To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

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.

Flash Memory Download Program for the E10A-USB Emulator

Introduction

The E10A-USB emulator is equipped with the download function to the flash memory. The user needs to prepare a download program to use this function. In this Application Note, the sample program of the download program is given, together with customization and downloading procedure using the E10A-USB emulator.

Contents

1. Important Information

Before creating this document, the operation has been confirmed by Renesas. However, this does not mean that the Renesas is responsible for guaranteed operation.

Before reading this Application Note, please refer to the section of Download Function to the Flash Memory Area, in the Debugger Part of the E10A-USB emulator user's manual for each microcomputer family (HS0005KCU01HE).

In this Application Note, an example of downloading the user program to the external flash memory connected to the SH7727 SolutionEngine (manufactured by Hitachi ULSI Systems, Co., Ltd.) is shown. If the specifications of the user system is different from those of the SH7727 SolutionEngine, the write and erase modules supplied as the sample programs require customization. When using the sample programs, read this Application Note thoroughly and understand their contents before customization, and confirm their operation.

2. Description of Functions

By the sample program of this Application Note, the user program can be downloaded to the external flash memory area using the flash memory download function of the E10A-USB emulator.

The followings are required to download the user program:

- This sample program including:
  - Program for writing to the flash memory (called the write module hereafter)
  - Program for erasing the flash memory contents (called the erase module hereafter)
- RAM area in which the write and erase modules are downloaded and executed

For details on the flash memory download function of the E10A-USB emulator, please refer to the section of Download Function to the Flash Memory Area, in the Debugger Part of the E10A-USB emulator user's manual for each microcomputer family (HS0005KCU01HE).

This Application Note describes the contents to be modified for customization of the write and erase modules, which are used for downloading the user program to the external flash memory connected to the microcomputer, to suit the specifications of the user system (connection form with the target microcomputer).

3. Creating the Write and Erase Modules

The sample program (fmtool.src) described in section 7, Modifying the Sample Program, needs to be changed in accordance with the specifications of the user system. The parts that need to be changed are the addresses of the RAM on the user system used for the write and erase modules, and the addresses of the flash memory in which the user program is downloaded. Moreover, when the CUI command type flash memory is used, the start addresses of each sector block need to be changed.

3.1 Customization Details

The addresses of the RAM on the user system, where the write and erase modules are downloaded and executed, need to be changed. Check the following locations in the fmtool.src file, and refer to section 7, Modifying the Sample Program. Some of the addresses shown below are also used in section 5.3, Setting the [Configuration] Dialog Box.

```
O_FMErase       .equ    H'0C001000    Start address of the erase module
O_FMWrite       .equ    H'0C001100    Start address of the write module
FM_TOOL_STACK   .equ    H'0C002000    Start address of the stack area for write and erase modules
FM_TOP_ADDRESS .equ     H'00000000    Start address of the flash memory of the user system
```

When the CUI command type flash memory is used, the start addresses of each sector block should be listed to suit the specifications of the flash memory.

```
FM_ERASE_ADDRESS:       List of the start addresses of each sector block
       .data.l  H'00000000, H'00080000, H'00100000, H'00180000
       .data.l  H'00200000, H'00280000, H'00300000, H'00380000
       .data.l  H'00400000, H'00480000, H'00500000, H'00580000
       .data.l  H'00600000, H'00680000, H'00700000, H'00780000
                   :                          :
       .data.l  H'FFFFFFFF
```

4. Creating the Execution File

After modifying the fmtool.src file as required by referring to section 3, Creating the Write and Erase Modules, create the execution file by the following procedure.

Press the [All Build] button of the HEW to execute a build. This performs assembly and linkage and creates an execution file.

When the build is successfully completed, the execution file fmtool.mot is created under the folder with the same name as the build configuration (usually in folder “release”).

5. Procedure for Downloading the User Program

This section describes the procedure for downloading the user program to the flash memory using file fmtool.mot which is created in section 4, Creating the Execution File. An example using the SH7727 SolutionEngine (manufactured by Hitachi ULSI Systems, Co., Ltd.) is shown below.

5.1 Preparation for Download Environment

1. Connect the E10A-USB emulator connected to a host computer and the user system.
2. Activate the HEW, and open the user program workspace.
   The [CPU Select] dialog box shown in figure 1 appears.
3. Select the CPU used from the drop-down list box, and press the [OK] button.

Figure 1 [CPU Select] Dialog Box

4. The [Connecting] dialog box is displayed, indicating the connection state of the E10A-USB emulator.
   Then, the dialog box shown in figure 2 appears.

Figure 2 Dialog Box Requesting the Signal to be Input

5. Turn on the power of the user system.
6. Input the RESET signal from the user system, then click the [OK] button.
   When “Connected” is displayed in the [Output] window of the HEW, the emulator initiation is completed.

5.2 Setting the Bus State Controller

Set the bus state controller in the microcomputer so that the RAM and the flash memory can be accessed by the CPU. Display the HEW [IO] window. In the [IO] window shown in figure 3, set the I/O registers in accordance with the user system

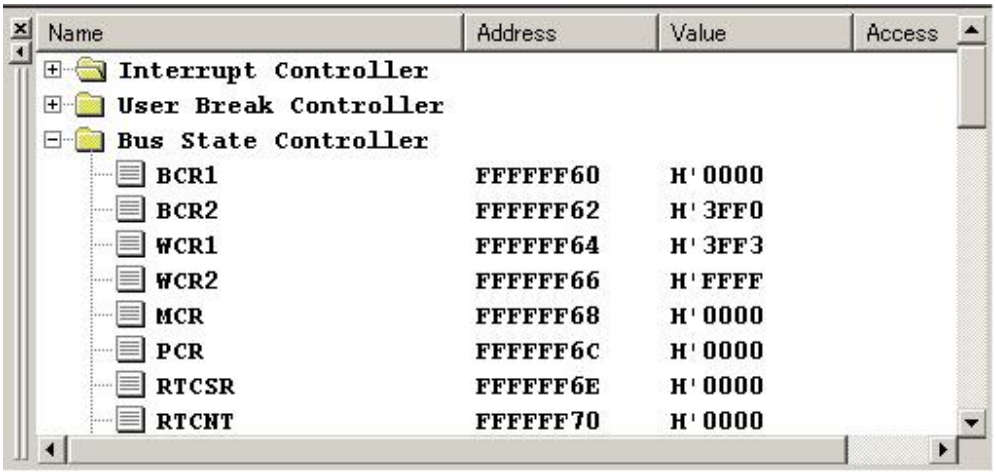

Figure 3 [IO] Window

The sample setting for the SH7727 SolutionEngine is given below:

BCR1	FFFFFF60	H'0008
BCR2	FFFFFF62	H'2FF3
MCR	FFFFFF68	H'512C
RTCNT	FFFFFF70	H'a500
RTCOR	FFFFFF72	H'a50C
RFCR	FFFFFF74	H'a400
RTCSR	FFFFFF6E	H'a518
SDMR	FFFFE880	H'0

5.3 Setting the [Configuration] Dialog Box

Set the [Loading flash memory] page in the [Configuration] dialog box shown in figure 4 to download the user program to the external flash memory using the E10A-USB emulator.

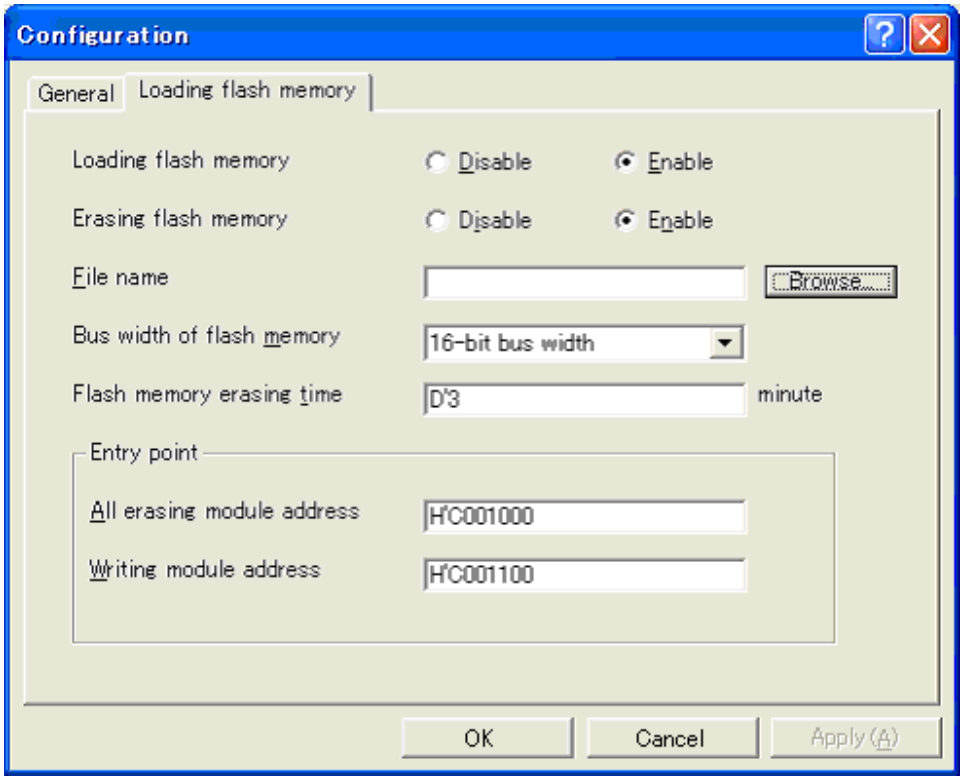

Figure 4 [Loading flash memory] Page in the [Configuration] Dialog Box

Set each item as follows:

- Loading flash memory: Select Enable.
- Erasing flash memory: Select Enable.
- File name: Specify fmtool.mot created in section 4, Creating the Execution File.
- Bus width of flash memory: Select the bus width for connecting the target microcomputer and the flash memory.
- All erasing module address: Enter the start address of the erase module.
- Writing module address: Enter the start address of the write module.

When all the items have been set, click the [OK] button.

5.4 Downloading the User Program

When the [Configuration] dialog box setting has been completed, download the user program which you want to download to the flash memory using the HEW program loading function shown in figure 5.

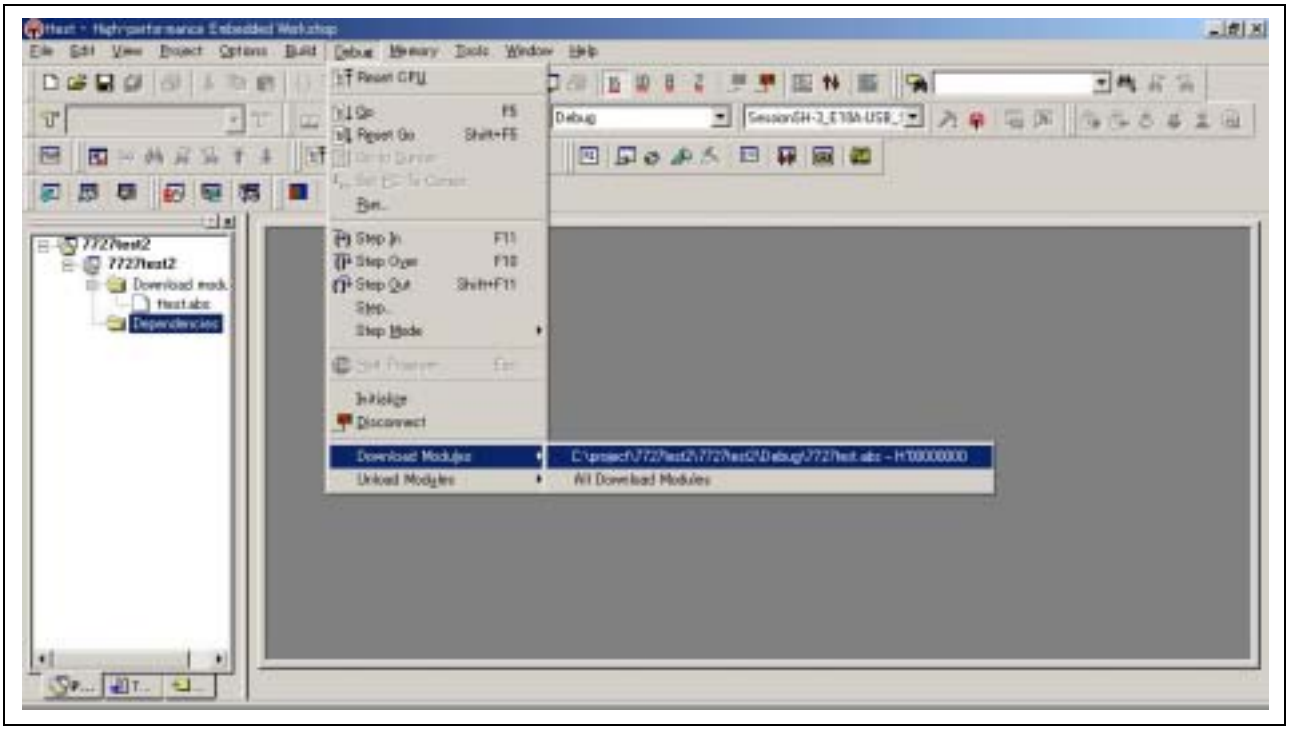

Figure 5 Downloading the User Program

5.5 Processing after Downloading the User Program

When a program is downloaded by the HEW program load function shown in figure 5, the operation according to the settings made in the [Configuration] dialog box in advance is performed, as shown in section 5.3, Setting the [Configuration] Dialog Box. Accordingly, before loading a program to a memory other than the external flash memory after the user program is downloaded to the external flash memory, set [Loading flash memory] to Disable in the [Configuration] dialog box as shown in figure 6.

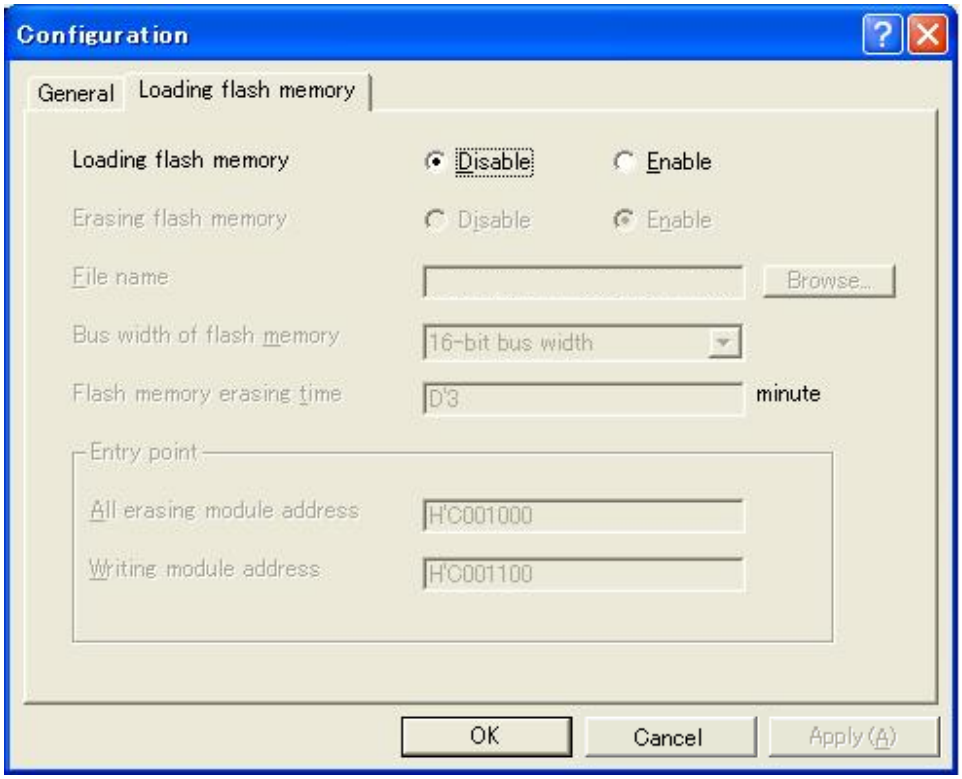

Figure 6 Loading Program to Memory Other Than External Flash Memory

6. Questions and Answers

1. About the operation procedure, first, connect to the user system, select [Option]-[Emulator]-[System], and set [Loading flash memory]in the Configuration screen, and download the user program. Is this correct?
   — Yes, that is correct.
2. Don’t you need to include the CPU initialization processing in the S-type files of the write and erase modules?
   — No, the CPU initialization processing is not necessary for the S-type files of the write and erase modules. You need to initialize the CPU before downloading the user program. Refer to the answer for question 5.
3. Is the required RAM area size that for the write and erase modules?
   — Yes, the required RAM area size is that for the write and erase modules. The user program is directly written to the flash memory from the E10A-USB emulator using the write and erase modules in the RAM area.
4. Is the transfer destination address of the write and erase modules in the S-type file?
   — If the address is allocated using .section in the write or erase module, the transfer destination address is the address set using the .section.
5. About the SDRAM setting, after connecting the user system, is the register manipulation display directly edited for setting the SDRAM? In Note 2 of the section of Downloading a Program, in the Debugger Part of the E10A-USB emulator user's manual for each microcomputer family (HS0005KCU01HE), describes that “When a program is downloaded to the external RAM, the bus state controller must be initially set in the area for downloading. Specifically, check that the initialization of SDRAM or setting of the bus width is appropriate for the user system.” How should I operate the E10A-USB emulator to initialize the SDRAM?
   — Use the [I/O register] window of the E10A-USB emulator to directly input the target register values to initialize the SDRAM. As is explained earlier, set the RAM address to which write and erase modules are downloaded to readable/writable, and select [Option]-[Emulator]-[System] to display the [Configuration] dialog box, set [Loading flash memory], then download the user program.

7. Modifying the Sample Program

The sample programs given below are for word-mode connection to the flash memory from Renesas and FUJITSU . The portions which modification is required for any manufacturer are (1) to (4) for the .EQU assembler directive. For the CUI command type flash memory, portion in (5) requires modification in addition. The modified parts are the same in byte or word mode. In byte mode, the bus width of 8 bits (when one flash memory unit is used), 16 bits (when two flash memory units are used), or 32 bits (when four flash memory units are used) is supported. Similarly, in word mode, the bus width of 16 bits (when one flash memory unit is used) or 32 bits (when 2 flash memory units are used) is supported.

Flash Memory from Renesas
File name: FMTOOL.src

```
;+=========================================================================+
;|                                                                         |
;| Flash memory tool program sample                                        |
;|    SuperH Family Flash memory load is supported                         |
;|    Copyright (C) 2004 Renesas Technology Corp. All rights reserved.     |
;|    Licensed Material of Renesas Technology Corp.                        |
;|    Erasing flash memory routine : O_FMErase                             |
;|    Writing flash memory routine : O_FMWrite                             |
;|    Stack pointer                : FM_TOOL_STACK                         |
;|    Flash memory top address     : FM_TOP_ADDRESS                        |
;|                                                                         |
;| Target flash memory : RENESAS [Word mode]                               |
;|                                                                         |
;+=========================================================================+
;
;+=============================================
;|       EQU
;+=============================================
O_FMErase       .equ  H'0C001000
O_FMWrite       .equ  H'0C001100
FM_TOOL_STACK   .equ  H'0C002000
FM_TOP_ADDRESS  .equ  H'00000000
;
FM_CMD_ERASE    .equ  H'00000020
FM_CMD_WRITE    .equ  H'00000040
FM_CMD_READ     .equ  H'ffffffff
FM_CMD_STSCLR   .equ  H'00000050
FM_CMD_CONFIRM  .equ  H'000000d0
;
FM_CHK_SR7      .equ  H'00000080
FM_CHK_SR5      .equ  H'00000020
FM_CHK_SR4      .equ  H'00000010
FM_CHK_SR3      .equ  H'00000008
;
FM_OK           .equ  H'0000
FM_BT           .equ  H'4254
;
TYPE_BYTE       .equ  H'4220
TYPE_WORD       .equ  H'5720
TYPE_LONG       .equ  H'4c20
;
;
;
   .align      4
;=========================================================================
;
; NAME = FM_TOOL_ERASE;
; FUNC = The routine of erasing all flash memories.
; NOTE = NEW;
; HIST = 2004.09.01;
; INPU = R4.L  : access size = 0x4220("B ");
```

(1) **Start address of the erase module**
The RAM area in the user system is used. Secure the size of H'100 bytes or more.

(2) **Start address of the write module**
The RAM area in the user system is used. Secure the size of H'800 bytes or more.

(3) **Start address of the stack area for write/erase module**
The RAM area in the user system is used. Secure the size of H'100 bytes or more.

(4) **Start address of the flash memory in the user system**
Set the start address of the flash memory area in the user system.

```
;                       = 0x5720("W ");
;                       = 0x4c20("L ");
; OUTP = R0.L  : status           = 0 (OK);
;                       !=0 (ERROR);
;
;=====================================================================
;  .org     O_FMErase
   .section fm_erase,CODE,LOCATE=O_FMErase
FM_TOOL_ERASE:
   mov.l    #FM_TOOL_STACK,r15     ; Set stack address to R15 register
   sts.l    pr,@-r15               ;
   mov.l    r3,@-r15               ;
;
   mov      r4,r0                  ;>>> Check an argument
   shlr8    r0                     ;
   cmp/eq   #H'42,r0               ; 'B' ? -> Illegal
   bt       FM_ERASE_BYTE          ;
   cmp/eq   #H'57,r0               ; 'W' ?
   bt       FM_ERASE_WORD          ;
   bf       FM_ERASE_LONG          ; 'L' jump
;
;
;>>> Call a module for the bus width of 8 bits --> Illegal call
FM_ERASE_BYTE:
   bra      FM_ERASE_END           ;
   nop                             ;
;
;
;>>> Call a module for the bus width of 16 bits
FM_ERASE_WORD:
   bsr      ClearAllStatusOfWord   ;>>> Clear the status of flash memory
   nop                             ;
;
   bsr      FmEraseWord            ;>>> Erase flash memory
   nop                             ; .. The routine has no parameter
;
   mov.l #CheckStatusWord,r3       ;
   jsr      @r3                    ;>>> Check the status of flash memory
   nop                             ; .. A status returns to R0
;
   bsr      ClearAllStatusOfWord   ;
   nop                             ;
;
   bra      FM_ERASE_END           ;
   nop                             ;
;
;
;>>> Call a module for the bus width of 32 bits
FM_ERASE_LONG:
   bsr      ClearAllStatusOfLong   ;>>> Clear the status of flash memory
   nop                             ;
;
   bsr      FmEraseLong            ;>>> Erase flash memory
```

```
   nop                              ; .. The routine has no parameter
;
   mov.l #CheckStatusLong,r3
   jsr      @r3                     ;>>> Check the status of flash memory
   nop                              ; .. A status returns to R0
;
   bsr      ClearAllStatusOfLong    ;
   nop                              ;
;
FM_ERASE_END:
;
   mov.l    @r15+,r3                ;
   lds.l    @r15+,pr                ;
   rts                              ;
   nop                              ;
;
;
;
;
;=====================================================================
;
; NAME = FM_TOOL_WRITE;
; FUNC = The routine of writing data in flash memory.
; NOTE = NEW;
; HIST = 2004.09.01
; INPU = R4.L  : write address;
;      R5.L : access size    = 0x4220("B ");
;                            = 0x5720("W ");
;                            = 0x4c20("L ");
;      R6.L : write data;
;      R7.L : verify flag    = 0 (non verify);
;                            = 1 (verify);
; OUTP = R0.L  : status              = 0 (OK);
;                            !=0 (ERROR);
;
;=====================================================================
;; .org     O_FMWrite
   .section fm_write,CODE,LOCATE=O_FMWrite
   .align      4
FM_TOOL_WRITE:
   mov.l    #FM_TOOL_STACK,r15      ; Set stack address to R15 register
   sts.l    pr,@-r15                ;
   mov.l    r1,@-r15                ;
   mov.l    r4,@-r15                ;
   mov.l    r5,@-r15                ;
   mov.l    r6,@-r15                ;
   mov.l    r7,@-r15                ;
;
   mov      r5,r0                   ;>>> Check an argument
   shlr8    r0                      ;
   cmp/eq   #H'42,r0                ; 'B' ? -> Illegal
   bt       FM_WRITE_BYTE           ;
   cmp/eq   #H'57,r0                ; 'W' ?
```

```
   bt       FM_WRITE_WORD          ;
   bf       FM_WRITE_LONG          ; 'L' jump
;
;
;>>> Call a module for the bus width of 8 bits --> Illegal call
FM_WRITE_BYTE:
   bra      FM_WRITE_END           ;
   nop                             ;
;
;
;>>> Call a module for the bus width of 16 bits
FM_WRITE_WORD:
   extu.w   r6,r6                  ;
;
   bsr      ClearAllStatusOfWord   ;>>> Clear the status of flash memory
   nop                             ;
;
   bsr      FmWriteWord            ;>>> Write a data to flash memory
   nop                             ; .. The routine has R4 and R6 (address and
                                   ; data) parameters
;
   bsr      CheckStatusWord        ;>>> Check the status of flash memory
   nop                             ; ... A status returns to R0
;
   cmp/eq   #0,r0                  ; if return == NG
   bf       FM_WRITE_WORD_E        ; then End
;
   cmp/pl   r7                     ; if verify flag is OFF
   bf       FM_WRITE_WORD_E        ; then end
;
   bsr      CheckVerifyWord        ;>>> Call a verify routine
   nop                             ;
;

FM_WRITE_WORD_E:
   bsr      ClearAllStatusOfWord   ;>>> Clear the status of flash memory
   nop                             ;
;
   bra      FM_WRITE_END           ;
   nop                             ;
;
;
;>>> Call a module for the bus width of 32 bits
FM_WRITE_LONG:
   bsr      ClearAllStatusOfLong   ;>>> Clear the status of flash memory
   nop                             ;
;
   bsr      FmWriteLong            ;>>> Write a data to flash memory
   nop                             ; .. The routine has R4 and R6 (address and
                                   ; data) parameters
;
   bsr      CheckStatusLong        ;>>> Check the status of flash memory
   nop                             ; ... A status returns to R0
```

```
;
   cmp/eq   #0,r0                ; if return == NG
   bf       FM_WRITE_LONG_E      ; then End
;
   cmp/pl   r7                   ; if verify flag is OFF
   bf       FM_WRITE_LONG_E      ; then end
;
   bsr      CheckVerifyLong      ;>>> Call a verify routine
   nop                           ;
;

FM_WRITE_LONG_E:
   bsr      ClearAllStatusOfLong ;>>> Clear the status of flash memory
   nop                           ;
;
;
;
FM_WRITE_END:
   mov.l    @r15+,r7             ;
   mov.l    @r15+,r6             ;
   mov.l    @r15+,r5             ;
   mov.l    @r15+,r4             ;
   mov.l    @r15+,r1             ;
   lds.l    @r15+,pr             ;
;
   rts                           ;
   nop                           ;
;
;
;
;=====================================================================
;
; NAME = FmEraseWord;
; FUNC = The routine of erasing flash memory(Bus width is 16 bits).
; NOTE = NEW;
; HIST = 2004.09.01
; INPU = none;
; OUTP = R0.L  = status;
;
;=====================================================================
FmEraseWord:
   mov.l    r1,@-r15             ;
   mov.l    r2,@-r15             ;
   mov.l    r3,@-r15             ;
   mov.l    r4,@-r15             ;
   mov.l    r8,@-r15             ;
;
   mov.l    #FM_TOP_ADDRESS,r3   ; R5 <- Top address of flash memory
   mov.l    #FM_ERASE_ADDRESS,r4 ; R4 <- Table of flash memory entry address
   mov.l    @r4,r8               ;
;
FmEraseWord_Main:
   add      r3,r8                ; R8 <- add flash top address (offset)
```

```
    mov.l    #FM_CMD_ERASE,r1       ;>>> Write the Erase command to flash memory
    mov.w    r1,@r8                 ;
;
    mov.l    #FM_CMD_CONFIRM,r1     ;>>> Write the Write confirm command to
                                    ; flash memory
    mov.w    r1,@r8                 ;
;
FmEraseWord_Loop:
    mov.w    @r8,r0                 ; Read status
    mov.l    #FM_CHK_SR7,r1         ;
    and      r1,r0                  ;
    cmp/eq   r1,r0                  ; if status.7 == 0
    bf       FmEraseWord_Loop       ; then loop
;
    mov.w    @r8,r0                 ; Set a return code to R8
    extu.w   r0,r0                  ;
;
    mov.l    #FM_CMD_READ,r1        ;>>> Write the Read command to flash memory
    mov.w    r1,@r8                 ;
;
    add      #4,r4                  ;>>> R8 <- Next entry address
    mov.l    @r4,r8                 ;
    mov      #-1,r1                 ;
    cmp/eq   r1,r8                  ;
    bf       FmEraseWord_Main       ;
;
FmEraseWord_End:
    mov.l    @r15+,r8               ;
    mov.l    @r15+,r4               ;
    mov.l    @r15+,r3               ;
    mov.l    @r15+,r2               ;
    mov.l    @r15+,r1               ;
    rts                             ;
    nop                             ;
;
;
;
;=====================================================================
;
; NAME = FmEraseLong;
; FUNC = The routine of erasing flash memory(Bus width is 32 bits).
; NOTE = NEW;
; HIST = 2004.09.01
; INPU = none;
; OUTP = R0.L  = status;
;
;=====================================================================
FmEraseLong:
    mov.l    r1,@-r15               ;
    mov.l    r2,@-r15               ;
    mov.l    r3,@-r15               ;
    mov.l    r4,@-r15               ;
    mov.l    r8,@-r15               ;
```

```
;
   mov.l    #FM_TOP_ADDRESS,r3     ; R5 <- Top address of flash memory
   mov.l    #FM_ERASE_ADDRESS,r4   ; R4 <- Table of flash memory entry address
   mov.l    @r4,r8                 ;
;
FmEraseLong_Main:
   add      r3,r8                  ; R8 <- add flash top address (offset)
   mov.l    #FM_CMD_ERASE,r1       ;>>> Write the Erase command to flash memory
   mov      r1,r2                  ;
   shll16   r2                     ;
   or       r2,r1                  ; R1 <- 32-bit status
   mov.l    r1,@r8                 ;
;
   mov.l #FM_CMD_CONFIRM,r1        ;>>> Write the Write confirm command to
                                   ; flash memory
   mov      r1,r2                  ;
   shll16   r2                     ;
   or       r2,r1                  ; R1 <- 32-bit status
   mov.l    r1,@r8                 ;
;
FmEraseLong_Loop:
   mov.l    @r8,r0                 ; Read status
   mov.l    #FM_CHK_SR7,r1         ;
   mov      r1,r2                  ;
   shll16   r2                     ;
   or       r2,r1                  ; R1 <- 32-bit status
   and      r1,r0                  ;
   cmp/eq   r1,r0                  ; if status.7 == 0
   bf       FmEraseLong_Loop       ; then loop
;
   mov.l    @r8,r0                 ; Set a return code to R8
;
   mov.l    #FM_CMD_READ,r1        ;>>> Write the Read command to flash memory
   mov      r1,r2                  ;
   shll16   r2                     ;
   or       r2,r1                  ; R1 <- 32-bit status
   mov.l    r1,@r8                 ;
;
   add      #4,r4                  ;>>> R8 <- Next entry address
   mov.l    @r4,r8                 ;
   mov      #-1,r1                 ;
   cmp/eq   r1,r8                  ;
   bf       FmEraseLong_Main       ;
;
FmEraseLong_End:
   mov.l    @r15+,r8               ;
   mov.l    @r15+,r4               ;
   mov.l    @r15+,r3               ;
   mov.l    @r15+,r2               ;
   mov.l    @r15+,r1               ;
   rts                             ;
   nop                             ;
;
```

```
;
;
;
;=====================================================================
;
; NAME = FmWriteWord
; FUNC = It is the routine of writing data to flash memory
; NOTE = NEW;
; HIST = 2004.09.01
; INPU = R4.L  = write address;
;      R6.L = write data;
; OUTP = R0.L  = status;
;
;=====================================================================
FmWriteWord:
   mov.l    r1,@-r15              ;
   mov.l    r2,@-r15              ;
;
   mov.l    #FM_CMD_WRITE,r1      ;>>> Write the Write command to flash memory
   mov.w    r1,@r4                ;
;
   mov.w    r6,@r4                ; Write data to flash memory
;
FmWriteWord_Check:
   mov.w    @r4,r0                ;>>> Check the status of flash memory
   extu.w   r0,r0                 ;
   mov.l    #FM_CHK_SR7,r1        ;
;
   and      r1,r0                 ;
   cmp/eq   r1,r0                 ; if status & 0x8080 != 0x8080
   bf       FmWriteWord_Check     ; then loop
;
   mov.w    @r4,r0                ; Set a return code to R4
   extu.w   r0,r0                 ;
;
   mov.l    #FM_CMD_READ,r1       ;>>> Write the Read command to flash memory
   mov.w    r1,@r4                ;
;
   mov.l    @r15+,r2              ;
   mov.l    @r15+,r1              ;
   rts                            ;
   nop                            ;
;
;
;
;
;=====================================================================
;
; NAME = CheckStatusWord
; FUNC = It is the routine of checking the status of flash memory
; NOTE = NEW;
; HIST = 2004.09.01
; INPU = R0.L  = check status value;
```

```
; OUTP = R0.L  = return code = 0 (OK);
;                     !=0 (NG);
;
;=====================================================================
CheckStatusWord:
   mov.l    r1,@-r15              ;
   mov.l    r2,@-r15              ;
;
   mov.l    #FM_CHK_SR3,r1        ;>>> Check SR.3, SR.5, and SR.4 of status
                                  ; register
   mov.l    #FM_CHK_SR5,r2        ;
   or       r2,r1                 ;
   mov.l    #FM_CHK_SR4,r2        ;
   or       r2,r1                 ;
;
   extu.w   r0,r0                 ;
   and      r1,r0                 ;
   cmp/eq   #0,r0                 ; if SR != 0
   bf       CSW_Error             ; then error
;
   bra      CSW_END               ;
   mov      #0,r0                 ; Set the OK code
CSW_Error:
   mov      #-1,r0                ; Set an error code
CSW_END:
   mov.l    @r15+,r2              ;
   mov.l    @r15+,r1              ;
   rts                            ;
   nop                            ;
;
;
;
;
;=====================================================================
;
; NAME = CheckVerifyWord
; FUNC = It is the routine of checking the written data
; NOTE = NEW;
; HIST = 2004.09.01
; INPU = R4.L  = check data address;
;      R6.L = write data
;
;=====================================================================
CheckVerifyWord:
   mov.l    r1,@-r15              ;
;
   extu.w   r6,r6                 ; Verify check
   mov.w    @r4,r1                ;
;
   extu.w   r1,r1                 ;
;
   cmp/eq   r6,r1                 ; if read == write data
   bt       CheckVeriW_OK         ; then OK
```

```
;
   mov.w    #FM_BT,r0             ; Set an error code
   bra      CheckVeriW_End        ;
   nop                            ;
;
CheckVeriW_OK:
   mov      #0,r0                 ; Set the OK code
CheckVeriW_End:
   mov.l    @r15+,r1              ;
   rts                            ;
   nop                            ;
;
;
;
;
;=====================================================================
;
; NAME = FmWriteLong
; FUNC = It is the routine of writing data to flash memory
; NOTE = NEW;
; HIST = 2004.09.01
; INPU = R4.L  = write address;
;      R6.L = write data;
; OUTP = R0.L  = status;
;
;=====================================================================
FmWriteLong:
   mov.l    r1,@-r15              ;
   mov.l    r2,@-r15              ;
;
   mov.l    #FM_CMD_WRITE,r1      ;>>> Write the Write command to flash memory
   mov      r1,r2                 ;
   shll16   r2                    ;
   or       r2,r1                 ; R1 <- 32-bit status
   mov.l    r1,@r4                ;
;
   mov.l    r6,@r4                ; Write data to flash memory
;
FmWriteLong_Check:
   mov.l    @r4,r0                ;>>> Check status of flash memory
   mov.l    #FM_CHK_SR7,r1        ;
   mov      r1,r2                 ;
   shll16   r2                    ;
   or       r2,r1                 ; R1 <- 32bit status
   and      r1,r0                 ;
   cmp/eq   r1,r0                 ; if status & 0x80808080 != 0x80808080
   bf       FmWriteLong_Check     ; then loop
;
   mov.l    @r4,r0                ; Set a return code to R4
;
   mov.l    #FM_CMD_READ,r1       ;>>> Write the Read command to flash memory
   mov      r1,r2                 ;
   shll16   r2                    ;
```

```
   or       r2,r1                ; R1 <- 32-bit status
   mov.l    r1,@r4               ;
;
   mov.l    @r15+,r2             ;
   mov.l    @r15+,r1             ;
   rts                           ;
   nop                           ;
;
;
;
;
;=====================================================================
;
; NAME = CheckStatusLong
; FUNC = It is the routine of checking the status of flash memory
; NOTE = NEW;
; HIST = 2004.09.01
; INPU = R0.L  = check status value;
; OUTP = R0.L  = return code = 0 (OK);
;                      !=0 (NG);
;
;=====================================================================
CheckStatusLong:
   mov.l    r1,@-r15             ;
   mov.l    r2,@-r15             ;
;
   mov.l    #FM_CHK_SR3,r1       ;>>> Check SR.3, SR.5, and SR.4 of status
                                 ; register
   mov.l    #FM_CHK_SR5,r2       ;
   or       r2,r1                ;
   mov.l    #FM_CHK_SR4,r2       ;
   or       r2,r1                ;
   mov      r1,r2                ;
   shll16   r2                   ;
   or       r2,r1                ; R1 <- 32-bit status

   and      r1,r0                ;
   cmp/eq   #0,r0                ; if SR != 0
   bf       CSL_Error            ; then error
;
   bra      CSL_END              ;
   mov      #0,r0                ; Set the OK code
CSL_Error:
   mov      #-1,r0               ; Set an error code
CSL_END:
   mov.l    @r15+,r2             ;
   mov.l    @r15+,r1             ;
   rts                           ;
   nop                           ;
;
;
;
;
```

```
;=====================================================================
;
; NAME = CheckVerifyLong
; FUNC = It is the routine of checking the written data
; NOTE = NEW;
; HIST = 2004.09.01
; INPU = R4.L  = check data address;
;      R6.L = write data
;
;=====================================================================
CheckVerifyLong:
   mov.l    r1,@-r15              ;
;
   mov.l    @r4,r1                ; Verify check
   cmp/eq   r6,r1                 ; if read == write data
   bt       CheckVeriL_OK         ; then OK
;
   mov.w    #FM_BT,r0             ; Set an error code
   bra      CheckVeriL_End        ;
   nop                            ;
;
CheckVeriL_OK:
   mov      #0,r0                 ; Set the OK code
CheckVeriL_End:
   mov.l    @r15+,r1              ;
   rts                            ;
   nop                            ;
;
;
;
;
;=====================================================================
;
; NAME = ClearStatus
; FUNC = It is the routine which clears the status of flash memory.
; NOTE = NEW;
; HIST = 2004.09.01
; INPU = R4.L  : access address;
;      R5.L : access size    = 0x4220("B ");
;                            = 0x5720("W ");
;                            = 0x4c20("L ");
;
;=====================================================================
ClearStatus:
   mov.l    r0,@-r15              ;
   mov.l    r1,@-r15              ;
   mov.l    r2,@-r15              ;
   mov.l    r3,@-r15              ;
;
   mov.l    #FM_TOP_ADDRESS,r3    ; R7 <- Top address of flash memory
   add      r3,r4                 ; R4 <- add flash top address (offset)
   mov.l    #FM_CMD_STSCLR,r1     ; R1 <- Status clear command
;
```

```
   mov      r5,r0                ;
   shlr8    r0                   ;
   cmp/eq   #H'42,r0             ; if access size == 'B ' -> Illegal
   bt       ClearByte            ; then ClearByte
;
   cmp/eq   #H'57,r0             ; if access size == 'W '
   bt       ClearWord            ; then ClearWord
;
   bf       ClearLong            ; then ClearLong
;
;>>> Write the status clear command to flash memory
ClearByte:                       ;>>> flash memory of 8-bit bus width ->
                                 ; Illegal
   bra      ClearEnd             ;
   nop                           ;
;
ClearWord:                       ;>>> flash memory of 16-bit bus width
   mov.w    r1,@r4               ;
   bra      ClearEnd             ;
   nop
;
ClearLong:                       ;>>> flash memory of 32-bit bus width
   mov      r1,r2                ;
   shll16   r2                   ;
   or       r2,r1                ; R1 <- 32-bit status clear command
   mov.l    r1,@r4               ;
;
ClearEnd:
   mov.l    @r15+,r3             ;
   mov.l    @r15+,r2             ;
   mov.l    @r15+,r1             ;
   mov.l    @r15+,r0             ;
   rts                           ;
   nop                           ;
;
;
;
;
;=====================================================================
;
; NAME = ClearAllStatusOfWord
; FUNC = It is the routine which clears all the status of flash memory.
; NOTE = NEW;
; HIST = 2004.09.01
; INPU = none;
;
;=====================================================================
ClearAllStatusOfWord:
   sts.l    pr,@-r15             ;
   mov.l    r0,@-r15             ;
   mov.l    r4,@-r15             ;
   mov.l    r5,@-r15             ;
   mov.l    r8,@-r15             ;
```

```
;
   mov.l    #FM_ERASE_ADDRESS,r8   ;>>> Clear all the status register
;
ClearAllStatusW_LOOP:
   mov.l    @r8,r4                 ;
   mov.l    #TYPE_WORD,r5          ;
   bsr      ClearStatus            ;
   nop                             ;
;
   add      #4,r8                  ;>>> Check finished
   mov.l    @r8,r4                 ;
   mov      #-1,r0                 ;
   cmp/eq   r0,r4                  ;
   bf       ClearAllStatusW_LOOP   ;
;
   mov.l    @r15+,r8               ;
   mov.l    @r15+,r5               ;
   mov.l    @r15+,r4               ;
   mov.l    @r15+,r0               ;
   lds.l    @r15+,pr               ;
   rts                             ;
   nop                             ;
;
;
;
;=====================================================================
;
; NAME = ClearAllStatusOfLong
; FUNC = It is the routine which clears all the status of flash memory.
; NOTE = NEW;
; HIST = 2004.09.01
; INPU = none;
;
;=====================================================================
ClearAllStatusOfLong:
   sts.l    pr,@-r15               ;
   mov.l    r0,@-r15               ;
   mov.l    r4,@-r15               ;
   mov.l    r5,@-r15               ;
   mov.l    r8,@-r15               ;
;
   mov.l    #FM_ERASE_ADDRESS,r8   ;
;
ClearAllStatusL_LOOP:
   mov.l    @r8,r4                 ;>>> Clear all the status register
   mov.l    #TYPE_LONG,r5          ;
   bsr      ClearStatus            ;
   nop
;
   add      #4,r8                  ;>>> Check finished
   mov.l    @r8,r4                 ;
   mov      #-1,r0                 ;
   cmp/eq   r0,r4                  ;
```

```
   bf       ClearAllStatusL_LOOP   ;
;
   mov.l    @r15+,r8               ;
   mov.l    @r15+,r5               ;
   mov.l    @r15+,r4               ;
   mov.l    @r15+,r0               ;
   lds.l    @r15+,pr               ;
   rts                             ;
   nop                             ;
;
;
;
;+=================================================================+
;|       FM DATA TABLE                                             |
;+=================================================================+
;  .align   4
;
FM_ERASE_ADDRESS:
   .data.l  H'00000000, H'00080000, H'00100000, H'00180000
   .data.l  H'00200000, H'00280000, H'00300000, H'00380000
   .data.l  H'00400000, H'00480000, H'00500000, H'00580000
   .data.l  H'00600000, H'00680000, H'00700000, H'00780000
   .data.l  H'FFFFFFFF
;
;
;
;
;
   .end
```

For CUI command type flash memory only:

(5)

Start addresses of each sector block

A list of start addresses of each sector block in the flash memory. Do not change H'FFFFFFFF at the end.

Flash Memory from FUJITSU
File name: FMTOOL.src

```
;+=========================================================================+
;|                                                                         |
;| Flash memory tool program sample                                        |
;|    SuperH Family Flash memory load is supported                         |
;|    Copyright (C) 2004 Renesas Technology Corp. All rights reserved.     |
;|    Licensed Material of Renesas Technology Corp.                        |
;|                                                                         |
;|    Erasing flash memory routine top address : O_FMErase                 |
;|    Writing flash memory routine top address : O_FMWrite                 |
;|    Stack pointer address                    : FM_TOOL_STACK             |
;|    Flash memory top address                 : FM_TOP_ADDRESS            |
;|                                                                         |
;| Target flash memory : FUJITSU [Word mode]                               |
;|                                                                         |
;+=========================================================================+
;
;+=============================================
;|       EQU
;+=============================================
;
O_FMErase       .equ  H'2c001000
O_FMWrite       .equ  H'2c001100
FM_TOOL_STACK   .equ  H'2c002000
FM_TOP_ADDRESS  .equ  H'20000000
;
FM_CMD_RESET    .equ  H'000000F0
FM_CHK_DQ7      .equ  H'00000080
FM_CHK_DQ5      .equ  H'00000020
;
FM_OK           .equ  H'0000
FM_BT           .equ  H'4254
;
TYPE_BYTE       .equ  H'4220
TYPE_WORD       .equ  H'5720
TYPE_LONG       .equ  H'4c20
;
;
;
   .align   4
;=========================================================================
;
; NAME = FM_TOOL_ERASE;
; FUNC = The routine of erasing all flash memories.
; NOTE = NEW;
; HIST = 2004.09.01;
; INPU = R4.L  : access size = 0x4220("B ");
;                            = 0x5720("W ");
;                            = 0x4C20("L ");
; OUTP = R0.L  : status      = 0 (OK);
;                       !=0 (ERROR);
;
```

(1) **Start address of the erase module**
The RAM area in the user system is used. Secure the size of H'100 bytes or more.

(2) **Start address of the write module**
The RAM area in the user system is used. Secure the size of H'800 bytes or more.

(3) **Start address of the stack area for write/erase module**
The RAM area in the user system is used. Secure the size of H'100 bytes or more.

(4) **Start address of the flash memory in the user system**
Set the start address of the flash memory area in the user system.

```
;======================================================================
;  .org     O_FMErase
   .section fm_erase,CODE,LOCATE=O_FMErase
;
FM_TOOL_ERASE:
   mov.l    #FM_TOOL_STACK,r15     ; Set stack address to R15 register
   sts.l    pr,@-r15               ;
   mov.l    r3,@-r15               ;
;
   mov      r4,r0                  ; Check an argument
   shlr8    r0                     ;
   cmp/eq   #H'42,r0               ; 'B' ? -> Illegal
   bt       FM_ERASE_BYTE          ;
   cmp/eq   #H'57,r0               ; 'W' ?
   bt       FM_ERASE_WORD          ;
   bf       FM_ERASE_LONG          ; 'L' jump
;
;
;>>> Call a module for the bus width of 8 bits  --> Illegal call
FM_ERASE_BYTE:
;
   bra      FM_ERASE_END           ;
   nop                             ;
;
;
;>>> Call a module for the bus width of 16 bits
FM_ERASE_WORD:
;
   bsr      ClearAllStatusWord     ; Clear the status of flash memory
   nop                             ;
;
   bsr      FmEraseWord            ; Erase flash memory
   nop                             ; .. The routine has no parameter
;
   bsr      ClearAllStatusWord     ; Clear the status of flash memory
   nop                             ;
;
   bra      FM_ERASE_END           ;
   nop                             ;
;
;
;>>> Call a module for the bus width of 32 bits
FM_ERASE_LONG:
;
   bsr      ClearAllStatusLong     ; Clear the status of flash memory
   nop                             ;
;
   bsr      FmEraseLong            ; Erase flash memory
   nop                             ; .. The routine has no parameter
;
   bsr      ClearAllStatusLong     ; Clear the status of flash memory
   nop                             ;
;
```

```
FM_ERASE_END:
   mov.l    @r15+,r3              ;
   lds.l    @r15+,pr              ;
   rts                            ;
   nop                            ;
;
;
;=====================================================================
;
; NAME = FM_TOOL_WRITE;
; FUNC = The routine of writing data in flash memory.
; NOTE = NEW;
; HIST = 2004.09.01;
; INPU = R4.L  : write address;
;      R5.L : access size    = 0x4220("B ");
;                            = 0x5720("W ");
;                            = 0x4c20("L ");
;      R6.L : write data;
;      R7.L : verify flag    = 0 (non verify);
;                            = 1 (verify);
; OUTP = R0.L  : status      = 0 (OK);
;                     !=0 (ERROR);
;
;=====================================================================
;  .org  O_FMWrite
   .section fm_write,CODE,LOCATE=O_FMWrite
;
FM_TOOL_WRITE:
   mov.l    #FM_TOOL_STACK,r15    ; Set stack address to R15 register
   sts.l    pr,@-r15              ;
   mov.l    r1,@-r15              ;
   mov.l    r4,@-r15              ;
   mov.l    r5,@-r15              ;
   mov.l    r6,@-r15              ;
   mov.l    r7,@-r15              ;
;
   mov      r5,r0                 ; Check an argument
   shlr8    r0                    ;
   cmp/eq   #H'42,r0              ; 'B' ? Illegal
   bt       FM_WRITE_BYTE         ;
   cmp/eq   #H'57,r0              ; 'W' ?
   bt       FM_WRITE_WORD         ;
   bf       FM_WRITE_LONG         ; 'L' jump
;
;
;>>> Call a module for the bus width of 8 bits  --> Illegal call
FM_WRITE_BYTE:
;
   mov.l    #1,r0                 ; Error
   bra      FM_WRITE_END          ;
   nop                            ;
;
;
```

```
;>>> Call a module for the bus width of 16 bits
FM_WRITE_WORD:
   bsr      ClearAllStatusWord     ; Clear the status of flash memory
   nop                             ;
;
   extu.w   r6,r6                  ;
;
   bsr      FmWriteWord            ; Write a data to flash memory
   nop                             ; .. The routine has R4 and R6 (address and
                                   ; data) parameters
;
   cmp/eq   #0,r0                  ; if return == NG
   bf       FM_WRITE_END_WORD      ; then End
;
   cmp/pl   r7                     ; if verify flag is OFF
   bf       FM_WRITE_END_WORD      ; then end
;
   bsr      CheckVerifyWord        ; Call a verify routine
   nop                             ;
;
FM_WRITE_END_WORD:
   bsr      ClearAllStatusWord     ; Clear the status of flash memory
   nop                             ;
;
   bra      FM_WRITE_END           ;
   nop                             ;
;
;
;>>> Call a module for the bus width of 32 bits
FM_WRITE_LONG:
   bsr      ClearAllStatusLong     ; Clear the status of flash memory
   nop                             ;
;
   bsr      FmWriteLong            ; Write a data to flash memory
   nop                             ; .. The routine has R4 and R6 (address and
                                   ; data) parameters
;
   cmp/eq   #0,r0                  ; if return == NG
   bf       FM_WRITE_END_LONG      ; then End
;
   cmp/pl   r7                     ; if verify flag is OFF
   bf       FM_WRITE_END_LONG      ; then end
;
   bsr      CheckVerifyLong        ; Call a verify routine
   nop                             ;
;
FM_WRITE_END_LONG:
   bsr      ClearAllStatusLong     ; Clear the status of flash memory
   nop                             ;
;
   bra      FM_WRITE_END           ;
   nop                             ;
;
```

```
FM_WRITE_END:

   mov.l    @r15+,r7              ;
   mov.l    @r15+,r6              ;
   mov.l    @r15+,r5              ;
   mov.l    @r15+,r4              ;
   mov.l    @r15+,r1              ;
   lds.l    @r15+,pr              ;
   rts                            ;
   nop                            ;
;
;
;
;=====================================================================
;
; NAME = FmEraseWord;
; FUNC = The routine of erasing flash memory(Bus width is 16 bits).
; NOTE = NEW;
; HIST = 2004.09.01;
; INPU = none;
; OUTP = R0.L  = status;
;
;=====================================================================
;
FmEraseWord:
   mov.l    r1,@-r15              ;
   mov.l    r2,@-r15              ;
   mov.l    r4,@-r15              ;
   mov.l    r5,@-r15              ;
   mov.l    r8,@-r15              ;
;
   mov.l    #FM_TOP_ADDRESS,r5    ; R5 <- Top address of flash memory
   mov.l    #FM_ERASE_DATA,r4     ; R4 <- Table of flash memory command data
   mov.l    #H'FFFFFFFF,r0        ; R0 <- Table end code
;
FmEraseWord_Main:
   mov.l    @r4+,r8               ; R8 <- command address
   add      r5,r8                 ; R8 <- add flash top address (offset)
   mov.l    @r4+,r1               ; R1 <- command data
   cmp/eq   r0,r1                 ; if Table end code
   bt       FmEraseWord_Loop      ; then next
;
   mov.w    r1,@r8                ; Write the Erase command to flash memory
   bra      FmEraseWord_Main      ; loop
   nop                            ;
;
FmEraseWord_Loop:
   mov.w    @r8,r0                ; Read status
   mov      r0,r2                 ;
   mov.l    #FM_CHK_DQ7,r1        ;
   and      r1,r0                 ;
   cmp/eq   r1,r0                 ; if status.DQ7 == 1 (0 = Busy / 1 = End)
   bt       FmEraseWord_Loop_Next ; then exit
```

```
    nop
;
    mov      r2,r0                  ;
    mov.l    #FM_CHK_DQ5,r1         ;
    and      r1,r0                  ;
    cmp/eq   r1,r0                  ; if status.DQ5 == 0 (0 = OK / 1 = Fail)
    bf       FmEraseWord_Loop       ; then loop
;
    bra      FmEraseWord_End        ; error end
    nop                             ;
;
FmEraseWord_Loop_Next:
    mov.l    #0,r0                  ; set OK
;
FmEraseWord_End:
    mov.l    @r15+,r8               ;
    mov.l    @r15+,r5               ;
    mov.l    @r15+,r4               ;
    mov.l    @r15+,r2               ;
    mov.l    @r15+,r1               ;
    rts                             ;
    nop                             ;
;
;
;
;
;=====================================================================
;
; NAME = FmEraseLong;
; FUNC = The routine of erasing flash memory(Bus width is 32 bits).
; NOTE = NEW;
; HIST = 2004.09.01;
; INPU = none;
; OUTP = R0.L  = status;
;
;=====================================================================
;
FmEraseLong:
    mov.l    r1,@-r15               ;
    mov.l    r2,@-r15               ;
    mov.l    r4,@-r15               ;
    mov.l    r5,@-r15               ;
    mov.l    r8,@-r15               ;
;
    mov.l    #FM_TOP_ADDRESS,r5     ; R5 <- Top address of flash memory
    mov.l    #FM_ERASE_DATA,r4      ; R4 <- Table of flash memory command data
    mov.l    #H'FFFFFFFF,r0         ; R0 <- Table end code
;
FmEraseLong_Main:
    mov.l    @r4+,r8                ; R8 <- command address
    shll     r8                     ; adjust address
    add      r5,r8                  ; R8 <- add flash top address (offset)
    mov.l    @r4+,r1                ; R1 <- command data
```

```
    cmp/eq    r0,r1                 ; if Table end code
    bt        FmEraseLong_Loop      ; then next
    mov       r1,r2                 ;
    shll16    r2                    ; command for upper word
    or        r2,r1                 ; R1 <- 32-bit command
;
    mov.l     r1,@r8                ; Write the Erase command to flash memory
    bra       FmEraseLong_Main      ; loop
    nop                             ;
;
FmEraseLong_Loop:
    mov.l     @r8,r0                ; Read status
    mov       r0,r2                 ;
    mov.l     #FM_CHK_DQ7,r1        ;
    shll16    r1                    ;
    mov.l     #FM_CHK_DQ7,r5        ;
    or        r5,r1                 ; R1 <- 32-bit status
    and       r1,r0                 ;
    cmp/eq    r1,r0                 ; if status.DQ7 == 1 (0 = Busy / 1 = End)
    bt        FmEraseLong_Loop_Next ; then exit
;
    mov       r2,r0
    and       r5,r0                 ;
    cmp/eq    #H'0,r0               ; if status.DQ7 == 1 (0 = Busy / 1 = End)
    bf        FmEraseLong_Loop_u    ; then next

    mov       r2,r0                 ;
    mov.l     #FM_CHK_DQ5,r1        ;
    and       r1,r0                 ;
    cmp/eq    r1,r0                 ; if status.DQ5 == 0 (0 = OK / 1 = Fail)
    bf        FmEraseLong_Loop_u    ; then next
;
    bra       FmEraseLong_End       ; error end
    nop                             ;
;
FmEraseLong_Loop_u:
    mov       r2,r0                 ;
    shlr16    r0                    ;
    and       r5,r0                 ;
    cmp/eq    #H'0,r0               ; if status.DQ7 == 1 (0 = Busy / 1 = End)
    bf        FmEraseLong_Loop      ; then loop
;
    mov       r2,r0                 ;
    mov.l     #FM_CHK_DQ5,r1        ;
    shll16    r1                    ;
    and       r1,r0                 ;
    cmp/eq    r1,r0                 ; if status.DQ5 == 0 (0 = OK / 1 = Fail)
    bf        FmEraseLong_Loop      ; then loop
;
    bra       FmEraseLong_End       ; error end
    nop                             ;
;
FmEraseLong_Loop_Next:
```

```
   mov.l    #0,r0                ; set OK
;
FmEraseLong_End:
   mov.l    @r15+,r8             ;
   mov.l    @r15+,r5             ;
   mov.l    @r15+,r4             ;
   mov.l    @r15+,r2             ;
   mov.l    @r15+,r1             ;
   rts                           ;
   nop                           ;
;
;
;
;=====================================================================
;
; NAME = FmWriteWord
; FUNC = The routine of writing data to flash memory
; NOTE = NEW;
; HIST = 2004.09.01;
; INPU = R4.L  = write address;
;   R6.L = write data;
; OUTP = R0.L  = status;
;
;=====================================================================
FmWriteWord:
   mov.l    r1,@-r15             ;
   mov.l    r2,@-r15             ;
   mov.l    r5,@-r15             ;
   mov.l    r6,@-r15             ;
   mov.l    r7,@-r15             ;
   mov.l    r8,@-r15             ;
;
   mov.l    #FM_TOP_ADDRESS,r7   ; R7 <- Top address of flash memory
   mov.l    #FM_WRITE_DATA,r5    ; R5 <- Table of flash memory command data
   mov.l    #H'FFFFFFFF,r0       ; R0 <- Table end code
;
FmWriteWord_Main:
   mov.l    @r5+,r8              ; R8 <- command address
   add      r7,r8                ; R8 <- add flash top address (offset)
   mov.l    @r5+,r1              ; R1 <- command data
   cmp/eq   r0,r1                ; if Table end code
   bt       FmWriteWord_Write    ; then next
;
   mov.w    r1,@r8               ; Write the Write command to flash memory
   bra      FmWriteWord_Main     ; loop
   nop                           ;
;
FmWriteWord_Write:
   mov.w    r6,@r4               ; Write data to flash memory
   mov.l    #FM_CHK_DQ7,r7       ;
   and      r6,r7                ;
;
FmWriteWord_Loop:
```

```
    mov.w    @r4,r0                ; Read memory
    mov      r0,r2                 ;
    mov.l    #FM_CHK_DQ7,r1        ;
    and      r1,r0                 ;
    cmp/eq   r7,r0                 ; if end the write
    bt       FmWriteWord_Loop_Next ; then exit
    nop
;
    mov      r2,r0                 ;
    mov.l    #FM_CHK_DQ5,r1        ;
    and      r1,r0                 ;
    cmp/eq   r1,r0                 ; if status.DQ5 == 0 (0 = OK / 1 = Fail)
    bf       FmWriteWord_Loop      ; then loop
;
    bra      FmWriteWord_End       ; error end
    nop                            ;
;
FmWriteWord_Loop_Next:
    mov.l    #0,r0                 ;
;
FmWriteWord_End:
    mov.l    @r15+,r8              ;
    mov.l    @r15+,r7              ;
    mov.l    @r15+,r6              ;
    mov.l    @r15+,r5              ;
    mov.l    @r15+,r2              ;
    mov.l    @r15+,r1              ;
    rts                            ;
    nop                            ;
;
;
;
;=====================================================================
;
; NAME = FmWriteLong
; FUNC = The routine of writing data to flash memory
; NOTE = NEW;
; HIST = 2004.09.01;
; INPU = R4.L  = write address;
;   R6.L = write data;
; OUTP = R0.L  = status;
;
;=====================================================================
FmWriteLong:
    mov.l    r1,@-r15              ;
    mov.l    r2,@-r15              ;
    mov.l    r5,@-r15              ;
    mov.l    r6,@-r15              ;
    mov.l    r7,@-r15              ;
    mov.l    r8,@-r15              ;
;
    mov.l    #FM_TOP_ADDRESS,r7    ; R7 <- Top address of flash memory
    mov.l    #FM_WRITE_DATA,r5     ; R5 <- Table of flash memory command data
```

```
    mov.l    #H'FFFFFFFF,r0         ; R0 <- Table end code
;
FmWriteLong_Main:
    mov.l    @r5+,r8                ; R8 <- command address
    shll     r8                     ; adjust address (address signal connecting
                                    ; A2)
    add      r7,r8                  ; R8 <- add flash top address (offset)
    mov.l    @r5+,r1                ; R1 <- command data
    mov      r1,r2                  ;
    shll16   r2                     ;
    or       r2,r1                  ;
    cmp/eq   r0,r1                  ; if Table end code
    bt       FmWriteLong_Write      ; then next
;
    mov.l    r1,@r8                 ; Write the Write command to flash memory
    bra      FmWriteLong_Main       ; loop
    nop                             ;
;
FmWriteLong_Write:
    mov.l    r6,@r4                 ; Write data to flash memory
    mov.l    #FM_CHK_DQ7,r7         ;
    shll16   r7                     ;
    mov.l    #FM_CHK_DQ7,r5         ;
    or       r5,r7                  ;
    and      r6,r7                  ;
;
FmWriteLong_Loop:
    mov.l    @r4,r0                 ; Read memory
    mov      r0,r2                  ;
    mov.l    #FM_CHK_DQ7,r1         ;
    shll16   r1                     ;
    mov.l    #FM_CHK_DQ7,r5         ;
    or       r5,r1                  ;
    and      r1,r0                  ;
    cmp/eq   r7,r0                  ; if status.DQ7 == 1 (0 = Busy / 1 = End)
    bt       FmWriteLong_Loop_Next  ; then exit
;
    mov      r7,r1                  ;
    and      r5,r1                  ;
    mov      r2,r0                  ;
    and      r5,r0                  ;
    cmp/eq   r1,r0                  ; if status.DQ7 == write data
    bt       FmWriteLong_Loop_u     ; then next
;
    mov      r2,r0                  ;
    mov.l    #FM_CHK_DQ5,r1         ;
    and      r1,r0                  ;
    cmp/eq   r1,r0                  ; if status.DQ5 == 0 (0 = OK / 1 = Fail)
    bf       FmWriteLong_Loop_u     ; then next
;
    bra      FmWriteLong_End        ; error end
    nop                             ;
;
```

```
FmWriteLong_Loop_u:
   shll16   r5                  ;
   mov      r7,r1               ;
   and      r5,r1               ;
   mov      r2,r0               ;
   and      r5,r0               ;
   cmp/eq   r1,r0               ; if status.DQ7 == write data
   bt       FmWriteLong_Loop    ; then loop
   nop                          ;
;
   mov      r2,r0               ;
   mov.l    #FM_CHK_DQ5,r1      ;
   shll16   r1                  ;
   and      r1,r0               ;
   cmp/eq   r1,r0               ; if status.DQ5 == 0 (0 = OK / 1 = Fail)
   bf       FmWriteLong_Loop    ; then loop
;
   bra      FmWriteLong_End     ; error end
   nop                          ;
;
FmWriteLong_Loop_Next:
   mov.l    #0,r0               ;
;
FmWriteLong_End:
   mov.l    @r15+,r8            ;
   mov.l    @r15+,r7            ;
   mov.l    @r15+,r6            ;
   mov.l    @r15+,r5            ;
   mov.l    @r15+,r2            ;
   mov.l    @r15+,r1            ;
   rts                          ;
   nop                          ;
;
;
;
;=====================================================================
;
; NAME = CheckVerifyWord
; FUNC = The routine of checking the written data
; NOTE = NEW;
; HIST = 2004.09.01;
; INPU = R4.L  = check data address;
;      R6.L = write data
;
;=====================================================================
CheckVerifyWord:
   mov.l    r1,@-r15            ;
;
   extu.w   r6,r6               ; Verify check
   mov.w    @r4,r1              ;
   extu.w   r1,r1               ;
   cmp/eq   r6,r1               ; if read == write data
   bt       CheckVeriW_OK       ; then OK
```

```
;
   mov.w    #FM_BT,r0             ; Set an error code
   bra      CheckVeriW_End        ;
   nop                            ;
;
CheckVeriW_OK:
   mov      #0,r0                 ; Set the OK code
CheckVeriW_End:
   mov.l    @r15+,r1              ;
   rts                            ;
   nop                            ;
;
;
;
;
;=====================================================================
;
; NAME = CheckVerifyLong
; FUNC = The routine of checking the written data
; NOTE = NEW;
; HIST = 2004.09.01;
; INPU = R4.L  = check data address;
;      R6.L = write data
;
;=====================================================================
CheckVerifyLong:
   mov.l    r1,@-r15              ;
;
   mov.l    @r4,r1                ; Verify check
   cmp/eq   r6,r1                 ; if read == write data
   bt       CheckVeriL_OK         ; then OK
   mov.w    #FM_BT,r0             ; Set an error code
   bra      CheckVeriL_End        ;
   nop                            ;
;
CheckVeriL_OK:
   mov      #0,r0                 ; Set the OK code
CheckVeriL_End:
   mov.l    @r15+,r1              ;
   rts                            ;
   nop                            ;
;
;
;
;
;=====================================================================
;
; NAME = ClearAllStatusWord
; FUNC = The routine which clears the status of flash memory.
; NOTE = NEW;
; HIST = 2004.09.01;
; INPU = none;
;
```

```
;======================================================================
ClearAllStatusWord:
   mov.l    r0,@-r15              ;
   mov.l    r1,@-r15              ;
;
   mov.l    #FM_CMD_RESET,r1      ; R1 <- Status of clear command
   mov.l    #FM_TOP_ADDRESS,r0    ; R0 <- Flash memory top address
   mov.w    r1,@r0                ;
   nop                            ;
   nop                            ;
   nop                            ;
   nop                            ;
   nop                            ;
   nop                            ;
;
   mov.l    @r15+,r1              ;
   mov.l    @r15+,r0              ;
   rts                            ;
   nop                            ;
;
;
;
;======================================================================
;
; NAME = ClearAllStatusLong
; FUNC = The routine which clears the status of flash memory.
; NOTE = NEW;
; HIST = 2004.09.01;
; INPU = none;
;
;======================================================================
ClearAllStatusLong:
   mov.l    r0,@-r15              ;
   mov.l    r1,@-r15              ;
;
   mov.l    #FM_CMD_RESET,r0      ;
   mov.l    #FM_CMD_RESET,r1      ;
   shll16   r1                    ;
   or       r0,r1                 ; R1 <- 32-bit status clear command
   mov.l    #FM_TOP_ADDRESS,r0    ; R0 <- Flash memory top address
   mov.l    r1,@r0                ;
   nop                            ;
   nop                            ;
   nop                            ;
   nop                            ;
   nop                            ;
   nop                            ;
;
   mov.l    @r15+,r1              ;
   mov.l    @r15+,r0              ;
   rts                            ;
   nop                            ;
;
```

```
;
;
;+=====================================================================+
;|       FM DATA TABLE                                                 |
;+=====================================================================+
   .align   4
;
;          [ADDRESS(W)],[DATA(W)]
FM_ERASE_DATA:
   .data.l  H'00000AAA,H'000000AA  ; 1
   .data.l  H'00000554,H'00000055  ; 2
   .data.l  H'00000AAA,H'00000080  ; 3
   .data.l  H'00000AAA,H'000000AA  ; 4
   .data.l  H'00000554,H'00000055  ; 5
   .data.l  H'00000AAA,H'00000010  ; 6
   .data.l  H'00000000,H'FFFFFFFF  ; .. END ID.
;
;
;         [ADDRESS(W)],[DATA(W)]
FM_WRITE_DATA:
   .data.l  H'00000AAA,H'000000AA  ; 1
   .data.l  H'00000554,H'00000055  ; 2
   .data.l  H'00000AAA,H'000000A0  ; 3
   .data.l  H'00000000,H'FFFFFFFF  ; .. END ID.
;
;
   .end
```

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep.23.04	—	First edition issued

Keep safety first in your circuit designs!

1. Renesas Technology Corp. puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
   Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corp. product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corp. or a third party.
2. Renesas Technology Corp. assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors.
   Renesas Technology Corp. assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corp. by various means, including the Renesas Technology Corp. Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corp. or an authorized Renesas Technology Corp. product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corp. is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corp. for further details on these materials or the products contained therein.