RX63N Group, RX631 Group

USB Host Flash Boot Loader

R01AN1241EJ0100
Rev.1.00
Apr. 05, 2013

Abstract

This application note describes a USB host flash boot loader that operates, in single-chip mode, the RX63N and RX631 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 Series Simple Flash API for RX600 Rev.2.20 (R01AN0544EU0220)
- 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.

- An 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 an 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 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

RX63N Group, RX631 Group

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 RX63N RSK board.

If a reset is cleared with switch SW3 on the RX63N RSK not held down, the 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 RX63N 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

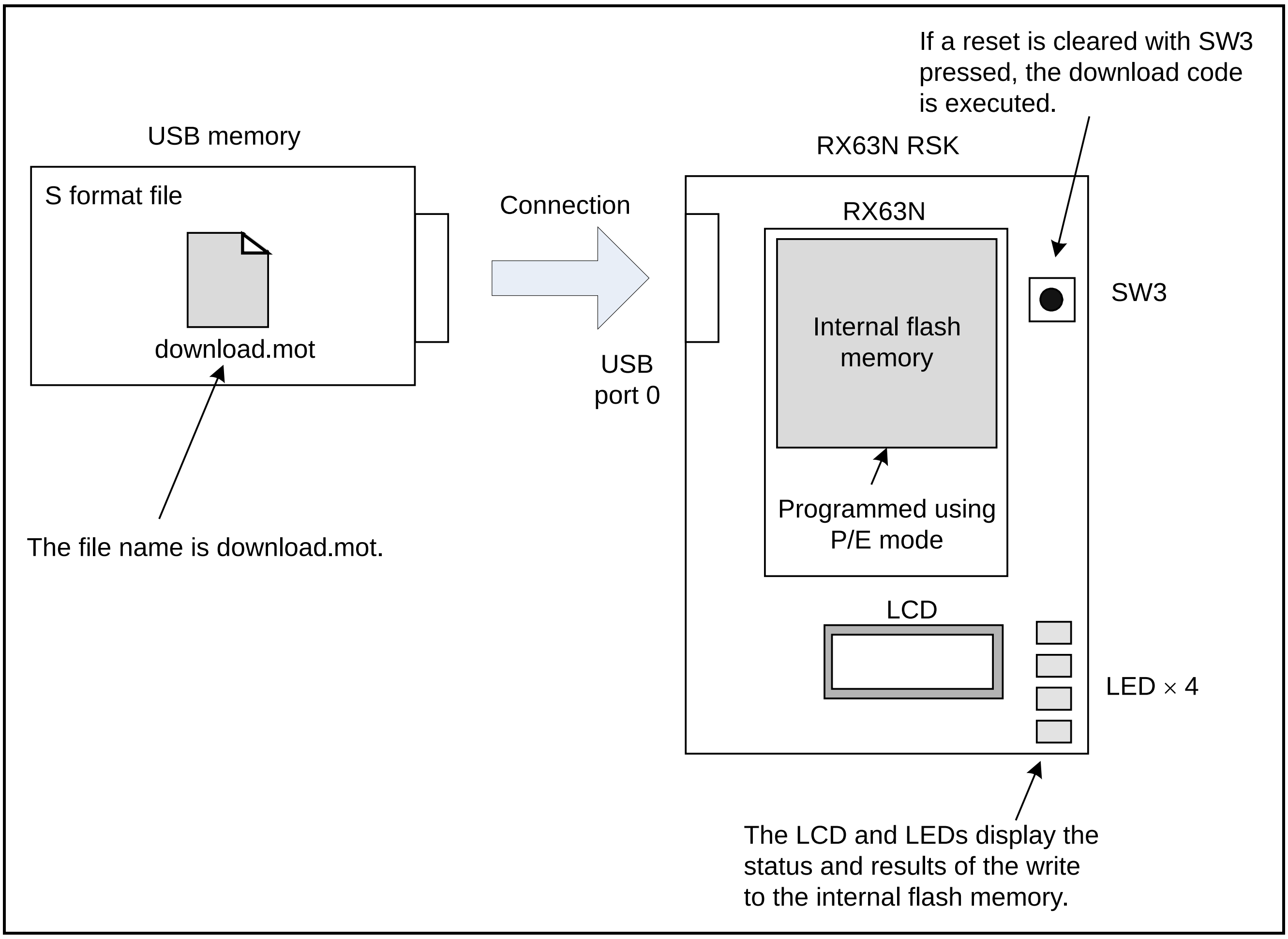

Figure 1.1 Usage Example

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	R5F563NEDDFC (RX63N 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	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 1.2.1.0
	'-cpu=rx600 include="$(WORKSPDIR)\WorkSpace\USBSTDFW\include" -include="$(WORKSPDIR)\WorkSpace\SmplMain\APL" -include="$(WORKSPDIR)\WorkSpace\HwResourceForUSB\USBHW" -include="$(WORKSPDIR)\WorkSpace\HwResourceForUSB\USBHW\DEF" -include="$(WORKSPDIR)\WorkSpace\HwResourceForUSB\USBHW\REG" -include="$(WORKSPDIR)\WorkSpace\ANSI" -include="$(WORKSPDIR)\WorkSpace\HwResourceForUSB\USRCFG" -include="$(WORKSPDIR)\WorkSpace\MSCFW\include" -include="$(WORKSPDIR)\WorkSpace\MSCFW\TFAT\lib_src" -include="$(WORKSPDIR)\WorkSpace\FLASH" -define=USB_FW_PP=USB_FW_NONOS_PP,USB_TFAT_USE_PP=1, R_FLASH_USB -output=obj="$(CONFIGDIR)\$(FILELEAF).obj" -debug -nostuff -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 RX63N

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 Series Simple Flash API for RX600 Rev.2.20 (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.
P16/USB0_VBUSEN-H	Output	VBUS (5 V) supply enable signal for port 0 external power supply chip
P14/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.
P07	Input	Sample code mode selection pin
P03	Output	LED connection pin
P05	Output	LED connection pin
P10	Output	LED connection pin
P11	Output	LED connection pin
PJ5	Output	LCD module control pin
PF5	Output	LCD module control pin
P84	Output	LCD module control pin
P85	Output	LCD module control pin
P86	Output	LCD module control pin
P87	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 P07). If this switch is not being pressed (if pin P07 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 P07 is low), it runs the download code.

Figure 5.1 shows the operation after a reset.

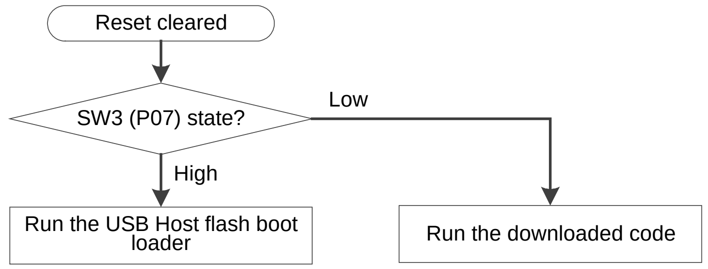

Figure 5.1 Operation After a Reset Is Cleared

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, FFFD C000h to FFFF FFFFh, is not overwritten.

Figure 5.2 shows the memory allocation.

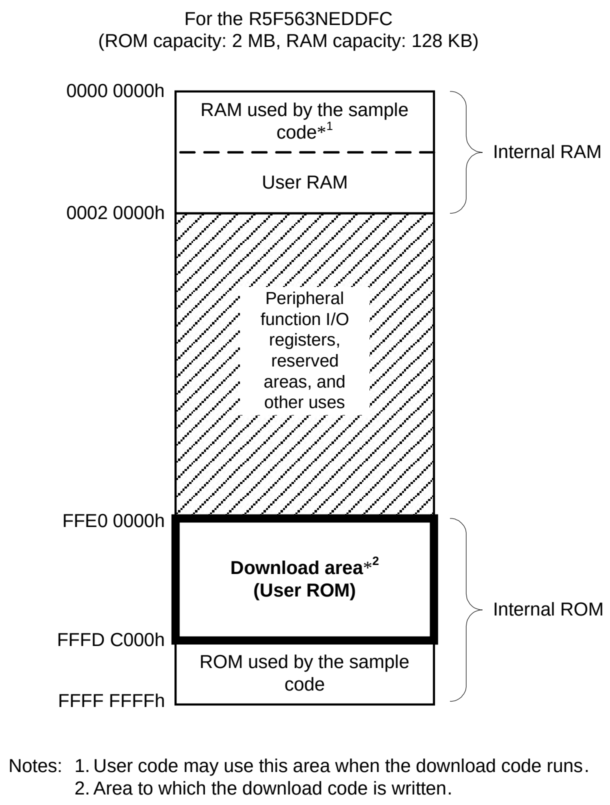

Figure 5.2 Memory Allocation

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 an S format file with the filename download.mot.
(4) If an 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 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 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.

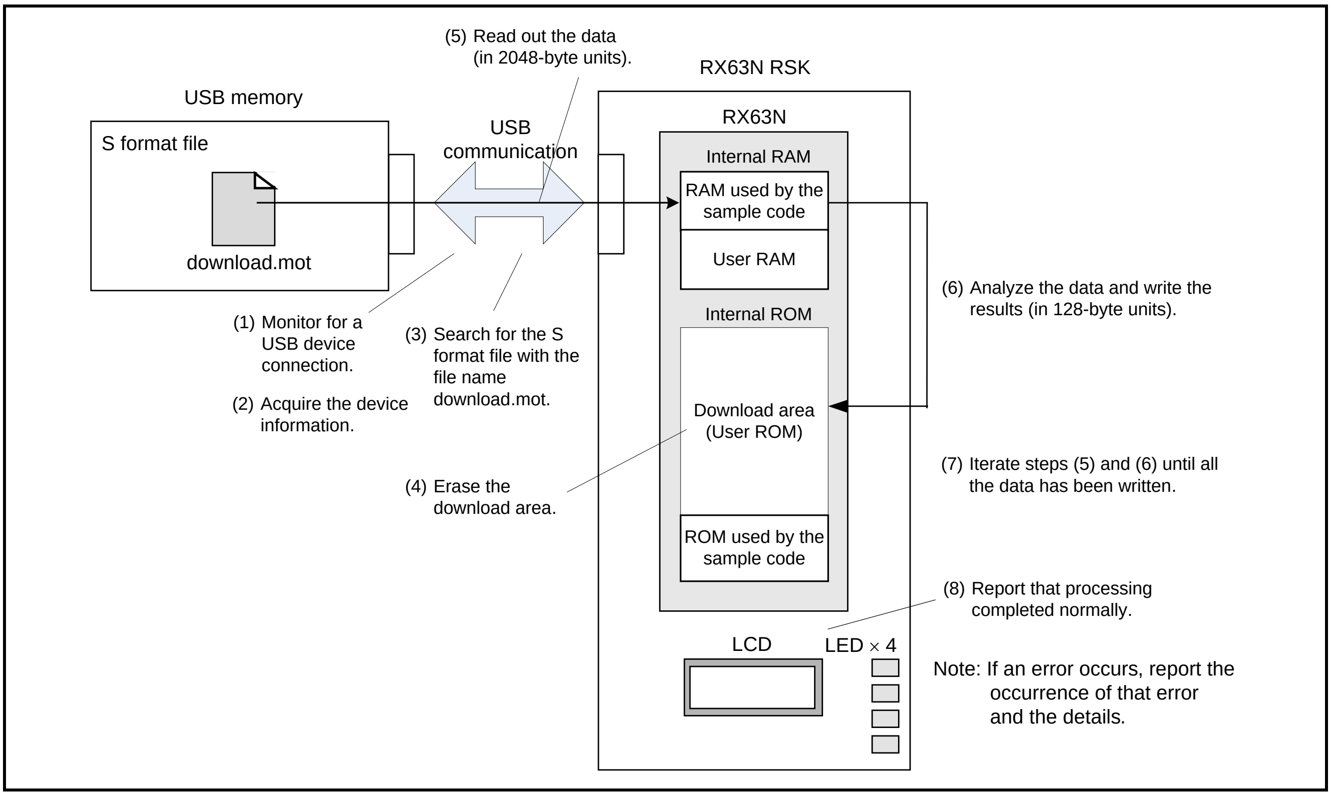

Figure 5.3 Programming the Download Area

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 BFFCh. That is, the reset vector for the download code is FFFD BFFCh. Therefore the download code must store its start address at location FFFD BFFCh.

Figure 5.4 shows the reset vector for the download code.

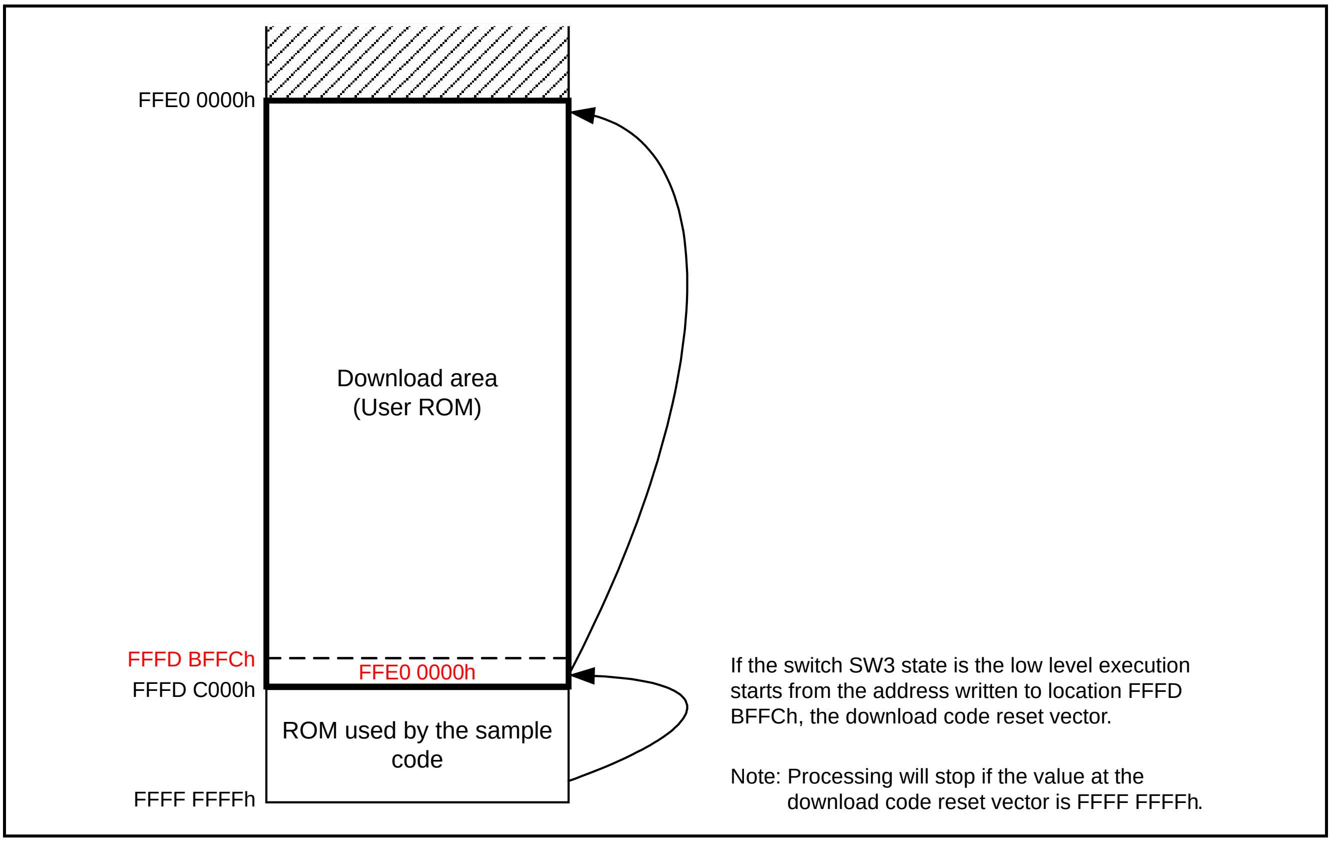

Figure 5.4 Download Code Reset Vector

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 RX Family RX600 Simple Flash API 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.

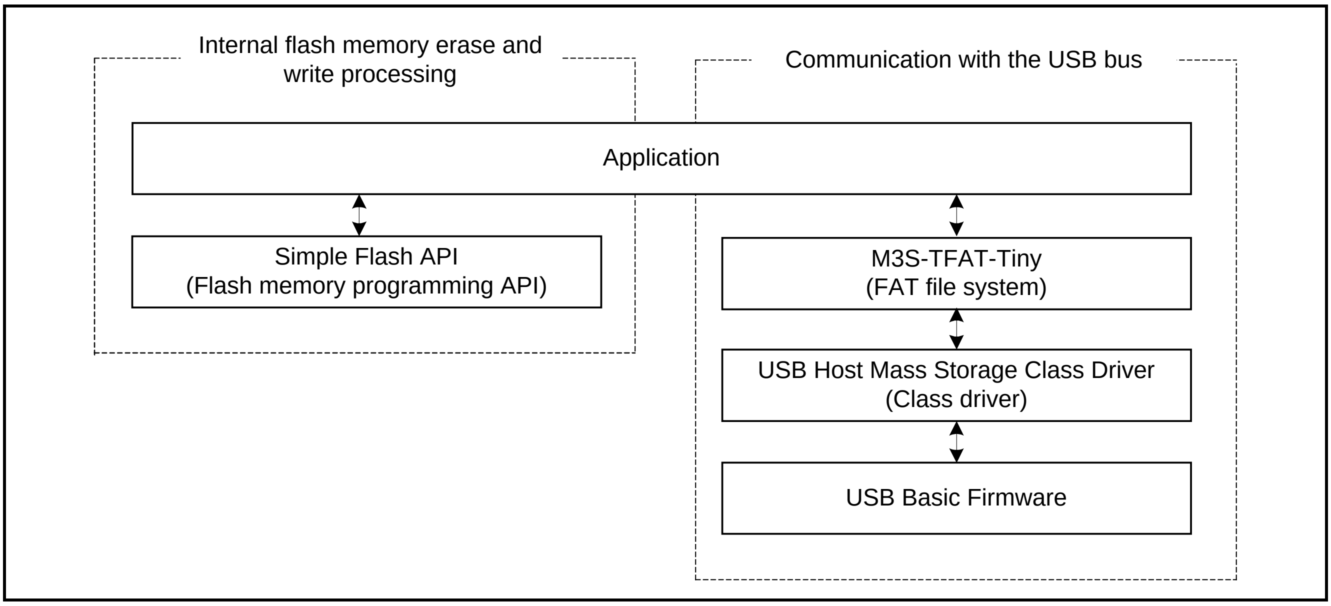

Figure 5.5 Software Structure of the Sample Code

Table 5.1 Software Overview

Module	Overview
Application	The application uses FAT library functions to read an 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 RX600	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 S format data is copied to an S format buffer (this is ASCII data).
(3) At the same time as analyzing the S format data header section, the ASCII data is converted to binary and stored in an S format buffer (for binary data).
See section 7, S Format, for the S format data analysis specifications used in this application note.
(4) The data is stored in a write buffer.
In the RX63N and RX631 group microcontrollers, 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.

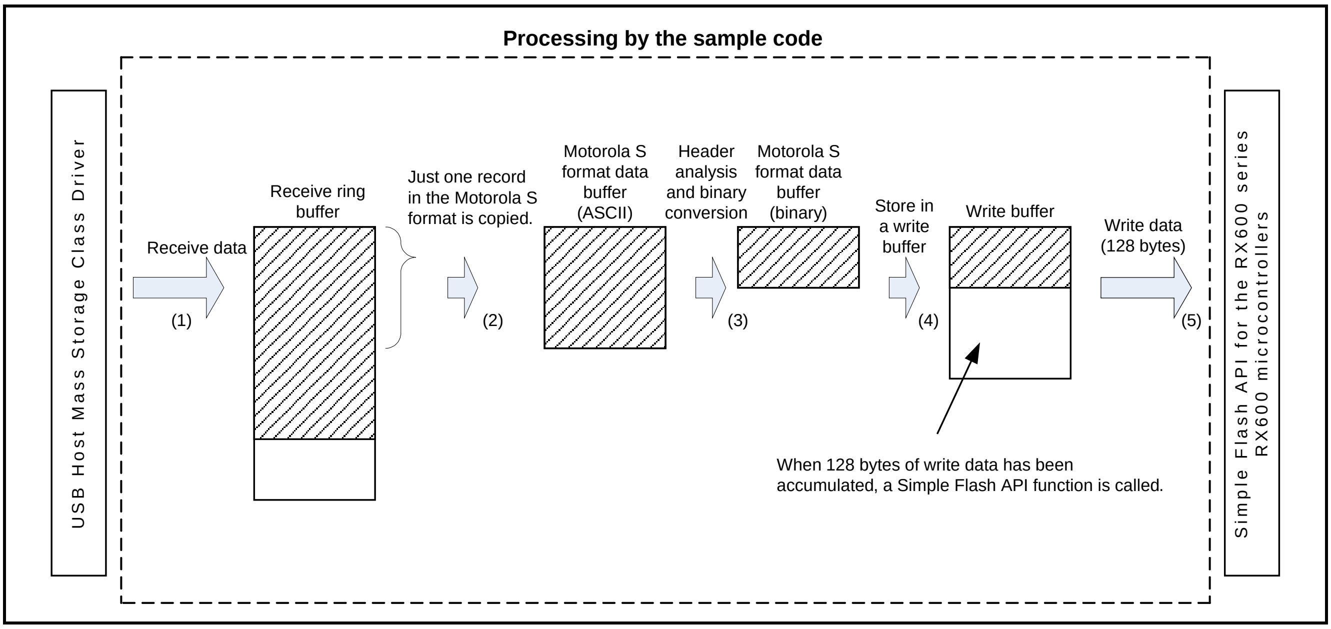

Figure 5.6 Data Flow During Write

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

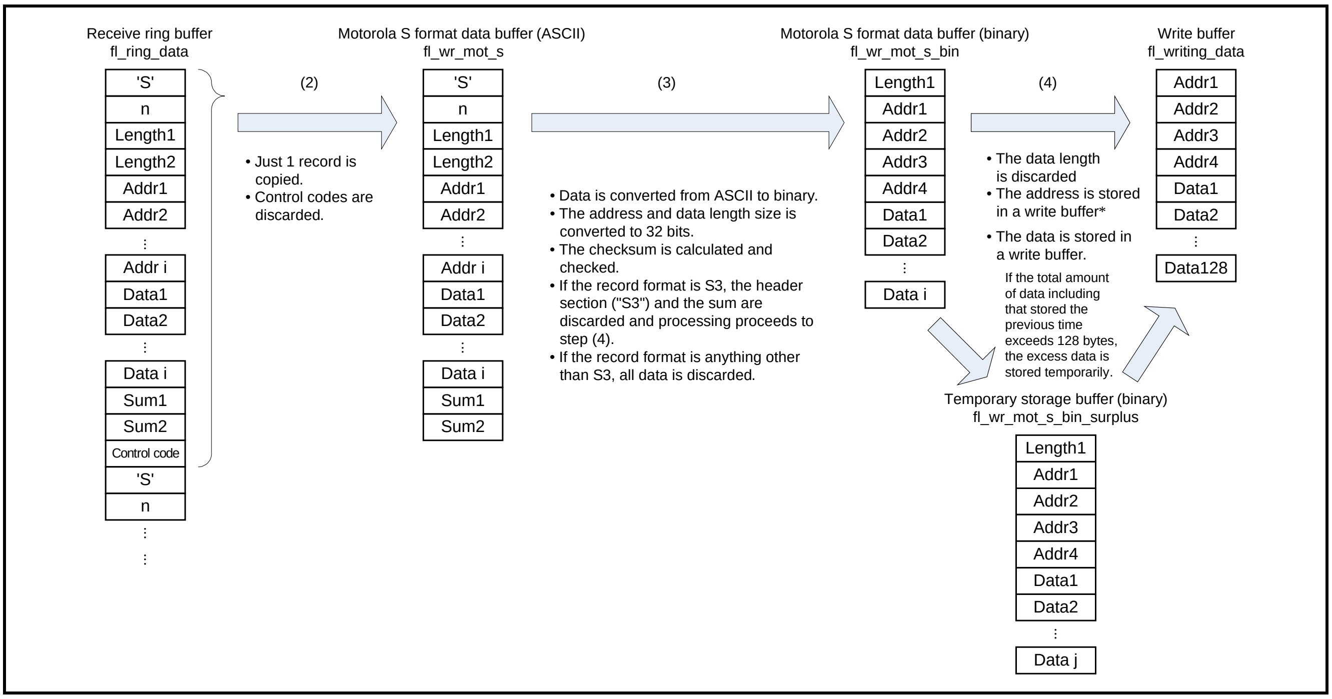

Figure 5.7 Data Structures Used for Writing

Note: In the RX63N and RX631 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 RX63N-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					Description
LED3	LED2	LED1	LED0	Order	
×	×	×	×		The LEDs display a binary counter during the download processing. The LED display is updated every 128 bytes (the unit of writes to the RX63N/RX63N 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]
PROGRAM 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, S Format. (Checksum error)
ERROR!! MOTS	See section 7, 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, 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 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	125,110 bytes	Since the sample code is allocated to locations FFFD C000h to FFFF FFFFh, the amount of ROM that can be written is the total ROM capacity minus 147,456 bytes.
RAM	38,200 bytes	The user code can use this area when it runs.
Maximum user stack usage	584 bytes	
Maximum interrupt stack usage	72 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 generated by the integrated development environment are not included in this table.

Table 5.5 Files Used in the Sample Code

File Name	Description	Remarks
r_flash_api_rx600.c	The RX600 Series RX600 Simple Flash API program	For details, see the RX600 Series RX600 Simple Flash API application note.
r_flash_api_rx600.h	External reference include header for the RX600 Series RX600 Simple Flash API program.	
r_flash_api_rx600_private.h	External reference include header for the RX600 Series RX600 Simple Flash API program.	
r_flash_api_rx600_config.h	Parameter settings include header for the RX600 Series RX600 Simple Flash API program.	
mcu_info.h	Parameter settings include header for the RX600 Series RX600 Simple Flash API program.	
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. HOCO oscillation is ignored after a reset.
MDES*1	FFFF FF83h to FFFF FF80h	 FFFF FFFFh FFFF FFF8h	(Single-chip mode) Little endian 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 lists the constants used in the sample code.

Table 5.7 Constants Used in the Sample Code

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	FFFDBFFCh	Reset vector address of download code
FL_START_BLOCK_NUM	15	First block in the download area
FL_END_BLOCK_NUM	69	Last block in the download area
FL_START_WRITE_ADDRESS	FFE00000h	First address in the download area
FL_END_WRITE_ADDRESS	FFFDBFFFh	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 S type format
FL_MOTS_LNG_SIZE	1	Data length buffer size of S type format
FL_MOTS_ADDR_MIN_SIZE	2	Min. address buffer size of S type format
FL_MOTS0_ADDR_SIZE	2	S0 format data address buffer size
FL_MOTS3_ADDR_SIZE	4	S3 format data address buffer size
FL_MOTS7_ADDR_SIZE	4	S7 format data address buffer size
FL_MOTS8_ADDR_SIZE	3	S8 format data address buffer size
FL_MOTS9_ADDR_SIZE	2	S9 format data address buffer size
FL_MOTS_SUM_SIZE	1	S format data checksum buffer size
FL_USB_RCV_BLANK_SIZE	FL_RINGBUFF_SIZE / 2	Ring buffer capacity
FL_PORT_MDE	PORT0.PIDR.BIT.B7	PORT register for the port connected to SW3
FL_DDR_MDE	PORT0.PDR.BIT.B7	DDR 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_rx600_private.h, and it is referenced when this file is included.

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

5.10 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

5.11 Functions

Table 5.8 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 5.8 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 S format data
R_Fl_Prg_ProcessForMotS_data	Header analysis, binary conversion, and write of an S format record
R_Fl_Prg_MotS_AsciiToBinary	Convert 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 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

5.12 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, iodefine.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 S format file in the USB memory. • Calls functions that analyze the 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 an 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 S format record. • When a single 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 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 S format data one byte at a time. • Discards all data until the first 'S' (ASCII data) is acquired.
Arguments	First argument: mot_data : S format data
Return Value	• If a single S format data item (from the 'S' to the checksum) was stored: FLASH_API_SAMPLE_OK • If a single S format data item was not stored: FLASH_API_SAMPLE_NG
Notes	• This function is used by passing 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 an S format record
Header	None
Declaration	static void R_Fl_Prg_ProcessForMotS_data(void)
Description	• Analyses the 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 S format is received.
Arguments	None
Return Value	None

R_Fl_Prg_MotS_AsciiToBinary

Outline	Convert 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 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 S format is received. • Calls an error handler if a checksum error occurs.
Arguments	First argument: *tmp_mot_s : Pointer to 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 RX63N/RX631 Group user MAT.
Arguments	None
Return Value	• If creation of 128 bytes (the unit of writes to the RX63N/RX631 Group user MAT) of write data completed: FLASH_API_SAMPLE_OK • If creation of 128 bytes (the unit of writes to the RX63N/RX631 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 S format data
Header	None
Declaration	static void R_Fl_Prg_ClearMotSVariables(void)
Description	Clears the variables related to the 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: *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: *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.

5.13 Flowcharts

5.13.1 Main USB Processing

Figure 5.9 shows the flowchart for main USB processing.

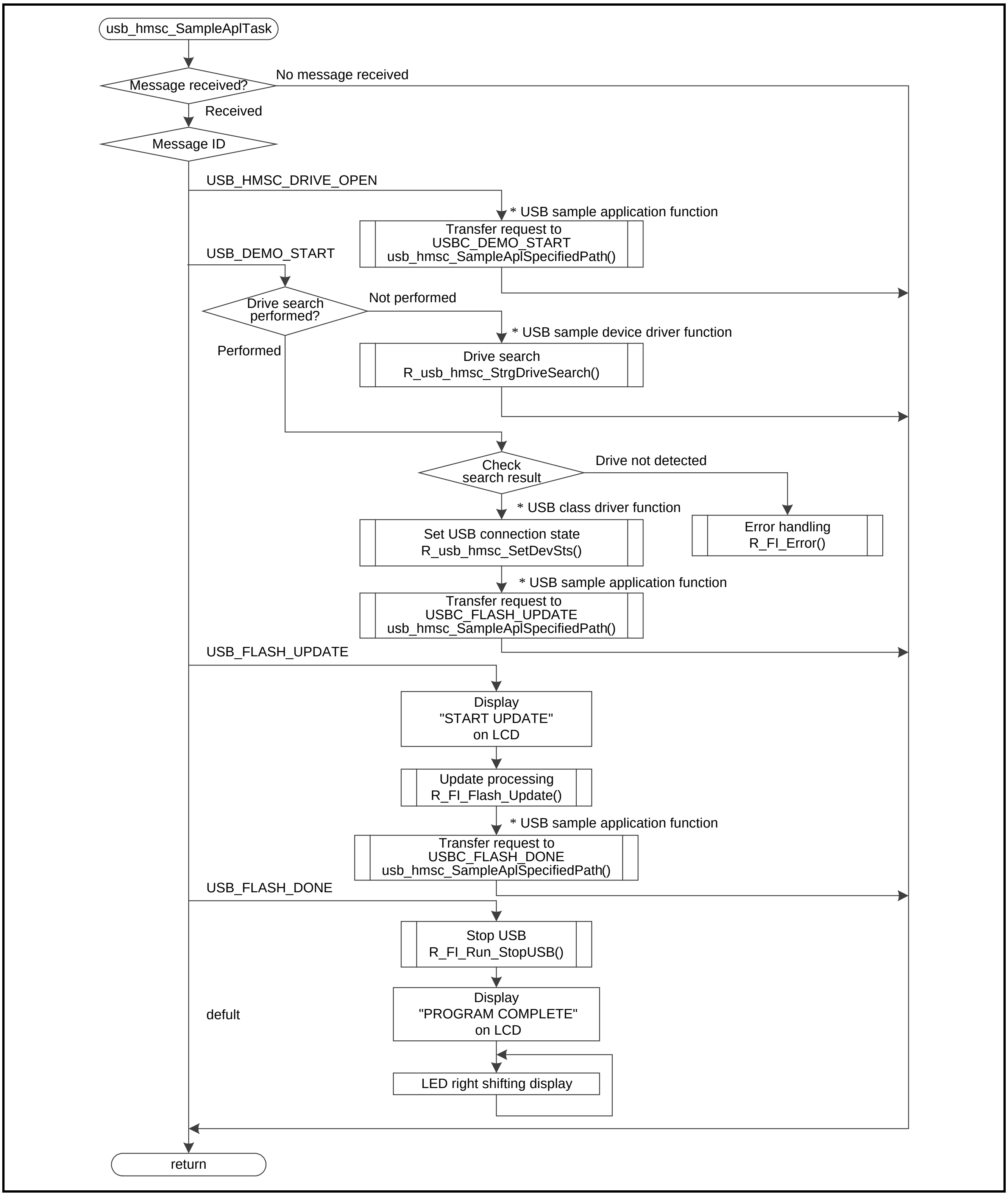

Figure 5.9 Main USB Processing

RENESAS

5.13.2 Mode Entry

Figure 5.10 shows the flowchart for mode entry.

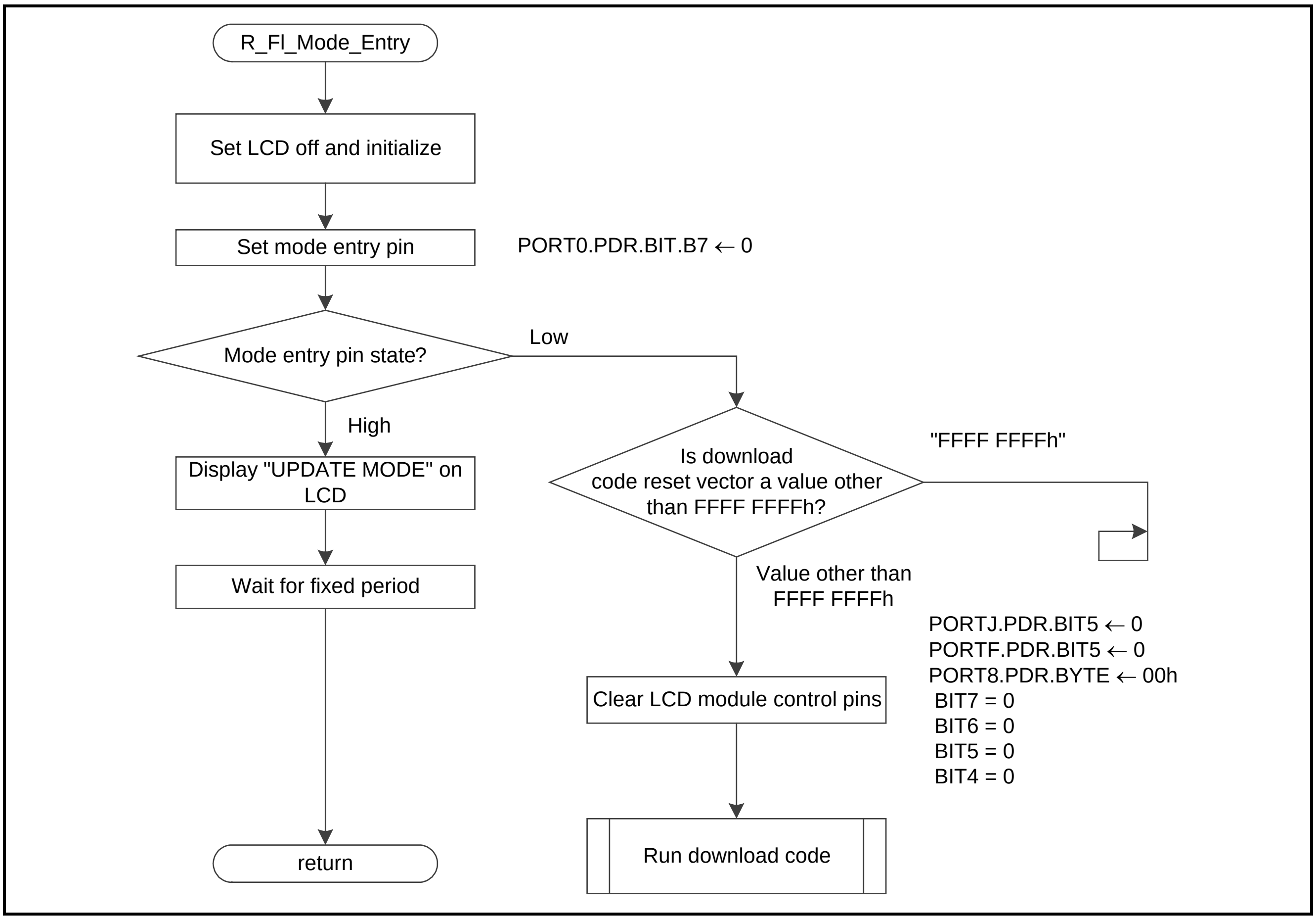

Figure 5.10 Mode Entry

5.13.3 Main Write Processing

Figure 5.11 shows the flowchart for main write processing.

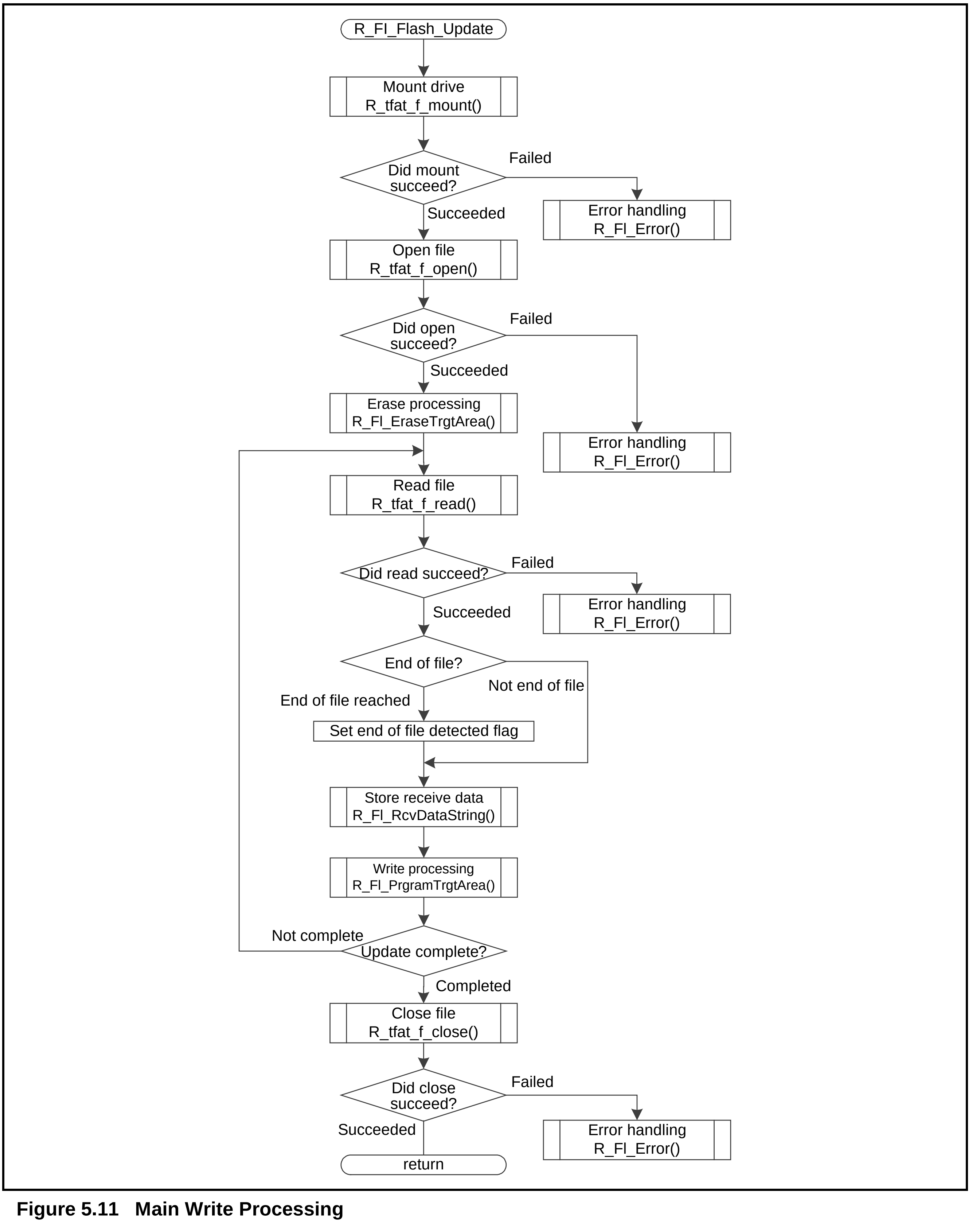

Figure 5.11 Main Write Processing

5.13.4 Erase Processing

Figure 5.12 shows the flowchart for erase processing.

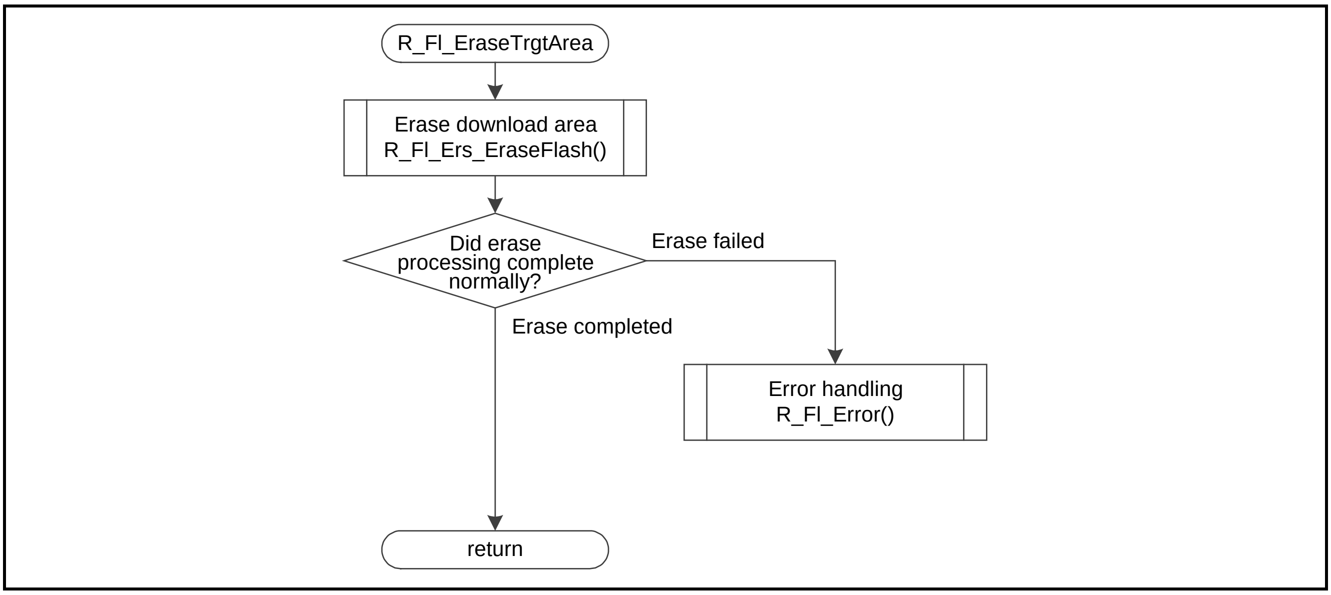

Figure 5.12 Erase Processing

5.13.5 Erase Download Area

Figure 5.13 shows the flowchart for erase download area.

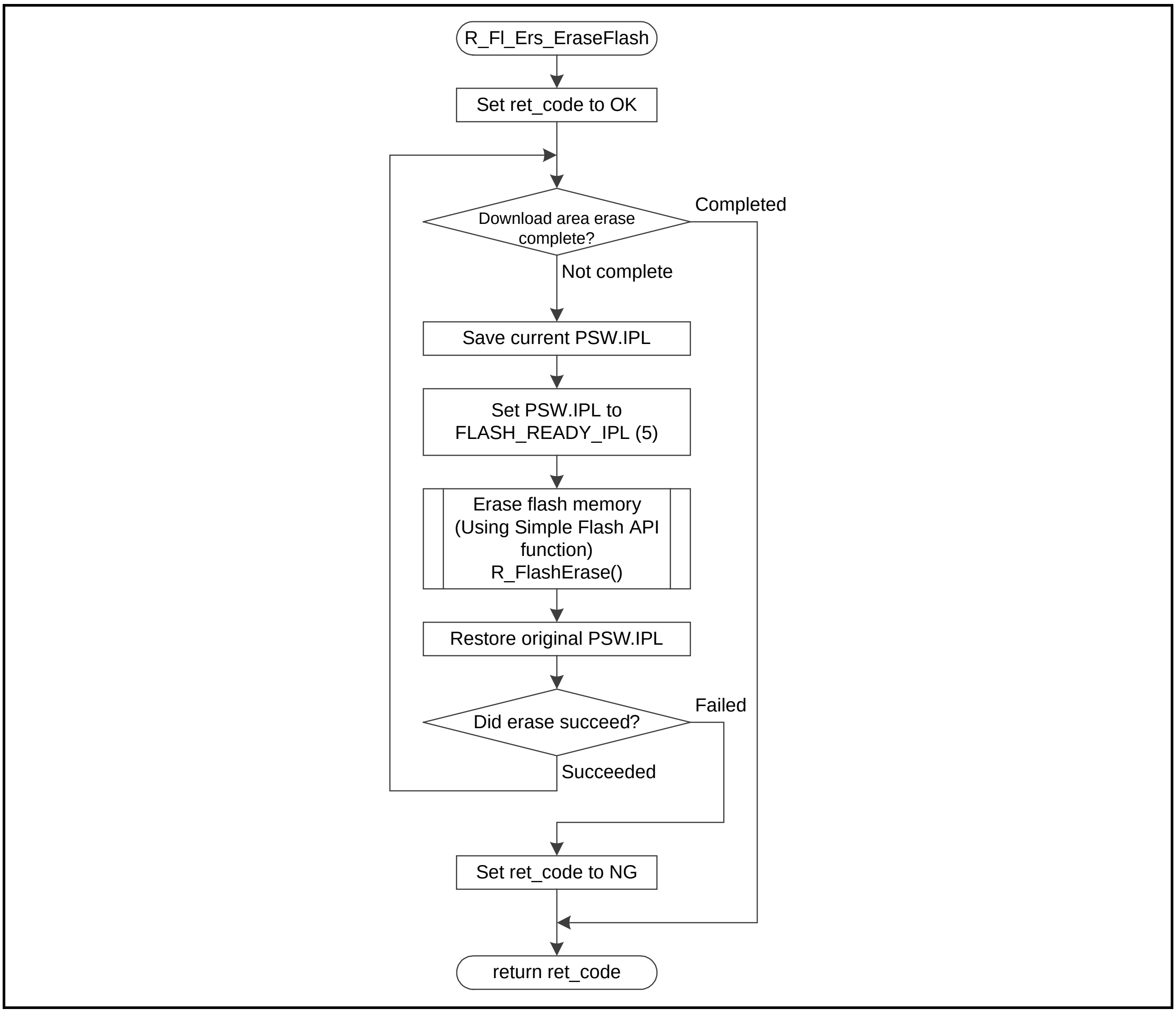

Figure 5.13 Erase Download Area

5.13.6 Write Processing

Figure 5.14 shows the flowchart for write processing.

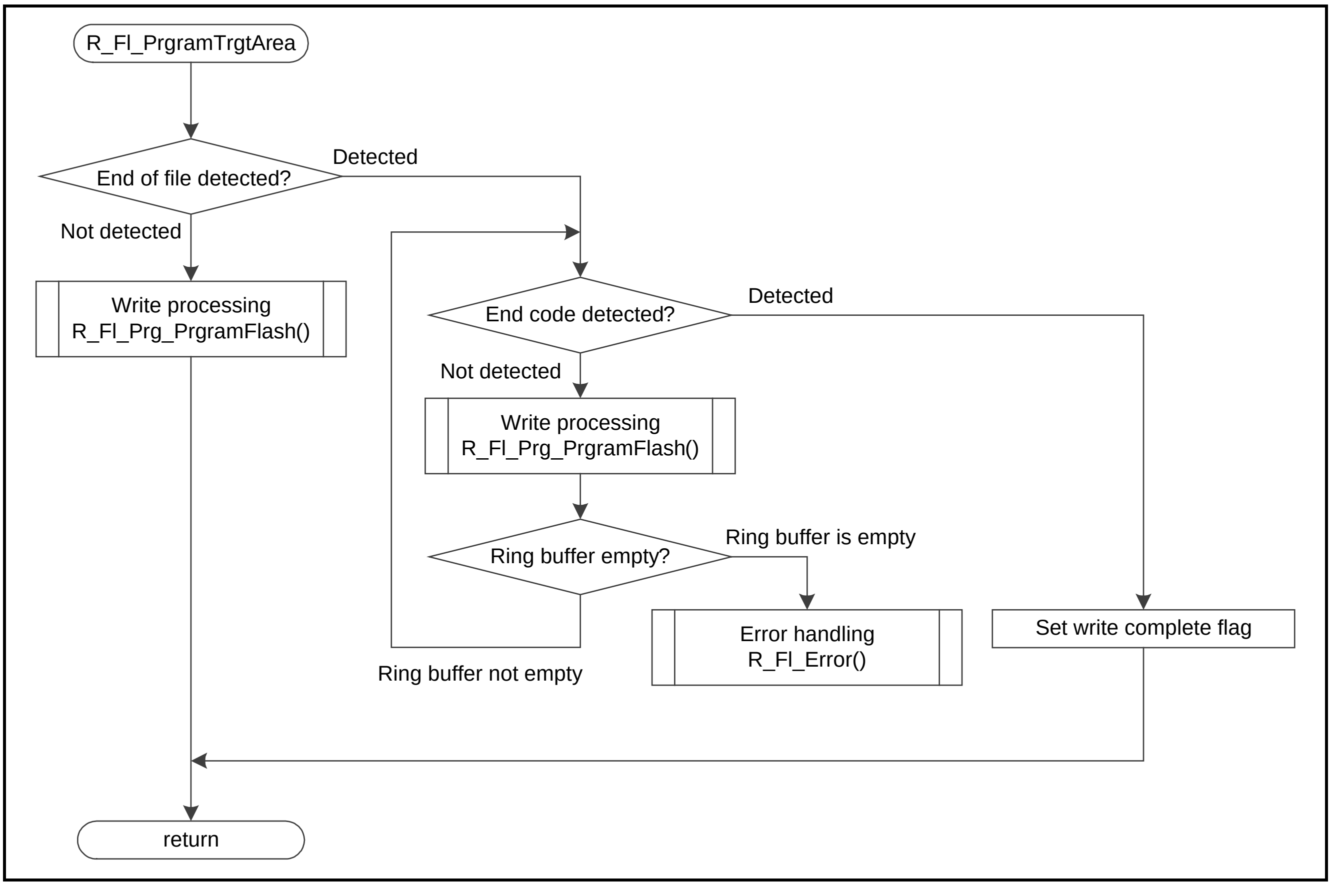

Figure 5.14 Write Processing

5.13.7 Download Area Write Operation

Figure 5.15 shows the flowchart for download area write operation.

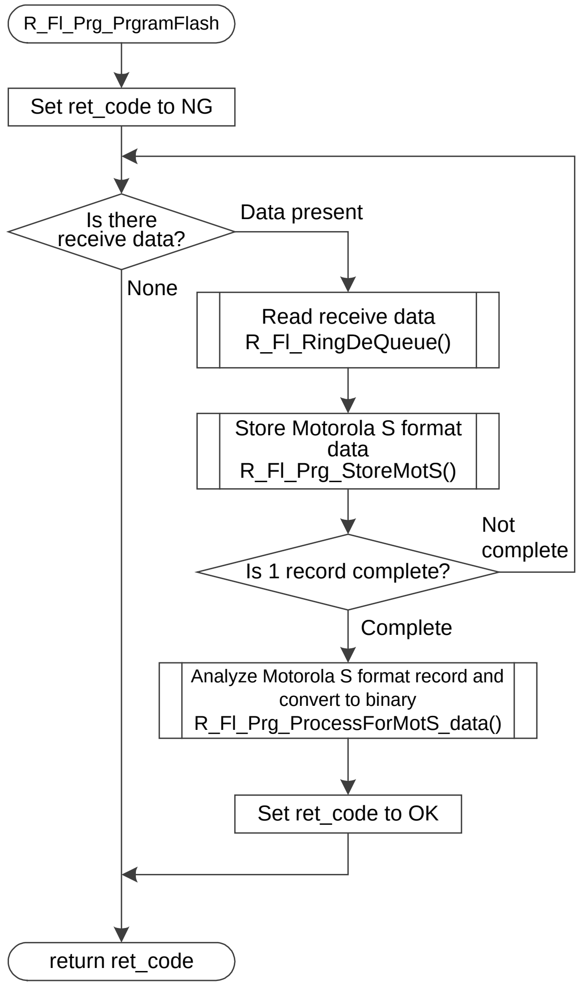

Figure 5.15 Download Area Write Operation

5.13.8 S Format Header Analysis, Conversion to Binary, and Write Operations

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

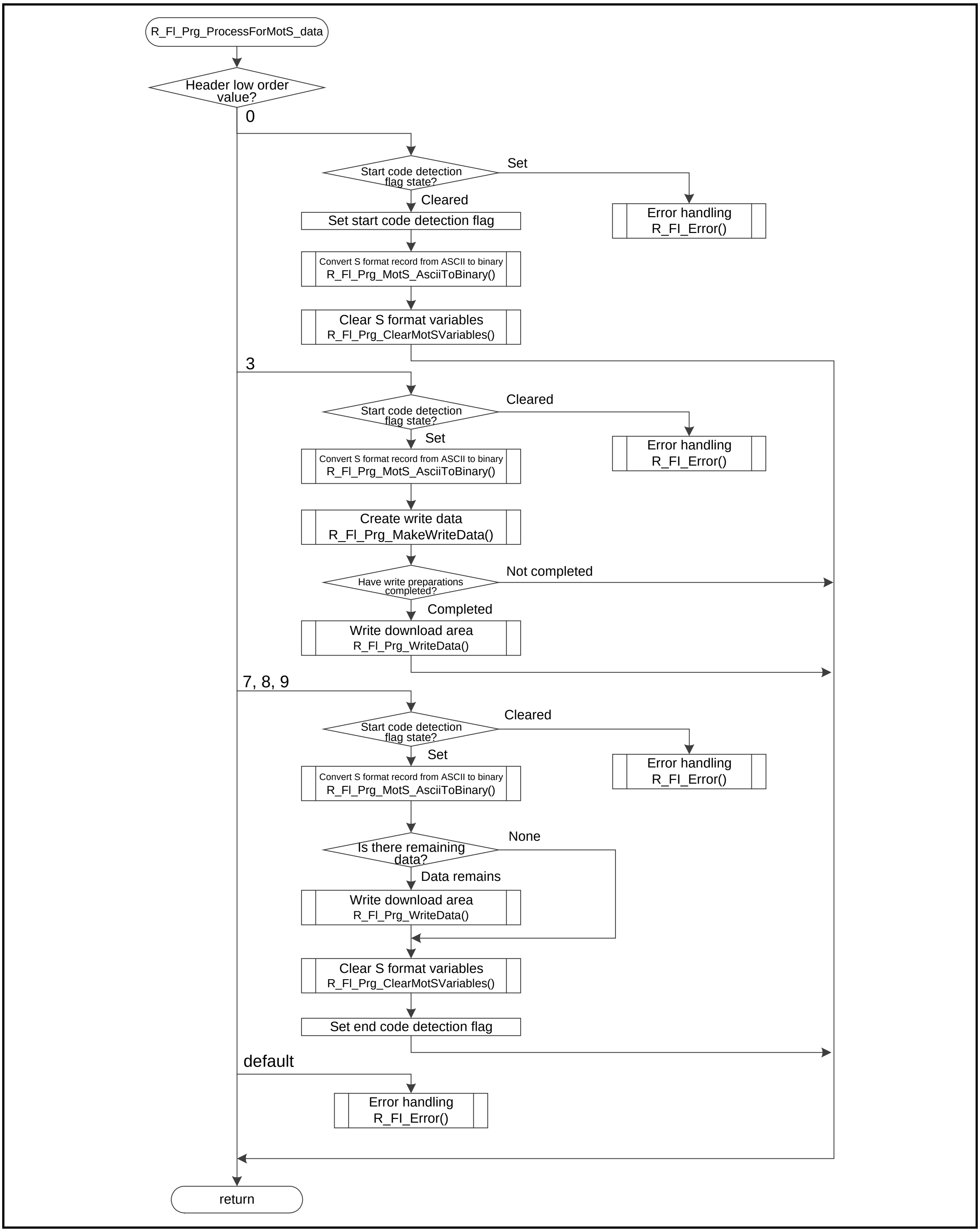

Figure 5.16 Format Header Analysis, Conversion to Binary, and Write Operations

5.13.9 S Format Data Store Operation

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

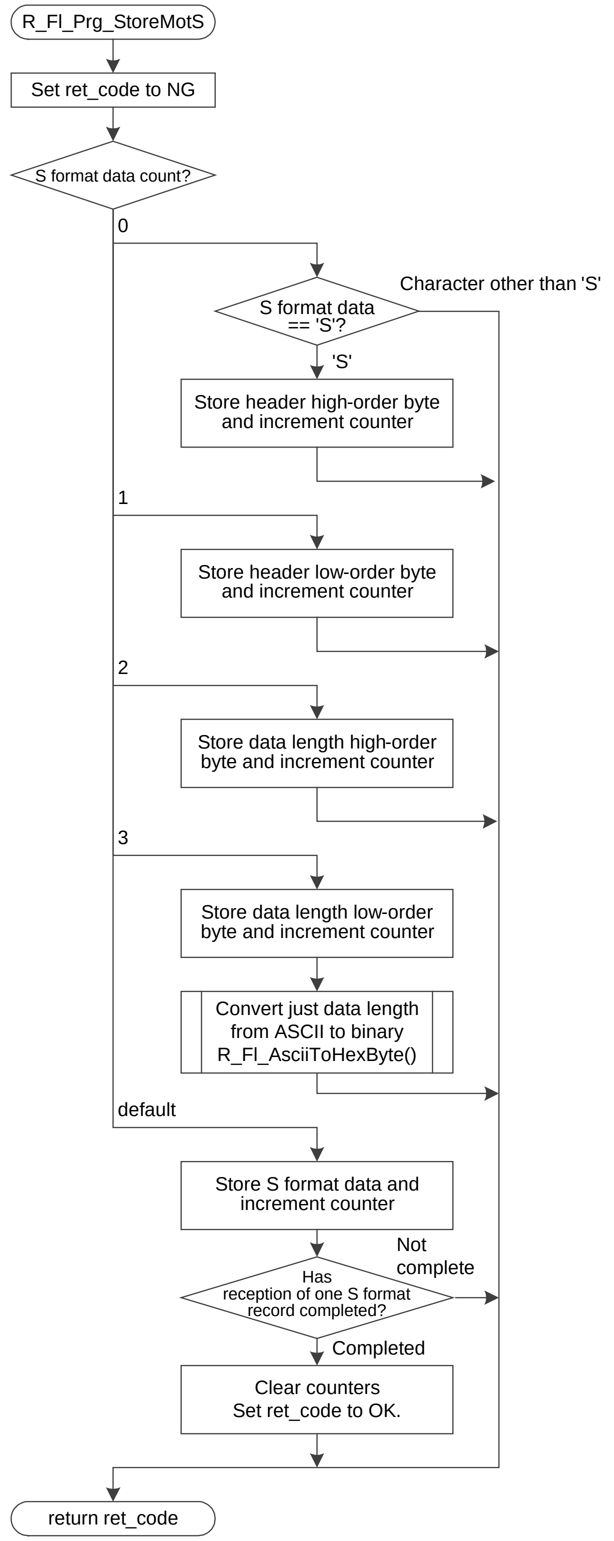

Figure 5.17 S Format Data Store Operation

5.13.10 S Format Data ASCII to Binary Conversion

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

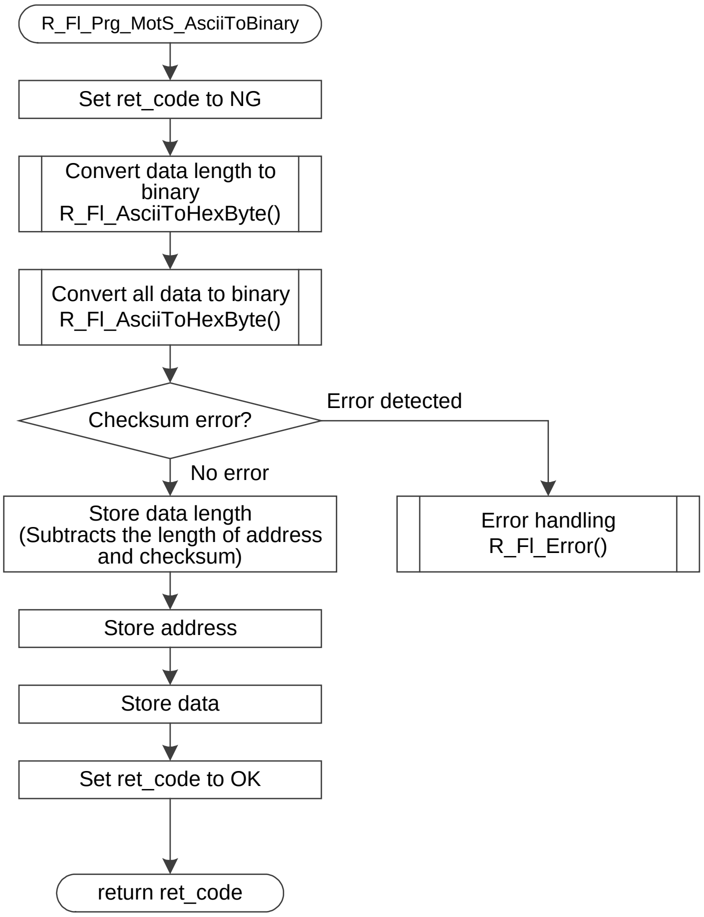

Figure 5.18 S Format Data ASCII to Binary Conversion

5.13.11 Download Area Write Data Creation

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

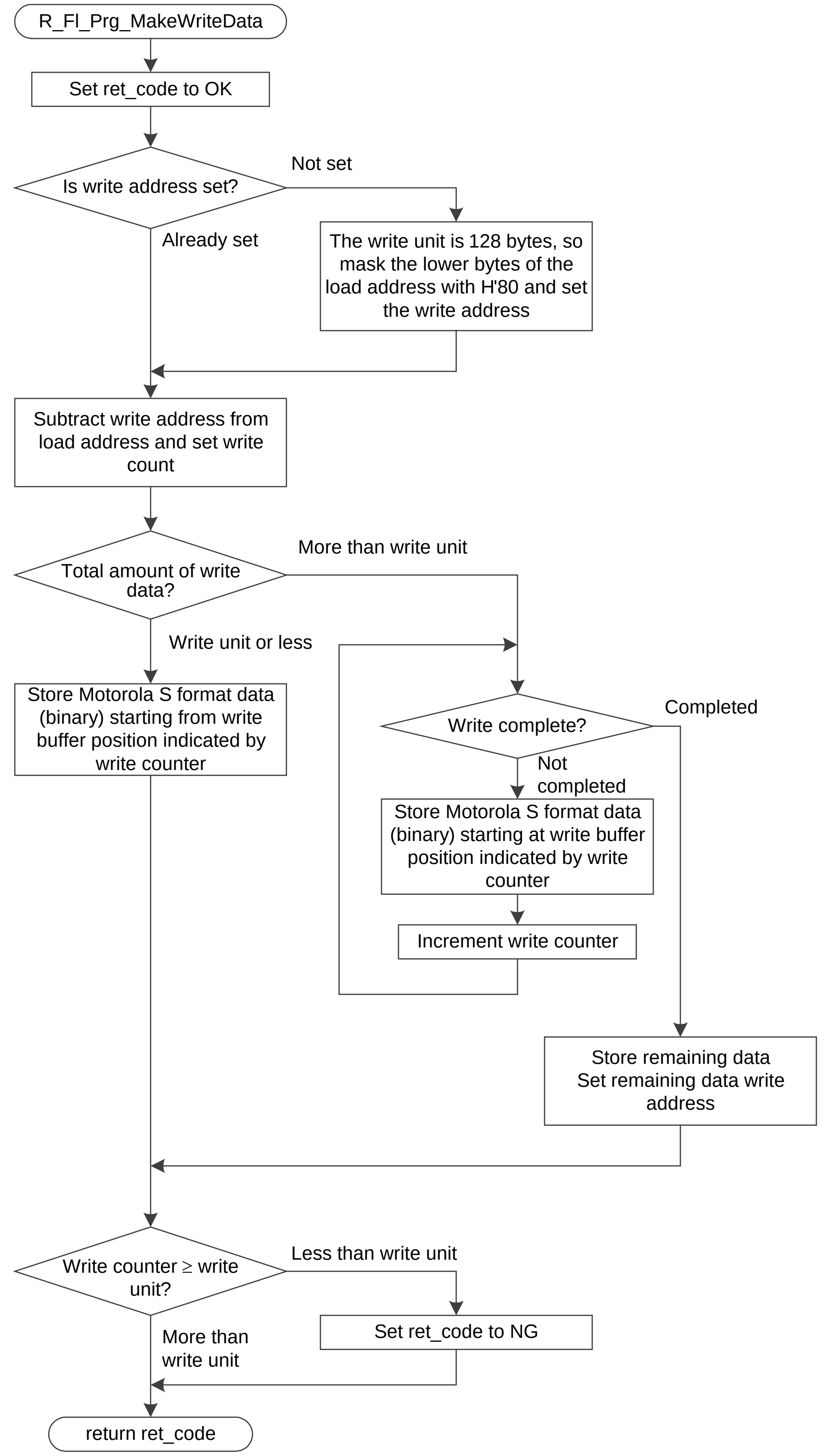

Figure 5.19 Download Area Write Data Creation

5.13.12 Download Area Write

Figure 5.20 shows the flowchart for download area write.

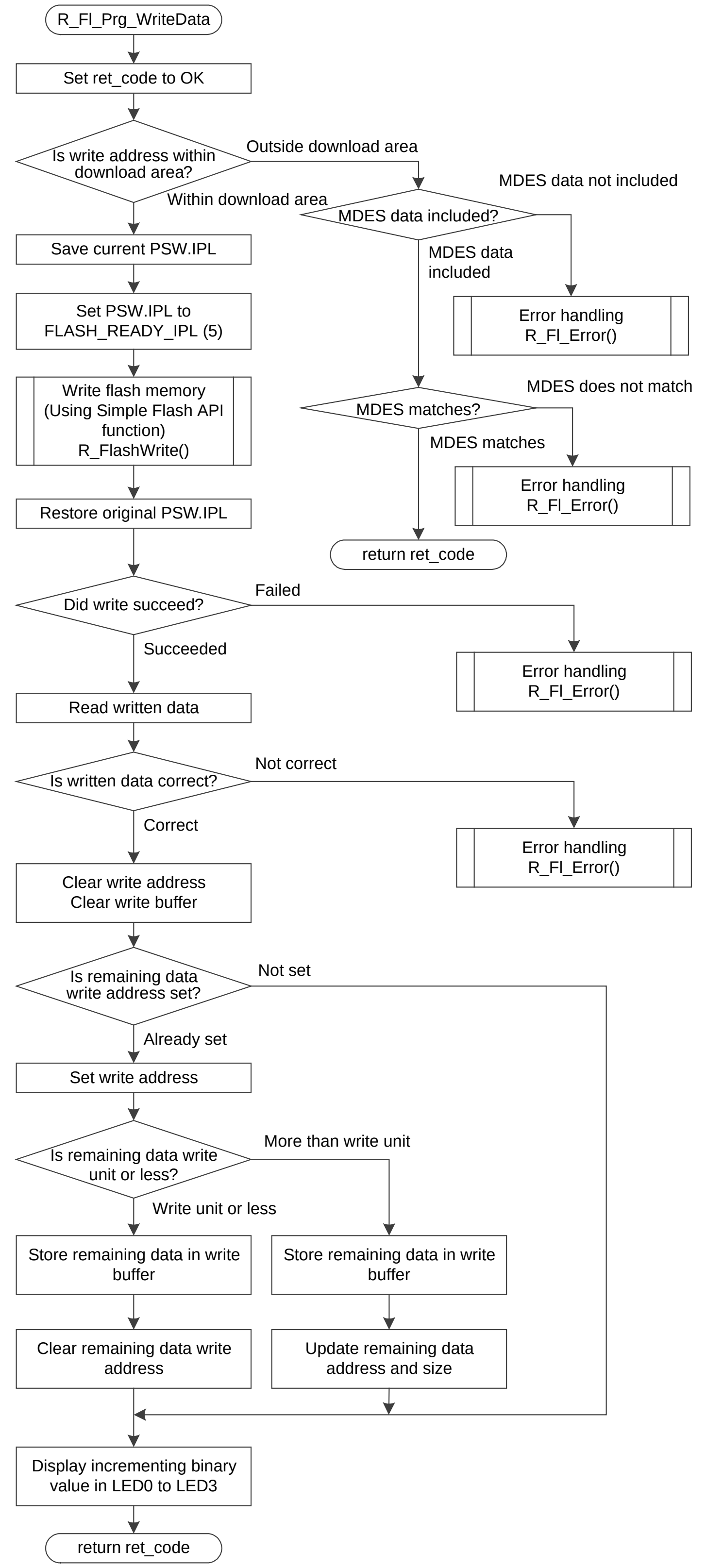

Figure 5.20 Download Area Write

5.13.13 Clear S Format Data Related Variables

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

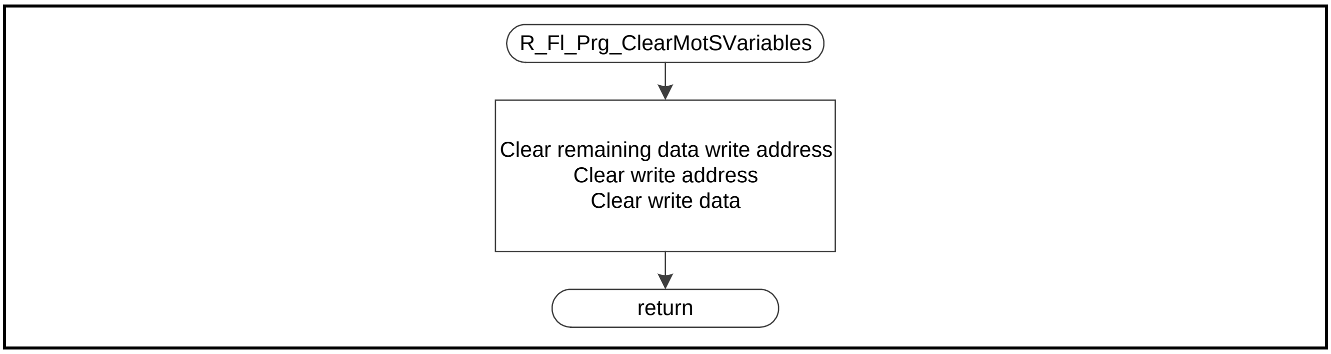

Figure 5.21 Clear S Format Data Related Variables

5.13.14 Store USB Receive Data

Figure 5.22 shows the flowchart for store USB receive data.

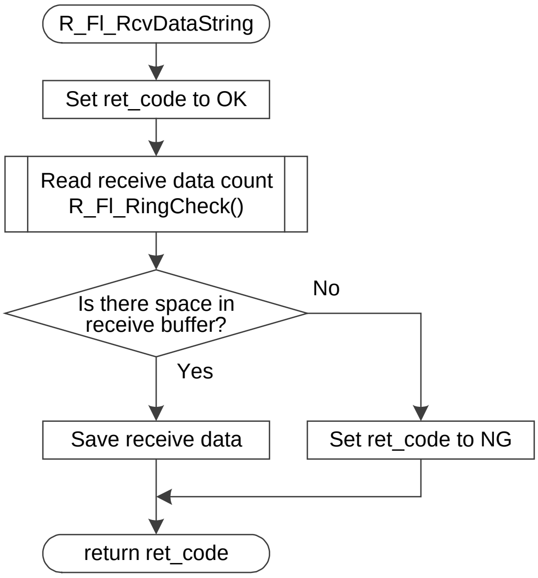

Figure 5.22 Store USB Receive Data

5.13.15 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.

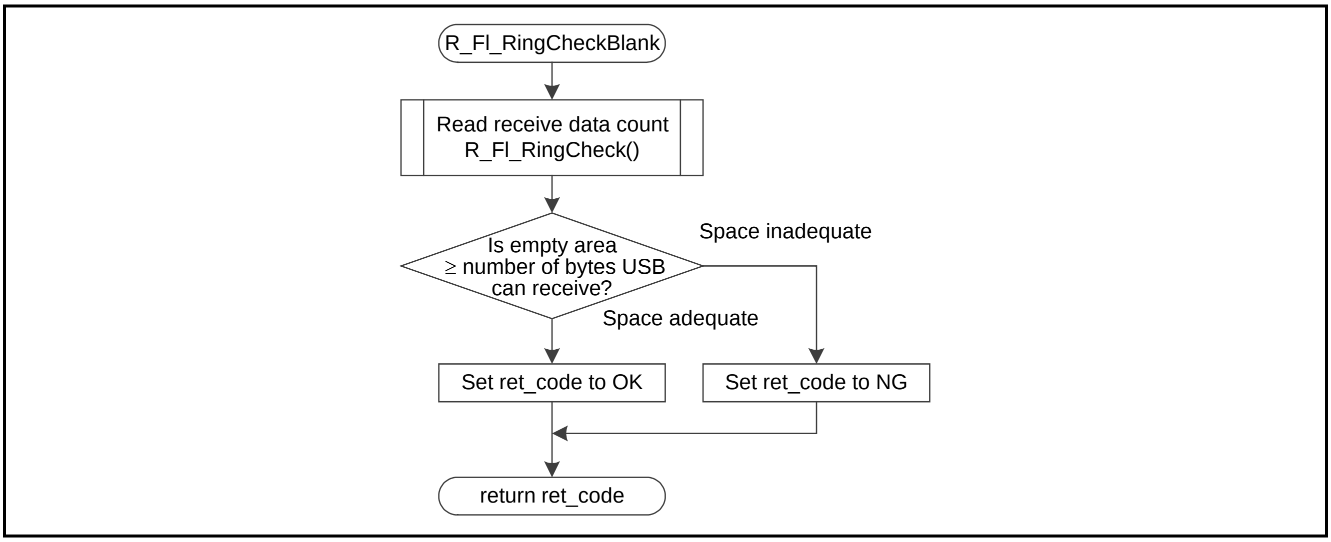

Figure 5.23 Check for Empty Space in Receive Data Storage Ring Buffer

5.13.16 Stop USB

Figure 5.24 shows the flowchart for stop USB.

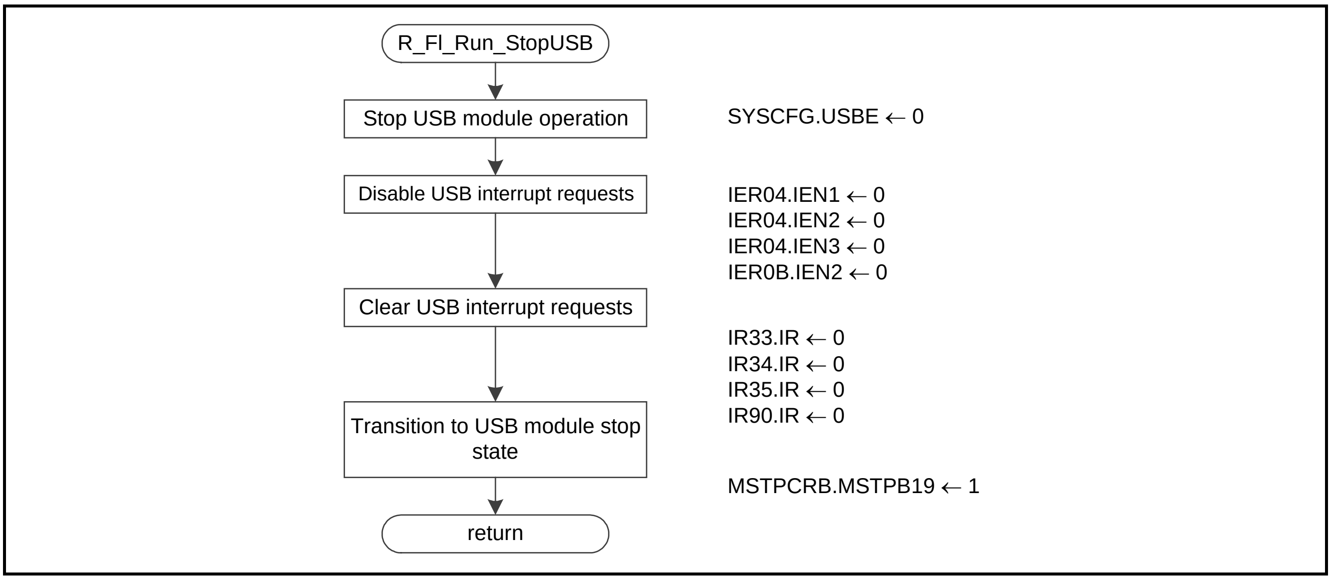

Figure 5.24 Stop USB

5.13.17 Error Handling

Figure 5.25 shows the flowchart for error handling.

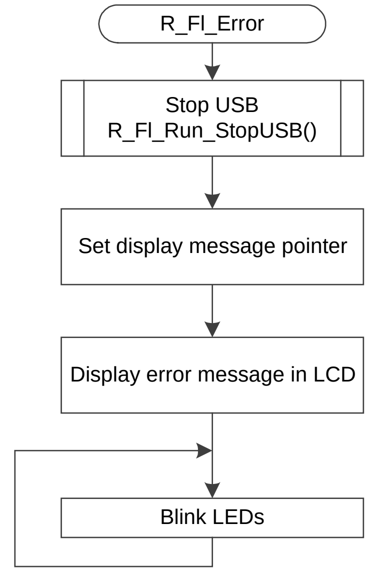

Figure 5.25 Error Handling

5.13.18 Store Data in Receive Data Ring Buffer

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

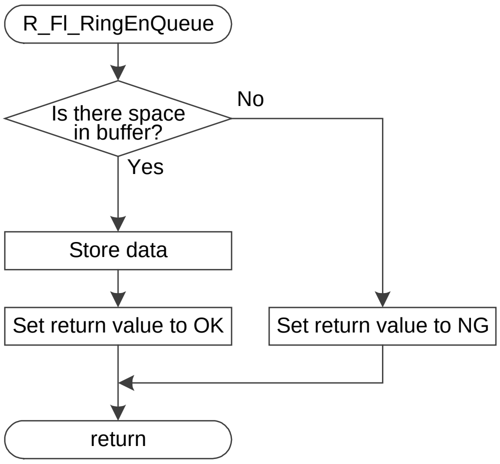

Figure 5.26 Store Data in Receive Data Ring Buffer

5.13.19 Read Data from Receive Data Ring Buffer

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

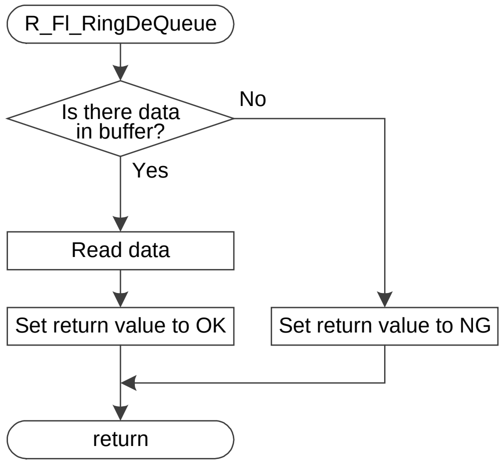

Figure 5.27 Read Data from Receive Data Ring Buffer

5.13.20 Check Data Count in Receive Data Ring Buffer

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

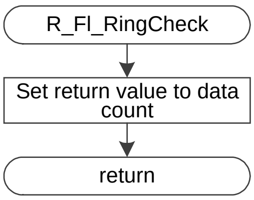

Figure 5.28 Check Data Count in Receive Data Ring Buffer

5.13.21 Convert Data from ASCII to Binary

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

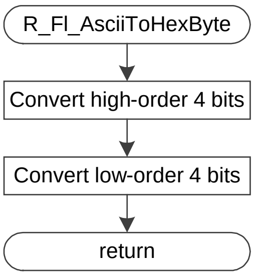

Figure 5.29 Convert Data from ASCII to Binary

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 2 MB of ROM.

7. S Format

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

7.1 Record Formats

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

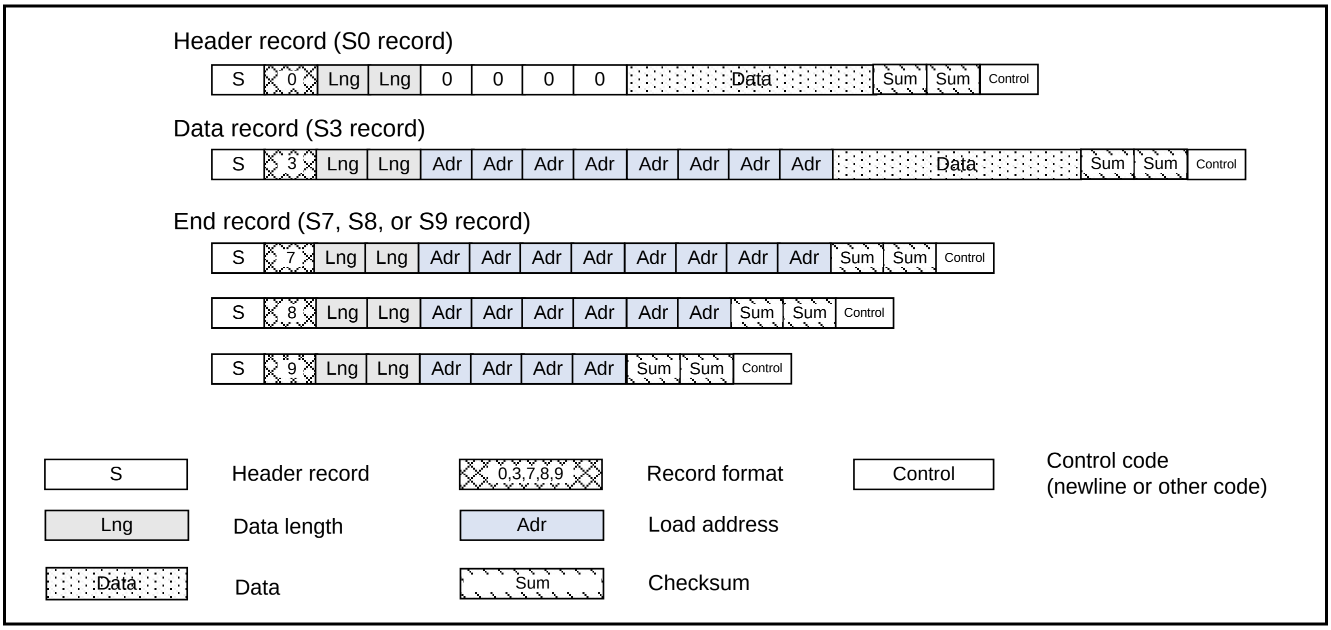

Figure 7.1 Record Formats Supported by Sample Code

7.2 Record Structure

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

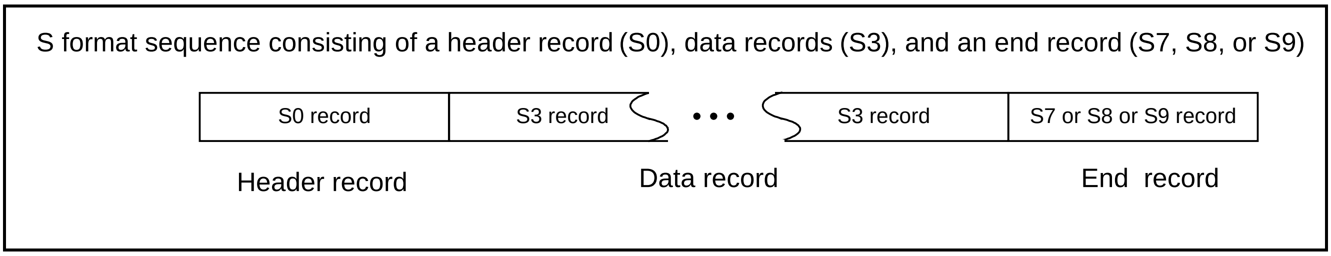

Figure 7.2 Record Structure Supported by the Sample Code

7.3 Load Address

The sample code only supports S type format files with increasing load addresses. Do not use S type 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 S format file received.

(a) Checksum error

The sample code verifies the checksum at each received 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 an 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.

Figure 7.3 shows the format error detection conditions.

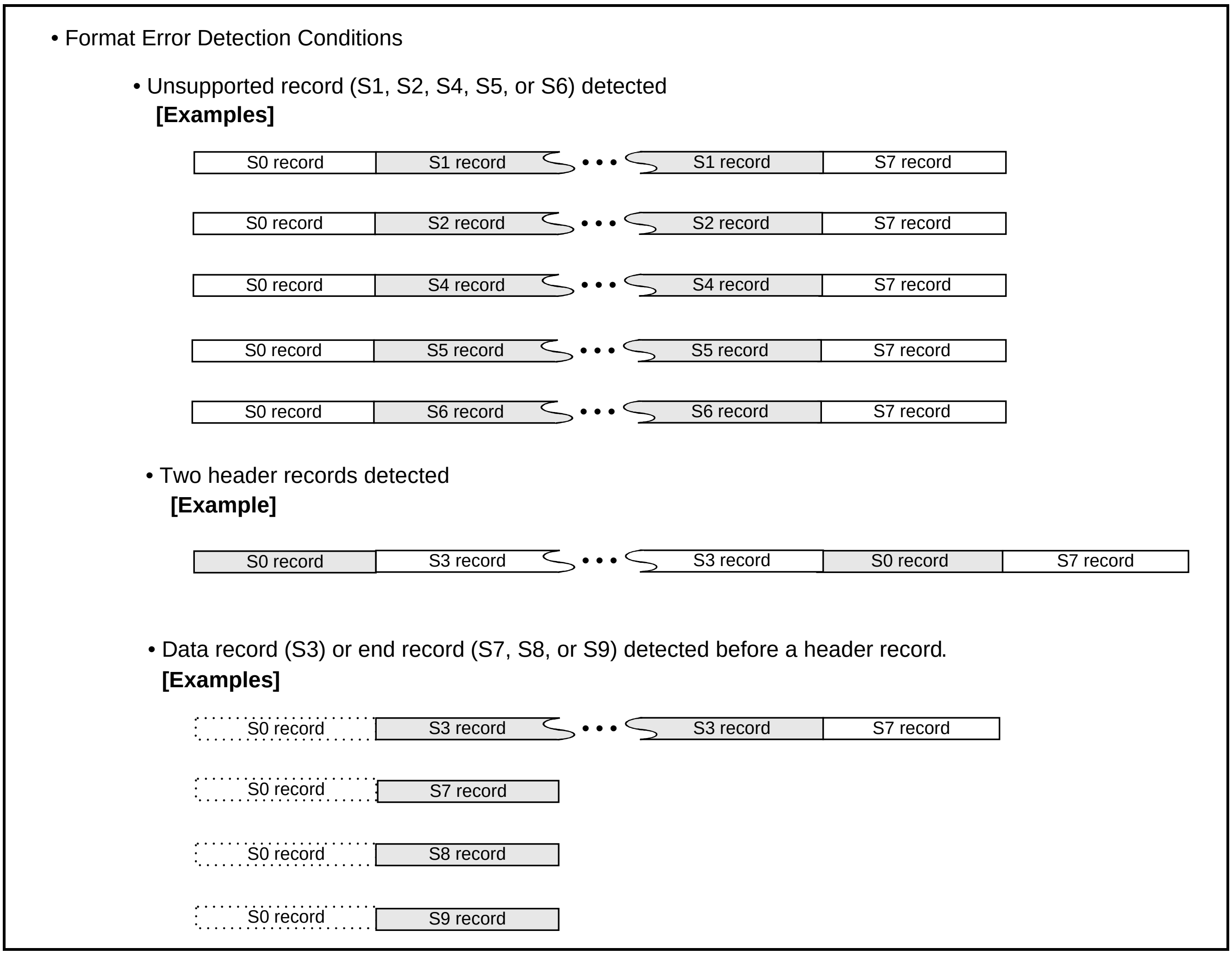

Figure 7.3 Format Error Detection Conditions

(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 RX Family RX600 Simple Flash API 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 BFFCh). Therefore the download code must be set up so that its reset vector is allocated at FFFD BFFCh. 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 2 MB. If a microcontroller with a ROM capacity of 1.5 MB, 1 MB, or 768 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
R5F563xE	2 M	496K	FFE0 0000h	EB15 to EB69
R5F563xD	1.5 M	432K	FFE8 0000h	EB15 to EB61
R5F563xB	1 M	368K	FFF0 0000h	EB15 to EB53
R5F563xA	768 K	336K	FFF4 0000h	EB15 to EB45

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_rx_little_v100.lib or tfat_rx_big_v100.lib)
   Specify $(WORKSPDIR)\WorkSpace\MSCFW\TFAT\lib\tfat_rx600_little_v100.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_rx_little_v100.lib or tfat_rx_big_v100.lib)
   Specify $(WORKSPDIR)\WorkSpace\MSCFW\TFAT\lib\tfat_rx600_big_v100.lib as the linker input library file option.

8.8 Changes to the RX600 Simple Flash API

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

8.8.1 Changes

The files in the RX600 Simple Flash API that are changed are r_flash_api_rx600_config.h and mcu_info.h.

- Changes to the file r_flash_api_rx600_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 FLASH_READY_IPL 5

  (2) The following Simple Flash API settings are changed.
  Before change: #define IGNORE_LOCK_BITS
  #define COPY_CODE_BY_API
  #define FLASH_API_USE_R_BSP
  After change: //#define IGNORE_LOCK_BITS
  //#define COPY_CODE_BY_API
  //#define FLASH_API_USE_R_BSP

- Changes to the file mcu_info.h
  (1) The files contained in the Simple Flash API r_bsp/board/rskrx63n folder are stored in $(WORKSPDIR)\WorkSpace\Flash\ and used.
  (2) 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\rx_rsk.c, which is included in the USB Host Mass Storage Class Driver files.

Before change:	#define ROM_SIZE_BYTES	(1048576)
	#define ICLK_HZ	(96000000)
	#define PCLK_HZ	(48000000)
	#define BCLK_HZ	(24000000)
	#define FCLK_HZ	(48000000)
After change:	#define ROM_SIZE_BYTES	(2097152)

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
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 is added to the include directories.

8.9.5 Macro Definitions (Compiler options)

- R_FLASH_USB is added as a compiler option macro definition.
- USBC_SDRAM_USE_PP is removed from 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

- RX63N Group, RX631 Group User's Manual: Hardware, Rev.1.50
  (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.)
- C Compiler Manual
  RX Family C/C++ Compiler Package V.1.01
  RX Family C/C++ Compiler Package 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
  (The latest version can be downloaded from the Renesas Electronics Web site.)
  Renesas USB Device USB Basic Firmware Rev.2.00
  (The latest version can be downloaded from the Renesas Electronics Web site.)
  M3S-TFAT-Tiny: Fat File System Software Rev.1.00
  (The latest version can be downloaded from the Renesas Electronics Web site.)
  RX600 Series Simple Flash API for RX600 Rev.2.20
  (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	RX63N Group, RX631 Group Application Note USB Host Flash Boot Loader

Rev.	**Date**	**Description**	
		Page	**Summary**
1.00	**Apr. 05, 2013**	—	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 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.

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