg.c, in the USB peripheral communication device class driver have been changed.

- Changes to r_usb_PCDC_apl.c:
  (1) Addition of files to be included
  Added: #include "r_Flash_main.h"
  (2) Places changed other than the above are made conditional with #ifdef R_FLASH_USB
- Changes to dbsct_pcdc.c
  The sections indicated by the command "// Flash table"
- Changes to resetprg.c
  The place where INTB is set in the function PowerON_Reset_PC() has been changed.
  Before change: set_intb(0xFFF80000);
  After change: set_intb(_sectop("INTERRUPT_VECTOR"));

8.12.2 Added Files

See section 5.8, File Structure, for the files added to the USB peripheral communication device class driver.

8.12.3 Added Sections

Table 8.1 lists the added sections.

Table 8.1 Added Sections

Section	Overview
R_flash_api_sec	Section used for variables in the flash programming code that runs in RAM.
RPFRAM	Section used for the flash programming code that runs in RAM.
TRGT_DMMY_FIXEDVECT	Section for the fixed vector in the target program.

8.12.4 Include File Directory

The directory WorkSpace\FLASH was added as an include file directory.

8.12.5 Linker Settings

Linker settings to map from ROM to RAM were added.

- ROM PFRAM is mapped to RPFRAM.
- ROM D_flash_api_sec is mapped to R_flash_api_sec.

9. Sample Code

The sample code can be downloaded from the Renesas Electronics Corporation web site.

10. Reference Documents

- RX62N Group, RX621 Group User's Manual: Hardware, Revision 1.10
  (The latest version can be downloaded from the Renesas Electronics Web site.)
- Technical Updates and Technical News
  (The latest information can be accessed at the Renesas Electronics Web site.)
- RX Family C Compiler Package, Version. 1.01R00
  (The latest version can be downloaded from the Renesas Electronics Web site.)
- C Compiler User's Manual, Revision 1.01R00
  (The latest version can be downloaded from the Renesas Electronics Web site.)
- Application Note
  RX600 Series: Simple Flash API for RX600 Rev.2.20 (R01AN0544EU)
  (The latest version can be downloaded from the Renesas Electronics Web site.)

  Renesas USB Device USB Basic Firmware Rev.1.10 (R01AN0512EJ)
  (The latest version can be downloaded from the Renesas Electronics Web site.)

  Renesas USB Device USB Peripheral Communication Device Class Driver Rev.1.10 (R01AN0273EJ)
  (The latest version can be downloaded from the Renesas Electronics Web site.)

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Mar.27.12	—	First edition issued
1.01	Mar.07.13	—	Title updated

General Precautions in the Handling of MPU/MCU Products

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

1. Handling of Unused Pins

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

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

2. Processing at Power-on

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

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

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

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

4. Clock Signals

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

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

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different 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.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

RENESAS

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

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

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, 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: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

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: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, 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.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 2.2