User’s Manual

78K0/Kx2

8-Bit Single-Chip Microcontrollers

Flash Memory Self Programming

μPD78F0500
μPD78F0501
μPD78F0502
μPD78F0503
μPD78F0503D
μPD78F0511
μPD78F0512
μPD78F0513
μPD78F0513D
μPD78F0514
μPD78F0515
μPD78F0515D
μPD78F0521
μPD78F0522
μPD78F0523
μPD78F0524
μPD78F0525
μPD78F0526
μPD78F0527
μPD78F0527D
μPD78F0531
μPD78F0532
μPD78F0533
μPD78F0534
μPD78F0535
μPD78F0536
μPD78F0537
μPD78F0537D
μPD78F0544
μPD78F0545
μPD78F0546
μPD78F0547
μPD78F0547D

Document No. U17516EJ1V0UM00 (1st edition)
Date Published August 2005 N CP(K)

[MEMO]

NOTES FOR CMOS DEVICES

① 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, etc., 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).

② HANDLING OF UNUSED INPUT PINS

Unconnected CMOS device inputs can be cause of malfunction. If an input pin is unconnected, it is possible that an internal input level may be generated due to noise, etc., causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using pull-up or pull-down circuitry. Each unused pin should be connected to V_{DD} or GND via a resistor if there is a possibility that it will be an output pin. All handling related to unused pins must be judged separately for each device and according to related specifications governing the device.

③ PRECAUTION AGAINST ESD

A strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it when it has occurred. Environmental control must be adequate. When it is dry, a humidifier should be used. It is recommended to avoid using insulators that 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 should be grounded. The operator should be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with mounted semiconductor devices.

④ STATUS BEFORE INITIALIZATION

Power-on does not necessarily define the initial status of a MOS device. Immediately after the power source is turned ON, devices with reset functions have not yet been initialized. Hence, power-on does not guarantee output pin levels, I/O settings or contents of registers. A device is not initialized until the reset signal is received. A reset operation must be executed immediately after power-on for devices with reset functions.

⑤ POWER ON/OFF SEQUENCE

In the case of a device that uses different power supplies for the internal operation and external interface, as a rule, switch on the external power supply after switching on the internal power supply. When switching the power supply off, as a rule, switch off the external power supply and then the internal power supply. Use of the reverse power on/off sequences may result in the application of an overvoltage to the internal elements of the device, causing malfunction and degradation of internal elements due to the passage of an abnormal current.

The correct power on/off sequence must be judged separately for each device and according to related specifications governing the device.

⑥ INPUT OF SIGNAL DURING POWER OFF STATE

Do not input signals or an I/O pull-up power supply while the device is not powered. 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.

Input of signals during the power off state must be judged separately for each device and according to related specifications governing the device.

- **The information in this document is current as of July, 2005. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".

  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.

  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.

  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).

  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)

(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.

(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02. 11-1

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC Electronics product in your application, pIease contact the NEC Electronics office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

[GLOBAL SUPPORT]
http://www.necel.com/en/support/support.html

NEC Electronics America, Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782

NEC Electronics (Europe) GmbH
Duesseldorf, Germany
Tel: 0211-65030

- **Sucursal en España**
  Madrid, Spain
  Tel: 091-504 27 87

- **Succursale Française**
  Vélizy-Villacoublay, France
  Tel: 01-30-67 58 00

- **Filiale Italiana**
  Milano, Italy
  Tel: 02-66 75 41

- **Branch The Netherlands**
  Eindhoven, The Netherlands
  Tel: 040-265 40 10

- **Tyskland Filial**
  Taeby, Sweden
  Tel: 08-63 87 200

- **United Kingdom Branch**
  Milton Keynes, UK
  Tel: 01908-691-133

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-558-3737

NEC Electronics Shanghai Ltd.
Shanghai, P.R. China
Tel: 021-5888-5400

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377

NEC Electronics Singapore Pte. Ltd.
Novena Square, Singapore
Tel: 6253-8311

J05.6

INTRODUCTION

Readers

This manual is intended for users who wish to understand the functions of the flash memory versions of the 78K0/Kx2 and design application systems using these microcontrollers.

Purpose

This manual is intended to give users an understanding of the usage of the flash memory self programming sample library which is used when rewriting the 78K0/Kx2 flash memory.

Organization

This manual can be generally divided into the following sections.

- Description of flash environment
- Description of flash memory self programming sample library

How to Read This Manual

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

To check the hardware functions of the 78K0/Kx2
→ Refer to the user's manual of each 78k0/Kx2 product.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active low representation:	$\overline{\text{xxx}}$ (overscore over pin or signal name)
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numerical representation:	Binary … xxxx or xxxxB Decimal … xxxx Hexadecimal … xxxxH

Terminology The following describes the meanings of certain terms used in this manual.

- Self programming
  Self programming operations are flash memory write operations that are performed by user programs.
- Flash memory self programming sample library
  This is the library that is provided by the 78K0/Kx2 for flash memory manipulation.
- Flash environment
  This is the environment that supports flash memory manipulations. It has restrictions that differ from those applied to ordinary program execution.
- Block number
  Block numbers indicate blocks in flash memory. They are used as units during manipulations such as erasures and blank checks.
- Boot cluster
  This is the area that is used for boot swapping. Boot cluster 0 and boot cluster 1 are provided and the cluster to be booted can be selected.
- Entry RAM
  This is the area in RAM that is used by the flash memory self programming sample library. The user program reserves this area and specifies the start address of the specific area to be used when the library is called.
- Internal verification
  After writing to flash memory, signal levels are checked internally to confirm correct reading of data. When an internal verification error occurs, the corresponding device is judged as faulty.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 GENERAL

1.1 Overview

The self programming sample library is firmware provided on the 78K0/Kx2, and is software which is used to rewrite data in the flash memory.

By calling the self programming sample library from a user program, the contents of the flash memory can be rewritten and, consequently, the period for software development can be substantially shortened.

Cautions 1. Because the self programming sample library rewrites the contents of the flash memory by using the CPU, registers, and RAM of the 78K0/Kx2, a user program cannot be executed while processing of the self programming sample library is being executed.

2. The self programming sample library uses the CPU (register bank 3) and a work area (100 bytes of entry RAM). Therefore, the user must save the data necessary for the user program in that area immediately before calling the self programming sample library.

1.2 Calling Self Programming Sample Library

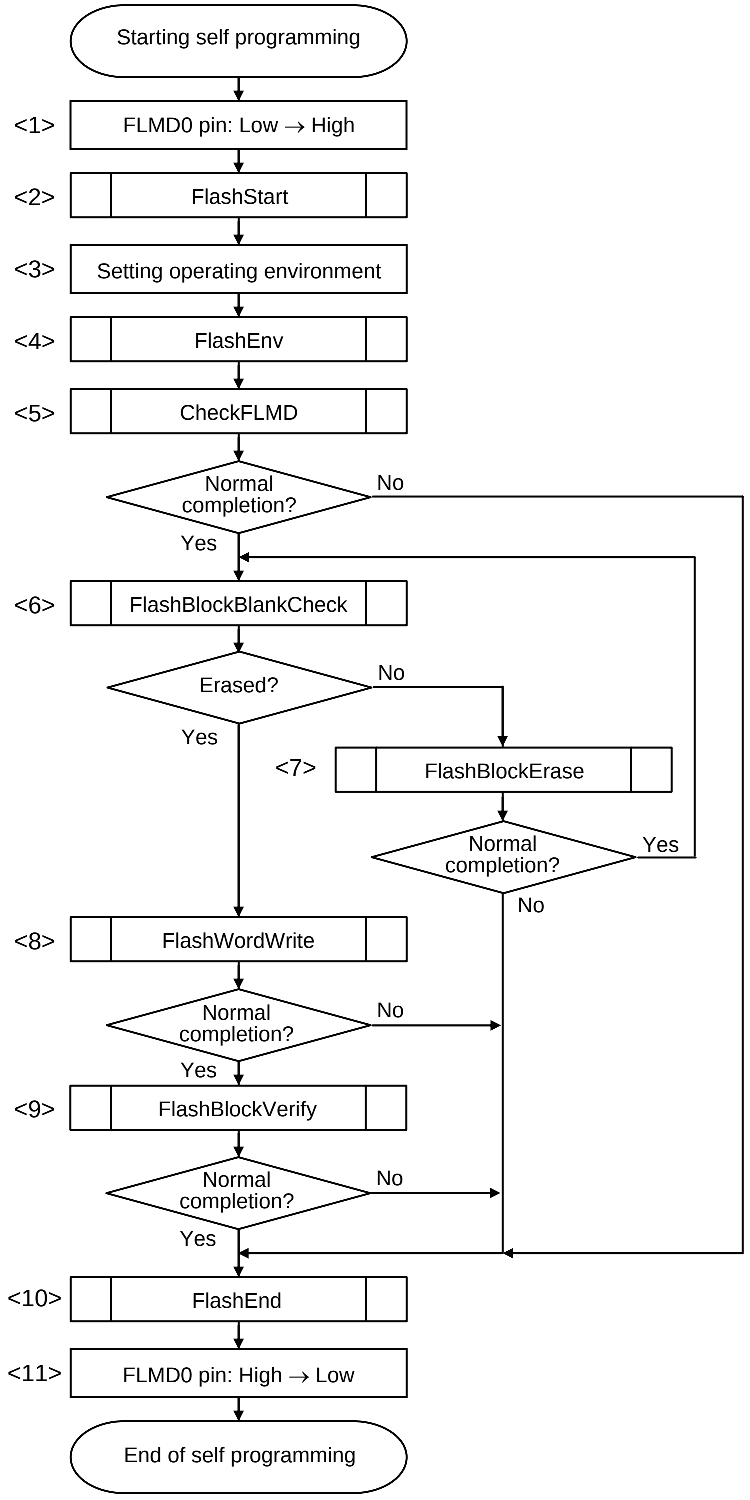
The self programming sample library can be called by a user program in C or an assembly language.

If the -SM option (that uses an object as a static model) is specified when a file written in C is complied, use (link) the library for static models. If the -SM option is not specified, link the library for normal models.

If the file is written in an assembly language, use (link) the library for static models.

The following flowchart illustrates how to rewrite the contents of the flash memory by using the self programming sample library.

Figure 1-1. Flow of Self Programming (rewriting contents of flash memory)



<1> Preprocessing (setting of hardware environment)
As preprocessing, make the FLMD0 pin high (refer to **2.1 Hardware Environment**).

<2> Preprocessing (declaring start of self programming)
As preprocessing, call the self programming start library FlashStart to declare the start of self programming.

<3> Preprocessing (setting of software environment)
As preprocessing, save register bank 3 and specify a work area (refer to **2.2 Software Environment**).

<4> Preprocessing (initializing entry RAM)
As preprocessing, call the initialize library FlashEnv to initialize the entry RAM.

<5> Preprocessing (checking voltage level)
As preprocessing, call the mode check library CheckFLMD and check the voltage level.

<6> Checking erasing of specified block (1 KB)
Call the block blank check library FlashBlockBlankCheck to check if the specified block (1 KB) has been erased.

<7> Erasing specified block (1 KB)
Call the block erase library FlashBlockErase to erase a specified block (1 KB).

<8> Writing data of 1 to 64 words to specified addresses
Call the word write library FlashWordWrite to write data of 1 to 64 words to specified addresses.

<9> Verifying specified block (1 KB) (internal verification)
Call the block verify library FlashBlockVerify to verify a specified block (1 KB) (internal verification).

<10> Post-processing (declaring end of self programming)
As post-processing, call the self programming end library FlashEnd to declare the end of self programming.

<11> Post-processing (setting of hardware environment)
As post-processing, return the level of the FLMD0 pin to the low level.

1.3 Bank Number and Block Number

Products in the 78K0/Kx2 Series having flash memory of up to 60 KB have their flash memory divided into 1 KB blocks. Erasing, blank checking, and verification (internal verification) for self programming are performed in these block units. To call the self programming sample library, a block number is specified.

Addresses 0000H to 0FFFH and 1000H to 1FFFH of the 78K0/Kx2 are allocated for boot clusters. A boot cluster is an area that is used to prevent the vector table data and basic functions of the program from being destroyed, and to prevent the user program from being unable to start due to a power failure or because the device was reset while an area including a vector area was being rewritten. For details on the boot cluster, refer to **CHAPTER 4 BOOT SWAP FUNCTION**.

Figure 1-2 shows the block numbers and boot clusters of a flash memory of up to 60 KB.

78K0/Kx2 products having flash memory of more than 96 KB have banks in an area that is larger than 32 KB. For these products, not only a block number but also a bank number must be specified to call the self programming sample library when performing erasing, blank checking, or verification (internal verification) in the area that is larger than 32 KB during self programming.

Figure 1-3 shows the block numbers and boot clusters of a flash memory of more than 96 KB.

Figure 1-2. Block Numbers and Boot Clusters (flash memory of up to 60 KB)

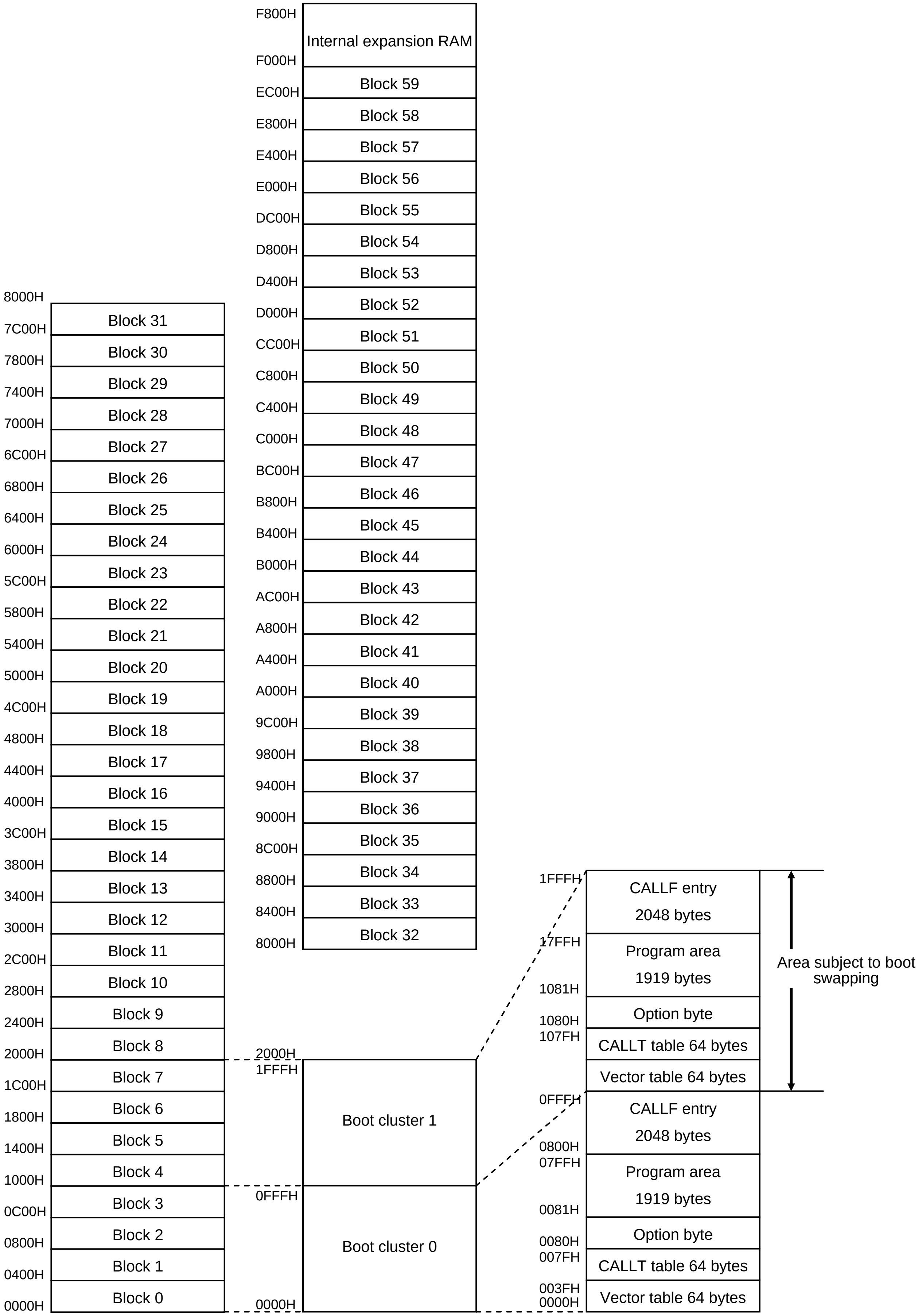
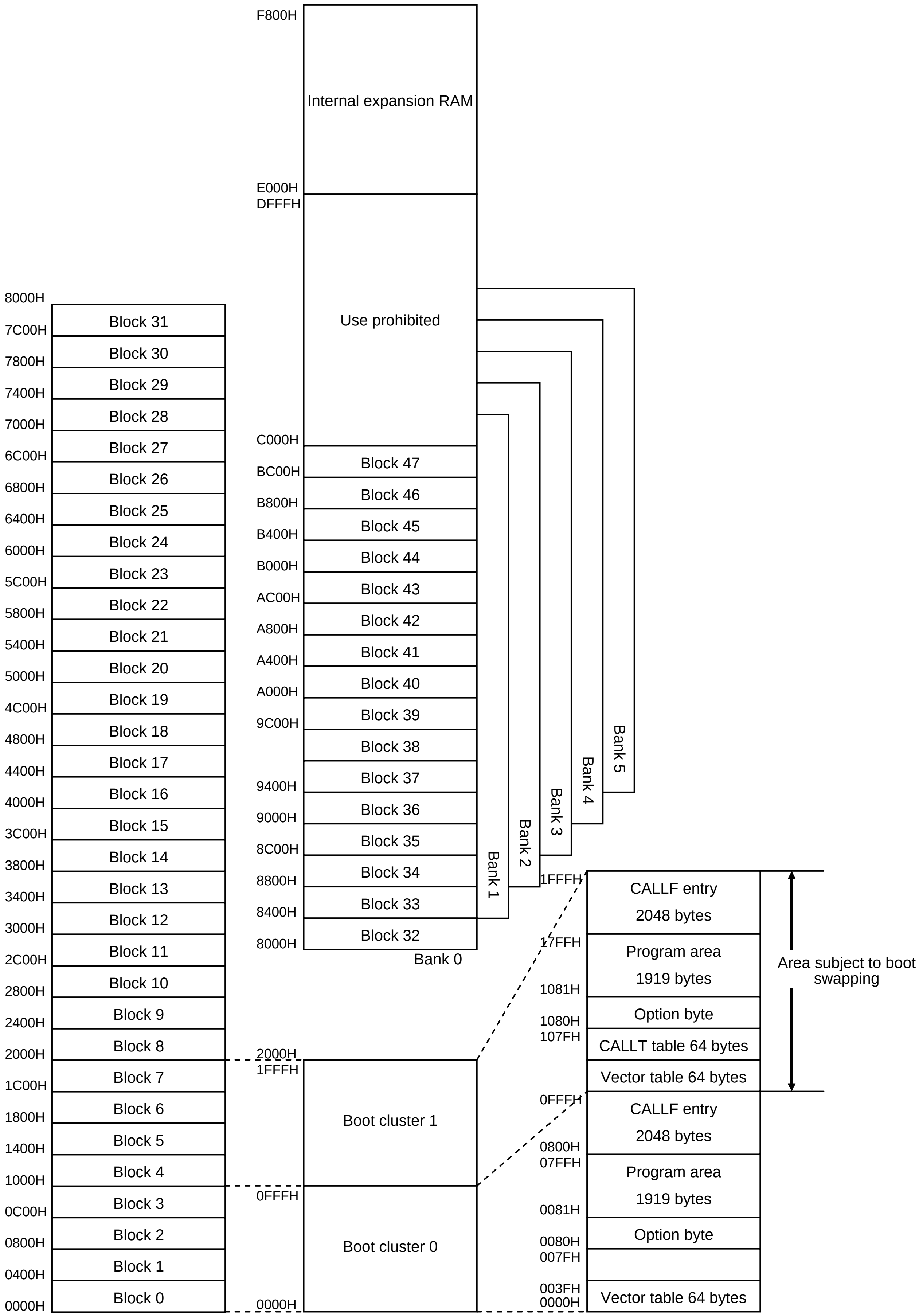


Figure 1-3. Block Numbers and Boot Clusters (flash memory of 96 KB or more)



1.4 Processing Time and Acknowledging Interrupt

Table 1-1 and Table 1-2 show the processing time of the self programming sample library and whether interrupts can be acknowledged. Table 1-1 shows a case where an internal high-speed oscillator is used for the main system clock and Table 1-2 shows a case where an external system clock is used for the main system clock.

The self programming sample library that can acknowledge interrupts has a function to check if an interrupt is generated while processing of the self programming sample library is under execution, and a function to perform post-processing if an interrupt has been generated.

For details on interrupts, refer to **CHAPTER 3 INTERRUPT SERVICING DURING SELF PROGRAMMING**.

Table 1-1. Processing Time and Acknowledging Interrupt (with internal high-speed oscillator)

Library Name	Processing Time (unit: microseconds)								Acknowledging Interrupt
	Outside short direct addressing range				In short direct addressing range				
	Normal model		Static model		Normal model		Static model		
	Min	Max	Min	Max	Min	Max	Min	Max	
self programming start library	4.25								Not acknowledged
initialize library	977.75				443.5				Not acknowledged
mode check library	753.875		753.125		219.625		218.875		Not acknowledged
block blank check library	12770.875		12765.875		12236.625		12231.625		Acknowledged
block erase library	36909.5	356318	36904.5	356296.25	36363.25	355771.75	36358.25	355750	Acknowledged
word write library	1214 (1214.375)	2409 (2409.375)	1207 (1207.375)	2402 (2402.375)	679.75 (680.125)	1874.75 (1875.125)	672.75 (673.125)	1867.75 (1868.125)	Acknowledged
block verify library	25618.875		25613.875		25072.625		25067.625		Acknowledged
self programming end library	4.25								Not acknowledged
get information library (option value: 03H)	871.25 (871.375)		866 (866.125)		337 (337.125)		331.75 (331.875)		Not acknowledged
get information library (option value: 04H)	863.375 (863.5)		858.125 (858.25)		329.125 (239.25)		323.875 (324)		Not acknowledged
get information library (option value: 05H)	1042.75 (1043.625)		1037.5 (1038.375)		502.25 (503.125)		497 (497.875)		Not acknowledged
set information library	105524.75	790809.375	105523.75	790808.375	104978.5	541143.125	104977.5	541142.125	Acknowledged
EEPROM write library	1496.5 (1496.875)	2691.5 (2691.875)	1489.5 (1489.875)	2684.5 (2684.875)	962.25 (962.625)	2157.25 (2157.625)	955.25 (955.625)	2150.25 (2150.625)	Acknowledged

Remark Values in parentheses are when the write start address structure is placed outside of internal high-speed RAM.

Table 1-2. Processing Time and Acknowledging Interrupt (with external system clock used)

Library Name	Processing Time (unit: microseconds)								Acknowledging Interrupt
	Outside short direct addressing range				In short direct addressing range				
	Normal model		Static model		Normal model		Static model		
	Min	Max	Min	Max	Min	Max	Min	Max	
self programming start library	34/fx[Note]								Not acknowledged
initialize library	49x[Note] + 485.8125				49/fx[Note] + 224.6875				Not acknowledged
mode check library	35/fx[Note] + 374.75		29/fx[Note] + 374.75		35/fx[Note] + 113.625		29/fx[Note] + 113.625		Not acknowledged
block blank check library	174/fx[Note] + 6382.0625		134/fx[Note] + 6382.0625		174/fx[Note] + 6120.9375		134/fx[Note] + 6120.9375		Acknowledged
block erase library	174/fx[Note] + 31093.875	174/fx[Note] + 298948.125	134/fx[Note] + 31093.875	134/fx[Note] + 298948.125	174/fx[Note] + 30820.75	174/fx[Note] + 298675	134/fx[Note] + 30820.75	134/fx[Note] + 298675	Acknowledged
word write library	318(321)/fx[Note] + 644.125	318(321)/fx[Note] + 1491.625	262(265)/fx[Note] + 644.125	262(265)/fx[Note] + 1491.625	318(321)/fx[Note] + 383	318(321)/fx[Note] + 1230.5	262(265)/fx[Note] + 383	262(265)/fx[Note] + 1230.5	Acknowledged
block verify library	174/fx[Note] + 13448.5625		134/fx[Note] + 13448.5625		174/fx[Note] + 13175.4375		134/fx[Note] + 13175.4375		Acknowledged
self programming end library	34x[Note]								Not acknowledged
get information library (option value: 03H)	171(172)/fx[Note] + 432.4375		129(130)/fx[Note] + 432.4375		171(172)/fx[Note] + 171.3125		129(130)/fx[Note] + 171.3125		Not acknowledged
get information library (option value: 04H)	181(182)/fx[Note] + 427.875		139(140)/fx[Note] + 427.875		181(182)/fx[Note] + 166.75		139(140)/fx[Note] + 166.75		Not acknowledged
get information library (option value: 05H)	404(411)/fx[Note] + 496.125		362(369)/fx[Note] + 496.125		404(411)/fx[Note] + 231.875		362(369)/fx[Note] + 231.875		Not acknowledged
set information library	75/fx[Note] + 79157.6875	75/fx[Note] + 652400	67/fx[Note] + 79157.6875	67/fx[Note] + 652400	75/fx[Note] + 78884.5625	75/fx[Note] + 527566.875	67/fx[Note] + 78884.5625	67/fx[Note] + 527566.875	Acknowledged
EEPROM write library	318(321)/fx[Note] + 799.875	318(321)/fx[Note] + 1647.375	262(265)/fx[Note] + 799.875	262(265)/fx[Note] + 1647.375	318(321)/fx[Note] + 538.75	318(321)/fx[Note] + 1386.25	262(265)/fx[Note] + 538.75	262(265)/fx[Note] + 1386.25	Acknowledged

Note fx: Operating frequency of external system clock

Remark Values in parentheses are when the write start address structure is placed outside of internal high-speed RAM.

CHAPTER 2 PROGRAMMING ENVIRONMENT

This chapter explains the hardware environment and software environment necessary for the user to rewrite flash memory by using the self programming sample library.

2.1 Hardware Environment

To execute self programming, a circuit that controls the voltage on the FLMD0 pin of the 78K0/Kx2 is necessary.

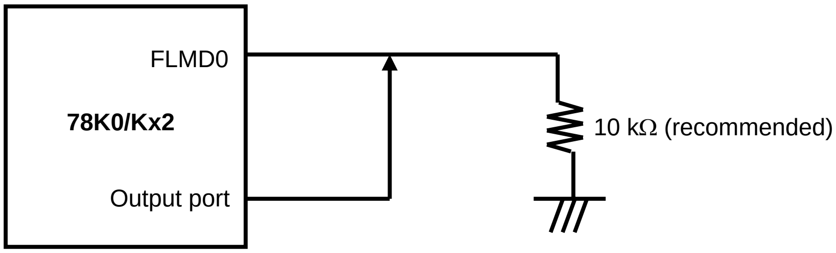
The voltage on the FLMD0 pin must be low while an ordinary user program is being executed (in normal operation mode) and high while self programming is being executed (in flash rewriting mode).

While the FLMD0 pin is low, the firmware and software for rewriting run, but the circuit for rewriting flash memory does not operate. Therefore, the flash memory is not actually rewritten.

A self programming sample library that makes the FLMD0 pin high is not provided. Therefore, to rewrite the flash memory, the voltage level of the FLMD0 pin must be made high by manipulating a port through user program, before calling the self programming start library.

Here is an example of the circuit that changes the voltage on the FLMD0 pin by manipulating a port.

Figure 2-1. FLMD0 Voltage Generator



2.2 Software Environment

The self programming sample library allocates its program to a user area and consumes about 500 bytes of the program area. The self programming sample library itself uses the CPU (register bank 3), work area (entry RAM), stack, and data buffer.

The following table lists the necessary software resources.

Table 2-1. Software Resources

Item	Description	Restriction
CPU	Register bank 3	–
Work area	Entry RAM: 100 bytes	Internal high-speed RAM outside short addressing range or internal high-speed RAM in short direct addressing range with first address as FE20H (Refer to 2.2.1 Entry RAM.)
Stack	39 bytes max. **Remark** Use the same stack as for the user program.	Internal high-speed RAM other than FE20H to FE83H (Refer to 2.2.2 Stack and data buffer.)
Data buffer	1 to 256 bytes **Remark** The size of this buffer varies depending on the writing unit specified by the user program.	Internal high-speed RAM other than FE20H to FE83H (Refer to 2.2.2 Stack and data buffer.)
Program area	Normal model: 525 bytes Static model: 432 bytes **Remark** Supplied as an assembly-language source.	Within 0000H to 7FFFH (32 KB) **Caution The self programming sample library and the user program that calls the self programming sample library must always be located within the above range, because the firmware built into the product is allocated to addresses starting from 8000H.**

Cautions 1. The self programming operation is not guaranteed if the user manipulates the above resources. Do not manipulate these resources during a self programming operation.

2. The user must release the above resources used by the self programming sample library before calling the self programming sample library.

2.2.1 Entry RAM

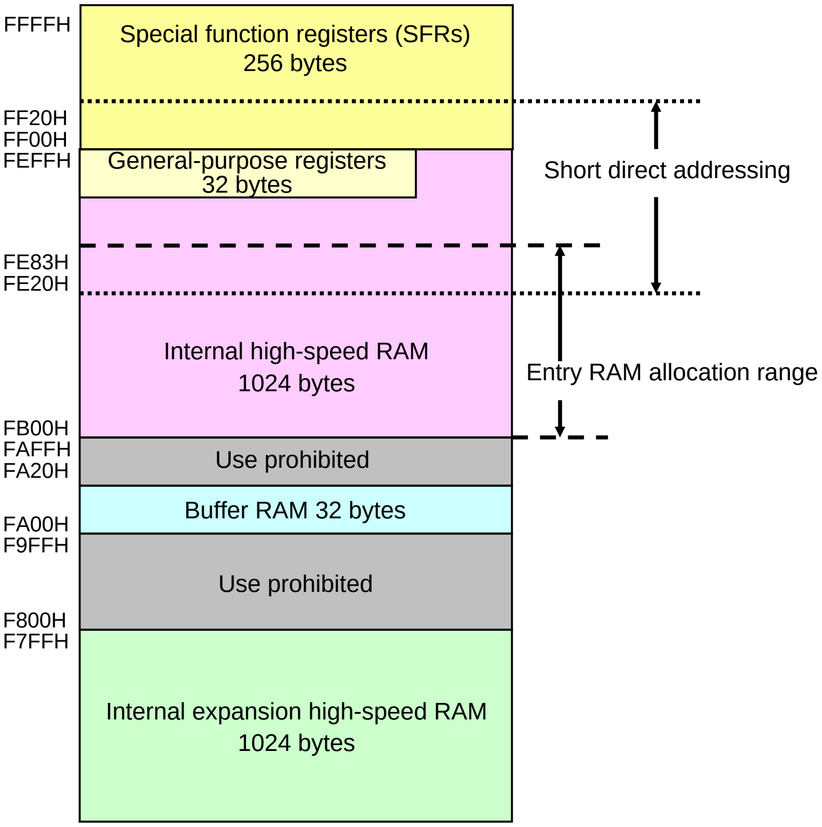
The self programming sample library uses a work area of 100 bytes. This area is called entry RAM.

As the entry RAM, 100 bytes are automatically allocated, starting from the first address that is specified when the initialize library is called. Therefore, the first address of the entry RAM can be specified in the range from FB00H to FE20H.

In addition, a data buffer used by the initialize library to actually write data to the flash memory must be allocated to an area that is within the range from 1 to 256 bytes and is other than the work area. For details on the data buffer, refer to 2.2.2 Stack and data buffer.

The range in which the entry RAM can be allocated is shown below.

Figure 2-2. Allocation Range of Entry RAM



Caution The size of the internal expansion high-speed RAM varies depending on the product.
For the size of the internal expansion high-speed RAM, refer to the user's manual of each product.

2.2.2 Stack and data buffer

The self programming sample library writes data to flash memory by using the CPU. Therefore, a self programming operation is performed by using the stack specified by the user program.

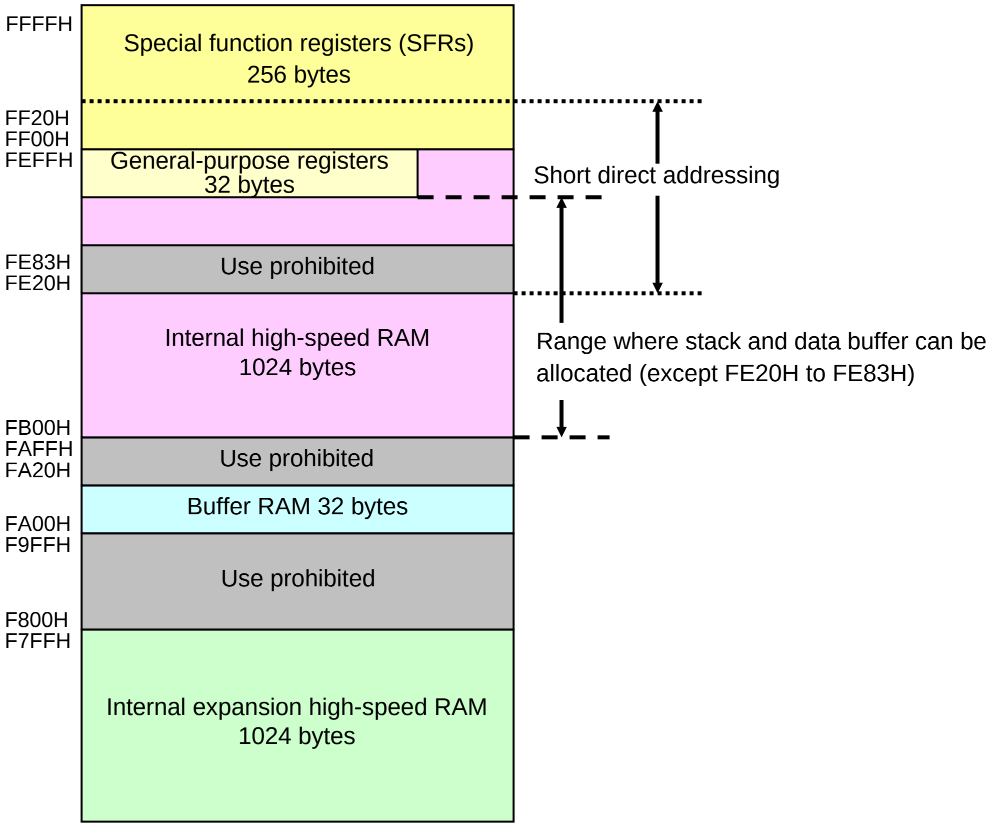
The stack must be allocated by stack processing of the self programming operation so that the entry RAM and the RAM used by the user are not cleared. Therefore, the stack can be allocated in the internal high-speed RAM at addresses other than FE20H to FE83H.

A data buffer is automatically allocated from the first address and by the number of data specified when the word write library is called. Therefore, the first address of the data buffer can be specified in the internal high-speed RAM at an address other than FE20H to FE83H, just as for the stack pointer.

Note that data to be written to the flash memory must be appropriately set and processed before the word write library is called.

The following figure shows the range in which the stack pointer and data buffer can be allocated.

Figure 2-3. Allocatable Range for Stack Pointer and Data Buffer



Caution The size of the internal expansion high-speed RAM varies depending on the product.
For the size of the internal expansion high-speed RAM, refer to the user's manual of each product.

CHAPTER 3 INTERRUPT SERVICING DURING SELF PROGRAMMING

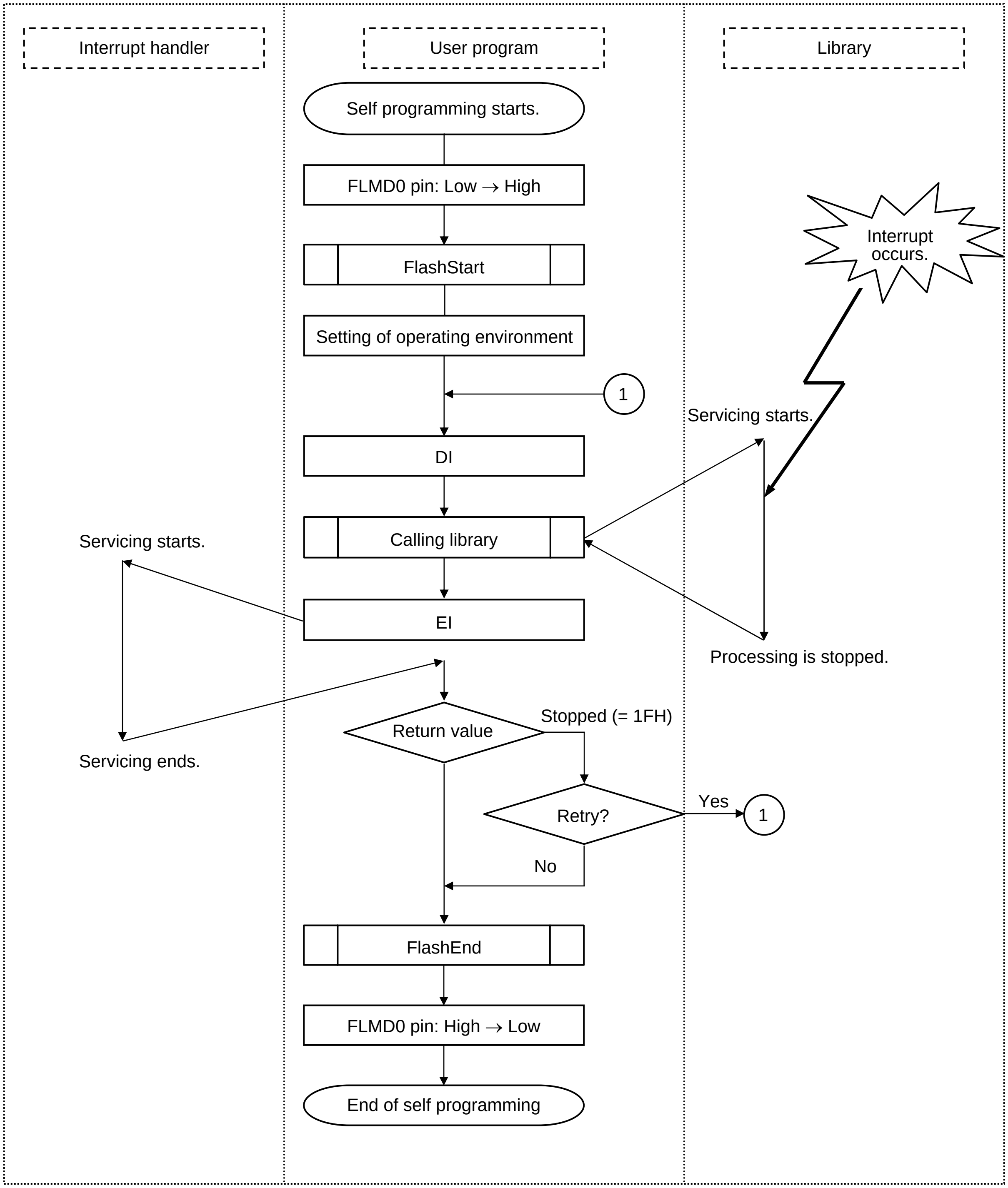
3.1 Overview

An interrupt can be generated, even while self programming is executed, in some self programming sample libraries of the 78K0/Kx2.

However, unlike the case for an ordinary interrupt, the user must decide whether the processing that has been interrupted should be resumed, by checking the return value from the self programming sample library.

The following figure illustrates the flow of processing if an interrupt is generated while processing of the self programming sample library is being executed.

Figure 3-1. Flow of Processing in Case of Interrupt



The following table shows how the processing of the self programming sample libraries that acknowledge interrupts is resumed after the processing has been stopped by the occurrence of an interrupt.

Table 3-1. Resume Processing Stopped by Interrupt

Library Name	Resuming Method
block blank check library	Call the block blank check library FlashBlockBlankCheck to resume processing to check block erasure that has been stopped by the occurrence of an interrupt.
block erase library	To resume processing to erase blocks that was stopped by the occurrence of an interrupt, call the block blank check library FlashBlockBlankCheck and check whether blocks that should be erased have been erased. Then, call the block erase library FlashBlockErase.
word write library	Call the word write library FlashWordWrite to resume data write processing that was stopped by the occurrence of an interrupt.
block verify library	Call the block verify library FlashBlockVerify to resume block verify processing that was stopped by the occurrence of an interrupt.
set information library	Call the set information library FlashSetInfo to resume flash information setting processing that was stopped by the occurrence of an interrupt.
EEPROM write library	Call the EEPROM write library FlashEEPROMWrite to resume processing to write data during EEPROM emulation that was stopped by the occurrence of an interrupt.

Remark An interrupt is not acknowledged until all of the processing of the above self programming sample libraries has been completed, because these libraries execute their processing with interrupts disabled.

3.2 Interrupt Response Time

Unlike the case for an ordinary interrupt, generation of an interrupt during execution of self programming is accomplished via post-interrupt servicing in the self programming sample library (such as setting 0x1F as the return value from the self programming sample library). Consequently, the response time is longer than that for an ordinary interrupt.

When an interrupt occurs during self programming execution, both the interrupt response time of the self programming sample library, as well as the interrupt response time of the device used, are necessary.

Remark For the response time of each device, refer to the user's manual of each device.

Table 3-2 and Table 3-3 show the interrupt response time of the self programming sample library. Table 3-2 is a case where the internal high-speed oscillator is used to generate the main system clock, and Table 3-3 is a case where an external system clock is used as the main system clock.

Table 3-2. Interrupt Response Time (with Internal High-Speed Oscillator)

Library Name	Interrupt Response Time (Unit: Microseconds)			
	Entry RAM outside short direct addressing range		Entry RAM inside short direct addressing range (from FE20H)	
	Min	Max	Min	Max
block blank check library	391.25	1300.5	81.25	727.5
block erase library	389.25	1393.5	79.25	820.5
word write library	394.75	1289.5	84.75	716.5
block verify library	390.25	1324.5	80.25	751.5
set information library	387	852.5	77	279.5
EEPROM write library	399.75	1395.5	89.75	822.5

Remark An interrupt is not acknowledged until all of the processing of the above self programming sample libraries has been completed, because these libraries execute their processing with interrupts disabled.

Table 3-3. Interrupt Response Time (with External System Clock)

Library Name	Interrupt Response Time (Unit: Microseconds)			
	Entry RAM outside short direct addressing range		Entry RAM inside short direct addressing range (from FE20H)	
	Min	Max	Min	Max
block blank check library	18/fx[Note] + 192	28/fx[Note] + 698	18/fx[Note] + 55	28/fx[Note] + 462
block erase library	18/fx[Note] + 186	28/fx[Note] + 745	18/fx[Note] + 49	28/fx[Note] + 509
word write library	22/fx[Note] + 189	28/fx[Note] + 693	22/fx[Note] + 52	28/fx[Note] + 457
block verify library	18/fx[Note] + 192	28/fx[Note] + 709	18/fx[Note] + 55	28/fx[Note] + 473
set information library	16/fx[Note] + 190	28/fx[Note] + 454	16/fx[Note] + 53	28/fx[Note] + 218
EEPROM write library	22/fx[Note] + 191	28/fx[Note] + 783	22/fx[Note] + 54	28/fx[Note] + 547

Note fx: Operating frequency of external system clock

Remark An interrupt is not acknowledged until all of the processing of the above self programming sample libraries has been completed, because these libraries execute their processing with interrupts disabled.

3.3 Description Example

This section shows an example of writing a user program that resumes erase processing that was stopped by the occurrence of an interrupt during execution of a self programming sample library (block erase library).

```
ERS_RTRY:
      ; Main processing
      MOV    A, #0                ; Sets 0 as the bank number of the block to be erased.
      MOV    B, #10               ; Sets 10 as the block number of the block to be erased.
      DI                          ; Disables interrupts.
      CALL   !_FlashBlockErase    ; Calls the block erase library.
      EI                          ; Enables interrupts.
      CMP    A, #1FH              ; Checks whether a stop status is set.
      BZ     $BLN_RTRY            ; If the stop status is set,
                                  ; jumps to resume processing BLN_RTRY.
      CMP    A, #00H              ; Checks whether execution has been correctly
                                    completed.
      BNZ    $ERS_FALSE_END       ; Jumps to abnormal termination ERS_FALSE_END if
                                    execution has not been correctly completed.
      BR     ERS_TRUE_END

BLN_RTRY:
      ; Resume processing
      MOV    A, #0                ; Sets 0 as the bank number of the block to be
                                    blank-checked.
      MOV    B, #10               ; Sets 10 as the block number of the block to be
                                    blank-checked.
      DI                          ; Disables interrupts.
                                  ; Calls the block blank check library.
      CALL !_FlashBlockBlankCheck
      EI                          ; Enables interrupts.
      CMP    A, #1FH              ; Checks whether a stop status is set.
      BZ     $BLN_RTRY            ; If the stop status is set,
                                  ; retries the resume processing.
      CMP    A, #00H              ; Checks whether execution has been correctly
                                    completed.
      BNZ    $ERS_RTRY            ; Retries the main processing if execution has not been
                                    correctly completed.

      ; Clears the internal status of the stop processing
      MOVW   AX, #EntryRAM        ; Sets the first address of entry RAM.
      CALL   !_FlashEnv           ; Calls the initialize library.

ERS_TRUE_END:
      ; Normal completion

ERS_FALSE_END:
      ; Abnormal termination
```

Caution   It is assumed that the entry RAM has already been set.

3.4 Cautions

This section explains points to be noted during interrupt servicing.

- If processing related to self programming is performed or a setting related to it is changed during processing of an interrupt that has occurred during execution of self programming, then the operation is not guaranteed. Do not perform processing related to self programming and change settings related to it during interrupt servicing.

- Do not use register bank 3 during interrupt servicing, because self programming uses register bank 3.

- Save and restore registers used for interrupt servicing during interrupt servicing.

- If the set time of the watchdog timer is too short, processing of the set information library may not be completed. Therefore, do not set a time that is too short to the watchdog timer.
  If an interrupt successively occurs during a specific period while processing of the set information library is being executed, an infinite loop may occur if processing of the set information library is resumed after it has been stopped by the interrupt, because the processing is started from the beginning. Therefore, do not allow an interrupt to occur successively at an interval shorter than that within which processing of the set information library is to be completed.

  Remark Processing time of set information library (at 8 MHz)
  Min.: 108 milliseconds
  Max.: 696 milliseconds

- If multiple interrupts occur during execution of self programming, then the operation is not guaranteed. Disable the acknowledging of multiple interrupts during execution of self programming.

- If processing of the self programming sample library that was stopped by the occurrence of an interrupt is not resumed and processing of another block is to be performed, then the initialize library must be called before the processing of another block is started.

  Example To not resume erase processing of block 0 that was stopped and to execute erase processing of block 1, call the initialize library and then start the erase processing of block 1.

- Do not erase the entry RAM, stack, and data buffer until the series of processing tasks has been completed.

- Allocate an interrupt servicing program in an area other than that of the blocks to be rewritten, just as for the self programming program.

CHAPTER 4 BOOT SWAP FUNCTION

If rewriting of the vector table data, the basic functions of the program, or the self programming area fails because of a momentary power failure or the occurrence of a reset due to an external cause, then the data being rewritten is lost, the user program cannot be restarted by a reset, and rewriting can no longer be performed. This problem can be avoided by using a boot swap function through self programming.

The boot swap function is to replace boot program area, boot cluster 0[Note], with the boot swap target area, boot cluster 1[Note].

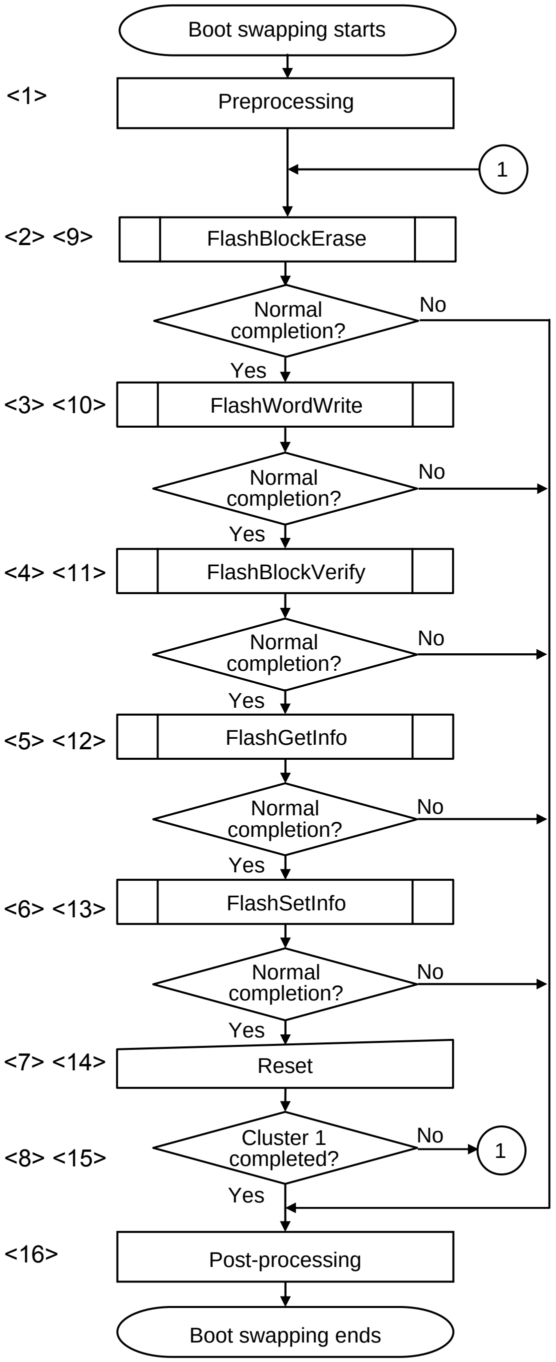
Before rewrite processing is started, a new boot program is written to boot cluster 1. This boot cluster 1 and boot cluster 0 are swapped and boot cluster 1 is used as a boot program area.

As a result, even if a power failure occurs while the boot program area is rewritten, the program is executed correctly because the next reset start program is booted from boot cluster 1. After that, boot cluster 0 can be erased or written as necessary.

Note Boot cluster 0 (0000H to 0FFFH): Original boot program area
Boot cluster 1 (1000H to 1FFFH): Boot swap target area

Figure 4-1 shows the flow of boot swapping by using the self programming sample library.

Figure 4-1. Flow of Boot Swapping



<1> Preprocessing

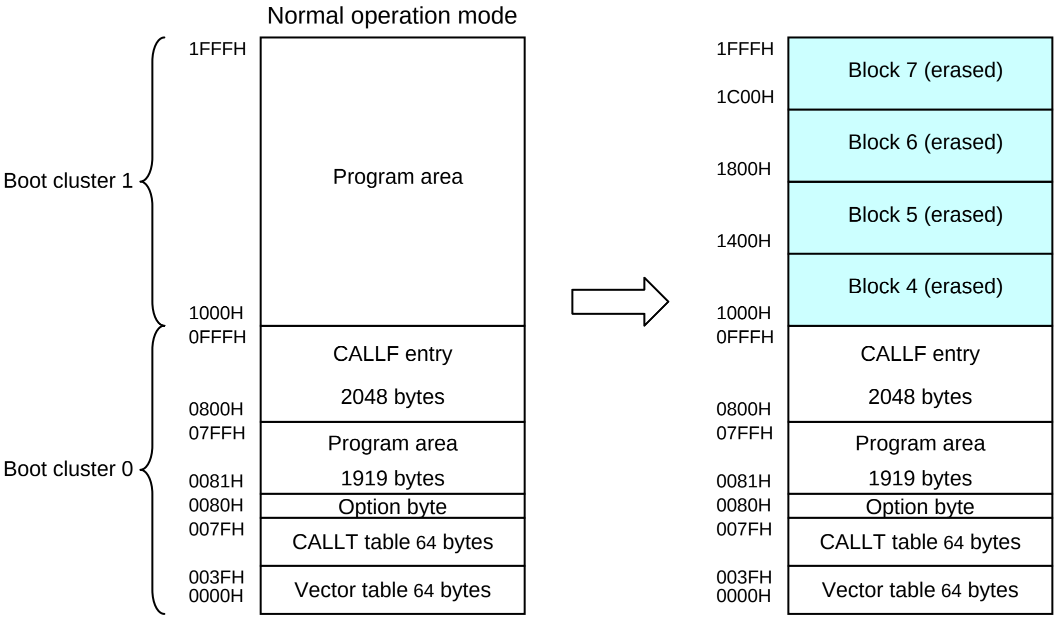
The following preprocessing of boot swapping is performed.

– Setting of hardware environment
– Declaring start of self programming
– Setting of software environment
– Initializing entry RAM
– Checking voltage level

<2> Erasing boot cluster 1

Blocks 4 to 7 are erased by calling the block erase library FlashBlockErase.

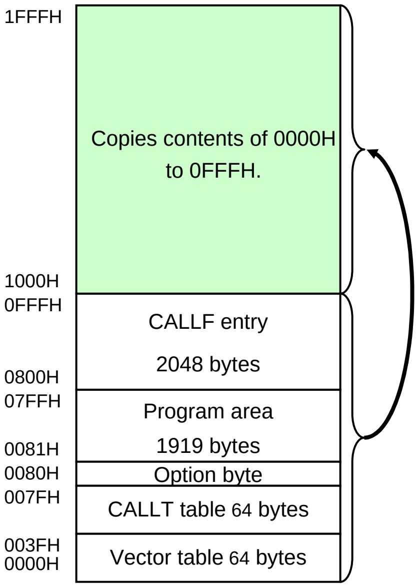
Remark The block erase library erases each block one by one.



<3> Copying boot cluster 0

The contents of 0000H to 0FFFH are written to 1000F to 1FFFH by calling the word write library FlashWordWrite.

Remark The word write library writes data in word units (256 bytes max.).



<4> Verifying boot cluster 1

Blocks 4 to 7 are verified by calling the block verify library FlashBlockVerify.

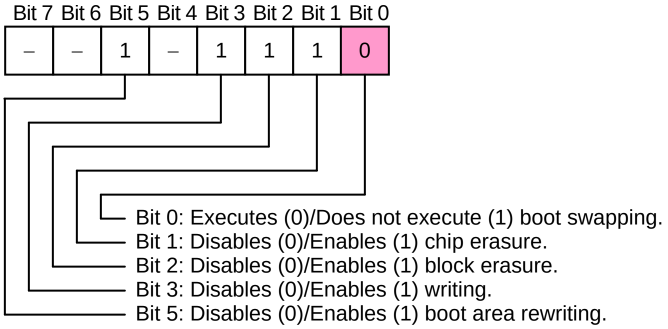
Remark The block verify library verifies each block one by one.

<5> Reading set status of boot swapping

The set status of boot swapping can be read by calling the get information library FlashGetInfo.

<6> Setting of boot swap bit

Set the boot swap bit to “execute boot swapping (0)” by calling the set information library FlashSetInfo.


Bit 7 Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 Bit 0
– – 1 – 1 1 1 0
Bit 0: Executes (0)/Does not execute (1) boot swapping.
Bit 1: Disables (0)/Enables (1) chip erasure.
Bit 2: Disables (0)/Enables (1) block erasure.
Bit 3: Disables (0)/Enables (1) writing.
Bit 5: Disables (0)/Enables (1) boot area rewriting.

<7> Occurrence of event

Boot cluster 1 is used as a boot program area when an external reset or overflow of the watchdog timer is generated.

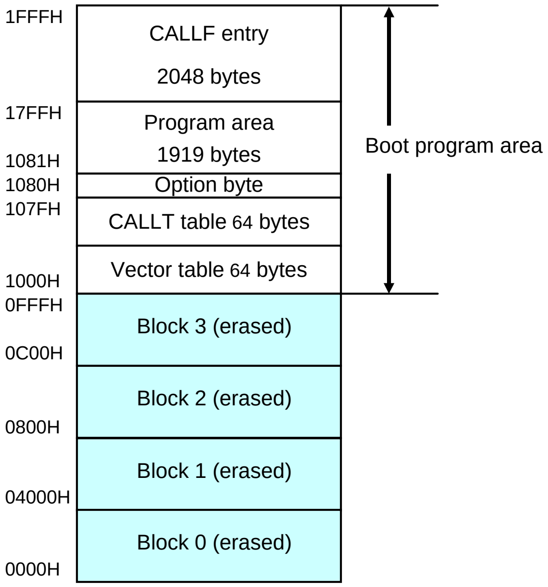
<8> End of swap processing (boot cluster 1)

Operations <2> to <7> complete the swap processing of boot cluster 1

<9> Erasing boot cluster 0

Blocks 0 to 3 are erased by calling the block erase library FlashBlockErase.

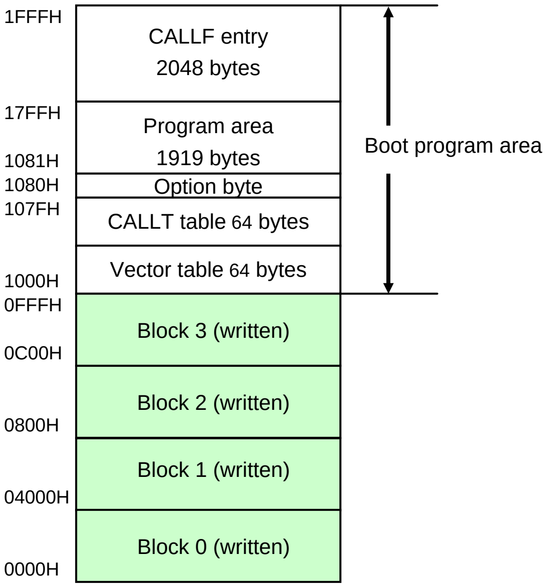
Remark The block erase library erases each block one by one.



<10> Writing new program to boot cluster 0

The contents of the new program are written to 0000H to 0FFFH by calling the word write library FlashWordWrite.

Remark The word write library writes the program in word units (256 bytes max.).



<11> Verifying boot cluster 0

Blocks 0 to 3 are verified by calling the block verify library FlashBlockVerify.

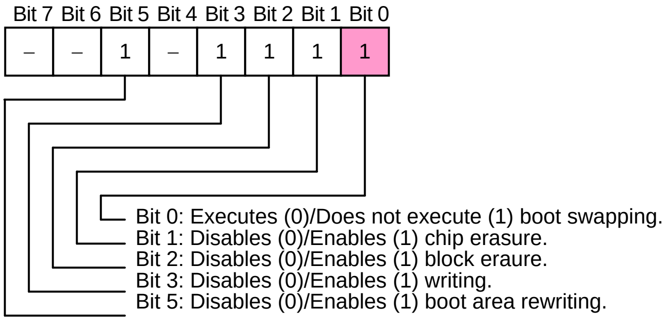
Remark The block verify library verifies each block one by one.

<12> Reading set status of boot swapping.

The set status of boot swapping is read by calling the get information library FlashGetInfo.

<13> Setting of boot swap bit

Set the boot swap bit to “not execute boot swapping (1)” by calling the set information library FlashSetInfo.


Bit 7
Bit 6
Bit 5
Bit 4
Bit 3
Bit 2
Bit 1
Bit 0
–
–
1
–
1
1
1
1
Bit 0: Executes (0)/Does not execute (1) boot swapping.
Bit 1: Disables (0)/Enables (1) chip erasure.
Bit 2: Disables (0)/Enables (1) block eraure.
Bit 3: Disables (0)/Enables (1) writing.
Bit 5: Disables (0)/Enables (1) boot area rewriting.

<14> Occurrence of event

Boot cluster 0 is used as a boot program area when an external reset or overflow of the watchdog timer is generated.

<15> End of swap processing (boot cluster 0)

Operations <9> to <14> complete the swap processing of boot cluster 0.

<16> Post-processing

As post-processing of boot swapping, the following is performed.

- Declaring end of self programming
- Setting of hardware environment

CHAPTER 5 SELF PROGRAMMING SAMPLE LIBRARY

This chapter explains details on the self programming sample library.
For the source program of each library, refer to **APPENDIX A SAMPLE PROGRAM**.

5.1 Type of Self Programming Sample Library

The self programming sample library consists of the following libraries.

Table 5-1. Self programming sample library List

Library Name	Call Example (C language)	Outline
	Call Example (assembly language)	
self programming start library	`FlashStart();`	Declares start of self programming.
	`CALL !_FlashStart`	
initialize library	`FlashEnv( &EntryRAM[0] );`	Initializes entry RAM.
	`CALL !_FlashEnv`	
mode check library	`Status = CheckFLMD( );`	Checks voltage level.
	`CALL !_CheckFLMD`	
block blank check library	`Status = FlashBlockBlankCheck(BlankCheckBANK, BlankCheckBlock );`	Checks erasing of specified block (1 KB).
	`CALL !_FlashBlockBlankCheck`	
block erase library	`Status = FlashBlockErase( EraseBANK, EraseBlock );`	Erases specified library (1 KB).
	`CALL !_FlashBlockErase`	
word write library	`Status = FlashWordWrite( &WordAddr, WordNumber, &DataBuffer );`	Writes 1- to 64-word data to specified address.
	`CALL !_FlashWordWrite`	
block verify library	`Status = FlashBlockVerfy( VerifyBANK, VerifyBlock );`	Verifies specified block (1 KB) (internal verification).
	`CALL !_FlashBlockVerify`	
self programming end library	`FlashEnd( );`	Declares end of self programming.
	`CALL !_FlashEnd`	
get information library	`Status = FlashGetInfo( &GetInfo, &DataBuffer );`	Reads flash information.
	`CALL !_FlashGetInfo`	
set information library	`Status = FlashSetInfo( SetInfoData );`	Changes setting of flash information.
	`CALL !_FlashSetInfo`	
EEPROM write library	`Status = FlashEEPROMWrite( &WordAdder, WordNumber, &DataBuffer );`	Writes 1- to 64-word data to specified address (during EEPROM emulation).
	`CALL !_EEPROMWrite`	

5.2 Explanation of Self Programming Sample Library

Each self programming sample library is explained in the following format.

self programming sample library name

[Outline]

Outlines the function of the self programming sample library.

[Format]

Indicates a format to call the self programming sample library from a user program described in C or an assembly language.

Caution In this manual, the data type name is defined as follows.

Definition Name	Data Type
UCHAR	unsigned char
USHORT	unsigned short

[Argument]

Indicates the argument of the self programming sample library.

[Return value]

Indicates the return value from the self programming sample library.

[Function]

Indicates the function details and points to be noted for the self programming sample library.

[Register status after calling]

Indicates the status of registers after the self programming sample library is called.

[Stack size]

Indicates the size of the stack used by the self programming sample library.

[ROM capacity]

Indicates the ROM capacity necessary for self programming.

[Call example]

Indicates an example of calling the self programming sample library from a user program described in C or an assembly language.

[Supplement]

Indicates supplementary information on a self programming sample library other than the above.

[Flow]

This indicates the program flow of the self programming sample library.

self programming start library

[Outline]

Declares the start of self programming.

[Format]

<C language>

```
void   FlashStart( void )
```

<Assembly language>

```
CALL   !_FlashStart
```

[Argument]

None

[Return value]

None

[Function]

This self programming sample library declares the start of self programming.

Therefore, call this library first as a self programming operation.

Caution The operation is not guaranteed if this library is called with interrupts enabled. Before calling this library, execute the DI instruction, and execute the EI instruction after execution of this library is completed, so that acknowledgment of an interrupt is disabled while this library is executed.

[Register status after calling]

No register is cleared.

[Stack size]

0 bytes

[ROM capacity]

12 bytes

[Call example]

<C language>

```
di();                      /* Disables interrupts.  */
FlashStart();              /* Calls self programming start library. */
ei();                      /* Enables interrupts. */
```

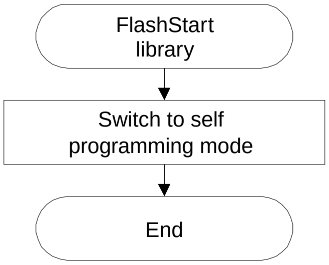
<Assembly language>

```
DI                         ; Disables interrupts.
CALL   !_FlashStart        ; Calls self programming start library.
EI                         ; Enables interrupts.
```

[Flow]

Figure 5-1 shows the flow of the self programming start library.

Figure 5-1. Flow of Self Programming Start Library



initialize library

[Outline]

Initializes entry RAM.

[Format]

<C language>

```
void   FlashEnv( USHORT EntryRAM )
```

<Assembly language>

```
CALL   !_FlashEnv
```

[Argument]

<C language>

Argument	Explanation
USHORT *EntryRAM*	First address of entry RAM[Note]

<Assembly language>

Argument	Explanation
AX	First address of entry RAM[Note]

Note For details on entry RAM, refer to 2.2.1 Entry RAM.

[Return value]

None

[Function]

This self programming sample library secures and initializes the entry RAM used for self programming.

As initialize processing, this library secures 100 bytes from an address specified by the parameter as a work area where the flash memory writing firmware operates, and sets the initial value to the first address +06H to +16H. The other areas are cleared to 0.

Remark Call this library after calling the self programming start library.
Also call this library to resume processing of a library executing self programming that was stopped by the occurrence of an interrupt.

[Register status after calling]

No register is cleared.

[Stack size]

30 bytes

[ROM capacity]

11 bytes

[Call example]

<C language>

```
USHORT EntryRAM;            /* Declares variable. */

FlashEnv( &EntryRAM[0] );   /* Calls initialize library. */
```

<Assembly language>

```
SELF_RAM      DSEG   AT     0FDBCH
EntryRAM: DS 100

SELF_PROG     CSEG
MOVW   AX, #EntryRAM        ; Sets first address of entry RAM.
CALL   !_FlashEnv           ; Calls initialize library.
```

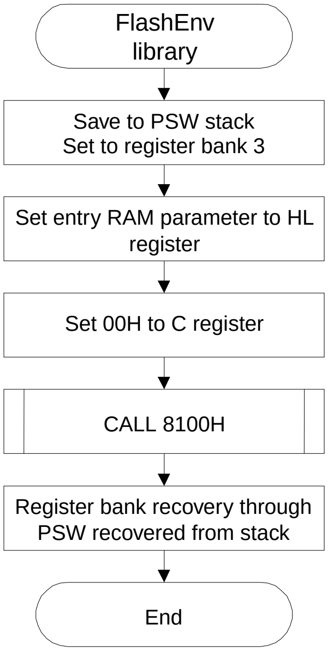
Caution Allocate the entry RAM at any address of the internal high-speed RAM outside of the short direct addressing range.

To allocate it in the internal high-speed RAM in the short direct addressing range, the first address is set to FE20H.

[Flow]

Figure 5-2 shows the flow of the initialize library.

Figure 5-2. Flow of Initialize Library


FlashEnv library
Save to PSW stack
Set to register bank 3
Set entry RAM parameter to HL register
Set 00H to C register
CALL 8100H
Register bank recovery through PSW recovered from stack
End

mode check library

[Outline]

Checks the voltage level.

[Format]

<C language>

```
UCHAR  CheckFLMD( void )
```

<Assembly language>

```
CALL   !_CheckFLMD
```

[Argument]

None

[Return value]

Status	Explanation
00H	Normal completion – FLMD0 pin is at high level.
01H	Abnormal termination – FLMD0 pin is at low level.

Remark The status is the UCHAR type in C and is stored in the A register in an assembly language.

[Function]

This library checks the voltage level (high or low) of the FLMD0 pin.

Remark Call this library after calling the self programming start library to check the voltage level of the FLMD0 pin.

Caution If the FLMD0 pin is at low level, operations such as erasing and writing the flash memory cannot be performed. To manipulate the flash memory by self programming, it is necessary to call this library and confirm that the FLMD0 pin is at high level.

[Register status after calling]

Memory Model	Register Status
Normal model	Registers cleared: A, BC Registers held: X, DE, HL
Static model	Registers cleared: A Registers held: X, BC, DE, HL

[Stack size]

28 bytes

[ROM capacity]

Memory Model	ROM Capacity
Normal model	14 bytes
Static model	11 bytes

[Call example]

<C language>

```
UCHAR  Status;              /* Declares variable.*/

Status = CheckFLMD();       /* Calls mode check library and */
                            /* stores status information. */
```

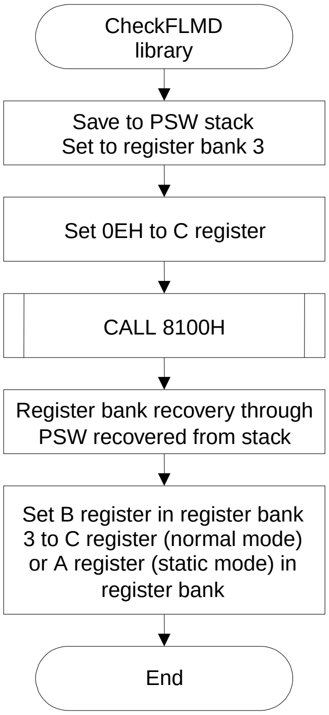
<Assembly language>

```
SELF_RAM     DSEG
Status: DS 1

SELF_PROG    CSEG

CALL   !_CheckFLMD          ; Calls mode check library.
MOV    !Status, A           ; Stores status information.
```

[Flow]

Figure 5-3 shows the flow of the mode check library.

Figure 5-3.　Flow of Mode Check Library


CheckFLMD library
Save to PSW stack
Set to register bank 3
Set 0EH to C register
CALL 8100H
Register bank recovery through PSW recovered from stack
Set B register in register bank 3 to C register (normal mode) or A register (static mode) in register bank
End

block blank check library

[Outline]

Checks erasing of a specified block (1 KB).

[Format]

<C language>

```
UCHAR  FlashBlockBlankCheck( UCHAR BlankCheckBANK, UCHAR BlankCheckBlock )
```

<Assembly language>

```
CALL   !_FlashBlockBlankCheck
```

[Argument]

<C language>

Argument	Explanation
UCHAR *BlankCheckBANK*	Bank number of block to be blank-checked.
UCHAR *BlankCheckBlock*	Block number of block to be blank-checked.

<Assembly language>

Argument	Explanation
A	Bank number of block to be blank-checked.
B	Block number of block to be blank-checked.

Remark Set the bank number to 0 when a product with which no bank number has to be set is used.

[Return value]

Status	Explanation
00H	Normal completion Specified block is blank (erase processing has been completed).
05H	Parameter error Specified bank number or block number is outside the settable range.
1BH	Blank check error Specified block is not blank (erase processing has not been completed).
1FH	Processing is stopped because an interrupt occurs. An interrupt occurs while processing of this library is under execution.

Remark The status is the UCHAR type in C and is stored in the A register in an assembly language.

[Function]

This library checks if a specified block (1 KB) has been erased.

Remark Because only one block is checked at a time, call this library as many times as required to check two or more blocks.

Caution The operation is not guaranteed if this library is called with interrupts enabled. Before calling this library, execute the DI instruction, and execute the EI instruction after execution of this library is completed, so that acknowledgment of an interrupt is disabled while this library is executed.

[Register status after calling]

Memory Model	Register Status
Normal model	Registers cleared: AX, BC Registers held: DE, HL
Static model	Registers cleared: A, BC Registers held: X, DE, HL

[Stack size]

Memory Model	Stack Size
Normal model	37 bytes
Static model	35 bytes

[ROM capacity]

Memory Model	ROM Capacity
Normal model	67 bytes (of which 30 bytes are common routine)
Static model	54 bytes (of which 30 bytes are common routine)

[Call example]

<C language>

```
UCHAR  Status;             /* Declares variable. */
UCHAR  BlankCheckBANK;     /* Declares variable. */
UCHAR  BlankCheckBlock;    /* Declares variable. */

BlankCheckBANK = 0;        /* Sets bank number of block to be blank-checked to 0. */
BlankCheckBlock = 10;      /* Sets block number of block to be blank-checked to 10. */

                           /* Calls block blank check library and */
                           /* stores status information.*/
di();                      /* Disables interrupts.  */
Status = FlashBlockBlankCheck ( BlankCheckBANK, BlankCheckBlock );
ei();                      /* Enables interrupts. */
```

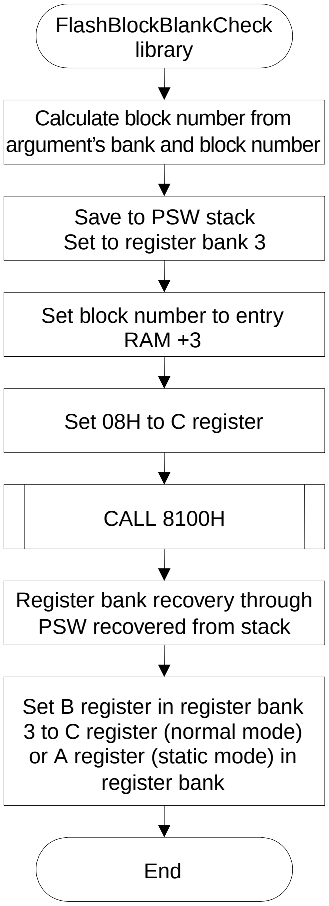
<Assembly language>

```
SELF_RAM      DSEG
Status: DS 1

SELF_PROG     CSEG
MOV    A, #0               ; Sets bank number of block to be blank-checked to 0.
MOV    B, #10              ; Sets block number of block to be blank-checked to 10.
                           ; Calls block blank check library.
DI                         ; Disables interrupts.
CALL   !_FlashBlockBlankCheck
MOV    !Status, A          ; Stores status information.
EI                         ; Enables interrupts.
```

[Flow]

Figure 5-4 shows the flow of the block blank check library.

Figure 5-4. Flow of Block Blank Check Library


FlashBlockBlankCheck library
Calculate block number from argument's bank and block number
Save to PSW stack
Set to register bank 3
Set block number to entry RAM +3
Set 08H to C register
CALL 8100H
Register bank recovery through PSW recovered from stack
Set B register in register bank 3 to C register (normal mode) or A register (static mode) in register bank
End

block erase library

[Outline]

Erases a specified block (1 KB).

[Format]

<C language>

```
UCHAR  FlashBlockErase( UCHAR EraseBANK, UCHAR EraseBlock )
```

<Assembly language>

```
CALL   !_FlashBlockErase
```

[Argument]

<C language>

Argument	Explanation
UCHAR *EraseBANK*	Bank number of block to be erased
UCHAR *EraseBlock*	Block number of block to be erased.

<Assembly language>

Argument	Explanation
A	Bank number of block to be erased
B	Block number of block to be erased.

Remark Set the bank number to 0 when a product with which no bank number has to be set is used.

[Return value]

Status	Explanation
00H	Normal completion
05H	Parameter error Specified bank number or block number is outside the settable range.
10H	Protect error Specified block is included in the boot area and rewriting the boot area is disabled.
1AH	Erase error An error occurred during processing of this library.
1FH	Processing is stopped by the occurrence of an interrupt. An interrupt occurred while processing of this library was under execution.

Remark The status is the UCHAR type in C and is stored in the A register in an assembly language.

[Function]

This library erases a specified block (1 KB).

Remark Because only one block is erased at a time, call this library as many times as required to erase two or more blocks.

Caution The operation is not guaranteed if this library is called with interrupts enabled. Before calling this library, execute the DI instruction, and execute the EI instruction after execution of this library is completed, so that acknowledgment of an interrupt is disabled while this library is executed.

[Register status after calling]

Memory Model	Register Status
Normal model	Registers cleared: AX, BC Registers held: DE, HL
Static model	Registers cleared: A, BC Registers held: X, DE, HL

[Stack size]

Memory Model	Stack Size
Normal model	39 bytes
Static model	37 bytes

[ROM capacity]

Memory Model	ROM Capacity
Normal model	67 bytes (of which 30 bytes are common routine)
Static model	54 bytes (of which 30 bytes are common routine)

[Call example]

<C language>

```
UCHAR  Status;                    /* Declares variable. */
UCHAR  EraseBANK;                 /* Declares variable. */
UCHAR  EraseBlock;                /* Declares variable. */

EraseBANK = 0;                    /* Sets bank number of block to be erased to 0. */
EraseBlock = 10;                  /* Sets block number of block to be erased to 10. */

di();                             /* Disables interrupts.  */
                                  /* Calls block erase library and stores status */
                                  /* information. */
Status = FlashBlockErase( EraseBANK, EraseBlock );
ei();                             /* Enables interrupts. */
```

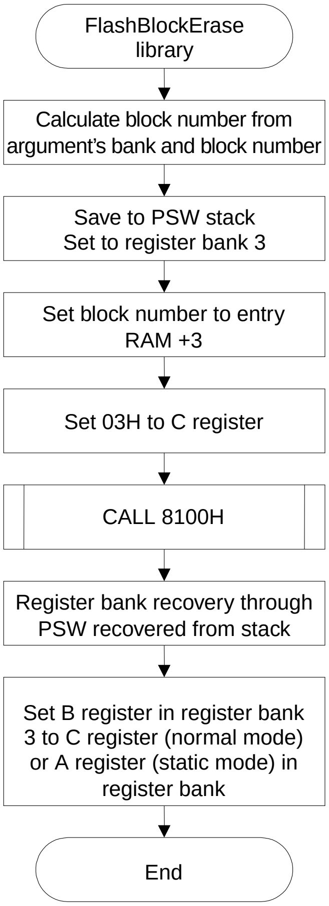
<Assembly language>

```
SELF_RAM      DSEG
Status: DS 1

SELF_PROG     CSEG
MOV    A, #0                      ; Sets bank number of block to be erased to 0.
MOV    B, #10                     ; Sets block number of block to be erased to 10.
DI                                ; Disables interrupts.
CALL !_FlashBlockErase            ; Calls block erase library.
MOV    !Status, A                 ; Stores status information.
EI                                ; Enables interrupts.
```

[Flow]

Figure 5-5 shows the flow of the block erase library.

Figure 5-5. Flow of Block Erase Library


FlashBlockErase library
Calculate block number from argument's bank and block number
Save to PSW stack
Set to register bank 3
Set block number to entry RAM +3
Set 03H to C register
CALL 8100H
Register bank recovery through PSW recovered from stack
Set B register in register bank 3 to C register (normal mode) or A register (static mode) in register bank
End

word write library

[Outline]

Writes 1- to 64-word data to specified addresses.

[Format]

<C language>

```
UCHAR  FlashWordWrite( struct stWordAddress *ptr, UCHAR WordNumber,
                           USHORT DataBufferAddress )
```

<Assembly language>

```
CALL   !_FlashWordWrite
```

[Argument]

<C language>

Argument	Explanation
struct stWordAddress **ptr*	First address of write start address structure (stWordAddress)[Note 1]. This structure must be 3 bytes in size and at a 4-byte boundary and must be secured by the user.
UCHAR *WordNumber*	Number of data to be written (1 to 64)
USHORT *DataBufferAddress*	First address of write data buffer[Note 2]

<Assembly language>

Argument	Explanation
AX	First address of data having structure same as that of write start address structure[Note 1] in C (Refer to **APPENDIX A SAMPLE PROGRAM**.)
B	Number of data to be written (1 to 64)
HL	First address of write data buffer[Note 2]

Notes 1. Write start address structure

```
struct stWordAddress{
       USHORT WriteAddress;       /* Write start address*/
       UCHAR  WriteBank;          /* Bank number of write start address*/
};
```

Remarks 1. Specify the write start address as a multiple of 4 bytes.

2. Set the bank number to 0 when a product with which no bank number has to be set is used.

Caution Before calling this library, set a value to each member of this structure.

2. Before calling this library, set write data to the write data buffer (whose first address is indicated by DataBufferAddress).

Caution Set the write start address and the number of data to be written so that they do not straddle over the boundary of each block.

[Return value]

Status	Explanation
00H	Normal completion
05H	Parameter error – Start address not is a multiple of 1 word (4 bytes). – The number of data to be written is 0. – The number of data to be written exceeds 64 words. – Write end address (Start address + (Number of data to be written × 4 bytes)) exceeds the flash memory area.
10H	Protect error – Specified range includes the boot area and rewriting the boot area is disabled.
1CH	Write error – Data is verified but does not match after execution of the processing of this library.
1FH	Processing is stopped by the occurrence of an interrupt. – An interrupt occurred while processing of this library was under execution.

Remark The status is the UCHAR type in C and is stored in the A register in an assembly language.

[Function]

This library writes the specified number of data from a specified address.
Set a RAM area containing the data to be written as a data buffer and call this library.
Data of up to 256 bytes can be written (in 4-byte units) at one time.

Remark Call this library as many times as required to write data of more than 256 bytes.

Cautions 1. After writing data, execute verification (internal verification) of the block including the range in which the data has been written. If verification is not executed, the written data is not guaranteed.

2. The operation is not guaranteed if this library is called with interrupts enabled. Before calling this library, execute the DI instruction, and execute the EI instruction after execution of this library is completed, so that acknowledgment of an interrupt is disabled while this library is executed.

[Register status after calling]

Memory Model	Register Status
Normal model	Registers cleared: AX, BC, DE Registers held: HL
Static model	Registers cleared: AX, C Registers held: B, DE, HL

[Stack size]

39 bytes

[ROM capacity]

Memory Model	ROM Capacity
Normal model	117 bytes (of which 57 bytes are common routine)
Static model	100 bytes (of which 57 bytes are common routine)

[Call example]

<C language>

```
struct stWordAddress WordAddr;   /* Declares variable. */
UCHAR  DataBuffer[4];            /* Declares variable. */
UCHAR  WordNumber;               /* Declares variable. */
UCHAR  Status;                   /* Declares variable. */

DataBuffer[0] = 0x11;            /* Sets data to be written. */
DataBuffer[1] = 0x22;            /* Sets data to be written. */
DataBuffer[2] = 0x33;            /* Sets data to be written. */
DataBuffer[3] = 0x44;            /* Sets data to be written. */
WordNumber = 1;                  /* Sets number of data to be written. */
WordAddr.WriteAddress = 0xA000;  /* Sets 0xA000H as write start address. */
WordAddr.WriteBANK = 0;          /* Sets bank number of write start address to 0. */

di();                            /* Disables interrupts.  */
                                 /* Calls word write library and stores status */
                                 /* information. */
Status = FlashWordWrite( &WordAddr, WordNumber, &DataBuffer );
ei();                            /* Enables interrupts. */
```

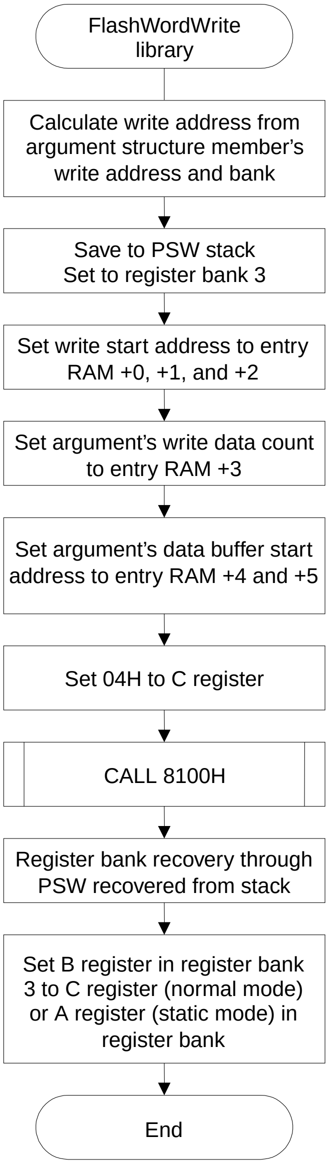
<Assembly language>

```
SELF_RAM      DSEG
DataBuffer: DS 4
WordAddr:
WriteAddress: DS 2
WriteBank: DS 1
Status: DS 1

SELF_PROG     CSEG

MOV    A, #11H
MOV    !DataBuffer, A             ; Sets data to be written.
MOV    A, #22H
MOV    !DataBuffer+1, A           ; Sets data to be written.
MOV    A, #33H
MOV    !DataBuffer+2, A           ; Sets data to be written.
MOV    A, #44H
MOV    !DataBuffer+3, A           ; Sets data to be written.

MOVW   AX, #0A000H
MOVW   !WriteAddress, AX          ; Sets 0xA000H as write start address.

MOV    A, #0
MOV    !WriteBANK, A              ; Sets bank number of write start address to 0.

MOVW   AX, #WordAddr              ; Sets first address of write start address
structure.
MOV    B, #1                      ; Sets number of data to be written.
MOVW   HL, #DataBuffer            ; First address of write data buffer

DI                                ; Disables interrupts.
CALL   !_FlashWordWrite           ; Calls word write library.
MOV    !Status, A                 ; Stores status information.
EI                                ; Enables interrupts.
```

[Flow]

Figure 5-6 shows the flow of the word write library.

Figure 5-6. Flow of Word Write Library


FlashWordWrite library
Calculate write address from argument structure member's write address and bank
Save to PSW stack
Set to register bank 3
Set write start address to entry RAM +0, +1, and +2
Set argument's write data count to entry RAM +3
Set argument's data buffer start address to entry RAM +4 and +5
Set 04H to C register
CALL 8100H
Register bank recovery through PSW recovered from stack
Set B register in register bank 3 to C register (normal mode) or A register (static mode) in register bank
End

block verify library

[Outline]

Verifies (internal verification) a specified block (1 KB).

Caution Verification (internal verification) is a function to check if data written to the flash memory is written at a sufficient level, and is different from verification that compares data.

[Format]

<C language>

```
UCHAR  FlashBlockVerify( UCHAR VerifyBANK, UCHAR VerifyBlock )
```

<Assembly language>

```
CALL   !_FlashBlockVerify
```

[Argument]

<C language>

Argument	Explanation
UCHAR *VerifyBANK*	Bank number of block to be verified
UCHAR *VerifyBlock*	Block number to be verified

<Assembly language>

Argument	Explanation
A	Bank number of block to be verified
B	Block number to be verified

Remark Set the bank number to 0 when a product with which no bank number has to be set is used.

[Return value]

Status	Explanation
00H	Normal completion
05H	Parameter error Specified bank number or block number is outside the settable range.
1BH	Verify (internal verify) error An error occurs during processing of this library.
1FH	Processing is stopped by the occurrence of an interrupt. An interrupt occurred while processing of this library was under execution.

Remark The status is the UCHAR type in C and is stored in the A register in an assembly language.

[Function]

This library verifies (internal verification) a specified block (1 KB).

Remark Call this library as many times as required to verify two or more blocks, because only one block is verified at a time.

Cautions 1. After writing data, verify (internal verification) the block including the range in which the data has been written. If verification is not executed, the written data is not guaranteed.

2. The operation is not guaranteed if this library is called with interrupts enabled. Before calling this library, execute the DI instruction, and execute the EI instruction after execution of this library is completed, so that acknowledgment of an interrupt is disabled while this library is executed.

[Register status after calling]

Memory Model	Register Status
Normal model	Registers cleared: AX, BC Registers held: DE, HL
Static model	Registers cleared: A, BC Registers held: X, DE, HL

[Stack size]

Memory Model	Stack Size
Normal model	37 bytes
Static model	35 bytes

[ROM capacity]

Memory Model	ROM Capacity
Normal model	67 bytes (of which 30 bytes are common routine)
Static model	54 bytes (of which 30 bytes are common routine)

[Call example]

<C language>

```
UCHAR  Status;               /* Declares variable. */
UCHAR  VerifyBANK;           /* Declares variable. */
UCHAR  VerifyBlock;          /* Declares variable. */

VerifyBANK = 0;              /* Sets bank number of block to be verified to 0. */
VerifyBlock = 10;            /* Sets block number of block to be verified to 10. */

di();                        /* Disables interrupts.  */
                             /* Calls block verify library and stores */
                             /* status information. */
Status = FlashBlockVerify( VerifyBANK, VerifyBlock );
ei();                        /* Enables interrupts. */
```

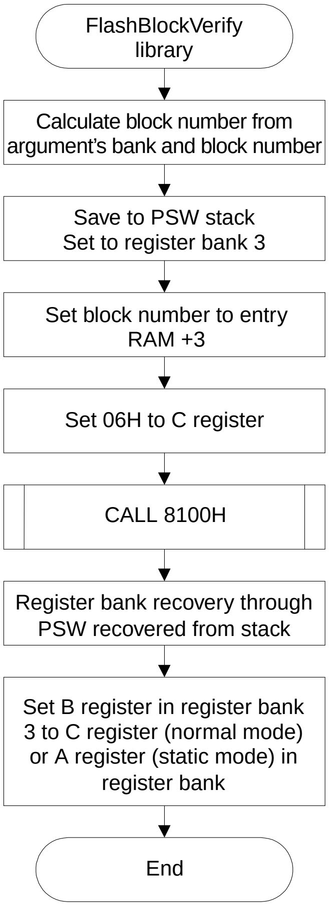
<Assembly language>

```
SELF_RAM     DSEG
Status: DS 1

SELF_PROG    CSEG
MOV    A, #0                      ; Sets bank number of block to be verified to 0.
MOV    B, #10                     ; Sets block number of block to be verified to 10.
DI                                ; Disables interrupts.
CALL   !_FlashBlockVerify         ; Calls block verify library.
MOV    !Status, A                 ; Stores status information.
EI                                ; Enables interrupts.
```

[Flow]

Figure 5-7 shows the flow of the block verify library.

Figure 5-7. Flow of Block Verify Library


FlashBlockVerify library
Calculate block number from argument's bank and block number
Save to PSW stack
Set to register bank 3
Set block number to entry RAM +3
Set 06H to C register
CALL 8100H
Register bank recovery through PSW recovered from stack
Set B register in register bank 3 to C register (normal mode) or A register (static mode) in register bank
End

self programming end library

[Outline]

Declares the end of self programming.

[Format]

<C language>

```
void   FlashEnd( void )
```

<Assembly language>

```
CALL   !_FlashEnd
```

[Argument]

None

[Return value]

None

[Function]

This library declares the end of self programming.

It completes writing to the flash memory and restores the normal operation mode.

Remarks 1. Call this library at the end of the self programming operation.

2. After execution of this library is completed, the level of the FLMD0 pin is returned to low.

Caution **The operation is not guaranteed if this library is called with interrupts enabled. Before calling this library, execute the DI instruction, and execute the EI instruction after execution of this library is completed, so that acknowledgment of an interrupt is disabled while this library is executed**

[Register status after calling]

No register is cleared.

[Stack size]

0 bytes

[ROM capacity]

12 bytes

[Call example]

<C language>

```
di();                      /* Disables interrupts.  */
FlashEnd();                /* Calls self programming end library. */
ei();                      /* Enables interrupts. */
```

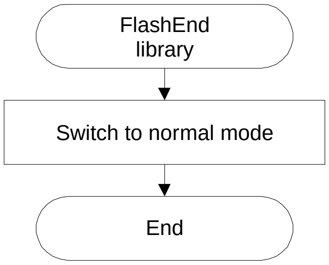
<Assembly language>

```
DI                         ; Disables interrupts.
CALL   !_FlashEnd          ; Calls self programming end library.
EI                         ; Enables interrupts.
```

[Flow]

Figure 5-8 shows the flow of the self programming end library.

Figure 5-8. Flow of Self Programming End Library



get information library

[Outline]

Reads flash information.

[Format]

<C language>

```
UCHAR  FlashGetInfo( struct stGetInfo *ptr, USHORT DataBufferAddress )
```

<Assembly language>

```
CALL   !_FlashGetInfo
```

[Argument]

<C language>

Argument	Explanation
struct stGetInfo **ptr*	First address of flash information acquisition structure (stGetInfo)[Note]. This structure is 3 bytes in size and must be secured by the user.
USHORT *DataBufferAddress*	First address of acquired data storage buffer

<Assembly language>

Argument	Explanation
AX	First address of data having the same structure as flash information acquisition structure in C[Note] (Refer to **APPENDIX A SAMPLE PROGRAM**.)
BC	First address of acquired data storage buffer

Note Flash information acquisition structure

```
Struct stGetInfo{
       UCHAR  OptionNumber;   /* Option value[Note] */
       UCHAR  GetInfoBank;    /* Bank number (valid if option value is 05H) */
       UCHAR  GetInfoBlock;   /* Block number (valid if option value is 05H) */
};
```

Note Refer to **[Supplement]**.

Remark Set the bank number to 0 when a product with which no bank number has to be set is used.

Cautions 1. Setting of a bank number and a block number is invalid when security flag information and boot flag information are checked.

2. Before calling this library, set a value to each member of this structure.

[Return value]

Status	Explanation
00H	Normal completion
05H	Parameter error - Specified option value is outside the settable range.
20H	Read error - Security flag is read twice and different data are read when the option value is set to 03H.

Remark The status is the UCHAR type in C and is stored in the A register in an assembly language.

Caution Flash information corresponding to a specified option value is stored in the data buffer. For details on the flash information, refer to [Supplement].

[Function]

This library reads flash information.

It is used to check the set information (security flag, boot flag information, and last address of a specified block) of the flash memory.

Caution The operation is not guaranteed if this library is called with interrupts enabled. Before calling this library, execute the DI instruction, and execute the EI instruction after execution of this library is completed, so that acknowledgment of an interrupt is disabled while this library is executed.

[Register status after calling]

Memory Model	Register Status
Normal model	Registers cleared: AX, BC, DE Registers held: HL
Static model	Registers cleared: AX, BC, HL Registers held: DE

[Stack size]

38 bytes

[ROM capacity]

Memory Model	ROM Capacity
Normal model	161 bytes (of which 30 bytes are common routine)
Static model	148 bytes (of which 30 bytes are common routine)

[Call example]

<C language>

```
Struct stGetInfo GetInfo;    /* Declares variable. */
UCHAR  DataBuffer[3];        /* Declares variable. */
UCHAR  Status;               /* Declares variable. */

GetInfo.OptionNumber = 5;    /* Specifies option value to "get last address */
                             /* of specified block". */
GetInfo.GetInfoBank = 0;     /* Sets bank number of block whose flash */
                             /* information is to be acquired to 0. */
GetInfo.GetInfoBlock = 10;   /* Sets block number of block whose flash  */
                             /* information is to be acquired to 10. */

di();                        /* Disables interrupts.  */
                             /* Calls get information library and stores status */
                             /* information. */
Status = FlashGetInfo( &GetInfo, &DataBuffer );
ei();                        /* Enables interrupts. */
```

<Assembly language>

```
SELF_RAM     DSEG
DataBuffer: DS 3
GetInfo:
OptionNumber: DS 1
GetInfoBank: DS 1
GetInfoBlock: DS 1
Status: DS 1

SELF_PROG    CSEG

MOV    A, #5
MOV    OptionNumber, A       ; Specifies option value to "get last address of
MOV    A, #0                 ; specified block".
MOV    GetInfoBank, A        ; Sets bank number of block whose flash
MOV    A, #10                ; information is to be acquired to 0.
MOV    GetIngoBlock, A       ; Sets block number of block whose flash
                             ; information is to be acquired to 10.

MOVW   AX, #GetInfo
MOVW   BC, #DataBuffer
DI                           ; Disables interrupts.
CALL   !_FlashGetInfo        ; Calls get information library.
MOV    !Status, A            ; Stores status information.
EI                           ; Enables interrupts.
```

[Supplement]

The flash information that can be acquired differs depending on the option value specified by the flash information acquisition structure.

The information corresponding to each option value is shown below.

Option Value	Information Acquired
03H	Security flag information (2 bytes)
04H	Boot flag information (1 byte)
05H	Last address of specified block (3 bytes)

Each piece of information is detailed below.

(1) Security flag information (option value: 03H)

The setting status of the security flag is stored as data of 2 bytes in the data buffer from its beginning.

Offset	Contents
+0	Security flag information [Note]
+1	Last block number of boot area (fixed to 03H)

Note Details on security flag information

Security Flag	Contents
Bit 0	Chip erase enable flag 0: Disabled 1: Enabled
Bit 1	Block erase enable flag 0: Disabled 1: Enabled
Bit 2	Write enable flag 0: Disabled 1: Enabled
Bit 4	Boot area rewrite disable flag 0: Disabled 1: Enabled
Other than above	Always 1

(2) Boot flag information (option value: 04H)

The boot flag information (setting status of boot swapping) is stored in the data buffer as data of 1 byte.

Offset	Contents
+0	Boot flag information[Note]

Note Details on boot flag information

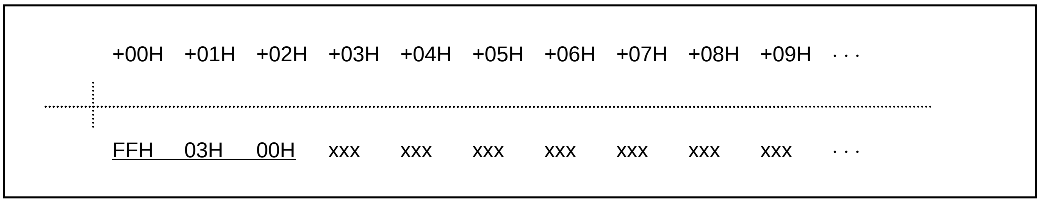
Offset	Contents
00H	Boot areas are not swapped. (Reset and started from address 0000H)
01H	Boot areas are swapped. (Reset and started from address 1000H)

(3) Last address of specified block (option value: 05H)

The last address of the specified block is stored in the data buffer from its beginning as data of 3 bytes.

Offset	Contents
+0	Block last address (Low)
+1	Block last address (High)
+2	Bank number

Example Where the last address for block of block number 00H is 0003FFH

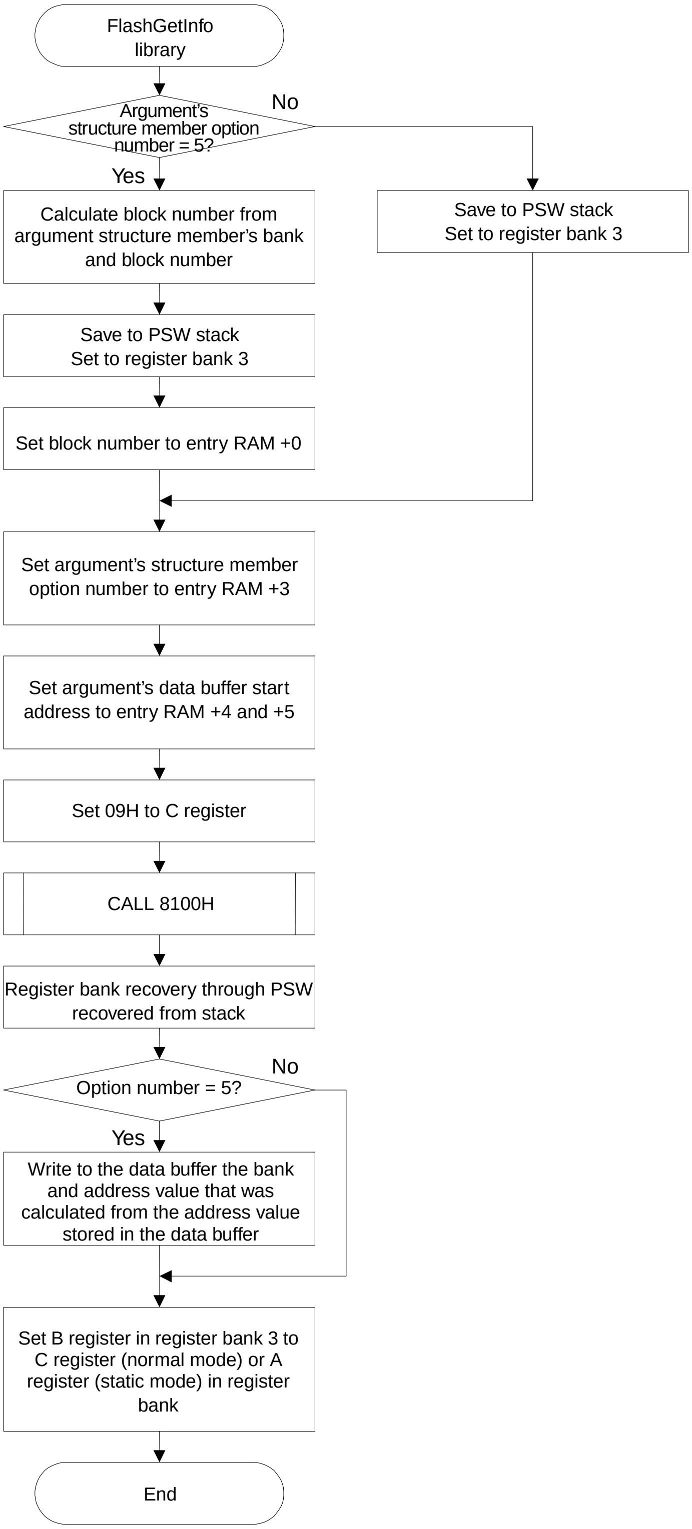


+00H	+01H	+02H	+03H	+04H	+05H	+06H	+07H	+08H	+09H	· · ·
FFH	03H	00H	xxx	xxx	xxx	xxx	xxx	xxx	xxx	· · ·

[Flow]

Figure 5-9 shows the flow of the get information library.

Figure 5-9. Flow of Get Information Library


FlashGetInfo library
Argument's structure member option number = 5?
No
Yes
Calculate block number from argument structure member's bank and block number
Save to PSW stack Set to register bank 3
Save to PSW stack Set to register bank 3
Set block number to entry RAM +0
Set argument's structure member option number to entry RAM +3
Set argument's data buffer start address to entry RAM +4 and +5
Set 09H to C register
CALL 8100H
Register bank recovery through PSW recovered from stack
Option number = 5?
No
Yes
Write to the data buffer the bank and address value that was calculated from the address value stored in the data buffer
Set B register in register bank 3 to C register (normal mode) or A register (static mode) in register bank
End

set information library

[Outline]

Changes setting of flash information.

[Format]

<C language>

```
UCHAR  FlashSetInfo( UCHAR SetInfoData )
```

<Assembly language>

```
CALL   !_FlashSetInfo
```

[Argument]

<C language>

Argument	Explanation
UCHAR *SetInfoData*	Flash information data[Note]

<Assembly language>

Argument	Explanation
A	Flash information data[Note]

Note Details on flash information data

Flash Information Data	Contents
Bit 0	0: Swaps boot areas. 1: Does not swap boot areas.
Bit 1	0: Disables chip erasure. 1: Enables chip erasure.
Bit 2	0: Disables block erasure. 1: Enables block erasure.
Bit 3	0: Disables writing. 1: Enables writing.
Bit 5	0: Disables writing boot area. 1: Enables writing boot area.
Other than above	Always 1

[Return value]

Status	Explanation
00H	Normal completion
05H	Parameter error Bit 0 of the information flag value was cleared to 0 for a product that does not support boot swapping.
10H	Protect error - Attempt was made to enable a flag that has already been disabled. - Attempt was made to change the boot area swap flag while rewriting of the boot area was disabled.
1AH	Erase error - An erase error occurred while processing of this library was under execution.
1BH	Verify (internal verify) error - A verify error occurred while processing of this library was under execution.
1CH	Write error - A write error occurred while processing of this library was under execution.
1FH	Processing is stopped by the occurrence of an interrupt. - An interrupt occurred while processing of this library was under execution.

Remark The status is the UCHAR type in C and is stored in the A register in an assembly language.

[Function]

This library changes the setting of the flash information.

It is used to change the set information (security flag and boot flag information) of the flash memory.

Cautions 1. A flag that has already disabled processing cannot be changed to enable the processing.

2. The operation is not guaranteed if this library is called with interrupts enabled. Before calling this library, execute the DI instruction, and execute the EI instruction after execution of this library is completed, so that acknowledgment of an interrupt is disabled while this library is executed.

[Register status after calling]

Memory Model	Register Status
Normal model	Registers cleared: A, BC Registers held: X, DE, HL
Static model	Registers cleared: A Registers held: X, BC, DE, HL

[Stack size]

37 bytes

[ROM capacity]

Memory Model	ROM Capacity
Normal model	27 bytes
Static model	23 bytes

[Call example]

<C language>

```
UCHAR  Status;              /* Declares variable. */
UCHAR  SetInfoData;         /* Declares variable. */

SetInfoData = 0b11111101;  /* Sets flash information data to “disable chip erase”.*/

di();                       /* Disables interrupts.  */
                            /* Calls set information library and stores status */
                            /* information. */
Status = FlashSetInfo( SetInfoData );
ei();                       /* Enables interrupts. */
```

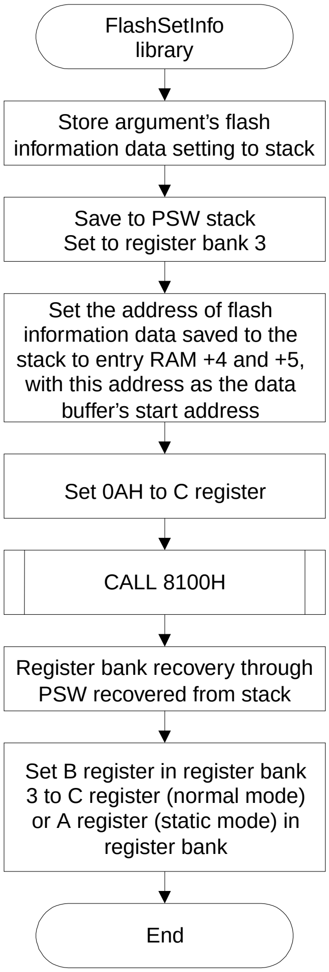
<Assembly language>

```
SELF_RAM     DSEG
Status: DS 1

SELF_PROG    CSEG

MOV    A, #11111101B       ; Sets flash information data to “disable chip erase”.
DI                         ; Disables interrupts.
CALL   !_FlashSetInfo      ; Calls set information library.
MOV    !Status, A          ; Stores status information.
EI                         ; Enables interrupts.
```

[Flow]

Figure 5-10 shows the flow of the set information library.

Figure 5-10. Flow of Set Information Library


FlashSetInfo library
Store argument's flash information data setting to stack
Save to PSW stack
Set to register bank 3
Set the address of flash information data saved to the stack to entry RAM +4 and +5, with this address as the data buffer's start address
Set 0AH to C register
CALL 8100H
Register bank recovery through PSW recovered from stack
Set B register in register bank 3 to C register (normal mode) or A register (static mode) in register bank
End

EEPROM write library

[Outline]

Writes 1 to 64 word data to a specified address (during EEPROM emulation).

[Format]

<C language>

```
UCHAR  EEPROMWrite( struct stWordAddress *ptr, UCHAR WordNumber,
                            USHORT DataBufferAddress )
```

<Assembly language>

```
CALL !_EEPROMWrite
```

[Argument]

<C language>

Argument	Explanation
struct stWordAddress *ptr*	First address of write start address structure (stWordAddress)[Note 1]. This structure must be 3 bytes in size and at a 4-byte boundary, and must be secured by the user.
UCHAR *WordNumber*	Number of data to be written (1 to 64)
USHORT *DataBufferAddress*	First address of write data buffer[Note 2]

<Assembly language>

Argument	Explanation
AX	First address of data having the same structure as the write start address structure[Note 1] in C (Refer to **APPENDIX A SAMPLE PROGRAM**.)
B	Number of data to be written (1 to 64)
HL	First address of write data buffer[Note 2]

Notes 1. Write start address structure

```
Struct stWordAddress{
       USHORT WriteAddress;        /* Write start address*/
       UCHAR  WriteBANK;           /* Bank number of write start address */
};
```

Remarks 1. Set the write start address as a multiple of 4 bytes.

2. Set the bank number to 0 when a product with which no bank number has to be set is used.

Caution Set a value to each member of this structure before calling this library.

2. Set write data to the write data buffer (first address indicated by *DataBufferAddress*) before calling this library.

Caution Set the write start address and the number of data to be written so that they do not straddle over the boundary of each block.

[Return value]

Status	Explanation
00H	Normal completion
05H	Parameter error – Start address is not a multiple of 1 word (4 bytes). – The number of data to be written is 0. – The number of data to be written exceeds 64 words. – Write end address (Start address + (Number of data to be written x 4 bytes)) exceeds the flash memory area.
10H	Protect error – A boot area is included in the specified range and rewriting of the boot area is disabled.
1CH	Write error – Data cannot be written correctly.
1DH	Verify (MRG12) error – Data is verified but does not match after it has been written.
1EH	Blank error – Area equal to the number of data to be written was not a vacant area.
1FH	Processing is stopped by the occurrence of an interrupt. – An interrupt occurred while processing of this library was under execution.

Remark The status is the UCHAR type in C and is stored in the A register in an assembly language.

[Function]

This library writes the specified number of data to the flash memory starting from a specified address during EEPROM emulation. Set a RAM area storing the data to be written as a data buffer and call this library.
Data of up to 256 bytes can be written (in 4-byte units) at one time.

Remark Call this library as many times as required to write data of more than 256 bytes.

Caution The operation is not guaranteed if this library is called with interrupts enabled. Before calling this library, execute the DI instruction, and execute the EI instruction after execution of this library is completed, so that acknowledgment of an interrupt is disabled while this library is executed.

[Register status after calling]

Memory Model	Register Status
Normal model	Registers cleared: AX, BC, DE Registers held: HL
Static model	Registers cleared: AX, C Registers held: B, DE, HL

[Stack size]

36 bytes

[ROM capacity]

Memory Model	ROM Capacity
Normal model	117 bytes (of which 57 bytes are common routine)
Static model	100 bytes (of which 57 bytes are common routine)

[Call example]

<C language>

```
Struct stWordAddress WordAddr;    /* Declares variable. */
UCHAR  DataBuffer[4];             /* Declares variable. */
UCHAR  WordNumber;                /* Declares variable. */
UCHAR  Status;                    /* Declares variable. */

DataBuffer[0] = 0x11;             /* Sets data to be written. */
DataBuffer[1] = 0x22; ;           /* Sets data to be written. */
DataBuffer[2] = 0x33; ;           /* Sets data to be written. */
DataBuffer[3] = 0x44; ;           /* Sets data to be written. */
WordNumber = 1;                   /* Sets number of data to be written. */
WordAddr.WriteAddress = 0xA000;   /* Sets 0xA000 to write start address.* /
WordAddr.WriteBANK = 0;           /* Sets bank number of write start address to 0. */

di();                             /* Disables interrupts.  */
                                  /* Calls EEPROM write library and stores status */
                                  /* information.*/
Status = EEPROMWrite( &WordAddr, WordNumber, &DataBuffer );
ei();                             /* Enables interrupts. */
```

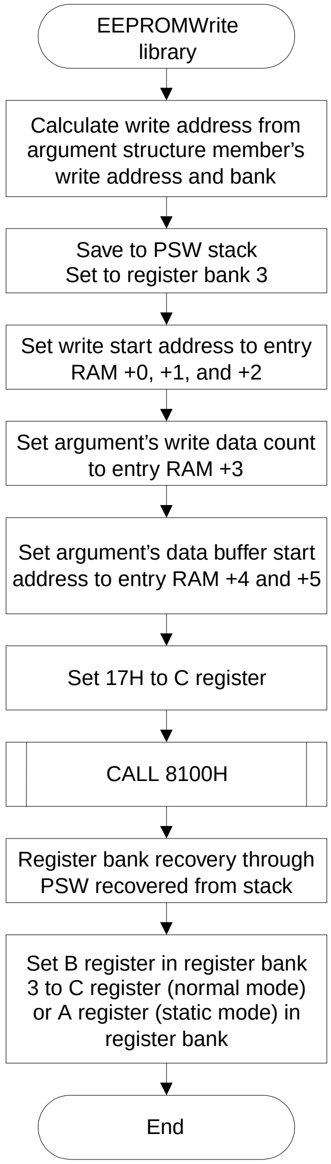
<Assembly language>

```
SELF_RAM     DSEG
DataBuffer: DS 4
WordAddr:
WriteAddress: DS 2
WriteBank: DS 1
Status: DS 1

SELF_PROG    CSEG

MOV    A, #11H
MOV    !DataBuffer, A            ; Sets data to be written.
MOV    A, #22H
MOV    !DataBuffer+1, A          ; Sets data to be written.
MOV    A, #33H
MOV    !DataBuffer+2, A          ; Sets data to be written.
MOV    A, #44H
MOV    !DataBuffer+3, A          ; Sets data to be written.

MOVW   AX, #0A000H
MOVW   !WriteAddress, AX         ; Sets A000H to write address.

MOV    A, #0
MOV    !WriteBANK, A             ; Sets bank number of write start address to 0.

MOVW   AX, #WordAddr             ; Sets address of write start address structure.
MOV    B, #4                     ; Sets number of data to be written.
MOVW   HL, #DataBuffer           ; Sets first address of write data buffer.

DI                               ; Disables interrupts.
CALL   !_EEPROMWrite             ; Calls EEPROM write library.
MOV    !Status, A                ; Stores status information.
EI                               ; Enables interrupts.
```

[Flow]

Figure 5-11 shows the flow of the EEPROM write library.

Figure 5-11. Flow of EEPROM Write Library


EEPROMWrite library
Calculate write address from argument structure member’s write address and bank
Save to PSW stack
Set to register bank 3
Set write start address to entry RAM +0, +1, and +2
Set argument’s write data count to entry RAM +3
Set argument’s data buffer start address to entry RAM +4 and +5
Set 17H to C register
CALL 8100H
Register bank recovery through PSW recovered from stack
Set B register in register bank 3 to C register (normal mode) or A register (static mode) in register bank
End

CHAPTER 6 DETAILS OF SELF PROGRAMMING CONTROL

This chapter describes the registers that are used to control flash memory access, and the entry RAM.

6.1 Registers That Control Self Programming

6.1.1 Flash programming mode control register (FLPMC)

This register is used to enable/disable flash memory access (write, erase, etc.), and indicate the self programming operation mode.

A particular sequence must be used when writing to this register, in order to prevent inadvertent settings due to noise or manipulation errors. For the specific sequence, refer to **6.1.2 Flash protect command register (PFCMD)**.

After reset: 08H R/W[Note]

Symbol	7	6	5	4	3	2	1	0
FLPMC	0	0	0	0	FWEDIS	FWEPR	FLSPM1	FLSPM0

Note Bit 2 is a read-only bit.

[FWEDIS]

This flag is used to control flash memory access (write, erase, etc.) enable/disable through software. The initial value of this flag is 1, and flash memory access is enabled by writing 0 to this flag.

FWEDIS	Function
0	Enable write/erase
1	Disable write/erase

[FWEPR]

This flag is used to control flash memory access (write, erase, etc.) enable/disable through hardware. It directly reflects the voltage of the FLMD0 pin.

FLMD0 Pin Voltage	FWEPR[Note]	Function
Low level (V_{SS})	0	Disable write/erase
High level (V_{DD})	1	Enable write/erase

Note The FWEPR bit is a read-only bit. Its value cannot be changed by software. However, when using ICE, the value can be changed even by overwriting.

Flash memory access can be enabled through the combination of FWEDIS and FWEPR.

FWEDIS	FWEPR	Flash Memory Write/Erase Enable
0	1	Enable write/erase
Other than above		Disable write/erase

Cautions 1. When executing flash memory write/erase, be sure to set FWEDIS to 0.

2. In the normal mode, be sure to set FWEDIS to 1.

[FLSPM0 and FLSPM1]

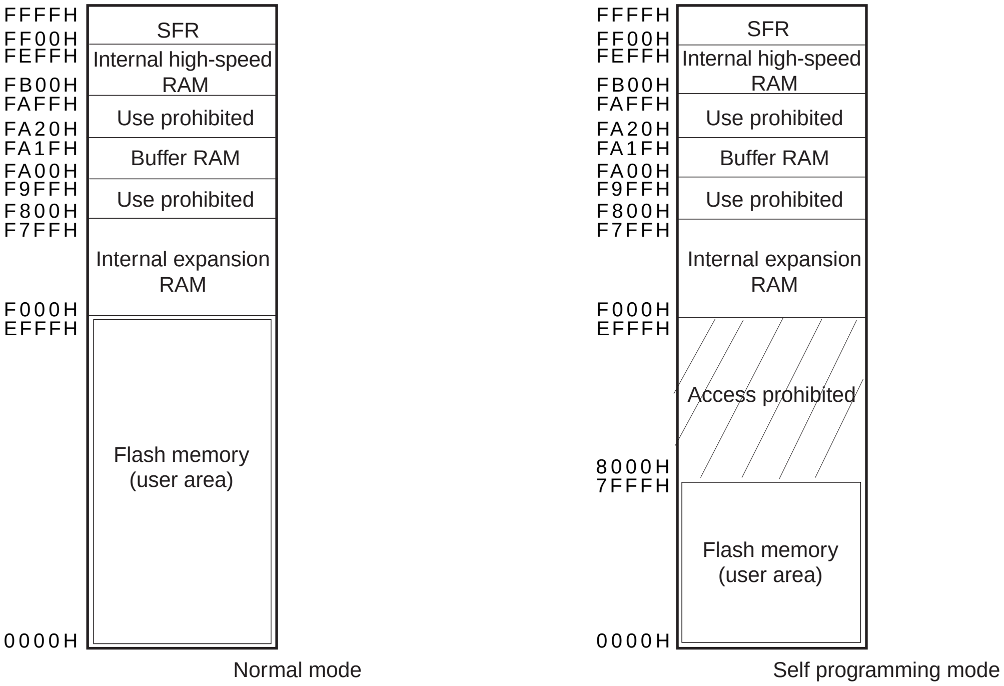
These control flags are used to select the self programming operation mode.

FLSPM1	FLSPM0	Mode Selection
0	0	Normal mode • Access (instruction fetch, data read) to the entire address range of flash memory is possible.
0	1	Self programming mode • Self programming by “CALL #8100H” is possible. • Access (instruction fetch, data read) to flash memory (in products with 32 KB or more of ROM, 0000H to 7FFFH) is possible.

Caution Setting FLSPM1, FLSPM0 = 1, 0 or 1, 1 is prohibited.

Figure 6-1 shows the self programming operation mode and memory map.

Figure 6-1. Self Programming Operation Mode and Memory Map (μPD78F0545)



Caution Place the program that controls the self programming in the address range of 0000H to 7FFFH.

6.1.2 Flash protect command register (PFCMD)

To prevent erroneous flash memory write or erase caused by an inadvertent program loop, etc., protection is implemented by this register for flash programming mode control register (FLPMC) write.

The FLPMC register is a special register that is valid for write operations only when the write operations are performed via following special sequence.

<1> Write a specified value (= A5H) to the PFCMD register.
<2> Write the value to be set to the FLPMC register (writing is invalid at this step).
<3> Write the inverted value of the value to be set to the FLPMC register (writing is invalid at this step).
<4> Write the value to be set to the FLPMC register (writing is valid at this step).

Caution The above sequence must be executed every time the value of the FLPMC register is changed.

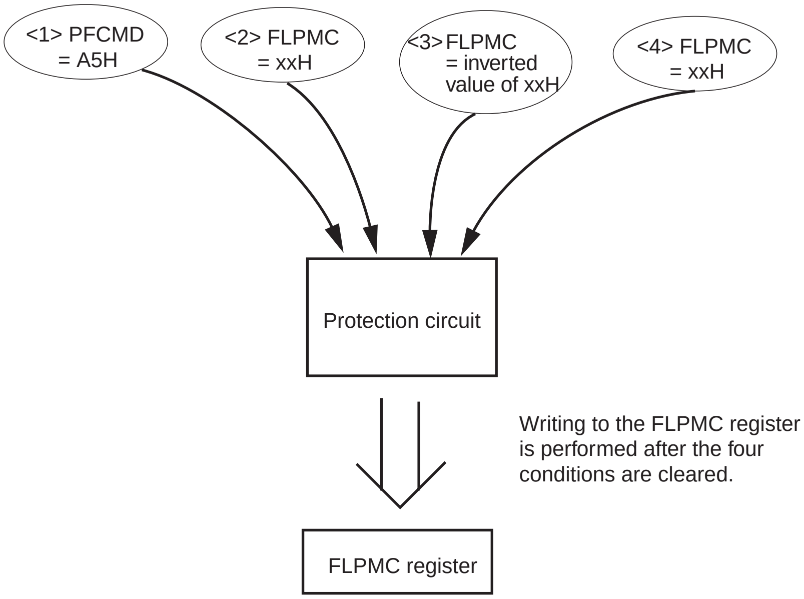
After reset: Undefined W

Symbol	7	6	5	4	3	2	1	0
PFCMD	REG7	REG6	REG5	REG4	REG3	REG2	REG1	REG0

<Coding example of special sequence>
When writing 05H to FLPMC register:

```
MOV PFCMD, #0A5H   ; Writes A5H to PFCMD
MOV FLPMC, #05H    ; Writes 05H to FLPMC
MOV FLPMC, #0FAH   ; Writes 0FAH (inverted value of 05H) to FLPMC
MOV FLPMC, #05H    ; Writes 05H to FLPMC
```

Figure 6-2. Write Protection


<1> PFCMD = A5H
<2> FLPMC = xxH
<3> FLPMC = inverted value of xxH
<4> FLPMC = xxH
Protection circuit
Writing to the FLPMC register is performed after the four conditions are cleared.
FLPMC register

6.1.3 Flash status register (PFS)

If the flash programming mode control register (FLPMC) is not written in the correct sequence, the FLPMC register is not set and a protection error occurs. At this time, bit 0 (FPRERR) of the PFS register is set to 1.

This flag is a cumulative flag.

After reset: 00H R/W

Symbol	7	6	5	4	3	2	1	0
PFS	0	0	0	0	0	0	0	FPRERR

The FPRERR flag's operation conditions are as follows.

<Setting conditions>

- When the PFCMD register is written to at a time when the store instruction's operation for the latest peripheral register was not a write operation to the PFCMD register using a specified value (A5H)
- When the first store instruction operation after <1> above is for a peripheral register other than the FLPMC register
- When the first store instruction operation after <2> above is for a peripheral register other than the FLPMC register
- When the first store instruction operation after <2> above writes a value other than the inverted value of the value to be set to the FLPMC register
- When the first store instruction operation after <3> above is for a peripheral register other than the FLPMC register
- When the first store instruction operation after <3> above writes a value other than the value (write value in <2>) to be set to the FLPMC register.

Remark The numbers shown in angle brackets above correspond to the numbers shown in angle brackets in section 6.1.2 above.

<Reset conditions>

- When 0 is written to bit 0 (FPRERR) in the PFS register.
- When a system reset is performed.

6.1.4 Self programming control parameters

The self programming operation includes setting the FLMD0 pin to 1, setting the required values to the FLPMC register, and setting up entry RAM, after which the function number (refer to Table 6-1) is set to register bank 3’s C register and CALL8100H processing is performed.

The parameters involved in this operation are described below.

(1) Register bank 3’s parameters

In the self programming sample library, register bank 3’s C register is used to select functions to control self programming, while its B register is used to store execution results and the HL register is used to specify the start address of entry RAM.

Since settings to register bank 3 are all performed within a library, register bank 3 should be included in user programs.

Table 6-1. Register Bank 3 Parameter List

Register / Function Name	C register Function Number	B Register Return Value	HL Register	AX/DE Register
Initialize	00H	00H: Normal completion	Start address of entry RAM[Note]	Not used (used by self programming sample library)
Block erase	03H	00H: Normal completion 05H: Parameter error 10H: Protect error 1AH: Erase error 1FH: Stopped		
Word write	04H	00H: Normal completion 05H: Parameter error 10H: Protect error 1CH: Write error 1FH: Stopped		
Block verify	06H	00H: Normal completion 05H: Parameter error 1BH: Verify (internal verify) error 1FH: Stopped		
Block blank check	08H	00H: Normal completion 05H: Parameter error 1BH: Blank check error 1FH: Stopped		
Get information	09H	00H: Normal completion 05H: Parameter error 20H: Read error		
Set information	0AH	00H: Normal completion 05H: Parameter error 10H: Protect error 1AH: Erase error 1BH: Verify (internal verify) error 1CH: Write error 1FH: Stopped		
Mode check	0EH	00H: Normal completion 01H: Error		
EEPROM write	17H	00H: Normal completion 05H: Parameter error 10H: Protect error 1CH: Write error 1DH: Verify (MRG12) error 1EH: Blank error 1FH: Stopped		

Note Entry RAM can be allocated to any address in the internal high-speed RAM except in the short direct addressing area (entry RAM can be allocated to addresses in the internal high-speed RAM within the short direct addressing area only when the start address is FE20H).

(2) Entry RAM

Entry RAM is a 100-byte RAM area that is used for self programming. Parameters that control self programming are set six bytes from the start address of the entry RAM area. Once these parameters have been set, the self programming sample library is called to begin controlling self programming operations. The placement of the parameters for various functions relative to the start of the entry RAM area is listed in Table 6-2 below.

Allocate the Entry RAM to any address in the high-speed RAM area except in the short direct addressing area (it is possible to allocate the entry RAM to addresses in internal high-speed RAM within the short direct addressing area only when the start address is FE20H).

Entry RAM is used as a work area for self programming. Consequently, nothing in the entry RAM area except for parameters should be changed during self programming operations.

Table 6-2. Entry RAM Parameter List

Offset Value Function Name	+00H	+01H	+02H	+03H	+04H, +05H	+06H to +99H
Initialize	–	–	–	–	–	–
Block erase	–	–	–	Block number	–	–
Word write	Start address lower bits	Start address higher bits	Start address MSB	Number of words	Data buffer start address	–
Block verify	–	–	–	Block number	–	–
Block blank check	–	–	–	Block number	–	–
Get information	Block number	–	–	Option value	Data buffer start address	–
Set information	–	–	–	–	Data buffer start address	–
Mode check	–	–	–	–	–	–
EEPROM write	Start address lower bits	Start address higher bits	Start address MSB	Number of words	Data buffer start address	–

Remark Do not modify the content of any single description.

(3) Data buffer

The data buffer is used to pass data and setting-related information written to flash memory; its specific contents depend on the self programming function being used. The data buffer can be placed at any address in internal high-speed RAM, and its start address is specified in the entry RAM. The data buffer's size also depends on the function, but it must be in range from 1 to 256 bytes.

Table 6-3. Data Buffer Parameter List

Function	Data Buffer Size (Bytes)	Data Buffer Contents					
			+00H	+01H	+02H	+03H	+04H to +FFH
Initialize	–	–	Not used				
Block erase	–	–	Not used				
Word write	4 to 256	Write data	Write data				
Block verify	–	–	Not used				
Block blank check	–	–	Not used				
Get information	1 to 8	Flash information	Flash information (refer to Table 6-4 for details)				
Set information	1	Information flag	Bit 0: Execute boot swap (0)/ Do not execute (1) Bit 1: Prohibit chip erase (0)/Enable (1) Bit 2: Prohibit block erase (0)/Enable (1) Bit 3: Prohibit write (0)/Enable (1) Bit 5: Prohibit boot area overwrite (0)/ Enable (1)	Not used			
Mode check	–	–	Not used				
EEPROM write	4 to 256	Write data	Write data				

Remark If a function is used with an area marked as "not used", the area cannot be used as a data buffer.

Table 6-4. Detailed Flash Information for Get Information Function

Flash Information Type	Option Value	Data Buffer's Offset Value							
		+00H	+01H	+02H	+03H	+04H	+05H	+06H	+07H
Security flag	03H	Security flag information	Boot area's final block number	Not used					
		<Security flag information: Details> Bit 1: Chip erase enable flag (0: Prohibit, 1: Enable) Bit 2: Block erase enable flag (0: Prohibit, 1: Enable) Bit 3: Write enable flag (0: Prohibit, 1: Enable) Bit 4: Boot area overwrite prohibit flag (0: Prohibit, 1: Enable) Bits 3, 5, 6, and 7 are always 1. <Boot area's final block number> 03H (fixed)							
Boot flag	04H	Boot flag information	Not used						
		<Boot flag information: details> 00H: Boot area is not being switched 01H: Boot area is being switched							
Last address of specified block	05H	Block's end address			Not used				
		Lower bits	Higher bits	MSB					

Remark If a function is used with an area marked as "not used", the area cannot be used as a data buffer.

APPENDIX A   SAMPLE PROGRAM

This appendix shows the sample program provided.

Caution This sample program must be used at the user's own risk. Correct operation is not guaranteed if this sample program is used.

A.1 User Program

<sample.c>

```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*  System           : Sample program that uses self programming sample library
*  File name        : sample.c
*  Target CPU       : 78K0/Kx2
*  Last updated     : 2005/02/25
*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/*---------------------------------------------------------------------
*      Expanded functions
*--------------------------------------------------------------------*/
#pragma      sfr
#pragma      DI
#pragma      EI
#pragma      NOP

/*---------------------------------------------------------------------
*      Type declarations
*--------------------------------------------------------------------*/
typedef unsigned char      UCHAR;
typedef unsigned short     USHORT;

/*---------------------------------------------------------------------
*      Constant definitions
*--------------------------------------------------------------------*/
#define      STATE_OF_ABORT             (0x1F)       /* State of abort */
#define      FLASHFIRM_NORMAL_END       (0x00)       /* Normal completion */
#define      FLASHFIRM_ABNORMAL_END     (0xFF)       /* Abnormal completion */

#define      TRUE                       (0x00)       /* Normal */
#define      FALSE                      (0xFF)       /* Abnormal */
#define      PARAMETER_ERROR            (0x05)       /* Parameter error */

#define      BANKNUM                    (5)          /* Bank number */
#define      BLOCK                      (32)         /* Block number */
#define      ADDR                       (0x8000)     /* Write Address */

struct stWordAddress{
       USHORT WriteAddress;
       UCHAR  WriteBank;
};

struct stWordWriteData{
       UCHAR  WordNumber;
       UCHAR  WriteDataBuffer[256];
       USHORT WriteAddressData;
       UCHAR  WriteBankData;
};

struct stGetInfo{
       UCHAR  OptionNumber;
       UCHAR  GetInfoBank;
       UCHAR  GetInfoBlock;
};

struct stGetInfoData{
       UCHAR  OptionNumberData;
       UCHAR  GetInfoBankData;
       UCHAR  GetInfoBlockData;
};


/*-----------------------------------------------------------------------
*      Prototype declarations
*---------------------------------------------------------------------*/
extern void   FlashStart( void );
extern void   FlashEnd( void );
extern void   FlashEnv( USHORT EntryRAM );
extern UCHAR  FlashBlockErase( UCHAR EraseBank, UCHAR EraseBlock );
extern UCHAR  FlashWordWrite( struct stWordAddress *ptr, UCHAR WordNumber, USHORT
DataBufferAddress );
extern UCHAR  FlashBlockVerify( UCHAR VerifyBank, UCHAR VerifyBlock );
extern UCHAR  FlashBlockBlankCheck( UCHAR BlankCheckBank, UCHAR BlankCheckBlock );
extern UCHAR  FlashGetInfo( struct stGetInfo *ptr, USHORT DataBufferAddress );
extern UCHAR  FlashSetInfo( UCHAR SetInfoData );
extern UCHAR  CheckFLMD( void );

UCHAR  FlashBlockErase_Call( UCHAR EraseBank, UCHAR EraseBlock );
UCHAR  FlashWordWrite_Call( struct stWordAddress *ptr1, USHORT DataBufferAddress,
struct stWordWriteData *ptr2 );
UCHAR  FlashBlockVerify_Call( UCHAR VerifyBank, UCHAR VerifyBlock );
UCHAR  FlashBlockBlankCheck_Call( UCHAR BlankCheckBank, UCHAR BlankCheckBlock );
UCHAR  FlashGetInfo_Call( struct stGetInfo *ptr1, USHORT DataBufferAddress, struct
stGetInfoData *ptr2 );
UCHAR  FlashGetInfo_Call5( struct stGetInfo *ptr1, USHORT DataBufferAddress, struct
stGetInfoData *ptr2 );
UCHAR  FlashSetInfo_Call3( UCHAR SetInfoData, struct stGetInfo *ptr, USHORT
DataBufferAddress );
UCHAR  FlashSetInfo_Call4( UCHAR SetInfoData, struct stGetInfo *ptr, USHORT
DataBufferAddress );
UCHAR  CheckFLMD_Call( void );


/* FlashEnv */
sreg   UCHAR  EntryRAM[100];


/*-------------------------------------------------------------------------
*      Sample program
*------------------------------------------------------------------------*/
void main( void ){
       USHORT i;
       UCHAR  Status;

       /* FlashBlockErase */
       UCHAR  EraseBank;
       UCHAR  EraseBlock;

       /* FlashBlockVerify */
       UCHAR  VerifyBank;
       UCHAR  VerifyBlock;

       /* FlashBlockBlankCheck */
       UCHAR  BlankCheckBank;
       UCHAR  BlankCheckBlock;

       /* FlashWordWrite */
       struct stWordAddress WordAddr;
       UCHAR  DataBuffer[256];
       struct stWordWriteData WordWriteData;

       DI();

       IMS = 0xCC;
       IXS = 0x00;

       PCC    = 0x00;              /* Clock select( division ratio ) */
                                   /* -> IN: 5MHZ = OUT: 5MHZ */

       MSTOP = 0;                  /* MOC.bit7:X1 oscillator operation */
       OSTS = 0x05;                /* Oscillation stabilization time */
       while( OSTC.0 == 0);
       MCM0 = 0;
       XSEL = 0;
       while( MCS == 1 );
```

```
        RSTOP = 1;

        LVIM = 0x00;                    /* Prohibits low voltage detection */
        LVIS = 0x00;

        EI();

        /* FlashStart( Self programming start library ) call processing */
        FlashStart();

        /* FlashEnv( Initialization library ) call processing */
        FlashEnv( ( USHORT )&EntryRAM );

        /* CheckFLMD( Mode check library ) call processing */
        Status = CheckFLMD_Call();

        if( Status == TRUE ){
                while(1){
                        /* FlashBlockBlankCheck call processing */
                        BlankCheckBank = BANKNUM;
                        BlankCheckBlock = BLOCK;
                                                        /* Block blank check library */
                        Status = FlashBlockBlankCheck_Call( BlankCheckBank,
BlankCheckBlock );

                        if( Status == TRUE ){
                                break;
                        }else if ( Status == PARAMETER_ERROR ){
                                break;
                                /* Abnormal end */
                        }else{
                                /* FlashBlockErase call processing */
                                EraseBank = BANKNUM;
                                EraseBlock = BLOCK;
                                                        /* Block erase library */
                                Status = FlashBlockErase_Call( EraseBank, EraseBlock );

                                if( Status != TRUE ){
                                        break;
                                        /* Abnormal end */
                                }
                        }
                }
        }

        if( Status == TRUE ){
                /* FlashWordWrite call processing */
                for( i=0; i<=255; i++ ){
                     WordWriteData.WriteDataBuffer[i] = ( UCHAR )i;
              }
              WordWriteData.WordNumber = 64;
              WordWriteData.WriteAddressData = ADDR;
              WordWriteData.WriteBankData = BANKNUM;
              Status = FlashWordWrite_Call( &WordAddr, ( USHORT )&DataBuffer,
&WordWriteData );                              /* Word write library */

              if( Status == TRUE ){
                     /* FlashBlockVerify call processing */
                     VerifyBank = BANKNUM;
                     VerifyBlock = BLOCK;
                                               /* Block verify library */
                     Status = FlashBlockVerify_Call( VerifyBank, VerifyBlock );
              }
       }

       if( Status == TRUE ){
              /* FlashEnd( Self programming end library ) call processing */
              FlashEnd();
              /* Normal end */
       }else{
              /* FlashEnd( Self programming end library ) call processing */
              FlashEnd();
              /* Abnormal end */
       }


       while( 1 ){
              NOP();
              NOP();
       }
}

/* Call processing in each library */

/*-------------------------------------------------------------------
* Function name :   FlashBlockErase_CALL
* Input :           EraseBank = Erase bank
*                   EraseBlock = Erase block number
* Output :          Status = Return value from firm
*                   ( When the retry time exceeds 10 times,
*                   return PARAMETER ERROR from this function )
* Summary :         FlashBlockErase library call processing.
*-------------------------------------------------------------------*/
UCHAR  FlashBlockErase_Call( UCHAR EraseBank, UCHAR EraseBlock ){
       UCHAR  Status;
       UCHAR  Counter;
       Counter = 0;                            /* Retry counter reset */

       while( 1 ){
              Counter++;   /* When the retry time exceeds 10 times, it ends. */
              if( Counter >= 10 ){
                     Status = FLASHFIRM_ABNORMAL_END;
                     break;
              }

              DI();

                                               /* Erase library call */
              Status = FlashBlockErase( EraseBank, EraseBlock );

              EI();

              if( Status == STATE_OF_ABORT ){  /* State of abort?, YES */
                     while( 1 ){
                            DI();

                                               /* Block blank check library call */
                            Status = FlashBlockBlankCheck( EraseBank, EraseBlock );

                            EI();

                                               /* State of abort?, NO */
                            if( Status != STATE_OF_ABORT ){
                                   break;
                            }
                     }

                                               /* Normal completion?, YES */
                     if( Status == FLASHFIRM_NORMAL_END ){
                                               /* Initialization library call */
                            FlashEnv( ( USHORT )&EntryRAM );
                            break;
                     }
              }else{
                     break;
              }
       }

       return(Status);                         /* Return value = Status */
}

/*---------------------------------------------------------------------
* Function name :   FlashWordWrite_CALL
* Input :           *ptr1 = Address of writing beginning address structure
*                   DataBufferAddress = Address in writing data buffer
*                   ptr2 = Address of writing beginning address structure
*                   ( Member of structure ...  Number of writing data
*                                              Writing starting address
*                                              Bank of writing starting address )
* Output :          Status = Return value from firm
* Summary :         WordWrite library call processing.
*-------------------------------------------------------------------*/
UCHAR  FlashWordWrite_Call( struct stWordAddress *ptr1, USHORT DataBufferAddress,
struct stWordWriteData *ptr2 ){
       UCHAR  Status;
       USHORT i;
       UCHAR  *p;

       p = ( UCHAR * )DataBufferAddress;

                                        /* Writing data setting to data buffer. */
       for( i=0; i<=(ptr2->WordNumber)*4-1; i++){
             *p = ptr2->WriteDataBuffer[i];
             p++;
       }

       /* Writing address and the bank are set to writing beginning address structure.
*/
       ptr1->WriteAddress = ptr2->WriteAddressData;
       ptr1->WriteBank = ptr2->WriteBankData;

       while(1){
             DI();

                                               /* Word write library call */
             Status = FlashWordWrite( ptr1, ptr2->WordNumber, DataBufferAddress );

             EI();

             if( Status != STATE_OF_ABORT ){   /* State of abort?, NO */
                    break;
             }
       }

       return(Status);                         /* Return value = Status */
}

/*-------------------------------------------------------------------
* Function name :   FlashBlockVerify_CALL
* Input :           VerifyBank = Verify bank
*                   VerifyBlock = Verify block number
* Output :              Status = Return value from firm
* Summary :             FlashBlockVerify library call processing.
*-------------------------------------------------------------------*/
UCHAR   FlashBlockVerify_Call( UCHAR VerifyBank, UCHAR VerifyBlock ){
        UCHAR   Status;

        while( 1 ){
                DI();

                                                        /* Block verify library call */
                Status = FlashBlockVerify(VerifyBank, VerifyBlock);

                EI();

                if( Status != STATE_OF_ABORT ){         /* State of abort?, NO */
                        break;
                }
        }

        return(Status);                                 /* Return value = Status */
}

/*-------------------------------------------------------------------
* Function name :       FlashBlockBlankCheck_CALL
* Input :               BlankCheckBank = Blank check bank
*                       BlankCheckBlock = Blank check block number
* Output :              Status = Return value from firm
* Summary :             FlashBlockBlankCheck library call processing.
*-------------------------------------------------------------------*/
UCHAR   FlashBlockBlankCheck_Call( UCHAR BlankCheckBank, UCHAR BlankCheckBlock ){
        UCHAR   Status;

        while( 1 ){
                DI();

                                                        /* Block blank check library call */
                Status = FlashBlockBlankCheck( BlankCheckBank, BlankCheckBlock );

                EI();

                if( Status != STATE_OF_ABORT ){         /* State of abort?, NO */
                        break;
                }
        }

        return(Status);                                 /* Return value = Status */
}
```

```
/*-------------------------------------------------------------------
* Function name :   FlashGetInfo_CALL
* Input :           *ptr1 = Address of flash information acquisition structure
*                   DataBufferAddress = The first address in buffer where get data is
stored
*                   *ptr2 = Address of flash information acquisition structure
*                   ( Member of structure ... Option number )
* Output :          Status = Return value from firm
* Summary :         FlashGetInfo library call processing.
*                   ( When Security flag information or Boot flag information is
acquired )
*-------------------------------------------------------------------*/
UCHAR  FlashGetInfo_Call( struct stGetInfo *ptr1, USHORT DataBufferAddress, struct
stGetInfoData *ptr2 ){
       UCHAR  Status;

       /* Setting of option number of flash information acquisition structure */
       ptr1->OptionNumber = ptr2->OptionNumberData;

                                              /* Get information library call */
       Status = FlashGetInfo( ptr1, DataBufferAddress );

       return(Status);                        /* Return value = Status */
}

/*-------------------------------------------------------------------
* Function name : FlashGetInfo_CALL5
* Input :     *ptr1 = Address of flash information acquisition structure
*             DataBufferAddress = The first address in buffer where get data is stored
*             *ptr2 = Address of flash information acquisition structure
*             ( Member of structure ...  Option number
*                                        Bank
*                                        Block number)
*             #Option Number=Only 05H#
* Output :    Status = Return value from firm
* Summary :   FlashGetInfo library call processing.
*             (When block final address information is acquired)
*-------------------------------------------------------------------*/
UCHAR  FlashGetInfo_Call5( struct stGetInfo *ptr1, USHORT DataBufferAddress, struct
stGetInfoData *ptr2 ){
       UCHAR  Status;

              /* Setting of data of flash information acquisition structure */
       ptr1->OptionNumber = ptr2->OptionNumberData;
       ptr1->GetInfoBank = ptr2->GetInfoBankData;
       ptr1->GetInfoBlock = ptr2->GetInfoBlockData;

                                              /* Get information library call */
       Status = FlashGetInfo( ptr1, DataBufferAddress );

       return(Status);                         /* Return value = Status */
}

/*---------------------------------------------------------------------
* Function name :   FlashSetInfo_CALL3
* Input :           SetInfoData=Flash information data
*                   *ptr = Address of flash information acquisition structure( For
GetInfo )
*                   DataBufferAddress=The first address in buffer where get data is
stored( For GetInfo )
* Output :          Status = Return value from firm
* Summary :         FlashSetInfo library call processing
*                   ( When security flag information is set )
*--------------------------------------------------------------------*/
UCHAR  FlashSetInfo_Call3( UCHAR SetInfoData, struct stGetInfo *ptr, USHORT
DataBufferAddress ){
       UCHAR  Status;
       UCHAR  SecurityFlag;
       UCHAR  *p;

       p = ( UCHAR * )DataBufferAddress;

       while( 1 ){
              DI();

              while( 1 ){
              ptr->OptionNumber = 0x03;  /* Security flag information acquisition */
                                               /* Get information library call */
                     Status = FlashGetInfo( ptr, DataBufferAddress );

                     if( Status == FLASHFIRM_NORMAL_END ){
                            /* The state of a present security flag is maintained */
                            /* in the variable. */
                            SecurityFlag = *p;
                            break;
                     }
              }

                                               /* Set information library call */
              Status = FlashSetInfo( SetInfoData );

              EI();

              if( Status == STATE_OF_ABORT ){   /* State of abort?, YES */
                     while(1){
                            ptr->OptionNumber = 0x03;
```

```
                                               /* Get information library call */
                           Status = FlashGetInfo( ptr, DataBufferAddress );

                                               /* Normal completion?, YES */
                           if( Status == FLASHFIRM_NORMAL_END ){
                                 break;
                           }
                    }

                           /* Flash information rewriting completion?, YES */
                    if( SecurityFlag != *p ){
                           break;
                    }
             }else{
                    break;
             }
       }

       return( Status );                       /* Return value = Status */
}

/*-------------------------------------------------------------------
* Function name :   FlashSetInfo_CALL4
* Input :           SetInfoData = Flash information data
*                   *ptr = Address of flash information acquisition structure( For
GetInfo )
*                   DataBufferAddress = The first address in buffer where get data is
stored( For GetInfo )
* Output :          Status = Return value from firm
* Summary :         FlashSetInfo library call processing
*                   ( When boot flag information is set )
*-------------------------------------------------------------------*/
UCHAR  FlashSetInfo_Call4( UCHAR SetInfoData, struct stGetInfo *ptr, USHORT
DataBufferAddress ){
       UCHAR  Status;
       UCHAR  BootFlag;
       UCHAR  *p;

       p = ( UCHAR * )DataBufferAddress;

       while(1){
             DI();

             while(1){
                                        /* Boot flag information acquisition */
                    ptr->OptionNumber = 0x04;
                                               /* Get information library call */
                    Status = FlashGetInfo( ptr, DataBufferAddress );
```

```
                        if( Status == FLASHFIRM_NORMAL_END ){
                                                /* The state of a present boot flag */
                                                /* is maintained in the variable. */
                                BootFlag = *p;
                                break;
                        }
                }

                                                        /* Set information library call */
                Status = FlashSetInfo( SetInfoData );

                EI();

                if( Status == STATE_OF_ABORT ){         /* State of abort ?, YES */
                        while( 1 ){
                                ptr->OptionNumber = 0x04;
                                                        /* Get information library call */
                                Status = FlashGetInfo( ptr, DataBufferAddress );

                                                        /* Normal completion?, YES */
                                if( Status == FLASHFIRM_NORMAL_END ){
                                        break;
                                }
                        }

                        if( BootFlag != *p ){
                                break;
                        }
                }else{
                        break;
                }
        }

        return( Status );                               /* Return value = Status */
}

/*-------------------------------------------------------------------
* Function name :       CheckFLMD_CALL
* Input :               None
* Output :              Status = Return value from firm
* Summary :             CheckFLMD library call processing.
*-------------------------------------------------------------------*/
UCHAR  CheckFLMD_Call( void ){
        UCHAR  Status;

        Status = CheckFLMD();                           /* Mode check library call */
        return( Status );                     /* Return value = Status */
}
```

A.2 Self Programming Library (Normal Model)

<SelfLibrary_normal.asm>

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  System :          Self programming library( Normal model )
;  File name :       SelfLibrary_normal.asm
;  Version :         2.00
;  Target CPU :      78K0/Kx2
;  Last updated :    2005/07/08
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

PUBLIC _FlashStart
PUBLIC _FlashEnd
PUBLIC _FlashEnv
PUBLIC _FlashBlockErase
PUBLIC _FlashWordWrite
PUBLIC _FlashBlockVerify
PUBLIC _FlashBlockBlankCheck
PUBLIC _FlashGetInfo
PUBLIC _FlashSetInfo
PUBLIC _CheckFLMD
PUBLIC _EEPROMWrite

;----------------------------------------------------------------------
;      EQU settings
;----------------------------------------------------------------------
FLASH_ENV                  EQU    00H    ; Initialization
FLASH_BLOCK_ERASE          EQU    03H    ; Block erace
FLASH_WORD_WRITE           EQU    04H    ; Word write
FLASH_BLOCK_VERIFY         EQU    06H    ; Block verify
FLASH_BLOCK_BLANKCHECK     EQU    08H    ; Block blank check
FLASH_GET_INF              EQU    09H    ; Flash memory information read
FLASH_SET_INF              EQU    0AH    ; Flash memory information setting
FLASH_CHECK_FLMD           EQU    0EH    ; Mode check
FLASH_EEPROM_WRITE         EQU    17H    ; EEPROM write

FLASHFIRM_PARAMETER_ERROR  EQU    05H    ; Parameter error

BANK_BLC_ERROR             EQU    0FFH   ; Bank number error( BLOCK )
BANK_ADDR_ERROR            EQU    0FFFFH ; Bank number error( ADDRESS )


SELF_PROG    CSEG

;--------------------------------------------------------------------
; Function name :          _FlashStart
; Input :                  None
; Output :                 None
; Destroyed register :     None
; Summary :                Self programming start processing.
;-------------------------------------------------------------------
_FlashStart:
      MOV    PFCMD, #0A5H        ; PFCMD register control
      MOV    FLPMC, #001H        ; FLPMC register control ( set value )
      MOV    FLPMC, #0FEH        ; FLPMC register control ( inverted set value )
      MOV    FLPMC, #001H        ; FLPMC register control ( set value )
      RET

;-------------------------------------------------------------------
; Function name :          _FlashEnd
; Input :                  None
; Output :                 None
; Destroyed register :     None
; Summary :                Self programming end processing.
;-------------------------------------------------------------------
_FlashEnd:
      MOV    PFCMD, #0A5H        ; PFCMD register control
      MOV    FLPMC, #000H        ; FLPMC register control ( set value )
      MOV    FLPMC, #0FFH        ; FLPMC register control ( inverted set value )
      MOV    FLPMC, #000H        ; FLPMC register control ( set value )
      RET

;-------------------------------------------------------------------
; Function name :          _FlashEnv
; Input :                  AX = Entry RAM address
; Output :                 None
; Destroyed register :     None
; Summary :                Initialization processing of self programming.
;-------------------------------------------------------------------
_FlashEnv:
; Initialization processing
      PUSH   PSW                 ; Save register bank in STACK.
      PUSH   AX
      SEL    RB3                 ; Sets to register bank 3.
      POP    HL                  ; Sets Entry RAM address to HL register
      MOV    C, #FLASH_ENV       ; Sets function number to C register
      CALL   !8100H              ; Calls flash firmware
      MOV    A, #09H
      MOV    [HL+13H], A         ; Set Block Erase Retry Number
      MOV    [HL+14H], A         ; Set Chip Erase Retry Number
      POP    PSW                 ; Restores register bank from STACK.
      RET

;-------------------------------------------------------------------
; Function name :          _FlashBlockErase
; Input :                  AX = Erase bank
;                           STACK = Erase block number
; Output :                  BC = Status
; Destroyed register :      AX,BC
; Summary :                 Erases of specified block ( 1Kbyte ).
;----------------------------------------------------------------------
_FlashBlockErase:
       PUSH   HL

; Calculate Erase block number from block number and bank.
       MOVW   BC, AX
       MOVW   AX, SP
       MOVW   HL, AX
       MOV    A, [HL+4]     ; Read STACK data( = Erase block number )
       MOV    B, A
       MOV    A, C          ; A ... Erase bank, B ... Erase block number
                            ; Block number is calculated from block number and bank.
                            ; ( Return A = Erase block number after it calculates )
       CALL   !ExchangeBlockNum
       BZ     $FBE_PErr     ; It is error if the bank number is outside the range.

; Block erase processing
       PUSH   PSW                         ; Save register bank in STACK.
       PUSH   AX
       SEL    RB3                         ; Sets to register bank 3.
       POP    AX
       MOV    [HL+3], A                   ; Sets entry RAM+3 to Erase block number
                                          ; after it calculates
       MOV    C, #FLASH_BLOCK_ERASE       ; Sets function number to C register
       CALL   !8100H                      ; Calls flash firmware
       POP    PSW                         ; Restores register bank from STACK.

; Get flash firmware error information
       MOV    A,0FEE3H      ; Sets flash firmware error information to return value
                            ; ( 0FEE3H = B register of Bank 3 )
       BR     FlashBlockErase00

; Parameter error
FBE_PErr:
       MOV    A, #FLASHFIRM_PARAMETER_ERROR      ; Sets parameter error to
                                                 ; return value

FlashBlockErase00:
       MOV    C, A
       MOV    B, #00H
       POP    HL
       RET

;----------------------------------------------------------------------
```

```
; Function name : _FlashWordWrite
; Input :                  AX = Address of writing beginning address structure
;                          ( Member of structure ...
;                                       Writing starting address
;                                       Bank of writing starting addres )
;                                       STACK1 = Number of writing data
;                                       STACK2 = Address in writing data buffer
; Output :                 BC = Status
; Destroyed register :     AX, BC, DE
; Summary :                Data on RAM is written in the flash memory.
;                          256 bytes or less ( Every 4 bytes ) are written at a time.
;-------------------------------------------------------------------
_FlashWordWrite:
       PUSH  HL

; Calculate Writing address from writing address and bank.
       MOVW  DE, AX
       MOVW  AX, SP
       MOVW  HL, AX
       MOV   A, [HL+4]            ; Read STACK data( =Number of writing data )
       MOV   B, A
       MOV   A, [HL+6]            ; Read STACK data( =Address in writing data buffer )
       XCH   A, X
       MOV   A, [HL+7]
       MOVW  HL, AX
       MOVW  AX, DE               ; AX ...      Address of writing beginning address
                                  ;             structure address,
                                  ; B ...       Number of writing data,
                                  ; HL ...      Address in writing data buffer
       CALL  !ExchangeAddress     ; Writing address is calculated from structure
                                  ; member's writing address and bank
                                  ; ( Return AX=Writing address )
       BZ    $FWW_PErr      ; It is error if the bank number is outside the range.

; Word write processing
       PUSH  PSW            ; Save register bank in STACK.
       PUSH  AX
       PUSH  BC
       PUSH  HL
       SEL   RB3            ; Sets to register bank 3.
       POP   AX
       MOV   [HL+5], A      ; Sets entry RAM+5 to higher address in writing data buffer
       MOV   A, X
       MOV   [HL+4], A      ; Sets entry RAM+4 to lower address in writing data buffer
       POP   AX
       MOV   [HL+3], A      ; Sets entry RAM+3 to Number of writing data
       MOV   A,X
       MOV   [HL+0], A      ; Sets entry RAM+0 to Writing address lower bytes
       POP   AX
       MOV   [HL+2], A    ; Sets entry RAM+2 to Writing address most higher bytes
       MOV   A, X
       MOV   [HL+1], A    ; Sets entry RAM+1 to Writing address higher bytes
       MOV   C, #FLASH_WORD_WRITE      ; Sets function number to C register
       CALL  !8100H       ; Calls flash firmware
       POP   PSW          ; Restores register bank from STACK.

; Get flash firmware error information
       MOV   A, 0FEE3H    ; Sets flash firmware error information to return value
                          ;( 0FEE3H = B register of Bank 3 )
       BR    FlashWordWrite00

; Parameter error
FWW_PErr:
       MOV   A, #FLASHFIRM_PARAMETER_ERROR    ; Sets parameter error to
                                              ; return value

FlashWordWrite00:
       MOV   C, A
       MOV   B, #00H
       POP   HL
       RET

;--------------------------------------------------------------------
; Function name :          _FlashBlockVerify
; Input :                  AX = Verify bank
;                          STACK = Verify block number
; Output             :     BC = Status
; Destroyed register :     AX, BC
; Summary :                Internal verify of specified block ( 1Kbyte ).
;--------------------------------------------------------------------
_FlashBlockVerify:
       PUSH  HL

; Calculate Verify block number from block number and bank.
       MOVW  BC, AX
       MOVW  AX, SP
       MOVW  HL, AX
       MOV   A, [HL+4]           ; Read STACK data( =Verify block number )
       MOV   B, A
       MOV   A, C                ; A ... Verify bank, B ... Verify block number
       CALL  !ExchangeBlockNum   ; Block number is calculated from block number
                                 ; and bank.
                                 ; (Return A=Verify block number after
                                 ; it calculates)
       BZ    $FBV_PErr           ; It is error if the bank number is outside
                                 ; the range.
; Block verify processing
       PUSH   PSW                              ; Save register bank in STACK.
       PUSH   AX
       SEL    RB3                              ; Sets to register bank 3.
       POP    AX
       MOV    [HL+3], A                        ; Sets entry RAM+3 to Verify block
                                               ; number after it calculates
       MOV    C, #FLASH_BLOCK_VERIFY           ; Sets function number to C register
       CALL   !8100H                           ; Calls flash firmware
       POP    PSW                              ; Restores register bank from STACK.

; Get flash firmware error information
       MOV    A, 0FEE3H    ; Sets flash firmware error information to return value
                           ; ( 0FEE3H = B register of Bank 3 )
       BR     FlashBlockVerify00

; Parameter error
FBV_PErr:
       MOV    A, #FLASHFIRM_PARAMETER_ERROR    ; Sets parameter error to return
                                               ; value

FlashBlockVerify00:
       MOV    C, A
       MOV    B, #00H
       POP    HL
       RET

;-------------------------------------------------------------------
; Function name :          _FlashBlockBlankCheck
; Input :                  AX = Blank check bank
;                          STACK = Blank check block number
; Output :                 BC = Status
; Destroyed register :     AX, BC
; Summary :                Blank check of specified block ( 1Kbyte ).
;-------------------------------------------------------------------
_FlashBlockBlankCheck:
       PUSH   HL

; Calculate Blank check block number from block number and bank.
       MOVW   BC, AX
       MOVW   AX, SP
       MOVW   HL, AX
       MOV    A, [HL+4]    ; Read STACK data( =Blank check block number )
       MOV    B, A
       MOV    A, C         ; A ... Blank check bank, B ... Blank check block number
       CALL   !ExchangeBlockNum        ; Block number is calculated from block
                                       ; number and bank.
                           ; ( Return A = Blank check block number after it calculates )
       BZ     $FBBC_PErr   ; It is error if the bank number is outside the range.

; Block blank check processing
       PUSH   PSW                                 ; Save register bank in STACK.
       PUSH   AX
       SEL    RB3                                 ; Sets to register bank 3.
       POP    AX
       MOV    [HL+3], A                           ; Sets entry RAM+3 to Blank check
block number after it calculates
       MOV    C, #FLASH_BLOCK_BLANKCHECK          ; Sets function number to C register
       CALL   !8100H                              ; Calls flash firmware
       POP    PSW                                 ; Restores register bank from STACK.

; Get flash firmware error information
       MOV    A, 0FEE3H    ; Sets flash firmware error information to return value
                           ; ( 0FEE3H = B register of Bank 3 )
       BR     FlashBlockBlankCheck00

;Parameter error
FBBC_PErr:
       MOV    A, #FLASHFIRM_PARAMETER_ERROR       ; Sets parameter error to return
                                                  ; value

FlashBlockBlankCheck00:
       MOV    C, A
       MOV    B, #00H
       POP    HL
       RET

;--------------------------------------------------------------------
; Function name :          _FlashGetInfo
; Input :                  AX = Address of flash information acquisition structure
;                          (Member of structure ...   Option number
;                                                     Bank
;                                                     Block number)
;                          STACK = The first address in buffer where get data is stored
; Output :                 BC = Status
; Destroyed register :     AX,BC,DE
; Summary :                The set up information of the flash memory is read.
;--------------------------------------------------------------------
_FlashGetInfo:
       PUSH   HL

; Check of Option number
       MOVW   BC, AX
       MOVW   AX, SP
       MOVW   HL, AX
       MOV   A, [HL+4]            ; Read STACK data( =The first address in buffer
                                  ; where get data is stored )
       XCH   A, X
       MOV   A, [HL+5]
       XCHW  AX, BC               ; AX ... Address of flash information
                                  ; acquisition structure
                                  ; BC ... The first address in buffer where get data
                                  ; is stored
       MOVW  HL, AX
       MOVW  AX, BC
       MOVW  DE, AX
       MOV   A, [HL+0]            ; Read data from flash information acquisition
                                  ; structure( =Option number )
       CMP   A, #05H              ; Option number = 5 ?
       BNZ   $FlashGetInfo10      ; NO

; Calculate Block number from block number and bank.
       MOV   X, A
       MOV   A, [HL+2]            ; Read data from flash information acquisition
                                  ; structure( =Block number )
       MOV   B, A
       MOV   A, [HL+1]            ; Read data from flash information acquisition
                                  ; structure(=Bank)
                                  ; A...Bank, B...Block number
       CALL  !ExchangeBlockNum    ; Block number is calculated from block number
                                  ; and bank.
                                  ; ( Return A = Block number after it calculates )
       BZ    $FlashGetInfo20      ; It is error if the bank number is outside
                                  ; the range.
       XCH   A, X                 ; A...Option number, X...Block number

; Get info processing( When Option number = 5 )
       PUSH  PSW                  ; Save register bank in STACK.
       PUSH  DE
       PUSH  AX
       SEL   RB3                  ; Sets to register bank 3.
       POP   AX
       XCH   A, X
       MOV   [HL+0],A             ; Sets entry RAM+0 to Block number
       MOV   A, X                 ; A...Option number
       BR    FlashGetInfo40

; Check of Option number error
FlashGetInfo10:
       CMP   A, #03H              ; Option number = 3 ?
       BZ    $FlashGetInfo30      ; YES
       CMP   A, #04H              ; Option number = 4?
       BZ    $FlashGetInfo30      ; YES
```

```
FlashGetInfo20:
       MOV    A, #FLASHFIRM_PARAMETER_ERROR     ; The parameter error is returned,
                                                ; except when option NO is 3, 4,
                                                ; and 5.
       BR     FlashGetInfo50

; Get info processing( When Option number = 3, 4 )
FlashGetInfo30:
       PUSH   PSW                   ; Save register bank in STACK.
       PUSH   DE
       PUSH   AX
       SEL    RB3                   ; Sets to register bank 3.
       POP    AX
FlashGetInfo40:
       MOV    [HL+3], A             ; Sets entry RAM+3 to Option number
       POP    AX
       MOV    [HL+5], A             ; Sets entry RAM+5 to Storage buffer higher address
       MOV    A, X
       MOV    [HL+4], A             ; Sets entry RAM+4 to Storage buffer lower address
       MOV    C, #FLASH_GET_INF     ; Sets function number to C register
       CALL   !8100H                ; Calls flash firmware
       POP    PSW                   ; Restores register bank from STACK.

; Calculate Address from Storage buffer and bank. Nothing to do
; when Option number = 3or4 or Bank = 0 or Block number( Previous ) < 32
; or Block number( Previous ) >= 48.
; A = Option number, B = Bank, C ... Block number(Previous),
; DE = Storage buffer first address of get data
       CMP    A, #05H                      ; Option number = 5?
       BNZ    $ReturnAddress_end           ; NO
       MOV    A, B
       CMP    A, #0                        ; Bank = 0 ?
       BZ     $ReturnAddress_end           ; YES
       XCH    A, C
       CMP    A, #32                       ; Block number( Previous ) < 32?
       BC     $ReturnAddress_end           ; YES
       CMP    A, #48                       ; Block number( Previous ) >= 48?
       BNC    $ReturnAddress_end           ; YES
       MOV    A, C

; Calculation of address( 40H*Bank is pulled from address in two high rank bytes.
; Lower address is the state as it is. )
       XCHW   AX, DE
       MOVW   HL, AX
       MOV    A, [HL+1]
       MOV    X, A
       MOV    A, [HL+2]             ; A ... Most higher address, X ... Higher address
       XCHW   AX, DE                ; A ... Bank, D ... Most higher address,
                                  ; E ... Higher address
       MOV    [HL+2], A           ; Sets Storage buffer+2 to Bank.
       MOV    X, #0
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROLC   A, 1
       XCH    A, X
       ROLC   A, 1                ; AX = 40H*Bank
       XCHW   AX, DE
       XCH    A, X
       SUB    A, E
       XCH    A, X
       SUBC   A, D
       MOV    A, X
       MOV    [HL+1], A           ; Sets Storage buffer+1 to Calculated
                                  ; address(higher).
ReturnAddress_end:

; Get flash firmware error information
       MOV    A, 0FEE3H           ; Sets flash firmware error information to
                                  ; return value
                                  ;( 0FEE3H = B register of Bank 3 )
FlashGetInfo50:
       MOV    C, A
       MOV    B, #00H
       POP    HL
       RET

;--------------------------------------------------------------------
; Function name :          _FlashSetInfo
; Input :                  AX = Flash information data
; Output :                 BC = Status
; Destroyed register :     A, BC
; Summary :                Setting of flash information.
;--------------------------------------------------------------------
_FlashSetInfo:
; Set infomation processing
       MOV    A, X
       PUSH   AX                  ; Save Flash information data in STACK.
       PUSH   PSW                 ; Save register bank in STACK.
       SEL    RB3                 ; Sets to register bank 3.
       MOVW   AX, SP
       ADDW   AX, #2
       MOV    [HL+5], A           ; Sets entry RAM+5 to higher address of flash
                                  ; information data secured for stack
       MOV    A, X
       MOV    [HL+4], A           ; Sets entry RAM+4 to lower address of flash
                                  ; information data secured for stack
       MOV    C, #FLASH_SET_INF   ; Sets function number to C register
       CALL   !8100H              ; Calls flash firmware
       POP    PSW                 ; Restores register bank from STACK.
       POP    AX

; Get flash firmware error information
       MOV    A, 0FEE3H    ; Sets flash firmware error information to return value
                           ; ( 0FEE3H = B register of Bank 3 )
       MOV    C, A
       MOV    B, #00H
       RET

;-------------------------------------------------------------------
; Function name :          _CheckFLMD
; Input :                  None
; Output :                 BC = Status
; Destroyed register :     A, BC
; Summary :                Checks voltage level of FLMD pin.
;-------------------------------------------------------------------
_CheckFLMD:
; Set infomation processing
       PUSH   PSW                       ; Save register bank in STACK.
       SEL    RB3                       ; Sets to register bank 3.
       MOV    C, #FLASH_CHECK_FLMD      ; Sets function number to C register
       CALL   !8100H                    ; Calls flash firmware
       POP    PSW                       ; Restores register bank from STACK.

; Get flash firmware error information
       MOV    A, 0FEE3H    ; Sets flash firmware error information to return value
                           ; ( 0FEE3H = B register of Bank 3 )
       MOV    C, A
       MOV    B, #00H
       RET

;-------------------------------------------------------------------
; Function name :          _EEPROMWrite
; Input                    : AX = Address of writing beginning address structure
;                          ( Member of structure ... Writing starting address
;                          Bank of writing starting address)
;                          STACK1 = Number of writing data
;                          STACK2 = Address in writing data buffer
; Output :                 BC=Status
; Destroyed register :     AX,BC,DE
; Summary :                Data on RAM is written in the flash memory.
;                          256 bytes or less ( Every 4 bytes ) are written at a time.
```

```
;-------------------------------------------------------------------
_EEPROMWrite:
       PUSH   HL

; Calculate Writing address from writing address and bank.
       MOVW   DE, AX
       MOVW   AX, SP
       MOVW   HL, AX
       MOV    A, [HL+4]           ; Read STACK data( =Number of writing data )
       MOV    B, A
       MOV    A, [HL+6]           ; Read STACK data( =Address in writing data buffer )
       XCH    A, X
       MOV    A, [HL+7]
       MOVW   HL, AX
       MOVW   AX, DE              ; AX ... Address of writing beginning address
                                  ; structure address,
                                  ; B ... Number of writing data,
                                  ; HL ... Address in writing data buffer
       CALL   !ExchangeAddress    ; Writing address is calculated from structure
                                  ; member's writing address and bank
                                  ; ( Return AX = Writing address )
       BZ     $EW_PErr            ; It is error if the bank number is outside
                                  ; the range.

; EEPROM write processing
       PUSH   PSW          ; Save register bank in STACK.
       PUSH   AX
       PUSH   BC
       PUSH   HL
       SEL    RB3          ; Sets to register bank 3.
       POP    AX
       MOV    [HL+5], A    ; Sets entry RAM+5 to higher address in writing data buffer
       MOV    A, X
       MOV    [HL+4], A    ; Sets entry RAM+4 to lower address in writing data buffer
       POP    AX
       MOV    [HL+3], A    ; Sets entry RAM+3 to Number of writing data
       MOV    A, X
       MOV    [HL+0], A    ; Sets entry RAM+0 to Writing address lower bytes
       POP    AX
       MOV    [HL+2], A    ; Sets entry RAM+2 to Writing address most higher bytes
       MOV    A, X
       MOV    [HL+1], A    ; Sets entry RAM+1 to Writing address higher bytes
       MOV    C, #FLASH_EEPROM_WRITE     ; Sets function number to C register
       CALL   !8100H       ; Calls flash firmware
       POP    PSW          ; Restores register bank from STACK.

; Get flash firmware error information
       MOV    A,0FEE3H     ; Sets flash firmware error information to return value
```

```
                           ;( 0FEE3H = B register of Bank 3 )
       BR     EEPROMWrite00

; Parameter error
EW_PErr:
       MOV    A, #FLASHFIRM_PARAMETER_ERROR    ; Sets parameter error to return
                                               ; value

EEPROMWrite00:
       MOV    C, A
       MOV    B, #00H
       POP    HL
       RET

;-------------------------------------------------------------------
; Function name :   ExchangeBlockNum
; Input :           A = Bank
;                   B = Block number
; Output :          A = Block number( New )
;                   B = Bank
;                   C = Block number( Previous )
; Summary :         Block number is converted into the real address
;                   from bank information.
;-------------------------------------------------------------------
ExchangeBlockNum:
; It calculates from 32 to 47 block number.
       XCH    A, B
       CMP    A, #32
       BC     $EBN_end
       CMP    A, #48
       BNC    $EBN_end

; Calculation of block number( Bank*16 is added to block number. )
       XCH    A, B
       MOV    C, A                ; C ... Bank
       CMP    A, #6
       BNC    $EBN_error_end
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1                ; A = 16*Bank
       ADD    A, B
       XCH    A, C
       XCH    A, B
       XCH    A, C                ; A = Block number after it calculates, B = Bank,
                                  ; C = Block number before it calculates
       BR     EBN_end
```

```
; Bank error
EBN_error_end:
       MOV    A, #BANK_BLC_ERROR  ; Return error number

EBN_end:
       CMP    A, #BANK_BLC_ERROR  ; Bank error?
       RET

;-------------------------------------------------------------------
; Function name :   ExchangeAddress
; Input :           AX = Address of writing beginning address structure
;                   ( Member of structure ...  Writing starting address
;                                              Bank of writing starting address )
; Output      :     AX = Writing starting address( Address in two high rank bytes )
;                   C = Writing starting address( Lower address )
; Summary :         Writing starting address of structure is converted
;                   into the real address from bank information.
;-------------------------------------------------------------------
ExchangeAddress:
       PUSH   HL

; It calculates from 8000H to BFFFH address.
       MOVW   HL, AX
       MOV    A, [HL+0]           ; Read data from writing beginning address
                                  ; structure(=Write address)
       MOV    X, A
       MOV    A, [HL+1]
       CMPW   AX, #8000H
       BC     $EA_end
       CMPW   AX, #0C000H
       BNC    $EA_end

; Calculation of address( Bank*40H is added to address in two high rank bytes.
; Lower address is the state as it is. )
       MOV    D, A
       XCH    A, X
       MOV    C, A
       MOV    X, #0
       MOV    A, [HL+2]     ; Read data from writing beginning address structure
                            ; ( =Bank of writing starting address )
       CMP    A, #6
       BNC    $EA_error_end
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROLC   A, 1
       XCH   A, X
       ROLC  A, 1          ; AX=40H*Bank
       XCH   A, X
       ADD   A, D          ; Addition of Higher address
       XCH   A, X
       ADDC  A, #0         ; Addition of Most higher address
                           ; A ... Most higher address after it calculates
                           ; X ... higher address after it calculates,
                           ; C ... Lower address
       BR    EA_normal_end

; Bank error
EA_error_end:
       MOVW  AX, #BANK_ADDR_ERROR
       BR    EA_normal_end

EA_end:
       XCH   A, X
       MOV   C, A
       MOV   A, #0         ; A ... Most higher address after it calculates
                           ; X ... higher address after it calculates,
                           ; C ... Lower address
EA_normal_end:
       POP   HL
       CMPW  AX, #BANK_ADDR_ERROR       ; Bank error?
       RET



       END
```

A.3 Self Programming Library (Static Model)

<SelfLibrary_static.asm>

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  System :          Self programming library( Static model )
;  File name :       SelfLibrary_static.asm
;  Version :         2.00
;  Target CPU :      78K0/Kx2
;  Last updated :    2005/07/08
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

PUBLIC _FlashStart
PUBLIC _FlashEnd
PUBLIC _FlashEnv
PUBLIC _FlashBlockErase
PUBLIC _FlashWordWrite
PUBLIC _FlashBlockVerify
PUBLIC _FlashBlockBlankCheck
PUBLIC _FlashGetInfo
PUBLIC _FlashSetInfo
PUBLIC _CheckFLMD
PUBLIC _EEPROMWrite

;---------------------------------------------------------------------
;      EQU settings
;---------------------------------------------------------------------
FLASH_ENV                  EQU    00H    ; Initialization
FLASH_BLOCK_ERASE          EQU    03H    ; Block erace
FLASH_WORD_WRITE           EQU    04H    ; Word write
FLASH_BLOCK_VERIFY         EQU    06H    ; Block verify
FLASH_BLOCK_BLANKCHECK     EQU    08H    ; Block blank check
FLASH_GET_INF              EQU    09H    ; Flash memory information read
FLASH_SET_INF              EQU    0AH    ; Flash memory information setting
FLASH_CHECK_FLMD           EQU    0EH    ; Mode check
FLASH_EEPROM_WRITE         EQU    17H    ; EEPROM write

FLASHFIRM_PARAMETER_ERROR  EQU    05H    ; Parameter error

BANK_BLC_ERROR             EQU    0FFH   ; Bank number error(BLOCK)
BANK_ADDR_ERROR            EQU    0FFFFH ; Bank number error(ADDRESS)


SELF_PROG    CSEG

;-------------------------------------------------------------------
; Function name :          _FlashStart
; Input :                  None
; Output :                 None
; Destroyed register :     None
; Summary :                Self programming start processing.
;-------------------------------------------------------------------
_FlashStart:
       MOV    PFCMD, #0A5H        ; PFCMD register control
       MOV    FLPMC, #001H        ; FLPMC register control ( set value )
       MOV    FLPMC, #0FEH        ; FLPMC register control ( inverted set value )
       MOV    FLPMC, #001H        ; FLPMC register control ( set value )
       RET

;-------------------------------------------------------------------
; Function name :          _FlashEnd
; Input :                  None
; Output :                 None
; Destroyed register :     None
; Summary :                Self programming end processing.
;-------------------------------------------------------------------
_FlashEnd:
       MOV    PFCMD, #0A5H        ; PFCMD register control
       MOV    FLPMC, #000H        ; FLPMC register control ( set value )
       MOV    FLPMC, #0FFH        ; FLPMC register control ( inverted set value )
       MOV    FLPMC, #000H        ; FLPMC register control ( set value )
       RET

;-------------------------------------------------------------------
; Function name :          _FlashEnv
; Input :                  AX = Entry RAM address
; Output :                 None
; Destroyed register :     None
; Summary :                Initialization processing of self programming.
;-------------------------------------------------------------------
_FlashEnv:
; Initialization processing
       PUSH   PSW                 ; Save register bank in STACK.
       PUSH   AX
       SEL    RB3                 ; Sets to register bank 3.
       POP    HL                  ; Sets Entry RAM address to HL register
       MOV    C,#FLASH_ENV        ; Sets function number to C register
       CALL   !8100H              ; Calls flash firmware
       MOV    A, #09H
       MOV    [HL+13H], A         ; Set Block Erase Retry Number
       MOV    [HL+14H], A         ; Set Chip Erase Retry Number
       POP    PSW                 ; Restores register bank from STACK.
       RET

;-------------------------------------------------------------------
; Function name :          _FlashBlockErase
; Input :                  A = Erase bank
;                          B = Erase block number
; Output :                 A = Status
; Destroyed register :     A, BC
; Summary :                Erases of specified block ( 1Kbyte ).
;--------------------------------------------------------------------
_FlashBlockErase:
; Calculate Erase block number from block number and bank.
       CALL   !ExchangeBlockNum          ; Block number is calculated from block
                                         ; number and bank.
                           ; ( Return A = Erase block number after it calculates )
       BZ     $FBE_PErr    ; It is error if the bank number is outside the range.

; Block erase processing
       PUSH   PSW          ; Save register bank in STACK.
       PUSH   AX
       SEL    RB3          ; Sets to register bank 3.
       POP    AX
       MOV    [HL+3], A    ; Sets entry RAM+3 to Erase block number after
                           ; it calculates
       MOV    C, #FLASH_BLOCK_ERASE      ; Sets function number to C register
       CALL   !8100H       ; Calls flash firmware
       POP    PSW          ; Restores register bank from STACK.

; Get flash firmware error information
       MOV    A, 0FEE3H    ; Sets flash firmware error information to return value
                           ; ( 0FEE3H = B register of Bank 3 )
       BR     FlashBlockErase00

; Parameter error
FBE_PErr:
       MOV    A, #FLASHFIRM_PARAMETER_ERROR    ; Sets parameter error to return
                                               ; value

FlashBlockErase00:
       RET

;--------------------------------------------------------------------
; Function name :          _FlashWordWrite
; Input :                  AX = Address of writing beginning address structure
;                          ( Member of structure ...
;                                                Writing starting address
;                                                Bank of writing starting address )
;                          B = Number of writing data
;                          HL = Address in writing data buffer
; Output :                 A = Status
; Destroyed register :     AX, C
; Summary :                Data on RAM is written in the flash memory.
;                          256 bytes or less ( Every 4 bytes ) are written at a time.
```

```
;-------------------------------------------------------------------
_FlashWordWrite:
       PUSH   DE

; Calculate Writing address from writing address and bank.
       CALL   !ExchangeAddress    ; Writing address is calculated from structure
                                  ; member's writing address and bank
                                  ; ( Return AX = Writing address )
       BZ     $FWW_PErr           ; It is error if the bank number is outside
                                  ; the range.

; Word write processing
       PUSH   PSW                 ; Save register bank in STACK.
       PUSH   AX
       PUSH   BC
       PUSH   HL
       SEL    RB3                 ; Sets to register bank 3.
       POP    AX
       MOV    [HL+5],A            ; Sets entry RAM+5 to higher address in writing
                                  ; data buffer
       MOV    A,X
       MOV    [HL+4],A            ; Sets entry RAM+4 to lower address in writing
                                  ; data buffer
       POP    AX
       MOV    [HL+3],A            ; Sets entry RAM+3 to Number of writing data
       MOV    A,X
       MOV    [HL+0],A            ; Sets entry RAM+0 to Writing address lower bytes
       POP    AX
       MOV    [HL+2],A            ; Sets entry RAM+2 to Writing address most
                                  ; higher bytes
       MOV    A,X
       MOV    [HL+1],A            ; Sets entry RAM+1 to Writing address higher bytes
       MOV    C,#FLASH_WORD_WRITE ; Sets function number to C register
       CALL   !8100H              ; Calls flash firmware
       POP    PSW                 ; Restores register bank from STACK.

; Get flash firmware error information
       MOV    A,0FEE3H            ; Sets flash firmware error information to
                                  ; return value
                                  ; ( 0FEE3H = B register of Bank 3 )
       BR     FlashWordWrite00

; Parameter error
FWW_PErr:
       MOV    A,#FLASHFIRM_PARAMETER_ERROR     ; Sets parameter error to return
                                               ; value

FlashWordWrite00:
       POP    DE
       RET

;-----------------------------------------------------------------------
; Function name :           _FlashBlockVerify
; Input :                   A = Verify bank
;                           B = Verify block number
; Output :                  A = Status
; Destroyed register :      A, BC
; Summary :                 Internal verify of specified block ( 1Kbyte ).
;-----------------------------------------------------------------------
_FlashBlockVerify:
; Calculate Verify block number from block number and bank.
       CALL   !ExchangeBlockNum   ; Block number is calculated from block number
                                  ; and bank.
                                  ; ( Return A = Verify block number after it
                                  ;  calculates )
       BZ     $FBV_PErr           ; It is error if the bank number is outside
                                  ; the range.

; Block verify processing
       PUSH   PSW                        ; Save register bank in STACK.
       PUSH   AX
       SEL    RB3                        ; Sets to register bank 3.
       POP    AX
       MOV    [HL+3], A                  ; Sets entry RAM+3 to Verify block number
                                         ; after it calculates
       MOV    C, #FLASH_BLOCK_VERIFY     ; Sets function number to C register
       CALL   !8100H                     ; Calls flash firmware
       POP    PSW                        ; Restores register bank from STACK.

; Get flash firmware error information
       MOV    A, 0FEE3H    ; Sets flash firmware error information to return value
                           ; ( 0FEE3H = B register of Bank 3 )
       BR     FlashBlockVerify00

; Parameter error
FBV_PErr:
       MOV    A, #FLASHFIRM_PARAMETER_ERROR    ; Sets parameter error to return
                                               ; value

FlashBlockVerify00:
       RET

;-----------------------------------------------------------------------
; Function name     :      _FlashBlockBlankCheck
; Input :                   A = Blank check bank
;                           B = Blank check block number
; Output :                 A = Status
; Destroyed register :     A, BC
; Summary :                Blank check of specified block ( 1Kbyte ).
;-------------------------------------------------------------------
_FlashBlockBlankCheck:
; Calculate Blank check block number from block number and bank.
       CALL   !ExchangeBlockNum   ; Block number is calculated from block number
                                  ; and bank.
                                  ; ( Return A = Blank check block number after
                                  ; it calculates )
       BZ     $FBBC_PErr          ; It is error if the bank number is outside
                                  ; the range.

; Block blank check processing
       PUSH   PSW                              ; Save register bank in STACK.
       PUSH   AX
       SEL    RB3                              ; Sets to register bank 3.
       POP    AX
       MOV    [HL+3], A                        ; Sets entry RAM+3 to Blank check
                                               ; block number after it calculates
       MOV    C, #FLASH_BLOCK_BLANKCHECK       ; Sets function number to C register
       CALL   !8100H                           ; Calls flash firmware
       POP    PSW                              ; Restores register bank from STACK.

; Get flash firmware error information
       MOV    A, 0FEE3H    ; Sets flash firmware error information to return value
                           ; ( 0FEE3H = B register of Bank 3 )
       BR     FlashBlockBlankCheck00

; Parameter error
FBBC_PErr:
       MOV    A, #FLASHFIRM_PARAMETER_ERROR    ; Sets parameter error to return
                                               ; value

FlashBlockBlankCheck00:
       RET

;-------------------------------------------------------------------
; Function name :          _FlashGetInfo
; Input :                  AX = Address of flash information acquisition structure
;                          ( Member of structure ...  Option number
;                                                     Bank
;                                                     Block number )
;                          BC = The first address in buffer where get data is stored
; Output :                 A = Status
; Destroyed register :     AX, BC, HL
; Summary :                The set up information of the flash memory is read.
;-------------------------------------------------------------------
```

```
_FlashGetInfo:
       PUSH   DE

; Check of Option number
       MOVW   HL, AX
       MOVW   AX, BC
       MOVW   DE, AX
       MOV    A, [HL+0]           ; Read data from flash information acquisition
                                  ; structure( =Option number )
       CMP    A, #05H             ; Option number = 5 ?
       BNZ    $FlashGetInfo10     ; NO

; Calculate Block number from block number and bank.
       MOV    X, A
       MOV    A, [HL+2]           ; Read data from flash information acquisition
structure(=Block number)
       MOV    B, A
       MOV    A, [HL+1]           ; Read data from flash information acquisition
structure(=Bank)
                                  ; A ... Bank, B ... Block number
       CALL   !ExchangeBlockNum   ; Block number is calculated from block number
                                  ; and bank.
                                  ; ( Return A=Block number after it calculates )
       BZ     $FlashGetInfo20     ; It is error if the bank number is outside
                                  ; the range.
       XCH    A, X                ; A ... Option number, X ... Block number

; Get info processing( When Option number = 5 )
       PUSH   PSW                 ; Save register bank in STACK.
       PUSH   DE
       PUSH   AX
       SEL    RB3                 ; Sets to register bank 3.
       POP    AX
       XCH    A, X
       MOV    [HL+0], A           ; Sets entry RAM+0 to Block number
       MOV    A, X                ; A ... Option number
       BR     FlashGetInfo40

; Check of Option number error
FlashGetInfo10:
       CMP    A, #03H             ; Option number = 3?
       BZ     $FlashGetInfo30     ; YES
       CMP    A, #04H             ; Option number = 4?
       BZ     $FlashGetInfo30     ; YES
FlashGetInfo20:
       MOV    A, #FLASHFIRM_PARAMETER_ERROR    ; The parameter error is returned,
                                               ; except when option NO is 3, 4,
                                               ; and 5.
       BR     FlashGetInfo50

; Get info processing( When Option number = 3, 4 )
FlashGetInfo30:
       PUSH   PSW                  ; Save register bank in STACK.
       PUSH   DE
       PUSH   AX
       SEL    RB3                  ; Sets to register bank 3.
       POP    AX
FlashGetInfo40:
       MOV    [HL+3], A            ; Sets entry RAM+3 to Option number
       POP    AX
       MOV    [HL+5], A            ;S ets entry RAM+5 to Storage buffer higher address
       MOV    A, X
       MOV    [HL+4], A            ; Sets entry RAM+4 to Storage buffer lower address
       MOV    C, #FLASH_GET_INF    ; Sets function number to C register
       CALL   !8100H               ; Calls flash firmware
       POP    PSW                  ; Restores register bank from STACK.

; Calculate Address from Storage buffer and bank.Nothing to do
; when Option number = 3or4 or Bank = 0 or Block number( Previous ) < 32
; or Block number(Previous) >= 48.
; A = Option number, B = Bank, C ... Block number( Previous ),
; DE = Storage buffer first address of get data
       CMP    A, #05H              ; Option number = 5?
       BNZ    $ReturnAddress_end   ; NO
       MOV    A, B
       CMP    A, #0                ; Bank = 0 ?
       BZ     $ReturnAddress_end   ; YES
       XCH    A, C
       CMP    A, #32               ; Block number( Previous ) < 32?
       BC     $ReturnAddress_end   ; YES
       CMP    A, #48               ; Block number( Previous ) >= 48?
       BNC    $ReturnAddress_end   ; YES
       MOV    A, C

; Calculation of address( 40H*Bank is pulled from address in two high rank bytes.
; Lower address is the state as it is. )
       XCHW   AX, DE
       MOVW   HL, AX
       MOV    A, [HL+1]
       MOV    X, A
       MOV    A, [HL+2]            ; A ... Most higher address, X ... Higher address
       XCHW   AX, DE               ; A ... Bank, D ... Most higher address,
                                   ; E ... Higher address
       MOV    [HL+2], A            ; Sets Storage buffer+2 to Bank.
       MOV    X, #0
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROLC   A, 1
       XCH    A, X
       ROLC   A, 1                ; AX = 40H*Bank
       XCHW   AX, DE
       XCH    A, X
       SUB    A, E
       XCH    A, X
       SUBC   A, D
       MOV    A, X
       MOV    [HL+1], A           ; Sets Storage buffer+1 to Calculated address
                                  ; ( higher ).
ReturnAddress_end:

; Get flash firmware error information
       MOV    A, 0FEE3H    ; Sets flash firmware error information to return value
                           ; ( 0FEE3H = B register of Bank 3 )
FlashGetInfo50:
       POP    DE
       RET

;--------------------------------------------------------------------
; Function name :          _FlashSetInfo
; Input :                  A = Flash information data
; Output :                 A = Status
; Destroyed register :     A
; Summary :                Setting of flash information.
;--------------------------------------------------------------------
_FlashSetInfo:
; Set infomation processing
       PUSH   AX                  ; Save Flash information data in STACK.
       PUSH   PSW                 ; Save register bank in STACK.
       SEL    RB3                 ; Sets to register bank 3.
       MOVW   AX, SP
       ADDW   AX, #2
       MOV    [HL+5], A           ; Sets entry RAM+5 to higher address of flash
                                  ; information data secured for stack
       MOV    A, X
       MOV    [HL+4], A           ; Sets entry RAM+4 to lower address of flash
                                  ; information data secured for stack
       MOV    C, #FLASH_SET_INF   ; Sets function number to C register
       CALL   !8100H              ; Calls flash firmware
       POP    PSW                 ; Restores register bank from STACK.
       POP    AX
```

```
; Get flash firmware error information
       MOV    A, 0FEE3H     ; Sets flash firmware error information to return value
                            ; ( 0FEE3H = B register of Bank 3 )
       RET

;--------------------------------------------------------------------
; Function name :           _CheckFLMD
; Input :                   None
; Output :                  A = Status
; Destroyed register :      A
; Summary :                 Checks voltage level of FLMD pin.
;--------------------------------------------------------------------
_CheckFLMD:
; Set infomation processing
       PUSH   PSW                         ; Save register bank in STACK.
       SEL    RB3                         ; Sets to register bank 3.
       MOV    C, #FLASH_CHECK_FLMD        ; Sets function number to C register
       CALL   !8100H                      ; Calls flash firmware
       POP    PSW                         ; Restores register bank from STACK.

; Get flash firmware error information
       MOV    A, 0FEE3H     ; Sets flash firmware error information to return value
                            ; ( 0FEE3H = B register of Bank 3 )
       RET

;--------------------------------------------------------------------
; Function name :           _EEPROMWrite
; Input :                   AX = Address of writing beginning address structure
;                           ( Member of structure ...
                                                Writing starting address
;                                               Bank of writing starting address )
;                           B = Number of writing data
;                           HL = Address in writing data buffer
; Output :                  A = Status
; Destroyed register :      AX, C
; Summary :                 Data on RAM is written in the flash memory.
;                           256 bytes or less ( Every 4 bytes ) are written at a time.
;--------------------------------------------------------------------
_EEPROMWrite:
       PUSH   DE

; Calculate Writing address from writing address and bank.
       CALL   !ExchangeAddress    ; Writing address is calculated from structure
                                  ; member's writing address and bank
                                  ; ( Return AX = Writing address )
       BZ     $EW_PErr      ; It is error if the bank number is outside the range.

; EEPROM write processing
```

```
       PUSH   PSW                 ; Save register bank in STACK.
       PUSH   AX
       PUSH   BC
       PUSH   HL
       SEL    RB3                 ; Sets to register bank 3.
       POP    AX
       MOV    [HL+5], A           ; Sets entry RAM+5 to higher address in writing data
buffer
       MOV    A,X
       MOV    [HL+4], A           ; Sets entry RAM+4 to lower address in writing data
buffer
       POP    AX
       MOV    [HL+3], A           ; Sets entry RAM+3 to Number of writing data
       MOV    A,X
       MOV    [HL+0], A           ; Sets entry RAM+0 to Writing address lower bytes
       POP    AX
       MOV    [HL+2], A     ; Sets entry RAM+2 to Writing address most higher bytes
       MOV    A, X
       MOV    [HL+1], A           ; Sets entry RAM+1 to Writing address higher bytes
       MOV    C, #FLASH_EEPROM_WRITE     ; Sets function number to C register
       CALL   !8100H              ; Calls flash firmware
       POP    PSW                 ; Restores register bank from STACK.

;Get flash firmware error information
       MOV    A,0FEE3H            ;Sets flash firmware error information to return
value
                                  ;(0FEE3H = B register of Bank 3)
       BR     EEPROMWrite00

; Parameter error
EW_PErr:
       MOV    A, #FLASHFIRM_PARAMETER_ERROR    ; Sets parameter error to return
                                               ; value

EEPROMWrite00:
       POP    DE
       RET

;-------------------------------------------------------------------
; Function name :   ExchangeBlockNum
; Input :           A=Bank
;                   B = Block number
; Output            A = Block number( New )
;                   B = Bank
;                   C = Block number( Previous )
; Summary :         Block number is converted into the real address from bank
;                   information.
;-------------------------------------------------------------------
```

```
ExchangeBlockNum:
; It calculates from 32 to 47 block number.
       XCH    A, B
       CMP    A, #32
       BC     $EBN_end
       CMP    A, #48
       BNC    $EBN_end

; Calculation of block number( Bank*16 is added to block number. )
       XCH    A, B
       MOV    C, A                ; C ... Bank
       CMP    A, #6
       BNC    $EBN_error_end
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1                ; A = 16*Bank
       ADD    A, B
       XCH    A, C
       XCH    A, B
       XCH    A, C                ; A = Block number after it calculates, B = Bank,
                                  ; C = Block number before it calculates
       BR     EBN_end

; Bank error
EBN_error_end:
       MOV    A, #BANK_BLC_ERROR  ; Return error number

EBN_end:
       CMP    A,#BANK_BLC_ERROR   ;Bank error ?
       RET

;--------------------------------------------------------------------
; Function name :   ExchangeAddress
; Input :           AX = Address of writing beginning address structure
;                   ( Member of structure ...  Writing starting address
;                                              Bank of writing starting address )
; Output :          AX = Writing starting address( Address in two high rank bytes )
;                   C = Writing starting address( Lower address )
; Summary :         Writing starting address of structure is converted into the
;                   real address from bank information.
;--------------------------------------------------------------------
ExchangeAddress:
       PUSH   HL

; It calculates from 8000H to BFFFH address.
       MOVW   HL, AX
       MOV    A, [HL+0]           ; Read data from writing beginning address
```

```
                                 ; structure( =Write address )
       MOV    X, A
       MOV    A, [HL+1]
       CMPW   AX, #8000H
       BC     $EA_end
       CMPW   AX, #0C000H
       BNC    $EA_end

; Calculation of address( Bank*40H is added to address in two high rank bytes.
; Lower address is the state as it is. )
       MOV    D, A
       XCH    A, X
       MOV    C, A
       MOV    X, #0
       MOV    A, [HL+2]          ; Read data from writing beginning address
structure
                                 ; ( =Bank of writing starting address )
       CMP    A, #6
       BNC    $EA_error_end
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROL    A, 1
       ROLC   A, 1
       XCH    A, X
       ROLC   A, 1               ; AX = 40H*Bank
       XCH    A, X
       ADD    A, D               ; Addition of Higher address
       XCH    A, X
       ADDC   A, #0              ; Addition of Most higher address
                                 ; A ... Most higher address after it calculates
                                 ; X ... higher address after it calculates,
                                 ; C ... Lower address
       BR     EA_normal_end

; Bank error
EA_error_end:
       MOVW   AX, #BANK_ADDR_ERROR
       BR     EA_normal_end

EA_end:
       XCH    A, X
       MOV    C, A
       MOV    A, #0              ; A ... Most higher address after it calculates
                                 ; X ... higher address after it calculates,
                                 ; C...Lower address
EA_normal_end:
       POP    HL
       CMPW   AX, #BANK_ADDR_ERROR       ; Bank error?
       RET


       END
```

A.4 Boot Swap

<boot.asm>

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; System :           Sample program that uses self programming library
;                    ( Bootswap )
; File name :        boot.asm
; Target CPU :       78K0/Kx2
; Last updated :     2005/04/04
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

EXTRN  _FlashStart
EXTRN  _FlashEnd
EXTRN  _FlashEnv
EXTRN  _FlashBlockErase
EXTRN  _FlashWordWrite
EXTRN  _FlashGetInfo
EXTRN  _FlashSetInfo
EXTRN  _CheckFLMD

;---------------------------------------------------------------------
;      EQU settings
;---------------------------------------------------------------------
STATE_OF_ABORT             EQU   1FH    ; State of abort
FLASHFIRM_NORMAL_END       EQU   00H    ; Normal completion

TRUE                       EQU   00H    ; Normal
FALSE                      EQU   0FFH   ; Abnormal
PARAMETER_ERROR            EQU   05H    ; Parameter error

BANKNUMBER                 EQU   0      ; Bank number
BLOCK                      EQU   32     ; Block number

;---------------------------------------------------------------------
;      Stores stack
;---------------------------------------------------------------------
DSTACK DSEG   AT     0FB00H
              DS     80H    ; STACK AREA
STACKINI:

;---------------------------------------------------------------------
;      Sets interrupt vector table
;---------------------------------------------------------------------
VCTTBL        CSEG   AT     0000H
                                  ; addr
       DW     MAIN                ; 00H

S_RAM1 DSEG   AT     0FE20H
```

```
;FlashEnv
       EntryRAM:           DS     100

S_RAM2 DSEG   AT     0FC00H
; FlashWordWrite
; FlashGetInfo
       DataBuffer:         DS     128

S_RAM3 DSEG

; FlashWordWrite
       WordAddr:
       WriteAddress:       DS     2
       WriteBank:          DS     1

; FlashGetInfo
       GetInfo:
       OptionNumber:       DS     1
       FlashGetInfoData:   DS     1

; FlashBlockErase
       EraseBlock:         DS     1

M_PROG CSEG   AT     0400H
;---------------------------------------------------------------------
;      Sample program
;---------------------------------------------------------------------
MAIN:
       DI
       SEL    RB0                 ; Sets to register bank 0.

       MOVW   AX, #STACKINI
       MOVW   SP, AX              ; Sets stack pointer

       MOV    IMS, #0CCH
       MOV    IXS, #00H

       MOV    PCC, #00H

       CLR1   MSTOP
       MOV    OSTS, #05H
MAIN_00:
       NOP
       BF     OSTC.0, $MAIN_00
       CLR1   MCM0
       CLR1   XSEL
MAIN_01:
       NOP
       BT     MCS, $MAIN_01
       CLR1   RSTOP

       MOV    LVIM, #00H
       MOV    LVIS, #00H

       EI

       CALL   !_FlashStart       ; FlashStart( Self programming start library )
                                 ; call processing

;------------------------------------------------------------------
;FlashEnv( Initialization library ) call processing
;------------------------------------------------------------------
       MOVW   AX, #EntryRAM
       CALL   !_FlashEnv         ; Initialization library call

;------------------------------------------------------------------
;CheckFLMD(Mode check library) call processing
;------------------------------------------------------------------
       CALL   !_CheckFLMD        ; Mode check library call

       CMP    A,#TRUE            ; Normal completion?
       BZ     $MAIN_02           ; YES
       BR     MAIN_09

MAIN_02:
;------------------------------------------------------------------
;0000H-0FFFH data is copied to 1000H-1FFFH.
;( Block4-7 is first erased, and 0000H-0FFFH data is written afterwards. )
;------------------------------------------------------------------
       MOV    A, #4
       MOV    !EraseBlock, A

MAIN_03:
; Erase Block4-Block7
       DI

       MOV    A, !EraseBlock
       MOV    B, A
       MOV    A, #0                     ; A ... Erase bank, B ... Erase block number
       CALL   !_FlashBlockErase         ; Erase library call

       EI

       CMP    A, #TRUE                  ; Normal completion?
       BNZ    $MAIN_09                  ; NO
       MOV   A, !EraseBlock
       INC   A
       MOV   !EraseBlock,A
       CMP   A, #8
       BC    $MAIN_03

; Write 0000H-0FFFH data to Block4-Block7
       MOVW  AX, #1000H
       MOVW  !WriteAddress, AX
       MOV   A, #0
       MOV   !WriteBank, A
       MOVW  HL, #0000H
MAIN_04:
       MOV   B, #32*4
       MOVW  DE, #DataBuffer
MAIN_05:
       MOV   A, [HL]
       MOV   [DE], A
       INCW  HL
       INCW  DE
       DBNZ  B, $MAIN_05
       PUSH  HL

       DI

       MOV   A, #32
       MOV   B, A
       MOVW  AX, #WordAddr
       MOVW  HL, #DataBuffer
       CALL  !_FlashWordWrite    ; Word write library call

       EI

       CMP   A, #TRUE            ; Normal completion?
       BNZ   $MAIN_09            ; NO

       POP   HL
       MOVW  AX, !WriteAddress
       ADDW  AX, #128
       MOVW  !WriteAddress,AX
       CMPW  AX, #2000H
       BC    $MAIN_04

MAIN_06:
;------------------------------------------------------------------
;FlashGetInfo call processing( Boot flag information )
;------------------------------------------------------------------
       MOV   A, #04H
       MOV    !OptionNumber, A
       MOVW   AX, #GetInfo
       MOVW   BC, #DataBuffer     ; AX ... Address of flash information
                                  ; acquisition structure,
                                  ; BC ... The first address in buffer where get data
                                  ; is stored
       CALL   !_FlashGetInfo      ; Get information library call

       CMP    A, #TRUE            ; Normal completion ?
       BNZ    $MAIN_09            ; NO

       MOVW   HL, #DataBuffer     ; Get boot flag information
       MOV    A, [HL]
       MOV    !FlashGetInfoData, A

MAIN_07:
;-------------------------------------------------------------------
;FlashSetInfo call processing
;-------------------------------------------------------------------
       DI

       MOV    A, #0FEH
       CALL   !_FlashSetInfo      ; Set information library call

       EI

       CMP    A, #TRUE            ; Normal completion ?
       BNZ    $MAIN_09            ; NO

MAIN_08:
       CALL   !_FlashEnd          ; FlashEnd( Self programming end library )
                                  ; call processing
       ; Normal end

       BR     MAIN_LOOP

MAIN_09:
       CALL   !_FlashEnd          ; FlashEnd( Self programming end library )
                                  ; call processing
       ; Abnormal end

MAIN_LOOP:
       NOP
       NOP
       BR     MAIN_LOOP


       END
```

A.5 Compiling the Flash Self Programming Sample Library and Sample Program

Use the static model sample library and compile options only when using a static model.
Otherwise, use the normal model.

<1> Normal model compile method and options for C

- ra78K0.exe -cF054780 -yC:¥NECTools32¥DEV SelfLibrary_normal.asm
- cc78k0.exe -cF054780 -yC:¥NECTools32¥DEV EEPROMCtrl.c
- cc78k0.exe -cF054780 -yC:¥NECTools32¥DEV Main.c
- lk78K0.exe -yC:¥NECTools32¥DEV -oMain.lmf C:¥NECTools32¥LIB78K0¥s0l.rel -bcl0.lib -s Main.rel SelfLibrary_normal.rel EEPROMCtrl.rel
- oc78K0.exe -yC:¥NECTools32¥DEV Main.lmf

<2> Static model compile method and options for C

- ra78K0.exe -cF054780 -yC:¥NECTools32¥DEV SelfLibrary_static.asm
- cc78k0.exe -sm0 -cF054780 -yC:¥NECTools32¥DEV EEPROMCtrl.c
- cc78k0.exe -sm0 -cF054780 -yC:¥NECTools32¥DEV Main.c
- lk78K0.exe -yC:¥NECTools32¥DEV -oMain.lmf C:¥NECTools32¥LIB78K0¥s0sml.rel -bcl0sm.lib -s Main.rel SelfLibrary_static.rel EEPROMCtrl.rel
- oc78K0.exe -yC:¥NECTools32¥DEV Main.lmf

<3> Normal model compile method and options for assembler

- ra78K0.exe -cF054780 -yC:¥NECTools32¥DEV USER_MAIN.asm
- ra78K0.exe -cF054780 -yC:¥NECTools32¥DEV EEPROM.asm
- ra78K0.exe -cF054780 -yC:¥NECTools32¥DEV SelfLibrary_normal.asm
- lk78K0.exe -yC:¥NECTools32¥DEV USER_MAIN.rel EEPROM.rel SelfLibrary_normal.rel
- oc78K0.exe -yC:¥NECTools32¥DEV USER_MAIN.lmf

<4> Static model compile method and options for assembler

- ra78K0.exe -cF054780 -yC:¥NECTools32¥DEV USER_MAIN.asm
- ra78K0.exe -cF054780 -yC:¥NECTools32¥DEV EEPROM.asm
- ra78K0.exe -cF054780 -yC:¥NECTools32¥DEV SelfLibrary_static.asm
- lk78K0.exe -yC:¥NECTools32¥DEV USER_MAIN.rel EEPROM.rel SelfLibrary_static.rel
- oc78K0.exe -yC:¥NECTools32¥DEV USER_MAIN.lmf

APPENDIX B INDEX