Old Company Name in Catalogs and Other Documents

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

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

April 1st, 2010 Renesas Electronics Corporation

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

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

Notice

- 1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
- 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. 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; anticrime 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 majorityowned subsidiaries.
- (Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.



SuperH RISC engine C/C++ Compiler Package

APPLICATION NOTE: [IDE User's Guide] Simulator Usage Guide

This document explains useful simulator functionality.

Table of contents

1.	Preface	2
2. 2.1 2.2 2.3 2.4 2.4. 2.4.	••••	3 5 .11 .12 .12
3. 3.1 3.2 3.3	Image display Overview Supported image formats Sample program	16 18
4. 4.1 4.2 4.3	Profiler Overview Usage Sample program	22 23
5. 5.1 5.2 5.2. 5.2.		30 32 32
6. 6.1 6.2	Timer simulation Usage Sample program	36
7. 7.1 7.2	Eventpoints Usage Sample program	42
8. 8.1 8.1. 8.1. 8.1. 8.1. 8.2 Web	2 Label display 3 LED display	45 46 47 48 49 50

1. Preface

ENESA

This document explains the High-performance Embedded Workshop (herein as *Renesas IDE*) simulator functionality shown in Table 1-1.

Functionality	Chapter with explanation
Simulated I/O	Chapter 2
Image display	Chapter 3
Profiler	Chapter 4
Performance analysis	Chapter 4
Pseudo-interrupts	Chapter 5
Timer simulation	Chapter 6
Eventpoints	Chapter 7
Virtual I/O panels	Chapter 8

Each type of functionality comes with sample projects, which are provided from the download site with this document. SH-2A and SH-4 sample projects are provided for pseudo-interrupts. For functionality other than pseudo-interrupts, only SH-2A sample projects are provided. Table 1-2 shows the project name for each sample project. Make sure that you place the sample project workspace in C:\WorkSpace\sample.

Contents	Project name
Simulated I/O	sample_file_io
Image display	sample_img
Profiler / performance analysis	sample_profile
Pseudo-interrupts (for SH-2A)	sample_trigger_sh2a
Pseudo-interrupts (for SH-4)	sample_trigger_sh4
Timer simulation	sample_timer
Eventpoints	sample_ep
Virtual I/O panel	sample_panel

Table 1-2 Sample projects explained in this document

For details about the sample projects, see the sample program explanations given in each chapter. Make sure that sample projects are created on the following environment, as sample projects cannot be opened on earlier environments. Note that these sample projects are for simulator use. Sample programs for simulated I/O will not run on an emulator.

- Renesas SuperH Family C/C++ Compiler Package
- High-performance Embedded Workshop
- Toolchain
- Simulator

... V.9.01 Release 01 ... V.4.03.00

- ... V.9.1.1.0
- ... V.9.0.7

2. Simulated I/O

2.1 Overview

The simulator comes with simulated I/O functionality to virtually check files and standard I/O in the simulator. This chapter gives an overview of simulated I/O functionality, and explains sample programs that use this functionality to check files and standard I/O.

(1) Low-level Interface Routine

When C/C++ is used to develop programs, the standard I/O library (fopen(), printf(), scanf(), and other functions) is often used to perform file and standard I/O. In such cases, the standard I/O library calls the user-implemented functions actually performing I/O to perform file and standard I/O. The functions that actually perform I/O are called *Low-level Interface Routines*. These Low-level Interface Routines are needed because embedded applications have a variety of I/O destinations for standard I/O, including LCDs, hard disks, printers, CD-R/RW drives, DIP switches, keyboards, mice, mobile phone buttons, and touch panels, and each of these requires its own I/O processing applied to the user system. As such, the standard I/O library is comprised of a group of user-defined functions called *low-level interface routines*.

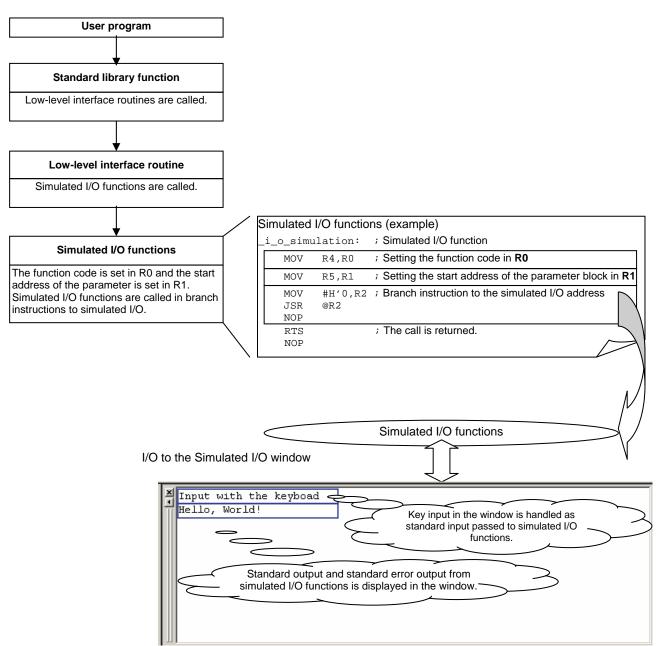
Low-level interface routines are implemented according to the specification in SuperHTM RISC engine C/C++ Compiler, Assembler, Optimizing Linkage Editor User's Manual 9.2.2 Execution Environment Settings (6) Low-level interface routines. This chapter introduces sample programs for the low-level interface routines for virtually performing file and standard I/O in a simulator. These sample programs correspond to standard input (stdin), standard output (stdout), standard error output (stderr), and file I/O.

(2) Simulated I/O functionality

Functionality to virtually perform file and standard I/O in a simulator is called *simulated I/O functionality*. Simulated I/O functionality is executed using branch instructions to a specific address (simulated I/O address) set in the simulator. Branch instructions to simulated I/O addresses are only called by simulated I/O functionality, as branching itself is not performed. Parameters for simulated I/O functionality are passed using the R0 register and R1 register. The simulated I/O functionality used (function code) is set in the R0 register. The simulator performs file I/O and string output to the simulator windows according to the function code set in the R0 register. The start address of the memory area (parameter block) in which file names, output characters, and other parameters specific to simulated I/O functionality are set is set in the R1 register. For details about how to set simulated I/O function codes and parameter blocks, see *2.2 Functionality*.

When simulated I/O functionality is called (a branch instruction is executed) as shown above, parameters need to be set in R0 and R1. As such, the parts of simulated I/O functionality called (simulated I/O functions) need to be coded in assembly language. The sample programs call simulated I/O function from low-level interface routines, so that file and standard I/O can be performed in the simulator.









2.2 Functionality

The function code needs to be set in the R0 register, and the parameter block start address needs to be set in the R1 register, before simulated I/O functionality is called. Figure 2-2 shows the contents set in the R0 register and R1 register. When I/O processing is finished, simulation is restarted from the next instruction of the branch instructions to the simulated I/O address.

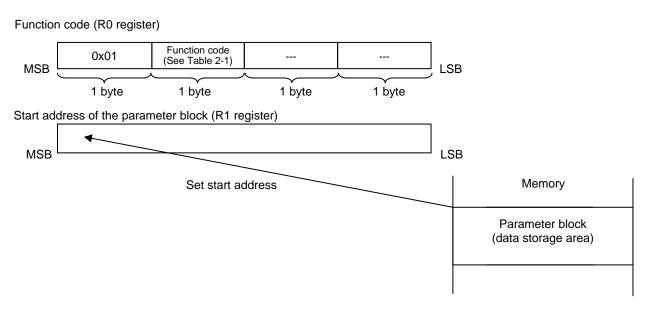


Figure 2-2

Table 2-1 shows the functionality supported by simulated I/O.

Table 2-1 List functionality

No.	Function Code	Function Name	Description
1	H'21	GETC	Inputs one byte from the standard input device
2	H'22	PUTC	Outputs one byte to the standard output device
3	H'23	GETS	Inputs one line from the standard input device
4	H'24	PUTS	Outputs one line to the standard output device
5	H'25	FOPEN	Opens a file
6	H'06	FCLOSE	Closes a file
7	H'27	FGETC	Inputs one byte from a file
8	H'28	FPUTC	Outputs one byte to a file
9	H'29	FGETS	Inputs one line from a file
10	H'2A	FPUTS	Outputs one line to a file
11	H'0B	FEOF	Checks for end of the file
12	H'0C	FSEEK	Moves the file pointer
13	H'0D	FTELL	Returns the current position of the file pointer

The following explains each kind of I/O functionality.

(1) Number shown in Table 2-1	(2) Function Name	(4) I/O overview
(1) Number snown in Table 2-1	(3) Function Code	(4) I/O overview
(5) I/O parameter block		(6) I/O parameters



GETC: Inputs one byte from the standard input device

1	GETC	Inputs one bute from the standard input device
1	H'21	Inputs one byte from the standard input device
	1 byte 1 byte +0 +2 Input buffer start address	 Input buffer start address (input) Start address of the buffer in which the output data is stored.

PUTC: Outputs one byte to the standard output device

2	PUTC H'22	Outputs one byte to the standard output device
	1 byte 1 byte +0 +2 Output buffer start address	 Output buffer start address (input) Start address of the buffer in which the output data is stored.

GETS: Inputs one line from the standard input device

2	GETS	Inputs one line from the standard input device
5	Н'23	Inputs one line from the standard input device
	1 byte 1 byte	 Input buffer start address (input) Start address of the buffer in which the output data is stored.
	+0 +2 Input buffer start address	

PUTS: Outputs one line to the standard output device

4	PUTS	Outputs one line to the standard output device
4	Н'24	Outputs one line to the standard output device
	1 byte 1 byte	 Output buffer start address (input) Start address of the buffer in which the output data is
	+0 +2 Output buffer start address	stored.



FOPEN: Opens a file

5	FOPEN	Orang a file
5	Н'25	Opens a file
The numl numl files. same	H'25 [FOPEN] opens a file and returns the file ber must be used to input, output, or close A maximum of 256 files can be open at the time. 1 byte 1 byte +0 Return value File number +2 Open mode Unused +4 +6 Start address of file name	 Return value (output) Normal completion 1 Error File number (output) The number to be used in all file accesses after opening. Open mode (input) H'00 "r" H'01 "w" H'02 "a" H'03 "r+" H'04 "w+" H'05 "a+" H'10 "b" H'12 "ab" H'13 "r+b" H'14 "w+b" H'15 "a+b" These modes are interpreted as follows. "r" Open for reading. "w" Open an empty file for writing. "a" Open for reading and writing. "r+" Open for reading and writing. "w+" Open an empty file for reading and writing. "a+" Open for reading and appending.
		• Start address of file name (input) The start address of the area for storing the file name.

FCLOSE: Closes a file

6	FCLOSE H'06	Closes a file
-	1 byte 1 byte +0 Return value File number	 Return value (output) 0 Normal completion -1 Error File number (input) The number returned when the file was opened.



FGETC: Inputs one byte from a file

7		GETC '27		Inputs one byte from a file
	1 +0 +2 +4 +6	Return value Unu	1 byte File number ised start address	 Return value (output) Normal completion Error File number (input) The number returned when the file was opened. Input buffer start address (input) Start address of the buffer in which the output data is stored.

FPUTC: Outputs one byte to a file

8 FPUTC H'28	Outputs one byte to a file
1 byte 1 byte +0 Return value File number +2 Unused +4 +6 Output buffer start address	 Return value (output) 0 Normal completion -1 Error File number (input) The number returned when the file was opened. Output buffer start address (input) Start address of the buffer in which the output data is stored.

FGETS: Inputs one line from a file

9		GETS 29		Inputs one line from a file
read	ds cl unt ad, o	naracter string d il either a new l or until the buff byte Return value Buffe	1 byte File number	 Return value (output) Normal completion 1 Error File number (input) The number returned when the file was opened. Buffer size (input) The size of the area for storing the read data. A maximum of 256 bytes can be stored. Input buffer start address (input) Start address of the buffer in which the output data is stored.

RENESAS

FPUTS: Outputs one line to a file

10	FPUTS H'2A	Outputs one line to a file
code	es character string data to a file. The NULL e that terminates the character string is not ten to the file. 1 byte 1 byte +0 Return value File number +2 Unused +4 +6 Output buffer start address	 Return value (output) Normal completion Error File number (input) The number returned when the file was opened. Output buffer start address (input) Start address of the buffer in which the output data is stored.

FEOF: Checks for end of the file

11	FEOF H'0B	Checks for end of the file
	1 byte 1 byte +0 Return value File number	 Return value (output) 0 File pointer is not at EOF -1 EOF detected File number (input) The number returned when the file was opened.

FSEEK: Moves the file pointer

12 -	FSEEK H'0C		Moves the file pointer
+(+2 +4 +6	0 Return value 2 Direction	1 byte File number Unused	 Return value (output) Normal completion Error File number (input)



FTELL: Returns the current position of the file pointer

13 FTELL H'0D	Returns the current position of the file pointer
1 byte 1 byte +0 Return value File number +2 Unused +4 +6 Offset	 Return value (output) Normal completion Error File number (input) The number returned when the file was opened. Offset (output) The current position of the file pointer, as a byte count from the start of the file.



2.3 Usage

To use simulated I/O, first enable simulated I/O functionality. Note that the Simulated I/O window needs to be displayed to check standard I/O.

(1) Enabling simulated I/O

To enable simulated I/O, from **Simulator** in the **Setup** menu, choose **System** to display the Simulator System dialog box. In the Simulator System dialog box, select the **Enable** checkbox and set the simulated I/O address.

Simulator System	? ×
System Memory	
<u>O</u> PU: SH2A-FPU (ROM Disable)	
<u>B</u> it size: D'32	Sjmulated I/O Address: 🔽 Enable
Endian: Big Endian	Execution Mode: Stop
Clock Rate:	Response: D'40000
	OK Cancel Apply



Simulated I/O functionality will not operate if **Simulated I/O Address** in the Simulator System dialog box and the simulated I/O address in the program do not match. For example, assume the simulated I/O address in the attached sample program is H'00000000. For an SH-4/SH-4A CPU, the simulated I/O address is H'00000004 by default. This means that if the sample program is used with SH-4/SH-4A, **Simulated I/O Address** needs to be changed in the Simulator System dialog box, or the simulated I/O address must be changed in the program.

(2) Displaying the simulated I/O window

To use simulated I/O, the Simulated I/O window must also be displayed. From **CPU** in the **View** menu, choose **Simulated I/O** to display simulated I/O. Note that the Simulated I/O window must also be displayed when file I/O is used.





(3) Using the Simulated I/O window

When a low-level interface routine is implemented using simulated I/O, standard output from the application program is output to this window. Likewise, key input in this window is the standard input to the application program.



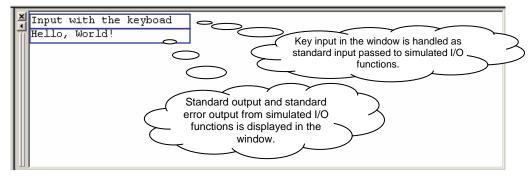


Figure 2-5

2.4 Sample program

lowsrc.src and lowsrc.c are implemented in the sample program so that file and standard I/O can be checked in simulated I/O.lowsrc.c implements the low-level interface routines used by the standard library functions: open(), close(), read(), write(), and lseek(). Note that with lowsrc.src, simulated I/O functions are implemented as called from low-level interface routines.

2.4.1 Source files

(1) lowsrc.src

Simulated I/O functions need to be coded in the assembler. The simulated I/O function $i_0_simulation$ is defined in lowsrc.src. This simulated I/O function is used by the low-level interface routines open(), close(), read(), write(), and lseek() to call simulated I/O functionality.

The following processing is performed before i_o_simulation calls simulated I/O functionality:

- The function code specified in the first argument is set in R0.
- The parameter block address specified in the second argument is set in R1.

(2) lowsrc.c

The low-level interface routines open(), close(), read(), write(), and lseek(), as well as the standard library initialization programs _INIT_IOLIB() and _CLOSEALL() are implemented in lowsrc.c.



Function name	Functionality	Processing
open	Opening files	This calls the i_o_simulation function with the FOPEN function code specified. If stdin is specified, the specified open mode is ignored, and the file is opened as read-only. If anything other than stdin is specified, the file is opened according to the open mode.
close	Closing files	This calls the i_o_simulation function with the FCLOSE function code specified, and closes the file.
read	Inputting data	If stdin is specified, this calls the $i_o_simulation$ function with the GETC function code specified, and loads input from the standard input. If anything other than stdin is specified, the $i_o_simulation$ function is called with the FGETC function code specified, and the string is read from the specified file.
write	Outputting data	When stdout/stderr is specified, the i_o_simulation function is called with the PUTC function code specified, and the specified string is output to the standard output. If anything other than stdout/stderr is specified, the i_o_simulation function is specified with the FPUTC function code specified, and the string is output to the specified file.
lseek	Moving a file pointer to the specified location	This calls the i_o_simulation function with the FSEEK function code specified, and moves the file pointer to the specified location. It then calls the i_o_simulation function with the FTELL function code specified, to obtain the current offset value.
_INIT_IOLI B	Performing initial standard I/O settings	This opens stdin/stdout/stderr.
_CLOSEALL	Closing all files	This closes all open files (including stdin/stdout/stderr).

Note In the sample program, the _CLOSEALL() function is called when main() terminates, and processing is performed to close all files. During program execution, when execution is performed after reset but before close processing is called, since the _CLOSEALL() function has not been called, operation may not be performed as expected. When using the _INIT_IOLIB() and open() functions, execute them after calling the _CLOSEALL() function, and after reset. To reset the debugger without executing the _CLOSEALL() function, from the File menu, choose Refresh Session.



2.4.2 Main processing

The main function of the sample program uses the scanf() function to load the string input to the standard input, and then uses the printf() function to output the loaded string to the standard output. It then uses the fopen() function, to open the C:\WorkSpace\sample\file.txt file, and uses the fscanf() function to load the string from the file, and uses the printf() function to output the loaded string to the standard output.

```
#include <stdio.h>
unsigned char buf[100];
void main (void)
{
    FILE* fp;
    scanf ("%s", buf);
    printf ("Input=%s\n", buf);
    fp = fopen ("C:\\WorkSpace\\sample\\file.txt", "r");
    if ( fp != NULL ) {
        fscanf (fp, "%s", buf);
        printf ("File=%s\n", buf);
        fclose (fp);
    }
}
```

The scanf(), printf(), fopen(), fscanf(), and fclose() functions in the standard library call each of the following low-level interface routines:

- scanf(): Calls the read() function for the standard input (stdin).
- printf(): Calls the write() function for the standard output (stdout).
- fopen(): Calls the open() function for the specified file for fopen().
- fscanf(): Calls the read() function for the specified file for fscanf().
- fclose(): Calls the close() function for the specified file pointer for fclose().



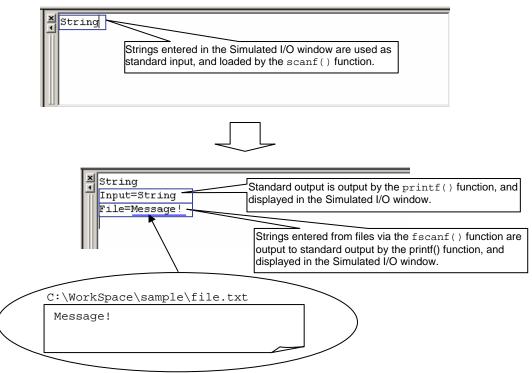


Figure 2-6

Note When checking file I/O for simulated I/O, use an absolute path to specify the file name specified for file open (fopen()). Make sure that \setminus , not \setminus , is specified as the separator in the absolute path.



3. Image display

3.1 Overview

Image display functionality can be used to visually confirm the contents of image data during program execution. From **Graphic** in the **View** menu, choose **Image** to display the Image Properties dialog box.

	ng: 4:4:4
MONOCHROME Bit/Pixe Samplin BGR YObOr Eormat	ig: 4:4:4
BGR Samplin Eormat	
C BGR C YObOr Buffer Information:	
Buffer Information:	
Data <u>A</u> ddress: 00000000	
	•
Palette Address: 00000000	
-Width/Height Size(Pixel):	
Width: 0 Bu	iffer Size(Hex): 00000000 Byte
Height: 0	0000000 Dyte
View Information: View Mode: Full Size X Position:	
	, i i i i i i i i i i i i i i i i i i i
0.7	t Size(Pixel):
• Top Width:	0
O Bottom Height:	0



In the Image Properties dialog box, when the buffer address and size are specified in **Buffer Information**, and the format of image data in the buffer is specified in **Color Information**, image data is displayed in the Image window during program execution. The following chapter explains the image formats that can be displayed.

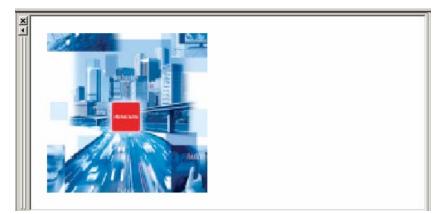


Figure 3-2

To update the images during program execution in real-time, right-click the Image window to display the pop-up menu, and then from **Auto Refresh**, choose **Real time**.







Figure 3-3



3.2 Supported image formats

The following image formats can be selected from the Image Properties.

Table 3-1 List of image formats - Monochrome/RGE	3/BGR
--	-------

Color information		Data	Data image				
Mode	Bits/pixel						
Monochrome	_	One bit per pixel Bit data - 1: white, 0: black	Bit data: B'10100000 □□□□□□□□□				
RGB/BGR	8 bits (Index Color)	A 256-color (RGB/BGR represented in 32 bits) palette is used, in which image data is stored as numbers (0 to 255) referencing the palette table.	Pixel data: H'05 This references the sixth color from the palette table. The sixth pixel data from the palette table is H'E7 H'C3 H'81 H'00				
		The pixel data from the palette table is represented in Blue: 8 Green: 8 Red: 8 alpha: 8-bit.	BGR: Blue: 231 Green: 195 Red: 129 Alpha: 0 231 195 129				
		Image display functionality ignores the value of alpha for transparency information.					
	16 bits (5:5:5)	RGB: Red: 5 Green: 5 Blue: 5 BGR: Blue: 5 Green: 5 Red: 5	Pixel data H'4677 (=B'0100011001110111) RGB: Red: 17 Green: 19 Blue: 23				
		The highest bit is padding.	17 19 23 BGR: Blue: 17 Green: 19 Red: 23				
			17 19 23				
	16 bits (5:6:5)	BGR: Blue: 5 Green: 6 Red: 5-bit	Pixel data H'8A77 (=B'1000101001110111) RGB: Red: 17 Green: 19 Blue: 23				
		The highest bit for green is padding.	BGR: Blue: 17 Green: 19 Red: 23				
	24 bits	RGB: Red: 8 Green: 8 Blue: 8-bit BGR: Blue: 8 Green: 8 Red: 8-bit	Pixel data H'81 H'C3 H'E7 RGB: Red: 129 Green: 195 Blue: 231 129 195 231				
			BGR: Blue: 129 Green: 195 Red: 231				
	32 bits	RGB: Alpha: 8 Red: 8 Green: 8 Blue: 8-bit BGR: Alpha: 8 Blue: 8 Green: 8 Red: 8-bit	Pixel data H'00 H'81 H'C3 H'E7 RGB: Alpha: 0 Red: 129 Green: 195 Blue: 231 129 195 231				
		Image display functionality ignores the value of alpha for transparency information.	BGR: Alpha: 0 Blue: 129 Green: 195 Red: 231 <u>129 195 231</u>				



Color ir	nformation		ata	Data image
Mode	Sampling	Source data	Sampling data	
INIOUE	ratio			
YCbCr	4:4:4	11,12,13,14	11,12,13,14	chunky
		21,22,23,24	21,22,23,24	Y ₁₁ ,Cb ₁₁ ,Cr ₁₁ ,Y ₁₂ ,Cb ₁₂ ,Cr ₁₂ ,Y ₁₃ ,Cb ₁₃ ,Cr ₁₃ ,
		31,32,33,34	31,32,33,34	,Y _{nm} ,Cb _{nm} ,Cr _{rnm}
		41,42,43,44	41,42,43,44	
				planar
				$Y_{11}, Y_{12}, Y_{13}, \dots, Y_{nm}, Cb_{11}, Cb_{12}, Cb_{13}, \dots, Cb_{nm},$
				$Cr_{11}, Cr_{12}, Cr_{13}, \dots, Cr_{nm}$
				planar2
				Y ₁₁ , Y ₁₂ , Y ₁₃ ,,Y _{nm} ,
				$Cb_{11}, Cr_{11}, Cb_{12}, Cr_{12}, Cb_{13}, Cr_{13}, \dots, Cb_{nm}, Cr_{nm}$
	4:2:2	11,12,13,14	11,11,13,13	chunky
		21,22,23,24	21,21,23,23	$Y_{11}, Y_{12}, Cb_{11}, Cr_{11}, Y_{13}, Y_{14}, Cb_{13}, Cr_{13},$
		31,32,33,34	31,31,33,33	$, Y_{n \ (m-1)}, Y_{nm}, Cb_{n \ (m-1)}, Cr_{n \ (m-1)}$
		41,42,43,44	41,41,43,43	
				planar
				$Y_{11}, Y_{12}, Y_{13}, \dots, Y_{nm}, Cb_{11}, Cb_{13}, Cb_{15}, \dots, Cb_{n \ (m-1)},$
				$Cr_{11}, Cr_{13}, Cr_{15}, \dots, Cr_{n (m-1)}$
				planar2
				$Y_{11}, Y_{12}, Y_{13}, \dots, Y_{nm},$
				$Cb_{11}, Cr_{11}, Cb_{13}, Cr_{13}, Cb_{15}, Cr_{15}, \dots, Cb_{n \ (m-1)}, Cr_{n \ (m-1)}$
	4:1:1	11,12,13,14	11,11,11,11	chunky
		21,22,23,24	21,21,21,21	$Y_{11}, Y_{12}, Y_{13}, Y_{14}, Cb_{11}, Cr_{11},$
		31,32,33,34	31,31,31,31	$Y_{21}, Y_{22}, Y_{23}, Y_{24}, Cb_{23}, Cr_{21},$
		41,42,43,44	41,41,41,41	$,Y_{n\ (m-3)},Y_{n\ (m-2)},Y_{n\ (m-1)},Y_{nm},Cb_{n\ (m-3)},Cr_{n\ (m-3)}$
				planar
				$Y_{11}, Y_{12}, Y_{13},, Y_{nm}, Cb_{11}, Cb_{13}, Cb_{15},, Cb_{n (m-1)},$
				$Cr_{11}, Cr_{13}, Cr_{15}, \dots, Cr_{n (m-1)}$
				planar2
				$Y_{11}, Y_{12}, Y_{13}, \dots, Y_{nm},$
				Cb ₁₁ ,Cr ₁₁ ,Cb ₁₃ ,Cr ₁₃ ,Cb ₁₅ ,Cr ₁₅ ,,Cb _{n (m-1)} ,Cr _{n (m-1)}
	4:2:0	11,12,13,14	11,11,13,13	planar
		21,22,23,24	11,11,13,13	Y ₁₁ ,Y ₁₂ ,Y ₁₃ ,,,Y _{nm} ,
		31,32,33,34	31,31,33,33	$Cb_{11}, Cb_{13}, Cb_{15}, \dots, Cb_{n (m-1)},$
		41,42,43,44	31,31,33,33	Cr ₁₁ ,Cr ₁₃ ,Cr ₁₅ ,,Cr _{n (m-1)}
				planar2
				$Y_{11}, Y_{12}, Y_{13}, \dots, Y_{nm},$
				$Cb_{11}, Cr_{11}, Cb_{13}, Cr_{13}, Cb_{15}, Cr_{15}, \dots, Cb_{n (m-1)}, Cr_{n (m-1)}$

Table 3-2 List of image formats (2) - YCbCr



3.3 Sample program

In the sample program, the two pieces of image data, data0 and data1, are alternately copied to the buffer buf. The image data used is in a 200x200-pixel, 24-bit RGB format.

```
#define SIZE (120000)
extern const unsigned char data0[SIZE]; /* Image Data0 */
extern const unsigned char data1[SIZE]; /* Image Data1 */
unsigned char buf[SIZE]; /* Buffer */
void main (void)
{
    int i;
    for (;;)
    ł
        for (i=0; i < SIZE; i++ ) {</pre>
            buf[i] = data0[i]; /* Image Data0 */
        for (i=0; i < SIZE; i++ ) {</pre>
            buf[i] = data1[i]; /* Image Data1 */
        }
    }
}
```

From **Graphic** in the **View** menu, choose **Image** to display the Image Properties dialog box, and perform the following settings:

- Color Information
 - For **Mode**, choose **RGB**
 - From the **Bit/Pixel** drop-down list, choose **24 bits**
- Buffer Information
 - Click the button to the right of the Data Address field to display the Select Label dialog box, and then select _buf
 - For Width/ Height Size (Pixel), enter 200 in decimal for both Width and Height





– Mode: ––––––––––––––––––––––––––––––––––––		_		
	OME	<u>B</u> it/Pixel:	24bit	-
RGB		Sampling:	4:4:4	~
O BGR				
C YCbCr		<u>F</u> ormat:	chunky	<u></u>
Buffer Information:				
Data <u>A</u> ddress:	_buf			• 🕫
Palette Address:	00000000			
-Width/Height Size	(Pixel):			
Width:		200 Buffer	r Size(Hex):	.
<u>H</u> eight:		200	0001 D4C0	Byte
View Information: —				
- <u>V</u> iew Mode:		'osition: ——		0
Full Size		<u>X</u> Position:		
C Part Size		Y Position:		0
		Width/Height S	ize(Pixel): —	
-S <u>t</u> art Position:		Width:		0
-S <u>t</u> art Position:		AATO NUS		



The image is displayed in the Image window. To have the image update in real-time, right-click the Image window to display the pop-up menu, and in **Auto Refresh**, choose **Real time**. Swapping between the two image data can be seen.



Figure 3-5



4. Profiler

4.1 Overview

The simulator comes with profiler functionality and performance analysis functionality to measure the execution performance of application programs. The following lists characteristics of profiler functionality and performance analysis functionality.

Table 4-1 Characteristics of	profiler functionality	and performance	analysis functionality

Item	Performance analysis functionality	Profiler functionality
Measurement coverage	Specific functions	All functions
Obtainable execution performance items	Few	Many
Child function measurement	Not possible	Possible
Simulation speed	Fast	Slow

(1) Performance analysis functionality

Performance analysis functionality measures the following items for specific functions in an application program:

- Execution cycle count
- Call count
- Ratio of execution cycle count for the corresponding function as a ratio of program-wide execution cycle count
- Histogram of above ratio

Since only specific functions are measured, faster simulations are possible compared to profiling. Use this when measuring large-scale applications that require significant time to simulate.

Note that measurement is not performed for child functions. To measure a child function, include the child function as part of the measurement target for performance analysis.

(2) Profiler functionality

Profiler functionality measures various execution performance items for all functions in an application program. It can be used for detailed analysis of area and causes of performance degradation with an application program, as well as displaying execution results, including those for child functions.

For details about the execution performance items that can be obtained, see $SuperH^{TM}$ RISC engine Simulator/Debugger User's Manual 3.6.10 Types and Purposes of Displayed Data.

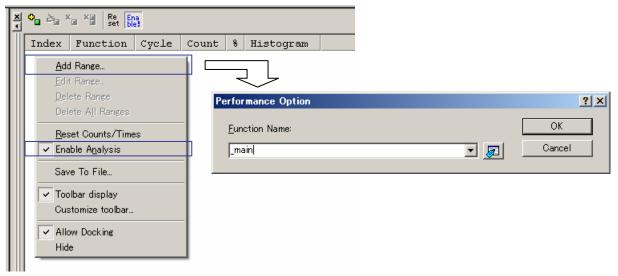
RENESAS

4.2 Usage

The following explains how to use performance analysis functionality and profiler functionality.

(1) Performance analysis functionality

From **Performance** in the **View** menu, choose **Performance Analysis** to display the Performance Analysis window. In the Performance Analysis window, right-click to display the pop-up menu, and then choose **Add Range** to display the Performance Option dialog box. In the Performance Option dialog box, specify the name of the function to be measured. Then, from the same pop-up menu, choose **Enable Analysis** (a check mark will be displayed in the pop-up menu).





When the program is executed, the specified function is measured. The following figure shows the measurement results.

×	• <u>•</u> ≥ ₀ :	K Re Re set	Ena ple1			
Ш	Index	Function	Cycle	Count	8	Histogram
	0	_main	35	1	0%	

Figure 4-2

The items displayed are as follows:

Index	Index number of the set condition
Function	Name (or start address) of the measured function
Cycle	Cumulative execution cycle count for the corresponding function
Count	Cumulative call count for the corresponding function
% count	Execution cycle count for the corresponding function, as a ratio of the program-wide execution cycle
Histogram	Histogram display of the above ratio

RENESAS

(2) Profiler functionality

From **Performance** in the **View** menu, choose **Profile** to display the Profiling window. Right-click in the Profiling window to display the pop-up menu, and choose **Enable Profiler** (a check mark will be displayed in the pop-up menu).

To include child functions in the displayed execution results, from **Setting** in the pop-up menu, choose **Include Data of Child Functions** (a check mark will be displayed in the pop-up menu).

×	🗈 - 🗧 Show Functions/Variables		▼ TE Data	0			
	Function/Variable F/V Addre	33 	Size	Times	Cycle	ICache	miss
	Vie <u>w</u> Source View Profile- <u>C</u> hart						
	Vertical Antices and the second secon	_					
	Setting Properties			Functions/ <u>F</u> unctions	Variables		
	Eind Ctrl+F Cle <u>a</u> r Data	_		<u>V</u> ariables ixecuted F	unctions		
	Output Profile Information Files Output Text File	T	Include	e Data of (Child Func	tions	
	✓ Toolbar display Customize toolbar						
	Allow Docking Hide						
	List Tree /						

Figure 4-3

The program is executed, and the function and variable information is obtained. The function execution count or variable access count is displayed for **Times**, and the execution cycle count is displayed for **Cycle**. For details about other items displayed, see *SuperHTM RISC engine Simulator/Debugger User's Manual 3.6.10 Types and Purposes of Displayed Data*.

The Profiling window contains a List sheet and a Tree sheet, as shown in the following figure.

- List
 - The measurement results are displayed in a list.

×	🗈 🤟 📴 Show Function	ns/Variat	oles 📘	. 📴 🖉							
	Function/Variable	F/V	Address	Size	Times	Cycle	I.	0. E	. I	. I	<u> </u>
H	_PowerON_Reset_PC	F	00000800	н'00000022	1	10301	0	03	0	1	
	freeptr	v	FFF8062C	н'00000004	1	0	0	0 0	0	0	
	_errno	v	FFF80624	н'00000004	4	0	0	0 0	0	0	
	_buf	v	FFF801CO	н'00000004	12	0	0	0 0	0	0	
	_flmod	v	FFF801AC	н'00000004	20	0	0	0 0	0	0	
	_sml_buf	v	FFF80198	н'00000004	20	0	0	0 0	0	0	
	iob	v	FFF80008	н'00000004	23	0	0	0 0	0	0	
	ctype	v	0000736C	H'00000001	1	0	0	0 0	0	0	
	_alocbuf	F	000053BC	н'00000074	2	114	0	02	0	29	
	_strlen	F	00004 F 3C	н'00000010	1	31	0	0 0	0	5	
H	flshbuf	F	00004898	H'00000D8	4	797	0	03	0 0	168	-
	List Tree										

Figure 4-4



• Tree

- The **Function** column displays the measurement results in tree format. Double-click the **Function** column to expand or hide the tree structure.

🞽 🗈 🗲 🔚 Show Functions/Variables		Tere Ø							
Function	Address	Size	Stack Size	Times	Cycle	I	0	Ε.	I
□ C:\WorkSpace\sample									
PowerON_Reset_PC	00000800	н'00000022	н'00000000	1	10301	0	0	з.	1
INITSCT	0000144C	н'00000000	н'00000000	1	3236	0	0	11	407
main	000013 F 8	н'00000038	н'00000000	1	4155	0	0	1.	842
scanf	0000158C	H'000003C	н'00000000	1	2178	0	0	59	452
printf	00001550	H'000003C	н'00000000	1	1942	0	0	60	386
CLOSEALL	000013B4	н'00000044	н'00000000	1	748	0	0	37	108
fclose	000014B4	н'00000040	н'00000000	3	487	0	0	33	78
init_Iolib	00001304	н'000000в0	н'00000000	1	2132	0	0	1.	362
freopen	000014F4	H'0000005c	н'00000000	3	1382	0	0	1.	205
List Tree									

Figure 4-5

The measurement results in the Profiling window can be used to display the call relationship for functions according to a specific function.

Select a function as shown in Figure 4-4 or Figure 4-5, and right-click it to display a pop-up menu. From the pop-up menu, choose **View Profile-Chart** to display the Profile-Chart window.

The chosen function is displayed in the center of the Profile-Chart window, with the function from which it was called on the right, and the functions it called on the left. Note that the number of times each function is called is also displayed.

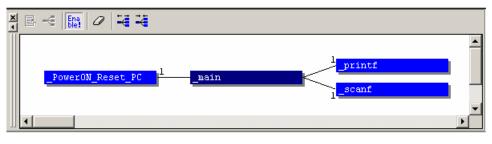


Figure 4-6



4.3 Sample program

The following shows examples of performance measurement using performance analysis functionality and profiler functionality, which in turn uses the three types of sort programs provided as sample programs (quicksort: quicksort(), heapsort(), and bubblesort: bubblesort()).

Note that the order of unbiased efficiency for sorted data is: quicksort > heapsort > bubblesort.

Main processing

```
#include <string.h>
extern void quicksort (int n, int a[]);
extern void bubblesort (int n, int a[]);
extern void heapsort (int n, int a[]);
#define N (80)
const int array[N] = {
    0, 690, 505, 591, 554, 378, 257, 207, 626, 340,
43, 68, 409, 879, 319, 980, 85, 907, 102, 921,
  843,
  507, 872, 333, 692, 556, 361,
                                       31, 858,
                                                   98, 877,
  449\,,\ 432\,,\ 606\,,\ 927\,,\ 664\,,\ 395\,,\ 438\,,\ 652\,,\ 928\,,\ 949\,,
  307, 596, 783, 338, 805, 942,
                                      66, 857, 977, 889,
  545, 864, 457, 800, 873, 821, 185, 86, 638, 233,
462, 7, 635, 421, 953, 210, 970, 261, 857, 581,
  707, 285, 318, 643, 858, 668, 443, 55, 777, 594,
};
void main (void)
{
    int dist1[N];
    int dist2[N];
    int dist3[N];
    memcpy (dist1, array, sizeof (int) * N);
    quicksort (N, dist1);
    memcpy (dist2, array, sizeof (int) * N);
    bubblesort (N, dist2);
    memcpy (dist3, array, sizeof (int) * N);
    heapsort (N, dist3);
```



• Quicksort: quicksort()

```
/* Quick Sort */
void qsort (int a[], int first, int last);
void quicksort (int n, int a[])
{
    qsort (a, 0, n-1);
}
void qsort (int a[], int first, int last)
{
    int i, j;
    int pivot;
    int tmp;
    pivot = a[ (first + last) / 2];
    i = first; j = last;
    for (;;) {
        while (a[i] < pivot) i++;</pre>
        while (pivot < a[j]) j--;</pre>
        if (i >= j) break;
        tmp = a[i]; a[i] = a[j]; a[j] = tmp;
i++; j--;
    }
    if (first < i - 1) qsort (a, first , i - 1);
    if (j + 1 < last) qsort (a, j + 1, last);</pre>
```

• Heapsort: heapsort()

```
/* Heap Sort */
void heapsort (int n, int a[])
{
    int i, j, k;
    int x;
    for (k=n/2; k >= 1; k--) {
        i = k; x = a[i];
        while ( (j=2*i) <= n) {
            if (j < n && a[j] < a[j + 1]) j++;
            if (x \ge a[j]) break;
            a[i] = a[j]; i = j;
        }
        a[i] = x;
    }
    while (n > 1) {
        x = a[n]; a[n] = a[1]; n--;
        i = 1;
        while ( (j=2*i) <= n) {
            if (j < n && a[j] < a[j + 1]) j++;
            if (x \ge a[j]) break;
            a[i] = a[j]; i = j;
        }
        a[i] = x;
    }
}
```



• Bubblesort: bubblesort()

```
/* Bubble Sort */
void bubblesort (int n, int a[])
{
    int i, j, k;
    int tmp;
    k = n - 1;
    while (k >= 0) {
        i = -1;
        for (i=1; i <= k; i++)</pre>
             if (a[i - 1] > a[i]) {
                 j = i - 1;
                 tmp = a[j]; a[j] = a[i]; a[i] = tmp;
             }
        k = j;
    }
}
```

(1) Performance analysis functionality

In the Performance Analysis window, set the function to be measured. The measured functions are quicksort: quicksort(), heapsort(), bubblesort: bubblesort(), and qsort() as called from quicksort(). For details about these settings, in 4.2 Usage, see (1) Performance analysis functionality.

When the program is executed, the measurement results are output to the Performance Analysis window.

×	° <mark>∎</mark> è _{la} ×	Re Ena set blet				
	Index	Function	Cycle	Count	8	Histogram
	0	_quicksort	992	1	1%	
	1	_qsort	9528	35	13%	>
	2	_heapsort	10784	1	15%	>
	3	_bubblesort	44005	1	64%	>>>>>>

Figure 4-7

Table 4-2 lists the cycle count for each sort from Figure 4-7.

Table 4-2 Measurement results for performance analysis functionality

Sort name	Functions	Cycle count
quicksort	quicksort() +	10520
	qsort()	
heapsort	heapsort()	10784
bubblesort	bubblesort()	44005

(2) Profiler functionality

In the Profiling window, enable **Include Data of Child Functions**. For details about this setting, in 4.2 Usage, see (2) *Profiler functionality*.

When the program is executed, the measurement results are output to the Profiling window.



× -∈ Ena Show Function	s/Variat	bles 💌	Data 🖉							
Function/Variable	F/V	Address	Size	Times	Cycle	Ι.	ο.	Е	Ι.	In
_PowerON_Reset_PC	F	00000800	H'0000001A	1	68230	0	0	261	0	13364
_array	V	00001910	н'00000004	3	0	0	0	0	0	0
_moveLong	F	00001882	H'000003A	3	1158	0	0	240	0	294
_memcpy	F	00001278	H'0000002C	3	1224	0	0	243	0	294
INITSCT	F	00001210	н'00000000	1	1608	0	0	10	0	268
_bubblesort	F	000011D0	H'000003E	1	44005	0	0	0	0	9135
heapsort	F	00001110	H'000000CO	1	10784	0	0	0	0	1920
_qsort	F	00001092	H'0000007E	35	9528	0	0	0	0	1603
_quicksort	F	00001088	H'0000000A	1	10520	0	0	0	0	1734
main	F	00001000	H'0000006C	1	66599	0	0	248	0	13095
List Tree /										

Figure 4-8

Table 4-3 lists the cycle count for each sort from Figure 4-8.

Table 4-3 Profiler functionality measurement results
--

Sort name	Function	Cycle count
quicksort	quicksort()	10520
heapsort	heapsort()	10784
bubblesort	bubblesort()	44005



5. Pseudo-interrupts

The simulator window can be used manually to make an interrupt occur virtually. This allows interrupts that cannot be made to occur, such as those from external devices, to be simulated as well. This chapter explains how to use pseudo-interrupts.

5.1 Usage

To use a pseudo-interrupt, use the trigger button placed in the Trigger window. The following explains how to use the trigger button to make a pseudo-interrupt occur. First, from **CPU** in the **View** menu, choose **Trigger** to display the Trigger window. In the Trigger window, right-click to display the pop-up menu, and choose **Setting** from this menu to display the Trigger Setting dialog box and set the trigger button.

× 1 [±]			
Setting	4	5	б
Button Nu <u>m</u> ber Size	6	17	18
Z Toolbar display		29	30
G Customize toolbar	_10	41	42
Allow Docking Hide	i2	53	54
AT 62 63	64	45	66



In the Trigger Setting dialog box, set the contents of the interrupt to occur when the trigger button is clicked. Note that as many as 256 settings can be performed for a button.

Trigger Setting		<u>?</u> ×
Trigger <u>N</u> o.	-	<u>O</u> K
		<u>C</u> ancel
🔲 <u>E</u> nable		
N <u>a</u> me:	1	
Interrupt Type <u>1</u> :	H'00000000	
Interrupt Type <u>2</u> :	H'0000000	
<u>P</u> riority:	0 💌	

Figure 5-2

The following lists the contents set in the Trigger Setting dialog box:



[Trigger No.]	Selects the trigger button to be specified in detail
[Enable]	Checking this box enables the trigger button.
[Name]	Specifies a name for the selected trigger button; the name will be displayed in the [Trigger] window
[Interrupt Type1]	Sets the following values for each CPU
	• SH-1, SH-2, SH2-DSP, and SH2A-FPU series
	Interrupt vector number
	• SH-3, SH-4 and SH3-DSP series
	INTEVT (H'0 to H'FFF)
	• SH-4A series
	INTEVT (H'0 to H'3FFF)
[Interrupt Type2]	Only selectable for the SH3-DSP series: INTEVT2 (H'0 to H'FFF)
[Priority]	Interrupt priority (0 to 17)
	When 16 is specified, the interrupt is always accepted regardless of the value of the I bit value
	in SR, but is masked by the BL bit in SR. When 17 is specified, the interrupt is always accepted
	regardless of the I and BL bit values in SR.

The Simulator System dialog box can be used to set whether the simulation stops or continues to be executed when an interrupt occurs. From **Simulator** in the **Setup** menu, choose **System** to display the Simulator System dialog box. To continue execution of the simulation, choose **Continue** from the **Execution Mode** drop-down list. To stop the simulation, choose **Stop** from the **Execution Mode** drop-down list, and click the **Detail** button to display the Stoppage Setting dialog box. In this dialog box, select the **Interrupt Exception** checkbox for **Break cause**.

	Stoppage Setting	<u>? ×</u>
Simulator System	Break cause: ☑[interrupt Exception	OK Cancel
System Memory CPU: SH2A-FPU (ROM Disable)		
D'32 <u>E</u> ndian: Big Endian C <u>l</u> ock Rate:	Simulated I/O Address: Enable H'00000000 Execution Mode: Stop Detail. Response: D'40000	
	OK Cancel Apply	

Figure 5-3



5.2 Sample program

The interrupt specification differs between the SH-1, SH-2, and SH-2A series, and the SH-3, SH-4, and SH-4A series, as do the settings in the Trigger Setting dialog box. The following explains how to use pseudo-interrupts using the sample programs for SH-2A and SH-4.

5.2.1 SH-2A

The following explains how to use a pseudo-interrupt in the SH-2A simulator. To receive interrupts in the sample program, use set_imask() in the main function to set the interrupt mask to 0. This means that interrupts with a priority level of 1 or higher are accepted. Then, once the interrupt mask is set, processing is only performed for infinite loops. When an interrupt occurs, the function corresponding to the vector number registered in the interrupt vector table is called.

<pre>#include <machine.h></machine.h></pre>
void main (void)
<pre>set_imask (0);</pre>
<pre>for (;;) { nop();</pre>
}
}

For **Interrupt Type1** in the Trigger Setting dialog box, specify the vector number in the interrupt vector table. In Figure 5-4, H' 00000040 is specified as the vector number, with 5 specified as the interrupt priority. In the sample program, the function corresponding to vector number H' 00000040 is INT_IRQ0().

Trigger Setting		? ×
Trigger <u>N</u> o.		<u>O</u> K
		<u>C</u> ancel
N <u>a</u> me:	1	
Interrupt Type <u>1</u> :	H'000000040	
Interrupt Type <u>2</u> :	H'00000000	
<u>P</u> riority:	5 💌]



Use the Simulator System dialog box to set the simulation to stop when an interrupt occurs. For details about this setting, see *5.1 Usage*. Once the program is executed, click the trigger button set in the Trigger window.

×	л±			
	1	2	3	4
	8	9	10	11
Ш	15	16	17	18



When the trigger button is clicked, a virtual interrupt occurs, and $INT_IRQ0()$, the function corresponding to vector number H'00000040, is called.





5.2.2 SH-4

The following explains how to use pseudo-interrupts for the SH-4 simulator. To have the sample program receive interrupts, use set_cr() in the power-on reset function PowerON_Reset() to set the interrupt mask to 0. This means that interrupts with a priority level of 1 or higher are accepted. Then, once the interrupt mask is set, processing is only performed for infinite loops. When an interrupt occurs, the exception processing handler _IRQHandler is called, and the interrupt processing function corresponding to the INTEVT value in _IRQHandler is called.

<pre>#include <machine.h></machine.h></pre>
#define SR_Init 0x0000000
•
•
void PowerON_Reset (void)
•
<pre>set_cr (SR_Init);</pre>
for (;;) {
nop();
}

For **Interrupt Type1** in the Trigger Setting dialog box, specify the value of INTEVT. In Figure 5-7, H'00000200 is specified as the value of INTEVT, and **4** is specified as the interrupt priority. In the sample program, when the value of INTEVT is H'00000200, an attempt is made to call the interrupt processing function INT_Extern_0000() in _IRQHandler.



APPLICATION NOTE

Trigger Setting	<u>?</u> ×	
Trigger <u>N</u> o.		<u>O</u> K
		<u>C</u> ancel
N <u>a</u> me:	1	
Interrupt Type <u>1</u> :	H'00000200]
Interrupt Type <u>2</u> :	H'00000000	
<u>P</u> riority:	4	



Use the Simulator System dialog box to set the simulation to stop when an interrupt occurs. For details about this setting, see *5.1 Usage*. Once the program is executed, click the trigger button set in the Trigger window.

×	× 1 [±]					
	1	2	3	4		
	8	9	10	11		
	15	16	17	18		

Figure 5-8

When the trigger button is clicked, an interrupt occurs, and the simulation stops during an infinite loop. Here, step-in (F11) can be performed to see that the exception processing handler _IRQHandler is called. Step-in can be repeated to see that the value of INTEVT is H'00000200, and the corresponding interrupt processing function of INT_Extern_0000() is called.



Line	Source A	C.,	. S	Source
68 69	00001000			void PowerON_Reset(void) {
70	00001002			<pre>set_vbr((void *)((_UINT)INTHandlerPRG - INT_OFFSET));</pre>
72	00001000			_INITSCT();
73 74				// _CALL_INIT(); // Remove the comment when you use glot
75 76				// _INIT_IOLIB(); // Enable I/O in the application(both S
77				// errno=0; // Remove the comment when you use errr
79				// srand((_UINT)1); // Remove the comment when you use // _s1ptr=NULL; // Remove the comment when you use strt
81 82				// HardwareSetup(); // Use Hardware Setup
83 84	00001012			set_cr(SR_Init);
85 86				for(;;) {
87 88	0000101E		⇔	nop();
00				
Line	Source A	C	S	Source
158 159				: IRQ ::
160 161				
162				_IRQHandler:
163 164	00000D00		\$	PUSH_EXP_BASE_REG;
165 166	000000D18 00000D1A			mov.l #INTEVT,r0 ; set event address mov.l @r0,r1 ; set exception code
167 168	00000D1C 00000D1E			mov.1 #_INT_Vectors,r0 ; set vector table address add #-(h'40),r1 ; exception code - h'40
169	00000D20			shir2 r1
170	000000D22 00000D24			shir r1 mov.l @(r0,r1),r3 ; set interrupt function addr
172 173	00000D26			; mov.! #_INT_MASK,r0 ; interrupt mask table addr
174 175	000000D28 00000D2A			shlr2 r1 mov.b @(r0,r1),r1 ; interrupt mask
176 177	00000D2C			extu.b r1,r1 ;
178 179	00000D2E 00000D30			stc sr,r0 ; save sr mov.l #(RBBLcIr&IMASKcIr),r2 ; RB,BL,mask clear data
180 181	00000D32 00000D34			and r2,r0 ; clear mask data
182	00000D34			or r1,r0 ; set interrupt mask Idc r0,ssr ; set current status
183 184	00000D38			; Idc.I r3,spc
185 186	00000D3A 00000D3C			mov.1 #int_term,r0 ; set interrupt terminate Ids r0,pr
187 188	00000D3E			; rte
189 190	00000D40			nop
191 192				,
192				
Line	Source A	C.	S.	Source
1 2	00002000			void INT_Extern_0000(void) {
	00002000		4	

Figure 5-9

6. Timer simulation

Timer control is used in many embedded applications. The simulator allows timer functionality to be partially simulated, allowing timer-based application debugging.

Note that the timers supported by the simulator differ depending on the CPU used. For details about the timers supported by the simulator, see $SuperH^{TM}$ RISC engine Simulator/Debugger User's Manual 2.9.2 Control Registers. For simulation of functionality for which terminal I/O such as input capture is used, use a pseudo-interrupt (see 5. Pseudo-interrupts). Note that the type and usage of device timers differ depending on the device used. For details about timers, see the hardware documentation for the device used.

6.1 Usage

To simulate timer control in the simulator, the peripheral function simulation module for the timer used must be registered. Note that if the register address of the device used differs from the default simulator settings, the register address for the peripheral function simulation module needs to be changed. The following explains how to register the peripheral function simulation module, and change the register address of the peripheral function simulation module.

(1) Registering the peripheral function simulation module

The peripheral function simulation module can be registered from the Set Peripheral Function Simulation dialog box displayed when the simulator starts up. From the Set Peripheral Function Simulation dialog box, select the checkbox of the timer to be used, from the **Peripheral Functions** list.

Set	Peripheral Fur	oction Simulation		? ×
f	Peripheral <u>F</u> unctio	ns:		
	Module Name	File Name		Enable All
	СМТ	c:¥program files¥renesas¥hew_l)01¥tools¥renesas¥	<u>D</u> isable All
		_		De <u>t</u> ail
	•		Þ	
E	eripheral Clock F	late: 1		
Γ	Don't <u>s</u> how this	s dialog box	ОК	Cancel



The following explains the contents set in each item in the Set Peripheral Function Simulation dialog box.

[Peripheral Function	on] Shows infor	mation on the peripheral function simulation modules.		
	[Module Name]	Names of peripheral functions to be simulated		
	[File Name]	Names of files holding peripheral function simulation modules		
	Check the checkbox	under [Module Name] to register the corresponding peripheral function		
	simulation module a	nd make it available.		
[Enable All]	Enables all j	peripheral functions.		
[Disable All] Disables all	peripheral functions.		
[Detail]	Opens the []	Peripheral Module Configuration] dialog box, allowing you to view		
		on the corresponding peripheral function, and change the address where it		
[Peripheral (between the peripheral clock and the internal clock (the number of cycles of		
	The clock ra	ate setting can be selected as 1, 2, 3, 4, 6, 8, 12, 16, 24, or 32.		
[Disable All [Detail]	Enables all j Disables all Opens the [l information starts and th Clock Rate] The ratio the internal	peripheral functions. peripheral functions. Peripheral Module Configuration] dialog box, allowing you to view on the corresponding peripheral function, and change the address where it he interrupt-source information.		

When the **Don't show this dialog box** checkbox is selected in the Set Peripheral Function Simulation dialog box, the Set Peripheral Function Simulation dialog box is no longer displayed when the simulator starts up. To have it displayed again, from the **Setup** menu, choose **Options** to display the Options dialog box, and then select the **Confirmation** tab. When the **Display Set Peripheral Function Simulation dialog box at start up** checkbox is selected in the **Display confirmation**



dialogs for list in the **Confirmation** tab, the Set Peripheral Function Simulation dialog box is displayed when the simulator starts up.

Options			? ×
Build Editor Debug Workspace Confirmation Network			
Display confirmation dialogs for:			
		S-4 AU	
✓Delete All PA Ranges		<u>S</u> et All	
✓ Delete All PA Ranges at Loading		<u>C</u> lear All	
✓Delete all watches ✓Delete Event			
✓ Delete Event ✓ Delete file from project			
✓ Delete PA Range			
✓ Delete Template			
✓ Delete test from test suite			
✓ Display all array elements in Watch/Locals.			
✓Display Set Peripheral Function Simulation dialogbox at start up.			
☑ Display Splash Screen			
Display static member on the variable expansion in Watch/Locals	š.		
✓Display Welcome Dialog			
Do not show Debug only workspace browse dialog on download			
Download modules after build			
External editor change warning	-		
	OK	Cano	el

Figure 6-2

(2) Changing the address of the peripheral function simulation module

The register address of the peripheral function simulation module can be changed from the Peripheral Module Configuration dialog box. In the **Peripheral Function** field in the Set Peripheral Function Simulation dialog box, select the peripheral function for which the register address is to be changed, and then click the **Detail** button to display the Peripheral Module Configuration dialog box. In the Peripheral Module Configuration dialog box, set the register address of the device used in **Begin Address (Register**).



Peripheral Modul	e Configuration	?×
Address Interru	pt	
<u>M</u> odule:		
CMT		
<u>B</u> egin Address [,]	(Register):	
H'FFFEC000	•	
<u>R</u> egister Addre	ss:	
Register	Address	
CMSTR	FFFEC000	
CMCSR0	FFFEC002	
CMCNT0 CMCOR0	FFFEC004 FFFEC006	
CINCORU	FFFECUUD	
,		
ОК	Cancel	Apply
01		======

Figure 6-3

The following items are displayed and set in the Address tab of the Peripheral Module Configuration dialog box.

[Module]

[Begin Address(Register)]

[Register Address]

Name of the peripheral function supported by the selected peripheral function simulation module Start address of the peripheral function selected in [Module] Names and addresses of registers of the peripheral function selected in [Module]. It is not possible to change the register addresses.

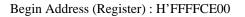
Note that interrupt cause information for peripheral functions can be viewed and changed in the **Interrupt** tab of the Peripheral Module Configuration dialog box.

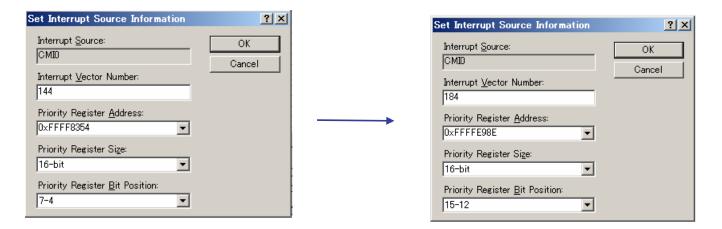
Peripheral Module Configuration	
Address Interrupt	Set Interrupt Source Information
Interrupt Source Information: Interrupt So Vector Num Priority Register CMID 144 FFFF8354/7-4 Double-clicking	Interrupt <u>S</u> ource: CMID Interrupt <u>Vector Number:</u> 144 Priority Register <u>A</u> ddress: 0×FFFF8354 Priority Register Size: 16-bit Priority Register <u>B</u> it Position:
OK Cancel Apply	7-4

[Interrupt Vector Number] [Priority Register Address] [Priority Register Size] [Priority Register Bit Position] Number of Interrupt Vector Address of Priority Register Size of Priority Register Bit Position of Priority Register

[Example of setting SH7085]

eripheral Mod	ule Configuration	? ×	(Peripheral Modu	le Configuration	<u>?</u> ×
Address Intern	rupt			Address Interru	ipt]	
<u>M</u> odule: CMT <u>B</u> egin Addres	s(Register):			<u>M</u> odule: CMT <u>B</u> egin Address	(Register):	
H'FFFF83D0 <u>R</u> egister Addr	ress:			H'FFFFCE00 <u>R</u> egister Addre		
Register CMSTR CMCSR0 CMCNT0 CMCOR0	Address FFFF83D0 FFFF83D2 FFFF83D4 FFFF83D6			Register CMSTR CMCSR0 CMCNT0 CMCOR0	Address FFFFCE00 FFFFCE02 FFFFCE04 FFFFCE06	
ОК	Cancel	Apply		ОК	Cancel	Арру





Interrupt Vector Number : 184 Priority Register Address : 0xFFFFE98E Priority Register Bit Position : 15 - 12

6.2 Sample program

The following explains how to use the sample program for timer simulation. In the sample program, interrupts based on a compare match timer (CMT) are used to increment a variable at a fixed interval. The CMT is a timer for which an interrupt occurs when the timer count and compare match value match.

The following gives a detailed explanation of the sample program. Note that the structure corresponding the register for the interrupt controller (INTC) and CMT peripheral function module is defined in iodefine.h.

- main function
 - (1) The count variable for counting is initialized.
 - (2) set_imask() is used to set the interrupt mask to 0 so that interrupts can be received.
 - (3) The CMT interrupt priority is set.
 - (4) The CMT compare match value is set.
 - (5) CMT interrupt occurrence is permitted.
 - (6) The CMT timer count is started.
 - (7) An infinite loop occurs, and the CMT interrupt continues to be received.

```
#include <machine.h>
#include "iodefine.h"
unsigned int count;
void main (void)
{
    count = 0;
                               /* (1) */
    set_imask (0);
                               /* (2) */
    INTC.IPR08.WORD = 0xF000; /* (3) */
    CMT0.CMCOR = 0 \times 0 FFF; /* (4) */
    CMT0.CMCSR.WORD = 0x0040; /* (5) */
    CMT.CMSTR.WORD = 0x01; /* (6) */
                                /* (7) */
    for (;;) {
        nop();
    }
```

- Timer interrupt function
 - (1) The count variable for counting is incremented.
 - (2) Interrupt is permitted again, since timer interrupt occurrence is prohibited when the CMT timer count matches the compare match value. Note that the CMT timer count returns to 0 when it matches the compare match value, and that incrementing is restarted.

When starting the simulator, register the peripheral function simulation module in the Set Peripheral Function Simulation dialog box. For details about registration, see *6.1 Usage*.

Check the Watch window for the value of the count variable during program execution. From **Symbol** in the **View** menu, choose **Watch** to display the Watch window. In the Watch window, register the symbol for the count variable, and then enable auto-update for the registered count symbol. When the sample program is executed, the count variable is incremented in the Watch window (Figure 6-4). This reflects the fact that timer interrupts are occurring regularly.

×	R R 🗖 🖊 🗙	(🛃 📌 🖻						
	Name	Value	Туре					
	R count	H'000000e6 { FFF80000 }	(unsigned int)					
	Watch1 / Watch2 / Watch3 / Watch4 /							

Figure 6-4

7. Eventpoints

Breakpoints (software breakpoints) set from the editor window stop execution of user programs immediately before the instruction at the set address is executed. In contrast, eventpoints can be set to provide more advanced program stop conditions than software breakpoints. For example, breakpoints can be set to stop a program when conditions such as the following are satisfied:

- (1) The value of a certain register or piece of data reaches a specified value
- (2) The specified data area is accessed
- (3) The execution cycle count reaches the specified cycle count

Note that for an eventpoint, instead of stopping the program when a set condition is satisfied, data can be written from a file to memory or from memory to a file, and an interrupt can occur.

7.1 Usage

Eventpoints can be set by choosing **Eventpoints** from **Code** in the **View** menu, to display the Eventpoints window. Right-click in the Eventpoints window to display a pop-up menu, and then choose **Add** to display the Select Break Type dialog box. In the Select Break Type dialog box, set the conditions for stopping user programs, and the operations performed when conditions are satisfied. When settings are completed, if **Stop** is specified for **Action type**, the contents of the set eventpoint are displayed in the **Software Break** tab of the Eventpoints window. If anything other than **Stop** is specified, the contents of the set eventpoint are displayed in the **Software Event** tab of the Eventpoints window. For details about the items that can be set for eventpoints, see *SuperHTM RISC engine Simulator/Debugger User's Manual - 3.4 Using the Simulator/Debugger Breakpoints*.

×	₽ Z	×							
	Type	State	Condition	Action					
	<u>A</u> dd. Edi <u>t</u> , E <u>n</u> at			─/	<u>B</u> reak typ		<u>D</u> etail	_1	<u>?</u> ×
	Di <u>s</u> a <u>D</u> ele D <u>e</u> le				PC Breal <u>A</u> ction ty Stop		D <u>e</u> tail		Cancel
	 <u>C</u> los	o Source e File e All <u>F</u> iles		_					
		bar display tomize toolbar							
	Allow Hide	w Docking	/ Software Event /						

Figure 7-1

The following lists the information displayed in the Eventpoints window:



[Type] Break types	
[BP]:	PC break
[BA]:	Access break
[BD]:	Data break
[BR]:	Register break (register name)
[BS]:	Sequential break
[BCY]:	Number-of cycles break
[State] Whether the	e breakpoint is enabled or disabled.
[Enable]:	Valid
[Disable]:	Invalid
	ion that causes a break. The contents to be displayed depend on the type of the break. When the
• •	f the break is BR, the register name is displayed, and when the type of the break is BCY, the
numbe	er of cycles is displayed.
BP:	PC = Program counter (Corresponding file name, line, and symbol name)
BA:	Address=Address (Symbol name)
BD:	Address=Address (Symbol name)
BR:	Register=Register name
BS:	PC = Program counter (Corresponding file name, line, and symbol name)
BCY:	Cycle = Number of cycles (displayed in hexadecimal)
[Action] Operation	of the simulator/debugger when a break condition is satisfied.
[Stop]:	Execution halts
[File Input]:	(file name) [File state]: Memory data is read from file
[File Output]:	(file name) [File state]: Memory data is written to file
[Interrupt]:	(Interrupt type/priority): Interrupt processing.
	Only for the SH3-DSP, (Interrupt type 1, interrupt type 2/priority) is displayed.

7.2 Sample program

The following explains how to use the sample program for using eventpoint break data. In the sample program, the global variable x is incremented in a loop. This program is used as an example in which the program is stopped when the global variable x reaches a specific value.

The sample program is as follows:

```
volatile long x;
void main (void)
{
    int i;
    x = 0;
    for (i=0; i < 10000; i++ ) {
        x++;
    }
}
```

The following explains how to stop a program when the value of the global variable x reaches 5000. First, from the Select Break Type dialog box, choose **break data** for **Break type**, and then click the **Detail** button to display the Set Data Break Condition dialog box. Then, set the items in the Set Data Break Condition dialog box as follows:

- Address: _x
- Option: Equal
- **Data**: D'5000
- Data mask: Cleared
- Size: Long word



× • × =					
Type State Conditi	on Action	Select Break Ty	уре		? ×
<u>A</u> dd Edi <u>b</u> E <u>n</u> able Digable Delete Dglete All		Break type: Break Data Action type: Stop			OK Cancel
Go to Source		Set Data Break	Condition		? ×
<u>C</u> lose File Close All <u>F</u> iles		<u>A</u> ddress:		- 🔊	ок
		Option:	Equal	<u> </u>	Cancel
✓ Toolbar display Customize toolbar		<u>D</u> ata:	D'5000		
Allow Docking		🔲 Data <u>m</u> ask:	H'FFFFFFFF		
Hide		<u>S</u> ize:	Long word	•	
Software Break / Software B	Event /	Sign:	Signed	V	

Figure 7-2

The set eventpoint is displayed in the Eventpoints window.

×	1 1	×			
Ш	Type	State	Condition	Action	
	BD	Enable	Address=FFF80000(_x)	Stop	
		oftware Break 🖌	Software Event /		



When the sample program is executed, it is stopped immediately after the value of the variable x reaches 5000.

Line	Source A	C	S	Source
27				volatile long x;
28	00001000			void main(void)
30	00001000			{
31				int i;
32	00001000			x = 0;
	00001006			for(i=0; i < 10000; i++) {
	0000100A		⇔	x++;
36	00001014			}

Figure 7-4



8. Virtual I/O panels

Virtual I/O panels can be used to check data visually, by placing virtual buttons and LEDs on the simulator window. Virtual I/O panels can use the following GUI components:

- Buttons
- Buttons can be clicked to input data to the specified address, or make a virtual interrupt occur.
- Labels

Specific strings can be displayed and removed when specified values are written to a specified address or bit.

• LEDs

Specified colors can be displayed (in place of an LED light) when specified values are written to a specified address or bit.

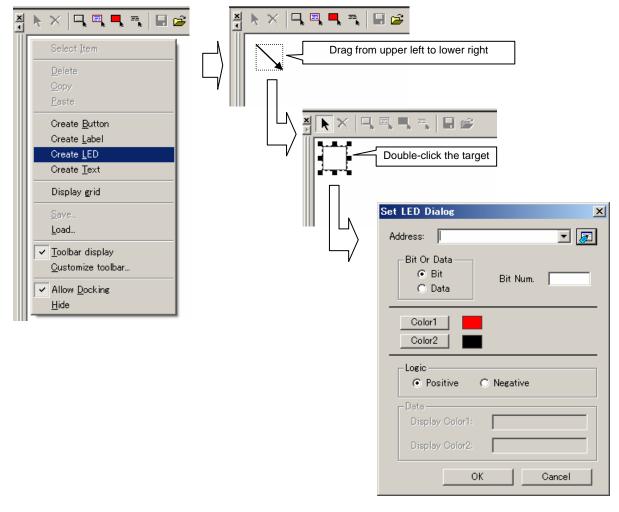
• Text

Specified strings are always displayed.

8.1 Usage

Virtualized output panels are used by being placed in the GUI I/O window. The following explains how to set up a panel.

From **Graphic** in the **View** menu, choose **GUI I/O** to display the GUI I/O window. To place the panel, right-click in the GUI I/O window to display a pop-up menu, and then choose the item for the panel to be created. The mouse cursor will change to a plus sign (+), at which point the panel to be created can be dragged between from the upper left to the lower right to create an output frame. As output frames are selected once they are created, the created output frame can be double-clicked to display a dialog box for setting the panel display contents.



The panel settings for the GUI I/O window are saved independently of the workspace, which means that they need to be saved to a file separate from the workspace and project. To save panel settings, from the pop-up menu, choose **Save** to display the Save GUI I/O Panel File dialog box. In the Save GUI I/O Panel File dialog box, specify the save destination file name, whose extension is .pnl by default. To load a file to which panel settings have been saved, from the pop-up menu in the GUI I/O window, choose **Load**, and then select the saved file.

8.1.1 Button display

To create a button panel, from the pop-up menu, choose **Create** Button. Once the button output frame is created, select the output frame, and double-click to display the Set Button Dialog dialog box.

Set Button Dial	og	x
Button Name:	Button	
Select Button T		
-Input		
Туре:	Address	
Address:		
Data:	Length: Byte 💌	
Bit Mask	se	
Mask Data	ar 🖉	
Bit No.: 7	6 5 4 3 2 1 0	
Bit Symbol:	💌 Bit Value: 0 💌	[
- Interrupt		
Interrupt Type1	H'0000000	
Interrupt Type2	H'0000000	
Priority:	0	
	OK Cancel	

Figure 8-2

Perform the following settings in the Set Button Dialog dialog box.



Button Name	Specify the name disp	played in the button panel.			
Select Button Type	Select Input, Interrupt, or Input and Interrupt.				
Input: specify the follow	wing when Input or Inp	out and Interrupt is selected for Select Button Type:			
Туре		nge the specified address to a specific value, or Address & Bit No. to and change the value of the specified address, when the button panel			
Address	Specify the address for	or changing the value when the button panel is clicked.			
Data		eplace the specified address when the button panel is clicked. Specify cted for Type , or Address & Bit No. is selected and a bitwise mask is			
Length	Specify the size of the	e data.			
Bit Mask	Set the following whe	en Address & Bit No. is set for Type:			
	When not using a bitw bitwise mask, set Mas	vise mask, clear the Not use checkbox, and set Bit No . To use a sk Data .			
Mask Dat	-	the specified address to the bits of the value specified for the data, or the specified mask value is 1.			
Bit No	For bits 7 to 0, specify	*			
210110		ose value is to be set to 0.			
		ose value is to be set to 0.			
		ose value is to be set to 1.			
Bit Symbol at	1 1	input for SH microcomputer simulators.			
Interrupt					
Interrupt	[Interrupt Type1]	Sets the following values for each CPU			
	[interrupt Type1]	• SH-1, SH-2, SH2-DSP, and SH2A-FPU series			
		Interrupt vector number			
		• SH-3, SH-4 and SH3-DSP series			
		INTEVT (H'0 to H'FFF)			
		• SH-4A series			
		INTEVT (H'0 to H'3FFF)			
	[Interrupt Type2] [Priority]	Only selectable for the SH3-DSP series: INTEVT2 (H'0 to H'FFF) Interrupt priority (0 to 17)			
	[i nonty]	When 16 is specified, the interrupt is always accepted regardless of the value of the I bit value in SR, but is masked by the BL bit in SR. When 17 is specified, the interrupt is always accepted regardless of the I and BL bit values in SR.			

8.1.2 Label display

To create a label panel, from the pop-up menu, choose **Create Label**. Once the label output frame is created, select the output frame, and double-click to display the Set Label Dialog dialog box.

Set Label Dialog	×
Address:	
Bit Or Data © Bit © Data Bit Num.	
Name1: Name2:	
Logic © Positive © Negative	
Data Display Name1: Display Name2:	
OK Cancel	

Figure 8-3

Perform the following settings in the Set Label Dialog dialog box:

(1) When the **Bit** radio button is selected for **Bit Or Data**:

Address	Set the start address for byte data.
Bit Num	Set the bit position (0 to 7) from the LSB in the byte data.
Name1	Set the string to be displayed.
Name2	Set the string to be displayed.
Logic	The label panel is displayed as follows:
	If the Positive radio button is selected, Name1 is displayed when the set bit is 1, and Name2 is
	displayed when the set bit is 0.
	If the Negative radio button is selected, Name2 is displayed when the set bit is 1, and Name1 is
	displayed when the set bit is 0.
1 /1 D /	

(2) When the **Data** radio button is selected for **Bit Or Data**:

Address Set the start address for byte data

- Name1 Set the string to be displayed.
- Name2 Set the string to be displayed.
- **Data** The label panel is displayed as follows:

Name1 is displayed if the data specified in Address is the value set for Display Name1, and Name2 is displayed if the data specified in Address is the value set for Display Name2. Note that no string is displayed for values other than those set for Display Name1 and Display Name2.

8.1.3 LED display

To create an LED panel, from the pop-up menu, select **Create LED**. Once the LED output frame is created, select the output frame and double-click to display the Set LED Dialog dialog box.

Set LED Dialog 🛛 🗙
Address:
Bit Or Data © Bit © Data
Color1 Color2
Cogic Positive C Negative
Data Display Color1: Display Color2:
OK Cancel

Figure 8-4

Perform the following settings in the Set LED Dialog dialog box:

(1) When the **Bit** radio button is selected for **Bit Or Data**:

Address	Set the start address for byte data.
Bit Num	Set the bit position (0 to 7) from the LSB in the byte data.
Color1	Select the color to be displayed.
Color2	Select the color to be displayed.
Logic	The label panel is displayed as follows:
	If the Positive radio button is selected, Color1 is displayed when the set bit is 1, and Color2 is
	displayed when the set bit is 0.
	If the Negative radio button is selected, Color2 is displayed when the set bit is 1, and Color1 is
	displayed when the set bit is 0.

(2) When the **Data** radio button is selected for **Bit Or Data**:

Address	Set the start	address for	byte data.
---------	---------------	-------------	------------

- **Color1** Select the color to be displayed.
- **Color2** Select the color to be displayed.
- **Data** The label panel is displayed as follows:

Color1 is displayed if the data specified in **Address** is the value set for **Display Color1**, and **Color2** is displayed if the data specified in **Address** is the value set for **Display Color2**. Note that nothing is displayed for values other than those set for **Display Color1** and **Display Color2**.

8.1.4 Text display

To create a text panel, from the pop-up menu, choose **Create Text**. Once the text output frame is created, select the output frame, and then double-click to display the Set Text Dialog dialog box.



Set Text Dialog			×
Text: Text]
Font Font Name:	FixedSys		
Font Size:	11	Font	
-Color			
Text			
Back			
	OK	Cancel	

Figure 8-5

Perform the following settings in the Set Text Dialog dialog box:

TextSet the string to be displayed in the text panel.FontClick the Font button to set the font and font size of the string displayed.ColorClick the Text button to set the color of the string displayed. Click the Back button to set the background color of the text panel.

8.2 Sample program

The following uses a sample program to explain how to use virtual I/O panels.

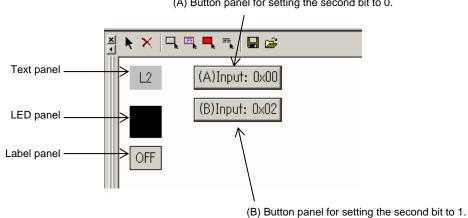
The virtual I/O panel for the sample program contains an LED panel that is lit based on the value of a given variable, and a button panel to change the value of the variable. The sample program shows an example in which the value of a variable is changed by the button panel, and the results are indicated by a lit LED panel. Note that in the sample program, using flashing LEDs for variable access implies I/O port access. On actual hardware, I/O port initialization processing is required when displaying LEDs or performing input from buttons.

The sample program is as follows. The main function performs only an infinite loop.

```
#include <machine.h>
volatile unsigned char port;
void main (void)
{
    Port = 0;
    for (;;) {
        nop();
      }
}
```

Place the output frame of the virtual I/O panel as follows in the GUI I/O window. Place an LED panel and label panel for which the display changes according to the second bit in the variable port. Also place a button panel that changes the second bit of the variable port to 0, and a button panel that changes the second bit of the variable port to 1.





(A) Button panel for setting the second bit to 0.



Set the contents of each panel as follows.

Text panel •

Create a text panel to display the string L2.

Set Text Dialog		×
Text: L2		
- Font		
Font Name:	FixedSys	
Font Size:	11	Font
Color		
Text		
Back		
	ОК	Cancel

Figure 8-7



• LED panel

Create an LED panel that changes color based on the value of the second bit of the variable port. Specify 1 for **Bit Num**, and set positive logic for **Logic**.

Set LED Dialog X
Address: _port 💽 👰
Bit Or Data © Bit © Data Bit Num. 1
Color1 Color2
O Negative
Display Color1:
Display Color2:
OK Cancel

Figure 8-8

• Label panel

Create a label panel that changes its displayed string based on the value of the second bit in the variable port. Specify 1 for **Bit Num**, and set positive logic for **Logic**.

Set Label Dialog 🛛 🔀
Address: _port 💌 🗾
Bit Or Data © Bit © Data Bit Num. 1
Name1: ON Name2: OFF
Logic Positive O Negative
Data Display Name1:
Display Name2: OK Cancel

Figure 8-9

• Button panel

KENESA

- Create the following button panels for setting the data in the variable port:
- (A) A button panel that sets the second bit of the variable port to 0.
- (B) A button panel that sets the second bit of the variable port to 1.

(A)	(B)
Set Button Dialog 🛛 🗙	Set Button Dialog
Button Name: (A)Input: 0x00	Button Name: (B)Input: 0x02
Select Button Type Input C Interrupt C Input and Interrupt	Select Button Type Input C Interrupt C Input and Interrupt
_ Input	
Type: Address & Bit No.	Type: Address & Bit No.
Address: _port 💌 😿	Address: _port 💌 💽
Data: Length: Byte	Data: Length: Byte
Bit Mask	Bit Mask
7 6 5 4 3 2 1 0 Bit No: - - - - - - 0 - 1 - - - - - - - - - - - - - - - - </th <th>7 6 5 4 3 2 1 0 Bit No: - - - - - - - 1 0</th>	7 6 5 4 3 2 1 0 Bit No: - - - - - - - 1 0
Bit Symbol: 🔽 Bit Value: 0 💌	Bit Symbol: 💽 Bit Value: 0 💌
_ Interrupt	_ Interrupt
Interrupt Type1: H'00000000	Interrupt Type1: H'00000000
Interrupt Type2: H'00000000	Interrupt Type2: H'00000000
Priority: 0	Priority: 0
OK Cancel	OK Cancel

Figure 8-10

To prevent the GUI I/O window from updating during program execution, execute cache off in the Command window (Figure 8-11). To display the Command window, choose **Command Line** from the **View** menu.

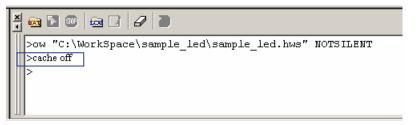
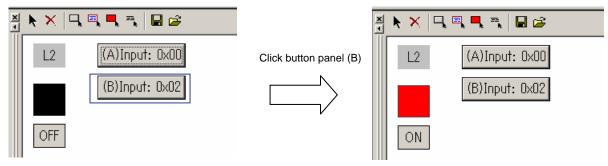


Figure 8-11

Click the (B) button panel during program execution to set the second bit of the variable port to 1. This will cause the color of the LED panel to change from black to red. Likewise, the string in the label panel will change from OFF to ON.





Click the (A) button panel during program execution to set the second bit of the variable port to 0. This will cause the color of the LED panel to change from red to black. Likewise, the string in the label panel will change from ON to OFF.

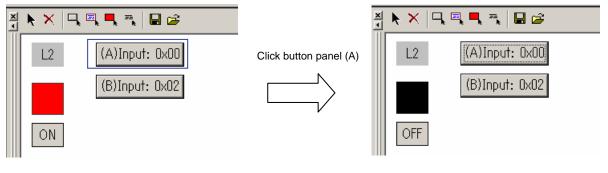


Figure 8-13



Website and Support <website and support,ws>

Renesas Technology Website

http://japan.renesas.com/

Inquiries

http://japan.renesas.com/inquiry

csc@renesas.com

Revision Record <revision history,rh>

		Description	
Rev.	Date	Page	Summary
1.00	Apr.01.08		First edition issued

Notes regarding these materials

- 1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
- Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
- 3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
- 4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, 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 products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
- 5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
- 6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
- 7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
 - Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below: (1) artificial life support devices or systems
 - (2) surgical implantations

8

- (3) healthcare intervention (e.g., excision, administration of medication, etc.)
- (4) any other purposes that pose a direct threat to human life

Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.

- 9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
- 10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas 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 applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
- 11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
- 12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
- 13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.

© 2008. Renesas Technology Corp., All rights reserved.