M16C/Tiny Series

Rewriting the User ROM Area in EW0 Mode

R01AN0440EJ0101
Rev. 1.01
Dec 28, 2010

1. Abstract

This application note presents a method for rewriting the flash memory in EW0 mode of the M16C/Tiny-series microcomputers.

2. Introduction

The example application presented here is intended for use in the following type of microcomputer.
Applicable microcomputer: Flash memory version of M16C/Tiny-series microcomputers

This sample program may also be used in other M16C-family microcomputers that have the SFR (Special Function Registers) similar to those in the M16C/Tiny series. However, since it is possible that some functions of your microcomputer will have been altered for functional enhancements, etc., please consult the user's manual for confirmation. Note also that careful evaluation is required before the sample program in this application note can be used.

Table of Contents

3. Explanation of Example Usage

3.1 Overview

The M16C/Tiny series has a special mode known as CPU rewrite mode in which the user ROM area can be rewritten from the CPU by executing software commands. This mode consists of two modes: EW0 and EW1.

- EW0 mode
  Advantages : The CPU continues operating even while programming and erasing.
  : Interrupts generated while programming or erasing are responded quickly.
  (This applies when the interrupt handler routines are located in the RAM.)
  Disadvantages : The flash memory program cannot be executed while programming or erasing.
  (Flash memory cannot be accessed for read.)
  : A large amount of RAM is used.
  (The programming/erasing routines must be located in the RAM.)

- EW1 mode
  Advantages : The amount of RAM used is small.
  (The programming/erasing routines can be located in the flash memory.)
  Disadvantages : The CPU remains idle (in hold state) while programming and erasing.
  : Interrupts generated while programming or erasing cannot be responded quickly.
  (The response time in the M16C/26, for example, is 100 μs typ. during programming and 8 ms max. during erasing.)

In this application note, a description is made of the programming procedure in EW0 mode, showing how to read/write and erase the flash memory.
The procedure described in the following pages will help you understand how to rewrite the flash memory in EW0 mode of the M16C/Tiny-series microcomputers.

- **Rewriting operation specifications (for the M16C/26)**

Rewrite mode	EW0
Operating CPU frequency during rewrite	10 MHz Note
Programming time (2 bytes)	75 μs, typ.
Erase time, 2 Kbyte block	0.2 s, typ.
Erase operation to erase suspend transition time	8 ms, max.

Note: For details about limitations on the on-chip oscillator and PLL, refer to Section 3.5.1, "Auto Program and Auto Erase Procedure."
These limitations apply to flash memory rewrite operation in CPU rewrite mode, and not to normal operation of the microcomputer.

3.2 About Flash Memory

Flash memory is electrically programmable and erasable nonvolatile memory.
The following shows the manner in which the flash memory of the M16C/Tiny series is accessed:

- The flash memory can be programmed in units of one word.
- The flash memory is erased in block units.
- The flash memory cannot be accessed for read during programming and erasing.

3.2.1 Operations on Flash Memory

The following shows operations performed on flash memory.

Table 3-1. Operations and Limitations on Flash Memory

Operation name	Description of operation	Limitation
Read	Means reading out written data.	Any blocks cannot be accessed for read while programming and erasing.
Write	Means changing bit value from 1 to 0.	Only blank addresses can be written to.
Erase	Means changing bit value in the entire block to 1 (changed all to FF_{16}).	Must be performed in block units.

The following shows how to resolve the limitations imposed on data rewrite operation of flash memory.

Table 3-2. Flash Memory Limitations and How to Resolve

Limitation	Solution
Flash memory cannot be accessed for read while programming and erasing. (Programs cannot be run in flash memory.)	Locate the programming/erasing routines in other than the flash memory. (EW0 mode)
	The CPU automatically halts while programming or erasing, unable to read from flash memory. (EW1 mode)
Flash memory can only be erased in block units.	Devise the data retention method so as to reduce the number of times the flash memory needs to be erased. Note

Note: This method is not discussed in this application note.

This application note uses EW0 mode and the software in the RAM to rewrite the flash memory.

3.3 Rewriting the Flash Memory in EW0 Mode

3.3.1 EW0 Mode

EW0 mode allows the user ROM area to be rewritten by issuing the programming (Auto Program) and erasing (Auto Erase) commands from the program located in the RAM. The CPU continues operating even during Auto Program and Auto Erase.
The interrupts whose vectors are accommodated in the variable vector table can be made usable by moving the vectors and the interrupt handler routines into the RAM area.

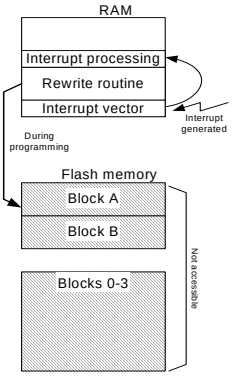

Figure 3-1. Conceptual Diagram of Auto Program Operation

Auto Erase has an “erase suspend” function. The erase suspend function means suspending the auto erase processing during Auto Erase to read data from the flash memory. In order for the erase suspend function to be controlled during EW0 mode, the registers associated with it must be set up in software. For details, refer to Section 3.5, “EW0 Auto Program and Auto Erase Procedure.”
During erase suspend mode, it is possible to call a process located in the flash memory or read data out of the flash memory.

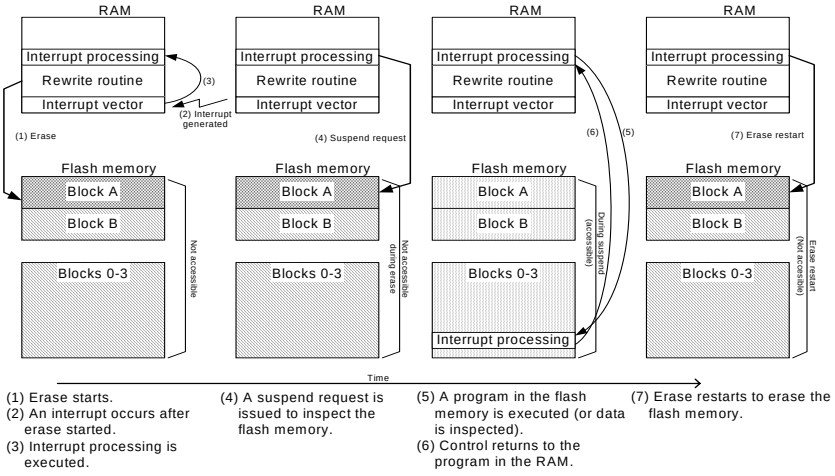

Figure 3-2. Conceptual Diagram of Auto Erase Operation

In EW0 mode, unlike in EW1 mode, the CPU continues operating even during Auto Erase or Auto Program, so that interrupts generated during that time can be responded to as quickly as in normal operation. However, a finite amount of space must be provided in the RAM for the rewrite routine, variable vector table and the relevant interrupt handler routines to be able to work. In addition, program transfer processing is needed.

3.3.2 Flash Memory Mode Transition

The flash memory control registers and software commands are used to control the flash memory in the M16C/Tiny series of microcomputers.
Software commands are generated by writing to the flash memory.

The diagram below shows operation modes of the flash memory during EW0 mode.

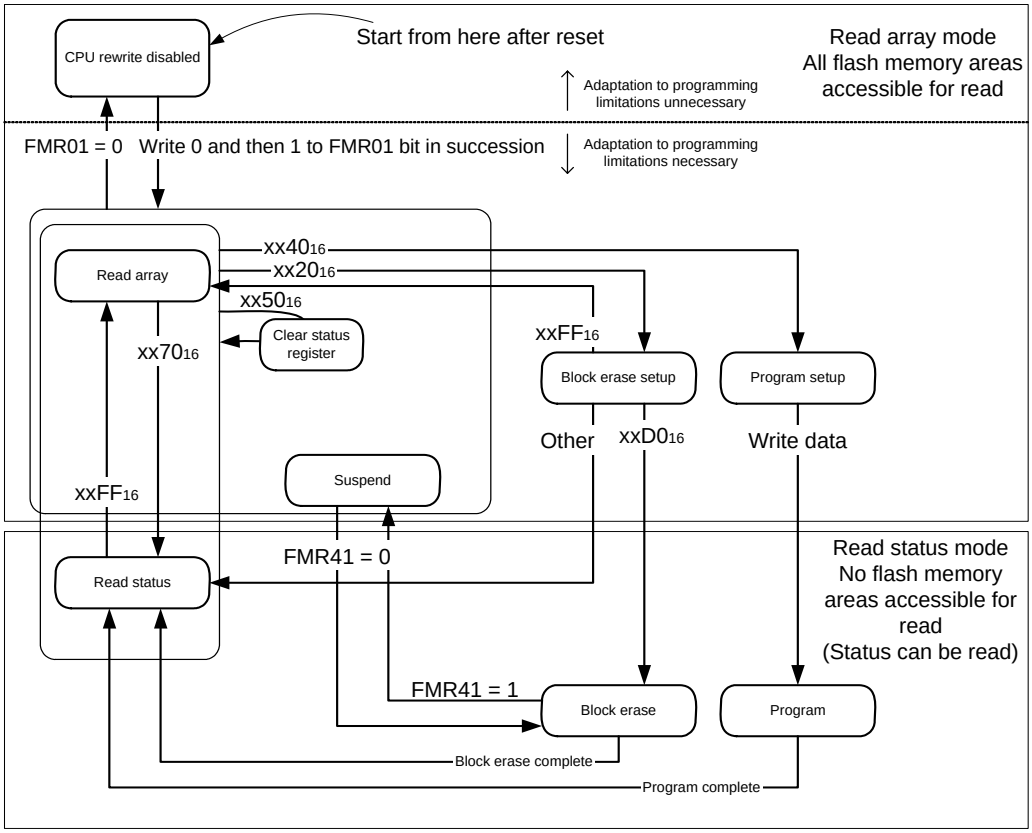

Figure 3-3. Flash Memory Operation Modes

In Figure 3-3, "Flash Memory Operation Modes," transitions of "$xx40_{16}$," etc. are software commands. "FMR41 = 0" means setting the indicated register bit to 0.

During EW0 mode, the flash memory can be placed in either read array mode where the written content can be read or status read mode where the status can be read.

Table 3-3. Flash Memory Read Modes

Read mode	Feature
Read array mode	The written content can be read out.
Read status mode	The status can be read from any area of the flash memory. The status indicates whether the flash memory is operating or in error. (Written content cannot be read out.)

While the flash memory is in read status mode, no programs can be run in the flash memory. This is because the program written to the flash memory cannot be read out during read status mode.

3.3.3 Interrupts during Flash Memory Rewrite

If the erase suspend function is used in EW1 mode, be aware that the erase operation performed differs depending on the type of interrupt used (maskable interrupt or nonmaskable interrupt). During programming too, if interrupts are used in other than byte write, the programming operation performed differs depending on the type of interrupt used, as for the erase operation.
Operational differences due to the type of interrupt are outlined below.

Table 3-4. Auto Erase/Auto Program and Interrupt Operation

Mode	State	When a maskable interrupt is accepted	When a nonmaskable interrupt (NMI interrupt, watchdog timer, oscillation stoppage detection or voltage detection interrupt) is accepted
EW0	During auto erase	Maskable interrupts can be used (and serviced) by locating their vectors in the RAM.	When an interrupt request is accepted, auto erase or auto program is forcibly stopped and the flash memory is reset. When the flash memory restarts a certain time later, the CPU starts processing the interrupt. Since auto erase or auto program is forcibly stopped, correct values may not be read from the block that was being erased or the address that was being programmed. Therefore, execute auto erase/auto program again after the flash memory has restarted and confirm that the operation is terminated normally. The watchdog timer remains idle during command execution.
	During auto program		

Note 1: The vector for the address match interrupt is located in the ROM. Do not use it while executing an erase/program command.
Note 2: Since the fixed vectors are located in block 0, do not use nonmaskable interrupts while auto-erasing block 0.

3.4 EW0 Auto Program and Auto Erase Procedure

3.4.1 Auto Program and Auto Erase Procedure

The following shows an auto program flowchart following the procedure described in the manual. This flowchart is shown for comparison with the auto program procedure for the RAM-reduced version, and not a flowchart for the sample program. For details about the sample program flowchart, refer to Figure 3-11, “RAM Reduced Version Auto Program Flowchart”.

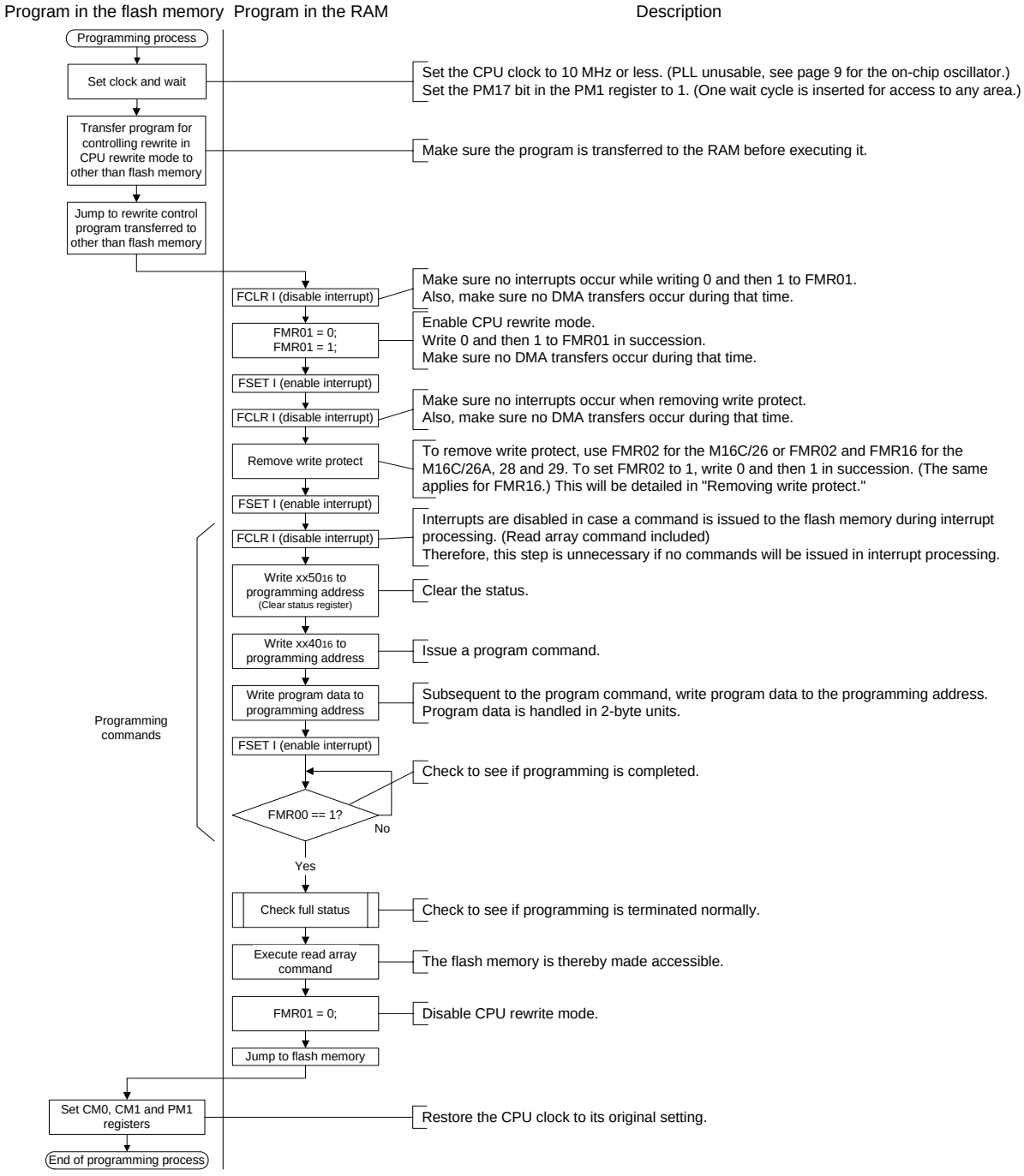

Figure 3-4. Auto Program Flowchart (Reference)

The following shows an auto erase flowchart following the procedure described in the manual.
This flowchart is shown for comparison with the auto program procedure for the RAM-reduced version, and not a flowchart for the sample program. For details about the sample program flowchart, refer to Figure 3-12, “RAM Reduced Version Auto Erase Flowchart”.

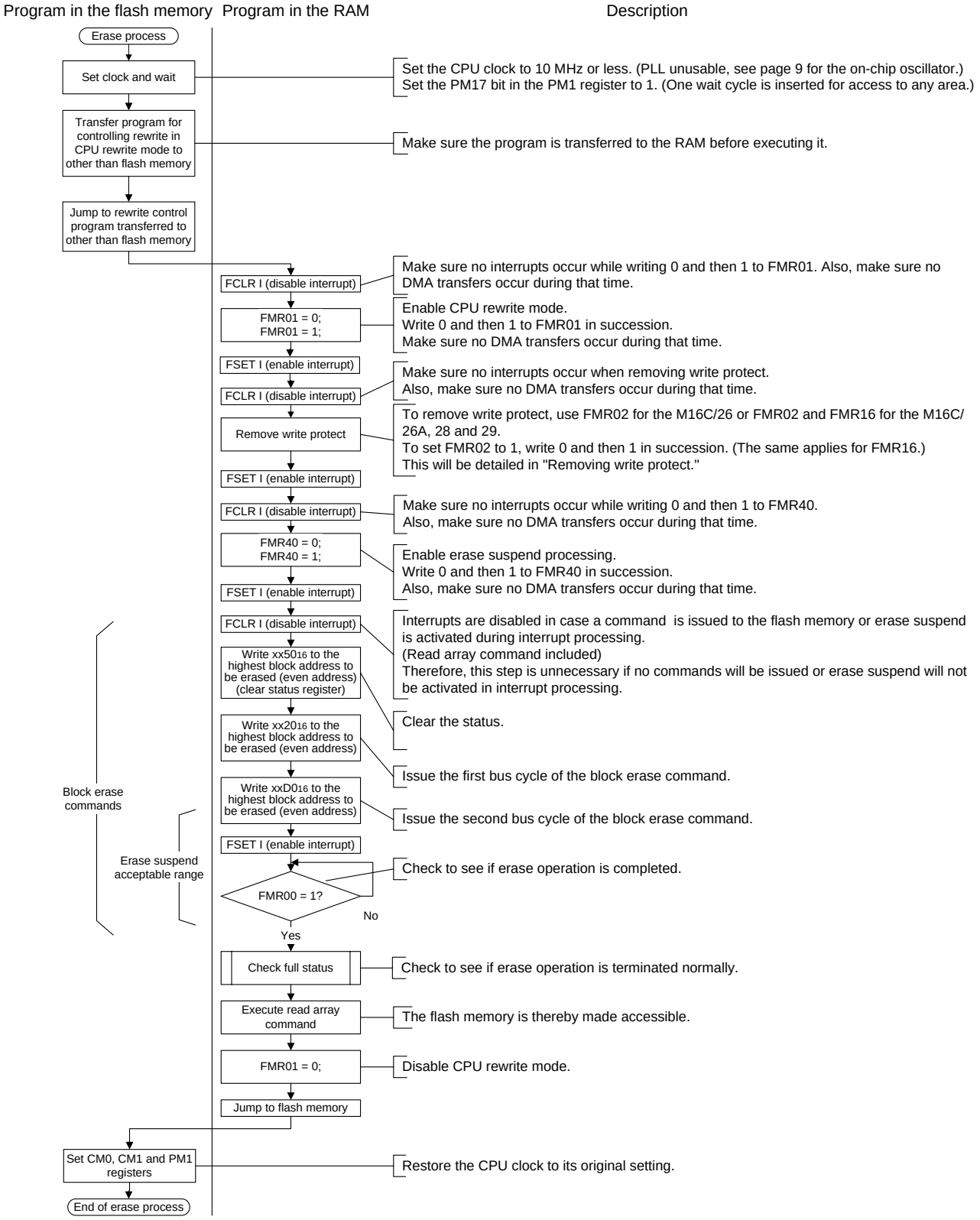

Figure 3-5. Auto Erase Flowchart (Reference)

The following shows an erase suspend flowchart.

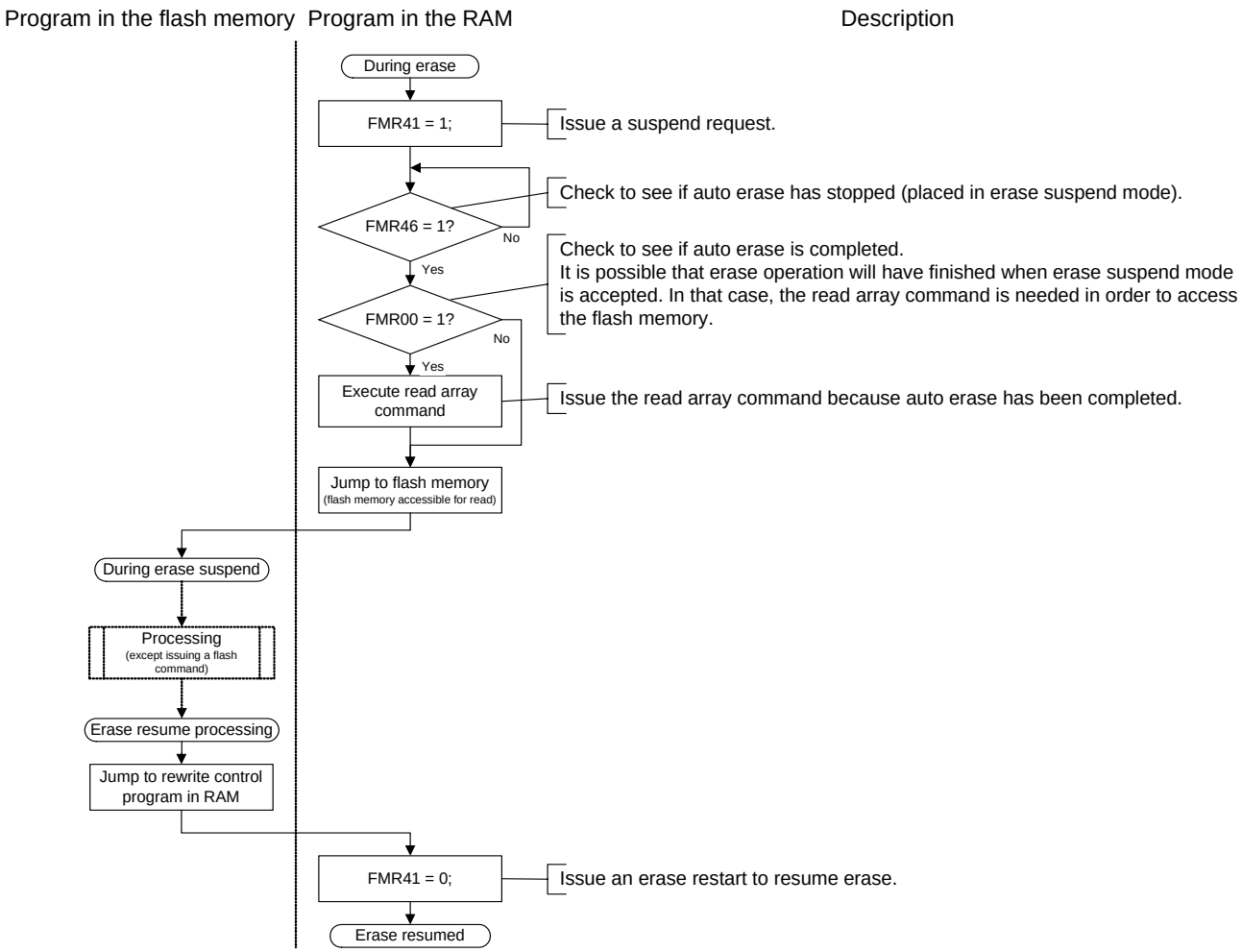

Figure 3-6. Erase Suspend Processing Flowchart

The flowcharts shown in the preceding pages are described in detail below.

- Setting clock and wait

When the flash memory is placed in CPU rewrite mode (by setting FMR01 to 1), the following limitations apply.

・A wait cycle must be inserted for access to any ROM and RAM area (by setting PM17 to 1).

・The CPU clock must satisfy the conditions below.

Table 3-5. Clock Limitations for CPU Rewrite Mode

Operating clocks	Limitations	Remarks
Main clock	10 MHz or less	Set by CM0 and CM1.
On-chip oscillator (M16C/26A, 28 and 29)	f1 (ROC), f2 (ROC) or f3 (ROC) and the ROCR register is set to divide by 4 or 8.	Set by ROCR.
PLL clock (M16C/26A, 28 and 29)	Unusable	Change from the PLL clock to the main clock using the system clock control bit (CM11) in System Clock Control Register 1 (CM1). There is no need to stop the operation of the PLL frequency synthesizer.

Furthermore, if blocks A and B are rewritten 100 times or more, blocks A and B must be accessed for read with one wait cycle by setting FMR17 to 1, even in other than CPU rewrite mode.
These are summarized below.

Table 3-6. CPU Rewrite Mode and Limitations

State		Clock limitations	Use of PLL	Wait
CPU rewrite mode enabled	Read from any block Issuance of software command	Yes	Unusable	Necessary
CPU rewrite mode disabled	Read from blocks A and B (Rewritten less than 100 times)	No	Usable	Unnecessary
	Read from blocks A and B (Rewritten 100 times or more)	No	Usable	Necessary
	Read from any blocks other than A and B	No	Usable	Unnecessary

Limitations on the CPU clock can be lifted during erase suspend. To remove limitations on the CPU clock, reset the CPU rewrite mode select bit (FMR01) to 0 (disable) during erase suspend. The erase suspend state is retained intact even when CPU rewrite mode is disabled.
To resume auto erase, set up registers, etc. to meet conditions for CPU rewrite mode, enable CPU rewrite mode, and then clear the erase suspend request bit (FMR14) to 0 (erase restart).
An example operation is shown below.

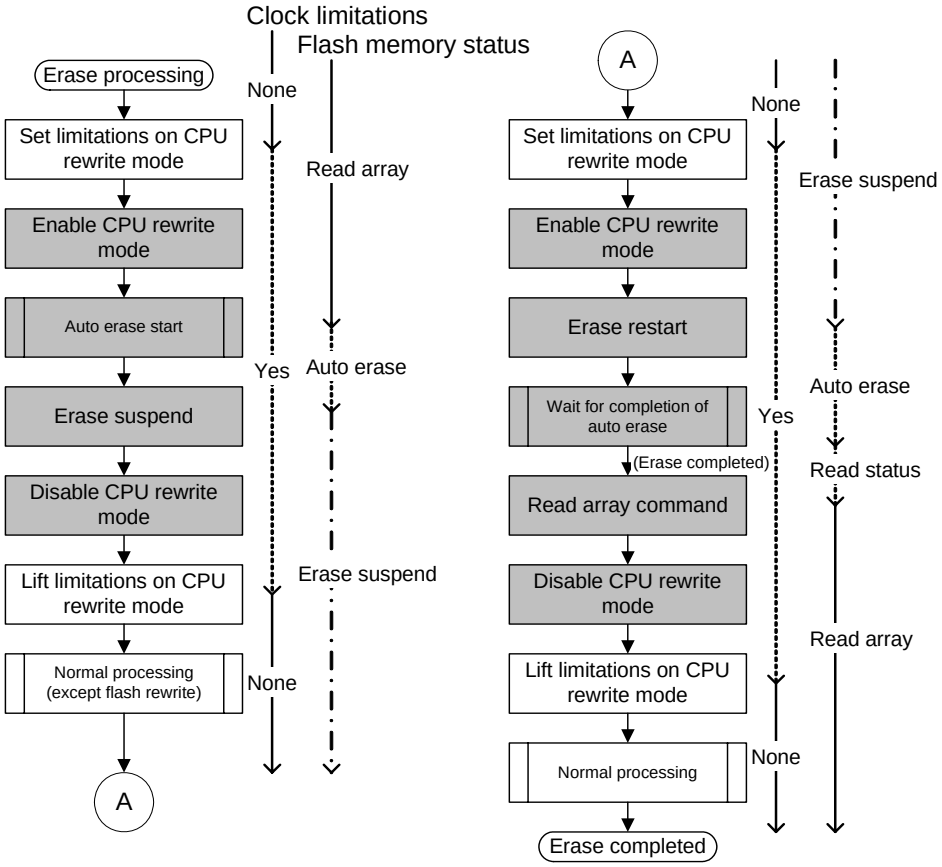

Figure 3-7. Clock Control during Erase Suspend

- Transferring a program to the RAM area
  Refer to Section 3.7, “About Programs Running in the RAM Area.”

- Entering CPU rewrite mode

Set bit 1 (FMR01) in Flash Memory Control Register 0 (FMR0).
To set the FMR01 bit to 1, write 0 and then 1 in succession. Make sure no interrupts or DMA transfers occur before writing 1 after writing 0.

- Removing write protect

Some areas must have their write protect removed before they can be rewritten.
To remove write protect, the FMR02 and FMR16 bits (M16C/26A, 28 and 29) must be set.
The tables below show how write protect is set for each type of microcomputer.

Table 3-7. Write Protect Set for the M16C/26

Register settings	Rewrite areas		
FMR02	Blocks A and B	Blocks 0 and 1	Other blocks
0	○	×	○
1	○	○	○

○: Rewritable ×: Not rewritable (write protected)

Table 3-8. Write Protect Set for the M16C/26A, 28 and 29

Register settings		Rewrite areas		
FMR16	FMR02	Blocks A and B	Blocks 0 and 1	Other blocks
0	0	○	×	×
0	1	○	×	×
1	0	○	×	○
1	1	○	○	○

○: Rewritable ×: Not rewritable (write protected)

The following shows how to set FMR02 and FMR16.

Table 3-9. How to Set FMR02 and FMR16

Bit name	If FMR01 bit = 0	If FMR01 bit = 1
FMR02	Always 0	Can be set. To set the bit to 1, write 0 and then 1 in succession. Note
FMR16	Cannot be set. (Value retained)	Can be set. To set the bit to 1, write 0 and then 1 in succession. Note

Note: Make sure no interrupts or DMA transfers occur before writing 1 after writing 0.

The data areas (blocks A and B) have an access enable bit (PM10). Set the PM10 bit to 1 when accessing the data area for read. When CPU rewrite mode is enabled (FMR01 = 1), the PM10 bit is automatically set to 1. For this reason, blocks A and B do not have write protect.
For details about the PM10 bit, refer to Section 3.6, “Readout Procedure.”

As shown in Table 3-9, the FMR02 bit retains its value only when CPU rewrite mode is enabled (FMR01 bit = 1), whereas the FMR16 bit always retains its value regardless of whether CPU rewrite mode is enabled or not.
In the sample program, therefore, FMR02 and MFR16 are set in the processing described below.

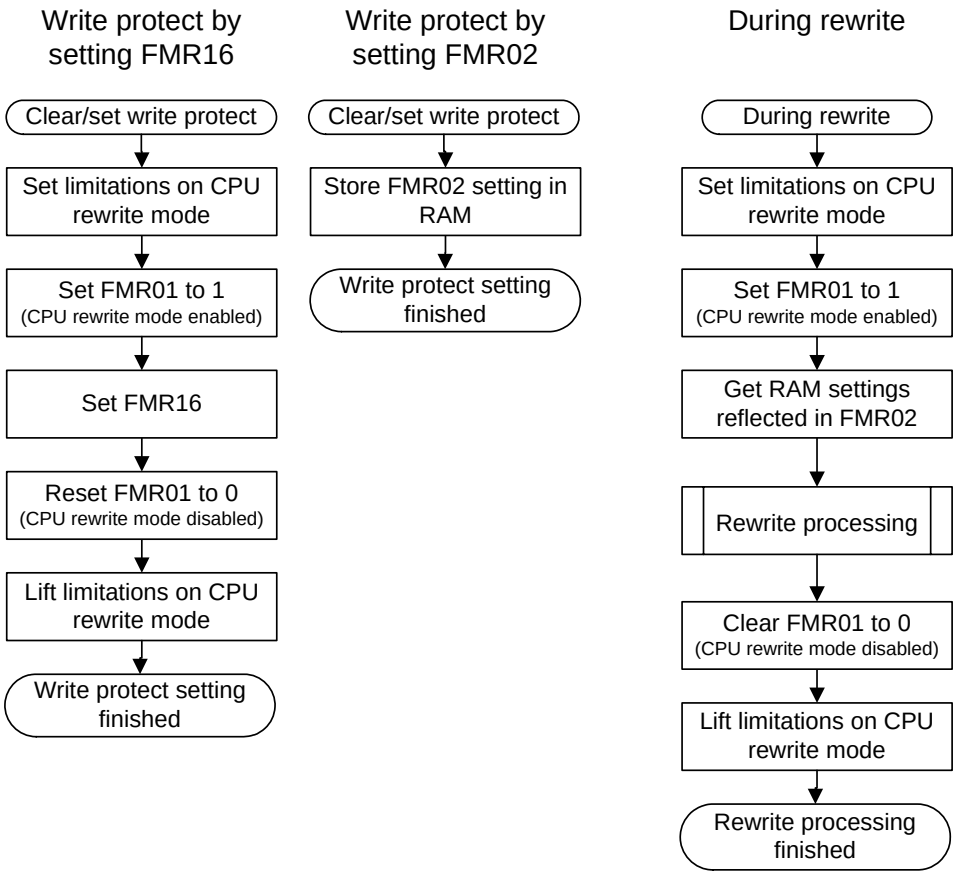

Figure 3-8. Write Protect Settings in the Sample Program

In the sample program, the “write protect setting” interface function and the “rewrite” interface function are separated.
The FMR16 bit has its value retained even when CPU rewrite mode is disabled (FMR01 bit = 0). On the other hand, the FMR02 bit has its value not retained and changed to 0 when CPU rewrite mode is disabled (FMR01 bit = 0).
The FMR16 bit is set in the “write protect setting” interface function. The FMR02 bit is set up back again from its last set content during rewrite.
Before the “rewrite” interface function can be called, the “write protect setting” interface function must be called in order to remove write protect. The write protect information once set remains effective unless it is set again in the “write protect setting” interface function.

- Program command

This command writes data to the flash memory one word (two bytes) at a time.
When the program command is issued, the CPU auto-programs the flash memory (by writing program data and verifying).
The auto program operation is depicted below.

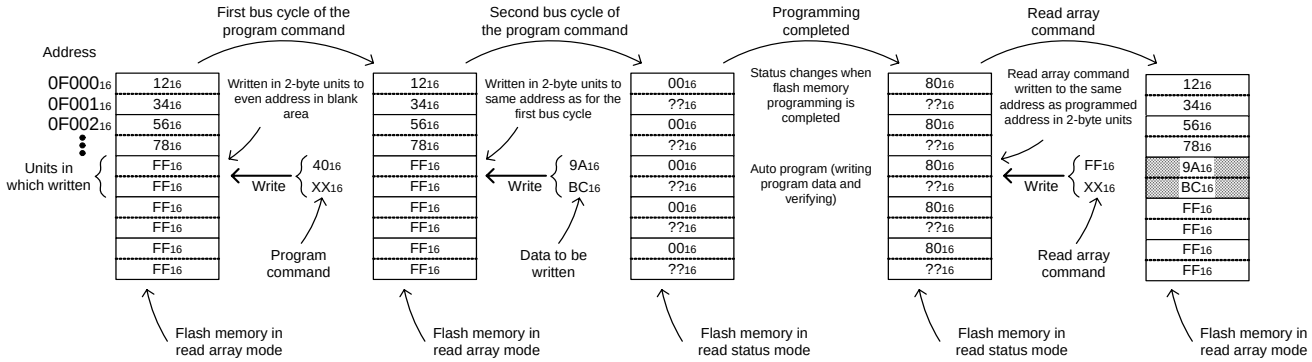

Figure 3-9. Auto Program Procedure

Make sure the command is written in 16-bit units to the even addresses in the user ROM area.
For details about the change of flash memory modes after the CPU started programming the flash memory, refer to Figure 3-3, “Flash Memory Operation Modes.”

- Erase command

This command erases data from the flash memory in block units.
When the erase command is issued, the CPU auto-erases the flash memory (by erasing data and verifying).
The auto erase operation is depicted below.

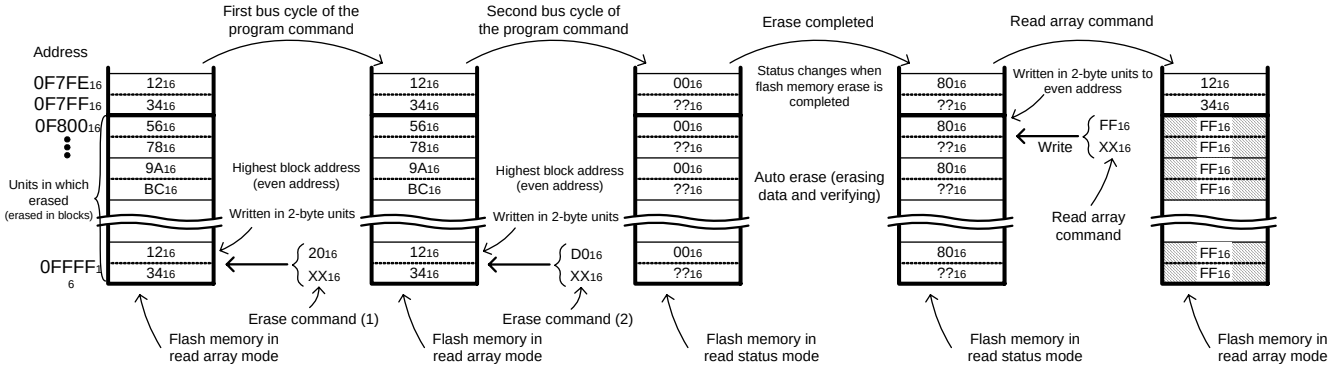

Figure 3-10. Auto Erase Procedure

Make sure the command is written in 16-bit units to the even addresses in the user ROM area.
For details about the change of flash memory modes after the CPU started erasing the flash memory, refer to Figure 3-3, “Flash Memory Operation Modes.”

- Full status check

To confirm whether auto program or auto erase has been executed normally, inspect the FMR06 and FMR07 bits.
The table below shows the relationship between the status register status and errors.

Table 3-10. Status of the FMR0 Register and Errors

Status of the FMR00 register (status register)		Error	Error occurrence conditions
FMR07	FMR06		
1	1	Command sequence error	・When the command was not written correctly ・When invalid data (not "xxD0$_{16}$" or "xxFF$_{16}$) was written in the second bus cycle of the block erase command Note 1 ・When the block erase command was executed on a write protected block. ・When the program command was executed on a write protected block.
1	0	Erase error	・When the block erase command was executed on a write "enabled" block, but the block was not auto-erased correctly
0	1	Program error	・When the program command was executed on a write "enabled" block, but the block was not auto-programmed correctly
0	0	No error	Successfully executed.

Note 1: Writing "xxFF$_{16}$" in the second bus cycle of these commands places the flash memory into read array mode, in which case the command code written in the first bus cycle is invalidated.

- Read array command

This command places the flash memory into read array mode.
In read array mode, the content recorded in the flash memory can be read out.
The flash memory is placed into read array mode by writing "xxFF16" in the first bus cycle. Then, when the address from which to read is entered in the next or the subsequent bus cycle, the content of the specified address can be read out in 16-bit units.
Read array mode is retained intact until another command is written to the flash memory, the contents of multiple addresses can be read out successively.

- Disabling CPU rewrite mode

Reset bit 1 (FMR01) in Flash Memory Control Register 0 (FMR0) to 0.

- Erase suspend

To use the erase suspend function in EW0 mode, set the erase suspend request bit (FMR41 bit in the FMR4 register) to 1, and then check the erase status bit (FMR46 bit in the FMR4 register) to see that the flash memory is placed into erase suspend mode.
The FMR46 bit is 0 during auto erase operation, and set to 1 when auto erase has stopped (placed into erase suspend mode).
If the flash memory is placed into erase suspend mode at the same time auto erase has finished, issue the read array command before accessing the flash memory.
During erase suspend mode, it is possible to call a process residing in the flash memory or read data out of the flash memory.
To resume auto erase, clear the erase suspend request bit (FMR41 bit in the FMR4 register) to 0 from the program in the RAM.

- About differences between microcomputers in the M16C/Tiny series

  The M16C/26, 26A, 28 and 29 differ in the following points with respect to the flash memory related features.

Table 3-11. Differences between the M16C/26, 26A, 28 and 29 with respect to the flash memory related features

Item	M16C/26	M16C/26A	M16C/28	M16C/29
Write protect	Write protected by FMR02	Write protected by FMR16 and FMR02	←	←
FMR16 included or not	Not included	Included	←	←
PLL included or not, and PLL based rewriting	PLL not included	PLL based rewriting prohibited	←	←

3.4.2 Reducing the RAM Size

To reduce the RAM size used by a program, separate processes into those that can be and those that cannot be run in the flash memory area, and then move the processes that do not need to be run in the RAM into the flash memory area.

It is only when the flash memory is in read status mode (including cases where auto program or auto erase is underway) that processes need to be run in the RAM.

Alter the following contents from the procedure described in Section 3.5.1, "Auto Program and Auto Erase Procedure," to reduce the RAM size.

When programming the flash memory

Run all processing up to immediately preceding the first bus cycle of the program command in the flash memory.
Perform operations from the first bus cycle of the program command to check for completion of programming in the RAM. (No changes needed)
To perform full status check, place the flash memory back into read array mode prior to the full status check.
Perform full status check in the flash memory.

When erasing the flash memory

Run all processing up to immediately preceding the first bus cycle of the erase command in the flash memory.
Perform operations from the first bus cycle of the erase command to check for completion of auto erase in the RAM. (No changes needed)
To perform full status check, place the flash memory back into read array mode prior to the full status check.
Perform full status check in the flash memory.

Next, a flowchart with the above contents incorporated will be shown.

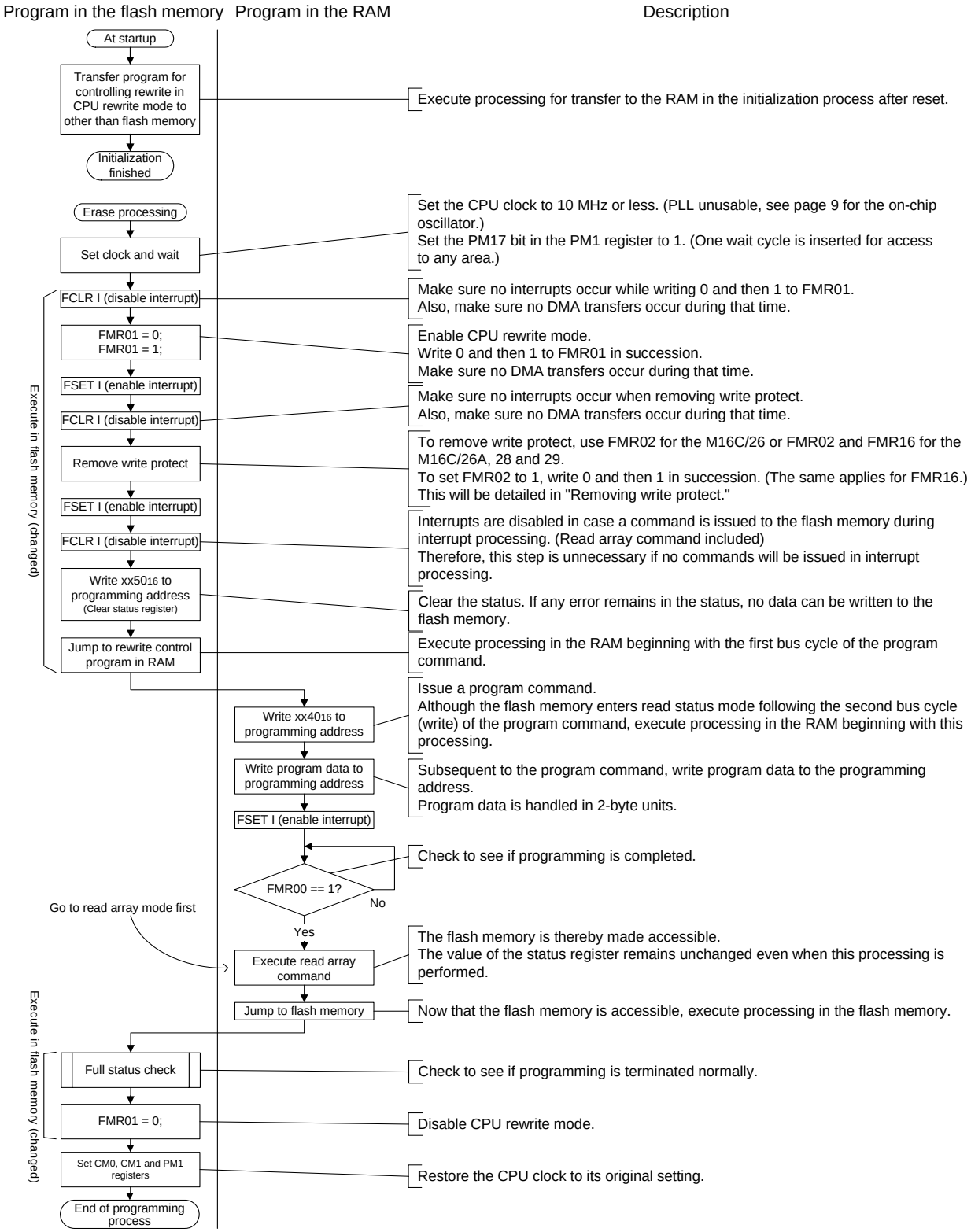

Figure 3-11. Auto Program Flowchart for the RAM Reduced Version

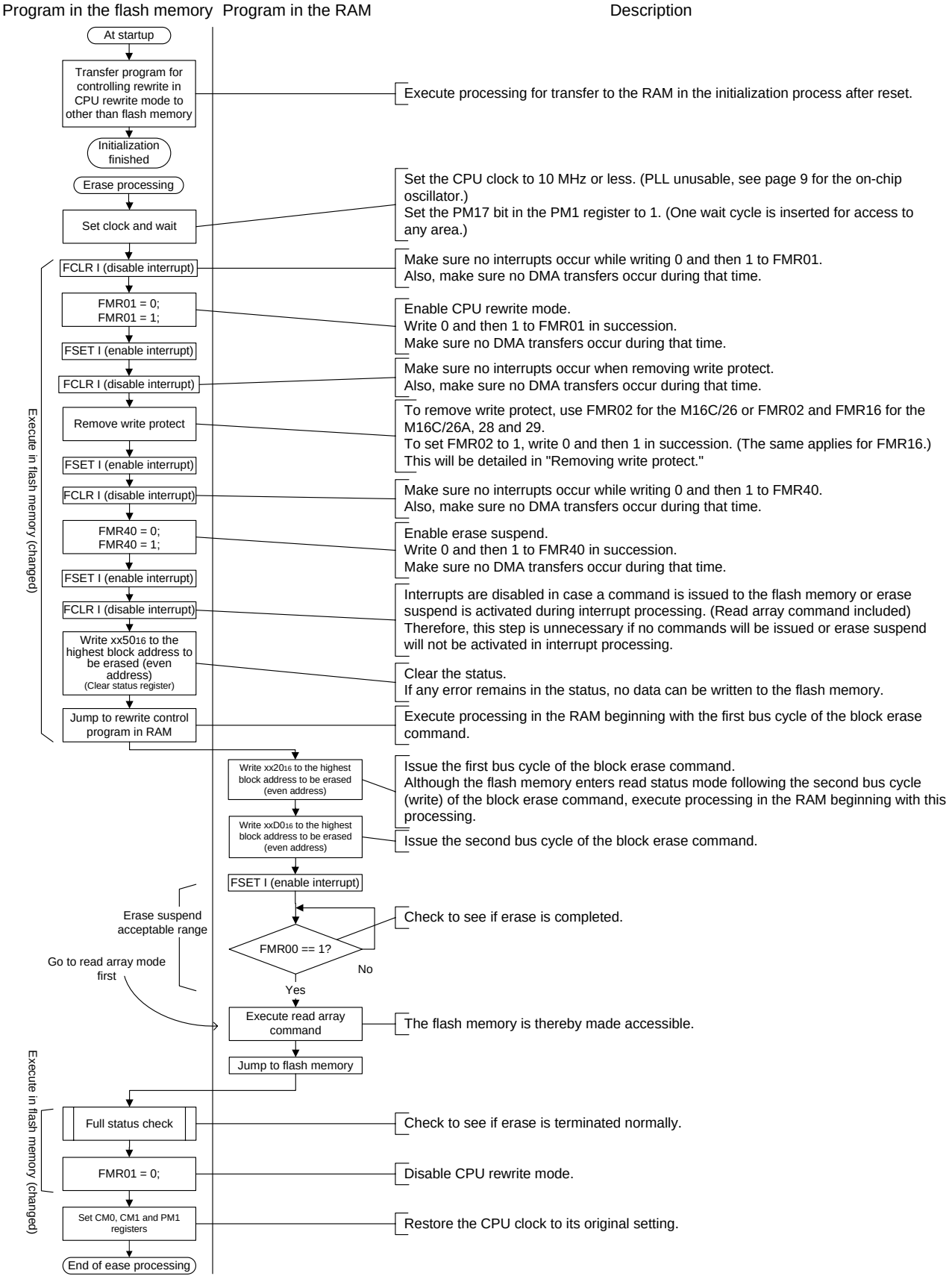

Figure 3-12. Auto Erase Flowchart for the RAM Reduced Version

In the sample program, the auto erase procedure consists of the processing shown above plus erase suspend processing based on the erase suspend function.

3.5 Readout Procedure

The flash memory can be accessed for data readout during read array mode.
Writing "xxFF$_{16}$" in the first bus cycle places the flash memory into read array mode. (When the program is running in the flash memory, read array mode is already entered into. When the flash memory is being rewritten following the procedure described in this application note, it is placed into read array mode when a programming process is completed.)

Furthermore, before data can be read from the data area (block A or B), the data area access enable bit (PM10) in Processor Mode Register 1 (PM1) must be set to 1.

About the PM10 bit

- Set the PRC1 bit in the PRCR register to 1 (write enabled) before rewriting the PM10 bit.
- When CPU rewrite mode is enabled (FMR01 = 1), the PM10 bit is automatically set to 1.

Table 3-12. PM10 Bit and the Status of Blocks A and B

PM10	Status of blocks A and B
0	Cannot be read (Always PM10 = 1 during rewrite)
1	Can be read Can be rewritten

In the sample program, the PM10 bit is set to 1 in the device initialization process.

3.6 Programs Running in the RAM Area

The sample program includes some programs that run in the RAM.
In the example below, the program data is stored in $0FB000_{16}$ and is executed in the RAM.

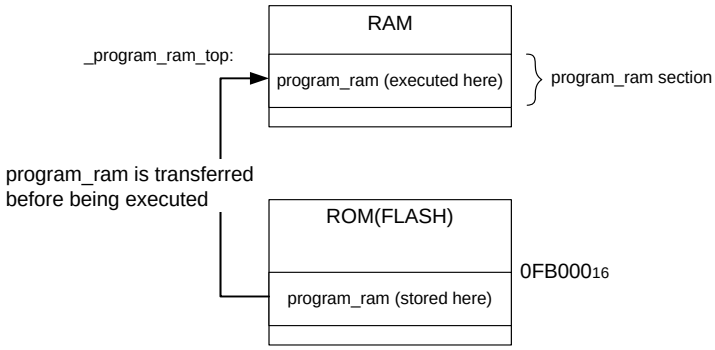

Figure 3-13. Program Location Diagram

3.6.1 Altering the Section Name

The following explains a method for adding another section named “program_ram” and locating in it a program that is to be run in the RAM.

To relocate the program from the program section into the program_ram section, write a program statement as follows:

```
void ThisProgramOnROM(void)
{
      /* This program is located in the program section. */
}
/* Defined after the #program SECTION declaration are located in
program_ram. */
#pragma SECTION program program_ram
void ThisProgramOnRAM(void)
{
      /* This program is located in the program_ram section. */
}
```

Next, add to sect30.inc the program_ram section and a label description _program_ram_top at the beginning.
The position at which the program will be run is determined by sect30.inc. (In the example here, it is located after heap section.)
Note that _program_ram_top is used to transfer the program.

```
;------------------------------------------------------
;     heap section
;------------------------------------------------------
      .section    heap,DATA
heap_top:
      .blkb HEAPSIZE
```

Insert here.

A section to be inserted

```
;------------------------------------------------
; RAM program area
;------------------------------------------------
      .section    program_ram,ALIGN
_program_ram_top:
.glb  _program_ram_top
```

3.6.2 Transferring the Program

Next, write a program statement to transfer the program from the ROM to the RAM.
In the example explained below, the program is transferred at startup.

Use the N_BCOPY macro to transfer the program from the far area to the near area or the BCOPY macro to transfer to the far area. (The N_BCOPY and BCOPY macros are defined in ncrt0.a30.)
Add the transfer processing to the C language startup routine ncrt0.a30 by using this macro.
The following shows an example in which N_BCOPY is used.

crt0.a30

```
;=====================================================================
; Program RAM Initialize
;     _from_addr is specified in the option -D_from_addr=0FB000h
;---------------------------------------------------------------------
    N_BCOPY _from_addr,_program_ram_top,program_ram
```

3.6.3 Specifying Locations where the Program is Stored and Executed

The compiler must be directed to allocate separate addresses for the program at which it is to be stored and executed.
This can be accomplished by specifying an option in ln30 as shown below.

```
ln30 -LOC program_ram=0FB000
```

The above linker option -LOC specifies the address at which the data in the specified section should be stored by specifying the location of the program_ram section (program to be stored at the address $0FB000_{16}$).

Note: This option "-LOC" and the related option "-ORDER" cannot be specified in -ln30 for NC30. Make sure the option is set in ln30. The programs relocated by ALIGN specification may not always run normally. Therefore, if the beginning address of a relocated section is an even address, set an even address for the location to which the program is to be transferred. The same applies to odd addresses. (In the example here, the address specified by -LOC is set to an even address, and the ALIGN option is specified for the program_ram section, so that both sections will start on an even address.)
This option only specifies the registered address of a set section, and does not have a function to transfer the program to its run-time address area. To call a process in the set section, be sure to transfer the desired process to the address set by the program before calling it.

3.6.4 Setting Vector Addresses in the RAM Area

Locate the interrupt vectors in the RAM.

Here, explanation is made of the following operations:

- Reserve storage for the variable vector table in the RAM.
- Copy the necessary variable vectors from vector_table which is included as a standard component.
- Copy them to the variable vectors in the RAM at startup..

First, in sect30.inc, reserve 4 bytes x 64 = 256 bytes for the vector area in the RAM.

```
;------------------------------------------------------------------
; RAM vector area
;------------------------------------------------------------------
    .section vector_ram,data,ALIGN
    .ALIGN
_vector_table:
    .blkl   64
.glb    _vector_table
```

At startup, copy the default interrupt vectors to the interrupt vectors in the RAM to alter the vector addresses. Add this setting to the startup routine ncrt0.a30.

```
;------------------------------------------------------------------------
; vector area
;------------------------------------------------------------------
    N_BCOPY VECTOR_ADR,_vector_table,vector_ram
    ldc  #(_vector_table >> 16),INTBH
    ldc  #(_vector_table & 0FFFFh),INTBL
```

Note that since the contents described in Section 3.7, “About Programs Running in the RAM Area,” all are written in the sample program, please refer to the sample program for details about the entire description.

4. Sample Program

The sample program explained here includes the content described in Section 3.5.2, “Reducing the RAM Size.”
The sample program is written as a flash memory device driver.
Following are defined as the driver interfaces.
If an error occurs during flash memory auto program or auto erase, error code is returned. In such a case, please execute the processing described in the manual. (After executing the clear status command, reexecute auto program for a programming error or auto erase for an erase error up to three times repeatedly.)

Table 4-1. Function Table

Function Name	Description	Remark
StartEraseFlash()	Starts erasing flash memory.	Unusable in an interrupt. Interrupts are controlled using the I flag internally. If interrupts need to be disabled in order to use this function, disable interrupts in the IPL.
RestartEraseFlash()	Resumes suspended erase.	Unusable in an interrupt. Interrupts are controlled using the I flag internally. If interrupts need to be disabled in order to use this function, disable interrupts in the IPL.
WriteFlash()	Writes to flash memory.	Unusable in an interrupt. Interrupts are controlled using the I flag internally. If interrupts need to be disabled in order to use this function, disable interrupts in the IPL.
ReadFlash()	Reads from flash memory.	
UnlockBlockFlash()	Unlocks flash memory from write protect.	
LockBlockFlash()	Locks flash memory to write protect.	
SuspendErase()	Issues an event requesting that flash memory erase be suspended	If this function is called after calling the StartEraseFlash() function, StartEraseFlash() suspends the erase operation and returns with F_SUSPEND. Thereafter, the erase operation is resumed by RestartEraseFlash(). (At this time, the erase operation will be suspended again by SuspendErase().)
ResumErase ()	Cancels an event requesting that flash memory erase be suspended	If this function is called after calling the SuspendErase() function, the request to suspend erase operation issued by SuspendErase() is canceled.
SuspendFlash()	Suspends erasing flash memory.	Used for interrupts only. If this function is called while the flash memory is being erased, the flash memory goes to an erase-suspended state. After return from this function, the flash memory can be accessed for read.
ResumFlash()	Cancels suspended erase of flash memory	Used for interrupts only. This function is called only when the flash memory placed in an erase-suspended state by SuspendFlash() needs to be erased again from where it stopped from the beginning.

4.1 File Configuration

The sample program is comprised of the files listed below.

Table 4-2. File Configuration Table

File Name	Description
flashdevconf.h	This file sets up the driver.
flashdevdrv.h	This header file is included when using the driver.
flashm16c.h	This is the include file in the flash memory driver for the M16C type dependent part.
flashdrvdev_ew0.c	This is the EW0 mode flash memory driver file.
depend_m16c.c	This is the source of the flash memory driver for the M16C type dependent part.
ncrt0_EW0.a30	This is a C language initialization file. It is an upgraded version from the standard file (ncrt0.a30) with the RAM transfer processing at startup added.
sect30_EW0.inc	This is a C language section file. It has had a new program section that runs in the RAM added from the earlier version.
sfr26.h sfr26a.h sfr28.h sfr29.h	These are include files for the M16C/26, 26A, 28 and 29. Please be sure to obtain the latest file.
main_m16c.c	This file contains a sample program usage example.
M16C_EW0.tmk	This is the Makefile. (Specify make -f M16C_EW0.tmk to compile it.)

4.2 Program Operation

This section explains operation of the sample program as flash memory driver.
The driver is always used when reading or writing to the flash memory, as well as when erasing the flash memory.
During read or write, the flash memory cannot be accessed for data readout (fetch or read).
The driver is assumed to be run in the RAM.
A sequence flow of the driver during auto erase is shown below.

4.2.1 Auto Program and Auto Erase Operations

The following shows an example driver operation using auto program and auto erase APIs.

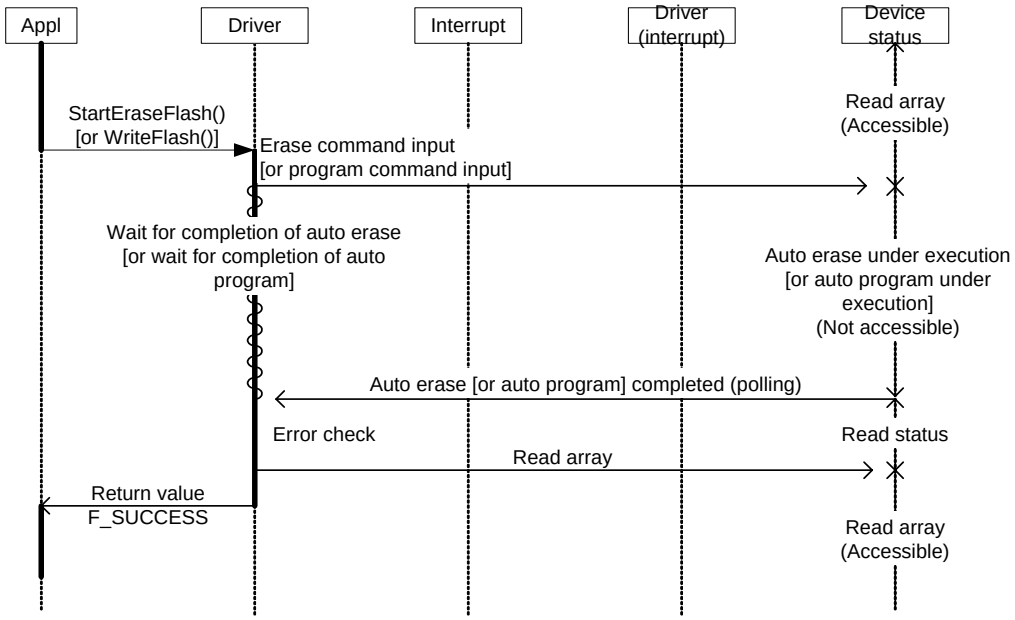

Figure 4-1. Operation of Auto Program and Auto Erase APIs -1

The auto program and the auto erase APIs return F_SUCCESS when the respective operations are successfully completed.

4.2.2 Erase Suspend Processing

The SuspendErase() function suspends the erase processing being executed by StartEraseFlash() or RestartEraseFlash(). The operation sequence is shown below.

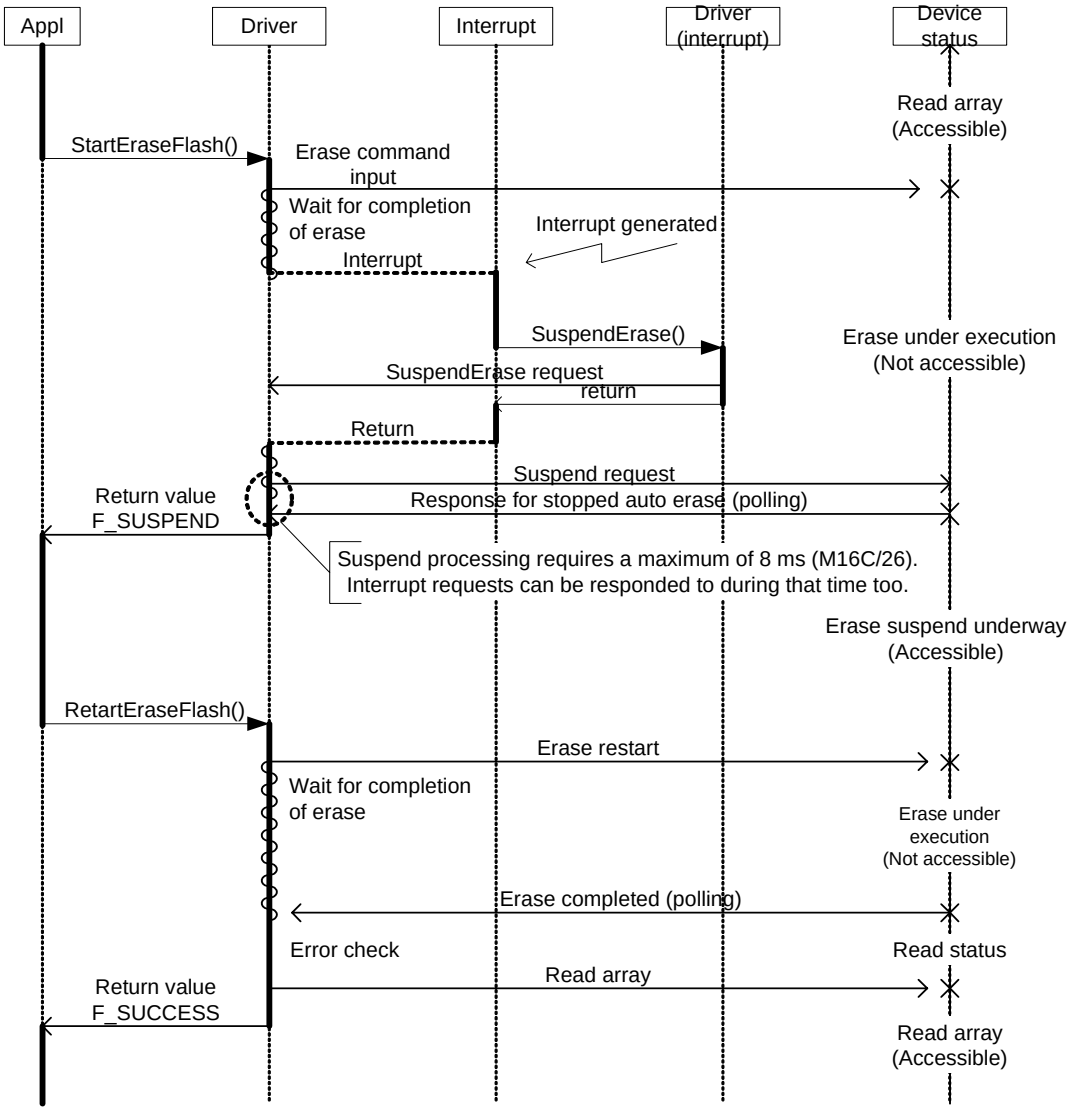

Figure 4-2. Erase Suspend Processing Sequence Flow -1

Called in an interrupt, SuspendErase() only issues a request to the driver, and does not perform any operation on the flash memory. The flash memory cannot be accessed for read during an interrupt. When a suspend request is issued, a finite time of up to 8 ms is required (for the M16C/26) until auto erase of the flash memory stops.

For details on how to read data from the flash memory during an interrupt, refer to Section 4.2.3, “Flash Memory Read during Interrupt Processing.”

The request from SuspendErase() processing holds the currently erase-suspended call too. ResumeErase() may be used to avoid it. A sequence is shown below.

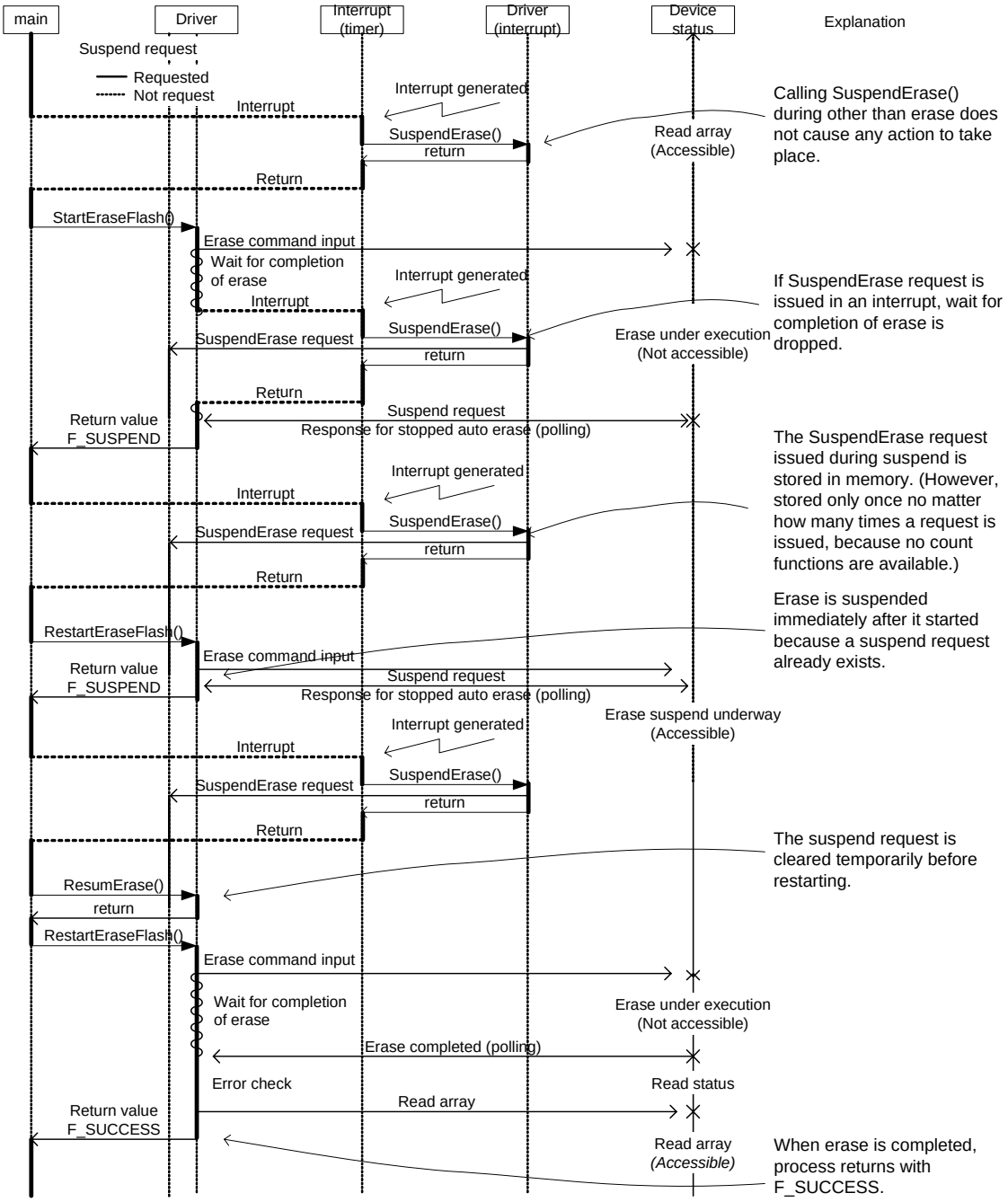

Figure 4-3. Erase Suspend Processing Sequence Flow -2

Although the erase interface consists of synchronous functions, use of the SuspendErase() function allows for return from the erase interface.
As in Figure 4-3, “Erase Suspend Processing Sequence Flow -2,” SuspendErase() processing may be periodically invoked using a timer in order to prevent the CPU from being occupied by StartFlashErase() or RestartFlashErase().

4.2.3 Flash Memory Read during Interrupt Processing

Use the SuspendFlash() function to read data from the flash memory in an interrupt process during auto erase or auto program. Use ResumFlash() to restart auto erase after reading data from the flash memory.

The operation sequence is shown below.

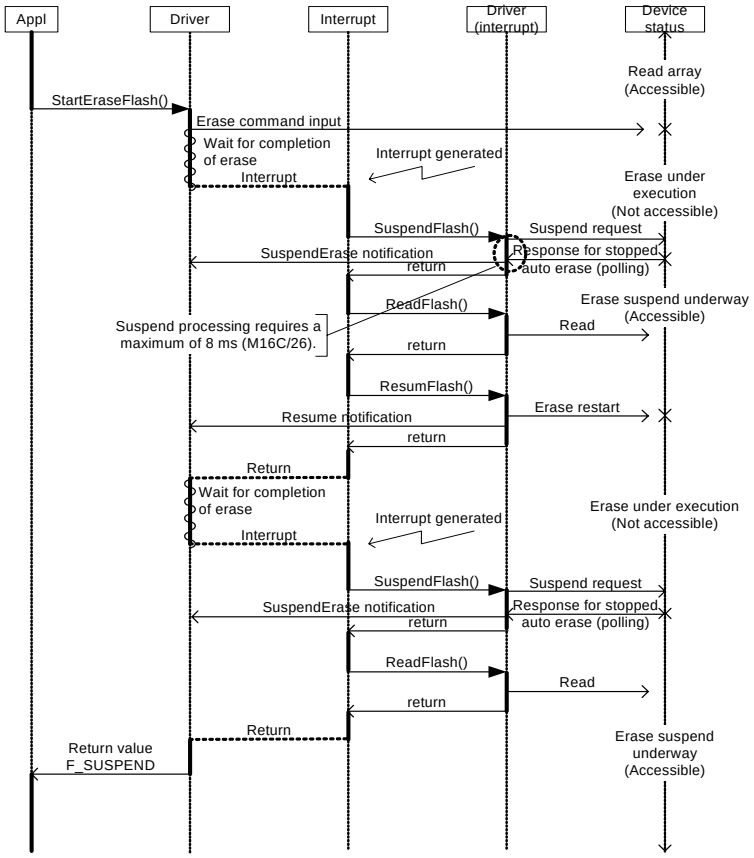

Figure 4-4. Flash Memory Read during Interrupt Processing (Auto Erase)

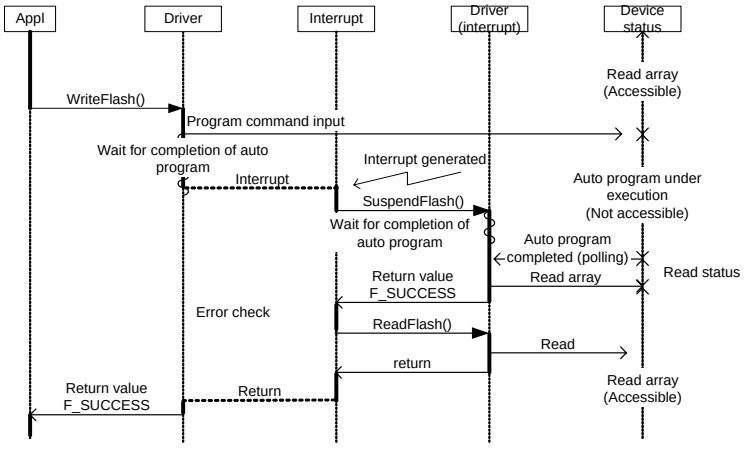

Figure 4-5. Flash Memory Read during Interrupt Processing (Auto Program)

No problems may be incurred by executing SuspendFlash() during other than auto program or auto erase.

4.3 Software Interface

The sample program interface is described below.

Table 4-3. StartEraseFlash()

Outline	Erases the device.	
Declaration	FlashResult StartEraseFlash(F_ADR flashAddress);	
Include	flash_ew0drv.h	
Parameter	Meaning	Remark
F_ADR flashAddress	Highest address of the flash memory to be erased (even address)	
Return value	Meaning	Value
FlashResult	Error code	F_SUCCESS Successfully executed Other Error code See Table 4-13
Description		
This function erases the flash memory (all 0xFF). If SuspendErase() or SuspendFlash() is called in an interrupt process during erase and the interrupt processing is completed, this function is terminated in the middle, with the error code F_SUSPEND returned. In this case, RestartEraseFlash() described next should be called to resume suspended erase and complete it.		
Error code		
F_SUCCESS Successfully executed Other Error code See Table 4-13		
Remark		
If an error occurs (responded with error from the flash memory device), no retry operation is attempted. If an error occurs, therefore, the erase operation should be retried a number of times as specified in the manual. This function cannot be used in an interrupt. Interrupts are controlled using the I flag internally. If interrupts need to be disabled in order to use this function, disable interrupts in the IPL.		

Table 4-4. RestartEraseFlash()

Outline	Resumes device erase.	
Declaration	FlashResult RestartEraseFlash(void);	
Include	flash_ew0drv.h	
Parameter	Meaning	Remark
None		
Return value	Meaning	Value
FlashResult	Error code	F_SUCCESS Successfully executed Other Error code See Table 4-13
Description		
This function resumes erase of the flash memory (all 0xFF). If StartEraseFlash() or RestartEraseFlash() returns with the error code F_SUSPEND (erase suspended), this function is called in order to resume suspended erase. When erase is completed after being resumed, F_SUCCESS is flagged. As for StartEraseFlash(), the erase operation resumed by this function can be suspended again by SuspendErase() or SuspendFlash().		
Error code		
F_SUCCESS Successfully executed Other Error code See Table 4-13		
Remark		
If an error occurs (responded with error from the flash memory device), no retry operation is attempted. If an error occurs, therefore, the erase operation should be retried a number of times as specified in the manual. This function cannot be used in an interrupt. Interrupts are controlled using the I flag internally. If interrupts need to be disabled in order to use this function, disable interrupts in the IPL.		

Table 4-5. WriteFlash()

Outline	Writes data to the flash memory.	
Declaration	FlashResult WriteFlash(F_ADR flashAddress, const void * buffer, unsigned short size);	
Include	flash_ew0drv.h	
Parameter	Meaning	Remark
F_ADR flashAddress,	Beginning address of the area to which data is to be written	Because data is written in 2-byte units, specify an even address.
const void * buffer	Beginning address of the data to be written	
unsigned short size	The data size to be written	Be sure to specify in 2-byte units.
Return value	Meaning	Value
FlashResult	Error code	F_SUCCESS Successfully executed Other Error code See Table 4-13
Description		
This function writes data to the flash memory. Since data is written to the flash memory in 2-byte units, an even address must be specified for flashAddress and an multiple of 2 bytes must be specified for size. An error is assumed unless the conditions are met.		
Error code		
F_SUCCESS Successfully executed Other Error code See Table 4-13		
Remark		
A write alignment error or write size error can be deleted by commenting out #define ENABLE_FLASH_ERR_CHECK. In this case, if a parameter error (2-byte data size check or alignment) occurs, the program behavior is undefined. This function cannot be used in an interrupt. Interrupts are controlled using the I flag internally. If interrupts need to be disabled in order to use this function, disable interrupts in the IPL.		

Table 4-6. ReadFlash()

Outline	Reads data from the flash memory.	
Declaration	FlashResult ReadFlash(F_ADR flashAddress, void * buffer, unsigned short size);	
Include	flash_ew0drv.h	
Parameter	Meaning	Remark
F_ADR flashAddress	Beginning address of the read data	
void * buffer	Address into which data is read	
unsigned short size	The data size to be written	
Return value	Meaning	Value
FlashResult	Error code	F_SUCCESS Successfully executed Other Error code See Table 4-13
Description		
This function reads data from the flash memory.		
Error code		
F_SUCCESS Successfully executed Other Error code See Table 4-13		
Remark		

Table 4-7. UnlockBlockFlash()

Outline	Unlocks flash memory blocks to lift write protect.	
Declaration	void UnlockBlockFlash(enum FlashBlock blockNumber);	
Include	flash_ew0drv.h	
Parameter	Meaning	Remark
enum FlashBlock blockNumber	Level of write protect to be lifted	When F_ALLBLOCK, all blocks are removed of write protect
Return value	Meaning	Value
None		
Description		
This function unlocks flash memory blocks to lift write protect. F_BLOCK_2 to F_BLOCK_5: Blocks 2, 3, 4, 5, A and B are write enabled. (M16C/26A, 28 and 29) F_BLOCK_0 to F_BLOCK_1: Blocks 0, 1, 2, 3, 4, 5, A and B are write enabled.		
Error code		
None		
Remark		

Table 4-8. LockBlockFlash()

Outline	Locks flash memory blocks to write protect.	
Declaration	void LockBlockFlash(enum FlashBlock blockNumber);	
Include	flash_ew0drv.h	
Parameter	Meaning	Remark
enum FlashBlock blockNumber	Level be write protected	When F_ALLBLOCK, all blocks are write protected
Return value	Meaning	Value
None		
Description		
This function locks flash memory blocks to write protect. F_BLOCK_2 to F_BLOCK_5: Blocks 0, 1, 2, 3, 4 and 5 are write enabled. F_BLOCK_0 to F_BLOCK_1: Blocks 0 and 1 are write protected.		
Error code		
None		
Remark		

Table 4-9. SuspendErase()

Outline	Issues a request to suspend wait for completion of erase	
Declaration	void SuspendErase(void);	
Include	flash_ew0drv.h	
Parameter	Meaning	Remark
None		
Return value	Meaning	Value
None		
Description		
This function issues a request to suspend processing while completion of erase by StartEraseFlash() or RestartEraseFlash() is being waited for. The content of processing varies depending on the flash memory status, as follows: During programming : No operation performed During erase : A request to suspend wait for completion of erase is issued. During normal : No operation performed During erase suspend : A request to suspend wait for completion of erase is issued. If this function is called after calling the StartEraseFlash() function, StartEraseFlash() suspends erase and returns with F_SUSPEND. Thereafter, erase is restarted by RestartEraseFlash(). (In this case, erase may be suspended again by SuspendErase().)		
Error code		
None		
Remark		

Table 4-10. ResumErase()

Outline	Cancels a request to suspend wait for completion of erase	
Declaration	void ResumErase(void);	
Include	flash_ew0drv.h	
Parameter	Meaning	Remark
None		
Return value	Meaning	Value
None		
Description		
This function drops the request to suspend erase issued by SuspendErase(). If no requests occur, no operation is performed.		
Error code		
None		
Remark		
If this function is called after calling SuspendErase(), the request to suspend erase issued by SuspendErase() is dropped. However, this does not apply if this function is called after SuspendErase() has once been accepted.		

Table 4-11. SuspendFlash()

Outline	Checks whether the flash memory can be accessed for read during erase or program.	
Declaration	FlashResult SuspendFlash(void);	
Include	flash_ew0drv.h	
Parameter	Meaning	Remark
None		
Return value	Meaning	Value
FlashStatus	Flash memory status	
Description		
Used for interrupts only. This function checks whether the flash memory can be accessed for read while it is being erased. It is used for the purpose of inspection when the flash memory needs to be accessed for read in an interrupt sequence. The content of processing varies depending on the flash memory status, as follows: During programming : Completion of programming is waited (software loop). During erase : A suspend command is issued and completion of suspend or programming is waited (software loop). During normal : No operation performed During erase suspend : No operation performed		
Error code		
None		
Remark		

Table. 4-12 ResumFlash()

Outline	Restores from suspend state	
Declaration	FlashResult ResumFlash(void);	
Include	flash_ew0drv.h	
Parameter	Meaning	Remark
None		
Return value	Meaning	Value
FlashStatus	Flash memory status	
Description		
Used for interrupts only. This function restores the driver from a suspend state and let it start erasing the flash memory. This function is used in pairs with SuspendFlash(). If this function is called during other than suspend, no operation is performed.		
Error code		
None		
Remark		

Table 4-13. Error Code (enum FlashResult)

Name	Value	Meaning
F_SUCCESS	0	Successfully executed.
F_WRITE_ERROR	1	Write error. A write error was notified from the flash memory.
F_ERASE_ERROR	2	Erase error.
F_CMD_SEQUENCE_ERROR	3	Sequence error. A sequence error was notified from the flash memory.
F_WRITE_ADDRESS_ERROR	4	Write address alignment error (2 bytes aligned)
F_WRITE_SIZE_ERROR	5	Invalid write size error (written in 2-byte units)
F_DEVICE_BUSY	6	Device busy (read during programming or erase)
F_SUSPEND	7	Suspended by a suspend request in the middle of programming
F_RESUM_ERROR	8	Called by ResumFlash() when not suspended
F_RESET_OCCURRED	9	When the flash memory is reset

4.4 Customization

The sample program requires partial setup that needs to be made for each system.
The following describes how to customize the sample program to make it suitable for each system.

4.4.1 Customizing CPU Clock Settings

In CPU rewrite mode, the CPU clock settings are subject to limitations.
To meet the conditions, customize the processing in the functions listed below.
In the sample program, the CPU clock settings are created assuming M16C/28 Xin = 20 MHz.

Table 4-14. Functions That Need to Be Customized

Function name	Description
void SlowMCU(ProcessorMode * save);	Corrects the CPU clock settings to overcome the limitations and saves the settings prior to correction in save.
void RestoreMCU(ProcessorMode * save);	Restores the save data as CPU clock settings.

For details about clock limitations, refer to the contents described in Section 3.5.1.

4.4.2 Customizing Operation of the Driver Software

Use flashdevconf.h to customize the driver software.

Table 4-15. Definition of Sample Program Options

Setup content	Define name	Default	Contents to be set or description
CPU series setting	M16C_SERIES	M16C_SERIES	Add a M16C type dependent part to the source.
CPU group setting	M16C_26 M16C_26A M16C_28 M16C_29	M16C_28	Set a CPU group. Include the header provided for each type. Add a type dependent part of lock bit settings.
Programming mode setting	FLASH_MODE_EW0	FLASH_MODE_EW0	Select the programming mode used.
Watchdog timer clear	ENABLE_WATCHDOG_RESET	Unspecified	Specify whether or not to clear the watchdog timer while waiting for completion of auto erase or auto program. Normally, do this in the main processing of the user program rather than using this processing.
Address error check specification	ENABLE_FLASH_ERR_CHECK	ENABLE_FLASH_ERR_CHECK	Check address alignment and 2-byte unit programming.
Definition for use of flash memory suspend function	USE_SUSPEND_FLASH_FUNCTION	Unspecified	Define this when the SuspendFlash() function is to be used. Normally, use the SuspendFlash() function.
Definition of flash memory address	F_ADR_SIZE	Far	Specify the definition of the write pointer that indicates the flash memory location to which to write. If only the data area needs to be written to, this can be defined as near.

5. Using the Sample Program

An example method for using the sample program is shown in the main of main_m16c.c.
For details on how to use, refer to the main_m16c.c file.

The following shows specifications of the main processing.

- Operates with M16C/28 Xin = 20 MHz.
- The main loop executed every 20 ms is created by timer A interrupt processing.
- Block B is erased every 1 s.
  - Erase is suspended every 20 ms, with control returned to the main loop. This processing is performed in timer A interrupt processing.
  - After completion of erase, data is written to the flash memory every 32 bytes beginning with 0F00016 and the written data is read out.

5.1 Source Code

5.1.1 flashdevconf.h

```
/***********************************************************************/
/*    FILE NAME : flashdevconf.h                                       */
/*    Ver       : 1.00                                                 */
/*    CPU       : M16C/Tiny R8C/Tiny                                   */
/*    FUNCTION  : Flash erase/read/write driver.                       */
/*                by EW0 or EW1 mode operation                         */
/*---------------------------------------------------------------------*/
/*  Copyright(C)2004, Renesas Technology Corp.                         */
/*  Copyright(C)2004, Renesas Solutions Corp.                          */
/*  All rights reserved.                                               */
/***********************************************************************/
/*
// $Id: flashdevconf.h,v 1.9 2004/07/02 06:12:02 kato Exp $
*/

#ifndef __FLASHDEVCONF_H__
#define __FLASHDEVCONF_H__

//#define DEBUG

/** Select CPU TYPE ************************************************/
//#define R8C_SERIES
//#define R8C_10
//#define R8C_11
//#define R8C_12
//#define R8C_13
#define M16C_SERIES
#define M16_26
//#define M16_26A
//#define M16_28
//#define M16_29

/** Write Mode Define (EW0 or EW1)  *********************************/
#define FLASH_MODE_EW0
//#define FLASH_MODE_EW1

// Watch Dog Reset Enable
//#define ENABLE_WATCHDOG_RESET

//#define ENABLE_FLASH_ERR_CHECK

// USE SuspendFlash function
#define USE_SUSPEND_FLASH_FUNCTION

#if defined(M16C_SERIES)
#define F_ADR_SIZE far
#elif defined(R8C_SERIES)
#define F_ADR_SIZE
#undef ENABLE_FLASH_ERR_CHECK
#else
#error "Please choose either R8C_SERIES\ or M16C_SERIES\. "
#endif

/** Check !! ********************************************/
#if (!defined(FLASH_MODE_EW0) && !defined(FLASH_MODE_EW1))
#error "Please choose FLASH_MODE_EW0 , FLASH_MODE_EW0 or both. "
#endif

#endif /* #ifndef __FLASHDEVCONF_H__ */
```

5.1.2 flashdevdrv.h

```
/***********************************************************************/
/*    FILE NAME : flashdevdrv.h                                        */
/*    Ver       : 1.00                                                 */
/*    CPU       : M16C/26                                              */
/*    FUNCTION  : Flash erase/read/write driver.                       */
/*                by EW0 or EW1 mode operation                         */
/*---------------------------------------------------------------------*/
/*  Copyright(C)2004, Renesas Technology Corp.                         */
/*  Copyright(C)2004, Renesas Solutions Corp.                          */
/*  All rights reserved.                                               */
/***********************************************************************/
/*
// $Id: flashdevdrv.h,v 1.5 2004/07/22 04:06:41 ikari Exp $
*/

#include "flashdevconf.h"

#ifndef __FLASHDEVDRV_H__
#define __FLASHDEVDRV_H__


/** Write Mode Define (EW0 or EW1)  **********************************/
#if (defined(FLASH_MODE_EW0) && !defined(FLASH_MODE_EW1))
#define WriteFlash(a,b,c)       WriteFlashEW0(a,b,c)
#define ReadFlash(a,b,c)        ReadFlashEW0(a,b,c)
#define StartEraseFlash(a)      StartEraseFlashEW0(a)
#define RestartEraseFlash()     RestartEraseFlashEW0()
#define SuspendErase()          SuspendEraseEW0()
#define ResumErase()            ResumEraseEW0()
#define GetFlashStatus()        GetFlashStatusEW0()
#endif  /* #if (defined(FLASH_MODE_EW0) && !defined(FLASH_MODE_EW1)) */

#if (!defined(FLASH_MODE_EW0) && defined(FLASH_MODE_EW1))
#define WriteFlash(a,b,c)       WriteFlashEW1(a,b,c)
#define ReadFlash(a,b,c)        ReadFlashEW1(a,b,c)
#define StartEraseFlash(a)      StartEraseFlashEW1(a)
#define RestartEraseFlash()     RestartEraseFlashEW1()
#define SuspendErase()          SuspendEraseEW1()
#define ResumErase()            ResumEraseEW1()
#define GetFlashStatus()        GetFlashStatusEW1()
#endif /* #if (!defined(FLASH_MODE_EW0) && defined(FLASH_MODE_EW1)) */


/*======================================================================*/
/*! The address definition of the flash memory */
typedef void F_ADR_SIZE * F_ADR;

/*======================================================================*/
/*! Error code */
typedef enum FlashResult{
    F_SUCCESS,              /*! Success                                    */
    F_WRITE_ERROR,          /*! Program error  (from the flash device)     */
    F_ERASE_ERROR,          /*! Block erase error (from the Flash device)  */
    F_CMD_SEQUENCE_ERROR,   /*! Command sequence error(from the Flash device) */
    F_WRITE_ADDRESS_ERROR,  /*! The address alignment error of the argument */
    F_WRITE_SIZE_ERROR,     /*! The size error of the argument             */
    F_DEVICE_BUSY,          /*! When a Read/Programing/Erasing requirement  */
                            /*  in Programing/Erasing occurs.              */
    F_SUSPEND,              /*! Suspend requirement acceptance.            */
    F_RESUM_ERROR,          /*! When Suspend isn't being done with         */
                            /*  ResumFlash(), call                         */
    F_RESET_OCCURRED,       /*! Resetting was done during the              */
                            /*  Programing/Erasing as to the call.         */
} FlashResult;
```

```
/*=========================================================================*/
/*!
 * Initialize flash device.
 */
void FlashInitialize(void);
/*=========================================================================*/
/*!
 * Device resetting.
 */
void FlashReset(void);
/*=========================================================================*/
/*!
 * Erase flash memory in the EW0 mode.
 * @param  flashAddress             [in] physical address of the head of
 *                                       the block on flash.
 * @return Success or Error code.
 * @retval F_SUCCESS                Success
 * @retval F_ERASE_ERROR            Block erase error (from the Flash device)
 * @retval F_CMD_SEQUENCE_ERROR     Command sequence error(from the Flash device)
 * @retval F_WRITE_ADDRESS_ERROR    The address alignment error of the argument.
 * @retval F_DEVICE_BUSY            When a Read/Programing/Erasing requirement
 *                                  in Programing/Erasing occurs.
 * @retval F_SUSPEND                Suspend requirement acceptance.
 */
FlashResult StartEraseFlashEW0(F_ADR flashAddress);

/*=========================================================================*/
/*!
 * Erase flash memory in the EW1 mode.
 * @param  flashAddress             [in] physical address of the head of
 *                                       the block on flash.
 * @return Success or Error code.
 * @retval F_SUCCESS                Success
 * @retval F_ERASE_ERROR            Block erase error (from the Flash device)
 * @retval F_CMD_SEQUENCE_ERROR     Command sequence error(from the Flash device)
 * @retval F_WRITE_ADDRESS_ERROR    The address alignment error of the argument.
 * @retval F_DEVICE_BUSY            When a Read/Programing/Erasing requirement
 *                                  in Programing/Erasing occurs.
 * @retval F_SUSPEND                Suspend requirement acceptance.
 */
FlashResult StartEraseFlashEW1(F_ADR flashAddress);

/*=========================================================================*/
/*!
 * Restart to erase a block of the flash memory in the EW0 mode.
 * @return Success or Error code.
 * @retval F_SUCCESS                Success
 * @retval F_ERASE_ERROR            Block erase error (from the Flash device)
 * @retval F_CMD_SEQUENCE_ERROR     Command sequence error(from the Flash device)
 * @retval F_WRITE_ADDRESS_ERROR    The address alignment error of the argument.
 * @retval F_DEVICE_BUSY            When a Read/Programing/Erasing requirement
 *                                  in Programing/Erasing occurs.
 * @retval F_SUSPEND                Suspend requirement acceptance.
 */
FlashResult RestartEraseFlashEW0(void);

/*=========================================================================*/
/*!
 * Restart to erase a block of the flash memory in the EW1 mode.
 * @return Success or Error code.
 * @retval F_SUCCESS                Success
 * @retval F_ERASE_ERROR            Block erase error (from the Flash device)
 * @retval F_CMD_SEQUENCE_ERROR     Command sequence error(from the Flash device)
 * @retval F_WRITE_ADDRESS_ERROR    The address alignment error of the argument.
 * @retval F_DEVICE_BUSY            When a Read/Programing/Erasing requirement
```

```
 *                                  in Programing/Erasing occurs.
 * @retval F_SUSPEND                Suspend requirement acceptance.
 */
FlashResult RestartEraseFlashEW1(void);

/*=====================================================================*/
/*!
 * Write data to flash memory in the EW0 mode.
 * @param  flashAddress             [in] physical address on flash to begin write
 * @param  buffer                   [in] address in buffer to write from
 * @param  size                     [in] number of byte to write.
 * @return Success or Error code.
 * @retval F_SUCCESS                Success
 * @retval F_WRITE_ERROR            Program error  (from the flash device)
 * @retval F_CMD_SEQUENCE_ERROR     Command sequence error(from the Flash device)
 * @retval F_WRITE_ADDRESS_ERROR    The address alignment error of the argument.
 * @retval F_WRITE_SIZE_ERROR,      The size error of the argument
 * @retval F_DEVICE_BUSY            When a Read/Programing/Erasing requirement
 *                                  in Programing/Erasing occurs.
 */
FlashResult WriteFlashEW0(F_ADR flashAddress,
                          const void * buffer,
                          unsigned short size);

/*=====================================================================*/
/*!
 * Write data to flash memory in the EW1 mode.
 * @param  flashAddress             [in] physical address on flash to begin write.
 * @param  buffer                   [in] address in buffer to write from.
 * @param  size                     [in] number of byte to write.
 * @return Success or Error code.
 * @retval F_SUCCESS                Success
 * @retval F_WRITE_ERROR            Program error  (from the flash device)
 * @retval F_CMD_SEQUENCE_ERROR     Command sequence error(from the Flash device)
 * @retval F_WRITE_ADDRESS_ERROR    The address alignment error of the argument.
 * @retval F_WRITE_SIZE_ERROR,      The size error of the argument
 * @retval F_DEVICE_BUSY            When a Read/Programing/Erasing requirement
 *                                  in Programing/Erasing occurs.
 */
FlashResult WriteFlashEW1(F_ADR flashAddress,
                          const void * buffer,
                          unsigned short size);

/*=====================================================================*/
/*!
 * Read data from flash memory for the EW1 mode.
 * @param  flashAddress             [in]  physical address on flash to begin read.
 * @param  buffer                   [out] address in buffer to read to.
 * @param  size                     [in]  number of byte to read.
 * @return Success or Error code.
 * @retval F_SUCCESS                Success
 * @retval F_DEVICE_BUSY            When a Read/Programing/Erasing requirement
 *                                  in Programing/Erasing occurs.
 */
FlashResult ReadFlashEW0(F_ADR flashAddress,
                         void * buffer,
                         unsigned short size);

/*=====================================================================*/
/*!
 * Read data from flash memory for the EW1 mode.
 * @param  flashAddress             [in]  physical address on flash to begin read.
 * @param  buffer                   [out] address in buffer to read to.
 * @param  size                     [in]  number of byte to read.
 * @return Success or Error code.
 * @retval F_SUCCESS                Success
 * @retval F_DEVICE_BUSY            When a Read/Programing/Erasing requirement
```

```
 *                                in Programing/Erasing occurs.
 */
FlashResult ReadFlashEW1(F_ADR flashAddress,
                         void * buffer,
                         unsigned short size);

/*======================================================================*/
/*! Block number */
enum FlashBlock{
    F_BLOCK_0,
    F_BLOCK_1,
    F_BLOCK_2,
    F_BLOCK_3,
    F_BLOCK_4,
    F_BLOCK_5,
    F_ALLBLOCK = -1,
    F_BLOCK_A =  -2,
    F_BLOCK_B =  -3,
};
/*======================================================================*/
/*!
 * Disable write protect.
 * @param  blockNumber              [in]  The block number to protect it.
 */
void UnlockBlockFlash(enum FlashBlock blockNumber);

/*======================================================================*/
/*!
 * Enable write protect.
 * @param  blockNumber              [in]  The block number to unprotect it.
 */
void LockBlockFlash(enum FlashBlock blockNumber);

/*======================================================================*/
/*!
 * Send the suspension request of a flash memory in the EW0 mode.
 */
void SuspendEraseEW0(void);
/*======================================================================*/
/*!
 * Send the suspension request of a flash memory in the EW1 mode.
 */
void SuspendEraseEW1(void);
/*======================================================================*/
/*!
 * Clear the suspension request of a flash memory in the EW0 mode.
 */
void ResumEraseEW0(void);
/*======================================================================*/
/*!
 * Clear the suspension request of a flash memory in the EW0 mode.
 */
void ResumEraseEW1(void);


/*======================================================================*/
/*! Status of flash memory or this driver. */
typedef enum FlashStatus{
    FLASH_READY,                /*! ready                                 */
    FLASH_WRITE,                /*! programing                            */
    FLASH_ERASE,                /*! erasing                               */
    FLASH_SUSPEND,              /*! suspend erasing                       */
    FLASH_INT_SUSPEND,          /*! suspend erasing during the interruption  */
}FlashStatus;

/*======================================================================*/
/*!
```

```
 * Get flash memory status for the EW0 mode.
 */
FlashStatus GetFlashStatusEW0(void);
/*=========================================================================*/
/*!
 * Get flash memory status for the EW1 mode.
 */
FlashStatus GetFlashStatusEW1(void);

#ifdef FLASH_MODE_EW0
/*=========================================================================*/
/*!
 * Suspend at interrupt.
 */
FlashResult SuspendFlashEW0(void);
/*=========================================================================*/
/*!
 * This function resumes the erasure processing suspended by SuspendFlashEW0().
 */
FlashResult ResumFlashEW0(void);
#endif


#endif /* #ifndef __FLASHDEVDRV_H__ */
```

5.1.3 flashm16c.h

```
/***********************************************************************/
/*    FILE NAME : flashm16c.h                                          */
/*    Ver       : 1.00                                                 */
/*    CPU       : M16C                                                 */
/*    FUNCTION  : Flash erase/read/write driver.                       */
/*                by EW0 or EW1 mode operation                         */
/*---------------------------------------------------------------------*/
/*  Copyright(C)2004, Renesas Technology Corp.                         */
/*  Copyright(C)2004, Renesas Solutions Corp.                          */
/*  All rights reserved.                                               */
/***********************************************************************/
/*
// $Id: flashm16c.h,v 1.4 2004/06/30 02:28:41 nagayoshi Exp $
*/

#include "flashdevconf.h"

#if     defined( M16_26  )
#include "sfr26.h"
#elif   defined( M16_26A )
#include "sfr26a.h"
#elif   defined( M16_28  )
#include "sfr28.h"
#elif   defined( M16_29  )
#include "sfr29.h"
#else
#error "This cpu type not support."
#endif

#ifndef __FLASHM16C_H__
#define __FLASHM16C_H__

//typedef volatile unsigned short F_ADR_SIZE * DEPEND_F_ADR;

typedef volatile unsigned short DEPEND_FSIZE;

#define FLASH_READARRAY_CMD             ((unsigned short)0xff)
#define FLASH_STSREGS_CMD               ((unsigned short)0x70)
#define FLASH_CLEAR_STSREGS_CMD         ((unsigned short)0x50)
#define FLASH_PRG_CMD                   ((unsigned short)0x40)
#define FLASH_BLOCK_ERASE_1_CMD         ((unsigned short)0x20)
#define FLASH_BLOCK_ERASE_2_CMD         ((unsigned short)0xd0)

#define _FLASH_E_EW() {\
    fmr01 = 0;  \
    asm("");    \
    fmr01 = 1;  \
    asm("");    \
}

#define _FLASH_DIS_EW() {\
    fmr01 = 0;  \
}

#define _FLASH_E_SUSPEND() {\
    fmr40 = 0;  \
    asm("");    \
    fmr40 = 1;  \
    asm("");    \
}

#define _FLASH_SUSPEND_ERASE() {\
    fmr41 = 1;\
}
#define _FLASH_RESUME_ERASE() {\
```

```
    fmr41 = 0;\
}

#define _FLASH_RESUME_ERASE_EW1(a) {\
    fmr41 = a;\
}

#define _FLASH_EW1_MODE() {\
    fmr11 = 0;\
    fmr11 = 1;\
}

#define _CLEAR_WATCHDOG() {\
    wdts = 0x7fff;\
}

#define _FLASH_BUSY() (fmr00 == 0)

#define _FLASH_DEVICE_INIT() {\
    fmr17 = 1;          \
    prcr = 0x3;      /* Unlock CM0, CM1, PM1 */\
    pm10 = 1;        /* enable flash data block (4KB Virtual EEPROM) access */ \
    prcr = 0;        /* Lock the System Clock Control Register */ \
}

#define ERASE_ERR 0x40
#define PRGRAM_ERR 0x80
#define FMR07_06 (PRGRAM_ERR | ERASE_ERR)

#define _FLASH_GET_STAT_FLG() (fmr0 & FMR07_06)

#define DEBUG_OUT_ERASE_START   0x04    /* xxxx100 */
#define DEBUG_OUT_SUSPEND 0x02          /* xxxx010 */
#define DEBUG_OUT_READY   0x07          /* xxxx111 */
#define DEBUG_OUT_WAIT_ERASE 0xFF       /* not use */

//#define DEBUG_M16C
#ifdef DEBUG_M16C
#define DEBUG_OUT(a) {\
    if(a != 0xFF){  \
        p7_0 = (0x01 & a);          /*  0:Eraseing 1:Eraseed                */ \
        p7_1 = (0x01 & (a>>1));     /*  0 Erase  1:suspend or erase complete */ \
        p7_2 = (0x01 & (a>>2));     /*  0:Suspend   1: not Suspend          */ \
    }   \
}
#else
#define DEBUG_OUT(a)
#endif /* #ifdef DEBUG_M16C */

struct  LockBitStatus{
    unsigned char s_fmr16:1;
    unsigned char s_fmr02:1;
};

/*! for SlowMCU/RestoreMCU */
#define USE_CLOCKGEAR
typedef struct ProcessorMode
{
     unsigned char p_pm1;
     unsigned char p_cm0;
     unsigned char p_cm1;
} ProcessorMode;
void SlowMCU(ProcessorMode * save);
void RestoreMCU(ProcessorMode * save);

#endif /* #ifndef __FLASHM16C_H__ */
```

5.1.4 flashdrvdev_ew0.c

```
/*********************************************************************/
/*    FILE NAME : flashdrvdev_ew0.c                                  */
/*    Ver       : 1.00                                               */
/*    CPU       : R8C M16C/26,26A,28,29                              */
/*    FUNCTION  : Flash low level (erase/read/write) driver.         */
/*                by EW0 mode operation                              */
/*-------------------------------------------------------------------*/
/*  Copyright(C)2004, Renesas Technology Corp.                       */
/*  Copyright(C)2004, Renesas Solutions Corp.                        */
/*  All rights reserved.                                             */
/*********************************************************************/
/*
// $Id: flashdrvdev_ew0.c,v 1.11 2004/07/21 09:04:39 kato Exp $
*/
#include <string.h>
#include "flashdevdrv.h"

#if defined(M16C_SERIES)
#include "flashm16c.h"
#elif defined(R8C_SERIES)
#include "flashr8c.h"
#endif

#ifdef FLASH_MODE_EW0


/*================ RAM  ================*/
/*! Flash status                        */
enum FlashStatus stat = FLASH_READY;

/*! Flash status                    */
struct {
    unsigned char suspReq:1;
    unsigned char rstOccerred:1;
}f_flags = {0,0};
#define suspendReq  f_flags.suspReq
#define resetOccerrd f_flags.rstOccerred

/*! Command target address.  */
DEPEND_FSIZE F_ADR_SIZE * com_adr;


/*=== Prototype  ======================*/
FlashResult EraseCheckInternal(void);
FlashResult FullStatusCheck(void);
void SetUpLockBit(void);

/* There functions are running on RAM.*/
void RestartFlashEraseRam(void);
void FlashEraseCommandRam(DEPEND_FSIZE F_ADR_SIZE *  flashAddress);
void FlashWriteCommandRam(DEPEND_FSIZE F_ADR_SIZE *  flashAddress ,
                          unsigned short data);
void WaitEraseCompleteRam(DEPEND_FSIZE F_ADR_SIZE * flashAddress);
void FlashSuspendRam(void);

/*************************************************************************
Name        :FlashInitialize()
Purpose     :Initialization of a flash memory.
Returns     :Result.
Notice      :None.
*************************************************************************/
void FlashInitialize(void)
{
    _FLASH_DEVICE_INIT();
    FlashReset();
}
```

```
/***********************************************************************
Name        :ReadFlashEW0()
Purpose     :Read data from flash memory.
Arguments   :
    [in]  flashAddress  physical add on flash to begin read
    [out] buf           address in buffer to read to
    [in]  size          number of byte to read
Return      :Result
Notice      :None
***********************************************************************/
FlashResult ReadFlashEW0(F_ADR flashAddress,
                         void * buf,
                         unsigned short size)
{
    if(stat != FLASH_READY && stat != FLASH_SUSPEND)return F_DEVICE_BUSY;
    memcpy( buf , flashAddress , size);
    return F_SUCCESS;
}

/***********************************************************************
Name        :StartEraseFlashEW0()
Purpose     :Start to erase 1 block of the flash memory.
Arguments   :physical address of the head of the block on flash.
Return      :Result
Notice      :This function erase and return result only.
             This function does not perform retry at the time of error.
***********************************************************************/
FlashResult StartEraseFlashEW0(F_ADR flashAddress)
{
    FlashResult ret;

#if defined(USE_CLOCKGEAR)
    ProcessorMode save_dat;
#endif

    DEPEND_FSIZE F_ADR_SIZE *  adr = flashAddress;

#ifdef ENABLE_FLASH_ERR_CHECK
    if(0x01 & ((unsigned long)adr ))return F_WRITE_ADDRESS_ERROR;
#endif

    // disable interrupt.
    asm("FCLR I");
    if(stat != FLASH_READY){
        asm("FSET I");
        return F_DEVICE_BUSY;
    }
    asm("FSET I");

    suspendReq = 0;     /* initialize request */
    resetOccerrd = 0;   /* initialize reset_flags */

#if defined(USE_CLOCKGEAR)
    SlowMCU(&save_dat);     // Must change main clock speed to meet flash
#endif

    asm("FCLR I");
    _FLASH_E_EW();
    asm("FSET I");

    asm("FCLR I");
    SetUpLockBit();
    asm("FSET I");

    asm("FCLR I");
    _FLASH_E_SUSPEND();
```

```
    asm("FSET I");

    ////////////////////////////////
    // Clear status register.
    ////////////////////////////////
    asm("FCLR I");
    *adr = FLASH_CLEAR_STSREGS_CMD;      // Clear status register
    com_adr = adr;
    stat = FLASH_ERASE;
    ////////////////////////////////
    // Flash command (Erase) entry.
    ////////////////////////////////
    DEBUG_OUT(DEBUG_OUT_ERASE_START);

    FlashEraseCommandRam(adr);  // I-flag is set in this function.

    ret = EraseCheckInternal();

    _FLASH_DIS_EW();

#if defined(USE_CLOCKGEAR)
    RestoreMCU(&save_dat);      // Restore clock back to original speed
#endif

    return ret;
}


/*****************************************************************************
Name        :RestartEraseFlashEW0()
Purpose     :Restart to erase 1 block of the flash memory.
Return      :Result
Notice      :This function erases flash memory with operating erase command.
             It does not perform retry at the time of error.
*****************************************************************************/
FlashResult RestartEraseFlashEW0(void)
{
    FlashResult ret;
#if defined(USE_CLOCKGEAR)
    ProcessorMode save_dat;
#endif

    asm("FCLR I");
    if(stat != FLASH_SUSPEND){
        asm("FSET I");
        return F_DEVICE_BUSY;
    }
    asm("FSET I");

#if defined(USE_CLOCKGEAR)
    SlowMCU(&save_dat);         // Must change main clock speed to meet flash
#endif

    asm("FCLR I");
    _FLASH_E_EW();
    asm("FSET I");

    asm("FCLR I");
    SetUpLockBit();
    asm("FSET I");

    DEBUG_OUT(DEBUG_OUT_ERASE_START);

    asm("FCLR I");
    stat = FLASH_ERASE;
    RestartFlashEraseRam();     // I-flag is set in this function.
```

```
    ret = EraseCheckInternal();

    _FLASH_DIS_EW();

#if defined(USE_CLOCKGEAR)
    RestoreMCU(&save_dat);      // Restore clock back to original speed
#endif
    return ret;

}

/************************************************************************
Name        :EraseCheckInternal()
Purpose     :The eraseed result is judged.
Return      :Result.
Notice      :
************************************************************************/
FlashResult EraseCheckInternal(void)
{
    FlashResult ret;
    if(_FLASH_BUSY()){

        DEBUG_OUT(DEBUG_OUT_SUSPEND);

        stat = FLASH_SUSPEND;
        ret = F_SUSPEND;
    }
    else{

        DEBUG_OUT(DEBUG_OUT_READY);

        ret = FullStatusCheck();        // Erasing error?
        asm("FCLR I");
        _FLASH_RESUME_ERASE();          // take the precaution of resumeing.
        stat = FLASH_READY;
        asm("FSET I");

    }
    return ret;
}


/************************************************************************
Name        :WriteFlashEW0()
Purpose     :Write data to flash memory.
Arguments   :Address
    [in]  flashAddress  physical add on flash to begin write
    [in]  buf           address in buffer to write from
    [in]  size          number of byte to write
Return      :Result
Notice      :This function writes flash memory with operating write command.
             It does not perform retry at the time of error.
************************************************************************/
FlashResult WriteFlashEW0(F_ADR flashAddress,
                          const void * buffer,
                          unsigned short size)
{
#if defined(USE_CLOCKGEAR)
    ProcessorMode save_dat;
#endif
    FlashResult ret;
    DEPEND_FSIZE F_ADR_SIZE * adr = flashAddress;
    const DEPEND_FSIZE * buf = buffer;

#ifdef ENABLE_FLASH_ERR_CHECK
    if(0x01 & ((unsigned long)adr ))return F_WRITE_ADDRESS_ERROR;
    if(0x01 & size)return F_WRITE_SIZE_ERROR;
```

```
#endif

    // disable interrupt.
    asm("FCLR I");
    if(stat != FLASH_READY){
        asm("FSET I");
        return F_DEVICE_BUSY;
    }
    asm("FSET I");

    resetOccerrd = 0;   /* initialize reset_flags */

#if defined(USE_CLOCKGEAR)
    SlowMCU(&save_dat);     // Must change main clock speed to meet flash
#endif

    asm("FCLR I");
    _FLASH_E_EW();
    asm("FSET I");

    asm("FCLR I");
    SetUpLockBit();
    asm("FSET I");

    ret = F_SUCCESS;
    while(size > 0) {
        asm("FCLR I");
        stat = FLASH_WRITE;
        * adr = FLASH_CLEAR_STSREGS_CMD;    /* Clear status register            */
        FlashWriteCommandRam(adr , *buf);   /* I-flag is set in this function.  */
        ret = FullStatusCheck();            /* Write error?                     */
        stat = FLASH_READY;

        if(ret != F_SUCCESS) break;
        size -= sizeof(DEPEND_FSIZE);       /* subract 2 from byte counter      */
        buf++;                              /* increase to next data index      */
        adr++;                              /* increase to next flash index     */
    }

    // disable interrupt.
    asm("FCLR I");

    _FLASH_DIS_EW();

#if defined(USE_CLOCKGEAR)
    RestoreMCU(&save_dat);      // Restore clock back to original speed
#endif

    return ret;           // Write Pass
}


/***********************************************************************
Name        :FullStatusCheck()
Purpose     :Check the status of flash memory.
Arguments   :None
Return      :Result
***********************************************************************/
FlashResult FullStatusCheck(void)
{
    unsigned char reg;
    reg = _FLASH_GET_STAT_FLG();
    if(resetOccerrd)return F_RESET_OCCURRED;
    if(reg == 0)return F_SUCCESS;
    if(reg == PRGRAM_ERR) return F_WRITE_ERROR;
    if(reg == ERASE_ERR)  return F_ERASE_ERROR;
    return F_CMD_SEQUENCE_ERROR;
```

```
}


/***********************************************************************
 running on RAM
***********************************************************************/
#pragma SECTION program program_ram
/***********************************************************************
Name        :SuspendEraseEW0()
Purpose     :Send the suspension request of a flash memory.
Return      :None
***********************************************************************/
void SuspendEraseEW0(void)
{
    /* send event to wait roop  */
    suspendReq = 1;
}

/***********************************************************************
Name        :ResumEraseEW0()
Purpose     :This function clears suspension request.
Return      :None
***********************************************************************/
void ResumEraseEW0(void)
{
    /* send event to wait roop  */
    suspendReq = 0;
}

/***********************************************************************
Name        :FlashReset()
Purpose     :Reset flash memory.
Return      :None
Notice      :If you use "KD30 debugger", it is necessary to set a setup of
             "KD30 debugger" to "FreeRunMode."
***********************************************************************/
void FlashReset(void)
{
    unsigned char fmr01_reg;
    unsigned char cnt;
#if defined(USE_CLOCKGEAR)
    ProcessorMode save_dat;
    SlowMCU(&save_dat);     // Must change main clock speed to meet flash
#endif

    // disable interrupt.
    asm("FCLR I");
    fmr01_reg = fmr01;

    _FLASH_E_EW();

    fmstp = 1;

//    cnt = 1;while(cnt --);

    fmstp = 0;

    if(fmr01_reg == 0)_FLASH_DIS_EW();

    resetOccerrd = 1;
    /* Wait 10us                                *
     * 1/10MHz = 100ns                          *
     * DEC.B 1cycle + JNE 2+2cycle = 5 cycle    *
     * 10us/100ns/5 = 20                         */
    cnt = 20;while(-- cnt);

    asm("FSET I");
```

```
#if defined(USE_CLOCKGEAR)
    RestoreMCU(&save_dat);      // Restore clock back to original speed
#endif

}

/***********************************************************************
Name        :FlashWriteCommandRam
Purpose     :Operate write command on RAM and enable interrupt.
Return      :None
Notice      :This functions on the RAM.
***********************************************************************/
void FlashWriteCommandRam(DEPEND_FSIZE F_ADR_SIZE *  flashAddress ,unsigned short data)
{
    *flashAddress = FLASH_PRG_CMD;        // Send write command
    *flashAddress = data;                 // Write next word of data
    asm("FSET I");
    while( _FLASH_BUSY() );
    *flashAddress = FLASH_READARRAY_CMD;    // Read allay command
    return ;
}

/***********************************************************************
Name        :FlashEraseCommandRam
Purpose     :Erase flash memory and enable interrupt.
Return      :None
Notice      :This functions on the RAM.
***********************************************************************/
void FlashEraseCommandRam(DEPEND_FSIZE F_ADR_SIZE *  flashAddress)
{
    *flashAddress = FLASH_BLOCK_ERASE_1_CMD;     // Send erase command
    *flashAddress = FLASH_BLOCK_ERASE_2_CMD;     // Send erase confirm command
    asm("FSET I");
    WaitEraseCompleteRam(flashAddress);
}

/***********************************************************************
Name        :RestartFlashEraseRam()
Purpose     :Restart Erasing.
Return      :None
Notice      :This functions on the RAM.
***********************************************************************/
void RestartFlashEraseRam(void)
{
    _FLASH_RESUME_ERASE();
    asm("FSET I");
    WaitEraseCompleteRam(com_adr);
    return ;
}

/***********************************************************************
Name        :WaitEraseCompleteRam
Purpose     :Wait until complete or suspended.
             Flash memory will be set "Read Array" mode (you can read data on
             flash memory) ,if this function return.
Return      :None
Notice      :This functions on the RAM.
***********************************************************************/
void WaitEraseCompleteRam(DEPEND_FSIZE F_ADR_SIZE * flashAddress)
{
     DEBUG_OUT(DEBUG_OUT_WAIT_ERASE);
    while( _FLASH_BUSY() ){
#ifdef ENABLE_WATCHDOG_RESET
        _CLEAR_WATCHDOG();
#endif
        if(suspendReq){
```

```
            suspendReq = 0;
            FlashSuspendRam();
            /* Data on the flash memory can't be read if Read-Array Mode isn't
            taken when elimination is completed when a Suspend requirement is accepted. */
            if( !_FLASH_BUSY() )break;
            return;
        }
    }
    *flashAddress = FLASH_READARRAY_CMD;// Read allay command
    return ;
}

/***********************************************************************
Name        :WaitEraseCompleteRam
Purpose     :Block erase command is suspended.
             Wait until suspended.
Return      :None
Notice      :This functions on the RAM.
***********************************************************************/
void FlashSuspendRam(void)
{
    _FLASH_SUSPEND_ERASE();
    while(fmr46 != 1);
}

/***********************************************************************
Name        :SuspendFlashEW0()
Purpose     :Suspend at interrupt.
Return      :Result.
Notice      :Don't use under multiplex interruption.
***********************************************************************/
#ifdef USE_SUSPEND_FLASH_FUNCTION
unsigned char susModeBackup = 0;
FlashResult SuspendFlashEW0(void)
{
    switch(stat){
    case FLASH_ERASE:
        DEBUG_OUT(DEBUG_OUT_SUSPEND);
        suspendReq = 1;
        stat = FLASH_INT_SUSPEND;
        susModeBackup = fmr4;
        FlashSuspendRam();
        if( !_FLASH_BUSY() )return F_SUCCESS;
        *com_adr = FLASH_READARRAY_CMD;    /* Read allay command */
        return F_SUSPEND;
    case FLASH_WRITE:
        while( _FLASH_BUSY() );
        *com_adr = FLASH_READARRAY_CMD;    /* Read allay command */
        break;
    default:
        break;
    }
    return F_SUCCESS;
}

/***********************************************************************
Name        :ResumFlashEW0(void)
Purpose     :This function resumes the erasure processing suspended by SuspendFlashEW0().
Return      :None
Notice      :Don't use under multiplex interruption.
***********************************************************************/
FlashResult ResumFlashEW0(void)
{
    if(stat == FLASH_INT_SUSPEND){
        DEBUG_OUT(DEBUG_OUT_ERASE_START);
        stat = FLASH_ERASE;
        fmr4 = (0x03 & susModeBackup);
```

```
        suspendReq = 0;
        return F_SUCCESS;
    }
    return F_RESUM_ERROR;
}

#endif /* #ifdef USE_SUSPEND_FLASH_FUNCTION */
/***********************************************************************
Name        :GetFlashStatusEW0()
Purpose     :This function returns the condition of the flash memory.
Return      :None
Notice      :This functions on the RAM.
***********************************************************************/
FlashStatus GetFlashStatusEW0(void)
{
    return stat;
}

#endif /* #ifdef FLASH_MODE_EW0 */
```

5.1.5 depend_m16c.c

```
/***********************************************************************/
/*    FILE NAME : depend_m16c.c                                        */
/*    Ver       : 1.00                                                 */
/*    CPU       : M16C/26,26A,28,29                                    */
/*    FUNCTION  : Flash low level (erase/read/write) driver for M16C */
/*---------------------------------------------------------------------*/
/*  Copyright(C)2004, Renesas Technology Corp.                         */
/*  Copyright(C)2004, Renesas Solutions Corp.                          */
/*  All rights reserved.                                               */
/***********************************************************************/
/*
// $Id: depend_m16c.c,v 1.2 2004/06/29 00:40:23 nagayoshi Exp $
*/

#include "flashdevdrv.h"
#include "flashm16c.h"

#ifndef M16C_SERIES
#error "This file is only for M16C."
#endif

/*! Protect flash status */
struct  LockBitStatus fmr_status = {0,0};

/************************************************************************
Name        :SlowMCU()
Purpose     :In order to rewrite a flash memory, the clock of MCU operation is made late.
             (Flash Accsess time nomal -> slow)
Arguments   :address in area to save CPU clock setting.
Return      :None
Notice      :It depends on your system.
             You must modify this function for your system.
             This function is the example of "M16C/26 Xin=20MHz."
************************************************************************/
void SlowMCU(ProcessorMode * save)
{
    asm("FCLR I");
    save->p_cm0 = cm0;    // Save current CPU clock setting
    save->p_cm1 = cm1;
    save->p_pm1 = pm1;

/* "Modify for your system. This code is sample for M16C/26 Xin=20MHz." */

    prcr = 3;           // Unprotect registers CM0 and PM0
//    cm1 = 0x60;         // Use Xin, Xin drive HIGH, Xin/2 (f2)
    cm16 = 1;         // Use Xin, Xin drive HIGH, Xin/2 (f2)
    cm17 = 0;         // Use Xin, Xin drive HIGH, Xin/2 (f2)
    cm06 = 0;           // CM16 and CM17 are valid
    pm17 = 1;           // enable flash data block 1 wait access
    prcr = 0;           // Protection register back on
    asm("FSET I");
}
/************************************************************************
Name        :RestoreMCU()
Purpose     :Restore MCU clock.
Arguments   :address in area to load CPU clock setting.
Return      :None
Notice      :It depends on your system.
             You must modify this function for your system.
             This function is the example of "M16C/26 Xin=20MHz."
************************************************************************/
void RestoreMCU(ProcessorMode * save)
{
    asm("FCLR I");
    prcr = 3;           // Unprotect registers CM0 and PM0
```

```
    pm1 = save->p_pm1;
    cm1 = save->p_cm1;
    cm0 = save->p_cm0;

    prcr = 0;           // Protection register back on
    asm("FSET I");
}

/***********************************************************************
Name        :UnlockBlockFlash()
Purpose     :Disable write protect.
Arguments   :F_BLOCK_0,F_BLOCK_1,F_ALLBLOCK     All area writing is possible.
             F_BLOCK_2,F_BLOCK_3,F_BLOCK_4,F_BLOCK_5
                        Block2 to 4 area writing is possible.(only M16C/26A,28,29)
Return      :None
Notice      :It depends on your system.
***********************************************************************/
void UnlockBlockFlash(enum FlashBlock blockNumber)
{
#if (defined(M16_26A) || defined(M16_28) || defined(M16_29))
    ProcessorMode save_dat;
#endif
    asm("FCLR I");
    switch(blockNumber){
    case F_BLOCK_0: case F_BLOCK_1: case F_ALLBLOCK:
        fmr_status.s_fmr02 = 1;
        /* Not break */
    #if (defined(M16_26A) || defined(M16_28) || defined(M16_29))
    case F_BLOCK_2: case F_BLOCK_3: case F_BLOCK_4: case F_BLOCK_5:
        SlowMCU(&save_dat);     // Must change main clock speed to meet flash
        _FLASH_E_EW();
        fmr16 = 0;
        fmr16 = 1;
        _FLASH_DIS_EW();
        RestoreMCU(&save_dat);      // Restore clock back to original speed
    #endif
        break;
//    case F_ALLBLOCK: case F_BLOCK_A : case F_BLOCK_B :
    default:
        break;
    }
    asm("FSET I");
}

/***********************************************************************
Name        :LockBlockFlash()
Purpose     :Enable write protect.
Arguments   :F_BLOCK_0,F_BLOCK_1            Block0,1 are locked.
             F_BLOCK_2,F_BLOCK_3,F_BLOCK_4,F_BLOCK_5
                                            all area is locked.
             F_BLOCK_A,F_BLOCK_B don't have bit for write to lock.
Return      :None
Notice      :It depends on your system.
***********************************************************************/
void LockBlockFlash(enum FlashBlock blockNumber)
{
#if (defined(M16_26A) || defined(M16_28) || defined(M16_29))
ProcessorMode save_dat;
#endif
    asm("FCLR I");
    switch(blockNumber){
    #if (defined(M16_26A) || defined(M16_28) || defined(M16_29))
    case F_BLOCK_2: case F_BLOCK_3: case F_BLOCK_4: case F_BLOCK_5: case F_ALLBLOCK:
        SlowMCU(&save_dat);     // Must change main clock speed to meet flash
        _FLASH_E_EW();
        fmr16 = 0;
        _FLASH_DIS_EW();
```

```
        RestoreMCU(&save_dat);      // Restore clock back to original speed
        /* not break */
    #endif
    case F_BLOCK_0: case F_BLOCK_1:
        fmr_status.s_fmr02 = 0;
        break;
//    case F_ALLBLOCK: case F_BLOCK_A : case F_BLOCK_B :
    default:
        break;
    }
    asm("FSET I");
}
/***********************************************************************
Name        :SetUpLockBit()
Purpose     :set up lock bit.
Notice      :It depends on your system.
***********************************************************************/
void SetUpLockBit(void)
{
    if(fmr_status.s_fmr02){
        fmr02 = 0;
        fmr02 = 1;
    }
}
```

5.1.6 ncrt0_EW0.a30

```
;*********************************************************************** ;
; C COMPILER for R8C/Tiny, M16C/60,30,20,10
; COPYRIGHT(C) 1999(2000-2002) RENESAS TECHNOLOGY CORPORATION
; AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
;
;
; ncrt0.a30 : NC30 startup program
;
; This program is applicable when using the basic I/O library
;
; $Id: ncrt0_EW0.a30,v 1.4 2004/06/29 00:40:22 nagayoshi Exp $
;
;***********************************************************************

;------------------------------------------------------------------------
; HEEP SIZE definition
;------------------------------------------------------------------------
.if   __HEAP__ == 1
HEAPSIZE     .equ    0H
.else

.if   __HEAPSIZE__ == 0
HEAPSIZE     .equ    50H
.else
HEAPSIZE     .equ    __HEAPSIZE__
.endif

.endif

;------------------------------------------------------------------------
; STACK SIZE definition
;------------------------------------------------------------------------
.if   __USTACKSIZE__ == 0
STACKSIZE    .equ    300h
.else
STACKSIZE    .equ    __USTACKSIZE__
.endif

;------------------------------------------------------------------------
; INTERRUPT STACK SIZE definition
;------------------------------------------------------------------------
.if   __ISTACKSIZE__ == 0
ISTACKSIZE   .equ    300h
.else
ISTACKSIZE   .equ    __ISTACKSIZE__
.endif

;------------------------------------------------------------------------
; INTERRUPT VECTOR ADDRESS  definition
;------------------------------------------------------------------------
.if   __R8C__ != 1
;VECTOR_ADR  .equ    0ffd00h
VECTOR_ADR   .equ    0ff700h
.else
VECTOR_ADR   .equ    0fedch
.endif

;------------------------------------------------------------------------
; Section allocation
;------------------------------------------------------------------------
      .list OFF
      .include sect30_EW0.inc
      .list ON

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

```
; SBDATA area definition
;------------------------------------------------------------------------
      .glb    __SB__
__SB__        .equ    data_SE_top

;========================================================================
; Initialize Macro declaration
;------------------------------------------------------------------------
N_BZERO       .macro  TOP_ ,SECT_
      mov.b   #00H, R0L
      mov.w   #(TOP_ & 0FFFFH), A1
      mov.w   #sizeof SECT_ , R3
      sstr.b
      .endm

N_BCOPY .macro       FROM_,TO_,SECT_
      mov.w   #(FROM_ & 0FFFFH),A0
      mov.b   #(FROM_ >>16),R1H
      mov.w   #TO_ ,A1
      mov.w   #sizeof SECT_ , R3
      smovf.b
      .endm

BZERO .macro  TOP_,SECT_
      push.w  #sizeof SECT_ >> 16
      push.w  #sizeof SECT_  & 0ffffh
      pusha   TOP_ >>16
      pusha   TOP_ & 0ffffh
      .stk    8
      .glb    _bzero
      .call   _bzero,G
      jsr.a   _bzero
      .endm

BCOPY .macro  FROM_ ,TO_ ,SECT_
      push.w  #sizeof SECT_  >> 16
      push.w  #sizeof SECT_  & 0ffffh
      pusha   TO_ >>16
      pusha   TO_  & 0ffffh
      pusha   FROM_ >>16
      pusha   FROM_  & 0ffffh
      .stk    12
      .glb    _bcopy
      .call    _bcopy,G
      jsr.a   _bcopy
      .endm

.if   __R8C__ != 1
;
; for M16C/60,30,20,10 series
;
;             .glb    __BankSelect
;__BankSelect        .equ    0BH

;---------------------------------------------------------------
; special page definition
;---------------------------------------------------------------
;       macro define for special page
;
;Format:
;     SPECIAL number
;

SPECIAL       .macro  NUM
      .org    0FFFFEH-(NUM*2)
      .glb    __SPECIAL_@NUM
      .word   __SPECIAL_@NUM  & 0FFFFH
```

```
.endm
;======================================================================
; Interrupt section start
;----------------------------------------------------------------------
      .insf   start,S,0
      .glb    start
      .section       interrupt
start:
;----------------------------------------------------------------------
; after reset,this program will start
;----------------------------------------------------------------------
      ldc     #istack_top,   isp     ;set istack pointer
      mov.b   #03h,0ah
      mov.b   #00h,04h               ;set processer mode
      mov.b   #00100000B,07h ;set CM1
      mov.b    #00000000B,06h  ;set CM0
      mov.b    #00000000B,0Ch  ;set CM2
      mov.b   #00h,0ah
      ldc     #0080h, flg
      ldc     #stack_top,    sp      ;set stack pointer
      ldc     #data_SE_top,  sb      ;set sb register
;     ldintb  #VECTOR_ADR
;======================================================================
; Program RAM Initialize
;----------------------------------------------------------------------
; program area
;----------------------------------------------------------------------
    N_BCOPY _from_addr,_program_ram_top,program_ram
;----------------------------------------------------------------------
; vector area
;----------------------------------------------------------------------
    N_BCOPY VECTOR_ADR,_vector_table,vector_ram
;     ldintb  #_vector_table
      ldc  #(_vector_table >> 16),INTBH
    ldc  #(_vector_table & 0FFFFh),INTBL

;======================================================================
; NEAR area initialize.
;----------------------------------------------------------------------
; bss zero clear
;----------------------------------------------------------------------
      N_BZERO bss_SE_top,bss_SE
      N_BZERO bss_SO_top,bss_SO
      N_BZERO bss_NE_top,bss_NE
      N_BZERO bss_NO_top,bss_NO

;----------------------------------------------------------------------
; initialize data section
;----------------------------------------------------------------------
      N_BCOPY data_SEI_top,data_SE_top,data_SE
      N_BCOPY data_SOI_top,data_SO_top,data_SO
      N_BCOPY data_NEI_top,data_NE_top,data_NE
      N_BCOPY data_NOI_top,data_NO_top,data_NO

;======================================================================
; FAR area initialize.
;----------------------------------------------------------------------
; bss zero clear
;----------------------------------------------------------------------
;     BZERO   bss_FE_top,bss_FE
;     BZERO   bss_FO_top,bss_FO

;----------------------------------------------------------------------
; Copy edata_E(O) section from edata_EI(OI) section
;----------------------------------------------------------------------
;     BCOPY   data_FEI_top,data_FE_top,data_FE
;     BCOPY   data_FOI_top,data_FO_top,data_FO
```

```
      ldc     #stack_top,sp
      .stk    -40

;======================================================================
; heap area initialize
;----------------------------------------------------------------------
.if __HEAP__ != 1
      .glb    __mbase
      .glb    __mnext
      .glb    __msize
      mov.w   #(heap_top&0FFFFH), __mbase
      mov.w   #(heap_top>>16), __mbase+2
      mov.w   #(heap_top&0FFFFH), __mnext
      mov.w   #(heap_top>>16), __mnext+2
      mov.w   #(HEAPSIZE&0FFFFH), __msize
      mov.w   #(HEAPSIZE>>16), __msize+2
.endif

;======================================================================
; Initialize standard I/O
;----------------------------------------------------------------------
.if __STANDARD_IO__ != 1
      .glb    _init
      .call   _init,G
      jsr.a   _init
.endif

;======================================================================
; Call main() function
;----------------------------------------------------------------------
      ldc     #0h,fb  ; for debuger

      .glb    _main
      jsr.a   _main

.else ; __R8C__

;----------------------------------------------------------------------
; for R8C/Tiny
;----------------------------------------------------------------------

;======================================================================
; Interrupt section start
;----------------------------------------------------------------------
      .insf   start,S,0
      .glb    start
      .section       interrupt
start:
;----------------------------------------------------------------------
; after reset,this program will start
;----------------------------------------------------------------------
      ldc     #istack_top,   isp     ;set istack pointer
      mov.b   #02h,0ah
      mov.b   #00h,04h               ;set processer mode
      mov.b   #00h,0ah
      ldc     #0080h, flg
      ldc     #stack_top,    sp      ;set stack pointer
      ldc     #data_SE_top,  sb      ;set sb register
;     ldintb  #VECTOR_ADR

;======================================================================
; NEAR area initialize.
;----------------------------------------------------------------------
; bss zero clear
;----------------------------------------------------------------------
      N_BZERO bss_SE_top,bss_SE
```

```
      N_BZERO bss_SO_top,bss_SO
      N_BZERO bss_NE_top,bss_NE
      N_BZERO bss_NO_top,bss_NO

;---------------------------------------------------------------------
; initialize data section
;---------------------------------------------------------------------
      N_BCOPY data_SEI_top,data_SE_top,data_SE
      N_BCOPY data_SOI_top,data_SO_top,data_SO
      N_BCOPY data_NEI_top,data_NE_top,data_NE
      N_BCOPY data_NOI_top,data_NO_top,data_NO

;=====================================================================
; FAR area initialize.
;---------------------------------------------------------------------
; bss zero clear
;---------------------------------------------------------------------
;     BZERO   bss_FE_top,bss_FE
;     BZERO   bss_FO_top,bss_FO

;---------------------------------------------------------------------
; Copy edata_E(O) section from edata_EI(OI) section
;---------------------------------------------------------------------
;     BCOPY   data_FEI_top,data_FE_top,data_FE
;     BCOPY   data_FOI_top,data_FO_top,data_FO

      ldc     #stack_top,sp
;     .stk    -40

;=====================================================================
; heap area initialize
;---------------------------------------------------------------------
.if __HEAP__ != 1
      .glb    __mbase
      .glb    __mnext
      .glb    __msize
      mov.w   #(heap_top&0FFFFH), __mbase
      mov.w   #(heap_top&0FFFFH), __mnext
      mov.w   #(HEAPSIZE&0FFFFH), __msize
.endif

;=====================================================================
; Program RAM Initialize
;---------------------------------------------------------------------
; program area
;---------------------------------------------------------------------
      N_BCOPY _from_addr,_program_ram_top,program_ram
;---------------------------------------------------------------------
; vector area
;---------------------------------------------------------------------
    N_BCOPY VECTOR_ADR,_vector_table,vector_ram
      ldc  #(_vector_table >> 16),INTBH
      ldc  #(_vector_table & 0FFFFh),INTBL

;=====================================================================
; Initialize standard I/O
;---------------------------------------------------------------------
.if __STANDARD_IO__ != 1
      .glb    _init
      .call   _init,G
      jsr.a   _init
.endif

;=====================================================================
; Call main() function
;---------------------------------------------------------------------
      ldc     #0h,fb  ; for debuger
```

```
      .glb    _main
      jsr.a   _main

.endif       ; __R8C__

;===============================================================================
; exit() function
;-------------------------------------------------------------------------------
      .glb    _exit
      .glb    $exit
_exit:                          ; End program
$exit:
      jmp     _exit
      .einsf
;===============================================================================
; dummy interrupt function
;-------------------------------------------------------------------------------
dummy_int:
      reit

      .end
;****************************************************************************
;
; C COMPILER for R8C/Tiny, M16C/60,30,20,10
; COPYRIGHT(C) 1999(2000-2002) RENESAS TECHNOLOGY CORPORATION
; AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
;
;****************************************************************************
```

5.1.7 sect30_EW0.inc

```
;************************************************************************
;
; C Compiler for R8C/Tiny, M16C/60,30,20,10
; COPYRIGHT(C) 1999(2000-2002) RENESAS TECHNOLOGY CORPORATION
; AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
;
;
; Written by T.Aoyama
;
; sect30.inc     : section definition
; This program is applicable when using the basic I/O library
;
; $Id: sect30_EW0.inc,v 1.1 2004/06/25 01:29:56 nagayoshi Exp $
;
;************************************************************************

.if   __R8C__ != 1
;
;    for M16C/60,30,20,10
;
;----------------------------------------------------------------
;
;    Arrangement of section
;
;----------------------------------------------------------------
; Near RAM data area
;----------------------------------------------------------------
; SBDATA area
      .section       data_SE,DATA
      .org    400H
data_SE_top:

      .section       bss_SE,DATA,ALIGN
bss_SE_top:

      .section       data_SO,DATA
data_SO_top:

      .section       bss_SO,DATA
bss_SO_top:

; near RAM area
      .section       data_NE,DATA,ALIGN
data_NE_top:

      .section       bss_NE,DATA,ALIGN
bss_NE_top:

      .section       data_NO,DATA
data_NO_top:

      .section       bss_NO,DATA
bss_NO_top:

;----------------------------------------------------------------
; Stack area
;----------------------------------------------------------------
      .section       stack,DATA
      .blkb   STACKSIZE
stack_top:

      .blkb   ISTACKSIZE
istack_top:

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

```
;     heap section
;------------------------------------------------------------------
      .section       heap,DATA
heap_top:
      .blkb   HEAPSIZE

;------------------------------------------------------------------
; RAM program area
;------------------------------------------------------------------
      .section       program_ram,ALIGN
_program_ram_top:
      .glb    _program_ram_top

;------------------------------------------------------------------
; RAM vector area
;------------------------------------------------------------------
    .section vector_ram,data,ALIGN
    .ALIGN
_vector_table:
    .blkl   64
      .glb    _vector_table

;------------------------------------------------------------------
; Near ROM data area
;------------------------------------------------------------------
      .section       rom_NE,ROMDATA
      .org    0f000H
rom_NE_top:

      .section       rom_NO,ROMDATA
rom_NO_top:

;------------------------------------------------------------------
; Far RAM data area
;------------------------------------------------------------------
      .section       data_FE,DATA
      .org           10000H
data_FE_top:

      .section       bss_FE,DATA,ALIGN
bss_FE_top:

      .section       data_FO,DATA
data_FO_top:

      .section       bss_FO,DATA
bss_FO_top:


;------------------------------------------------------------------
; Far ROM data area
;------------------------------------------------------------------
      .section       rom_FE,ROMDATA
      .org           0F8000H
rom_FE_top:

      .section       rom_FO,ROMDATA
rom_FO_top:

;------------------------------------------------------------------
; Initial data of 'data' section
;------------------------------------------------------------------
      .section       data_SEI,ROMDATA
data_SEI_top:

      .section       data_SOI,ROMDATA
data_SOI_top:
```

```
      .section       data_NEI,ROMDATA
data_NEI_top:

      .section       data_NOI,ROMDATA
data_NOI_top:

      .section       data_FEI,ROMDATA
data_FEI_top:

      .section       data_FOI,ROMDATA
data_FOI_top:

;------------------------------------------------------------------
; Switch Table Section
;------------------------------------------------------------------
      .section        switch_table,ROMDATA
switch_table_top:


;------------------------------------------------------------------
; code area
;------------------------------------------------------------------

      .section       program

      .section       interrupt
      ;.org   ;must be set internal ROM area
      .section       program_S

;------------------------------------------------------------------
; variable vector section
;------------------------------------------------------------------
      .section       vector,ROMDATA ; variable vector table
      .org    VECTOR_ADR

.if   M60TYPE == 1
      .lword  dummy_int             ; vector 0 (BRK)
      .lword  dummy_int             ; vector 1
      .lword  dummy_int             ; vector 2
      .lword  dummy_int             ; vector 3
      .lword  dummy_int             ; vector 4
      .lword  dummy_int             ; vector 5
      .lword  dummy_int             ; vector 6
      .lword  dummy_int             ; vector 7
      .lword  dummy_int             ; vector 8
      .lword  dummy_int             ; vector 9
      .lword  dummy_int             ; vector 10
      .lword  dummy_int             ; DMA0 (for user) (vector 11)
      .lword  dummy_int             ; DMA1 2 (for user) (vector 12)
      .lword  dummy_int             ; input key (for user) (vector 13)
      .lword  dummy_int             ; AD Convert (for user) (vector 14)
      .lword  dummy_int             ; vector 15
      .lword  dummy_int             ; vector 16
      .lword  dummy_int             ; uart0 trance (for user) (vector 17)
      .lword  dummy_int             ; uart0 receive (for user) (vector 18)
      .lword  dummy_int             ; uart1 trance (for user) (vector 19)
      .lword  dummy_int             ; uart1 receive (for user) (vector 20)
      .lword  dummy_int             ; TIMER A0 (for user) (vector 21)
      .lword  dummy_int             ; TIMER A1 (for user) (vector 22)
      .lword  dummy_int             ; TIMER A2 (for user) (vector 23)
      .lword  dummy_int             ; TIMER A3 (for user) (vector 24)
      .lword   dummy_int            ; TIMER A4 (for user) (vector 25)
      .lword  dummy_int             ; TIMER B0 (for user) (vector 26)
      .lword  dummy_int             ; TIMER B1 (for user) (vector 27)
      .lword  dummy_int             ; TIMER B2 (for user) (vector 28)
      .lword  dummy_int             ; INT0 (for user) (vector 29)
```

```
      .lword  dummy_int             ; INT1 (for user) (vector 30)
      .lword  dummy_int             ; INT2 (for user) (vector 31)
.else
      .glb    _timerA1_int
      .glb     _vec_dummy_int
      .lword  _vec_dummy_int        ; BRK   (vector 0)
      .lword  _vec_dummy_int        ;       (vector 1)
      .lword  _vec_dummy_int        ;       (vector 2)
      .lword  _vec_dummy_int        ;       (vector 3)
      .lword  _vec_dummy_int        ; int3(for user)(vector 4)
      .lword  _vec_dummy_int        ; timerB5(for user)(vector 5)
      .lword  _vec_dummy_int        ; timerB4(for user)(vector 6)
      .lword  _vec_dummy_int        ; timerB3(for user)(vector 7)
      .lword  _vec_dummy_int        ; si/o4 /int5(for user)(vector 8)
      .lword  _vec_dummy_int        ; si/o3 /int4(for user)(vector 9)
      .lword  _vec_dummy_int        ; Bus collision detection(for user)(v10)
      .lword  _vec_dummy_int        ; DMA0(for user)(vector 11)
      .lword  _vec_dummy_int        ; DMA1(for user)(vector 12)
      .lword  _vec_dummy_int        ; Key input interrupt(for user)(vect 13)
      .lword  _vec_dummy_int        ; A-D(for user)(vector 14)
      .lword  _vec_dummy_int        ; uart2 transmit(for user)(vector 15)
      .lword  _vec_dummy_int        ; uart2 receive(for user)(vector 16)
      .lword  _vec_dummy_int        ; uart0 transmit(for user)(vector 17)
      .lword  _vec_dummy_int        ; uart0 receive(for user)(vector 18)
      .lword  _vec_dummy_int        ; uart1 transmit(for user)(vector 19)
      .lword  _vec_dummy_int        ; uart1 receive(for user)(vector 20)
      .lword  _vec_dummy_int        ; timer A0(for user)(vector 21)
      .lword  _timerA1_int          ; timer A1(for user)(vector 22)
      .lword  _vec_dummy_int        ; timer A2(for user)(vector 23)
      .lword  _vec_dummy_int        ; timer A3(for user)(vector 24)
      .lword  _vec_dummy_int        ; timer A4(for user)(vector 25)
      .lword  _vec_dummy_int        ; timer B0(for user)(vector 26)
      .lword  _vec_dummy_int        ; timer B1(for user)(vector 27)
      .lword  _vec_dummy_int        ; timer B2(for user)(vector 28)
      .lword  _vec_dummy_int        ; int0 (for user)(vector 29)
      .lword  _vec_dummy_int        ; int1 (for user)(vector 30)
      .lword  _vec_dummy_int        ; int2 (for user)(vector 31)
.endif
      .lword  dummy_int             ; vector 32 (for user or MR30)
      .lword  dummy_int             ; vector 33 (for user or MR30)
      .lword  dummy_int             ; vector 34 (for user or MR30)
      .lword  dummy_int             ; vector 35 (for user or MR30)
      .lword  dummy_int             ; vector 36 (for user or MR30)
      .lword  dummy_int             ; vector 37 (for user or MR30)
      .lword  dummy_int             ; vector 38 (for user or MR30)
      .lword  dummy_int             ; vector 39 (for user or MR30)
      .lword  dummy_int             ; vector 40 (for user or MR30)
      .lword  dummy_int             ; vector 41 (for user or MR30)
      .lword  dummy_int             ; vector 42 (for user or MR30)
      .lword  dummy_int             ; vector 43 (for user or MR30)
      .lword   dummy_int            ; vector 44 (for user or MR30)
      .lword  dummy_int             ; vector 45 (for user or MR30)
      .lword  dummy_int             ; vector 46 (for user or MR30)
      .lword  dummy_int             ; vector 47 (for user or MR30)
      .lword  dummy_int             ; vector 48
      .lword  dummy_int             ; vector 49
      .lword  dummy_int             ; vector 50
      .lword  dummy_int             ; vector 51
      .lword  dummy_int             ; vector 52
      .lword  dummy_int             ; vector 53
      .lword  dummy_int             ; vector 54
      .lword  dummy_int             ; vector 55
      .lword  dummy_int             ; vector 56
      .lword  dummy_int             ; vector 57
      .lword  dummy_int             ; vector 58
      .lword  dummy_int             ; vector 59
      .lword  dummy_int             ; vector 60
```

```
      .lword  dummy_int             ; vector 61
      .lword  dummy_int             ; vector 62
      .lword  dummy_int             ; vector 63

;===============================================================
; fixed vector section
;---------------------------------------------------------------
      .section       fvector,ROMDATA                     ; fixed vector table
;===============================================================
; special page defination
;---------------------------------------------------------------
;     macro is defined in ncrt0.a30
;     Format: SPECIAL number
;
;---------------------------------------------------------------
;     SPECIAL 255
;     SPECIAL 254
;     SPECIAL 253
;     SPECIAL 252
;     SPECIAL 251
;     SPECIAL 250
;     SPECIAL 249
;     SPECIAL 248
;     SPECIAL 247
;     SPECIAL 246
;     SPECIAL 245
;     SPECIAL 244
;     SPECIAL 243
;     SPECIAL 242
;     SPECIAL 241
;     SPECIAL 240
;     SPECIAL 239
;     SPECIAL 238
;     SPECIAL 237
;     SPECIAL 236
;     SPECIAL 235
;     SPECIAL 234
;     SPECIAL 233
;     SPECIAL 232
;     SPECIAL 231
;     SPECIAL 230
;     SPECIAL 229
;     SPECIAL 228
;     SPECIAL 227
;     SPECIAL 226
;     SPECIAL 225
;     SPECIAL 224
;     SPECIAL 223
;     SPECIAL 222
;     SPECIAL 221
;     SPECIAL 220
;     SPECIAL 219
;     SPECIAL 218
;     SPECIAL 217
;     SPECIAL 216
;     SPECIAL 215
;     SPECIAL 214
;     SPECIAL 213
;     SPECIAL 212
;     SPECIAL 211
;     SPECIAL 210
;     SPECIAL 209
;     SPECIAL 208
;     SPECIAL 207
;     SPECIAL 206
;     SPECIAL 205
;     SPECIAL 204
```

```
;     SPECIAL 203
;     SPECIAL 202
;     SPECIAL 201
;     SPECIAL 200
;     SPECIAL 199
;     SPECIAL 198
;     SPECIAL 197
;     SPECIAL 196
;     SPECIAL 195
;     SPECIAL 194
;     SPECIAL 193
;     SPECIAL 192
;     SPECIAL 191
;     SPECIAL 190
;     SPECIAL 189
;     SPECIAL 188
;     SPECIAL 187
;     SPECIAL 186
;     SPECIAL 185
;     SPECIAL 184
;     SPECIAL 183
;     SPECIAL 182
;     SPECIAL 181
;     SPECIAL 180
;     SPECIAL 179
;     SPECIAL 178
;     SPECIAL 177
;     SPECIAL 176
;     SPECIAL 175
;     SPECIAL 174
;     SPECIAL 173
;     SPECIAL 172
;     SPECIAL 171
;     SPECIAL 170
;     SPECIAL 169
;     SPECIAL 168
;     SPECIAL 167
;     SPECIAL 166
;     SPECIAL 165
;     SPECIAL 164
;     SPECIAL 163
;     SPECIAL 162
;     SPECIAL 161
;     SPECIAL 160
;     SPECIAL 159
;     SPECIAL 158
;     SPECIAL 157
;     SPECIAL 156
;     SPECIAL 155
;     SPECIAL 154
;     SPECIAL 153
;     SPECIAL 152
;     SPECIAL 151
;     SPECIAL 150
;     SPECIAL 149
;     SPECIAL 148
;     SPECIAL 147
;     SPECIAL 146
;     SPECIAL 145
;     SPECIAL 144
;     SPECIAL 143
;     SPECIAL 142
;     SPECIAL 141
;     SPECIAL 140
;     SPECIAL 139
;     SPECIAL 138
;     SPECIAL 137
```

```
;     SPECIAL 136
;     SPECIAL 135
;     SPECIAL 134
;     SPECIAL 133
;     SPECIAL 132
;     SPECIAL 131
;     SPECIAL 130
;     SPECIAL 129
;     SPECIAL 128
;     SPECIAL 127
;     SPECIAL 126
;     SPECIAL 125
;     SPECIAL 124
;     SPECIAL 123
;     SPECIAL 122
;     SPECIAL 121
;     SPECIAL 120
;     SPECIAL 119
;     SPECIAL 118
;     SPECIAL 117
;     SPECIAL 116
;     SPECIAL 115
;     SPECIAL 114
;     SPECIAL 113
;     SPECIAL 112
;     SPECIAL 111
;     SPECIAL 110
;     SPECIAL 109
;     SPECIAL 108
;     SPECIAL 107
;     SPECIAL 106
;     SPECIAL 105
;     SPECIAL 104
;     SPECIAL 103
;     SPECIAL 102
;     SPECIAL 101
;     SPECIAL 100
;     SPECIAL 99
;     SPECIAL 98
;     SPECIAL 97
;     SPECIAL 96
;     SPECIAL 95
;     SPECIAL 94
;     SPECIAL 93
;     SPECIAL 92
;     SPECIAL 91
;     SPECIAL 90
;     SPECIAL 89
;     SPECIAL 88
;     SPECIAL 87
;     SPECIAL 86
;     SPECIAL 85
;     SPECIAL 84
;     SPECIAL 83
;     SPECIAL 82
;     SPECIAL 81
;     SPECIAL 80
;     SPECIAL 79
;     SPECIAL 78
;     SPECIAL 77
;     SPECIAL 76
;     SPECIAL 75
;     SPECIAL 74
;     SPECIAL 73
;     SPECIAL 72
;     SPECIAL 71
;     SPECIAL 70
```

```
;     SPECIAL 69
;     SPECIAL 68
;     SPECIAL 67
;     SPECIAL 66
;     SPECIAL 65
;     SPECIAL 64
;     SPECIAL 63
;     SPECIAL 62
;     SPECIAL 61
;     SPECIAL 60
;     SPECIAL 59
;     SPECIAL 58
;     SPECIAL 57
;     SPECIAL 56
;     SPECIAL 55
;     SPECIAL 54
;     SPECIAL 53
;     SPECIAL 52
;     SPECIAL 51
;     SPECIAL 50
;     SPECIAL 49
;     SPECIAL 48
;     SPECIAL 47
;     SPECIAL 46
;     SPECIAL 45
;     SPECIAL 44
;     SPECIAL 43
;     SPECIAL 42
;     SPECIAL 41
;     SPECIAL 40
;     SPECIAL 39
;     SPECIAL 38
;     SPECIAL 37
;     SPECIAL 36
;     SPECIAL 35
;     SPECIAL 34
;     SPECIAL 33
;     SPECIAL 32
;     SPECIAL 31
;     SPECIAL 30
;     SPECIAL 29
;     SPECIAL 28
;     SPECIAL 27
;     SPECIAL 26
;     SPECIAL 25
;     SPECIAL 24
;     SPECIAL 23
;     SPECIAL 22
;     SPECIAL 21
;     SPECIAL 20
;     SPECIAL 19
;     SPECIAL 18
;
;=====================================================================
; fixed vector section
;---------------------------------------------------------------------
      .org    0fffdch
UDI:
      .lword  dummy_int
OVER_FLOW:
      .lword  dummy_int
BRKI:
      .lword  dummy_int
ADDRESS_MATCH:
      .lword  dummy_int
SINGLE_STEP:
      .lword  dummy_int
```

```
WDT:
      .lword  dummy_int
DBC:
      .lword  dummy_int
NMI:
      .lword  dummy_int
      .org    0ffffcH
RESET:
      .lword  start

.else ; __R8C__

;
; for R8C/Tiny
;
;------------------------------------------------------------
;
;     Arrangement of section
;
;------------------------------------------------------------
; Near RAM data area
;------------------------------------------------------------
; SBDATA area
      .section       data_SE,DATA
      .org    400H
data_SE_top:

      .section       bss_SE,DATA,ALIGN
bss_SE_top:

      .section       data_SO,DATA
data_SO_top:

      .section       bss_SO,DATA
bss_SO_top:

; near RAM area
      .section       data_NE,DATA,ALIGN
data_NE_top:

      .section       bss_NE,DATA,ALIGN
bss_NE_top:

      .section       data_NO,DATA
data_NO_top:

      .section       bss_NO,DATA
bss_NO_top:

;------------------------------------------------------------
; Stack area
;------------------------------------------------------------
      .section       stack,DATA,ALIGN
      .blkb   STACKSIZE
stack_top:

      .blkb   ISTACKSIZE
istack_top:

;------------------------------------------------------------
;     heap section
;------------------------------------------------------------
      .section       heap,DATA
heap_top:
      .blkb   HEAPSIZE

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

```
; RAM program area
;------------------------------------------------------------------
      .section       program_ram,ALIGN
_program_ram_top:
      .glb    _program_ram_top

;------------------------------------------------------------------
; RAM vector area
;------------------------------------------------------------------
    .section vector_ram,data,ALIGN
    .ALIGN
_vector_table:
    .blkl   64
      .glb    _vector_table

;------------------------------------------------------------------
; Near ROM data area
;------------------------------------------------------------------
      .section       rom_NE,ROMDATA
      .org           0e000H
rom_NE_top:

      .section       rom_NO,ROMDATA
rom_NO_top:

;------------------------------------------------------------------
; Initial data of 'data' section
;------------------------------------------------------------------
      .section       data_SEI,ROMDATA,ALIGN
data_SEI_top:

      .section       data_SOI,ROMDATA
data_SOI_top:

      .section       data_NEI,ROMDATA,ALIGN
data_NEI_top:

      .section       data_NOI,ROMDATA
data_NOI_top:

;------------------------------------------------------------------
; Switch Table Section
;------------------------------------------------------------------
      .section        switch_table,ROMDATA
switch_table_top:

;------------------------------------------------------------------
; code area
;------------------------------------------------------------------

      .section       program,CODE,ALIGN

      .section       interrupt,CODE,ALIGN

;------------------------------------------------------------------
; variable vector section
;------------------------------------------------------------------
      .section       vector,ROMDATA ; variable vector table
      .org    VECTOR_ADR
      .glb    _int_timerx

      .lword  dummy_int             ; vector 0
      .lword  dummy_int             ; vector 1
      .lword  dummy_int             ; vector 2
      .lword  dummy_int             ; vector 3
      .lword  dummy_int             ; vector 4
      .lword  dummy_int             ; vector 5
```

```
      .lword  dummy_int             ; vector 6
      .lword  dummy_int             ; vector 7
      .lword  dummy_int             ; vector 8
      .lword  dummy_int             ; vector 9
      .lword  dummy_int             ; vector 10
      .lword  dummy_int             ; vector 11
      .lword  dummy_int             ; vector 12
      .lword  dummy_int             ; vector 13
      .lword  dummy_int             ; vector 14
      .lword  dummy_int             ; vector 15
      .lword  dummy_int             ; vector 16
      .lword  dummy_int             ; vector 17
      .lword  dummy_int             ; vector 18
      .lword  dummy_int             ; vector 19
      .lword  dummy_int             ; vector 20
      .lword  dummy_int             ; vector 21
      .lword   _int_timerx          ; timerx(vector 22)
      .lword  dummy_int             ; vector 23
      .lword  dummy_int             ; vector 24
      .lword  dummy_int             ; vector 25
      .lword  dummy_int             ; vector 26
      .lword  dummy_int             ; vector 27
      .lword  dummy_int             ; vector 28
      .lword  dummy_int             ; vector 29
      .lword  dummy_int             ; vector 30
      .lword  dummy_int             ; vector 31
      .lword  dummy_int             ; vector 32
      .lword  dummy_int             ; vector 33
      .lword  dummy_int             ; vector 34
      .lword  dummy_int             ; vector 35
      .lword  dummy_int             ; vector 36
      .lword  dummy_int             ; vector 37
      .lword  dummy_int             ; vector 38
      .lword  dummy_int             ; vector 39
      .lword  dummy_int             ; vector 40
      .lword  dummy_int             ; vector 41
      .lword  dummy_int             ; vector 42
      .lword  dummy_int             ; vector 43
      .lword   dummy_int            ; vector 44
      .lword  dummy_int             ; vector 45
      .lword  dummy_int             ; vector 46
      .lword  dummy_int             ; vector 47
      .lword  dummy_int             ; vector 48
      .lword  dummy_int             ; vector 49
      .lword  dummy_int             ; vector 50
      .lword  dummy_int             ; vector 51
      .lword  dummy_int             ; vector 52
      .lword  dummy_int             ; vector 53
      .lword  dummy_int             ; vector 54
      .lword  dummy_int             ; vector 55
      .lword  dummy_int             ; vector 56
      .lword  dummy_int             ; vector 57
      .lword  dummy_int             ; vector 58
      .lword  dummy_int             ; vector 59
      .lword  dummy_int             ; vector 60
      .lword  dummy_int             ; vector 61
      .lword  dummy_int             ; vector 62
      .lword  dummy_int             ; vector 63

;===============================================================
; fixed vector section
;---------------------------------------------------------------
      .section       fvector,ROMDATA              ; fixed vector table
;     .org    0ffdcH
;UDI:
;     .lword  dummy_int
;OVER_FLOW:
```

```
;     .lword  dummy_int
;BRKI:
;     .lword  dummy_int
;ADDRESS_MATCH:
;     .lword  dummy_int
;SINGLE_STEP:
;     .lword  dummy_int
;WDT:
;     .lword  dummy_int
;DBC:
;     .lword  dummy_int
;NMI:
;     .lword  dummy_int
      .org    0fffcH
RESET:
      .lword  start | 0FF000000H

.endif       ; __R8C

;------------------------------------------------------------------
; far ROM data area
;------------------------------------------------------------------
;
;     .section       rom_FE,ROMDATA
;     .org           10000H
;
;     .section       rom_FO,ROMDATA
;
;     .section       data_FEI,ROMDATA,ALIGN
;data_FEI_top:
;
;     .section       data_FOI,ROMDATA
;data_FOI_top:
;
;*****************************************************************************
;
;     C Compiler for R8C/Tiny, M16C/60,30,20,10
; COPYRIGHT(C) 1999(2000-2002) RENESAS TECHNOLOGY CORPORATION
; AND RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
;
;*****************************************************************************
```

5.1.8 main_m16c.c

```
/***********************************************************************/
/*    FILE NAME : main_m16c.c                                          */
/*    Ver       : 1.00                                                 */
/*    CPU       : M16C/26                                              */
/*    FUNCTION  : Data Flash rewrite Apllication Note sample program */
/*                for EW0 mode operation                               */
/*---------------------------------------------------------------------*/
/*  Copyright(C)2004, Renesas Technology Corp.                         */
/*  Copyright(C)2004, Renesas Solutions Corp.                          */
/*  All rights reserved.                                               */
/***********************************************************************/
// $Id: main_m16c.c,v 1.6 2004/06/30 02:28:41 nagayoshi Exp $
#include <string.h>
#include <stdio.h>
#include "sfr28.h"
#include "flashdevdrv.h"

#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE 1
#endif

typedef unsigned char BOOL;

#define     BLOCK_SIZE              2048

#pragma INTERRUPT timerA1_int;
void timerA1_int(void);

/*** Interrupt dummy (running on ram) ***/
#pragma INTERRUPT vec_dummy_int;
void far vec_dummy_int(void);

void ErrorDisp(const char * dt);

void mcu_init(void);

int CheckErasedBlank(void F_ADR_SIZE * f_addr, short size);
int CmpBlank(unsigned char F_ADR_SIZE * buf , short size);

/*=====================================================*/
/** Timer ********/
/*=====================================================*/
enum TimerSource{
    TIMER_DEV_1  = 0x00,
    TIMER_DEV_8  = 0x40,
    TIMER_DEV_32 = 0x80,
    TIMER_SUB_32 = 0xC0,
};
#define TIMER_CLOCK_Hz    20000000
#define TIMER_SUBCLOCK_Hz 32768
#define TA1_DEV32_MS(a) ((TIMER_CLOCK_Hz / 32000) * (a) - 1)
#define TA1_DEV8_MS(a)  ((TIMER_CLOCK_Hz / 8000) * (a) - 1)
#define TA1_DEV1_MS(a)  ((TIMER_CLOCK_Hz / 1000) * (a) - 1)

BOOL CheckTa1Passed(void);

void SetTickTimer(unsigned char tm);

void TimerA1InitTimerMode(enum TimerSource source,
                          unsigned short timer);

inline void StartTimerA1(void);
```

```
inline void StartTimerA1(void)
{
    ta1s = 1;                   // TimerA1 start
}

void ClearTotalTimer(void);

void SystemTimerInc(void);
/** RAM for Timer **/
BOOL tm_ps = FALSE;
unsigned char tm_ms = 0;
unsigned long totalTimer = 0;  // ms

inline unsigned long GetTotalTimer(void);
inline unsigned long GetTotalTimer(void)
{
    return totalTimer;
}
/*======================================================*/
/** Timer End ****/
/*======================================================*/
enum MainMode{
    ERASE_TEST_START,
    ERASE_TEST_RESTART,
    ERASE_TEST_CHECK,
    PROGRAM_TEST,
    OTHER,
} mode = ERASE_TEST_START;
const char TestData[33] = "0123456789ABCDEF0123456789ABCDEF";
/*********************************************************************
// Main loop
*********************************************************************/
void main(void)
{
    unsigned char buffer_addr[32];
    FlashResult err_code = F_SUCCESS;

    mcu_init();         /* initialize MCU                   */

    FlashInitialize();  /*  FlashMemory Initialize          */

    StartTimerA1();

    asm("fset i");

    /* Unlock gives a Flash block to write. (This example is unnecessary.) */
    UnlockBlockFlash(F_BLOCK_3);

    /* example : When a main loop is done in 20mS and made to work.

    The turn of the movement
    mode                    Address and contents of a test
     << start >>
    ERASE_TEST_START        0xF000-0xF7FF Start Erasing
    ERASE_TEST_RESTART      0xF000-0xF7FF Restart Erasing
    ERASE_TEST_CHECK        0xF000-0xF7FF Erasing confirmation
    PROGRAM_TEST            0xF000-0xF01F Write and confirmation
    OTHER                   Wait until it passes from the erasing start for one second.
                            Judge it as the error, and reset a flash memory
                            when erasing isn't completed in one second.
      << Repetition >>
    */

    for(;;){

        /* Waiting 20ms passed */
        while(!CheckTa1Passed());
```

```
        switch(mode){
        case ERASE_TEST_START:
        case ERASE_TEST_RESTART:
            /* Start/Restart erasing    */
            /* So that it may clear a suspend requirement to do ResumErase()
               before resuming elimination.                                    */
            if(mode == ERASE_TEST_START){
                ClearTotalTimer();  /* Make totalTimer 0 for the erasing time acquisition. */
                err_code = StartEraseFlash((void F_ADR_SIZE *)0xF7FE);
            }
            else {
                ResumErase();
                err_code = RestartEraseFlash();
            }
            /* Check err code. */
            switch(err_code){
            case F_SUSPEND: mode = ERASE_TEST_RESTART;  break;
            case F_SUCCESS: mode = ERASE_TEST_CHECK;    break;
            default:        ErrorDisp("EraseER1");      break;
            }
            /* TimeOut Check */
            /* Take an error when F_SUSPEND occurs for one second after you start elimination. */
            if(GetTotalTimer() >= 1000){
                if(mode == ERASE_TEST_RESTART){
                    FlashReset();       /* Make condition revert with FlashReset(). */
                    ErrorDisp("Err Tout");
                }
            }
            break;
        case ERASE_TEST_CHECK:
            /* Check whether even an erase block (0xF000-0xF7FFF) is being erased.  */
            if(0 == CheckErasedBlank((void F_ADR_SIZE *)0xF000, BLOCK_SIZE)){
                ErrorDisp("EraseER2");
            }
            mode = PROGRAM_TEST;
            break;
        case PROGRAM_TEST:
            /* Writing 32byte and error code check      */
            err_code = WriteFlash((void F_ADR_SIZE *)0xF000,TestData,32);
            if(err_code != F_SUCCESS)ErrorDisp("WriteERR");
            /* The data being written check whether it begins to read it. */
            err_code = ReadFlash((void F_ADR_SIZE *)0xF000 , buffer_addr, 32);
            if(err_code != F_SUCCESS)ErrorDisp("Read ERR");
            /* Compare data of write and data of read. */
            if(memcmp(buffer_addr , TestData , 32)){
                ErrorDisp("Comp ERR");
            }
            mode = OTHER;
            break;
         default:
            if(GetTotalTimer() >= 1000){
                mode = ERASE_TEST_START;
            }
        }
    }

}


/*================================
 * Confirm whether a flash memory is in the blank.
 * f_addr : physical address on flash memory to confirm.
 * size   : Number of bytes to confirm.
 * return : 1:blank
 *        : 0:not blank
 ================================*/
```

```
int CheckErasedBlank(void F_ADR_SIZE * f_addr, short size)
{
    unsigned char F_ADR_SIZE * faddr = f_addr;
    FlashResult err_code = F_SUCCESS;
    unsigned short r_buf[16];
    unsigned short r_size;
    for(; size > 0 ; ){
        r_size = (size > sizeof(r_buf))? sizeof(r_buf):size;
        err_code = ReadFlash(faddr , r_buf , r_size);
        if(CmpBlank((unsigned char F_ADR_SIZE * )r_buf , r_size)){
            return 0;
        }
        faddr += r_size;
        size -= r_size;
    }
    return 1;
}

/*================================
 * Compare the matter whether designated data are "BLANK_PATTERN".
 * f_addr : physical address on flash memory to confirm.
 * size   : Number of bytes to confirm.
 * return : 1:blank
 *        : 0:not blank
 ================================*/
#define BLANK_PATTERN 0xff
int CmpBlank(unsigned char F_ADR_SIZE * buf , short size)
{
    while(size --){
        if(* buf ++ != BLANK_PATTERN)return -1;
    }
    return 0;
}

/*================================
 Error display and cancellation of a movement.
 ================================*/
void ErrorDisp(const char * dt)
{
//    DISPLAY(1, dt);
    while(1);
}

/*================================
 initialize MCU
 ================================*/
void mcu_init( void )
{

   /* Select full speed operation */
   /* Switch port initialization */
   pd10_5 = 0;      // change switch ports to inputs
   pd10_6 = 0;
   pd10_7 = 0;

   /* LED initialization */
   pd7_0 = 1;       // Change LED ports to outputs (connected to LEDs)
   pd7_1 = 1;
   pd7_2 = 1;

   /* unused pins - configure as outputs to decrease power consumption */
   pd6 = 0x90;

   pd8_0 = 1;
   pd8_1 = 1;
   pd8_2 = 1;
   pd8_3 = 1;
```

```
    prc2 = 1;        // P9 is write protected - disable protection before writing to P9
    pd9_0 = 1;
    pd9_1 = 1;
    pd9_2 = 1;
    pd9_3 = 1;
    prc2 = 0;        // Write protect P9

    pd10_0 = 1;
    pd10_1 = 1;
    pd10_2 = 1;
    pd10_3 = 1;
    pd10_4 = 1;

// Set up a Timer A1
     TimerA1InitTimerMode(TIMER_DEV_32,TA1_DEV32_MS(20));
     SetTickTimer(20);

     ta1ic = 0x07;                      // Set Timer-A1 Interrupt-Priolity-Level

// Port Initialize
     p6   = 0x00;                       // Port-6 clear
//     pd6  = 0xff;                        // Port-6 is output port

     p7   = 0x07;                       // p7_0 - p7_2 LED off
     pd7  = 0x07;                       // p7_0 to p7_2 output select

     pu25 = 1;                          // p10_4 to P10_7 pull-up
     p10  = 0x00;                       // Port-10 clear
     pd10 = 0x1f;                       // p10_5 to p10_7 is Key-in port

}


/*************************************/
/* Timer                             */
/*************************************/
/*================================
 Check tick time.
 ================================*/
BOOL CheckTa1Passed(void)
{
    if(tm_ps){
        tm_ps = FALSE;
        return TRUE;
    }
    return FALSE;
}
/*================================
 Set up value of tick time.
 ================================*/
void SetTickTimer(unsigned char tm)
{
    tm_ms = tm;
}
/*================================
 Initialize Timer A1.
 ================================*/
void TimerA1InitTimerMode(enum TimerSource source,
                          unsigned short timer)
{
// Set up a Timer A1
    ta1s  = 0;                          // Timer-A1 Stopped
    ta1ic = 0x07;                       // Set Timer-A1 Interrupt-Priolity-Level

    ta1mr = source;                     // Set Timer-A1 mode regster
                                        //  Mode=Timer-mode,Count-src=f1
```

```
    ta1   = timer;              // Set Timer-A1 timer-value (50ms)
}
/*================================
 Clear totaltimer
 ================================*/
void ClearTotalTimer(void)
{
    totalTimer = 0;
}
/*****************************************************************************
 The following is a program to work by the RAM.(EW0 only)
*****************************************************************************/
#ifdef FLASH_MODE_EW0
#pragma SECTION program program_ram
#endif
/*================================
 Timer A1 Interrupt function
 ================================*/
void timerA1_int(void)
{
    /* Suspend erasing and advance timer. */
    SuspendErase();
    SystemTimerInc();
}

/*================================
 This is function for Timer A1 interrupt.
 ================================*/
void SystemTimerInc(void)
{
    // 1ms
    tm_ps = TRUE;
    totalTimer += tm_ms;
}

/*================================
 dummy Interrupt
 ================================*/
void vec_dummy_int(void)
{
}
```

5.1.9 M16C_EW0.tmk

```
  ######################################################################
  # Makefile for TM V.3.20A
  # COPYRIGHT(C) 1998(1998-2003)
  # RENESAS TECHNOLOGY CORPORATION ALL RIGHTS RESERVED
  # AND
  # RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
  #
  #     Notice  :       Don't edit.
  #     Date    :       2004 06(June) 25(Friday) AM.10.42
  #     Project :       M16C_EW0
  ######################################################################

  DELETE       =       @-del
  LNLIST       =       $(PROJECT).cmd
  FROM_ADDR    =       0FB000
  LMC   =      LMC30
  CC    =      NC30
  AR    =      LB30
  UTL   =      utl30
  AS    =      AS30
  LIBFILE      =       $(PROJECT).lib
  OUTDIR       =       M16CEW0
  MKDIR =      @-mkdir
  ABSFILE      =       $(PROJECT).x30
  ODINCMD      =       $(OUTDIR)
  LN    =      LN30
  TARGET       =       $(ABSFILE)
  ECHO  =      @-echo
  MKFILE       =       $(PROJECT).tmk
  PROJECT      =       M16C_EW0
  TYPE  =      @-type
  LFLAGS       =       -MS -L nc30lib -G -LOC program_ram=$(FROM_ADDR) -O $(OUTDIR)\$(TARGET)
  UTLFLAGS     =
  CFLAGS       =       -c -dir $(OUTDIR) -g -gbool_to_char -OR -O4 -OSA -finfo -fUD -fNA -fSA -WNP -
WUP -WNC -Wall -WUV -WNUA
  LMCFLAGS     =       -L
  LIBFLAGS     =       -C
  AFLAGS       =       -LM -D__HEAP__=1 -D__STANDARD_IO__=1 -
D_from_addr=$(FROM_ADDR)h:__USTACKSIZE__=160h:__ISTACKSIZE__=160h -finfo -O$(OUTDIR)
  .SUFFIXES: .a30 .r30 .c .x30 .lib
  .PHONY: all
  all: \
               $(OUTDIR)\$(TARGET)
  .PHONY: clean
  clean:
       $(DELETE) $(OUTDIR)\$(TARGET)
       $(DELETE) $(ODINCMD)\$(LNLIST)
       $(DELETE) $(OUTDIR)\ncrt0_EW0.r30
       $(DELETE) $(OUTDIR)\depend_m16c.r30
       $(DELETE) $(OUTDIR)\flashdrvdev_ew0.r30
       $(DELETE) $(OUTDIR)\main_m16c.r30
  $(ODINCMD)\$(LNLIST): \
               .\$(MKFILE)
       $(ECHO)\$(MRLFLAGS) $(LFLAGS) >       $(ODINCMD)\$(LNLIST)
       $(ECHO)\$(OUTDIR)\ncrt0_EW0.r30 >>    $(ODINCMD)\$(LNLIST)
       $(ECHO)\$(OUTDIR)\depend_m16c.r30 >>  $(ODINCMD)\$(LNLIST)
       $(ECHO)\$(OUTDIR)\flashdrvdev_ew0.r30 >>     $(ODINCMD)\$(LNLIST)
       $(ECHO)\$(OUTDIR)\main_m16c.r30 >>    $(ODINCMD)\$(LNLIST)
  $(OUTDIR)\$(TARGET): \
               $(ODINCMD)\$(LNLIST) \
               $(OUTDIR)\ncrt0_EW0.r30 \
               $(OUTDIR)\depend_m16c.r30 \
               $(OUTDIR)\flashdrvdev_ew0.r30 \
               $(OUTDIR)\main_m16c.r30
       $(LN) @$(ODINCMD)\$(LNLIST)
```

```
$(OUTDIR)\depend_m16c.r30: \
             .\depend_m16c.c \
             .\flashdevdrv.h \
             .\flashdevconf.h \
             .\flashm16c.h
     $(CC) $(MRCFLAGS) $(CFLAGS) depend_m16c.c
$(OUTDIR)\flashdrvdev_ew0.r30: \
             .\flashdrvdev_ew0.c \
             .\flashdevdrv.h \
             .\flashdevconf.h \
             .\flashm16c.h
     $(CC) $(MRCFLAGS) $(CFLAGS) flashdrvdev_ew0.c
$(OUTDIR)\main_m16c.r30: \
             .\main_m16c.c \
             .\flashdevdrv.h \
             .\flashdevconf.h
     $(CC) $(MRCFLAGS) $(CFLAGS) main_m16c.c
$(OUTDIR)\ncrt0_EW0.r30: \
             .\ncrt0_EW0.a30 \
             .\sect30_EW0.inc
     $(AS) $(MRAFLAGS) $(AFLAGS) ncrt0_EW0.a30
########################################################################
# End of makefile for TM V.3.20A
# COPYRIGHT(C) 1998(1998-2003)
# RENESAS TECHNOLOGY CORPORATION ALL RIGHTS RESERVED
# AND
# RENESAS SOLUTIONS CORPORATION ALL RIGHTS RESERVED
########################################################################
```

6. Reference

Hardware Manual
M16C/26 Group Hardware Manual Rev.0.90
M16C/28 Group Hardware Manual Rev.0.60
M16C/29 Group Hardware Manual Rev.1.00
The latest version can be downloaded from the Renesas Electronics website.

TECHNICAL UPDATE/TECHNICAL NEWS
The latest information can be downloaded from the Renesas Electronics website.

7. Website and Support

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

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

All trademarks and registered trademarks are the property of their respective owners.

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Mar.16.05	—	First edition issued
1.01	Dec.28.10	9	Figure 3-5 “Auto Erase Flowchart (Reference)” partially modified
		15	Figure 3-10 “Auto Erase Procedure” partially modified
		20	Figure 3-12 “Auto Erase Flowchart for the RAM Reduced Version partially modified
		31	Table 4-3 “StartEraseFlash()” parameter meaning “Beginning address of the flash memory to be erased” → “Highest address of the flash memory to be erased (even address)”
		80	9th line 0xF000 → 0xF7FE

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

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

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas 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 Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   - "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.
   - "High Quality": 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; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

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

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

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan, R.O.C.
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

© 2010 Renesas Electronics Corporation. All rights reserved.

Colophon 1.0