SH Family

Simple Flash API for SH2 and SH2A

R01AN0719EU0102
Rev.1.02
Jul 1, 2011

Introduction

A simple Application Program Interface (API) has been created to allow users of flash based SH2 and SH2A devices to easily integrate reprogramming abilities into their applications using User Mode programming. User Mode programming is the term used to describe a Renesas MCU's ability to reprogram its own internal flash memory while running in its normal operational mode. This application note focuses on using that API and integrating it with an application program.

The API source files comply with Renesas SH compiler only.

Target Device

The following is a list of devices able to use this API:

SH2/7137 Group: SH7136, SH7137

SH2A/7280 Group: SH7285, SH7286, SH7243

SH2A/7216 Group: SH7216, SH7214

SH2A/7239 Series: SH7239, SH7237

Contents

1. API Files

The API files have been separated according to MCU devices belonging to the same 'Group'. The table below shows what files need to be added to your project for your desired device.

Devices	Group	API Files
SH7136 (SH-2) SH7137 (SH-2)	SH7137 Group	Flash_API_SH7137.h Flash_API_SH7137.c
SH7243 (SH-2A) SH7285 (SH-2A) SH7286 (SH-2A)	SH7280 Group	Flash_API_SH7280.h Flash_API_SH7280.c
SH7216 (SH-2A) SH7214 (SH-2A)	SH7216 Group	Flash_API_SH7216.h Flash_API_SH7216.c
SH7239 (SH-2A) SH7237 (SH-2A)	SH7239 Group SH7237 Group	Flash_API_SH7239.h Flash_API_SH7239.c

2. Configuring the API

Before using the API, you must first configure the code. All general settings are done by modifying *#define* statements in the Flash_API_SHxxxx.h file.

2.1 Specifying your system clock speed

NOTE: This setting is not required for the SH7216.

The actual programming of your device is done by the CPU by means of code execution. Therefore, the CPU's execution speed needs to be known in order for the correct software delays to be accurate. Incorrect delays could cause damage to the flash memory. Specify the speed at which the CPU will be running during the time of the flash operation. It is perfectly acceptable to run your system at a different speed when not performing an erase/write operation.

The operating speed of the CPU is set by modifying the OPERATING_FREQUENCY parameter in the Flash_API_SHxxxx.h file. The speed is set in increments of Hertz. For example, if your CPU will be running 25MHz at the time you call the Erase and Write functions, then you would specify '25000000'.

```
#define OPERATING_FREQUENCY 25000000
```

NOTE: For some SH devices, a maximum operating speed for flash operations may be specified. If a maximum speed value exists for a device, it will be explained in the Flash_API_SHxxxx.h file where you set the 'OPERATING_FREQUENCY' value.

2.2 Specifying your input clock speed (SH7216 & SH7239 Only)

NOTE: Note that the following settings only apply to the SH7216, SH7239 Groups of devices.

Reprogramming (erase/write) the SH7216/SH7239 flash is performed by the flash control unit (FCU) operated by the peripheral clock (Pφ). As the peripheral clock controls the flash reprogramming time, Pφ must be set in the FCU.

Specifically, in the PLL_FREQUENCY parameter in the Flash_API_SH7216.h or Flash_API_SH7239.h file, set the input clock to a frequency multiplied by 16. The input clock operation speed is measured in MHz. For example, when the input clock is 12.5 MHz, multiply it by 16 to get 200 MHz (= 12.5 MHz × 16). Therefore, set the FCU to 200. After performing the multiplication, when there is a value other than 0 after the decimal point, round up the value.

```
#define PLL_FREQUENCY   200
```

Once this setting value is converted to the Pφ value in the API, it is set to the FCU.

2.3 Selecting RAM for Flash Operations

NOTE: This section does not apply to the SH7216 and SH7239.

The software algorithm for erasing and programming the flash memory must be executed from RAM. For the SH2 and SH2A devices discussed in this application note, that executable binary is stored in a hidden area of the MCU. The Flash API code will download that hidden binary into a RAM location to be used for erasing and writing. The size of the RAM required is 3Kbytes and the location is selected by setting the FTDAR_VALUE settings in the Flash_API_SHxxxx.h file. The choices for the RAM location are limited to a few select locations and vary from MCU to MCU. Please refer to the Flash_API_SHxxxx.h file of your intended device in order to select the location you wish to use.

NOTE: You may use this RAM location for your application up until the time you want to do an erase or program operation. When you call the FlashErase or FlashWrite function, the RAM area selected by FTDAR_VALUE will then be overwritten and your application data will be lost. If after your flashing operation you wish to use that RAM location again, please note that you cannot call the FlashErase or FlashWrite routine again until you reset the global variable 'current_firmware' to 0x00. This will signal the FlashAPI to re-download the internal binary the next time FlashErase or FlashWrite is called.

3. Precautions / Limitations

3.1 Debugging within HEW

First, please note that not all SH devices allow you to do in circuit application erasing and reprogramming while debugging with an E10A JTAG debugger. At the time this application note was created, the E10A User Manuals said "On-chip flash memory cannot be programmed while the user program is running" for the devices discussed in this application note.

Additionally, if you attempt to disconnect and then re-connect to your system with HEW, the entire flash memory will be erased upon re-connecting with the E10A so you will not be able to observe the effect of your application's reprogramming efforts. This is also true if you use the FDT programming software because an SH device will automatically erase all flash memory when it enters boot mode as its built-in security feature.

For the reasons mentioned above, the sample applications that are provided with these APIs output the contents of the flash using a serial port that should be connected to a terminal application (like Windows HyperTerminal) because that is an easy and reliable way to confirm the 'actual' contents of the flash.

The SH7216 contains a separate 'Data Flash' area that can be monitored and reprogrammed while operating under the E10A JTAG debugger.

3.2 SH7137 Group

SH7136, SH7137

- Flash operation cannot be performed while debugging with the E10A debugger.

3.3 SH7280 Group

SH7285, SH7286, SH7243

- Flash operation cannot be performed while debugging with the E10A debugger.

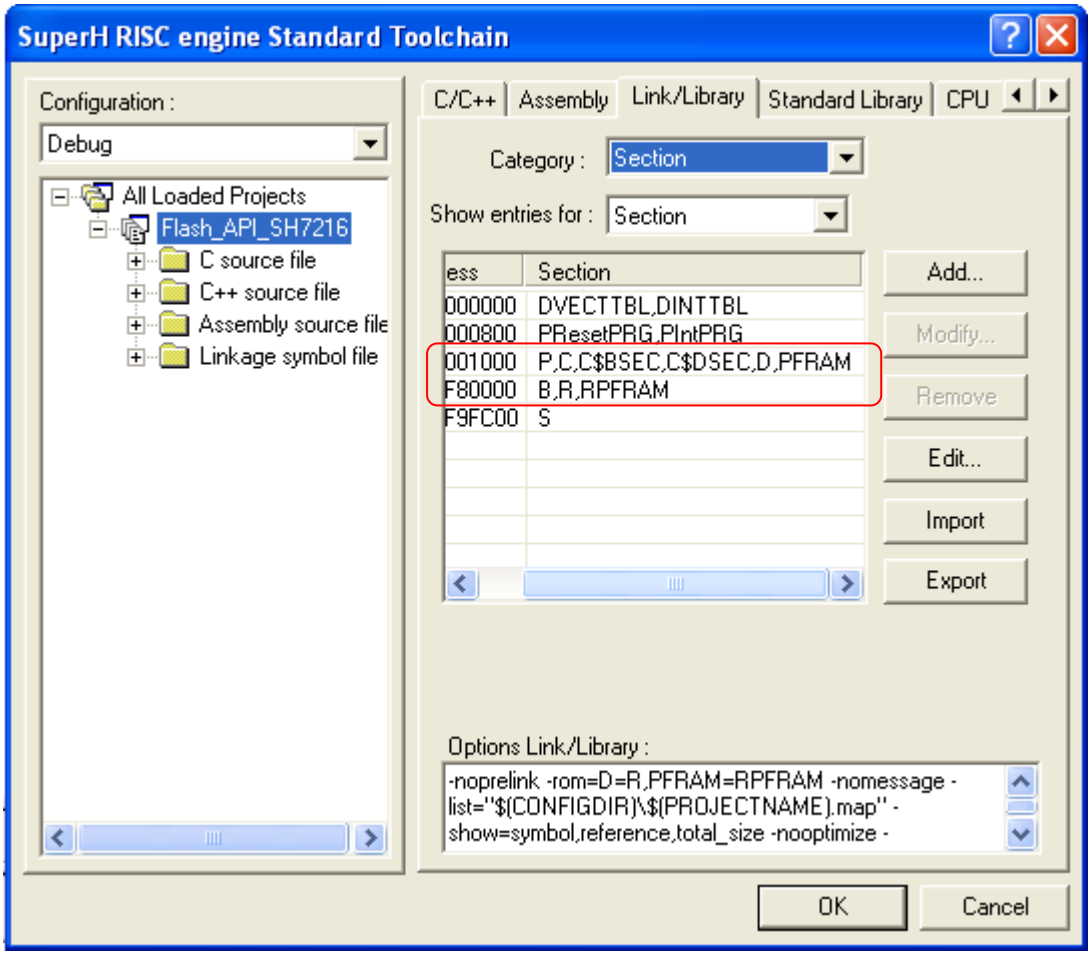
3.4 SH7216 Group, SH7239 Series

SH7216, SH7214, SH7239, SH7237

- The flash implementation for this device requires that sections in RAM and Flash be created to hold the API functions for reprogramming the user program area. Also, the RAM section will need to be initialized after RESET. Note that this is only for user program area programming. If you are only programming the Data Flash area, you do not need these settings, but you should change the configuration setting 'ENABLE_ROM_PROGRAMMING' to '0' in file 'Flash_API_SH7216.h'. Please follow steps 1-4 below:

Step 1: In HEW, add a new section titled '**RPFRAM**' in a RAM area.

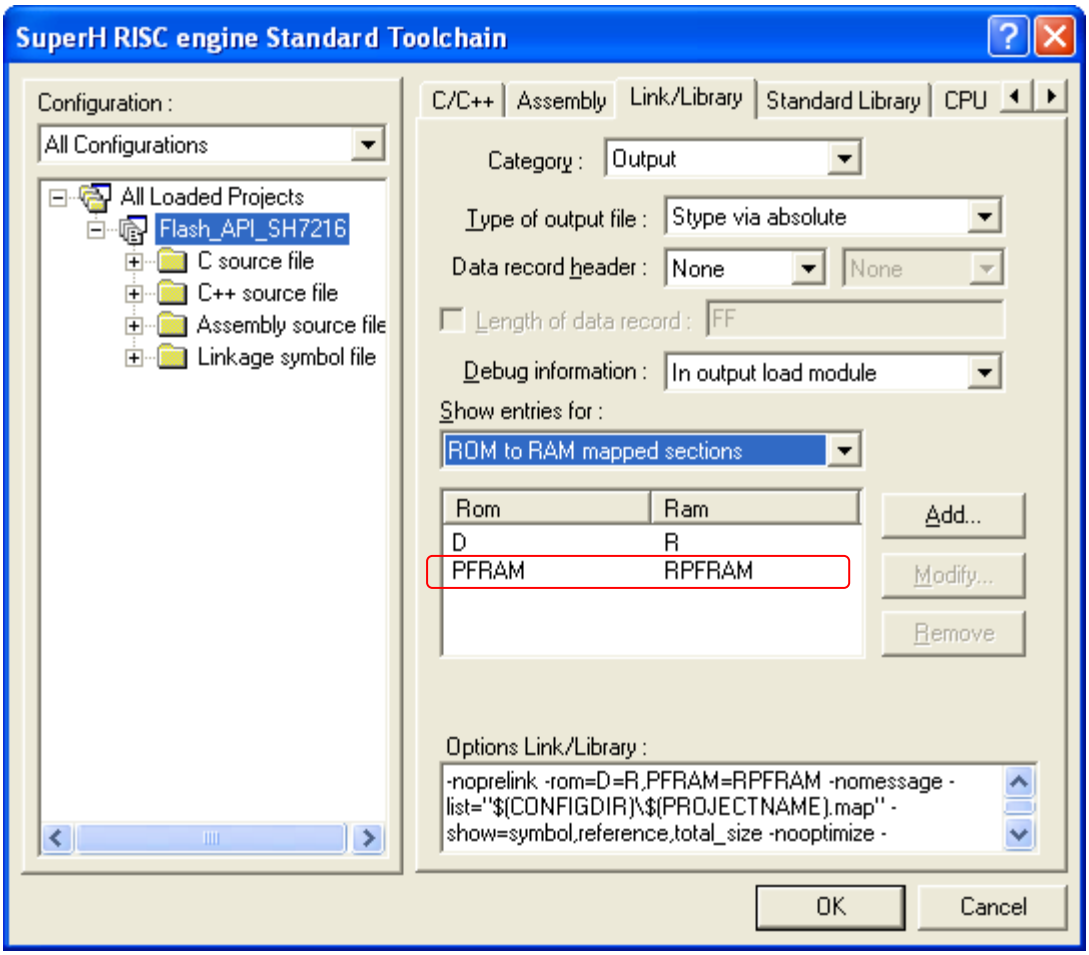
Step 2: In HEW, add a new section titled '**PFRAM**' in a Flash area.



Step 3: In the start file '**dbsct.c**', add the initialization of this code for the RAM section as seen below in RED (note, don't forget to add the comma on the previous line)

```
-- FILE [dbsct.c] --
#pragma section $DSEC
static const struct {
    _UBYTE *rom_s;       /* Initial address on ROM of initialization data section */
    _UBYTE *rom_e;       /* Final address on ROM of initialization data section */
    _UBYTE *ram_s;       /* Initial address on RAM of initialization data section */
}   DTBL[] = {
    { __sectop("D"), __secend("D"), __sectop("R") } ,
    { __sectop("PFRAM"), __secend("PFRAM"), __sectop("RPFRAM") }
};
```

Step 4: In HEW, add the linker option to map the ROM section (PFRAM) address to RAM section address (RPFRAM) as seen below.



4. Moving an entire application to RAM

If you wish to create a boot loader that will have the ability to erase/program the entire memory space of the device, then you will need to move your entire boot loader application to RAM since you cannot erase a block of flash that you are currently running out of. Below are some steps and considerations for doing this.

- Allocate a section in RAM within HEW's linker settings where you want your program to execute from. Make sure when you name the section that the first letter is an 'F' which signifies a 'Fixed' area section. For example, if you want a section called MY_APP_RAM, you would name the section named 'FMY_APP_RAM' in the linker settings.
- Because your RAM executable code will need a ROM location to be loaded and stored, you must first allocate a section within HEW's linker settings. Make sure when you name the section that the first letter is a 'P' which is required by the toolchain to signify a 'Program' area. For example, if you want a section called MY_APP, you would name the section named 'PMY_APP' in the linker settings.
- The SH linker has a special option that will assign RAM memory addresses for functions (and data), but then physically place it in a ROM location. This is done with the assumption that the application program will copy the code or data from the ROM storage location to the RAM execution location before it is referenced. After your code is moved to RAM, all the absolute address references will match your RAM location. Also whenever any other source module attempts to reference this code, it will be given its RAM address, not its ROM storage address. This is done by using the linker's ROM-to-RAM mapping option "-rom=xxxx=yyyy" where 'xxxx' would be the ROM section you have allocated and 'yyyy' would be the RAM section you have allocated. In our example, it would look like:

  -rom=PMY_APP=FMY_APP_RAM

- When it is time to execute your RAM based program, you must first copy the executable binary from its ROM storage location to its RAM executable location. Below is an example of how you could do that.

```
unsigned char *scr;
unsigned char *dst;

src = (unsigned char *)(__sectop("PMY_APP"));
dst = (unsigned char *)(__sectop("FMY_APP_RAM"));
for( ; src < (unsigned char *)(__secend("PMY_APP")); src++, dst++)
{
    *dst = *src;
}
```

- When writing your code, you will also need to tell the linker which functions should be part of this special ROM section that will be relocated to RAM. To do that, place "#pragma section MY_APP" before the function. Not that the 'P' before 'MY_APP' has been removed. This is because the compiler will automatically insert a 'P' at the beginning of each section that is intended to hold executable code. Please note that once the '#pragma section' is changed in a source file, all function and data declarations following it will be placed in that section as well until the end of the file. For more information, please refer to the SH Toolchain Manual. Below is an example of its usage.

```
#pragma section MY_APP
void function1( void )
{
    {THIS FUNCTION WILL BE PLACED IN 'PMY_APP'}
}
void function2( void )
{
    {THIS FUNCTION WILL BE PLACED IN 'PMY_APP'}
}
```

- One final consideration is to make sure that all reference functions be part of that section that will be relocated to RAM. It will be no good moving and executing your code from RAM if your code still accidentally calls a function in ROM (that you might have already erased). In some cases, compiler optimization may have your code call a common standard library function to increase code efficiency because calling a single library function will use less code than implementing the functionality multiple times throughout the code. An example of this would be doing 32-bit multiple operations in your application code. This sometimes may be tricky to spot unless you are examining the compiler's generated output. Since these libraries will be located in their default ROM based locations (not your special ROM-to-RAM section), your special re-programming code may execute OK for erasing a few blocks, but then after erasing the block with the library function call in it, your application will terminally crash.

5. API Functions

This function allows an entire flash block to be erased.

5.1 R_FlashErase

Format

```
uint8_t R_FlashErase(uint8_t block );
```

Parameters

block

Specifies the block to erase. This value is defined in the Flash_API_SHxxxx.h file. The blocks are labeled in the same fashion as they are in the device's Hardware Manual. For example, the block located at address 0x00000000 is called Block 0 in the hardware manual therefore "BLOCK_0" should be passed for this parameter.

Return Values

Returns the outcome of the erase operation:

0: Erase successful

Non-zero: An error occurred. Because the error codes vary from device to device, please see the function header in the .c file more information.

Properties

Prototyped in file "Flash_API_SHxxxx.h"

Implemented in file "Flash_API_SHxxxx.c"

Description

Erases a single block of Flash memory.

NOTE: Do not attempt to erase a flash block that you are currently executing out of.

5.2 R_FlashWrite

This function allows data to be written into flash.

Format

```
uint8_t R_FlashWrite( uint32_t   flash_addr,
                      uint32_t   buffer_addr,
                      uint16_t   bytes);
```

Parameters

flash_addr

This is a pointer to the flash area to write. The address must be on a programming line boundary. See *Description* below for important restrictions regarding this parameter.

buffer_addr

This is a pointer to the buffer containing the data to write to flash. This address must point to a location in RAM memory.

bytes

The number of bytes contained in the *buffer_addr* buffer. This number must be an increment of either 128 or 256 (depending on the MCU). See *Description* below for important restrictions regarding this parameter.

Return Values

Returns the outcome of the write operation:

0 = Operation Successful

Non-zero: An error occurred. Because the error codes vary from device to device, please see the function header in the .c file more information.

Properties

Prototyped in file ”Flash_API_SHxxx.h”

Implemented in file “Flash_API_SHxxx.c”

Description

Writes data to flash memory.

Because SH2 and SH2A device can only write in multiples of 128 or 256 bytes (depending on the MCU), you must pass a buffer that is an even multiple of the MCU’s programming ‘line’ size (128 or 256). Furthermore, the starting flash address you specify must be aligned to a line boundary. For example, if the programming line size is 128, then the flash address you pass must have bits B0-B6 must all be ‘0’. If the line is 256, then B7-B0 must all be ‘0’.

For the SH7216 and SH7239 only, this device contains a separate Data Flash (FLD) area that acts differently than program area flash. Please also see API functions ‘FlashFLDBlankCheck’ and ‘FlashFLDAreaAccess’ for the SH7216 and SH7239.

Below is a table show what the programming line size is for each Group.

MCU	Programming Line Size
SH7137 Group	128 byte
SH7286 Group	256 byte
SH7216 Group (Program Area) SH7239 Series (Program Area)	256 byte
SH7216 Group (Data Area) SH7239 Series (Data Area)	8 byte or 128 byte

5.3 R_FlashDataAreaAccess (SH7216 & SH7239 Only)

This function allows the Data Flash area to be accessed or modified.

Format

```
void R_FlashDataAreaAccess( uint16_t read_en_mask,
                            uint16_t write_en_mask);
```

Parameters

read_en_mask

This is a bitmapped value where bits 0 – 3 are used to determine which Data blocks should be able to be read by the MCU. A ‘0’ indicates the block cannot be accessed and a ‘1’ indicates it can. Bits 0-3 represent Data Blocks 0-3 respectively.

write_en_mask

This is a bitmapped value where bits 0 – 3 are used to determine which Data blocks should be able to be modified (Erase/Write) by the FCU. A ‘0’ indicates the block cannot be modified and a ‘1’ indicates it can. Bits 0-3 represent Data Blocks 0-3 respectively.

Return Values

None.

Properties

Prototyped in file ”Flash_API_SHxxx.h”
Implemented in file “Flash_API_SHxxx.c”

Description

After RESET, the Data Flash area is not readable by the MCU. It is also enabled for reprogramming. This function is used to select what blocks you would like to read or modify. You only have to set this function once if you like at the beginning of your application.

NOTE this function only applies the Data Flash (FLD) area of the SH7216 Group and SH7239 Series MCUs.

5.4 R_FlashDataAreaBlankCheck (SH7216 & SH7239 Only)

This function is used to determine an area in the Data Flash area is blank or not since this cannot be determine by simply reading the memory location.

Format

```
uint8_t R_FlashDataAreaBlankCheck(uint32_t address,
                                  uint8_t size);
```

Parameters

address

The address to check if is blank.
If the parameter 'size' is specified as 'BLANK_CHECK_8_BYTE', this should be set to an 8-byte address boundary.
If the parameter 'size' is specified as 'BLANK_CHECK_ENTIRE_BLOCK', this should be set to a defined Data Block Number ('BLOCK_DB0', 'BLOCK_DB1', 'BLOCK_DB2' or 'BLOCK_DB3').

size

This specifies if you are checking an 8-byte location, or an entire 8Kbyte block. You must set this to either 'BLANK_CHECK_8_BYTE' or 'BLANK_CHECK_ENTIRE_BLOCK'.

Return Values

Returns the outcome of the write operation:
0x00 = The location is blank
0x01 = The location is not blank
0x02 = The operation failed and a blank check was not properly completed.

Properties

Prototyped in file ”Flash_API_SHxxx.h”
Implemented in file “Flash_API_SHxxx.c”

Description

Before you can write to any flash area in an MCU, the area must already be blank. Since the memory locations with the SH7216/SH7239 Data Flash area are not represented by a defined ‘blank’ value of 0xFF like they are in the User Program area, an additional function is needed to test a section of flash to determine if it is blank.

The SH7216/SH7239 has two methods for check for blank areas; One is checking a single 8-byte location (since this flash area is 8-byte programmable) and the other method is to check the entire Data Flash block at once. This function does not have to be called for each section prior to programming. It is simply here to assist in application programming.

NOTE this function only applies the Data Flash (FLD) area of the SH7216 Group and SH7329 Series MCUs.

6. Example Code

The following is an example of using the API in order to erase and program a Block 2 on an SH7137 MCU.

```
#include "Flash_API_SH7137.h"

uint8_t write_buffer[256];   // Enough to hold two 128 byte lines
void main(void)
{
    uint8_t result;

    /* Erase Block 2 */
    result = R_FlashErase( BLOCK_2 );

    /* Write our data buffer into Flash block 2.
       Note that we are writing two 128 byte lines. */
    result = R_FlashWrite( (uint32_t)0x2000,       // Where to write
                           (uint32_t)write_buffer, // Our Data to write
                            256);                  // How much to write

    while(1);   /* END OF DEMO */
}
```

The following is an example of using the API in order to erase and program a Data Block 0 on an SH7216 MCU.

```
#include "Flash_API_SH7216.h"

uint8_t write_buffer[8];   // Data to program
void main(void)
{
    uint8_t result;

    /* Enable Read and Write access for the all the Data Flash blocks */
    R_FlashDataAreaAccess( 0xF, 0xF);

    /* Checks if a 8-bytes area is blank */
    result = R_FlashDataAreaBlankCheck( (uint32_t)0x80100400,
                                        BLANK_CHECK_8_BYTE );

    /* Erase Data Block 0 */
    result = R_FlashErase(BLOCK_DB0 );

    /* Write our data buffer into Flash Data Block 0. */
    result = R_FlashWrite((uint32_t) 0x80100000,   // Where to write
                          (uint32_t) write_buffer, // Our Data to write
                                     8);           // How much to write

    while(1);   /* END OF DEMO */
}
```

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Oct.23.09	—	First edition issued
1.01	Aug.18.10	all	New API format, added SH7214
1.02	Jul 1.11	most	New doc number, Bug fixes for SH7216 code, added SH7239

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 part number, confirm that the change will not lead to problems.

   — The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part 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: +49-211-65030, 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.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
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 1.1