User's Manual

16

RL78 Family

Flash Self-Programming Library Type 01
Japanese Release

Installer name: RENESAS_RL78_FSL_T01_xVxx

16-Bit Single-Chip Microcontrollers

Renesas Electronics
www.renesas.com

Rev.1.10 Dec 2023

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit 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. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. 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 the 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.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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.

HOW TO USE THIS MANUAL

Readers

This manual is intended for user engineers who wish to understand the functions of the Flash Self-Programming Library Type 01 for the RL78 microcontrollers and design and develop application systems and programs for these devices.
Refer to the following list for the target MCUs.

Self-Programming Library (Japanese Release) and Supported MCUs (R20UT2861XJxxxx)
RL78 Family Self RAM list of Flash Self Programming Library (R20UT2944)

Purpose

This manual is intended to give users an understanding of the methods (described in the **Organization** below) for using flash self-programming library Type 01 to rewrite the code flash memories.

Organization

This user's manual is separated into the following parts:

- Overview
- Programming Environment
- Interrupts During Execution of Flash Self-programming
- Security Setting
- Boot Swap Function
- Flash Self-Programming Function

How to Read This Manual

It is assumed that the readers of this manual have general knowledge of electrical engineering, logic circuits, and microcontrollers.

- To gain a general understanding of functions:
  → Read this manual in the order of the **CONTENTS**.
- To know details of the RL78 Microcontroller instructions:
  → Refer to **CHAPTER 6 FLASH FUNCTION**.

Conventions

Data significance:	Higher digits on the left and lower digits on the right		
Active low representations:	$\overline{\times\times\times}$ (overscore over pin and signal name)		
Note:	Footnote for item marked with **Note** in the text		
Caution:	Information requiring particular attention		
Remark:	Supplementary information		
Numerical representations:	Binary	···×××× or ××××B	
	Decimal	···××××	
	Hexadecimal	···××××H or '0x'xxxx	

CONTENTS

RL78 Family
Flash Self-Programming Library Type 01
R01US0050EJ0110
Rev.1.10
Dec 26, 2023

CHAPTER 1 OVERVIEW

1. 1 Overview

The flash self-programming library is software to rewrite data in the code flash memory with the firmware installed on the RL78 microcontroller.

The content of the code flash memory can be rewritten by calling the flash self-programming library from the user program, which can significantly shorten the software development period.

Use this Flash Self-Programming Library User's Manual along with the manual of the target device.

Terms The meanings of the terms used in this manual are described below.

- Flash self-programming
  Write operation to the code flash memory by the user program itself.
- Flash self-programming library
  Library for code flash memory operation with the functions provided by the RL78 microcontroller.
  Operation to the data flash memory cannot be done.
- Flash environment
  State in which operation to the code flash memory is available. There are restrictions different from those in the execution of normal programs. Operation to the data flash memory cannot be done.
- Block number
  Number indicating a block of flash memory. Operation unit for erasure, blank check, and verification (internal verification).
- Boot cluster
  Boot area used for boot swapping. For the availability of the boot swap function, refer to the user's manual of the target RL78 microcontroller.
- Internal verification
  To check if the signal level of the flash memory cell is appropriate after writing to the flash memory. If an error is detected in internal verification, the device is determined as failed. However, if data erasure, data writing, and internal verification are performed and completed normally after the internal verification error, the device is determined as normal.
- FSL
  Abbreviation of "Flash Self-Programming Library."
- FSW
  Abbreviation of "Flash Shield Window."
- Flash function
  Function comprising the flash self-programming library.

- Sequencer
  The RL78 microcontroller has a dedicated circuit for controlling the flash memory. In this document, this circuit is called the "sequencer."
- BGO (background operation)
  State in which rewriting of the flash memory can be done while operating the user program by letting the sequencer to control the flash memory.
- Status check
  When the sequencer is used, the processing to check the state of the sequencer (state of control for the flash memory) with the program controlling the flash memory is required. In this document, the processing to check the state of the sequencer is called "status checking."
- ROMization (program)
  In flash self-programming of the RL78 microcontroller, user programs and flash self-programming library need to be allocated in the RAM to perform the processing depending on the control method. In this document, allocating the program for operating on the RAM in the code flash memory to use it is called "ROMization."
  To perform ROMization, the functions such as the development tools need to be used.
- EEPROM emulation library
  Software library that provides the function to store data in the installed flash memory like an EEPROM.
- Data flash library
  Software library to perform operation to the data flash memory.

1. 2 Calling Flash Self-Programming Library

To perform flash self-programming, the initialization processing for flash self-programming and the functions corresponding to the functions used need to be executed from the user program in the C language or assembly language.

In Flash Self-Programming Library Type 01, the code flash memory cannot be referred to while it is being rewritten. Therefore, some segments of the flash self-programming library or the user program need to be allocated on the RAM depending on the usage.

Figure 1-1 shows the state transition diagram of flash self-programming. Figure 1-2 shows an example of the code flash memory rewriting flow by using the flash self-programming library. Figure 1-3 shows an example of the code flash memory rewriting flow during background operation (BGO).

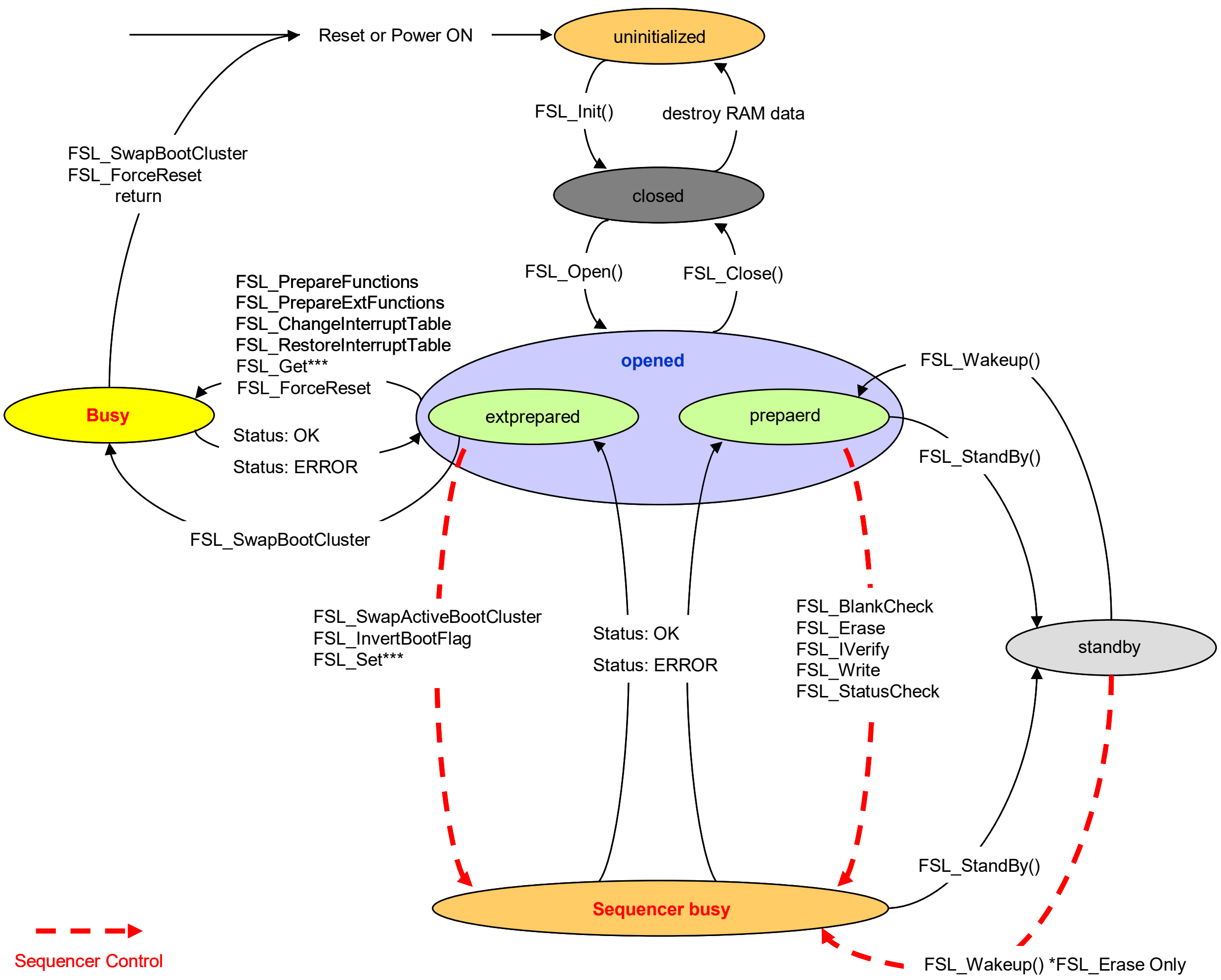


Figure 1-1. State Transition Diagram of Flash Self-Programming

[Overview of the state transition diagram]
To operate the code flash memory by using the flash self-programming library, the provided functions need to be executed sequentially to perform processing.

(1) uninitialized
State at Power ON and Reset. A transition to this state occurs also when the EEPROM emulation library or the data flash library is executed.

(2) closed
State in which the FSL_Init() function has been executed and the data to execute the flash self-programming has been initialized (operation to the code flash memory is stopped). To execute the EEPROM emulation library, the data flash library, STOP mode, or HALT mode after operating flash self-programming, execute FSL_Close from the opened state to make a transition to this state.

(3) opened
State in which the FSL_Open() function has been executed from the closed state and flash self-programming can be executed. This state is called the "flash environment." In the period from the execution of FSL_Close to the transition to the closed state, the EEPROM emulation library, the data flash library, STOP mode, or HALT mode cannot be executed.

(4) prepared
State in which the FSL_PrepareFunctions() function has been executed from the opened state and operations to the code flash memory such as writing, and erasure are enabled.

(5) extprepared
State in which the FSL_PrepareFunctions() and FSL_PrepareExtFunctions() function have been executed from the opened state in that order and rewriting of the security flag and boot swap processing can be executed.

(6) busy
State in which the specified processing is being executed. The resulting transition may change depending on the executed function and end state.

(7) sequencer busy
State in which the specified processing is being executed with the sequencer. The code flash memory cannot be referred to while the sequencer is being used. The resulting transition may change depending on the executed function and end state.

(8) standby
State in which flash self-programming is paused by the FSL_StandBy function. Flash self-programming can be restarted by using the FSL_WakeUp function. When a pause occurred during the execution of the FSL_Erase function, the processing of the FSL_Erase function is restarted.

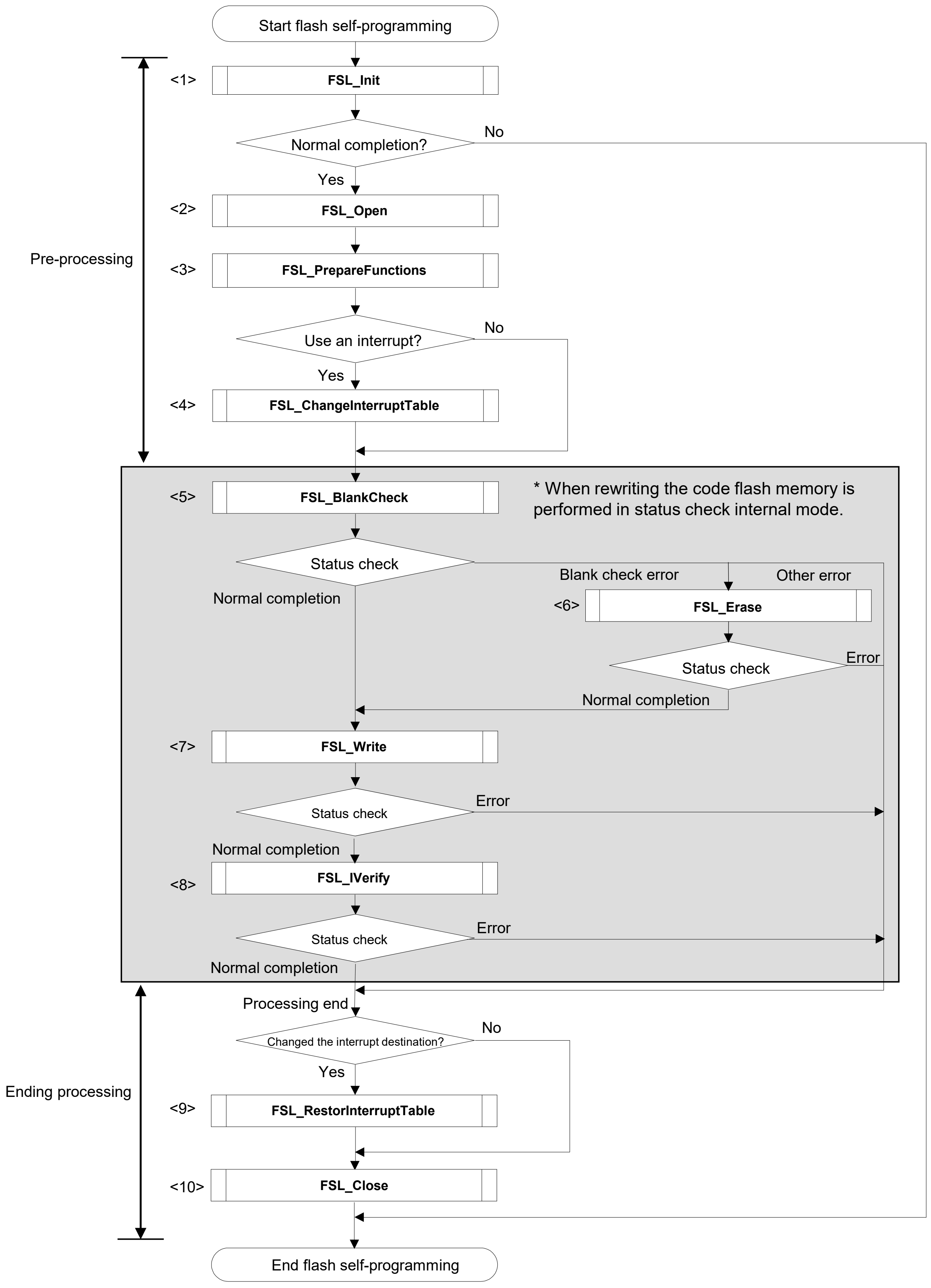


Figure 1-2. Example of Flow of Flash Self-Programming (Rewriting of Code Flash Memory)

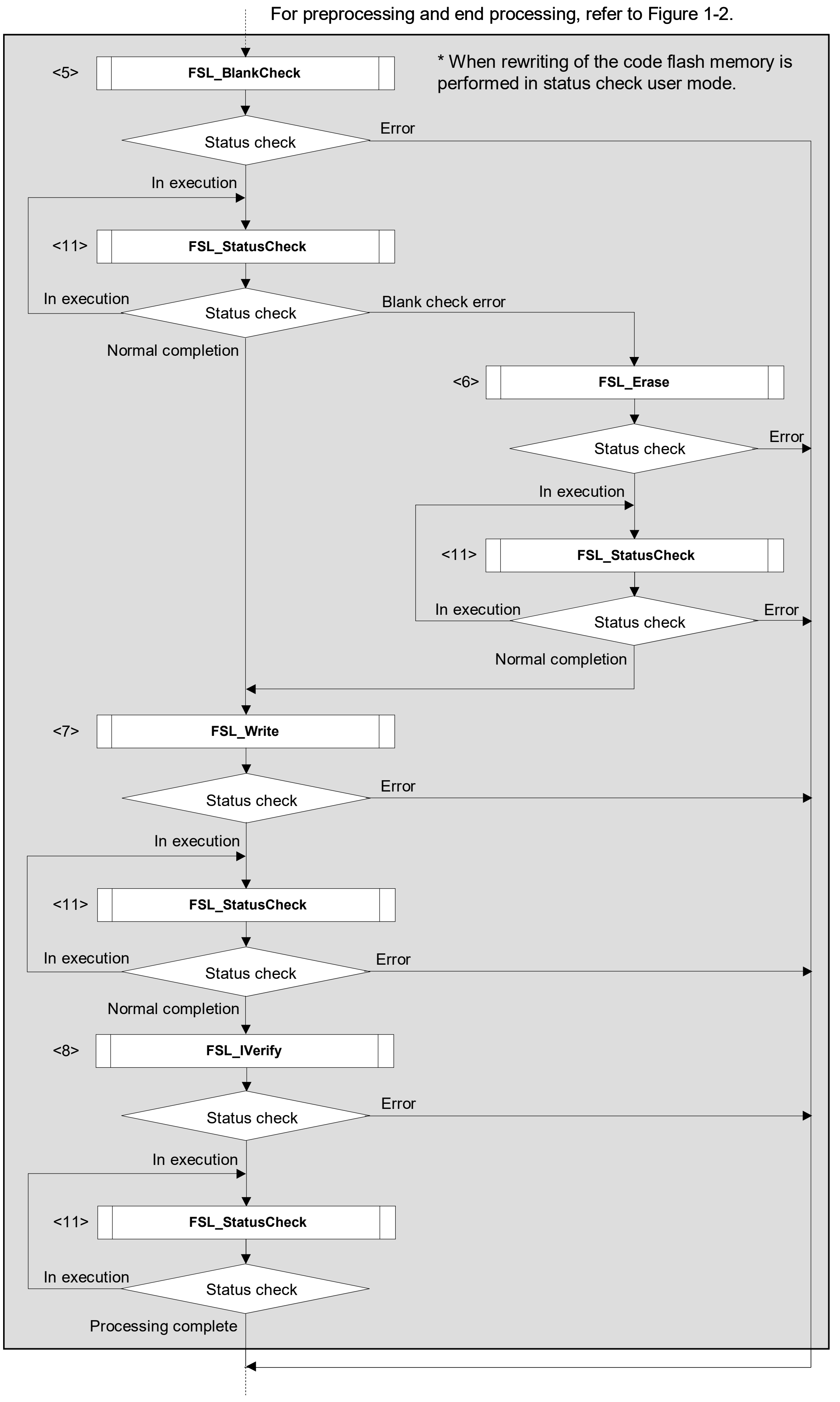


Figure 1-3. Example of Flow of Code Flash Memory Rewriting During Background Operation

<1> Initializing the RAM used for flash self-programming

The FSL_Init function is called to initialize the RAM used for flash self-programming and to set the parameters required for the operation.

<2> Starting the flash environment

The FSL_Open function is called to make flash self-programming available.

<3> Preparation processing

The FSL_PrepareFunctions function is called to prepare the functions used for flash self-programming. To use extension functions, the FSL_PrepareExtFunctions function must also be called.
For details of the FSL_PrepareFunctions function and FSL_PrepareExtFunctions function, refer to **CHAPTER 6 FLASH FUNCTIONS**.

<4> Changing interrupt reception to the RAM

When an interrupt is required during the execution of flash self-programming, the FSL_ChangeInterruptTable function is called to change the interrupt destination from the ROM to RAM.

<5> Blank checking of the specified block (1-Kbyte)

The FSL_BlankCheck function is called to perform a blank check of the specified block (1-Kbyte) (check that the block is ready to be written to).

<6> Erasing the specified block (1-Kbyte)

The FSL_Erase function is called to erase the specified block (1-Kbyte).

<7> Writing 1 word to 64 words (4 bytes to 256 bytes) data to the specified address

The FSL_Write function is called to write 1 word to 64 words (4 bytes to 256 bytes) data to the specified address.
If writing to the specified block cannot be completed at one time, the FSL_Write function is executed multiple times to complete writing all data to the specified block before a transition to the next processing. Data can be written only to blank or erased blocks; data cannot be written again (overwritten) to an area that has been written to.

<8> Verification (internal verification) of the specified block (1-Kbyte)

The FSL_IVerify function is called for verification (internal verification) of the specified block (1-Kbyte).

Note: Internal verification checks if the signal levels of the flash memory cells are correct. It does not compare data.

<9> Changing interrupt reception back to the ROM

If the interrupt destination was changed to the RAM in <4>, the FSL_RestoreInterruptTable function is called to change the interrupt reception destination back to the ROM.

<10> Ending the flash environment

The FSL_Close function is called to end flash self-programming. The FSL_Close function should be executed when all writing processing is completed or when flash self-programming should be terminated.

<11> Status checking

When the status check user mode is used, status checking must be performed until the control of the code flash memory is finished.

Remark 1 word = 4 bytes

CHAPTER 2 PROGRAMMING ENVIRONMENT

This chapter describes the hardware environment and software environment required to rewrite the code flash memory using the flash self-programming library.

2. 1 Hardware Environment

Flash self-programming of the RL78 microcontroller uses the sequencer to control rewriting of the flash memory. During the control of the sequencer, the code flash memory cannot be referred to. Therefore, if the user program needs to be operated during sequencer control such as an interrupt[Note], some segments of the flash self-programming library and the user program must be allocated in the RAM to control erasure and writing to the code flash memory or setting of the security flag. If it is not necessary to operate the user program during sequencer control, the flash self-programming library and user program can be allocated on the ROM for operation.

Figure 2-1 shows the state during a rewrite of the code flash memory. Figure 2-2 and Figure 2-3 show examples of execution of flash functions for rewriting of the code flash memory.

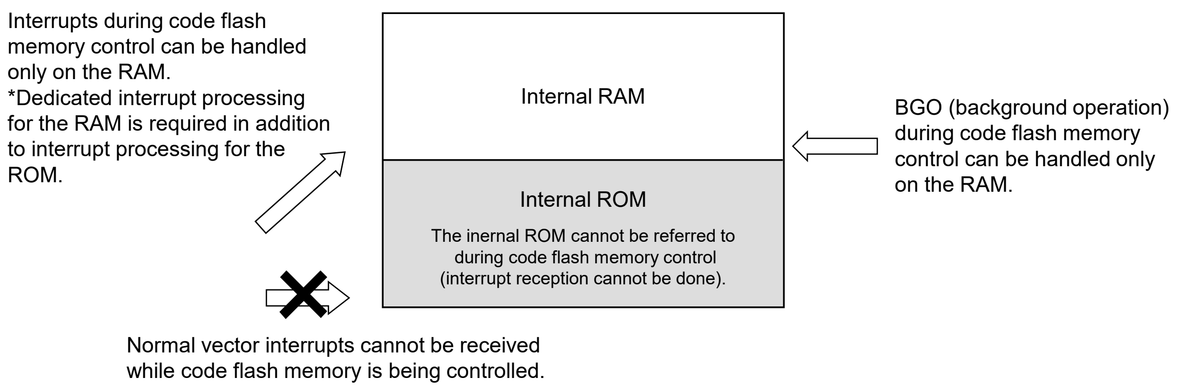

Interrupts during code flash memory control can be handled only on the RAM.
*Dedicated interrupt processing for the RAM is required in addition to interrupt processing for the ROM.
Internal RAM
Internal ROM
The inernal ROM cannot be referred to during code flash memory control (interrupt reception cannot be done).
BGO (background operation) during code flash memory control can be handled only on the RAM.
Normal vector interrupts cannot be received while code flash memory is being controlled.

Figure 2-1. State during Rewrite of Code Flash Memory

Note. Some RL78 microcontrollers do not support an interrupt during the execution of flash self-programming. Refer to the user’s manual of the target RL78 microcontroller to see whether the RL78 microcontroller to be used supports an interrupt during the execution of the flash self-programming.

- After a request of the desired processing execution is made to the sequencer of the RL78 microcontroller, the control is immediately returned to the user program. Because the sequencer controls the code flash memory, the user program can operate during the rewrite of the code flash memory. This is called BGO (background operation). To use this mode, select the status check user mode when initializing the Flash Self-Programming Library Type 01.

  However, the code flash memory cannot be referred to while the sequencer is controlling the code flash memory. Therefore, the user program that operates during code flash memory operation, the branch destination of the interrupt, the interrupt processing, and some segments of the flash self-programming library need to be allocated on the RAM.

  For the result of the control of the code flash memory, the status check function (FSL_StatusCheck function) must be called from the user program to check the control state of the code flash memory.

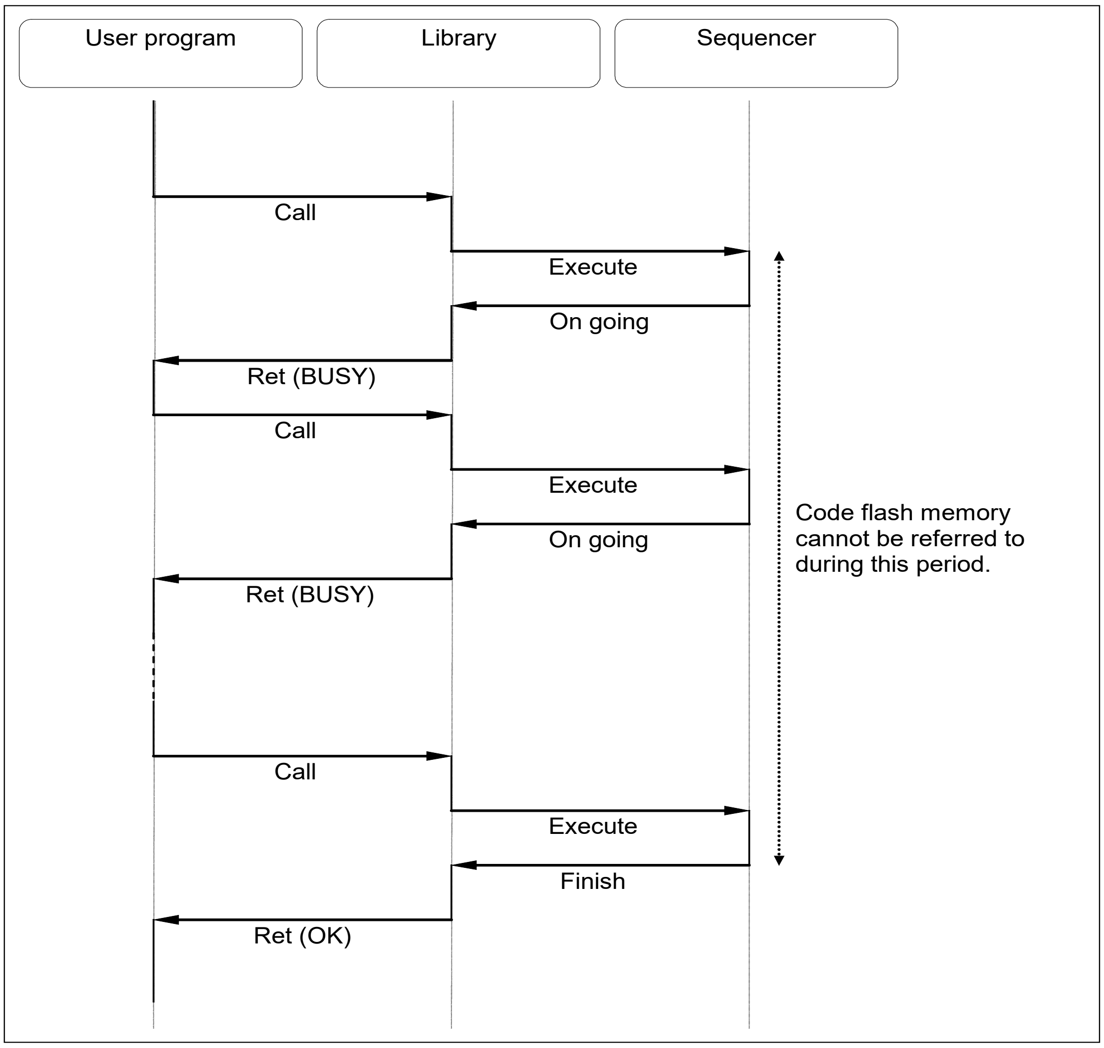


Figure 2-2. Example 1 Rewrite Control of Flash (When User Program Operates during Rewrite)

- After a request of the desired processing execution is made to the sequencer of the RL78 microcontroller, the control is not returned to the user program until the corresponding processing of the sequencer is completed. Because the control returns to the user program after the control of the code flash memory is completed, the user program and flash self-programming can be allocated on the ROM. To use this mode, select the status check internal mode when initializing the Flash Self-Programming Library Type 01.

  However, if it is required to receive an interrupt during the control of the code flash memory, the branch destination of the interrupt and interrupt processing must be allocated on the RAM. If they are allocated on the ROM, part of the flash functions cannot be used. For details of the flash functions, refer to **CHAPTER 6 FLASH FUNCTIONS**.

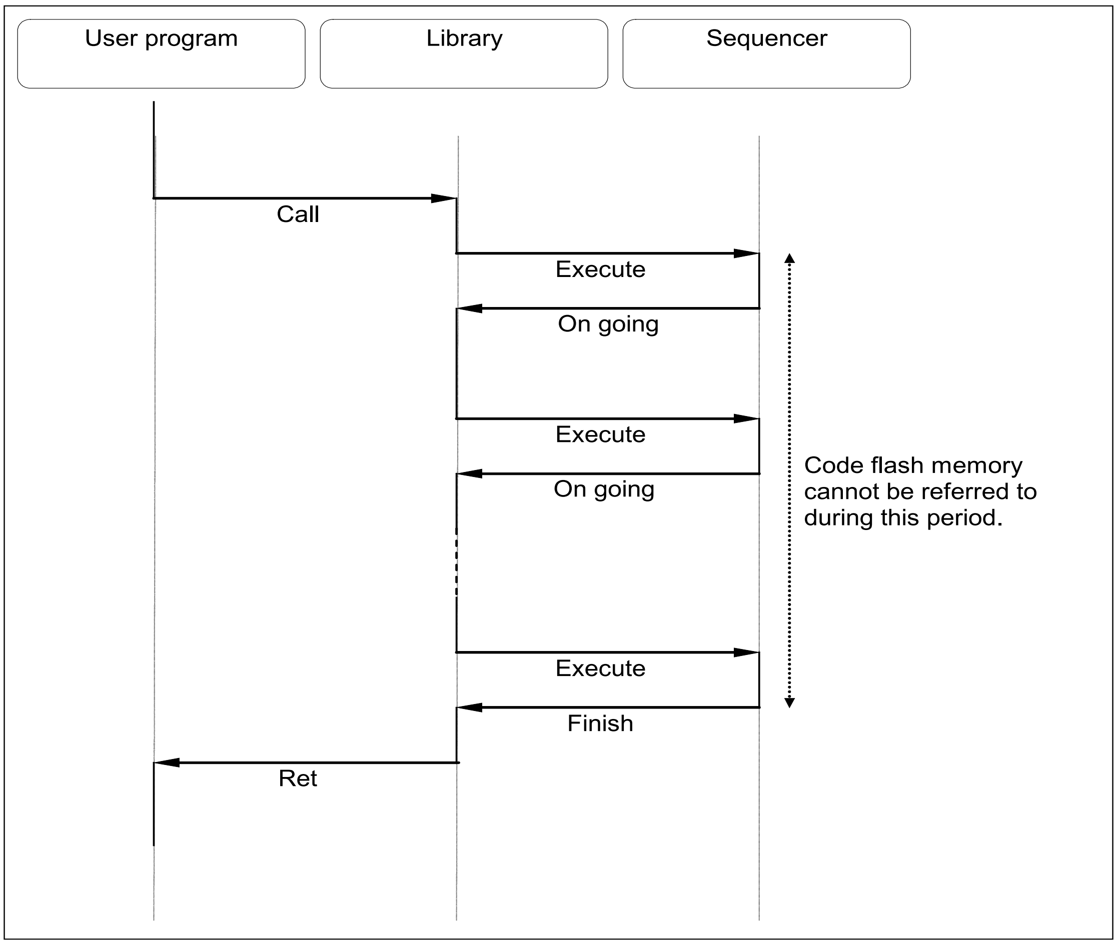


Figure 2-3. Example 2 Rewrite Control of Flash (When User Program Does Not Operate during Rewrite)

2. 1. 1 Initialization

When rewriting the code flash memory by using the flash self-programming library, make the following settings.

(1) Starting high-speed on-chip oscillator

During use of the flash self-programming library, keep the high-speed on-chip oscillator running. When the high-speed on-chip oscillator is stopped, start it before using the flash self-programming library.

(2) Setting CPU operating frequency[Note 1]

In order to calculate the timing in the flash self-programming library, set the CPU operating frequency at initialization. See the description of the FSL_Init() function for the method for setting the CPU operating frequency.

(3) Setting flash memory programming mode[Note 2]

In order to set the flash memory programming mode for erasing or writing, either of the flash memory programming modes shown below should be specified when initializing the flash self-programming library. See the description of the FSL_Init() function for the settings of the flash memory programming modes.

- Full speed mode
- Wide voltage mode

Notes 1. The CPU operating frequency is used as a parameter for the calculation of internal timing in the flash self-programming library. This setting does not affect the CPU operating frequency. This is not the operating frequency for the high-speed on-chip oscillator.

2. For details of the flash memory programming mode, see the target RL78 microcontroller user's manual.

2. 1. 2 Blocks

The flash memory of the RL78 microcontroller is divided into 1 Kbyte blocks. In flash self-programming, erasure processing, blank check processing, and verification (internal verification) processing are performed for the code flash memory in the units of the blocks. To call these flash self-programming library functions, specify a block number.

The boot cluster[Note] is the area provided to prevent the user program from being unable to boot up due to destruction of the vector table data or program basic functions caused by an instantaneous power interruption or resetting during a rewrite while the area including the vector area is being rewritten. For details, refer to **CHAPTER 5 BOOT SWAP FUNCTION**[Note].

Figure 2-4 shows block numbers and boot clusters.

Note To use this function, the RL78 microcontroller supporting the boot swap function is required.
To find if your RL78 microcontroller supports the boot swap function, refer to the user's manual of the target RL78 microcontroller.

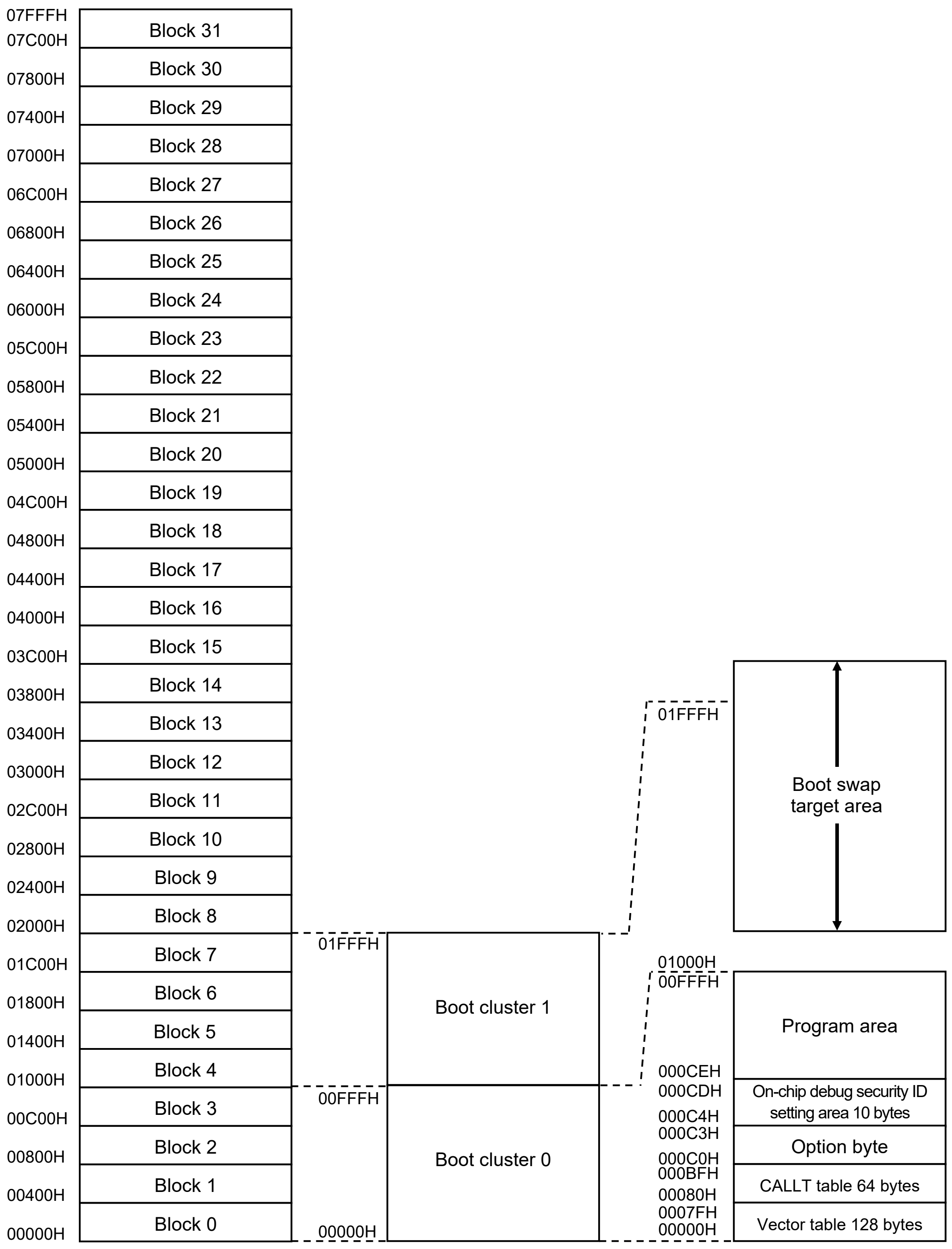


Figure 2-4. Example of Block Numbers and Boot Clusters (RL78/G13: When Code Flash Memory is 32 Kbytes)

2. 1. 3 Processing time of flash self-programming

This section describes the time required to process the Flash Self-Programming Library Type 01 functions. The number of clock cycles required to execute flash functions differs depending on whether the flash functions are allocated to the internal ROM area (flash memory) or they are allocated to the internal RAM area. When the functions are executed in the RAM, the processing time may increase to a maximum of double the time needed when they are executed in the ROM.

This section shows the processing time when the FSL_RCD segment is executed in the RAM and the other segments are executed in the ROM. For each segment of flash functions, see Table 6-2 Segment List of Flash Functions.

(1) Flash self-programming library function processing time in status check user mode

The flash self-programming library function processing time is the time required from when a user-created program calls a flash function until the processing ends and control returns to the user-created program. The flash function processing time differs depending on the status check mode.

This section shows the flash function processing time in the status check user mode.

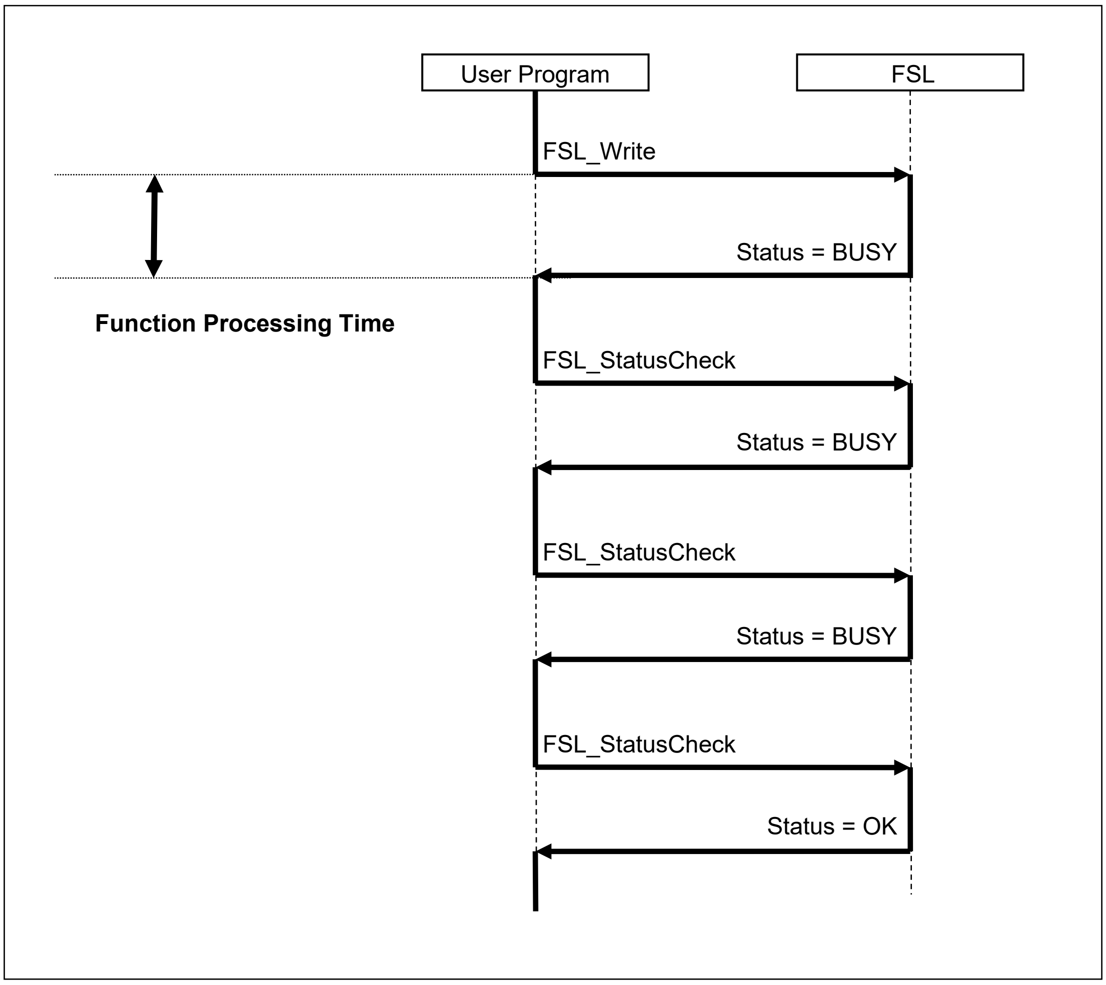


Figure 2-5. Overview of Flash Self-Programming Library Function Processing Time in Status Check User Mode

Table 2-1. Flash Function Processing Time in Status Check User Mode (Full Speed Mode)

FSL_Functions		**Max. (μs)**
FSL_Init		5021 / f_{CLK}
FSL_Open		10 / f_{CLK}
FSL_Close		10 / f_{CLK}
FSL_PrepareFunctions		2484 / f_{CLK}
FSL_PrepareExtFunctions		1259 / f_{CLK}
FSL_ChangeInterruptTable		253 / f_{CLK}
FSL_RestoreInterruptTable		229 / f_{CLK}
FSL_BlankCheck		2069 / f_{CLK} + 30
FSL_Erase		2192 / f_{CLK}+ 30
FSL_IVerify		2097 / f_{CLK} + 30
FSL_Write		2451 / f_{CLK} + 30
FSL_GetSecurityFlags		331 / f_{CLK}
FSL_GetBootFlag		328 / f_{CLK}
FSL_GetSwapState		206 / f_{CLK}
FSL_GetBlockEndAddr		368 / f_{CLK}
FSL_GetFlashShieldWindow		307 / f_{CLK}
FSL_SwapBootCluster		419 / f_{CLK} + 32
FSL_SwapActiveBootCluster		2316 / f_{CLK} + 30
FSL_InvertBootFlag		2341 / f_{CLK} + 30
FSL_SetBlockEraseProtectFlag		2347 / f_{CLK} + 30
FSL_SetWriteProtectFlag		2346 / f_{CLK}+ 30
FSL_SetBootClusterProtectFlag		2347 / f_{CLK} + 30
FSL_SetFlashShieldWindow		2141 / f_{CLK} + 30
FSL_StatusCheck		1135 / f_{CLK} + 50
FSL_StandBy	Erase	935 / f_{CLK} + 31
	Other than Erase (when FSL_SetXXX are supported)	140367 / f_{CLK} + 513844
	Other than Erase (when FSL_SetXXX are not supported)	76101 / f_{CLK} + 35952
FSL_WakeUp	Suspended Erase	2144 / f_{CLK} + 30
	Other than Erase	148 / f_{CLK}
FSL_ForceReset		—
FSL_GetVersionString		10 / f_{CLK}

Remark f_{CLK}: CPU operating frequency (For example, when using a 20 MHz clock, f_{CLK} is 20.)

Table 2-2. Flash Function Processing Time in Status Check User Mode (Wide Voltage Mode)

FSL_Functions		Max. (μs)
FSL_Init		5021 / f_{CLK}
FSL_Open		10 / f_{CLK}
FSL_Close		10 / f_{CLK}
FSL_PrepareFunctions		2484 / f_{CLK}
FSL_PrepareExtFunctions		1259 / f_{CLK}
FSL_ChangeInterruptTable		253 / f_{CLK}
FSL_RestoreInterruptTable		229 / f_{CLK}
FSL_BlankCheck		2068 / f_{CLK} + 30
FSL_Erase		2192 / f_{CLK} + 30
FSL_IVerify		2097 / f_{CLK} + 30
FSL_Write		2451 / f_{CLK} + 30
FSL_GetSecurityFlags		331 / f_{CLK}
FSL_GetBootFlag		328 / f_{CLK}
FSL_GetSwapState		206 / f_{CLK}
FSL_GetBlockEndAddr		368 / f_{CLK}
FSL_GetFlashShieldWindow		307 / f_{CLK}
FSL_SwapBootCluster		419 / f_{CLK} + 32
FSL_SwapActiveBootCluster		2316 / f_{CLK} + 30
FSL_InvertBootFlag		2341 / f_{CLK} + 30
FSL_SetBlockEraseProtectFlag		2347 / f_{CLK} + 30
FSL_SetWriteProtectFlag		2346 / f_{CLK} + 30
FSL_SetBootClusterProtectFlag		2347 / f_{CLK} + 30
FSL_SetFlashShieldWindow		2141 / f_{CLK} + 30
FSL_StatusCheck		1135 / f_{CLK} + 50
FSL_StandBy	Erase	935 / f_{CLK} + 44
	Other than Erase (when FSL_SetXXX are supported)	123274 / f_{CLK} + 538046
	Other than Erase (when FSL_SetXXX are not supported)	73221 / f_{CLK} + 69488
FSL_WakeUp	Suspended Erase	2144 / f_{CLK} + 30
	Other than Erase	148 / f_{CLK}
FSL_ForceReset		—
FSL_GetVersionString		10 / f_{CLK}

Remark f_{CLK}: CPU operating frequency (For example, when using a 20 MHz clock, f_{CLK} is 20.)

(2) Flash self-programming library function processing time in status check internal mode

This section shows the flash function processing time in the status check internal mode.

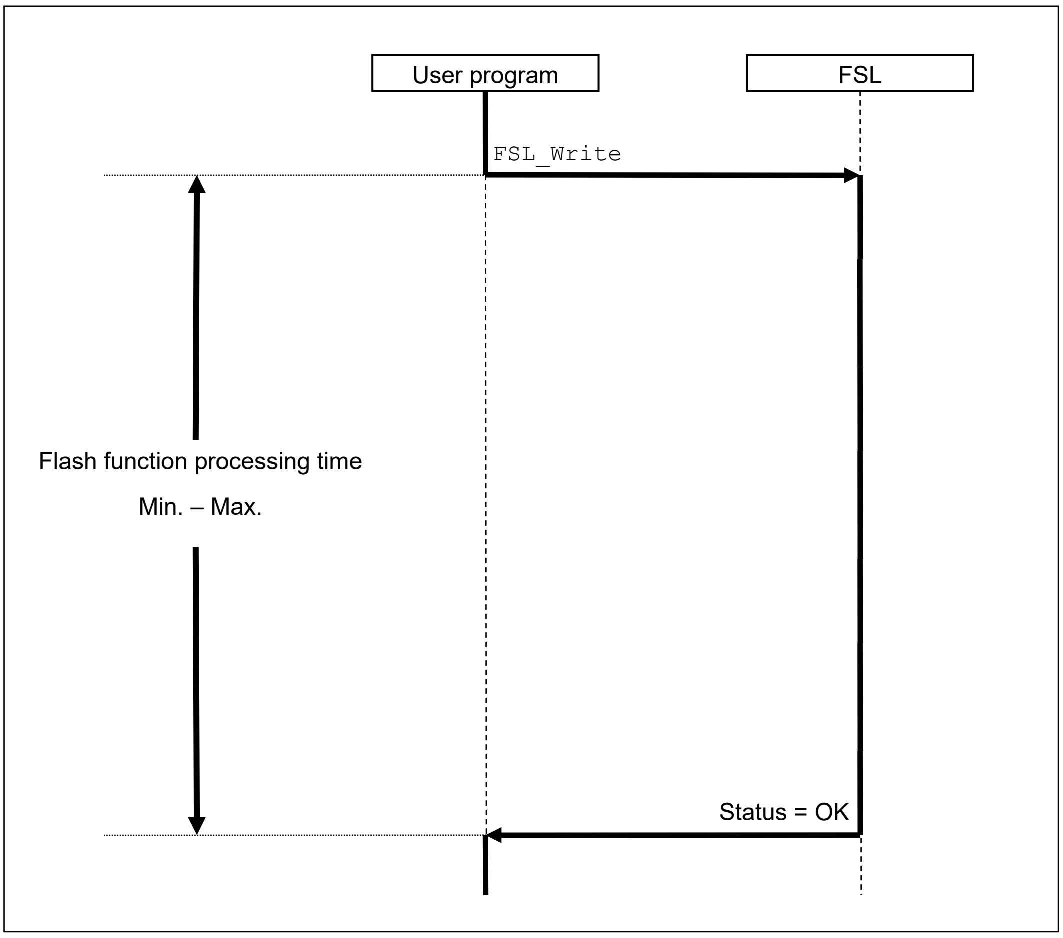


Figure 2-6. Overview of Flash Function Processing Time in Status Check Internal Mode

Table 2-3. Flash Function Processing Time in Status Check Internal Mode (Full Speed Mode)

FSL_Functions	Min. (μs)	Max. (μs)
FSL_Init	—	5021 / f_{CLK}
FSL_Open	—	10 / f_{CLK}
FSL_Close	—	10 / f_{CLK}
FSL_PrepareFunctions	—	2484 / f_{CLK}
FSL_PrepareExtFunctions	—	1259 / f_{CLK}
FSL_ChangeInterruptTable	—	253 / f_{CLK}
FSL_RestoreInterruptTable	—	229 / f_{CLK}
FSL_BlankCheck	3302 / f_{CLK} + 84	4833 / f_{CLK} + 164
FSL_Erase	4877 / f_{CLK} + 163	73339 / f_{CLK} + 255366
FSL_IVerify	—	10474 / f_{CLK} + 1107
FSL_Write	3121 / f_{CLK} + 66 + (595 / f_{CLK} + 60) × W	3121 / f_{CLK} + 66 + (1153 / f_{CLK} + 561) × W
FSL_GetSecurityFlags	—	331 / f_{CLK}
FSL_GetBootFlag	—	328 / f_{CLK}
FSL_GetSwapState	—	206 / f_{CLK}
FSL_GetBlockEndAddr	—	368 / f_{CLK}
FSL_GetFlashShieldWindow	—	307 / f_{CLK}
FSL_SwapBootCluster	—	419 / f_{CLK} + 32
FSL_SwapActiveBootCluster	1938 / f_{CLK} + 50	141314 / f_{CLK} + 513862
FSL_InvertBootFlag	1565 / f_{CLK} + 18	140940 / f_{CLK} + 513830
FSL_SetBlockEraseProtectFlag	1571 / f_{CLK} + 18	140946 / f_{CLK} + 513830
FSL_SetWriteProtectFlag	1569 / f_{CLK} + 18	140945 / f_{CLK} + 513830
FSL_SetBootClusterProtectFlag	1571 / f_{CLK} + 18	140946 / f_{CLK} + 513830
FSL_SetFlashShieldWindow	1356 / f_{CLK} + 18	140739 / f_{CLK} + 513830
FSL_StatusCheck	—	—
FSL_StandBy	—	—
FSL_WakeUp	—	—
FSL_ForceReset	—	—
FSL_GetVersionString	—	10 / f_{CLK}

Remarks 1. f_{CLK}: CPU operating frequency (For example, when using a 20 MHz clock, f_{CLK} is 20.)

2. W: The number of words to be written (1 word = 4 bytes)
(For example, when specifying 2 words = 8 bytes, W is 2.)

Table 2-4. Flash Function Processing Time in Status Check Internal Mode (Wide Voltage Mode)

FSL_Functions	Min. (μs)	Max. (μs)
FSL_Init	—	5021 / f_{CLK}
FSL_Open	—	10 / f_{CLK}
FSL_Close	—	10 / f_{CLK}
FSL_PrepareFunctions	—	2484 / f_{CLK}
FSL_PrepareExtFunctions	—	1259 / f_{CLK}
FSL_ChangeInterruptTable	—	253 / f_{CLK}
FSL_RestoreInterruptTable	—	229 / f_{CLK}
FSL_BlankCheck	3298 / f_{CLK} + 124	4574 / f_{CLK} + 401
FSL_Erase	4675 / f_{CLK} + 401	64468 / f_{CLK} + 266193
FSL_IVerify	—	7659 / f_{CLK} + 7534
FSL_Write	3121 / f_{CLK} + 66 + (591 / f_{CLK} + 112) × W	3121 / f_{CLK}+ 66 + (1108 / f_{CLK} + 1085) × W
FSL_GetSecurityFlags	—	331 / f_{CLK}
FSL_GetBootFlag	—	328 / f_{CLK}
FSL_GetSwapState	—	206 / f_{CLK}
FSL_GetBlockEndAddr	—	368 / f_{CLK}
FSL_GetFlashShieldWindow	—	307 / f_{CLK}
FSL_SwapBootCluster	—	419 / f_{CLK} + 32
FSL_SwapActiveBootCluster	1938 / f_{CLK} + 50	124221 / f_{CLK} + 538064
FSL_InvertBootFlag	1565 / f_{CLK} + 18	123847 / f_{CLK} + 538032
FSL_SetBlockEraseProtectFlag	1571 / f_{CLK} + 18	123853 / f_{CLK} + 538032
FSL_SetWriteProtectFlag	1569 / f_{CLK} + 18	123852 / f_{CLK} + 538032
FSL_SetBootClusterProtectFlag	1571 / f_{CLK} + 18	123853 / f_{CLK} + 538032
FSL_SetFlashShieldWindow	1356 / f_{CLK} + 18	123646 / f_{CLK} + 538032
FSL_StatusCheck	—	—
FSL_StandBy	—	—
FSL_WakeUp	—	—
FSL_ForceReset	—	—
FSL_GetVersionString	—	10 / f_{CLK}

Remarks 1. f_{CLK}: CPU operating frequency (For example, when using a 20 MHz clock, f_{CLK} is 20.)

2. W: The number of words to be written (1 word = 4 bytes)
(For example, when specifying 2 words = 8 bytes, W is 2.)

(3) Recommended interval of FSL_StatusCheck (status check)

The FSL_StatusCheck function is used to check the status in the status check user mode. However, correct results cannot be obtained if the FSL_StatusCheck function is executed before control by the sequencer finishes. Therefore, spacing each process executed by each flash function by a specific time is useful to enhance the efficiency of status checking. In addition, because a write process using the FSL_Write function must be triggered by status check processing every 4-byte, the status must be checked each time 4-byte is written.

When writing 12 bytes in the status check user mode, the sequencer writes data in a 4-byte unit. Therefore, when 4-byte is written, the FSL_StatusCheck function must trigger the next write. If the FSL_StatusCheck function is not executed while there are still bytes to be written, the next write does not start, and thus the write process does not end.

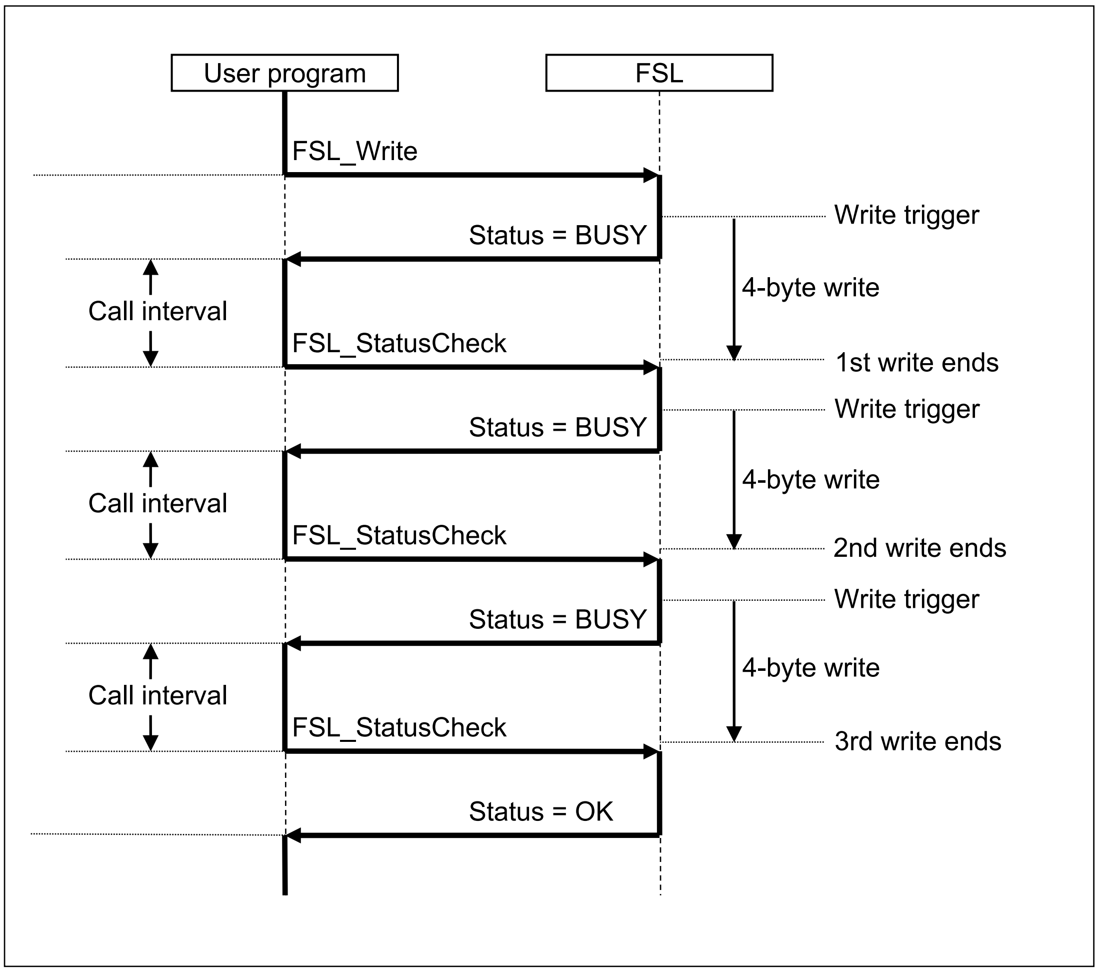


Figure 2-7. Overview of Interval for Checking Status When Using FSL_Write (When Writing 12 bytes)

When a process is executed by a function other than FSL_Write in the status check user mode, the sequencer is in the busy state until all processes end. A trigger by the FSL_StatusCheck function is therefore not required.

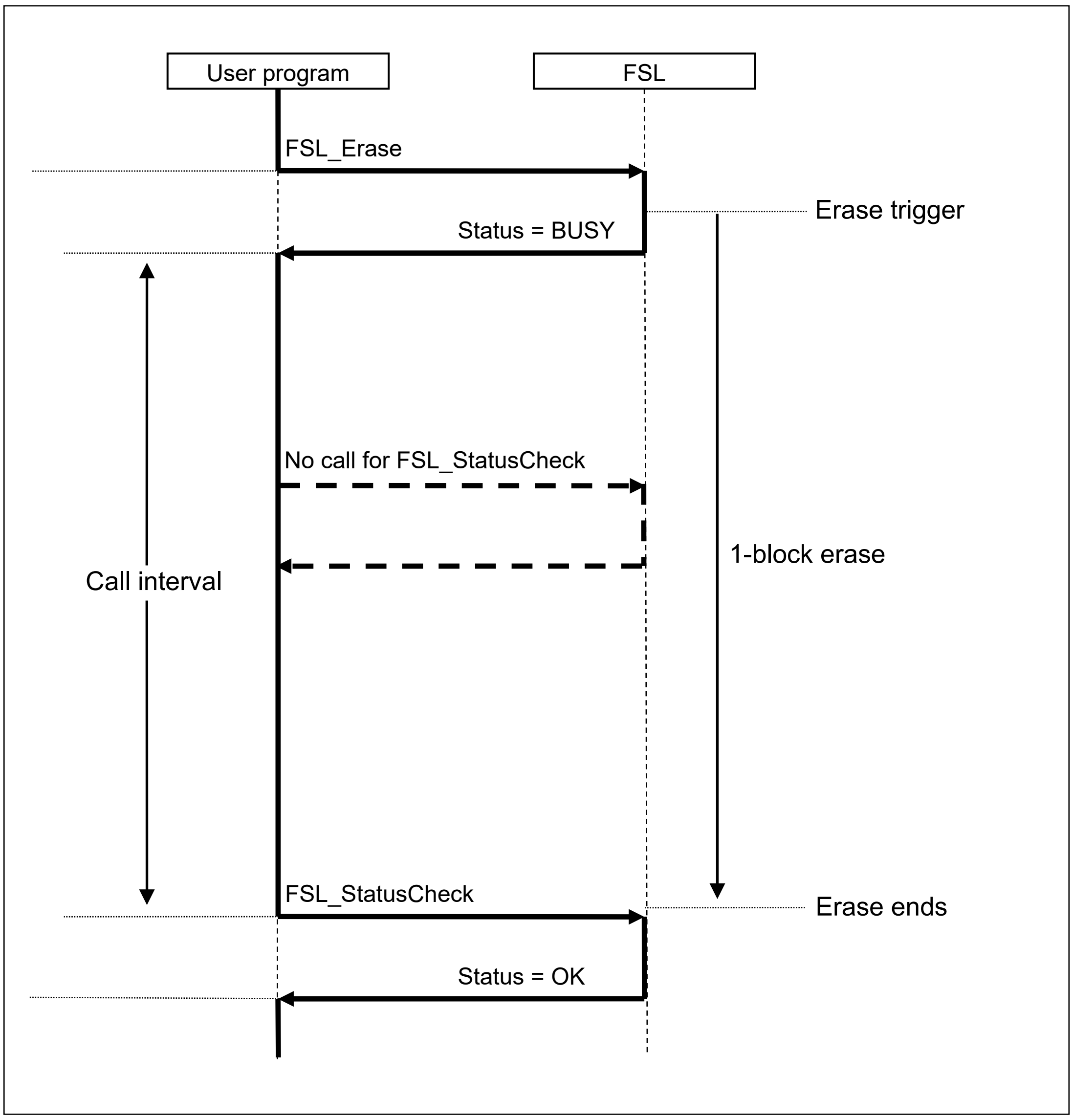


Figure 2-8. Overview of Interval for Checking Status When Using a Function Other Than FSL_Write (When Erasing Flash Memory)

Table 2-5. Recommended Interval of Status Check in Status Check User Mode (Full Speed Mode)

FSL_Functions		Call Interval (μs)
FSL_Init		—
FSL_Open		—
FSL_Close		—
FSL_PrepareFunctions		—
FSL_PrepareExtFunctions		—
FSL_ChangeInterruptTable		—
FSL_RestoreInterruptTable		—
FSL_BlankCheck		1569 / f_{CLK} + 98
FSL_Erase	When block is blanked	1490 / f_{CLK} + 97
	When block is not blanked	3092 / f_{CLK} + 6471
FSL_IVerify		7181 / f_{CLK} + 1041
FSL_Write[Note]		72 / f_{CLK} + 60
FSL_GetSecurityFlags		—
FSL_GetBootFlag		—
FSL_GetSwapState		—
FSL_GetBlockEndAddr		—
FSL_GetFlashShieldWindow		—
FSL_SwapBootCluster		—
FSL_SwapActiveBootCluster		6431 / f_{CLK} + 7053
FSL_InvertBootFlag		
FSL_SetBlockEraseProtectFlag		
FSL_SetWriteProtectFlag		
FSL_SetBootClusterProtectFlag		
FSL_SetFlashShieldWindow		
FSL_StatusCheck		—
FSL_StandBy		—
FSL_WakeUp	When block is blanked	1490 / f_{CLK} + 97
	When block is not blanked	3092 / f_{CLK} + 6471
FSL_ForceReset		—
FSL_GetVersionString		—

Remark f_{CLK}: CPU operating frequency (For example, when using a 20 MHz clock, f_{CLK} is 20.)

Note The value shown for the FSL_Write function indicates the recommended interval per 4-byte.

Table 2-6. Recommended Interval of Status Check in Status Check User Mode (Wide Voltage Mode)

FSL_Functions		Call Interval (μs)
FSL_Init		—
FSL_Open		—
FSL_Close		—
FSL_PrepareFunctions		—
FSL_PrepareExtFunctions		—
FSL_ChangeInterruptTable		—
FSL_RestoreInterruptTable		—
FSL_BlankCheck		1310 / f_{CLK} + 335
FSL_Erase	When block is blanked	1289 / f_{CLK} + 335
	When block is not blanked	2689 / f_{CLK} + 6959
FSL_IVerify		4366 / f_{CLK} + 7468
FSL_Write[Note]		67 / f_{CLK} + 112
FSL_GetSecurityFlags		—
FSL_GetBootFlag		—
FSL_GetSwapState		—
FSL_GetBlockEndAddr		—
FSL_GetFlashShieldWindow		—
FSL_SwapBootCluster		—
FSL_SwapActiveBootCluster		5728 / f_{CLK} + 8445
FSL_InvertBootFlag		
FSL_SetBlockEraseProtectFlag		
FSL_SetWriteProtectFlag		
FSL_SetBootClusterProtectFlag		
FSL_SetFlashShieldWindow		
FSL_StatusCheck		—
FSL_StandBy		—
FSL_WakeUp	When block is blanked	1289 / f_{CLK} + 335
	When block is not blanked	2689 / f_{CLK} + 6959
FSL_ForceReset		—
FSL_GetVersionString		—

Remark f_{CLK}: CPU operating frequency (For example, when using a 20 MHz clock, f_{CLK} is 20.)

Note The value shown for the FSL_Write function indicates the recommended interval per 4-byte.

2. 2 Software Environment

Because the flash self-programming library program needs to be allocated to a user-created program area, the size of the program code will be consumed in the program area.

To run the flash self-programming library, the CPU, stack, and data buffer are used.

<R> Flash Self-Programming Library Type 01 has three versions: one is for the CA78K0R compiler (V2.20) and the other is for the CC-RL compiler (V2.21) and the LLVM compiler (V2.21). In some tables below, the library for the CA78K0R compiler (V2.20) is abbreviated to CA78 and that for the CC-RL compiler (V2.21) is abbreviated to CCRL and that for the LLVM compiler (V2.21) is abbreviated to LLVM.

Tables 2-7 lists the software resources required[Note1, 2], and Figures 2-9 and 2-10 show examples of arrangement in RAM.

<R> **Table 2-7. Software Resources Used by Flash Self-Programming Library Type 01**

Item	Size (byte)		Restrictions on Allocation and Usage[Notes1,2]
	CA78	CCRL LLVM	
Self-RAM[Note3]	0 to 1024[Note3]	0 to 1024[Note3]	The self-RAM area used by RL78 Family Flash Self-Programming Library Type 01 differs depending on the device. For details, refer to "RL78 Family Self RAM list of Flash Self Programming Library(R20UT2944)".
Stack (see Table 2-8)	46 max.	50 max.	Can be allocated to a RAM area other than the self-RAM and the area from FFE20H to FFEFFH
Data buffer [Note4] (see Table 2-9)	1 to 256	1 to 256	
Arguments of library functions	0 to 8	0 to 8	
Library size (see Tables 2-10 and 2-11)	ROM: 1,252 max.	ROM: 1,294 max.	Can be allocated to a program area other than the self-RAM and the area from FFE20H to FFEFFH
	RAM: 0 to 447	RAM: 0 to 468	Can be allocated to a program area other than the self-RAM, the area from FFE20H to FFEFFH, and the internal ROM.

Notes: 1. For devices not shown in the RL78 Family Self RAM list of Flash Self Programming Library (R20UT2944), contact your Renesas sales agency.
2. The R5F10266 product does not support the self-programming function.
3. An area used as the working area by the flash self-programming library is called self-RAM in this manual and the Release Note. The self-RAM requires no user settings because it is an area that is not mapped and automatically used at execution of the flash self-programming library (previous data is discarded). When the flash self-programming library is not used, the self-RAM can be used as a normal RAM space.
4. The data buffer is used as the working area for flash self-programming library internal processing or the area where the data to be set is allocated by the FSL_Write function. The required size depends on the function to be used.

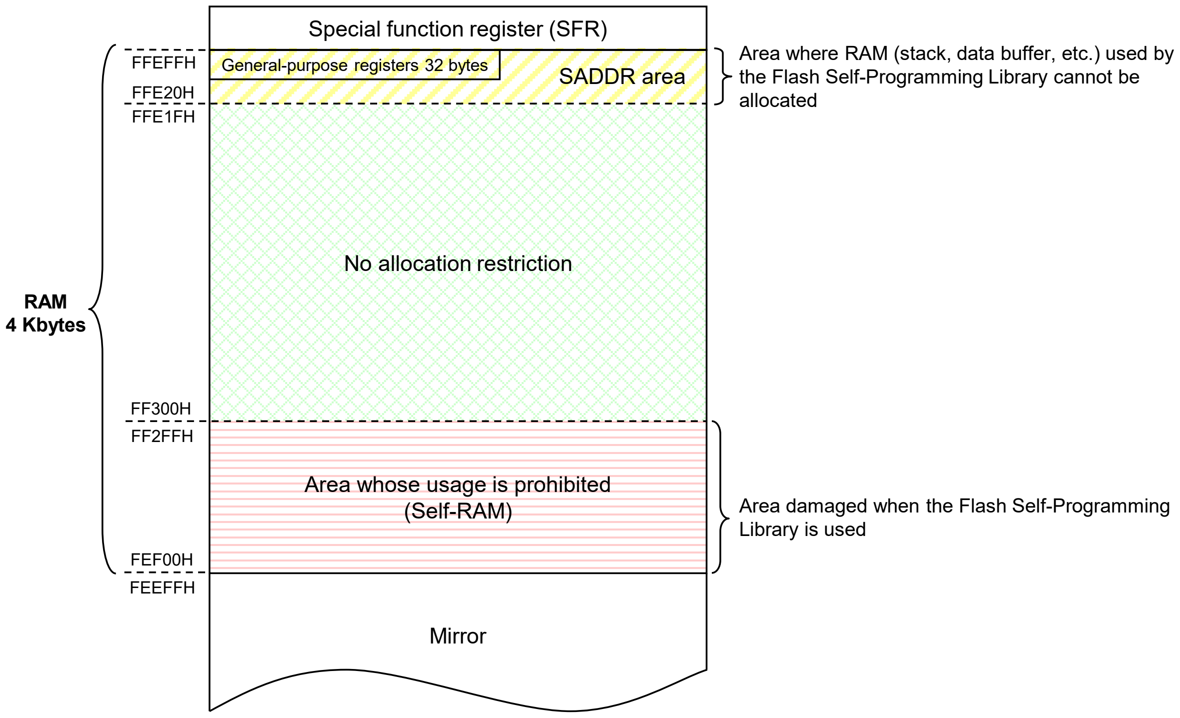

Special function register (SFR)
FFEFFH
General-purpose registers 32 bytes
SADDR area
FFE20H
FFE1FH
Area where RAM (stack, data buffer, etc.) used by the Flash Self-Programming Library cannot be allocated
RAM
4 Kbytes
No allocation restriction
FF300H
FF2FFH
Area whose usage is prohibited (Self-RAM)
Area damaged when the Flash Self-Programming Library is used
FEF00H
FEEFFH
Mirror

Figure 2-9 Example 1 of Arrangement in RAM Including Self-RAM (RL78/G13: product with 4 Kbytes RAM and 64 Kbytes ROM)

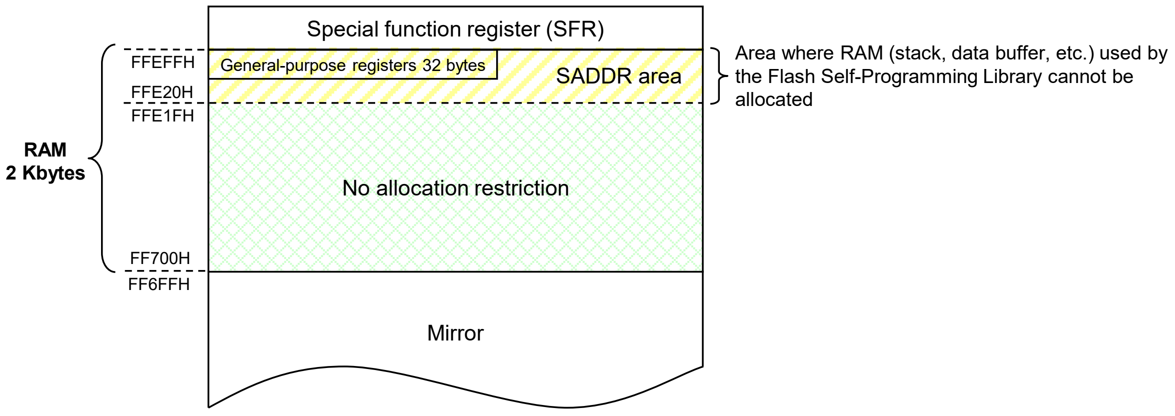

Special function register (SFR)
FFEFFH
General-purpose registers 32 bytes
SADDR area
FFE20H
FFE1FH
Area where RAM (stack, data buffer, etc.) used by the Flash Self-Programming Library cannot be allocated
RAM
2 Kbytes
No allocation restriction
FF700H
FF6FFH
Mirror

Figure 2-10 Example 2 of Arrangement in RAM without Self-RAM (RL78/G13: product with 2 Kbytes RAM and 32 Kbytes ROM)

<R>

Table 2-8. Stack Size Used by Flash Functions

Function Name	byte		Function Name	byte	
	CA78	CCRL LLVM		CA78	CCRL LLVM
`FSL_Init`	40	44	`FSL_GetBlockEndAddr`	36	40
`FSL_Open`	0	2	`FSL_GetFlashShieldWindow`	46	50
`FSL_Close`	0	2	`FSL_SwapBootCluster`	38	40
`FSL_PrepareFunctions`	10	12	`FSL_SwapActiveBootCluster`	42	46
`FSL_PrepareExtFunctions`	10	12	`FSL_InvertBootFlag`	42	46
`FSL_ChangeInterruptTable`	30	32	`FSL_SetBlockEraseProtectFlag`	42	46
`FSL_RestoreInterruptTable`	30	32	`FSL_SetWriteProtectFlag`	42	46
`FSL_BlankCheck`	42	46	`FSL_SetBootClusterProtectFlag`	42	46
`FSL_Erase`	42	46	`FSL_SetFlashShieldWindow`	42	46
`FSL_IVerify`	42	46	`FSL_StatusCheck`	30	34
`FSL_Write`	42	46	`FSL_StandBy`	30	34
`FSL_GetSecurityFlags`	46	50	`FSL_WakeUp`	42	46
`FSL_GetBootFlag`	46	50	`FSL_ForceReset`	0	2
`FSL_GetSwapState`	36	40	`FSL_GetVersionString`	0	2

Note Each size does not include the stack size used by the caller to call the FSL function.

Table 2-9. Data Buffer Size Used by Flash Functions

Function Name	byte	Function Name	byte
`FSL_Init`	0	`FSL_GetBlockEndAddr`	4
`FSL_Open`	0	`FSL_GetFlashShieldWindow`	4
`FSL_Close`	0	`FSL_SwapBootCluster`	0
`FSL_PrepareFunctions`	0	`FSL_SwapActiveBootCluster`	0
`FSL_PrepareExtFunctions`	0	`FSL_InvertBootFlag`	0
`FSL_ChangeInterruptTable`	0	`FSL_SetBlockEraseProtectFlag`	0
`FSL_RestoreInterruptTable`	0	`FSL_SetWriteProtectFlag`	0
`FSL_BlankCheck`	0	`FSL_SetBootClusterProtectFlag`	0
`FSL_Erase`	0	`FSL_SetFlashShieldWindow`	4
`FSL_IVerify`	0	`FSL_StatusCheck`	0
`FSL_Write`^Note	4 to 256	`FSL_StandBy`	0
`FSL_GetSecurityFlags`	2	`FSL_WakeUp`	0
`FSL_GetBootFlag`	1	`FSL_ForceReset`	0
`FSL_GetSwapState`	1	`FSL_GetVersionString`	0

Note The FSL_Write function requires an amount of memory equal to the data to be written (in words).
For example, when writing 2 words (1 word = 4 bytes), the required amount of memory is: 2 × 4 = 8 bytes

Flash Self-Programming Library Code Size

(1) Code size when allocating all functions to ROM

Table 2-10 shows the code size required when all flash self-programming library functions are allocated to ROM. Allocating the code to RAM is not required, but usage restrictions will prevent some functions being used if all functions are allocated to ROM. For details, see **6.2 Segments of Flash Functions**.

<R> **Table 2-10. Code Size When Allocating All Functions to ROM**

Conditions	CA78K0R compiler (V2.20)		CC-RL compiler (V2.21) LLVM compiler (V2.21)	
	RAM Size (byte)	ROM Size (byte)	RAM Size (byte)	ROM Size (byte)
Code size when all functions are registered * Some functions cannot be used.	0	1,252	0	1294
Code size when all the following functions are used: • FSL_Init • FSL_Open • FSL_Close • FSL_PrepareFunctions • FSL_BlankCheck • FSL_Erase • FSL_IVerify • FSL_Write • FSL_StatusCheck	0	500	0	502

(2) Code size when allocating some functions to RAM (when using BGO)

Table 2-11 shows the code size required when using the background operation (BGO) feature during flash self-programming. When using the BGO feature, the FSL_RCD segment must be allocated to RAM. To copy the FSL_RCD segment to RAM, the program must be ROMized. Therefore, an additional ROM capacity equivalent to the FSL_RCD segment size is required.

<R> **Table 2-11. Code Size When Allocating Some Functions to RAM**

Conditions	CA78K0R compiler (V2.20)		CC-RL compiler (V2.21) LLVM compiler (V2.21)	
	RAM Size (byte)	ROM Size (byte)	RAM Size (byte)	ROM Size (byte)
Code size when all functions are registered	447 (FSL_RCD)	805 + size of program that must be ROMized (447)	468 (FSL_RCD)	826 + size of program that must be ROMized (468)
Code size when all the following functions are used: • FSL_Init • FSL_Open • FSL_Close • FSL_PrepareFunctions • FSL_BlankCheck • FSL_Erase • FSL_IVerify • FSL_Write • FSL_StatusCheck	66 (FSL_RCD)	434 + size of program that must be ROMized (66)	88 (FSL_RCD)	502 + size of program that must be ROMized (88)

Remark The above table only describes the code size of the flash self-programming library. When using BGO, the user-created program must be allocated to RAM, and therefore a RAM capacity equivalent to the user-created program is also required. Moreover, a RAM capacity equivalent to the program ROMized and copied to RAM is required. For details about ROMization, see user's manual of the development tools to be used.

2. 2. 1 Self-RAM

The flash self-programming library may use a RAM area of 1 Kbyte as the working area. This area is called the "self-RAM." The data used in the self-RAM is defined within the library, so no user definition is required.

<R> When a flash self-programming library function is called, the data in the self-RAM area is rewritten.

The self-RAM area used for flash self-programming varies depending on the microcontroller, and the user RAM may be used in some devices. In such a device, the user needs to allocate the self-RAM area to the user RAM; be sure to allocate the self-RAM area at linkage. (In the CA78K0R compiler, the self-RAM area can be specified in the link directive file. In the CC-RL compiler, leave a space unallocated to any section so that the area can automatically be used as the self-RAM area. In the LLVM compiler, the self-RAM area can be specified in the linker script file.)

For the settings in the link directive file, refer to the section "Defining the Internal RAM Area" in the Release Note.

2. 2. 2 Register bank

The flash self-programming library uses the general registers, ES/CS registers, SP, and PSW of the register bank selected by the user.

2. 2. 3 Stack and data buffer

The flash self-programming library uses the sequencer to write to the code flash memory, but it uses the CPU for pre-setting and control. Therefore, to use the flash self-programming library, the stack specified by the user program is also required.

<R> Remark To allocate the stack and data buffer to the user-specified addresses, use the link directive in the CA78K0R compiler, make section allocation settings through a linker option in the CC-RL compiler or use the linker script file in the LLVM compiler.

- Stack
  In addition to the stack used by the user program, the stack space required for flash functions must be reserved in advance, and they must be allocated so that the RAM used by the user will not be destroyed in stack processing during flash self-programming operation. The available range for stack specification is the internal RAM excluding the self-RAM and addresses FFE20H-FFEFFH.
- Data buffer
  The data buffer is used as the working area used for flash self-programming library internal processing or the area where the data to be set is allocated in the FSL_Write function.
  The available range for the start address of the data buffer is the internal RAM excluding the self-RAM and addresses FFE20H-FFEFFH., as in the stack.

2. 2. 4 Flash self-programming library

Not all the flash functions are linked. Only the flash functions to be used are linked[Note].

- Memory allocation of the flash self-programming library

  Segments are assigned to the functions and variables used in the flash self-programming library. Areas used in the flash self-programming library can be specified at the specific locations.

  For details, refer to **6.2 Segments of Flash Functions**, or refer to the document "Release note" attached to the installer.

Note For the assembly language, delete unnecessary flash functions from the include file to link only the functions to be used.

2. 2. 5 Program area

This is the area in which the flash self-programming library and the user program using the flash self-programming library are allocated.

In flash self-programming of the RL78 microcontroller, the user program can operate during rewriting of the code flash memory because the code flash memory is rewritten by using the sequencer (background operation).

However, the program allocated in the code flash memory cannot be referred to during rewriting of the code flash memory, so some segments used by the user program and flash functions need to be allocated on the RAM depending on usage.

For details, refer to the sections of **CHAPTER 6 FLASH FUNCTIONS**.

2. 2. 6 ROMization of programs

To allocate the user program and library using flash self-programming on the RAM, the target program must be ROMized and allocated to the code flash memory, and the program must be copied to the RAM before it is used in flash self-programming.

For the ROMization function of the program allocated on the RAM, refer to the user's manual attached to the development tool used.

2. 3 Cautions on Programming Environment

(1) Do not execute the EEPROM emulation library or data flash library during the execution of flash self-programming. When using the EEPROM emulation library or data flash library, always execute up to FSL_Close to close the flash self-programming library.
When using the flash self-programming library after the execution of the EEPROM emulation library or data flash library, the flash self-programming processing must be started from the initialization function (FSL_Init).

(2) Do not execute the STOP or HALT instruction during the execution of flash self-programming. If the STOP or HALT instruction needs to be executed, pause flash self-programming with the FSL_StandBy function, or execute processing up to the FSL_Close function to close flash self-programming.

(3) The watchdog timer does not stop during the execution of self-programming. In the status check internal mode, do not make the watchdog timer interrupt interval shorter than the execution time of FSL_SetXXX, FSL_SwapActiveBootCluster, and FSL_InvertBootFlag.

(4) The code flash memory cannot be read during code flash memory operation by flash self-programming.

(5) Do not allocate the data buffer (arguments) or stack used in the flash function to an area starting from address 0xFFE20 (0xFE20).

(6) When using the data transfer controller (DTC) during the execution of flash self-programming, do not allocate the RAM area used by the DTC to the self-RAM or an area starting from address FFE20H(FE20H).

(7) Do not destroy the RAM area (including self-RAM) used by flash self-programming until flash self-programming is complete.

(8) Do not execute a flash function within interrupt processing. The flash function does not support nested execution of functions. If a flash function is executed within interrupt processing, operation cannot be guaranteed.

(9) When executing flash self-programming on the operating system, do not execute flash functions from multiple tasks. The flash function does not support multiple executions of functions. If a flash function is executed in multiple tasks, operation cannot be guaranteed.

(10) Before starting flash self-programming, the high-speed on-chip oscillator needs to be started. (The RL78 microcontroller hardware uses it during flash programming.)

(11) Note the following regarding the operating frequency of the CPU and the operating frequency value set with the initialization function (FSL_Init).

- When a frequency below 4 MHz[Note] is used as the operating frequency of the CPU, 1 MHz, 2 MHz, or 3 MHz can be used (a frequency such as 1.5 MHz that is not an integer value cannot be used). Also, set an integer value such as 1, 2, or 3 as the operating frequency value with the initialization function.
- When 4 MHz[Note] or a higher frequency is used as the operating frequency of the CPU, a frequency with decimal places can be used. However, set a rounded up integer value as the operating frequency with the initialization function (FSL_Init).
  (Example: For 4.5 MHz, set "5" with the initialization function.)
- This operating frequency is not the frequency of the high-speed on-chip oscillator (when the high-speed on-chip oscillator is not used to generate the operating frequency of the CPU).

Note For the range of the maximum operating frequency of the CPU, refer to the user's manual of the target RL78 microcontroller.

(12) Initialize the arguments (RAM) that are used by the flash self-programming library functions. When they are not initialized, a RAM parity error is detected and the RL78 microcontroller might be reset.
For a RAM parity error, refer to the user’s manual of the target RL78 microcontroller.

(13) In the code flash memory, only an area in the blank state or the area that has been erased can be written to. An area that has been written cannot be rewritten (overwritten) unless it has been erased. When rewriting is performed without erasing data, the code flash memory might be damaged.

(14) The R5F10266 product cannot use the flash self-programming function.

(15) Some RL78 microcontrollers do not support an interrupt during the execution of flash self-programming. Refer to the user’s manual of the target RL78 microcontroller to see whether the RL78 microcontroller to be used supports an interrupt during the execution of the flash self-programming.

(16) Some RL78 microcontrollers do not support the boot swap function. Refer to the user’s manual of the target RL78 microcontroller to see whether the RL78 microcontroller to be used supports the boot swap function.

(17) Some RL78 microcontrollers do not support the security setting function by the flash self-programming. Refer to the user’s manual of the target RL78 microcontroller to see whether the RL78 microcontroller to be used supports the security setting function by the flash self-programming.

(18) Do not arrange the segments FSL_BCD and FSL_BECD in the final address at a 64 Kbytes boundary (?FFFEH to ?FFFFH) when using the flash self-programming library. (This applies only to V2.20 and older versions of flash self-programming library; it does not apply to V2.21 and later versions because the issue regarding this restriction has been resolved.)

(19) Each segment (FSL_FCD, FSL_FECD, FSL_RCD, FSL_BCD, or FSL_BECD) of the Flash Self-Programming Library for the CC-RL compiler for the RL78 family cannot be allocated to extend across the 64 Kbytes boundary. Be sure to allocate segments so that they do not extend across the 64 Kbytes boundary.

(20) When using an assembler of the CC-RL compiler from Renesas Electronics, the hexadecimal prefix representation (0x..) cannot be mixed together with the suffix representation (..H). Specify the representation method by editing the symbol definition in fsl.inc to match the user environment.

fsl.inc

```
; FSL_INC_BASE_NUMBER_SUFFIX .SET 1
```

When symbol "FSL_INC_BASE_NUMBER_SUFFIX" is not defined (initial state), the prefix representation will be selected.

fsl.inc

```
FSL_INC_BASE_NUMBER_SUFFIX .SET 1
```

When symbol "FSL_INC_BASE_NUMBER_SUFFIX" is defined, the suffix representation will be selected.

CHAPTER 3 INTERRUPTS DURING EXECUTION OF FLASH SELF-PROGRAMMING

3. 1 Overview

Interrupt processing can be used even in the flash environment state[Note]. However, when the code flash memory is controlled, the interrupt vector of a normal user application cannot be used. The interrupt vector needs to be set to the RAM by using the interrupt vector change function (FSL_ChangeInterruptTable). Also, the interrupt routine needs to be allocated on the RAM. After the setting, execution branches to one vector on the RAM when any interrupt occurs. Therefore, if there are multiple interrupt sources for which you want to execute different processing, the interrupt sources need to be identified.

To restore the interrupt to the original vector state after the completion of the rewrite of the code flash memory, use the interrupt vector restoration function (FSL_RestoreInterruptTable) to restore the interrupt destination to the original state.

Note. Some RL78 microcontrollers do not support an interrupt during the execution of flash self-programming. Refer to the user's manual of the target RL78 microcontroller to see whether the RL78 microcontroller to be used supports an interrupt during the execution of the flash self-programming.

3. 2 Interrupts During Execution of Flash Self-Programming

Interrupts during code flash memory control cannot be received through the normal interrupt vector because the code flash memory cannot be referred to. Therefore, to receive interrupts, they need to be received on the RAM.

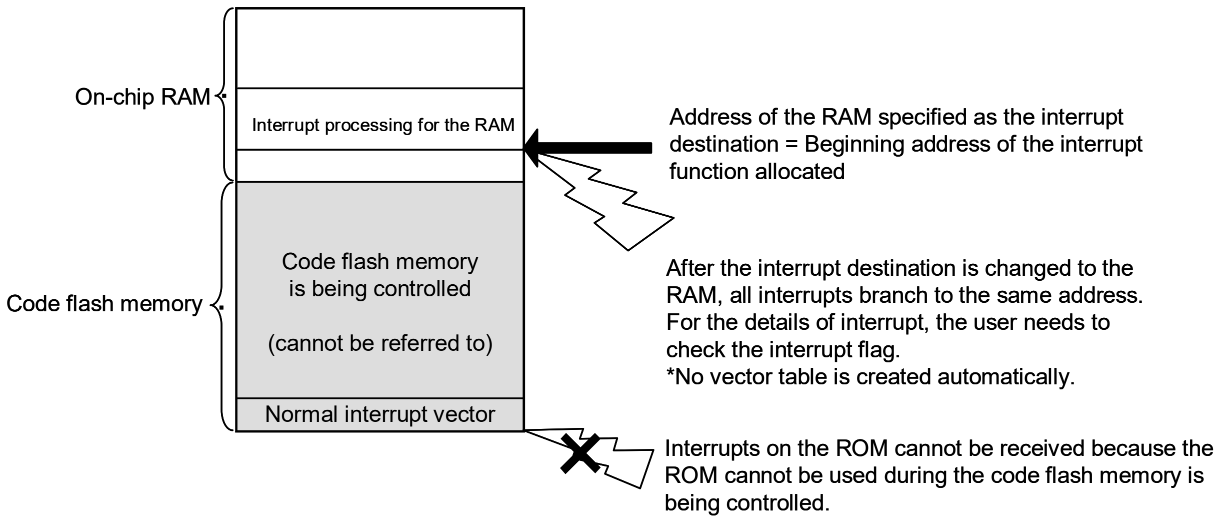

On-chip RAM
Interrupt processing for the RAM
Address of the RAM specified as the interrupt destination = Beginning address of the interrupt function allocated
Code flash memory
Code flash memory is being controlled
(cannot be referred to)
After the interrupt destination is changed to the RAM, all interrupts branch to the same address. For the details of interrupt, the user needs to check the interrupt flag. *No vector table is created automatically.
Normal interrupt vector
Interrupts on the ROM cannot be received because the ROM cannot be used during the code flash memory is being controlled.

Figure 3-1. Interrupts During Execution of Flash Self-Programming

3. 3 Cautions on Interrupts

- When changing the interrupt vector with the interrupt vector change function in the application, disable interrupts from the start to the end of the switching procedure.

- Do not specify a value over FFE20H as the changed destination of the interrupt vector address.

- Access to the code flash memory area is prohibited in interrupt processing during the execution of flash self-programming.

- The execution of flash functions is prohibited in interrupt processing.

- Save and restore the registers used in interrupt processing.

- The interrupt source can be determined by referring to SFR (interrupt request flag IF) when an interrupt occurs on the RAM. After the determination, clear the interrupt request flag (set to 0).

- The response time for interrupt processing on the RAM increases by a maximum of 20 clock cycles compared to the normal interrupt response time.

- To restore the original interrupt vector after the interrupt destination is changed with the interrupt vector change function, the interrupt vector restoration function must be executed. If the interrupt vector restoration function is not executed, the interrupt destination will remain changed even when flash self-programming is finished.

- When a reset is done after the interrupt destination is changed with the interrupt vector change function, the system starts up with the interrupt destination recovered.

- Some RL78 microcontrollers do not support an interrupt during the execution of flash self-programming. Refer to the user’s manual of the target RL78 microcontroller to see whether the RL78 microcontroller to be used supports an interrupt during the execution of the flash self-programming.

CHAPTER 4 SECURITY SETTINGS

The security function that prohibits rewriting of the user program written in the code flash memory is supported to prevent falsification of programs by a third party.

For details of security settings, refer to the manual of the target device.

Note. Some RL78 microcontrollers do not support the security setting function provided by the flash self-programming.
Refer to the user's manual of the target RL78 microcontroller to see whether the RL78 microcontroller to be used supports the security setting function by the flash self-programming.

4. 1 Security Flags

The flash self-programming library has functions to set the security flags (for details of the API of the functions, refer to **CHAPTER 6 FLASH FUNCTION**).

Functions setting the security flag	Description
FSL_SetBlockEraseProtectFlag	Sets the block erasure protection flag to protected.
FSL_SetWriteProtectFlag	Sets the write protection flag to protected.
FSL_SetBootClusterProtectFlag	Sets the boot area (boot cluster 0) rewrite protection flag to protected.

4. 2 Flash Shield Window Function

One of the security functions that can be used during the execution of flash self-programming is the flash shield window function. The flash shield window function is a security function that prohibits writing and erasure outside the specified window range only during the execution of flash self-programming. The window range can be set by specifying the start block and end block. The areas other than the window range are write-protected and erasure-protected during the execution of flash self-programming.

Function setting the flash shield window	Description
FSL_SetFlashShieldWindow	Sets the flash shield window.

CHAPTER 5 BOOT SWAP FUNCTION

5. 1 Overview

When rewriting fails due to an instantaneous power interruption or resetting caused by an external factor while the area in which the vector table data, program basic functions, and the flash self-programming library are allocated is being rewritten, the data being rewritten is destroyed, so restart or rewrite of the user program due to the subsequent reset cannot be done. The boot swap function avoids this situation[Note].

Note To use this function, the RL78 microcontroller supporting the boot swap function is required. To find if your RL78 microcontroller supports the boot swap function, refer to the user's manual of the target RL78 microcontroller.

5. 2 Boot Swap Function

The boot swap function replaces the boot program area Boot Cluster 0[Note] with the boot swap target area Boot Cluster 1[Note].

Before performing rewrite processing, a new boot program is written to Boot Cluster 1 in advance. Boot Cluster 1 and Boot Cluster 0 are swapped to make Boot Cluster 1 the boot program area.

As a result, the program operates normally because booting is done from Boot Cluster 1 in the next reset start even when an instantaneous power interruption occurs during rewriting of the boot program area. After that, erasure or write processing to Boot Cluster 0 can be performed if necessary.

Note Boot Cluster 0: Boot program area
Boot Cluster 1: Boot swap target area

5. 3 Boot Swapping Procedure

Figure 5-1 shows an example of the flow of boot swapping using the flash self-programming library.

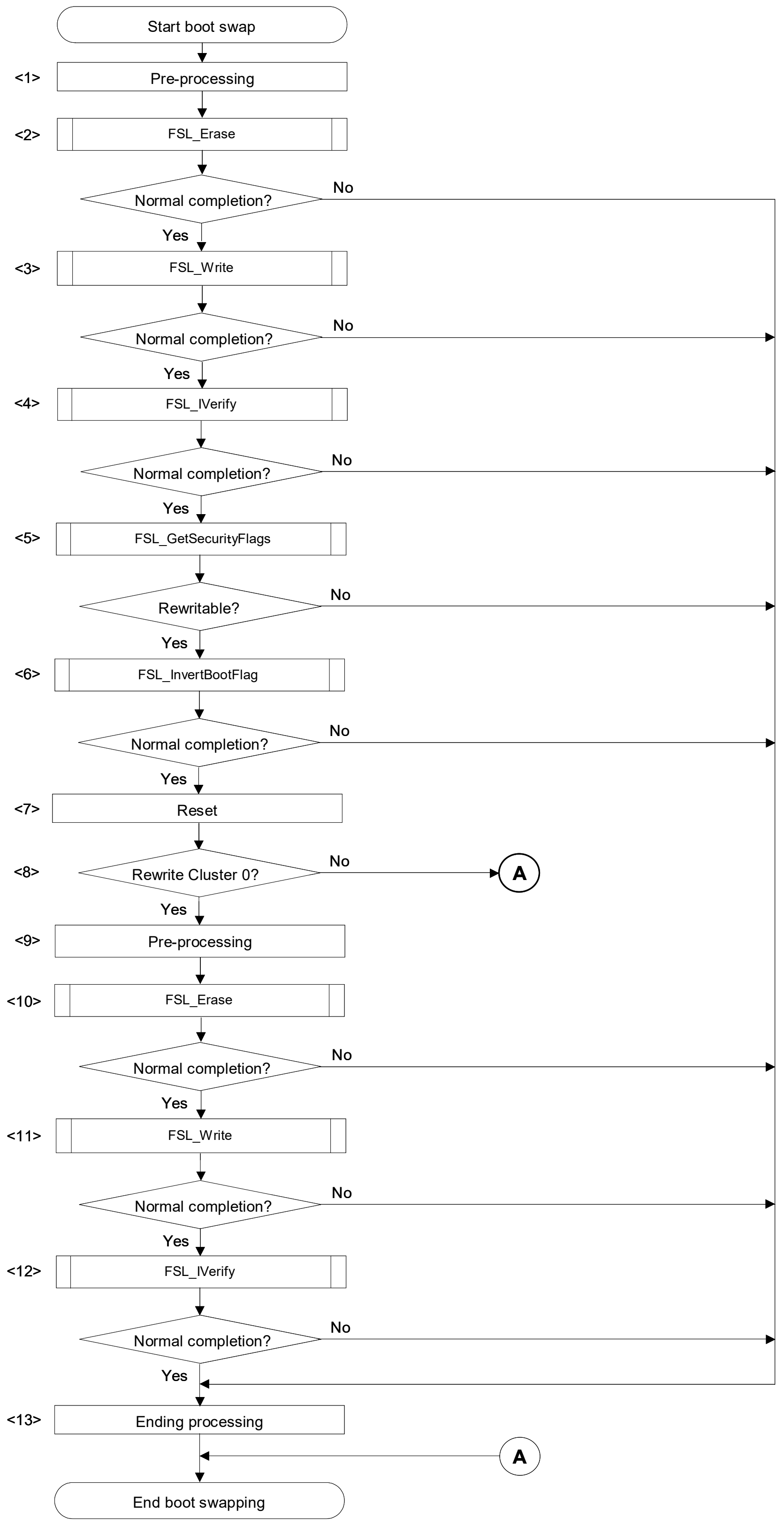


Figure 5-1. Example of Flow of Boot Swapping

<1> Preprocessing

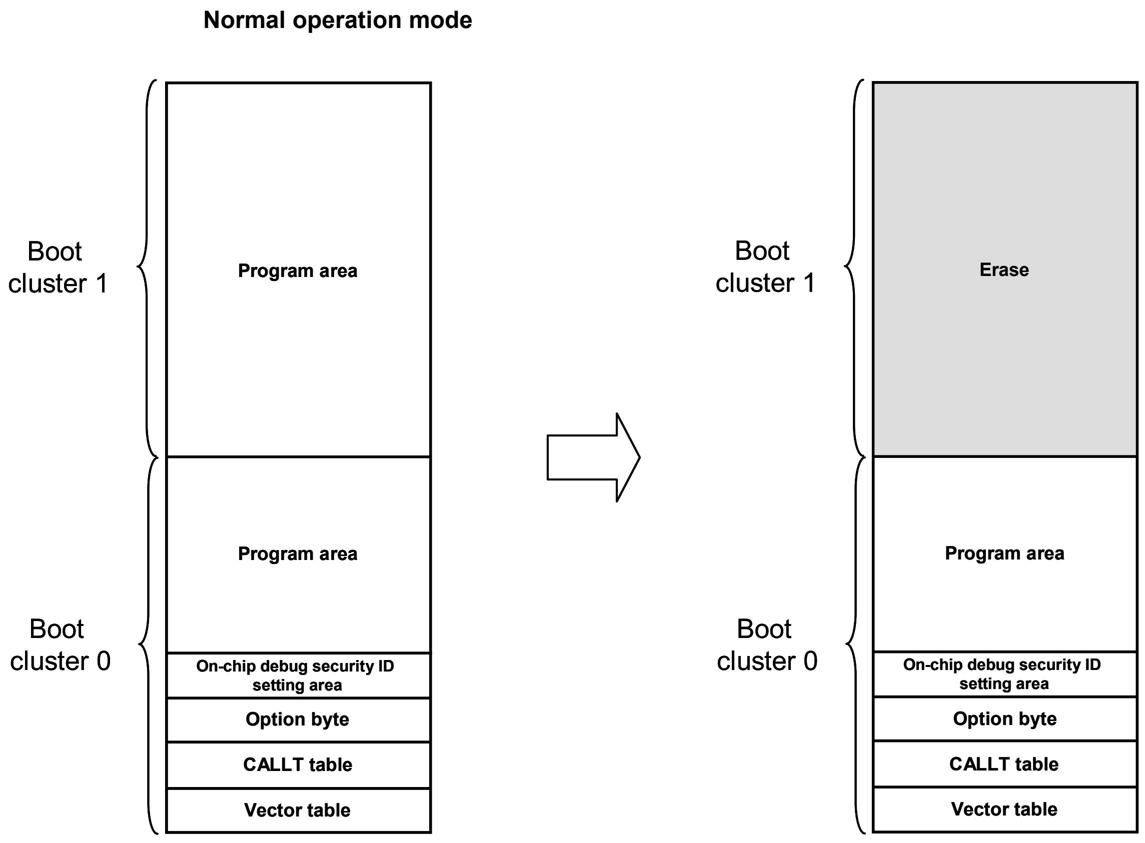
Boot swap preprocessing

- Setting of the software environment (reserving data buffer, etc.)
- Initialization of flash self-programming (execution of the FSL_Init function)
- Start of the flash environment (execution of the FSL_Open function)
- Preparation processing of the flash function (execution of the FSL_PrepareFunctions function)
- Preparation processing of the flash function (extension function) (execution of the FSL_PrepareExtFunctions function)
- RAM expansion processing of the ROMization code if the rewrite program is ROMized

<2> Erasure of Boot Cluster 1

All blocks in Boot Cluster 1 are erased by calling the FSL_Erase function.

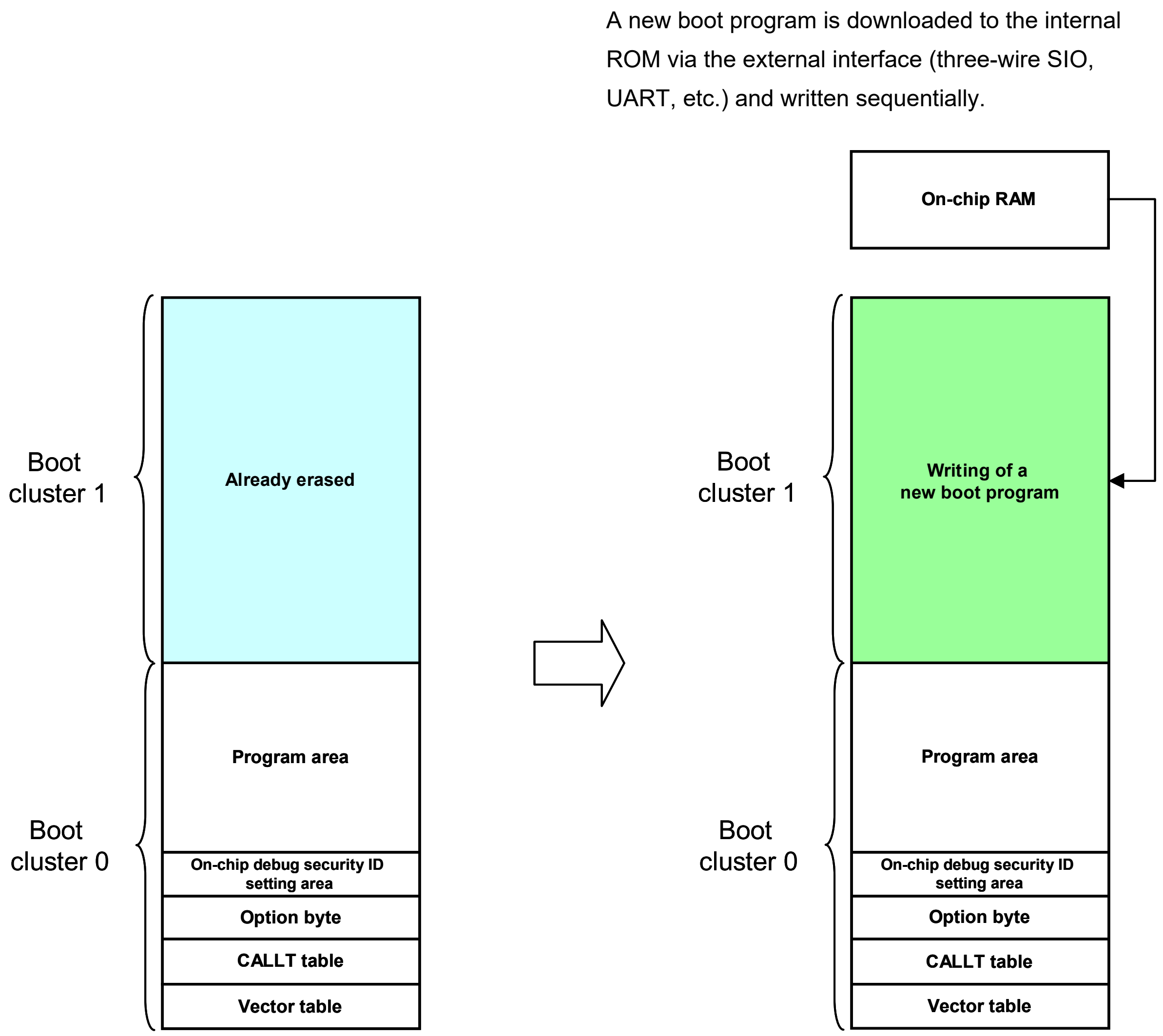
Remark The FSL_Erase function performs erasure in units of blocks.



<3> Copying of the new boot program to Boot Cluster 1

A new boot program (the program you want to allocate as the boot program area after boot swap processing) is written to Boot Cluster 1 by calling the FSL_Write function.

Remark The FSL_Write function performs writing in units of words (1 word = 4 bytes, up to 64 words (256 bytes))


A new boot program is downloaded to the internal ROM via the external interface (three-wire SIO, UART, etc.) and written sequentially.
Boot cluster 1
Already erased
Boot cluster 0
Program area
On-chip debug security ID setting area
Option byte
CALLT table
Vector table
On-chip RAM
Boot cluster 1
Writing of a new boot program
Boot cluster 0
Program area
On-chip debug security ID setting area
Option byte
CALLT table
Vector table

<4> Verification of Boot Cluster 1

All blocks of Boot Cluster 1 to which writing has been done are verified by calling the FSL_IVerify function.

Remark The FSL_IVerify function performs verification in units of blocks.

<5> Confirmation of the boot swap bit (recommended)

The security flag information is obtained by calling the FSL_GetSecurityFlags function. Check that the boot area (Boot Cluster 0) rewrite protection flag is 1 (permitted).

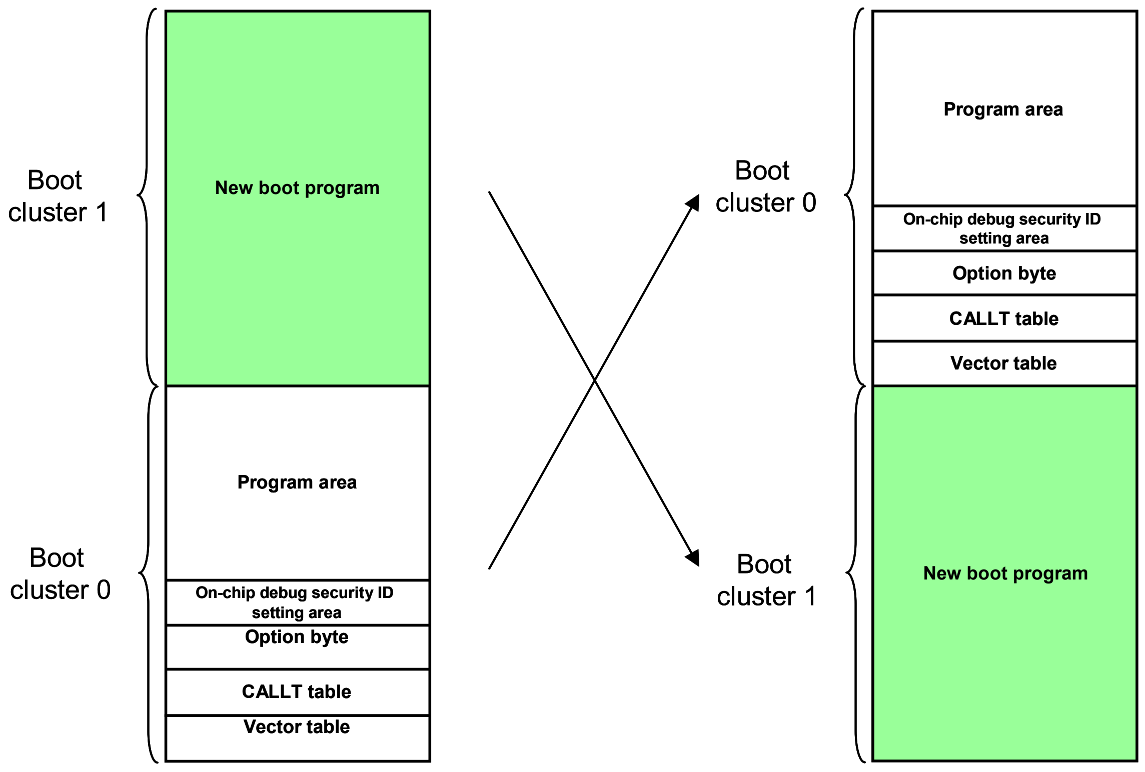
Remark If the (Boot Cluster 0) rewrite protection flag is 0 (protected), an error occurs when the FSL_InvertBootFlag function is called in <6>.

<6> Setting of the boot swap bit

Switching of the boot flag is performed by executing the FSL_InvertBootFlag function.

<7> Occurrence of an event

When a reset is generated, Boot Cluster 1 becomes the boot program area.



<8> Ending of swap processing (Boot Cluster 1)

Swap processing for Boot Cluster 1 is finished after steps <2> to <7>.

If Boot Cluster 0 does not have to be rewritten, terminate processing.

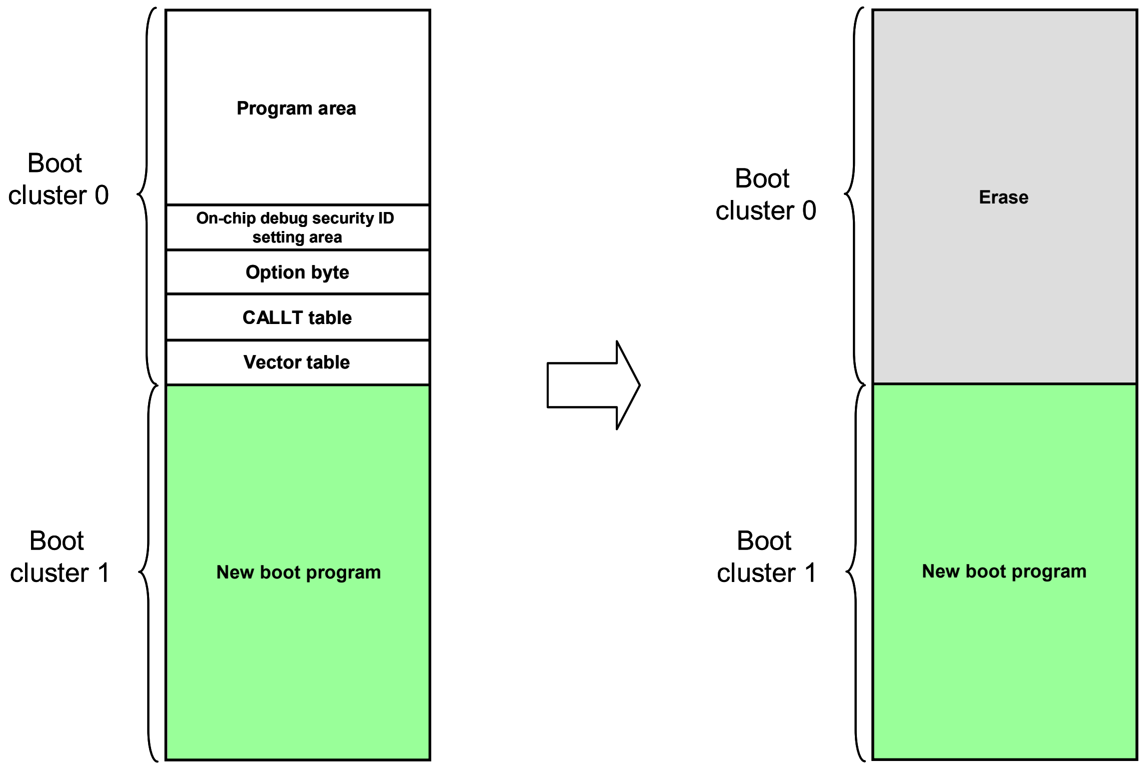
If Boot Cluster 0 has to be rewritten, perform the processing of <9> and on.

<9> Preprocessing

The same processing as <1> is performed.

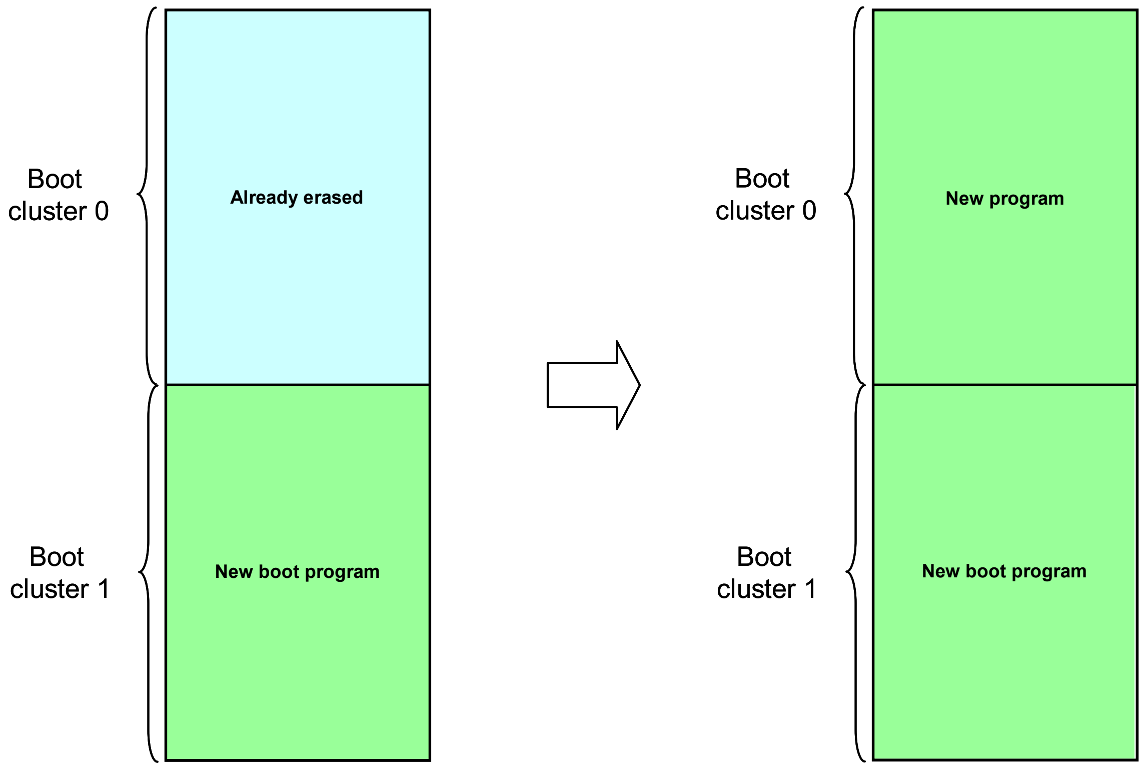
<10> Erasure of Boot Cluster 0

All blocks of Boot Cluster 0 are erased by calling the FSL_Erase function.



<11> Writing of the new program to Boot Cluster 0

The content of the new program is written to Boot Cluster 0 by calling the FSL_Write function.



<12> Verification of Boot Cluster 0

All blocks of Boot Cluster 0 to which writing has been done are verified by calling the FSL_IVerify function.

<13> End processing

As the end processing of boot swapping, the FSL_Close function is called.

5. 4 Cautions on Boot Swapping

- Boot swapping cannot be executed when the boot area (Boot Cluster 0) rewrite protection flag is set to 0 (protected).

- After a function for boot swapping is executed, control returns to the area from which the function was called. Therefore, the program that calls boot swap functions must not be stored in either Boot Cluster 0 or 1.
  Remark Applicable function: FSL_SwapBootCluster and FSL_SwapActiveBootCluster

- Specific cautions should be taken for each flash function providing the boot swap function. For details, refer to **CHAPTER 6 FLASH FUNCTIONS**.

CHAPTER 6 FLASH FUNCTIONS

This chapter describes the details of the flash functions (functions in the flash self-programming library).

6. 1 Types of Flash Functions

The flash self-programming library consists of the following flash functions.

Table 6-1. List of Flash Functions

Function name	Description	Basic function[Note]	Function for G11[Note]
FSL_Init	Initialization of the flash self-programming environment	○	○
FSL_Open	Starting of the flash environment (start declaration of flash self-programming)	○	○
FSL_Close	Ending of the flash environment (end declaration of flash self-programming)	○	○
FSL_PrepareFunctions	Preparation processing for flash functions	○	○
FSL_PrepareExtFunctions	Preparation processing for flash functions (extension functions)	-	○
FSL_ChangeInterruptTable	Interrupt vector change processing (changing the interrupt destination from ROM to RAM)	-	-
FSL_RestoreInterruptTable	Interrupt vector restoration processing (changing the interrupt destination from RAM to ROM)	-	-
FSL_BlankCheck	Blank checking of the specified block	○	○
FSL_Erase	Erasure of the specified block.	○	○
FSL_IVerify	Verification (internal verification) of the specified block	○	○
FSL_Write	Writing of 1 word to 64 words data into the specified address (1 word=4 bytes)	○	○
FSL_GetSecurityFlags	Acquisition of security information	-	-
FSL_GetBootFlag	Acquisition of boot flag information	-	○
FSL_GetSwapState	Acquisition of swap information	-	○
FSL_GetBlockEndAddr	Acquisition of the final address of the specified block	-	-
FSL_GetFlashShieldWindow	Acquisition of the start block number and end block number of the flash shield window	-	○
FSL_SwapBootCluster	Execution of boot swapping and jumping to the registered address of the reset vector	-	○
FSL_SwapActiveBootCluster	Inverting of the current value of the boot flag and execution of boot swapping	-	-
FSL_InvertBootFlag	Inverting of the current value of the boot flag	-	○
FSL_SetBlockEraseProtectFlag	Setting of the block erasure protection flag to protected	-	-
FSL_SetWriteProtectFlag	Setting of the write protection flag to protected	-	-
FSL_SetBootClusterProtectFlag	Setting of the boot area (boot cluster 0) rewrite protection flag to protected	-	-
FSL_SetFlashShieldWindow	Setting of the start block and end block of the flash shield window	-	○
FSL_StatusCheck	Status check processing	○	-
FSL_StandBy	Pause processing of flash self-programming	-	-
FSL_WakeUp	Restart processing of flash self-programming	-	-
FSL_ForceReset	Resetting of the microcontroller in use	-	○
FSL_GetVersionString	Version acquisition processing of the flash self-programming library	-	-

Note For the RL78/G12, L12, and G1G groups, only basic functions are supported and the other functions are not supported. For the RL78/G11 group, only the status check internal mode is supported as the status check mode.

6. 2 Segments of Flash Functions

<R> The codes of the flash functions are divided into some groups and must be allocated to specified areas. These groups are used as segments for memory allocation in the CA78K0R compiler. They are used as sections in the CC-RL compiler and the LLVM compiler.

The segments (sections) are classified as follows.

- FSL_FCD: A group of functions that initialize the environment. They can be allocated to the ROM or RAM.
- FSL_FECD: A group of functions that read security information, etc. They can be allocated to the ROM or RAM.
- FSL_RCD: A group of functions required to rewrite the flash. They can be allocated to the RAM. There are some usage restrictions[Note] when they are allocated to the ROM.
- FSL_BCD: Area used by the FSL_PrepareFunctions function. They can be allocated to the ROM or RAM.
- FSL_BECD: Area used by the FSL_PrepareExtFunctions function. They can be allocated to the ROM or RAM.

<R> When using the flash self-programming library for the CC-RL compiler or the LLVM compiler, read "segment" as "section" in the following descriptions.

Table 6-2. List of Flash Function Segments

Function name	Segment name	ROM allocation	RAM allocation
FSL_Init	FSL_FCD	○	○
FSL_Open	FSL_FCD	○	○
FSL_Close	FSL_FCD	○	○
FSL_PrepareFunctions	FSL_FCD / FSL_BCD	○	○
FSL_PrepareExtFunctions	FSL_FCD / FSL_BECD	○	○
FSL_ChangeInterruptTable	FSL_FCD	○	○
FSL_RestoreInterruptTable	FSL_FCD	○	○
FSL_BlankCheck	FSL_RCD	△ Note	○
FSL_Erase	FSL_RCD	△ Note	○
FSL_IVerify	FSL_RCD	△ Note	○
FSL_Write	FSL_RCD	△ Note	○
FSL_GetSecurityFlags	FSL_FECD	○	○
FSL_GetBootFlag	FSL_FECD	○	○
FSL_GetSwapState	FSL_FECD	○	○
FSL_GetBlockEndAddr	FSL_FECD	○	○
FSL_GetFlashShieldWindow	FSL_FECD	○	○
FSL_SwapBootCluster	FSL_RCD	△ Note	○
FSL_SwapActiveBootCluster	FSL_RCD	×	○
FSL_InvertBootFlag	FSL_RCD	△ Note	○
FSL_SetBlockEraseProtectFlag	FSL_RCD	△ Note	○
FSL_SetWriteProtectFlag	FSL_RCD	△ Note	○
FSL_SetBootClusterProtectFlag	FSL_RCD	△ Note	○
FSL_SetFlashShieldWindow	FSL_RCD	△ Note	○
FSL_StatusCheck	FSL_RCD	△ Note	○
FSL_StandBy	FSL_RCD	△ Note	○
FSL_WakeUp	FSL_RCD	△ Note	○
FSL_ForceReset	FSL_RCD	△ Note	○
FSL_GetVersionString	FSL_FCD	○	○

Note There are the following usage restrictions when they are allocated to the ROM.

- Do not use the FSL_SwapActiveBootCluster() function.
- Set the status check mode to the status check internal mode with the FSL_Init() function.

6. 3 Interrupts and BGO (Background Operation)

The flash functions can be divided into processing that does not use the sequencer and processing that uses the sequencer, and they differ in the interrupt reception methods. For the processing that uses the sequencer, BGO (background operation) can be performed.

The following table shows a list of the flash functions with the presence of sequencer control and their interrupt reception areas.

Table 6-3. List of Interrupt Reception Areas and BGO of Flash Functions

Function name	Sequencer control	Interrupt reception[Note1]	BGO function
FSL_Init	No	ROM: Allowed RAM: Allowed	No
FSL_Open			
FSL_Close			
FSL_PrepareFunctions			
FSL_PrepareExtFunctions			
FSL_ChangeInterruptTable		Not allowed	
FSL_RestoreInterruptTable			
FSL_BlankCheck	Yes	ROM: Not allowed RAM: Allowed	Yes Only on the RAM[Note2]
FSL_Erase			
FSL_IVerify			
FSL_Write			
FSL_GetSecurityFlags	No	ROM: Allowed RAM: Allowed	No
FSL_GetBootFlag			
FSL_GetSwapState			
FSL_GetBlockEndAddr			
FSL_GetFlashShieldWindow			
FSL_SwapBootCluster		Not allowed	
FSL_SwapActiveBootCluster	Yes	ROM: Not allowed RAM: Allowed	Yes Only on the RAM[Note2]
FSL_InvertBootFlag			
FSL_SetBlockEraseProtectFlag			
FSL_SetWriteProtectFlag			
FSL_SetBootClusterProtectFlag			
FSL_SetFlashShieldWindow			
FSL_StatusCheck[Note3]			No
FSL_StandBy[Note3]			
FSL_WakeUp[Note3]			
FSL_ForceReset	No	ROM: Allowed	
FSL_GetVersionString		RAM: Allowed	

Notes 1. Whether or not interrupt reception during the execution of the function or during sequencer control is allowed
ROM: Normal vector interrupt; RAM: Interrupt on the RAM
2. To execute BGO, the user program and part of the library must be allocated on the RAM.
3. This function does not have the BGO function because it checks the state of the sequencer or stops and restarts the sequencer control during block erasure.

6. 4 Status Check Mode

For the functions that can perform background operation by using the sequencer, a status check must be performed to check the control state of the code flash memory.

There are the following two status check modes, which can be set with the FSL_Init() function. They have different status check methods.

- Status check user mode[Note]

  After the control setting of the sequencer is done by the flash function, execution returns to the user program. The user needs to check the status of the sequencer with the status check function (FSL_StatusCheck), but the user program can operate until the sequencer processing is completed. The user programs and interrupt program to operate during sequencer control need to be allocated on the RAM.

- Status check internal mode[Note]

  Execution does not return to the user program until the status of the sequencer is checked with the flash function and the sequencer processing is completed. To process interrupts during the execution of the function (during sequencer control), the interrupt program needs to be allocated on the RAM.

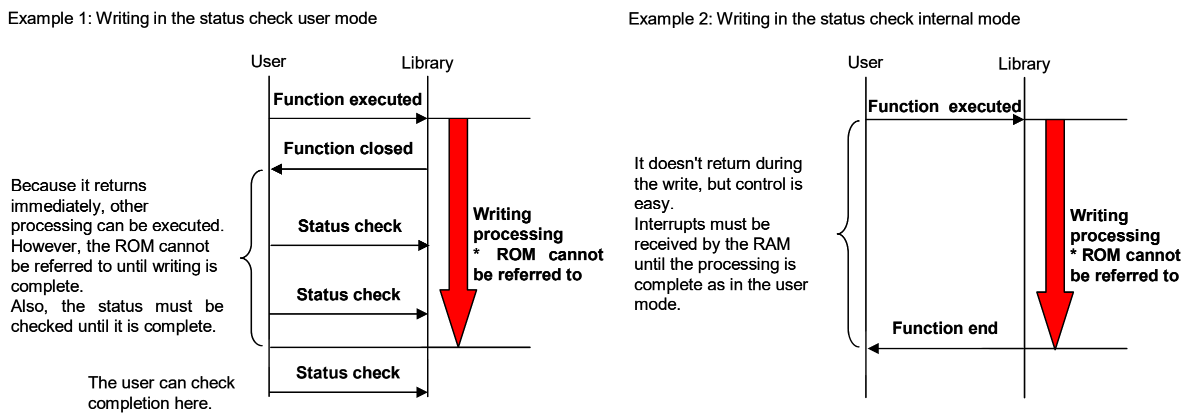

Example 1: Writing in the status check user mode
User
Library
Function executed
Function closed
Because it returns immediately, other processing can be executed. However, the ROM cannot be referred to until writing is complete. Also, the status must be checked until it is complete.
Status check
Writing processing * ROM cannot be referred to
Status check
Status check
The user can check completion here.
Example 2: Writing in the status check internal mode
User
Library
Function executed
It doesn't return during the write, but control is easy. Interrupts must be received by the RAM until the processing is complete as in the user mode.
Writing processing * ROM cannot be referred to
Function end

Figure 6-1 Example of Status Check Mode

Note Only the status check internal mode can be used for the segment (FSL_RCD) of some flash functions with restrictions on the allocation on the ROM or when the user program is allocated on the ROM.

Table 6-4. List of Status Checking of Flash Functions

Function name	Sequencer control	Status check
FSL_Init	No	Not required
FSL_Open		
FSL_Close		
FSL_PrepareFunctions		
FSL_PrepareExtFunctions		
FSL_ChangeInterruptTable		
FSL_RestoreInterruptTable		
FSL_BlankCheck	Yes	Required
FSL_Erase		
FSL_IVerify		
FSL_Write		
FSL_GetSecurityFlags	No	Not required
FSL_GetBootFlag		
FSL_GetSwapState		
FSL_GetBlockEndAddr		
FSL_GetFlashShieldWindow		
FSL_SwapBootCluster		
FSL_SwapActiveBootCluster	Yes	Required
FSL_InvertBootFlag		
FSL_SetBlockEraseProtectFlag		
FSL_SetWriteProtectFlag		
FSL_SetBootClusterProtectFlag		
FSL_SetFlashShieldWindow		
FSL_StatusCheck^{Note1}		Not required
FSL_StandBy^{Note1}		
FSL_WakeUp^{Note1,2}		
FSL_ForceReset	No	
FSL_GetVersionString		

Notes 1. The processing of this function does not require status checking because it is the function to perform status checking or the function to stop or restart the sequencer control during block erasure.

2. To restart the block erasure processing (FSL_Erase), status checking is required to check the erasure state of the block.

6. 4. 1　Status Check User Mode

In the status check user mode, the back ground operation (BGO) can be performed on the RAM.　The operation examples of each procedure are shown in the following figure.

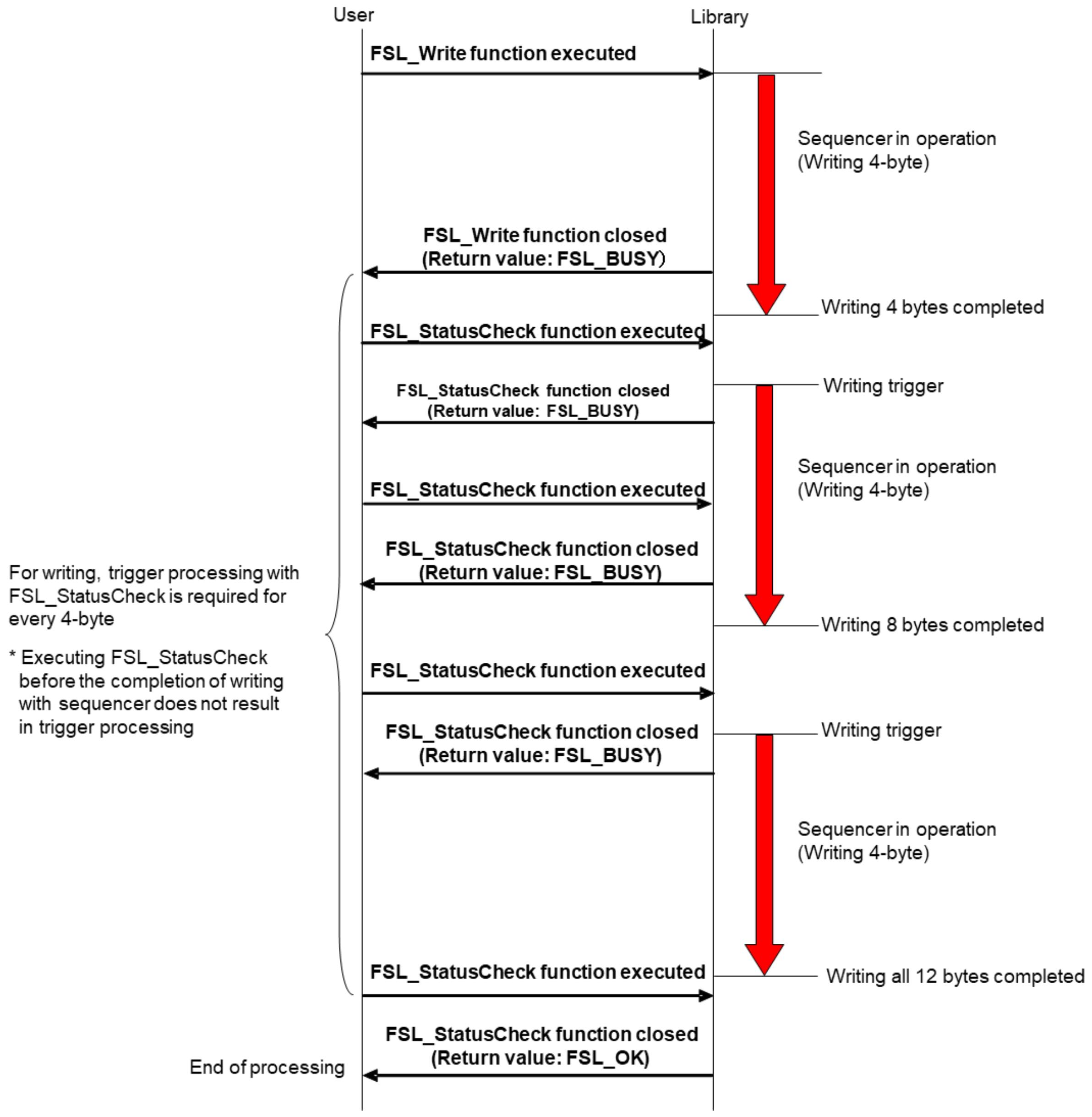

User
Library
FSL_Write function executed
Sequencer in operation (Writing 4-byte)
FSL_Write function closed (Return value: FSL_BUSY)
Writing 4 bytes completed
FSL_StatusCheck function executed
Writing trigger
FSL_StatusCheck function closed (Return value: FSL_BUSY)
Sequencer in operation (Writing 4-byte)
FSL_StatusCheck function executed
FSL_StatusCheck function closed (Return value: FSL_BUSY)
For writing, trigger processing with FSL_StatusCheck is required for every 4-byte
* Executing FSL_StatusCheck before the completion of writing with sequencer does not result in trigger processing
Writing 8 bytes completed
FSL_StatusCheck function executed
Writing trigger
FSL_StatusCheck function closed (Return value: FSL_BUSY)
Sequencer in operation (Writing 4-byte)
FSL_StatusCheck function executed
Writing all 12 bytes completed
FSL_StatusCheck function closed (Return value: FSL_OK)
End of processing

Figure 6-2　Example 1 of Status Check User Mode (FSL_Write: When writing 12 bytes data)

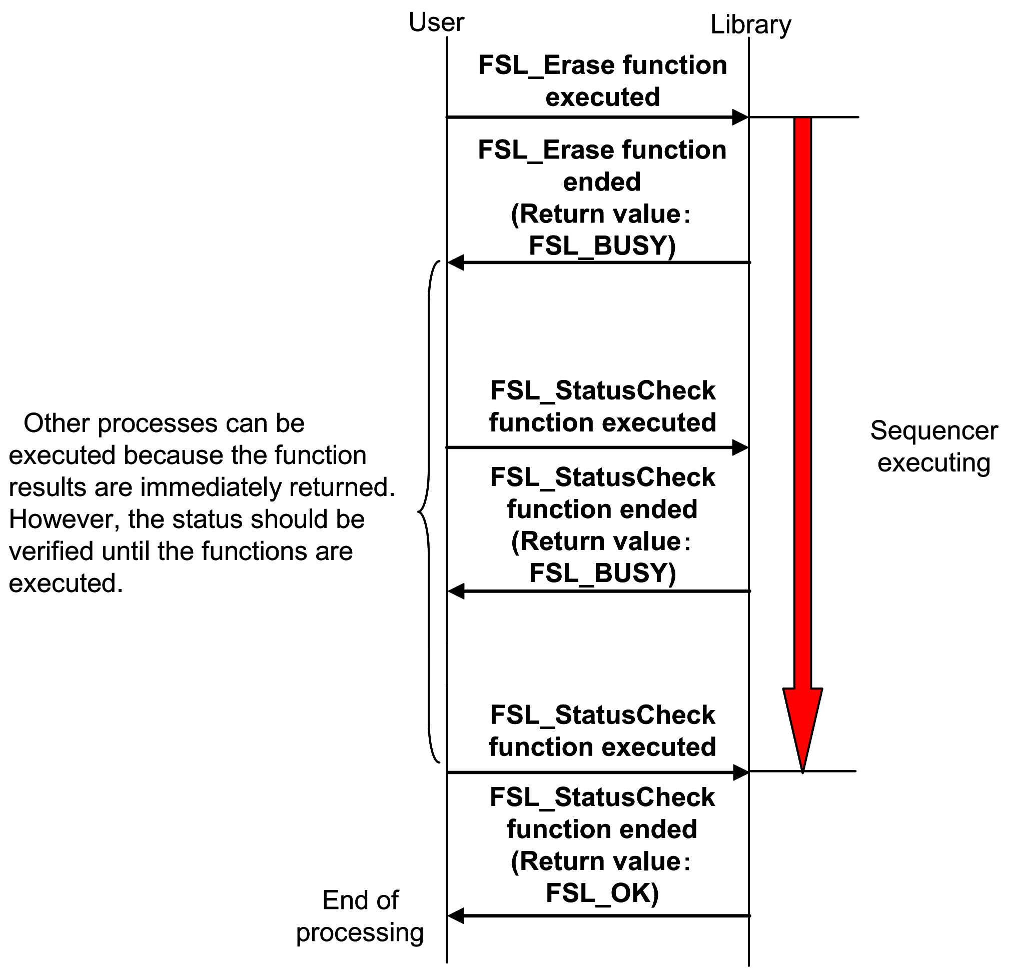

User
Library
FSL_Erase function executed
FSL_Erase function ended (Return value: FSL_BUSY)
FSL_StatusCheck function executed
FSL_StatusCheck function ended (Return value: FSL_BUSY)
FSL_StatusCheck function executed
FSL_StatusCheck function ended (Return value: FSL_OK)
Other processes can be executed because the function results are immediately returned. However, the status should be verified until the functions are executed.
Sequencer executing
End of processing

Figure 6-3 Example 2 of Status Check User Mode (Other than FSL_Write)

6. 5 Pausing of Flash Self-Programming

When you need to pause the sequencer control during block erasure while the flash function is being executed in the status check user mode, the stand-by function (FSL_StandBy) can be used to pause the erasure processing to put flash self-programming to the pause state. When the stand-by function is executed in a state other than block erasure state, it waits until the previous processing is completed, and makes a transition to the pause state after the completion.

When a transition to the pause state occurs, the code flash memory cannot be controlled. To return from the pause state, the wakeup function (FSL_WakeUp) needs to be executed. If the block erasure is suspended, the pause state is released to restart the block erasure. In other cases, only the pause state is released.

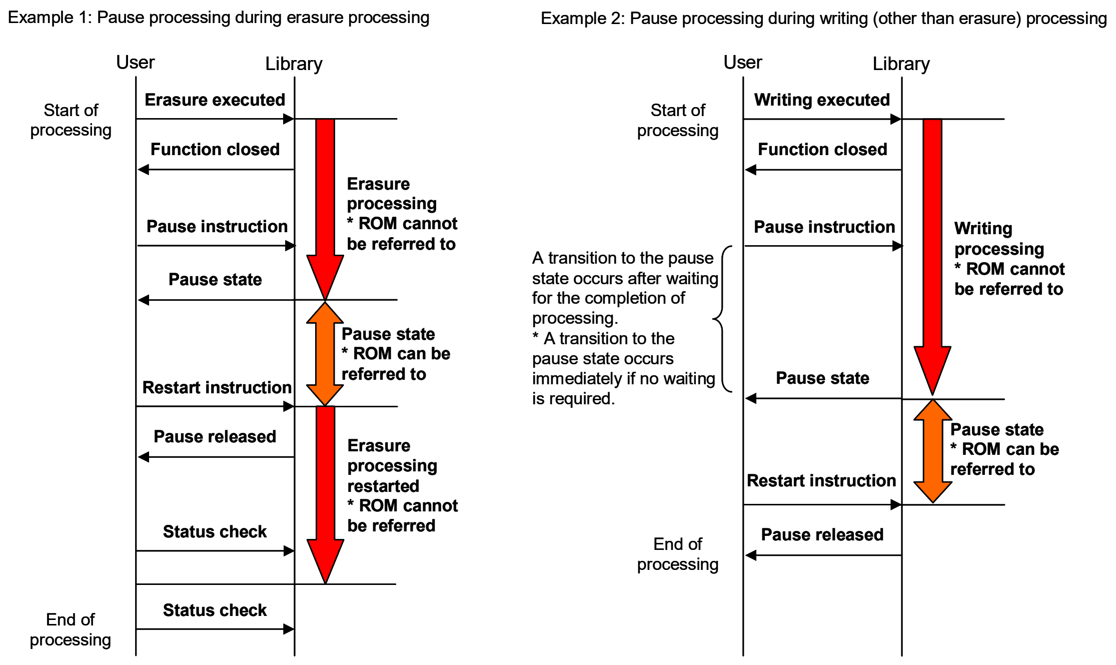

Example 1: Pause processing during erasure processing
User
Library
Start of processing
Erasure executed
Function closed
Erasure processing * ROM cannot be referred to
Pause instruction
Pause state
Pause state * ROM can be referred to
Restart instruction
Pause released
Erasure processing restarted * ROM cannot be referred
Status check
Status check
End of processing
Example 2: Pause processing during writing (other than erasure) processing
User
Library
Start of processing
Writing executed
Function closed
Pause instruction
Writing processing * ROM cannot be referred to
A transition to the pause state occurs after waiting for the completion of processing.
* A transition to the pause state occurs immediately if no waiting is required.
Pause state
Pause state * ROM can be referred to
Restart instruction
Pause released
End of processing

Figure 6-4 Example of Pausing of Flash Self-Programming

Table 6-5. List of Execution States of Stand-by Function

Function name	Sequencer control	State when the stand-by function is executed
FSL_Init	No	Not available
FSL_Open		
FSL_Close		
FSL_PrepareFunctions		
FSL_PrepareExtFunctions		
FSL_ChangeInterruptTable		
FSL_RestoreInterruptTable		
FSL_BlankCheck	Yes	Waits until the processing is completed, and makes a transition to the pause state.
FSL_Erase		Pauses the erasure processing, and makes a transition to the pause state.
FSL_IVerify		Waits until the processing is completes, and makes a transition to the pause state.
FSL_Write		
FSL_GetSecurityFlags	No	Not available
FSL_GetBootFlag		
FSL_GetSwapState		
FSL_GetBlockEndAddr		
FSL_GetFlashShieldWindow		
FSL_SwapBootCluster		
FSL_SwapActiveBootCluster	Yes	Waits until the processing is completed, and makes a transition to the pause state.
FSL_InvertBootFlag		
FSL_SetBlockEraseProtectFlag		
FSL_SetWriteProtectFlag		
FSL_SetBootClusterProtectFlag		
FSL_SetFlashShieldWindow		
FSL_StatusCheck		Not available
FSL_StandBy		
FSL_WakeUp		
FSL_ForceReset	No	
FSL_GetVersionString		
Flash function idle state		Makes a transition to the pause state.

6. 6 List of Data Types and Return Values

The data types are as follows

Table 6-6. List of Data Types

Definition	Data type	Description
fsl_u08	unsigned char	1-byte (8-bit) unsigned integer
fsl_u16	unsigned int	2-byte (16-bit) unsigned integer
fsl_u32	unsigned long int	4-byte (32-bit) unsigned integer

The meaning of each return value is as follows.

Table 6-7. List of Return Values

Definition	Return value	Description
FSL_OK	0x00	Normal completion
FSL_ERR_PARAMETER	0x05	Parameter error - The specified parameter has an error.
FSL_ERR_PROTECTION	0x10	Protect error - The target area is protected.
FSL_ERR_ERASE	0x1A	Erasure error - Erasure of the target area failed.
FSL_ERR_BLANKCHECK	0x1B	Blank check error - The target area is not in the blank state.
FSL_ERR_IVERIFY	0x1B	Internal verification error - An error occurred during internal verification processing of the target area.
FSL_ERR_WRITE	0x1C	Writing error - Writing to the target area failed.
FSL_ERR_FLOW	0x1F	Flow error - The processing of the flash function executed immediately before has not been completed. - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.
FSL_IDLE	0x30	Idle state - Flash self-programming is not executed.
FSL_SUSPEND	0x43	Pause state - Flash self-programming is paused.
FSL_BUSY	0xFF	Execution start of the flash function or the flash function in execution - The function of the flash function is in execution.

<R> The general register used to pass a return value differs between the RENESAS CA78K0R, the RENESAS CC-RL and the LLVM compilers.

The return value and the general register used in each compiler are as follows.

Table 6-8. List of Return Values Used in Each Compiler

Development tool	Return value	
	C language	Assembly language
RENESAS CA78K0R compiler	fsl_u08	C(General-purpose register)
RENESAS CC-RL compiler	fsl_u08	A(General-purpose register)
<R> LLVM compiler	fsl_u08	A(General-purpose register)

6. 7 Description of Flash Functions

The flash functions are described in the following format.

Flash function name

[Overview]

Describes the function overview of this function.

[Format]

<C language>

Describes the format to call this function from a user program written in the C language.

<Assembler>

Describes the format to call this function from a user program written in the assembly language.

Note Header file for assembly language (fsl.inc) are provided only for the CA78K0R compiler and the CC-RL compiler. When using assembly language with the LLVM compiler, refer to "fsl.inc" for the CC-RL compiler to create a header file that defines the status code returned by the FSL function.

[Presetting]

Describes the presetting of this function.

[Function]

Describes the function details and cautions of this function.

[Register State After Call]

Describes the register state after this function is called.

[Argument]

Describes the argument of this function.

[Return Value]

Describes the return values from this function.

[Flow] (FSL_SwapBootCluster() function only)

Describes the internal flow of this function.

[Operation Example] (FSL_ChangeInterruptTable() and FSL_RestoreInterruptTable() functions only)

Describes operation examples for using this function.

FSL_Init

[Overview]

Initialization of the flash self-programming environment

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_Init( __far fsl_descriptor_t* descriptor_pstr)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_Init(const __far fsl_descriptor_t* descriptor_pstr)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_Init(const __far fsl_descriptor_t* descriptor_pstr)
                                          __attribute__ ((section ("FSL_FCD")))
```

<Assembler>

```
CALL !_FSL_Init or CALL !!_FSL_Init
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

- The flash self-programming library, data flash library, the program used to operate the data flash memory, and EEPROM emulation library are not executed or have been ended.
- The high-speed on-chip oscillator is running.

[Function]

- Reserves and initializes the self-RAM used for flash self-programming. If a self-RAM[Note1] exists, do not use it until flash self-programming is finished.
- Define the flash memory programming mode [Note2] for flash self-programming in the argument fsl_flash_voltage_u08.

  0x00: Full-speed mode

  Other than above: Wide voltage mode
- Set the operating frequency of the CPU in the argument fsl_u08 fsl_frequency_u08. The set value is used for the calculation of timing data in the flash self-programming library.[Note3]

  For the value of the operating frequency of the CPU (fsl_frequency_u08), note the following.
  - When a frequency below 4 MHz[Note4] is used as the operating frequency of the RL78 microcontroller, 1 MHz, 2 MHz, or 3 MHz can be used (a frequency such as 1.5 MHz that is not an integer value cannot be used). Set an integer value such as 1, 2, or 3 as the operating frequency value with the initialization function.
  - When a frequency equal to or over 4 MHz[Note4] is used as the operating frequency of the RL78 microcontroller, a frequency with decimal places can be used. However, set a rounded up integer value as the operating frequency with the initialization function (FSL_Init).
    (Example: For 4.5 MHz, set "5" with the initialization function.)
  - This operating frequency is not the frequency of the high-speed on-chip oscillator.

- Set the status check mode in the argument fsl_auto_status_check_u08.Note5 For differences between the status check user mode and status check internal mode, refer to **2.1 Hardware Environment** or **6.4 Status Check Mode**.

  0x00: Status check user mode

  Other than above: Status check internal mode

Notes 1. For the self-RAM, refer to the document "Release note" attached to the installer, or refer to the user's manual of the target RL78 microcontroller.
2. For details of the flash memory programming mode, refer to the user's manual of the target RL78 microcontroller.
3. This is a required parameter for timing calculation in the flash self-programming library. This setting does not change the operating frequency of the RL78 microcontroller.
4. For the range of the operating frequency, refer to the user's manual of the target RL78 microcontroller.
5. When allocating the FSL_RCD segment on the ROM, always use it in the status check internal mode.

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
LLVM compiler	A(General-purpose register)	-

<R>

[Argument]

Definition of argument

Argument	Description
__far fsl_descriptor_t* descriptor_pstr	Initial setting value of the Flash Self-Programming Library Type 01 (flash memory programming mode, CPU frequency, status check mode)

Definition of __far fsl_descriptor_t*

<R>

Development tool	C language (Structure definition)	Assembly language (Example of definition)

RENESAS CA78K0R compiler	typedef struct {
	fsl_u08 fsl_flash_voltage_u08;
	fsl_u08 fsl_frequency_u08;
	fsl_u08 fsl_auto_status_check_u08;
	} fsl_descriptor_t;
		fsl_descriptor_str:
		DB fsl_flash_voltage_u08
		DB fsl_frequency_u08
		DB fsl_auto_status_check_u08

RENESAS CC-RL compiler	typedef struct {
	fsl_u08 fsl_flash_voltage_u08;
	fsl_u08 fsl_frequency_u08;
	fsl_u08 fsl_auto_status_check_u08;
	} fsl_descriptor_t;
		fsl_descriptor_str:
		.DB fsl_flash_voltage_u08
		.DB fsl_frequency_u08
		.DB fsl_auto_status_check_u08

LLVM compiler	typedef struct {
	fsl_u08 fsl_flash_voltage_u08;
	fsl_u08 fsl_frequency_u08;
	fsl_u08 fsl_auto_status_check_u08;
	} fsl_descriptor_t;
		Check the compiler specifications.

Parameters in __far fsl_descriptor_t*

Argument	Description
fsl_u08 fsl_flash_voltage_u08	Setting of the flash memory programming mode
fsl_u08 fsl_frequency_u08	CPU frequency during the execution of flash self-programming
fsl_u08 fsl_auto_status_check_u08	Setting of the status check mode

Contents of argument settings

Development tool	Argument Type/Register	
	C language	Assembly language
RENESAS CA78K0R compiler	__far fsl_descriptor_t *descriptor_pstr	AX(0-15), C(16-23) The start address of the variable (24 bits)
RENESAS CC-RL compiler	const __far fsl_descriptor_t *descriptor_pstr	DE(0-15), A(16-23) The start address of the variable (24 bits)
<R> LLVM compiler	const __far fsl_descriptor_t *descriptor_pstr	DE(0-15), A(16-23) The start address of the variable (24 bits)

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion - Initial setting is complete.
0x05(FSL_ ERR_PARAMETER)	Parameter error - The frequency value is outside the allowable setting range. - The high-speed on-chip oscillator is not running.

FSL_Open

[Overview]

Start declaration of flash self-programming (starting of the flash environment)

[Format]

<C language>

RENESAS CA78K0R compiler

```
void FSL_Open(void)
```

RENESAS CC-RL compiler

```
void __far FSL_Open(void)
```

<R> LLVM compiler

```
void __far FSL_Open(void) __attribute__ ((section ("FSL_FCD")))
```

<Assembler>

```
CALL !_FSL_Open or CALL !!_FSL_Open
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, the FSL_Init function must be completed normally.

[Function]

Performs start declaration of flash self-programming (starting of the flash environment). Call this function in the beginning of flash self-programming operation.

[Register State After Call]

The registers are not destructed.

[Argument]

None

[Return Value]

None

FSL_Close

[Overview]

End declaration of flash self-programming (ending of the flash environment)

[Format]

<C language>

RENESAS CA78K0R compiler

```
void FSL_Close(void)
```

RENESAS CC-RL compiler

```
void __far FSL_Close(void)
```

<R> LLVM compiler

```
void __far FSL_Close(void) __attribute__ ((section ("FSL_FCD")))
```

<Assembler>

```
CALL !_FSL_Close or CALL !!_FSL_Close
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function must make execution complete.

[Function]

Performs end declaration of flash self-programming (ending of the flash environment). It ends write operation to the code flash memory and returns execution to the normal operation mode.

[Register State After Call]

The registers are not destructed.

[Argument]

None

[Return Value]

None

FSL_PrepareFunctions

[Overview]

Preparation for use of the flash functions (standard rewrite functions) requiring execution in RAM

[Format]

<C language>

RENESAS CA78K0R compiler

```
void FSL_PrepareFunctions( void )
```

RENESAS CC-RL compiler

```
void __far FSL_PrepareFunctions( void )
```

<R> LLVM compiler

```
void __far FSL_PrepareFunctions(void) __attribute__ ((section ("FSL_FCD")))
```

<Assembler>

```
CALL !_FSL_PrepareFunctions or CALL !!_FSL_PrepareFunctions
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function must make execution complete.

[Function]

Prepares the following functions for use.

- FSL_BlankCheck
- FSL_Erase
- FSL_Write
- FSL_IVerify
- FSL_StatusCheck
- FSL_StandBy
- FSL_WakeUp

[Register State After Call]

The registers are not destructed.

[Argument]

None

[Return Value]

None

FSL_PrepareExtFunctions

[Overview]

Preparation for use of the flash functions (extension functions) requiring execution in RAM

[Format]

<C language>

RENESAS CA78K0R compiler

```
void FSL_PrepareExtFunctions(void)
```

RENESAS CC-RL compiler

```
void __far FSL_PrepareExtFunctions(void)
```

<R> LLVM compiler

```
void __far FSL_PrepareExtFunctions(void) __attribute__ ((section ("FSL_FCD")))
```

<Assembler>

```
CALL !_FSL_PrepareExtFunctions or CALL !!_FSL_PrepareExtFunctions
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function must make execution complete.

[Function]

Prepares the following functions for use.

- FSL_SwapBootCluster
- FSL_SwapActiveBootCluster
- FSL_InvertBootFlag
- FSL_SetBlockEraseProtectFlag
- FSL_SetWriteProtectFlag
- FSL_SetBootClusterProtectFlag
- FSL_SetFlashShieldWindow

[Register State After Call]

The registers are not destructed.

[Argument]

None

[Return Value]

None

FSL_ChangeInterruptTable

[Overview]

Changing of all interrupt destinations to the specified addresses on the RAM

[Format]

<C language>

RENESAS CA78K0R compiler

```
void FSL_ChangeInterruptTable(fsl_u16 fsl_interrupt_destination_u16)
```

RENESAS CC-RL compiler

```
void __far FSL_ChangeInterruptTable(fsl_u16 fsl_interrupt_destination_u16)
```

<R> LLVM compiler

```
void __far FSL_ChangeInterruptTable(fsl_u16 fsl_interrupt_destination_u16)
                                             __attribute__ ((section ("FSL_FCD")))
```

<Assembler>

```
CALL !_FSL_ChangeInterruptTable or CALL !!_FSL_ChangeInterruptTable
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

None

[Function]

Changes the destinations of all interrupt functions to the specified addresses on the RAM. After the execution of this function, when an interrupt occurs, execution goes to the address on the RAM specified with this function instead of jumping to the interrupt table.

Cautions 1. The type of the interrupt must be determined by the user by checking the interrupt flag. Because the type of the interrupt must be determined by the user after the execution of this function, the interrupt flag will not be cleared automatically. The user must clear the flag after determining the type of the interrupt.

2. Do not specify a RAM address in the area that has restrictions on usage during the execution of flash self-programming. The flash function may not operate correctly.

3. The interrupt change destination cannot be set to the ROM side (only the address range of FxxxxH can be specified).

4. When the interrupt destination is changed with this function, the interrupt destination remains changed even after flash self-programming until the interrupt destination is restored with the FSL_RestoreInterruptTable() function or a reset is performed.

5. When changing the interrupt destination to the RAM with this function, disable interrupts from the start to end of the processing.

[Register State After Call]

The registers are not destructed.

[Argument]

Definition of argument

Argument	Description
fsl_u16 fsl_interrupt_destination_u16	RAM address of the interrupt destination (lower 16 bits: FxxxxH) *Upper bits are not required.

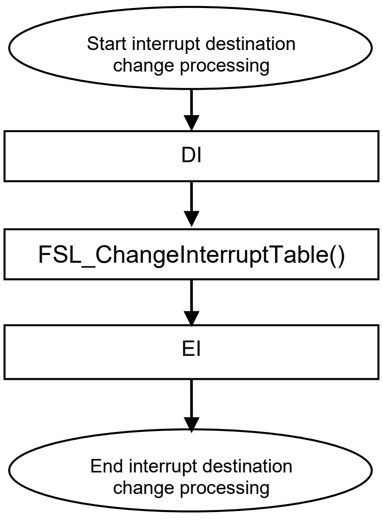
Contents of argument settings

	Development tool	Argument Type/Register	
		C language	Assembly language
	RENESAS CA78K0R compiler	fsl_u16 fsl_interrupt_destination_u16	AX(0-15): RAM address (lower 16 bits)
	RENESAS CC-RL compiler	fsl_u16 fsl_interrupt_destination_u16	AX(0-15): RAM address (lower 16 bits)
<R>	LLVM compiler	fsl_u16 fsl_interrupt_destination_u16	AX(0-15): RAM address (lower 16 bits)

[Return Value]

None

[Operation Example]


Start interrupt destination change processing
DI
FSL_ChangeInterruptTable()
EI
End interrupt destination change processing

FSL_RestoreInterruptTable

[Overview]

Restoration of the interrupt destination changed to the RAM to the standard interrupt vector table

[Format]

<C language>

RENESAS CA78K0R compiler

```
void FSL_RestoreInterruptTable( void )
```

RENESAS CC-RL compiler

```
void __far FSL_RestoreInterruptTable( void )
```

<R> LLVM compiler

```
void __far FSL_RestoreInterruptTable(void) __attribute__ ((section ("FSL_FCD")))
```

<Assembler>

```
CALL !_FSL_RestoreInterruptTable or CALL !!_FSL_RestoreInterruptTable
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

None

[Function]

Restores the interrupt destination changed to the RAM to the standard interrupt vector table.

Cautions 1. If the interrupt destination is changed with the FSL_ChangeInterruptTable() function, the interrupt destination remains changed even after flash self-programming until a reset is performed unless the interrupt destination is restored with this function.

2. When changing the interrupt destination to the standard interrupt vector with this function, disable interrupts from the start to end of the processing.

[Register State After Call]

The registers are not destructed.

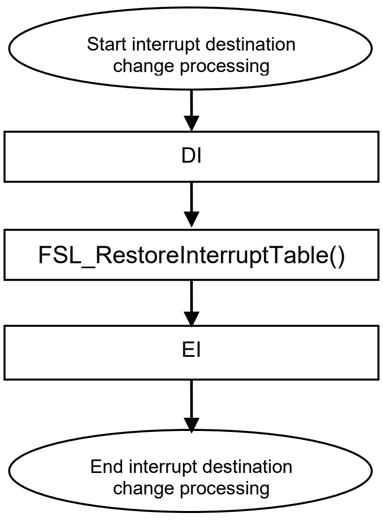
[Argument]

None

[Return Value]

None

[Operation Example]


Start interrupt destination change processing
DI
FSL_RestoreInterruptTable()
EI
End interrupt destination change processing

FSL_BlankCheck

[Overview]

Blank checking of the specified block

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_BlankCheck(fsl_u16 block_u16)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_BlankCheck(fsl_u16 block_u16)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_BlankCheck(fsl_u16 block_u16)
                                        __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_BlankCheck or CALL !!_FSL_BlankCheck
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareFunctions function must make execution complete. Also, when an interrupt must be received before the processing is completed, use the FSL_ChangeInterruptTable function to change the interrupt destination to the RAM.

[Function]

Checks that the code flash memory of the specified block is in the erasure level.

(The erasure level check cannot be done in checking of 0xFF with data read.)

In case of an error, execute the FSL_Erase function.

If the execution of the FSL_Erase function is completed normally, no blank checking is required.

If the specified block number does not exist, a parameter error (0x05) is returned.

Caution If both (1) and (2) below are satisfied, this function can be allocated on the internal ROM for use.

(1) The status check mode is set to the status check internal mode with the FSL_Init function.

(2) "Do not use interrupts" or "disable interrupts on the internal ROM" until the processing of this function is completed (the reception of interrupts on the RAM is permitted).

Remarks 1. The FSL_BlankCheck function checks if the cell of the code flash memory satisfies the erasure level with a sufficient margin. A blank check error does not indicate any problem in the code flash memory, but perform erasure processing before writing after the blank check error.

2. A blank check is performed only for one block. To perform blank checking of multiple blocks, call this function multiple times.

[Register State After Call]

	Development tool	Return value	Destructed register
	RENESAS CA78K0R compiler	C(General-purpose register)	-
	RENESAS CC-RL compiler	A(General-purpose register)	-
<R>	LLVM compiler	A(General-purpose register)	-

[Argument]

Definition of argument

Argument	Description
block_u16	Block number of the block to be blank-checked

Contents of argument settings

	Development tool	Argument Type/Register	
		C language	Assembly language
	RENESAS CA78K0R compiler	fsl_u16 block_u16	AX(0-15): Block number (16 bits)
	RENESAS CC-RL compiler	fsl_u16 block_u16	AX(0-15): Block number (16 bits)
<R>	LLVM compiler	fsl_u16 block_u16	AX(0-15): Block number (16 bits)

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion[Note1] - The specified block is in the blank state.
0x05(FSL_ERR_PARAMETER)	Parameter error - The specification of the block number is outside the allowable setting range.
0x1B(FSL_ERR_BLANKCHECK)	Blank check error[Note1] - The specified block is not in the blank state.
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.[Note2] - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.[Note2]
0xFF(FSL_BUSY)	Execution start of this function[Note2] - The execution of this function has been started. (Check the execution state with the FSL_StatusCheck function.)

Notes 1. Only in the status check internal mode.
2. Only in the status check user mode.

FSL_Erase

[Overview]

Erasure of the specified block

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_Erase(fsl_u16 block_u16)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_Erase(fsl_u16 block_u16)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_Erase(fsl_u16 block_u16) __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_Erase or CALL !!_FSL_Erase
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareFunctions function must make execution complete. Also, when an interrupt must be received before the processing is completed, use the FSL_ChangeInterruptTable function to change the interrupt destination to the RAM.

[Function]

Erases (0xFF) the content of the code flash memory in the specified block.

Caution If both (1) and (2) below are satisfied, this function can be allocated on the internal ROM for use.
(1) The status check mode is set to the status check internal mode with the FSL_Init function.
(2) "Do not use interrupts" or "disable interrupts on the internal ROM" until the processing of this function is completed (the reception of interrupts on the RAM is permitted).

Remark Erasure is performed only for one block. To erase multiple blocks, call this function multiple times.

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
<R> LLVM compiler	A(General-purpose register)	-

[Argument]

Definition of argument

Argument	Description
block_u16	Block number of the block to be erased

Contents of argument settings

Development tool	Argument Type/Register	
	C language	Assembly language
RENESAS CA78K0R compiler	fsl_u16 block_u16	AX(0-15): Block number (16 bits)
RENESAS CC-RL compiler	fsl_u16 block_u16	AX(0-15): Block number (16 bits)
<R> LLVM compiler	fsl_u16 block_u16	AX(0-15): Block number (16 bits)

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion[Note1]
0x05(FSL_ERR_PARAMETER)	Parameter error - The specification of the block number is outside the allowable setting range.
0x10(FSL_ERR_PROTECTION)	Protect error - The specified block is included in the boot area, and the boot area rewrite permission flag is set to protected. - The specified block is outside the FSW setting area.
0x1A(FSL_ERR_ERASE)	Erasure error[Note1] - An error occurred during erasure processing.
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.[Note2] - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.[Note2]
0xFF(FSL_BUSY)	Execution start of this function[Note2] - The execution of this function has been started. (Check the execution state with the FSL_StatusCheck function.)

Notes 1. Only in the status check internal mode.
2. Only in the status check user mode.

FSL_IVerify

[Overview]

Verification (internal verification) of the specified block

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_IVerify(fsl_u16 block_u16)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_IVerify(fsl_u16 block_u16)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_IVerify(fsl_u16 block_u16) __attribute__ ((section ("FSL_RCD")))
```

Assembler>

```
CALL !_FSL_IVerify or CALL !!_FSL_IVerify
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareFunctions function must make execution complete. Also, when an interrupt must be received before the processing is completed, use the FSL_ChangeInterruptTable function to change the interrupt destination to the RAM.

[Function]

Performs verification to check the write level in the specified block.

The verification checks if the data written to the code flash memory of the specified block is in the erasure level (data "1") or write level (data "0").

In case of an error, execute the FSL_Erase function, and then perform writing with FSL_Write again.

If the specified block number does not exist, a parameter error (0x05) is returned.

Cautions 1. After data is written, if no verification (internal verification) is done for the block including the range to which data is written, the written data is not guaranteed.

2. When data erasure, data write, and internal verification are performed and completed normally after an internal verification error, the device is determined as normal.

3. If both (1) and (2) below are satisfied, this function can be allocated on the internal ROM for use.

(1) The status check mode is set to the status check internal mode with the FSL_Init function.

(2) "Do not use interrupts" or "disable interrupts on the internal ROM" until the processing of this function is completed (the reception of interrupts on the RAM is permitted).

Remark Verification is performed only for one block. To perform verification of multiple blocks, call this function multiple times.

[Register State After Call]

	Development tool	Return value	Destructed register
	RENESAS CA78K0R compiler	C(General-purpose register)	-
	RENESAS CC-RL compiler	A(General-purpose register)	-
<R>	LLVM compiler	A(General-purpose register)	-

[Argument]

Definition of argument

Argument	Description
block_u16	Block number to be verified

Contents of argument settings

	Development tool	Argument Type/Register	
		C language	Assembly language
	RENESAS CA78K0R compiler	fsl_u16 block_u16	AX(0-15): Block number (16 bits)
	RENESAS CC-RL compiler	fsl_u16 block_u16	AX(0-15): Block number (16 bits)
<R>	LLVM compiler	fsl_u16 block_u16	AX(0-15): Block number (16 bits)

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion[Note1]
0x05(FSL_ERR_PARAMETER)	Parameter error - The specification of the block number is outside the allowable setting range.
0x1B(FSL_ERR_IVERIFY)	Verification (internal verification) error[Note1] - An error occurred during verification (internal verification) processing.
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.[Note2] - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.[Note2]
0xFF(FSL_BUSY)	Execution start of this function[Note2] - The execution of this function has been started. (Check the execution state with the FSL_StatusCheck function.)

Notes 1. Only in the status check internal mode.
2. Only in the status check user mode.

FSL_Write

[Overview]

Writing of 1 word to 64 words data into the specified address (1 word = 4 bytes)

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_Write (__near fsl_write_t* write_pstr)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_Write (__near fsl_write_t* write_pstr)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_Write(__near fsl_write_t* write_pstr)
                                        __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_Write or CALL !!_FSL_Write
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

- Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareFunctions function must make execution complete. Also, when an interrupt must be received before the processing is completed, use the FSL_ChangeInterruptTable function to change the interrupt destination to the RAM.
- Before calling this function, save the data to be written to the code flash memory in the data buffer.

[Function]

Writes to the code flash memory at the specified address.

After writing to a block, always execute FSL_IVerify for the block.

Execute the FSL_Write function only to an erased block.

Up to 256 bytes (in units of 4 bytes) of data can be written at once.

In the following cases (the specified word count or address is outside of the allowable setting range), a parameter error (0x05) is returned.

Word count check

- 0 words
- 65 words or more

Address check

- Not in units of 4 bytes from the start address
- The write end address exceeds the final address of the code flash memory.

Notes 1. After writing data, execute verification (internal verification) of the block including the range to which data is written. Otherwise, the written data is not guaranteed.

2. If both (1) and (2) below are satisfied, this function can be allocated on the internal ROM for use.

(1) The status check mode is set to the status check internal mode with the FSL_Init function.

(2) "Do not use interrupts" or "disable interrupts on the internal ROM" until the processing of this function is completed (the reception of interrupts on the RAM is permitted).

Remark To write data larger than 256 bytes, call this function multiple times.

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
<R> LLVM compiler	A(General-purpose register)	-

[Argument]

Definition of argument

Argument	Description
__near fsl_write_t* write_pstr	Write data storage settings (Data buffer address, write destination address, and write size)

Definition of fsl_write_t

<R>

Development tool	C language (Structure definition)	Assembly language (Example of definition)

RENESAS CA78K0R compiler	typedef struct {
	fsl_u08 __near *fsl_data_buffer_p_u08;
	fsl_u32 fsl_destination_address_u32;
	fsl_u08 fsl_word_count_u08;
	} fsl_write_t;
		fsl_write_str :
		fsl_data_buffer_p_u08: DS 2
		fsl_destination_address_u32: DS 2
		fsl_word_count_u08: DS 1

RENESAS CC-RL compiler	typedef struct {
	fsl_u08 __near *fsl_data_buffer_p_u08;
	fsl_u32 fsl_destination_address_u32;
	fsl_u08 fsl_word_count_u08;
	} fsl_write_t;
		fsl_write_str :
		fsl_data_buffer_p_u08: .DS 2
		fsl_destination_address_u32: .DS 2
		fsl_word_count_u08 : .DS 1

LLVM compiler	typedef struct {
	fsl_u08 __near *fsl_data_buffer_p_u08;
	fsl_u32 fsl_destination_address_u32;
	fsl_u08 fsl_word_count_u08;
	} fsl_write_t;
		Check the compiler specifications.

Contents of fsl_write_t

Argument	Description
fsl_u08 __near *fsl_data_buffer_p_u08	Start address of the buffer area where data to write is input (16 bits)
fsl_u32 fsl_destination_address_u32	Start address of the destination (32 bits)
fsl_u08 fsl_word_count_u08	Data count to write (1 to 64: in words)

Contents of argument settings

Development tool	Argument Type/Register	
	C language	Assembly language
RENESAS CA78K0R compiler	__near fsl_write_t* write_pstr	AX(0-15): The start address of the variable (16 bits)
RENESAS CC-RL compiler	__near fsl_write_t* write_pstr	AX(0-15): The start address of the variable (16 bits)
<R> LLVM compiler	__near fsl_write_t* write_pstr	AX(0-15): The start address of the variable (16 bits)

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion[Note1]
0x05(FSL_ERR_PARAMETER)	Parameter error - The start address is not a multiple of 1 word (4 bytes). - The written data count is 0. - The written data count exceeds 64 words. - The write end address (start address + (written data count × 4 bytes)) exceeds the code flash memory area.
0x10(FSL_ERR_PROTECTION)	Protect error - The specified range includes the boot area, and the boot area rewrite permission flag is set to protected. - The specified block is outside the FSW setting area.
0x1C(FSL_ERR_WRITE)	Writing error[Note1] - An error occurred during write processing.
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.[Note2] - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.[Note2]
0xFF(FSL_BUSY)	Execution start of this function[Note2] - The execution of this function has been started. (Check the execution state with the FSL_StatusCheck function.)

Notes 1. Only in the status check internal mode.
2. Only in the status check user mode.

FSL_GetSecurityFlags

[Overview]

Acquisition of security information

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_GetSecurityFlags(fsl_u08 __near *data_destination_pu08)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_GetSecurityFlags(fsl_u08 __near *data_destination_pu08)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_GetSecurityFlags(fsl_u08 __near *data_destination_pu08)
                                        __attribute__ ((section ("FSL_FECD")))
```

<Assembler>

```
CALL !_FSL_GetSecurityFlags or CALL !!_FSL_GetSecurityFlags
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function must make execution complete.

[Function]

Obtains the security flag information and inputs the value to the data storage buffer specified in the argument.

[Register State After Call]

	Development tool	Return value	Destructed register
	RENESAS CA78K0R compiler	C(General-purpose register)	-
	RENESAS CC-RL compiler	A(General-purpose register)	-
<R>	LLVM compiler	A(General-purpose register)	-

[Argument]

Definition of argument

Argument	Description
fsl_u08 __near *data_destination_pu08	Data storage buffer - Reserve a 1-byte data buffer.

Contents of argument settings

Development tool	Argument Type/Register	
	C language	Assembly language
RENESAS CA78K0R compiler	__near *data_destination_pu08	AX(0-15): The start address of the data buffer (16 bits)
RENESAS CC-RL compiler	__near *data_destination_pu08	AX(0-15): The start address of the data buffer (16 bits)
<R> LLVM compiler	__near *data_destination_pu08	AX(0-15): The start address of the data buffer (16 bits)

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.Note - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.Note

Note Only in the status check user mode.

Security bit information

The security bit information is written to the data storage buffer (data_destination_pu08) passed in the argument.

data_destination_pu08	Description
bit 1: 0b000000X0	Boot area rewrite protection flag (0: Protected, 1: Permitted)
bit 2: 0b00000X00	Block erasure protection flag (0: Protected, 1: Permitted)
bit 4: 0b000X0000	Write protection flag (0: Protected, 1: Permitted)
Other bits	1

FSL_GetBootFlag

[Overview]

Acquisition of boot flag information

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_GetBootFlag(fsl_u08 __near *data_destination_pu08)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_GetBootFlag(fsl_u08 __near *data_destination_pu08)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_GetBootFlag(fsl_u08 __near *data_destination_pu08)
                                          __attribute__ ((section ("FSL_FECD")))
```

<Assembler>

```
CALL !_FSL_GetBootFlag or CALL !!_FSL_GetBootFlag
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function must make execution complete.

[Function]

Obtains the boot cluster flag information and inputs the value to the data storage buffer specified in the argument.

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
<R> LLVM compiler	A(General-purpose register)	-

[Argument]

Definition of argument

Argument	Description
fsl_u08 __near *data_destination_pu08	Data storage buffer - Reserve a 1-byte data buffer.

Contents of argument settings

Development tool	Argument Type/Register	
	C language	Assembly language
RENESAS CA78K0R compiler	__near *data_destination_pu08	AX(0-15): The start address of the data buffer (16 bits)
RENESAS CC-RL compiler	__near *data_destination_pu08	AX(0-15): The start address of the data buffer (16 bits)
<R> LLVM compiler	__near *data_destination_pu08	AX(0-15): The start address of the data buffer (16 bits)

[Return Value]

[

State	Description
0x00(FSL_OK)	Normal completion
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.[Note] - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.[Note]

Note Only in the status check user mode.

Boot flag information

The boot flag information is written to the data storage buffer (data_destination_pu08) passed in the argument.

data_destination_pu08	Description
0x00	Starts up with Boot Cluster 0 as the boot area (from 0000H) after a reset.
0x01	Starts up with Boot Cluster 1 as the boot area (from 0000H) after a reset.

Remark For the swap state of the boot area before the reset, refer to the section on the FSL_GetSwapState function.

Example RL78/G13: The boot area size (one cluster) is 4KB.
RL78/F13: The boot area size (one cluster) is 8KB.

FSL_GetSwapState

[Overview]

Acquisition of the swap state

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_GetSwapState(fsl_u08 __near *data_destination_pu08)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_GetSwapState(fsl_u08 __near *data_destination_pu08)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_GetSwapState(fsl_u08 __near *data_destination_pu08)
                                          __attribute__ ((section ("FSL_FECD")))
```

<Assembler>

```
CALL !_FSL_GetSwapState or CALL !!_FSL_GetSwapState
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function must make execution complete.

[Function]

Obtains the current boot cluster swap state and inputs the value to the data storage buffer specified in the argument.

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
<R> LLVM compiler	A(General-purpose register)	-

[Argument]

Definition of argument

Argument	Description
fsl_u08 __near *data_destination_pu08	Data storage buffer - Reserve a 1-byte data buffer.

Contents of argument settings

Development tool	Argument Type/Register	
	C language	Assembly language
RENESAS CA78K0R compiler	__near *data_destination_pu08	AX(0-15): The start address of the data buffer (16 bits)
RENESAS CC-RL compiler	__near *data_destination_pu08	AX(0-15): The start address of the data buffer (16 bits)
<R> LLVM compiler	__near *data_destination_pu08	AX(0-15): The start address of the data buffer (16 bits)

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.Note - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.Note

Note Only in the status check user mode.

Boot swap status

The boot swap information is written to the data storage buffer (data_destination_pu08) passed in the argument.

data_destination_pu08	Description
0x00	The current boot area (The area from 0000H) is Boot Cluster 0.
0x01	The current boot area (The area from 0000H) is Boot Cluster 1.

Remark For the status of the boot area after the reset, refer to the section on the FSL_GetBootFlag function.

Example RL78/G13: The boot area size (one cluster) is 4KB.

RL78/F13: The boot area size (one cluster) is 8KB.

FSL_GetBlockEndAddr

[Overview]

Acquisition of the final address of the specified block

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_GetBlockEndAddr(__near fsl_getblockendaddr_t* getblockendaddr_pstr)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_GetBlockEndAddr(__near fsl_getblockendaddr_t*
                                                     getblockendaddr_pstr)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_GetBlockEndAddr(__near fsl_getblockendaddr_t*
             getblockendaddr_pstr) __attribute__ ((section ("FSL_FECD")));
```

<Assembler>

```
CALL !_FSL_GetBlockEndAddr or CALL !!_FSL_GetBlockEndAddr
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function must make execution complete.

[Function]

Obtains the final address of the block specified in the argument and inputs the value to the data storage buffer.

[Register State After Call]

	Development tool	Return value	Destructed register
	RENESAS CA78K0R compiler	C(General-purpose register)	-
	RENESAS CC-RL compiler	A(General-purpose register)	-
<R>	LLVM compiler	A(General-purpose register)	-

[Argument]

Definition of argument

Argument	Description
__near fsl_getblockendaddr_t* getblockendaddr_pstr	The pointer to the structure which obtains the end address of the specified block.

Definition of fsl_getblockendaddr_t

<R>

Development tool	C language (Structure definition)	Assembly language (Example of definition)

RENESAS CA78K0R compiler	typedef struct {
	fsl_u32 fsl_destination_address_u32;
	fsl_u16 fsl_block_u16
	} fsl_getblockendaddr_t;
		fsl_getblockendaddr_str:
		fsl_destination_address_u32: DS 4
		fsl_block_u16: DS 2

RENESAS CC-RL compiler	typedef struct {
	fsl_u32 fsl_destination_address_u32;
	fsl_u16 fsl_block_u16
	} fsl_getblockendaddr_t;
		fsl_getblockendaddr_str:
		fsl_destination_address_u32: .DS 4
		fsl_block_u16: .DS 2

LLVM compiler	typedef struct {
	fsl_u32 fsl_destination_address_u32;
	fsl_u16 fsl_block_u16
	} fsl_getblockendaddr_t;
		Check the compiler specifications.

Contents of fsl_getblockendaddr_t

Argument	Description
fsl_u32 fsl_destination_address_u32	End address storage buffer :Output - Reserve a 4-byte data buffer.
fsl_u16 fsl_block_u16	Block number :Input

Contents of argument settings

Development tool	Argument Type/Register	
	C language	Assembly language
RENESAS CA78K0R compiler	__near fsl_getblockendaddr_t* getblockendaddr_pstr	AX(0-15): The start address of the variable (16 bits)
RENESAS CC-RL compiler	__near fsl_getblockendaddr_t* getblockendaddr_pstr	AX(0-15): The start address of the variable (16 bits)
<R> LLVM compiler	__near fsl_getblockendaddr_t* getblockendaddr_pstr	AX(0-15): The start address of the variable (16 bits)

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion
0x05(FSL_ ERR_PARAMETER)	Parameter error - The specification of the block number is outside the allowable setting range.
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.[Note] - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.[Note]

Note Only in the status check user mode.

FSL_GetFlashShieldWindow

[Overview]

Acquisition of the start block number and end block number of the flash shield window

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_GetFlashShieldWindow(__near fsl_fsw_t* getfsw_pstr);
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_GetFlashShieldWindow(__near fsl_fsw_t* getfsw_pstr);
```

<R> LLVM compiler

```
fsl_u08 __far FSL_GetFlashShieldWindow(__near fsl_fsw_t* getfsw_pstr)
                                         __attribute__ ((section ("FSL_FECD")))
```

<Assembler>

```
CALL !_FSL_GetFlashShieldWindow or CALL !!_FSL_GetFlashShieldWindow
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function must make execution complete.

[Function]

Obtains the start block and end block of the flash shield window and inputs the values to the data storage buffers fsl_start_block_u16 (start block) and fsl_end_block_u16 (end block) specified in the arguments, respectively.

[Register State After Call]

	Development tool	Return value	Destructed register
	RENESAS CA78K0R compiler	C(General-purpose register)	-
	RENESAS CC-RL compiler	A(General-purpose register)	-
<R>	LLVM compiler	A(General-purpose register)	-

[Argument]

Definition of argument

Argument	Description
__near fsl_fsw_t* getfsw_pstr	Variable for obtaining FSW settings (FSW start block number and end block number)

Definition of fsl_fsw_t

<R>

Development tool	C language (Structure definition)	Assembly language (Example of definition)

RENESAS CA78K0R compiler	typedef struct {
	  fsl_u16 fsl_start_block_u16;
	  fsl_u16 fsl_end_block_u16;
	} fsl_fsw_t;
		getfsw_pstr:
		fsl_start_block_u16: DS 2
		fsl_end_block_u16 : DS 2

RENESAS CC-RL compiler	typedef struct {
	  fsl_u16 fsl_start_block_u16;
	  fsl_u16 fsl_end_block_u16;
	} fsl_fsw_t
		getfsw_pstr:
		fsl_start_block_u16: .DS 2
		fsl_end_block_u16 : .DS 2

LLVM compiler	typedef struct {
	  fsl_u16 fsl_start_block_u16;
	  fsl_u16 fsl_end_block_u16;
	} fsl_fsw_t
		Check the compiler specifications.

Contents of __near fsl_fsw_t

Argument	Description
fsl_u16 fsl_start_block_u16;	FSW start block storage buffer - Reserve a 2-byte data buffer.
fsl_u16 fsl_end_block_u16;	FSW end block storage buffer - Reserve a 2-byte data buffer.

Contents of argument settings

Development tool	Argument Type/Register	
	C language	Assembly language
RENESAS CA78K0R compiler	__near fsl_fsw_t* getfsw_pstr	AX(0-15): The start address of the variable (16 bits)
RENESAS CC-RL compiler	__near fsl_fsw_t* getfsw_pstr	AX(0-15): The start address of the variable (16 bits)
<R> LLVM compiler	__near fsl_fsw_t* getfsw_pstr	AX(0-15): The start address of the variable (16 bits)

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.Note - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.Note

Note Only in the status check user mode.

FSL_SwapBootCluster

[Overview]

Execution of boot swapping and jumping to the address registered in the reset vector in the swapped area

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_SwapBootCluster(void)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_SwapBootCluster(void)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_SwapBootCluster(void) __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_SwapBootCluster or CALL !!_FSL_SwapBootCluster
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareExtFunctions function must make execution complete. Also, when an interrupt must be received before the processing is completed, use the FSL_ChangeInterruptTable function to change the interrupt destination to the RAM.

[Function]

Disables interrupts (DI) and performs swapping of the boot clusters immediately after the execution of the function. Execution moves to the address registered to the reset vector in the swapped area (unlike the reset function of the RL78 microcontroller, program execution is started only from the reset vector address).

Cautions 1. Do not execute this function in an RL78 microcontroller that does not support boot swapping.
2. Before the execution of swapping, always write the setting information required for operation after swapping such as the option byte setting to the swap destination area.
3. When this function is executed normally, the code written after this function is not executed because execution moves to the address registered in the reset vector in the swapped boot cluster.
4. This function does not invert the boot flag. When a reset is performed, the boot cluster enters the state according to the boot flag setting.
5. When the FSL_ChangeInterruptTable function is executed with the interrupt destination changed, interrupt processing will go to the area changed by the FSL_ChangeInterruptTable function even after moving to the address registered in the reset vector. To move to the address registered in the restored reset vector, execute the FSL_RestoreInterruptTable function to restore the interrupt destination before executing this function.

6. If both (1) and (2) below are satisfied, this function can be allocated on the internal ROM for use.
   (1) The status check mode is set to the status check internal mode with the FSL_Init function.
   (2) "Do not use interrupts" or "disable interrupts on the internal ROM" until the processing of this function is completed (the reception of interrupts on the RAM is permitted).
7. The FSL_SwapBootCluster function execute exchange for the boot cluster 0 and the boot cluster 1. Do not locate the user program, data, and the flash self-programming library required for programming on a boot cluster.

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
<R> LLVM compiler	A(General-purpose register) -	

[Argument]
None

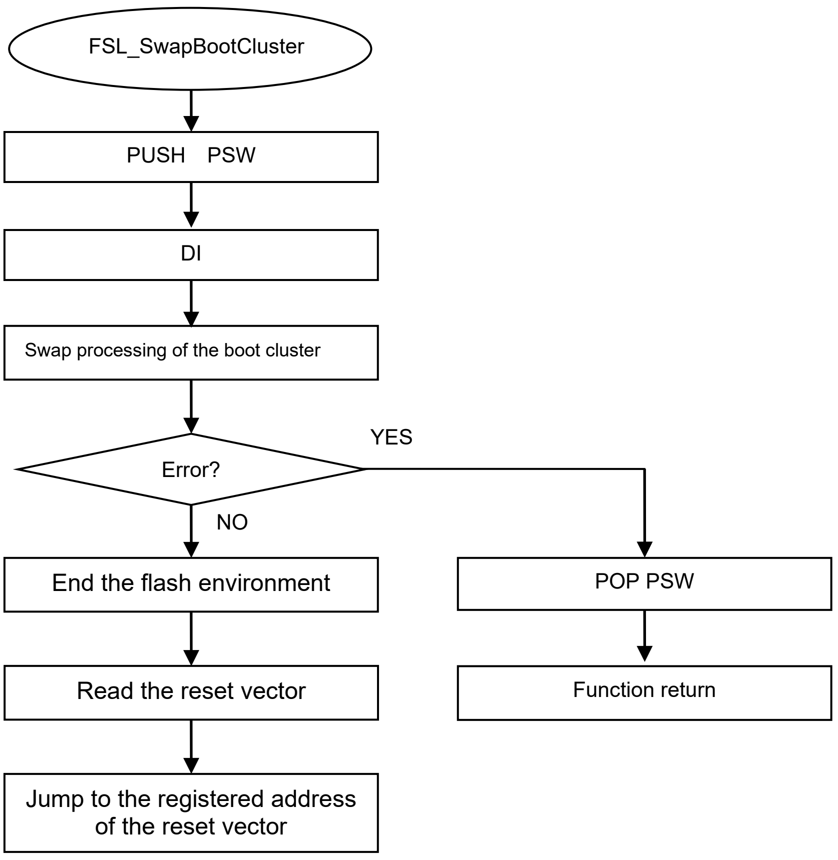
[Return Value]

State	Description
0x10(FSL_ERR_PROTECTION)	Protect error - Boot swapping was attempted in the boot area rewrite-protected state.
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.Note - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.Note

Note Only in the status check user mode.

Remark For normal completion, the return value cannot be checked.

[Flow]


FSL_SwapBootCluster
PUSH PSW
DI
Swap processing of the boot cluster
Error?
YES
NO
End the flash environment
Read the reset vector
Jump to the registered address of the reset vector
POP PSW
Function return

FSL_SwapActiveBootCluster

[Overview]

Inverting of the current value of the boot flag and execution of boot swapping

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_SwapActiveBootCluster(void)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_SwapActiveBootCluster(void)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_SwapActiveBootCluster(void) __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_SwapActiveBootCluster or CALL !!_FSL_SwapActiveBootCluster
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareExtFunctions function must make execution complete. Also, when an interrupt must be received before the processing is completed, use the FSL_ChangeInterruptTable function to change the interrupt destination to the RAM.

[Function]

When this function is executed, the current value of the boot flag is inverted, and boot clusters are swapped.

Cautions 1. Do not execute this function in an RL78 microcontroller that does not support boot swapping.

2. Before the execution of swapping, always write the setting information required for operation after swapping such as the option byte setting to the swap destination area.
3. The boot clusters are swapped without a reset. Do not allocate the user program, data, or flash self-programming library required for rewriting in the boot cluster. If it is required to refer to the program or data in the boot cluster after the execution of this function, use it with considering that the boot clusters are swapped.
4. This function cannot be executed from the ROM. To use this function, allocate the FSL_RCD segment on the RAM.
5. After the execution of this function, the interrupt vector on the ROM is also changed. To use interrupt processing on the ROM before and after the execution, use it with considering that the interrupt vector on the ROM switches during operation.
6. When this function is used, the functions contained in the FSL_RCD segment cannot be allocated on the ROM for use.

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
<R> LLVM compiler	A(General-purpose register)	-

[Argument]

None

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion[Note1]
0x10(FSL_ERR_PROTECTION)	Protect error - Changing of the flag from protected to permitted was attempted. - Changing of the boot area switching flag was attempted in the boot area rewrite-protected state.
0x1A(FSL_ERR_ERASE)	Erasure error[Note1] - An error occurred during erasure processing.
0x1B(FSL_ERR_IVERIFY)	Internal verification error[Note1] - An error occurred during verification (internal verification) processing.
0x1C(FSL_ERR_WRITE)	Writing error[Note1] - An error occurred during write processing.
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.[Note2] - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.[Note2]
0xFF(FSL_BUSY)	Execution start of this function[Note2] - The execution of this function has been started. (Check the execution state with the FSL_StatusCheck function.)

Notes 1. Only in the status check internal mode.
2. Only in the status check user mode.

FSL_InvertBootFlag

[Overview]

Inverting of the current value of the boot flag

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_InvertBootFlag(void)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_InvertBootFlag(void)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_InvertBootFlag(void) __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_InvertBootFlag or CALL !!_FSL_InvertBootFlag
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareExtFunctions function must make execution complete. Also, when an interrupt must be received before the processing is completed, use the FSL_ChangeInterruptTable function to change the interrupt destination to the RAM.

[Function]

Inverts the current value of the boot flag. After a reset, the boot cluster enters the state according to the boot flag setting.

Cautions 1. Do not execute this function in an RL78 microcontroller that does not support boot swapping.

2. The boot cluster is not inverted upon execution of this function.

3. If both (1) and (2) below are satisfied, this function can be allocated on the internal ROM for use.

(1) The status check mode is set to the status check internal mode with the FSL_Init function.

(2) "Do not use interrupts" or "disable interrupts on the internal ROM" until the processing of this function is completed (the reception of interrupts on the RAM is permitted).

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
LLVM compiler	A(General-purpose register)	-

[Argument]

None

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion[Note1]
0x10(FSL_ERR_PROTECTION)	Protect error - Changing of the flag from protected to permitted was attempted. - Changing of the boot area switching flag was attempted in the boot area rewrite-protected state.
0x1A(FSL_ERR_ERASE)	Erasure error[Note1] - An error occurred during erasure processing.
0x1B(FSL_ERR_IVERIFY)	Internal verification error[Note1] - An error occurred during verification (internal verification) processing.
0x1C(FSL_ERR_WRITE)	Writing error[Note1] - An error occurred during write processing.
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.[Note2] - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.[Note2]
0xFF(FSL_BUSY)	Execution start of this function[Note2] - The execution of this function has been started. (Check the execution state with the FSL_StatusCheck function.)

Notes 1. Only in the status check internal mode.
2. Only in the status check user mode.

FSL_SetBlockEraseProtectFlag

[Overview]

Setting of the block erasure protection flag to protected

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_SetBlockEraseProtectFlag(void)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_SetBlockEraseProtectFlag(void)
```

<R> LLVM compiler

```
Fsl_u08 __far FSL_SetBlockEraseProtectFlag(void)
                                            __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_SetBlockEraseProtectFlag or CALL !!_FSL_SetBlockEraseProtectFlag
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareExtFunctions function must make execution complete. Also, when an interrupt must be received before the processing is completed, use the FSL_ChangeInterruptTable function to change the interrupt destination to the RAM.

[Function]

Sets the block erasure protection flag to protected. Block erasure for the device by the programmer cannot be done.

Caution If both (1) and (2) below are satisfied, this function can be allocated on the internal ROM for use.
(1) The status check mode is set to the status check internal mode with the FSL_Init function.
(2) "Do not use interrupts" or "disable interrupts on the internal ROM" until the processing of this function is completed (the reception of interrupts on the RAM is permitted).

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
<R> LLVM compiler	A(General-purpose register)	-

[Argument]

None

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion[Note1]
0x1A(FSL_ERR_ERASE)	Erasure error[Note1] - An error occurred during erasure processing.
0x1B(FSL_ERR_IVERIFY)	Internal verification error[Note1] - An error occurred during verification (internal verification) processing.
0x1C(FSL_ERR_WRITE)	Writing error[Note1] - An error occurred during write processing.
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.[Note2] - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.[Note2]
0xFF(FSL_BUSY)	Execution start of this function[Note2] - The execution of this function has been started. (Check the execution state with the FSL_StatusCheck function.)

Notes 1. Only in the status check internal mode.
2. Only in the status check user mode.

FSL_SetWriteProtectFlag

[Overview]

Setting of the write protection flag to protected

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_SetWriteProtectFlag(void)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_SetWriteProtectFlag(void)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_SetWriteProtectFlag(void) __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_SetWriteProtectFlag or CALL !!_FSL_SetWriteProtectFlag
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareExtFunctions function must make execution complete. Also, when an interrupt must be received before the processing is completed, use the FSL_ChangeInterruptTable function to change the interrupt destination to the RAM.

[Function]

Sets the write protection flag to protected. When it is set to protected, writing to the device by the programmer cannot be done.

Caution If both (1) and (2) below are satisfied, this function can be allocated on the internal ROM for use.
(1) The status check mode is set to the status check internal mode with the FSL_Init function.
(2) "Do not use interrupts" or "disable interrupts on the internal ROM" until the processing of this function is completed (the reception of interrupts on the RAM is permitted).

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
<R> LLVM compiler	A(General-purpose register)	-

[Argument]

None

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion Note1
0x1A(FSL_ERR_ERASE)	Erasure error Note1 - An error occurred during erasure processing.
0x1B(FSL_ERR_IVERIFY)	Internal verification error Note1 - An error occurred during verification (internal verification) processing.
0x1C(FSL_ERR_WRITE)	Writing error Note1 - An error occurred during write processing.
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed. Note2 - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state. Note2
0xFF(FSL_BUSY)	Execution start of this function Note2 - The execution of this function has been started. (Check the execution state with the FSL_StatusCheck function.)

Notes 1. Only in the status check internal mode.
2. Only in the status check user mode.

FSL_SetBootClusterProtectFlag

[Overview]

Setting of the boot area rewrite protection flag to protected

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_SetBootClusterProtectFlag(void)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_SetBootClusterProtectFlag(void)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_SetBootClusterProtectFlag(void)
                                                   __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_SetBootClusterProtectFlag or CALL !!_FSL_SetBootClusterProtectFlag
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareExtFunctions function must make execution complete. Also, when an interrupt must be received before the processing is completed, use the FSL_ChangeInterruptTable function to change the interrupt destination to the RAM.

[Function]

Sets the boot area rewrite protection flag to protected. When it is set to protected, swapping, erasure, and writing to the boot cluster cannot be done.

Caution If both (1) and (2) below are satisfied, this function can be allocated on the internal ROM for use.
(1) The status check mode is set to the status check internal mode with the FSL_Init function.
(2) "Do not use interrupts" or "disable interrupts on the internal ROM" until the processing of this function is completed (the reception of interrupts on the RAM is permitted).

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
<R> LLVM compiler	A(General-purpose register)	-

[Argument]

None

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion[Note1]
0x1A(FSL_ERR_ERASE)	Erasure error[Note1] - An error occurred during erasure processing.
0x1B(FSL_ERR_IVERIFY)	Internal verification error[Note1] - An error occurred during verification (internal verification) processing.
0x1C(FSL_ERR_WRITE)	Writing error[Note1] - An error occurred during write processing.
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.[Note2] - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.[Note2]
0xFF(FSL_BUSY)	Execution start of this function[Note2] - The execution of this function has been started. (Check the execution state with the FSL_StatusCheck function.)

Notes 1. Only in the status check internal mode.
2. Only in the status check user mode.

FSL_SetFlashShieldWindow

[Overview]

Setting of the flash shield window

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_SetFlashShieldWindow(__near fsl_fsw_t* setfsw_pstr)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_SetFlashShieldWindow(__near fsl_fsw_t* setfsw_pstr)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_SetFlashShieldWindow(__near fsl_fsw_t* setfsw_pstr)
                                        __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_SetFlashShieldWindow or CALL !!_FSL_SetFlashShieldWindow
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function, FSL_PrepareFunctions function and FSL_PrepareExtFunctions function must make execution complete. Also, when an interrupt must be received before the processing is completed, use the FSL_ChangeInterruptTable function to change the interrupt destination to the RAM.

[Function]

Sets the flash shield window.

Caution If both (1) and (2) below are satisfied, this function can be allocated on the internal ROM for use.

(1) The status check mode is set to the status check internal mode with the FSL_Init function.

(2) "Do not use interrupts" or "disable interrupts on the internal ROM" until the processing of this function is completed (the reception of interrupts on the RAM is permitted).

Remark The flash shield window function is incorporated as a security function used during the execution of flash self-programming.
During the execution of flash self-programming, writing and erasure are permitted in the code flash memory in the range specified as the window, but prohibited in the code flash memory outside the specified range. However, during on-board/off-board programming, writing and erasure are permitted even in the code flash memory outside the range specified as the window. When the range specified as the window and the rewrite-protected area of Boot Cluster 0 overlap, rewrite protection of Boot Cluster 0 has precedence.

[Register State After Call]

	Development tool	Return value	Destructed register
	RENESAS CA78K0R compiler	C(General-purpose register)	-
	RENESAS CC-RL compiler	A(General-purpose register)	-
<R>	LLVM compiler	A(General-purpose register)	-

[Argument]

Definition of argument

Argument	Description
__near fsl_fsw_t* setfsw_pstr	Variable for FSW setting (FSW start block number and end block number)

Definition of __near fsl_fsw_t

<R>

Development tool	C language (Structure definition)	Assembly language (Example of definition)

RENESAS CA78K0R compiler	typedef struct {
	fsl_u16 fsl_start_block_u16;
	fsl_u16 fsl_end_block_u16;
	} fsl_fsw_t;
		fsl_fsw_str:
		fsl_start_block_u16: DS 2
		fsl_end_block_u16 : DS 2

RENESAS CC-RL compiler	typedef struct {
	fsl_u16 fsl_start_block_u16;
	fsl_u16 fsl_end_block_u16;
	} fsl_fsw_t;
		fsl_fsw_str:
		fsl_start_block_u16: .DS 2
		fsl_end_block_u16 : .DS 2

LLVM compiler	typedef struct {
	fsl_u16 fsl_start_block_u16;
	fsl_u16 fsl_end_block_u16;
	} fsl_fsw_t;
		Check the compiler specifications.

Contents of __near fsl_fsw_t

Argument	Description
fsl_u16 fsl_start_block_u16;	FSW start block storage buffer - Reserve a 2-byte data buffer.
fsl_u16 fsl_end_block_u16;	FSW end block storage buffer - Reserve a 2-byte data buffer.

Contents of argument settings

	Development tool	Argument Type/Register	
		C language	Assembly language
	RENESAS CA78K0R compiler	__near fsl_fsw_t* setfsw_pstr	AX(0-15): The start address of the variable (16 bits)
	RENESAS CC-RL compiler	__near fsl_fsw_t* setfsw_pstr	AX(0-15): The start address of the variable (16 bits)
<R>	LLVM compiler	__near fsl_fsw_t* setfsw_pstr	AX(0-15): The start address of the variable (16 bits)

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion[Note1]
0x05(FSL_ERR_PARAMETER)	Parameter error - The specification of the block number is outside the allowable setting range.
0x1A(FSL_ERR_ERASE)	Erasure error[Note1] - An error occurred during erasure processing.
0x1B(FSL_ERR_IVERIFY)	Internal verification error[Note1] - An error occurred during verification (internal verification) processing.
0x1C(FSL_ERR_WRITE)	Writing error[Note1] - An error occurred during write processing.
0x1F(FSL_ERR_FLOW)	Flow error - The processing of the flash function executed immediately before has not been completed.[Note2] - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.[Note2]
0xFF(FSL_BUSY)	Execution start of this function[Note2] - The execution of this function has been started. (Check the execution state with the FSL_StatusCheck function.)

Notes 1. Only in the status check internal mode.
2. Only in the status check user mode.

FSL_StatusCheck

[Overview]

Checking of the operation state of the flash function

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_StatusCheck( void )
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_StatusCheck( void )
```

<R> LLVM compiler

```
fsl_u08 __far FSL_StatusCheck(void) __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_StatusCheck or CALL !!_FSL_StatusCheck
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

- Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareFunctions function must make execution complete. Also, when an interrupt must be received before the processing is completed, use the FSL_ChangeInterruptTable function to change the interrupt destination to the RAM.
- This function can be used only in the status check user mode.

[Function]

Checks the start, progress, and status of the flash function executed immediately before.

[Register State After Call]

	Development tool	Return value	Destructed register
	RENESAS CA78K0R compiler	C(General-purpose register)	-
	RENESAS CC-RL compiler	A(General-purpose register)	-
<R>	LLVM compiler	A(General-purpose register)	-

[Argument]

None

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion[Note]
0x1A(FSL_ERR_ERASE)	Erasure error[Note] - An error occurred during erasure processing.
0x1B(FSL_ERR_IVERIFY)	Internal verification error[Note] - An error occurred during verification (internal verification) processing.
0x1B(FSL_ERR_BLANKCHECK)	Blank check error[Note] - The specified block is not in the blank state.
0x1C(FSL_ERR_WRITE)	Writing error[Note] - An error occurred during write processing.
0x1F(FSL_ERR_FLOW)	Flow error[Note] - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.
0x30(FSL_ERR_IDLE)	Non-execution error[Note] - No processing is in progress.
0xFF(FSL_BUSY)	Flash function in execution[Note] - The flash function is in execution.

Note Only in the status check user mode.

FSL_StandBy

[Overview]

Suspension of erasure processing (FSL_Erase) and pausing of flash self-programming

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_StandBy(void)
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_StandBy(void)
```

<R> LLVM compiler

```
fsl_u08 __far FSL_StandBy(void) __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_StandBy or CALL !!_FSL_StandBy
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

- Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareFunctions function must make execution complete.
- This function can be used only in the status check user mode.

[Function]

Suspends erasure processing (FSL_Erase) being executed, and holds the erasure processing (FSL_Erase) in the pause state until FSL_WakeUp is executed.
When this function is executed, flash self-programming enters the pause state, and flash self-programming cannot be executed until FSL_WakeUp is executed.

Cautions 1. During a pause of flash self-programming, the flash functions cannot be executed.
To restart flash self-programming, the FSL_WakeUp function must be executed.
2. A transition to the pause state occurs unless the return value is a flow error.

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
<R> LLVM compiler	A(General-purpose register)	-

[Argument]

None

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion[Note1]
0x1A(FSL_ERR_ERASE)	Erasure error[Note1] - An error occurred during erasure processing before suspension.
0x1B(FSL_ERR_BLANKCHECK)	Blank check error[Note1] - An error occurred during blank check processing before suspension.
0x1B(FSL_ERR_IVERIFY)	Internal verification error[Note1] - An error occurred during verification (internal verification) processing before suspension.
0x1C(FSL_ERR_WRITE)	Writing error[Note1] - An error occurred during write processing before suspension.
0x1F(FSL_ERR_FLOW)	Flow error[Note1] (does not result in the pause state) - The prerequisite defined in presetting is violated. - Flash self-programming is in the pause state.
0x30(FSL_ERR_IDLE)	Non-execution error[Note1] - No processing is in progress.
0x43(FSL_SUSPEND)	Pausing of the flash function[Note1, 2] - The processing of the flash function in execution is paused.

Notes 1. Only in the status check user mode.
2. Only when erasure processing is paused.

FSL_WakeUp

[Overview]

Canceling of the pause state to restart flash self-programming

[Format]

<C language>

RENESAS CA78K0R compiler

```
fsl_u08 FSL_WakeUp( void )
```

RENESAS CC-RL compiler

```
fsl_u08 __far FSL_WakeUp( void )
```

<R> LLVM compiler

```
fsl_u08 __far FSL_WakeUp(void) __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_WakeUp or CALL !!_FSL_WakeUp
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

- Before the execution of this function, after FSL_Init function is completed normally, FSL_Open function and FSL_PrepareFunctions function must make execution complete.
- This function can be used only in the status check user mode.

[Function]

Cancels the pause state and restarts flash self-programming. If block erasure processing is suspended, it is restarted.

[Register State After Call]

Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	C(General-purpose register)	-
RENESAS CC-RL compiler	A(General-purpose register)	-
<R> LLVM compiler	A(General-purpose register)	-

[Argument]

None

[Return Value]

State	Description
0x00(FSL_OK)	Normal completion[Note1]
0x1A(FSL_ERR_ERASE)	Erasure error[Note1] - An error occurred in the restarted erasure processing.
0x1F(FSL_ERR_FLOW)	Flow error[Note1] - The prerequisite defined in presetting is violated. - Flash self-programming is not in the pause state.
0xFF(FSL_BUSY)	Restarting of the flash function[Note1, 2] - The execution of the flash function was restarted. (Check the execution state with the FSL_StatusCheck function.)

Notes 1. Only in the status check user mode.
2. Only when erasure processing is restarted.

FSL_ForceReset

[Overview]

Resetting of the RL78 microcontroller in use

[Format]

<C language>

RENESAS CA78K0R compiler

```
void FSL_ForceReset(void)
```

RENESAS CC-RL compiler

```
void __far FSL_ForceReset(void)
```

<R> LLVM compiler

```
void __far FSL_ForceReset(void) __attribute__ ((section ("FSL_RCD")))
```

<Assembler>

```
CALL !_FSL_ForceReset or CALL !!_FSL_ForceReset
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

None

[Function]

Executes the command code of 0xFF to generate an internal reset of the RL78 microcontroller in use.

Cautions 1. The RL78 microcontroller in use is reset, so the code written after this function is not executed.

2. When this function is executed while E1,E2,E2 emulator Lite,E20 or IECUBE® is being used, a break occurs and processing stops.
   Normal operation cannot be done after the occurrence of a break. Execute a manual reset.
3. For the internal reset with the command code of 0xFF (internal reset through an illegal instruction), refer to the user's manual of the target RL78 microcontroller.

[Register State After Call]

The registers are not destructed.

[Argument]

None

[Return Value]

None

FSL_GetVersionString

[Overview]

Acquisition of the version of the flash self-programming library

[Format]

<C language>

RENESAS CA78K0R compiler

```
__far fsl_u08* FSL_GetVersionString( void )
```

RENESAS CC-RL compiler

```
__far fsl_u08* __far FSL_GetVersionString( void )
```

<R> LLVM compiler

```
__far fsl_u08 * __far FSL_GetVersionString(void)
                                        __attribute__ ((section ("FSL_FCD")))
```

<Assembler>

```
CALL !_FSL_GetVersionString or CALL !!_FSL_GetVersionString
```

Remark Call with "!" when the flash self-programming library is allocated at 00000H-0FFFFH, or call with "!!" otherwise.

[Presetting]

None

[Function]

Obtains the start address that holds the version information of the flash self-programming library.

[Register State After Call]

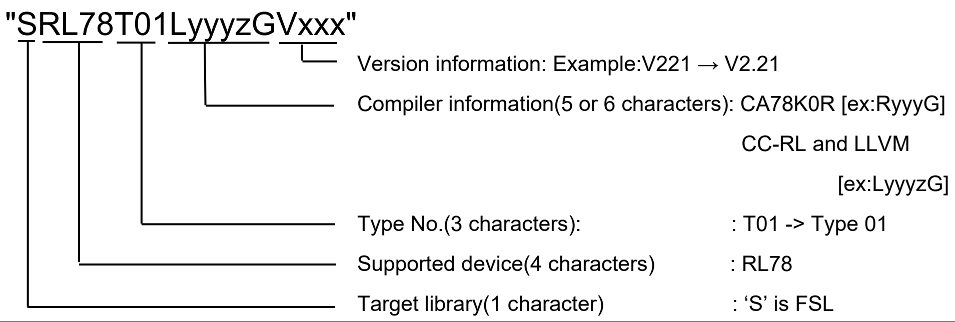
Development tool	Return value	Destructed register
RENESAS CA78K0R compiler	BC(0-15), DE(16-31)	-
RENESAS CC-RL compiler	DE(0-15), A(16-23)	-
<R> LLVM compiler	DE(0-15), A(16-23)	-

[Argument]

None

[Return Value]

<R>

Data type	Description
__far fsl_u08*	The version information storage start address of the flash self-programming library (far area) The version information of the flash self-programming library consists of ASCII characters. Example: Flash Self-Programming Library Type 01 "SRL78T01LyyyzGVxxx" Version information: Example:V221 → V2.21 Compiler information(5 or 6 characters): CA78K0R [ex:RyyyG] CC-RL and LLVM [ex:LyyyzG] Type No.(3 characters): : T01 -> Type 01 Supported device(4 characters) : RL78 Target library(1 character) : 'S' is FSL

APPENDIX A REVISION HISTORY

A. 1 Major Revisions in This Edition

Page	Description	Classification
Throughout the document		
-	Added information on the flash self-programming library for the LLVM compiler.	(b)
-	Moved the title of the figure number from the top of the figure to the bottom.	(c)
Chapter 2 Programming Environment		
P24	Added explanation about the LLVM compiler. Added software resources for the LLVM compiler in Table 2-7.	(c)
P26	Added stack size for the LLVM compiler in Table 2-8.	(c)
P28	Added explanation for allocating the self-RAM area when using the LLVM compiler.	(c)
P29	Added explanation about the LLVM compiler.	(c)
Chapter 6 Flash Function		
P45	Added explanation about the LLVM compiler.	(c)
P55	Added explanation about the LLVM compiler.	(c)
P56	Add "Note" in the description of <Assembler>.	(c)
P44-P111	Added settings for the LLVM compiler return values, C language format and argument definitions for each function.	(c)

Remark "Classification" in the above table classifies revisions as follows.

(a): Error correction, (b): Addition/change of specifications, (c): Addition/change of description or note,

(d): Addition/change of package, part number, or management division,

(e): Addition/change of related documents

A. 2 Revision History of Preceding Editions

Here is the revision history of the preceding editions. Chapter indicates the chapter of each edition.

Rev.	Description	Chapter
Rev.1.05	Correction of errors.	Throughout the document
	The ZIP file name was changed to the installer name.	Cover
	The supported device was corrected.	Chapter 6 Flash Function
	Operation example：Function name was changed from FSL_ChangeInterruptTable() to FSL_RestoreInterruptTable().	
	The emulator was added.	

Rev.	Description	Chapter
Rev.1.04	The English translation was reviewed and corrected.	Throughout the document
	The user's manual of the flash self-programming library for CC-RL was integrated into this manual.	
	Support of the RL78/G11 group microcontrollers was added.	
	In Table 2-5, the description of FSL_Erase Call Interval was corrected.	Chapter 2 Programming Environment
	A description regarding the supported compilers was added.	
	In Table 2-7, the software resources for the CC-RL compiler were added.	
	In Table 2-8, the stack sizes for the CC-RL compiler were added.	
	In Tables 2-10 and 2-11, the ROM and RAM sizes for the CC-RL compiler were added.	
	The methods for allocating the self-RAM area and specifying desired addresses in the CC-RL compiler were added.	
	A description regarding the start of the high-speed on-chip oscillator was added.	
	A description regarding operation frequency setting was added.	
	The applicable versions were added to the restriction on segment allocation.	
	The description that each segment must not extend across a 64-Kbyte boundary was added.	
	The method for representing hexadecimal numbers in the assembler in the Renesas CC-RL compiler package was added.	
	A restriction regarding allocation of the functions for boot swapping was added.	Chapter 5 Boot Swap Function
	In Table 6-1, the functions supported for the RL78/G11 group microcontrollers were added.	Chapter 6 Flash Function
	In note 1, the status check mode supported for the RL78/G11 group microcontrollers were added.	
	The heading of section 6.2 was changed ("Section" was added). The description regarding allocation to specified areas was reviewed and corrected.	
	The return values from each library function in the CC-RL compiler were added	
	The C-language format for each library function in the CC-RL compiler was added.	
	The definitions and descriptions of the arguments for each library function in the CC-RL compiler were added.	
	The descriptions regarding the return values and arguments in each function were corrected.	
	The descriptions regarding the boot area addresses were modified.	
	Caution 7 was added to the description of the FSL_SwapBootCluster function.	

Rev.	Description	Chapter
Rev.1.03	The corresponding ZIP file name and release version were added to the cover.	Throughout the document
	The target device descriptions were deleted.	
	References to the list of the target MCUs were added.	
	The term "voltage mode" was changed to "flash memory programming mode" for consistency of terminology.	
	Various types of operating frequency described in the former version were unified to the CPU operating frequency.	
	A description regarding boot cluster 0 was added to the boot area.	
	The FSL_IVerify state check processing was added to Figure 2-2.	Chapter 1 Overview
	A description of the case when flash functions are executed in the RAM was added.	Chapter 2 Programming Environment
	The formula for calculating the minimum time of FSL_BlankCheck was added.	
	In Table 2-4, the formula for calculating the processing time for each function was corrected.	
	The resources used to run the flash self-programming library were corrected.	
	In Table 2-7, the description of the self-RAM area was changed.	
	In note 1 on Table 2-7, the inquiry about device specifications was changed.	
	The restriction on versions up to 2.10 was deleted from Figure 2-9.	
	A note was added to Table 2-8 Stack Size Used by Flash Functions.	
	The description of the self-RAM was reviewed and corrected.	
	The available range for stack and data buffer specifications was corrected.	
	In (3), the description of the functions that require special care regarding the watchdog timer operation was reviewed and corrected.	
	In (18), note on the prohibition of 64KB boundary arrangement added.	
	In Table 6-1, the column of basic functions added and position and description of note was changed.	Chapter 6 Flash Function
	The status check mode was corrected from the user mode to the internal mode (in table titles, etc.).	

Rev.	Description	Chapter
Rev.1.02	The document on the data flash library, which was classified as the application note (old version of R01AN0350), was changed to the user's manual.	Throughout the document
	The corresponding installer and release version were added to the cover page.	
	Contents of the processing time and software resources were moved from the usage note to this document. Accordingly, the reference destination described in this document was also changed.	
	The supported device was added.	
	The notation of high-speed OCO was deleted to unify the notation of high-speed on-chip oscillator.	
	The description of the operating frequency was unified to the CPU operating frequency since individual descriptions had different notations.	
	The state transition from "prepared" to "extprepared" was added to figure 1-1.	Chapter 1 Overview
	Description on the FSL_PrepareFunctions function was added.	
	The names of functions were clearly stated in the overall description.	
	Notes on rewriting were added.	
	Notes on internal verification were added.	
	State when the FSL_Close function is executed was added.	
	Notes on the interrupt were added.	Chapter 2 Programming Environment
	Description of the mode selection was added to the example of controlling rewriting of the flash memory.	
	Description of the initial setting was added	
	Items regarding the processing time were added (the description of the processing time was moved from the usage note to this document).	
	Items regarding the resources were added (the description on the resources was moved from the usage note to this document).	
	Note on the frequency of the high-speed on-chip oscillator was added.	
	Note on the RAM parity error was added.	
	Note on the writing was added.	
	Description of the non-supported product (R5F10266) was added.	
	Note on the interrupt was added.	
	Note on the boot swap function was added.	
	Note on the security setting was added.	
	Notes on the interrupt were added.	Chapter 3 Interrupts During Execution of Flash Self-Programming
	Note on the security setting was added.	Chapter 4 Security Setting
	Tables of defining structures and tables of parameter contents were added. Descriptions were added to the sections for assembler in the field of settings of arguments.	Chapter 6 Flash Function
	Note on the internal reset was added.	

RL78 Microcontrollers User’s Manual: Flash Self-Programming Library Type 01

Publication Date:	Rev.1.00	Mar 31, 2011
	Rev.1.05	Oct 31, 2018
	Rev.1.10	Dec 26, 2023

Published by: Renesas Electronics Corporation

RL78 Family
Flash Self-Programming Library Type 01

Renesas Electronics Corporation

R01US0050EJ0110