RX63T Group

USB Host Flash Boot Loader

R01AN1418EJ0100
Rev.1.00
May 14, 2014

Abstract

This application note describes a USB host flash boot loader that operates, in single-chip mode, the RX63T Group microcontroller USB 2.0 host/function module in host mode and rewrites the microcontroller's on-chip flash memory over a USB connection.

Note that this application note uses the sample code and drivers described in the following application notes. Renesas USB Device USB Host Mass Storage Class Driver, Renesas USB Device USB Basic Firmware, and M3S-TFAT-Tiny: Fat File System Software are included in the package.

- Erasing and writing internal flash memory
  RX600 and RX200 Series Simple Flash API for RX Rev.2.40 (R01AN0544EU0240)
- USB communication
  Renesas USB Device USB Basic Firmware Rev.2.00 (R01AN0512EJ0200)
  Renesas USB Device USB Host Mass Storage Class Driver Rev.2.00 (R01AN0513EJ0200)
- FAT file system
  M3S-TFAT-Tiny: Fat File System Software Rev.1.00 (R20AN0038EJ0100)

This application note has the following features.

- A Motorola S format program stored on a USB memory device can be written to flash memory.
  When the connection of a USB mass storage device (USB memory) that holds a Motorola S format program is recognized, the microcontroller's internal flash memory is erased and that program is written to the erased flash memory.
- The written program can be run.
  The Motorola S format program written to the microcontroller's internal flash memory can be executed on the microcontroller.
- USB specifications
  USB 2.0 standard full speed transfers are supported.
  USB mass storage class bulk-only transport (BOT) is supported.
  USB mass storage subclass SFF-8070i (ATAPI) and SCSI are supported.

Products

RX63T Group, 144-pin and 120-pin versions

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

This application note's sample code operates on the RX63T-H RSK board.

If a reset is cleared with switch SW3 on the RX63T-H RSK not held down, the Motorola S format program (filename: download.mot) in the connected USB memory will be written to the microcontroller's internal flash memory. After this write has completed, if a reset is cleared with switch SW3 in the pressed state, the program written to the microcontroller's internal flash memory (also referred to as the downloaded code) will be executed.

Note that the area that this sample code can overwrite is limited to part of the user MAT and the area used by the sample code itself is not overwritten. See section 5.1, Operation Overview, for details.

The result of writing the program to internal flash memory is displayed in the LCD and LEDs on the RX63T-H RSK. See section 5.5, Sample Code LCD and LED Display, for details on the content displayed.

Table 1.1 lists the peripheral function used and their uses, and figure 1.1 shows an example of using this application note.

Table 1.1 Peripheral Functions and their Uses

Peripheral Function	Use
ROM (Flash memory used for storing program code)	The internal flash memory is programmed using ROM P/E mode.
USB 2.0 host/function module	Communication with the USB memory device

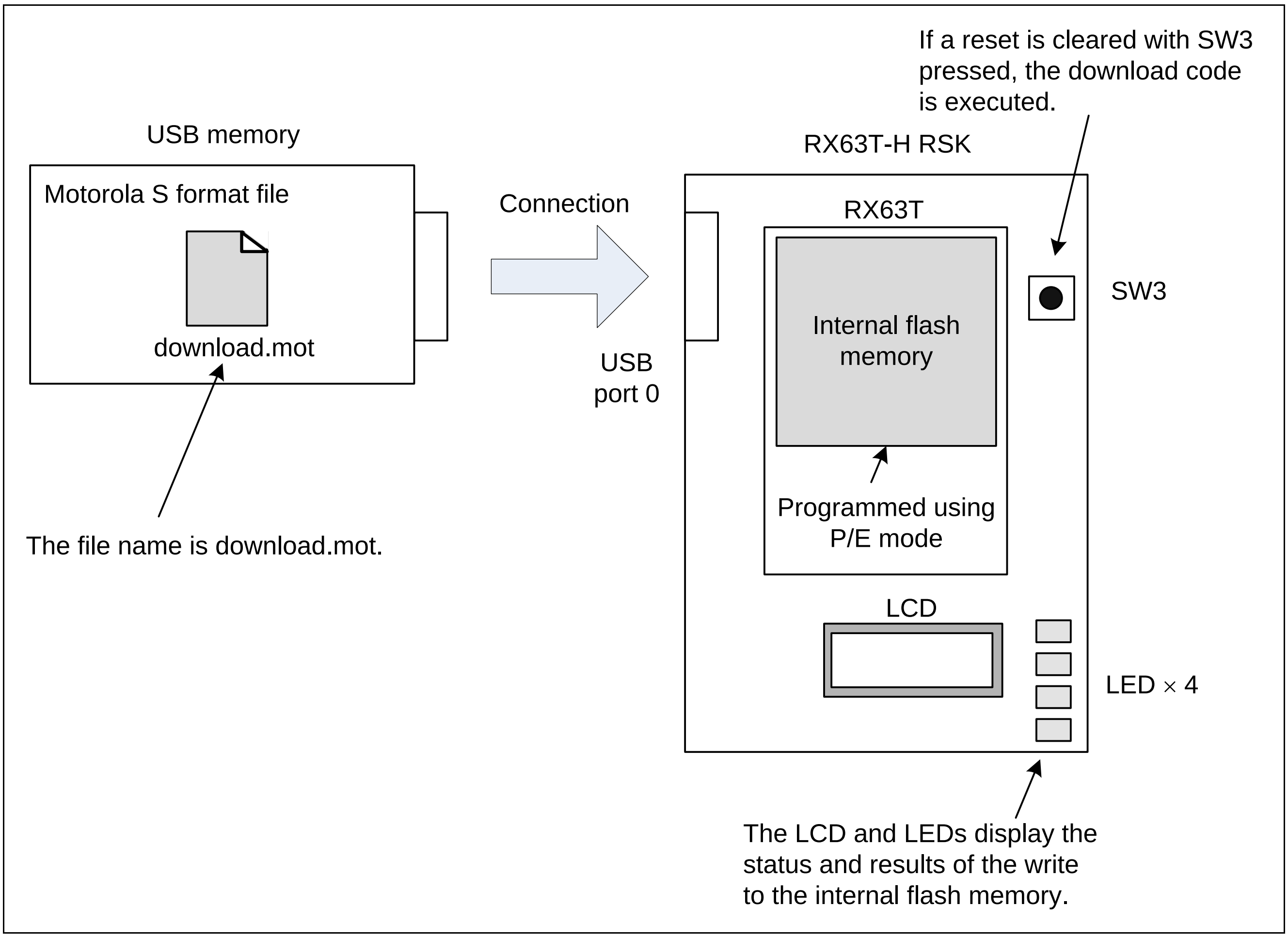

If a reset is cleared with SW3 pressed, the download code is executed.
USB memory
RX63T-H RSK
Motorola S format file
Connection
RX63T
SW3
Internal flash memory
download.mot
USB port 0
Programmed using P/E mode
The file name is download.mot.
LCD
LED × 4
The LCD and LEDs display the status and results of the write to the internal flash memory.

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	R5F563TEADFB (RX63T Group)
Operating frequency	Main clock: 12 MHz PLL: 192 MHz (main clock frequency divided by 1 and multiplied by 16) System clock (ICLK): 96 MHz (PLL clock frequency divided by 2) Peripheral module clock B (PCLKB): 48 MHz (PLL clock frequency divided by 4) USB clock supplied to USB (UCLK): 48 MHz (PLL clock frequency divided by 4) FlashIF clock (FCLK): 48 MHz (PLL clock frequency divided by 4)
Operating voltage	5.0 V
Integrated development environment	Renesas electronics High-performance Embedded Workshop Version 4.09.01.007
C compiler	Renesas electronics RX Standard Toolchain Version 1.2.1.0
	-cpu=rx600 -include="$(WORKSPDIR)\WorkSpace\ANSI" -include="$(WORKSPDIR)\WorkSpace\SmplMain\APL" -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\MSCFW\include" -include="$(WORKSPDIR)\WorkSpace\MSCFW\TFAT\lib_src" -include="$(WORKSPDIR)\WorkSpace\USBSTDFW\include" -include="$(WORKSPDIR)\WorkSpace\FLASH" -include="$(WORKSPDIR)\WorkSpace\FLASH\src" -include="$(WORKSPDIR)\WorkSpace\FLASH\r_bsp" -include="$(WORKSPDIR)\WorkSpace\FLASH\r_bsp\mcu\rx63t" -include="$(WORKSPDIR)\WorkSpace\FLASH\r_bsp\board\rskrx63t_144pin" -define=USB_FW_PP=USB_FW_NONOS_PP,USB_TFAT_USE_PP=1,R_FLASH_USB -output=obj="$(CONFIGDIR)\$(FILELEAF).obj" -debug -nostuff -listfile="$(CONFIGDIR)\$(FILELEAF).lst" -optimize=0 -nologo
iodefine.h version	0.50
Endian order	Little-endian
Operating mode	Single-chip mode
Sample code version	Version 1.00
Board used	Renesas Starter Kit for RX63T-H

3. Reference Application Notes

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

- Renesas USB Device USB Basic Firmware Rev.2.00 (R01AN0512EJ)
- Renesas USB Device USB Host Mass Storage Class Driver Rev.2.00 (R01AN0513EJ)
- M3S-TFAT-Tiny: FAT File System Software Rev.1.00 (R20AN0038EJ)
- RX600 & RX200 Series Simple Flash API for RX Rev.2.40 (R01AN0544EU)

4. Hardware

4.1 Used Pins

Table 4.1 lists the pins and their functions.

Table 4.1 Used Pins and Their Functions

Pin Name	I/O	Function
USB0_DP	I/O	D+ I/O pin of the port 0 USB on-chip transceiver This pin should be connected to the D+ pin of the USB bus.
USB0_DM	I/O	D– I/O pin of the port 0 USB on-chip transceiver This pin should be connected to the D– pin of the USB bus.
P13/USB0_VBUSEN	Output	VBUS (5 V) supply enable signal for port 0 external power supply chip
PE1/USB0_OVRCURA	Input	Port 0 external overcurrent detection signals should be connected to these pins. VBUS comparator signals should be connected to these pins when the OTG power supply chip is connected.
PE3	Input	Sample code mode selection pin
P71	Output	LED 0 connection pin
P72	Output	LED 1 connection pin
P73	Output	LED 2 connection pin
P33	Output	LED 3 connection pin
PG4	Output	LCD module control pin
PG5	Output	LCD module control pin
PG0	Output	LCD module control pin
PG1	Output	LCD module control pin
PG2	Output	LCD module control pin
PG3	Output	LCD module control pin

5. Software

5.1 Operation Overview

5.1.1 Operation After a Reset Is Cleared

After a reset is cleared, the sample code checks the state of switch SW3 (pin PE3). If this switch is not being pressed (if pin PE3 is high), it runs the USB host flash boot loader and programs the internal flash memory over the USB bus. If this switch is being pressed (if pin PE3 is low), it runs the download code.

Figure 5.1 shows the operation after a reset.

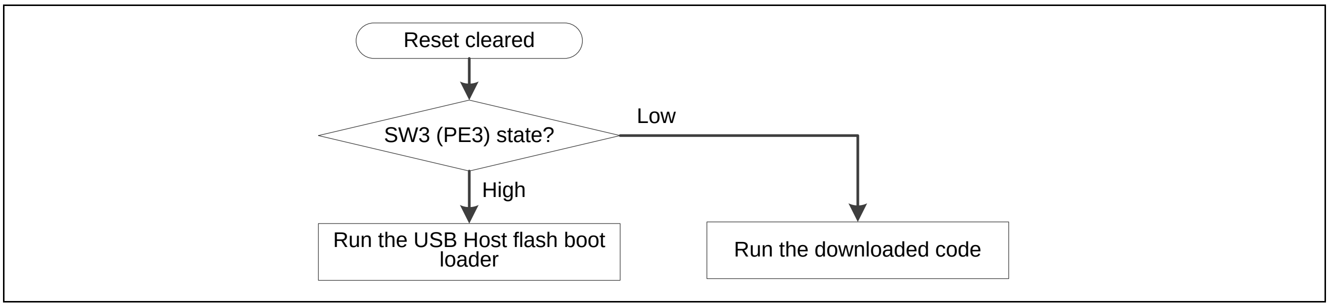
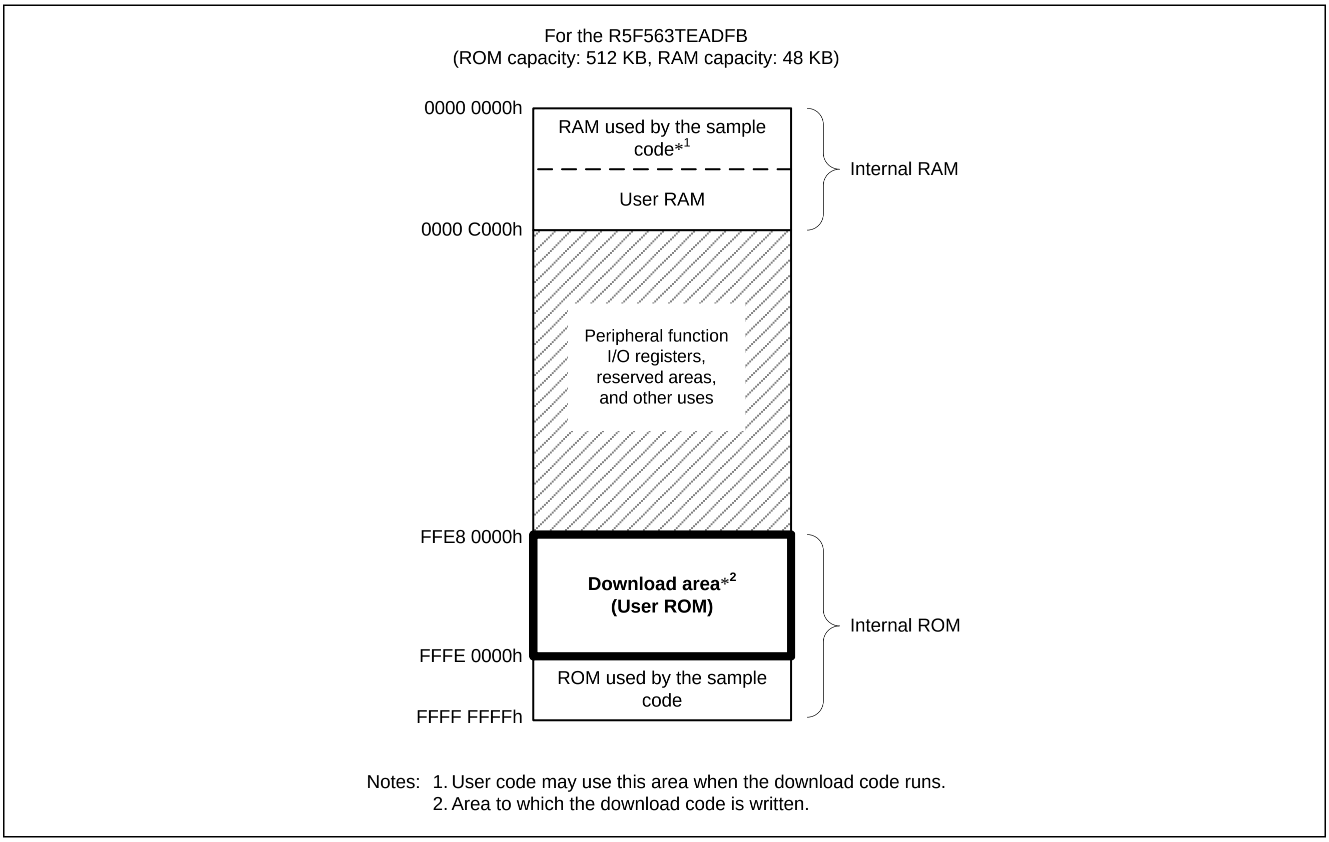


Figure 5.1 Operation After a Reset Is Cleared

5.1.2 Object of Overwriting

The object area that the USB host flash boot loader overwrites is restricted to a certain part of the user MAT (referred to as the download area in this document). The area used for the sample code itself, FFFE 0000h to FFFF FFFFh, is not overwritten.

Figure 5.2 shows the memory allocation.


For the R5F563TEADFB
(ROM capacity: 512 KB, RAM capacity: 48 KB)
0000 0000h
RAM used by the sample code*1
User RAM
Internal RAM
0000 C000h
Peripheral function I/O registers, reserved areas, and other uses
FFE8 0000h
Download area*2 (User ROM)
Internal ROM
FFFE 0000h
ROM used by the sample code
FFFF FFFFh

Notes: 1. User code may use this area when the download code runs.
2. Area to which the download code is written.

Figure 5.2 Memory Allocation

5.1.3 Programming the Download Area

The USB host flash boot loader uses the following procedure to program the download area. Figure 5.3 shows the download area programming procedure.

(1) Monitor the USB device connection.
(2) If a USB connection is detected, acquire the information for the connected device and determine whether or not access is possible.
(3) If access to the connected USB device (USB memory) is possible, search for a Motorola S format file with the filename download.mot.
(4) If a Motorola S format file with the filename download.mot is recognized, erase the download area.
(5) After erasing the download area, read 2048 bytes of data from the Motorola S format file in the USB memory and store it in internal RAM.
(6) After storing the data in RAM, analyze the data and write it to the download area in 128 byte units.
(7) Repeat the processing of steps (5) and (6) until all the data has been written.
Note that the end of the Motorola S format file is recognized by the occurrence of an end record (an S7, S8, or S9 record).
(8) If the erase and programming of the download area completes normally, report that normal completion in the LCD and LEDs connected to the I/O ports.

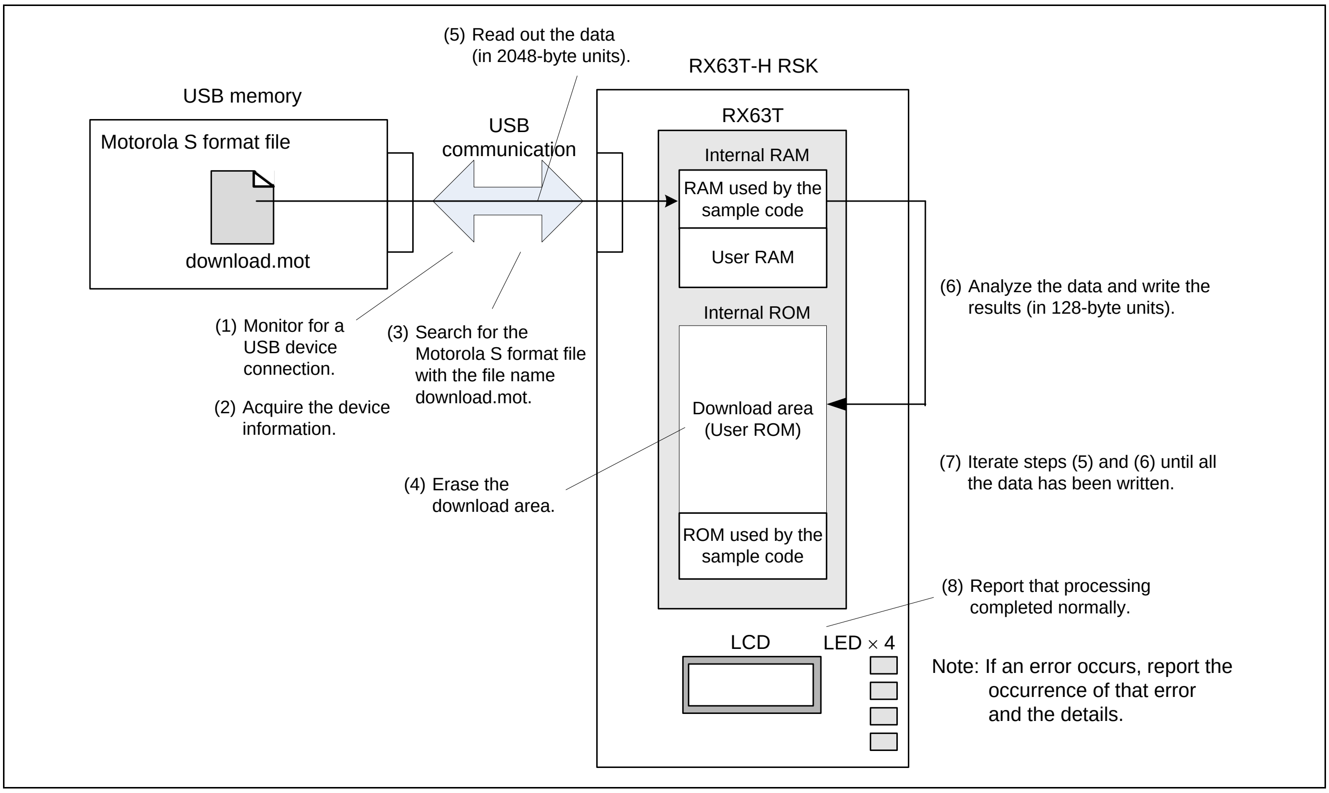

(5) Read out the data (in 2048-byte units).
RX63T-H RSK
USB memory
Motorola S format file
download.mot
USB communication
RX63T
Internal RAM
RAM used by the sample code
User RAM
Internal ROM
Download area (User ROM)
ROM used by the sample code
(1) Monitor for a USB device connection.
(2) Acquire the device information.
(3) Search for the Motorola S format file with the file name download.mot.
(4) Erase the download area.
(6) Analyze the data and write the results (in 128-byte units).
(7) Iterate steps (5) and (6) until all the data has been written.
(8) Report that processing completed normally.
LCD
LED × 4
Note: If an error occurs, report the occurrence of that error and the details.

Figure 5.3 Programming the Download Area

Note: If an error occurs during sample code execution, the details of that error are reported in the LCD and LEDs. See section 5.5, Sample Code LCD and LED Display, for details on error occurrence conditions and the LCD and LED display.

5.2 Download Code Execution Start Position

If the switch SW3 state is the low level when a microcontroller reset is cleared, the sample code will run the download code. At this time, the starts executing the download code from the address stored at location FFFD FFFCh. That is, the reset vector for the download code is FFFD FFFCh. Therefore the download code must store its start address at location FFFD FFFCh.

Figure 5.4 shows the reset vector for the download code.

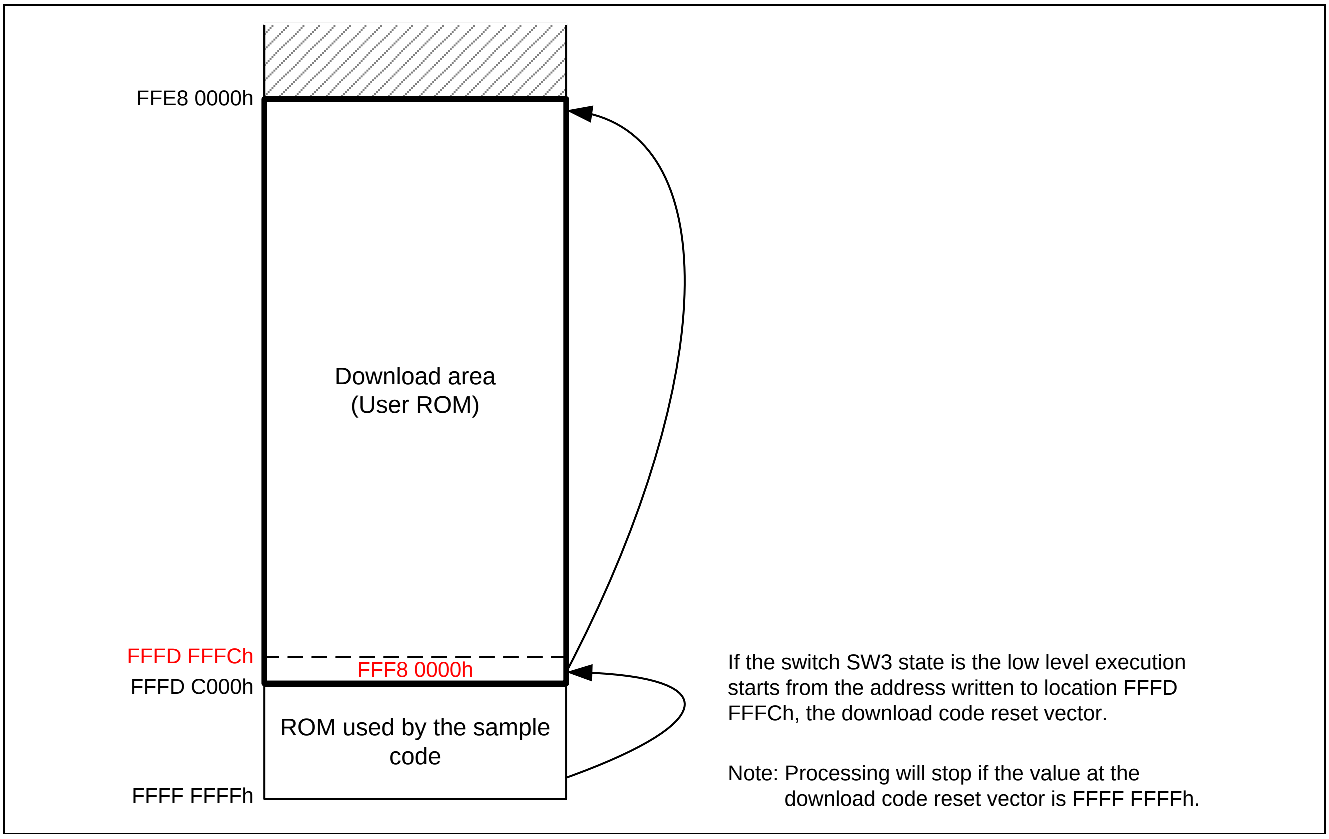

FFE8 0000h
Download area
(User ROM)
FFFD FFFCh
FFF8 0000h
FFFD C000h
ROM used by the sample code
FFFF FFFFh
If the switch SW3 state is the low level execution starts from the address written to location FFFD FFFCh, the download code reset vector.
Note: Processing will stop if the value at the download code reset vector is FFFF FFFFh.

Figure 5.4 Download Code Reset Vector

Note: If nothing is written to the download code reset vector (that is, if the value at the download code reset vector is FFFF FFFFh), the sample code executes a while(1) infinite loop to stop processing.

5.3 Software Structure of the Sample Code

The sample code uses the Renesas USB Device USB Basic Firmware and the Renesas USB Device USB Host Mass Storage Class Driver for USB communication.

It uses the M3S-TFAT-Tiny: Fat File System Software as its FAT file system.

It also uses the RX600 & RX200 Series Simple Flash API for RX for erase and write processing for the internal flash memory.

Figure 5.5 shows software structure of the sample code and table 5.1 gives an overview of the software.

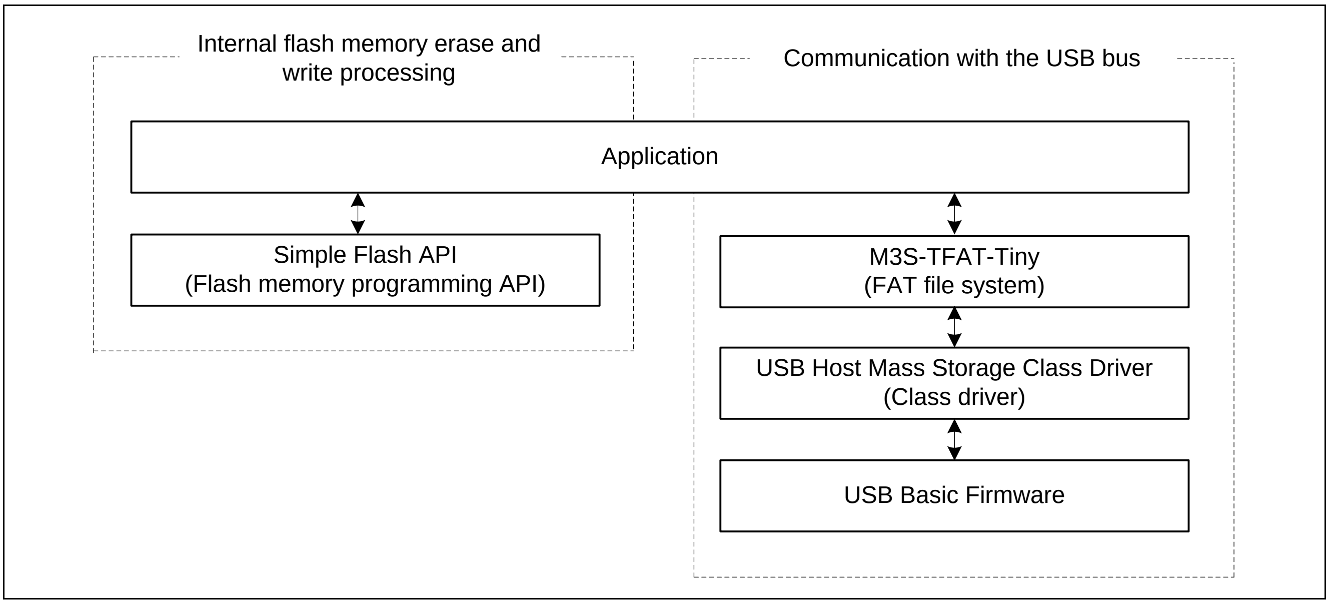

Internal flash memory erase and write processing
Communication with the USB bus
Application
Simple Flash API
(Flash memory programming API)
M3S-TFAT-Tiny
(FAT file system)
USB Host Mass Storage Class Driver
(Class driver)
USB Basic Firmware

Figure 5.5 Software Structure of the Sample Code

Table 5.1 Software Overview

Module	Overview
Application	The application uses FAT library functions to read a Motorola S format program from the USB memory. It then uses the Simple Flash API functions to erase the internal flash memory and write that program to internal flash memory.
Simple Flash API for RX	API used to erase and write the internal flash memory
M3S-TFAT-Tiny	A FAT file system that supports FAT12 and FAT16.
USB Host Mass Storage Class Driver	A class driver that supports the USB mass storage class bulk-only transport (BOT) protocol.
USB Basic Firmware	A sample program that controls the USB interface.

5.4 Data Flow During Write

Figure 5.6 shows the data flow internal to the microcontroller when the download code is written to flash memory.

(1) The data acquired from the USB driver is transferred to a receive ring buffer.

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

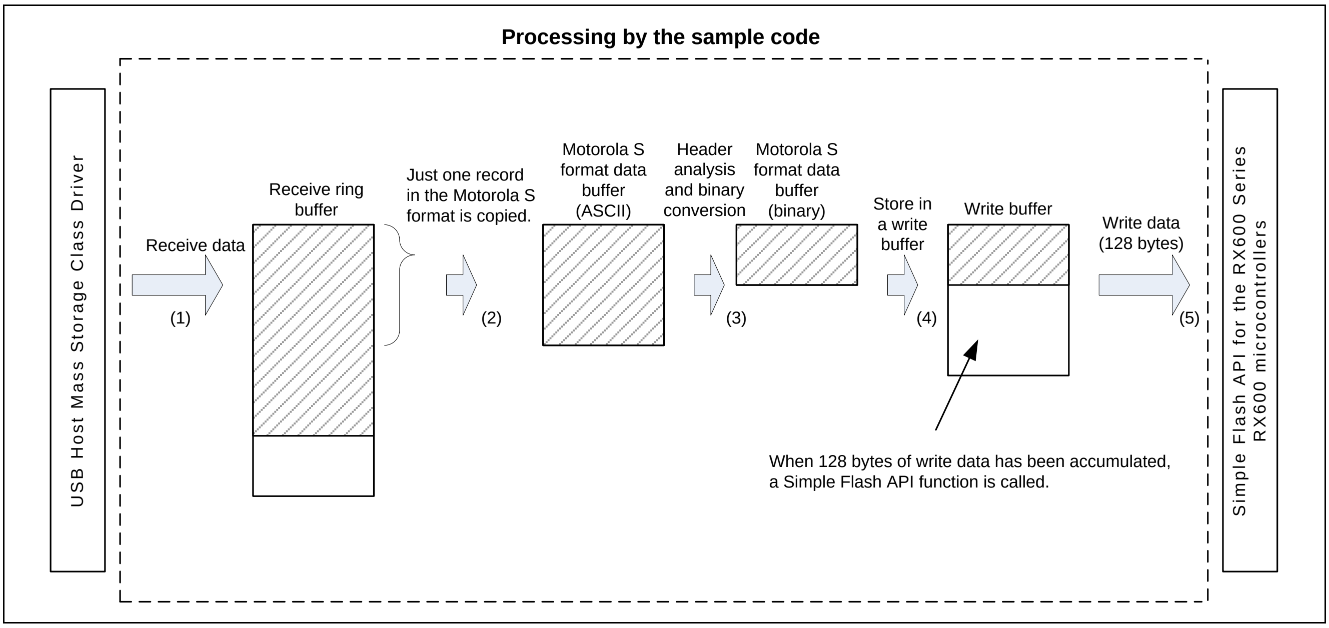
(3) At the same time as analyzing the Motorola S format data header section, the ASCII data is converted to binary and stored in a Motorola S format buffer (for binary data).

See section 7, Motorola S format, for the Motorola S format data analysis specifications used in this application note.

(4) The data is stored in a write buffer.

In the RX63T Group microcontroller, data is written to the user MAT in units of 128 bytes. Therefore, the sample code iterates steps (2) to (4) above until a total of 128 bytes of write data has been stored in the write buffer. Also, if the total amount of write data exceeds 128 bytes, the excess data is stored temporarily and used for the next write of 128 bytes of data.

(5) The assembled 128 bytes of data are written to flash memory using the Simple Flash API. After the write, if the size of the data stored in the temporary storage buffer is greater than the write unit, step (4) is repeated and write operation continues.



Processing by the sample code
USB Host Mass Storage Class Driver
Receive data
(1)
Receive ring buffer
Just one record in the Motorola S format is copied.
(2)
Motorola S format data buffer (ASCII)
Header analysis and binary conversion
(3)
Motorola S format data buffer (binary)
Store in a write buffer
(4)
Write buffer
Write data (128 bytes)
(5)
When 128 bytes of write data has been accumulated, a Simple Flash API function is called.
Simple Flash API for the RX600 Series RX600 microcontrollers

Figure 5.6 Data Flow During Write

Figure 5.7 shows the data structures used when writing data to flash memory.

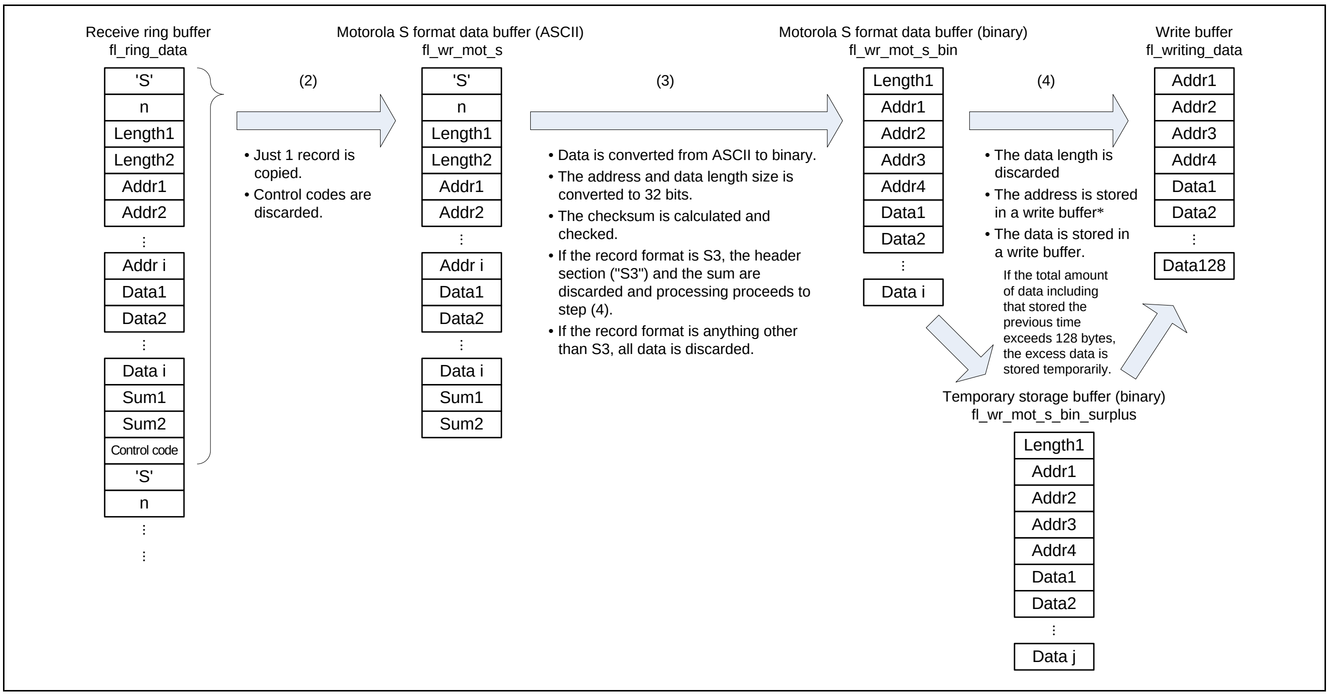

Receive ring buffer
fl_ring_data
'S'
n
Length1
Length2
Addr1
Addr2
Addr i
Data1
Data2
Data i
Sum1
Sum2
Control code
'S'
n
(2)
• Just 1 record is copied.
• Control codes are discarded.
Motorola S format data buffer (ASCII)
fl_wr_mot_s
'S'
n
Length1
Length2
Addr1
Addr2
Addr i
Data1
Data2
Data i
Sum1
Sum2
(3)
• Data is converted from ASCII to binary.
• The address and data length size is converted to 32 bits.
• The checksum is calculated and checked.
• If the record format is S3, the header section ("S3") and the sum are discarded and processing proceeds to step (4).
• If the record format is anything other than S3, all data is discarded.
Motorola S format data buffer (binary)
fl_wr_mot_s_bin
Length1
Addr1
Addr2
Addr3
Addr4
Data1
Data2
Data i
(4)
• The data length is discarded
• The address is stored in a write buffer*
• The data is stored in a write buffer.
If the total amount of data including that stored the previous time exceeds 128 bytes, the excess data is stored temporarily.
Write buffer
fl_writing_data
Addr1
Addr2
Addr3
Addr4
Data1
Data2
Data128
Temporary storage buffer (binary)
fl_wr_mot_s_bin_surplus
Length1
Addr1
Addr2
Addr3
Addr4
Data1
Data2
Data j

Figure 5.7 Data Structures Used for Writing

Note: In the RX63T Group microcontroller internal flash memory, a start address used for a write operation must be aligned on a 128-byte boundary. Accordingly, the sample code performs processing to assure that write start addresses are aligned on 128-byte boundaries when storing addresses to write buffers. See the flowchart in section 5.13.11, Download Area Write Data Creation, for details on this processing.

5.5 Sample Code LCD and LED Display

The sample code displays the state of progress of the program and the results in the LCD and LEDs mounted on the RX63T-H RSK board. Table 5.2 lists the LED patterns displayed by the sample code.

Table 5.2 Sample Code LED Display

○: On, ×: Off

LED Display State					
LED3	**LED2**	**LED1**	**LED0**	**Order**	**Description**
×	×	×	×	↓	The LEDs display a binary counter during the download processing. The LED display is updated every 128 bytes (the unit of writes to the RX63T Group user MAT) × 128 (16 KB).
×	×	×	○		
×	×	○	×		
×	×	○	○		
×	○	×	×		
×	○	×	○		
×	○	○	×		
×	○	○	○		
○	×	×	×		
○	×	×	○		
○	×	○	×		
○	×	○	○		
○	○	×	×		
○	○	×	○		
○	○	○	×		
○	○	○	○		
×	×	×	○	↓	When the sample code completes normally, a shifting display pattern is displayed in the LEDs. The LED display is updated once every 500 ms.
×	×	○	×		
×	○	×	×		
○	×	×	×		
×	×	×	×	↓	If the sample code terminates abnormally, the LEDs blink. The LED display is updated once every 500 ms.
○	○	○	○		

Table 5.3 lists the LCD patterns displayed by the sample code.

Table 5.3 Sample Code LCD Display

LCD Display State	Description
FLASH BOOT	Displayed when the USB host flash boot loader is run after a reset is cleared.
DETACH	Indicates that a USB was disconnected after having been connected.*1
ATTACH	Indicates that there was a USB connected after a reset was cleared. *1
FLASH UPDATE..	Indicates that download processing is in progress.
ERROR!! D OPEN	Indicates that no accessible drive was detected. (Drive open error)
ERROR!! D MOUNT	Indicates that drive mount processing failed. (Drive mount error)
ERROR!! F OPEN	Indicates that file open processing failed. (File open error)
ERROR!! F READ	Indicates that file read processing failed. (File read error)
ERROR!! F CLOSE	Indicates that file close processing failed. (File close error)
ERROR!! SUM	See section 7, Motorola S format. (Checksum error)
ERROR!! MOTS	See section 7, Motorola S format. (Format error)
ERROR!! ERASE	Indicates that erase of the download area failed. (Erase error)
ERROR!! WRITE	Indicates that write of the download area failed. (Write error)
ERROR!! ADDRESS	See section 7, Motorola S format. (Address error)
ERROR!! VERIFY	Indicates that the result of verifying the data written to the download area was that an abnormality was detected. (Verify error)
ERROR!! F END	Indicates that no Motorola S format end record had been received even though an end of file was detected by the FAT library. (File end error)
ERROR!! ILL DET	Indicates that a USB detach was detected during either drive open or download processing. (Illegal detach error)
ERROR!! ENDIAN	Indicates that the endian order (MDES value) of the sample code and the download code do not match. (Endian error)

Note: 1. The DETACH display used when a USB is disconnected, and the ATTACH display used when a USB is connected, follow the specifications of the USB Host Mass Storage Class Driver.

5.6 Required Memory Size

Table 5.4 lists the required memory sizes.

Table 5.4 Required Memory Size

Memory Used	Size	Remarks
ROM	121,022 bytes	Since the sample code is allocated to locations FFFE 0000h to FFFF FFFFh, the amount of ROM that can be written is the total ROM capacity minus 131,072 bytes.
RAM	38,002 bytes	The user code can use this area when it runs.
Maximum user stack usage	588 bytes	
Maximum interrupt stack usage	152 bytes	

Note: The required memory size varies depending on the C compiler version and compile options.

5.7 File Composition

Table 5.5 lists the files used in the sample code. Files not generated by the integrated development environment should not be listed in this table.

Table 5.5 Files Used in the Sample Code

File Name	Description	Remarks
r_flash_api_rx.c	The RX600 & RX200 Series RX Simple Flash API program	For details, see the RX600 & R200 Series RX Simple Flash API application note.
locking.c	The RX600 & RX200 Series RX Simple Flash API program	
mcu_locks.c	The RX600 & RX200 Series RX Simple Flash API program	
r_flash_api_rx_if.h	External reference include header for the RX600 & RX200 Series RX Simple Flash API program (r_flash_api_rx.c).	
r_flash_api_rx_private.h	Parameter settings include header for the RX600 & RX200 Series RX Simple Flash API program.	
r_flash_api_rx_config.h	Parameter settings include header for the RX600 & RX200 Series RX Simple Flash API program.	
r_bsp.h	External reference include header for the RX600 & R2X00 Series RX Simple Flash API program.	
r_bsp_config.h	Parameter settings include header for the RX600 & RX200 Series RX Simple Flash API program.	
r_flash_api_rx63t.h	Parameter settings include header for the RX600 & RX200 Series RX Simple Flash API program.	
mcu_info.h	Parameter settings include header for the RX600 & RX200 Series RX Simple Flash API program.	
locking.h	External reference include header for the RX600 & RX200 Series RX Simple Flash API program (locking.c).	
mcu_locks.h	External reference include header for the RX600 & RX200 Series RX Simple Flash API program (mcu_locks.c).	
r_Flash_main.c	Flash programming data processing	
r_Flash_main.h	External reference include header for the flash programming data processing	
r_Flash_buff.c	USB receive ring buffer related processing	
r_Flash_buff.h	External reference include header for the USB receive ring buffer related processing	
TrgtPrgDmmy.c	Dummy program for allocating the download code area	
Other files	The USB Host Mass Storage Class Driver program	See the Renesas USB device USB Host Mass Storage Class Driver and USB Basic Firmware application notes for details.

5.8 Option-Setting Memory

Table 5.6 lists the option-setting memory configured in the sample code. When necessary, set a value suited to the user system.

Table 5.6 Option-Setting Memory Configured in the Sample Code

Symbol	Address	Setting Value	Contents
OFS0	FFFF FF8Fh to FFFF FF8Ch	FFFF FFFFh	IWDT stops after a reset. WDT stops after a reset.
OFS1	FFFF FF8Bh to FFFF FF88h	FFFF FFFFh	Voltage monitor reset 0 is ignored after a reset.
MDES*[1]	FFFF FF83h to FFFF FF80h		(Single-chip mode)
		FFFF FFFFh	Little endian
		FFFF FFF8h	Big endian

Note: 1. The setting in the sample code is little endian. Refer to 8.7, Endian Order, for information on changing the endian setting.

5.9 Constants

Table 5.7 and table 5.8 list the constants used in the sample code 1 and constants used in the sample code 2.

Table 5.7 Constants Used in the Sample Code 1

Constant	Set Value	Description
FL_MODE_ENTRY_WAIT_LCD_PERIOD	100000	Wait time at mode entry
FL_UPDATE_WAIT_LED_PERIOD	128	Time for the interval between LED display updates during download processing
FL_ERROR_WAIT_LED_PERIOD	500	Time for the interval between LED display updates during error handling
FL_DONE_WAIT_LED_PERIOD	500	Time for the interval between LED display updates during normal termination
FL_RINGBUFF_SIZE	4096	Size of the USB data reception ring buffer
FL_TARGET_REST_VECT_ADDR	FFFDFFFCh	Reset vector address of download code
FL_START_BLOCK_NUM	14	First block in the download area
FL_END_BLOCK_NUM	37	Last block in the download area
FL_START_WRITE_ADDRESS	FFF80000h	First address in the download area
FL_END_WRITE_ADDRESS	FFFDFFFFh	Last address in the download area
MDES_START_ADDRESS	FFFFFF80h	MDES start address
MDES_END_ADDRESS	FFFFFF83h	MDES end address
FL_UPDATE_FILE_NAME	"download.mot"	Download code file name
FL_MOTS_LNG_MAX_DATA	0xFF	Max. data length value of Motorola S format
FL_MOTS_LNG_SIZE	1	Data length buffer size of Motorola S format
FL_MOTS_ADDR_MIN_SIZE	2	Min. address buffer size of Motorola S format
FL_MOTS0_ADDR_SIZE	2	S0 Motorola S format data address buffer size
FL_MOTS3_ADDR_SIZE	4	S3 Motorola S format data address buffer size
FL_MOTS7_ADDR_SIZE	4	S7 Motorola S format data address buffer size
FL_MOTS8_ADDR_SIZE	3	S8 Motorola S format data address buffer size
FL_MOTS9_ADDR_SIZE	2	S9 Motorola S format data address buffer size
FL_MOTS_SUM_SIZE	1	Motorola S format data checksum buffer size

Table 5.8 Constants Used in the Sample Code 2

Constant	Set Value	Description
FL_USB_RCV_BLANK_SIZE	FL_RINGBUFF_SIZE / 2	Ring buffer capacity
FL_PORT_MDE	PORTE.PIDR.BIT.B3	PORT register for the port connected to SW3
FL_DDR_MDE	PORTE.PDR.BIT.B3	PDR register for the port connected to SW3
ROM_PROGRAM_SIZE	128*[1]	Sets the unit for writes to the user MAT on the target device. This value is contained in r_flash_api_rx63t.h, and it is referenced when this file is included.

Note: 1. Value for RX63N/RX63N Group as the target device.

5.10 Variables

Table 5.9 lists the static Variables (1), and Table 5.10 lists the static Variables (2).

Table 5.9 static Variables (1)

Type	Variable Name	Contents	Function Used
static FIL	fl_file	Variable specifying the file object structure to be created*	R_Fl_Flash_Update
static FATFS	fl_fatfs	Variable specifying the work area (file system object structure) to be registered*	R_Fl_Flash_Update
static uint8_t	fl_usb_read_data[FL_RINGBUFF_SIZE / 2]	Area for storing data read from USB memory	R_Fl_Flash_Update
static Fl_prg_mot_s_t	fl_wr_mot_s	Motorola S format storage buffer (ASCII)	R_Fl_Prg_ProcessForMotS_data, R_Fl_Prg_StoreMotS
static Fl_prg_mot_s_binary_t	fl_wr_mot_s_bin	Motorola S format storage buffer (binary)	R_Fl_Prg_ProcessForMotS_data, R_Fl_Prg_MakeWriteData
static Fl_prg_mot_s_binary_t	fl_wr_mot_s_bin_surplus	Temporary storage area for redundant (exceeding the write unit) data (binary)	R_Fl_Prg_ProcessForMotS_data, R_Fl_Prg_MakeWriteData, R_Fl_Prg_ClearMotSVariables
static Fl_prg_writing_data_t	fl_writing_data	Variable specifying the write area (address)	R_Fl_Prg_ProcessForMotS_data, R_Fl_Prg_MakeWriteData, R_Fl_Prg_WriteData, R_Fl_Prg_ClearMotSVariables
static uint8_t	fl_communication_start_flg	Motorola S format start code (S0) detection flag 0: Start code not detected 1: Start code detected	R_Fl_Prg_ProcessForMotS_data
static uint8_t	fl_communication_end_flg	Motorola S format end code (S7, S8, S9) detection flag 0: End code not detected 1: End code detected	R_Fl_PrgramTrgtArea, R_Fl_Prg_ProcessForMotS_data
static uint8_t	fl_tfat_eof_flg	File end detection flag 0: File end not detected 1: File end detected	R_Fl_Flash_Update, R_Fl_PrgramTrgtArea

Note: * For details, refer to the M3S-TFAT-Tiny: FAT File System Software.

Table 5.10 static Variables (2)

Type	Variable Name	Contents	Function Used
static uint8_t	fl_ring_data[FL_RINGBUFF_SIZE]	Receive ring buffer	R_Fl_RingEnQueue, R_Fl_RingDeQueue
static uint32_t	fl_queue_head	Variable specifying receive ring buffer read position	R_Fl_RingEnQueue, R_Fl_RingDeQueue
static uint32_t	fl_queue_num	Data count stored in receive ring buffer	R_Fl_RingEnQueue, R_Fl_RingDeQueue, R_Fl_RingCheck
static const uint8_t	fl_tbl_msg_drive_open_err[]	Variable indicating drive open error	R_Fl_Error
static const uint8_t	fl_tbl_msg_drive_mount_err[]	Variable indicating drive mount error	R_Fl_Error
static const uint8_t	fl_tbl_msg_file_open_err[]	Variable indicating file open error	R_Fl_Error
static const uint8_t	fl_tbl_msg_file_read_err[]	Variable indicating file read error	R_Fl_Error
static const uint8_t	fl_tbl_msg_file_close_err[]	Variable indicating file close error	R_Fl_Error
static const uint8_t	fl_tbl_msg_program_erase_err[]	Variable indicating erase error	R_Fl_Error
static const uint8_t	fl_tbl_msg_program_write_err[]	Variable indicating write error	R_Fl_Error
static const uint8_t	fl_tbl_msg_s_format_sum_err[]	Variable indicating checksum error	R_Fl_Error
static const uint8_t	fl_tbl_msg_s_format_mots_err[]	Variable indicating format error	R_Fl_Error
static const uint8_t	fl_tbl_msg_program_verify_err[]	Variable indicating verify error	R_Fl_Error
static const uint8_t	fl_tbl_msg_program_address_err[]	Variable indicating address error	R_Fl_Error
static const uint8_t	fl_tbl_msg_illegal_detach_err[]	Variable indicating illegal detach error	R_Fl_Error
static const uint8_t	fl_tbl_msg_tfat_end_err[]	Variable indicating file end error	R_Fl_Error
static const uint8_t	fl_tbl_msg_endian_err[]	Variable indicating endian error	R_Fl_Error
static const uint8_t	*p_fl_output_msg[]	Pointer indicating error message	R_Fl_Error

5.11 Structures and Unions

Figure 5.8 shows the structures and unions used in the sample code.

```
/* buffer for mot S format data */
typedef struct {
    uint8_t type[2];                  /* "S0", "S1" and so on */
    uint8_t len[2];                   /* "00"-"FF" */
    uint8_t addr_data_sum[(FL_MOTS_LNG_MAX_DATA-FL_MOTS_LNG_SIZE)*2];
} 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[(FL_MOTS_LNG_MAX_DATA-FL_MOTS_LNG_SIZE-FL_MOTS_ADDR_MIN_SIZE];
} Fl_prg_mot_s_binary_t;

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

Figure 5.8 Structures and Unions Used in the Sample Code

5.12 Functions

Table 5.11 lists the functions. Note, however, that the USB Mass Storage Class Driver, Simple Flash API, and FAT file system software functions are not shown here.

Table 5.11 Functions

Function	Description
R_Fl_Mode_Entry	Mode selection
R_Fl_Flash_Update	Main function for flash memory write processing
R_Fl_EraseTrgtArea	Erase processing
R_Fl_Ers_EraseFlash	Erase download area
R_Fl_PrgramTrgtArea	Write download area
R_Fl_Prg_PrgramTrgtArea	Write processing
R_Fl_Prg_StoreMotS	Store Motorola S format data
R_Fl_Prg_ProcessForMotS_data	Header analysis, binary conversion, and write of a Motorola S format record
R_Fl_Prg_MotS_AsciiToBinary	Convert Motorola S format data from ASCII to binary
R_Fl_Prg_MakeWriteData	Create write data for the download area
R_Fl_Prg_WriteData	Write to download area
R_Fl_Prg_ClearMotSVariables	Clear the variables related to the Motorola S format data
R_Fl_Run_StopUSB	Stop USB
R_Fl_RcvDataString	Store USB receive data
R_Fl_Error	Error handling
R_Fl_RingCheckBlank	Verify amount of free space in receive ring buffer
R_Fl_RingEnQueue	Store data in receive ring buffer
R_Fl_RingDeQueue	Read data from receive ring buffer
R_Fl_RingCheck	Verify data count in receive ring buffer
R_Fl_AsciiToHexByte	Convert data from ASCII to binary

5.13 Function Specifications

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

R_Fl_Mode_Entry	
Outline	Mode entry
Header	r_Flash_main.h
Declaration	void R_Fl_Mode_Entry(void)
Description	• Checks the state of SW3. • If SW3 is not depressed, runs the USB host flash boot loader. • If SW3 is depressed, runs the download code.
Arguments	None
Return Value	None

R_Fl_Flash_Update	
Outline	Flash memory write processing main routine
Header	r_Flash_main.h
Declaration	void R_Fl_Flash_Update(void)
Description	• Calls the function that performs the download area erase processing. • Calls FAT library functions to read data from the Motorola S format file in the USB memory. • Calls functions that analyze the Motorola S format data and write the data to the download area. • Calls an error handler if execution of a FAT library function fails.
Arguments	None
Return Value	None

R_Fl_EraseTrgtArea	
Outline	Erase processing
Header	None
Declaration	static void R_Fl_EraseTrgtArea(void)
Description	• Calls the function that erases the download area. • Calls an error handler if erase of the download area fails.
Arguments	None
Return Value	None

R_Fl_Ers_EraseFlash	
Outline	Erase download area
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_Ers_EraseFlash(void)
Description	Erases the download area.
Arguments	None
Return Value	• If the erase operation completes normally: FLASH_API_SAMPLE_OK • If the erase operation does not complete normally: FLASH_API_SAMPLE_NG
Notes	The processor status word (PSW) interrupt priority level (IPL) is modified to prevent ROM access by interrupts during the erase operation.

R_Fl_PrgramTrgtArea

Outline Write download area

Header None

Declaration static void R_Fl_PrgramTrgtArea(void)

Description

- Calls the function that performs the required write processing.
- Calls an error handler if the end of file is reached before receiving a Motorola S format end record.

Arguments None

Return Value None

R_Fl_Prg_PrgramTrgtArea

Outline Write processing

Header None

Declaration static Fl_API_SMPL_rtn_t R_Fl_Prg_PrgramFlash(void)

Description

- If there is data in the receive ring buffer, calls the function that analyzes a single Motorola S format record.
- When a single Motorola S format record has been analyzed, calls the function that performs header analysis, conversion to binary, and writing to the download area.

Arguments None

Return Value

- If writing to the download area terminates normally: FLASH_API_SAMPLE_OK
- If writing to the download area did not terminate: FLASH_API_SAMPLE_NG

R_Fl_Prg_StoreMotS

Outline Store Motorola S format data

Header None

Declaration static Fl_API_SMPL_rtn_t R_Fl_Prg_StoreMotS(uint8_t)

Description

- Stores the data passed in the argument as Motorola S format data one byte at a time.
- Discards all data until the first 'S' (ASCII data) is acquired.

Arguments First argument: mot_data : Motorola S format data

Return Value

- If a single Motorola S format data item (from the 'S' to the checksum) was stored: FLASH_API_SAMPLE_OK
- If a single Motorola S format data item was not stored: FLASH_API_SAMPLE_NG

Notes

- This function is used by passing Motorola S format data 1 byte at a time in the argument.
- The checksum is not checked.

R_Fl_Prg_ProcessForMotS_data

Outline Header analysis, binary conversion, and write of a Motorola S format record

Header None

Declaration static void R_Fl_Prg_ProcessForMotS_data(void)

Description

- Analyses the Motorola S format header and calls the function that converts to binary.
- Calls the function that stores data in a write buffer.
- Calls the function that writes data to the download area.
- Calls an error handler if data that differs from the Motorola Motorola S format is received.

Arguments None

Return Value None

R_Fl_Prg_MotS_AsciiToBinary

Outline Convert Motorola S format data from ASCII to binary

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 data in ASCII code to binary data.
- Verifies the checksum of the converted binary data.
- Calls an error handler if data that differs from the Motorola Motorola S format is received.
- Calls an error handler if a checksum error occurs.

Arguments

First argument: *tmp_mot_s : Pointer to Motorola S format data in ASCII

Second argument: *tmp_mot_s_binary : Pointer to variable that holds the converted to binary data

Return Value

- If conversion completed normally: FLASH_API_SAMPLE_OK
- If conversion does not complete normally: FLASH_API_SAMPLE_NG

R_Fl_Prg_MakeWriteData

Outline Create write data for the download area

Header None

Declaration static Fl_API_SMPL_rtn_t R_Fl_Prg_MakeWriteData(void)

Description Creates data divided every 128 bytes, the unit of writes to the RX63T Group user MAT.

Arguments None

Return Value

- If creation of 128 bytes (the unit of writes to the RX63T Group user MAT) of write data completed: FLASH_API_SAMPLE_OK
- If creation of 128 bytes (the unit of writes to the RX63T Group user MAT) of write data did not complete: FLASH_API_SAMPLE_NG

R_Fl_Prg_WriteData	
Outline	Write to download area
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_Prg_WriteData(void)
Description	• Performs the write to the download area. • When the endian order (MDES) of the download code is received, it is compared to the endian order (MDES) of the sample code, and if they do not match an error handler is called. • Verifies the data written. • Calls the error handler if the write failed. • Calls an error handler if a verify error occurs.
Arguments	None
Return Value	• 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) interrupt priority level (IPL) is modified to prevent ROM access by interrupts during the write operation.

R_Fl_Prg_ClearMotSVariables	
Outline	Clear the variables related to the Motorola S format data
Header	None
Declaration	static void R_Fl_Prg_ClearMotSVariables(void)
Description	Clears the variables related to the Motorola S format data.
Arguments	None
Return Value	None

R_Fl_Run_StopUSB	
Outline	Stop USB
Header	r_Flash_main.h
Declaration	void R_Fl_Run_StopUSB(void)
Description	Stops the USB.
Arguments	None
Return Value	None

R_Fl_RcvDataString	
Outline	Store USB receive data
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_RcvDataString(void *, uint16_t)
Description	Stores the data received over the USB in a receive ring buffer.
Arguments	First argument: *p_tranadr : Pointer to a buffer to hold data received over the USB Second argument: length : Length of the data received over the USB
Return Value	• If the store completes: FLASH_API_SAMPLE_OK • If the receive ring buffer was full: FLASH_API_SAMPLE_NG

R_Fl_Error	
Outline	Error handling
Header	r_Flash_main.h
Declaration	void R_Fl_Error(Fl_err_tbl_num_t err_num)
Description	• Calls the USB stop function. • Displays the error in the LCD and LEDs.
Arguments	First argument: err_num : Error number
Return Value	None

R_Fl_RingCheckBlank	
Outline	Verify amount of free space in receive ring buffer
Header	r_Flash_buff.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_RingCheckBlank(void)
Description	Verifies whether or not there is space in the receive ring buffer for one data unit (2048 bytes) read by the file read function.
Arguments	None
Return Value	• If there is adequate free space: FLASH_API_SAMPLE_OK • If there is not adequate free space: FLASH_API_SAMPLE_NG

R_Fl_RingEnQueue	
Outline	Store data in receive ring buffer
Header	r_Flash_buff.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_RingEnQueue(uint8_t)
Description	Stores data in the receive ring buffer.
Arguments	First argument: enq_data : Data to be stored
Return Value	• If the store completed: FLASH_API_SAMPLE_OK • If the buffer was full: FLASH_API_SAMPLE_NG

R_Fl_RingDeQueue	
Outline	Read data from 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	First argument: *p_deq_data : Pointer to buffer to store data read
Return Value	• If the data was read out normally: FLASH_API_SAMPLE_OK • If there was no data to read: FLASH_API_SAMPLE_NG

R_Fl_RingCheck	
Outline	Verify data count in receive ring buffer
Header	r_Flash_buff.h
Declaration	uint32_t R_Fl_RingCheck(void)
Description	Verifies the number of data items in the receive ring buffer.
Arguments	None
Return Value	Returns the number of receive data items.

R_Fl_AsciiToHexByte

Outline	Convert data from ASCII to binary	
Header	r_Flash_buff.h	
Declaration	uint8_t R_Fl_AsciiToHexByte(uint8_t, uint8_t)	
Description	Converts a 2-byte ASCII coded data item to 1 byte of binary data.	
Arguments	First argument: in_upper	: ASCII code data (high order)
	Second argument: in_lower	: ASCII code data (low order)
Return Value	Returns the converted binary data.	

5.14 Flowcharts

5.14.1 Main USB Processing

Figure 5.9 shows the flowchart for main USB processing.

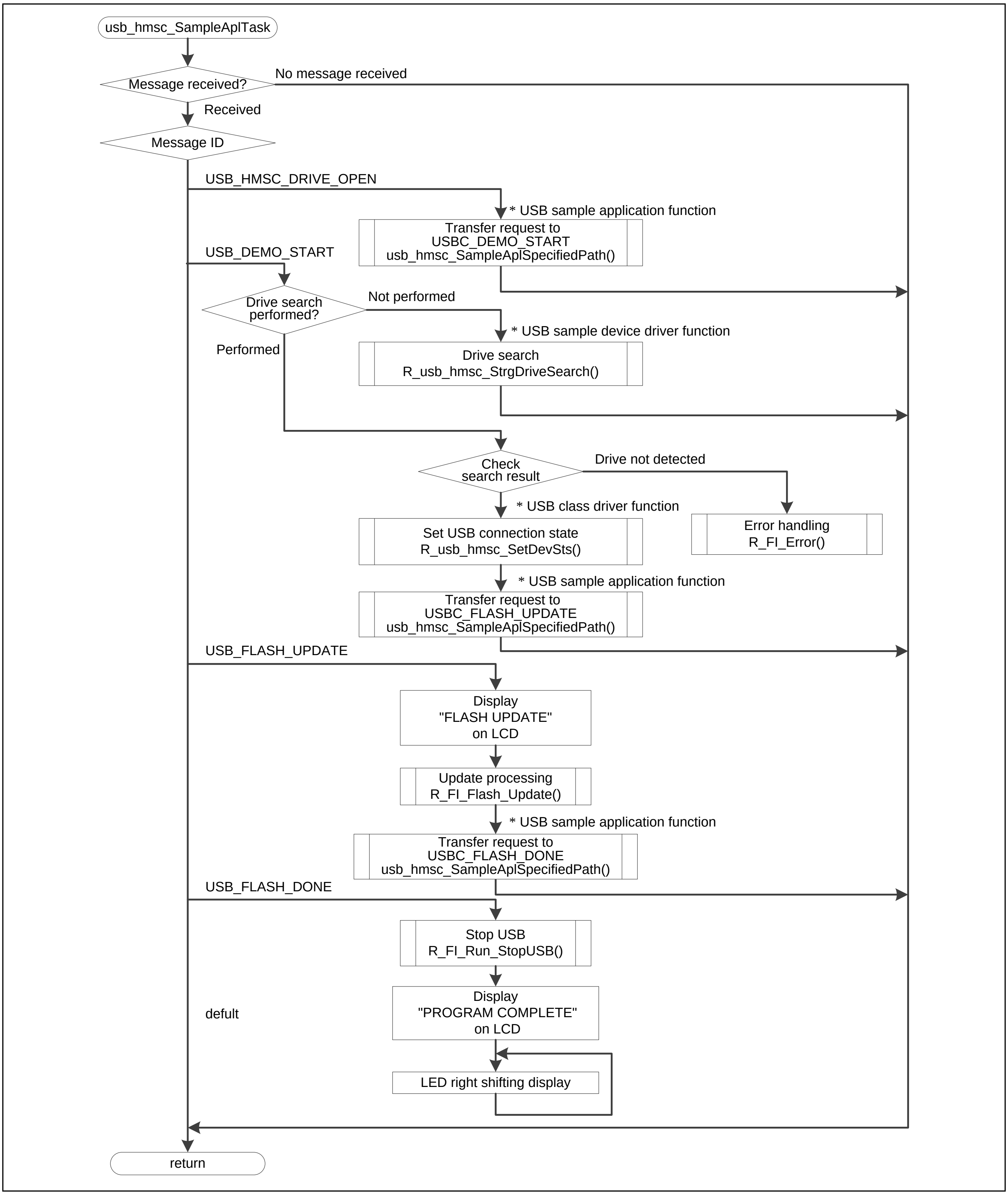

usb_hmsc_SampleAplTask
Message received?
No message received
Received
Message ID
USB_HMSC_DRIVE_OPEN
* USB sample application function
Transfer request to USBC_DEMO_START usb_hmsc_SampleAplSpecifiedPath()
USB_DEMO_START
Drive search performed?
Not performed
* USB sample device driver function
Performed
Drive search R_usb_hmsc_StrgDriveSearch()
Check search result
Drive not detected
* USB class driver function
Error handling R_FI_Error()
Set USB connection state R_usb_hmsc_SetDevSts()
* USB sample application function
Transfer request to USBC_FLASH_UPDATE usb_hmsc_SampleAplSpecifiedPath()
USB_FLASH_UPDATE
Display "FLASH UPDATE" on LCD
Update processing R_FI_Flash_Update()
* USB sample application function
Transfer request to USBC_FLASH_DONE usb_hmsc_SampleAplSpecifiedPath()
USB_FLASH_DONE
Stop USB R_FI_Run_StopUSB()
defult
Display "PROGRAM COMPLETE" on LCD
LED right shifting display
return

Figure 5.9 Main USB Processing

5.14.2 Mode Entry

Figure 5.10 shows the flowchart for mode entry.

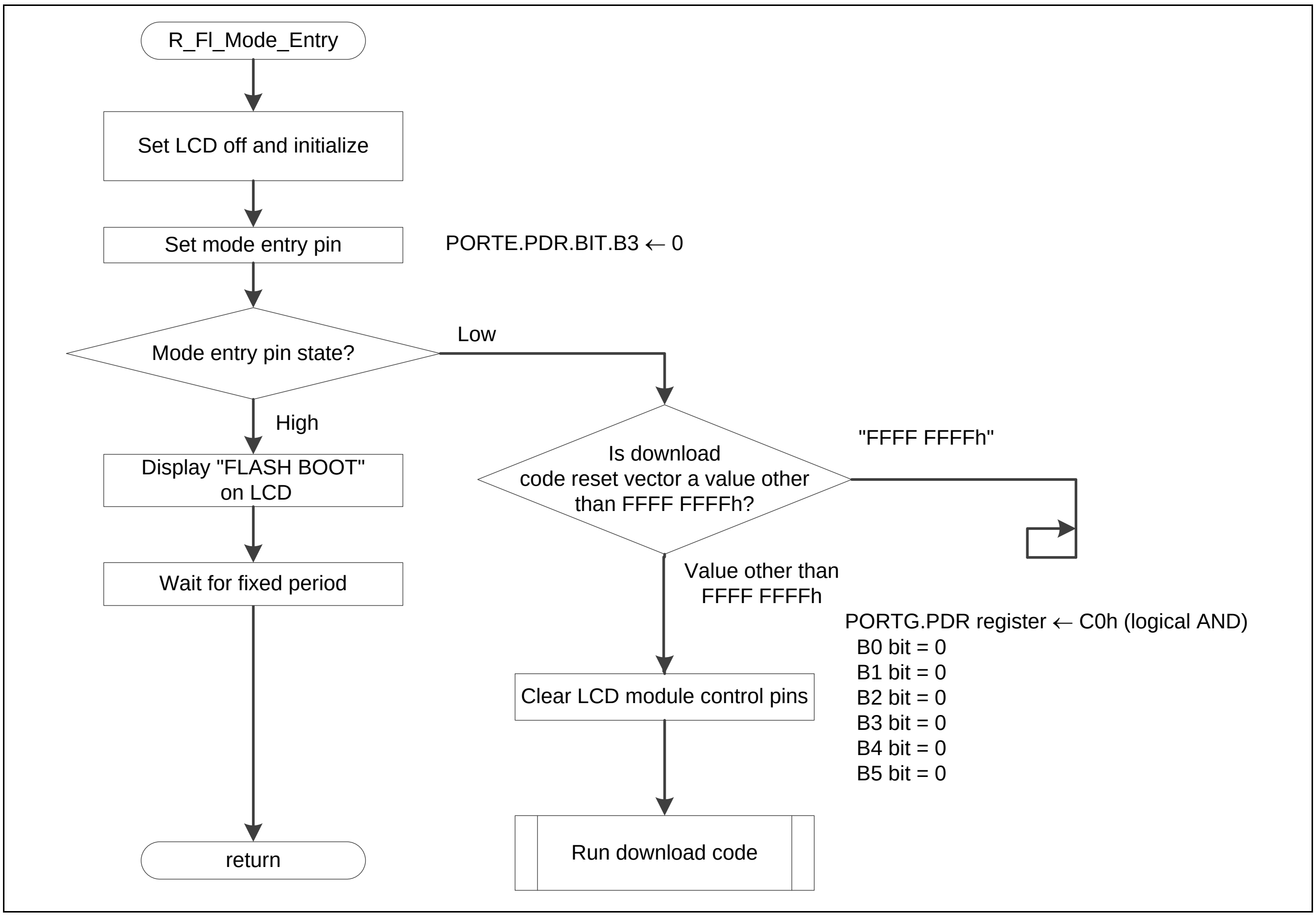

R_Fl_Mode_Entry
Set LCD off and initialize
Set mode entry pin
PORTE.PDR.BIT.B3 ← 0
Mode entry pin state?
Low
High
Display "FLASH BOOT" on LCD
Wait for fixed period
return
Is download code reset vector a value other than FFFF FFFFh?
"FFFF FFFFh"
Value other than FFFF FFFFh
PORTG.PDR register ← C0h (logical AND)
B0 bit = 0
B1 bit = 0
B2 bit = 0
B3 bit = 0
B4 bit = 0
B5 bit = 0
Clear LCD module control pins
Run download code

Figure 5.10 Mode Entry

5.14.3 Main Write Processing

Figure 5.11 shows the flowchart for main write processing.

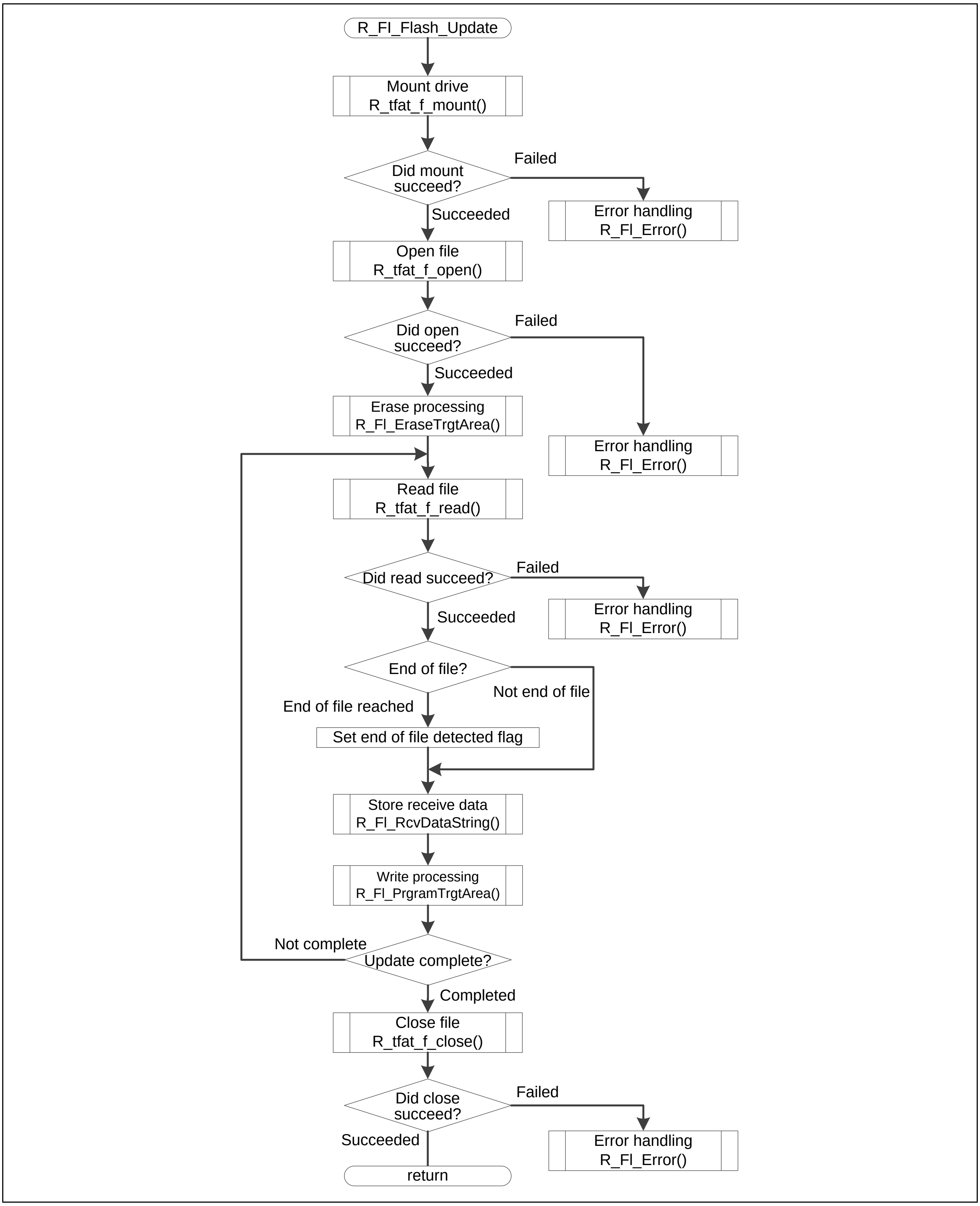


Figure 5.11 Main Write Processing

5.14.4 Erase Processing

Figure 5.12 shows the flowchart for erase processing.

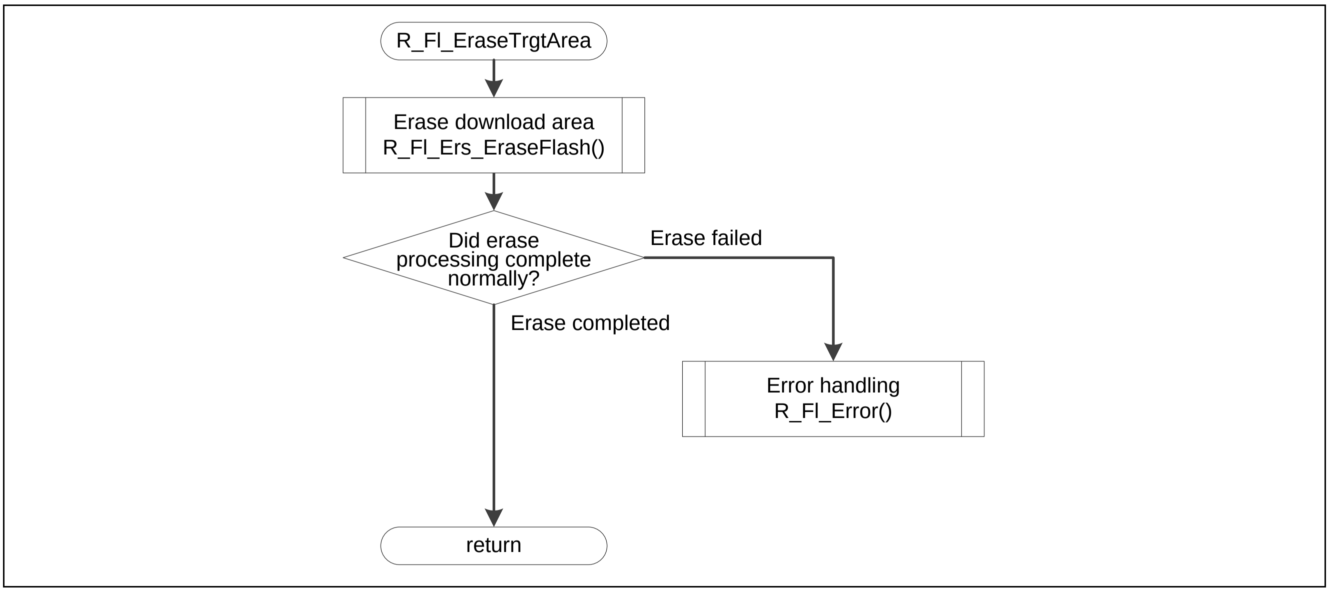


Figure 5.12 Erase Processing

5.14.5 Erase Download Area

Figure 5.13 shows the flowchart for erase download area.

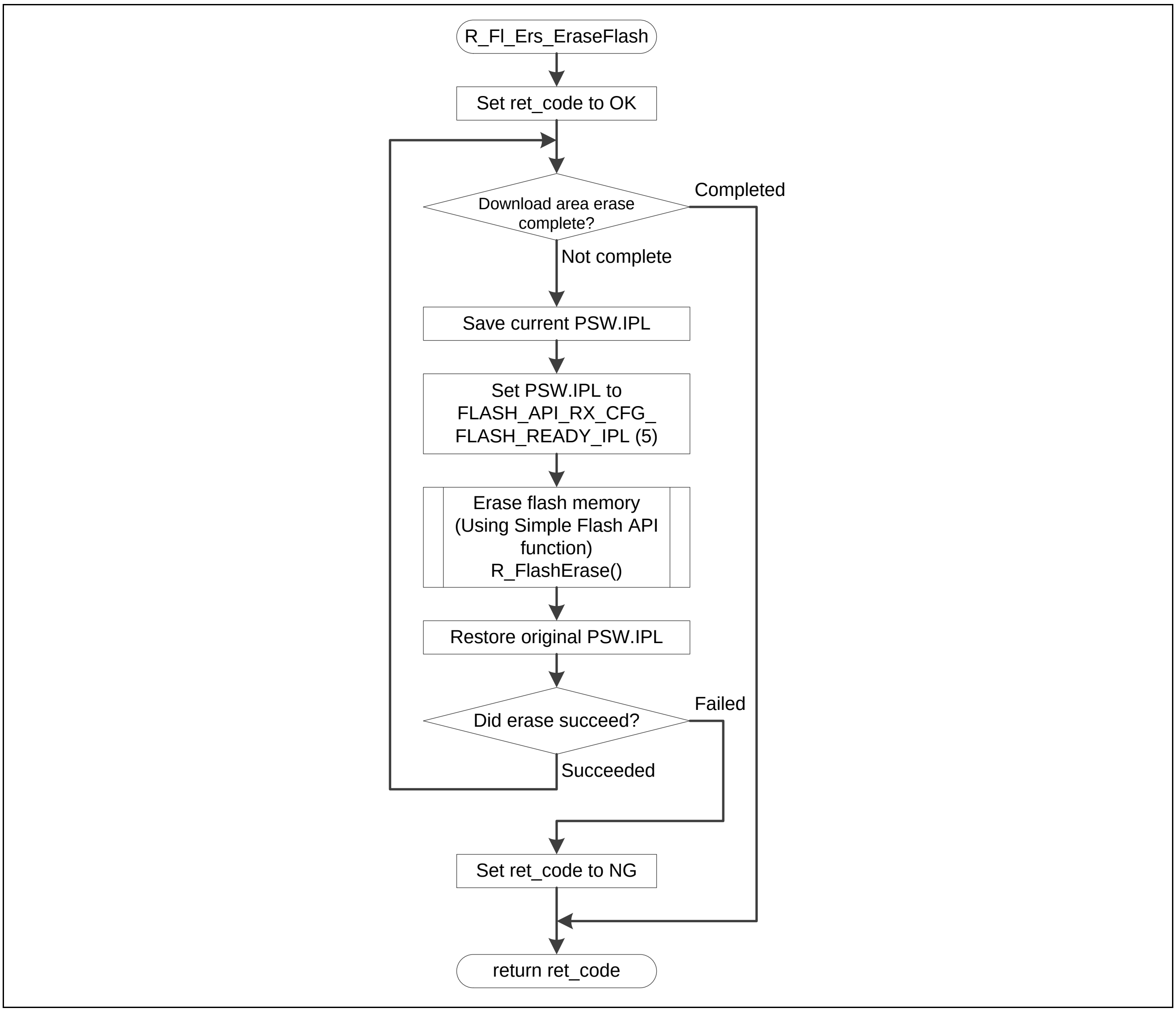


Figure 5.13 Erase Download Area

5.14.6 Write Processing

Figure 5.14 shows the flowchart for write processing.

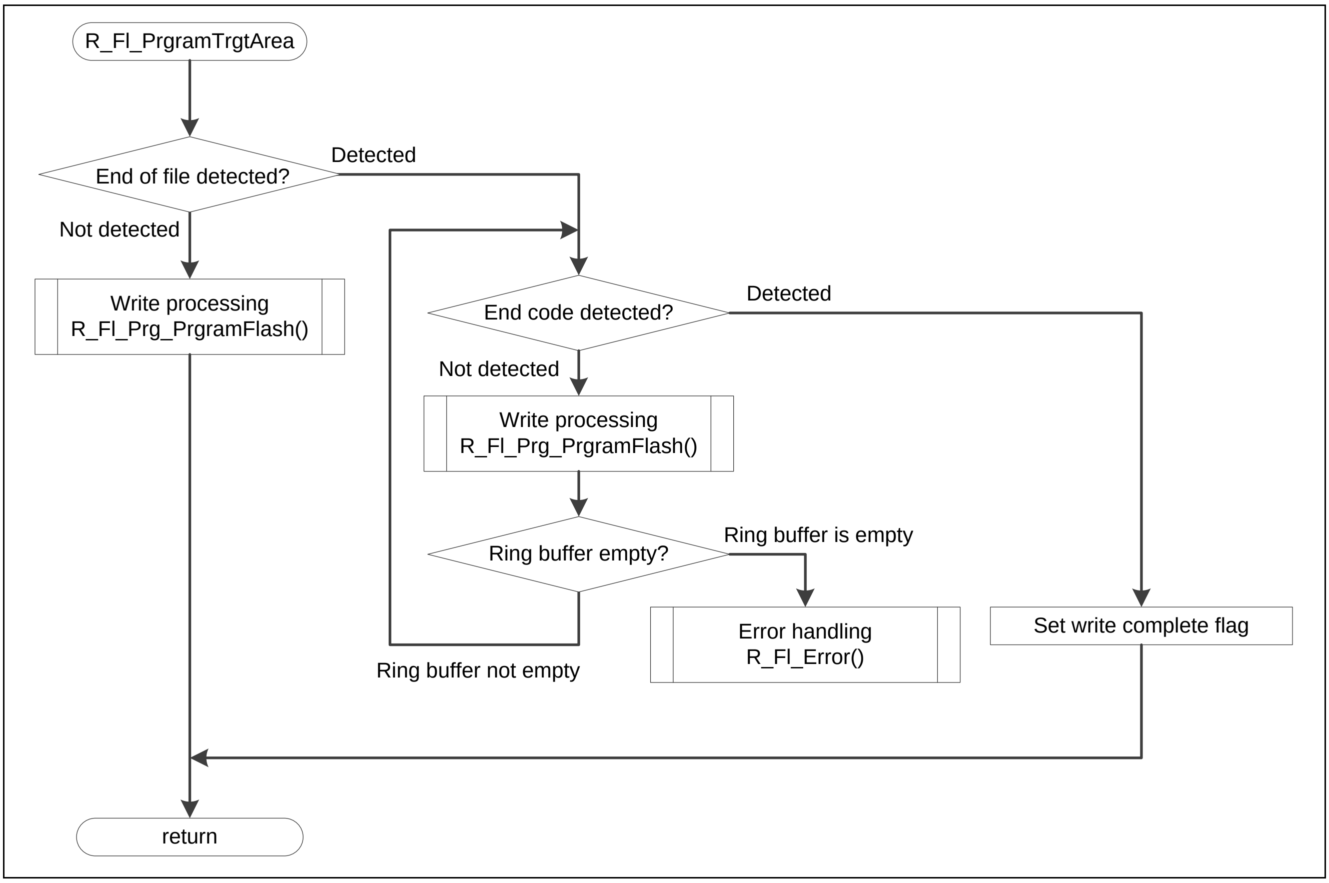


Figure 5.14 Write Processing

5.14.7 Download Area Write Operation

Figure 5.15 shows the flowchart for download area write operation.

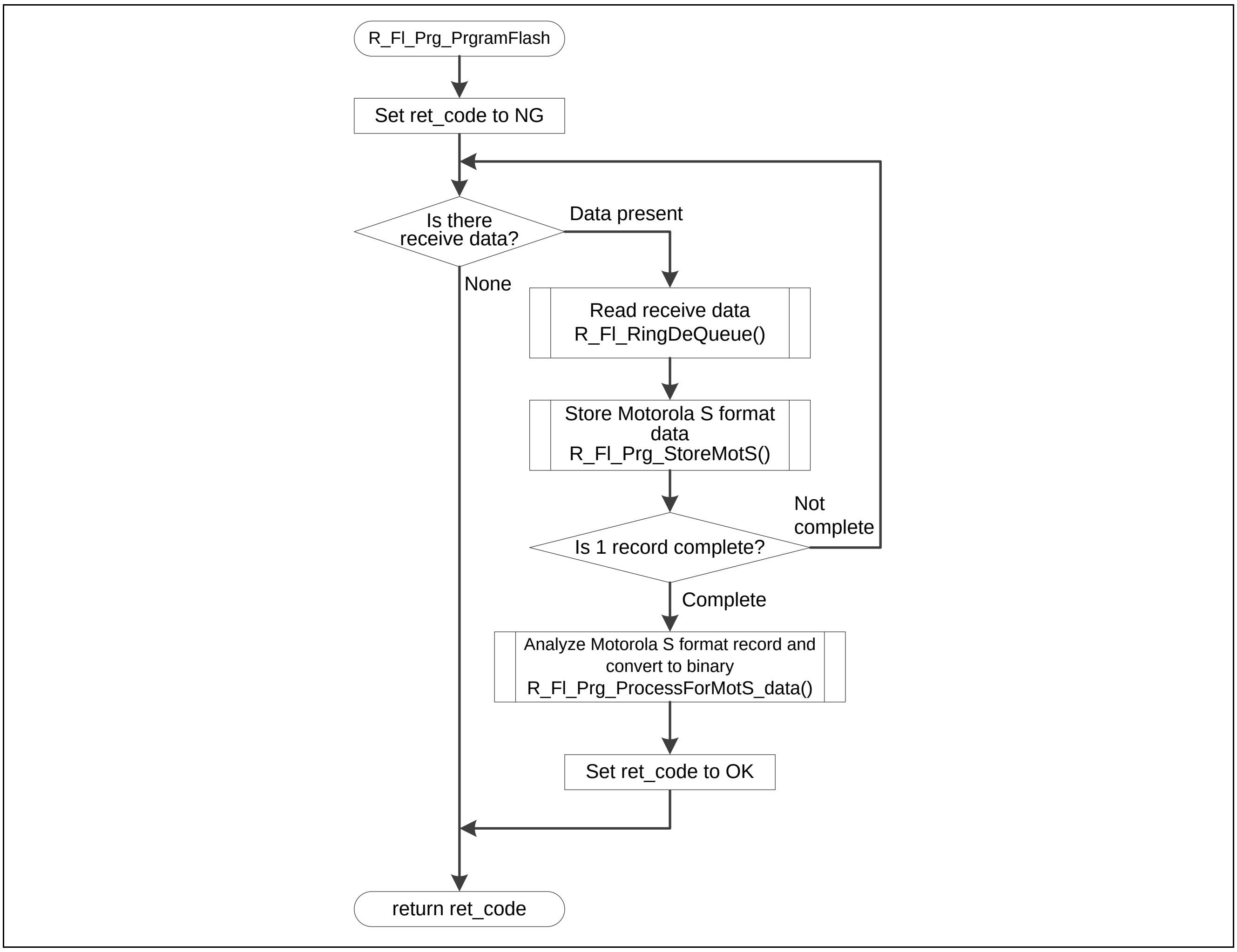


Figure 5.15 Download Area Write Operation

5.14.8 Motorola S Format Header Analysis, Conversion to Binary, and Write Operations

Figure 5.16 shows the flowchart for Motorola S format header analysis, conversion to binary, and write operations.

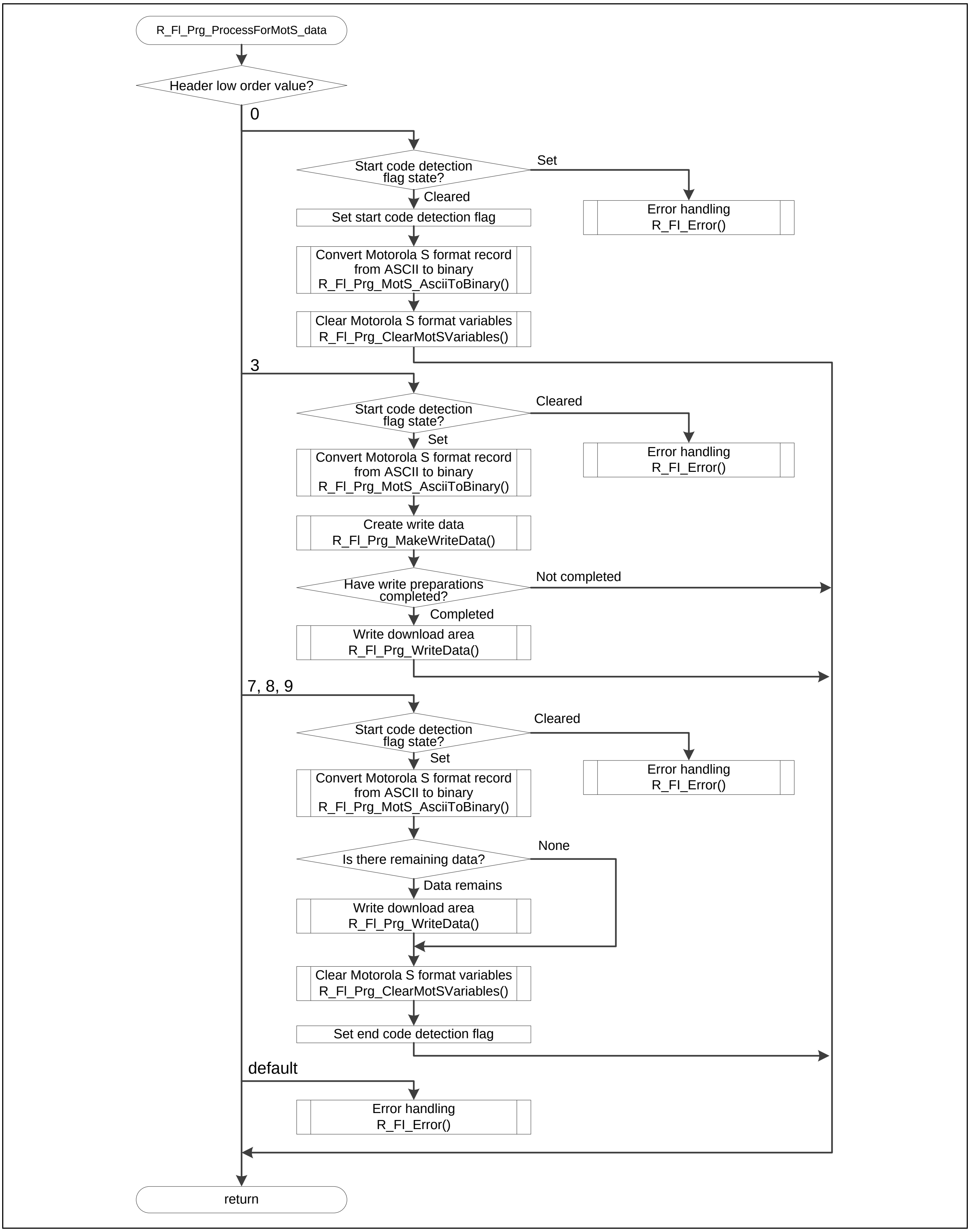

R_Fl_Prg_ProcessForMotS_data
Header low order value?
0
Start code detection flag state?
Set
Cleared
Set start code detection flag
Error handling R_Fl_Error()
Convert Motorola S format record from ASCII to binary R_Fl_Prg_MotS_AsciiToBinary()
Clear Motorola S format variables R_Fl_Prg_ClearMotSVariables()
3
Start code detection flag state?
Cleared
Set
Convert Motorola S format record from ASCII to binary R_Fl_Prg_MotS_AsciiToBinary()
Error handling R_Fl_Error()
Create write data R_Fl_Prg_MakeWriteData()
Have write preparations completed?
Not completed
Completed
Write download area R_Fl_Prg_WriteData()
7, 8, 9
Start code detection flag state?
Cleared
Set
Convert Motorola S format record from ASCII to binary R_Fl_Prg_MotS_AsciiToBinary()
Error handling R_Fl_Error()
Is there remaining data?
None
Data remains
Write download area R_Fl_Prg_WriteData()
Clear Motorola S format variables R_Fl_Prg_ClearMotSVariables()
Set end code detection flag
default
Error handling R_Fl_Error()
return

Figure 5.16 Motorola S Format Header Analysis, Conversion to Binary, and Write Operations

5.14.9 Motorola S Format Data Store Operation

Figure 5.17 shows the flowchart for Motorola S format data store operation.

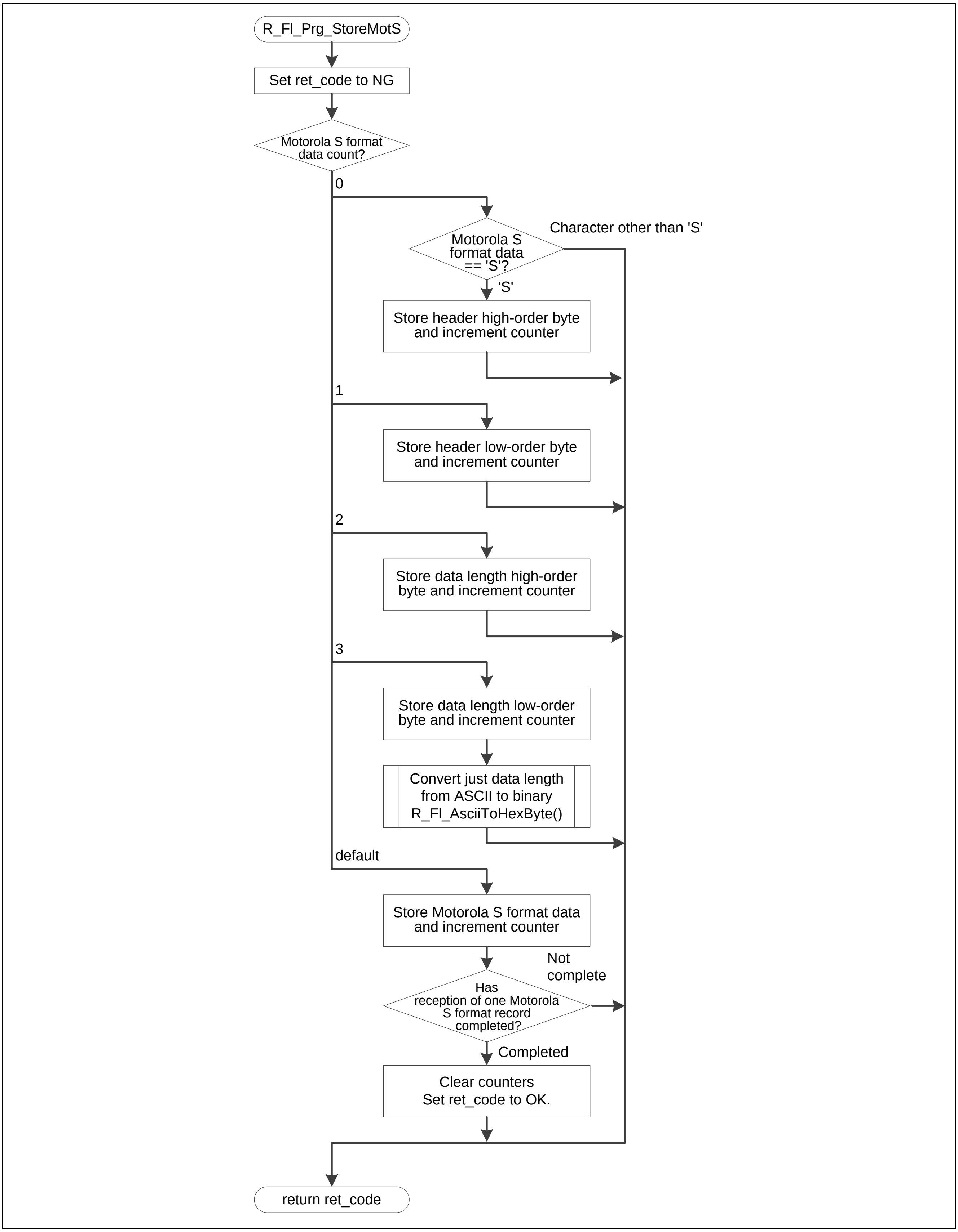

R_Fl_Prg_StoreMotS
Set ret_code to NG
Motorola S format data count?
0
Motorola S format data == 'S'?
Character other than 'S'
'S'
Store header high-order byte and increment counter
1
Store header low-order byte and increment counter
2
Store data length high-order byte and increment counter
3
Store data length low-order byte and increment counter
Convert just data length from ASCII to binary R_Fl_AsciiToHexByte()
default
Store Motorola S format data and increment counter
Has reception of one Motorola S format record completed?
Not complete
Completed
Clear counters
Set ret_code to OK.
return ret_code

Figure 5.17 Motorola S Format Data Store Operation

5.14.10 Motorola S Format Data ASCII to Binary Conversion

Figure 5.18 shows the flowchart for Motorola S format data ASCII to binary conversion.

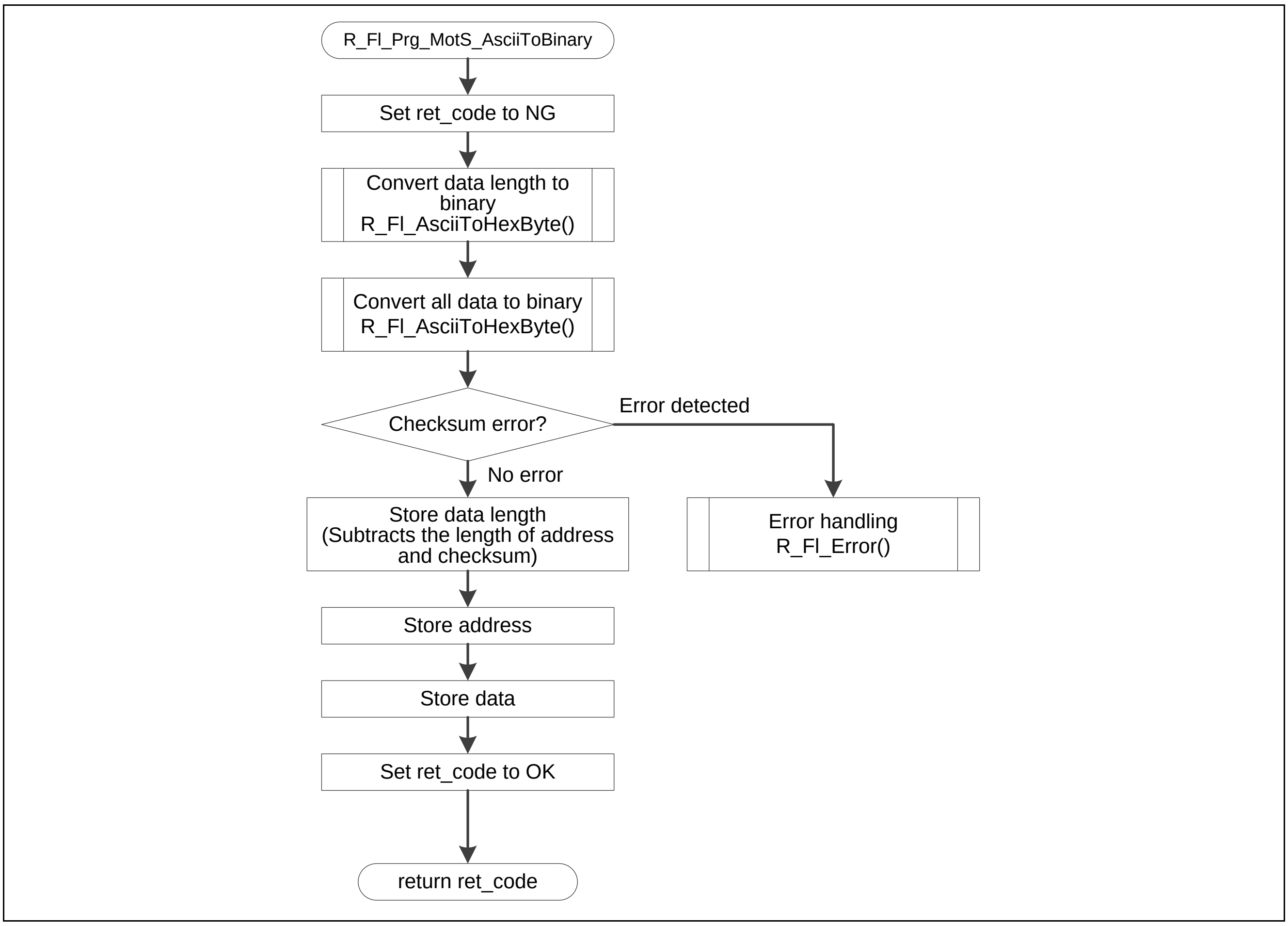


Figure 5.18 Motorola S Format Data ASCII to Binary Conversion

5.14.11 Download Area Write Data Creation

Figure 5.19 shows the flowchart for creating the data to be written to the download area.

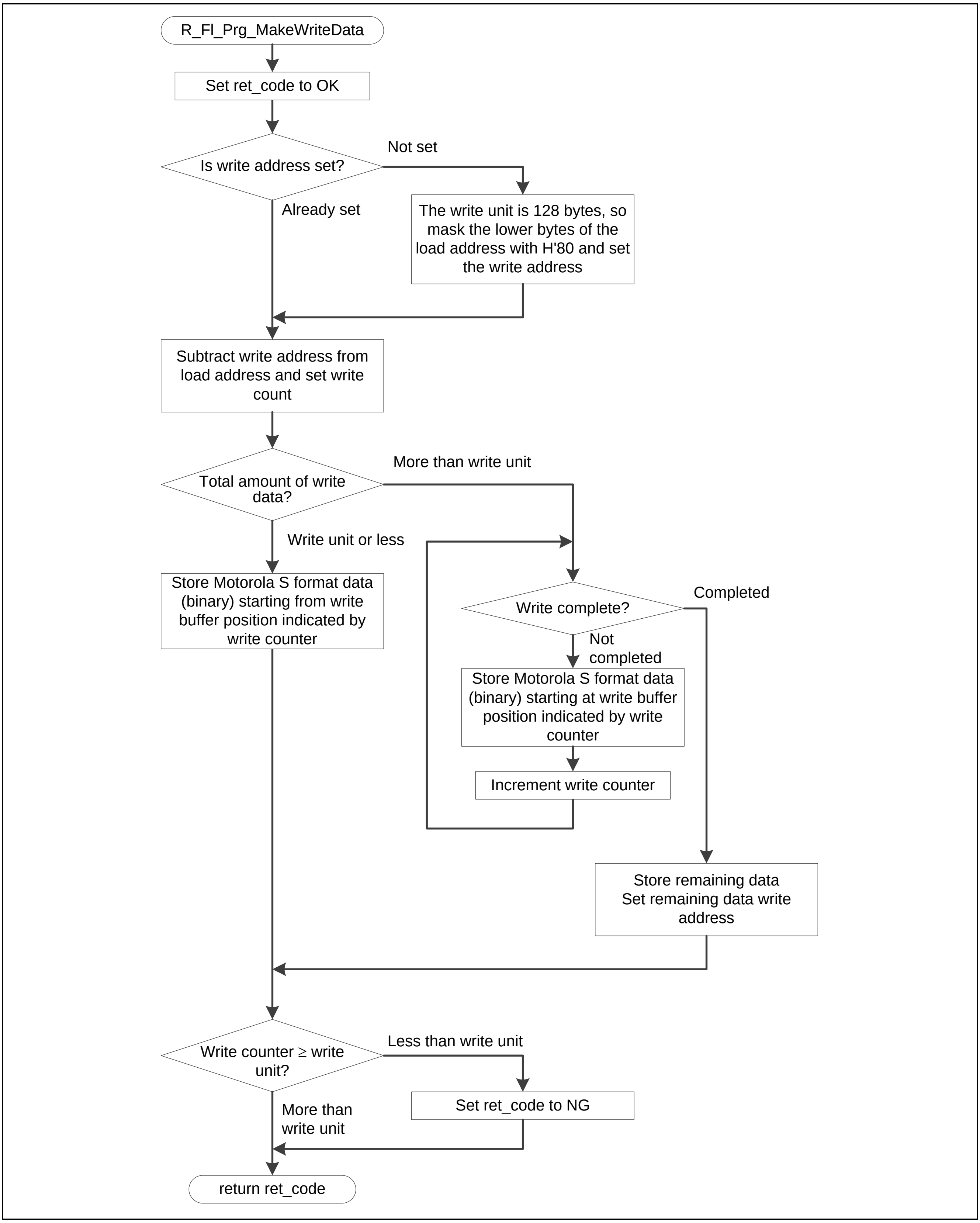

R_Fl_Prg_MakeWriteData
Set ret_code to OK
Is write address set?
Not set
Already set
The write unit is 128 bytes, so mask the lower bytes of the load address with H'80 and set the write address
Subtract write address from load address and set write count
Total amount of write data?
More than write unit
Write unit or less
Store Motorola S format data (binary) starting from write buffer position indicated by write counter
Write complete?
Completed
Not completed
Store Motorola S format data (binary) starting at write buffer position indicated by write counter
Increment write counter
Store remaining data Set remaining data write address
Write counter ≥ write unit?
Less than write unit
Set ret_code to NG
More than write unit
return ret_code

Figure 5.19 Download Area Write Data Creation

5.14.12 Download Area Write

Figure 5.20 shows the flowchart for download area write.

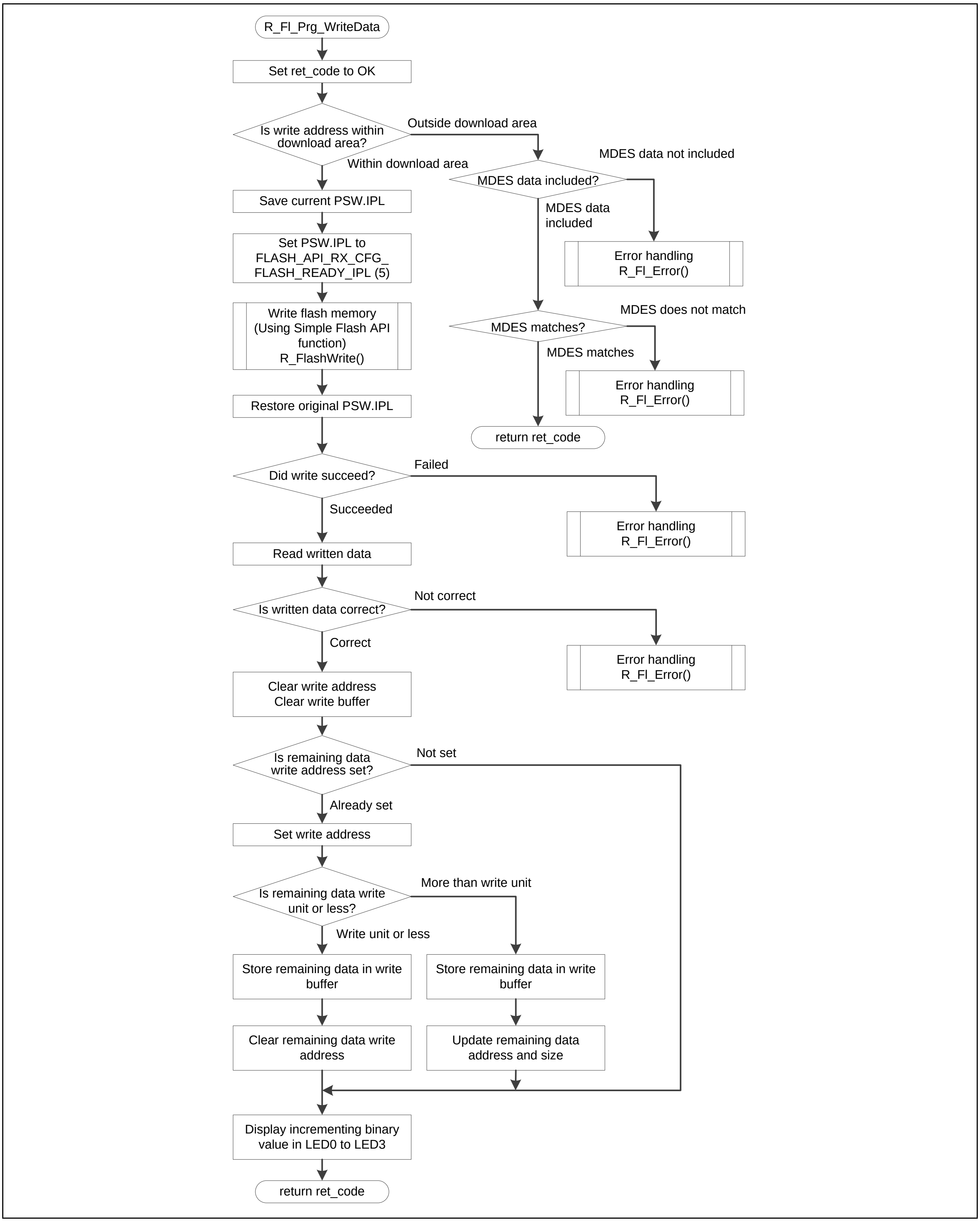

R_Fl_Prg_WriteData
Set ret_code to OK
Is write address within download area?
Outside download area
Within download area
MDES data included?
MDES data not included
MDES data included
Error handling R_Fl_Error()
MDES matches?
MDES does not match
MDES matches
Error handling R_Fl_Error()
return ret_code
Save current PSW.IPL
Set PSW.IPL to FLASH_API_RX_CFG_ FLASH_READY_IPL (5)
Write flash memory (Using Simple Flash API function) R_FlashWrite()
Restore original PSW.IPL
Did write succeed?
Failed
Succeeded
Error handling R_Fl_Error()
Read written data
Is written data correct?
Not correct
Correct
Error handling R_Fl_Error()
Clear write address Clear write buffer
Is remaining data write address set?
Not set
Already set
Set write address
Is remaining data write unit or less?
More than write unit
Write unit or less
Store remaining data in write buffer
Store remaining data in write buffer
Clear remaining data write address
Update remaining data address and size
Display incrementing binary value in LED0 to LED3
return ret_code

Figure 5.20 Download Area Write

5.14.13 Clear Motorola S Format Data Related Variables

Figure 5.21 shows the flowchart for clear Motorola S format data related variables.

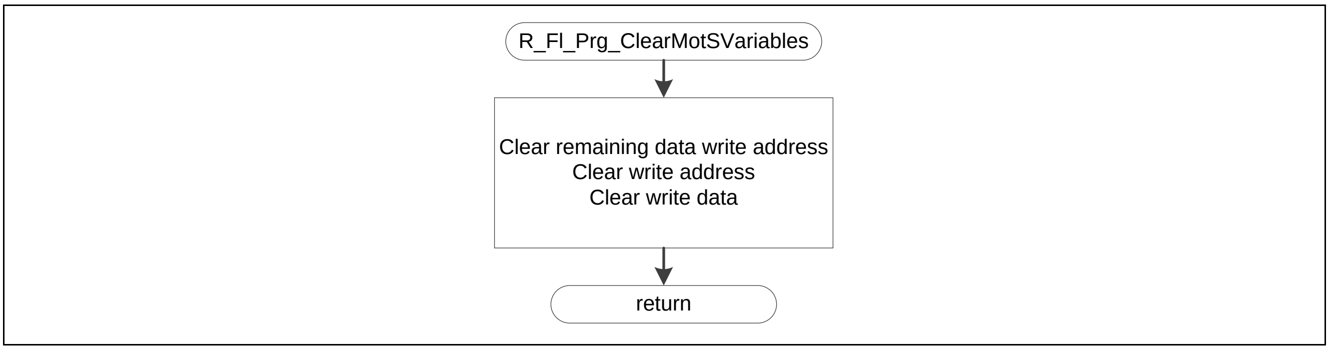

R_Fl_Prg_ClearMotSVariables
Clear remaining data write address
Clear write address
Clear write data
return

Figure 5.21 Clear Motorola S Format Data Related Variables

5.14.14 Store USB Receive Data

Figure 5.22 shows the flowchart for store USB receive data.

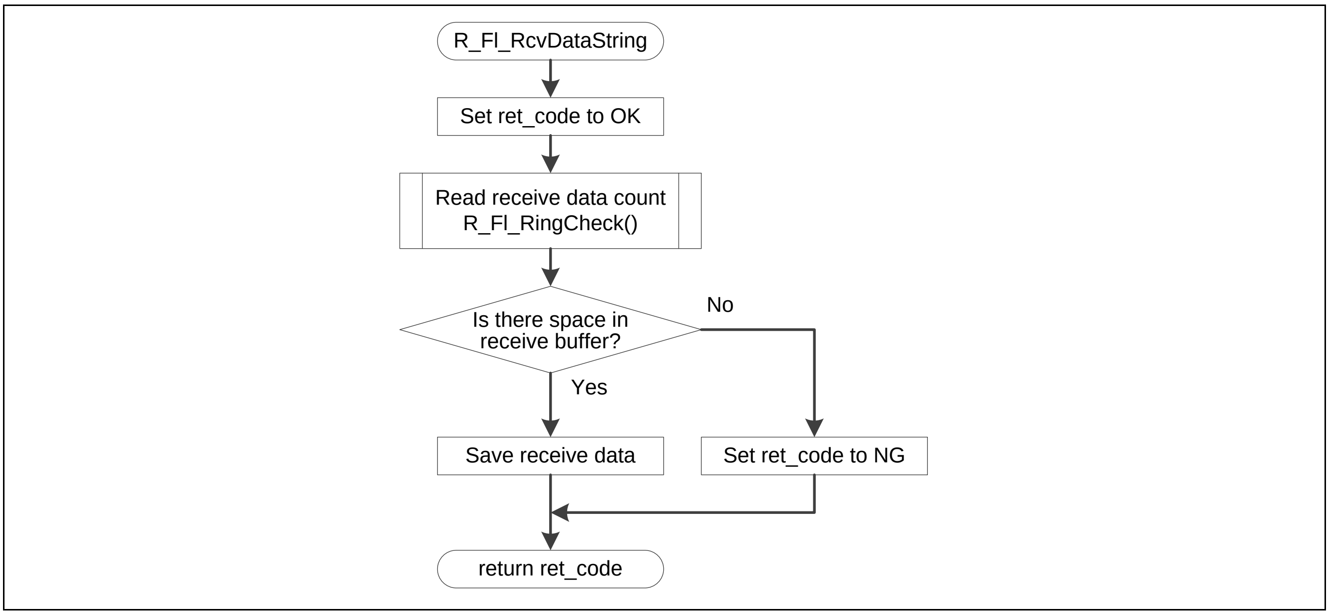


Figure 5.22 Store USB Receive Data

5.14.15 Check for Empty Space in Receive Data Storage Ring Buffer

Figure 5.23 shows the flowchart for check for empty space in receive data storage ring buffer.

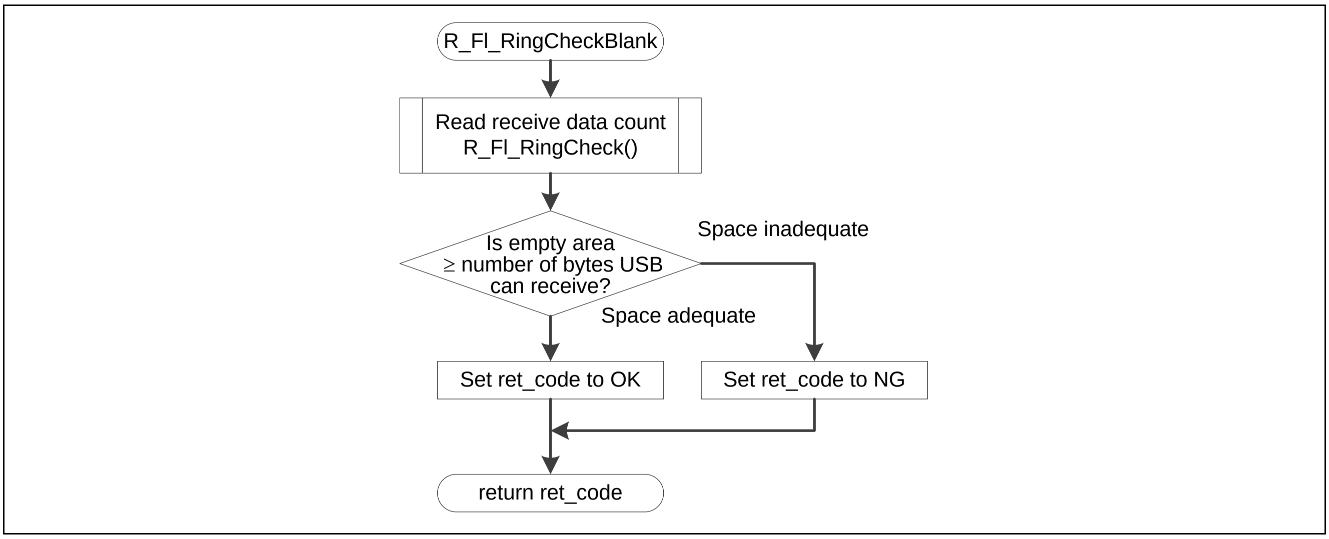


Figure 5.23 Check for Empty Space in Receive Data Storage Ring Buffer

5.14.16 Stop USB

Figure 5.24 shows the flowchart for stop USB.

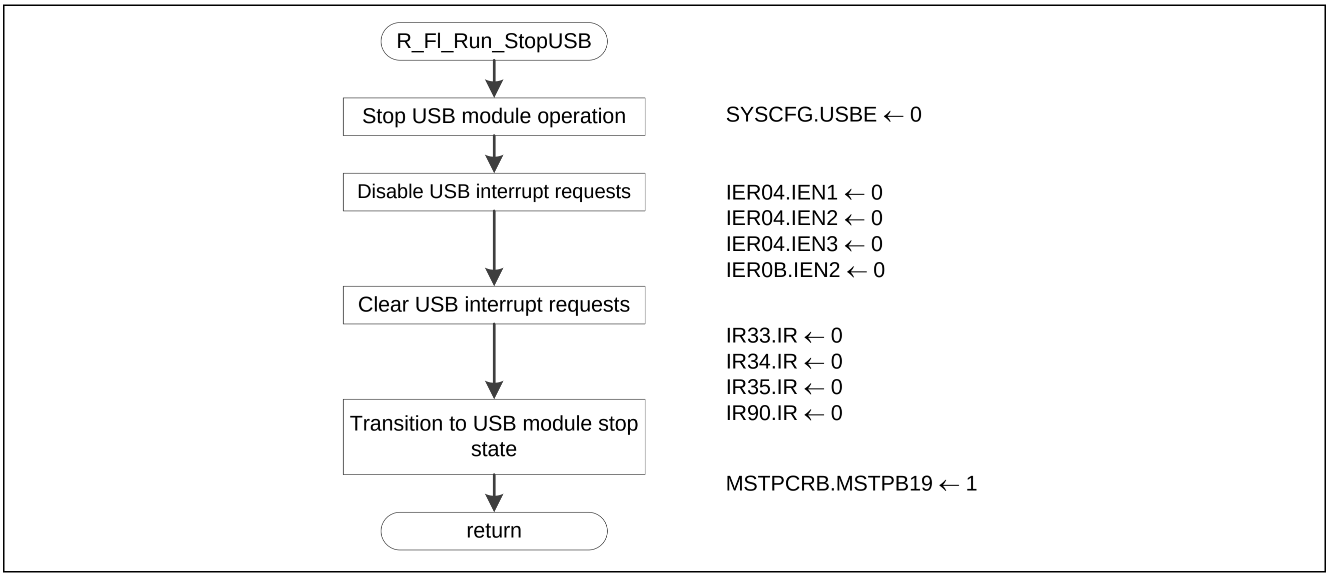


Figure 5.24 Stop USB

5.14.17 Error Handling

Figure 5.25 shows the flowchart for error handling.

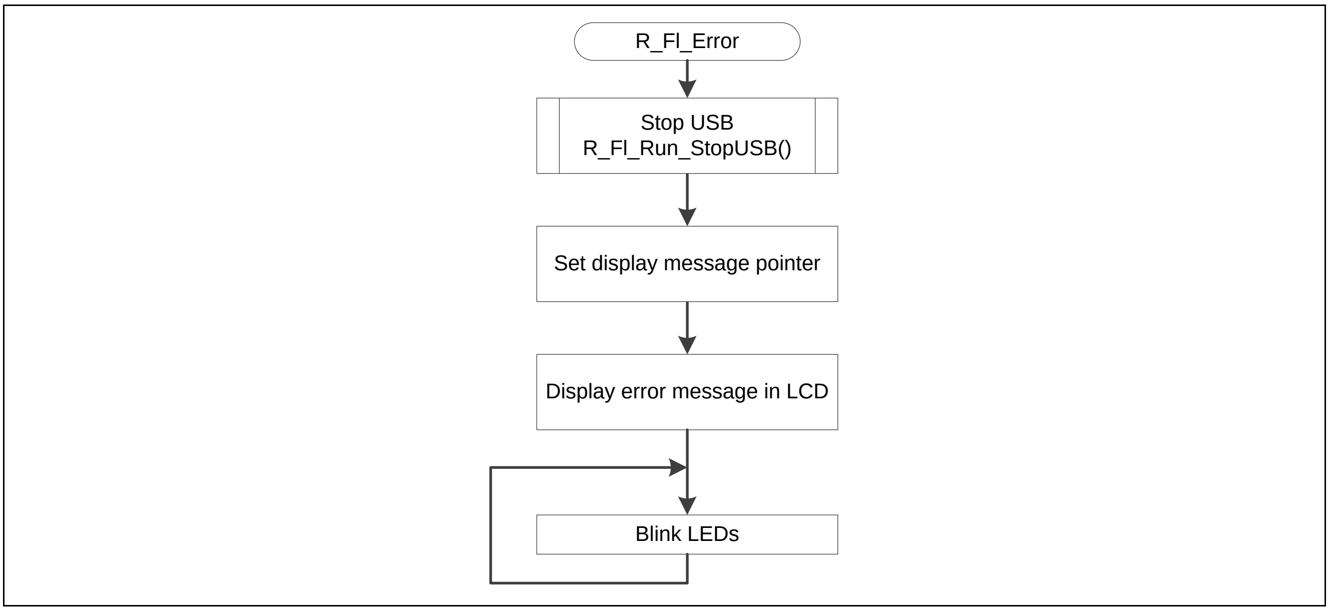


Figure 5.25 Error Handling

5.14.18 Store Data in Receive Data Ring Buffer

Figure 5.26 shows the flowchart for store data in receive data ring buffer.

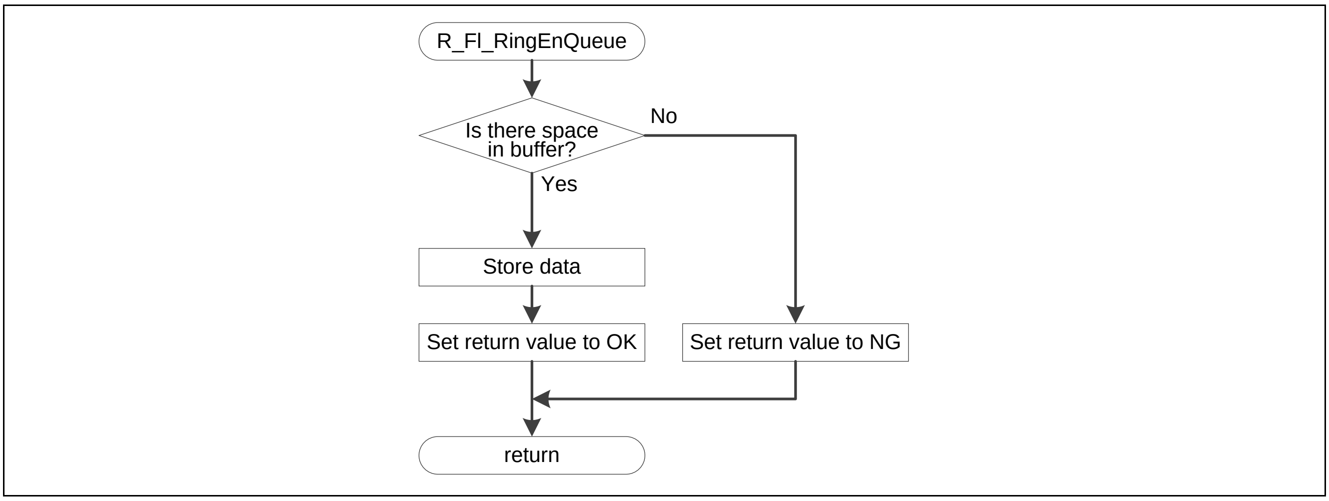


Figure 5.26 Store Data in Receive Data Ring Buffer

5.14.19 Read Data from Receive Data Ring Buffer

Figure 5.27 shows the flowchart for read data from receive data ring buffer.

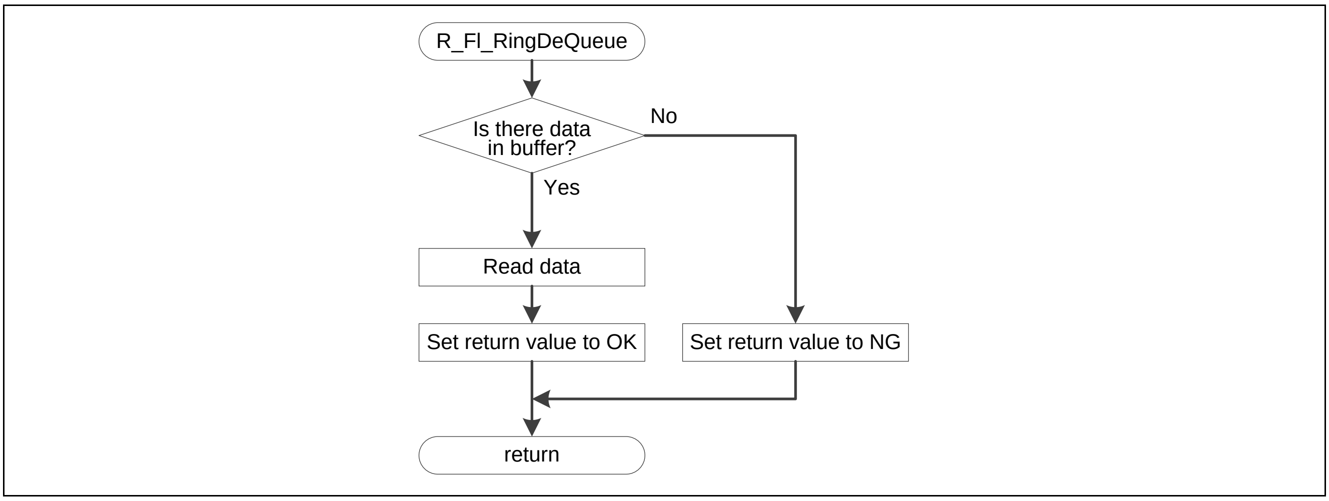


Figure 5.27 Read Data from Receive Data Ring Buffer

5.14.20 Check Data Count in Receive Data Ring Buffer

Figure 5.28 shows the flowchart for check data count in receive data ring buffer.

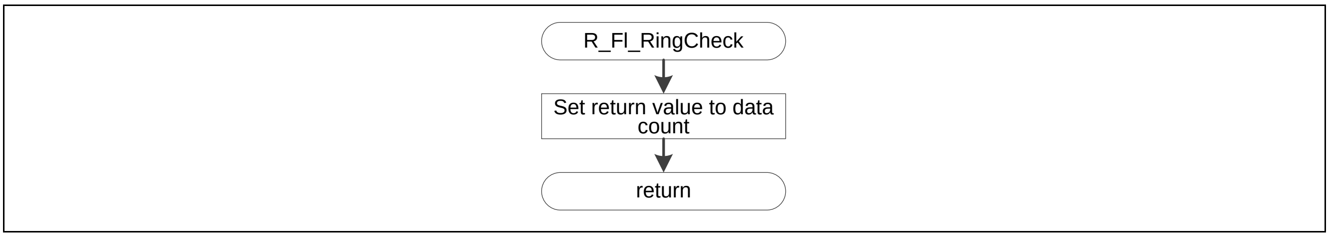


Figure 5.28 Check Data Count in Receive Data Ring Buffer

5.14.21 Convert Data from ASCII to Binary

Figure 5.29 shows the flowchart for convert data from ASCII to binary.

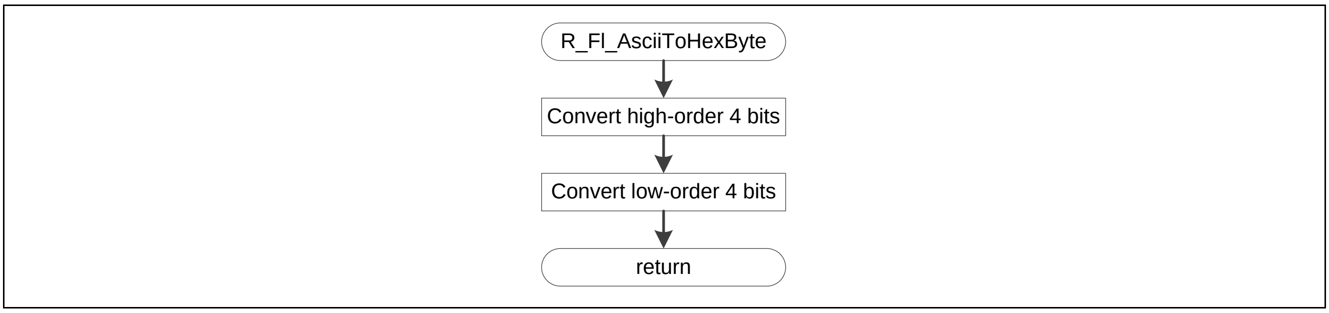


Figure 5.29 Convert Data from ASCII to Binary

6. Sample Download Code

This application note includes a sample download code file (download.zip). This program lights in sequence the LEDs on the board described in section 2, Operation Confirmation Conditions. Refer to this program for examples of download reset vector and section settings. Note that the download code is expected to use 512 KB of ROM.

7. Motorola S Format

This section describes the Motorola S format supported by the sample code.

7.1 Record Formats

Figure 7.1 shows the record formats supported by the sample code.

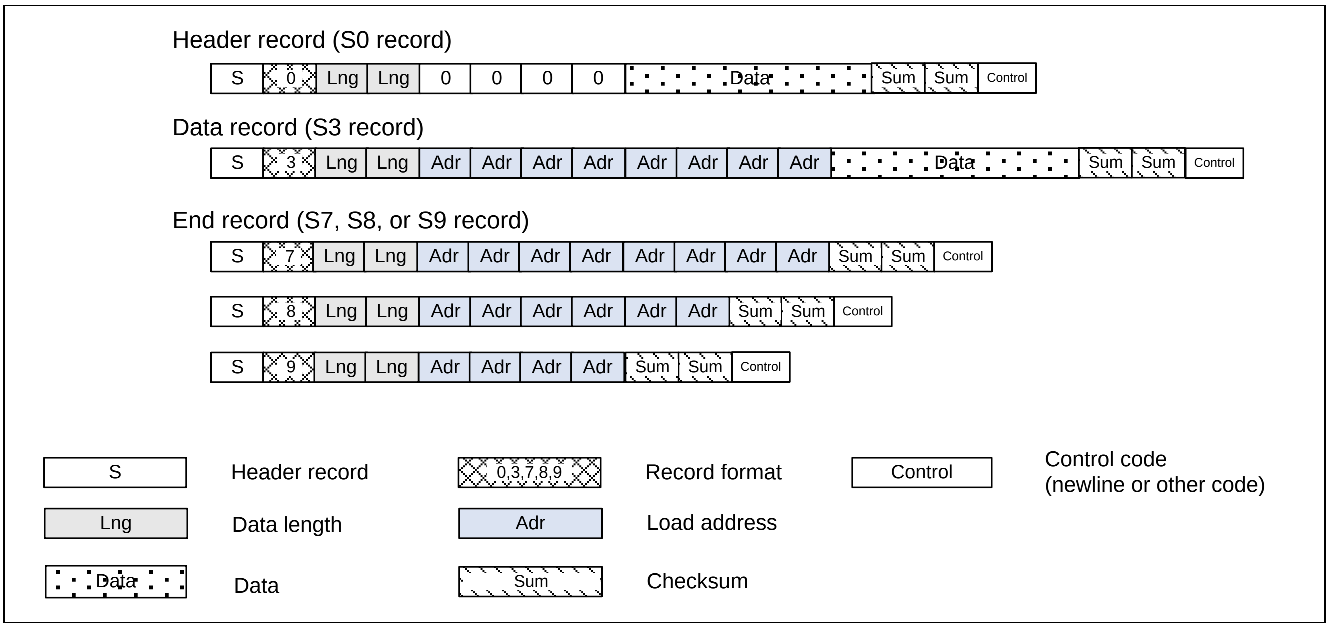

Header record (S0 record)
S 0 Lng Lng 0 0 0 0 Data Sum Sum Control
Data record (S3 record)
S 3 Lng Lng Adr Adr Adr Adr Adr Adr Adr Adr Data Sum Sum Control
End record (S7, S8, or S9 record)
S 7 Lng Lng Adr Adr Adr Adr Adr Adr Adr Adr Sum Sum Control
S 8 Lng Lng Adr Adr Adr Adr Adr Adr Sum Sum Control
S 9 Lng Lng Adr Adr Adr Adr Sum Sum Control
S
Header record
0,3,7,8,9
Record format
Control
Control code (newline or other code)
Lng
Data length
Adr
Load address
Data
Data
Sum
Checksum

Figure 7.1 Record Formats Supported by Sample Code

7.2 Record Structure

Figure 7.2 shows the record structure supported by the sample code. Motorola S format record sequences with orders other than those shown in figure 7.2 are not supported.

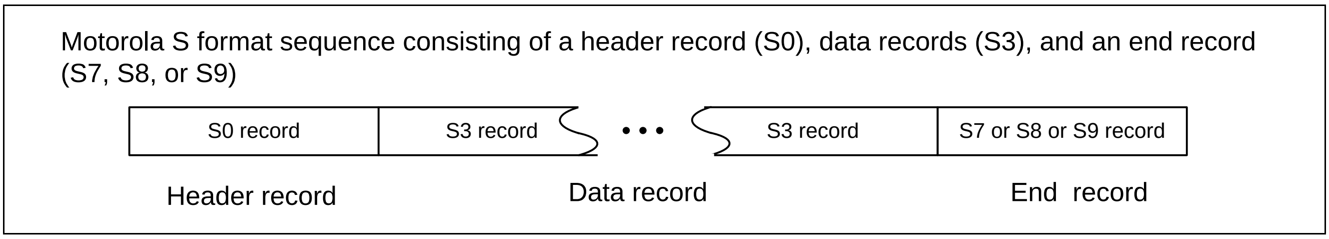

Motorola S format sequence consisting of a header record (S0), data records (S3), and an end record (S7, S8, or S9)
S0 record
S3 record
• • •
S3 record
S7 or S8 or S9 record
Header record
Data record
End record

Figure 7.2 Record Structure Supported by the Sample Code

7.3 Load Address

The sample code only supports Motorola S format files with increasing load addresses. Do not use Motorola S format files with decreasing order or out of order load addresses.

7.4 Error Detection

The sample code detects errors if there are problems with the Motorola S format file received.

(a) Checksum error

The sample code verifies the checksum at each received Motorola S format record. A checksum error is detected if that verification finds an abnormality.

(b) Format error

A format error is detected if the sample code receives a Motorola S format file that meets any of the following conditions.

- If an unsupported record (S1, S2, S4, S5, or S6) is detected
- If a header record (S0) is detected twice
- If a data record (S3) or an end record (S7, S8, or S9) is detected before a header record (S0).

Figure 7.3 shows the format error detection conditions.

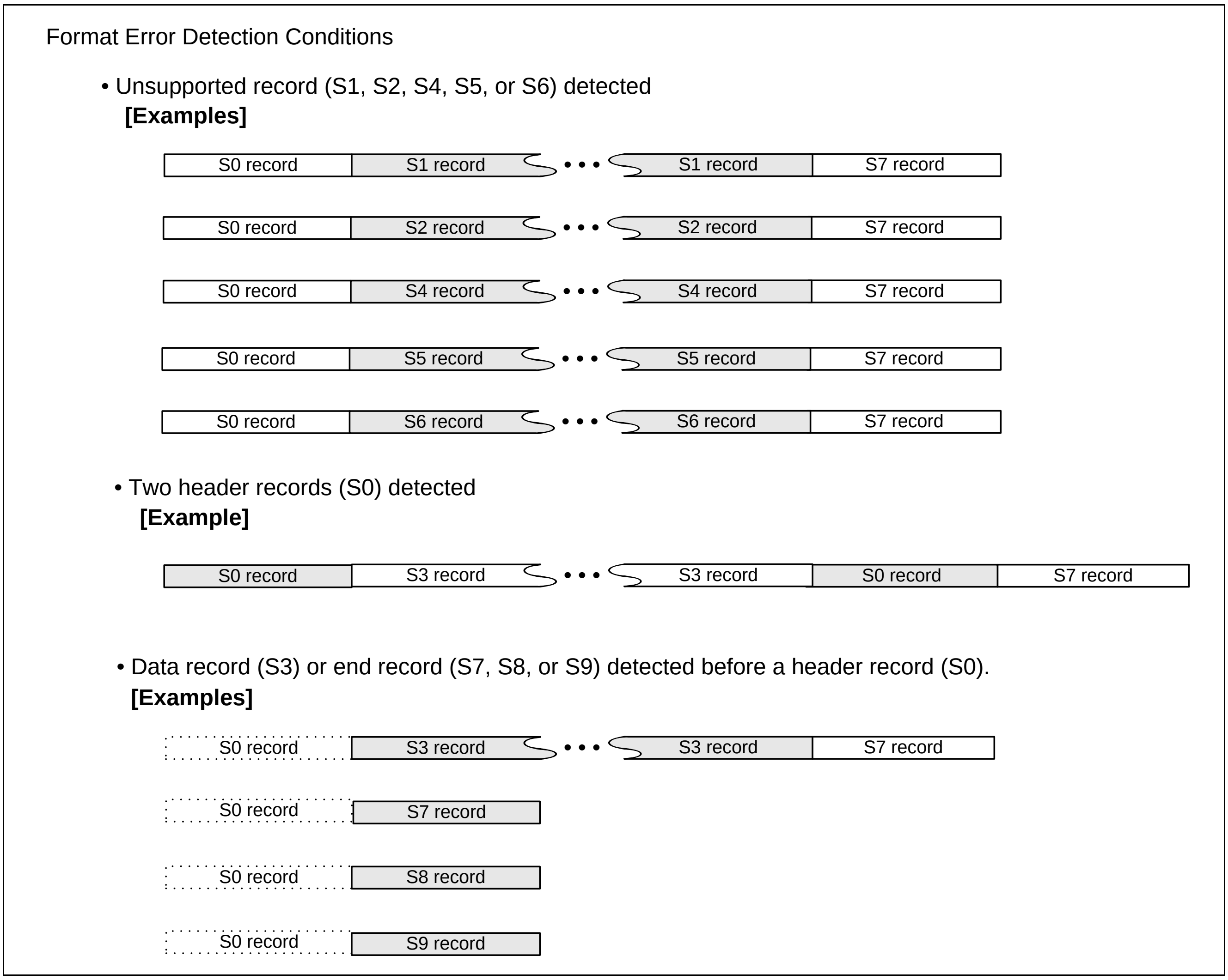


Figure 7.3 Format Error Detection Conditions

(c) Address error

An address error is detected if write data for any address outside the download area is received.

8. Notes

8.1 USB Disconnection During Write or Erase

Do not disconnect the USB while erasing or writing the download area.

8.2 HEW Settings

The sample code runs by copying the code in ROM to RAM during flash memory write operations. See the RX600 & RX200 Series Simple Flash API for RX application note for details on the settings.

8.3 Fixed Vector Table Interrupts

Of the fixed vector table interrupts, this sample code only handles the reset interrupt. If any other fixed vector table interrupts are used, the sample code must be modified to handle those interrupts.

8.4 Reset Vector for the Download Code

The execution start position for the download code written using the sample code is determined by the value written at the download reset vector (FFFD FFFCh). Therefore the download code must be set up so that its reset vector is allocated at FFFD FFFCh. See section 5.2, Download Code Execution Start Position, for details.

Also, see section 6, Sample Download Code, for details on the download code.

8.5 Changing the ROM Capacity

The ROM capacity of the microcontroller used by the sample code is 512 KB. If a microcontroller with a ROM capacity of 384 KB or 256 KB, is used, change FL_END_BLOCK_NUM #define directive in the file r_Flash_main.h to match the capacity used.

Table 8.1 lists the ROM capacities.

Table 8.1 ROM Capacities

Catalog Number	ROM Capacity	Download Area ROM Capacity	Download Area Start Address	Download Area Block Numbers
R5F563TE	512K	384K	FFF8 0000h	EB14 to EB37
R5F563TC	384K	256K	FFFA 0000h	EB14 to EB29
R5F563TB	256K	128K	FFFC 0000h	EB14 to EB21

Note: X: N or 1

8.6 while(1) Processing

Note that the sample code locks up by executing a while(1) loop if the USB ring buffer overflows.

8.7 Endian Order

The sample code in this application note supports both little endian and big endian orders. Note that the endian order settings of the flash boot loader and the download code must be the same.

8.7.1 Using Little Endian

When operating using the little endian order, perform the following settings.

1. Compiler option settings
   Specify "Little-endian data" as the compiler option endian setting. Use the little endian MDES value shown in 5.8, Option-Setting Memory.
2. Changes to the user system definitions file (r_usb_usrconfig.h)
   Set the value of the USB_CPUBYTE_PP macro definition to USB_BYTE_LITTLE_PP in the r_usb_usrconfig.h file.
3. Changing the TFAT library file (tfat_rx600_little.lib or tfat_rx600_big.lib)
   Specify $(WORKSPDIR)\WorkSpace\MSCFW\TFAT\lib\tfat_rx600_little.lib as the linker input library file option.

8.7.2 Using Big Endian

When operating using the big endian order, perform the following settings.

1. Compiler option settings
   Specify "Big-endian data" as the compiler option endian setting. Use the big endian MDES value shown in 5.8, Option-Setting Memory.
2. Changes to the user system definitions file (r_usb_usrconfig.h)
   Set the value of the USB_CPUBYTE_PP macro definition to USB_BYTE_BIG_PP in the r_usb_usrconfig.h file.
3. Changing the TFAT library file (tfat_rx600_little.lib or tfat_rx600_big.lib)
   Specify $(WORKSPDIR)\WorkSpace\MSCFW\TFAT\lib\tfat_rx600_big.lib as the linker input library file option.

8.8 Changes to the RX600 & RX200 Simple Flash API

This application note uses sample code from the RX600 & RX200 Simple Flash API. See the RX600 & RX200 Simple Flash API application note for the specifications of the RX600 & RX200 Simple Flash API.

8.8.1 Changes

The files in the RX600 & RX200 Simple Flash API that are changed are r_flash_api_rx_config, r_bsp_config.h, r_bsp.h, and r_flash_api_rx.c

- Changes to the file r_flash_api_rx_config.h

  (1) To prevent ROM access by interrupts during flash write and erase operations, the processor status word (PSW) interrupt priority level (IPL) field is changed to the value specified in the following macro definition. In this application note, the value 5 is used.

  ```
  Macro definition: #define F FLASH_API_RX_CFG_FLASH_READY_IPL     5
  ```

  (2) The following Simple Flash API settings are changed.

  ```
  Before change: //#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 change:  #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 to the file r_bsp_config.h

  (1) The Simple Flash API settings are changed. The settings of ICLK_HZ, PCLK_HZ, BCLK_HZ, and FCLK_HZ should match the settings of $(WORKSPDIR)\WorkSpace\HwResourceForUSB\src\rx_mcu.c, which is included in the USB Host Mass Storage Class Driver files.

  ```
  Before change: #define BSP_CFG_BCK_DIV (8)
  After change:  #define BSP_CFG_BCK_DIV (4)
  ```

- Changes to the file r_bsp.h

  (1) The files hwsetup.h, lowscr.h, vexttbl.h, and iodefine_rx63th.h are commented out to avoid duplication of program files included with the USB Host Mass Storage Class Driver. The file rskrx63t_144pin.h is not used, so it is also commented out.

  ```
  Before change: #include "mcu/rx63t/register_access/iodefine_rx63th.h"
                 #include "board/rskrx63t_144pin/rskrx63t_144pin.h"
                 #include "board/rskrx63t_144pin/hwsetup.h"
                 #include "board/rskrx63t_144pin/lowsrc.h"
                 #include "board/rskrx63t_144pin/vecttbl.h"
  After change:  //#include "mcu/rx63t/register_access/iodefine_rx63th.h"
                 //#include "board/rskrx63t_144pin/rskrx63t_144pin.h"
                 //#include "board/rskrx63t_144pin/hwsetup.h"
                 //#include "board/rskrx63t_144pin/lowsrc.h"
                 //#include "board/rskrx63t_144pin/vecttbl.h"
  ```

- Changes to the file r_flash_api_rx.c

  (1) The following changes have been to allow the sample program to access iodefine.h, which is included with the USB Host Mass Storage Class Driver.

  Before change:

  ```
  #if !defined(R_BSP_VERSION_MAJOR)
  #include "iodefine.h"
  #include "mcu_info.h"
  #endif
  ```

  After change:

  ```
  #if !defined(R_BSP_VERSION_MAJOR)
  #include "iodefine.h"
  #include "mcu_info.h"
  #else
  #include "iodefine.h"
  #endif
  ```

8.9 Changes to the USB Host Mass Storage Class Driver

The sample code uses the USB Host Mass Storage Class Driver program code. For specifications of the USB Device USB Host Mass Storage Class Driver, see the USB Device USB Host Mass Storage Class Driver and USB Device USB Basic Firmware application notes.

8.9.1 Changes

The following three files in the USB Host Mass Storage Class Driver are modified for used with this sample code.

- r_usb_hmsc_apl.c
- dbsct_hmsc.c
- resetprg.c
- Places changed in the file r_usb_hmsc_apl.c:
  Places indicated by #ifdef R_FLASH_USB.
- Places changed in the file dbsct_hmsc.c:
  Places indicated by the comment "// Flash table".
- Places changed in the file resetprg.c:
  1. An include file is added.
     Added: #include "r_Flash_main.h"
  2. The mode entry function is called in the function PowerON_Reset_PC().
     Added: R_Fl_Mode_Entry();

8.9.2 Added Files

See section 5.7, File Structure, for the files added to the USB Host Mass Storage Class Driver.

8.9.3 Added Sections

Table 8.2 lists the sections added to the USB Host Mass Storage Class Driver.

Table 8.2 Added Sections

Section	Description
R_flash_api_sec	Section for variables in the flash programming code that operates in RAM
D_flash_api_sec	Section for initialization area of flash programming code that operates in RAM
RPFRAM	Section for the flash programming code that operates in RAM
TRGT_DMMY_FIXEDVECT	Section for the download code fixed vector

8.9.4 Include File Directories

WorkSpace\FLASH, WorkSpace\FLASH\src, WorkSpace\FLASH\r_bsp, WorkSpace\FLASH\r_bsp\mcu\rx63t, and WorkSpace\FLASH\r_bsp\board\rskrx63t_144pin have been added to the include file directories.

8.9.5 Macro Definitions (Compiler options)

R_FLASH_USB is added as a compiler option macro definition.

8.9.6 Linker Settings

The following linker settings that map from ROM to RAM are added.

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

9. Sample Programs

The sample program can be downloaded from the Renesas Electronics Web site.

10. Reference Documents

User’s Manual: Hardware

RX63T Group User’s Manual: Hardware Rev.2.00
(The latest version can be downloaded from the Renesas Electronics Web site.)

Technical Updates/Technical News

(The latest information can be downloaded from the Renesas Electronics Web site.)

User’s Manual: Development Environment

RX Family C/C++ Compiler Package V.1.01 User’s Manual Rev.1.00 (includes V.1.02 supplementary documents)
(The latest version can be downloaded from the Renesas Electronics Web site.)

Application Notes

Renesas USB Device USB Host Mass Storage Class Driver Rev.2.00
Renesas USB Device USB Basic Firmware Rev.2.00
M3S-TFAT-Tiny: Fat File System Software Rev.1.00
RX600 & R200 Series Simple Flash API for RX Rev.2.40
(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/contact/

Revision History

		Description	
Rev.	Date	Page	Summary
1.00	May. 14, 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 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 type 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 301, Tower A, Central Towers, 555 LanGao Rd., Putuo District, Shanghai, China
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: +852-2886-9318, 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.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 3.0