RX62N Group, RX621 Group

Ethernet Flash Boot Loader Using M3S-T4-Tiny

R01AN0891EJ0100
Rev.1.00
Mar 13, 2012

Introduction

This application note presents an Ethernet flash boot loader that uses the M3S-T4-Tiny TCP/IP protocol stack. This flash boot loader writes an S format file transferred from the host PC over an Ethernet connection to the microcontroller's internal flash memory.

Note that this application note uses the sample code and libraries described in the following application notes.

- Data transfer over an Ethernet
  RX Family TCP/IP for Embedded system M3S-T4-Tiny: Introduction Guide Rev. 1.02 (R20AN0051EJ0102)
- Erasing and writing internal flash memory
  RX600 Series Simple Flash API for RX600 Rev2.20 (R01AN0544EU0220)

The application note has the following features.

- An S format program stored on a PC can be written to flash memory.
  This application note uses an application (the host PC sample program) running on the host PC to transfer an S format file over an Ethernet connection, and erases the microcontroller's internal flash memory and writes that file to the 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.
- Ethernet Specifications
  TCP is used as the transport layer protocol.

Target Device

RX62N and RX621 Group microcontrollers

If the code provided with this application note is used on any other microcontroller, it must be modified according to the specifications of that microcontroller and thoroughly tested.

Contents

1. Specifications

The sample code in this application note operates with a host PC connected with an Ethernet cable and the RX62N RSK.

If a reset is cleared with switch SW3 on the RX62N RSK not held down, the RX62N RSK will communicate with the host PC (which uses the host PC sample program) using the TCP/IP protocol and a program in the S format file stored on the host PC will be transferred as data to the RX62N RSK and written to the microcontroller's internal flash memory. 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 5.3, Operation Overview, for details.

If a reset is cleared with switch SW3 on the RX62N RSK held down, the program written to the microcontroller's internal flash memory (also referred to as the downloaded code) will be executed.

The result of writing the program to internal flash memory is displayed in the LEDs (LED0 to LED3) on the RX62N RSK. See section 5.6, Sample Code 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.
ETHERC: Ethernet controller	Communication with the host PC
EDMAC: Dedicated DMA controller used by the Ethernet controller	Controls transmission and reception of data in communication with the host PC
CMT: Compare match timer	Used for time management by the TCP/IP protocol stack (M3S-T4-Tiny)

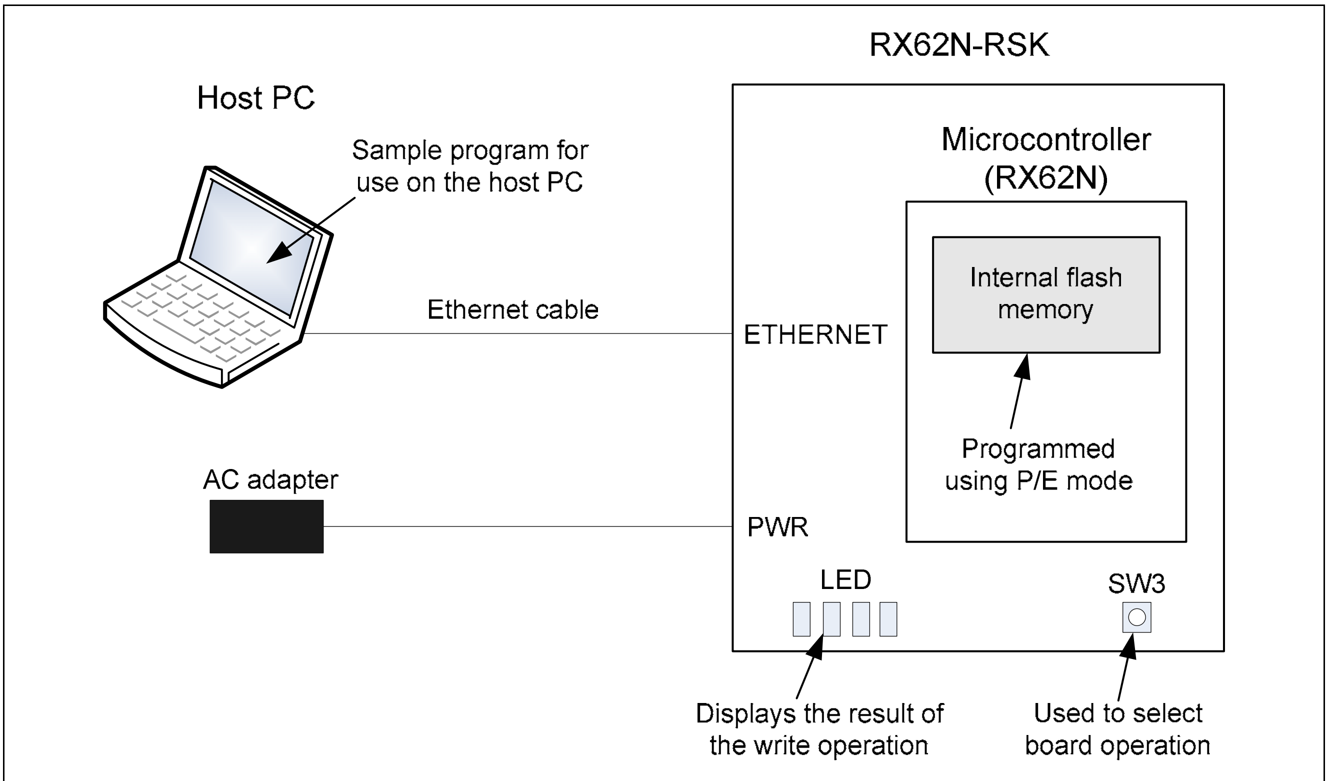

Figure 1.1 Usage Example

2. Confirmed Operating Conditions

The sample code provided with this application note has been confirmed to operate under the conditions listed in tables 2.1 and 2.2.

Table 2.1 Confirmed Operating Conditions

Item	Description
Microcontroller used	RX62N Group (R5F562N8BDBG)
Device used	R5F562N8BDBG
Operating frequency	• EXTAL: 12 MHz • ICLK: 96 MHz • PCLK: 48 MHz • BCLK: 24 MHz • SDCLK: 24 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics High-performance Embedded Workshop Version 4.09.00.007
C compiler	Renesas Electronics RX Standard Toolchain Version 1.1.0.0
	cpu=rx600 -include="$(PROJDIR)\src\bsp","$(PROJDIR)\src\FlashAPI", "$(PROJDIR)\src\driver","$(PROJDIR)\src\t4\lib", "$(PROJDIR)\src\user_app" -output=obj="$(CONFIGDIR)\$(FILELEAF).obj" -debug -nologo *1
Processor mode	Supervisor mode
Operation mode	Single-chip mode
Endian	Little endian / big endian
Version of the sample code	Ver.1.00
Evaluation board used	The RSK + RX62N packed with the Renesas Development Tools (catalog number: R0K5562N0S000BE) is used.

Note: Add the setting "-endian=big" if the big endian order is used.

Table 2.2 Confirmed Operating Conditions (Host PC Sample Program)

Item	Description
Hardware	PC/AT compatible (an Ethernet interface is required)
Operating system	Microsoft Windows XP Professional, Service Pack 3
Tools used	Command prompt (cmd.exe)

3. Related Application Notes

The following application notes are related to this document and should be referred to when using this application note.

- RX Family TCP/IP for Embedded system M3S-T4-Tiny: Introduction Guide Rev.1.02 (R20AN0051EJ)
- RX600 Series Simple Flash API for RX600 Rev.2.20 (R01AN0544EU)

4. Hardware Description

4.1 List of Used Pins

Table 4.1 lists the pins and functions used.

Table 4.1 Used Pins and their Functions

Pin Name	I/O	Description
ET_MDC	Output	Reference clock signal for information transfer via ET_MDIO Connected to the PHY MDC pin.
ET_MDIO	I/O	Bidirectional signal for exchange of management information between this LSI and PHY-LSI Connected to the PHY MDIO pin.
ET_LINKSTA	Input	Inputs link status from PHY-LSI Connected to the PHY LINK/PHYAD1 pin.
ET_TX_CLK	Input	Transmit clock Connected to the PHY TX_CLK pin.
ET_ETXD0	Output	4-bit transmit data Connected to the PHY TXD0 pin.
ET_ETXD1	Output	4-bit transmit data Connected to the PHY TXD1 pin.
ET_ETXD2	Output	4-bit transmit data Connected to the PHY TXD2 pin.
ET_ETXD3	Output	4-bit transmit data Connected to the PHY TXD3 pin.
ET_TX_EN	Output	Transmit enable signal Connected to the PHY TX_EN pin.
ET_TX_ER	Output	Sends error state occurred during data reception to the PHY-LSI Connected to the PHY TX_ER pin.
ET_COL	Input	Collision detection signal Connected to the PHY COL pin.
ET_CRS	Input	Carrier detection signal Connected to the PHY CRS pin.
ET_RX_CLK	Input	Receive clock Connected to the PHY CLK pin.
ET_ERXD0	Input	4-bit receive data Connected to the PHY RXD0 pin.
ET_ERXD1	Input	4-bit receive data Connected to the PHY RXD1 pin.
ET_ERXD2	Input	4-bit receive data Connected to the PHY RXD2 pin.
ET_ERXD3	Input	4-bit receive data Connected to the PHY RXD3 pin.
ET_RX_DV	Input	Indicates that valid receive data is on ET_ERXD3 to ET_ERXD0 Connected to the PHY RX_DV pin.
ET_RX_ER	Input	Receive error Identifies error state occurred during data reception Connected to the PHY RX_ER pin.
MDE	Input	Mode pin The endian order is changed under control of the MDE pin.

Pin Name	I/O	Description
MD0	Input	Mode pin The operating mode is changed under control of the MD0 pin.
MD1	Input	Mode pin The operating mode is changed under control of the MD1 pin.
P07	Input	Sample code operation selection pin
P02	Output	LED connection
P03	Output	LED connection
P05	Output	LED connection
P34	Output	LED connection

5. Software Description

5.1 Software Structure of the Sample Code

In the sample code, the RX Family M3S-T4-Tiny: Introduction Guide application note is used for Ethernet communication and the RX Family RX600 Simple Flash API application note is used for the erase and write processing for the internal flash memory.

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

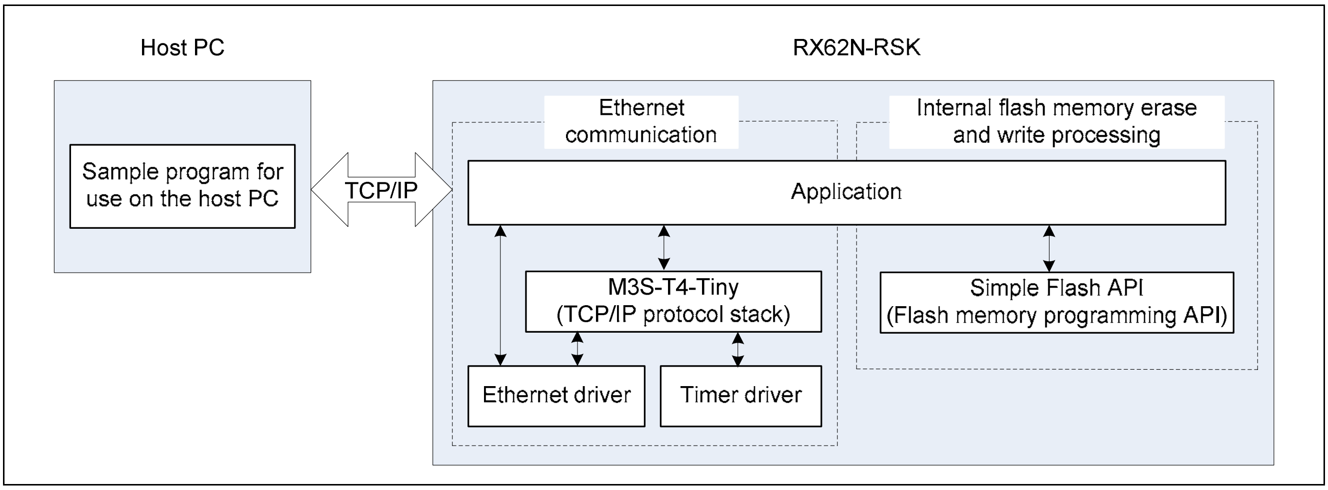

Figure 5.1 Software Structure of the Sample Code

Table 5.1 Software Overview

Module	Overview
Application	The application calls functions provided by M3S-T4-Tiny and receives an S-typ format file over the Ethernet from the Host PC. It also uses the Simple Flash API functions to erase and write to the internal flash memory.
M3S-T4-Tiny	The TCP/IP protocol stack
Ethernet driver	This driver allows applications to use the Ethernet controller (ETHERC) and the dedicated Ethernet DMA controller (EDMAC).
Timer driver	This driver allows applications to use the compare match timer (CMT).
Simple Flash API	API that allows applications to erase and write to the internal flash memory.
Host PC sample program	This sample program runs on the host PC. It communicates with the RX62N RSK from the host PC using the TCP/IP protocol over an Ethernet connection and sends an S format file on the host PC to the RX62N RSK. See section 7, Host PC Sample Program, for details.

5.2 Sample Code Folder Structure

Figure 5.2 shows the structure of the folders that hold the sample code.

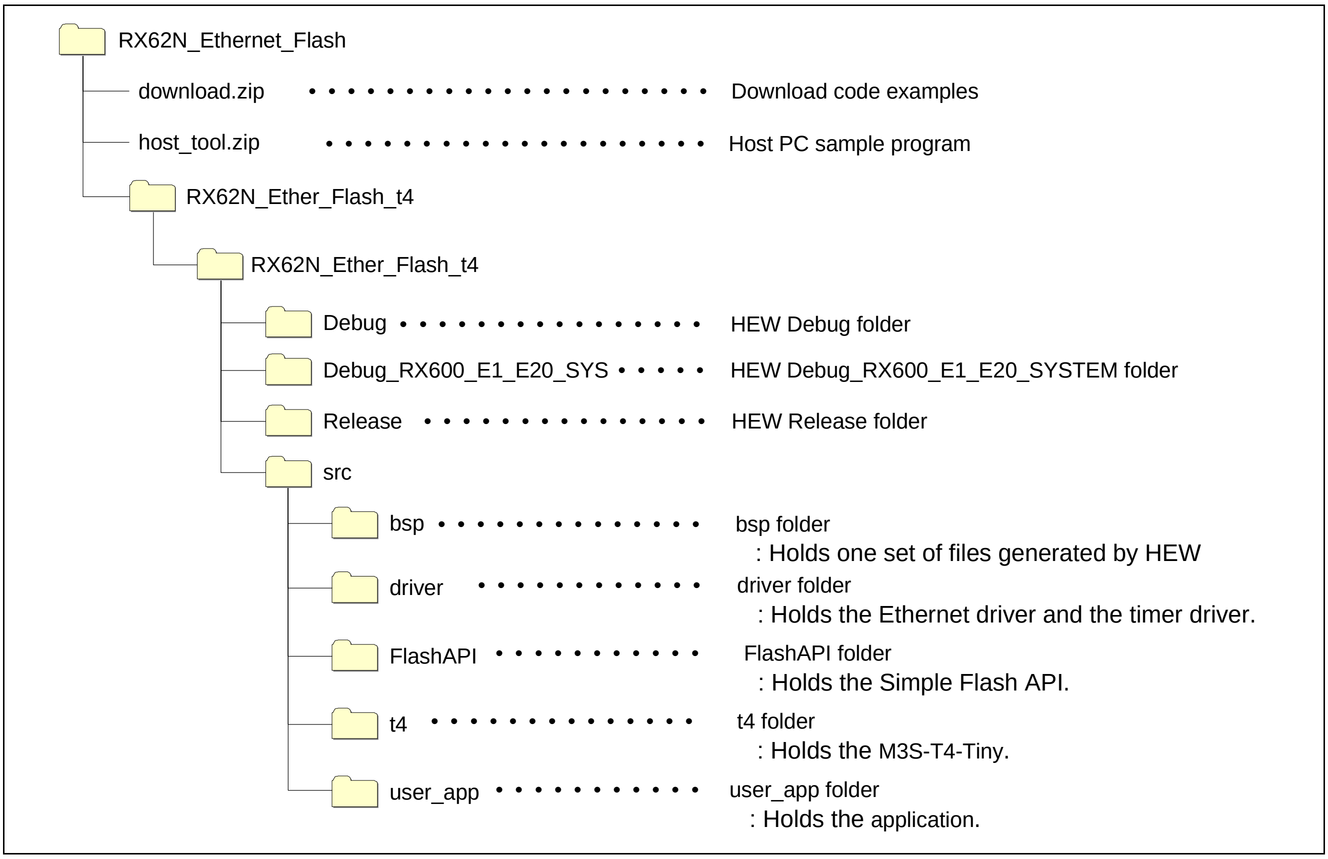

Figure 5.2 Sample Code Folder Structure

5.3 Operation Overview

5.3.1 Operation after a reset is cleared

After a reset is cleared, the sample code checks the state of switch SW3 (pin P07 on the microcontroller) on the RX62N RSK. If this switch is not being pressed (if microcontroller pin P07 is high), it runs the Ethernet flash boot downloader, which uses M3S-T4-Tiny, and rewrites the internal flash memory with data acquired over the Ethernet connection. If, however, the switch is being pressed (if microcontroller pin P07 is low), it runs the downloaded code.

Figure 5.3 shows the operation after a reset is cleared.

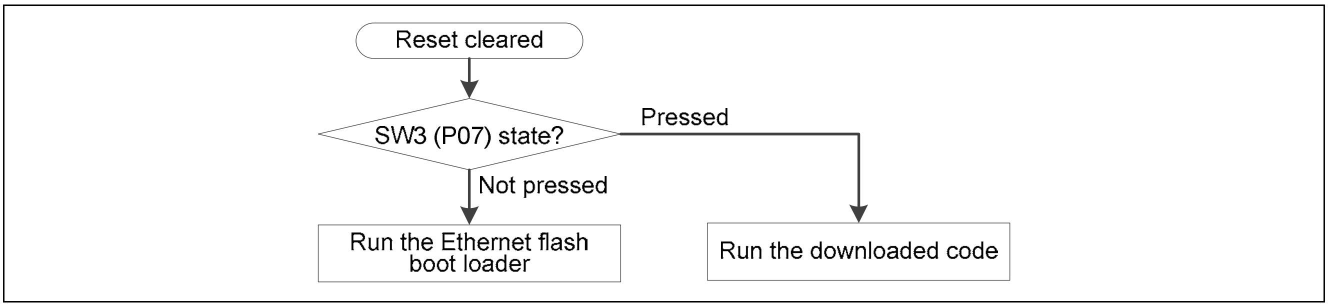

Figure 5.3 Operation After a Reset is Cleared

5.3.2 Object of Overwriting

The object area that the M3S-T4-Tiny based Ethernet 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, FFFF A000h to FFFF FFFFh, is not overwritten.

Figure 5.4 shows the memory allocation.

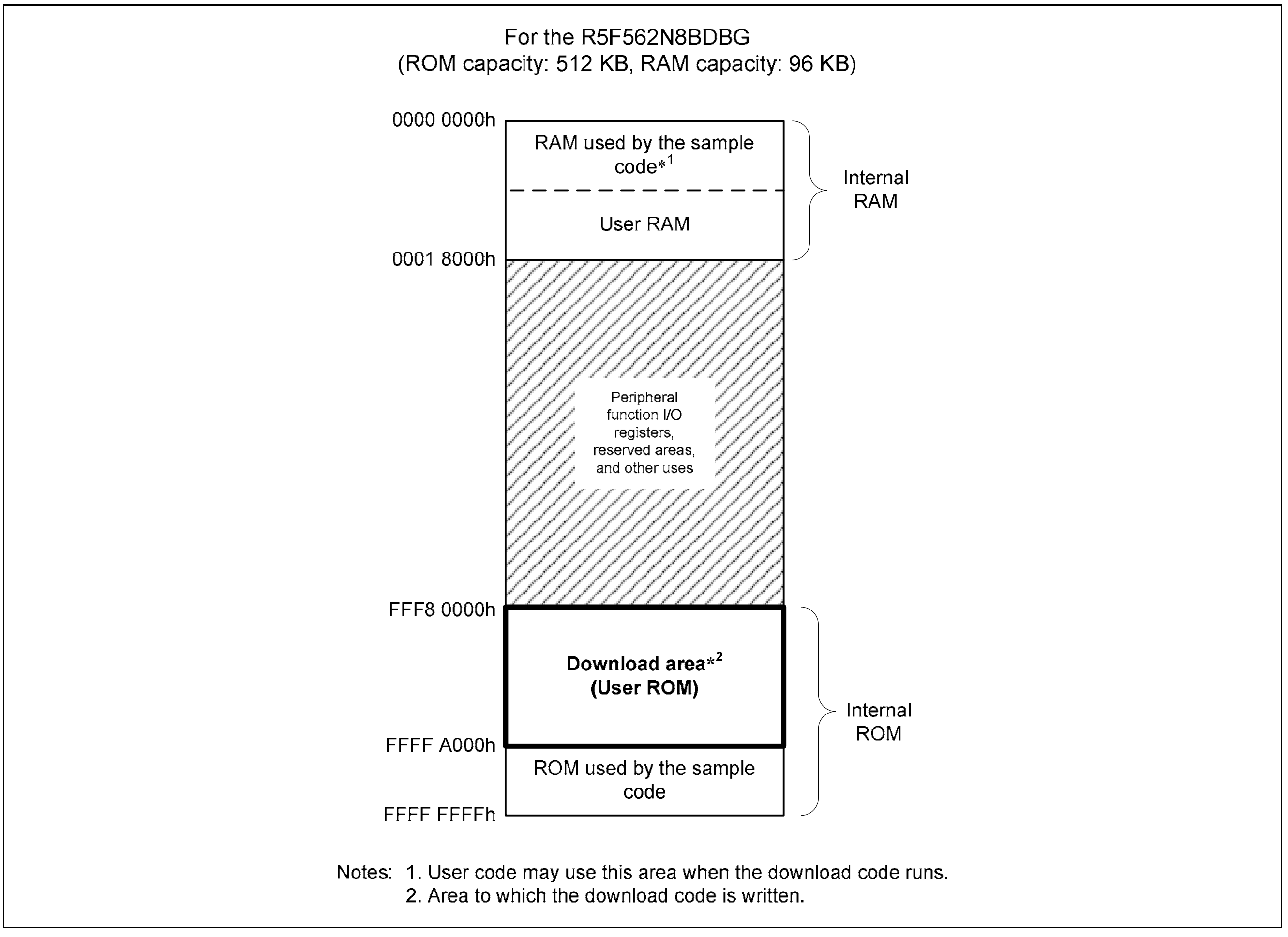

Figure 5.4 Memory Allocation

5.3.3 M3S-T4-Tiny Based Ethernet Flash Boot Loader Operation

The M3S-T4-Tiny based Ethernet flash boot loader uses the following procedure to program the download area. Figure 5.5 shows the download area programming procedure.

Preparations for running the M3S-T4-Tiny based Ethernet flash boot loader.

(1) Set the PC IP address and subnet mask as shown below and connect the RX62N RSK to the PC with an Ethernet cable.
IP address: 192.168.0.2
Subnet mask: 255.255.255.0

(2) Open a command prompt window on the PC and switch to the directory that holds the host PC sample program (RX62N-test_client.exe).

Run the M3S-T4-Tiny based Ethernet flash boot loader.

(1) After performing the above preparations, specify the RX62N RSK IP address (192.168.0.3), the port number (1024), the name of the S format file to be written (as the arbitrary file name) as command line arguments and run the host PC sample program (RX62N-test_client.exe) as shown in figure 5.6.
(2) After the RX62N RSK reset is cleared, a TCP connection will be established between the host PC and the RX62N RSK, and after the host PC checks the file size, it sends the file size to the RX62N RSK.
(3) After receiving the file size, the RX62N RSK erases the download area and reports completion of the erase operation to the host PC.
(4) After receiving confirmation of erase completion, the host PC sends S format data to the RX62N RSK.
(5) The RX62N RSK receives up to 1,400 bytes of S format data at a time and after analyzing the data, writes it in 256-byte units to the download area.
(6) The processing of steps (4) and (5) above are repeated until an S format end record (an S7, S8, or S9 record) is detected.
(7) The result of the write operation is displayed in the LEDs on the RX62N RSK and either normal completion or error termination of the write operation is reported to the host PC.
(8) If the erase and write of the download area completes normally, the result will be displayed on the host PC as shown in figure 5.7.

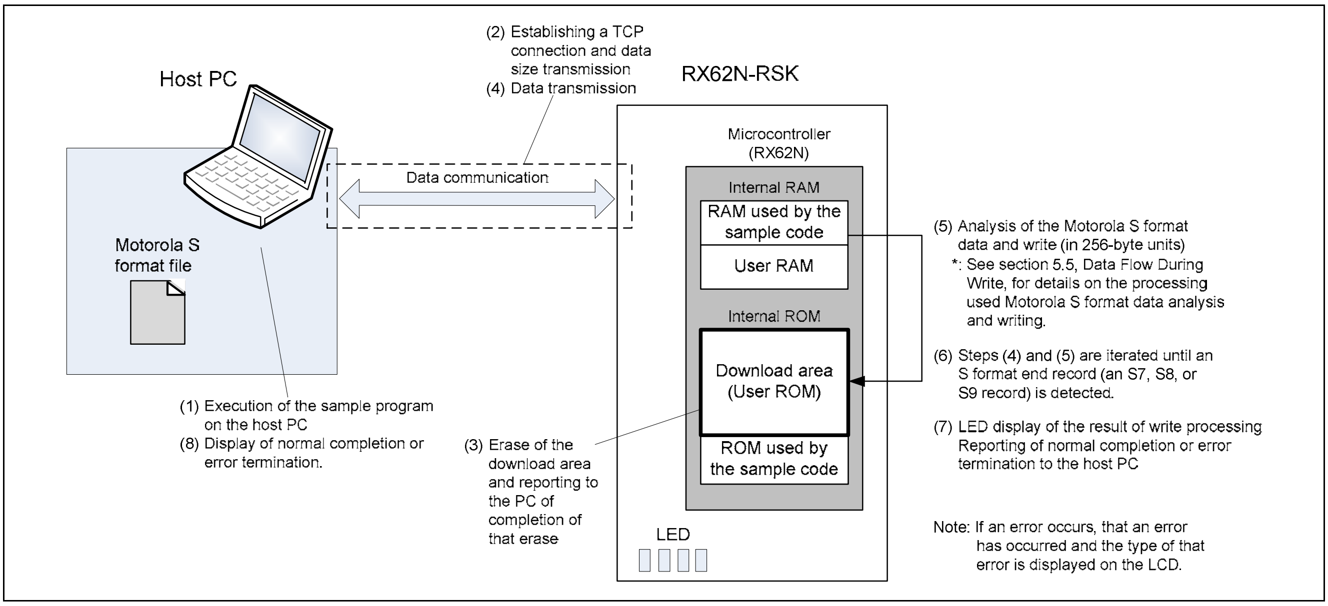

Figure 5.5 Programming the Download Area

```
Command Prompt
C:\>
C:\>RX62N-test_client.exe 192.168.0.3 1024 download.mot
```

Figure 5.6 Host PC Sample Program: Argument Input

```
Command Prompt
C:\>
C:\>RX62N-test_client.exe 192.168.0.3 1024 download.mot
***********************************
    RX62N-test_client
***********************************
connected to the server.

[send] Data size : 5536 byte
[recv] The flash programming start.
[send] Data      : 5536 byte
[recv] The flash programming was completed.

Please press <Enter> key.
```

Figure 5.7 Host PC Sample Program: Execution Results

5.3.4 M3S-T4-Tiny Based Ethernet Flash Boot Loader Operation

Figures 5.8 and 5.9 shows the overall flow of operations including operations on the host PC. Note that the arrows between the host PC and the RX62N RSK indicate TCP communication.

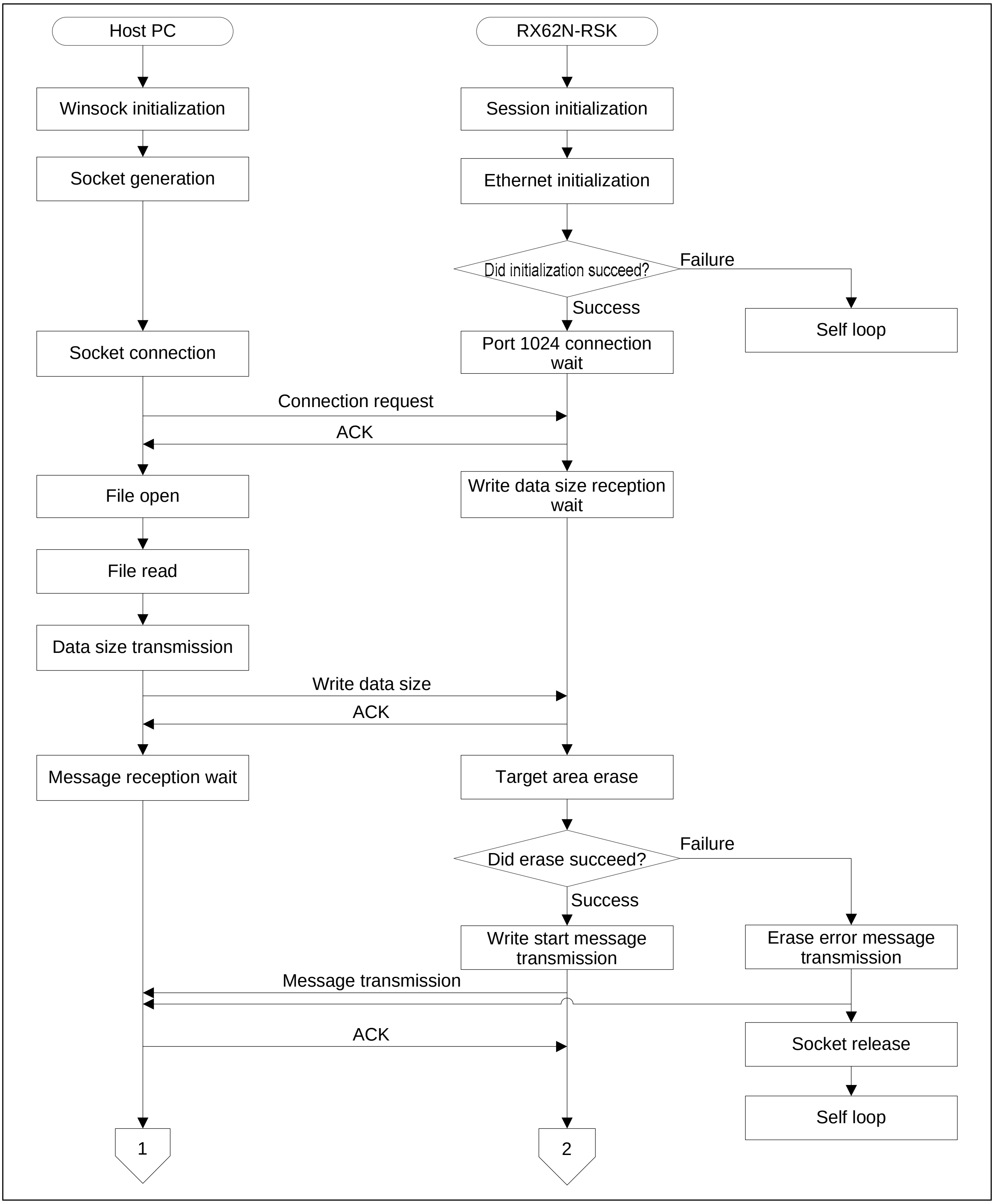

Figure 5.8 Overall Flow of Operations

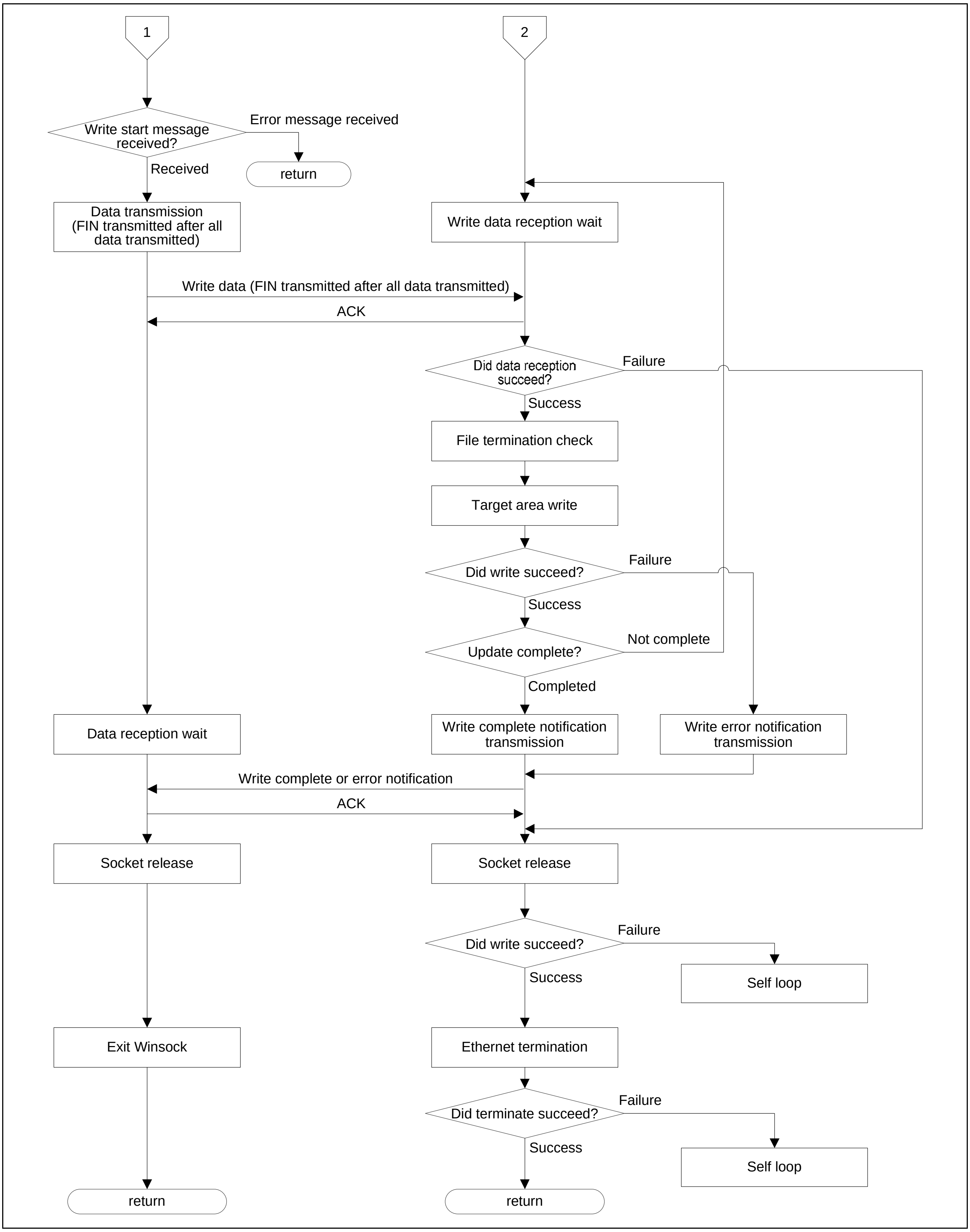

Figure 5.9 Overall Flow of Operations (continued)

5.4 Executing the Download Code

If switch SW3 is in the pressed state (the microcontroller pin P07 is low) when a microcontroller reset is cleared, the sample code will run the download code.

5.4.1 Download code execution start position

The sample program runs the download code by performing a function call to the address stored at location FFFF 9FFCh. Therefore the download code must store its start address at location FFFF 9FFCh.

Figure 5.10 shows the execution start position of the download code.

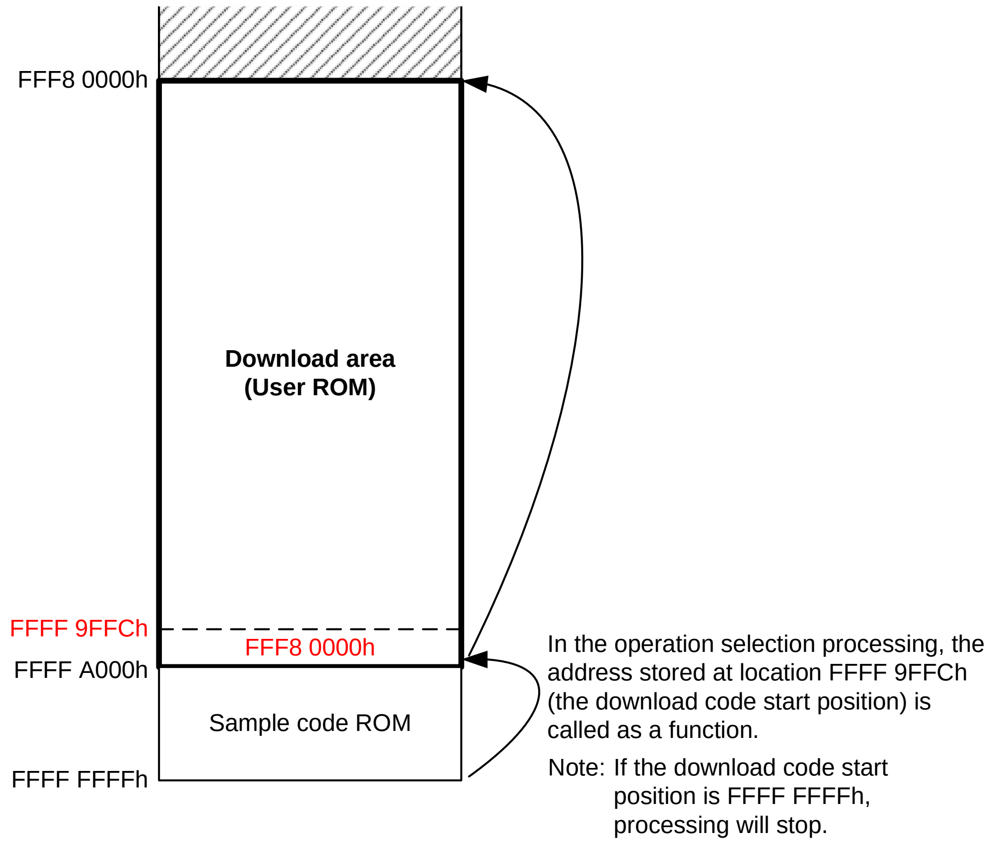

Figure 5.10 Download Code Execution Start Position

Note: If nothing was written to the download code execution start position (that is, if the download code execution start position is FFFF FFFFh), the sample code executes a while (1) infinite loop to stop processing.

5.5 Data Flow During Write

Figure 5.11 shows the data flow internal to the microcontroller when the download code is written to flash memory.

(1) The data received over the Ethernet 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 8, 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 RX62N and RX621 group microcontrollers, data is written to the user MAT in units of 256 bytes. Therefore, the sample code iterates steps (2) to (4) above until a total of 256 bytes of write data has been stored in the write buffer. Also, if the total amount of write data exceeds 256 bytes, the excess data is stored temporarily and used for the next write of 256 bytes of data.
(5) The assembled 256 bytes of data are written to flash memory using the Simple Flash API.

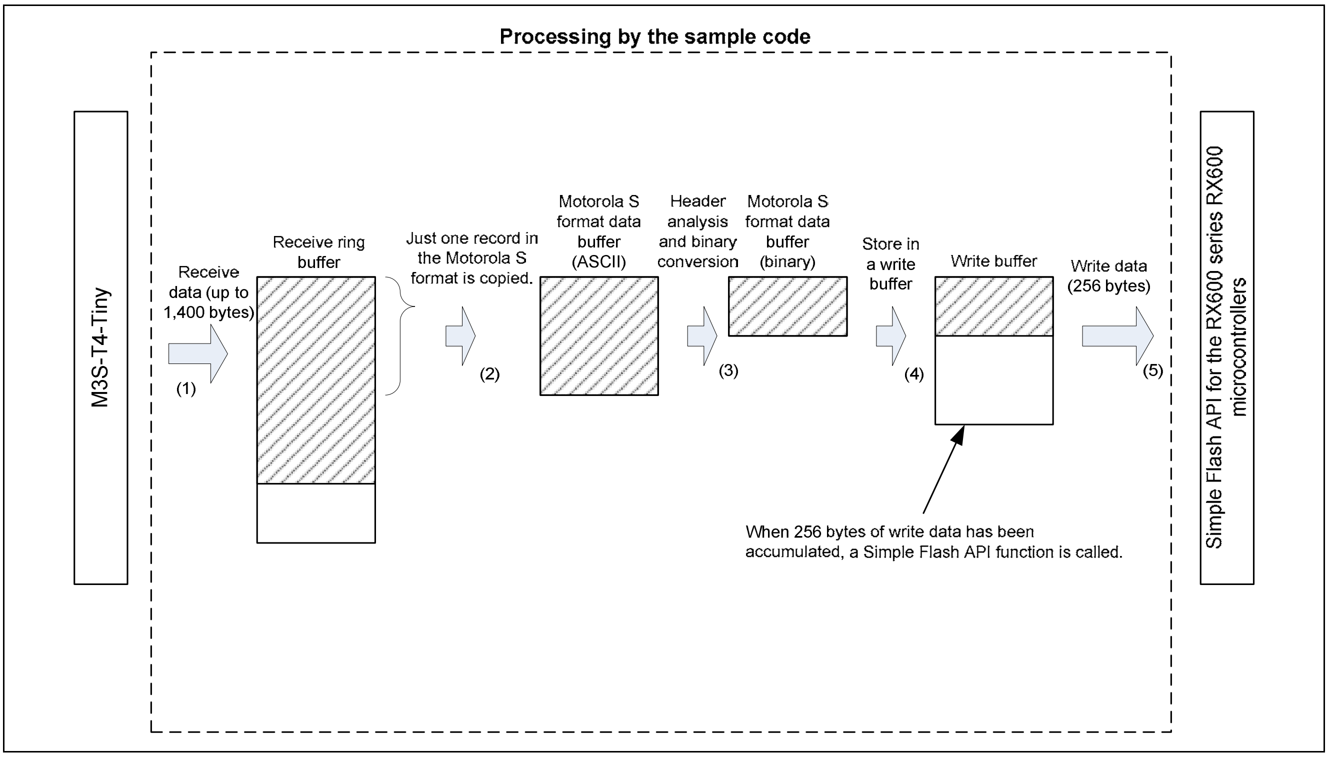

Figure 5.11 Data Flow During Write

Figure 5.12 shows the data structures used when writing data to flash memory.

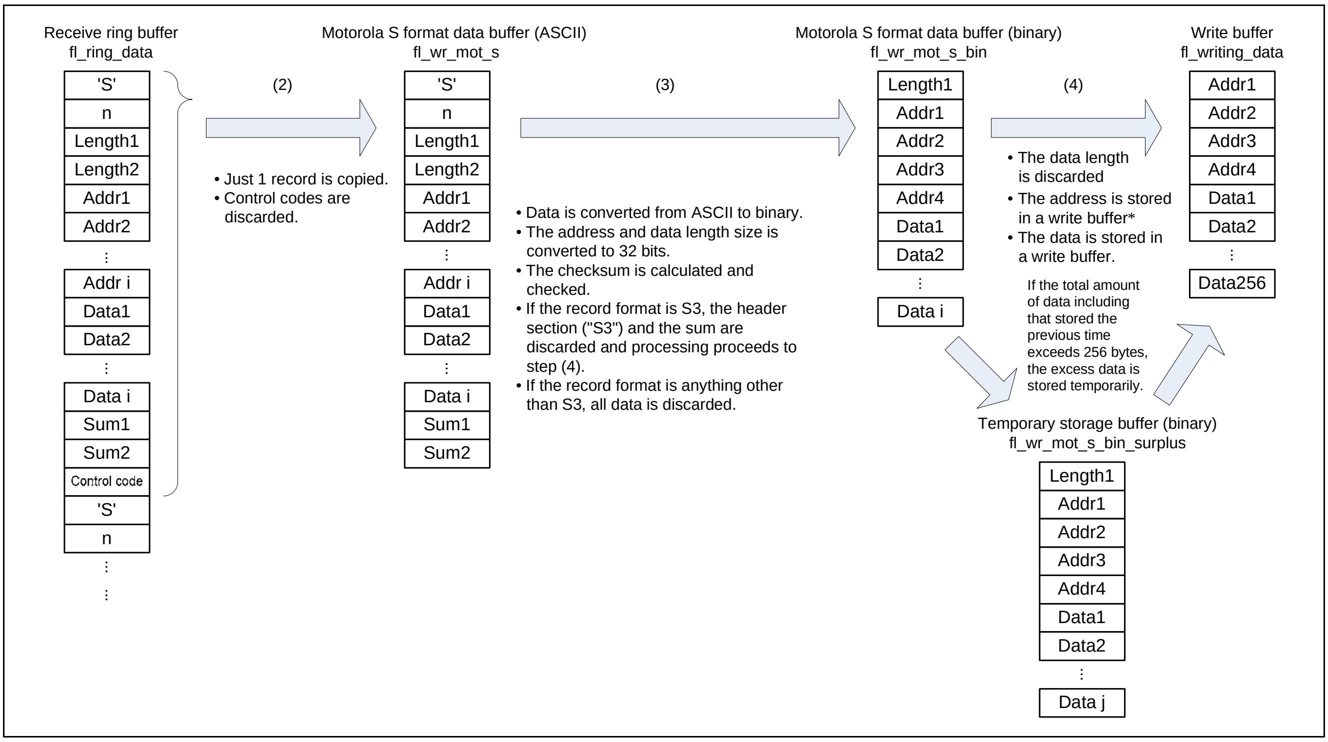

Figure 5.12 Data Structures Used for Writing

Note: In the RX62N and RX621 group microcontroller internal flash memory, a start address used for a write operation must be aligned on a 256-byte boundary. Accordingly, the sample code performs processing to assure that write start addresses are aligned on 256-byte boundaries when storing addresses to write buffers. See the flowchart in section 5.13.12, Download area write data creation, for details on this processing.

5.6 Sample Code LED Display

The sample code displays the result of writing to the internal flash memory on the LEDs on the RX62N RSK. Note that the LEDs (LED0 to LED3) will be turned off during the download processing.

Table 5.2 lists the LED display states produced by the sample code.

Table 5.2 Sample Code LED Display

O: On, ×: Off

LED Display State				
LED3	**LED2**	**LED1**	**LED0**	**Description**
O	O	O	O	Indicates that the write to the internal flash memory succeeded. (Write success)
×	×	×	O	Indicates that Ethernet initialization failed. (Ethernet initialization error)
×	×	O	×	Indicates that Ethernet termination failed. (Ethernet termination error)
×	×	O	O	Indicates that Ethernet connection failed. (Ethernet connection error)
×	O	×	×	Indicates that Ethernet reception failed. (Ethernet receive error)
×	O	×	O	Indicates that Ethernet disconnection failed. (Ethernet disconnect error)
×	O	O	×	Indicates that erase of the internal flash memory failed. (Erase error)
×	O	O	O	Indicates that write of the internal flash memory failed. (Write error)
O	×	×	×	Indicates that the post-write verification of internal flash memory failed. (Verify error)
O	×	×	O	Indicates that even though processing proceeded to the end of the file, there was no S format end record. (File end error)
O	×	O	×	Indicates that the download code start address was found to be FFFF FFFFh when the download code was run. (Download code not written error)
O	×	O	O	Indicates that an abnormality was detected in the S format data checksum. (Checksum error) See section 8, S Format.
O	O	×	×	Indicates that the download code was an unsupported S format. (Format error) See section 8, S Format.
O	O	×	O	Indicates that write data for locations outside the download area was detected. (Address error) See section 8, S Format.

5.7 Memory Requirements

Table 5.3 lists the required memory sizes.

Table 5.3 Memory Requirements

Memory Used	Size	Notes
ROM	24,318 bytes	Since the sample code is allocated to locations FFFF A000h to FFFF FFFFh, the amount of ROM that can be written is the total ROM capacity minus 24,576 bytes.
RAM	40,796 bytes	The user code can use this area when it runs.

Note: The sizes of required memory areas vary with the version and compiler options of the C compiler.

5.8 File Structure

Table 5.4 lists the files that make up the sample code.

Note that files automatically generated by the integrated development environment, download code examples, and the host PC sample program are not included.

Table 5.4 File Structure

File	Overview	Notes
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.	For details, see the RX600 Series RX600 Simple Flash API application note.
r_flash_api_rx600_private.h	External reference include header for the RX600 Series RX600 Simple Flash API program.	For details, see the RX600 Series RX600 Simple Flash API application note.
r_flash_api_rx600_config.h	Parameter settings include header for the RX600 Series RX600 Simple Flash API program.	For details, see the RX600 Series RX600 Simple Flash API application note.
mcu_info.h	Parameter settings include header for the RX600 Series RX600 Simple Flash API program.	For details, see the RX600 Series RX600 Simple Flash API application note.
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	Ethernet receive ring buffer related processing	
r_Flash_buff.h	External reference include header for the Ethernet receive ring buffer related processing	
TrgtPrgDmmy.c	Dummy program for allocating the download code area	
main.c	The main() function	
Other files	The programs from the RX Family M3S-T4-Tiny: Introduction Guide	See the RX Family M3S-T4-Tiny: Introduction Guide application note for details.

5.9 Constants

Table 5.5 lists the constants used in the sample code.

Table 5.5 Constants Used in the Sample Code

Constant	Set Value	Description
FL_T4_API_TIMEOUT	1000	M3S-T4-Tiny function timeout time
FL_INPUT_BUFSIZE	1400	Receive buffer size for data received from the Ethernet
FL_RINGBUFF_SIZE	1400	Receive ring buffer size for data received from the Ethernet
FL_MOTS_ADDR_SIZE	4	S format data address buffer size
FL_MOTS_SUM_SIZE	1	S format data checksum buffer size
FL_START_BLOCK_NUM	6	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	FFFF9FFFh	Last address in the download area
FL_RCV_BLANK_SIZE	1400	Ring buffer capacity

5.10 Structures and Unions

Figure 5.13 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];               /* "0-255" */
    uint8_t addr_data_sum[512];
} Fl_prg_mot_s_t;

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

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

Figure 5.13 Structures and Unions Used in the Sample Code

5.11 Functions

Table 5.6 lists the functions. Note, however, that the Simple Flash API, TCP/IP protocol stack, and Ethernet driver functions are not shown here.

Table 5.6 Functions

Function Name	Overview
R_Fl_Mode_Entry	Operation selection processing
R_Fl_Ether_Sample_Init	Ethernet initialization
R_Fl_Ether_Sample_Quit	Ethernet termination
R_Fl_Flash_Update	Main flash write processing
R_Fl_EraseTrgtArea	Erase processing
R_Fl_Ers_EraseFlash	Erase download area
R_Fl_PrgramTrgtArea	Write download area
R_Fl_Prg_PrgramFlash	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_RcvDataString	Store received Ethernet data
R_Fl_RingCheckBlank	Check the amount of free capacity in the ring buffer used to store data received over the Ethernet
R_Fl_RingInit	Initialize ring buffer used to store data received over the Ethernet
R_Fl_RingEnQueue	Store data in the ring buffer used to store data received over the Ethernet
R_Fl_RingDeQueue	Read data from the ring buffer used to store data received over the Ethernet
R_Fl_RingCheck	Verify number of data items in ring buffer used to store data received over the Ethernet
R_Fl_AsciiToHexByte	Convert data from ASCII to binary
R_Fl_LED_Ini	LED initialization
R_Fl_LED_Fnc	LED on/off state processing

5.12 Function Specifications

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

R_Fl_Mode_Entry

Overview	Operation selection processing
Header	r_Flash_main.h
Declaration	void R_Fl_Mode_Entry(void)
Description	• Selects the operation performed. • Performs LED initialization.
Arguments	None
Return values	None
Notes	Executes the download code if the RX62N RSK switch SW3 is pressed. If SW3 is not pressed, the sample code switches to M3S-T4-Tiny based Ethernet flash boot loader after this function returns.

R_Fl_Ether_Sample_Init

Overview	Ethernet initialization
Header	r_Flash_main.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_Ether_Sample_Init(void)
Description	• Calls the function that initializes and starts the LAN controller. • Calls the function that initializes the M3S-T4-Tiny protocol stack. • Initializes LED display on the RX62N RSK.
Arguments	None
Return values	• If initialization completes normally: FLASH_API_SAMPLE_OK • If initialization does not complete normally: FLASH_API_SAMPLE_NG
Notes	

R_Fl_Ether_Sample_Quit

Overview	Ethernet termination
Header	r_Flash_main.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_Ether_Sample_Quit(void)
Description	• Calls the function that terminates M3S-T4-Tiny operation. • Calls the function that stops the LAN controller.
Arguments	None
Return values	• If termination completes normally: FLASH_API_SAMPLE_OK • If termination does not complete normally: FLASH_API_SAMPLE_NG
Notes	

R_Fl_Flash_Update

Overview	Main flash write processing
Header	r_Flash_main.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_Flash_Update(void)
Description	• Calls an M3S-T4-Tiny function to receive an S format file from the host PC. • Calls the function that rewrites the internal flash memory with the contents of the received S format file.
Arguments	None
Return values	• If the flash write processing completes normally: FLASH_API_SAMPLE_OK • If the flash write processing does not complete normally: FLASH_API_SAMPLE_NG
Notes	

R_Fl_EraseTrgtArea

Overview	Erase processing
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_EraseTrgtArea(void)
Description	Calls the function that erases the download area.
Arguments	None
Return values	• If the flash erase processing completes normally: FLASH_API_SAMPLE_OK • If the flash erase processing does not complete normally: FLASH_API_SAMPLE_NG
Notes	

R_Fl_Ers_EraseFlash

Overview	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 values	• 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

Overview	Write download area
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_PrgramTrgtArea(void)
Description	Calls the function that performs the write processing.
Arguments	None
Return values	• If the write operation completes normally: FLASH_API_SAMPLE_OK • If the write operation does not complete normally: FLASH_API_SAMPLE_NG
Notes	

R_Fl_Prg_PrgramFlash

Overview	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. • If the end of file is reached, verifies whether an S format end record has been received. (If no end record has been received, returns FLASH_API_SAMPLE_NG.)
Arguments	None
Return values	• 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
Notes	

R_Fl_Prg_StoreMotS

Overview	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 values	• 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

Overview	Header analysis, binary conversion, and write of an S format record
Header	None
Declaration	static Fl_API_SMPL_rtn_t 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.
Arguments	None
Return values	• If the function completes normally: FLASH_API_SAMPLE_OK • If data that differs from the S format is found: FLASH_API_SAMPLE_NG
Notes	

R_Fl_Prg_MotS_AsciiToBinary

Overview	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.
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 values	• If conversion completed normally: FLASH_API_SAMPLE_OK • If a checksum error occurred: FLASH_API_SAMPLE_NG
Notes	

R_Fl_Prg_MakeWriteData

Overview	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 at each 256-byte unit.
Arguments	None
Return values	• If creation of 256 bytes of write data completed: FLASH_API_SAMPLE_OK • If creation of 256 bytes of write data did not complete: FLASH_API_SAMPLE_NG
Notes	

R_Fl_Prg_WriteData

Overview	Write to download area
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_Prg_WriteData(void)
Description	• Verifies that the write is to the download area. • Performs the write to the download area. • Verifies the data written. • Calls the error handler if the write failed.
Arguments	None
Return values	• If the write completed normally: FLASH_API_SAMPLE_OK • If the write did not complete normally: FLASH_API_SAMPLE_NG
Notes	The processor status word (PSW) interrupt priority level (IPL) is modified to prevent ROM access by interrupts during the write operation.

R_Fl_Prg_ClearMotSVariables

Overview	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 values	None
Notes	

R_Fl_RcvDataString

Overview	Store received Ethernet data
Header	None
Declaration	static Fl_API_SMPL_rtn_t R_Fl_RcvDataString(void *, uint16_t)
Description	Stores data received over the Ethernet in a receive ring buffer.
Arguments	First argument: *tranadr : Pointer to a buffer that holds data received over the Ethernet Second argument: length : Length of the data received over the Ethernet
Return values	• If the store completed normally: FLASH_API_SAMPLE_OK • If the store did not complete normally: FLASH_API_SAMPLE_NG
Notes	

R_Fl_RingCheckBlank

Overview	Check the amount of free capacity in the ring buffer used to store data received over the Ethernet
Header	r_Flash_buff.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_RingCheckBlank(void)
Description	Verifies that there is enough space in ring buffer used to store data received over the Ethernet for the amount of data received in one transfer (1400 bytes).
Arguments	None
Return values	• If there is enough space: FLASH_API_SAMPLE_OK • If there is not enough space: FLASH_API_SAMPLE_NG
Notes	

R_Fl_RingInit

Overview	Initialize ring buffer used to store data received over the Ethernet
Header	r_Flash_buff.h
Declaration	void R_Fl_RingInit(void)
Description	Initializes the ring buffer used to store data received over the Ethernet.
Arguments	None
Return values	None
Notes	

R_Fl_RingEnQueue

Overview	Store data in the ring buffer used to store data received over the Ethernet
Header	r_Flash_buff.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_RingEnQueue(uint8_t)
Description	Stores data the ring buffer used to store data received over the Ethernet.
Arguments	First argument: enq_data : Data to be stored
Return values	• If the store completed normally: FLASH_API_SAMPLE_OK • If a buffer full error occurred: FLASH_API_SAMPLE_NG
Notes	

R_Fl_RingDeQueue

Overview	Read data from the ring buffer used to store data received over the Ethernet
Header	r_Flash_buff.h
Declaration	Fl_API_SMPL_rtn_t R_Fl_RingDeQueue(uint8_t *)
Description	Reads data from the ring buffer used to store data received over the Ethernet
Arguments	First argument: *deq_data : Pointer to buffer to store read data
Return values	• If the data was read normally: FLASH_API_SAMPLE_OK • If there was no data to read: FLASH_API_SAMPLE_NG
Notes	

R_Fl_RingCheck

Overview	Verify number of data items in ring buffer used to store data received over the Ethernet
Header	r_Flash_buff.h
Declaration	uint32_t R_Fl_RingCheck(void)
Description	Verifies the number of data items in ring buffer used to store data received over the Ethernet.
Arguments	None
Return values	Returns the number of data items stored.
Notes	

R_Fl_AsciiToHexByte

Overview	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 values	Returns the converted binary data.
Notes	

R_FI_LED_Ini

Overview	LED initialization
Header	None
Declaration	static void R_FI_LED_Ini(void)
Description	Performs the processing required to set the initial states of the LEDs on the RX62N RSK.
Arguments	None
Return values	None
Notes	

R_FI_LED_Fnc

Overview	LED on/off state processing
Header	None
Declaration	static FI_API_SMPL_rtn_t R_FI_LED_Fnc(uint8_t)
Description	Performs the processing for turning the LEDs on the RX62N RSK on or off. See section 5.6, Sample Code LED Display, for details.
Arguments	First argument: in_data : Value used to set the LED on/off states
Return values	• If the operation completed normally: FLASH_API_SAMPLE_OK • If the operation did not complete normally: FLASH_API_SAMPLE_NG
Notes	The bits in the in_data argument are used as the on/off setting values for the individual LEDs. The correspondence with the LEDs is shown below. The bits in the argument should be set to 0 to turn the corresponding LED off, and to 1 to turn it on. bit[0]: LED0, bit[1]: LED1, bit[2]: LED2, bit[3]: LED3

5.13 Flowcharts

5.13.1 Operation Selection Processing

Figure 5.14 shows the flowchart for the operation selection processing.

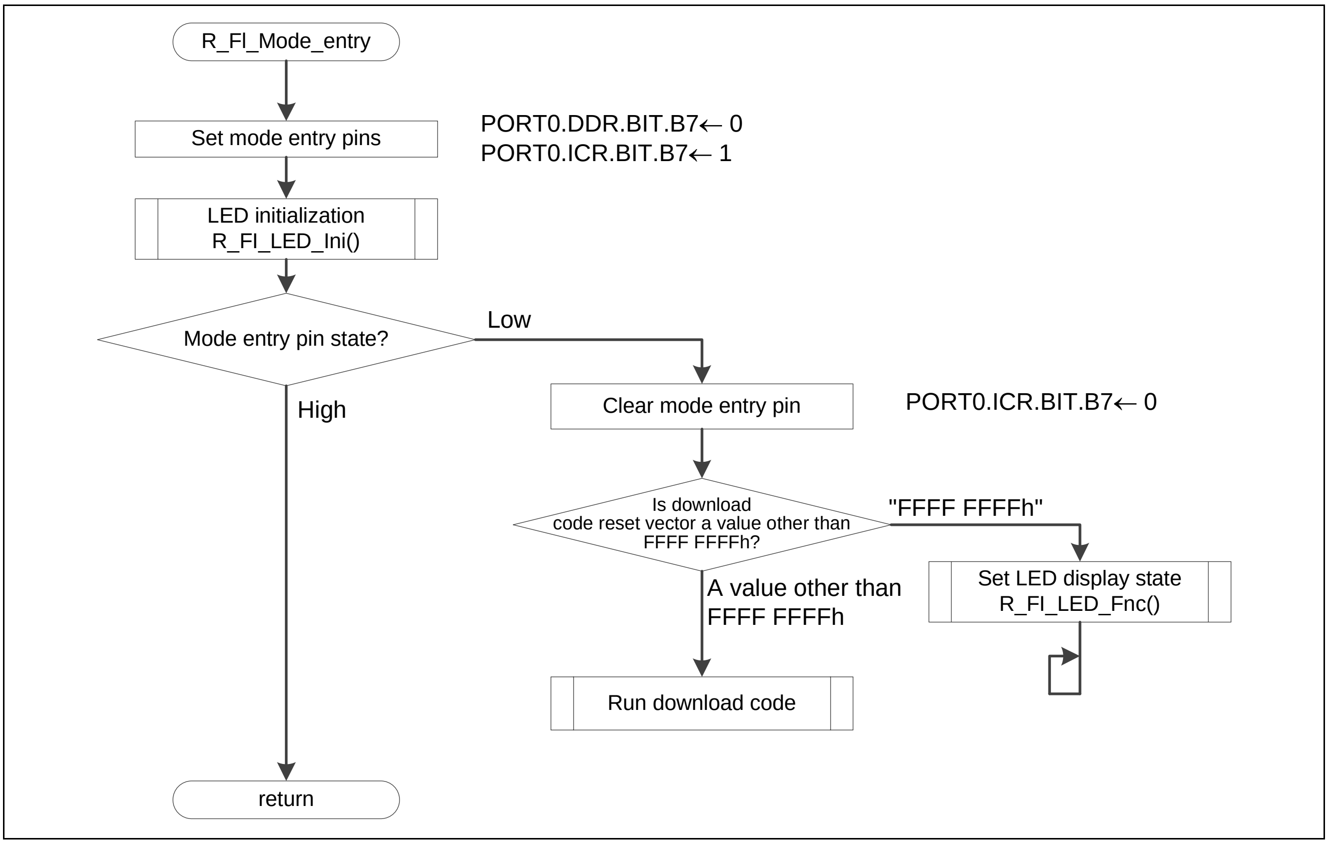

Figure 5.14 Operation Selection Processing

5.13.2 Ethernet Initialization Processing

Figure 5.15 shows the flowchart for the Ethernet initialization processing.

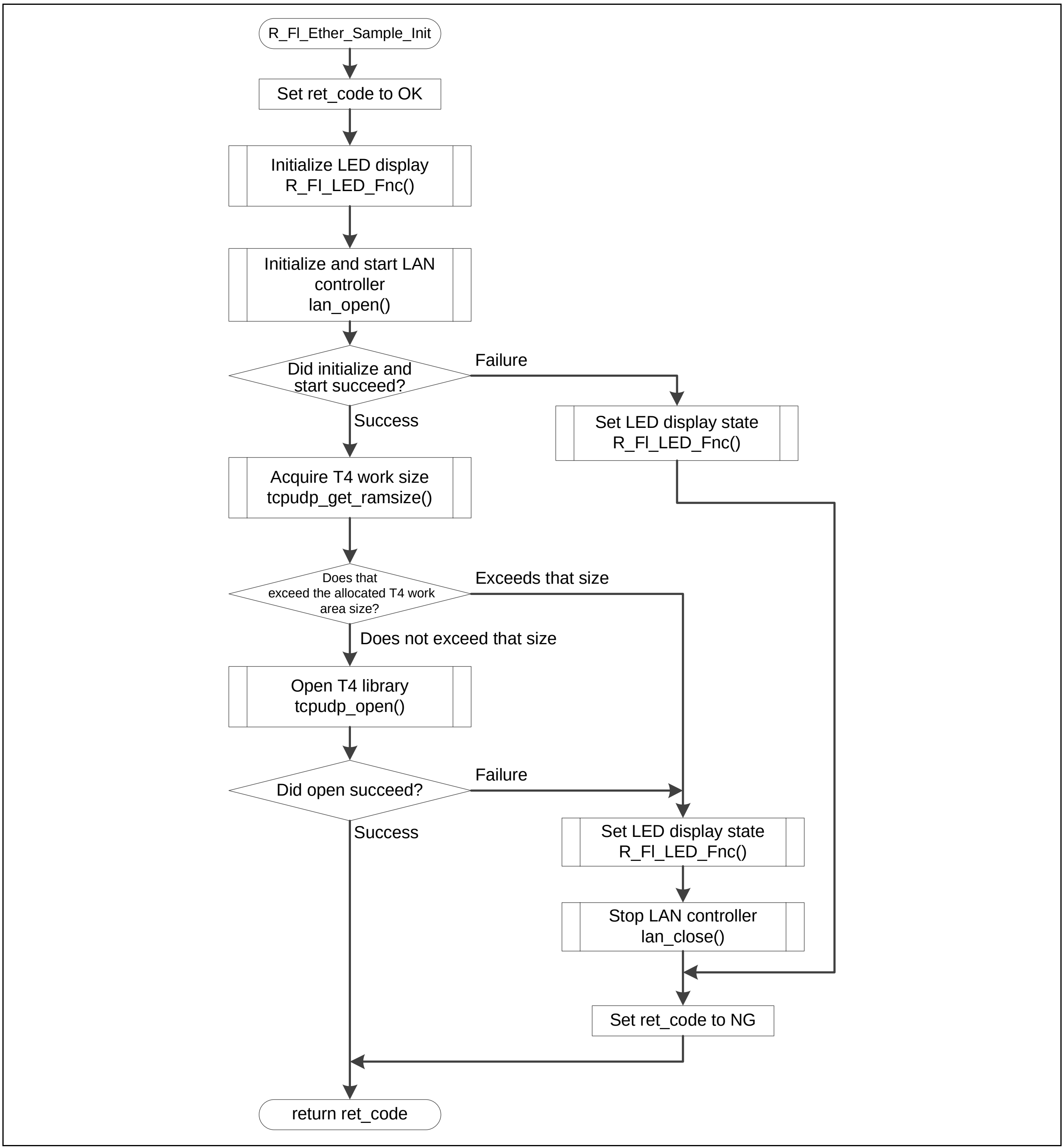

Figure 5.15 Ethernet Initialization Processing

5.13.3 Ethernet Termination Processing

Figure 5.16 shows the flowchart for the Ethernet termination processing.

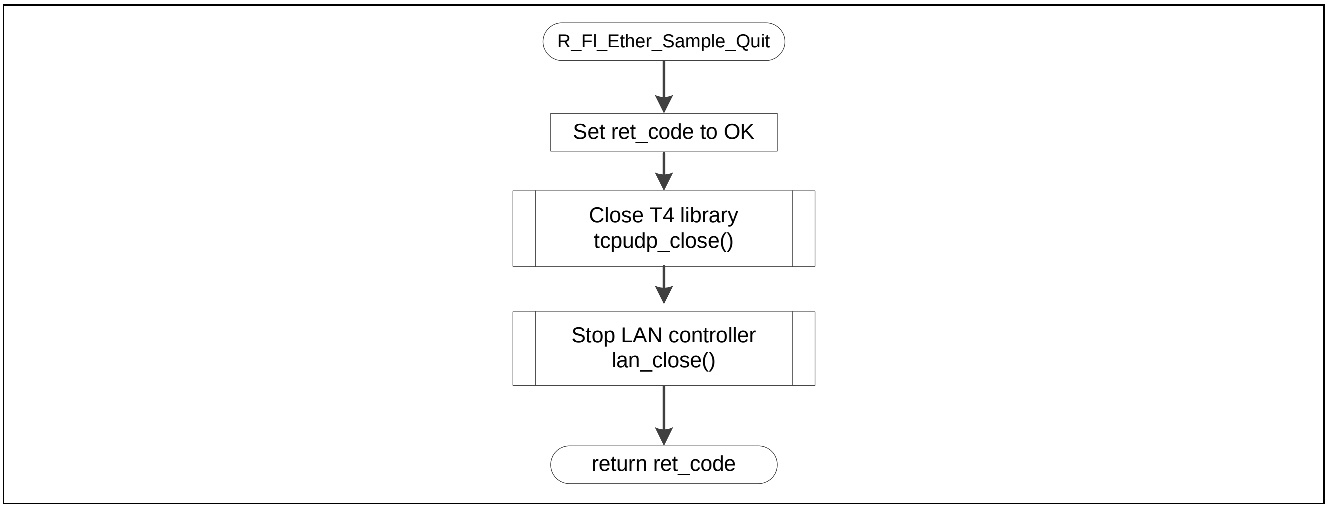

Figure 5.16 Ethernet Termination Processing

5.13.4 Main Write Processing

Figures 5.17 and 5.18 show the flowcharts for the main flash memory write processing.

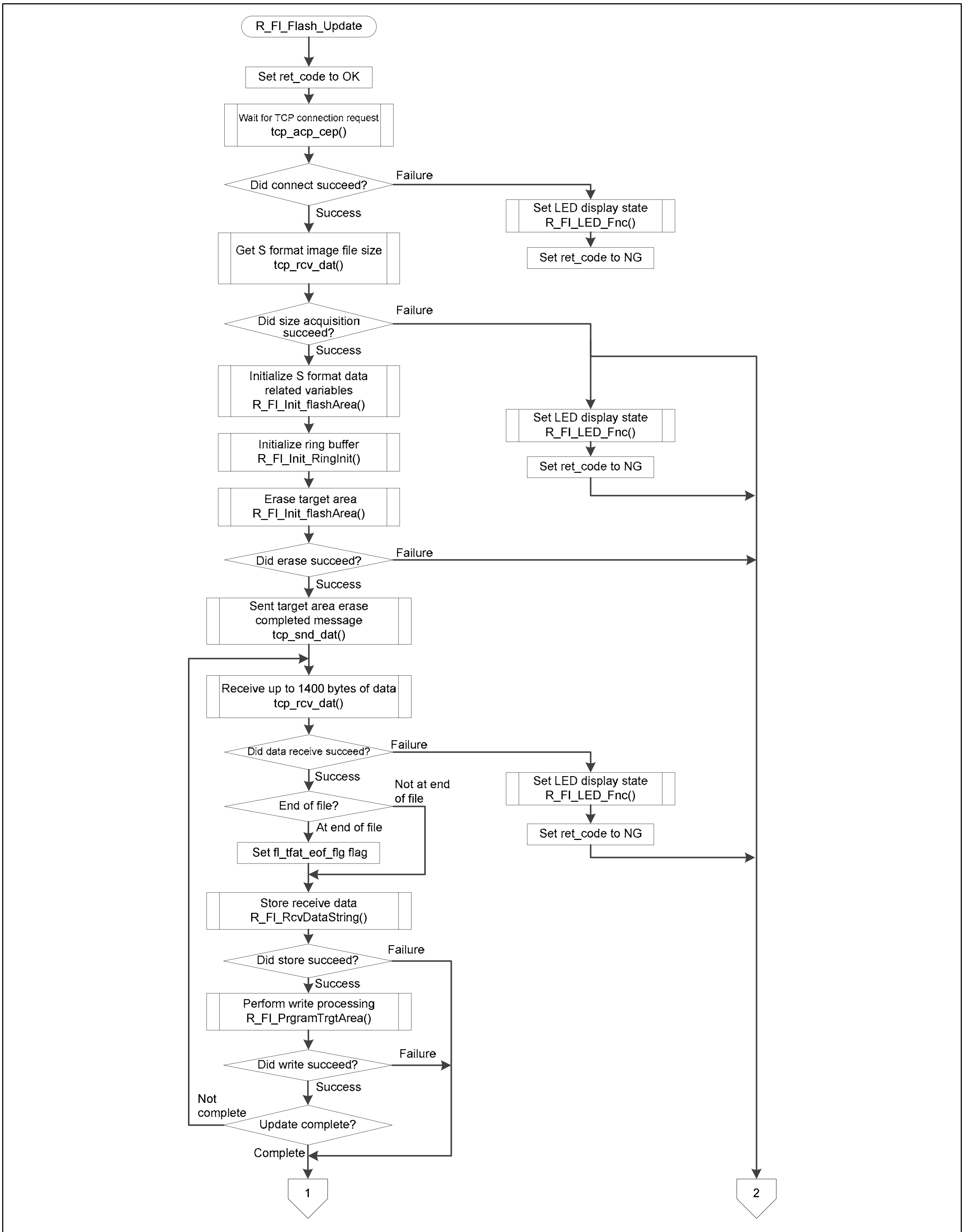

Figure 5.17 Main Write Processing

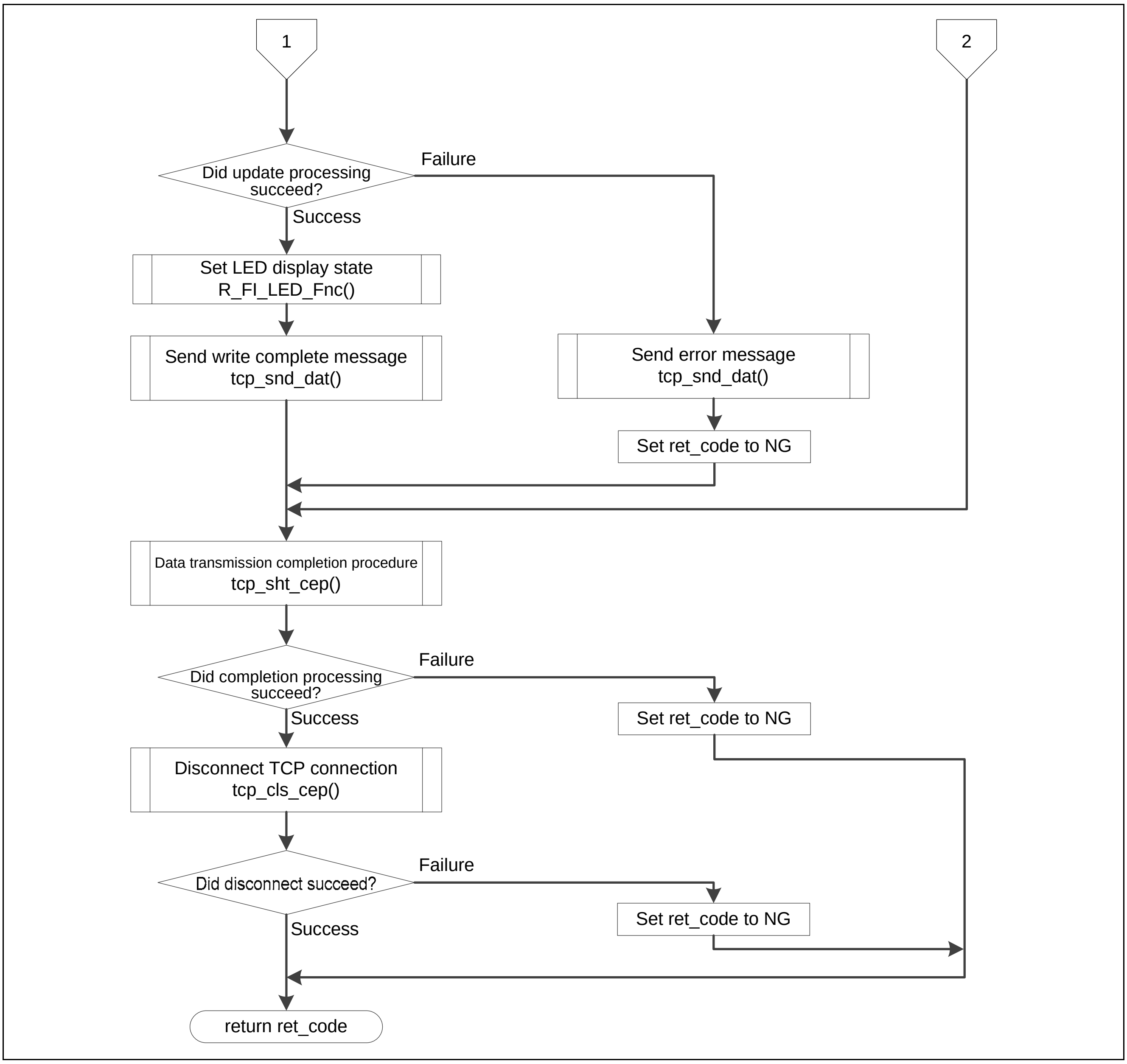

Figure 5.18 Main Write Processing (continued)

5.13.5 Erase Processing

Figure 5.19 shows the flowchart for the erase processing.

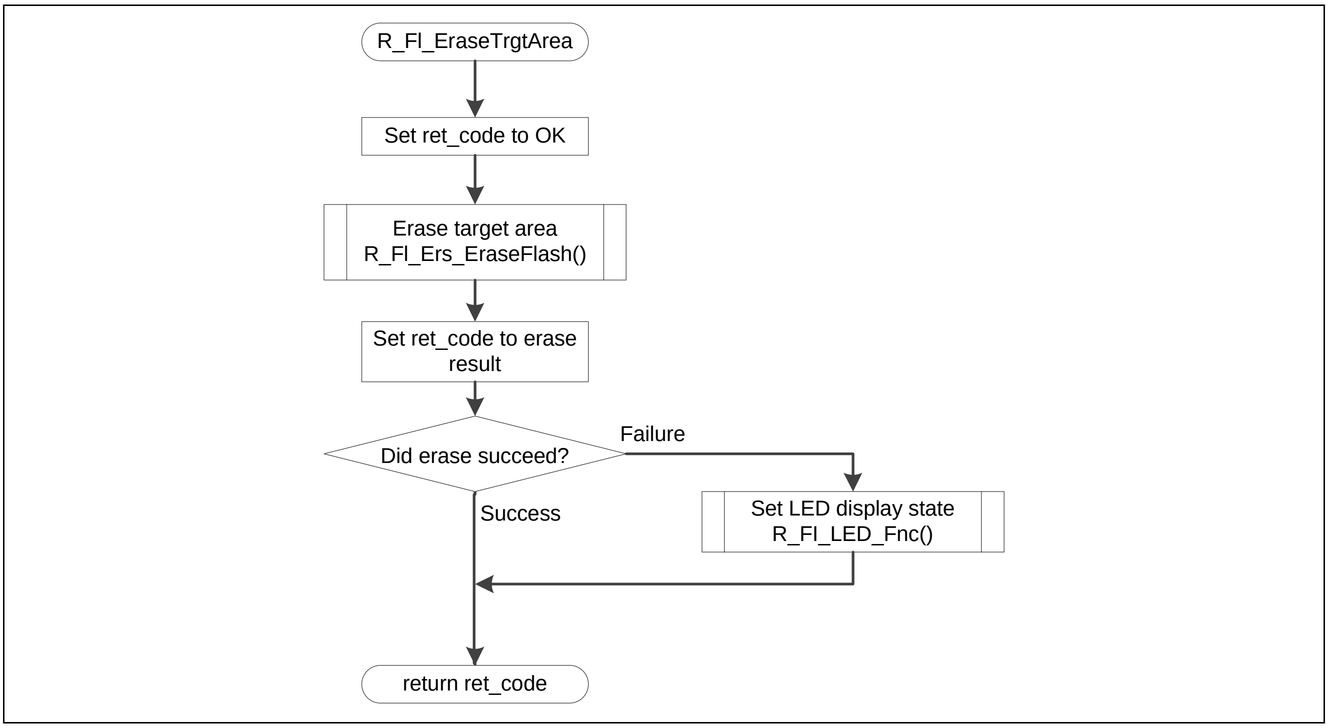

Figure 5.19 Erase Processing

5.13.6 Erase Download Area

Figure 5.20 shows the flowchart for the erase download area.

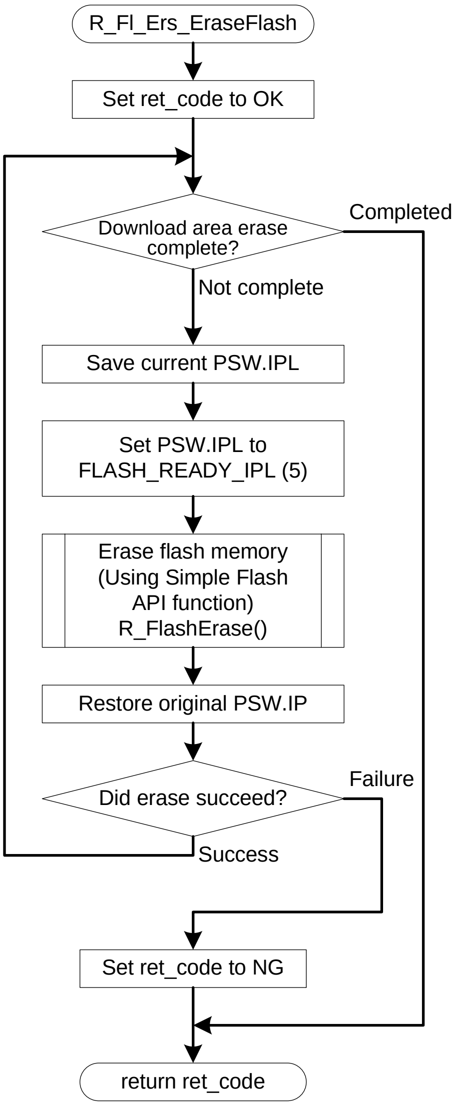

Figure 5.20 Erase Download Area

5.13.7 Write Processing

Figure 5.21 shows the flowchart for the write processing.

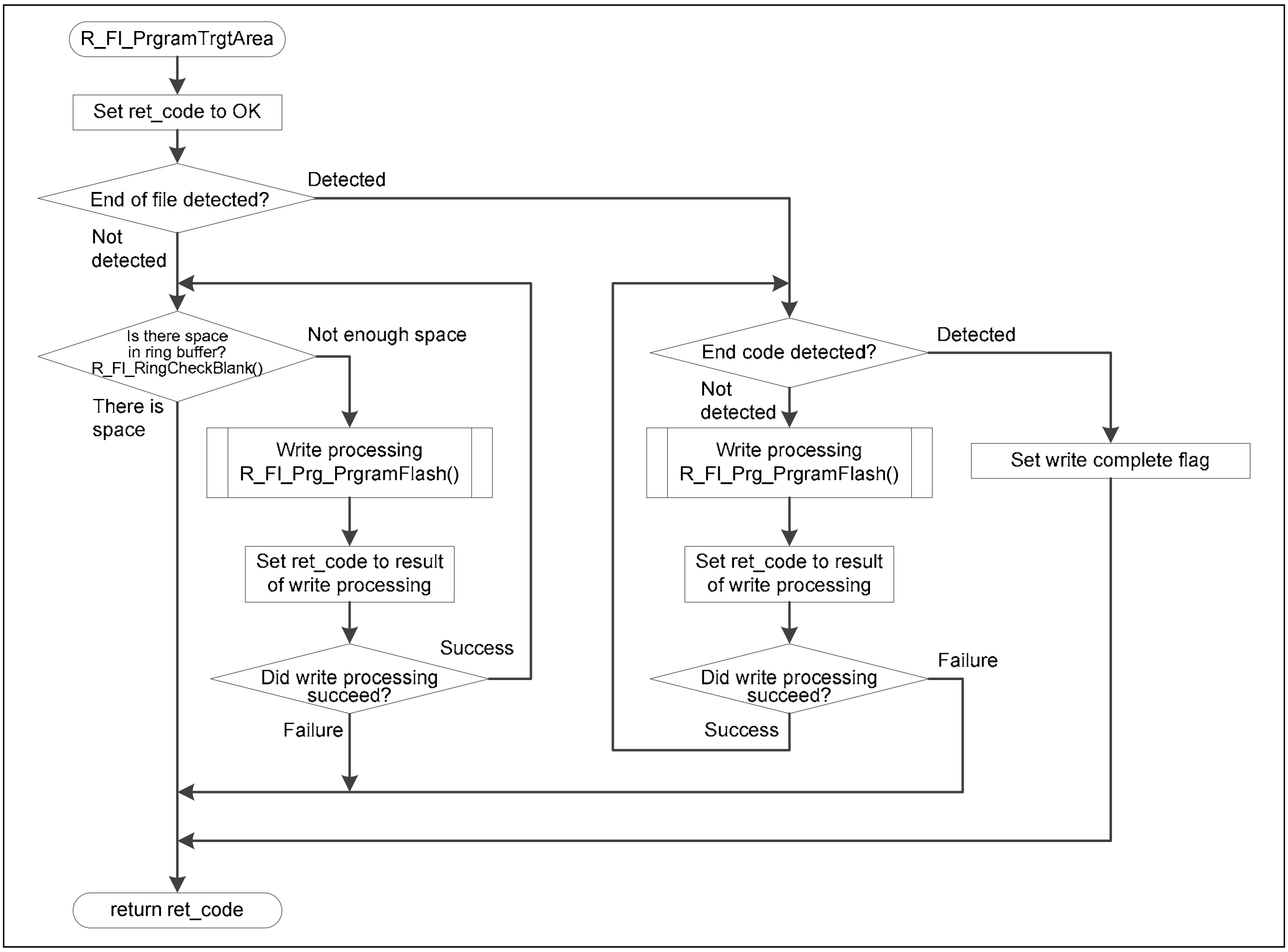

Figure 5.21 Write Processing

5.13.8 Download Area Write Operation

Figure 5.22 shows the flowchart for the download area write operation.

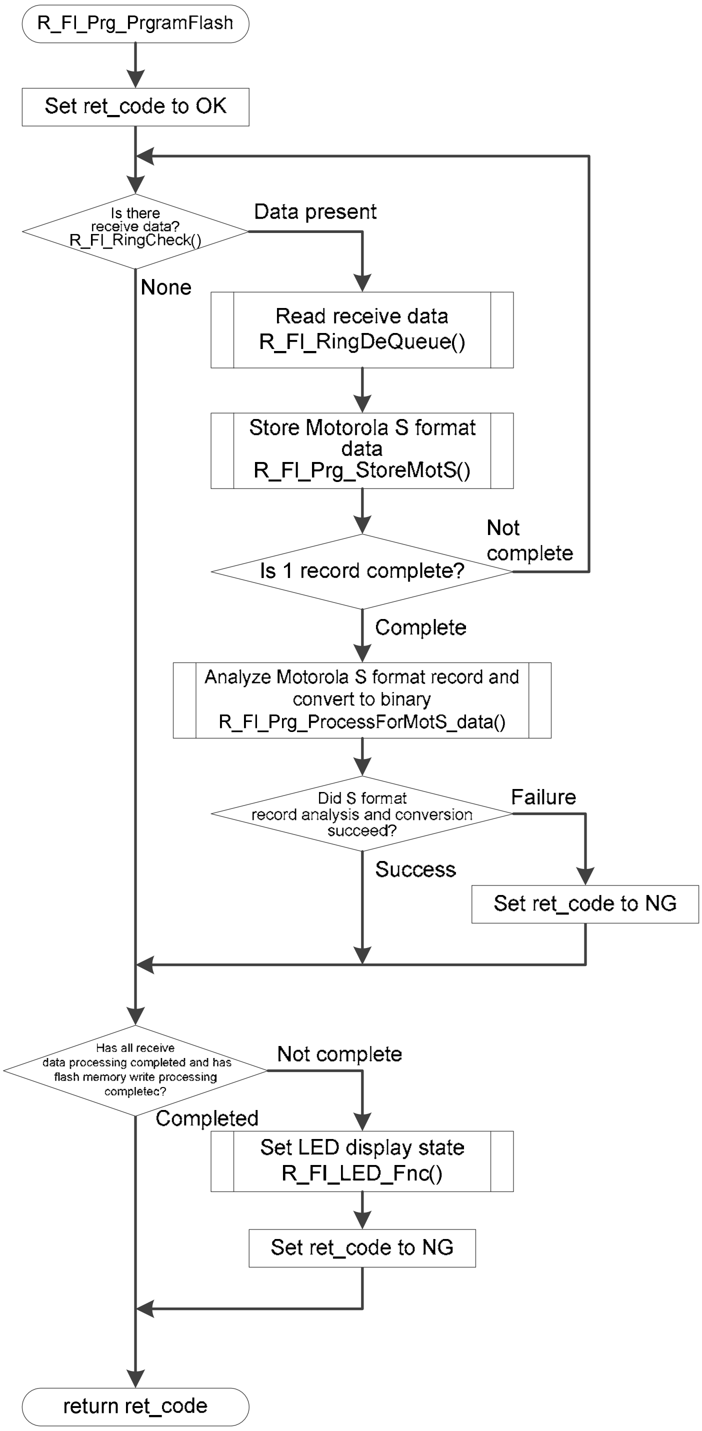

Figure 5.22 Download Area Write Operation

5.13.9 S Format Data Store Operation

Figure 5.23 shows the flowchart for the S format data store operation.

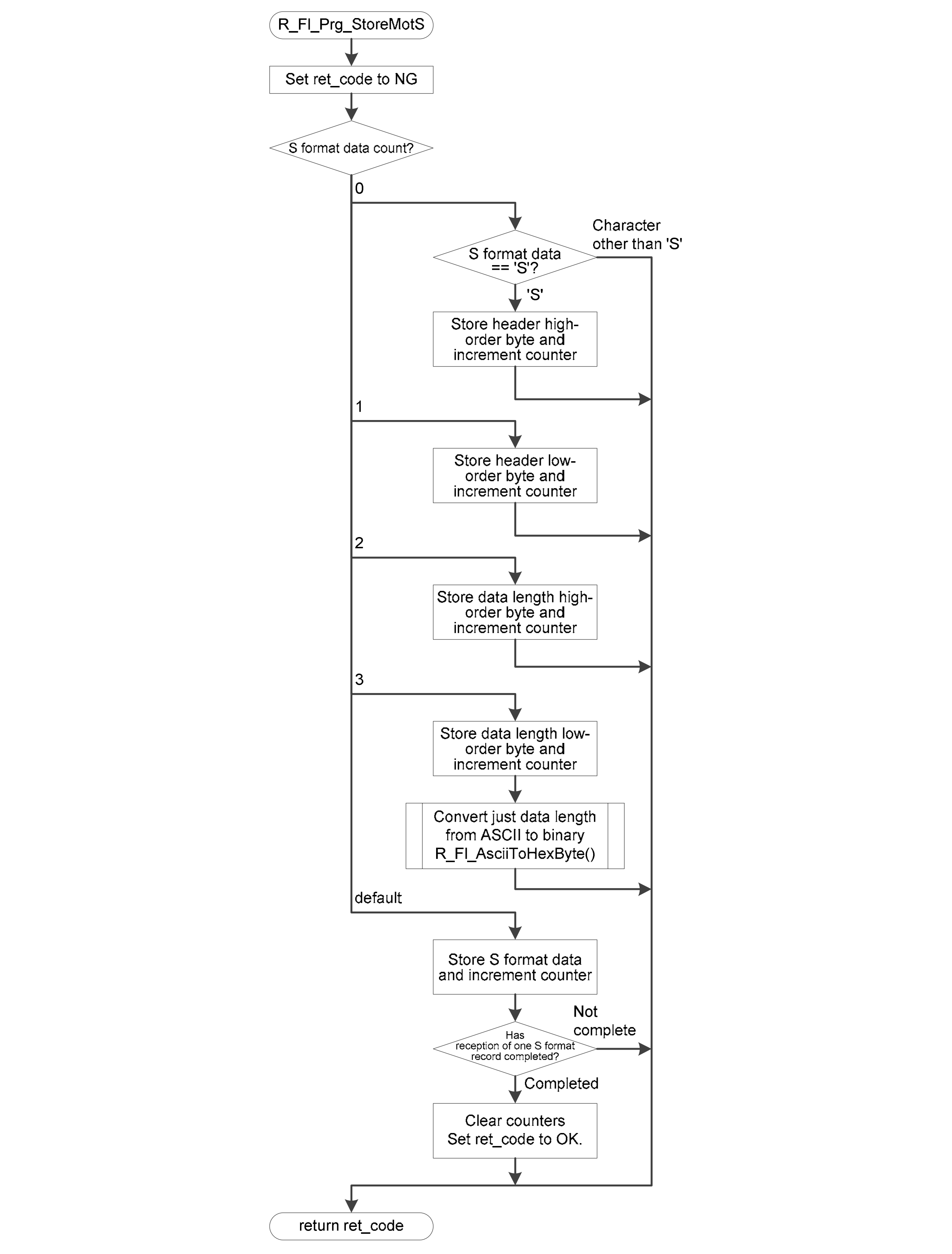

Figure 5.23 S Format Data Store Operation

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

Figure 5.24 shows the flowchart for the S format header analysis, conversion to binary, and write operations.

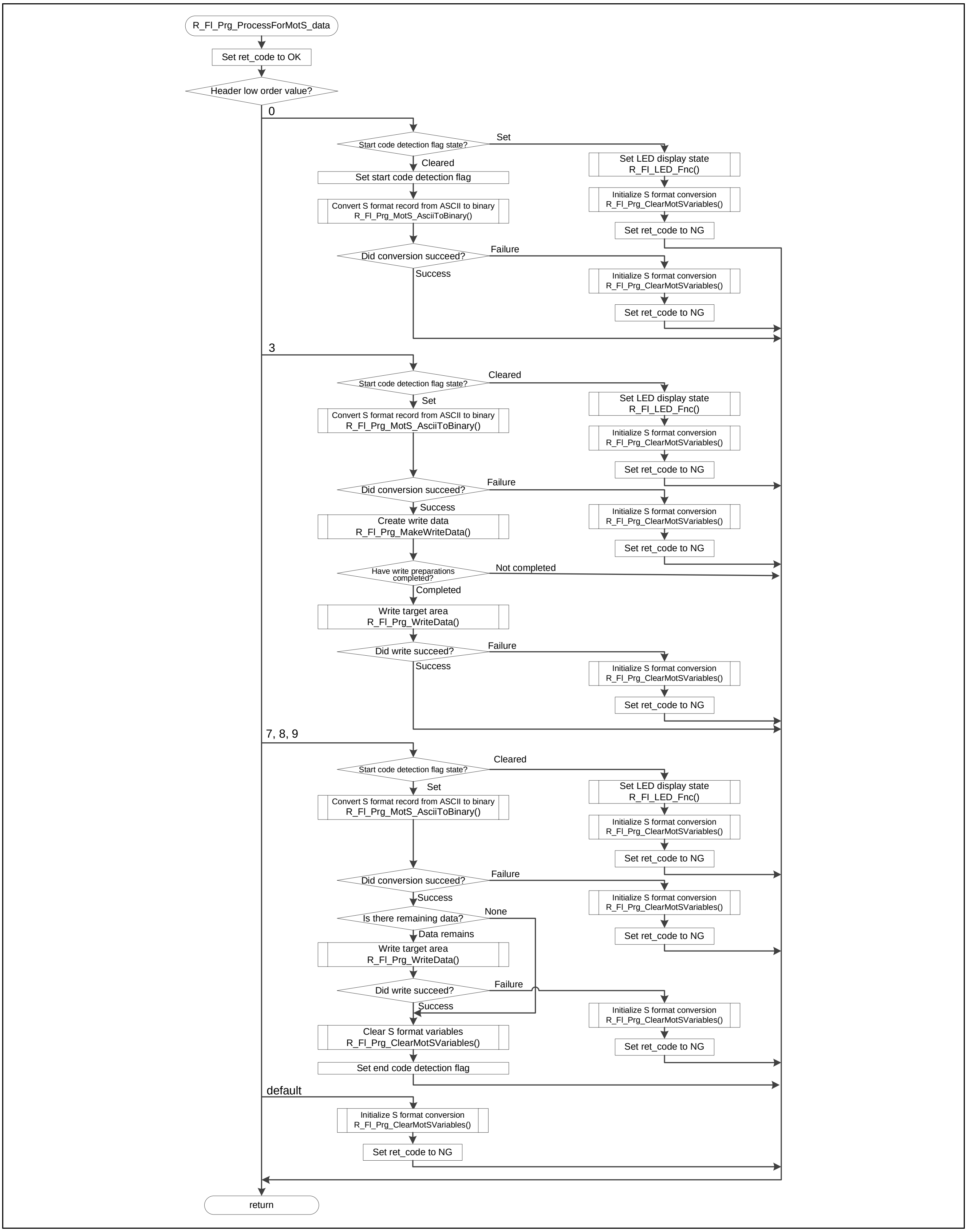

Figure 5.24 Format Header Analysis, Conversion to Binary, and Write Operations

5.13.11 S Format Data ASCII to Binary Conversion

Figure 5.25 shows the flowchart for the S format data ASCII to binary conversion.

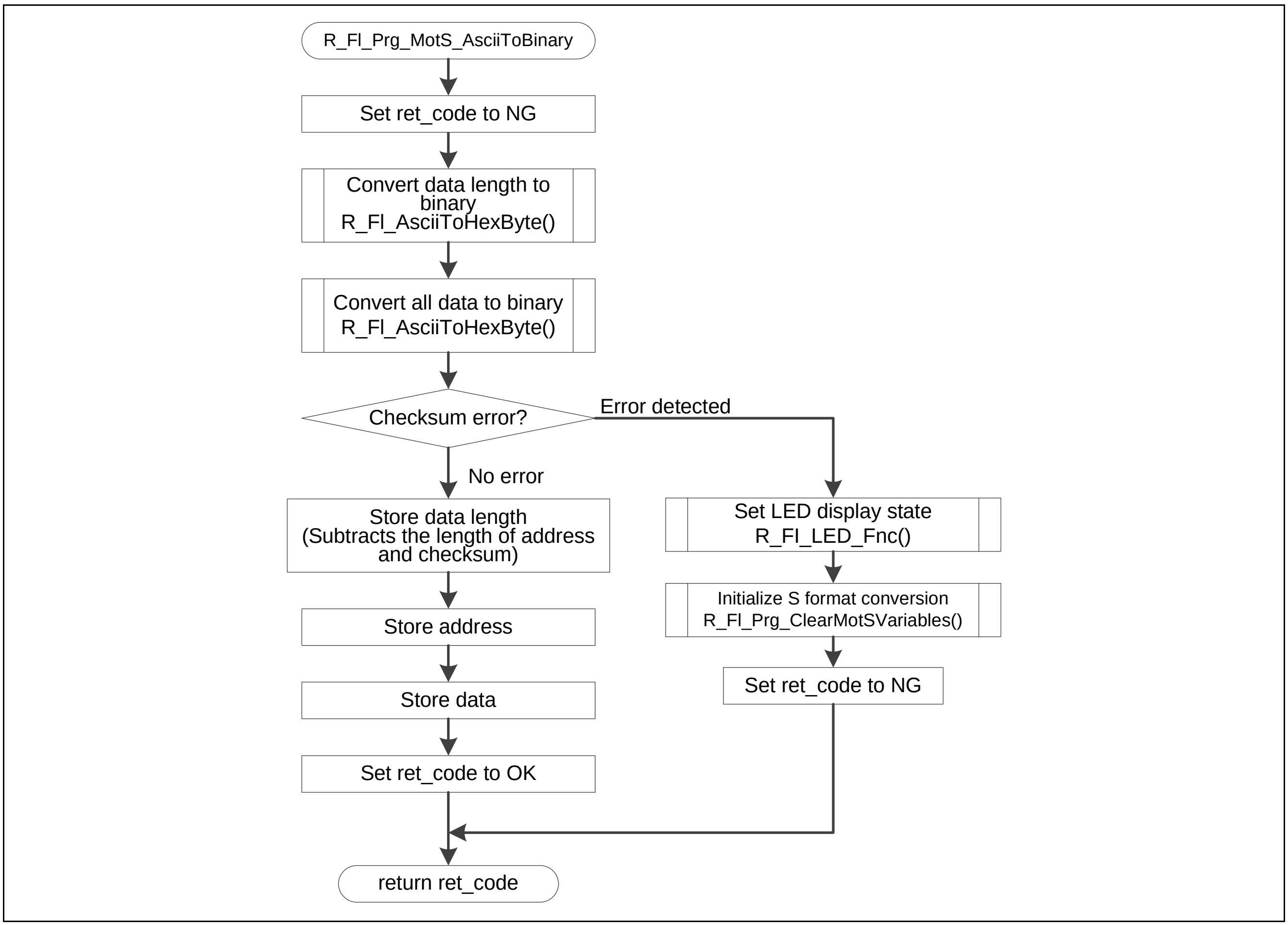

Figure 5.25 S Format Data ASCII to Binary Conversion

5.13.12 Download Area Write Data Creation

Figure 5.26 shows the flowchart for the download area write data creation.

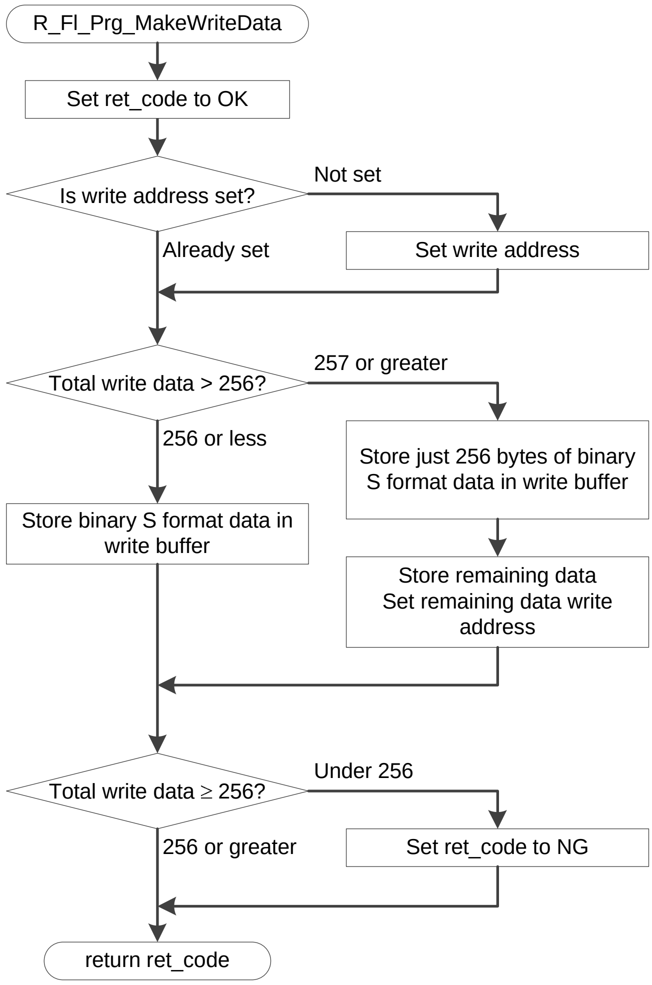

Figure 5.26 Download Area Write Data Creation

5.13.13 Download Area Write

Figure 5.27 shows the flowchart for the download area write.

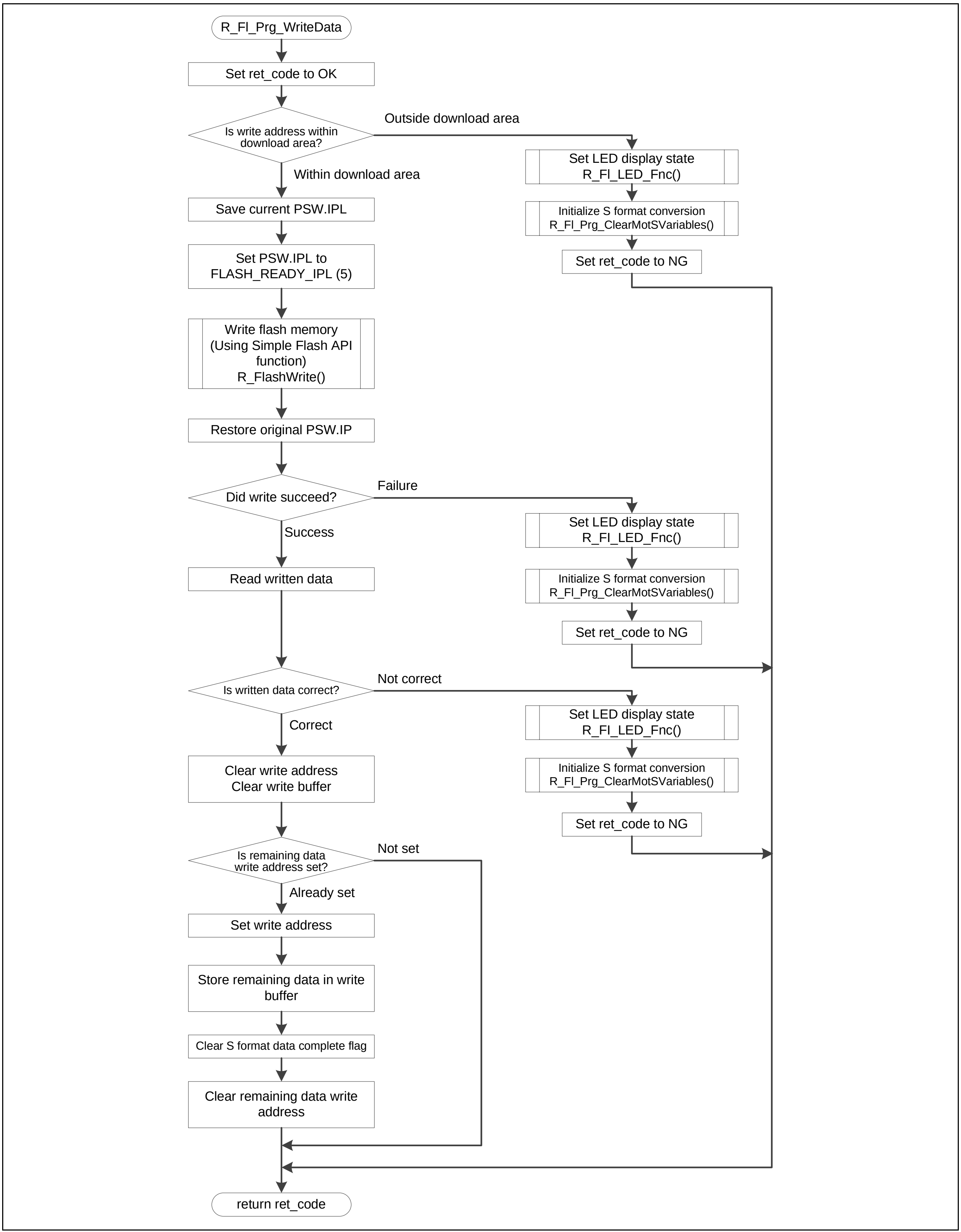

Figure 5.27 Download Area Write

5.13.14 Clear S Format Data Related Variables

Figure 5.28 shows the flowchart for the clear S format data related variables.

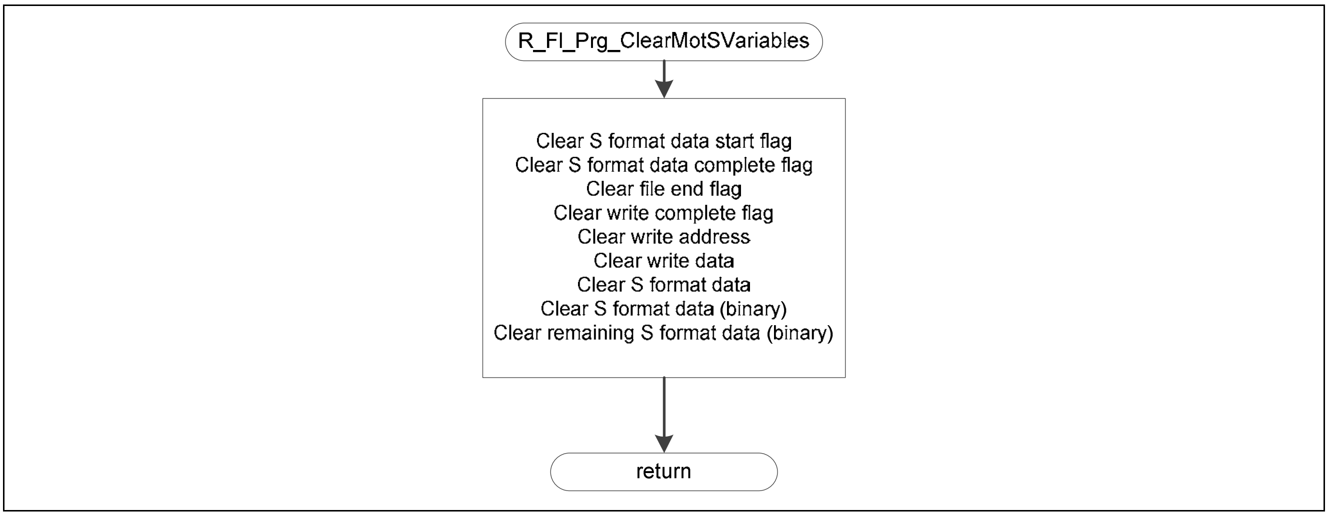

Figure 5.28 Clear S Format Data Related Variables

5.13.15 Store Ethernet Receive Data

Figure 5.29 shows the flowchart for the store Ethernet receive data.

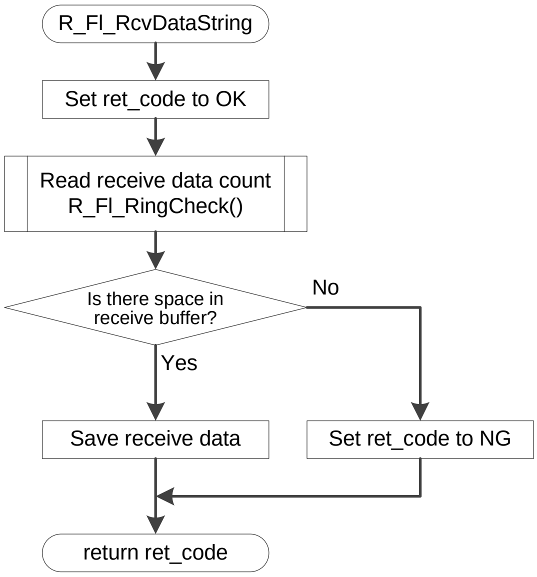

Figure 5.29 Store Ethernet Receive Data

5.13.16 Check for Empty Space in Ethernet Receive Data Storage Ring Buffer

Figure 5.30 shows the flowchart for the check for empty space in Ethernet receive data storage ring buffer.

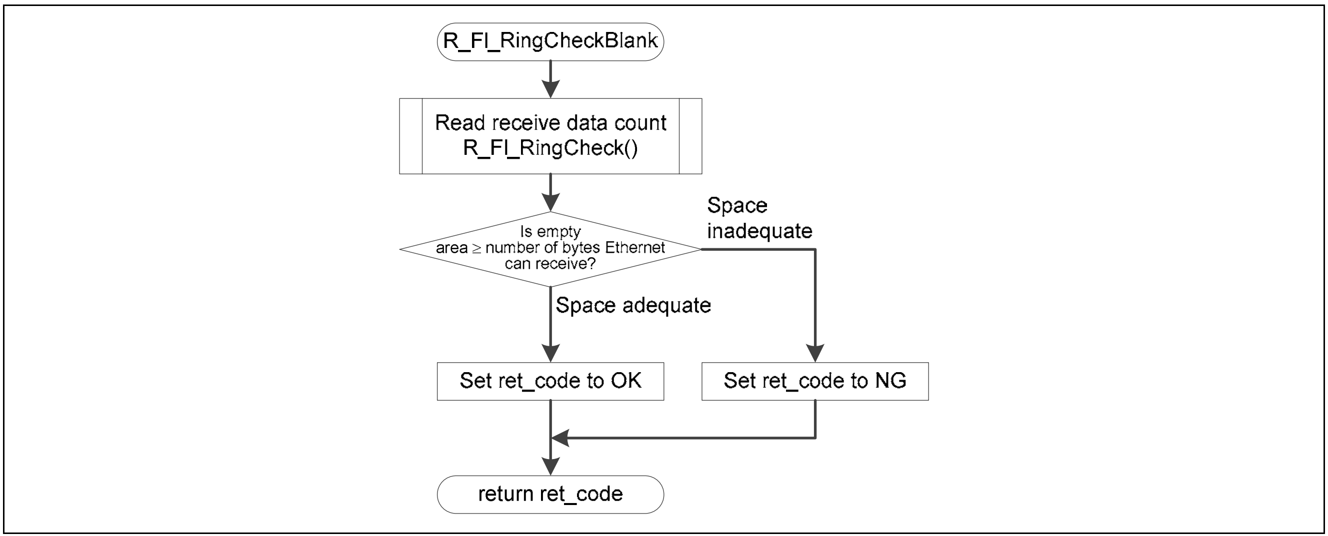

Figure 5.30 Check for Empty Space in Ethernet Receive Data Storage Ring Buffer

5.13.17 Initialize Ethernet Receive Data Storage Ring Buffer

Figure 5.31 shows the flowchart for the initialize Ethernet receive data storage ring buffer.

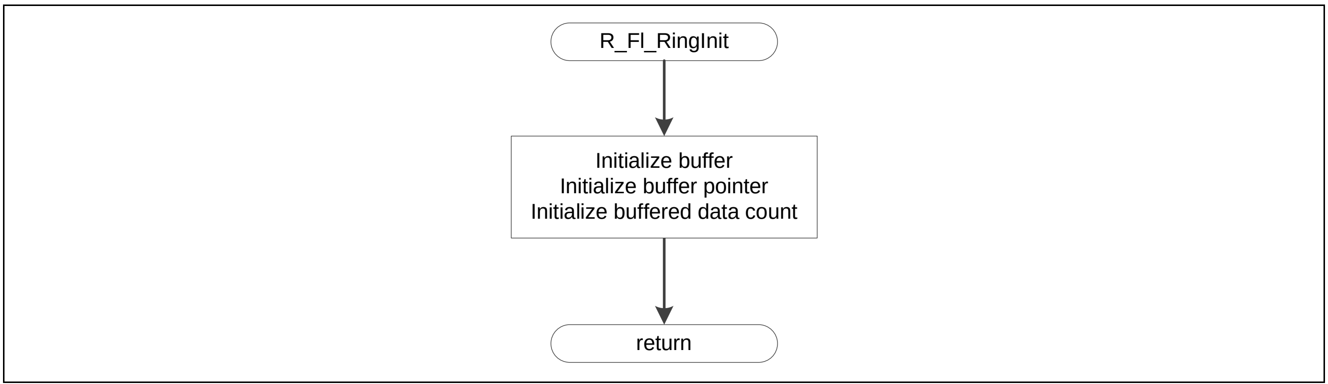

Figure 5.31 Initialize Ethernet Receive Data Storage Ring Buffer

5.13.18 Store Data in Ethernet Receive Data Ring Buffer

Figure 5.32 shows the flowchart for the store data in Ethernet receive data ring buffer.

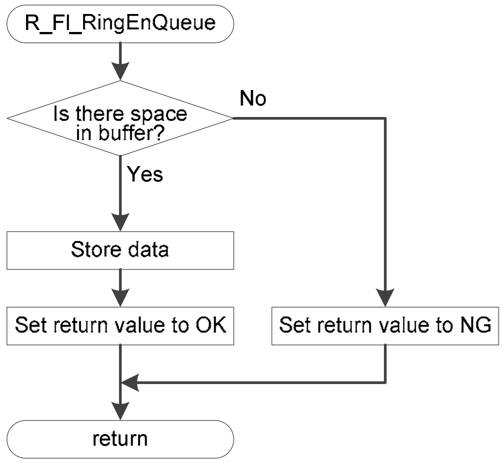

Figure 5.32 Store Data in Ethernet Receive Data Ring Buffer

5.13.19 Read Data from Ethernet Receive Data Ring Buffer

Figure 5.33 shows the flowchart for the read data from Ethernet receive data ring buffer.

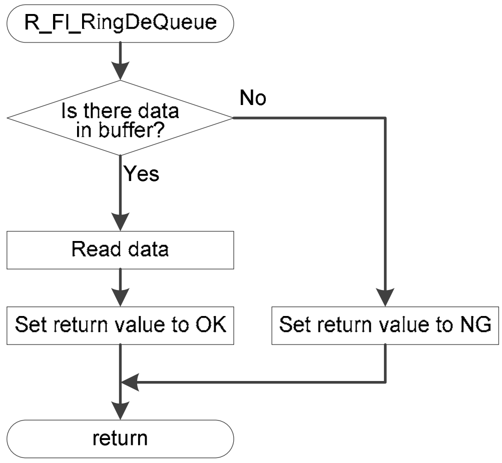

Figure 5.33 Read Data from Ethernet Receive Data Ring Buffer

5.13.20 Check Data Count in Ethernet Receive Data Ring Buffer

Figure 5.34 shows the flowchart for the check data count in Ethernet receive data ring buffer.

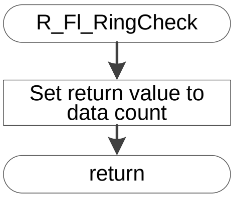

Figure 5.34 Check Data Count in Ethernet Receive Data Ring Buffer

5.13.21 Convert Data from ASCII to Binary

Figure 5.35 shows the flowchart for the convert data from ASCII to binary.

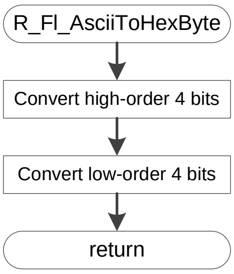

Figure 5.35 Convert Data from ASCII to Binary

5.13.22 LED Initialization Processing

Figure 5.36 shows the flowchart for the LED initialization processing.

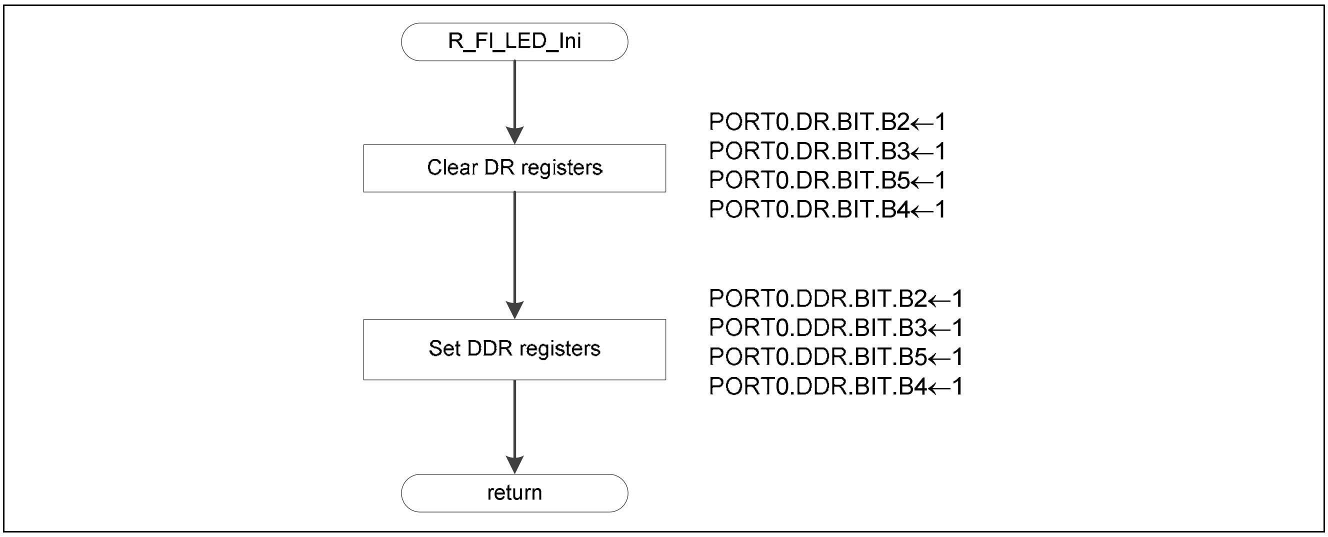

Figure 5.36 LED Initialization Processing

5.13.23 LED On/Off Processing

Figure 5.37 shows the flowchart for the LED On/Off processing.

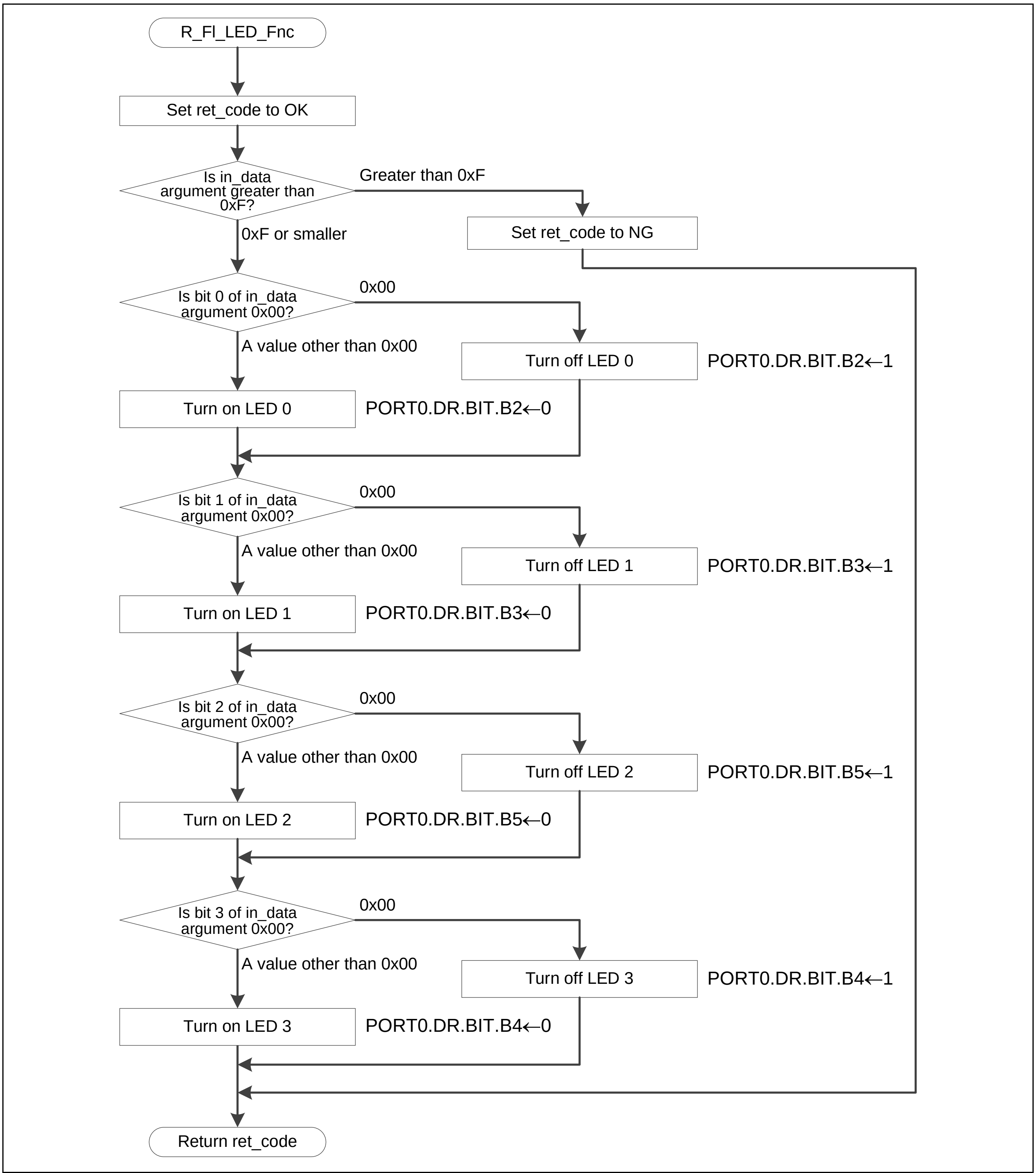

Figure 5.37 LED On/Off Processing

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, Confirmed Operating 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. Host PC Sample Program

This application note includes both the source code and executable file for a sample program that runs on the host PC (host_tool.zip).

The host PC sample program corresponds to the application layer in the TCP/IP model.

This program is a console application and is run by specifying the RX62N RSK IP address and port number and the file name of the S format file to be written. It attempts to connect to the specified IP address and port number and then transmits the data in the S format file to the RX62N RSK. Note that in this program, the maximum size of the S format file is set to be 2,048 KB.

This program uses the Winsock library to implement TCP data transfers.

8. S Format

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

8.1 Record Formats

Figure 8.1 shows the record formats supported by the sample code.

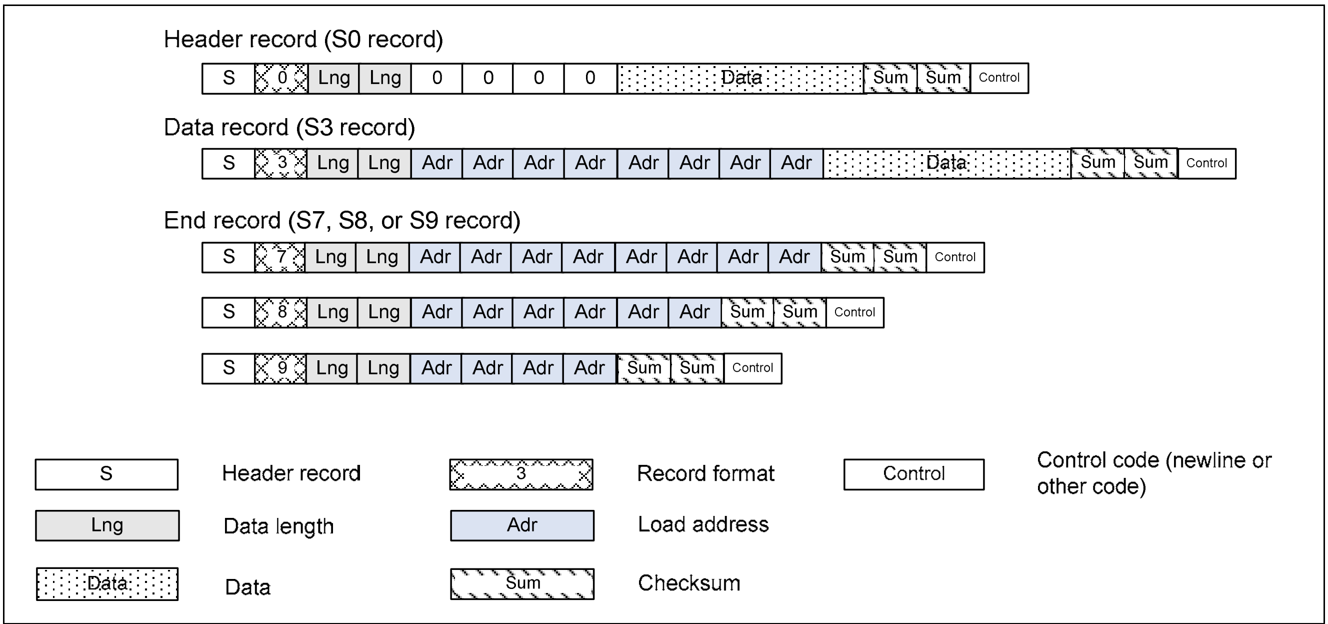

Figure 8.1 Record Formats Supported by Sample Code

8.2 Record Structure

Figure 8.2 shows the record structure supported by the sample code. S type format record sequences with orders other than those shown in figure 8.2 are not supported.

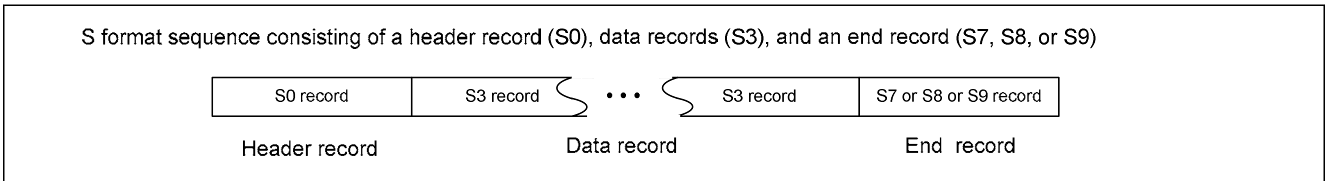

Figure 8.2 Record Structure Supported by the Sample Code

8.3 Load Address

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

8.4 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 8.3 shows the format error detection conditions.

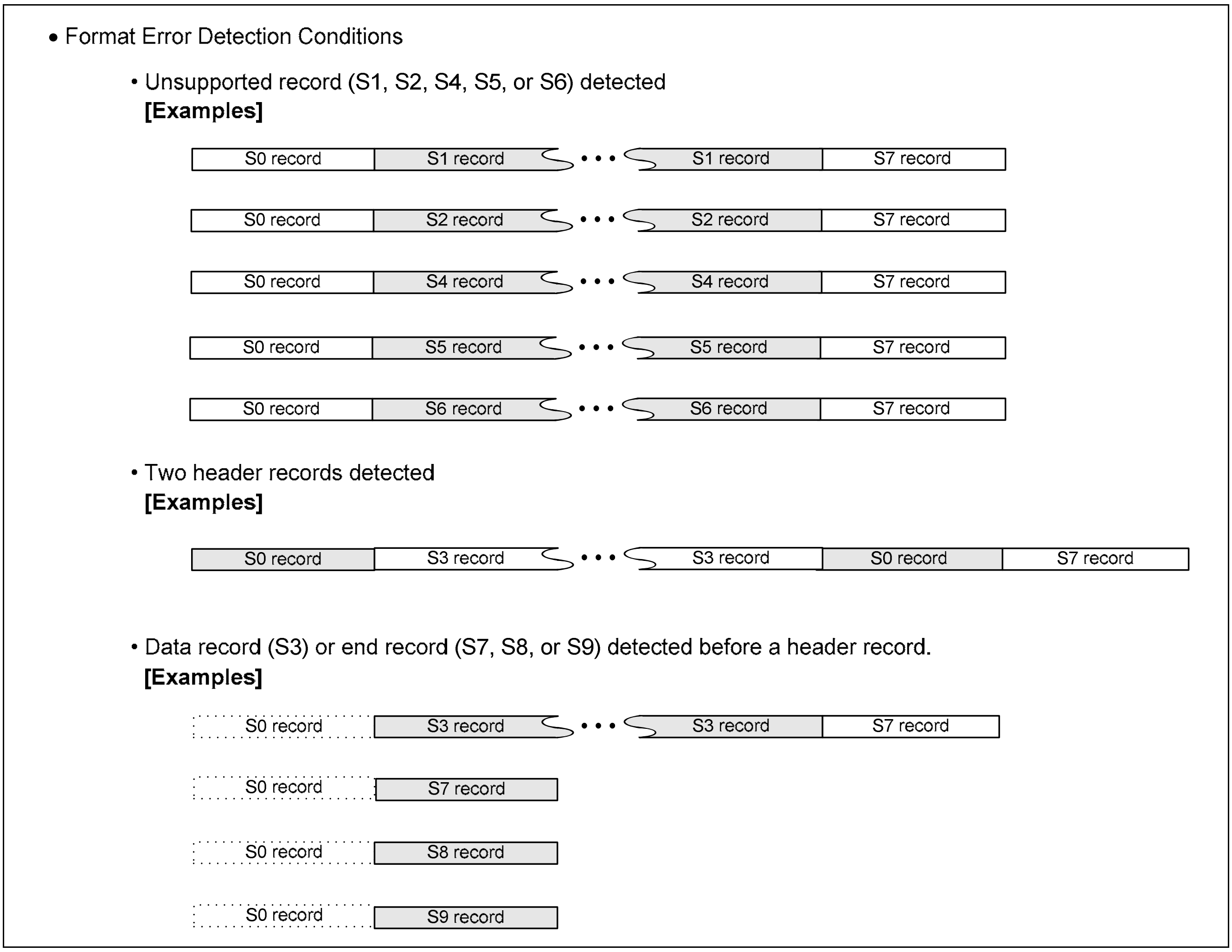

Figure 8.3 Format Error Detection Conditions

(c) Address error

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

9. Notes

9.1 Ethernet Cable Insertion/Removal During Erase or Write

Do not disconnect the Ethernet cable during erase or write of the download area.

9.2 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.

9.3 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 (FFFF 9FFCh). Therefore the download code must be set up so that its reset vector is allocated at FFFF 9FFCh. See section 5.3, Operation Overview, for details.

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

9.4 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 9.1 lists the ROM capacities.

Table 9.1 ROM Capacities

Catalog Number	ROM Capacity	Download Area ROM Capacity	Download Area Start Address	Download Area Block Numbers
R5F562x8	512 K	488 K	FFF8 0000h	EB6 to EB37
R5F562x7	384 K	360 K	FFFA 0000h	EB6 to EB29
R5F562x6	256 K	232 K	FFFC 0000h	EB6 to EB21

9.5 Operating Mode

The sample code only supports operation in single-chip mode.

Set Pin 1 and Pin 2 in SW4 on the RX62N RSK to OFF (MD0 = high, MD1 = high).

9.6 Endian Order

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

9.6.1 Using Little Endian

When operating using the little endian order, perform the following settings. Also, Pin 3 in SW4 on the RX62N RSK board must be set to ON (MDE = low).

1. Start up the RX Standard Toolchain from the Build item in the HEW toolbar.
2. Select Little-endian in the CPU endian tab and click OK.
3. In the HEW workspace window project tree, select "Exclude from build" in the context menu for the file T4_Library_rx600_ether_big.lib and select "Clear exclude from build:" in the context menu for the file T4_Library_rx600_ether_little.lib.
4. Rebuild the project.

9.6.2 Using Big Endian

When operating using the big endian order, perform the following settings. Also, Pin 3 in SW4 on the RX62N RSK board must be set to OFF (MDE = high).

1. Start up the RX Standard Toolchain from the Build item in the HEW toolbar.
2. Select Big-endian in the CPU endian tab and click OK.
3. In the HEW workspace window project tree, select "Exclude from build" in the context menu for the file T4_Library_rx600_ether_little.lib and select "Clear exclude from build:" in the context menu for the file T4_Library_rx600_ether_big.lib.
4. Rebuild the project.

9.7 Modifying the Host PC Sample Program

This application note includes both the source files and an executable file (host_tools.zip) for the host PC sample program.

The customer, however, must take responsibility for any modifications made to the host PC sample program.

The host PC sample program was developed using Microsoft Visual C++ 2010 (Microsoft Visual Studio 2010 Professional).

9.8 Changes to the RX Family RX600 Simple Flash API

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

9.8.1 Files Used

The files used are r_flash_api_rx600.c, r_flash_api_rx600.h, r_flash_api_rx600_private.h, r_flash_api_rx600_config.h and mcu_info.h.

9.8.2 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 stored in the Simple Flash API r_bsp/board/rskrx62n folder are used.
  (2) The following Simple Flash API settings are changed.
  Before change: #define BCLK_HZ (12000000)
  After change: #define BCLK_HZ (24000000)

9.9 Changes to the RX Family M3S-T4-Tiny: Introduction Guide

This application note uses sample code and library from the RX Family M3S-T4-Tiny: Introduction Guide. See the RX Family M3S-T4-Tiny: Introduction Guide application note for the RX Family M3S-T4-Tiny specifications.

9.9.1 Files Used

The files used are r_t4_itcpip.h, T4_Library_rx600_ether_big.lib, T4_Library_rx600_ether_little.lib, config_tcpudp.c, phy.c, phy.h, r_ether.c, r_ether.h, reg_accsess.h, t4_driver.c, timer.c and timer.h.

Note that the config_tcpdup.c T4 configuration file is used in the TCP Blocking Call file in the RX Family M3S-T4-Tiny: Introduction Guide project.

9.9.2 Changes

There are no changes to any of the files used.

9.10 Changes to HEW Generated Files

In this application note, certain parts of the HEW generated files are modified.

9.10.1 Changes

The following files are modified: dbsct.c, hwsetup.c, intprg.c, resetprg.c, stacksct.h, and vect.h.

- Changes to the file dbsct.c (2 places)
  (1) Addition of a section that performs the copy processing from the D section to the R section.
  Before change: { sectop("D_1"), secend("D_1"), sectop("R_1") }
  After change: { sectop("D_1"), secend("D_1"), sectop("R_1") },
  { sectop("PFRAM"), secend("PFRAM"), sectop("RPFRAM") }
  (2) Addition of a section that clears the B section to 0
  Before change: { sectop("B_1"), secend("B_1") }
  After change: { sectop("B_1"), secend("B_1") },
  { sectop("B_RX_DESC"), secend("B_RX_DESC") },
  { sectop("B_TX_DESC"), secend("B_TX_DESC") },
  { sectop("B_RX_BUFF_1"), secend("B_RX_BUFF_1") },
  { sectop("B_TX_BUFF_1"), secend("B_TX_BUFF_1") },
  { sectop("B_ETH_BUFF"), secend("B_ETH_BUFF") },
  { sectop("B_flash_api_sec"), secend("B_flash_api_sec") },
  { sectop("B_flash_api_sec_2"), secend("B_flash_api_sec_2") },
  { sectop("B_flash_api_sec_1"), secend("B_flash_api_sec_1") }

- Changes to the file hwsetup.c (4 places)
  (1) Addition of files to be included
  Before change: None
  After change: #include "r_ether.h"
  (2) System clock settings
  Before change: None
  After change: /* CPG setting */
  io_set_cpg();
  (3) I/O port settings
  Before change: None
  After change: /* Setup the port pins */
  ConfigurePortPins();
  (4) Clearing the module stop state for peripheral modules
  Before change: None
  After change: /* Enables peripherals */
  EnablePeripheralModules();

- Changes to the file intprg.c (1 place)
  (1) Changes to CMTU0_CMT0
  Before change: void Excep_CMTU0_CMT0(void){ }
  After change: void Excep_CMTU0_CMT0(void){ timer_interrupt(); }

- Changes to the file resetprg.c (1 place)
  (1) Addition of an operation selection processing function
  Before change: None
  After change: /* **** Mode entry **** */
  R_Fl_Mode_Entry();

- Changes to the file stacksct.h (1 place)
  (1) Modification of the stack size
  Before change: #pragma stacksize su=0x100
  After change: #pragma stacksize su=0x300

- Changes to the file vect.h (1 place)
  (1) Commenting out ETHER_EINT
  Before change: #pragma interrupt (Excep_ETHER_EINT(vect=32))
  void Excep_ETHER_EINT(void);
  After change://#pragma interrupt (Excep_ETHER_EINT(vect=32))
  //void Excep_ETHER_EINT(void);

9.10.2 Added Sections

Table 9.2 lists the added sections

Table 9.2 Added Sections

Section Name	Description
RPFRAM	Initialized data area (variables area) section for the flash memory programming code that operates in RAM
TRGT_DMMY_FIXEDVECT	Download code fixed vector section
B_RX_DESC	Uninitialized data area section for the Ethernet driver (receive descriptor)
B_TX_DESC	Uninitialized data area section for the Ethernet driver (transmit descriptor)
B_ETH_BUFF	Uninitialized data area section for Ethernet receive buffers and M3S-T4-Tiny working memory
B_ETH_BUFF_1	Uninitialized data area section for Ethernet receive buffers and M3S-T4-Tiny working memory (Align = 1)
B_RX_DESC_1	Uninitialized data area section for the Ethernet driver (receive descriptor) (Align = 1)
B_TX_DESC_1	Uninitialized data area section for the Ethernet driver (transmit descriptor) (Align = 1)
B_flash_api_sec	Uninitialized data area section for flash memory programming code
B_flash_api_sec_2	Uninitialized data area section for flash memory programming code (Align = 2)
B_flash_api_sec_1	Uninitialized data area section for flash memory programming code (Align = 1)

9.10.3 Include File Directory

The following include file directory settings are added.

- $(PROJDIR)\src\bsp is added to the include directories.
- $(PROJDIR)\src\FlashAPI is added to the include directories.
- $(PROJDIR)\src\driver is added to the include directories.
- $(PROJDIR)\src\t4\lib is added to the include directories.
- $(PROJDIR)\src\user_app is added to the include directories.

9.10.4 Linker Settings

The following linker setting that maps from ROM to RAM is added.

- ROM PFRAM is mapped to RPFRAM.

10. Sample Programs

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

11. Reference Documents

- RX62N Group, RX621 Group User's Manual: Hardware, Rev.1.20
  (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 Release 00
  RX Family C/C++ Compiler Package User's Manual V.1.0.1.0
  (The latest version can be downloaded from the Renesas Electronics Web site.)
- Application Notes
  RX600 Series Simple Flash API for RX600 Rev2.20 (R01AN0544EU)
  (The latest version can be downloaded from the Renesas Electronics Web site.)
  RX Family TCP/IP for Embedded system M3S-T4-Tiny: Introduction Guide Rev1.02 (R20AN0051EJ0102)
  (The latest version can be downloaded from the Renesas Electronics Web site.)

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Mar.13.12	—	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.

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: +44-1628-585-100, Fax: +44-1628-585-900

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.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

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 1.1