RZ/T1 Group

Example of Downloading to NOR Flash Memory by Using “Semihosting” of ARM® Development Studio 5 (DS-5TM)

R01AN2950EJ0120
Rev.1.20
Sep. 15, 2017

Summary

This application note presents a method for downloading programs to the NOR flash memory allocated to the external address space (CS0 space) of an RZ/T1 group microcontroller.

Note that the method of downloading described in this application note utilizes the “semihosting” (file operation) functionality of ARM® Development Studio 5 (DS-5TM, hereafter abbreviated as DS-5). You will need to obtain DS-5 separately. For details of the semihosting functionality of DS-5, refer to the documentation*1 provided by ARM®.

Note 1. Refer to “ARM® Compiler toolchain Developing Software for ARM® Processors, Semihosting” for details.

Applicable Devices

RZ/T1 Group

When applying the program covered in this application note to another microcontroller, modify the program according to the specifications for the target microcontroller and conduct an extensive evaluation of the modified program.

Table of Contents

1. Specifications 4

2. Conditions for Checking Operations 5

3. Related Application Notes 6

4. Description of Hardware 7

4.1 List of Pins 7

4.2 Reference Circuit 8

5. Outline of Downloading to the NOR Flash Memory 9

5.1 Terms Related to Downloading to the NOR Flash Memory 9

5.2 Schematic View of Flash Downloader Operation 10

5.3 Developing a Flash Downloader 11

5.3.1 Memory Map 12

5.4 Customizing the Examples of Downloading to the NOR Flash Memory 13

6. Example of Downloading to the RZ/T1 Evaluation Board (RTK7910022C00000BR) 14

6.1 Settings for the RZ/T1 Evaluation Board (RTK7910022C00000BR) 15

6.2 Copying DS-5 Scripts 15

6.3 Importing and Building Projects 16

6.4 Generating Application Binary Files 17

6.5 Copying the Flash Downloader Executable File 18

6.6 DS-5 Debug Configuration Settings 18

6.7 Connecting the RZ/T1 Evaluation Board with an ARM® Emulator 19

6.8 Running the Download Script 20

7. Flash Memory Interface Functions 21

7.1 Fixed-Width Integers 21

7.2 Constants 21

7.3 Variables 24

7.4 Flash Memory Interface Functions 24

7.5 Details of the Flash Memory Interface Functions 25

7.6 Flowcharts of Flash Memory Interface Functions 26

7.6.1 Initialization Interface Function 26

7.6.2 Write Interface Function 27

8. Operation of the Flash Downloader 32

8.1 Memory Allocation of the Application Program 32

8.2 Flow of Flash Downloader Processing 33

8.2.1 Calculating the Checksum of the Loader Parameter Information 36

9. Configuration of the Flash Downloader 37

9.1 Configuration of Projects 37

9.2 RZ/T1 Evaluation Board Initialization Script 38

9.3 Application Downloading Script 39

10. Application Examples 40
10.1 Changing the Binary File Names and Destination Addresses for Writing 40
10.1.1 Changing the Binary File Names for Writing to the Flash Memory 40
10.1.2 Changing the Destination Addresses for Writing to the Flash Memory 42
10.2 Customizing the Flash Memory Interface Functions to Suit the Given Flash Memory 43
10.2.1 Specifications of the Device for the Sample Program 43
10.2.2 Boot Types of Flash Memory Available through Customization 44
10.2.3 Details of Customization 46
10.2.4 Customizing the Sector Size and Number of Sectors of Uniform Type 48
10.2.5 Customizing the Boot Type of Flash Memory to Bottom-Boot Type 49
10.2.6 Customizing the Boot Type of Flash Memory to Top-Boot Type 50
10.2.7 Customizing the Boot Type of Flash Memory to Dual-Boot Type 51
10.2.8 Checking Flash Memory Commands 52
10.3 Customizing the Sample Program for Initial Settings of the Microcomputers Incorporating the R-IN Engine (Cortex-M3) 53
11. Sample Program 55
12. Documents for Reference 56

1. Specifications

The NOR flash memory is a type of nonvolatile memory typically used to store program codes and data. Writing to the NOR flash memory requires an appropriate algorithm for the flash memory in use. This application note presents such an algorithm as a C-language program that runs in the tightly-coupled memory (specifically, the ATCM) of an RZ/A1H group microcontroller. It also describes how to use the semihosting functionality of DS-5 to refer to application binary files which are stored on the hard disk of the host computer on which DS-5 is running, and to write them to the NOR flash memory.

Table 1.1 lists the peripheral modules used and their applications.

Note: See Table 5.1 for details of application binary files.

Table 1.1 Peripheral Modules and Their Applications

Peripheral Module	Application
Bus state controller (BSC)	• This is used to generate signals for use in access to the NOR flash memory connected to the external address space (CS0 space). • The sample program in this application note makes settings for access to the NOR flash memory on the RZ/T1 evaluation board (RTK7910022C00000BR).*1
ARM® Development Studio 5 (DS-5TM) “semihosting” functionality	• Semihosting is used to have code running on the target (the program running on the board) handle transfer to and from the I/O functions of the host computer on which the debugger is running. • This is used to refer to the terminal output from the target to the application console of DS-5 and to the handling of application binary files stored on the hard disk of the host computer.

Note 1. See Table 2.1 for the product type name of the NOR flash memory on the RZ/T1 evaluation board.

2. Conditions for Checking Operations

Operation of the sample program covered in this application note has been confirmed under the conditions below.

Table 2.1 Conditions for Checking Operations

Item	Description
MCU used	RZ/T1 Group
Operating frequency	CPUCLK = 450 MHz, CKIO = 50 MHz
Operating voltage	3.3 V
Integrated development environment	DS-5 Version 5.25.0 from ARM®
Operating mode	16-bit bus boot mode (NOR flash)
Board used	RZ/T1 evaluation board (RTK7910022C00000BR)
Devices used (functions to be used on the board)	NOR flash memory allocated to the CS0 space (16-bit bus width) • Manufacturer: Macronix International Co., Ltd. • Product type number: MX29GL512FLT2I-10Q

3. Related Application Notes

The application notes related to the descriptions in this application note are listed below. Also consult the following documents along with this application note.

- RZ/T1 Group Initial Settings (R01AN2554EJ)
- RZ/T1 Group Initial Settings of the Microcomputers Incorporating the R-IN Engine (R01AN2989EJ)

4. Description of Hardware

4.1 List of Pins

Table 4.1 lists the pins used and their functions.

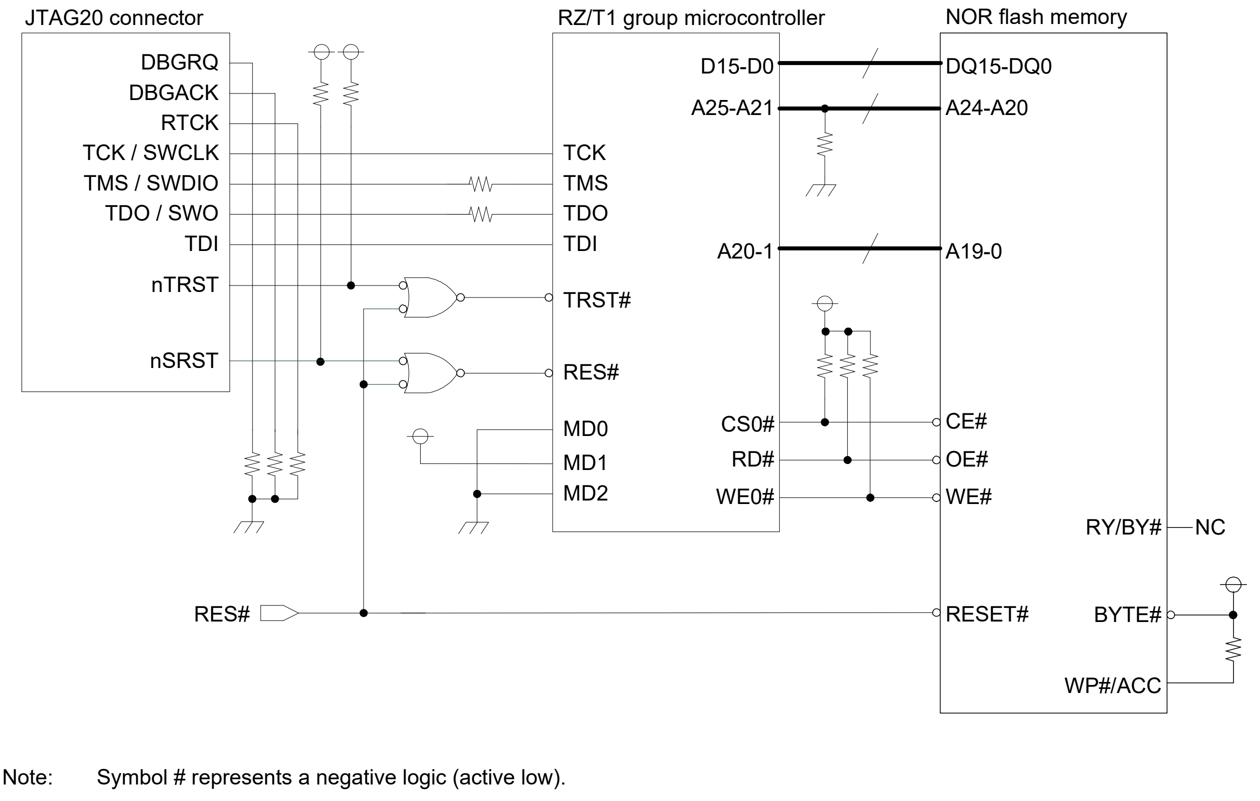
Table 4.1 Pins Used and Their Functions

Pin Name	I/O	Description
A25 to A1	Output	Output of address signals to the NOR flash memory
D15 to D0	I/O	Input/output of data signals of the NOR flash memory
CS0#	Output	Output of the device select signal to the NOR flash memory connected to the CS0 space
RD#	Output	Output of the read control signal to the NOR flash memory
WE0#	Output	Output of the write enable control signal to the NOR flash memory
MD2, MD1, MD0	Input	Selection of boot mode (set to 16-bit bus boot mode) MD2: “L” MD1: “H” MD0: “L”
TCK	Input	Clock input from the ARM® emulator
TMS	Input	Mode selection from the ARM® emulator
TRST#	Input	Reset input from the ARM® emulator
TDI	Input	Data input from the ARM® emulator
TDO	Output	Data output to the ARM® emulator
RES#	Input	System reset signal

Note: Symbol # represents a negative logic (or active low).

4.2 Reference Circuit

Figure 4.1 is a connection example.



Note: Symbol # represents a negative logic (active low).

Figure 4.1 Connection Example

5. Outline of Downloading to the NOR Flash Memory

This section gives an outline of downloading to the NOR flash memory.

5.1 Terms Related to Downloading to the NOR Flash Memory

Table 5.1 lists the terms related to downloading to the NOR flash memory that are used in this application note.

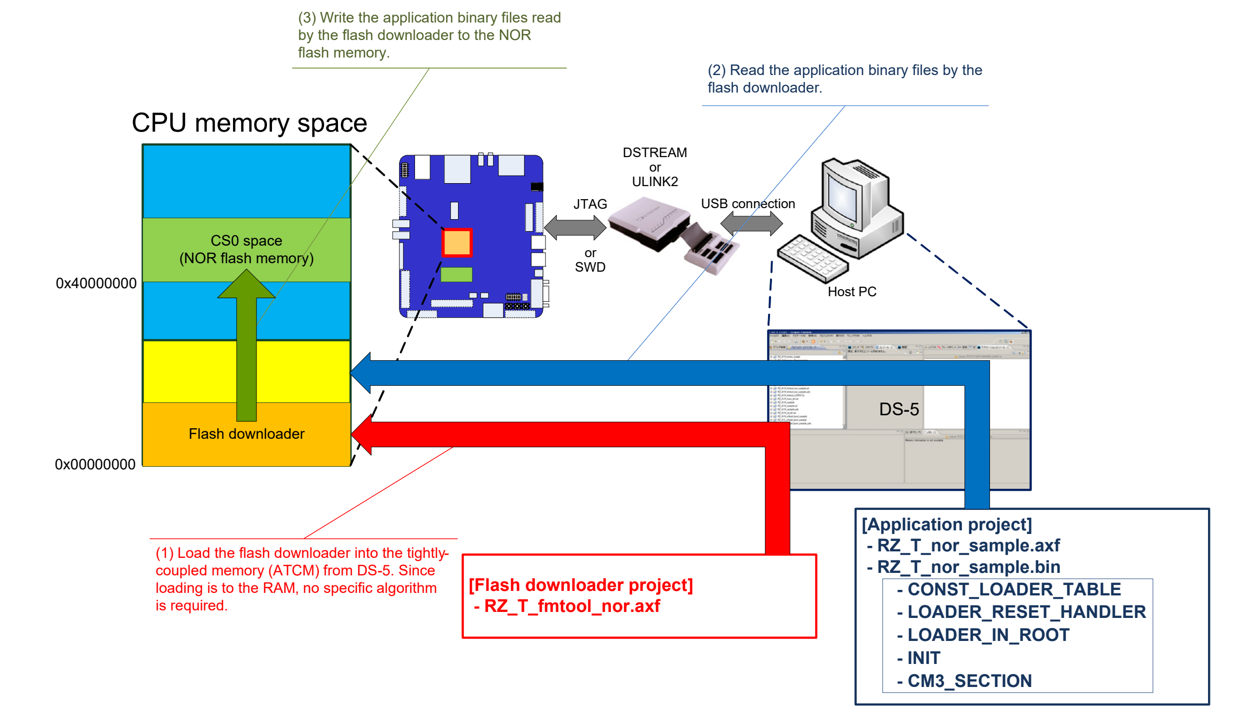
Table 5.1 Terms Related to Downloading to the NOR Flash Memory

Term	Description
Application program	The application program is a program which is created by the customer to suit the system.
Flash downloader	The flash downloader is a program for writing the application program to the NOR flash memory. Customers should use this application note for reference and create flash downloader programs to match their systems.
Semihosting	Semihosting is a mechanism where I/O request code running on an ARM® CPU uses the I/O functions of DS-5 through transfer to and from a debugger. Running standard C language functions such as printf, scanf, etc. on the ARM® CPU allows I/O processing on the screen and keyboard of the host PC through the I/O functions of DS-5 rather than through the I/O functions in the ARM® CPU on the target system. For details, see the document provided by ARM®.
Application project	This project is used to generate an executable file (axf file) of the application program in DS-5. The application program includes parameter information for the loader to be referred to by the RZ/T1 group microcontroller and the loader program itself.
Flash downloader project	This project is used to generate a flash downloader executable file (axf file) in DS-5. The application program includes parameter information for the loader to be referred to by the RZ/T1 group microcontroller and the loader program itself.
Application binary file	The application binary file is a data file containing the application program to be written to the NOR flash memory. A binary file generator tool (fromelf.exe)*1 is used to generate this file from the application program executable file (axf file) that is generated when the application project is built in DS-5.

Note 1. The binary file generator tool is included in DS-5. For details, see “ARM® DS-5TM DS-5 Getting Started Guide, ARM DS-5 Product Overview” provided by ARM®.

5.2 Schematic View of Flash Downloader Operation

Figure 5.1 is a schematic view of the operation of the flash downloader. The flash downloader runs in the tightly-coupled memory (ATCM) of the RZ/T1 group microcontroller; it uses semihosting to refer to the application binary files stored on the hard disk of the host computer on which DS-5 is running and to write them to the NOR flash memory.


(3) Write the application binary files read by the flash downloader to the NOR flash memory.
(2) Read the application binary files by the flash downloader.
CPU memory space
CS0 space (NOR flash memory)
0x40000000
Flash downloader
0x00000000
DSTREAM or ULINK2
JTAG or SWD
USB connection
Host PC
DS-5
(1) Load the flash downloader into the tightly-coupled memory (ATCM) from DS-5. Since loading is to the RAM, no specific algorithm is required.
[Flash downloader project]
- RZ_T_fmtool_nor.axf
[Application project]
- RZ_T_nor_sample.axf
- RZ_T_nor_sample.bin
- CONST_LOADER_TABLE
- LOADER_RESET_HANDLER
- LOADER_IN_ROOT
- INIT
- CM3_SECTION

Note: CM3_SECTION is a binary file for a Cortex-M3 program. For details, see the application note “RZ/T1 Group Initial Settings of the Microcomputers Incorporating the R-IN Engine”.

Figure 5.1 Schematic View of Flash Downloader Operation

5.3 Developing a Flash Downloader

Figure 5.2 shows the flow of developing a flash downloader. The flash downloader is developed as a DS-5 project. This project is called the flash downloader project. The flash downloader handles processing for reading the application binary files by means of semihosting, CPU initialization, and programming to suit the given NOR flash memory. The sample program covered in this application note handles programming of the NOR flash memory on the RZ/T1 evaluation board as a flash memory interface function. For details of the NOR flash memory interface functions, see Section 7, Flash Memory Interface Functions.

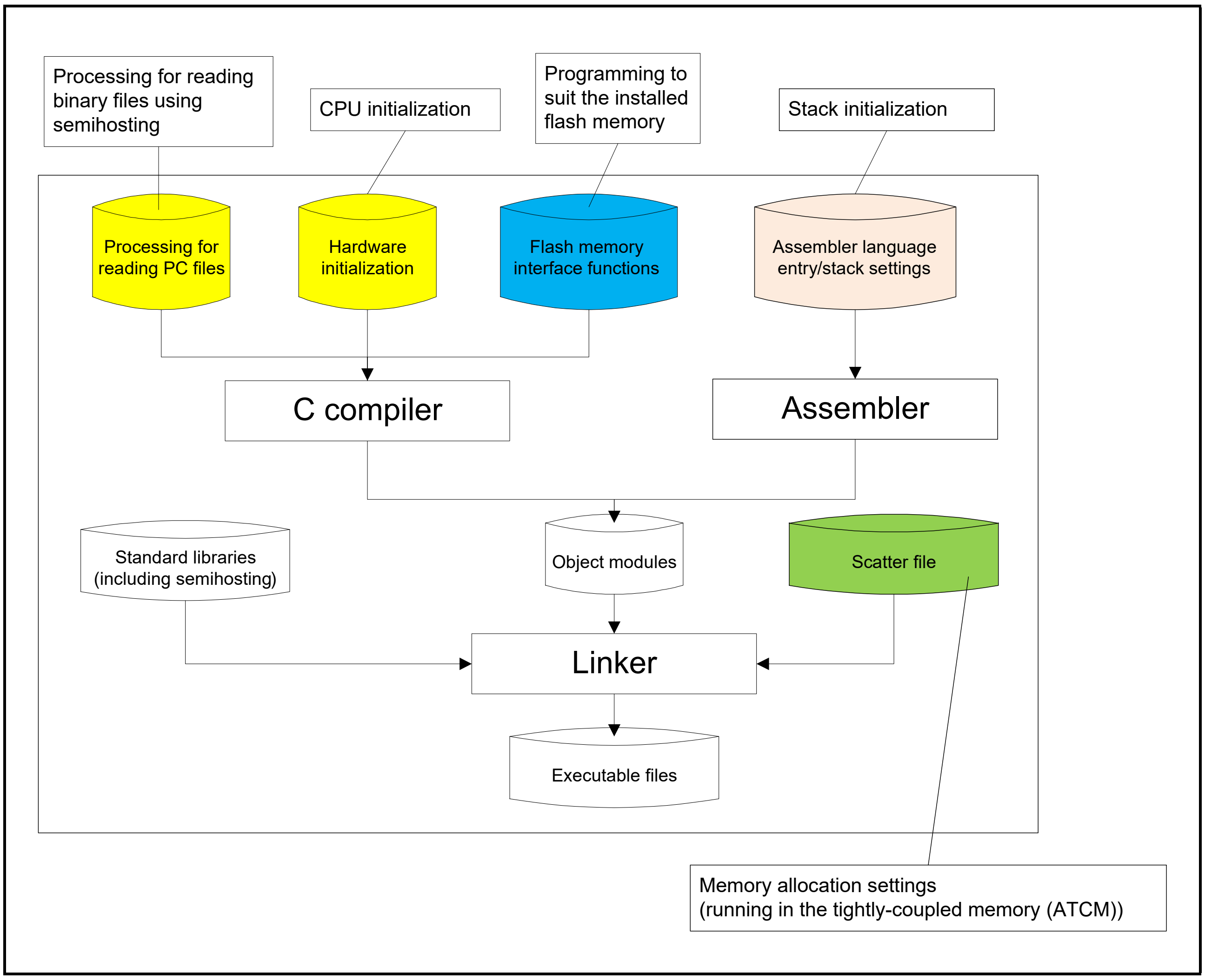

Processing for reading binary files using semihosting
CPU initialization
Programming to suit the installed flash memory
Stack initialization
Processing for reading PC files
Hardware initialization
Flash memory interface functions
Assembler language entry/stack settings
C compiler
Assembler
Standard libraries (including semihosting)
Object modules
Scatter file
Linker
Executable files
Memory allocation settings (running in the tightly-coupled memory (ATCM))

Figure 5.2 Flash Downloader Development Flow

5.3.1 Memory Map

Since the flash downloader runs in the tightly-coupled memory (ATCM) of the RZ/T1 group microcontroller, a scatter file*1 is used to allocate it to the tightly-coupled memory (ATCM). Figure 5.3 shows the memory allocation of the flash downloader.

Note 1. A scatter file is text in which memory layout and allocation of codes and data are described. For details, see “ARM® Compiler toolchain Using the Linker, Image structure and generation”.

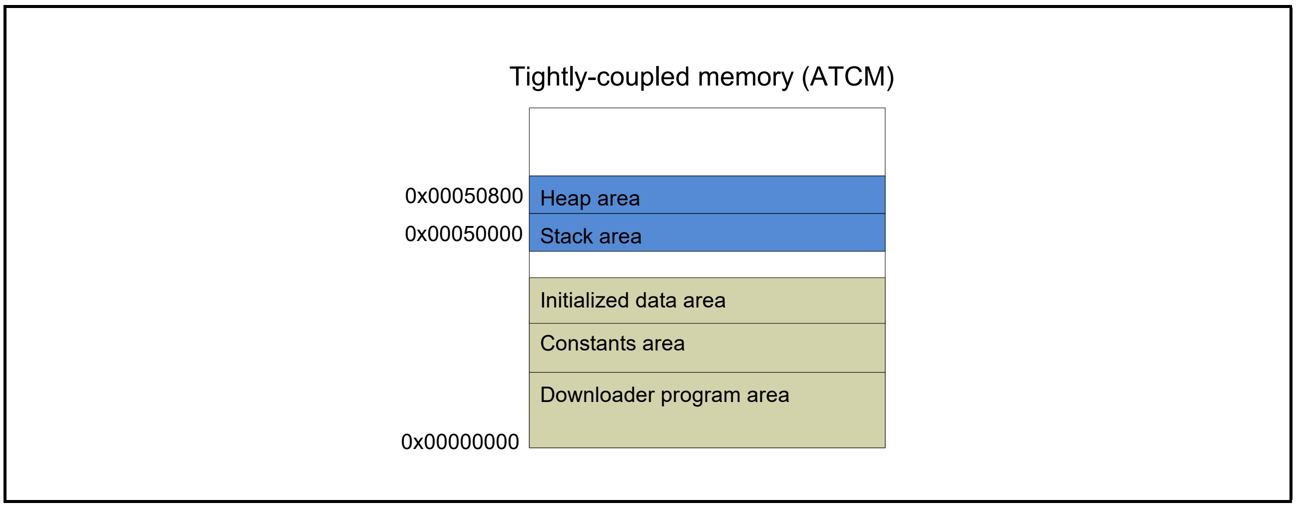


Figure 5.3 Memory Allocation of the Flash Downloader

1. The flash downloader is allocated to the tightly-coupled memory (ATCM) area of the RZ/T1 group microcontroller. Address 0x00000000 is set as the entry point of the flash downloader.
2. The stack area, heap area, etc. used by the flash downloader are allocated to the tightly-coupled memory (ATCM) area.
3. An exception handler vector table need not be implemented for the flash downloader since semihosting provides this functionality.

5.4 Customizing the Examples of Downloading to the NOR Flash Memory

This section describes the procedure for customizing the examples of downloading to the NOR flash memory presented in this application note.

You can customize the items listed in Table 5.2. Customize them to suit the specifications of your system.

Table 5.2 Customizable Items

Item	**Description**
Customization to suit the application project to be downloaded	The names of the application binary files and the write start addresses can be customized to suit the application project to be downloaded to the NOR flash memory. For details of the customization procedure, see Section 10.1, Changing the Binary File Names and Destination Addresses for Writing.
Customization of the flash memory interface functions	The flash memory interface functions can be customized to suit the flash memory to be programmed. For details of the customization procedure, see Section 10.2, Customizing the Flash Memory Interface Functions to Suit the Given Flash Memory.

6. Example of Downloading to the RZ/T1 Evaluation Board (RTK7910022C00000BR)

This section presents the procedure for downloading the application program (RZ_T_nor_sample) to the NOR flash memory on the RZ/T1 evaluation board (RTK7910022C00000BR) by using DS-5 and the ARM® emulator according to the method of downloading presented in this application note.

Figure 6.1 shows an outline of the downloading procedure.

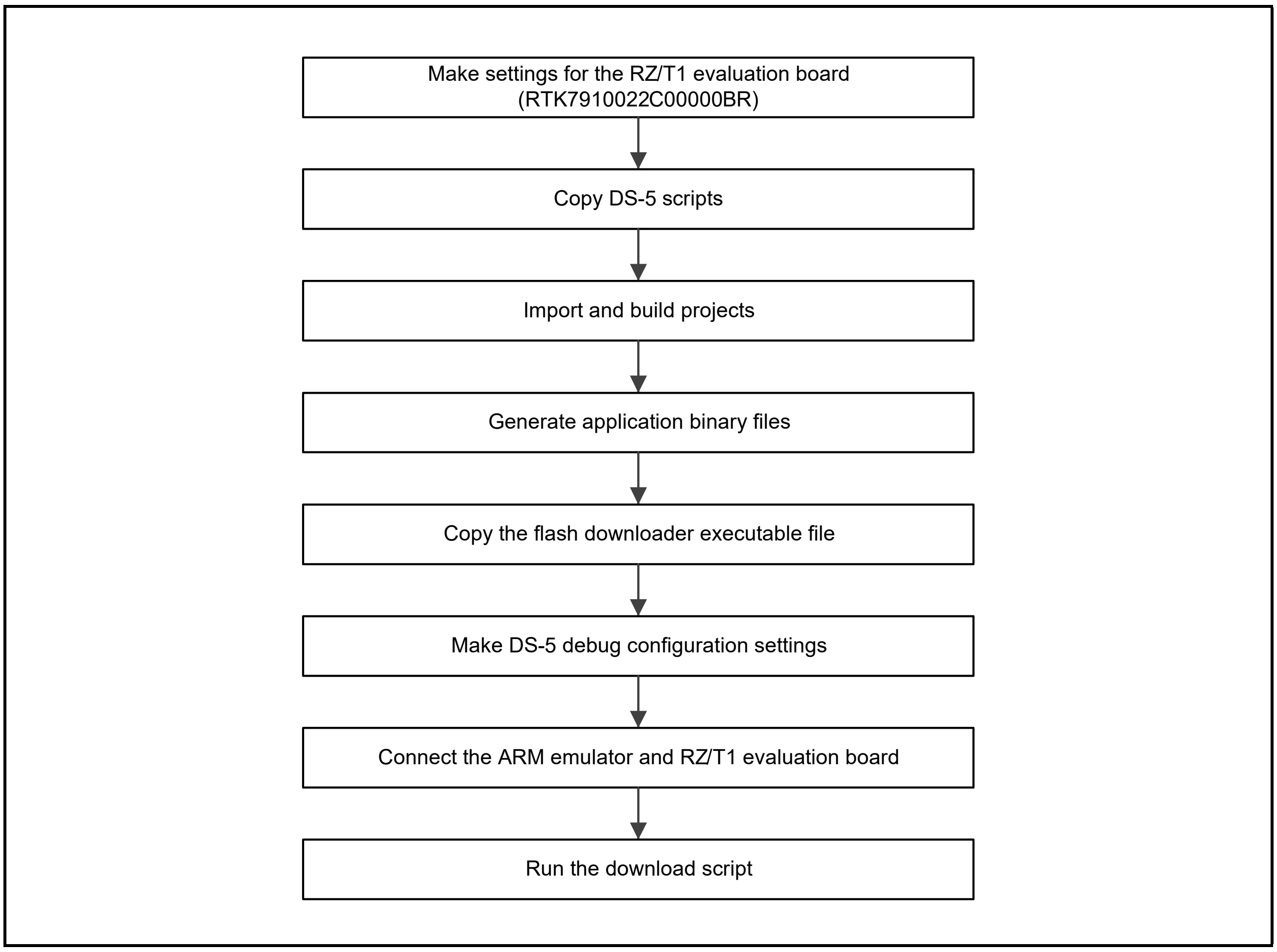

Make settings for the RZ/T1 evaluation board (RTK7910022C00000BR)
Copy DS-5 scripts
Import and build projects
Generate application binary files
Copy the flash downloader executable file
Make DS-5 debug configuration settings
Connect the ARM emulator and RZ/T1 evaluation board
Run the download script

Figure 6.1 Outline of Downloading Procedure

6.1 Settings for the RZ/T1 Evaluation Board (RTK7910022C00000BR)

Table 6.1 lists the settings for the RZ/T1 evaluation board (RTK7910022C00000BR) to run the sample program in this application note.

Make settings for the RZ/T1 evaluation board (RTK7910022C00000BR) as indicated in Table 6.1.

Table 6.1 Settings for the RZ/T1 Evaluation Board (RTK7910022C00000BR)

SW	Setting	Description
SW4-1	ON	MD0 = low level
SW4-2	OFF	MD1 = high level
SW4-3	ON	MD2 = low level
SW4-4	ON	BSCANP = low level
SW4-5	ON	OSCTH = low level
SW4-6	OFF	PU7 = high level

6.2 Copying DS-5 Scripts

Create a directory [script_nor] under the application project (RZ_T_nor_sample) directory and copy the DS-5 scripts listed in Table 6.2 into it.

Note: For details of the DS-5 work space directory, see “Using the ARM® DS-5TM Debugger” provided by ARM®.

Table 6.2 List of DS-5 Script Files

Script Name	Description
init_RZ-T.ds	This is the RZ/T1 evaluation board initialization script. This DS-5 script is for executing processing, such as enabling writing to the tightly-coupled memory (ATCM) of the RZ/T1 group microcontroller, when DS-5 is connected to the RZ/T1 evaluation board.
RZ_T_nor_sample.ds	This is the application downloading script. This DS-5 script contains the sequence of operations for writing the application program to the NOR flash memory allocated to the external address space (CS0 space) of the RZ/T1 group microcontroller.
init_RZ-T2.ds	This is the RZ/T1 evaluation board initialization script to be executed from the application downloading script. It is identical to init_RZ-T.ds, except that it does not make settings for the DS-5 memory area.

6.3 Importing and Building Projects

Import the projects listed in Table 6.3 to the DS-5 workspace directory. Then, build the projects to generate executable files.

[Procedure]

1. Select [All Programs] - [ARM DS-5 v5.21.1] - [Eclipse for DS-5]) from the DS-5 start menu.
2. Select [File (F)] - [Import (I)], then open the [Import - Select] window.
3. Select [General] - [Existing Projects into Workspace], then click [Next].
4. Display the projects by clicking [Reference] in the [Import - Import Projects] window, then select the projects to be imported. In the option, check [Copy Projects into Workspace (C)], then click [End].
5. Select the projects imported by the project explorer in order, then select [Project (P) - Build Projects (B)] to build the projects.

Table 6.3 List of Projects

Project	Description	Executable File
RZ_T_fmtool_nor	This project is used to build a simplified version of the flash downloader. We refer to it as the simplified version of the flash downloader project.	RZ_T_fmtool_nor.axf
RZ_T_nor_sample	This project is used to build the user application. We refer to it as the application project.	RZ_T_nor_sample .axf

6.4 Generating Application Binary Files

Run the command*1 in Figure 6.3 from [DS-5 Command Prompt] of DS-5 to generate application binary files (RZ_T_nor_sample.bin). Table 6.4 lists the binary files generated when this command is run.

In the project included in this application note, this processing is handled by a batch file (¥RZ_T_nor_sample¥Debug¥after_build.bat) when building the project.

[Procedure]

1. Select [All Programs] - [ARM DS-5 v5.21.1] - [DS-5 Command Prompt] from the DS-5 Command Prompt start menu.
2. Type [select_toolchain] and press enter. Select a toolchain to use and press enter (see Figure 6.2).
3. Set a path to the [fmtool] folder created in Section 6.3, Importing and Building Projects, and then run the command*1 listed in Figure 6.3.

Note 1. For details of the command, see “ARM® DS-5™ DS-5 Getting Started Guide, ARM DS-5 Product Overview” provided by ARM®.

```
You can change the compiler toolchain for this environment at any time by
running the 'select_toolchain' command. A default for all future environments
can be set  with the 'select_default_toolchain' command.


C:¥Program Files¥DS-5 v5.21.1¥bin>select_toolchain
Select a toolchain to use in the current environment

 1 - ARM Compiler 5 (DS-5 built-in)
 2 - GCC 4.x [arm-linux-gnueabihf] (DS-5 built-in)
Enter a number or <return> for no toolchain: 1

Environment configured for ARM Compiler 5 (DS-5 built-in)

C:¥Program Files¥DS-5 v5.21.1¥bin>
```

Figure 6.2 Configuring a Toolchain

```
fromelf --bin --output=RZ_T_nor_sample.bin RZ_T_nor_sample.axf
```

Figure 6.3 Application Binary File Generation Command

Table 6.4 Application Binary Files

Application Binary Files		Description
RZ_T_nor_sample.bin	CONST_LOADER_TABLE	Application (1) (loader parameter information) binary file
	LOADER_RESET_HANDLER	Application (2) (loader program) binary file
	LOADER_IN_ROOT	Application (3) (loader program) binary file
	INIT	Application (4) (user program) binary file
	CM3_SECTION*1	Application (5) (user program) binary file (Cortex-M3 program)

Note 1. CM3_SECTION is a binary file for a Cortex-M3 program. For details, refer to the application note “RZ/T1 Group Initial Settings of the Microcomputers Incorporating the R-IN Engine”.

6.5 Copying the Flash Downloader Executable File

Create a directory [fmtool] directly under the application project (RZ_T_nor_sample) directory that was imported in Section 6.3, Importing and Building Projects, and copy the flash downloader project executable file (RZ_T_fmtool_nor.axf) into it.

In the project included in this application note, this processing is handled by a batch file (¥RZ_T_fmtool_nor¥Debug¥after_build.bat) when building the project.

6.6 DS-5 Debug Configuration Settings

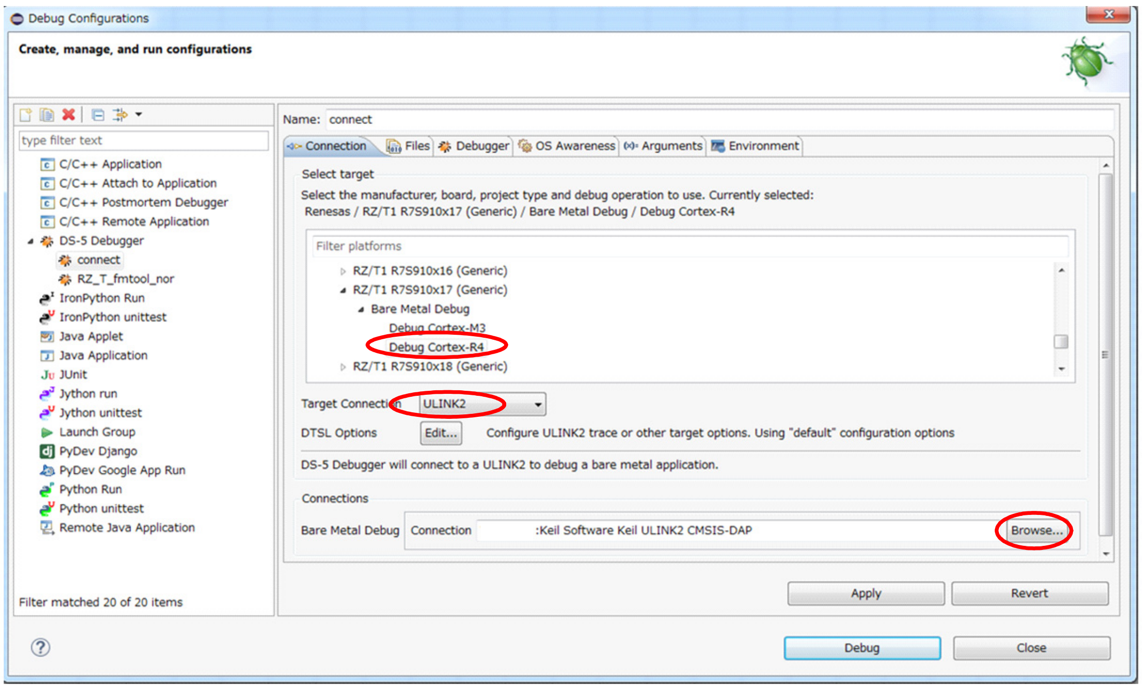
Follow the procedure below to make settings for a DS-5 debug configuration. The DS-5 debug configuration settings specify that the RZ/T1 evaluation board initialization script (init_RZ-T.ds) is run when DS-5 is connected to the RZ/T1 evaluation board*1. For details of processing by the RZ/T1 evaluation board initialization script (init_RZ-T.ds), see Section 9.2, RZ/T1 Evaluation Board Initialization Script.

[Procedure]

1. In DS-5, select [Run (R)] - [Debug Configurations (B)] to display the [Debug Configurations] window.
2. In the [Connection] tab of the DS-5 [Debug Configurations] window, select the target. As the target, select [Renesas] / RZ/T1 R7S910x17(Generic)] / [Bare Metal Debug] / [Debug of Cortex-R4]*2.
3. In the [Connection] tab of the DS-5 [Debug Configurations] window, select target connection and connection browser. In [Target Connection], select the debugger to connect, and then press the [Browse] button in [Bare Metal Debug] to select the connected debugger in [Connection Browser] (see Figure 6.4).
4. In the [Debugger] tab of the DS-5 [Debug Configurations] window, check the box for [Connect only] under Run control.
5. In the [Debugger] tab of the DS-5 [Debug Configurations] window, check the box for [Run target initialization debugger script (.ds/.py)] under Run control, and set a path to the RZ/T1 evaluation board initialization script (init_RZ-T.ds).

Note 1. The above procedure assumes that the RZ/T1 evaluation board has been registered with the DS-5 platform. If the RZ/T1 evaluation board has not been registered with the DS-5 platform, use the DS-5 debugger hardware configuration tools to register it.

Note 2. The name of the target to select may differ according to the version of the DS-5 you are using.



Name: connect

Select target
Select the manufacturer, board, project type and debug operation to use. Currently selected:
Renesas / RZ/T1 R7S910x17 (Generic) / Bare Metal Debug / Debug Cortex-R4

Target Connection: ULINK2

DTSL Options: Configure ULINK2 trace or other target options. Using "default" configuration options

DS-5 Debugger will connect to a ULINK2 to debug a bare metal application.

Connections
Bare Metal Debug Connection :Keil Software Keil ULINK2 CMSIS-DAP

Note: This screen shot is for when a debugger to connect to the target is ULINK2TM.

Figure 6.4 Selecting a Debugger in DS-5

6.7 Connecting the RZ/T1 Evaluation Board with an ARM® Emulator

Follow the procedure below to connect the RZ/T1 evaluation board with an ARM® emulator.

[Procedure]

1. In the [Debug Control] tab of DS-5, select the connection under the name specified in step 2 of Section 6.6, DS-5 Debug Configuration Settings. Then right click to select [Connect to Target].
2. Connection starts in step 1. After the connection is established, the RZ/T1 evaluation board initialization script (init_RZ-T.ds) registered in step 4 of Section 6.6, DS-5 Debug Configuration Settings, is run.

6.8 Running the Download Script

Follow the procedure below to run the download script (RZ_T_nor_sample.ds).

[Procedure]

1. In the [Scripts] tab of DS-5, register the download script (RZ_T_nor_sample.ds).
2. Double-click the download script (RZ_T_nor_sample.ds) registered in step 1 to run the script.
3. When the download script runs, the flash downloader is launched and starts writing to the flash memory. Figure 6.5 shows the message displayed in [Application Console].
4. When downloading is complete, the symbol information of the flash downloader is discarded, the RZ/T1 evaluation board initialization script (init_RZ-T2.ds) is run from the download script (RZ_T_nor_sample.ds), and the symbol information of the application program (RZ_T_nor_sample) is loaded.

```
RZ/T1 CPU Board NOR-Flash Programming Sample. Ver.1.00
Copyright (C) 2015 Renesas Electronics Corporation. All rights reserved.

Initializing Flash...
Start to load Binary Data to Flash Memory.
loop=1, file=CONST_LOADER_TABLE, flash address=0x40000000.
Calculating Data Size...
Data Size is 76
Programing Flash...
Calcurating Checksum of Loader Parameter.
Verifying Flash...
loop=1, Flash Programming Success!!
loop=2, file=LOADER_RESET_HANDLER, flash address=0x40000200.
Calculating Data Size...
Data Size is 3288
Programing Flash...
Verifying Flash...
loop=2, Flash Programming Success!!
loop=3, file=LOADER_IN_ROOT, flash address=0x40006200.
Calculating Data Size...
Data Size is 192
Programing Flash...
Verifying Flash...
loop=3, Flash Programming Success!!
loop=4, file=INIT, flash address=0x40020000.
Calculating Data Size...
Data Size is 2592
Programing Flash...
Verifying Flash...
loop=4, Flash Programming Success!!
loop=5, Could not open file. Exiting.
Flash Programming Complete
```

Note: Processing by loop = 5 is dedicated to writing a Cortex-M3 program. When writing a Cortex-R4F program, the required writing of the program is completed by loop = 4.

Figure 6.5 Messages Output to the Application Console

7. Flash Memory Interface Functions

This section describes the flash memory interface functions.

7.1 Fixed-Width Integers

Table 7.1 lists fixed-width integers used in the sample program.

Table 7.1 Fixed-Width Integers Used in the Sample Program

Symbol	Description
char8_t	8-bit signed integer
int16_t	16-bit signed integer
int32_t	32-bit signed integer
uint8_t	8-bit unsigned integer
uint16_t	16-bit unsigned integer
uint32_t	32-bit unsigned integer

7.2 Constants

Table 7.2 to Table 7.5 list the constants used in the sample program.

Table 7.2 Constants Used in the Sample Program (1)

Constant	Setting	Description
FM_TOOL_OK	(0)	Success
FM_TOOL_E_ERASE	(-1)	Sector erasure error
FM_TOOL_E_WRITE	(-2)	Programming error
FM_TOOL_E_VERIFY	(-3)	Verification error
FM_TYPE_BYTE	(0x4220)	Bus width of the CS0 space = 8 bits
FM_TYPE_WORD	(0x5720)	Bus width of the CS0 space = 16 bits
FM_TYPE_LONG	(0x4C20)	Bus width of the CS0 space = 32 bits
FM_ACCESS_SIZE	(FM_TYPE_WORD)	Bus width of the CS0 space for the sample program Sets FM_TYPE_BYTE, FM_TYPE_WORD, or FM_TYPE_LONG*1.
FM_UNIFORM	(0)	Sector type of the NOR flash memory = uniform type
FM_TOP_BOOT	(1)	Sector type of the NOR flash memory = top type
FM_BOTTOM_BOOT	(2)	Sector type of the NOR flash memory = bottom type
FM_DUAL_BOOT	(3)	Sector type of the NOR flash memory = dual type
FM_BOOT_TYPE	(FM_UNIFORM)	Sector type of the sample program Sets FM_UNIFORM, FM_TOP_BOOT, FM_BOTTOM_BOOT, and FM_DUAL_BOOT.

Note 1. The sample program does not support FM_TYPE_BYTE.

Table 7.3 Constants Used in the Sample Program (2)

Constant	Setting	Description
FM_CS0_NON_CACHE_START	(0x40000000uL)	Address where the CS0 space starts
FM_B_BOOT_SECTOR_START	(0x00000000uL)	Address where the bottom-boot type sector starts Note: Since the sample program is for uniform type, the setting is (0x00000000uL).
FM_B_BOOT_SECTOR_SIZE	(0x00000000uL)	Sector size of the bottom-boot type sector Note: Since the sample program is for uniform type, the setting is (0x00000000uL).
FM_B_BOOT_SECTOR_NUM	(0)	Number of sectors of the bottom-boot type sector Note: Since the sample program is for uniform type, the setting is (0).
FM_NORMAL_SECTOR_START	(0x00000000uL)	Address where the normal type sector starts
FM_NORMAL_SECTOR_SIZE	(0x00020000uL)	Sector size of the normal type sector
FM_NORMAL_BOOT_SECTOR_NUM	(512)	Number of sectors of the normal type sector
FM_T_BOOT_SECTOR_START	(0x00000000uL)	Address where the top-boot type sector starts Note: Since the sample program is for uniform type, the setting is (0x00000000uL).
FM_T_BOOT_SECTOR_SIZE	(0x00000000uL)	Sector size of the top-boot type sector Note: Since the sample program is for uniform type, the setting is (0x00000000uL).
FM_T_BOOT_SECTOR_NUM	(0)	Number of sectors of the top-boot type sector Note: Since the sample program is for uniform type, the setting is (0).
FM_END_ADDRESS	(0x03FFFFFEuL)	Address where the NOR flash memory ends*1

Note 1. Since the CS0 space has a bus width of 16 bits, the address set is 16-bit aligned.

Table 7.4 Constants Used in the Sample Program (3)

Constant	Setting Value	Description
FM_CMD_S_ERASE_ADDR_1	(FM_CS0_NON_CACHE_START | 0x0AAA)	Address issued in the first cycle of the sequence for issuing the sector erase command*1
FM_CMD_S_ERASE_ADDR_2	(FM_CS0_NON_CACHE_START | 0x0554)	Address issued in the second cycle of the sequence for issuing the sector erase command*1
FM_CMD_S_ERASE_ADDR_3	(FM_CS0_NON_CACHE_START | 0x0AAA)	Address issued in the third cycle of the sequence for issuing the sector erase command*1
FM_CMD_S_ERASE_ADDR_4	(FM_CS0_NON_CACHE_START | 0x0AAA)	Address issued in the fourth cycle of the sequence for issuing the sector erase command*1
FM_CMD_S_ERASE_ADDR_5	(FM_CS0_NON_CACHE_START | 0x0554)	Address issued in the fifth cycle of the sequence for issuing the sector erase command*1
FM_CMD_PROGRAM_ADDR_1	(FM_CS0_NON_CACHE_START | 0x0AAA)	Address issued in the first cycle of the sequence for issuing the programming command*1
FM_CMD_PROGRAM_ADDR_2	(FM_CS0_NON_CACHE_START | 0x0554)	Address issued in the second cycle of the sequence for issuing the programming command*1
FM_CMD_PROGRAM_ADDR_3	(FM_CS0_NON_CACHE_START | 0x0AAA)	Address issued in the third cycle of the sequence for issuing the programming command*1

Note 1. Since the CS0 space has a bus width of 16 bits, the address set is 16-bit aligned.

Table 7.5 Constants Used in the Sample Program (4)

Constant	Setting Value	Description
FM_CMD_RESET	(0x00F0)	This is for setting the reset command.
FM_CMD_S_ERASE_DATA_1	(0x00AA)	Command issued in the first cycle of the sequence for issuing the sector erase command
FM_CMD_S_ERASE_DATA_2	(0x0055)	Command issued in the second cycle of the sequence for issuing the sector erase command
FM_CMD_S_ERASE_DATA_3	(0x0080)	Command issued in the third cycle of the sequence for issuing the sector erase command
FM_CMD_S_ERASE_DATA_4	(0x00AA)	Command issued in the fourth cycle of the sequence for issuing the sector erase command
FM_CMD_S_ERASE_DATA_5	(0x0055)	Command issued in the fifth cycle of the sequence for issuing the sector erase command
FM_CMD_SECTOR_ERASE	(0x0030)	Command issued in the sixth cycle of the sequence for issuing the sector erase command
FM_CMD_PROGRAM_DATA_1	(0x00AA)	Command issued in the first cycle of the sequence for issuing the programming command
FM_CMD_PROGRAM_DATA_2	(0x0055)	Command issued in the second cycle of the sequence for issuing the programming command
FM_CMD_PROGRAM_DATA_3	(0x00A0)	Command issued in the third cycle of the sequence for issuing the programming command
FM_CHK_DQ7	(0x0080)	DQ7: Mask value for the DATA#polling bit
FM_CHK_DQ6	(0x0040)	DQ6: Mask value for the toggle bit
FM_CHK_DQ5	(0x0020)	DQ5: Mask value for the timing limit excess bit

7.3 Variables

Table 7.6 lists static variables.

Table 7.6 Static Variables

Type	Variable	Description	Function to be Used
static uint8_t	fmtool_pre_erase_sctno[];	Sector erasure flag Assigns a byte containing a flag to one sector of the NOR flash memory. The sector erasure flag is set to 0 (indicating the non-erased state) when running the initialization interface function. The flag is set to 1 (indicating the erased state) following sector erasure.	flash_init flash_write

7.4 Flash Memory Interface Functions

Table 7.7 lists the flash memory interface functions. Implement processing to suit the flash memory to be programmed in these functions.

Table 7.7 List of Flash Memory Interface Functions

Function	Description
flash_init	Initialization interface function Sets the peripheral modules for use in access to the NOR flash memory connected to the external bus (CS0 space) of the RZ/T1. Initializes the flash memory interface functions.
flash_write	Write interface function Handles writing to the NOR flash memory connected to the external bus (CS0 space) of the RZ/T1. If sector erasure is not executed for the specified address after the execution of the initialization interface function, this function also handles processing to erase the sector.

7.5 Details of the Flash Memory Interface Functions

The following tables list the details of the flash memory interface functions.

flash_init	
Synopsis	Initialization interface function
Header	"flash.h"
Declaration	int32_t flash_init(void);
Description	This function sets the peripheral modules for use in access to the NOR flash memory connected to the external bus (CS0 space) of the RZ/T1. It initializes the flash interface functions. It sets the sector erasure flag (fmtool_pre_erase_sctno) to 0 (indicating the non-erased state).
Arguments	None
Return value	0: Initialization has succeeded (always set to 0 in the sample program). -1: Initialization has failed.

flash_write		
Synopsis	Write interface function	
Header	"flash.h"	
Declaration	int32_t flash_write(uint32_t *fm_adrs, uint32_t *data, int32_t size);	
Description	This function handles writing to the NOR flash memory connected to the external bus (CS0 space) of the RZ/T1. It writes the amount specified by the argument 'size' of data specified by the argument 'data' to the address specified by the argument fm_adrs. If the sector including the address specified by the argument fm_adrs was not erased following a call of the initialization interface function, this function handles processing to erase the sector. Note that erasure or non-erasure of the sector is determined by the value of the sector erasure flag (fmtool_pre_erase_sctno). If the sector has been erased, the value of the sector erasure flag (fmtool_pre_erase_sctno) is set to 1 (indicating the erased state).	
Arguments	uint32_t *fm_adrs	Destination address for writing
	uint32_t *data	Write data storage address
	int32_t size	Amount of data to be written (in bytes)
Return value	Set the result of writing to the flash memory as a returned value. 0: Writing has succeeded. -1: Writing has failed. -2: Verification after writing has failed.	

7.6 Flowcharts of Flash Memory Interface Functions

7.6.1 Initialization Interface Function

Figure 7.1 shows the flow of the initialization interface function.

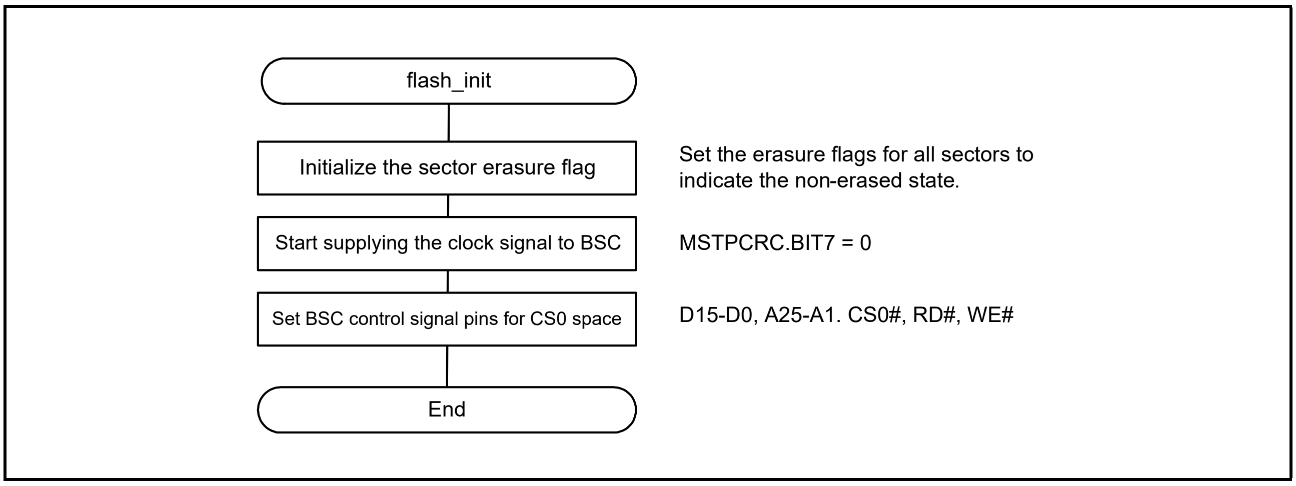

flash_init
Initialize the sector erasure flag
Start supplying the clock signal to BSC
Set BSC control signal pins for CS0 space
End
Set the erasure flags for all sectors to indicate the non-erased state.
MSTPCRC.BIT7 = 0
D15-D0, A25-A1. CS0#, RD#, WE#

Figure 7.1 Initialization Interface Function

7.6.2 Write Interface Function

Figure 7.2 shows the flow of the write interface function.

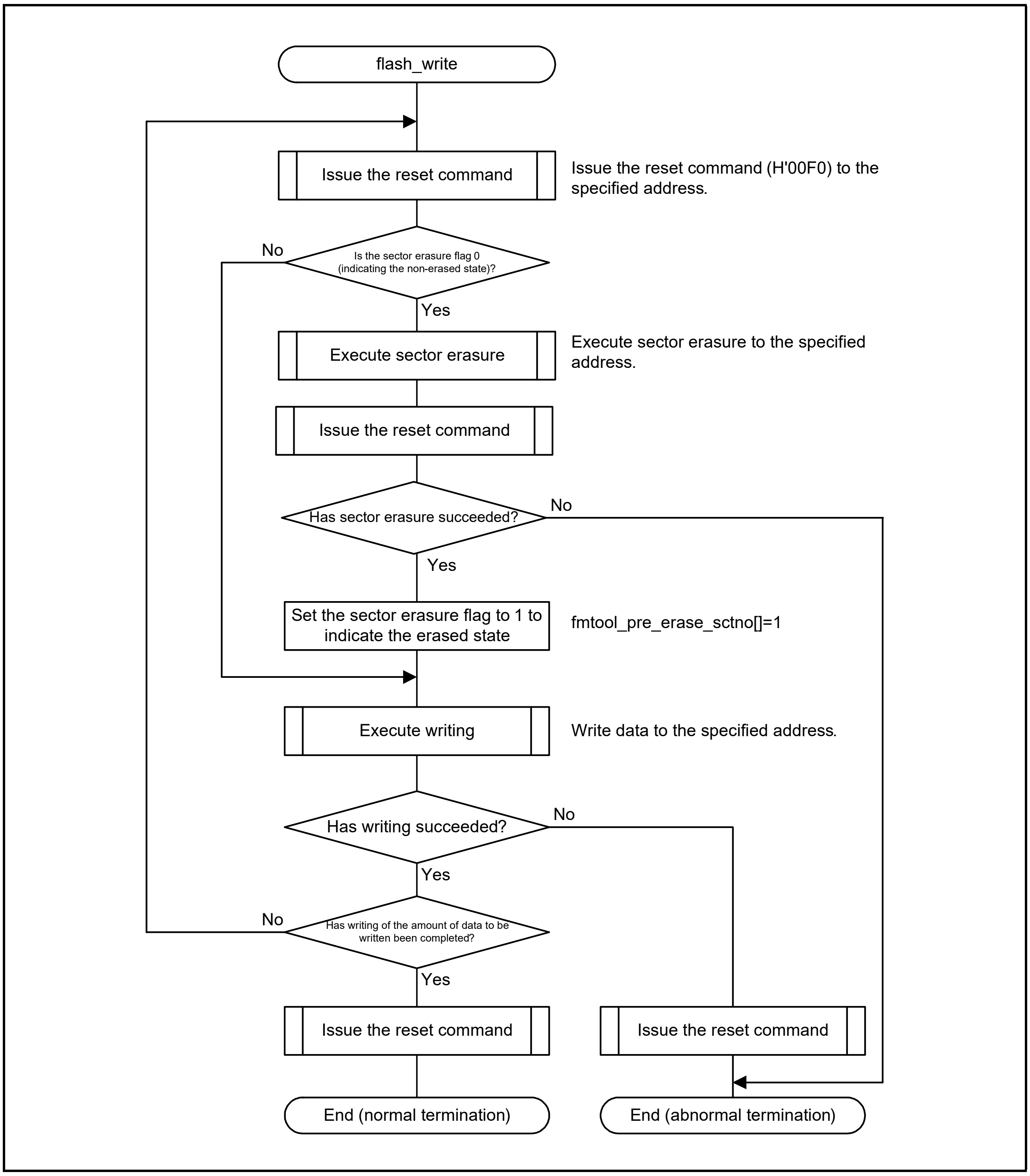

flash_write
Issue the reset command
Issue the reset command (H'00F0) to the specified address.
No
Is the sector erasure flag 0 (indicating the non-erased state)?
Yes
Execute sector erasure
Execute sector erasure to the specified address.
Issue the reset command
Has sector erasure succeeded?
No
Yes
Set the sector erasure flag to 1 to indicate the erased state
fmtool_pre_erase_sctno[]=1
Execute writing
Write data to the specified address.
Has writing succeeded?
No
Yes
No
Has writing of the amount of data to be written been completed?
Yes
Issue the reset command
Issue the reset command
End (normal termination)
End (abnormal termination)

Figure 7.2 Write Interface Function

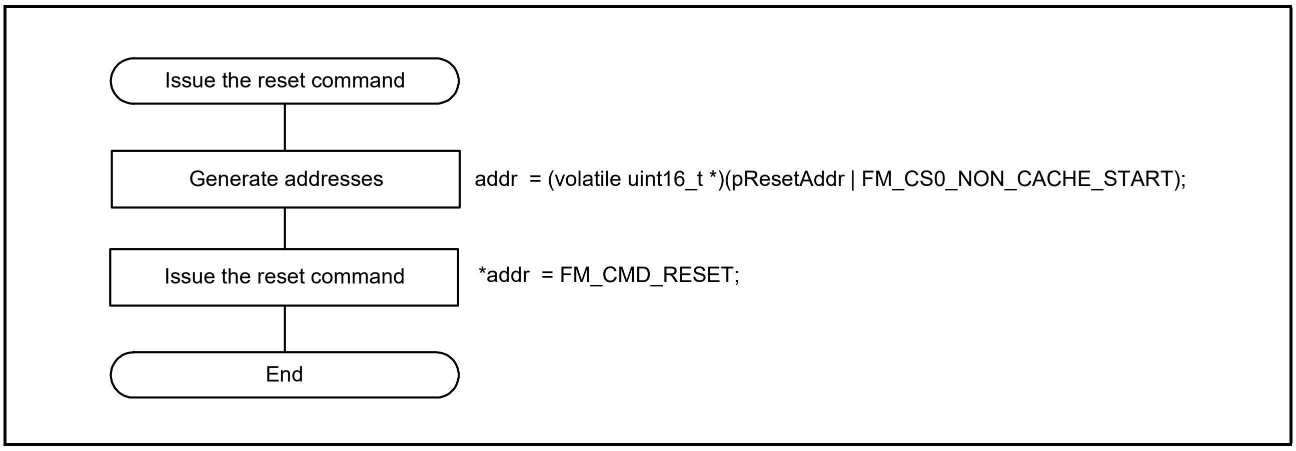


Figure 7.3 Issuing the Reset Command

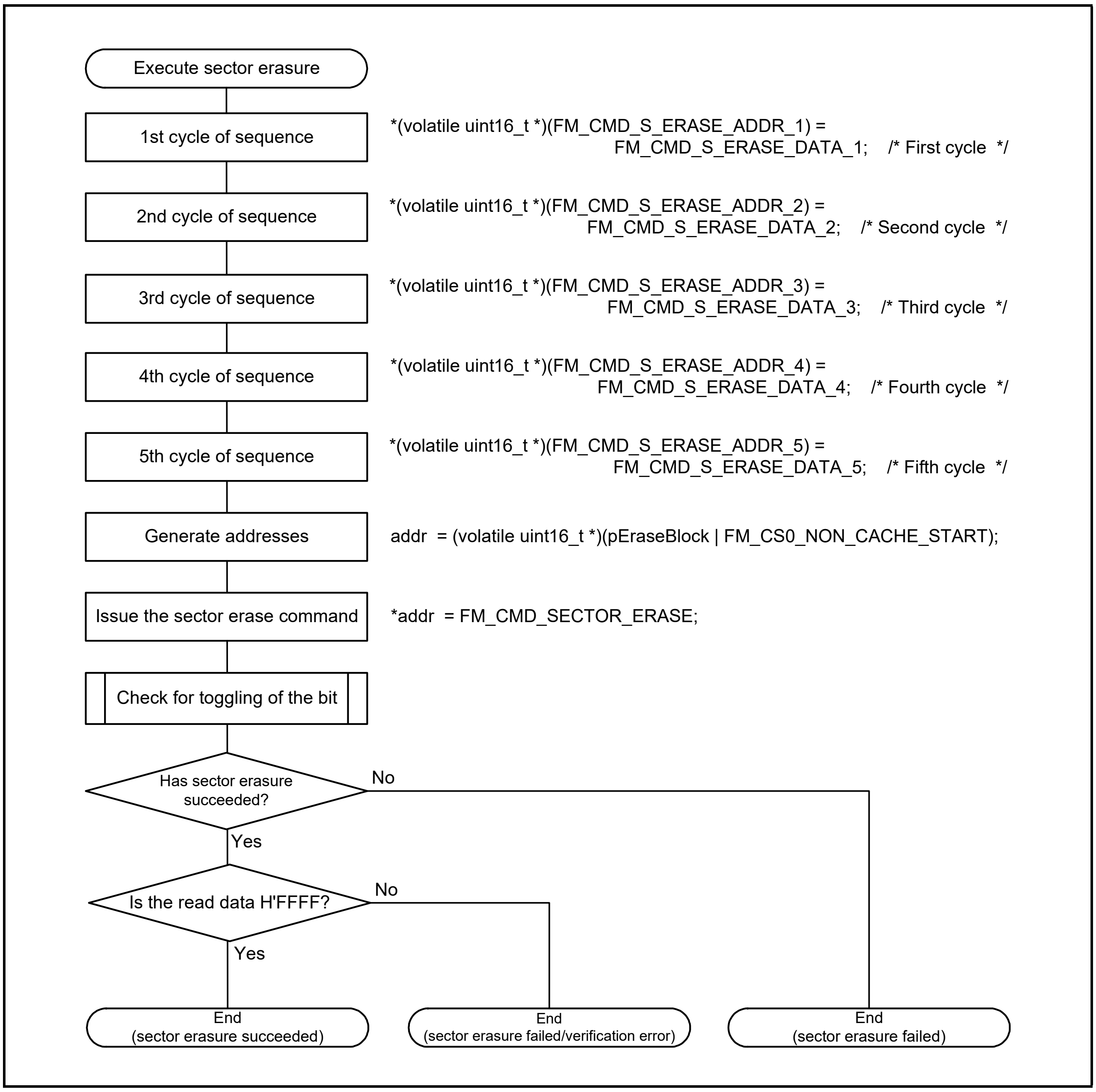
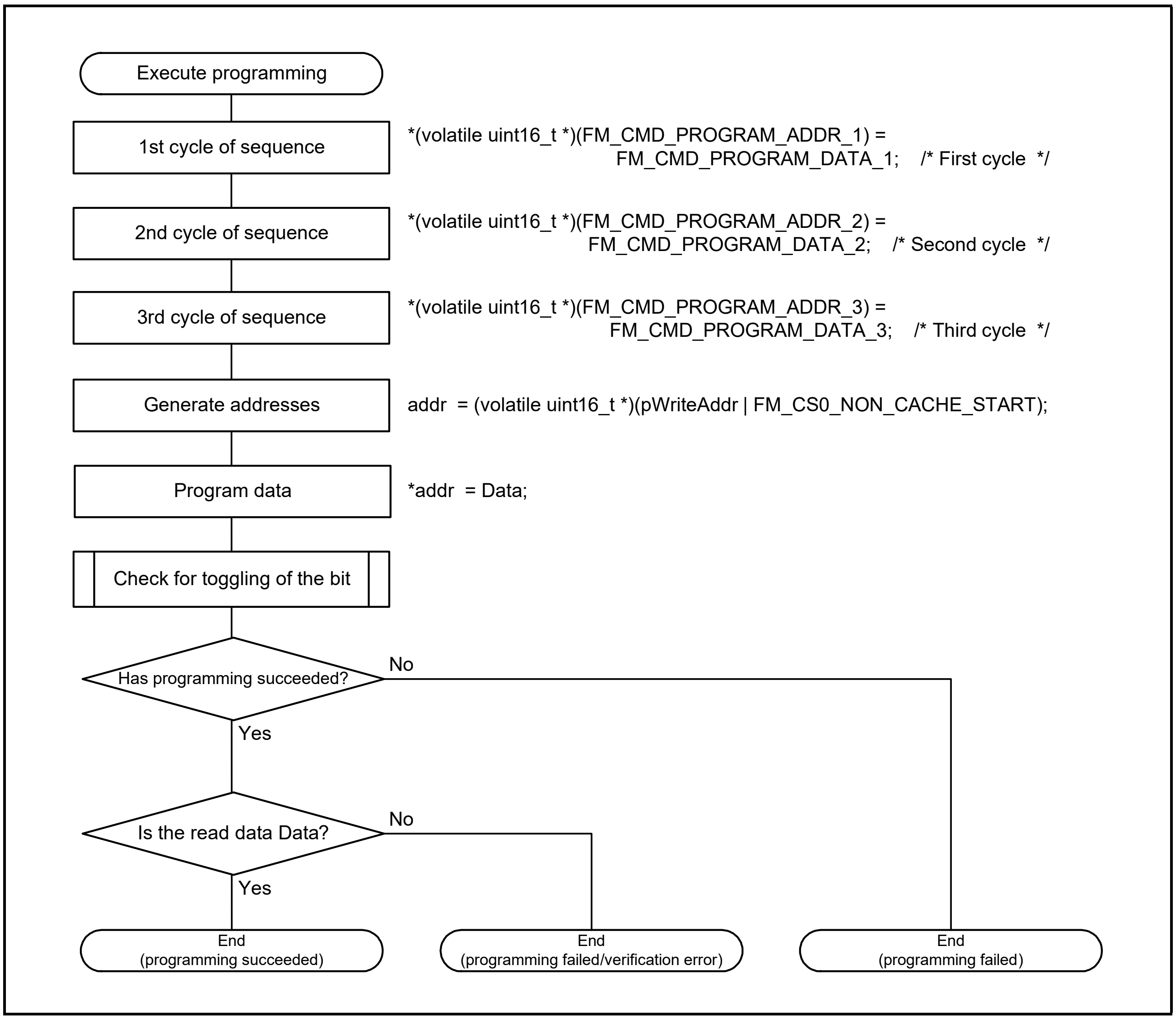

Execute sector erasure
1st cycle of sequence
*(volatile uint16_t *)(FM_CMD_S_ERASE_ADDR_1) = FM_CMD_S_ERASE_DATA_1; /* First cycle */
2nd cycle of sequence
*(volatile uint16_t *)(FM_CMD_S_ERASE_ADDR_2) = FM_CMD_S_ERASE_DATA_2; /* Second cycle */
3rd cycle of sequence
*(volatile uint16_t *)(FM_CMD_S_ERASE_ADDR_3) = FM_CMD_S_ERASE_DATA_3; /* Third cycle */
4th cycle of sequence
*(volatile uint16_t *)(FM_CMD_S_ERASE_ADDR_4) = FM_CMD_S_ERASE_DATA_4; /* Fourth cycle */
5th cycle of sequence
*(volatile uint16_t *)(FM_CMD_S_ERASE_ADDR_5) = FM_CMD_S_ERASE_DATA_5; /* Fifth cycle */
Generate addresses
addr = (volatile uint16_t *)(pEraseBlock | FM_CS0_NON_CACHE_START);
Issue the sector erase command
*addr = FM_CMD_SECTOR_ERASE;
Check for toggling of the bit
Has sector erasure succeeded?
No
Yes
Is the read data H'FFFF?
No
Yes
End (sector erasure succeeded)
End (sector erasure failed/verification error)
End (sector erasure failed)

Figure 7.4 Executing Sector Erasure



```
*(volatile uint16_t *)(FM_CMD_PROGRAM_ADDR_1) =
                         FM_CMD_PROGRAM_DATA_1;    /* First cycle  */
*(volatile uint16_t *)(FM_CMD_PROGRAM_ADDR_2) =
                         FM_CMD_PROGRAM_DATA_2;    /* Second cycle  */
*(volatile uint16_t *)(FM_CMD_PROGRAM_ADDR_3) =
                         FM_CMD_PROGRAM_DATA_3;    /* Third cycle  */
addr  = (volatile uint16_t *)(pWriteAddr | FM_CS0_NON_CACHE_START);
*addr  = Data;
```

Figure 7.5 Executing Programming

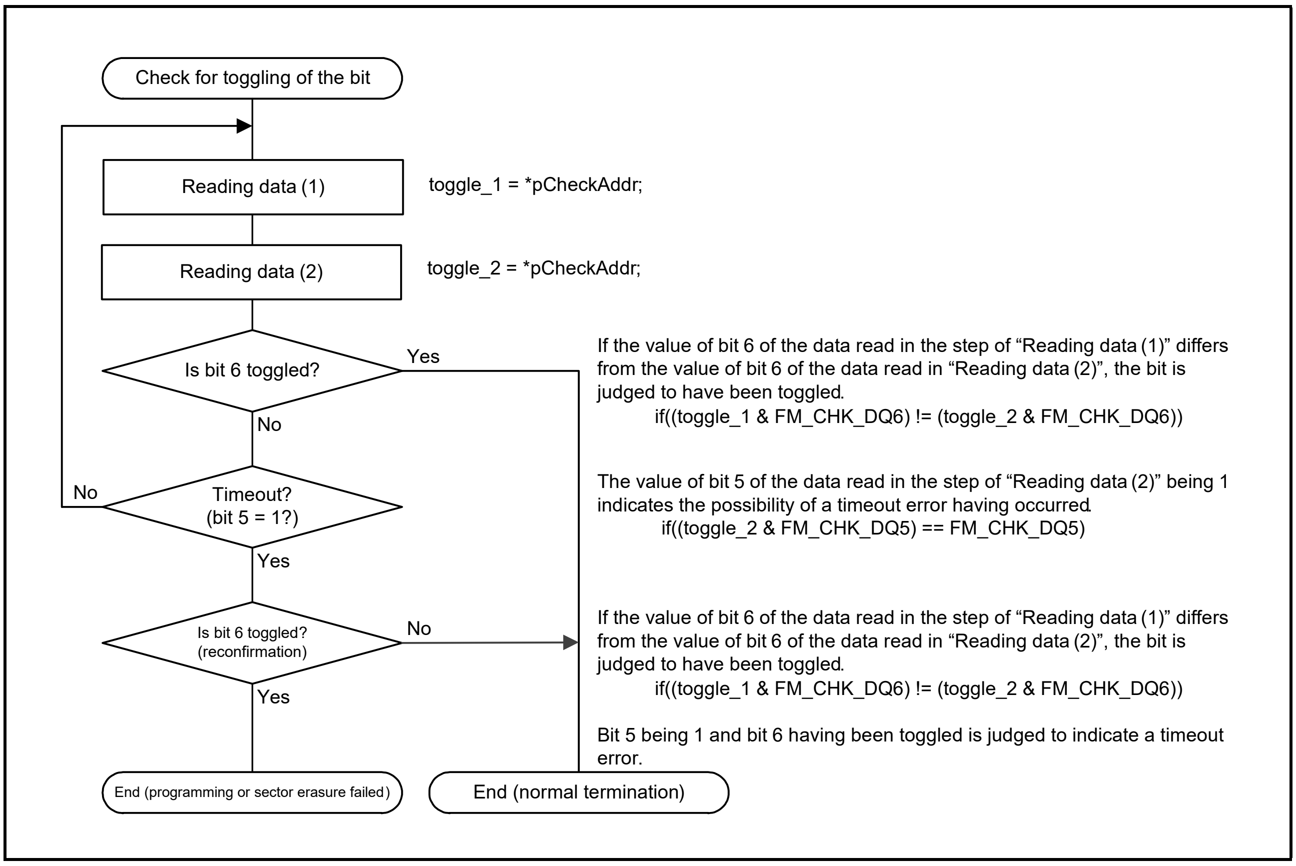

Check for toggling of the bit
Reading data (1)
toggle_1 = *pCheckAddr;
Reading data (2)
toggle_2 = *pCheckAddr;
Is bit 6 toggled?
Yes
No
If the value of bit 6 of the data read in the step of “Reading data (1)” differs from the value of bit 6 of the data read in “Reading data (2)”, the bit is judged to have been toggled.
if((toggle_1 & FM_CHK_DQ6) != (toggle_2 & FM_CHK_DQ6))
No
Timeout? (bit 5 = 1?)
Yes
The value of bit 5 of the data read in the step of “Reading data (2)” being 1 indicates the possibility of a timeout error having occurred.
if((toggle_2 & FM_CHK_DQ5) == FM_CHK_DQ5)
Is bit 6 toggled? (reconfirmation)
No
Yes
If the value of bit 6 of the data read in the step of “Reading data (1)” differs from the value of bit 6 of the data read in “Reading data (2)”, the bit is judged to have been toggled.
if((toggle_1 & FM_CHK_DQ6) != (toggle_2 & FM_CHK_DQ6))
Bit 5 being 1 and bit 6 having been toggled is judged to indicate a timeout error.
End (programming or sector erasure failed)
End (normal termination)

Figure 7.6 Checking for Toggling of the Bit

8. Operation of the Flash Downloader

This section describes operation of the flash downloader.

8.1 Memory Allocation of the Application Program

Figure 8.1 shows the memory allocation of the application program which is written by the flash downloader presented in this application note.

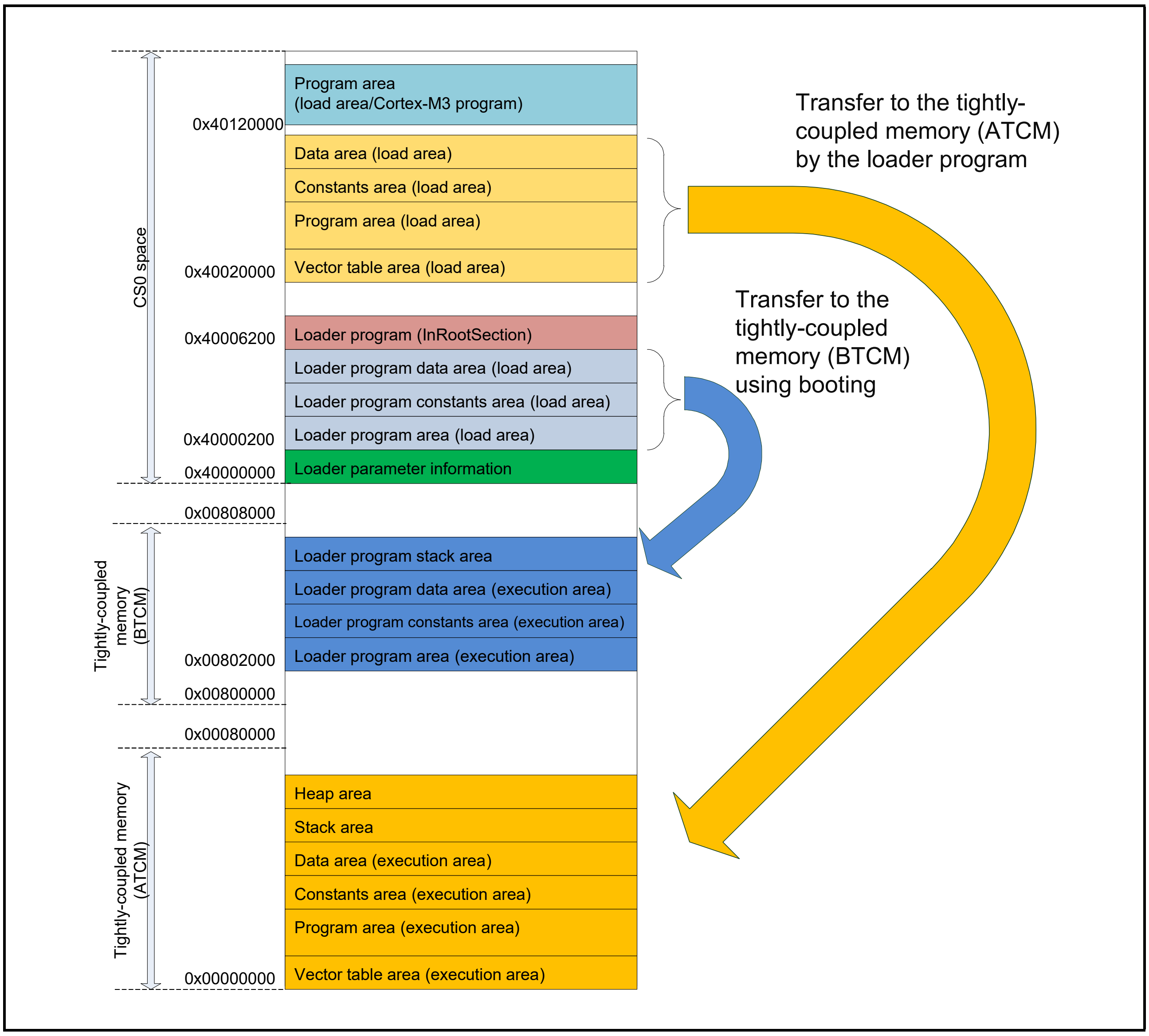

CS0 space
0x40120000
Program area (load area/Cortex-M3 program)
Data area (load area)
Constants area (load area)
Program area (load area)
0x40020000
Vector table area (load area)
0x40006200
Loader program (InRootSection)
Loader program data area (load area)
Loader program constants area (load area)
0x40000200
Loader program area (load area)
0x40000000
Loader parameter information
0x00808000
Tightly-coupled memory (BTCM)
Loader program stack area
Loader program data area (execution area)
Loader program constants area (execution area)
0x00802000
Loader program area (execution area)
0x00800000
0x00080000
Tightly-coupled memory (ATCM)
Heap area
Stack area
Data area (execution area)
Constants area (execution area)
Program area (execution area)
0x00000000
Vector table area (execution area)
Transfer to the tightly-coupled memory (ATCM) by the loader program
Transfer to the tightly-coupled memory (BTCM) using booting

Figure 8.1 Example Memory Allocation of the Application Program

1. The application program has four areas: the areas of the loader parameter information for reference by the RZ/T1 group microcontroller in booting and use by the loader program, the loader program (InRootSection) area, and the application program.
2. Binary data of the four areas are generated as three application binary files*[1] by the binary file generator tool from the executable file (axf file) that was generated from the application project.

Note 1. See Table 9.3 for the application binary files to be generated.

8.2 Flow of Flash Downloader Processing

Figure 8.2 to Figure 8.4 show the flow of processing by the flash downloader.

The flash downloader is loaded by DS-5 to the tightly-coupled memory (ATCM) area of the RZ/T1 group microcontroller and run from the entry point at address 0x00000000.

After the flash downloader initializes the stack pointers, it runs __main(), which is the entry function to the main function. The __main() function is supplied as a standard library function of the ARM® compiler; running this function enables semihosting functionality*1. The flash download main function (flash_main) is run from $Sub$$main(), which is run from the __main function. After flash_main runs, the prg_complete function is run to determine if downloading by the application downloading script (described below) has completed, and processing enters an infinite loop.

Note 1. For details, refer to "ARM® Compiler toolchain Developing Software for ARM® Processors, Embedded Software Development".

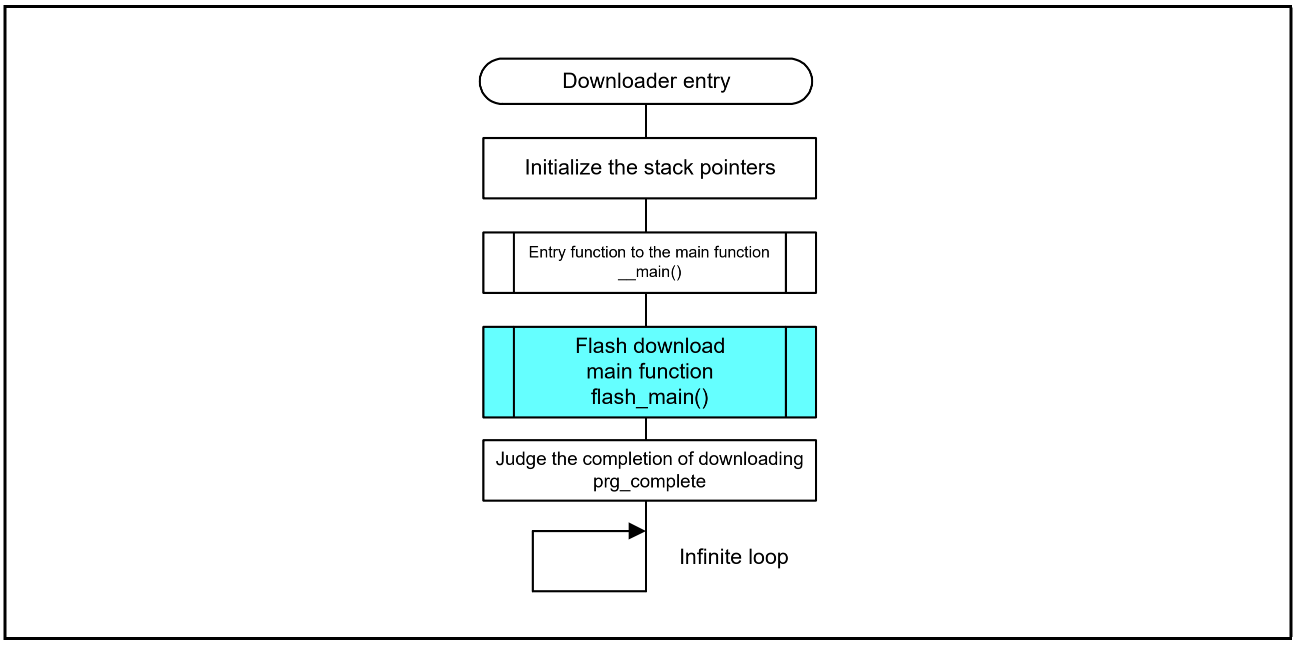
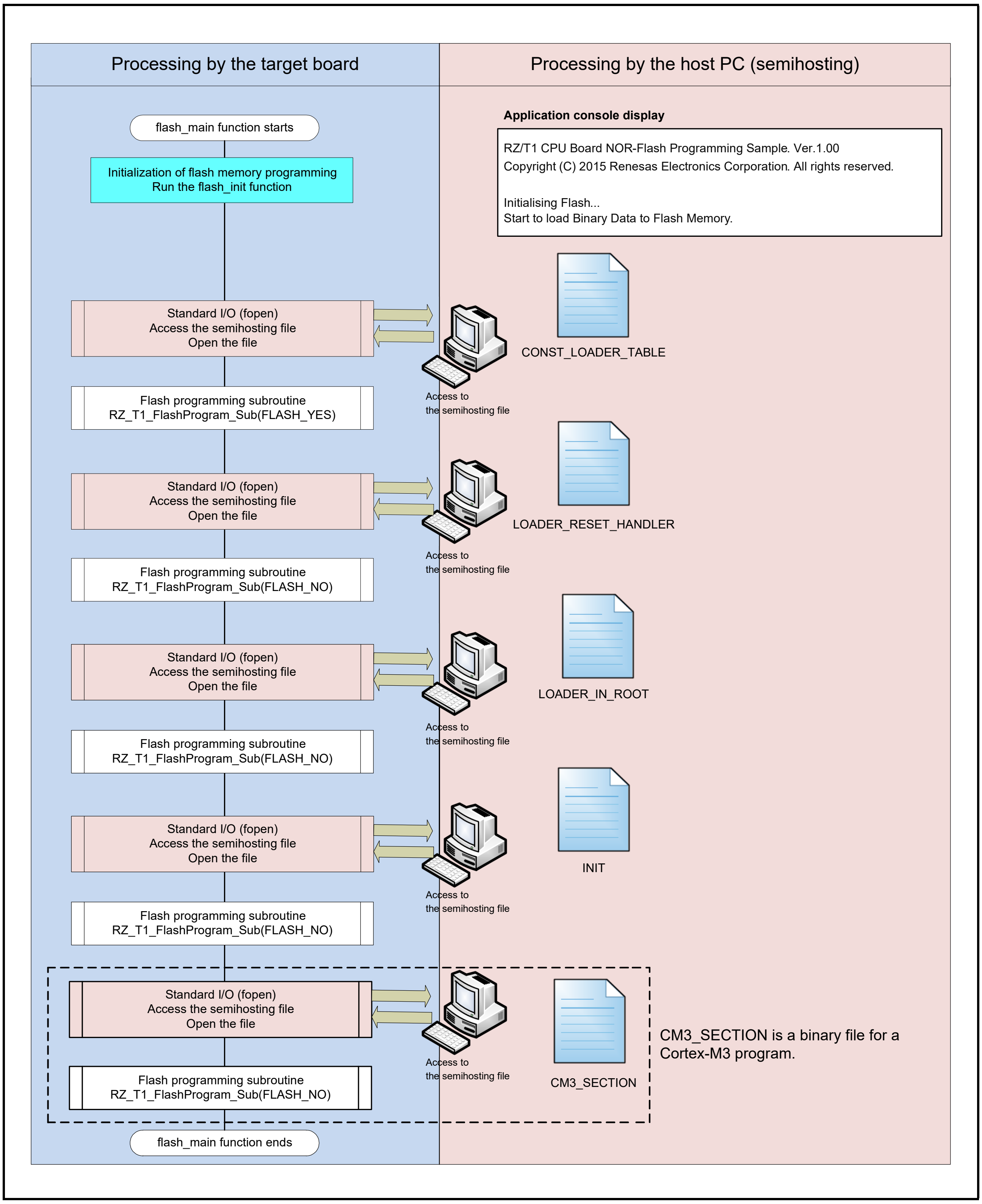

Downloader entry
Initialize the stack pointers
Entry function to the main function
__main()
Flash download
main function
flash_main()
Judge the completion of downloading
prg_complete
Infinite loop

Figure 8.2 Flash Downloader Processing Flow (1/3)

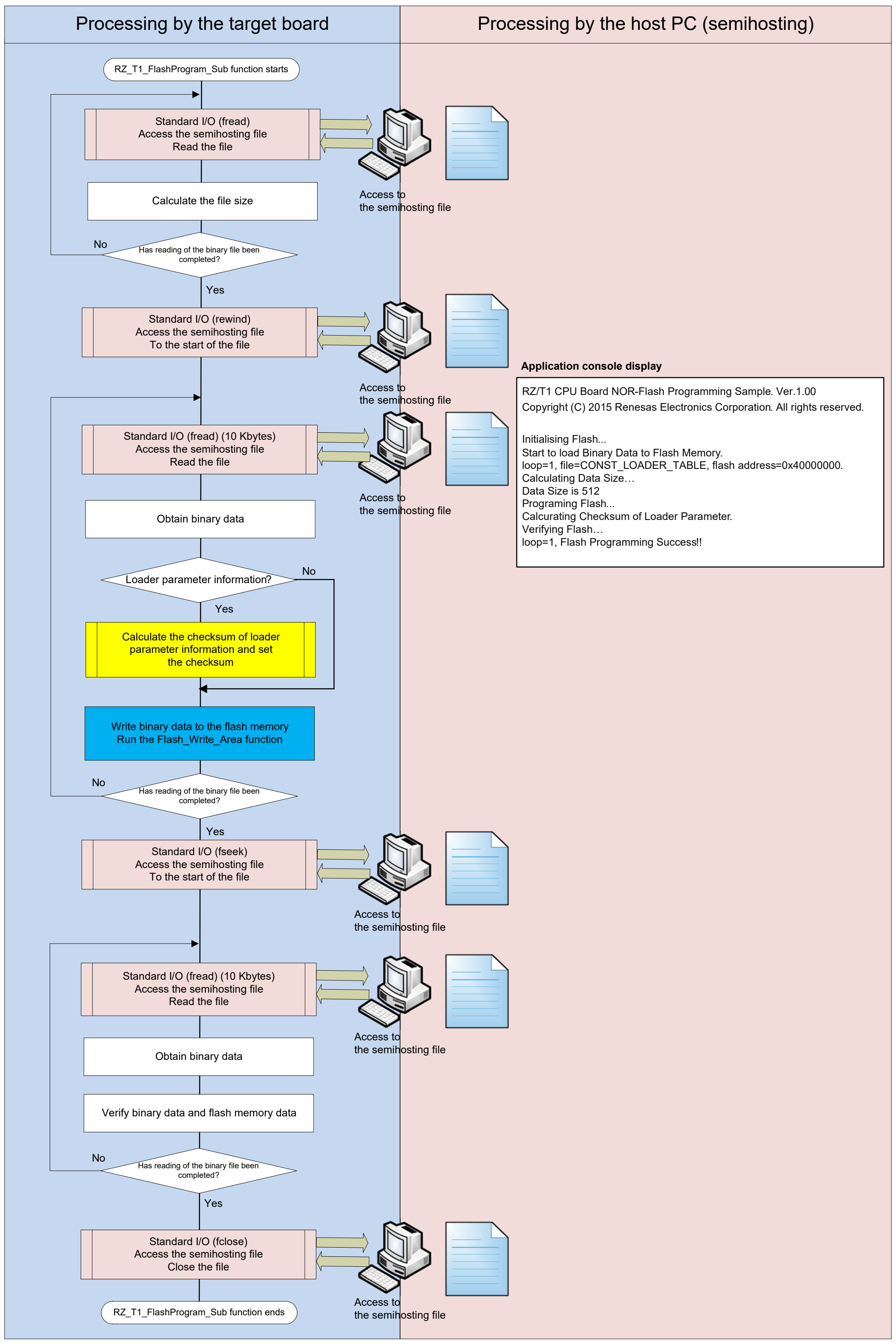
The flash_main function starts programming of the flash memory. Figure 8.3 shows the flow of processing by the flash_main function.

Programming of the flash memory proceeds by the RZ_T1_FlashProgram_Sub function. This uses semihosting to read data from an application binary file, and write data to the NOR flash memory. Figure 8.4 shows the flow of processing by the RZ_T1_FlashProgram_Sub function.


Processing by the target board
Processing by the host PC (semihosting)
flash_main function starts
Initialization of flash memory programming
Run the flash_init function
Standard I/O (fopen)
Access the semihosting file
Open the file
Flash programming subroutine
RZ_T1_FlashProgram_Sub(FLASH_YES)
Standard I/O (fopen)
Access the semihosting file
Open the file
Flash programming subroutine
RZ_T1_FlashProgram_Sub(FLASH_NO)
Standard I/O (fopen)
Access the semihosting file
Open the file
Flash programming subroutine
RZ_T1_FlashProgram_Sub(FLASH_NO)
Standard I/O (fopen)
Access the semihosting file
Open the file
Flash programming subroutine
RZ_T1_FlashProgram_Sub(FLASH_NO)
Standard I/O (fopen)
Access the semihosting file
Open the file
Flash programming subroutine
RZ_T1_FlashProgram_Sub(FLASH_NO)
flash_main function ends
Application console display
Access to the semihosting file
CONST_LOADER_TABLE
LOADER_RESET_HANDLER
LOADER_IN_ROOT
INIT
CM3_SECTION
CM3_SECTION is a binary file for a Cortex-M3 program.

```
RZ/T1 CPU Board NOR-Flash Programming Sample. Ver.1.00
Copyright (C) 2015 Renesas Electronics Corporation. All rights reserved.

Initialising Flash...
Start to load Binary Data to Flash Memory.
```

Figure 8.3 Flash Downloader Processing Flow (2/3)


Processing by the target board
Processing by the host PC (semihosting)
RZ_T1_FlashProgram_Sub function starts
Standard I/O (fread)
Access the semihosting file
Read the file
Access to the semihosting file
Calculate the file size
No
Has reading of the binary file been completed?
Yes
Standard I/O (rewind)
Access the semihosting file
To the start of the file
Access to the semihosting file
Standard I/O (fread) (10 Kbytes)
Access the semihosting file
Read the file
Access to the semihosting file
Obtain binary data
Loader parameter information?
No
Yes
Calculate the checksum of loader parameter information and set the checksum
Write binary data to the flash memory
Run the Flash_Write_Area function
No
Has reading of the binary file been completed?
Yes
Standard I/O (fseek)
Access the semihosting file
To the start of the file
Access to the semihosting file
Standard I/O (fread) (10 Kbytes)
Access the semihosting file
Read the file
Access to the semihosting file
Obtain binary data
Verify binary data and flash memory data
No
Has reading of the binary file been completed?
Yes
Standard I/O (fclose)
Access the semihosting file
Close the file
Access to the semihosting file
RZ_T1_FlashProgram_Sub function ends
Application console display

```
RZ/T1 CPU Board NOR-Flash Programming Sample. Ver.1.00
Copyright (C) 2015 Renesas Electronics Corporation. All rights reserved.

Initialising Flash...
Start to load Binary Data to Flash Memory.
loop=1, file=CONST_LOADER_TABLE, flash address=0x40000000.
Calculating Data Size…
Data Size is 512
Programing Flash...
Calcurating Checksum of Loader Parameter.
Verifying Flash…
loop=1, Flash Programming Success!!
```

Figure 8.4 Flash Downloader Processing Flow (3/3)

8.2.1 Calculating the Checksum of the Loader Parameter Information

The flash downloader is capable of calculating the checksum of the loader parameter information for reference by the RZ/T1 group microcontroller in booting and writing the result to the flash memory.

If FLASH_YES is specified for the argument check_sum_flag of the RZ_T1_FlashProgram_Sub function for execution, the binary file specified by the argument srcfile is taken as the binary file of loader parameter information and the checksum for the 72 bytes (H'48 bytes) from the start of the binary file is calculated. If the value up to the 72nd byte (byte H'48) from the start of the binary file is H'17320508, the calculated checksum is written to the flash memory as the checksum of the loader parameters. If the value up to the 72nd byte (byte H'48) from the start of the binary file is not H'17320508, the calculated checksum is compared with that for the given data, and if the values do not match, subsequent processing does not proceed and processing is abnormally terminated.

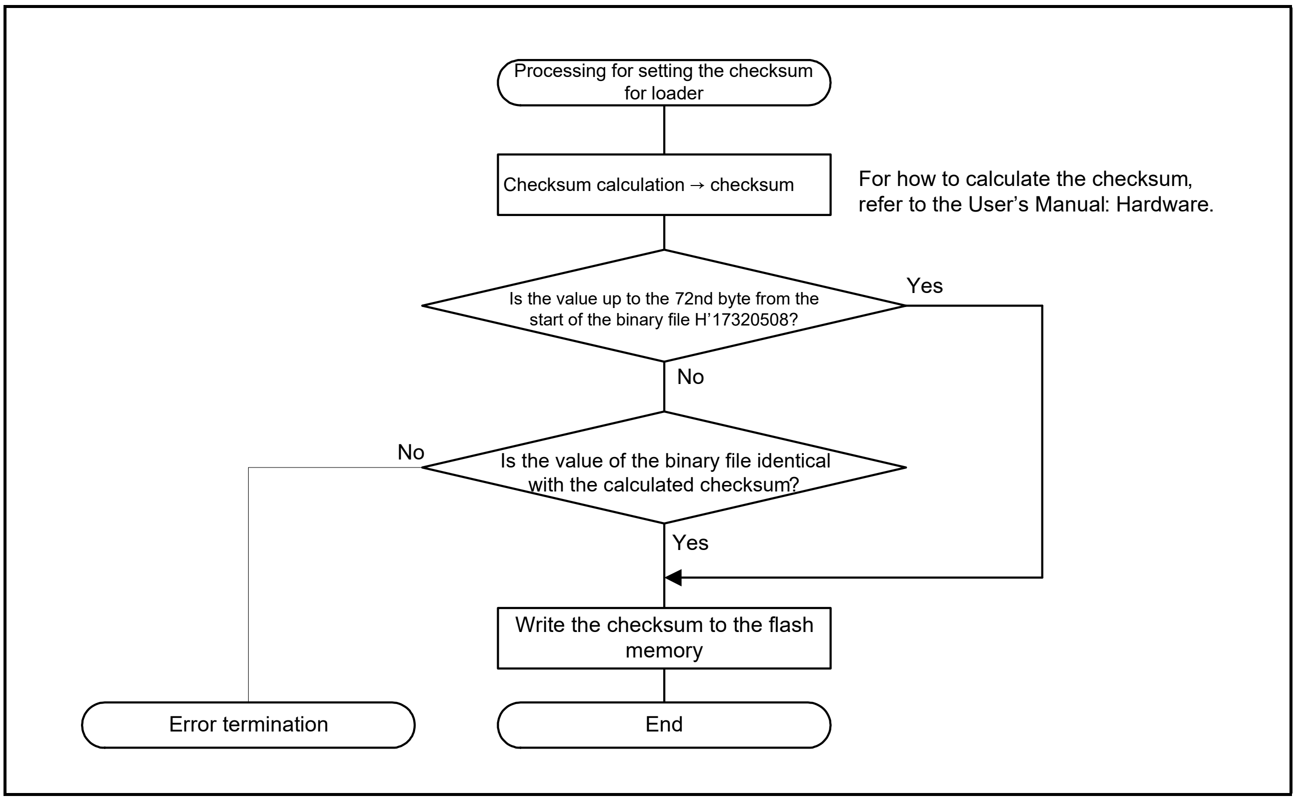

Processing for setting the checksum for loader
Checksum calculation → checksum
For how to calculate the checksum, refer to the User’s Manual: Hardware.
Is the value up to the 72nd byte from the start of the binary file H’17320508?
Yes
No
No
Is the value of the binary file identical with the calculated checksum?
Yes
Write the checksum to the flash memory
Error termination
End

Figure 8.5 Flow of Processing for Setting the Checksum of the Loader Parameter Information

9. Configuration of the Flash Downloader

9.1 Configuration of Projects

The flash downloader comprises the DS-5 projects and DS-5 scripts listed respectively in Table 9.1 and Table 9.2, Table 9.3 lists the application binary files generated from the application project. Follow the flow described in Section 8.2, Flow of Flash Downloader Processing, to write these application binary files to the flash memory.

Table 9.1 List of Projects

Project	Description
RZ_T_fmtool_nor	This project is used to build the flash downloader. We refer to it as the flash downloader project.
RZ_T_nor_sample	This project is used to build the user application. We refer to it as the application project. The binary generator tool (fromelf.exe) is used to generate an application binary file from the executable file (axf file) generated by building the project.

Table 9.2 List of Script Files

Script	Description
init_RZ-T.ds	This is the RZ/T1 evaluation board initialization script. This DS-5 script is for executing processing, such as enabling writing to the tightly-coupled memory (ATCM) of the RZ/T1 group microcontroller, when DS-5 is connected to the RZ/T1 evaluation board.
RZ_T_nor_sample.ds	This is the application downloading script This DS-5 script contains commands for the sequence of operations for writing the application program to the NOR flash memory allocated to the external address space (CS0 space) of the RZ/T1 group microcontroller.
init_RZ-T2.ds	This is the RZ/T1 evaluation board initialization script to be executed from the application downloading script. It is identical to init_RZ-T.ds, except that it does not make settings for the DS-5 memory area.

Table 9.3 List of Application Binary Files

Binary File	Write Start Address*1	Description
CONST_LOADER_TABLE	H'40000000	Application (1) (loader parameter information) binary file
LOADER_RESET_HANDLER	H'40000200	Application (2) (loader program) binary file
LOADER_IN_ROOT	H'40006200	Application (3) (loader program) binary file
INIT	H'40020000	Application (4) (user program) binary file
CM3_SECTION	H'40120000	Application (5) (user program) binary file (Cortex-M3 program)

Note 1. In the case of memory allocation of the application program shown in Figure 8.1.

9.2 RZ/T1 Evaluation Board Initialization Script

Figure 9.1 shows the details of processing by the RZ/T1 evaluation board initialization script.

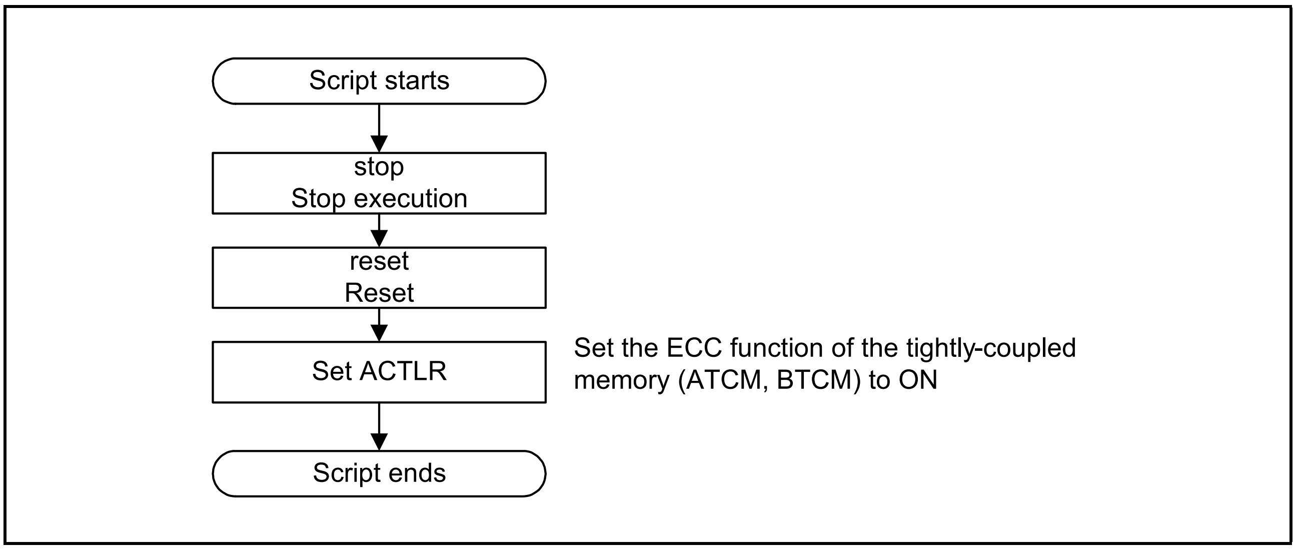

Script starts
stop
Stop execution
reset
Reset
Set ACTLR
Set the ECC function of the tightly-coupled memory (ATCM, BTCM) to ON
Script ends

Figure 9.1 Details of Processing by the RZ/T1 Evaluation Board Initialization Script

9.3 Application Downloading Script

Figure 9.2 shows the details of processing by the application downloading script for writing the application program RZ_T_nor_sample to the NOR flash memory allocated to the external address space (CS0 space) of the RZ/T1 group microcontroller. By running this script from DS-5, the application program RZ_T_nor_sample is written to the NOR flash memory allocated to the external address space (CS0 space) of the RZ/T1 group microcontroller and the symbol information of the application program RZ_T1_nor_sample is loaded into DS-5.

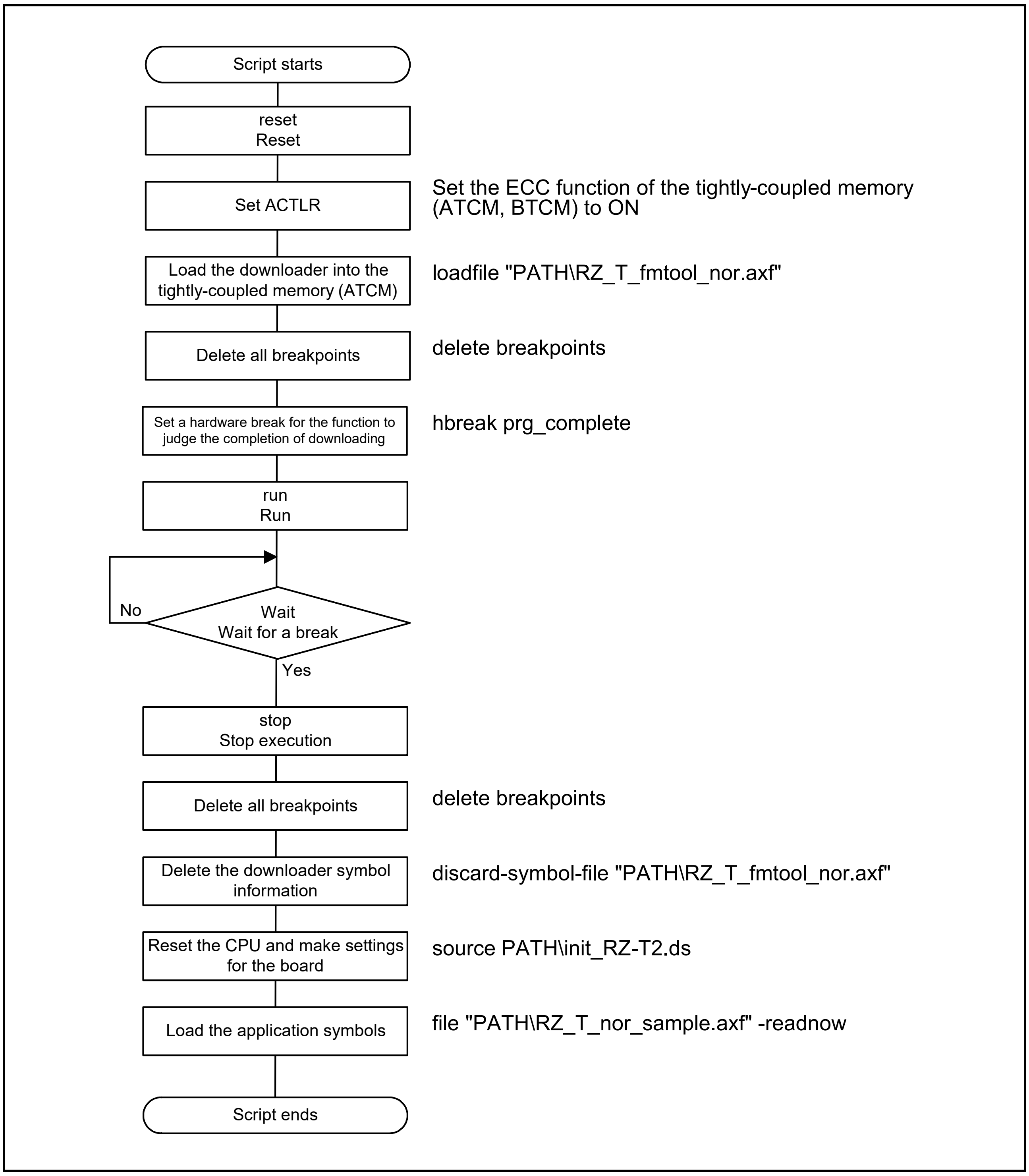

Script starts
reset
Reset
Set ACTLR
Set the ECC function of the tightly-coupled memory (ATCM, BTCM) to ON
Load the downloader into the tightly-coupled memory (ATCM)
loadfile "PATH\RZ_T_fmtool_nor.axf"
Delete all breakpoints
delete breakpoints
Set a hardware break for the function to judge the completion of downloading
hbreak prg_complete
run
Run
No
Wait
Wait for a break
Yes
stop
Stop execution
Delete all breakpoints
delete breakpoints
Delete the downloader symbol information
discard-symbol-file "PATH\RZ_T_fmtool_nor.axf"
Reset the CPU and make settings for the board
source PATH\init_RZ-T2.ds
Load the application symbols
file "PATH\RZ_T_nor_sample.axf" -readnow
Script ends

Figure 9.2 Details of Processing by the Application Downloading Script

10. Application Examples

This section describes how the customer can change the binary files for writing to the flash memory and how to customize the sample program to suit the flash memory used by the customer, as an example of the practical application of the sample program.

10.1 Changing the Binary File Names and Destination Addresses for Writing

This section describes how to change the binary file names for writing to the flash memory and destination addresses in flash memory for writing according to the flow described in Section 8.2, Flow of Flash Downloader Processing.

10.1.1 Changing the Binary File Names for Writing to the Flash Memory

The binary file names for writing to the flash memory are in the RZ_T1_FlashProgrammer function in the Flash_Programming.c file. The names of binary files to be written to the flash memory can be changed by changing the names of the binary files in the RZ_T1_FlashProgrammer function.

The current directory when DS-5 semihosting is executed is set by default to the DS-5 workspace directory*1. This allows development of the flash downloader by using relative paths in consideration of it running on another host PC.

Figure 10.1 and Figure 10.2 show examples of implementation.

Note 1. For information on the current directory, refer to “ARM® Compiler toolchain Developing Software for ARM® Processors, Semihosting” provided by ARM®.

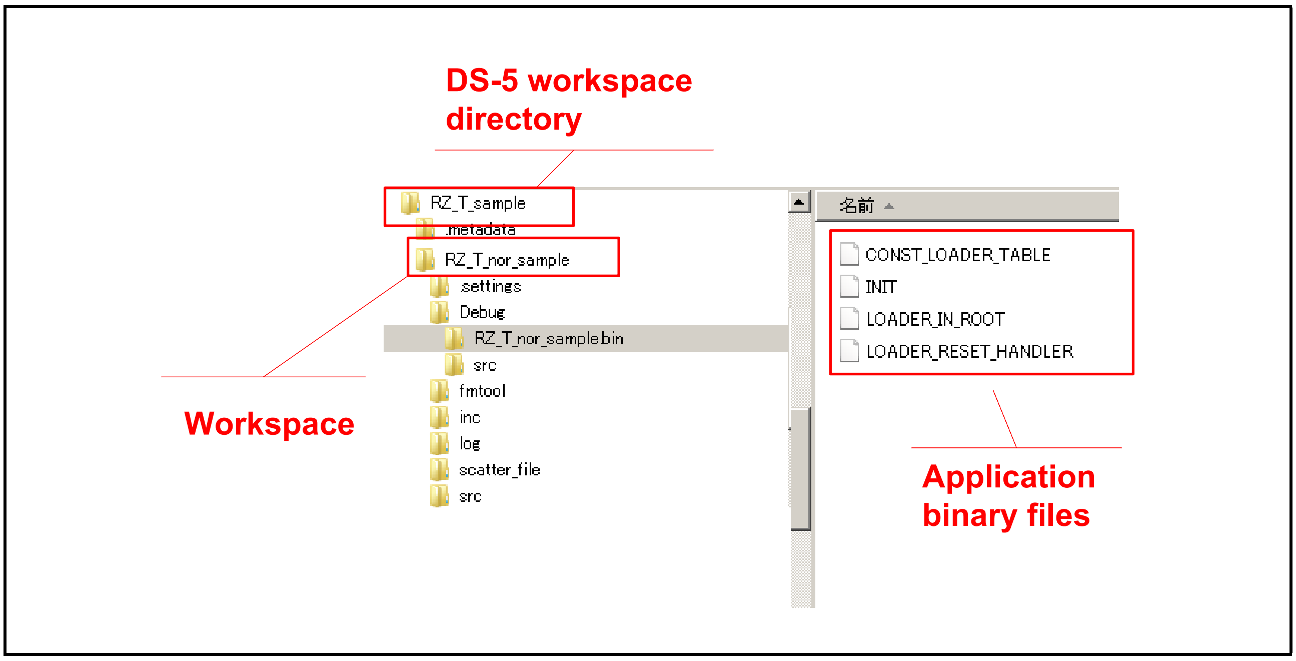


Figure 10.1 Structure of the Directory in the Implementation Example

Before change

```
srcfile = fopen(".¥¥RZ_T_nor_sample¥¥Debug¥¥RZ_T_nor_sample.bin¥¥INIT", "r");
if( srcfile == 0 )
{
    printf("loop=%d, Could not open file. Exiting.¥n", loop);
    return;
}
address = (uint32_t *)0x40020000;
printf("loop=%d, file=INIT, flash address=0x%08x.¥n", loop, (uint32_t)address);

ret = RZ_T1_FlashProgram_Sub( srcfile, address, FLASH_NO );
fclose( srcfile );
if( ret != 0 )
{
    printf("loop=%d, Flash Programming Error!!¥n", loop);
    return;
}
```

After change

```
srcfile = fopen(".¥¥RZ_T_nor_sample¥¥Debug¥¥RZ_T_nor_sample.bin¥¥INIT2", "r");
if( srcfile == 0 )
{
    printf("loop=%d, Could not open file. Exiting.¥n", loop);
    return;
}
address = (uint32_t *)0x40040000;
printf("loop=%d, file=INIT2, flash address=0x%08x.¥n", loop, (uint32_t)address);

ret = RZ_T1_FlashProgram_Sub( srcfile, address, FLASH_NO );
fclose( srcfile );
if( ret != 0 )
{
    printf("loop=%d, Flash Programming Error!!¥n", loop);
    return;
}
```

Figure 10.2 Implementation Example

10.1.2 Changing the Destination Addresses for Writing to the Flash Memory

As with file pathname input, the destination addresses for writing to the flash memory are in the RZ_T1_FlashProgrammer function in the Flash_Programming.c file. The destination addresses can be changed by changing the addresses for writing in the RZ_T1_FlashProgrammer function.

If you use a "scatter file" to set up the image layout, an application binary file is generated for each load module (LOAD_MODULE). The destination addresses for writing each generated application binary file to the flash memory will depend on the image layout which has been set in the application project.

Figure 10.3 shows an example image layout (scatter file) for the application program RZ_T_nor_sample with the memory allocation shown in Figure 8.1. Table 10.1 lists the destination addresses where writing to flash memory is to start for the various application binary files to be generated.

For the implementation examples, see Figure 10.1 and Figure 10.2.

```
LOAD_MODULE1 0x40000000     0x00000200
{
     CONST_LOADER_TABLE     0x40000000     FIXED
    {
        * (CONST_LOADER_TABLE)
    }
}
LOAD_MODULE2 0x40000200     0x00006000
{
    LOADER_RESET_HANDLER     0x00802000     FIXED
    {
        * (LOADER_RESET_HANDLER, +FIRST)
        * (USER_PROG_JUMP)
    }
    Omitted below
}
LOAD_MODULE3 0x40006200     (0x40020000 - 0x40006200)
{
    LOADER_IN_ROOT     0x40006200     FIXED
    {
        * (InRoot$$Sections)
    }
}
LOAD_MODULE4 0x40020000     (0x40120000 - 0x40020000)
{
    INIT               0x00000000     FIXED
    {
        * (VECTOR_TABLE, +FIRST)
        * (RESET_HANDLER)
        * (IRQ_FIQ_HANDLER)
    }
    Omitted below
}
LOAD_MODULE5 0x40120000     (0x44000000 - 0x40120000)
{
    CM3_SECTION        0x40120000     FIXED
    {
        cm3.o(sdram)
    }
}
```

Figure 10.3 Example Image Layout (Scatter File)

Table 10.1 Addresses where Writing to Flash Memory is to Start for the Various Application Binary Files to be Generated

Binary File	Write Start Address	Description
CONST_LOADER_TABLE	H'40000000	Application (1) (loader parameter information) binary file
LOADER_RESET_HANDLER	H'40000200	Application (2) (loader program) binary file
LOADER_IN_ROOT	H'40006200	Application (3) (loader program) binary file
INIT	H'40020000	Application (4) (user program) binary file
CM3_SECTION*1	H'40120000	Application (5) (user program) binary file

Note 1. CM3_SECTION is a binary file for a Cortex-M3 program. For details, refer to the application note “RZ/T1 Group Initial Settings of the Microcomputers Incorporating the R-IN Engine”.

10.2 Customizing the Flash Memory Interface Functions to Suit the Given Flash Memory

This section describes how to change the flash memory interface functions to suit the flash memory used by the customer, as an example of the practical application of the sample program.

Since the flash memory interface functions depend on the device specifications of the flash memory, you may need to customize the program if you change the device.

The sample program in this application note supports flash memory that is compatible with the JEDEC standard command system. If you are using a flash memory of this type, you can reuse the sample program.

If you are using a flash memory with commands of the CUI (command user interface) type, the sample program cannot be reused. In this case, you must create a new program to handle downloading.

Note 1. “JEDEC standard command system” refers to the method of issuing commands by writing to predetermined addresses (H'AAA, H'554, etc.).

Note 2. “CUI command system” refers to the method of issuing the programming command (H'40) and erase command (H'20) to the CUI.

10.2.1 Specifications of the Device for the Sample Program

Table 10.2 and Table 10.3 list the detailed specifications of the device used and the commands used in the sample program.

Table 10.2 Detailed Specifications of the Device Used

Item	Description
Manufacturer	Macronix International Co., Ltd
Product type name	MX29GL512F
Capacity	64 Mbytes
Data bus width	16 bits
Access time	90 ns
Sector structure	Uniform type
Sector size	128 Kbytes
Number of sectors	512
Method of programming	JEDEC standard command system

Table 10.3 Commands Used in the Sample Program

	Description		
Item	**Cycle**	**Address**	**Data**
Erase command (sector erasure)	1st cycle	H'AAA	H'AA
	2nd cycle	H'554	H'55
	3rd cycle	H'AAA	H'80
	4th cycle	H'AAA	H'AA
	5th cycle	H'554	H'55
	6th cycle	SA *1	H'30
Programming command	1st cycle	H'AAA	H'AA
	2nd cycle	H'554	H'55
	3rd cycle	H'AAA	H'A0
	4th cycle	PA*2	PD*3
Reset command	1st cycle	— *4	H'F0

Note 1. SA stands for “sector address”. It specifies the sector address to which the data are to be erased.
Note 2. PA stands for “programming address”. It specifies the address to which the data are to be written.
Note 3. PD stands for “programming data”. It specifies the data to be written.
Note 4. “—” indicates an address in the space where the flash memory is allocated. Any address can be specified as long as it is in the space with flash memory allocated to it.

10.2.2 Boot Types of Flash Memory Available through Customization

The following four boot types of flash memory can be made available by customizing the sample program.

1. Uniform type
2. Bottom-boot type
3. Top-boot type
4. Dual-boot type

Figure 10.4 is an example of the memory map of the boot types of flash memory that are made available by customizing the sample program.

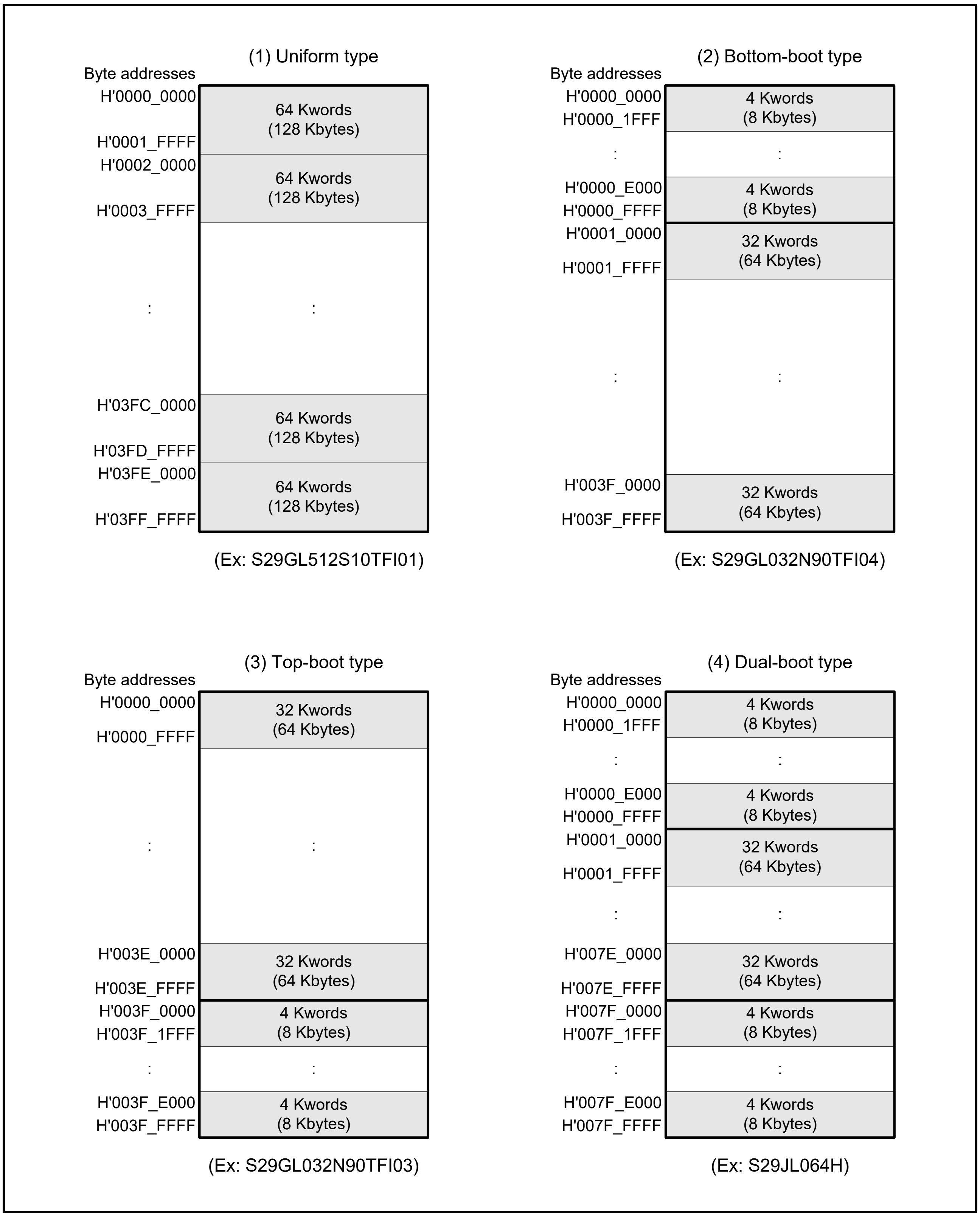


Figure 10.4 Boot Types of Flash Memory that are Made Available by Customizing the Sample Program

10.2.3 Details of Customization

The tables below list the cases where customization is required and the details of the modification.

Table 10.4 Cases where Customization is Required and Details of Customization (1/2)

Item	Details of Customization
Cases where the boot type of flash memory is different	The sample program supports erasure and programming of the flash memory with the uniform type allocation. If you are using flash memory with the top-boot type, bottom-boot type, or dual-boot type allocation, change the macro definition of FM_BOOT_TYPE to the following values. Top-boot type: FM_TOP_BOOT Bottom-boot type: FM_BOTTOM_BOOT Dual-boot type: FM_DUAL_BOOT The initial value defines “FM_UNIFORM” of uniform type.
Case 1 of changing the allocation of flash memory: bottom-boot type	Change the sector definitions. • FM_B_BOOT_SECTOR_START • FM_B_BOOT_SECTOR_SIZE • FM_B_BOOT_SECTOR_NUM • FM_NORMAL_SECTOR_START • FM_NORMAL_SECTOR_SIZE • FM_NORMAL_SECTOR_NUM The initial values of the following sector definitions are “0”. Use these definitions unchanged. • FM_T_BOOT_SECTOR_START • FM_T_BOOT_SECTOR_SIZE • FM_T_BOOT_SECTOR_NUM
Case 2 of changing the allocation of flash memory: top-boot type	Change the sector definitions. • FM_NORMAL_SECTOR_START • FM_NORMAL_SECTOR_SIZE • FM_NORMAL_SECTOR_NUM • FM_T_BOOT_SECTOR_START • FM_T_BOOT_SECTOR_SIZE • FM_T_BOOT_SECTOR_NUM The initial values of the following sector definitions are “0”. Use these definitions unchanged. • FM_B_BOOT_SECTOR_START • FM_B_BOOT_SECTOR_SIZE • FM_B_BOOT_SECTOR_NUM

Note: Since the flash memory interface functions depend on the specifications of the flash memory, check the data sheet of the device you are using and modify the flash memory interface functions according to the specifications of the device.

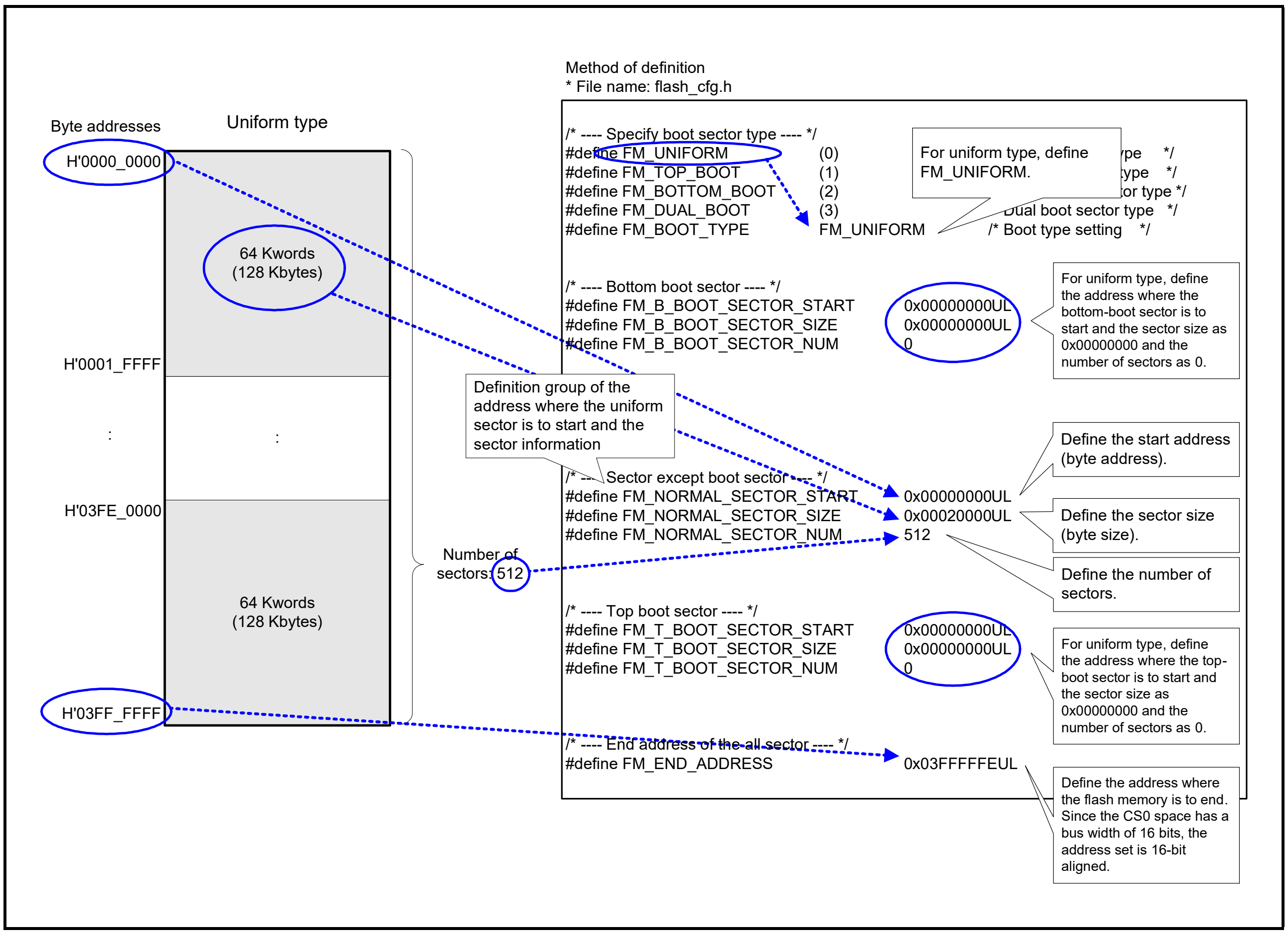
Table 10.5 Cases where Customization is Required and Details of Customization (2/2)

Item	Details of Customization
Case 3 of changing the allocation of flash memory: dual-boot type	Change the sector definitions. • FM_B_BOOT_SECTOR_START • FM_B_BOOT_SECTOR_SIZE • FM_B_BOOT_SECTOR_NUM • FM_NORMAL_SECTOR_START • FM_NORMAL_SECTOR_SIZE • FM_NORMAL_SECTOR_NUM • FM_T_BOOT_SECTOR_START • FM_T_BOOT_SECTOR_SIZE • FM_T_BOOT_SECTOR_NUM
Cases where the sector size and the number of sectors are different	Change the sector definitions. • FM_NORMAL_SECTOR_START • FM_NORMAL_SECTOR_SIZE • FM_NORMAL_SECTOR_NUM The initial values of the following sector definitions are “0”. Use these definitions unchanged. • FM_B_BOOT_SECTOR_START • FM_B_BOOT_SECTOR_SIZE • FM_B_BOOT_SECTOR_NUM • FM_T_BOOT_SECTOR_START • FM_T_BOOT_SECTOR_SIZE • FM_T_BOOT_SECTOR_NUM
Cases where the memory capacity is different	Change the setting of the FM_END_ADDRESS macro.
Cases where the boot type is different from any of the four shown in Figure 10.4	Customization of the flash memory operation functions is required. For details, refer to the sample program.
Cases where the specifications of the erasure, programming, and reset commands are different	
Cases where the programming command is for CUI command system	
Cases where the RZ/T1 group microcontroller and flash memory are connected with a width other than 16 bits	

Note: Since the flash memory interface functions depend on the specifications of the flash memory, check the data sheet of the device you are using and modify the flash memory interface functions according to the specifications of the device.

10.2.4 Customizing the Sector Size and Number of Sectors of Uniform Type

Figure 10.5 shows how to customize the sector size and number of sectors of the uniform type flash memory. If you are using the MX29GL512F flash memory and uniform type flash memory with a different sector size and number of sectors, change the setting of the macro for the sector information of uniform type. When the capacity of the flash memory is different, also change the setting of the macro for the end address.

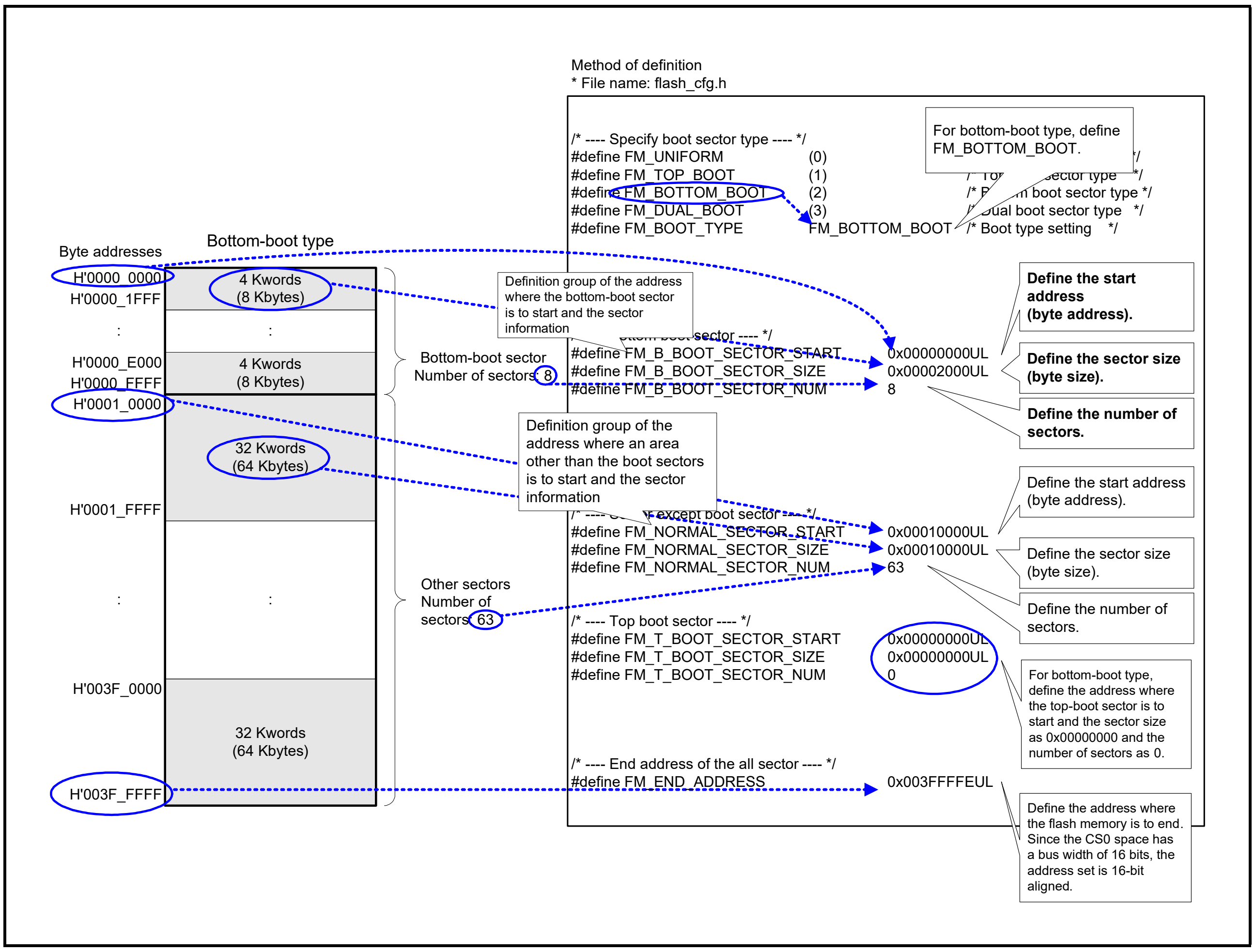

Byte addresses
Uniform type
H'0000_0000
64 Kwords
(128 Kbytes)
H'0001_FFFF
:
:
H'03FE_0000
64 Kwords
(128 Kbytes)
H'03FF_FFFF
Number of
sectors: 512
Method of definition
* File name: flash_cfg.h
For uniform type, define FM_UNIFORM.
For uniform type, define the address where the bottom-boot sector is to start and the sector size as 0x00000000 and the number of sectors as 0.
Definition group of the address where the uniform sector is to start and the sector information
Define the start address (byte address).
Define the sector size (byte size).
Define the number of sectors.
For uniform type, define the address where the top-boot sector is to start and the sector size as 0x00000000 and the number of sectors as 0.
Define the address where the flash memory is to end. Since the CS0 space has a bus width of 16 bits, the address set is 16-bit aligned.

```
/* ---- Specify boot sector type ---- */
#define FM_UNIFORM                    (0)                              ype    */
#define FM_TOP_BOOT                   (1)                              type   */
#define FM_BOTTOM_BOOT                (2)                              tor type */
#define FM_DUAL_BOOT                  (3)                  ual boot sector type   */
#define FM_BOOT_TYPE                  FM_UNIFORM           /* Boot type setting    */

/* ---- Bottom boot sector ---- */
#define FM_B_BOOT_SECTOR_START        0x00000000UL
#define FM_B_BOOT_SECTOR_SIZE         0x00000000UL
#define FM_B_BOOT_SECTOR_NUM          0

/* ---- Sector except boot sector ---- */
#define FM_NORMAL_SECTOR_START        0x00000000UL
#define FM_NORMAL_SECTOR_SIZE         0x00020000UL
#define FM_NORMAL_SECTOR_NUM          512

/* ---- Top boot sector ---- */
#define FM_T_BOOT_SECTOR_START        0x00000000UL
#define FM_T_BOOT_SECTOR_SIZE         0x00000000UL
#define FM_T_BOOT_SECTOR_NUM          0

/* ---- End address of the all sector ---- */
#define FM_END_ADDRESS                0x03FFFFFEUL
```

Figure 10.5 Customizing the Sector Size and Number of Sectors of the Uniform Type Flash Memory

10.2.5 Customizing the Boot Type of Flash Memory to Bottom-Boot Type

Figure 10.6 shows how to customize the boot type of flash memory to bottom-boot type. If you are using a bottom-boot type flash memory, change the settings of the macros for the sector information of two different areas: the bottom-boot sector area and areas other than the boot sectors. When the capacity of the flash memory is different, also change the setting of the macro for the end address.

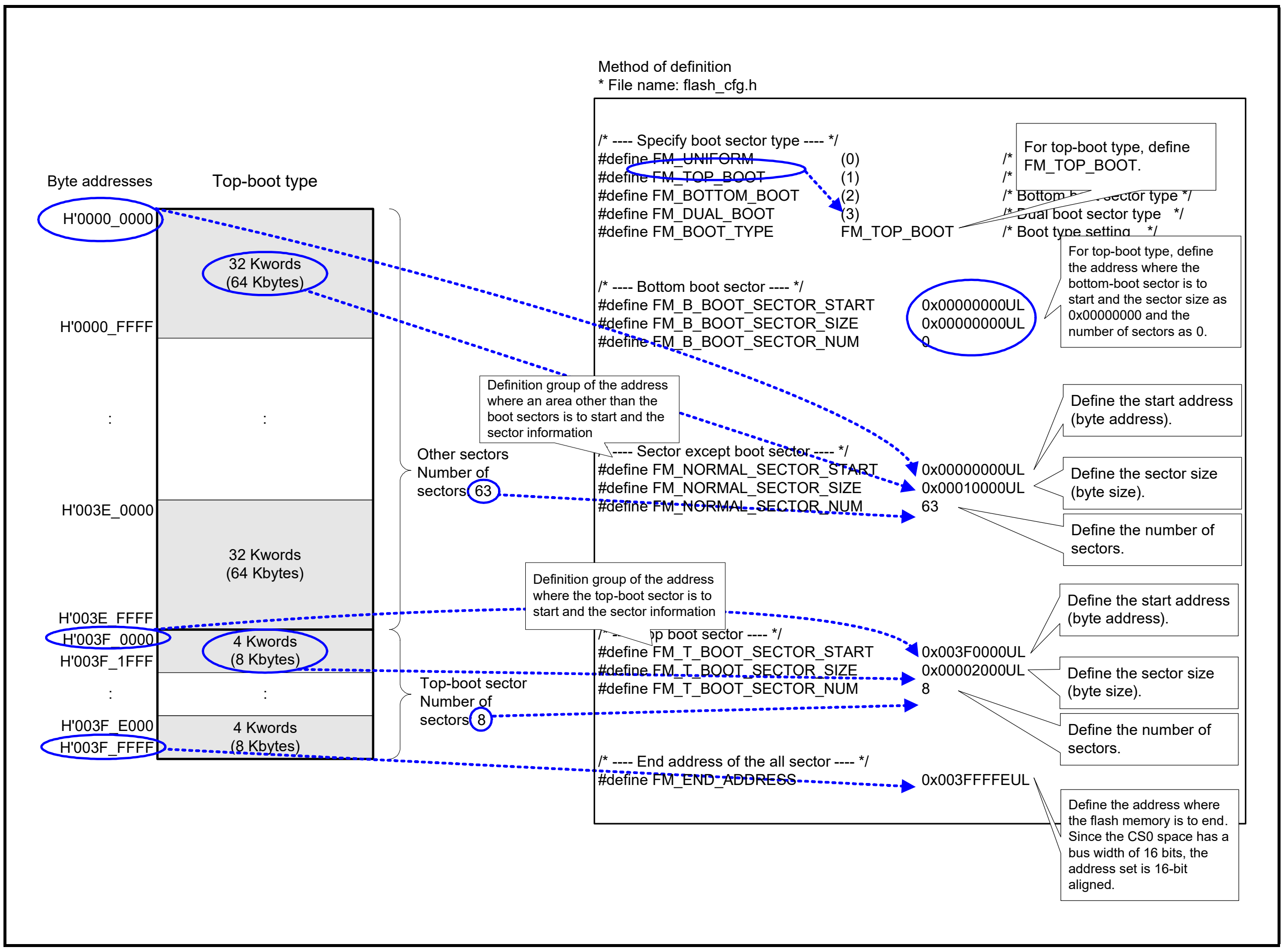

Byte addresses
Bottom-boot type
H'0000_0000
H'0000_1FFF
:
H'0000_E000
H'0000_FFFF
H'0001_0000
H'0001_FFFF
:
H'003F_0000
H'003F_FFFF
4 Kwords (8 Kbytes)
:
4 Kwords (8 Kbytes)
32 Kwords (64 Kbytes)
:
32 Kwords (64 Kbytes)
Bottom-boot sector Number of sectors 8
Other sectors Number of sectors 63
Method of definition
* File name: flash_cfg.h
For bottom-boot type, define FM_BOTTOM_BOOT.
Definition group of the address where the bottom-boot sector is to start and the sector information
Definition group of the address where an area other than the boot sectors is to start and the sector information
Define the start address (byte address).
Define the sector size (byte size).
Define the number of sectors.
Define the start address (byte address).
Define the sector size (byte size).
Define the number of sectors.
For bottom-boot type, define the address where the top-boot sector is to start and the sector size as 0x00000000 and the number of sectors as 0.
Define the address where the flash memory is to end. Since the CS0 space has a bus width of 16 bits, the address set is 16-bit aligned.

```
/* ---- Specify boot sector type ---- */
#define FM_UNIFORM          (0)                              */
#define FM_TOP_BOOT         (1)              /* Top      sector type      */
#define FM_BOTTOM_BOOT      (2)              /* B      m boot sector type */
#define FM_DUAL_BOOT        (3)              /*  ual boot sector type     */
#define FM_BOOT_TYPE        FM_BOTTOM_BOOT   /* Boot type setting         */

/*      ottom boot sector ---- */
#define FM_B_BOOT_SECTOR_START        0x00000000UL
#define FM_B_BOOT_SECTOR_SIZE         0x00002000UL
#define FM_B_BOOT_SECTOR_NUM          8

/* ----       except boot sector ---- */
#define FM_NORMAL_SECTOR_START        0x00010000UL
#define FM_NORMAL_SECTOR_SIZE         0x00010000UL
#define FM_NORMAL_SECTOR_NUM          63

/* ---- Top boot sector ---- */
#define FM_T_BOOT_SECTOR_START        0x00000000UL
#define FM_T_BOOT_SECTOR_SIZE         0x00000000UL
#define FM_T_BOOT_SECTOR_NUM          0

/* ---- End address of the all sector ---- */
#define FM_END_ADDRESS                0x003FFFFEUL
```

Figure 10.6 Customizing the Boot Type of Flash Memory to Bottom-Boot Type

10.2.6 Customizing the Boot Type of Flash Memory to Top-Boot Type

Figure 10.7 shows how to customize the boot type of flash memory to top-boot type. If you are using a top-boot type flash memory, change the settings of the macros for the sector information of two different areas: the top-boot sector area and areas other than the boot sectors. When the capacity of the flash memory is different, also change the setting of the macro for the end address.


Byte addresses
Top-boot type
H'0000_0000
32 Kwords (64 Kbytes)
H'0000_FFFF
:
:
H'003E_0000
32 Kwords (64 Kbytes)
H'003E_FFFF
H'003F_0000
4 Kwords (8 Kbytes)
H'003F_1FFF
:
:
H'003F_E000
4 Kwords (8 Kbytes)
H'003F_FFFF
Other sectors Number of sectors 63
Top-boot sector Number of sectors 8
Definition group of the address where an area other than the boot sectors is to start and the sector information
Definition group of the address where the top-boot sector is to start and the sector information
For top-boot type, define FM_TOP_BOOT.
For top-boot type, define the address where the bottom-boot sector is to start and the sector size as 0x00000000 and the number of sectors as 0.
Define the start address (byte address).
Define the sector size (byte size).
Define the number of sectors.
Define the start address (byte address).
Define the sector size (byte size).
Define the number of sectors.
Define the address where the flash memory is to end. Since the CS0 space has a bus width of 16 bits, the address set is 16-bit aligned.

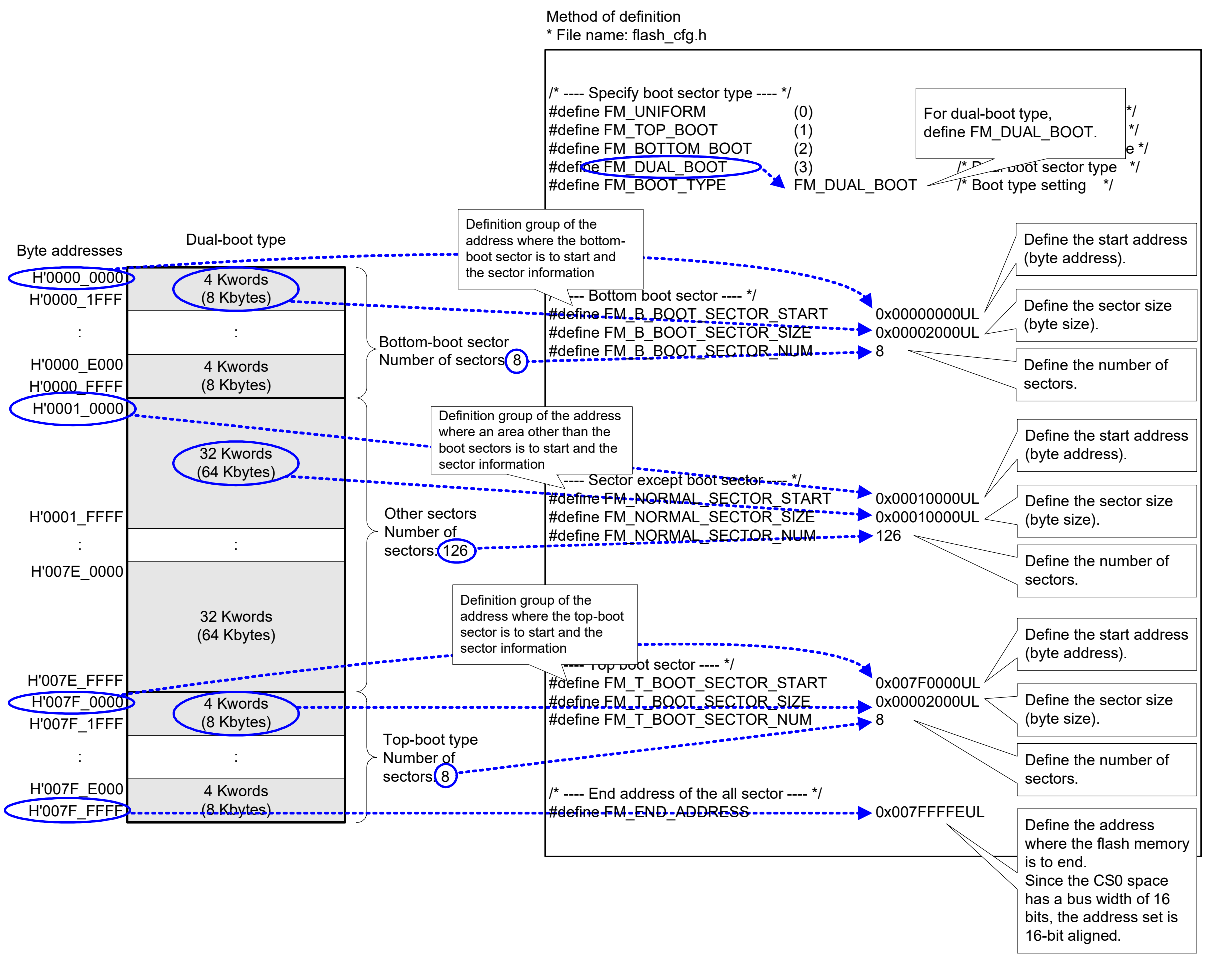
Method of definition
* File name: flash_cfg.h

```
/* ---- Specify boot sector type ---- */
#define FM_UNIFORM              (0)             /*
#define FM_TOP_BOOT             (1)             /*
#define FM_BOTTOM_BOOT          (2)             /* Bottom [illegible] sector type */
#define FM_DUAL_BOOT            (3)             /* Dual boot sector type  */
#define FM_BOOT_TYPE            FM_TOP_BOOT     /* Boot type setting   */

/* ---- Bottom boot sector ---- */
#define FM_B_BOOT_SECTOR_START          0x00000000UL
#define FM_B_BOOT_SECTOR_SIZE           0x00000000UL
#define FM_B_BOOT_SECTOR_NUM            0

/* ---- Sector except boot sector ---- */
#define FM_NORMAL_SECTOR_START          0x00000000UL
#define FM_NORMAL_SECTOR_SIZE           0x00010000UL
#define FM_NORMAL_SECTOR_NUM            63

/* ---- [illegible]p boot sector ---- */
#define FM_T_BOOT_SECTOR_START          0x003F0000UL
#define FM_T_BOOT_SECTOR_SIZE           0x00002000UL
#define FM_T_BOOT_SECTOR_NUM            8

/* ---- End address of the all sector ---- */
#define FM_END_ADDRESS                  0x003FFFFEUL
```

Figure 10.7 Customizing the Boot Type of Flash Memory to Top-Boot Type

10.2.7 Customizing the Boot Type of Flash Memory to Dual-Boot Type

Figure 10.8 shows how to customize the boot type of flash memory to dual-boot type. If you are using a dual-boot type flash memory, change the settings of the macros for the sector information of three different areas: the top-boot sector area, bottom-boot sector area, and areas other than the boot sectors. When the capacity of the flash memory is different, also change the setting of the macro for the end address.


Method of definition
* File name: flash_cfg.h

```
/* ---- Specify boot sector type ---- */
#define FM_UNIFORM               (0)                                    */
#define FM_TOP_BOOT              (1)                                    */
#define FM_BOTTOM_BOOT           (2)                                   e */
#define FM_DUAL_BOOT             (3)                  /* D    boot sector type   */
#define FM_BOOT_TYPE             FM_DUAL_BOOT         /* Boot type setting    */

/* ---- Bottom boot sector ---- */
#define FM_B_BOOT_SECTOR_START          0x00000000UL
#define FM_B_BOOT_SECTOR_SIZE           0x00002000UL
#define FM_B_BOOT_SECTOR_NUM            8

/* ---- Sector except boot sector ---- */
#define FM_NORMAL_SECTOR_START          0x00010000UL
#define FM_NORMAL_SECTOR_SIZE           0x00010000UL
#define FM_NORMAL_SECTOR_NUM            126

/* ---- Top boot sector ---- */
#define FM_T_BOOT_SECTOR_START          0x007F0000UL
#define FM_T_BOOT_SECTOR_SIZE           0x00002000UL
#define FM_T_BOOT_SECTOR_NUM            8

/* ---- End address of the all sector ---- */
#define FM_END_ADDRESS                  0x007FFFFEUL
```

For dual-boot type, define FM_DUAL_BOOT.

Definition group of the address where the bottom-boot sector is to start and the sector information

Definition group of the address where an area other than the boot sectors is to start and the sector information

Definition group of the address where the top-boot sector is to start and the sector information

Define the start address (byte address).

Define the sector size (byte size).

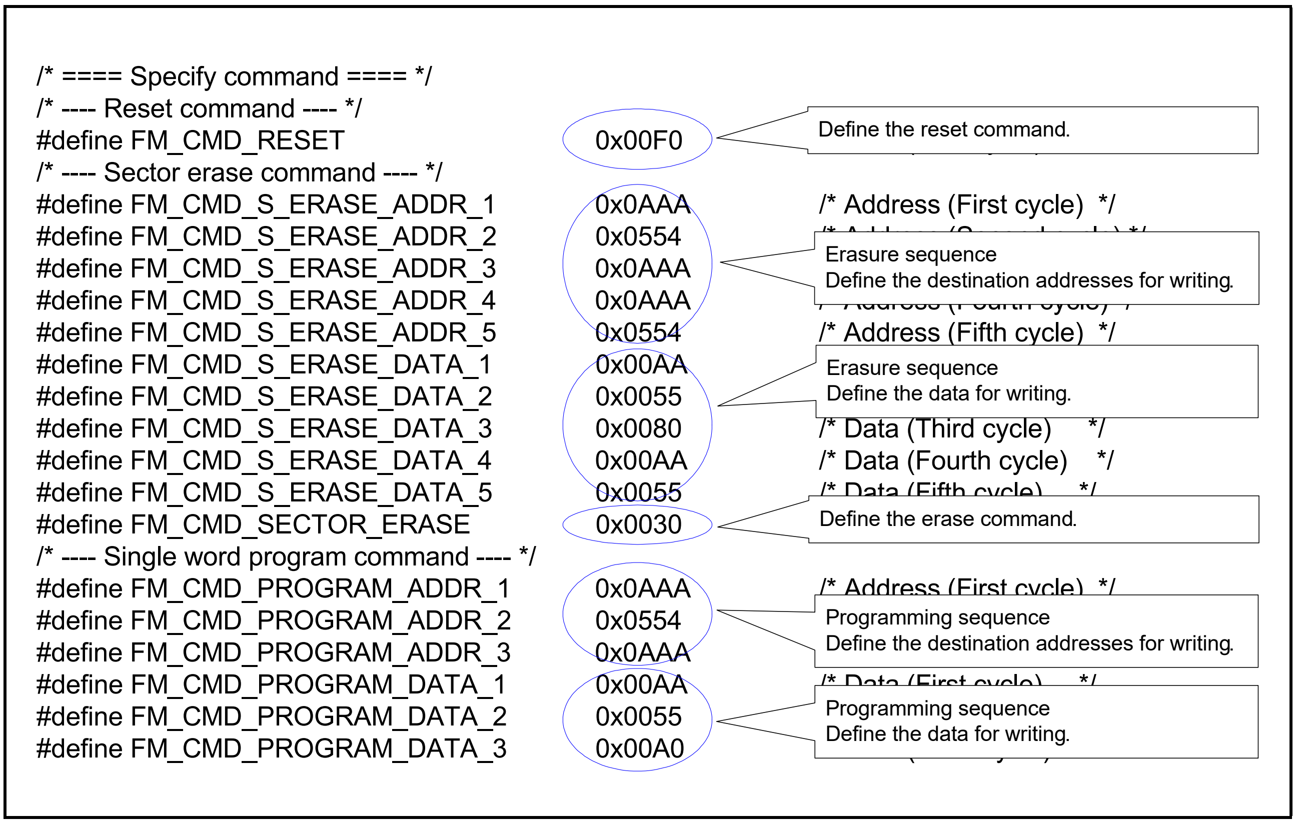
Define the number of sectors.

Define the address where the flash memory is to end.
Since the CS0 space has a bus width of 16 bits, the address set is 16-bit aligned.

Figure 10.8 Customizing the Boot Type of Flash Memory to Dual-Boot Type

10.2.8 Checking Flash Memory Commands

Figure 10.9 shows the command definitions of the flash memory. The sample program supports memory that is compatible with the JEDEC standard command system and defines the commands for the RZ/T1 group microcontroller when the width of the connection with the flash memory is 16 bits. Refer to the commands in word mode (x16-bit mode) in the data sheet of the flash memory and check that there are no differences in the values for the reset command, sector erase commands, and programming commands (write commands). The sample program does not support page programming (page writing).


Define the reset command.
Erasure sequence
Define the destination addresses for writing.
Erasure sequence
Define the data for writing.
Define the erase command.
Programming sequence
Define the destination addresses for writing.
Programming sequence
Define the data for writing.

```
/* ==== Specify command ==== */
/* ---- Reset command ---- */
#define FM_CMD_RESET                 0x00F0
/* ---- Sector erase command ---- */
#define FM_CMD_S_ERASE_ADDR_1        0x0AAA    /* Address (First cycle)  */
#define FM_CMD_S_ERASE_ADDR_2        0x0554
#define FM_CMD_S_ERASE_ADDR_3        0x0AAA
#define FM_CMD_S_ERASE_ADDR_4        0x0AAA
#define FM_CMD_S_ERASE_ADDR_5        0x0554    /* Address (Fifth cycle)  */
#define FM_CMD_S_ERASE_DATA_1        0x00AA
#define FM_CMD_S_ERASE_DATA_2        0x0055
#define FM_CMD_S_ERASE_DATA_3        0x0080    /* Data (Third cycle)     */
#define FM_CMD_S_ERASE_DATA_4        0x00AA    /* Data (Fourth cycle)    */
#define FM_CMD_S_ERASE_DATA_5        0x0055
#define FM_CMD_SECTOR_ERASE          0x0030
/* ---- Single word program command ---- */
#define FM_CMD_PROGRAM_ADDR_1        0x0AAA
#define FM_CMD_PROGRAM_ADDR_2        0x0554
#define FM_CMD_PROGRAM_ADDR_3        0x0AAA
#define FM_CMD_PROGRAM_DATA_1        0x00AA
#define FM_CMD_PROGRAM_DATA_2        0x0055
#define FM_CMD_PROGRAM_DATA_3        0x00A0
```

Figure 10.9 Command Definitions of Flash Memory (flash_cfg.h)

10.3 Customizing the Sample Program for Initial Settings of the Microcomputers Incorporating the R-IN Engine (Cortex-M3)

In the sample program for initial settings of devices with a built-in R-IN Engine, CM3_SECTION is generated as indicated in Table 10.6 when an application binary file is generated. In downloading, a binary file for the Cortex-M3 is also downloaded as processing for devices with a built-in R-IN Engine (loop = 5) and the messages are displayed in the application console as shown in Figure 10.10.

For details of initial settings of devices with a built-in R-IN Engine, refer to the application note “RZ/T1 Group Initial Settings of the Microcomputers Incorporating the R-IN Engine”.

Table 10.6 Application Binary Files

Application Binary File		Description
RZ_T_nor_sample.bin	CONST_LOADER_TABLE	Application (1) (loader parameter information) binary file
	LOADER_RESET_HANDLER	Application (2) (loader program) binary file
	LOADER_IN_ROOT	Application (3) (loader program) binary file
	INIT	Application (4) (user program) binary file
	CM3_SECTION	Application (5) (user program) binary file (Cortex-M3 program)

```
RZ/T1 CPU Board NOR-Flash Programming Sample. Ver.1.00
Copyright (C) 2015 Renesas Electronics Corporation. All rights reserved.

Initializing Flash...
Start to load Binary Data to Flash Memory.
loop=1, file=CONST_LOADER_TABLE, flash address=0x40000000.
Calculating Data Size...
Data Size is 76
Programing Flash...
Calcurating Checksum of Loader Parameter.
Verifying Flash...
loop=1, Flash Programming Success!!
loop=2, file=LOADER_RESET_HANDLER, flash address=0x40000200.
Calculating Data Size...
Data Size is 3288
Programing Flash...
Verifying Flash...
loop=2, Flash Programming Success!!
loop=3, file=LOADER_IN_ROOT, flash address=0x40006200.
Calculating Data Size...
Data Size is 192
Programing Flash...
Verifying Flash...
loop=3, Flash Programming Success!!
loop=4, file=INIT, flash address=0x40020000.
Calculating Data Size...
Data Size is 3008
Programing Flash...
Verifying Flash...
loop=4, Flash Programming Success!!
loop=5, file=CM3_SECTION, flash address=0x40120000.
Calculating Data Size...
Data Size is 1296
Programing Flash...
Verifying Flash...
loop=5, Flash Programming Success!!

 finish
Flash Programming Complete
```

Figure 10.10 Messages Output to the Application Console (when the Device with a Built-in R-IN Engine is in Use)

11. Sample Program

The sample program is available from the Renesas Electronics website.

12. Documents for Reference

- User's Manual: Hardware
  RZ/T1 Group User's Manual: Hardware
  (Download the latest version from the Renesas Electronics website.)

  RZ/T1 Evaluation Board RTK7910022C00000BR User's Manual
  (Download the latest version from the Renesas Electronics website.)

  ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C
  (Download the latest version from the ARM® website.)

  ARM Generic Interrupt Controller Architecture Specification Architecture version 1.0
  (Download the latest version from the ARM® website.)

- Technical Update and Technical News
  (Download the latest version from the Renesas Electronics website.)

- User's Manual: Development Environment
  For the ARM® software development tools (ARM Compiler toolchain, ARM DS-5, etc.), visit the ARM® website.
  (Download the latest version from the ARM® website.)

Website and Support

Renesas Electronics website

http://www.renesas.com/

Inquiries

http://www.renesas.com/inquiry

Revision History	Application Note: Example of Downloading to NOR Flash Memory by Using "Semihosting" of ARM® Development Studio 5 (DS-5™)

Rev.	Date	Description	
		Page	**Summary**
1.00	Nov. 17, 2015	—	First Edition issued
1.10	Sep. 16, 2016	All	Application Note "RZ/T1 Group Dual Core Control" changed to Application Note "RZ/T1 Group Initial Settings of the Microcomputers Incorporating the R-IN Engine"
1.20	Sep. 15, 2017	2. Conditions for Checking Operations	
		5	Table 2.1 Conditions for Checking Operations DS-5 version from ARM®, modified

General Precautions in the Handling of MPU/MCU Products

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

1. Handling of Unused Pins
   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.
   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.
   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

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

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

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

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

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

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

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, 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 India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 6.0