SH7262/SH7264 Group

Using a Custom FMTOOL with E10A-USB

To Program Serial SPI Connected Flash Memory

R01AN0321EG0100
Rev.1.00
Jan 12, 2011

Introduction

This application note describes the use of a custom version of an FMTOOL program with HEW (High-Performance Embedded Workshop) to program serial flash memory connected to the RSPI peripheral of a Renesas SH7264 microcontroller.

Target Device

SH7262/7264 MCU. (In this document, SH7262/SH7264 are referred to as SH7264).

Contents

Specification

This application activates channel 0 of the SH7264 RSPI interface connected to serial flash memory and erases and programs this memory under the control of HEW (High-Performance Embedded Workshop). This gives HEW the ability to download a program or download data for storage in the serial flash memory.

Modules Used

- Renesas Serial Peripheral Interface (RSPI)

Applicable Conditions

MCU	SH7262/SH7264
	Internal clock: 144MHz
Operating Frequencies	Bus clock: 72MHz
	Peripheral clock: 36MHz
IDE	High-Performance Embedded Workshop v4.08.00.011
	Renesas Technology SuperH RISC Engine Family
C/C++ Compiler	SH Series C/C++ Compiler Package v9.03.02.000
Compiler Options	-cpu=sh2afpu -include="$(PROJDIR)\..\src","$(PROJDIR)\..\inc","$(PROJDIR)" -object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -show=source,include,expansion,tab=4 -optimize=0 -gbr=auto -chgincpath -errorpath -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 -nologo

Related Application Notes

- SH7262/SH7264 Group Interfacing Serial Flash Memory Using the Renesas Serial Peripheral Interface
- SH7262/SH7264 Group Boot from the Serial Flash Memory
- Customization of Flash Memory Download Program for E10A-USB Emulator
- Guide to Using the Download Program to Flash Memory for the E10A-USB Emulator

2 Overview of FMTool

When developing with Renesas ROMless microcontrollers such as the SH7264 the code which the micro will execute typically resides in externally connected NOR flash memory. This causes a dilemma during the development phase as the debugger, in this case High-Performance Embedded Workshop (HEW) and the E10A-USB emulator, must have a way of programming code into this external flash memory. However, HEW has no way of knowing what flash memory is connected to the microcontroller's bus, what the access width is, what the speed is and what the erase and programming algorithms are. To solve this problem and give HEW the ability to erase and program externally connected flash memory a developer can supply HEW with a binary file (an s-record file) which HEW downloads into RAM on the device and calls functions in this binary file to erase the flash memory and to program a byte, word or long word at a particular address. The binary file which is downloaded by HEW and provides this functionality is called FMTool.

Figure 2-1 shows how an FMTool file is used with HEW and conventional NOR parallel flash memory.

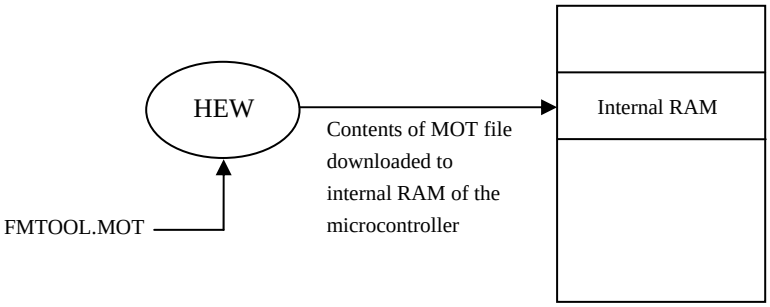

Fig. 2-1a FMTOOL.MOT file downloaded into Internal RAM of Microcontroller

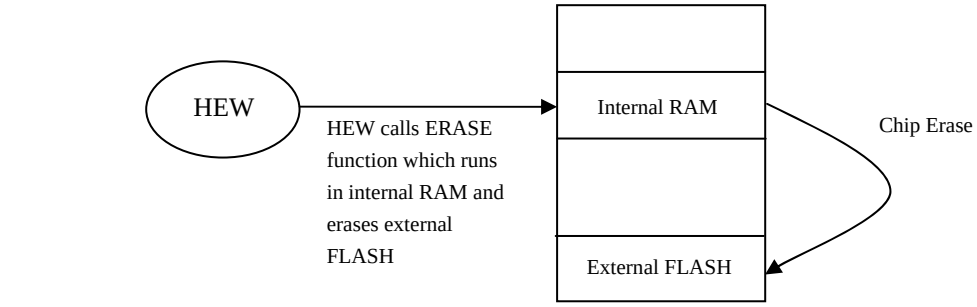

Fig. 2-1b Chip erase

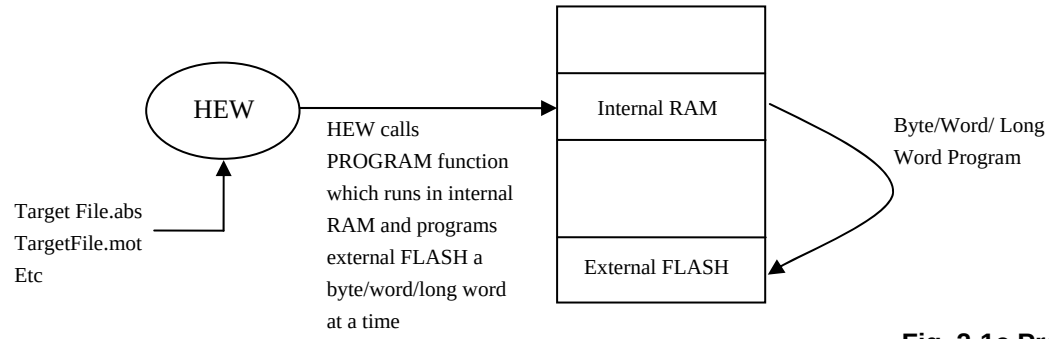

Fig. 2-1c Programming

When using a development board supplied by Renesas for a ROMless microcontroller FMTool will invariably be supplied with the board in binary, and sometimes source code form. In such cases there is nothing more to do than to ensure HEW is correctly configured to use the FMTool binary and often even this is unnecessary when using a sample project supplied with the hardware.

The FMTool binary file is added to HEW using the Configuration dialog reached from the menu Setup->Emulator->System and then selecting the “Loading flash memory” tab. Figure 2-2 shows this dialog for the SH7264 and this RSPI version of FMTool.

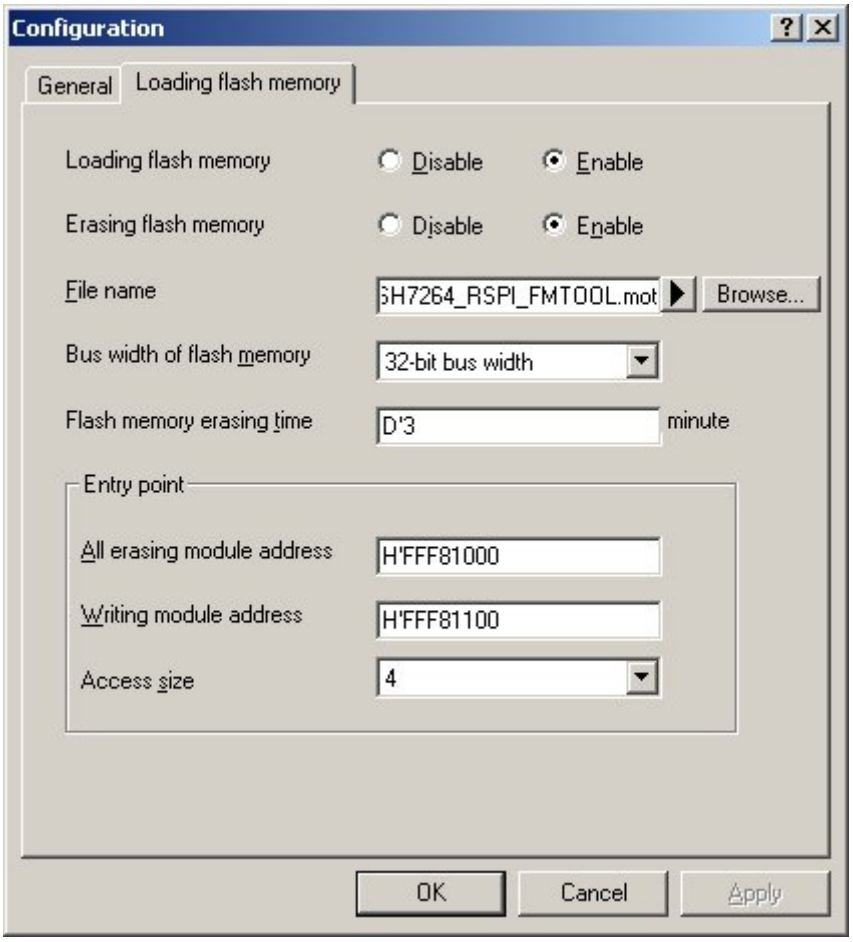

Fig. 2-2 – HEW Settings for FMTool Binary File

3 FMTool HEW Project

The FMTool HEW project contains the source code for the FMTool binary which erases and programs the serial flash memory device found on the RSK2+SH7264 board. The serial flash device is a Numonyx M25PX64 with the features shown in table 1.

Item	Description
Manufacturer	Numonyx
Part No.	M25PX65
Interface	SPI – SH7264 RSPI Channel 0
Speed	75MHz Maximum – SH7264 RSPI operating at 18MHz
Memory Size	64Mbit (8MByte)
Memory Arrangement	Uniform 64kB sectors
Characteristics	100,000 write cycles per sector. 20 years data retention

Table 1 – Serial Flash Information

The RSPI initialisation code and serial flash memory routines have been taken from application note “SH7262/SH7264 Group Interfacing Serial Flash Memory Using the Renesas Serial Peripheral Interface”. This application note should be referred to for further details on this code. One modification to this source code has been made relating to the number of flash memory sectors. This change is in the file SERIAL_FLASH.H and shown below.

```
From:
#define SF_NUM_OF_SECTOR   32
To:
#define SF_NUM_OF_SECTOR   128
```

The HEW workspace project “SH7264_RSPI_FMTOOL” consists of the following source files.

Item	Description
fmtool.src	Assembler code providing the entry points for the chip erase and program routines
cpg.c	Contains source code to initialise the SH7264 clock pulse generator Iclk = 144MHz, Bclk = 72MHz, Pclk = 36MHz
serial_flash.c	RSPI and serial flash routines including chip erase and program
iodefine.h	IO header file
serial_flash.h	Header file for the serial flash and RSPI routines
typedefine.h	Type definitions

3.1 Chip Erase

The figure below shows the flow chart of the chip erase function.

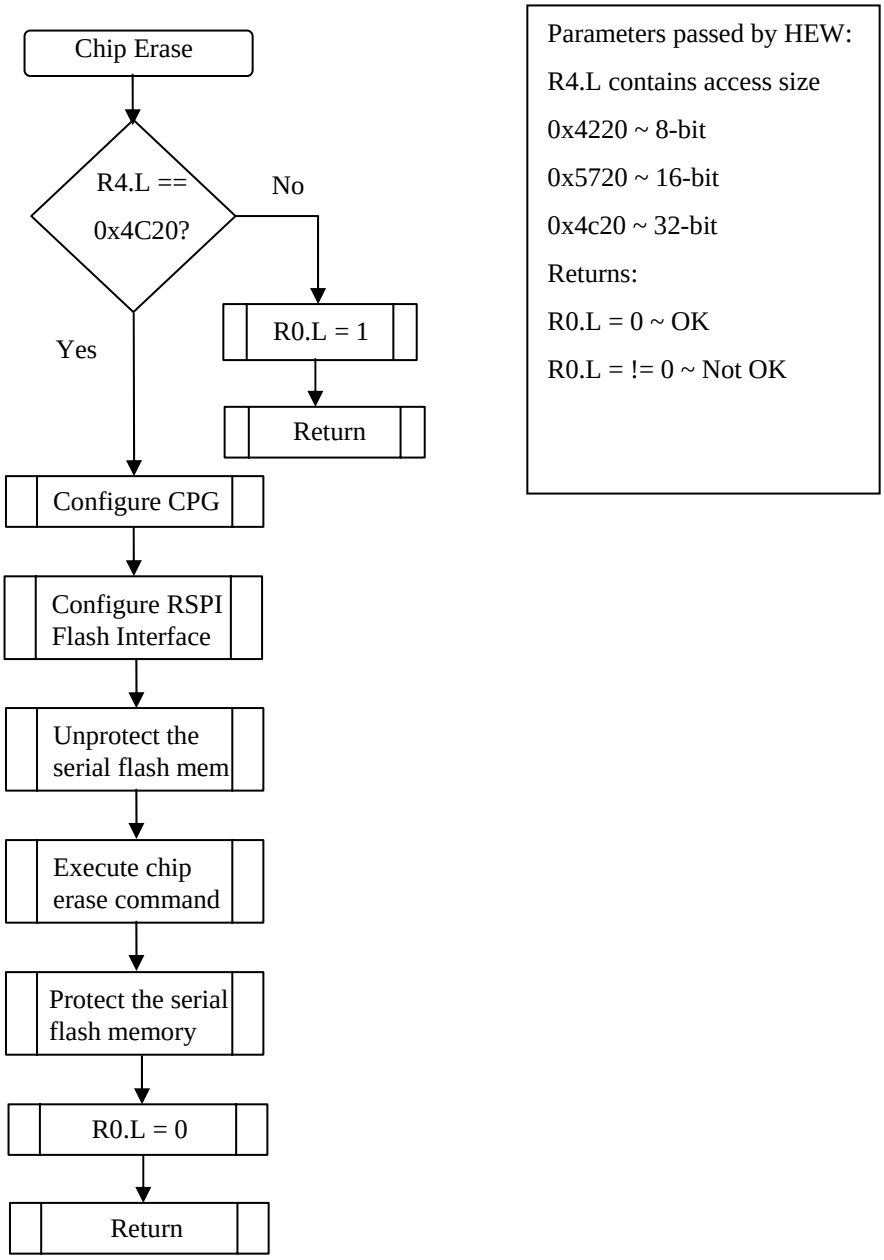

Figure 3-1 Chip Erase Flow Chart

3.2 Program Long Word

Only 32-bit long word programming is supported by this implementation of FMTool. The figure below shows the flow chart of the serial flash program function.

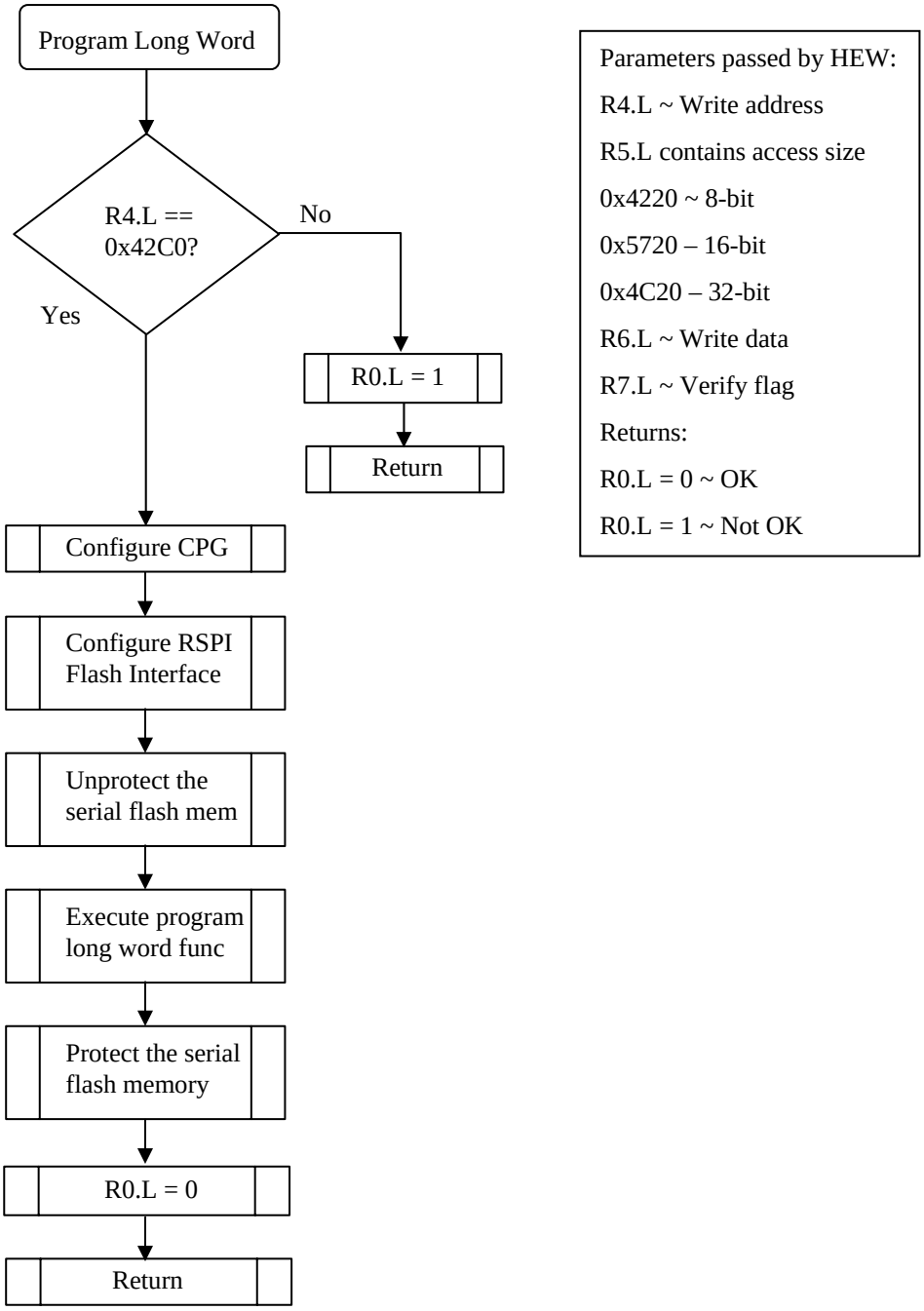

Figure 3-2 32-bit Long Word Programming Flow Chart

The serial flash memory starts at address zero. If the program code or data is linked to other addresses then the s-record file containing this code and/or data must have the addresses adjusted. This can be achieved using the open source and freely available command line application SREC_CAT.EXE. Details on using this utility can be found in section 4 of this application note.

3-2 HEW Configuration

The FMTool binary file is added to HEW using the configuration dialog reached from the menu Setup->Emulator->System and then selecting the “Loading flash memory” tab. Figure 2-2 shows this dialog for the SH7264 and this RSPI version of FMTool.

4 Serial Bootloader HEW Workspace Using FMTool

The Renesas SH7264 microcontroller supports various boot modes including boot modes 1 and 3 where the device can be booted from serial RSPI connected flash memory. This implementation of FMTool can be used to load the bootloader and application into the serial flash memory. The apps note “SH7262/SH7264 Group Boot from the Serial Flash Memory” describes in detail how to use this boot method and how to create the necessary boot and application code. The boot mode apps note is supplied with three HEW projects. For the purposes of this FMTool apps note two of these HEW projects are used. They are “sh7264_sflash_loader_prog” and “sh7264_sflash_app”. Both projects are supplied with this application note. Both projects have been slightly modified from their original form. The modifications are described below.

The target hardware for both projects is the RSK2+SH7264.

4-1 sh7264_sflash_loader_prog

SERIAL_FLASH.H modified to reflect the size of the serial flash being used.

This loader project creates the 8kB (maximum) binary file which is copied by the SH7264's built in boot mode bootloader into internal RAM and then executed. This program is linked to the RAM addresses H'FFF80000 to H'FFF81FFF where it is copied to automatically by the microcontroller as part of boot mode. In the serial flash memory the loader program is stored at addresses H'00000000 to H'00001FFF. Therefore, the output s-record binary file must have its addresses modified for correct storage in the serial flash. This modification is performed by the command line utility "SREC_CAT.EXE". This executable is supplied with the "sh7264_sflash_loader_prog" project. The source for this application with links to where the latest binaries are located can be found at the website at the below URL.

http://srecord.sourceforge.net/

A custom build phase has been added to HEW to invoke "SREC_CAT.EXE" to offset the addresses in the s-record file. The custom build settings (Build->Offset_SRecord_Addresses) in HEW for the Debug build are shown below.

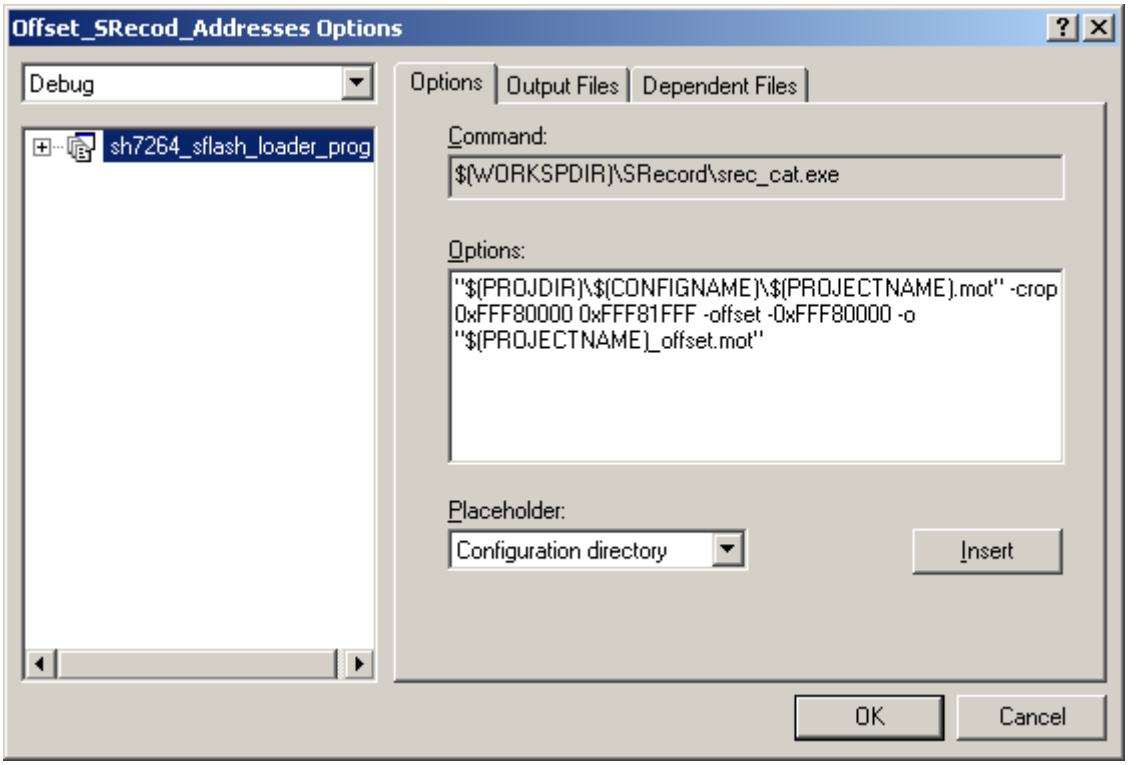

Figure 4-1 HEW Custom Build Phase Settings

Please note that using paths with spaces may cause issues with calling "SREC_CAT.EXE" from within HEW.

4-2 sh7264_sflash_app

This project creates the user's application which is copied from the serial flash by the loader program into internal RAM and then executed. This application is linked for the cache-enabled on-chip large capacity RAM space starting at address H'1C000000. In the serial flash memory it is stored above the 8kB space reserved for the loader program at address starting H'00002000 and so the user program s-record file must have its addresses modified by SREC_CAT.EXE. This utility is called as a custom HEW build phase with the configuration for the Debug build shown in the screenshot in figure 4-2.

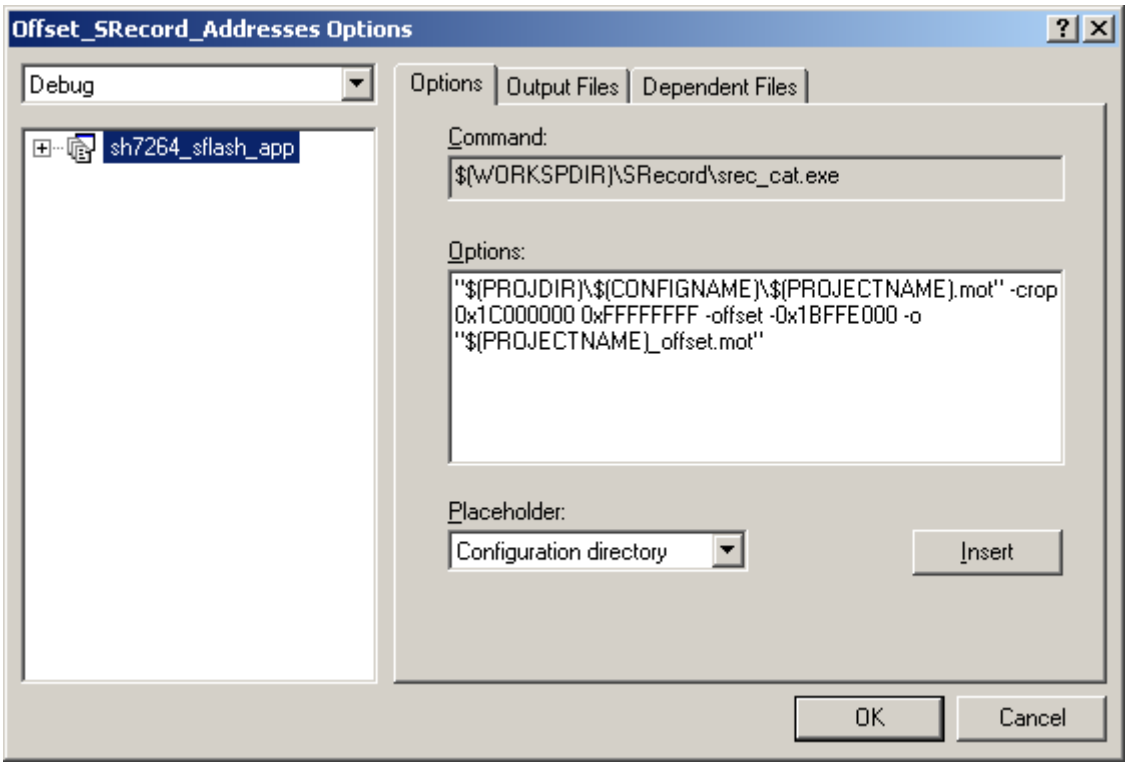

Figure 4-2 HEW Custom Build Phase Settings

The serial flash memory must contain both the loader application and the user application. The two s-record files are concatenated by SREC_CAT.EXE using another HEW custom build phase with the configuration shown in figure 4-3.

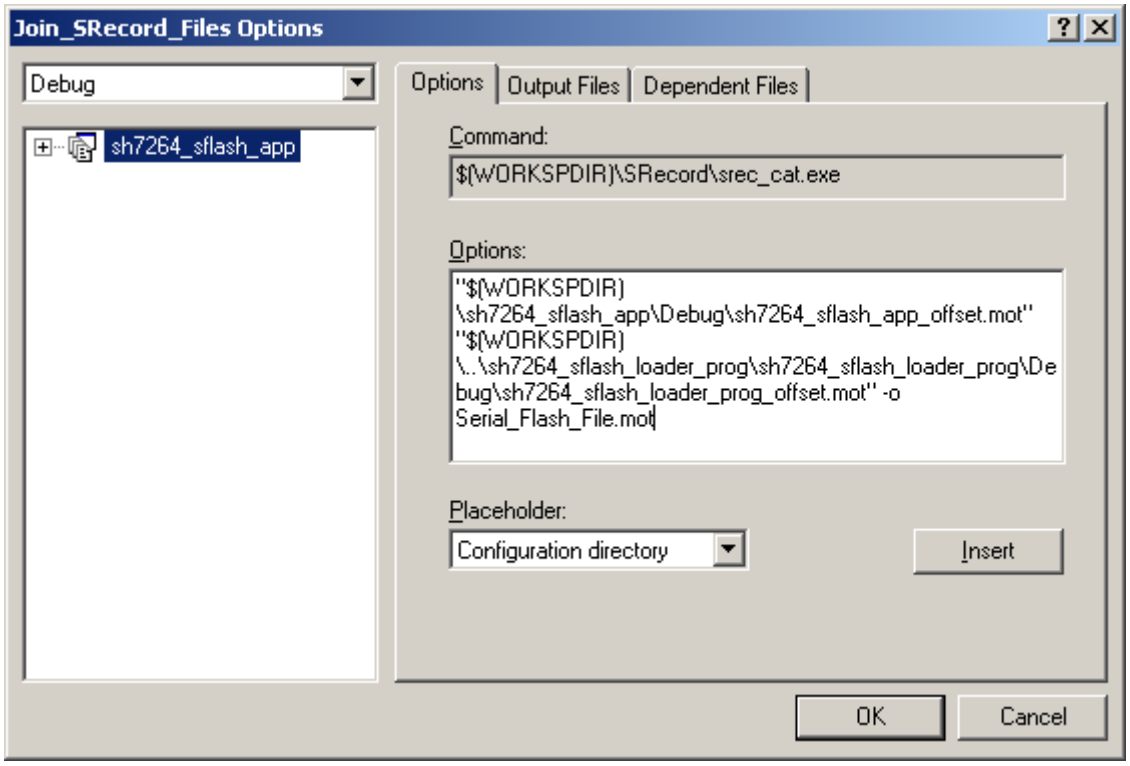

Figure 4-3 HEW Custom Build Phase Settings

Figure 4-4 shows the memory map of the final output from the sh7264_sflash_app project for both the serial flash memory and the internal RAM of the device after it has booted from the serial flash memory and loaded the application into the RAM.

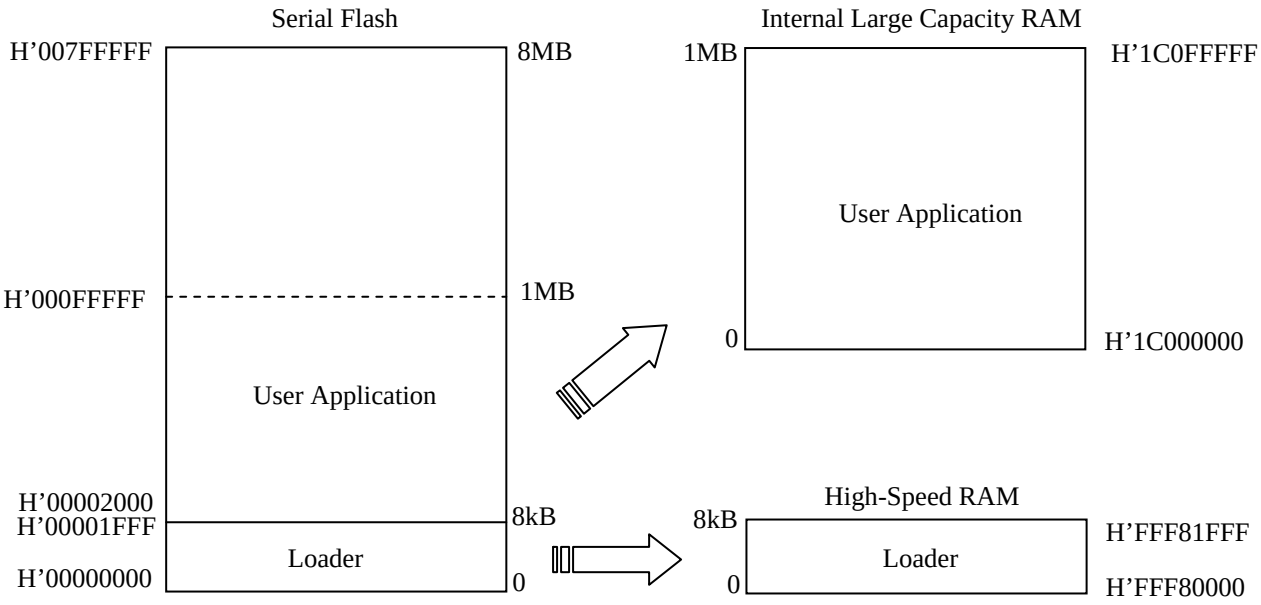

Figure 4-4 Memory Map Serial Flash and Internal RAM of sh7264_sflash_app Project after Boot from Serial Flash and Application Load into RAM

Please be aware that the capacity of the serial flash memory on the RSK board far exceeds the 1MB internal RAM of the SH7264. Therefore, care should be taken to ensure that the application fits within the available memory.

It is the resultant SERIAL_FLASH.MOT s-record file which is downloaded into the serial flash memory using this RSI version of FMTool. Please note that it may be necessary to manually modify the path to the SH7264_RSPI_FMTOOL.mot FMTool file as shown in figure 2-2.

To test booting from the serial flash the device should be reset with the mode pins set for boot mode 1 (MD_BOOT0=1, MD_BOOT1=0) or boot mode 3 (MD_BOOT0=1, MD_BOOT1=1). Connect a serial cable between CN1 on the RSK and an available COM port on a PC and start a terminal emulation program with the settings 57600 baud 8-N-1. When resetting the device the string “==== Serial Flash Boot Done. ====” should be output to the terminal window.

Using the serial boot mode of the SH7264 microcontroller family can eliminate the need for a parallel external flash memory and so help facilitate a lower cost final system.

5 Program Listing

5.1 FMTool Program

5.1.1 “fmtool.src”

```
;+=====================================================================
==+
;|                                                          |
;| Flash memory tool program sample                           |
;|    SuperH Family Flash memory load is supported              |
;|    Copyright (C) 2010 Renesas Electronics Europe Ltd. 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 : SPI Serial Flash [Longword mode]
  |
;|                                                          |
;+=====================================================================
==+
;

                .IMPORT _io_set_cpg
                .IMPORT _sf_init_serial_flash
                .IMPORT _sf_unprotect_flash
                .IMPORT _sf_chip_erase
                .IMPORT _sf_protect_flash
                .IMPORT _sf_program_longword
                .IMPORT _sf_program_byte

;+=================================================================+
;|       EQU                                                       |
;+=================================================================+
;
O_FMErase   .equ  H'FFF81000
O_FMWrite   .equ  H'FFF81100
FM_TOOL_STACK  .equ  H'FFF90000
;
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);
```

```
;
;=====================================================================
; .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.lpr,@-r15           ;
     mov.lr1,@-r15           ;
     mov.lr2,@-r15           ;
     mov.lr3,@-r15           ;
     mov.lr4,@-r15           ;
     mov.lr5,@-r15           ;
     mov.lr6,@-r15           ;
     mov.lr7,@-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' ? -> Illegal
  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:
;
     mov     #1, r0                 ; Error

     bra     FM_ERASE_END      ;
  nop                        ;
;
;
;>>> Call a module for the bus width of 8 bits  --> Illegal call
FM_ERASE_WORD:
;
  mov     #1, r0                  ; Error

     bra     FM_ERASE_END      ;
  nop                        ;
;
;>>> Call a module for the bus width of 32 bits
FM_ERASE_LONG:
;
     mov.l   #_io_set_cpg, r3
     jsr     @r3
     nop

     mov.l   #_sf_init_serial_flash, r3
     jsr     @r3
     nop

     mov.l   #_sf_unprotect_flash, r3
     jsr     @r3
     nop

     mov.l   #_sf_chip_erase, r3
```

```
    jsr     @r3
    nop

    mov.l   #_sf_protect_flash, r3
    jsr     @r3
    nop

    mov     #0, r0

FM_ERASE_END:
  mov.l  @r15+,r7          ;
    mov.l@r15+,r6          ;
    mov.l@r15+,r5          ;
  mov.l  @r15+,r4          ;
  mov.l  @r15+,r3          ;
    mov.l@r15+,r2          ;
    mov.l@r15+,r1          ;
  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.lpr,@-r15          ;
    mov.lr1,@-r15          ;
    mov.lr2,@-r15          ;
    mov.lr3,@-r15          ;
    mov.lr4,@-r15          ;
    mov.lr5,@-r15          ;
    mov.lr6,@-r15          ;
    mov.lr7,@-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' ? -> Illegal
  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  --> Illegal call
FM_WRITE_WORD:
  mov.l   #1,r0             ; Error
  bra     FM_WRITE_END      ;
  nop                       ;;
;
;>>> Call a module for the bus width of 32 bits
FM_WRITE_LONG:

    mov.l   #_io_set_cpg, r3
    jsr     @r3
    nop

    mov.l   #_sf_init_serial_flash, r3
    jsr     @r3
    nop

    mov.l   #_sf_unprotect_flash, r3
    jsr     @r3
    nop

    add.l   #4, r15
    mov.l@r15+,r6           ;
    add.l   #4, r15
    mov.l@r15+,r4           ;
    mov     r6, r5
    add.l   #-16, r15
    mov.l   #_sf_program_longword, r3
    jsr     @r3
    nop

    mov.l   #_sf_protect_flash, r3
    jsr     @r3
    nop

    mov.l#0,r0              ; OK

FM_WRITE_END:

  mov.l   @r15+,r7          ;
    mov.l@r15+,r6           ;
    mov.l@r15+,r5           ;
  mov.l   @r15+,r4          ;
  mov.l   @r15+,r3          ;
    mov.l@r15+,r2           ;
```

```
    mov.l@r15+,r1          ;
  lds.l  @r15+,pr          ;
  rts                      ;
  nop                      ;
;
  .end
```

5.1.2 “serial_flash.c”

```
/*""FILE COMMENT""************ Technical reference data
************************
*
*       System Name : SH7264 Sample Program
*       File Name   : serial_flash.c
*       Abstract    : Interfacing Serial Flash Memory Using the Renesas Serial
*                     Peripheral Interface
*       Version     : 1.00.00
*       Device      : SH7262/SH7264
*       Tool-Chain  : High-performance Embedded Workshop (Version 4.07.00.007)
*                   : C/C++ compiler package for the SuperH RISC engine family
*                   :                                 (Ver.9.03.02.000).
*       OS          : None
*       H/W Platform: RSK2+SH7264
*       Disclaimer  :
*
*       The information described here may contain technical inaccuracies or
*       typographical errors. Renesas Electronics
*       assume no responsibility for any damage, liability, or other loss
rising
*       from these inaccuracies or errors.
*
*       Copyright (C) 2009 Renesas Electronics Corp. All Rights Reserved
*
*       History     : June,2010 Ver.1.00.00
*""FILE COMMENT
END""*********************************************************/
#include <stdio.h>
#include <machine.h>
#include "iodefine.h"
#include "serial_flash.h"

/* ==== Macro definition ==== */
#define SFLASHCMD_CHIP_ERASE  0xc7
#define SFLASHCMD_SECTOR_ERASE0xd8
#define SFLASHCMD_BYTE_PROGRAM0x02
#define SFLASHCMD_BYTE_READ   0x0B
#define SFLASHCMD_BYTE_READ_LOW  0x03
#define SFLASHCMD_WRITE_ENABLE0x06
#define SFLASHCMD_WRITE_DISABLE  0x04
#define SFLASHCMD_READ_STATUS 0x05
#define SFLASHCMD_WRITE_STATUS0x01
#define UNPROTECT_WR_STATUS   0x00
#define PROTECT_WR_STATUS     0x3C

/* ==== Function prototype declaration ==== */
/*** Local function ***/
static void write_enable(void);
static void write_disable(void);
static void busy_wait(void);
static unsigned char read_status(void);
static void write_status(unsigned char status);
static void io_init_rspi(void);
static void io_cmd_exe(unsigned char *ope, int ope_sz, unsigned char *data,
int data_sz);
```

```
static void io_cmd_exe_rdmode(unsigned char *ope, int ope_sz, unsigned char
*rd, int rd_sz);
static void io_wait_tx_end(void);


void sf_protect_ctrl(enum sf_req req);

/* ==== Variable definition ==== */

/*""FUNC
COMMENT""************************************************************
 * ID          :
 * Outline     : Serial flash unprotect
 *-------------------------------------------------------------------------
--
 * Include     :
 *-------------------------------------------------------------------------
--
 * Declaration : void sf_unprotect_flash( void );
 *-------------------------------------------------------------------------
--
 * Function    : Wrapper call to sf_protect_ctrl( SF_REQ_UNPROTECT )
 *             : from asm code
  *-------------------------------------------------------------------------
---
 * Argument    : void
 *-------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""***********************************************************/
void sf_unprotect_flash( void )
{
    sf_protect_ctrl( SF_REQ_UNPROTECT );
}

/*""FUNC
COMMENT""************************************************************
 * ID          :
 * Outline     : Serial flash protect
 *-------------------------------------------------------------------------
--
 * Include     :
 *-------------------------------------------------------------------------
--
 * Declaration : void sf_protect_flash( void );
 *-------------------------------------------------------------------------
--
 * Function    : Wrapper call to sf_protect_ctrl( SF_REQ_PROTECT )
 *             : from asm code
  *-------------------------------------------------------------------------
---
 * Argument    : void
 *-------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""***********************************************************/
```

```
void sf_protect_flash( void )
{
    sf_protect_ctrl( SF_REQ_PROTECT );
}

/*""FUNC
COMMENT""*************************************************************
 * ID          :
 * Outline     : Serial flash protect
 *-------------------------------------------------------------------------
--
 * Include     :
 *-------------------------------------------------------------------------
--
 * Declaration : void sf_protect_flash( void );
 *-------------------------------------------------------------------------
--
 * Function    : Program data at address in serial flash memory
 *-------------------------------------------------------------------------
--
 * Argument    : unsigned long addr ; I : Address to start programming
 *             : unsigned long data ; I : Data to program into flash
 *-------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""*************************************************************/
void sf_program_longword( unsigned long addr, unsigned long data )
{
    unsigned char dat[4];

    dat[0] = (data >> 24) & 0xff;
    dat[1] = (data >> 16) & 0xff;
  dat[2] = (data >>  8) & 0xff;
  dat[3] =  data        & 0xff;

    sf_byte_program( addr, dat, 4 );
}

/*""FUNC
COMMENT""*************************************************************
 * ID          :
 * Outline     : Serial flash memory initialization
 *-------------------------------------------------------------------------
--
 * Include     :
 *-------------------------------------------------------------------------
--
 * Declaration : void sf_init_serial_flash(void);
 *-------------------------------------------------------------------------
--
 * Function    : Initializes serial flash memory for being accessed.
 *             : Initializes the channel 0 of the Renesas Serial Peripheral
 *             : Interface (RSPI).
 *-------------------------------------------------------------------------
--
 * Argument    : void
```

```
 *-------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""*********************************************************/
void sf_init_serial_flash(void)
{
  /* ==== Initializes the RSPI0 ==== */
  io_init_rspi();
}
/*""FUNC
COMMENT""*****************************************************************
 * ID          :
 * Outline     : Protect/unprotect operation
 *-------------------------------------------------------------------------
--
 * Include     :
 *-------------------------------------------------------------------------
--
 * Declaration : void sf_protect_ctrl(enum sf_req req);
 *-------------------------------------------------------------------------
--
 * Function    : Protects or unprotects serial flash memory.
 *             : Use the argument req to specify. Default setting and
unprotecting
 *             : method depends on the specifications of the serial flash
memory.
 *-------------------------------------------------------------------------
--
 * Argument    : enum sf_req req ; I : SF_REQ_UNPROTECT -> Write-enable all
sectors
 *             :                       SF_REQ_PROTECT   -> Write-protect all
sectors
 *-------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""*********************************************************/
void sf_protect_ctrl(enum sf_req req)
{
  if( req == SF_REQ_UNPROTECT ){
    write_status( UNPROTECT_WR_STATUS);    /* Protect total area */
  }
  else{
    write_status( PROTECT_WR_STATUS );     /* Unprotect total area */
  }
}
/*""FUNC
COMMENT""*****************************************************************
 * ID          :
 * Outline     : Chip erase
 *-------------------------------------------------------------------------
--
 * Include     :
 *-------------------------------------------------------------------------
--
 * Declaration : void sf_chip_erase(void);
```

```
 *------------------------------------------------------------------------
--
 * Function    : Erases all bits in serial flash memory.
 *             : Before erasing or programming, issue the Write Enable command.
 *             : After erasing or programming, make sure to check the status
of
 *             : serial flash memory if the busy status is reset.
 *------------------------------------------------------------------------
--
 * Argument    : void
 *------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""*********************************************************/
void sf_chip_erase(void)
{
  unsigned char cmd[1];
  cmd[0] = SFLASHCMD_CHIP_ERASE;

  write_enable();
  io_cmd_exe(cmd, 1, NULL, 0);
  busy_wait();
}

/*""FUNC
COMMENT""*****************************************************************
 * ID          :
 * Outline     : Sector erase
 *------------------------------------------------------------------------
--
 * Include     :
 *------------------------------------------------------------------------
--
 * Declaration : void sf_sector_erase(int sector_no);
 *------------------------------------------------------------------------
--
 * Function    : Erases the specified sector in serial flash memory.
 *             : Before erasing or programming, issue the Write Enable command.
 *             : After erasing or programming, make sure to check the status
of
 *             : serial flash memory if the busy status is reset.
 *------------------------------------------------------------------------
--
 * Argument    : int sector_no ; I : Sector number
 *------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""*********************************************************/
void sf_sector_erase(int sector_no)
{
  unsigned char cmd[4];
  unsigned long addr = sector_no * SF_SECTOR_SIZE;

  cmd[0] = SFLASHCMD_SECTOR_ERASE;
  cmd[1] = (addr >> 16) & 0xff;
  cmd[2] = (addr >>  8) & 0xff;
```

```
  cmd[3] =  addr        & 0xff;

  write_enable();
  io_cmd_exe(cmd, 4, NULL, 0);
  busy_wait();
}

/*""FUNC
COMMENT""*************************************************************
 * ID          :
 * Outline     : Program data
 *-------------------------------------------------------------------------
--
 * Include     :
 *-------------------------------------------------------------------------
--
 * Declaration : void sf_byte_program(unsigned long addr, unsigned char *buf,
int size);
 *-------------------------------------------------------------------------
--
 * Function    : Programs the specified data in serial flash memory.
 *             : Before erasing or programming, issue the Write Enable command.
 *             : After erasing or programming, make sure to check the status
of
 *             : serial flash memory if the busy status is reset.
 *             : The maximum write data size depends on the type of the device.
 *-------------------------------------------------------------------------
--
 * Argument    : unsigned long addr ; I : Address in serial flash memory to
write
 *             : unsigned char *buf ; I : Buffer address to store the write
data
 *             : int size           ; I : Number of bytes to write
 *-------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""*************************************************************/
void sf_byte_program(unsigned long addr, unsigned char *buf, int size)
{
  unsigned char cmd[4];

  cmd[0] = SFLASHCMD_BYTE_PROGRAM;
  cmd[1] = (unsigned char)((addr >> 16) & 0xff);
  cmd[2] = (unsigned char)((addr >>  8) & 0xff);
  cmd[3] = (unsigned char)( addr        & 0xff);
  write_enable();
  io_cmd_exe(cmd, 4, buf, size);
  busy_wait();
}

/*""FUNC
COMMENT""*************************************************************
 * ID          :
 * Outline     : Read data
 *-------------------------------------------------------------------------
--
 * Include     :
```

```
 *------------------------------------------------------------------------
--
 * Declaration : void sf_byte_read(unsigned long addr, unsigned char *buf, int
size);
 *------------------------------------------------------------------------
--
 * Function    : Reads the specified number of bytes from serial flash memory.
 *------------------------------------------------------------------------
--
 * Argument    : unsigned long addr ; I : Address in serial flash memory to
read
 *             : unsigned char *buf ; I : Buffer address to store the read
data
 *             : int size           ; I : Number of bytes to read
 *------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""*************************************************************/
void sf_byte_read(unsigned long addr, unsigned char *buf, int size)
{
  unsigned char cmd[4];

  cmd[0] = SFLASHCMD_BYTE_READ_LOW;
  cmd[1] = (unsigned char)((addr >> 16) & 0xff);
  cmd[2] = (unsigned char)((addr >>  8) & 0xff);
  cmd[3] = (unsigned char)( addr        & 0xff);
  io_cmd_exe_rdmode(cmd, 4, buf, size);
}

/*""FUNC
COMMENT""*************************************************************
 * ID          :
 * Outline     : Write enable
 *------------------------------------------------------------------------
--
 * Include     :
 *------------------------------------------------------------------------
--
 * Declaration : static void write_enable(void);
 *------------------------------------------------------------------------
--
 * Function    : Issues the Write Enable command to enable erasing or
programming
 *             : serial flash memory.
 *------------------------------------------------------------------------
--
 * Argument    : void
 *------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""*************************************************************/
static void write_enable(void)
{
  unsigned char cmd[1];
  cmd[0] = SFLASHCMD_WRITE_ENABLE;
  io_cmd_exe(cmd, 1, NULL, 0);
```

```
}

/*""FUNC
COMMENT""************************************************************
 * ID          :
 * Outline     : Write disable
 *-----------------------------------------------------------------------
--
 * Include     :
 *-----------------------------------------------------------------------
--
 * Declaration : static void write_disable(void);
 *-----------------------------------------------------------------------
--
 * Function    : Issues the Write Disable command to disable erasing or
programming
 *             : serial flash memory.
 *-----------------------------------------------------------------------
--
 * Argument    : void
 *-----------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""***********************************************************/
static void write_disable(void)
{
  unsigned char cmd[1];
  cmd[0] = SFLASHCMD_WRITE_DISABLE;
  io_cmd_exe(cmd, 1, NULL, 0);
}

/*""FUNC
COMMENT""************************************************************
 * ID          :
 * Outline     : Busy waiting
 *-----------------------------------------------------------------------
--
 * Include     :
 *-----------------------------------------------------------------------
--
 * Declaration : static void busy_wait(void);
 *-----------------------------------------------------------------------
--
 * Function    : Loops internally when the serial flash memory is busy.
 *-----------------------------------------------------------------------
--
 * Argument    : void
 *-----------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""***********************************************************/
static void busy_wait(void)
{
  while ((read_status() & 0x01) != 0) { /* RDY/BSY */
    /* serial flash is busy */
  }
```

```
}

/*""FUNC
COMMENT""*************************************************************
 * ID          :
 * Outline     : Read status
 *-----------------------------------------------------------------------
--
 * Include     :
 *-----------------------------------------------------------------------
--
 * Declaration : static unsigned char read_status(void);
 *-----------------------------------------------------------------------
--
 * Function    : Reads the status of serial flash memory.
 *-----------------------------------------------------------------------
--
 * Argument    : void
 *-----------------------------------------------------------------------
--
 * Return Value: Status register value
 *""FUNC COMMENT
END""*************************************************************/
static unsigned char read_status(void)
{
  unsigned char buf;
  unsigned char cmd[1];

  cmd[0] = SFLASHCMD_READ_STATUS;
  io_cmd_exe_rdmode(cmd, 1, &buf, 1);
  return buf;
}

/*""FUNC
COMMENT""*************************************************************
 * ID          :
 * Outline     : Write status
 *-----------------------------------------------------------------------
--
 * Include     :
 *-----------------------------------------------------------------------
--
 * Declaration : static void write_status(unsigned char status);
 *-----------------------------------------------------------------------
--
 * Function    : Writes the status of serial flash memory.
 *-----------------------------------------------------------------------
--
 * Argument    : unsigned char status ; I : status register value
 *-----------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""*************************************************************/
static void write_status(unsigned char status)
{
  unsigned char cmd[2];
```

```
  cmd[0] = SFLASHCMD_WRITE_STATUS;
  cmd[1] = status;

  write_enable();
  io_cmd_exe(cmd, 2, NULL, 0);
  busy_wait();
}
/*""FUNC
COMMENT""*************************************************************
 * ID          :
 * Outline     : RSPI initialization
 *-------------------------------------------------------------------------
--
 * Include     :
 *-------------------------------------------------------------------------
--
 * Declaration : static void io_init_rspi(void);
 *-------------------------------------------------------------------------
--
 * Function    : Initializes channel 0 of the RSPI.
 *             : Sets the RSPI in master mode to set parameters required to
transfer
 *             : according to the specifications of serial flash memory.
 *-------------------------------------------------------------------------
--
 * Argument    : void
 *-------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""*********************************************************/
static void io_init_rspi(void)
{
  if(PORT.PFCR3.BIT.PF12MD == 3){
        return;
    }

    /* ==== PORT ==== */
  PORT.PFCR3.BIT.PF12MD = 3;   /* PF12:MISO0 */
  PORT.PFCR2.BIT.PF11MD = 3;   /* PF11:MOSI0 */
  PORT.PFCR2.BIT.PF10MD = 3;   /* PF10:SSL00 */
  PORT.PFCR2.BIT.PF9MD  = 3;   /* PF9:RSPCK0 */

  /* ==== CPG ==== */
  CPG.STBCR5.BIT.MSTP51 = 0;   /* RSPI0 active */

  /* ==== RSPI ==== */
  RSPI0.SPCR.BYTE   = 0x00; /* Disables channel 0 of the RSPI */
  RSPI0.SPPCR.BYTE  = 0x30; /* MOSI idle fixed value = 1 */
  RSPI0.SPBR.BYTE   = 0x01; /* Specifies the base bit rate as 18 MHz
                                        (Bus clock = 72 MHz) */
  RSPI0.SPDCR.BYTE  = 0x20; /* Disables to send the dummy data */
                     /* Access width of the SPDR register: 8-bit */
  RSPI0.SPCKD.BYTE  = 0x00; /* RSPCK delay: 1 RSPCK */
  RSPI0.SSLND.BYTE  = 0x00; /* SSL negate delay: 1 RSPCK */
  RSPI0.SPND.BYTE   = 0x00; /* Next access delay: 1 RSPCK + 2 Bus clocks */
  RSPI0.SPSCR.BYTE  = 0x00; /* Sequence length: 1 (SPCMD0 is only used) */
  RSPI0.SPCMD0.WORD = 0xE783;  /* MSB first */
```

```
                    /* Data length: 8-bit */
                    /* Keeps the SSL signal level after transfer
                       is completed */
                    /* Bit rate: Base bit rate is divided by 1 */
                    /* RSPCK when it is idling is 1 */
                    /* Latches data on odd edge, outputs data on
                       even edge */
  RSPI0.SPBFCR.BYTE = 0xC0; /* Enable to reset data in the
                       transmit/receive buffer */
  RSPI0.SPBFCR.BYTE = 0x00; /* Disable to reset data in the
                       transmit/receive buffer */
                    /* Number of triggers in transmit buffer:
                       more than one byte available */
                    /* Number of triggers in receive buffer:
                       more than one byte received */
  RSPI0.SSLP.BYTE   = 0x00; /* SSLP = b'0 SSL signal 0-active */
  RSPI0.SPCR.BYTE   = 0x48; /* Master mode */
                    /* Disables interrupts */
                    /* Enables channel 0 of the RSPI */
}

/*""FUNC
COMMENT""************************************************************
 * ID          :
 * Outline     : Execute command (No read data).
 *-----------------------------------------------------------------------
--
 * Include     :
 *-----------------------------------------------------------------------
--
 * Declaration : static void io_cmd_exe(unsigned char *ope, int ope_sz,
 *             :                        unsigned char *data,int data_sz)
 *-----------------------------------------------------------------------
--
 * Function    : Executes the specified command.
 *             : Transmits the argument ope, and then transmits the argument
data.
 *             : Discards the receive data.
 *             : Set one of the values between 0 and 8 in the ope_sz.
 *             : Set one of the values between 0 and 256 in the data_sz.
 *-----------------------------------------------------------------------
--
 * Argument    : unsigned char *ope ; I : Start address of the opcode block
and
 *                                        address block to transmit
 *             : int ope_sz         ; I : Number of bytes in the opcode block
and
 *                                        address block
 *             : unsigned char *data; I : Start address of the data block to
transmit
 *             : int data_sz        ; I : Number of bytes in the data block
 *-----------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""*****************************************************/
static void io_cmd_exe(unsigned char *ope, int ope_sz, unsigned char *data,
int data_sz)
```

```
{
  unsigned char tmp;

  /* ==== Resets buffer ==== */
  RSPI0.SPBFCR.BYTE  = 0xC0u;
  RSPI0.SPBFCR.BYTE  = 0x00u;

  /* ---- Enables the SPI transfer ---- */
  RSPI0.SPCR.BIT.SPE = 1;

  /* ==== MOSI(command, address, write data) ==== */
  while(ope_sz--){
   RSPI0.SPDR.BYTE = *ope++;  /* Command size must be equal or less than 8
bytes */
  }
  while(data_sz--){
   while( RSPI0.SPSR.BIT.SPTEF == 0 ){
      /* wait */
   }
   RSPI0.SPDR.BYTE = *data++;
   if( RSPI0.SPSR.BIT.SPRF == 1 ){
      tmp = RSPI0.SPDR.BYTE;  /* Dummy read to avoid an overflow of data */
   }
  }
  io_wait_tx_end();           /* Waits fo transfer end */

  /* ---- SPI transfer end (SSL negation) ---- */
  RSPI0.SPCR.BIT.SPE = 0;
}
/*""FUNC
COMMENT""************************************************************
 * ID          :
 * Outline     : Execute command (With read data).
 *-----------------------------------------------------------------------------
--
 * Include     :
 *-----------------------------------------------------------------------------
--
 * Declaration : static void io_cmd_exe_rdmode(unsigned char *ope, int ope_sz,
 *             :                                unsigned char *rd,  int rd_sz)
 *-----------------------------------------------------------------------------
--
 * Function    : Executes the specified command.
 *             : Transmits the argument ope, and then receives data in the
argument rd.
 *             : Set one of the values between 0 and 8 in the ope_sz.
 *             : More than 0 can be set in the rd_sz.
 *-----------------------------------------------------------------------------
--
 * Argument    : unsigned char *ope ; I : Start address of the opcode block
and
 *                                      address block to transmit
 *             : int ope_sz         ; I : Number of bytes in the opcode block
and
 *                                      address block
 *             : unsigned char *rd  ; I : Buffer address to store the received
data
 *             : int rd_sz          ; I : Number of bytes in the data block
```

```
 *------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""*********************************************************/
static void io_cmd_exe_rdmode(unsigned char *ope, int ope_sz, unsigned char
*rd, int rd_sz)
{
  /* ==== Resets buffer ==== */
  RSPI0.SPBFCR.BYTE  = 0xC0u;
  RSPI0.SPBFCR.BYTE  = 0x00u;

  /* ---- Enables the SPI transfer ---- */
  RSPI0.SPCR.BIT.SPE = 1;

  /* ---- MOSI (command, address, dummy) ---- */
  while(ope_sz--){
    RSPI0.SPDR.BYTE = *ope++;  /* Command size must be equal or less than 8
bytes */
  }
  io_wait_tx_end();           /* Waits for transfer end */

  /* ---- MISO(read data) ---- */
  RSPI0.SPBFCR.BYTE  = 0xC0u;     /* Resets buffer */
  RSPI0.SPBFCR.BYTE  = 0x00u;

  RSPI0.SPDCR.BIT.TXDMY = 1;      /* Enables to transmit the dummy data */
  while(rd_sz--){
    while( RSPI0.SPSR.BIT.SPRF == 0){
       /* wait */
    }
    *rd++ = RSPI0.SPDR.BYTE;
  }
  RSPI0.SPDCR.BIT.TXDMY = 0;      /* Disable to transmit the dummy data */
  io_wait_tx_end();           /* Waits for transfer end */

  /* ---- SPI transfer end (SSL negation) ---- */
  RSPI0.SPCR.BIT.SPE = 0;
}

/*""FUNC
COMMENT""*************************************************************
 * ID          :
 * Outline     : Transfer end waiting
 *------------------------------------------------------------------------
--
 * Include     :
 *------------------------------------------------------------------------
--
 * Declaration : static void io_wait_tx_end(void);
 *------------------------------------------------------------------------
--
 * Function    : Loops internally until the transmission is completed.
 *------------------------------------------------------------------------
--
 * Argument    : void
 *------------------------------------------------------------------------
--
```

```
 * Return Value: void
 *""FUNC COMMENT
END""************************************************************/
static void io_wait_tx_end(void)
{
  while(RSPI0.SPSR.BIT.TEND == 0){
    /* wait */
  }
}

/* End of File */
```

5.1.3 “cpg.c”

```
/*""FILE COMMENT""************ Technical reference data
************************
*
*       System Name : SH7264 Sample Program
*       File Name   : cpg.c
*       Abstract    : CPG setting process
*       Version     : 1.00.00
*       Device      : SH7264/SH7262
*       Tool-Chain  : High-performance Embedded Workshop (Ver.4.07.00.007).
*                   : C/C++ compiler package for the SuperH RISC engine family
*                   :                                 (Ver.9.03.02.000).
*       OS          : None
*       H/W Platform: RSK2+SH7264
*       Disclaimer  :
*
*       The information described here may contain technical inaccuracies or
*       typographical errors. Renesas Electronics
*       assume no responsibility for any damage, liability, or other loss
rising
*       from these inaccuracies or errors.
*
*       Copyright (C) 2008 Renesas Electronics Corp. All Rights Reserved
*       AND Renesas Solutions Corp. All Rights Reserved
*
*       History     : June,2010 Ver.1.00.00
*""FILE COMMENT
END""*****************************************************/
#include "iodefine.h"


/* ==== Prototype Declaration ==== */
void io_set_cpg(void);

//#pragma section ResetPRG
/*""FUNC
COMMENT""*************************************************************
 * ID          :
 * Outline     : CPG settings
 *-------------------------------------------------------------------------
--
 * Include     : #include "iodefine.h"
 *-------------------------------------------------------------------------
--
 * Declaration : void io_set_cpg(void);
 *-------------------------------------------------------------------------
--
 * Description : Clock pulse generator (CPG) is set to set to the internal
clock
 *             : (I Clock), peripheral clock (P Clock), bus clock (B Clock),
and
 *             : I Clock = 144MHz, B Clock = 72MHz,  P Clock = 36MHz.
 *             : This setting example is the case that the function's input
clock
 *             : is 18MHz and clock mode is 2.
```

```
 *------------------------------------------------------------------------------
--
 * Argument    : void
 *------------------------------------------------------------------------------
--
 * Return Value: void
 *""FUNC COMMENT
END""************************************************************/
void io_set_cpg(void)
{
  /* ==== CPG Setting ==== */
  CPG.FRQCR.WORD = 0x1003u;    /* PLL1(x8),I:B:P= 8:4:2
                        * CKIO:Output at time usually,Output when bus right is
opened,output at standby"L"
                        * Clockin = 18MHz, CKIO = 72MHz
                        * I Clock = 144MHz, B Clock = 72MHz,
                        * P Clock = 36MHz
                        */


    /* ---- The clock of all modules is permitted. ---- */
  CPG.STBCR3.BYTE = 0x02u;     /* Port level is keep in standby mode
   */
                        /* IEBus, MTU2,SDHI0, SDHI1, A/D, [1], RTClock
  */
  CPG.STBCR4.BYTE = 0x00u;     /* SCIF0, SCIF1, SCIF2, SCIF3, SCIF4, SCIF5,
SCIF6, SCIF7*/
  CPG.STBCR5.BYTE = 0x10u;    /* I2C30, I2C31, I2C32, [1], RCAN0, RCAN1, RSPI0,
RSPI1    */
  CPG.STBCR6.BYTE = 0x00u;    /* SSI0, SSI1, SSI2, SSI3, CD-ROMDEC, SRC0, SRC1,
USB      */
  CPG.STBCR7.BYTE = 0x2au;     /* SIOF, RSPDIF, [1], VDC3, [1], CMT, [1], NAND
      */
  CPG.STBCR8.BYTE = 0x7eu;     /* PWM, [1], [1], [1], [1], [1], [1], DECOMP
   */

}

/* End of File */
```

Website and Support <website and support,ws>

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

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

Rev.	Date	Description	
		Page	Summary
1.00	Jan.12.11	—	First edition issued

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

Colophon 1.0