User’s Manual

RENESAS

RX Family Simulator/Debugger V.1.01

User’s Manual

Renesas Microcomputer Development Environment System

Renesas Electronics
www.renesas.com

Rev.1.00 2010.04

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.

Trademarks

Microsoft, MS-DOS, Windows, Windows NT are registered trademarks of Microsoft Corporation. Visual SourceSafe is a trademark of Microsoft Corporation.
IBM is a registered trademark of International Business Machines Corporation.
All brand or product names used in this manual are trademarks or registered trademarks of their respective companies or organizations.

About This Manual

This manual describes the HEW system. This manual is composed of two parts. HEW part describes information on the basic “look and feel” of the HEW and customizing the HEW environment and detail the build. Figure in the HEW part are those of the SH series. Simulator/Debugger part describes Debugger functions of the High-performance Embedded Workshop.

This manual does not intend to explain how to write C/C++ or assembly language programs, how to use any particular operating system or how best to tailor code for the individual devices. These issues are left to the respective manuals.

Document Conventions

This manual uses the following typographic conventions:

Table 1 Typographic Conventions

Convention	Meaning
[Menu->Menu Option]	Bold text with ‘->’ is used to indicate menu options (for example, **[File->Save As...]**).
FILENAME.C	Uppercase names are used to indicate filenames.
“enter this string”	Used to indicate text that must be entered (excluding the “” quotes).
Key + Key	Used to indicate required key presses. For example, **CTRL+N** means press the **CTRL** key and then, whilst holding the **CTRL** key down, press the **N** key.
➲ (The “how to” symbol)	When this symbol is used, it is always located in the left hand margin. It indicates that the text to its immediate right is describing “how to” do something.

For inquiries about the contents of this document or product, email to your local distributor.

Renesas Tools Homepage http://www.renesas.com/tools

Contents

Section 1 Overview

The simulator/debugger is a powerful development environment tool for embedded applications to run on Renesas Electronics microcomputers.

The simulator/debugger is used with the High-performance Embedded Workshop (HEW). The HEW provides a graphical user interface that eases the development and debugging of applications written in the C/C++ programming languages or assembly language for Renesas Electronics microcomputers. Its aim is to provide a powerful yet intuitive way of accessing, observing and modifying the debugging platform on which the application is running.

READ the simulator/debugger and HEW help information before using the simulator/debugger.

Section 2 Simulator/Debugger Functions

This section describes the functions of the RX600 series simulator/debugger.

2.1 Features

- Since the simulator/debugger runs on a host computer, software debugging can start without using an actual user system, thus reducing overall system development time.
- The simulator/debugger performs a simulation to calculate the number of instruction execution cycles for a program and time taken by instruction execution, thus enabling performance evaluation without using an actual user system.
- The simulator/debugger provides pseudo-interrupt and I/O-simulation functions for simple system-level simulation.
- The simulator/debugger offers the following functions that enable efficient program testing and debugging.
  - The ability to handle all of the RX600 series CPUs
  - Functions to stop or continue execution when an error occurs during user program execution
  - Profile data acquisition and function-unit performance measurement
  - A comprehensive set of break functions
  - Functions to set or edit memory maps
  - Functions to display function call history
  - Coverage information is displayed in the C/C++ or assembly-source level
  - Visual debugging functions provided through the display of images or waveforms
- The breakpoints, memory map, performance, and trace can be set through the dialog boxes under Windows®. Environments corresponding to each memory map of the RX600 series microcomputers can be set through the dialog box.
  - Intuitive user interface
  - Online help
  - Common display and operability

2.2 Target User Program

Load modules in the Elf/Dwarf2 format can be symbolically debugged with the simulator/debugger. Load modules in other formats can be downloaded, and their instructions can be executed; however, they cannot be symbolically debugged. For details, refer to the High-performance Embedded Workshop User's Manual.

2.3 Range

The simulator/debugger provides simulation functions for the RX600 series microcomputers.

The simulator/debugger supports the following RX600 series microcomputer functions:

- All CPU instructions
- Exception processing
- Registers
- All address space

The simulator/debugger does not support the following RX600 series MCU functions. Programs that use these functions must be debugged with the RX600 series emulator.

Item	Remarks
Low power state	Simulation is stopped on the execution of a WAIT instruction.
Non-maskable interrupt (NMI)	
Reception of an interrupt during execution of any of the following instructions: (RMPA, SCMPU, SMOVF, SMOVB, SMOVU, SSTR, SUNTIL, SWHILE)	The interrupt is accepted when execution of the instruction is completed.
Values in memory and registers that become undefined after the execution of instructions	
Lower-order 16 bits of the accumulator (ACC)	

2.4 Memory Management

Memory Map Specification: A memory map is used to calculate the number of memory access cycles during simulation. The following items can be specified:

- Memory type
- Start and end addresses of the memory area
- Number of memory access cycles
- Memory data bus width
- Endian

On the memory map, the endian is only specifiable for the external area.
For the internal ROM area and internal RAM area, the [Endian] specified on the [CPU Configuration] tabbed page of the [Set Simulator] dialog box (displayed when the simulator debugger is started up) applies.

For details, refer to section 3.3.3, Modifying the Memory Map and Memory Resource Settings.

Memory Resource Specification: A memory resource must be specified to load and execute a user program. The following items can be specified:

- Start address
- End address
- Access type

The access type is readable/writable, read-only, or write-only.

Since an error occurs if the user program attempts an illegal access (for example, trying to write to read-only memory), such an illegal access in the user program can be easily detected.

For details on memory resource setting, refer to section 3.3.3, Modifying the Memory Map and Memory Resource Settings.

2.5 Instruction-Execution Reset Processing

Counting by the simulator/debugger of executed instructions, cycles for instruction execution, and time taken by instruction execution is reset in the following cases.

- The program counter (PC) is modified after the instruction simulation stops and before it restarts.
- The Run command to which the execution start address has been specified is executed.
- Initialization is performed or the program is loaded.

2.6 Exception Processing

The simulator/debugger detects the generation of exceptions in the RX600 series and simulates exception processing. Accordingly, simulation can be performed even when an exception occurs.

The simulator/debugger simulates exception processing with the following procedures.

1. Detects an exception during instruction execution.
2. The PC and PSW are saved in the dedicated registers (for the fast interrupt) or the stack area (for a normal interrupt). If an error occurs when saving, the simulator/debugger stops exception processing, shows that the exception processing error has occurred, and returns to the command input wait state.
3. Bits of the PSW are set as follows.
   U = 0, I = 0, PM = 0
4. Reads the start address from the vector address corresponding to the vector number. If an error occurs when reading, the simulator/debugger stops exception processing, shows that the exception processing error has occurred, and returns to the command input wait state.
5. Starts instruction execution from the start address.

2.7 Endian

2.7.1 Endian of the CPU

The endian of the CPU can be specified in the [CPU Configuration] tabbed page in the [Set Simulator] dialog box, which is displayed at initiation of the simulator debugger. The endian of the CPU are applied to the internal ROM and the internal RAM. For details, refer to section 3.3.1, Setting the Endian and Frequency of CPU.

2.7.2 Endian of the External Memory Area

The endian of the external memory area can be set in the [Set Memory Map] dialog box. For details, refer to section 3.3.4, Set Memory Map Dialog Box.

2.8 Simulation of Peripheral Functions

2.8.1 Timer

(1) Supported Range

The RX600 series simulator/debugger supports a total of four compare match timer (CMT) channels, i.e. two CMT units (unit 0 and unit 1), each with two 16-bit timers.

(2) Control Registers

Table 2.1 lists the control registers of the CMT that are supported by the simulator/debugger.

In access to control registers, ensure that the unit of access is the same as the size of the register.

Table 2.1 Control Registers of the CMT Supported by the Simulator/Debugger

Peripheral Module	Unit	Supported Control Register	Support
CMT	Unit 0	CMSTR0	○
		CMCR0	○
		CMCNT0	○
		CMCOR0	○
		CMCR1	○
		CMCNT1	○
		CMCOR1	○
	Unit 1	CMSTR1	○
		CMCR2	○
		CMCNT2	○
		CMCOR2	○
		CMCR3	○
		CMCNT3	○
		CMCOR3	○

Note: ○: Supported

The addresses of the control registers can be referred to or modified in the [Peripheral Module Configuration] dialog box. Refer to section 3.4, Simulating Peripheral Functions, for details on this dialog box.

2.8.2 Serial Communications Interface

(1) Supported Range

The RX600 series simulator/debugger supports a total of seven serial communications interface (SCI) channels. Table 2.2 lists the supported SCI functions.

Table 2.2 SCI Functions Supported by the Simulator/Debugger

Item			Support
Serial communications mode	Asynchronous or clock synchronous		○
	Smart card interface		—
Clock sources for the on-chip baud rate generator	PCLK clock		○
	PCLK/4, PCLK/16, and PCLK/64		—
Full-duplex communications			○
Interrupt sources	Transmit-end, transmit-data-empty, receive-data-full, and receive error		○
Asynchronous mode	Data length	7 or 8 bits	○
	Transmission stop bit	1 or 2 bits	○
	Parity	Even, odd, or none	○
	Receive error detection	Parity, overrun, and framing errors	○
	Break detection		—
	Clock source	Internal clock	○
		External clock or transfer rate clock input from TMR	—
Clock synchronous mode	Data length	8 bits	○
	Receive error detection	Overrun errors	○

Note: ○: Supported
—: Not supported

(2) Control Registers

Table 2.3 shows control registers of the SCI supported by the simulator/debugger.

In access to control registers, ensure that the unit of access is the same as the size of the register.

Table 2.3 Control Registers of the SCI Supported by the Simulator/Debugger

Peripheral Module	Channel	Supported Control Register	Support
SCI	0 to 6	SMR	Δ
		BRR	O
		SCR	Δ
		TDR	O
		SSR	Δ
		RDR	O
		SCMR	Δ
		SEMR	Δ

Note: O: Supported
Δ: Partly supported (bits for the function described in section 2.8.2 (1), Supported Range)

The addresses of the control registers can be referred to or modified in the [Peripheral Module Configuration] dialog box. Refer to section 3.4, Simulating Peripheral Functions, for details on this dialog box.

(3) Input and Output of Data

For the simulator/debugger, some pins are allocated to memory as virtual ports. Programs being debugged and debuggers are only able to access those pins through the virtual ports. Table 2.4 lists the addresses of virtual ports for the SCI.

Table 2.4 Addresses of Virtual Ports for the SCI

Channel	Virtual Port Name	Address	Access Unit	Description
0	RxD0	H'00088224	16	Channel 0 receive data
	TxD0	H'00088226	16	Channel 0 transmit data
1	RxD1	H'00088228	16	Channel 1 receive data
	TxD1	H'0008822A	16	Channel 1 transmit data
2	RxD2	H'0008822C	16	Channel 2 receive data
	TxD2	H'0008822E	16	Channel 2 transmit data
3	RxD3	H'00088230	16	Channel 3 receive data
	TxD3	H'00088232	16	Channel 3 transmit data
4	RxD4	H'00088234	16	Channel 4 receive data
	TxD4	H'00088236	16	Channel 4 transmit data
5	RxD5	H'00088238	16	Channel 5 receive data
	TxD5	H'0008823A	16	Channel 5 transmit data
6	RxD6	H'0008823C	16	Channel 6 receive data
	TxD6	H'0008823E	16	Channel 6 transmit data

Tables 2.5 and 2.6 show the configurations of virtual ports RxD and TxD, respectively. Table 2.7 lists the functions of the bits in RxD and TxD.

Table 2.5 Configuration of RxD

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	SB	PE	FE	-	-	-	-	-	D7	D6	D5	D4	D3	D2	D1	D0

Table 2.6 Configuration of TxD

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	SB	-	-	-	-	-	-	-	D7	D6	D5	D4	D3	D2	D1	D0

Table 2.7 Bits in RxD and TxD

Bit	Bit Name	Initial Value	R/W	Description
0	D0	0	R/W	Data Bits D7 to D0 are used for reception or transmission of 8-bit data. D6 to D0 are used for reception or transmission of 7-bit data.
1	D1	0	R/W	
2	D2	0	R/W	
3	D3	0	R/W	
4	D4	0	R/W	
5	D5	0	R/W	
6	D6	0	R/W	
7	D7	0	R/W	
12 to 8	-	All 0	-	Reserved This bit is always read as 0. The write value should always be 0.
13	FE	0	R/W	Framing Error Bit The SCI detects a framing error if this bit included in a frame is 1.
14	PE	0	R/W	Parity Error Bit The SCI detects a parity error if this bit included in a frame is 1.
15	SB	1	R/W	Start Bit The value of this bit changes from 1 to 0 when transmission starts and from 0 to 1 when transmission ends.

Reception and transmission of data that are visible in the simulator/debugger are abstract: all data are transmitted and received at the same time. Figures 2.1 and 2.2 respectively show the reception and transmission of data in the simulator/debugger.

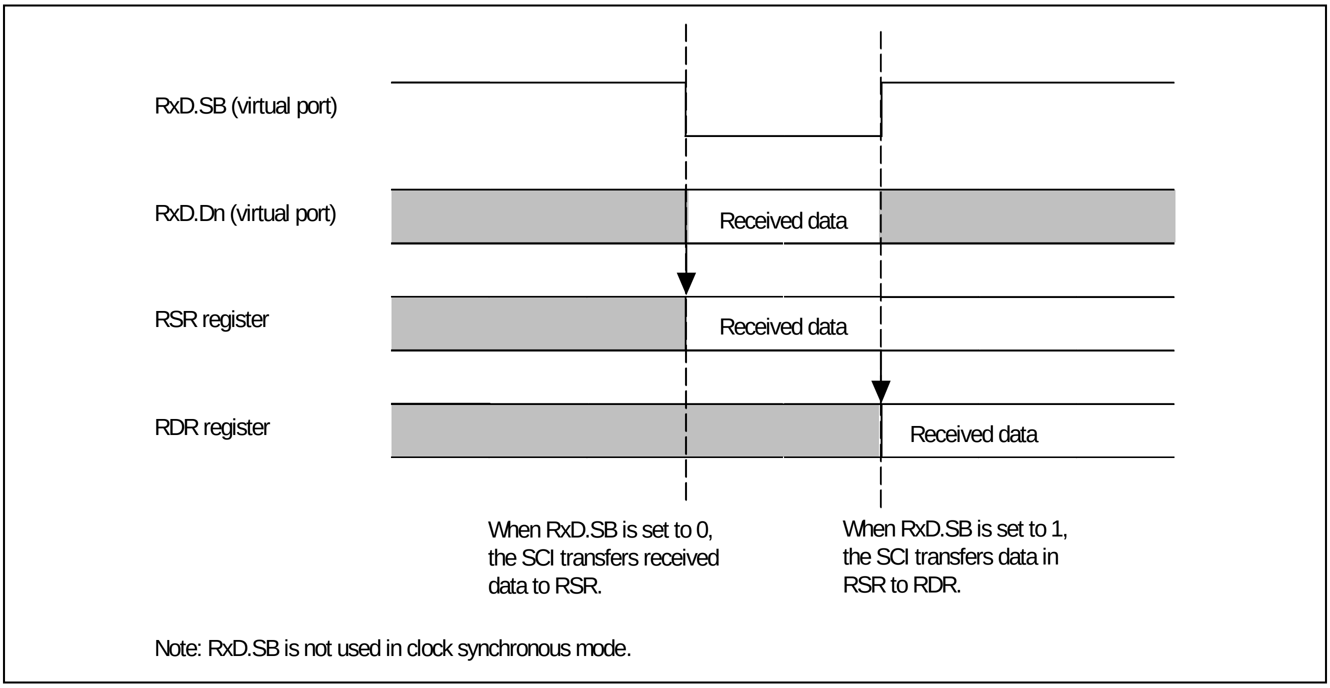


Figure 2.1 Reception of Data in the Simulator/Debugger

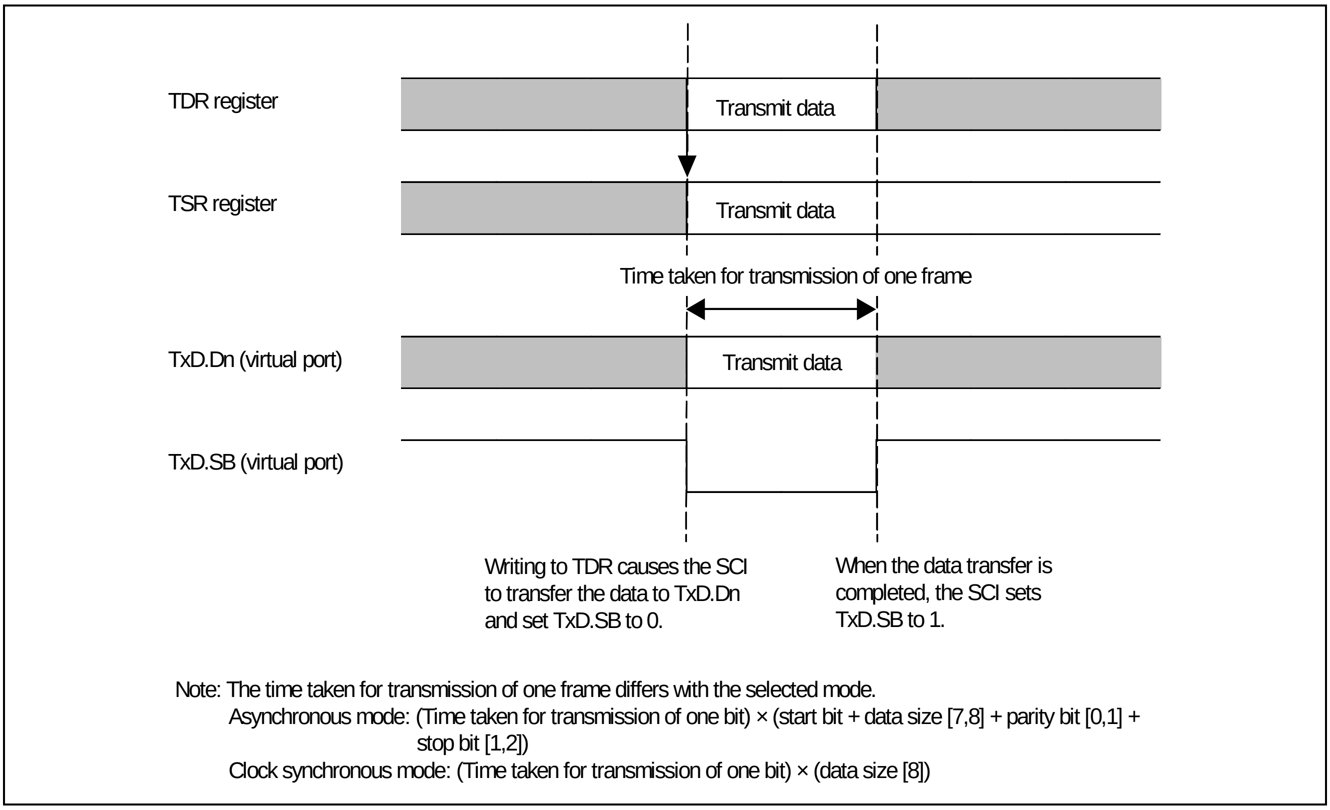


Figure 2.2 Transmission of Data in the Simulator/Debugger

The simulator/debugger allows input to and output from files through virtual ports. For details, refer to section 3.4.6, Input to and Output from Files through Virtual Ports.

2.8.3 Interrupt Controller

(1) Supported Range

The RX600 series simulator/debugger supports the interrupt controller unit (ICU) that is related to the CMT and SCI. The ICU can convey interrupts to the CPU but cannot activate the DTC or DMAC.

(2) Control Registers

Table 2.8 shows control registers of the ICU that are supported by the simulator/debugger.

In access to control registers, ensure that the unit of access is the same as the size of the register.

Table 2.8 Control Registers of the ICU Supported by the Simulator/Debugger

Peripheral Module	Supported Control Register	Support
ICU	IRn (n = 028 and 029, 214 to 241)	○
	ISELR028	Δ
	ISELR029	Δ
	ISELR030	Δ
	ISELR031	Δ
	ISELR215	Δ
	ISELR216	Δ
	ISELR219	Δ
	ISELR220	Δ
	ISELR223	Δ
	ISELR224	Δ
	ISELR227	Δ
	ISELR228	Δ
	ISELR231	Δ
	ISELR232	Δ
	ISELR235	Δ
	ISELR236	Δ
	ISELR239	Δ
	ISELR240	Δ
	IER03	Δ
	IER1A	Δ
	IER1B	○
	IER1C	○
	IER1D	○
	IER1E	Δ
	IPRm (m = 04 to 07, 80 to 86)	○
	FIR	○

Note: ○: Supported
Δ: Partly supported (bits for the function described in section 2.8.3 (1), Supported Range)

The addresses of the control registers, the interrupt vector numbers, and the position of the priority register can be referred to or modified in the [Peripheral Module Configuration] dialog box. Refer to section 3.4, Simulating Peripheral Functions, for details on this dialog box.

(3) Note on Using the ICU

To select whether an interrupt should cause a break in execution, use the [Simulator System] dialog box or EXEC_STOP_SET command.

2.8.4 Clocks

The simulator/debugger supports a system clock that provides timing in access to memory, a peripheral function clock, and clocks for operating the timers.

The numbers of cycles of the internal clock required for access to memory correspond to the specifications for the memory map. Set the frequency ratio of the system clock to the peripheral function clock in the [Set Peripheral Function Simulation] dialog box.

Use the timer control register to specify the division ratio to create the clock for operating the timers.

2.8.5 Using Peripheral Functions

To use a peripheral function, the corresponding module must be registered in the [Set Peripheral Function Simulation] dialog box, which is opened on initiation of the simulator/debugger.

For details on the module registration, refer to section 3.4, Simulating Peripheral Functions.

2.9 Trace

The simulator/debugger writes the execution results of each instruction into the trace buffer. The conditions for trace information acquisition can be specified in the [Trace Acquisition] dialog box. Right-clicking on the [Trace] window displays the pop-up menu. Choose [Acquisition...] from the pop-up menu to display the [Trace Acquisition] dialog box. The acquired trace information is displayed in the [Trace] window.

The trace information can be searched. The search conditions can be specified in the [Find] dialog box. Right-clicking on the [Trace] window displays the pop-up menu. Choose [Find -> Find…] from the pop-up menu to display the [Find] dialog box.

For details, refer to section 3.7, Viewing Trace Information.

2.10 Standard I/O and File I/O Processing

The simulator/debugger enables the standard I/O and file I/O processing to be executed by the user program. When the I/O processing is executed, the [Simulated I/O] window must be open.

Table 2.10 shows the supported I/O functions.

Table 2.10 I/O Functions

No.	Function Code	Function Name	Description
1	H'21	GETC	Inputs one byte from the standard input
2	H'22	PUTC	Outputs one byte to the standard output
3	H'23	GETS	Inputs one line from the standard input
4	H'24	PUTS	Outputs one line to the standard output
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

For details on I/O functions, refer to section 3.12, Standard I/O and File I/O Processing.

2.11 Break Conditions

The simulator/debugger provides the following conditions for interrupting the simulation of a user program during execution.

- Break due to the satisfaction of a break command condition
- Break due to the detection of an error during execution of the user program
- Break due to a trace buffer overflow
- Break due to execution of the WAIT instruction
- Break due to the [STOP] button

Break Due to Satisfaction of a Break Command Condition: There are nine break commands as follows:

- BREAKPOINT: Break based on the address of the instruction executed
- BREAK_ACCESS: Break based on access to a memory range
- BREAK_CYCLE: Break based on the instruction execution cycles
- BREAK_DATA: Break based on the value of data written to memory
- BREAK_DATA_DIFFERENCE: Break based on a difference between values in memory
- BREAK_DATA_INVERSE: Break based on sign inversion of a value in memory
- BREAK_DATA_RANGE: Break based on the range of values in memory
- BREAK_REGISTER: Break based on the value of data written to a register
- BREAK_SEQUENCE: Break based on a specified execution sequence

If [Stop] is specified as the action to take when a break condition is satisfied, user program execution stops when the break condition is satisfied. For details, refer to section 3.6, Using the Simulator/Debugger Breakpoints.

When a break condition is satisfied and user program execution stops, the instruction at the breakpoint may or may not be executed before a break depending on the type of break, as listed in table 2.11.

Table 2.11 Processing When a Break Condition is Satisfied

Command	Instruction When a Break Condition is Satisfied
BREAKPOINT	Not executed
BREAK_ACCESS	Executed
BREAK_CYCLE	Executed
BREAK_DATA	Executed
BREAK_DATA_DIFFERENCE	Executed
BREAK_DATA_INVERSE	Executed
BREAK_DATA_RANGE	Executed
BREAK_REGISTER	Executed
BREAK_SEQUENCE	Not executed

For BREAKPOINT and BREAK_SEQUENCE, if a breakpoint is specified at an address that is not the beginning of an instruction, the break will not be detected.

When a break condition is satisfied during user program execution, a break condition satisfaction message is displayed in the [Output] window and the execution stops.

Break Due to Error Detection during User Program Execution: The simulator/debugger detects simulation errors, that is, program errors that cannot be detected by the CPU exception generation functions. The [Simulator System] dialog box specifies whether to stop or continue the simulation when such an error occurs. Table 2.12 lists the error messages, error causes, and the action of the simulator/debugger in the continuation mode.

Table 2.12 Simulation Errors

Error Message	Error Cause	Processing in Continuation Mode
Memory Access Error (ADDRESS: H'nnnnnnnn)	Access to a memory area that has not been allocated	On memory write, nothing is written; on memory read, all bits are read as 1.
	Write to a memory area having the write-protected attribute	
	Read from a memory area having the read disable attribute	
	Access to an area where memory data do not exist	

When a simulation error occurs in the stop mode, the simulator/debugger returns to the command input wait state after stopping instruction execution and displaying the error message. Table 2.13 lists the states of the program counter (PC) at a simulation error stop. Also, after a stop due to a simulation error, the contents of the PSW are not changed.

Table 2.13 Register States at Simulation Error Stop

Error Message	PC Value
Memory Access Error	• When an instruction is read: The start address of the instruction that caused the error. • When an instruction is executed: The instruction address following the instruction that caused the error.

Use the following procedure when debugging programs that include instructions that generate simulation errors.

1. First execute the program in the stop mode and confirm that there are no errors except those in the intended locations.
2. After confirming the above, execute the program in the continuation mode.

Note: If an error occurs in the stop mode and simulation is continued after changing the simulator/debugger mode to the continuation mode, simulation may not be performed correctly. When restarting simulation, always restore the register contents and the memory contents to the state prior to the occurrence of the error.

Break Due to a Trace Buffer Overflow: After the [Stop] mode is specified with [Trace Buffer Full Handling] in the [Trace Acquisition] dialog box, the simulator/debugger stops execution when the trace buffer becomes full. The following message is displayed in the [Output] window when execution is stopped.

```
Trace Buffer Full
```

Break Due to Execution of a WAIT Instruction: Execution of a WAIT instruction causes execution by the simulator/debugger to stop. The following message is displayed in the [Output] window.

```
WAIT Instruction
```

Note: When restarting execution, change the PC value to the instruction address at the restart location.

Break Due to the [Stop] Button: Users can forcibly terminate execution by clicking the [HALT] button during instruction execution. The following message is displayed on the status bar when execution is stopped.

```
Stop
```

Execution can be resumed with the GO or STEP command.

2.12 Floating-Point Data

Floating-point numbers can be used for the following real-number data, which makes floating-point data processing easier. The following data can be specified for floating-point data:

- Data when the break type is set to [Break Data] or [Break Register] in the [Select Break Type] dialog box
- Data in the [Memory] window
- Data in the [Fill Memory] dialog box
- Data in the [Search Memory] dialog box
- Input data in the [Register] dialog box

The floating-point data format conforms to the ANSI C standard.

In the simulator/debugger, the round-to-nearest (RN) mode is applied as the rounding mode for floating-point decimal-to-binary conversion.

If a denormalized number is specified for binary-to-decimal or decimal-to-binary conversion, it is left as a denormalized number in RN mode. If an overflow occurs during decimal-to-binary conversion, the infinity is returned in RN mode.

2.13 Display of Function Call History

The simulator/debugger displays the function call history in the [Stack Trace] window when simulation stops, which enables program execution flow to be checked easily. Selecting a function name in the [Stack Trace] window displays the corresponding source program in the [Editor] window. This allows the function that has called the current function to also be checked.

The displayed function call history is updated in the following cases:

- When simulation stops due to the break conditions described in section 2.11, Break Conditions.
- When register values are modified while simulation stops due to the above break conditions.
- While single-step execution is performed.

For details, refer to the High-performance Embedded Workshop User’s Manual.

2.14 Performance Measurement

The simulator/debugger has the profiler function and performance analysis function for performance measurement of the user program.

2.14.1 Profiler

The profiler function displays the memory address and size allocated to functions and global variables, the number of function calls, and the profile data for the entire user program. The profile data to be displayed depends on the CPU.

Profile information is displayed in list, tree, and chart formats.

Profile information is useful in optimizing user programs by reducing the size and putting the most frequently called functions in-line.

When using the profile information saved in a file, it is possible to optimize user programs based on dynamic information using the optimizing linkage editor.

For details, refer to section 3.8, Viewing the Profile Information.

2.14.2 Performance Analysis

The performance analysis function displays the number of execution cycles and function calls for the specified function in the user program. Since performance data for only the specified function is acquired, faster simulation is possible. For details, refer to section 3.9, Analyzing Performance.

2.15 Pseudo-Interrupts

The simulator/debugger can generate pseudo-interrupts during simulation in the following two ways:

1. Pseudo-interrupts generated by satisfaction of break conditions
   A pseudo-interrupt can be generated using a break command to specify [Interrupt] as the action when a break condition is satisfied. For details, refer to refer to section 3.6, Using the Simulator/Debugger Breakpoints.
2. Pseudo-interrupts generated from windows
   A pseudo-interrupt can be generated by clicking a button in the [Trigger] or [GUI I/O] window. For details, refer to section 3.11, Generating a Pseudo-Interrupt Manually.

If another pseudo-interrupt occurs between a pseudo-interrupt occurrence and its acceptance, only the interrupt that has a higher priority can be accepted.

3. Break by pseudo-interrupts
   The user can select whether or not to cause a break when a pseudo-interrupt occurs. This can be set in the [Simulator System] dialog box or by the EXEC_STOP_SET command.

Note: For a pseudo-interrupt, the vector number and priority level of the interrupt are specified. The priority level of an interrupt can be specified as a value from 0 to 8 or from 0 to H'10. The fast interrupt is specified by the value 8 when the range is from 0 to 8 and H'10 when the range is from 0 to H'10.

2.16 Coverage

The simulator/debugger acquires instruction coverage information during instruction execution within the measurement range specified by the user.

In the measurement range, addresses are directly specified, and all functions in a file whose name has been specified are set.

The state of each instruction execution can be monitored through the instruction coverage information. In addition, this information can be used to determine which part of a program has not been executed.

The [Coverage] window displays the acquired instruction coverage information.

The instruction coverage information can be displayed in the [Editor] window by highlighting the column corresponding to the source line of the executed instruction.

For the address range or function to be measured, the coverage statistical information is displayed in percentage. This gives the user a clear idea how much the program has been executed.

The instruction coverage information can be saved in or loaded from a file. Only a file in the .COV format can be loaded.

For details, refer to section 3.10, Measuring Code Coverage.

Section 3 Debugging

This section describes the simulator/debugger operations and their related windows and dialog boxes.

For details on the functions common to the HEW listed below, refer to the HEW help information.

— Preparations for Debugging
— Viewing a Program
— Operating Memory
— Displaying Memory Contents as Waveforms
— Displaying Memory Contents as an Image
— Modifying the Memory Contents
— Viewing the I/O Memory
— Looking at Registers
— Executing Your Program
— Viewing the Current Status
— Synchronizing Multiple Debugging Platforms
— Debugging with the Command Line Interface
— Elf/Dwarf2 Support
— Looking at Labels

3.1 Creating the Workspace for Simulator/Debugger

To use the simulator/debugger, a workspace for the simulator/debugger must be created. This section only describes the procedures specific to the simulator/debugger. For details, refer to the High-performance Embedded Workshop user's manual.

3.1.1 Selecting a Debugging Platform

When you create a new workspace, the dialog box shown below appears. Specify the debugging platform in step 8.

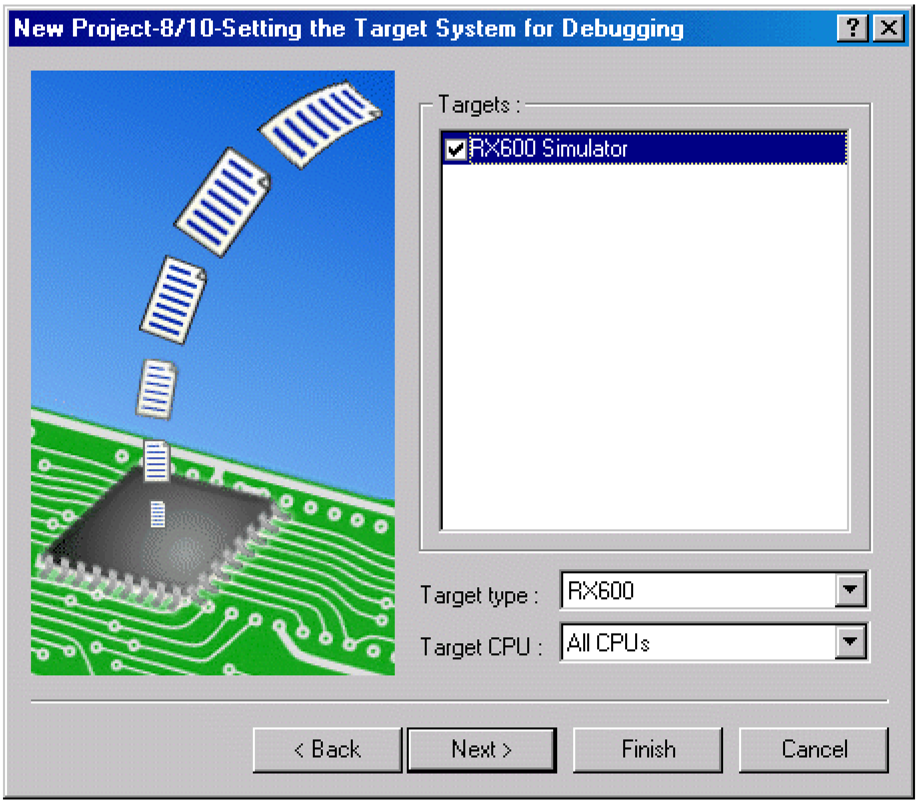


Figure 3.1 Debugger Target Setting Display (8/10)

[Targets]	Sets the debugger targets. Select (by checking) the debugger targets. No selection or a selection of more than one target is possible.
[Target type]	Specifies the type of the targets displayed under [Targets].
[Target CPU]	Specifies the type of the CPUs displayed under [Targets].

3.1.2 Setting up a Workspace for the Simulator/Debugger

Set up the workspace for the simulator/debugger in step 9/10.

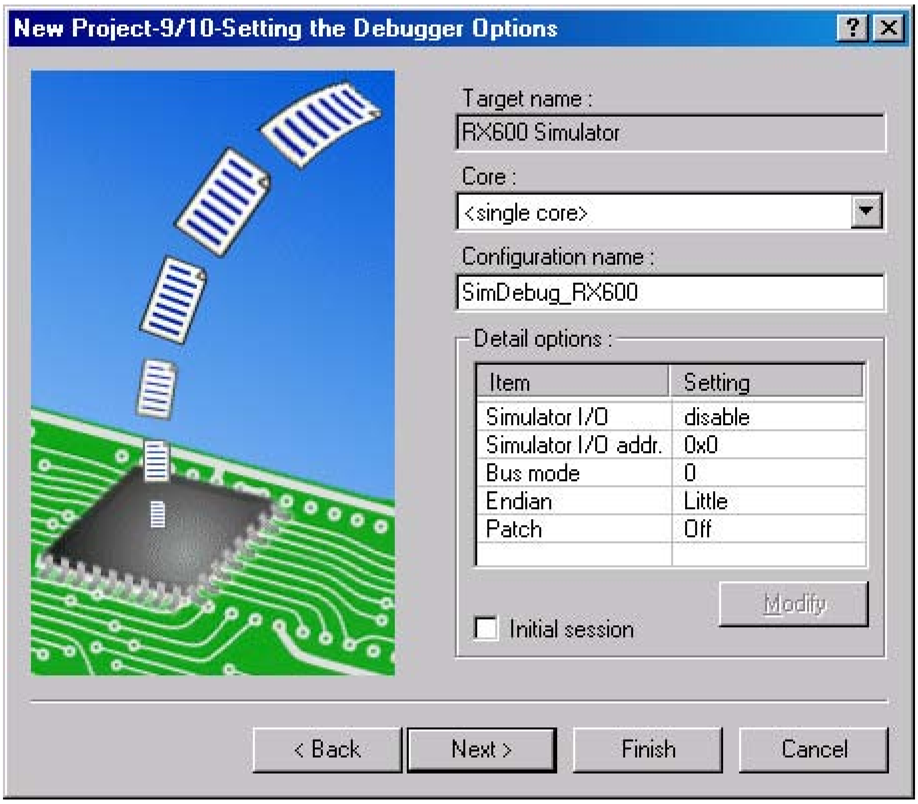


Figure 3.2 Debugger Option Setting Display (9/10)

[Detail options] Sets the debugger target options. To modify an option, select [Item] and click [Modify]. If the selected item cannot be modified, [Modify] remains gray even when [Item] is selected.

- [Simulator I/O] Simulation for standard I/O or file I/O from the user program is enabled ([Enable]) or disabled ([Disable]).
- [Simulator I/O addr.] Address for the above simulated I/O.
- [Bus mode] Currently not used by the simulator/debugger.
- [Endian] Displays the endian of CPU.
- [Patch] Priority levels of interrupts and whether the MVTIPL instruction is enabled or disabled.
  - [Off] Available priority levels for interrupts are from 0 to 15. The MVTIPL instruction is enabled.
  - [RX610] Available priority levels for interrupts are from 0 to 7. The MVTIPL instruction is disabled.

Refer to the High-performance Embedded Workshop User's Manual for items other than those listed under [Detail options].

3.2 Starting up the Simulator/Debugger

You can connect to the simulator/debugger by selecting a session file in which simulator/debugger settings have already been defined. When you have selected targets in the process of creating a project, the number of session files is the same as the number of selected targets. Select the session file that corresponds to the current target from the drop-down list shown in figure 3.3.

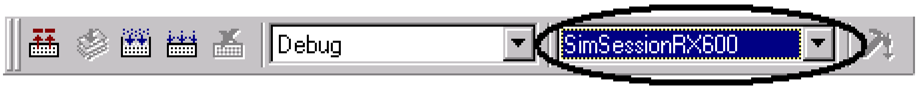


Figure 3.3 Selecting a Session File

If you have selected a session file with which the simulator/debugger has been registered but the simulator/debugger is disconnected, select [Debug -> Connect] or click on the [Connect] toolbar button .

To disconnect the simulator/debugger, on the other hand, select [Debug -> Disconnect] or click on the [Disconnect] toolbar button .

3.3 Modifying the Simulator/Debugger Settings

This section describes how to modify the simulator system after the simulator/debugger is started.

3.3.1 Setting the Endian and Frequency of CPU

The endian and operating frequency of CPU are set on the [CPU Configuration] tabbed page in the [Set Simulator] dialog box, which is displayed on initiation of the simulator/debugger.

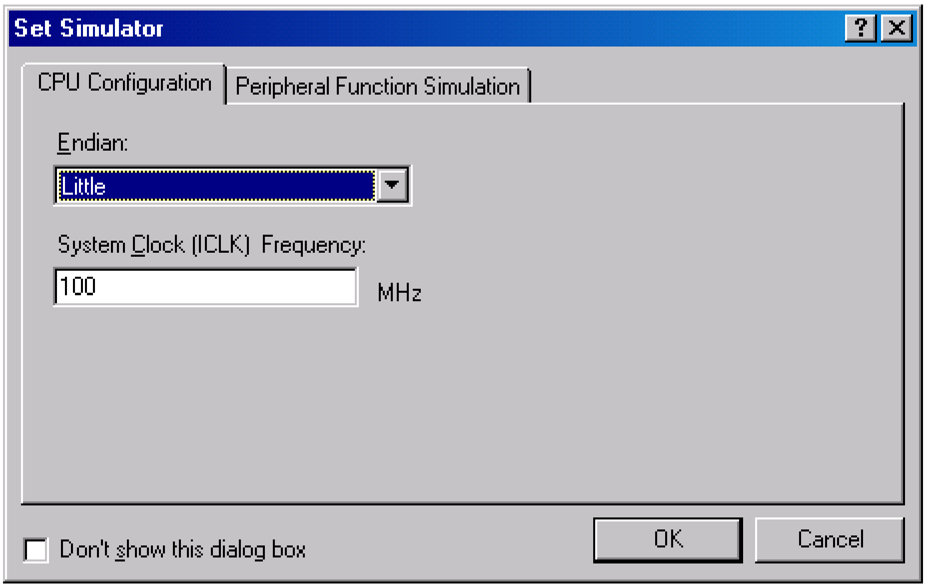


Figure 3.4 Set Simulator Dialog Box (CPU Configuration)

The following items can be specified in this dialog box:

[Endian] Endian of CPU.

[Big]	Big endian
[Little]	Little endian

[System Clock (ICLK) Frequency] Operating frequency of the CPU (unit: MHz)
Specifiable range: 1 to 1000

If you do not wish this dialog box to be opened when the simulator/debugger is subsequently initiated, check [Don’t show this dialog box].

3.3.2 Modifying the Simulator System

The [System] tab in the [Simulator System] dialog box is used to modify the location to start the simulated I/O and execution mode.

Choose [Setup -> Simulator -> System...] or click the [Simulator System] toolbar button to open the [System] tab in this dialog box.

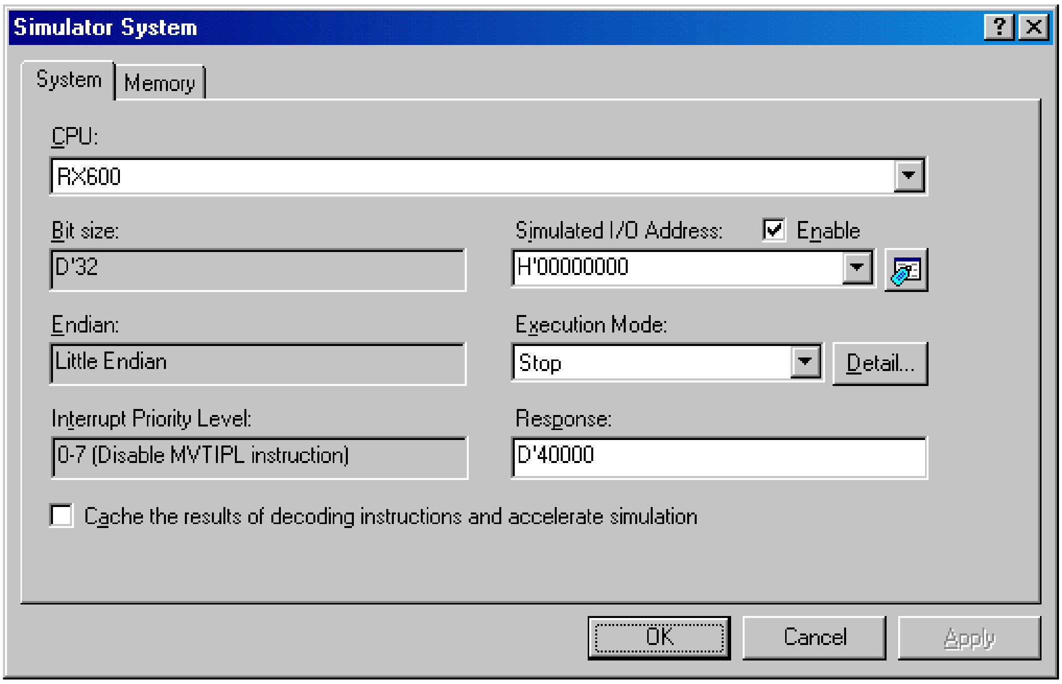


Figure 3.5 Simulator System Dialog Box (System Tab)

The following items can be specified in this dialog box:

[CPU] The current CPU.

[Bit Size] Size of the address space (as the number of bits in addresses.

[Endian] Endian of the CPU.

[Priority Level of Interrupts] Priority levels of interrupts and whether the MVTIPL instruction is enabled or disabled.
0 to 7 (Disable MVTIPL instruction): Available priority levels for interrupts are from 0 to 7.
0 to 15 (Enable MVTIPL instruction): Available priority levels for interrupts are from 0 to 15.

[Simulated I/O Address] Specifies the start address of a simulated I/O that performs standard input/output or file input/output processing from the user program.
[Enable] Checking this box enables the simulated I/O.

[Response] Specifies the window refresh timing; that is, how many instructions should be executed between refresh operations (D'1 to D'2,147,483,647. The default is D'40000).

[Execution Mode] Specifies whether the simulator/debugger stops or continues operation when a simulation error (including interrupts) occurs. It is also possible to specify an action to take place when an interrupt occurs by clicking the [Detail…] button.
[Stop] Stops simulation.
[Continue] Continues simulation.

[Cache the results of decoding instructions and accelerate simulation]
Selects whether or not to save the results of decoding instructions at the time of their execution and reuse the results of decoding when instructions at the same addresses are reused.
Selecting this box enables the caching facility for decoded instructions, making simulation faster.

Clicking the [OK] or [Apply] button stores the modified settings. Clicking the [Cancel] button closes this dialog box without modifying the settings.

Note: The caching facility for decoded instructions reuses results of decoding so is not applicable to programs that contain self-modifying code. Furthermore, errors in the form of an instruction being overwritten due to unexpected behavior of the program may not be correctly detected.

3.3.3 Modifying the Memory Map and Memory Resource Settings

The [Memory] tab in the [Simulator System] dialog box is used to set and modify the memory map and memory resource.

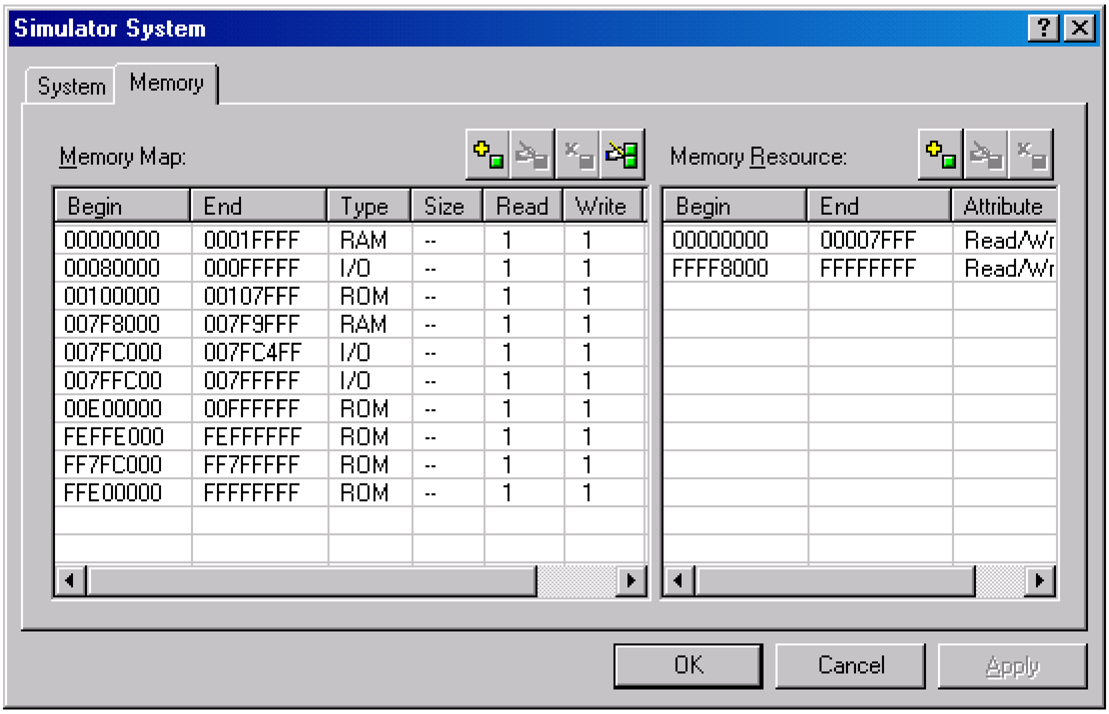


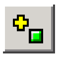
Figure 3.6 Simulator System Dialog Box (Memory Tab)

The following items can be specified in this dialog box:

[Memory Map] Displays the memory type, start and end addresses, data bus width, and the number of access cycles.

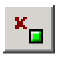
[Memory Resource] Displays the access type and start and end addresses of the current memory resource.

[Memory Map] can be added, modified, or deleted using the following buttons:



Adds [Memory Map] items. Clicking this button opens the [Set Memory Map] dialog box (figure 3.6), and memory map items can be added.

Modifies [Memory Map] items. Select an item to be modified in the list box and click this button. The [Set Memory Map] dialog box (figure 3.6) opens and memory map items can be modified.

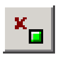


Deletes [Memory Map] items. Select an item to be deleted in the list box and click this button.

[Memory Resource] can be added, modified, or deleted using the following buttons:

Adds [Memory Resource] items. Clicking this button opens the [Set Memory Resource] dialog box, and memory map items can be specified.

Modifies [Memory Resource] items. Select an item to be modified in the list box and click this button. The [Set Memory Resource] dialog box opens and memory map items can be modified.



Deletes [Memory Resource] items. Select an item to be deleted in the list box and click this button.

[Memory Resource] is the same setting information as that of [Memory Resource] of the [Debugger] sheet in the [RX Standard Toolchain] dialog box. Modifications are reflected on both items.

[Memory Map] can be reset to the default value by the button. Clicking the [OK] or [Apply] button stores the modified settings. Clicking the [Cancel] button closes this dialog box without modifying the settings.

When there is a linkage list file (.map) output by the optimizing linkage editor, the memory resource can be automatically allocated according to the memory map and linkage map information. For details, refer to Automatically Allocating the Memory Resource, in the High-performance Embedded Workshop User's Manual.

3.3.4 Set Memory Map Dialog Box

The [Set Memory Map] dialog box specifies the memory map of the target CPU.

The contents displayed in this dialog box depend on the target CPU. The values are used in simulation of memory access by the simulator/debugger.

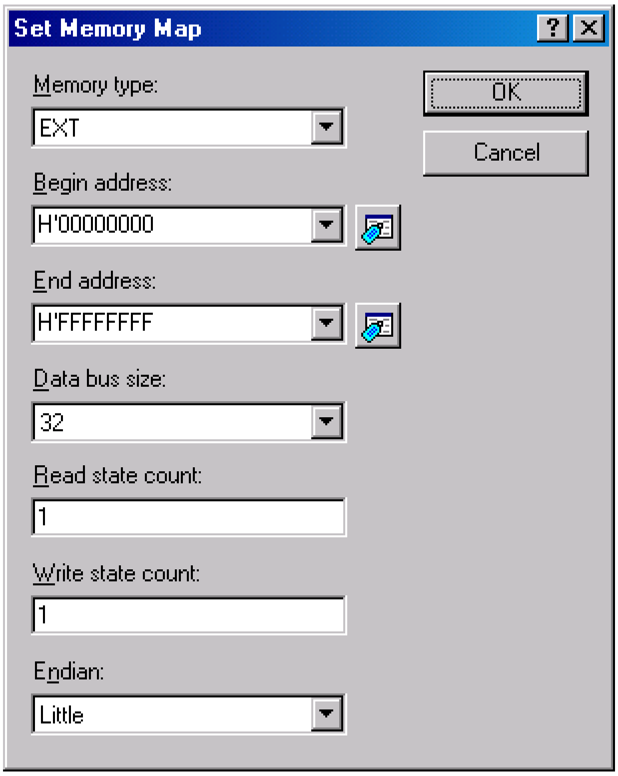


Figure 3.7 Set Memory Map Dialog Box

The following items are specified:

[Memory type]	Memory type	
	[ROM]	Internal ROM
	[RAM]	Internal RAM
	[EXT]	External memory
	[IO]	Internal I/O
[Begin address]	Start address of the memory corresponding the memory type	
[End address]	End address of the memory corresponding to the memory type	
[Data bus size]	Memory data bus width	
[Read state count]	Number of cycles ("states") for read access to the specified type of memory	
[Write state count]	Number of cycles ("states") for write access to the specified type of memory	
[Endian]	Endian of the specified area of memory	

Clicking the [OK] button stores the settings. Clicking the [Cancel] button closes this dialog box without modifying the settings.

Notes: 1. The memory map setting for the area allocated to a system memory resource cannot be deleted or modified. First delete the system memory resource allocation on the [Memory] tab of the [Simulator System] dialog box, then delete or modify the memory map setting.

2. The data bus size cannot be displayed or modified for any type of memory other than external memory.
3. The data bus size, read state count, and write state count do not affect to the instruction simulations. The number of states (cycles) for memory access is always 1.
4. The memory map must start and end on 16-byte boundaries. If any other setting is made, the map is adjusted to the closest 16-byte boundaries that include the set values.
5. It is not possible to view or modify the current endian for the internal I/O area.
6. The endian for the internal ROM and RAM areas is only modifiable through the [Set Simulator] dialog box. For details on the [Set Simulator] dialog box, refer to section 3.3.1, Setting the Endian and Frequency of CPU.

3.3.5 Set Memory Resource Dialog Box

The [Set Memory Resource] dialog box sets and modifies memory resources.

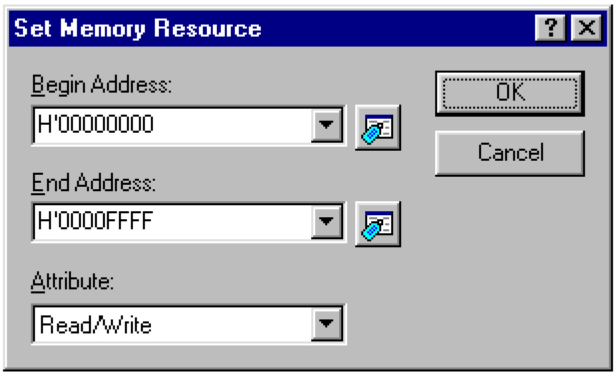


Figure 3.8 Set Memory Resource Dialog Box

The following items are specified:

[Begin Address] Address where the memory area to be secured starts

[End Address] Address where the memory area to be secured ends

[Attribute] Access type

[Read]	Read only
[Write]	Write only
[Read/Write]	Readable/writable

Click the [OK] button after specifying the [Begin Address], [End Address], and [Attribute]. Clicking the [Cancel] button closes this dialog box without modifying the settings.

Notes: 1. If memory resources are set, memory in the host computer will be used. If the user allocates too much memory resources, operation of the host computer will be extremely slow.

2. The memory area must start and end on 16-byte boundaries. If any other setting is made, the area is adjusted to the closest 16-byte boundaries that include the set values. Furthermore, concerning the type of access, boundaries become 16 bytes.
When using a resource with units smaller than 16 bytes, use the memory within an area in accord with the hardware manual.

3. Attempts by instructions to write to memory for which only reading is permitted or to read from memory for which only writing is permitted cause memory-access errors.

3.4 Simulating Peripheral Functions

The simulator/debugger is able to simulate peripheral functions by using DLL modules. This section describes how to register the peripheral function simulation modules to enable the simulation of individual peripheral functions, and how to set their configurations.

3.4.1 Registering Peripheral Function Simulation Modules

Peripheral function simulation modules can be registered in the [Peripheral Function Simulation] tabbed page of the [Set Simulator] dialog box, which is opened on initiation of the simulator/debugger.

Once a peripheral function simulation module has been registered in this dialog box, the simulated peripheral function provided by that simulation module becomes available. The registered settings cannot be modified after the simulator/debugger has fully started up. To change the peripheral function simulation modules that are in use, restart the simulator/debugger to bring up this dialog box.

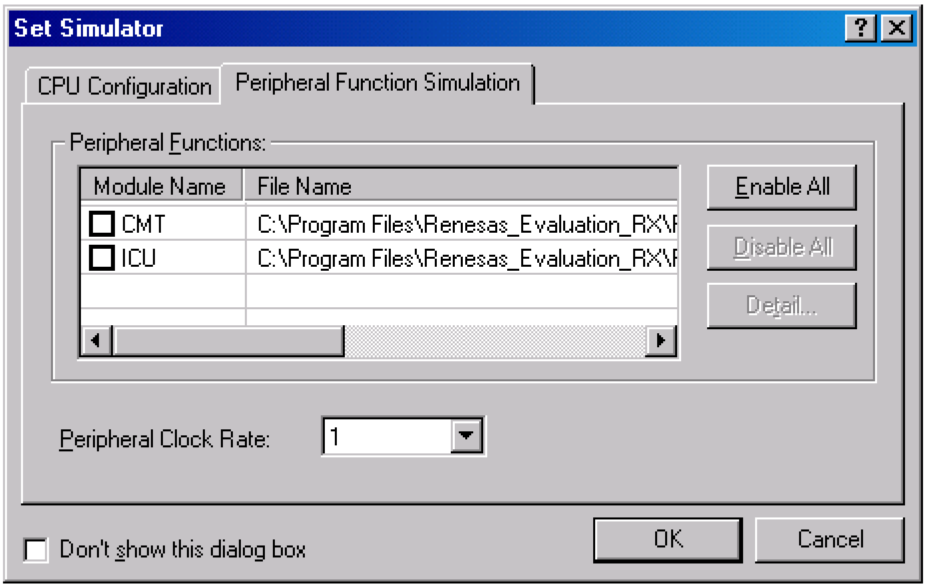


Figure 3.9 Set Simulator Dialog Box (Peripheral Function Simulation Tab)

The following items are specified in this dialog box:

[Peripheral Functions] Shows information 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 and make it available.

[Enable All] Enables all peripheral function simulation modules.

[Disable All] Disables all peripheral function simulation modules.

[Detail…] Opens the [Peripheral Module Configuration] dialog box, allowing you to view information on the corresponding peripheral function, and change the address where it starts and the interrupt-source information.

[Peripheral Clock Rate] The ratio between the peripheral clock and the system clock (the number of cycles of the system clock corresponding to one cycle of the peripheral clock) is specified here. The clock rate setting can be selected as 1, 2, 3, 4, 6, 8, 12, 16, 24, or 32.

Clicking the [OK] button makes the settings effective. Clicking the [Cancel] button closes this dialog box without storing the settings.

If you do not wish this dialog box to be opened when the simulator/debugger is subsequently initiated, check [Don't show this dialog box].

3.4.2 Changing the Addresses of Peripheral Functions

The addresses of peripheral functions can be changed on the [Peripheral Module Configuration] dialog box. The addresses of the peripheral functions which have interrupt source information can be changed on the [Address] tabbed page of the [Peripheral Module Configuration] dialog box. To open this dialog box, select a peripheral function in [Peripheral Functions] on the [Peripheral Function Simulation] tabbed page of the [Set Simulator] dialog box and then press the [Detail…] button.

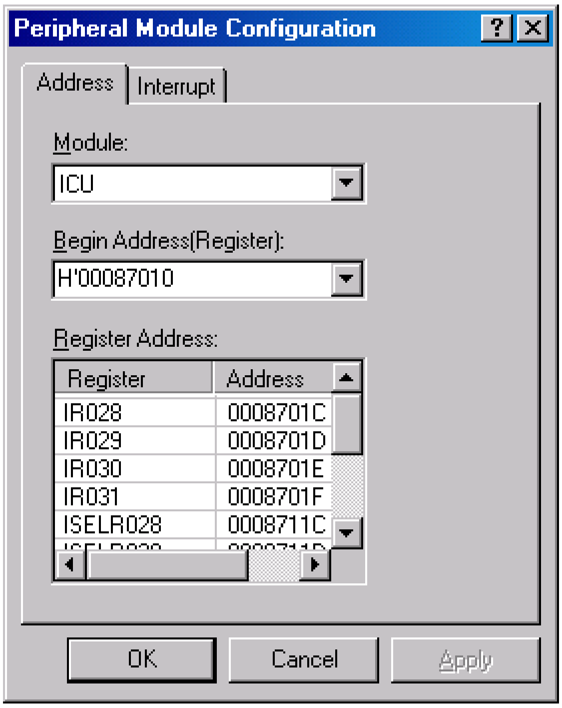


Figure 3.10 Peripheral Module Configuration Dialog Box

The following items can be set or displayed in this dialog box:

[Module] Name of the peripheral function supported by the selected peripheral function simulation module

[Start Address] Start address of the peripheral function selected in [Module]

[Register Address] Names and addresses of registers of the peripheral function selected in [Module]. It is not possible to change the register addresses.

Clicking the [OK] or [Set] button makes the settings effective. Clicking the [Cancel] button closes this dialog box without storing the settings.

3.4.3 Changing the Interrupt Source Information of Peripheral Functions

The interrupt source information of peripheral functions can be changed in the [Interrupt] tab of the [Peripheral Module Configuration] dialog box. To open this dialog box, select a peripheral function in [Peripheral Functions] on the [Peripheral Function Simulation] tabbed page of the [Set Simulator] dialog box and then press the [Detail…] button.

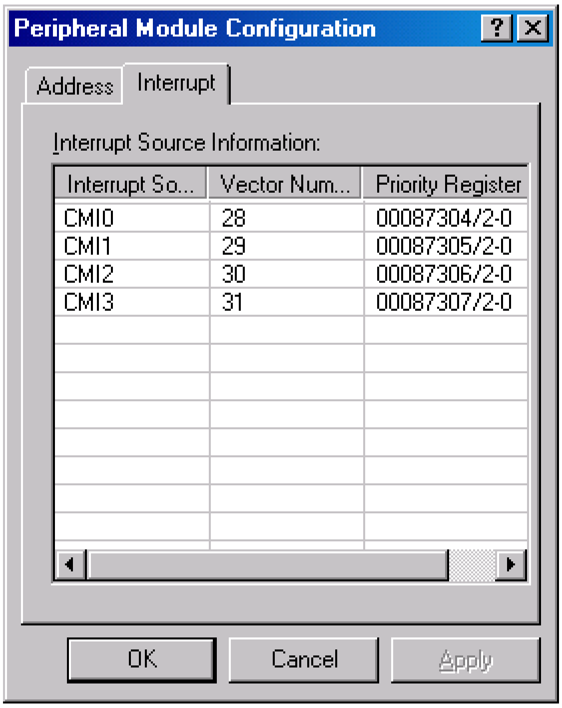


Figure 3.11 Peripheral Module Configuration Dialog Box (Interrupt Tab)

The following items can be displayed in this dialog box:

Interrupt Source: Name of the interrupt source (or sources) supported by the peripheral function

Vector Number: Interrupt vector number

Priority Register Address/ Bit Field Position: Address of the interrupt priority register and positions of bits in the register

To change the interrupt-source information, open the [Set Interrupt Source Information] dialog box by double-clicking on the line for the interrupt source to be changed.

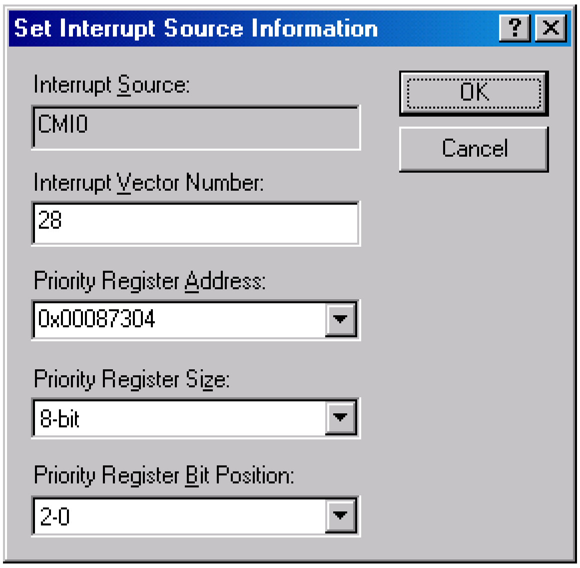


Figure 3.12 Set Interrupt Source Information Dialog Box

The following items can be set or displayed in this dialog box:

Interrupt Source:	Interrupt source name
Interrupt Vector Number:	Interrupt vector number (when the prefix is omitted, values input are taken as decimal, and the display is in decimal notation)
Priority Register Address:	Address of the interrupt priority register
Priority Register Size:	Size of the interrupt priority register
Priority Register Bit Position:	Positions of bits in the interrupt priority register

Clicking the [OK] button makes the settings effective. Clicking the [Cancel] button closes this dialog box without storing the settings.

3.4.4 Memory Resources for Control Registers

The peripheral function simulation module secures memory resources in the control register area. Do not perform operations that lead to the deletion or alteration of memory resources for control registers after they have been allocated. For details on the setting of memory resources, refer to section 3.3.3, Modifying the Memory Map and Memory Resource Settings.

3.4.5 Viewing the Names of Connected Peripheral Functions

After the simulator/debugger has been initiated, [Peripheral Modules] on the [Platform] sheet of the [Status] window shows the names of the peripheral functions that are connected.

3.4.6 Input to and Output from Virtual Ports

For the simulator/debugger, some pins are allocated to memory as virtual ports. These can be used for input to and output from files. For details on the virtual ports supported by the simulator/debugger, refer to section 2.8.2 (3), Input and Output of Data.

(1) Viewing the List of File Input and Output

To view the list of file input and output that is currently defined, open the [Port I/O] tabbed page of the [Simulator System] dialog box that is displayed by selecting [Setup -> Simulator -> System…]. If no modules with virtual ports have been registered, the [Port I/O] tab does not appear.

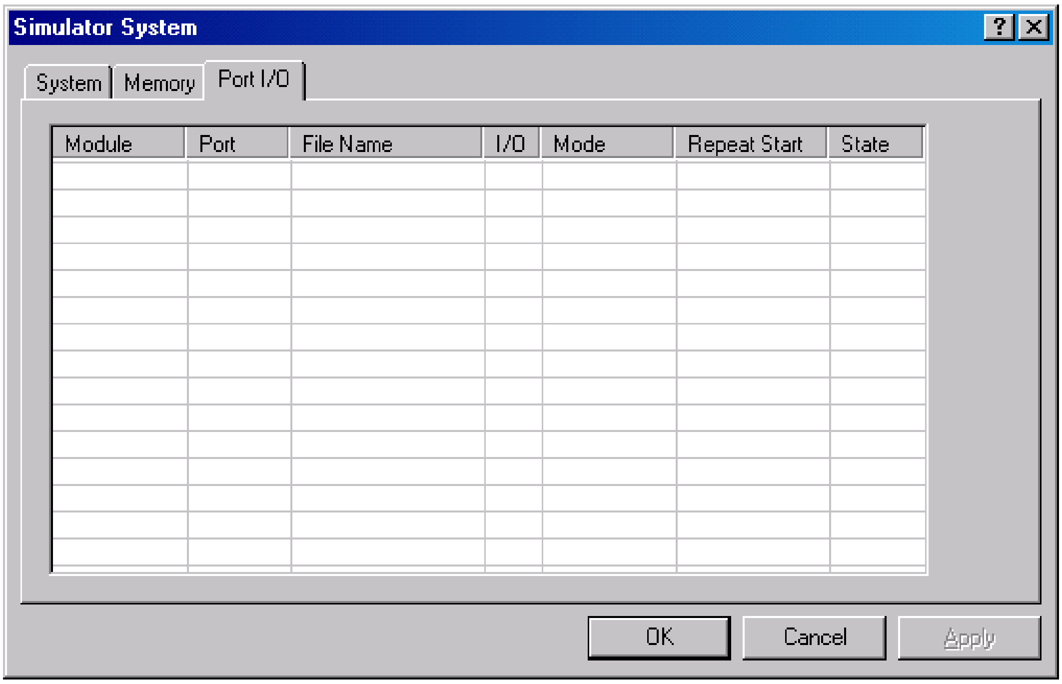


Figure 3.13 Simulator System Dialog Box (Port I/O Tab)

The following items are displayed in this dialog box:

[Module]: Module name

[Port]: Port name

[File Name]: Filename

[I/O]: Input or output
[In]: File input
[Out]: File output

[Mode]: Mode of file input or output
[Repeat]: Repeated input
[Once]: Input only once
[Overwrite]: Write output over existing files
[Append]: Append output to existing files

[Repeat Start]: Line number where repeated input starts

[State]: Whether the file is open or closed
[Open]: Open
[Close]: Closed

(2) Adding a File

Right-click on the [Port I/O] tabbed page and select [Add] from the popup menu or double-click on an item in the list to open the [Set Port I/O] dialog box.

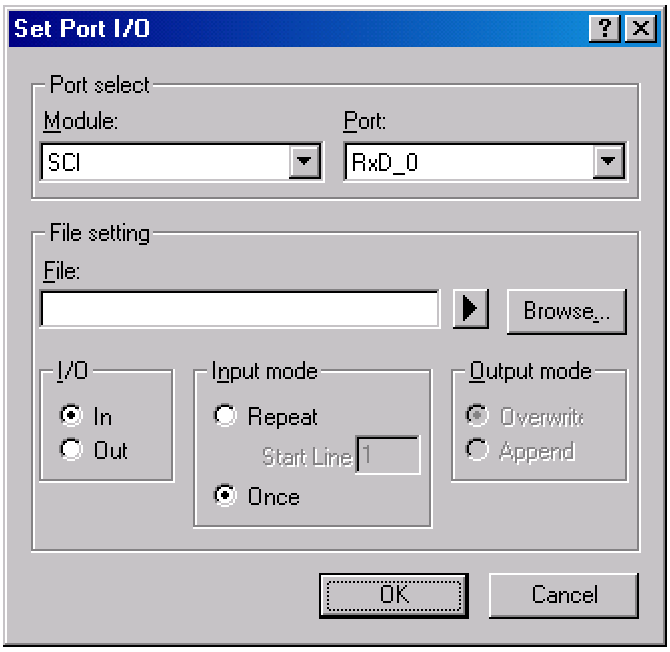


Figure 3.14 Set Port I/O Dialog Box

The following items can be set in this dialog box:

[Port select]
[Module]: Select the module for the port that data are to be input to or output

		from.
[Port]:		Select the port name.

[File setting]

[File]:		Specify the filename. If the filename extension is omitted, .csv is automatically appended.
[I/O]	[Input]:	File input
	[Output]:	File output
[Input mode]	[Repeat]:	When the end of the file is reached, the input is repeated from the start. [Start Line]: Line number where repeated input starts (1 to 65535)
	[Once]:	When the end of the file is reached, the input is ended.
[Output mode]	[Overwrite]:	If an output file with the specified name already exists, that file is overwritten.
	[Append]:	If an output file with the specified name already exists, output data are appended to the end of the file.

Each port can be allocated to one file for input and one file for output. A single file can also be allocated to two or more input ports.

(3) Opening a File

To open a file, click on the line where the filename appears on the [Port I/O] tabbed page and select [Open] from the popup menu.

(4) Opening All Files

To open all files, right-click on the [Port I/O] tabbed page and select [Open All] from the popup menu.

(5) Closing a File

To close a file, click on the line where the filename appears on the [Port I/O] tabbed page and select [Close] from the popup menu.

(6) Closing All Files

To close all files, right-click on the [Port I/O] tabbed page and select [Close All] from the popup menu.

(7) Modifying File Setting

Click on the line where the filename appears and select [Edit] from the popup menu or simply double-click on the line to open the [Set Port I/O] dialog box, where the settings for the file can be modified.

(8) Deleting a File

To delete a file, click on the line where the filename appears on the [Port I/O] tabbed page and select [Delete] from the popup menu.

(9) Format for Virtual Port Files

Virtual port files are in the CSV format. The input file format is as follows.

```
<Time>, <Data>
      :
```

Data values in input files must be accompanied by descriptions of the times they are input. Each time is the difference in picoseconds (integer value: must be 1 or larger) from the time for the previous value. The values are hexadecimal integers.

The output file format is as follows.

```
[Module]
<Module name>
[Port]
<Port name>
[Length]
<Number of bits in data>
[Data]
<Time>, <Data>
      :
```

The name of the module that outputs the data, port name, number of bits in the values, times, and the values themselves are output in an output file. The time indicates the duration from the start of simulation to the output of the value in picoseconds (as an integer).

3.5 Operations for Memory

3.5.1 Regularly Updating Contents of the [Memory] Window

Selecting [Auto Refresh] from the pop-up menu of the [Memory] window leads to regular updating of the contents displayed in the [Memory] window during execution of the user program.
The default value and specifiable range for the update interval are given below.
Default value for the update interval: 100 ms
Specifiable range for the update interval: 10 ms to 10,000 ms

3.5.2 Viewing and Modifying the Settings for the I/O Area

If you wish to view or modify the settings for the I/O area through the [Memory] window, ensure that the access size defined in the hardware manual is selected for display in the [Memory] window. Otherwise the settings may not be correctly displayed or modified.

3.6 Using the Simulator/Debugger Breakpoints

Sophisticated breakpoint functions are available in the simulator/debugger in addition to the HEW standard PC breakpoints. The user can specify break conditions and actions after a break condition is satisfied, and can display the breakpoints set.

3.6.1 Listing the Breakpoints

Choose [View -> Code -> Eventpoints] or click the [Eventpoints] toolbar button  to open the [Event] window, which lists the breakpoints set.

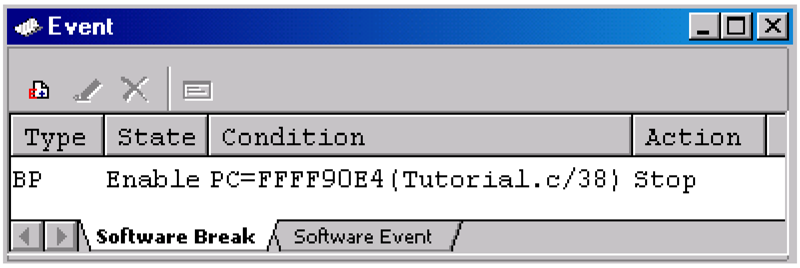


Figure 3.15 Event Window

The following items are displayed:

[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 breakpoint is enabled or disabled
- [Enable]: Valid
- [Disable]: Invalid

[Condition] Condition that causes a break. The contents to be displayed depend on the type of the break. When the type of the break is BR, the register name is displayed, and when the type of the break is BCY, the number 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
- [Trace Trigger]: Tracing starts

Conditions specifying [Stop] for [Action] is displayed on the [Software Break] tab and the conditions specifying another action type is on the [Software Event] tab.

3.6.2 Setting a Breakpoint

Selecting [Add...] from the pop-up menu in the [Event] window opens the [Select Break Type] dialog box, which allows the user to set a breakpoint.

Two further dialog boxes can be opened from the [Select Break Type] dialog box: [Set xx Condition] for specifying a break condition and [Set xx Action] for specifying an action to take when the break condition is satisfied. To open the [Select Break Type] dialog box from the [Event] window when you wish to select [Stop] as [Action type] in the [Select Break Type] dialog box, select [Add...] from the pop-up menu on the [Software Break] tab; if you wish to select another action type, select [Add...] from the pop-up menu on the [Software Event] tab.

Selecting a Break Type:

Selecting [Add...] from the popup menu on the [Event] window opens the [Select Break Type] dialog box. Select a break type in the [Break type] field of this dialog box.

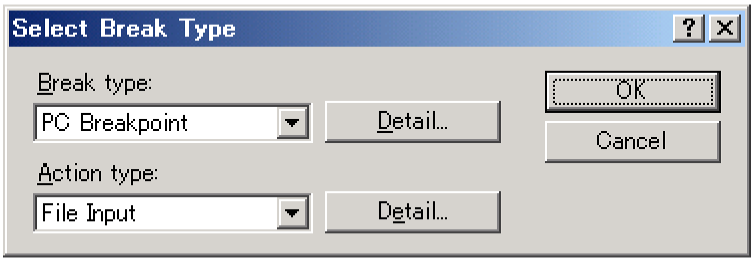


Figure 3.16 Select Break Type Dialog Box

The following options are available:

[Break type]

[PC Breakpoint]: Breakpoint set at an instruction
[Break Access]: Break on access to a memory range
[Break Data]: Break on detection of a memory value
[Break Register]: Break on detection of a register value
[Break Sequence]: Sequential breakpoints
[Break Cycle]: Break after the specified number of cycles

Setting Break Conditions:

Click on [Detail…] after selecting the break type in the [Select Break Type] dialog box. This opens a dialog box that allows you to set conditions for the selected break type.

- [PC Breakpoint]

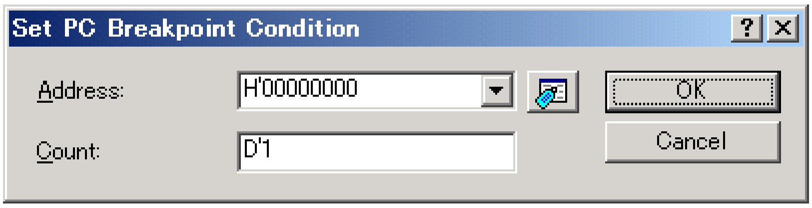


Figure 3.17 Set PC Breakpoint Condition Dialog Box

Up to 1024 PC-breakpoint conditions can be specified.

[Address]: Address of the instruction where a break will occur

[Count]: Number of times that the specified instruction is fetched (1 to 16,383; the default value is 1; if the prefix is omitted, values input are taken as decimal, and the display is in decimal notation).

- [Break Access]

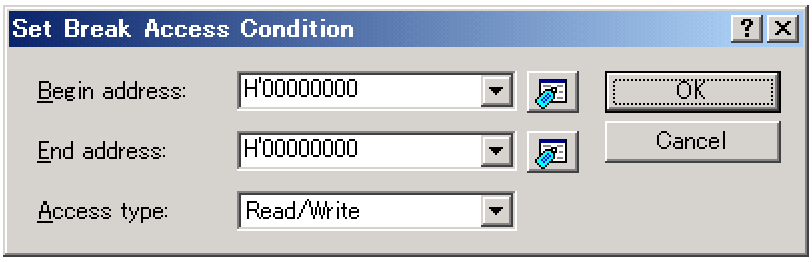


Figure 3.18 Set Break Access Condition Dialog Box

Up to 1024 access break conditions can be specified.

[Begin address]: First address of the range of memory for which access generates a break

[End address]: Last address of the range of memory for which access generates a break (if no data is input, the range corresponds to the first address alone)

[Access type]: Read, write, or read/write

Note: For string and multiply-and-accumulate instructions, only the last data-access operation is checked for access break conditions.

- [Break Register]

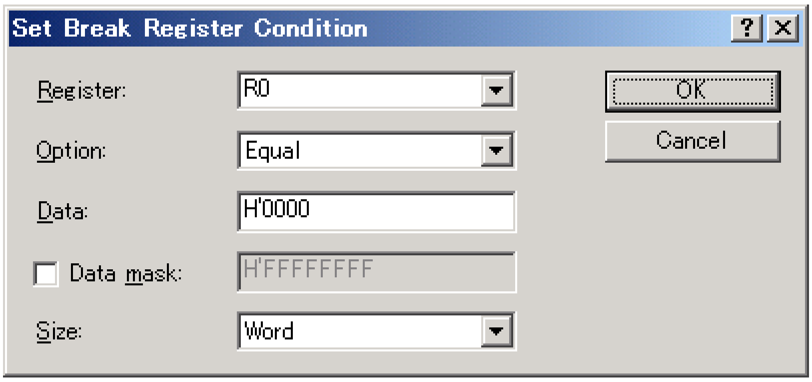


Figure 3.19 Set Break Register Condition Dialog Box

Up to 1024 register break conditions can be specified.

[Register]: Register name for which the break condition is specified

[Option]: Match or mismatch with the data

[Data]: Data value used in the break condition (if no data is input here, a break will occur whenever data is written to the register)

[Data mask]: Mask condition (specifying 0 for a bit masks the bit)

[Size]: Data size

Notes: 1. For string and multiply-and-accumulate instructions, only the last register-access operation is checked for register break conditions.

2. Checking of registers when stack pointer registers are specified as break registers is as shown below.

Register Specification	Accessed Register	
	ISP	**USP**
R0	Checked	Checked
ISP	Checked	Not Checked
USP	Not Checked	Checked

- [Break Sequence]

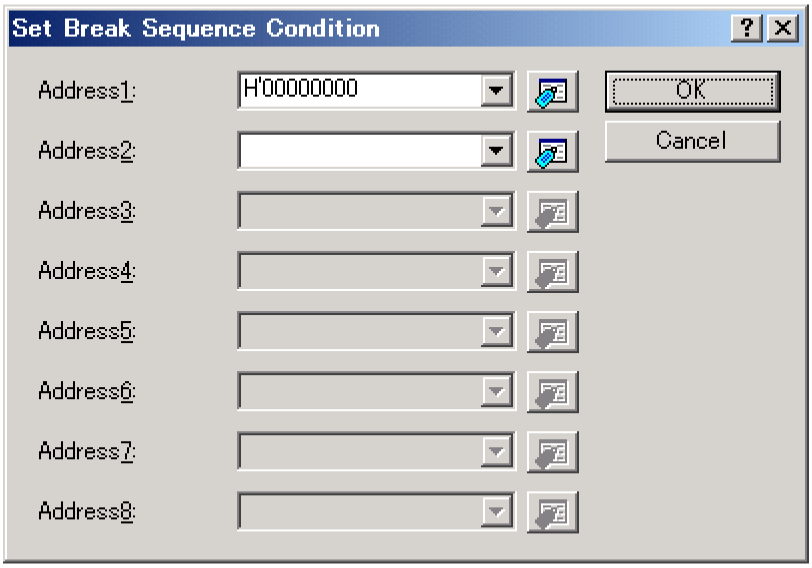


Figure 3.20 Set Break Sequence Condition Dialog Box

Only one sequential break condition can be specified.

[Address1] to [Address8]: Addresses that must be passed as conditions to generate the break (not all of the eight breakpoints have to be set).

- [Break Cycle]

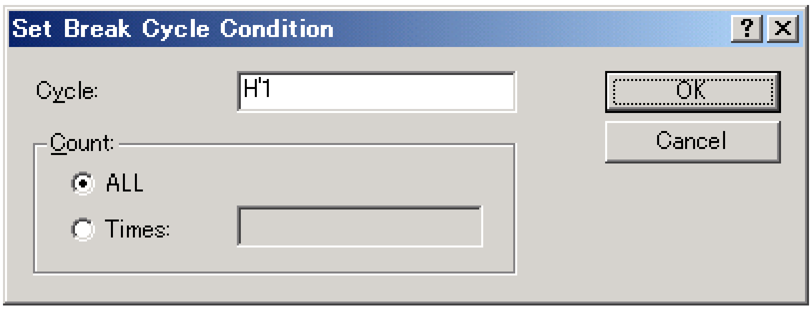


Figure 3.21 Set Number-of-Cycles Break Condition Dialog Box

Up to 1024 number-of-cycles break conditions can be specified.

[Cycle]: Number of cycles required to cause a break (H'1 to H'FFFFFFFF).
The condition will be satisfied by execution for the number of cycles in the [Cycle] setting × n.
However, the specified number of cycles may differ from the number of cycles on which the condition is satisfied.

[Count]: Number of times the break will occur

[ALL]: The break will occur whenever the condition is satisfied.

[Times]: The break will only occur up to the number of times specified as [Times] (1 to 65535; if the prefix is omitted, values input are taken as hexadecimal, and the display is in hexadecimal notation).

- [Break Data]

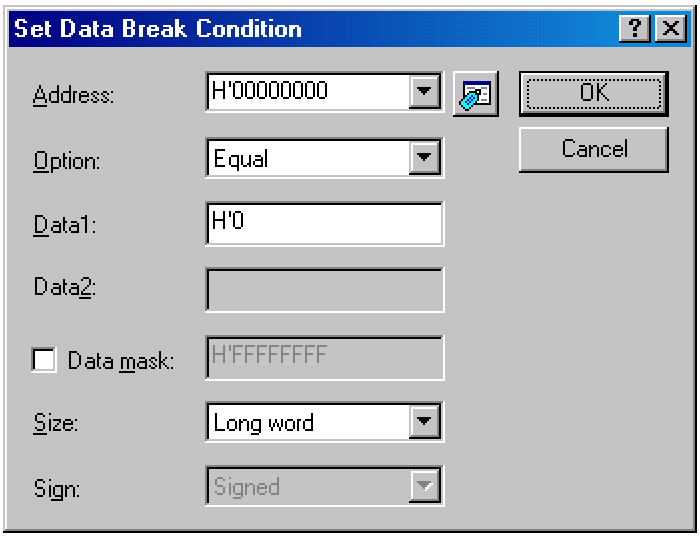


Figure 3.22 Set Break Data Condition Dialog Box

Data break conditions should be set as follows.

Up to 1024 data break conditions can be specified.

[Address]: Address in memory for which the break condition is specified

[Option]: How the data value is used to form the condition that must be satisfied for break generation

[Equal]: The value written to memory matches [Data].

[Not equal]: The value written to memory does not match [Data].

[Inverse sign]*[1]: The sign of the value written to memory is the inverse of that for the previous value.

[Difference]*[1]: The difference between the current and previous values written to memory exceeds [Data].

[GT(>)]: A value written to memory is greater than [Data].

[LT(<)]: A value written to memory is less than [Data].

[GE(>=)]: A value written to memory is greater than or equal to [Data].

[LE(<=)]: A value written to memory is less than or equal to [Data].

[IN]: A value written to memory is within the range between [Data 1] and [Data 2] ([Data 1] <= value written to memory <= [Data 2]).

[OUT]: A value written to memory is outside the range between [Data 1] and [Data 2] (value written to memory < [Data 1] | [Data 2] < value written to memory).

[Data 1]: Data value used in the break condition. When [IN] or [OUT] has been selected, [Data 1] is the beginning of a range for use in the break condition.

[Data 2]: Data value that is the end of a range for use in the break condition. This option is only available when [IN] or [OUT] has been selected.

[Data mask]: Mask condition (specifying 0 for a bit masks the bit). This option is not available when [Inverse sign] or [Difference] has been selected.

[Size]: Data size

[Sign]: Sign of the data.
This option is only available in the following cases.

- The selection for [Option] is [Difference].
- The selection for [Option] is [GT(>)], [LT(<)], [GE(>=)], [LE(<=)], [IN], or [OUT] and the selection for [Size] is [Byte], [Word], or [Long word].

Notes: 1. Since [Inverse sign] and [Difference] require comparison of the data with the value previously written to memory, the break will never occur on the first test after a reset or break generation when either of these conditions has been selected.

2. For string and multiply-and-accumulate instructions, only the last data-access operation is checked for data break conditions.

Selecting an Action in Response to a Break:

If you click on [OK] after setting break conditions in the dialog boxes described on the preceding pages, the [Select Break Type] dialog box is opened again. Select an action type in the [Action type] field of this dialog box.

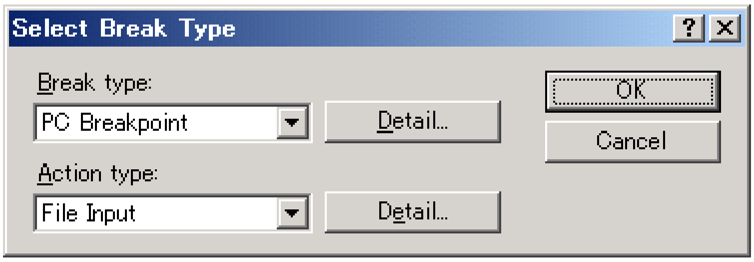


Figure 3.23 Select Break Type Dialog Box

The following options are available:

[Stop]:	Execution of the user program is stopped when the condition is satisfied.
[File Input]:	The contents of a specified file are read out and written to the specified memory when the condition is satisfied.
[File Output]:	The contents of the specified memory are read out and written to the specified file when the condition is satisfied.
[Interrupt]:	Interrupt processing proceeds when the condition is satisfied.
[Trace Trigger]:	Tracing starts when the condition is satisfied (only in cases where triggering of tracing by events has been enabled).

Setting Details of the Action:

Click on [Detail…] after selecting the action type in the [Select Break Type] dialog box. This opens a dialog box that allows you to set details of the selected action (except [Stop] and [Trace Trigger]).

- [File Input]

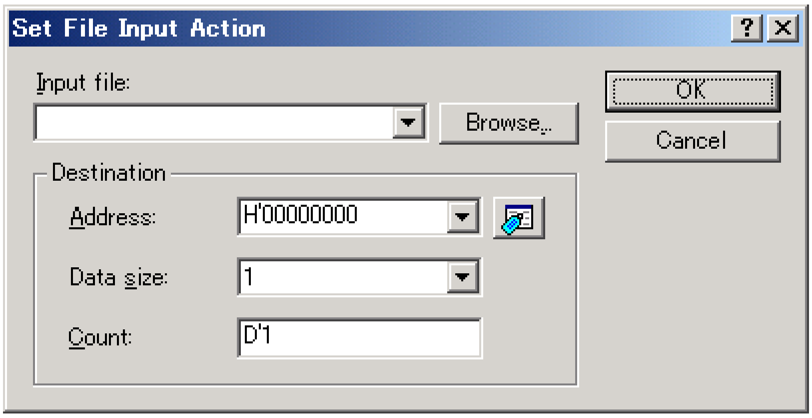


Figure 3.24 Set File Input Action Dialog Box

When the condition is satisfied, data are read out from the specified file and written to the specified location in memory.

[Input file]:	File from which data are to be read out. When reading out by the simulator/debugger reaches the end of the file, reading out recommences from the beginning of the same file.
[Address]:	Memory address to which data should be written.
[Data size]:	Size of each data value in bytes (1/2/4/8).
[Count]:	Number of values to be written (H'1 to H'FFFFFFFF; when the prefix is omitted, values input are taken as decimal, and the display is in decimal notation).

- [File Output]

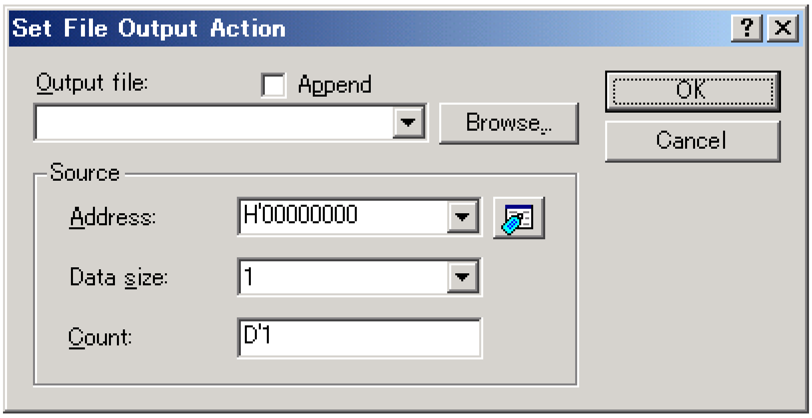


Figure 3.25 Set File Output Action Dialog Box

When the condition is satisfied, the contents at the specified location in memory are written to the specified file.

[Output file]:	Data file to which data are written.
[Append]:	Selects whether the data should be appended to the file if an existing file is specified as the output file.
[Address]:	Memory address to read data from.
[Data size]:	Size of each data value to be read (1/2/4/8).
[Count]:	Number of values to be read (H'1 to H'FFFFFFFF; when the prefix is omitted, values input are taken as decimal, and the display is in decimal notation).

- [Interrupt]

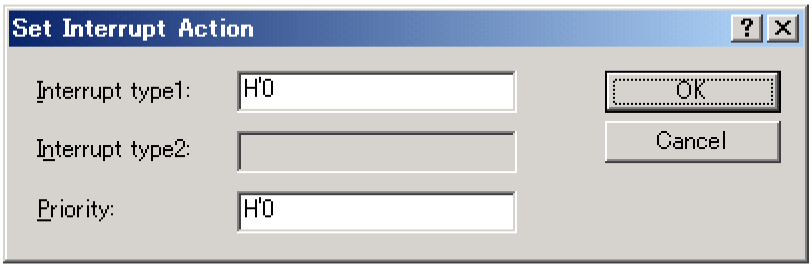


Figure 3.26 Set Interrupt Action Dialog Box

When the condition is satisfied, interrupt processing proceeds. For details, refer to section 2.15, Pseudo-Interrupts.

[Interrupt type 1]: Sets the following values for each CPU (when the prefix is omitted, values input are taken as hexadecimal, and the display is in hexadecimal notation)

[Priority] Interrupt priority (0 to 8 or 0 to H'10: if the prefix is omitted, values input are taken as hexadecimal, and the display is hexadecimal). The value is in the range from 0 to 8 or H'10.
The fast interrupt is specified by the value 8 when the range is from 0 to 8 and H'10 when the range is from 0 to H'10.

- Point for Caution

When the same file is specified for multiple [File Input] actions, the simulator/debugger will read data from the file in the order of condition satisfaction. When the same file is specified for multiple [File Output] actions, the simulator/debugger will write data to the file in the order of condition satisfaction. However, when the same file is specified for [File Input] and [File Output], the only valid action is that for the first condition to be satisfied.

3.6.3 Modifying Breakpoints

Select a breakpoint to be modified, and choose [Edit...] from the pop-up menu to open the [Select Break Type] dialog box, which allows the user to modify the break conditions. The [Edit...] menu is only available when one breakpoint is selected.

3.6.4 Enabling a Breakpoint

Select a breakpoint and choose [Enable] from the pop-up menu to enable the selected breakpoint.

3.6.5 Disabling a Breakpoint

Select a breakpoint and choose [Disable] from the pop-up menu to disable the selected breakpoint. When a breakpoint is disabled, the breakpoint will remain in the list, but a break will not occur when the specified conditions have been satisfied.

3.6.6 Deleting a Breakpoint

Select a breakpoint and choose [Delete] from the pop-up menu to remove the selected breakpoint. To retain the breakpoint but not have it cause a break when its conditions are met, use the [Disable] option (see section 3.6.5, Disabling a Breakpoint).

3.6.7 Deleting All Breakpoints

Choose [Delete All] from the pop-up menu to remove all breakpoints.

3.6.8 Viewing the Source Line for a Breakpoint

Select a breakpoint and choose [Go to Source] from the pop-up menu to open the [Source] or [Disassembly] window at the address of the breakpoint. The [Go to Source] menu is only available when one breakpoint is selected.

3.6.9 Closing Input or Output File

Select a breakpoint and choose [Close File] from the pop-up menu to close the selected [File Input] or [File Output] data file and to reset the address to read the file.

3.6.10 Closing All Input and Output Files

Choose [Close All Files] from the pop-up menu to close all [File Input] and [File Output] data files and to reset the address for reading the file.

3.7 Viewing Trace Information

The simulator/debugger acquires the results of each instruction execution as trace information and displays it in the [Trace] window. The conditions for the trace information acquisition can be specified in the [Trace Acquisition] dialog box.

3.7.1 Opening the Trace Window

To open the [Trace] window, choose [View -> Code -> Trace] or click the [Trace] toolbar button .

3.7.2 Specifying Trace Acquisition Conditions

After the [Trace] window opens, specify the trace acquisition conditions in the [Trace Acquisition] dialog box. To open this dialog box, choose [Acquisition...] from the pop-up menu.

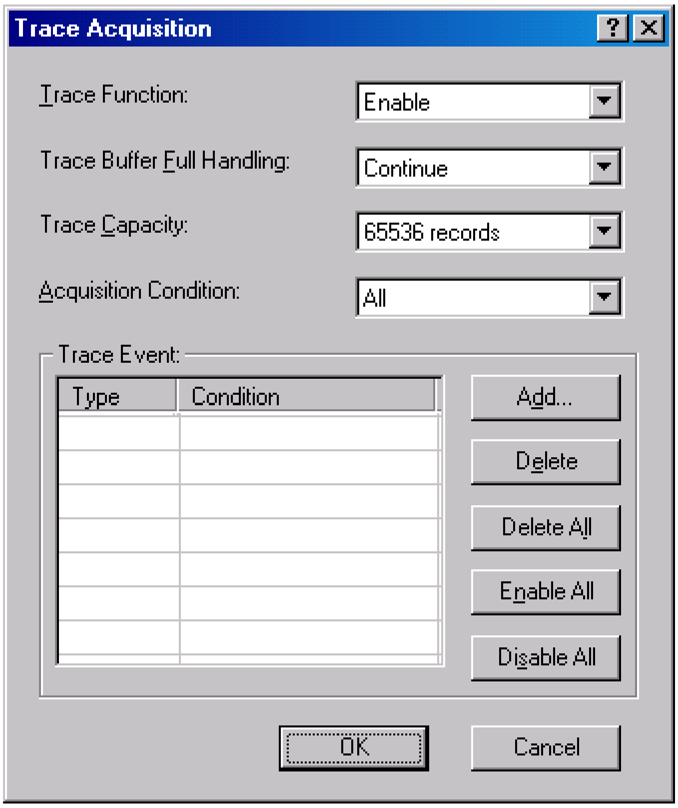


Figure 3.27 Trace Acquisition Dialog Box

This dialog box specifies the conditions for trace information acquisition.

[Trace Function]

[Disable]	Disables trace information acquisition.
[Enable]	Enables trace information acquisition.

[Trace Buffer Full Handling]

[Continue]	Continues acquiring trace information even if the trace information acquisition buffer becomes full.
[Break]	Stops execution when the trace information acquisition buffer becomes full.

[Trace Capacity]

[65536 records]	The size of the trace buffer is 64 Krecords.
[131072 records]	The size of the trace buffer is 128 Krecords.
[262144 records]	The size of the trace buffer is 256 Krecords.
[524288 records]	The size of the trace buffer is 512 Krecords.
[1048576 records]	The size of the trace buffer is 1 Mrecord.

[Acquisition Condition]

[All]	Trace information is acquired until execution of the program is stopped.
[Event Trigger]	A total of 512 records of trace data (i.e. 255 records before the event, the event point itself, and 256 records after the event) are acquired every time the trigger event is encountered.

[Trace Event]

Shows information on the events to start tracing.

The following items are displayed.

[Type]	Event type

[Condition]	Condition
Events of the type selected for [Type] (with the checkbox selected) are valid.	
[Add…]	Opens a dialog box in which events can be added.
[Delete]	Deletes the selected event.
[Delete All]	Deletes all events.
[Enable All]	Enables all events.
[Disable All]	Disables all events.

Modifying a setting in the [Trace Acquisition] dialog box clears the trace information.

Clicking the [OK] button stores the settings. Clicking the [Cancel] button closes this dialog box without modifying the settings.

3.7.3 Setting Events for Tracing

Break conditions are utilized as events for tracing. When a specified event is encountered, trace data around the event point are acquired. Such events can be set in the [Select Break Type] dialog box.

To open the [Select Break Type] dialog box, click on the [Add] button in the [Trace Acquisition] dialog box or select [Add…] from the popup menu opened by right-clicking on the [Software event] tabbed page of the [Event] window.

For details on the conditions and actions to take, refer to section 3.6, Using the Simulator/Debugger Breakpoints.

If you wish to modify the condition of an event for tracing, double-click on the event condition in the [Trace Event] section to open the [Select Break Type] dialog box.

3.7.4 Acquiring Trace Information

After trace acquisition is enabled, trace information is acquired during instruction execution. The acquired information will be displayed in the [Trace] window.
Bus display, disassembly display, and source display or mixtures of these are available.

(1) Bus Display Mode

In the pop-up menu, select [Display Mode -> BUS].

(a) “Acquire All” Mode

In this mode, the [Trace] window shows all trace data from the start to the end of simulation.

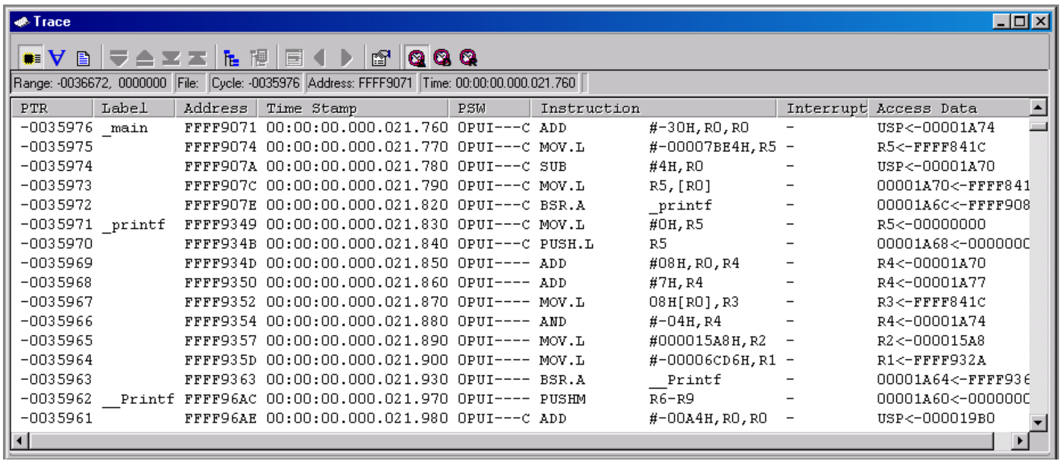


Figure 3.28 Trace Window in “All Acquire” Mode (Bus Display Mode)

This window displays the following trace information items:

[PTR]	Pointer in the trace buffer (0 for the last executed instruction)
[Label]	Label corresponding to the address (only displayed when a label is set).
[Address]	Instruction address
[Time Stamp]	Total instruction execution time (hours: minutes: seconds: milliseconds: microseconds: nanoseconds)
[PSW]	Display the value of the processor status word (PSW) as a mnemonic.
[Instruction]	Instruction mnemonic
[Interrupt]	Interrupt (″Interrupt″ if an interrupt is generated, ″-″ if not)
[Access Data]	Data access information (display format: destination <- accessed data)*

Note: For string and multiply-and-accumulate instructions, this is only the last data to have been accessed.

(b) Event Trigger Mode

In this mode, the [Trace] window shows a set of 512 records of data around an event that has been encountered. To view data on another event, select [Trace Point -> Trace Point Previous] or [Trace Point -> Trace Point Next] from the popup menu of the [Trace] window. After the simulation stops, the [Trace] window shows information on the oldest event.

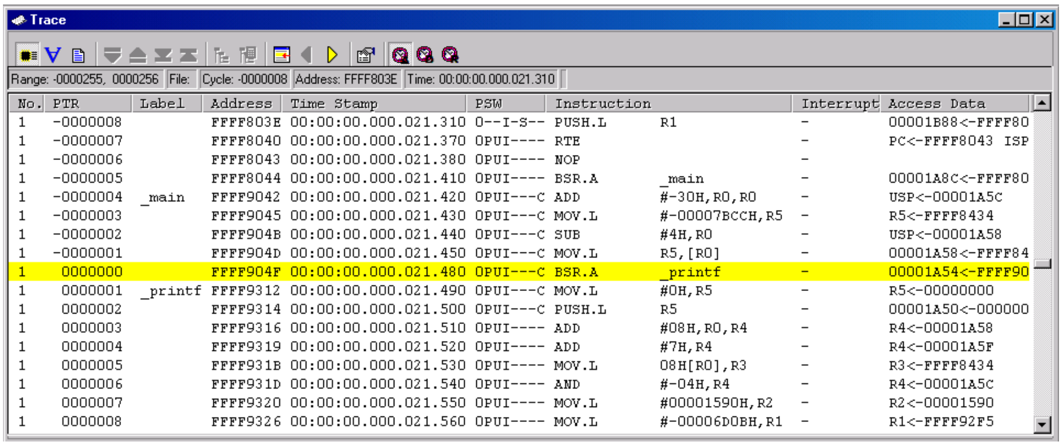


Figure 3.29 Trace Window in Event Trigger Mode (Bus Display Mode)

This window displays the following trace information items:

[No.]	Number of times that the trace point has been encountered once the simulation has started
[PTR]	Pointer to entry in the trace buffer (0 for the trigger of the event)
[Label]	Label corresponding to the address (only displayed when a label is set)
[Address]	Instruction address
[Time Stamp]	Total instruction execution time (hours: minutes: seconds: milliseconds: microseconds: nanoseconds)
[PSW]	Display the value of the processor status word (PSW) as a mnemonic.
[Instruction]	Instruction mnemonic
[Interrupt]	Interrupt ("Interrupt" if an interrupt is generated, "-" if not)
[Access Data]	Data access information (display format: destination <- accessed data)*

Note: For string and multiply-and-accumulate instructions, this is only the last data to have been accessed.

(2) Disassembly Display Mode

In the pop-up menu, select [Display Mode -> DIS]. This enables reference to executed instructions.

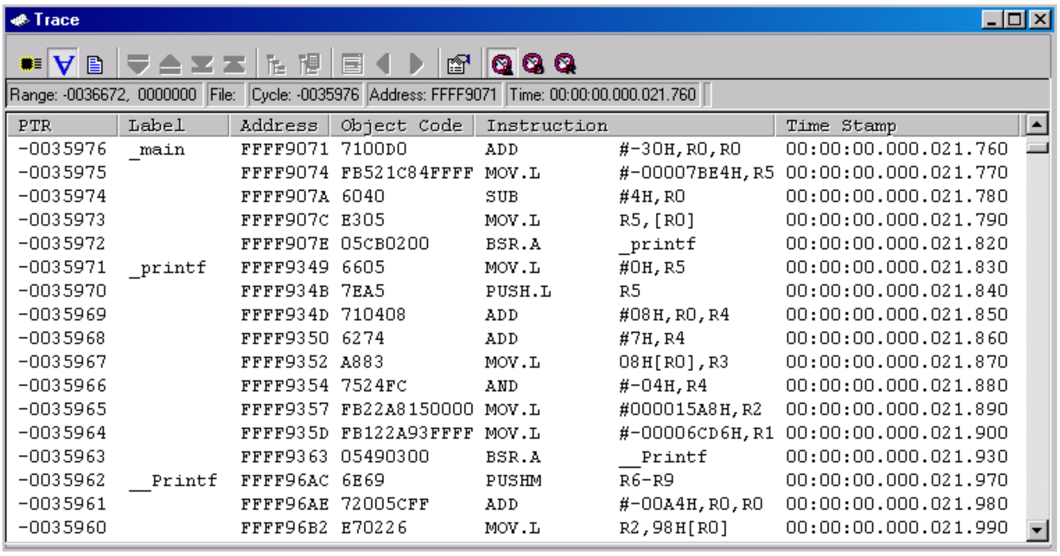


Figure 3.30 Trace Window (Disassembly Display Mode)

(3) Source Display Mode

From the pop-up menu, choose [Display Mode -> SRC]. This display mode allows you to inspect the source program's execution path. The execution path can be verified by stepping through the source within trace data forward or backward from the current trace cycle.

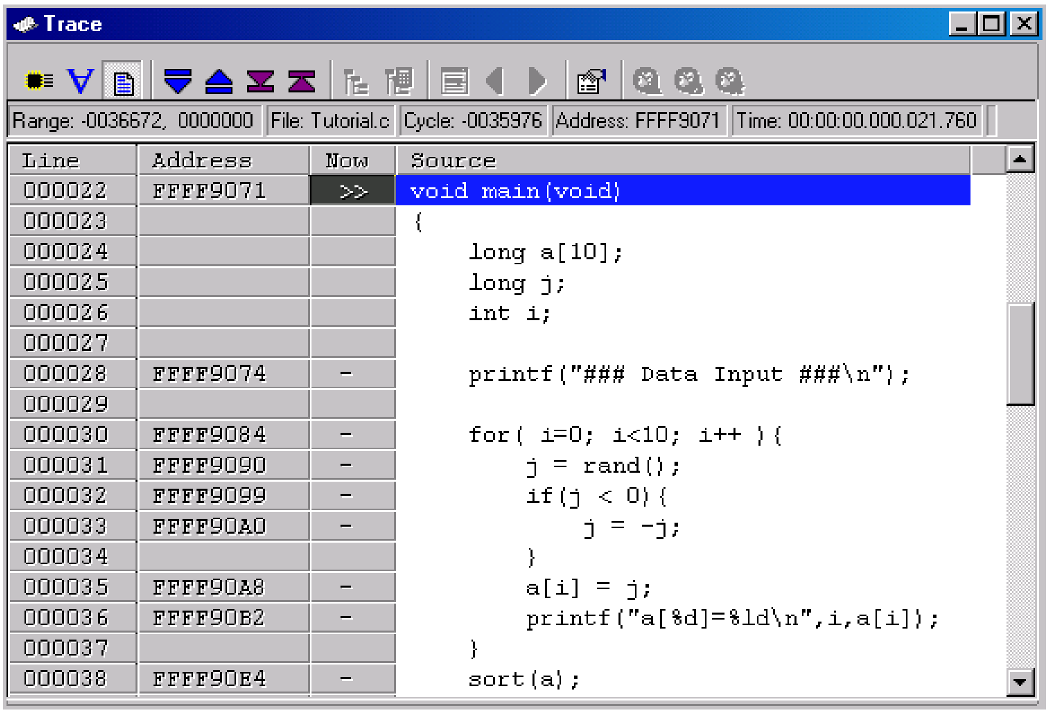


Figure 3.31 Trace Window (Source Display Mode)

(4) Mixed Display Mode

This display mode provides a mixed display of bus, disassemble or source display.

After choosing [Display Mode -> BUS] from the pop-up menu, select [Display Mode -> DIS]. That way, you can produce a bus and disassemble mixed display. In the same way, you can produce a bus and source, a disassemble and source or a bus, disassemble and source mixed display.

To revert to a bus only display after viewing a bus and disassemble mixed display, choose [Display Mode-> DIS] from the pop-up menu again.

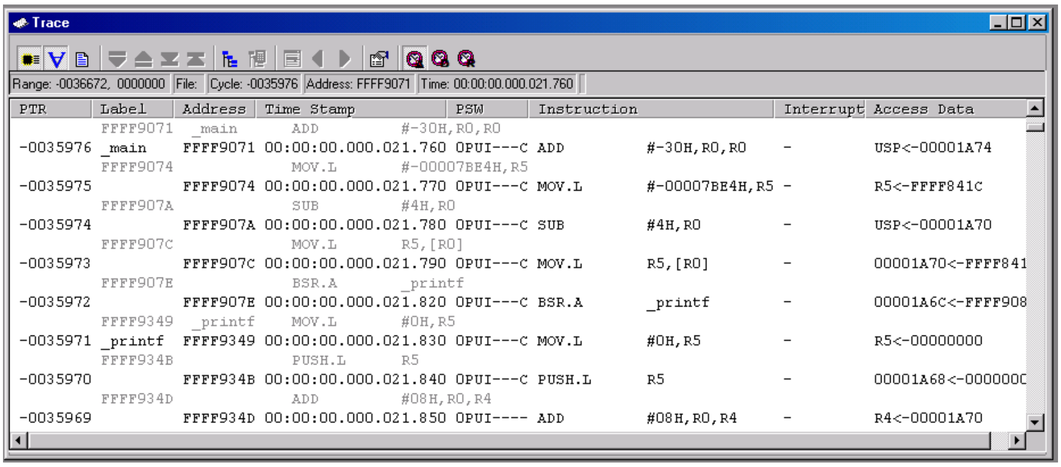


Figure 3.32 Trace Window (Mixed Display Mode)

3.7.5 Searching for Trace Information

Use the [Find] dialog box to search for trace information. To open it, select [Find -> Find...] from the pop-up menu.

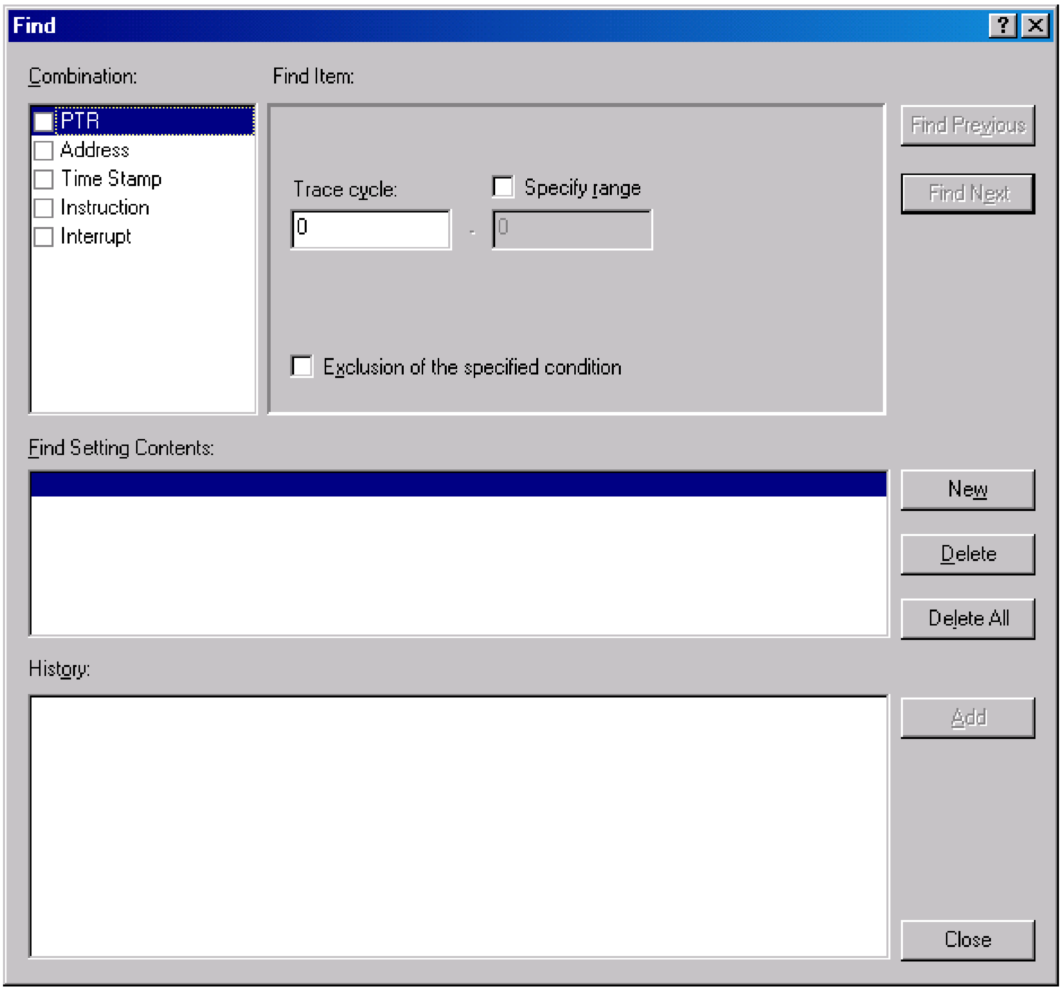


Figure 3.33 Trace Search Dialog Box

Select the conditions required for the search by checking the corresponding buttons in the [Combination] list. Details of the condition can be specified under [Find Item]. When several conditions have been chosen in the [Combination] list, specify the details of the individual conditions. The target of the search is the logical AND of the several conditions.

Item	Contents	Search Conditions
[PTR]	Pointer in the trace buffer	Specified decimal value A range is specifiable. Searching for values other than the specified value is selectable.
[Address]	Instruction address	Specified hexadecimal value A range is specifiable. Searching for values other than the specified value is selectable.
[Time stamp]	Execution time of total instruction	Value specified in an edit box in the unit of time A range is specifiable. Searching for values other than the specified value is selectable.
[Instruction]	Instruction mnemonic	Specified string Searching for values other than the specified value is selectable.
[Interrupt]	Interrupt occurrence	Fixed string: ″Interrupt″ Searching for values other than the specified value is selectable.

The conditions you have set are shown in the [Find Setting Contents] list box.
After setting search conditions, click the [Find Previous] or [Find Next] button to start a search.

When a matching trace record is found by a search, the relevant line in the [Trace] window is highlighted. If no matching trace records are found, a message dialog box is displayed.

When an instance of the trace record was successfully found, choose the [Find Previous] or [Find Next] button from the pop-up menu. The next instance of the trace record will be searched for.

3.7.6 Filtering Trace Information

Use the filter function to extract only the necessary records from the acquired trace information. To use the filter function, select [Auto Filter] from the pop-up menu of the [Trace] window. When [Auto Filter] is turned on, each column of the [Trace] window is marked with an auto-filter arrow [▼]. Click on an arrow and select [Options…] from the drop-down list to bring up the [Options…] dialog box to select the conditions for filtering. The available kinds of filtering and filtering conditions are the same as for the kinds of targets and search conditions for trace record searching.

Note: Filtering is not possible in the event trigger mode.

3.7.7 Clearing the Trace Information

Re-executing instruction simulation after trace information has been acquired clears the trace information.

3.7.8 Saving the Trace Information in a File

The trace information displayed in the [Trace] window is saved in text format and cannot be saved in binary format. Choose [File-> Save...] from the pop-up menu to open the [Save As] dialog box, which allows the user to save the contents of the trace buffer as a text file. A range can be specified based on [Start – End Cycle]. Note that this file cannot be reloaded into the trace buffer.

3.7.9 Viewing the Source File

An [Editor] window corresponding to a selected trace record can be displayed in the source display mode by selecting [File -> Edit Source] from the pop-up menu.

To display another source file in the source display mode of the [Trace] window, use the [Display Source] dialog box. Choose [File -> Display Source] from the pop-up menu to open the [Display Source] dialog box.

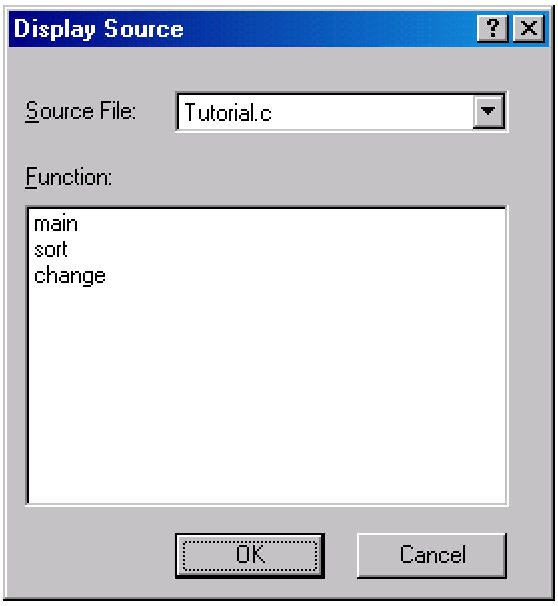


Figure 3.34 Display Source Dialog Box

The source file to be displayed in the [Trace] window can be selected in this dialog box. After setting the conditions, click on the [OK] button to display the source file in the [Trace] window, with the first line of the selected function highlighted.

3.7.10 Switching Timestamp Display

The timestamp displayed in the [Trace] window can be switched to absolute time, differential time or relative time. In the initial state, the timestamp is displayed in absolute time.

(1) Absolute time

From the pop-up menu, choose [Time -> Absolute Time] or click the [Absolute Time] button  in the toolbar.

(2) Differential time

From the pop-up menu, choose [Time -> Differences] or click the [Differences] button  in the toolbar.

(3) Relative time

From the pop-up menu, choose [Time -> Relative Time] or click the [Relative Time] button  in the toolbar.

3.7.11 Showing the History of Function Execution

To show the history of function execution from the acquired trace information, choose [Function Execution History -> Function Execution History] from the pop-up menu or click the [Function Execution History] button in the toolbar. An upper pane of the window will be displayed. (Initially, this window is blank.) When you choose [Analyze Execution History] from the pop-up menu or click the [Analyze Execution History] button in the toolbar, the simulator/debugger starts analyzing the execution history from the end of the trace result and shows the result in a tree structure.

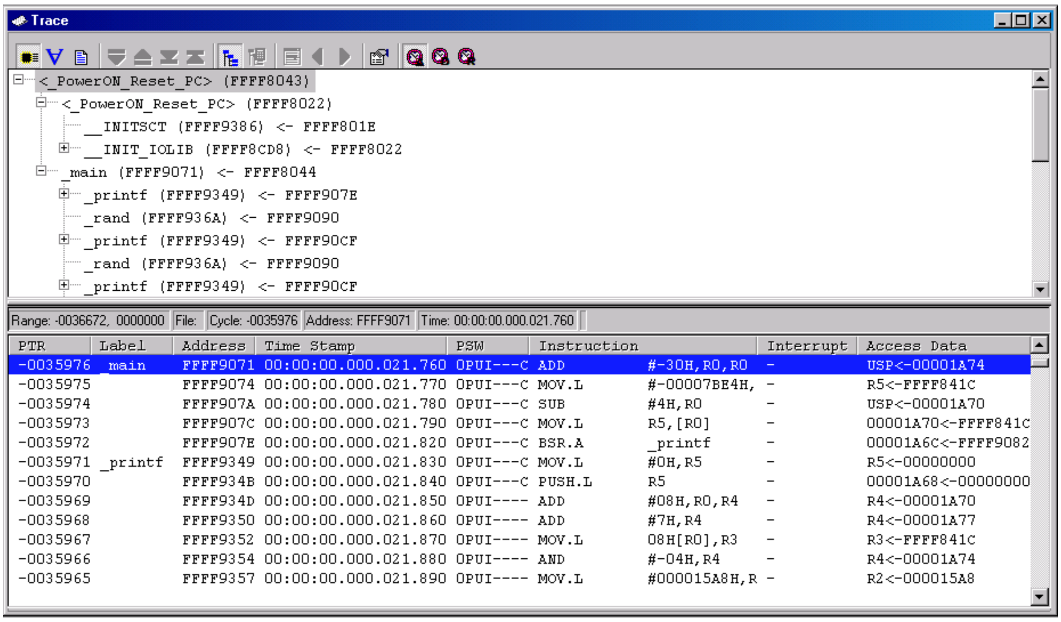


Figure 3.35 Trace Window

The lower pane of the window shows the trace result beginning with the cycle in which the function selected in the upper pane was called.

Note: The history of function execution is not displayable in the event trigger mode.

3.8 Viewing the Profile Information

The profile function enables function-by-function measurement of the performance of the application program in execution. This makes it possible to identify parts of an application program that degrade its performance and the reasons for such degradation.

The HEW displays the results of measurement in three windows, according to the method and purpose of viewing the profile data.

3.8.1 Stack Information Files

The profile function allows the HEW to read the stack information files (extension: .SNI) which are output by the optimizing linkage editor (ver. 7.0 or later). Each of these files contains information related to the calling of static functions in the corresponding source file. Reading the stack information file makes it possible for the HEW to display information related to the calling of functions without executing the user application (i.e. before measuring the profile data). However, this feature is not available when [Setting->Only Executed Functions] is checked in the pop-up menu of the [Profile] window.

When the HEW does not read any stack information files, only the data on the functions executed during measurement will be displayed by the profile function.

To make the linkage editor create a stack information file, choose [Build -> RX600 Standard Toolchain...], and select [Other] from the [Category] list box and check the [Stack information output] box in the [Link/Library] sheet of the [Standard Toolchain] dialog box.

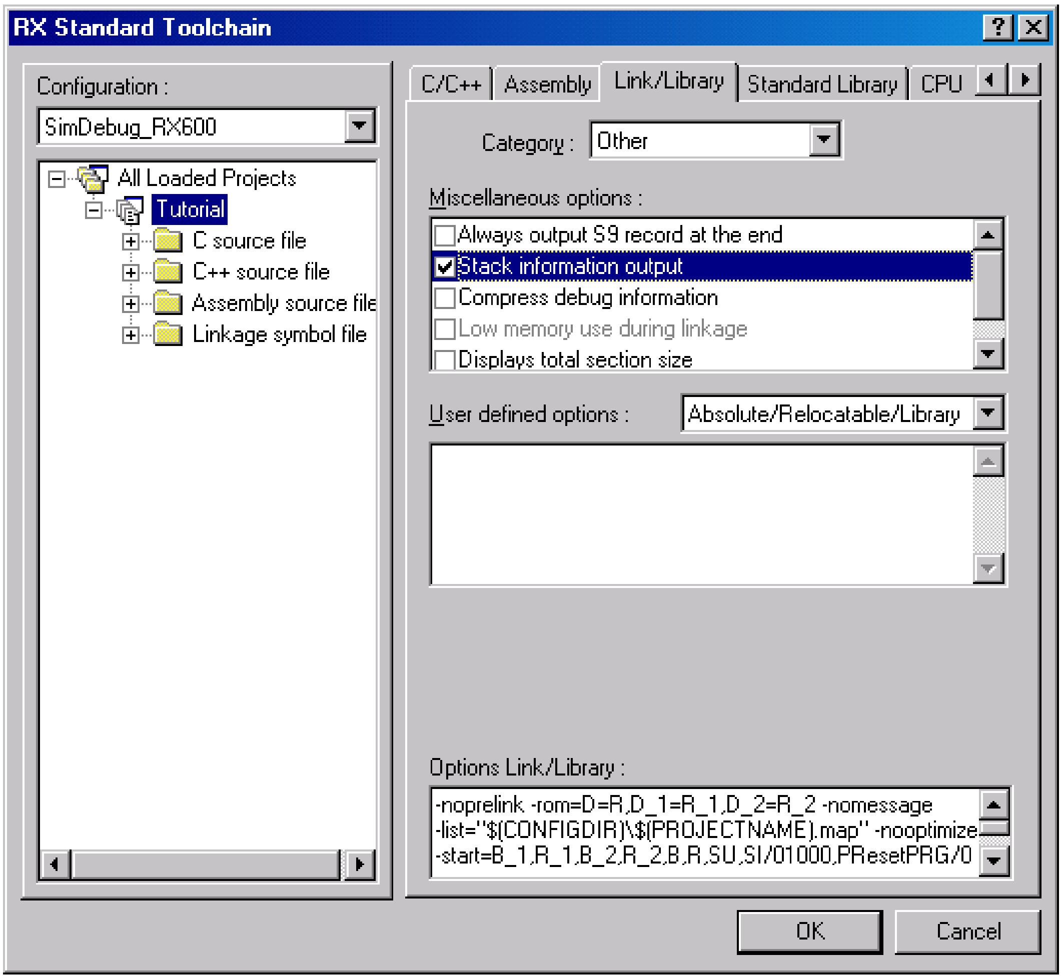


Figure 3.36 Standard Toolchain Dialog Box (1)

3.8.2 Loading Stack Information Files

You can select whether or not to read the stack information file in a message box for confirmation that is displayed when a load module is loaded. Clicking the [OK] button of the message box loads the stack information file. The message box for confirmation will be displayed when:

- There are stack information files (extension: .SNI)
- The [Load Stack Information Files (SNI files)] check box is checked in the [Confirmation] tab of the [Options] dialog box (figure 3.37) that can be opened by choosing [Setup -> Options…] from the main menu.

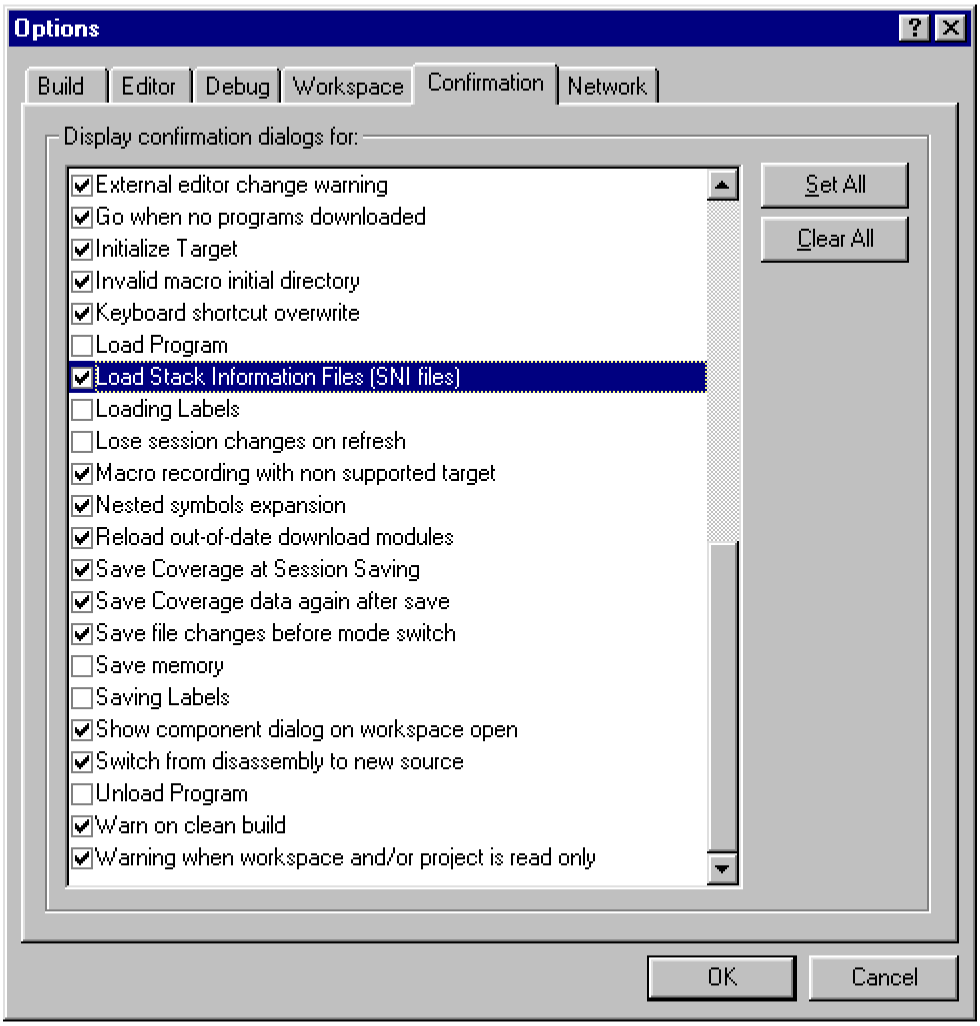


Figure 3.37 Options Dialog Box

3.8.3 Enabling the Profile

Choose [View->Performance->Profile] to open the [Profile] window.

Choose [Enable Profiler] from the pop-up menu of the [Profile] window. The item on the menu will be checked.

3.8.4 Specifying Measurement Mode

You can specify whether to trace functions calls while profile data is acquired. When function calls are traced, the relations of function calls during user program execution are displayed as a tree diagram. When not traced, the relations of function calls cannot be displayed, but the time for acquiring profile data can be reduced.

To stop tracing function calls, choose [Disable Tree (Not traces function call)] from the pop-up menu in the [Profile] window (a check mark is shown to the left of the menu item).

When acquiring profile data of the program in which functions are called in a special way, such as task switching in the OS, stop tracing function calls.

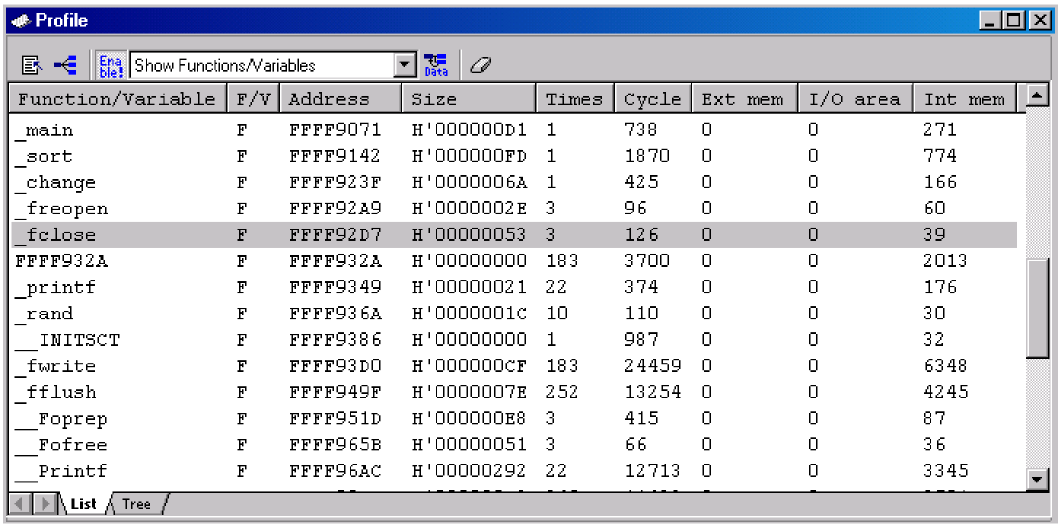
3.8.5 Executing the Program and Checking the Results

After the user program has been executed and execution has been halted, the results of measurement are displayed in the [Profile] window.

The [Profile] window has two sheets; a [List] sheet and a [Tree] sheet.

3.8.6 List Sheet

This sheet lists functions and global variables and displays the profile data for each function and variable.



Function/Variable	F/V	Address	Size	Times	Cycle	Ext mem	I/O area	Int mem
_main	F	FFFF9071	H'000000D1	1	738	0	0	271
_sort	F	FFFF9142	H'000000FD	1	1870	0	0	774
_change	F	FFFF923F	H'0000006A	1	425	0	0	166
_freopen	F	FFFF92A9	H'0000002E	3	96	0	0	60
_fclose	F	FFFF92D7	H'00000053	3	126	0	0	39
FFFF932A	F	FFFF932A	H'00000000	183	3700	0	0	2013
_printf	F	FFFF9349	H'00000021	22	374	0	0	176
_rand	F	FFFF936A	H'0000001C	10	110	0	0	30
__INITSCT	F	FFFF9386	H'00000000	1	987	0	0	32
_fwrite	F	FFFF93D0	H'000000CF	183	24459	0	0	6348
_fflush	F	FFFF949F	H'0000007E	252	13254	0	0	4245
__Foprep	F	FFFF951D	H'000000E8	3	415	0	0	87
__Fofree	F	FFFF965B	H'00000051	3	66	0	0	36
__Printf	F	FFFF96AC	H'00000292	22	12713	0	0	3345

Figure 3.38 List Sheet

Clicking the column header sorts the items in an alphabetical or ascending/descending order. Clicking the [Function/Variable] or [Address] column displays the source program corresponding to the address in the line.

Right-clicking on the mouse within the window displays a pop-up menu. For details on this pop-up menu, refer to section 3.8.7, Tree Sheet.

3.8.7 Tree Sheet

This sheet displays the relation of function calls along with the profile data that are values when the function is called. This sheet is available when [Disable Tree (Not traces function call)] is not selected from the pop-up menu in the [Profile] window.

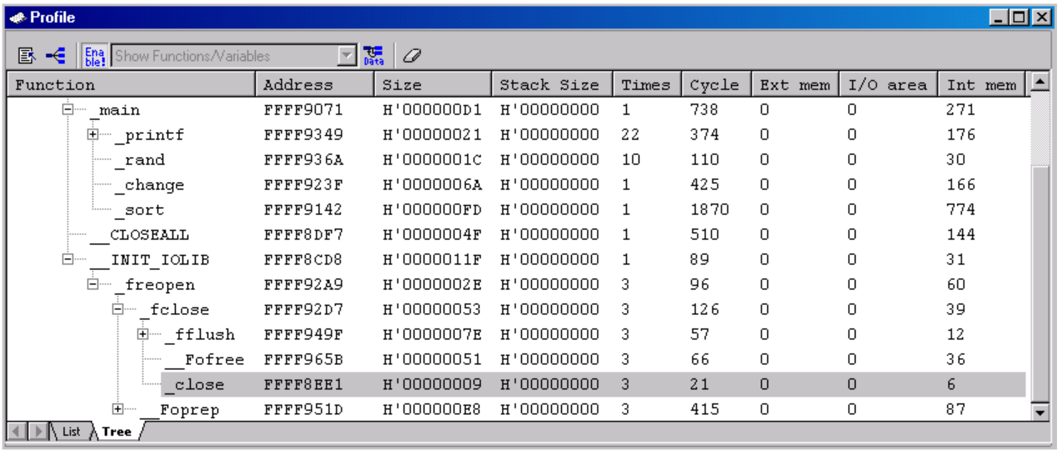


Figure 3.39 Tree Sheet

Double-clicking a function in the [Function] column expands or reduces the tree structure display. The expansion or reduction is also provided by the “+” or “-” key. Double-clicking the [Address] column displays the source program corresponding to the specific address.

Right-clicking on the mouse within the window displays a pop-up menu. Supported menu options are as follows:

- View Source
  Displays the source program or disassembled memory contents for the address in the selected line.
- View Profile-Chart
  Displays the [Profile-Chart] window focused on the function in the specified line.
- Enable Profiler
  Toggles acquisition of profile data. When profile data acquisition is enabled, a check mark is shown to the left of the menu text.
- Not trace the function call
  Stops tracing function calls while profile data is acquired. This menu is used when acquiring profile data of the program in which functions are called in a special way, such as task switching in the OS.
  To display the relation of function calls in the [Tree] sheet of the [Profile] window, acquire profile data without selecting this menu. In addition, do not select this menu when optimizing the program by the optimizing linkage editor using the acquired profile information file.
- Find…
  Displays the [Find Text] dialog box to find a character string in the [Function] column. Search is started by entering a character string to be found in the edit box and clicking [Find Next] or pressing the Enter key.
- Find Data…
  Displays the [Find Data] dialog box.

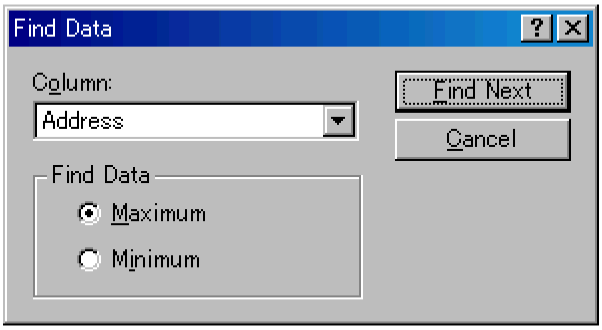


Figure 3.40 Find Data Dialog Box

By selecting the column to be searched in the [Column] combo box and the search type in the [Find Data] group then pressing [Find Next] button or Enter key, search is started. If the [Find Next] button or the Enter key is input repeatedly, the second larger data (the second smaller data when Minimum is specified) is searched for.

- Clear Data
  Clears the number of times functions are called and the profile data. Data in the [List] sheet of the [Profile] window and the data in the [Profile-Chart] window are also cleared.
- Output Profile Information Files…
  Displays the [Save Profile Information Files] dialog box. Profiling results are saved in a profile information file (.pro extension).
- Output Text File…
  Displays the [Save Text of Profile Data] dialog box. Displayed contents are saved in a text file.
- Setting
  This menu has the following submenus (the menus available only in the [List] sheet are also included).
  — Show Functions/Variables
    Displays both functions and global variables in the [Function/Variable] column.
  — Show Functions
    Displays only functions in the [Function/Variable] column.
  — Show Variables
    Displays only global variables in the [Function/Variable] column.
  — Only Executed Functions
    Only displays the executed functions. If a stack information file (.sni extension) output from the optimizing linkage editor does not exist in the directory where the load module is located, only the executed functions are displayed even if this check box is not checked.
  — Include Data of Child Functions
    Sets whether or not to display information for a child function called in the function as profile data.
- Properties...
  This menu cannot be used in this simulator/debugger.

3.8.8 Profile-Chart Window

The [Profile-Chart] window displays the relation of calls for a specific function. This window displays the specified function in the middle, with the callers of the function on the left and the callees of the function on the right. The numbers of times the function calls the functions or is called by the functions are also displayed in this window.

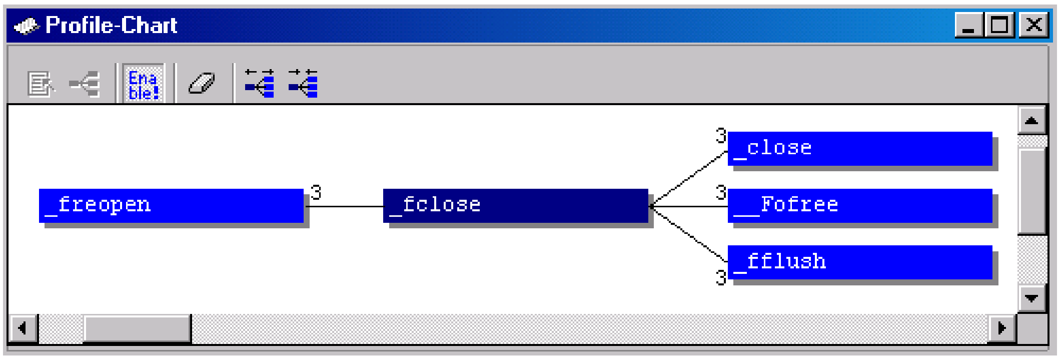


Figure 3.41 Profile-Chart Window

Right-clicking the mouse within the window displays a pop-up menu. Supported menu options are as follows:

- View Source
  Displays the source program or disassembled memory contents for the address of the function on which the cursor is placed when the right-hand mouse button is clicked. If the cursor is not placed on a function when the right-hand mouse button is clicked, this menu option remains gray.
- View Profile-Chart
  Displays the [Profile-Chart] window for the specific function on which the cursor is placed when the right-hand mouse button is clicked. If the cursor is not placed on a function when the right-hand mouse button is clicked, this menu option remains gray.
- Enable Profiler
  Toggles acquisition of profile data. When profile data acquisition is enabled, a check mark is shown to the left of the menu text.
- Clear Data
  Clears the number of times functions are called. Data in the [List] and [Tree] sheets of the [Profile] window are also cleared.
- Multiple View
  If a further [Profile-Chart] window is opened while an existing [Profile-Chart] window is already open, this option selects whether a new window is opened or the new data is displayed in the existing window. When a check mark is shown to the left of this menu item, a new window will be opened.
- Output Profile Information Files...
  Displays the [Save Profile Information Files] dialog box. Profiling results are saved in a profile information file (.pro extension). The optimizing linkage editor optimizes user programs according to the profile information in this file. For details on optimization with the profile information, refer to the user's manual for the optimizing linkage editor.
- Expands Size
  Redo the display with larger intervals between functions. The "+" key can also be used to do this.
- Reduces Size
  Redo the display with smaller intervals between functions. The "-" key can also be used to this.

3.8.9 Types and Purposes of Displayed Data

The profile function is able to acquire the following information:

Address	You can view the locations in memory to which the functions are allocated. Sorting the list of functions and global variables in order of their addresses allows the user to view the way the items are allocated in the memory space.
Size	Sorting in order of size makes it easy to find small functions that are frequently called. Setting such functions as inline may reduce the overhead of function calls.
Stack Size	When there is deep nesting of function calls, pursue the route of the function calls and obtain the total stack size for all of the functions on that route to estimate the amount of stack being used.
Times	Sorting by the number of calls or accesses makes it easy to identify the frequently called functions and frequently accessed global variables.
Profile Data	Measurement of a variety of CPU-specific data is also available as follows:

- [Cycle] (the number of cycles execution requires)
- [Ext_mem] (the number of external memory accesses)
- [I/O_area] (the number of internal I/O area accesses)
- [Int_mem] (the number of internal memory accesses)

The number of cycles is calculated by subtracting the number of cycles until the specified function is called from the number of cycles when the return instruction for the function is called.

Note: A string or multiply-and-accumulate instruction is treated as accessing data only once (i.e. the last data-access operation).

3.8.10 Creating Profile Information Files

To create a profile information file, choose the [Output Profile Information Files…] menu option from the pop-up menu. The [Save Profile Information Files] dialog box is displayed. Pressing the [Save] button after selecting a file name will write the profile information to the selected file. Pressing the [Save All] button will write the profile information to all of the profile information files.

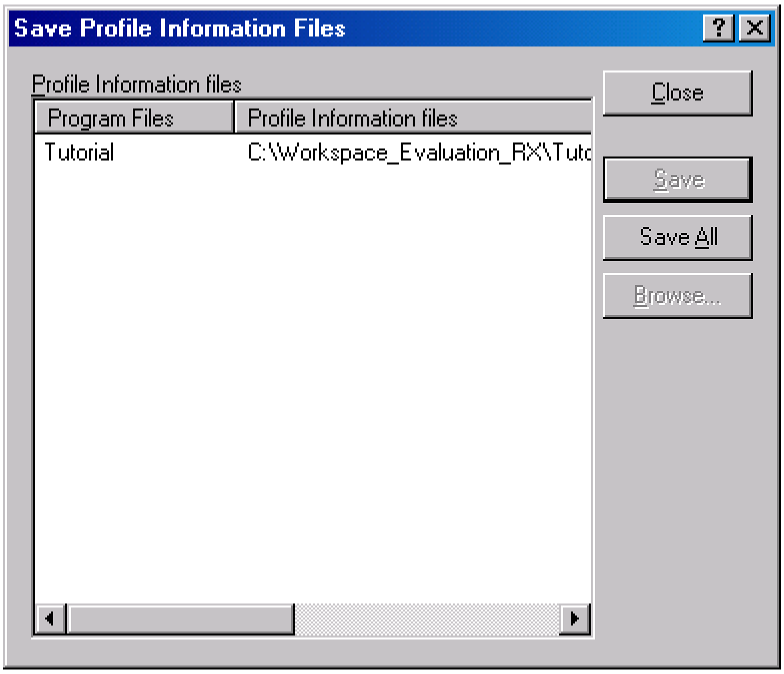


Figure 3.42 Save Profile Information Files Dialog Box

3.8.11 Notes

1. The number of executed cycles for an application program as measured by the profile function includes a margin of error. The profile function only allows the measurement of the proportions of execution time that the functions occupy in the overall execution of the application program. Use the Performance Analysis function to precisely measure the numbers of executed cycles.
2. The names of the corresponding functions may not be displayed when the profile information on a load module with no debugging information is measured.
3. The stack information file (extension: .SNI) must be in the same directory as the load module file (extension: .ABS).
4. It is not possible to store the results of measurement.
5. It is not possible to modify the results of measurement.

3.9 Analyzing Performance

Use the [Performance Analysis] window to select a function name and analyze the performance.

3.9.1 Opening the Performance Analysis Window

Choose [View -> Performance -> Performance Analysis] or click the [PA] toolbar button to open the [Performance Analysis] window.

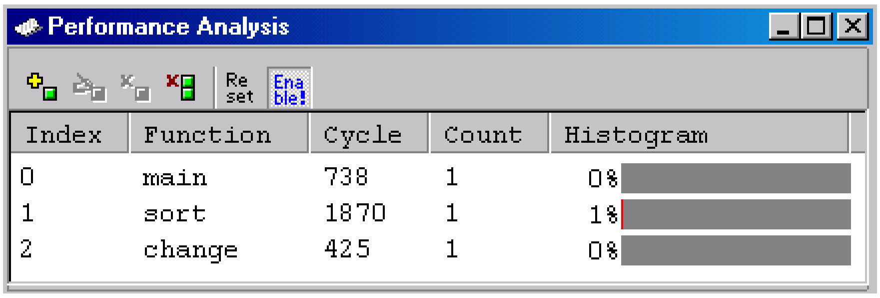


Figure 3.43 Performance Analysis Window

This window displays the number of execution cycles required for each specified function.

The number of execution cycles is calculated as follows:

Execution cycles = total number of execution cycles when execution returns from the function
– total number of execution cycles when the target function is called

The following items are displayed:

[Index] Index number of the set condition

[Function] Name of the function to be measured (or the start address of the function)

[Cycle] Total number of instruction execution cycles

[Count] Total number of calls for the function

[Histogram] Ratio of number of cycles for execution of the function to the number of cycles for execution of the whole program, displayed as a percentage and histogram

3.9.2 Specifying a Target Function

After the [Performance Analysis] window is open, choose [Add Range...] from the pop-up menu or press the Insert key to open the [Performance Option] dialog box, which allows the user to specify a function to be analyzed.

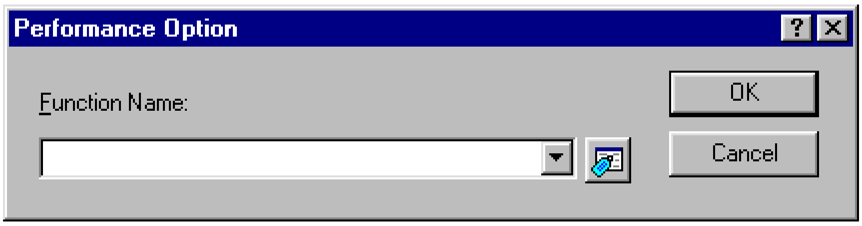


Figure 3.44 Performance Option Dialog Box

This dialog box specifies a function (including a label) to be evaluated. Up to 255 functions can be specified in total.

Clicking the [OK] button stores the setting. Clicking the [Cancel] button closes this dialog box without setting the function to be evaluated.

Select a function that has been set and choose [Edit Range] from the pop-up menu or press the Enter key to open the [Performance Option] dialog box and to change the function to be evaluated.

3.9.3 Starting Performance Data Acquisition

Choose [Enable Analysis] from the pop-up menu (a check mark is shown to the left of [Enable analysis]) to start acquiring performance analysis data.

3.9.4 Resetting Data

Choose [Reset Counts/Times] from the pop-up menu to clear the current performance analysis data.

3.9.5 Deleting a Target Function

Select a function and choose [Delete Range] from the pop-up menu to delete the selected target function and to recalculate the data within other ranges. The selected function can also be deleted by the Delete key.

3.9.6 Deleting All Target Functions

Choose [Delete All Ranges] from the pop-up menu to delete all the current target functions to be evaluated and to clear the performance analysis data.

3.9.7 Saving the Currently Displayed Contents

The contents currently displayed in the window can be saved in a text file. Select [Save to File…] from the pop-up menu.

3.10 Measuring Code Coverage

The [Coverage] window acquires code coverage information (C0 coverage and C1 coverage) in the range specified by the user, and displays the information.

3.10.1 Opening the Coverage Window

Choose [View -> Code -> Coverage...] or click the [Coverage] toolbar button to open the [Open Coverage] dialog box.

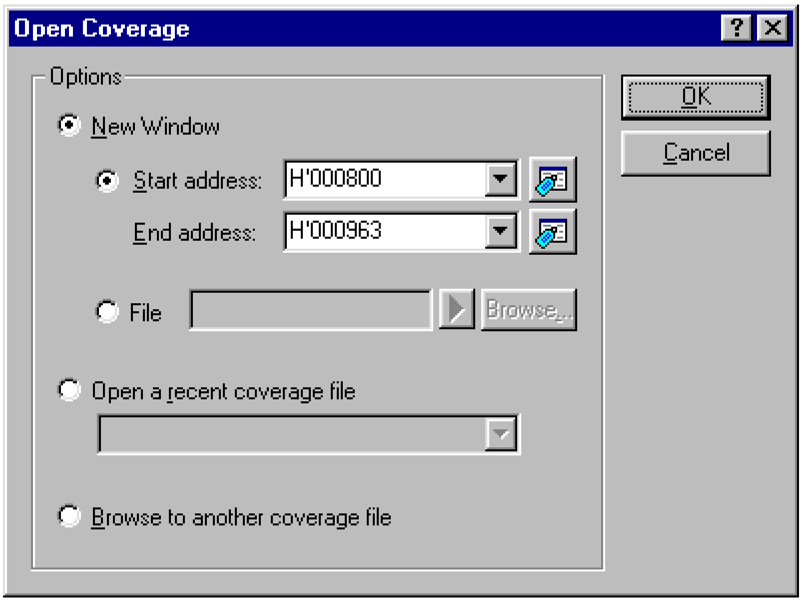


Figure 3.41 Open Coverage Dialog Box

This dialog box specifies the coverage measuring range. To set coverage for a new range, the following two ways are available:

- Specifying the start and end addresses on the new window

[Start Address] Start address of coverage information display
(When a prefix is omitted, values input are taken as hexadecimal.)

[End Address] End address of coverage information display
(When a prefix is omitted, values input are taken as hexadecimal.)

- Specifying the file on the new window

[File] Source file whose extension is .C or .CPP in the current project.
Functions in the specified file can be set as the coverage range.
If the extension of the file is omitted, .C is complemented.
The file that has other extensions than .C or .CPP cannot be specified.
A placeholder or the [Browse…] button is available.

To use the settings saved in a coverage information file, choose the file from [Open a recent coverage file], or open a file open dialog box by [Browse to another coverage file] and select the file. When [Open a recent coverage file] is selected, up to four recent files that have been saved are displayed.

Clicking [OK] opens the [Coverage] window.

When the [Coverage] window has already been displayed for specifying address, settings are added in the window.

- Coverage window (specifying address)

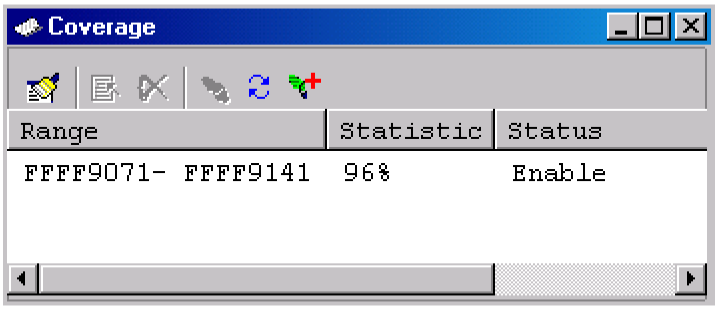


Figure 3.42 Coverage Window (Specifying Address)

This window displays the coverage range and statistical information. The following items are displayed:

[Range] Address range

[Statistic] Percentage of the instructions executed within the range

[Status] Enable or Disable status of the coverage range

When the [Coverage] window is closed, the acquired coverage information and the conditions to acquire information will be cleared.

- Coverage window (specifying source file)

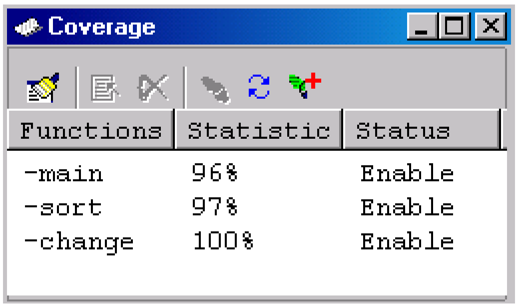


Figure 3.43 Coverage Window (Specifying Source File)

This window displays the coverage range and statistical information. The following item is displayed:

[Functions] List of functions

[Statistic] Percentage of the instruction executed within the function

[Status] Enable or Disable status of the respective function

Note: The functions can be sorted by their names or percentage, either in ascending or descending order, by clicking the column tab ([Functions] or [Statistic]).

When the [Coverage] window is closed, the acquired coverage information and the conditions to acquire information will be cleared.

3.10.2 Acquiring All Coverage Information

Choose [Enable All] from the pop-up menu and execute the user program to acquire all coverage information. By default, [Enable All] is selected.

3.10.3 Clearing All Coverage Information

Choosing [Clear All] from the popup menu clears all the coverage information that has been acquired.

3.10.4 Viewing the Source Window

Choose [View Source] from the pop-up menu to open the [Editor] window and to display the [Editor] window corresponding to the cursor location in the [Coverage] window.

3.10.5 Specifying the New Coverage Range

Choose [Add Range...] from the pop-up menu to open the [Open Coverage] dialog box (figure 3.41). For the [Open Coverage] dialog box, refer to section 3.10.1, Opening the Coverage Window.

3.10.6 Changing the Coverage Range

- Specifying the coverage range with an address

Choose the coverage range and [Edit Range...] from the pop-up menu to open the [Coverage Range] dialog box.

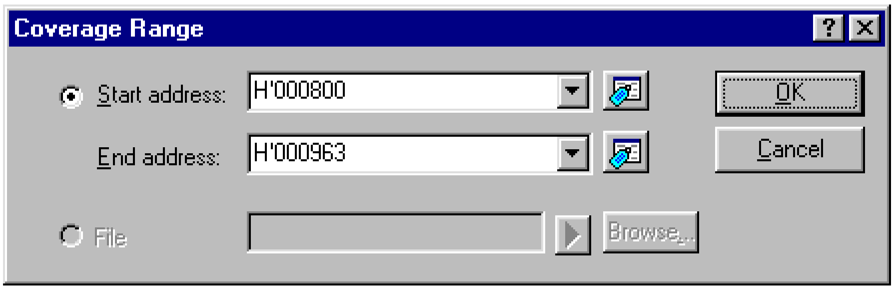


Figure 3.44 Coverage Range Dialog Box (Specifying Address)

This dialog box specifies the condition to acquire instruction execution information. The following items can be specified.

[Start address] Start address (When a prefix is omitted, values input are taken as hexadecimal.)

[End address] End address (When a prefix is omitted, values input are taken as hexadecimal.)

Clicking [OK] changes the coverage range.

- Specifying the coverage range with a source file

Choose [Edit Range...] from the pop-up menu to open the [Coverage Range] dialog box.

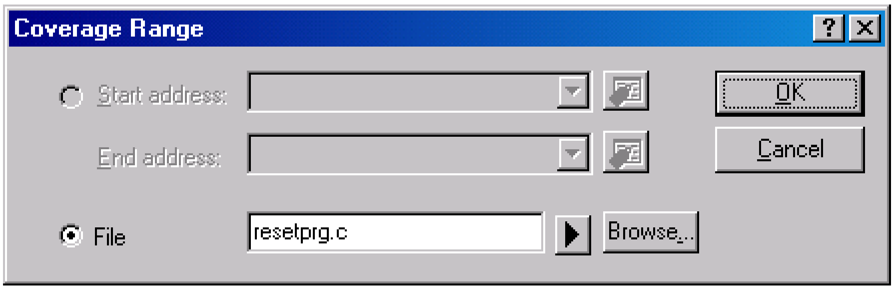


Figure 3.45 Coverage Range Dialog Box (Specifying Source File)

This dialog box specifies the condition to acquire instruction execution information. The following items can be specified.

[File] Source file whose extension is .C or .CPP in the current project.
Functions in the specified file can be set as the coverage range.
If the extension of the file is omitted, .C is complemented.
The file that has other extensions than .C or .CPP cannot be specified.
A placeholder or the [Browse…] button is available.

Clicking [OK] changes the coverage range.

3.10.7 Deleting the Selected Coverage Range

Select a coverage range and choose [Delete Range] from the pop-up menu to delete the selected coverage range.

3.10.8 Acquiring Coverage Information

Specify a coverage range, choose [Enable Coverage] from the pop-up menu, and execute the user program to acquire coverage information. By default, [Enable Coverage] is selected.

3.10.9 Clearing Coverage Information

Specify a coverage range and choose [Clear Data] from the pop-up menu to clear the acquired coverage information.

3.10.10 Saving Coverage Information in a File

Choose [Save Data...] from the pop-up menu to open the [Save Data] dialog box, which allows the user to save the coverage information in a file.

Figure 3.46 Save Data Dialog Box

This dialog box specifies the location and name of a coverage information file to be saved. The placeholder or the [Browse...] button can be used.

If a file name extension is omitted, .COV is automatically added. If a file name extension other than .COV or .TXT is specified, an error message will be displayed.

3.10.11 Loading Coverage Information from a File

Choose [Load Data...] from the pop-up menu to open the [Load Data] dialog box, which allows the user to load the coverage information from a file.

Figure 3.47 Load Data Dialog Box

This dialog box specifies the location and name of a coverage information file to be loaded. The placeholder or the [Browse...] button can be used.

Only .COV files can be loaded. If a file name extension other than .COV is specified, an error message will be displayed.

3.10.12 Updating the Information

Choose [Refresh] from the pop-up menu to update the [Coverage] window to the latest information.

3.10.13 Confirmation Request Dialog Box

A confirmation request dialog box will appear when [Clear All], [Clear Data], [Edit Range...], or [Delete Range] is clicked or an attempt is made to close the [Coverage] window.

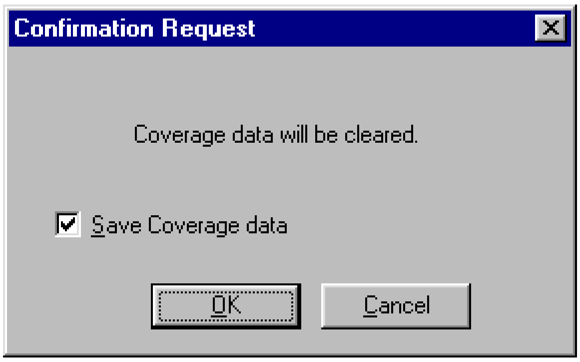


Figure 3.48 Confirmation Request Dialog Box

Clicking [OK] clears the coverage data. Choosing [Save Coverage data] opens the [Save Data] dialog box (figure 3.46) to save the coverage data in a file before it is cleared.

3.10.14 Save Coverage Data Dialog Box

When [File -> Save Session] menu option is clicked, the [Save Coverage Data] dialog box will appear, which allows the user to save the [Coverage] window data in separate files or a single file.

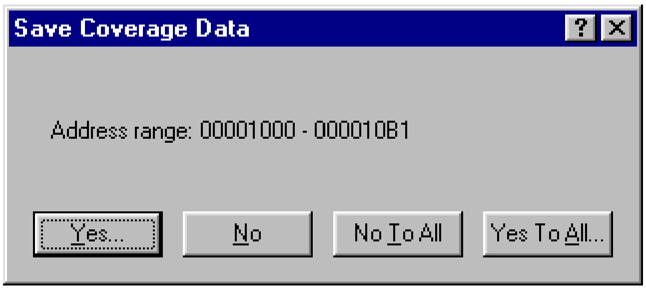


Figure 3.49 Save Coverage Data Dialog Box

When multiple [Coverage] windows are open, a [Save Coverage Data] dialog box will appear for each open coverage window.

Clicking the [No To All] button closes the dialog box without saving any coverage data.

Clicking the [Yes To All] button saves the data of all [Coverage] windows in a single file.

Note: If a file is specified for the coverage range, not all [Coverage] windows can be saved in a single file.

3.10.15 Displaying the Coverage Information in the Editor Window

The coverage information is reflected to the [Editor] window by highlighting the coverage columns corresponding to the source lines of executed instructions. When the coverage settings are modified in the [Coverage] window, the coverage column display will be updated.

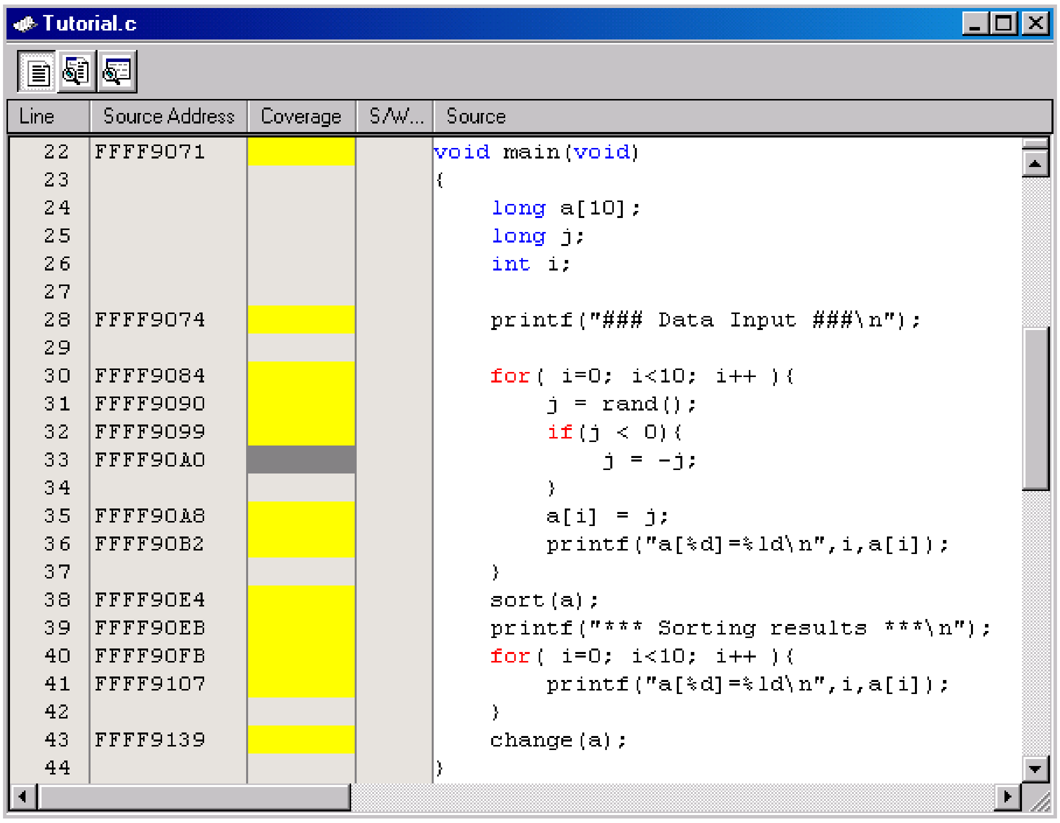


Figure 3.50 [Coverage] Column (Source)

3.10.16 Displaying the Coverage Information in the [Disassembly] Window

The coverage information is reflected to the [Disassembly] window by highlighting the [Coverage – ASM] columns corresponding to the disassembly lines of executed instructions. When the coverage settings are modified in the [Coverage] window, the [Coverage – ASM] column display will be updated.

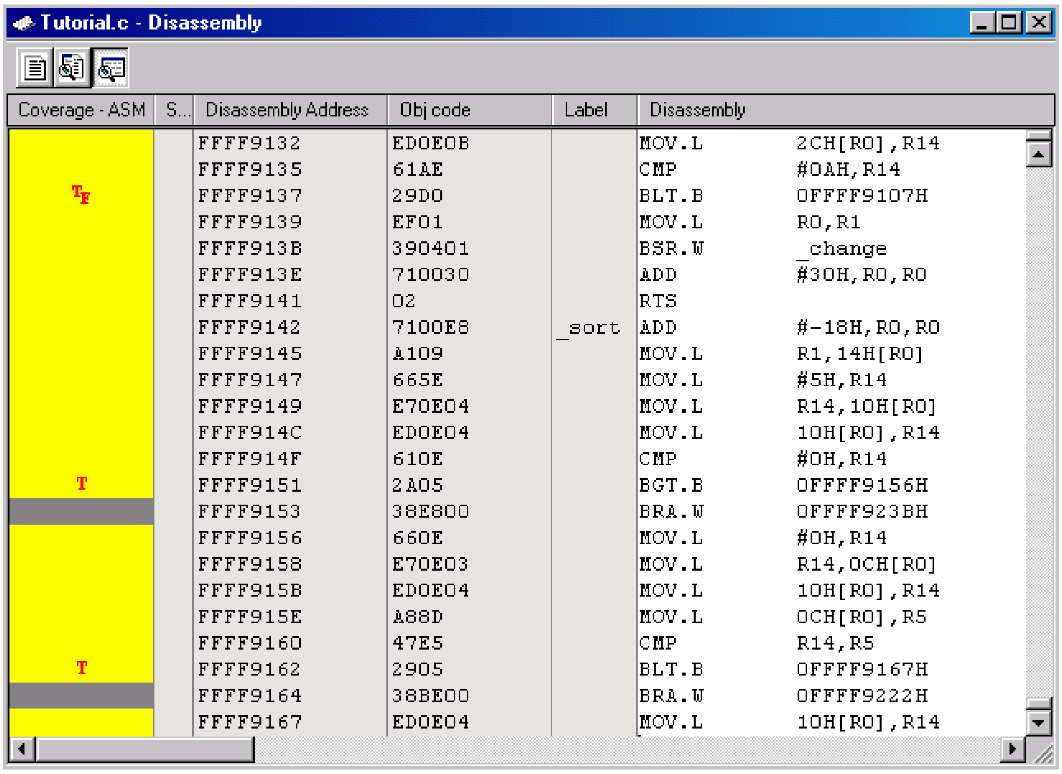


Figure 3.51 Coverage Column (Disassembly)

3.11 Generating a Pseudo-Interrupt Manually

Windows [Trigger] and [GUI I/O] allow the user to generate a pseudo-interrupt manually by pressing a button on the window.

3.11.1 [Trigger] Window

Choose [View -> CPU -> Trigger] or click the [Trigger] toolbar button  to open the [Trigger] window.

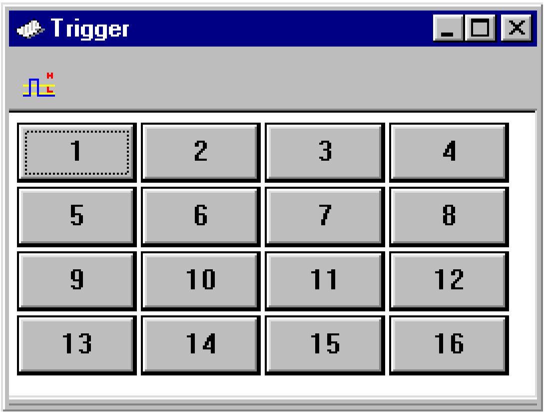


Figure 3.52 Trigger Window

This window displays trigger buttons that generate pseudo-interrupts manually. The details of the interrupt to be generated by pressing each trigger button can be specified in the [Trigger Setting] dialog box.

Up to 256 trigger buttons can be used.

For details on the interrupt processing in the simulator/debugger, refer to section 2.15, Pseudo-Interrupts.

- Setting a trigger button

Choose [Setting...] from the pop-up menu to open the [Trigger Setting] dialog box and to specify the details of the pseudo-interrupt to be generated by pressing each trigger button.

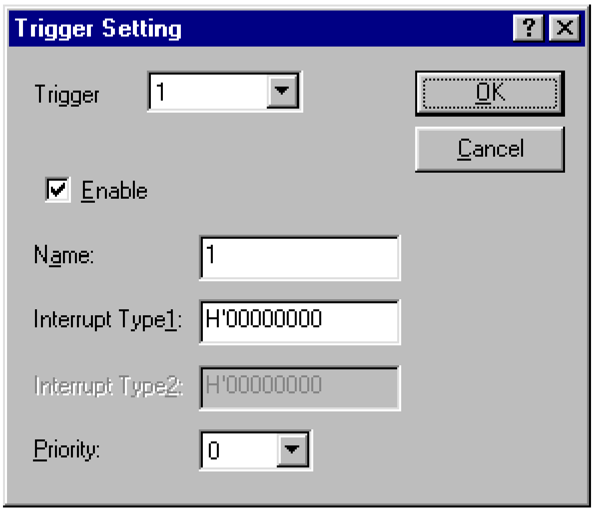


Figure 3.53 Trigger Setting Dialog Box

This dialog box allows the user to specify the details of the pseudo-interrupt to be generated by pressing each trigger button.

[Trigger]	Selects the trigger button to be specified in detail
[Name]	Specifies a name for the selected trigger button; the name will be displayed in the [Trigger] window
[Enable]	Checking this box enables the trigger button.
[Interrupt type1]	Interrupt vector number
[Priority]	Interrupt priority (0 to 8 or 0 to H'10; when the prefix is omitted, values input are taken as hexadecimal, and the display is in hexadecimal notation). The fast interrupt is specified by the value 8 when the range is from 0 to 8 and H'10 when the range is from 0 to H'10.

Clicking the [OK] button stores the setting. Clicking the [Cancel] button closes this dialog box without setting the details of the interrupt.

Note: If the [Cancel] button is clicked after multiple trigger button settings are modified, the modifications of all those buttons are canceled.

- Changing the number of trigger buttons

Specify the number of trigger buttons displayed in the [Trigger] window in the [Number of Buttons] submenu in the pop-up menu. [4], [16], [64], or [256] can be selected.

- Changing the size of trigger buttons

Specify the size of trigger buttons displayed in the [Trigger] window in the [Size] submenu in the pop-up menu. [Large], [Normal], or [Small] can be selected.

3.11.2 [GUI I/O] Window

Choose [View -> Graphic -> GUI I/O] or click the [GUI I/O] toolbar button to open the [GUI I/O] window.

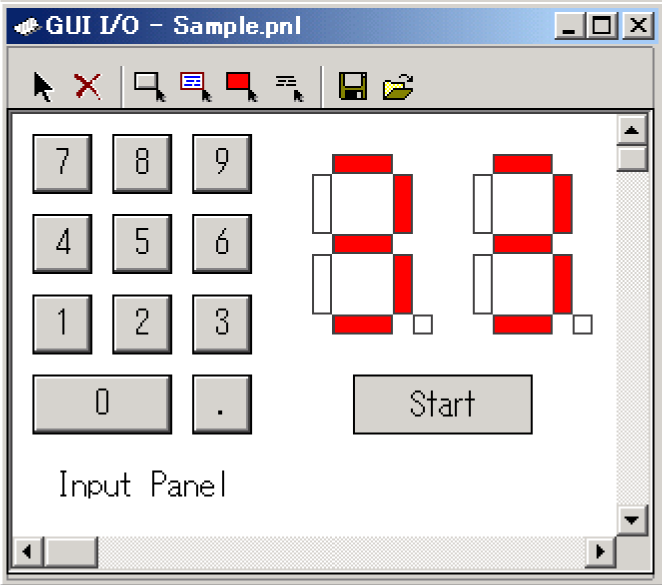


Figure 3.54 GUI I/O Window

This window displays buttons that generate pseudo-interrupts manually. The details of the interrupt to be generated by pressing each button can be specified in the [Set Button] dialog box.

For details on the interrupt processing in the simulator/debugger, refer to section 2.15, Pseudo-Interrupts.

- Setting a button

Choose [Create Button] from the pop-up menu or click the [Create Button] toolbar button ( ). The mouse cursor turns into a "+" symbol. Create the button by dragging the mouse cursor from a higher-left to a lower-right position.

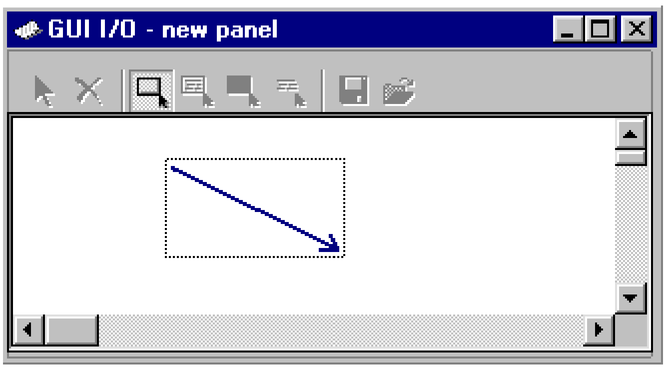


Figure 3.55 GUI I/O Window (Create Button)

Double-click the created button to open the [Set Button] dialog box.

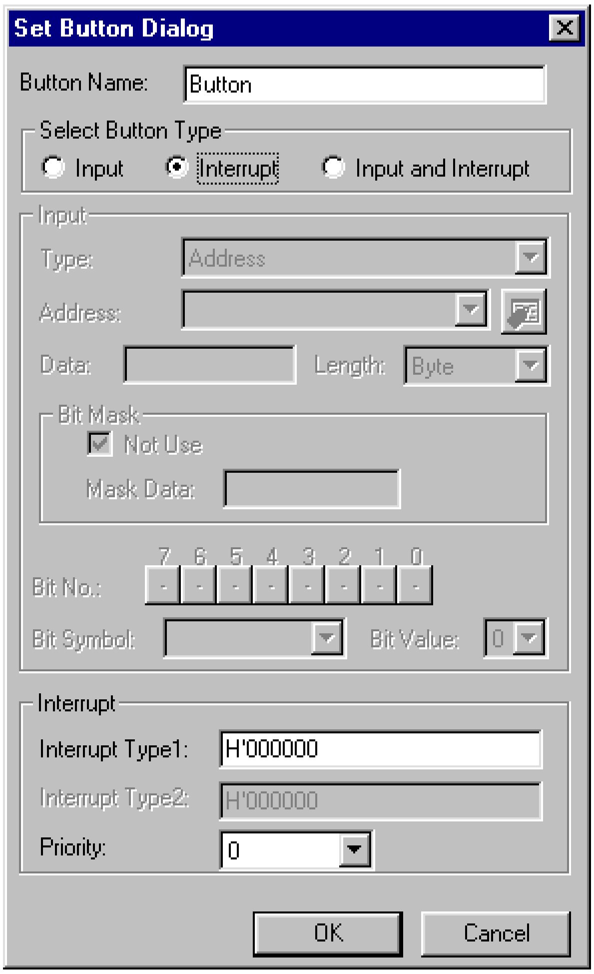


Figure 3.56 Set Button Dialog Box

This dialog box allows the user to specify the details of the pseudo-interrupt to be generated by pressing each button.

[Button Name] Specifies a name for the button; the name will be displayed in the [GUI I/O] window

[Select Button Type] Select [Input] or [Input and Interrupt].

[Interrupt] [Interrupt Type1] Interrupt vector number

[Priority] Interrupt priority (0 to 8, H'0 to H'10; when the prefix is omitted, values input are taken as hexadecimal, and the display is in hexadecimal notation).
The fast interrupt is specified by the value 8 when the range is from 0 to 8 and H'10 when the range is from 0 to H'10.

3.12 Standard I/O and File I/O Processing

Use the [Simulated I/O] window to enable the simulation for standard I/O and file I/O from the user program.

3.12.1 Opening the Simulated I/O Window

Choose [View -> CPU -> Simulated I/O] or click the [Simulated I/O] toolbar button to open the [Simulated I/O] window.

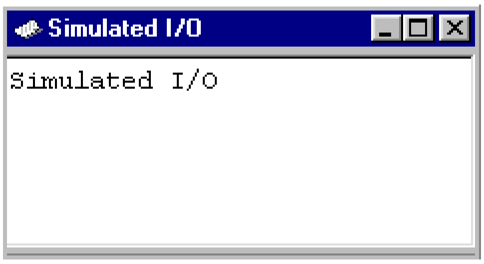


Figure 3.57 Simulated I/O Window

The standard output from the user program is displayed in this window. The key input from this window is handled as the standard input to the user program.

3.12.2 I/O Functions

Table 3.1 lists the supported I/O functions.

Table 3.1 I/O Functions

No.	Function Code	Function Name	Description
1	H'21	GETC	Inputs one byte from the standard input
2	H'22	PUTC	Outputs one byte to the standard output
3	H'23	GETS	Inputs one line from the standard input
4	H'24	PUTS	Outputs one line to the standard output
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

To perform I/O processing, use the [Simulated I/O Address] in the [Simulator System] dialog box (refer to section 3.3.2, Modifying the Simulator System) in the following procedure.

1. Set the address specialized for I/O processing in the [Simulated I/O Address], select [Enable] and execute the program.
2. When detecting a subroutine call instruction (BSR or JSR), that is, a simulated I/O instruction to the specified address during user program execution, the simulator/debugger performs I/O processing with the value in R1 and R2 as the parameters.

- Set the function code (table 3.1) in the R1 register

MSB 1 byte 1 byte LSB

H'01	Function code	———	———

- Set the parameter block address in the R2 register

MSB LSB

Parameter block address

- Reserve the parameter block and input/output buffer areas

Each parameter of the parameter block must be accessed in the parameter size.

After the I/O processing, the simulator/debugger resumes simulation from the instruction that follows the simulated I/O instruction.

Refer to the simulator/debugger help about each I/O function.

The following shows an example for inputting one character as a standard input (from a keyboard). Label SYS_CALL is specified as the simulated I/O address.

```
          MOV.L     #01210000h, R1
          MOV.L     #PARM, R2
          MOV.L     #SYS_CALL, R3
          JSR       R3
STOP      NOP
SYS_CALL  NOP
PARM      .LWORD    INBUF
          .SECTION  B, DATA
INBUF     .BLKB     2
          .END
```

3.13 Creating a Virtual I/O Panel

The simulator/debugger has a GUI I/O function for simulating a simple key-input or key-output panel of the user target system in a window. This virtual I/O panel is created in the [GUI I/O] window. That is, virtual buttons and virtual LEDs are arranged in this window to allow the input and output of data.

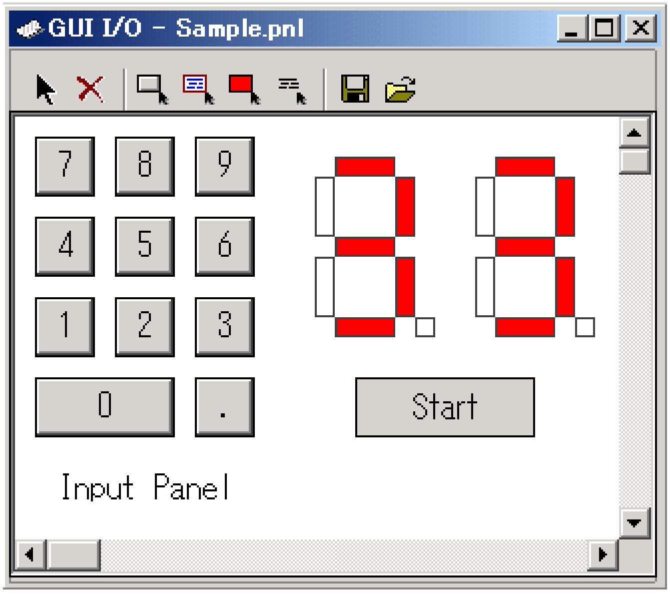


Figure 3.58 Example of a GUI I/O Window

3.13.1 Opening the [GUI I/O] Window

Choose [View -> Graphic -> GUI I/O] or click the [GUI I/O] toolbar button to open the [GUI I/O] window.

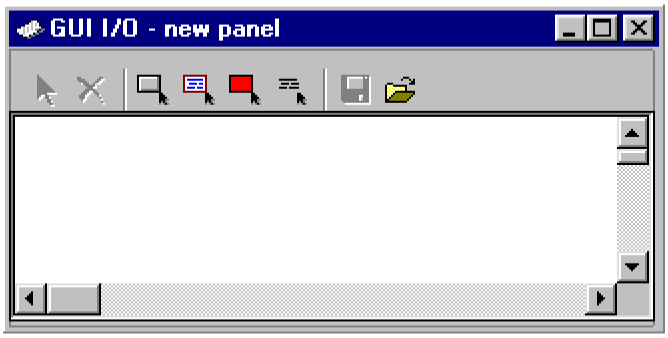


Figure 3.59 [GUI I/O] Window

This window is used to arrange the following items.

Button: Press a button for input of data to a virtual port or generation of a virtual interrupt.

Label: A character string which is shown when the value written to a selected address or bit was the specified value and hidden otherwise.

LED: A defined region in which a specified color is displayed (representing illumination of a LED) when the value written to a selected address or bit was the specified value.

Text: A region for the display of a text string.

3.13.2 Creating a Button

Click on the  button of the toolbar or choose [Create Button] from the pop-up menu. The mouse cursor turns into a "+" symbol. Create the button by dragging the mouse cursor from a higher-left to a lower-right position.

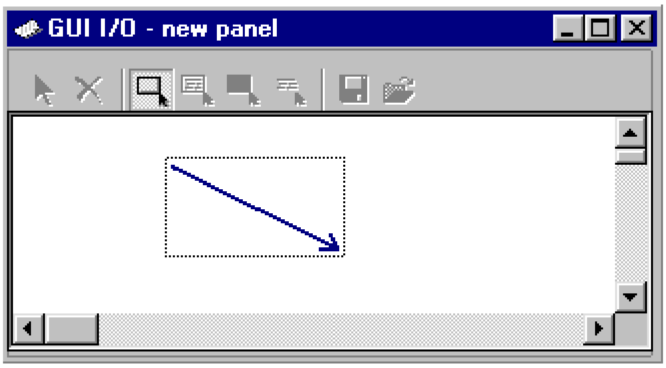


Figure 3.60 GUI I/O Window (Create Button)

- Specifying the event generated by clicking the button

Press the  button on the toolbar and double-click on the created button to open the [Set Button] dialog box.

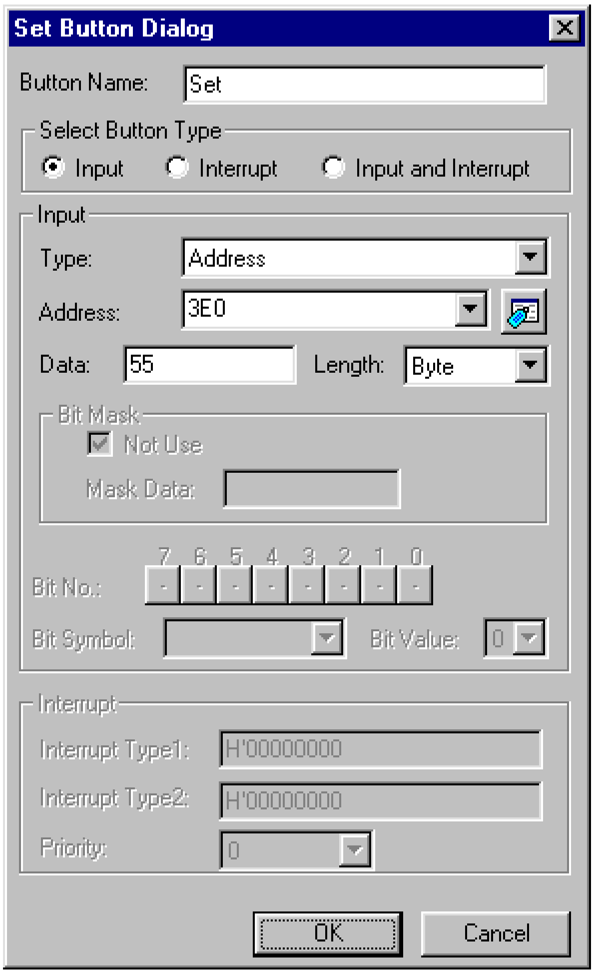


Figure 3.61 Set Button Dialog Box

Enter the name of the button, input port address, and input data. The button name must not include white space.

3.13.3 Creating a Label

Click on the button of the toolbar or choose [Create Label] from the pop-up menu. The mouse cursor turns into a “+” symbol. Drag the mouse cursor from a higher-left to a lower-right position. This shows the frame for the label.

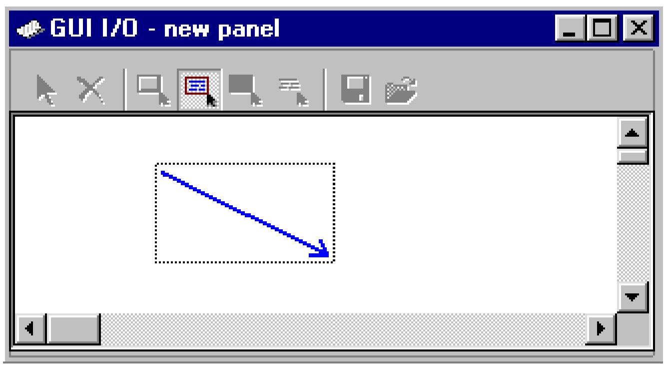


Figure 3.62 GUI I/O Window (Create Label)

Press the button on the toolbar or choose [Select Item] from the pop-up menu and double-click on the created label to open the [Set Label] dialog box. Specify the responses to events. The label name must not include white space.

- Response to writing of either value to a selected bit

The settings shown below set up display of the character string “Printing in progress” or “Printer ready” when the value of bit 3 at address 0x3E0 is 0 or 1, respectively.

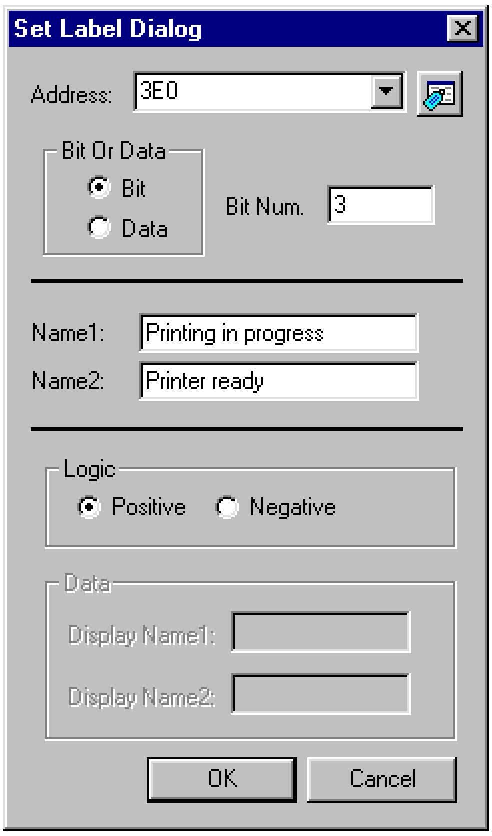


Figure 3.63 Set Label Dialog Box (Bit Selection)

- Response to writing of specified values to a selected address

The settings shown below set up display of the character string “Printing in progress” or “Printer ready” when the value 0x10 or 0x20, respectively, is written to address 0x3E0.

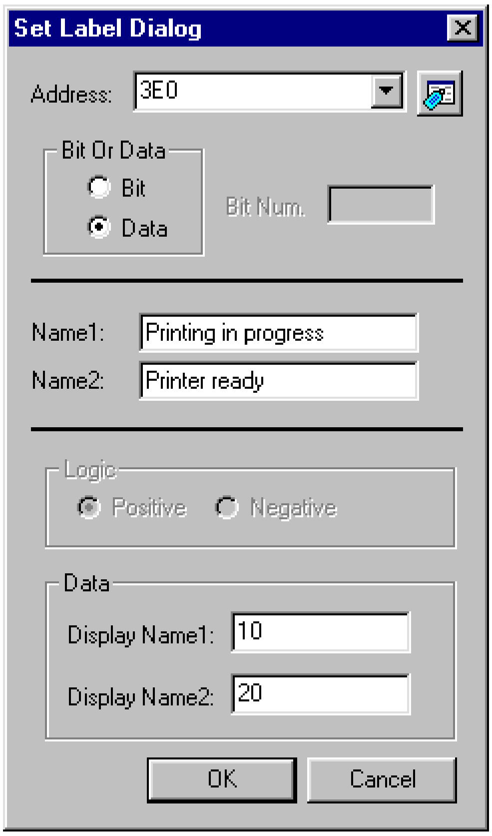


Figure 3.64 Set Label Dialog Box (Data Selection)

3.13.4 Creating an LED

Click on the  button on the toolbar or choose [Create LED] from the pop-up menu. The mouse cursor turns into a “+” symbol. Drag the mouse cursor from an upper left to a lower right position. This shows the frame for the LED output.

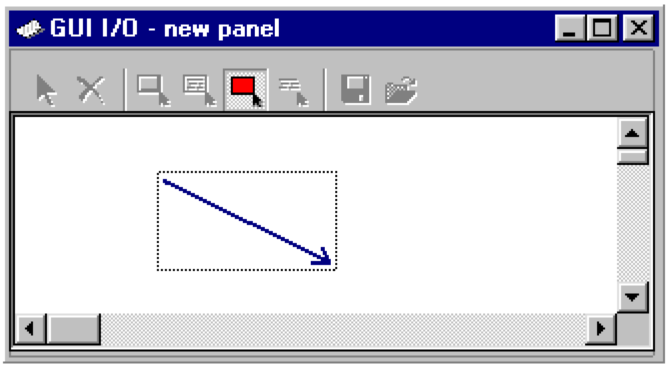


Figure 3.65 GUI I/O Window (Create LED)

Press the  button on the toolbar or choose [Select Item] from the pop-up menu and double-click on the created LED to open the [Set LED] dialog box. Specify the events and responses.

- Response to writing of either value to a selected bit

The settings shown below set up the display of green or red, respectively, in the LED area when the value of bit 2 at address 0x3E0 is 0 or 1.

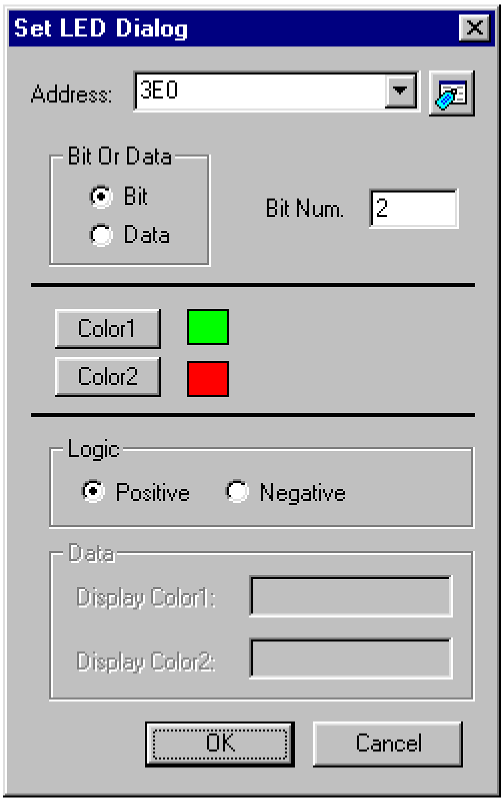


Figure 3.66   Set LED Dialog Box (Bit Selection)

- Response to writing of specified values to a selected address

The settings shown below set up the display of green or red, respectively, in the LED area when the value 0x10 or 0x20 is written to address 0x3E0.

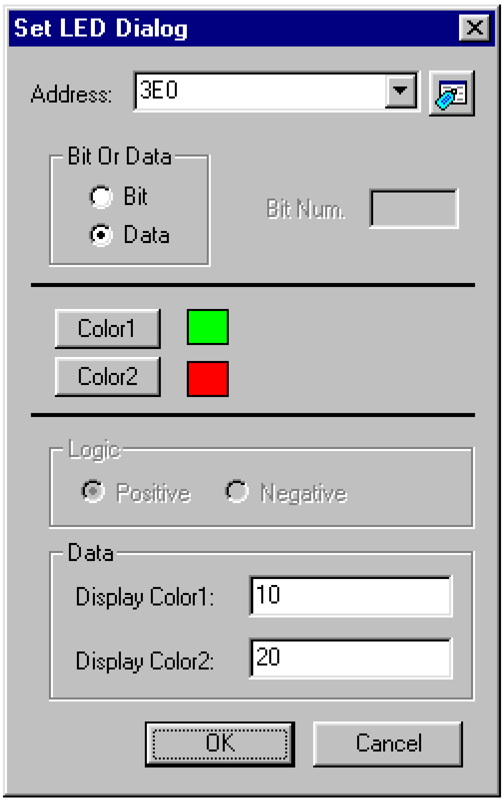


Figure 3.67 Set LED Dialog Box (Data Selection)

Clicking the [Color 1] or [Color 2] button opens the [Color] dialog box, which allows you to select the color.

3.13.5 Creating Fixed Text

Click the button on the toolbar or choose [Create Text] from the pop-up menu. The mouse cursor turns into a "+" symbol. Create the text box by dragging the mouse cursor from a higher-left to a lower-right position.

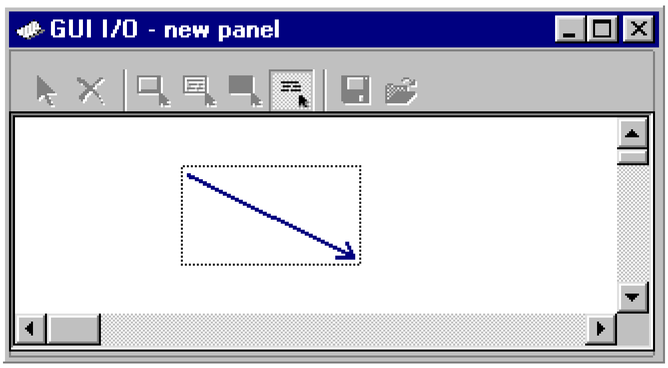


Figure 3.68 GUI I/O Window (Create Fixed Text)

- Setting the format for the text

Press the button on the toolbar and double-click on the created text to open the [Set Text] dialog box.

Figure 3.69 Set Text Dialog Box

Click the [Font…] button to select the font and size for the text. Then click the [Text] and [Back] buttons to specify the colors of the text and its background.

3.13.6 Changing the Size and Position of an Item

Press the button on the toolbar and click on the item. The item is selected as shown in the figure below.

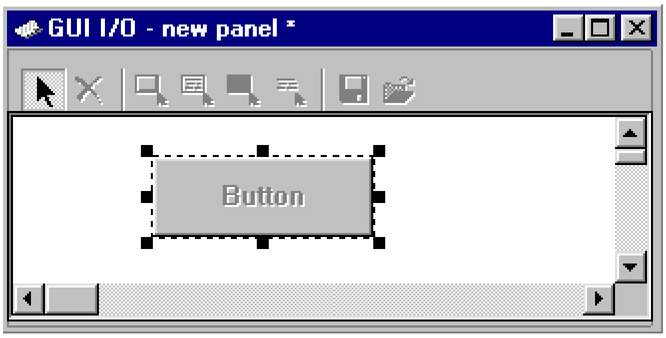


Figure 3.70 GUI I/O Window (Item Selected)

Drag the item to change its position or the control points to change its size.

3.13.7 Copying an Item

Press the button on the toolbar or choose [Copy] from the pop-up menu. The mouse cursor turns into a "+" symbol. In this state, click on the item you wish to copy. Press the button on the toolbar or choose [Paste] from the pop-up menu to create a new item with the same size and attributes.

3.13.8 Deleting an Item

Press the button on the toolbar or choose [Delete] from the pop-up menu. The mouse cursor turns into a “+” symbol. In this state, click on the item you wish to delete.

3.13.9 Showing the Grid

Press the button on the toolbar or choose [Display Grid] from the pop-up menu. This displays the grid on the background.

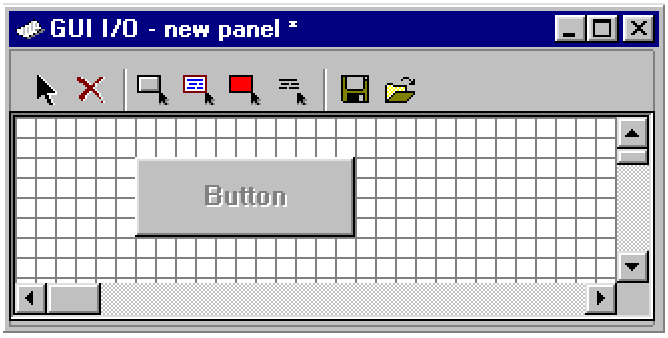


Figure 3.71 GUI I/O Window (Show Grid)

Clicking the button again hides the grid.

3.13.10 Saving I/O Panel Information

It is possible to reuse created I/O panels by saving the information in files. Press the button on the toolbar or choose [Save] from the pop-up menu to open the [Save GUI I/O Panel File] dialog box. Specify the directory where the file is to be stored and enter the file name.

3.13.11 Loading I/O Panel Information

Press the button on the toolbar or choose [Load] from the pop-up menu to open the [Load GUI I/O Panel File] dialog box. Specify the file you wish to load. Panel information prior to the load will be deleted.

Section 4 Windows

Table 4.1 lists the windows.

Refer to the simulator/debugger help about the toolbar buttons.

Table 4.1 Simulator/Debugger Windows

Window Name	Function
IO	Viewing the I/O Memory
Simulated I/O	Standard I/O and File I/O Processing
Event	Using the Simulator/Debugger Breakpoints
Watch	Looking at Variables (any variables)
Editor	Displaying the source code
Image	Displaying Memory Contents as an Image
Coverage	Measuring Code Coverage
Disassembly	Viewing the Assembly-Language Code
Command Line	Debugging with the Command Line Interface
Stack Trace	Viewing the Function Call History
Status	Viewing the Current Status
Trigger	Generating a Pseudo-Interrupt Manually
Trace	Viewing the Trace Information
Wave	Displaying Memory Contents as Waveforms
Analyzing Performance Register	Analyzing Performance
Profile/Profile-Chart	Viewing the Profile Information
Memory	Viewing a Memory Area
Label	Looking at Labels
Register	Looking at Registers
Local	Looking at Variables (local variables)
GUI I/O	Creating a Virtual I/O Panel
OS Object	Displaying the status of OS objects such as tasks and semaphores
Task Trace	Measuring the execution history of the program by using the realtime OS.
Task Analyze	Displaying the state of CPU occupancy.

Section 5 Command Lines

5.1 Commands (Functional Order)

The following tables show the commands in functional order.

Refer to the simulator/debugger help about each command.

5.1.1 Execution

Command Name	Abbr.	Function
GO	GO	Executes user program
GO_RESET	GR	Executes user program from reset vector
GO_TILL	GT	Executes user program until temporary breakpoint
HALT	HA	Halts the user program
RESET	RE	Resets CPU
STEP	ST	Steps program (by instructions or source lines)
STEP_MODE	SM	Selects the step mode
STEP_OUT	SP	Steps out of the current function
STEP_OVER	SO	Steps program, not stepping into functions
STEP_RATE	SR	Sets or displays rate of stepping

5.1.2 Download

Command Name	Abbr.	Function
BUILD	BU	Performs a build on the current project
BUILD_ALL	BL	Performs a build all on the current project
BUILD_FILE	BF	Compiles files
BUILD_MULTIPLE	BM	Builds multiple projects
CLEAN	CL	Deletes intermediate and output files produced in building
DEFAULT_OBJECT_FORMAT	DO	Sets the default object (program) format
FILE_LOAD	FL	Loads an object (program) file
FILE_LOAD_ALL	LA	Loads all object (program) files
FILE_SAVE	FS	Saves memory to a file
FILE_UNLOAD	FU	Unloads an object (program) file from memory
FILE_UNLOAD_ALL	UA	Unloads all object (program) files from memory
FILE_VERIFY	FV	Verifies file contents against memory
GENERATE_MAKE_FILE	GM	Generates a build makefile for the current workspace or deletion

5.1.3 Register

Command Name	Abbreviation	Function
REGISTER_DISPLAY	RD	Displays CPU register values
REGISTER_SET	RS	Changes CPU register contents

5.1.4 Memory

Command Name	Abbreviation	Function
CACHE	-	Sets caching on or off
MEMORY_COMPARE	MC	Compares memory contents
MEMORY_DISPLAY	MD	Displays memory contents
MEMORY_EDIT	ME	Modifies memory contents
MEMORY_FILL	MF	Modifies the content of a memory area by specifying data
MEMORY_FIND	MI	Finds a string in an area of memory
MEMORY_MOVE	MV	Moves a block of memory
MEMORY_TEST	MT	Tests a block of memory

5.1.5 Assemble/Disassemble

Command Name	Abbreviation	Function
ASSEMBLE	AS	Assembles instructions into memory
DISASSEMBLE	DA	Disassembles memory contents
SYMBOL_ADD	SA	Defines a symbol
SYMBOL_CLEAR	SC	Deletes a symbol
SYMBOL_LOAD	SL	Loads a symbol information file
SYMBOL_SAVE	SS	Saves a symbol information file
SYMBOL_VIEW	SV	Displays symbols

5.1.6 Break

Command Name	Abbreviation	Function
BREAKPOINT	BP	Sets a breakpoint at an instruction address
BREAK_ACCESS	BA	Specifies a memory range access as a break condition
BREAK_CLEAR	BC	Deletes breakpoints
BREAK_CYCLE	BCY	Specifies a cycle as a break condition
BREAK_DATA	BD	Specifies a memory data value as a break condition
BREAK_DATA_DIFFERENCE	BDD	Specifies a difference between two values of data in memory as a break condition
BREAK_DATA_INVERSE	BDI	Specifies inversion of the sign of a value of data in memory as a break condition
BREAK_DATA_RANGE	BDR	Specifies a range of values in memory as a break condition
BREAK_DISPLAY	BI	Displays a list of breakpoints
BREAK_ENABLE	BE	Enables or disables a breakpoint
BREAK_REGISTER	BR	Specifies a register data as a break condition
BREAK_SEQUENCE	BS	Sets sequential breakpoints
SET_DISASSEMBLY_SOFT_BREAK	SDB	Sets or deletes a software breakpoint at the disassembly level
SET_SOURCE_SOFT_BREAK	SSB	Sets or cancels a software breakpoint at source level
STATE_DISASSEMBLY_SOFT_BREAK	TDB	Enables or disables a software breakpoint at disassembly level
STATE_SOURCE_SOFT_BREAK	TSB	Enables or disables a software breakpoint at source level

5.1.7 Trace

Command Name	Abbr.	Function
TRACE	TR	Displays trace information
TRACE_CONDITION_SET	TCS	Sets trace information acquisition
TRACE_SAVE	TV	Outputs trace information into a file

5.1.8 Coverage

Command Name	Abbr.	Function
COVERAGE	CV	Enables or disables coverage measurement
COVERAGE_DISPLAY	CVD	Displays coverage information
COVERAGE_LOAD	CVL	Loads coverage information
COVERAGE_RANGE	CVR	Sets a coverage range
COVERAGE_SAVE	CVS	Saves coverage information

5.1.9 Performance

Command Name	Abbr.	Function
ANALYSIS	AN	Enables or disables performance analysis
ANALYSIS_RANGE	AR	Sets or displays performance analysis functions
ANALYSIS_RANGE_DELETE	AD	Deletes a performance analysis range
PROFILE	PR	Enables or disables profile
PROFILE_DISPLAY	PD	Displays profile information
PROFILE_SAVE	PS	Saves the profile information to file

5.1.10 Watch

Command Name	Abbr.	Function
WATCH_ADD	WA	Adds an item for watching
WATCH_AUTO_UPDATE	WU	Selects or cancels automatic updating of watched items
WATCH_DELETE	WD	Deletes a watched item
WATCH_DISPLAY	WI	Displays the contents of the Watch window
WATCH_EDIT	WE	Edits the value of a watched item
WATCH_EXPAND	WX	Expands or collapses a watched item
WATCH_RADIX	WR	Changes the radix for display of watched items
WATCH_RECORD	WO	Outputs the history of updating of the values of a watched item to a file
WATCH_SAVE	WS	Saves the contents of the Watch window to a file

5.1.11 Script/Logging

Command Name	Abbr.	Function
!	-	Comment
ASSERT	-	Checks if an expression is true or false
AUTO_COMPLETE	AC	Enables or disables the auto-complete function
ERASE	ER	Clears the [Command Line] window
EVALUATE	EV	Evaluates an expression
LOG	LO	Controls command output logging
SLEEP	-	Delays command execution
SUBMIT	SU	Executes a command file
TCL	-	Displays TCL information

5.1.12 Memory Resource

Command Name	Abbr.	Function
MAP_DISPLAY	MA	Displays memory resource settings
MAP_SET	MS	Allocates a memory area

5.1.13 Simulator/Debugger Settings

Command Name	Abbr.	Function
EXEC_MODE	EM	Sets and displays execution mode
EXEC_STOP_SET	ESS	Sets or displays the execution mode at the occurrence of an interrupt

5.1.14 Standard I/O and File I/O

Command Name	Abbr.	Function
SIMULATEDIO_CLEAR	SIOC	Clears the contents of the [Simulated I/O] window
TRAP_ADDRESS	TP	Sets a simulated I/O address
TRAP_ADDRESS_DISPLAY	TD	Displays simulated I/O address settings
TRAP_ADDRESS_ENABLE	TE	Enables or disables the simulated I/O

5.1.15 Utility

Command Name	Abbr.	Function
HELP	HE	Displays the command line help
INITIALIZE	IN	Initializes the debugging platform
QUIT	QU	Exits HEW
RADIX	RA	Sets default input radix
RESPONSE	RP	Sets an interval to refresh the window
STATUS	STA	Displays the debugging platform status
TOOL_INFORMATION	TO	Outputs information on the currently registered tool to a file

5.1.16 Project/Workspace

Command Name	Abbr.	Function
ADD_FILE	AF	Adds a file to the current project
CHANGE_CONFIGURATION	CC	Sets the current configuration
CHANGE_PROJECT	CP	Sets the current project
CHANGE_SESSION	CS	Changes the current session
CHANGE_SUB_SESSION	CB	Changes the currently active session when simultaneous debugging is enabled
CLEAR_OUTPUT_WINDOW	COW	Clears the contents of the specified tab in the [Output] window
CLOSE_WORKSPACE	CW	Close the current workspace
OPEN_WORKSPACE	OW	Opens a workspace
REFRESH_SESSION	RSE	Updates information on the session
REMOVE_FILE	REM	Removes a file from the current project
SAVE_SESSION	SE	Saves the current session
SAVE_WORKSPACE	SW	Saves the current workspace
UPDATE_ALL_DEPENDENCIES	UD	Updates all build dependencies of the current project

5.1.17 Test Tool Facility

Command Name	Abbr.	Function
CLOSE_TEST_SUITE	CTS	Closes a test suite
COMPARE_TEST_DATA	CTD	Compares test data
OPEN_TEST_SUITE	OTS	Opens a test suite
RUN_TEST	RT	Executes a test

5.1.18 Debugging Functions for the Realtime OS

Command Name	Abbr.	Function
OSOBJECT_ALL_ADD	OAA	Adds OS objects (of a specific object type)
OSOBJECT_ALL_DELETE	OAD	Deletes OS objects (in a specific sheet)
OSOBJECT_AUTO_UPDATE	OAU	Changes the automatic-update setting to “Auto” and “Break”.
OSOBJECT_DATA_LOWLINE	ODL	Moves an OS object to the next line.
OSOBJECT_DATA_SAVE	ODS	Saves the information on an OS object to a file.
OSOBJECT_DATA_UPLINE	ODU	Moves an OS object to the previous line.
OSOBJECT_DISPLAY	OD	Shows the information on an OS object.
OSOBJECT_NO_UPDATE	ONU	Changes the automatic-update setting to “Lock”.
OSOBJECT_ONE_ADD	OOA	Adds an OS object.
OSOBJECT_ONE_DELETE	OOD	Deletes an OS object.
OSOBJECT_ONE_EDIT	OOE	Edits an OS object.
OSOBJECT_SETTING_LOAD	OSL	Loads OS-object settings from a file.
OSOBJECT_SETTING_SAVE	OSS	Saves OS-object settings in a file.
OSOBJECT_STOP_UPDATE	OSU	Changes the automatic-update setting to “Break”.

5.1.19 File Input and Output through Virtual Ports

Command Name	Abbr.	Function
PORT_FILE_ADD	PFA	Adds a file for input or output through a virtual port.
PORT_FILE_CLOSE	PFC	Closes a file for input or output through a virtual port.
PORT_FILE_DELETE	PFD	Deletes the setting of a file for input or output through a virtual port.
PORT_FILE_OPEN	PFO	Opens a file for input or output through a virtual port.
PORT_FILE_STATUS	PFS	Shows the current state of a file for input or output through a virtual port.

5.2 Commands (Alphabetical Order)

Table 5.1 lists the commands in alphabetical order.

Refer to the simulator/debugger help about each command.

Table 5.1 Simulator/Debugger Commands

No.	Command Name	Abbr.	Function
1	!	-	Comment
2	ADD_FILE	AF	Adds a file to the current project
3	ANALYSIS	AN	Enables or disables performance analysis
4	ANALYSIS_RANGE	AR	Sets or displays performance analysis functions
5	ANALYSIS_RANGE_ DELETE	AD	Deletes a performance analysis range
6	ASSEMBLE	AS	Assembles instructions into memory
7	ASSERT	-	Checks if an expression is true or false
8	AUTO_COMPLETE	AC	Enables or disables the auto-complete function
9	BREAKPOINT	BP	Sets a breakpoint at an instruction address
10	BREAK_ACCESS	BA	Specifies a memory range access as a break condition
11	BREAK_CLEAR	BC	Deletes breakpoints
12	BREAK_CYCLE	BCY	Specifies a cycle as a break condition
13	BREAK_DATA	BD	Specifies a memory data value as a break condition
14	BREAK_DATA_ DIFFERENCE	BDD	Specifies a difference between two values of data in memory as a break condition
15	BREAK_DATA_ INVERSE	BDI	Specifies inversion of the sign of a value of data in memory as a break condition
16	BREAK_DATA_RANGE	BDR	Specifies a range of values in memory as a break condition
17	BREAK_DISPLAY	BI	Displays a list of breakpoints
18	BREAK_ENABLE	BE	Enables or disables a breakpoint
19	BREAK_REGISTER	BR	Specifies a register data as a break condition
20	BREAK_SEQUENCE	BS	Sets sequential breakpoints
21	BUILD	BU	Performs a build on the current project
22	BUILD_ALL	BL	Performs a build all on the current project
23	BUILD_FILE	BF	Compiles files
24	BUILD_MULTIPLE	BM	Builds multiple projects
25	CACHE	-	Sets caching on or off
26	CHANGE_CONFIGURATION	CC	Sets the current configuration
27	CHANGE_PROJECT	CP	Sets the current project
28	CHANGE_SESSION	CS	Changes the current session
29	CHANGE_SUB_SESSION	CB	Changes the currently active session when simultaneous debugging is enabled
30	CLEAN	CL	Deletes intermediate and output files produced in building

Table 5.1 Simulator/Debugger Commands (cont)

No.	Command Name	Abbr.	Function
31	CLEAR_OUTPUT_WINDOW	COW	Clears the contents of the specified tab in the [Output] window
32	CLOSE_TEST	CT	Closes a test suite
33	CLOSE_WORKSPACE	CW	Close the current workspace
34	COMPARE_TEST_DATA	CTD	Compares test data
35	COVERAGE	CV	Enables or disables coverage measurement
36	COVERAGE_DISPLAY	CVD	Displays coverage information
37	COVERAGE_LOAD	CVL	Loads coverage information
38	COVERAGE_RANGE	CVR	Sets a coverage range
39	COVERAGE_SAVE	CVS	Saves coverage information
40	DEFAULT_OBJECT_FORMAT	DO	Sets the default object (program) format
41	DISASSEMBLE	DA	Disassembles memory contents
42	ERASE	ER	Clears the [Command Line] window
43	EVALUATE	EV	Evaluates an expression
44	EXEC_MODE	EM	Sets and displays execution mode
45	EXEC_STOP_SET	ESS	Sets or displays the execution mode at the occurrence of an interrupt
46	FILE_LOAD	FL	Loads an object (program) file
47	FILE_LOAD_ALL	LA	Loads all object (program) files
48	FILE_SAVE	FS	Saves memory to a file
49	FILE_UNLOAD	FU	Unloads an object (program) file from memory
50	FILE_UNLOAD_ALL	UA	Unloads all object (program) files from memory
51	FILE_VERIFY	FV	Verifies file contents against memory
52	GENERATE_MAKE_FILE	GM	Generates a build makefile for the current workspace
53	GO	GO	Executes user program
54	GO_RESET	GR	Executes user program from reset vector
55	GO_TILL	GT	Executes user program until temporary breakpoint
56	HALT	HA	Halts the user program
57	HELP	HE	Displays the command line help
58	INITIALIZE	IN	Initializes the debugging platform
59	LOG	LO	Controls command output logging
60	MAP_DISPLAY	MA	Displays memory resource settings
61	MAP_SET	MS	Allocates a memory area
62	MEMORY_COMPARE	MC	Compares memory contents
63	MEMORY_DISPLAY	MD	Displays memory contents
64	MEMORY_EDIT	ME	Modifies memory contents
65	MEMORY_FILL	MF	Modifies the content of a memory area by specifying data
66	MEMORY_FIND	MI	Finds a string in an area of memory
67	MEMORY_MOVE	MV	Moves a block of memory
68	MEMORY_TEST	MT	Tests a block of memory

Table 5.1 Simulator/Debugger Commands (cont)

No.	Command Name	Abbr.	Function
69	OPEN_TEST_SUITE	OTS	Opens a test suite
70	OPEN_WORKSPACE	OW	Opens a workspace
71	OSOBJECT_ALL_ADD	OAA	Adds OS objects (of a specific object type)
72	OSOBJECT_ALL_DELETE	OAD	Deletes OS objects (in a specific sheet)
73	OSOBJECT_AUTO_UPDATE	OAU	Changes the automatic-update setting to “Auto” and “Break”.
74	OSOBJECT_DATA_LOWLINE	ODL	Moves an OS object to the next line.
75	OSOBJECT_DATA_SAVE	ODS	Saves the information on an OS object to a file.
76	OSOBJECT_DATA_UPLINE	ODU	Moves an OS object to the previous line.
77	OSOBJECT_DISPLAY	OD	Shows the information on an OS object.
78	OSOBJECT_NO_UPDATE	ONU	Changes the automatic-update setting to “Lock”.
79	OSOBJECT_ONE_ADD	OOA	Adds an OS object.
80	OSOBJECT_ONE_DELETE	OOD	Deletes an OS object.
81	OSOBJECT_ONE_EDIT	OOE	Edits an OS object.
82	OSOBJECT_SETTING_LOAD	OSL	Loads OS-object settings from a file.
83	OSOBJECT_SETTING_SAVE	OSS	Saves OS-object settings in a file.
84	OSOBJECT_STOP_UPDATE	OSU	Changes the automatic-update setting to “Break”.
85	PORT_FILE_ADD	PFA	Adds a file for input or output through a virtual port.
86	PORT_FILE_CLOSE	PFC	Closes a file for input or output through a virtual port.
87	PORT_FILE_DELETE	PFD	Deletes the setting of a file for input or output through a virtual port.
88	PORT_FILE_OPEN	PFO	Opens a file for input or output through a virtual port.
89	PORT_FILE_STATUS	PFS	Shows the current state of a file for input or output through a virtual port.
90	PROFILE	PR	Enables or disables the profile
91	PROFILE_DISPLAY	PD	Displays profile information
92	PROFILE_SAVE	PS	Saves the profile information to file
93	QUIT	QU	Exits HEW
94	RADIX	RA	Sets default input radix
95	REFRESH_SESSION	RSE	Updates information on the session
96	REGISTER_DISPLAY	RD	Displays CPU register values
97	REGISTER_SET	RS	Changes CPU register contents
98	REMOVE_FILE	REM	Removes a file from the current project
99	RESET	RE	Resets CPU
100	RESPONSE	RP	Sets an interval to refresh the window
101	RUN_TEST	RT	Executes a test
102	SLEEP	-	Delays command execution
103	SAVE_SESSION	SE	Saves the current session
104	SAVE_WORKSPACE	SW	Saves the current workspace
105	SET_DISASSEMBLY_SOFT_BREAK	SDB	Sets or deletes a software breakpoint at the disassembly level

Table 5.1 Simulator/Debugger Commands (cont)

No.	Command Name	Abbr.	Function
106	SET_SOURCE_SOFT_BREAK	SSB	Sets or deletes a software breakpoint at the source level
107	SIMULATEDIO_CLEAR	SIOC	Clears the contents of the [Simulated I/O] window
108	STATE_DISASSEMBLY_SOFT_ BREAK	TDB	Enables or disables a software breakpoint at the disassembly level
109	STATE_SOURCE_SOFT_BREAK	TSB	Enables or disables a software breakpoint at the source level
110	STATUS	STA	Displays the debugging platform status
111	STEP	ST	Steps program (by instructions or source lines)
112	STEP_MODE	SM	Selects the step mode
113	STEP_OUT	SP	Steps out of the current function
114	STEP_OVER	SO	Steps program, not stepping into functions
115	STEP_RATE	SR	Sets or displays rate of stepping
116	SUBMIT	SU	Executes a command file
117	SYMBOL_ADD	SA	Defines a symbol
118	SYMBOL_CLEAR	SC	Deletes a symbol
119	SYMBOL_LOAD	SL	Loads a symbol information file
120	SYMBOL_SAVE	SS	Saves a symbol information file
121	SYMBOL_VIEW	SV	Displays symbols
122	TCL	-	Enables or disables the TCL
123	TOOL_INFORMATION	TO	Outputs information on the currently registered tool to a file
124	TRACE	TR	Displays trace information
125	TRACE_CONDITION_SET	TCS	Sets trace information acquisition
126	TRACE_SAVE	TV	Outputs trace information into a file
127	TRACE_STATISTIC	TST	Analyzes statistic information
128	TRAP_ADDRESS	TP	Sets a simulated I/O address
129	TRAP_ADDRESS_DISPLAY	TD	Displays simulated I/O address settings
130	TRAP_ADDRESS_ENABLE	TE	Enables or disables the simulated I/O
131	UPDATE_ALL_DEPENDENCIES	UD	Updates all build dependencies of the current project
132	WATCH_ADD	WA	Adds an item for watching
133	WATCH_AUTO_UPDATE	WU	Selects or cancels automatic updating of watched items
134	WATCH_DELETE	WD	Deletes a watched item
135	WATCH_DISPLAY	WI	Displays the contents of the Watch window
136	WATCH_EDIT	WE	Edits the value of a watched item
137	WATCH_EXPAND	WX	Expands or collapses a watched item
138	WATCH_RADIX	WR	Changes the radix for display of watched items
139	WATCH_RECORD	WO	Outputs the history of updating of the values of a watched item to a file
140	WATCH_SAVE	WS	Saves the contents of the Watch window to a file

Section 6 Messages

6.1 Information Messages

The simulator/debugger outputs information messages as listed in table 6.1 to notify users of execution status.

Table 6.1 Information Messages

Message	Contents
Break Access (Access Address: H'nnnnnnnn, Type: xxxx, Access Size: yyyy)	An access break condition was satisfied so execution has stopped. The information in parentheses shows the satisfied access break condition (accessed address, access type, and access unit).
Break Cycle (Cycle: H'nnnnnnnn)	A number-of-cycles condition was satisfied so execution has stopped. The information in parentheses shows the satisfied number-of-cycles condition (number of cycles).
Break Data (Access Address: H'nnnnnnnn, Data: H'mmmm)	A data break condition (other than [Inverse sign] or [Difference]) was satisfied so execution has stopped. The information in parentheses shows the satisfied data break condition (accessed address and value).
Break Data (Access Address: H'nnnnnnnn, Previous Data: H'mmmm, Current Data: H'mmmm)	A data break condition ([Inverse sign] or [Difference]) was satisfied so execution has stopped. The information in parentheses shows the satisfied data break condition (accessed address, and previous and current values).
Break Register (Register: XX, Value: H'mmmm)	A register break condition was satisfied so execution has stopped. The information in parentheses shows the satisfied register break condition (register name and value).
Break Sequence (PC: H'nnnnnnnn)	A sequential break condition was satisfied so execution has stopped. The information in parentheses shows the satisfied sequential break condition (address of the last instruction).
I/O DLL Stop	The peripheral function has stopped.
PC Breakpoint (PC: H'nnnnnnnn)	A PC breakpoint condition was satisfied so execution has stopped. The information in parentheses shows the satisfied PC-breakpoint condition (instruction address).
Step Normal End	The step execution succeeded.
Stop	Execution has been stopped by the [Stop] button.
Trace Buffer Full	Since the Break mode was selected by [Trace buffer full handling] in the [Trace Acquisition] dialog box and the trace buffer became full, execution was terminated.
WAIT Instruction	Instruction execution has been suspended by a WAIT instruction.

6.2 Error Messages

The simulator/debugger outputs error messages to notify users of the errors of user programs or operation. Table 6.2 lists the error messages.

Table 6.2 Error Messages

Message	Contents
Undefined Instruction Exception	An error has occurred due to undefined instruction exception processing.
Privilege Instruction Exception	An error has occurred due to privileged instruction exception processing.
Floating-point Exception	An error has occurred due to floating-point exception processing.
Reset Exception	An error has occurred due to reset exception processing.
Interrupt Exception	An error has occurred at the interrupt exception.
INT Instruction Exception	An error has occurred due to unconditional trap (INT instruction) exception processing.
BRK Instruction Exception	An error has occurred due to unconditional trap (BRK instruction) exception processing.
I/O area not exist	An attempt was made to delete the I/O area. Be sure to set the I/O area.
I/O DLL Illegal Interrupt Information (errNum=2xx)	Information on interrupts is incorrect. [errNum] shows the details on this error. Correct the information. [errNum] 200: The specified vector is outside the supported range. 201: The specified priority is outside the supported range.
I/O DLL Memory Access Error (errNum=0xx, Address=0xXXXXXXXX)	An error has occurred during a memory access to the peripheral function. [errNum] shows the details on this error and [Address] shows the address where this error occurred. Correct the user program according to the error information. [errNum] 001: The specified address is outside the supported range. 002: No memory exists in the specified area. 003: The required memory cannot be allocated. 004: The specified data size is outside the supported range. 005: The specified address cannot be accessed.
I/O DLL Register Access Error (errNum=1xx, RegisterName=xxxx)	An error has occurred during a register access to the peripheral function. [errNum] shows the details on this error and [RegisterName] shows the register where this error occurred. Correct the user program according to the error information. [errNum] 100: The register description is incorrect. 101: The specified data value is incorrect.

Table 6.2 Error Messages (cont)

Message	Contents
Memory Access Error (Address: H'nnnnnnnn)	One of the following events occurred (the information in parentheses shows the target address for the operation that generated the error): • A memory area that had not been allocated was accessed. • Data was written to a memory area having the write-protected attribute. • Data was read from a memory area having the read-disabled attribute. • A memory area in which memory does not exist was accessed. Allocate memory, change the memory attribute, or correct the user program to prevent the memory from being accessed.
System Call Error	Simulated I/O error occurred. Modify the incorrect contents of registers R1, R2, and parameter block.
The memory resource has not been set up	The memory resource was set outside the range of memory mapping. Modify the memory resource settings so that no error will occur.

Section 7 Tutorial

7.1 Preparation

The basic functions of the simulator/debugger will be described in this section using a sample program.

Note: The contents of usage examples (figures) in this section will differ depending on the compiler version.

7.1.1 Sample Program

The HEW demonstration program is used for the sample program and is written in C language. It first sorts ten random data in the ascending order, and then in the descending order. The sample program:

(1) Generates random data for sorting using the main function.

(2) Inputs the array which stores the random data that is generated by the main function, then sorts the data in the ascending order using the sort function.

(3) Inputs the array generated by the sort function, and sorts the data in the descending order using the change function.

(4) Displays the random data and the sorted data using the printf function.

The HEW demonstration program is used as the sample program.

7.1.2 Creating the Sample Program

Note the following when creating the HEW demonstration program:

- Specify [Demonstration] for [Project Type] in [Creating a New Workspace].
- Specify [RX600] for [CPU Series:].
- Specify [RX600 Simulator] for [Target:].
- Specify [SimDebug_RX600] for the configuration on the toolbar before building the project.
- Specify [SimSessionRX600] for the session on the toolbar.
- This demonstration program uses no peripheral function. In the [Set Peripheral Function Simulation] dialog box that opens when the session is changed, check [Don't show this dialog box] and then press the [OK] button.

Since this section explains the debugging function, [Demonstration] has not been optimized. Do not change this setting.

7.2 Settings for Debugging

7.2.1 Allocating the Memory Resource

The allocation of the memory resource is necessary to run the application being developed. When using the demonstration project, the memory resource is allocated automatically, so check the setting.

- Select [Simulator->Memory Resource...] from the [Setup] menu, and display the allocation of the current memory resource.

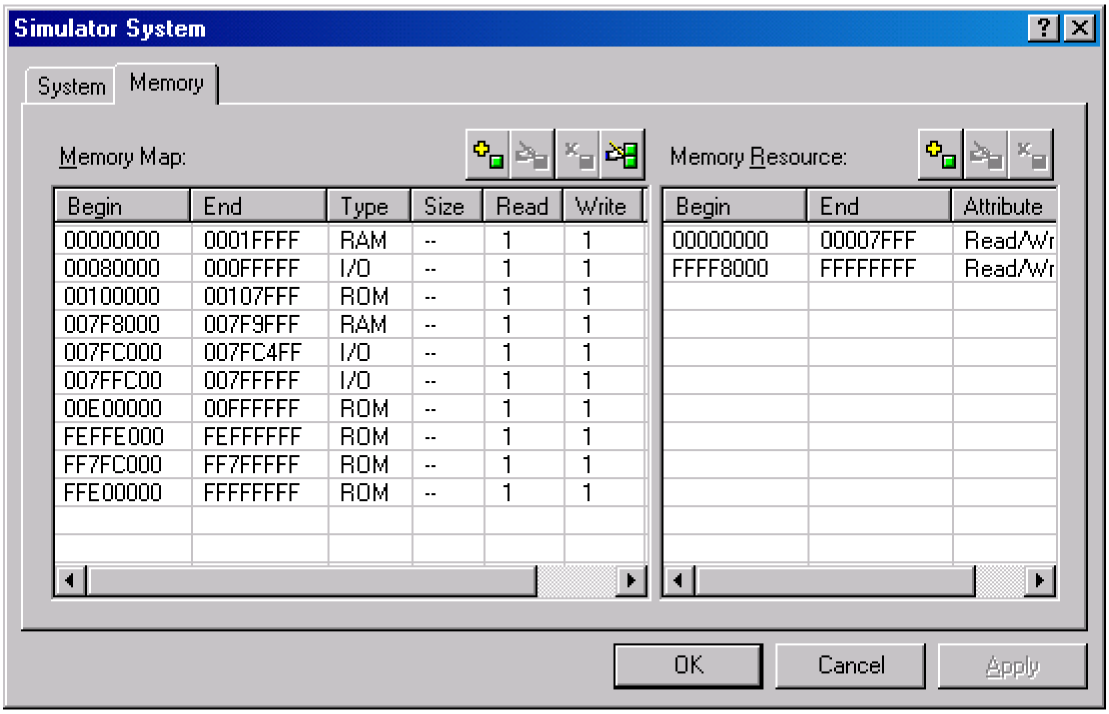


Figure 7.1 Simulator System Dialog Box (Memory Page)

The ranges of addresses from H'FFFF8000 to H'FFFFFFFF and H'00000000 to H'00007FFF are secured as readable and writable areas for storage of the program and data, respectively.

- Close the dialog box by clicking [OK].

The memory resource can also be referred to or modified by using the [Debugger] page on the [RX Standard Toolchain] dialog box. Changes made in either of the dialog boxes are reflected.

7.2.2 Downloading the Sample Program

When using the demonstration project, the sample program to be downloaded is automatically set, so check the settings.

- Open the [Debug Setting] dialog box by selecting [Debug Settings...] on the [Debug] menu.

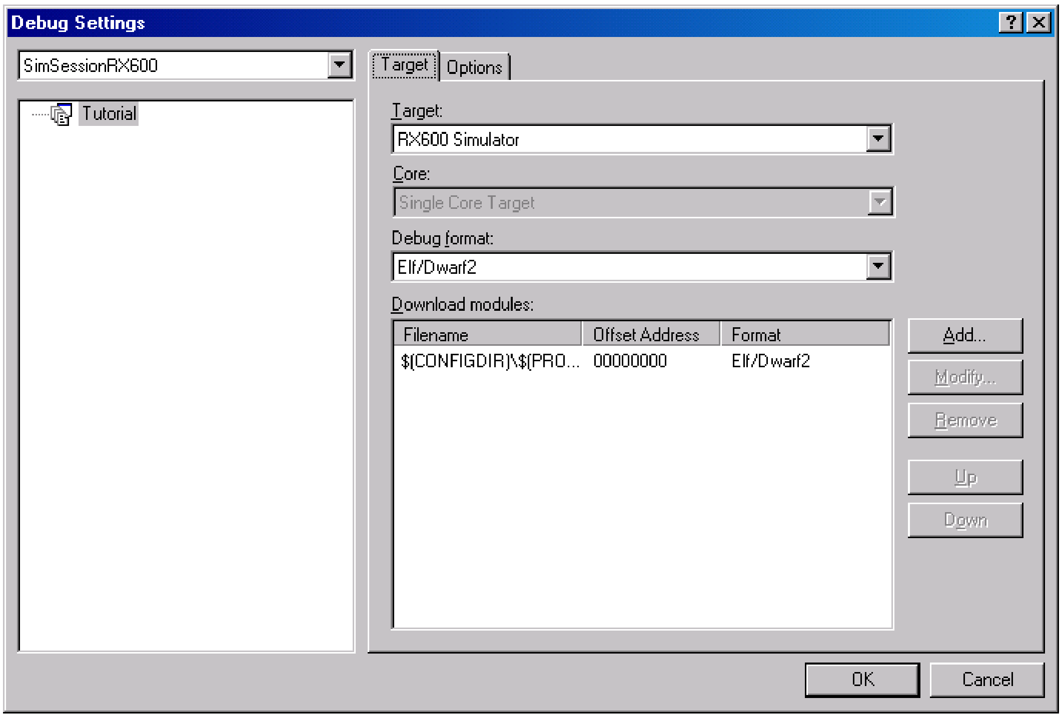


Figure 7.2 Debug Settings Dialog Box

- Files to be downloaded are listed in [Download Modules].
- Close the [Debug Settings] dialog box by clicking the [OK] button.
- Download the sample program by selecting [Download Modules->All Download Modules] from the [Debug] menu.

7.2.3 Displaying the Source Program

The HEW supports the source-level debugging. Display the source file ("Tutorial.c") in the [Source] window.

- Open the [Source] window by double-clicking Tutorial.c on the [Workspace] window.

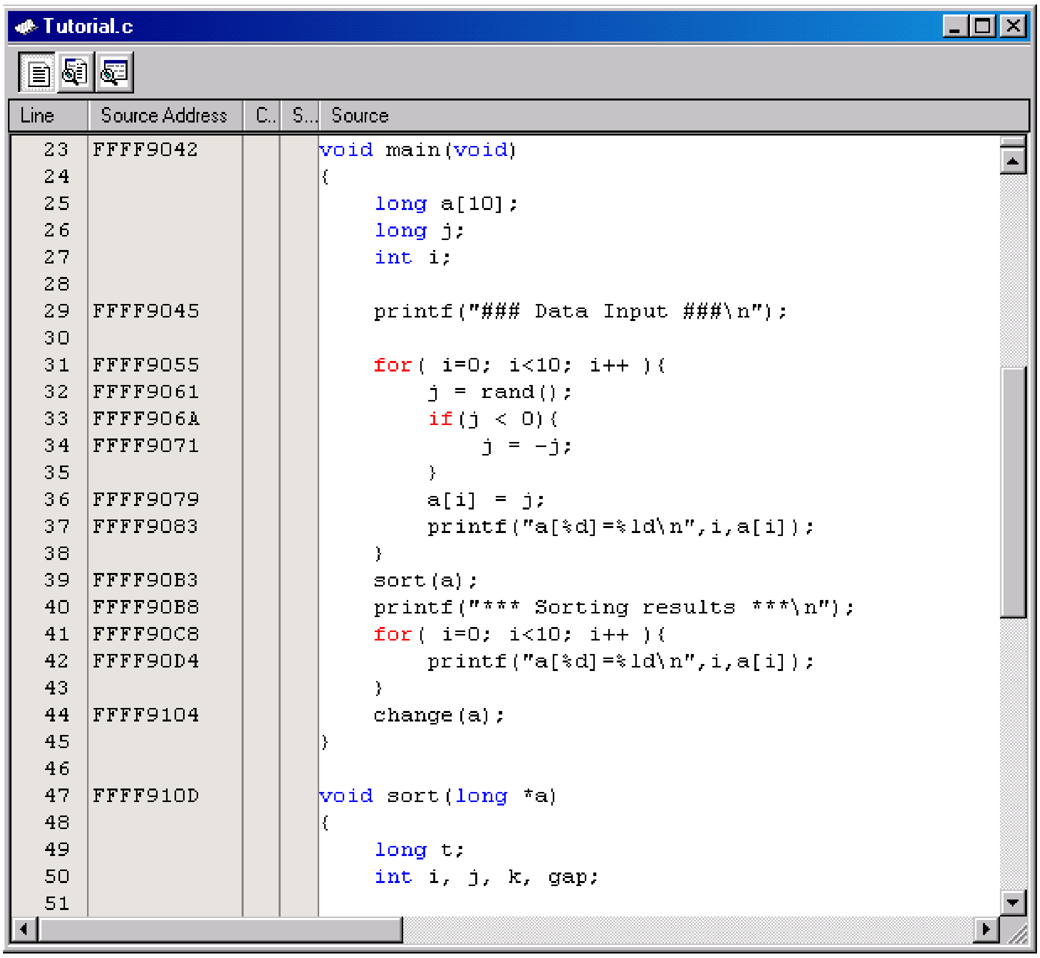


Figure 7.3 Source Window (Displaying the Source Program)

7.2.4 Setting a PC Breakpoint

Breakpoints can be set easily via the [Source] window. To set a breakpoint on a line that includes the sort function call:

- Place the cursor in the line that includes the sort function call and click the right mouse button to launch the pop-up menu, and select [Toggle Breakpoint] from the pop-up menu.

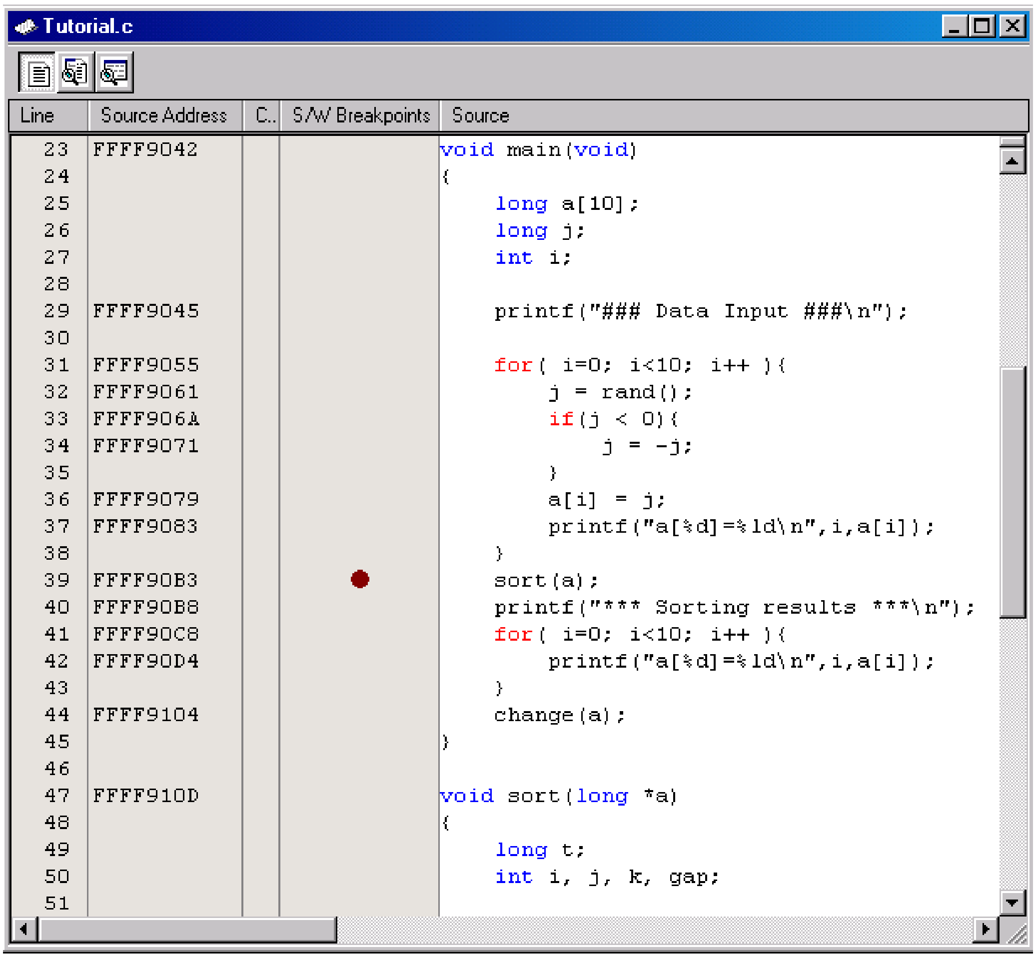


Figure 7.4 Source Window (Setting the Breakpoint)

A [•] is displayed at the line that includes the sort function call, indicating that the PC breakpoint is set at the address.

7.2.5 Setting the Profiler

- Open the [Profile] window by selecting [Profile] from the [View->Performance] menu.

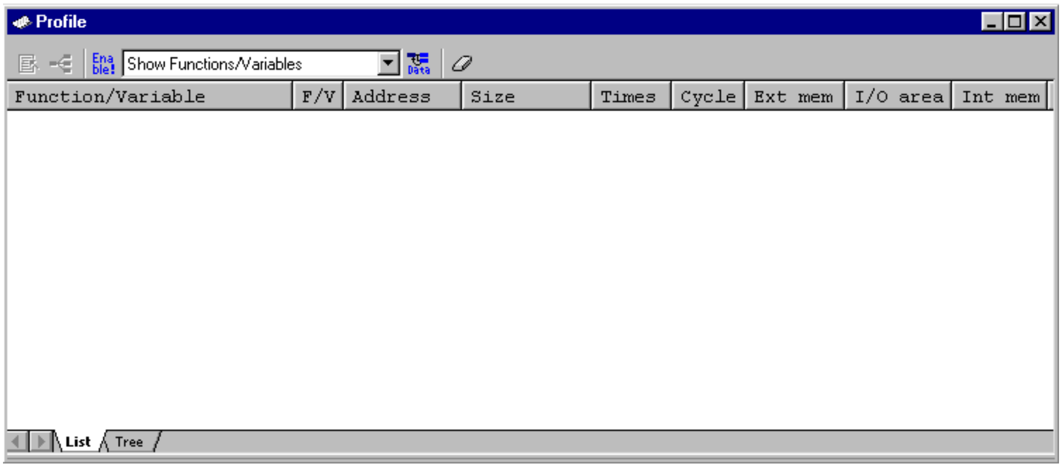


Figure 7.5 Profile Window

- Open the pop-up menu by right clicking the mouse on the [Profile] window, and select [Enable Profiler] to enable acquisition of the profile information.

7.2.6 Setting the Simulated I/O

When the demonstration project is used, the simulated I/O is automatically set, so check the setting.

- Open the [Simulator System] dialog box by selecting [Simulator->System] from the [Setup] menu.

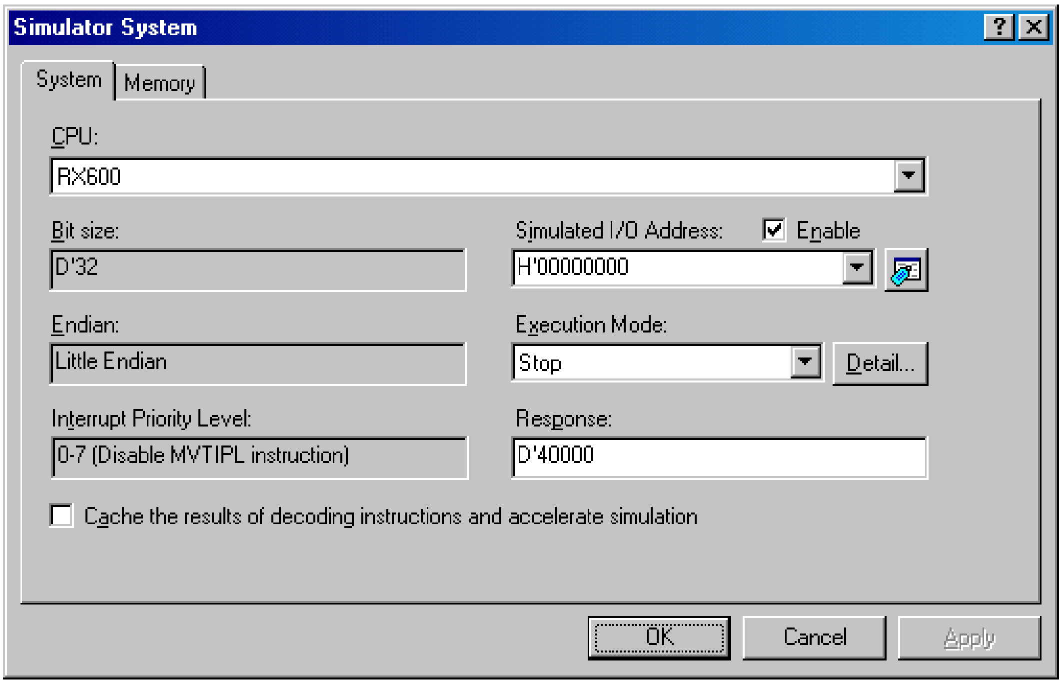


Figure 7.6 Simulator System Dialog Box (System Page)

- Confirm that [Enable] in [Simulated I/O Address] is checked.
- Click the [OK] button to enable the simulated I/O.
- Select [Simulated I/O] from the [View->CPU] menu and open the [Simulated I/O] window. The simulated I/O will not be enabled if the [Simulated I/O] window is not open.

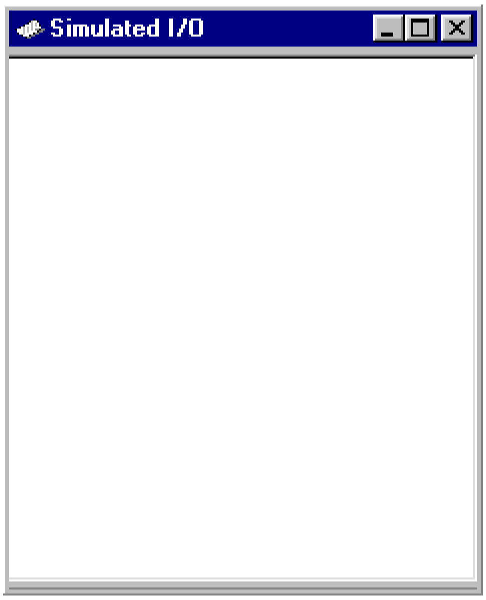


Figure 7.7 Simulated I/O Window

7.2.7 Setting the Trace Information Acquisition Conditions

- Select [Trace] from the [View->Code] menu to open the [Trace] window. Open the pop-up menu by right clicking the mouse on the [Trace] window, and select [Acquisition...] from the pop-up menu.

The [Trace Acquisition] dialog box below will be displayed.

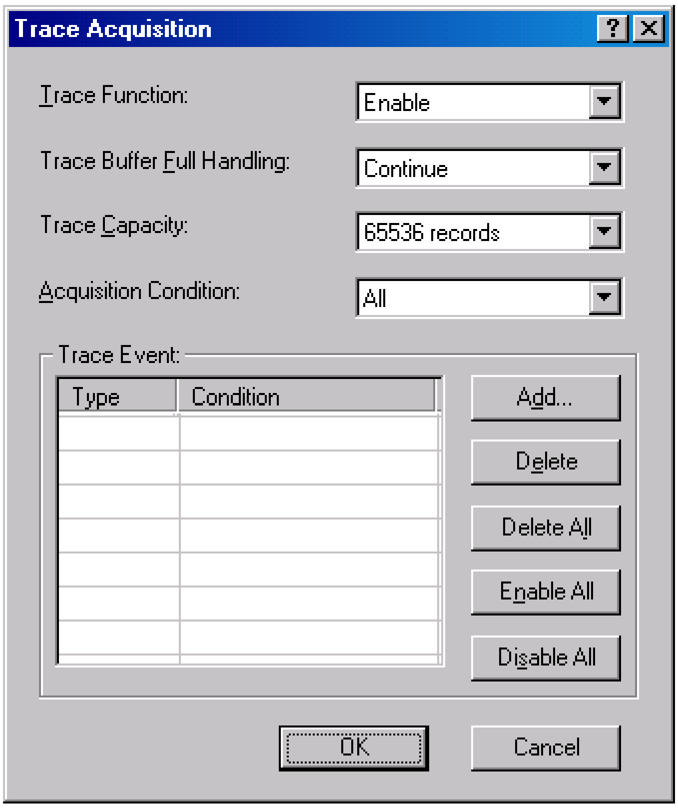


Figure 7.8 Trace Acquisition Dialog Box

- Set [Trace Function] to [Enable] in the [Trace Acquisition] dialog box, and click the [OK] button to enable the acquisition of the trace information.

7.2.8 Setting the Stack Pointer and Program Counter

To execute the program, the program counter must be set from the location of the reset vector. In the reset vector of the sample program, the PC value H'FFFF8000 is written.

- Select [Reset CPU] from the [Debug] menu, or click the [Reset CPU] button on the toolbar.

Set the program counter to H'FFFF8000 from the reset vector.

Figure 7.9 Reset CPU Button

7.3 Start Debugging

7.3.1 Executing a Program

- Select [Go] from the [Debug] menu, or click the [Go] button on the toolbar.

Figure 7.10 Go Button

The program halts where a breakpoint is set. An arrow is displayed in the [Source] window, indicating the location the execution has stopped. As the termination cause, [PC Breakpoint (PC: H'FFFF90E4)] is displayed in the [Output] window.

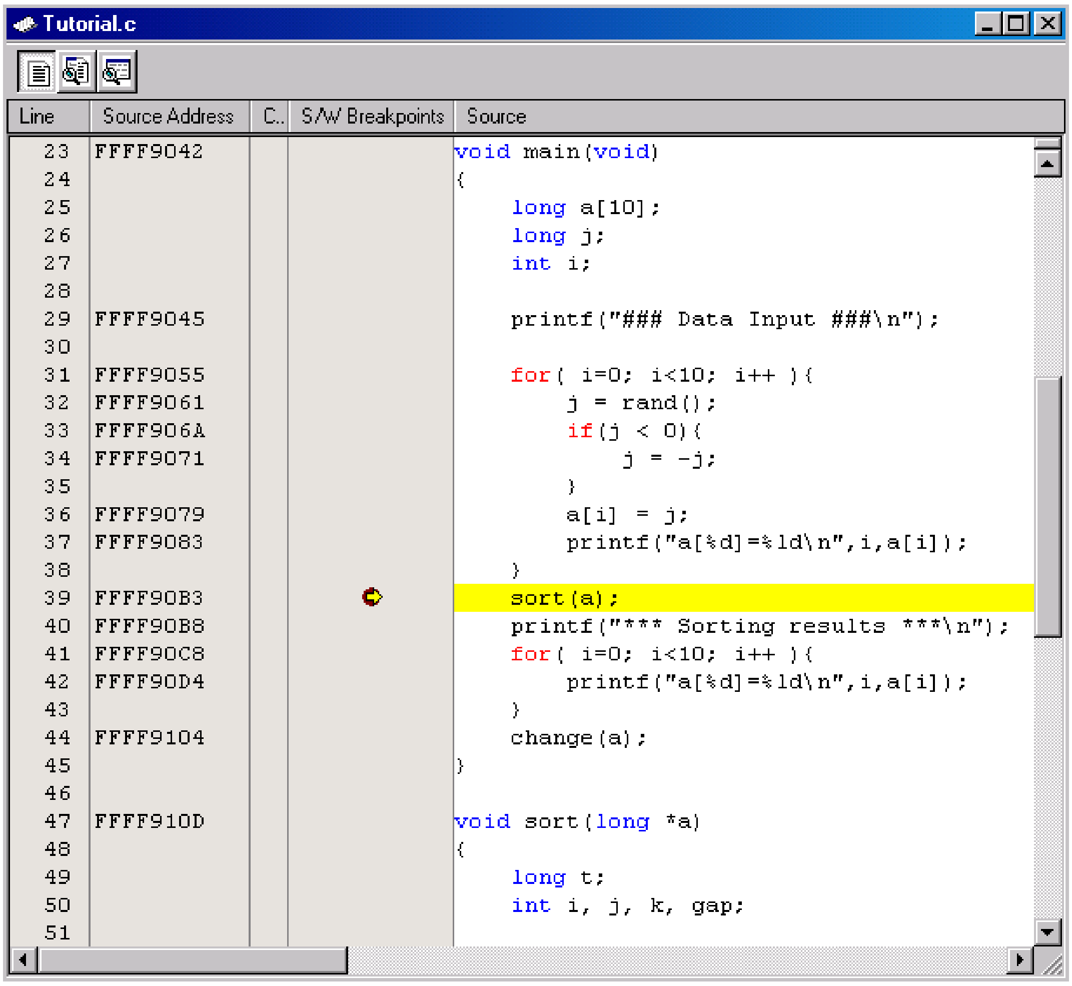


Figure 7.11 Source Window (Break Status)

The termination cause can be displayed in the [Status] window.

- Select [Status] from the [View->CPU] menu to open the [Status] window, and select the [Platform] sheet in the [Status] window.

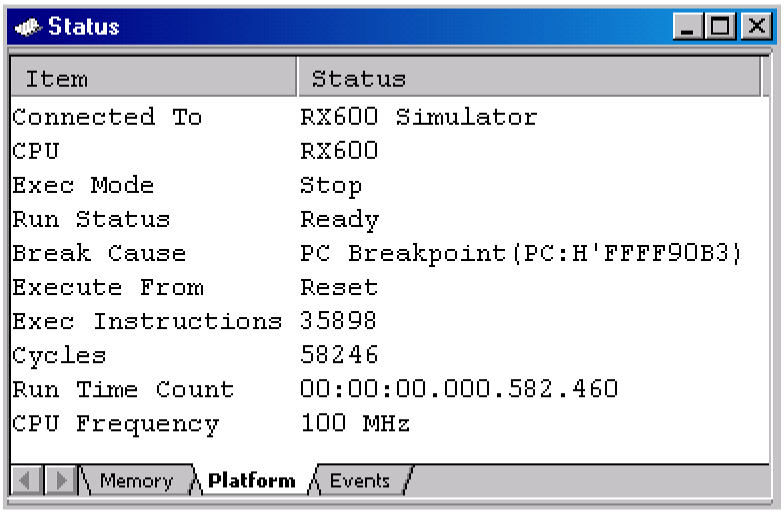


Figure 7.12 Status Window

The above status window indicates that:

(1) The cause of break is a PC breakpoint
(2) Execution is performed from the reset
(3) The number of instructions executed from a GO command following a reset is 35,898.
(4) The number of cycles of execution following a reset is 58,246.
(5) The execution time following a reset is 582.46 ms.
(6) The operating frequency of the CPU is 100 MHz.

Register values can be checked in the [Register] window.

- Select [Registers] from the [View->CPU] menu.

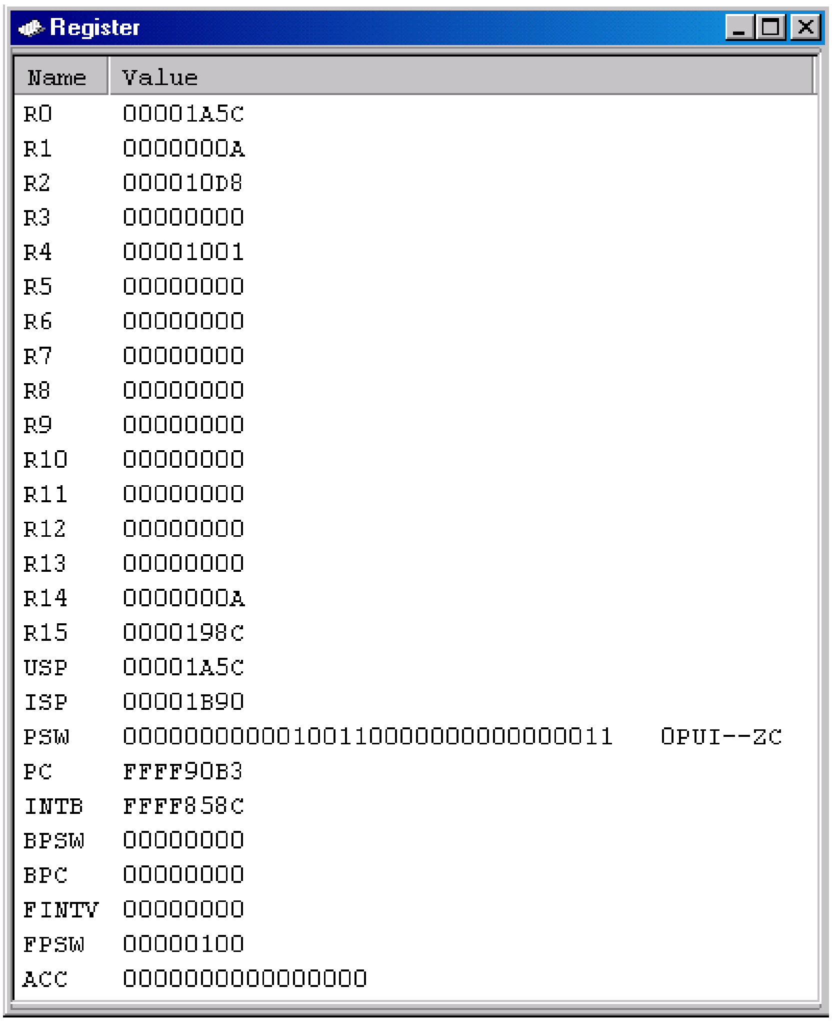


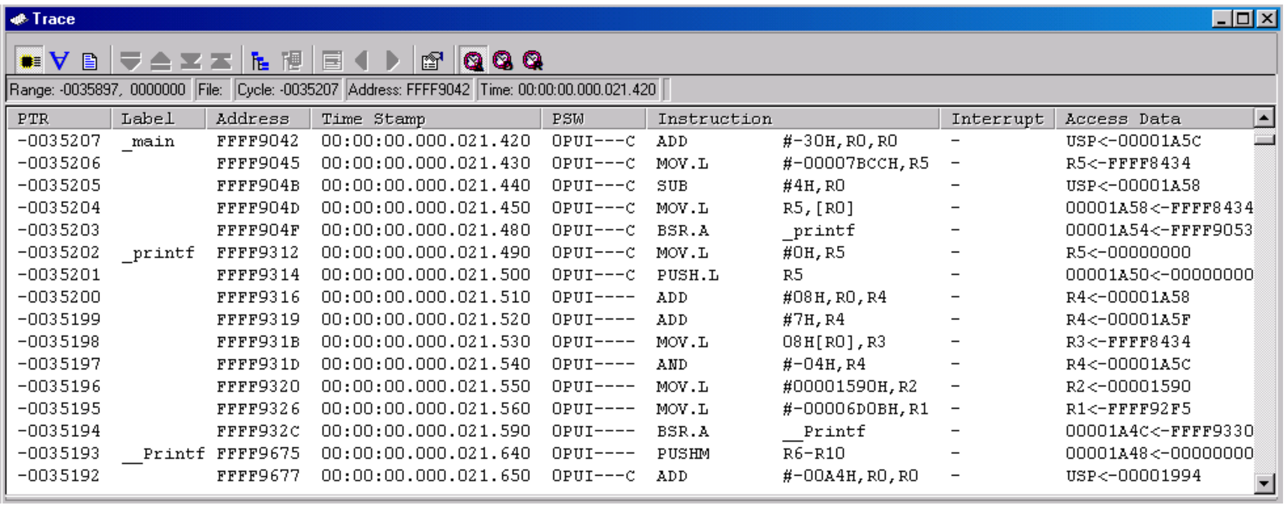
Figure 7.13 Register Window

Register values when the program is terminated can be checked.

7.3.2 Using the Trace Buffer

The trace buffer can be used to clarify the history of instruction execution.

- Select [Trace] from the [View->Code] menu and open the [Trace] window. Scroll up to the very top of the main() function.



Trace

Range: -0035897, 0000000 | File: | Cycle: -0035207 | Address: FFFF9042 | Time: 00:00:00.000.021.420

PTR	Label	Address	Time Stamp	PSW	Instruction		Interrupt	Access Data
-0035207	_main	FFFF9042	00:00:00.000.021.420	OPUI---C	ADD	#-30H,R0,R0	-	USP<-00001A5C
-0035206		FFFF9045	00:00:00.000.021.430	OPUI---C	MOV.L	#-00007BCCH,R5	-	R5<-FFFF8434
-0035205		FFFF904B	00:00:00.000.021.440	OPUI---C	SUB	#4H,R0	-	USP<-00001A58
-0035204		FFFF904D	00:00:00.000.021.450	OPUI---C	MOV.L	R5,[R0]	-	00001A58<-FFFF8434
-0035203		FFFF904F	00:00:00.000.021.480	OPUI---C	BSR.A	_printf	-	00001A54<-FFFF9053
-0035202	_printf	FFFF9312	00:00:00.000.021.490	OPUI---C	MOV.L	#0H,R5	-	R5<-00000000
-0035201		FFFF9314	00:00:00.000.021.500	OPUI---C	PUSH.L	R5	-	00001A50<-00000000
-0035200		FFFF9316	00:00:00.000.021.510	OPUI----	ADD	#08H,R0,R4	-	R4<-00001A58
-0035199		FFFF9319	00:00:00.000.021.520	OPUI----	ADD	#7H,R4	-	R4<-00001A5F
-0035198		FFFF931B	00:00:00.000.021.530	OPUI----	MOV.L	08H[R0],R3	-	R3<-FFFF8434
-0035197		FFFF931D	00:00:00.000.021.540	OPUI----	AND	#-04H,R4	-	R4<-00001A5C
-0035196		FFFF9320	00:00:00.000.021.550	OPUI----	MOV.L	#00001590H,R2	-	R2<-00001590
-0035195		FFFF9326	00:00:00.000.021.560	OPUI----	MOV.L	#-00006D0BH,R1	-	R1<-FFFF92F5
-0035194		FFFF932C	00:00:00.000.021.590	OPUI----	BSR.A	__Printf	-	00001A4C<-FFFF9330
-0035193	__Printf	FFFF9675	00:00:00.000.021.640	OPUI----	PUSHM	R6-R10	-	00001A48<-00000000
-0035192		FFFF9677	00:00:00.000.021.650	OPUI---C	ADD	#-00A4H,R0,R0	-	USP<-00001994

Figure 7.14 Trace Window (Trace Information Display)

7.3.3 Performing Trace Search

Click the right mouse button on the [Trace] window to launch the pop-up menu, and select [Find -> Find....] to open the [Find] dialog box.

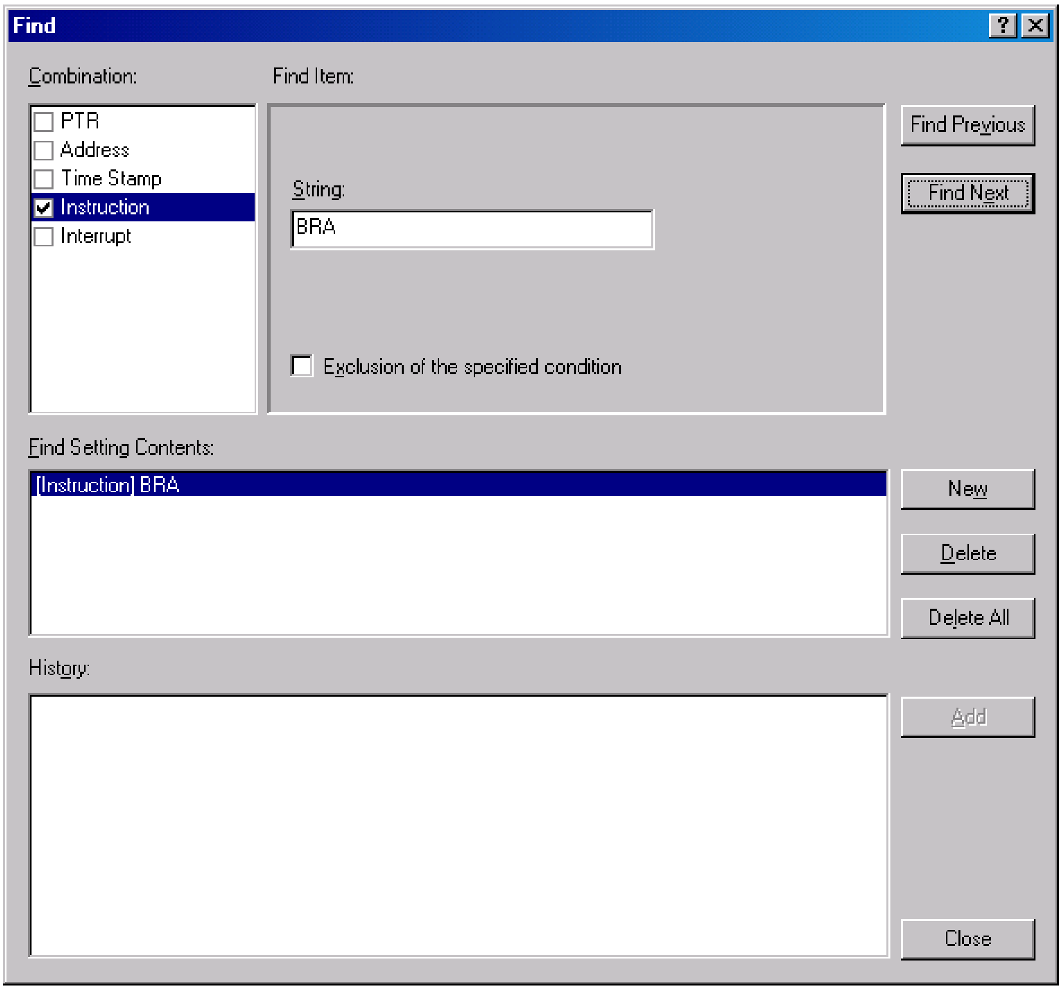


Figure 7.15 Trace Search Dialog Box

Check the check boxes for the conditions to be targets of the search in the [Combination] column, and specify the details of the conditions in the [Find Item] column.

The conditions you have set are shown in the [Find Setting Contents] list box.

After setting search conditions, click the [Find Previous] or [Find Next] button to start a search.

When a matching trace record is found by a search, the relevant line in the [Trace] window is highlighted. When an instance of the trace record was successfully found, choose the [Find Previous] or [Find Next] button from the pop-up menu. The next instance of the trace record will be searched for.

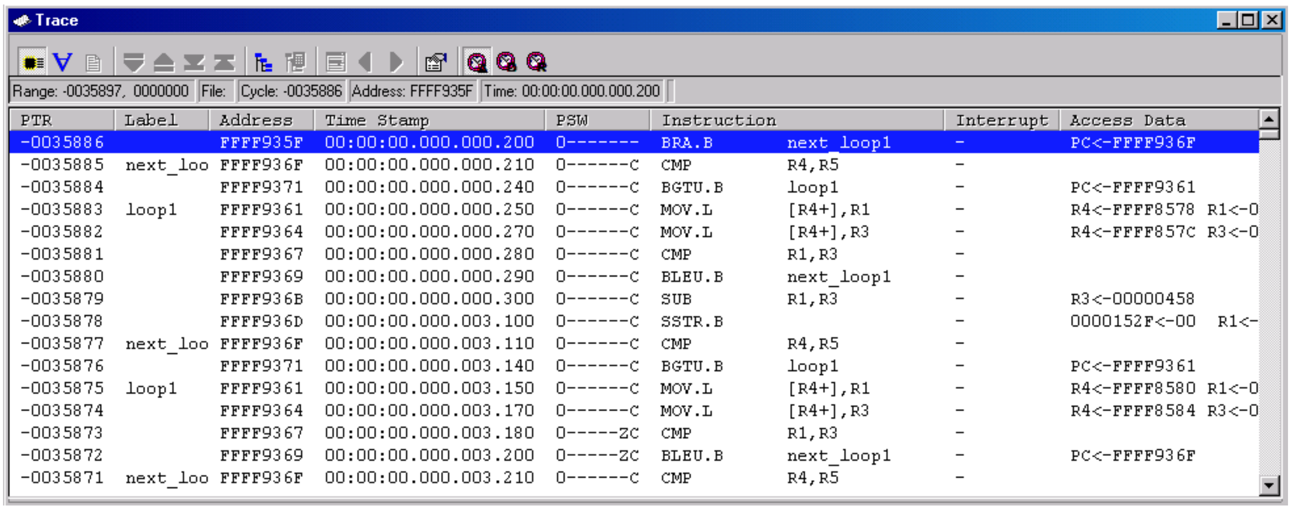


Figure 7.16 Trace Window (Searched Result)

7.3.4 Checking Simulated I/O

Random data that is displayed by the printf function can be checked in the [Simulated I/O] window.

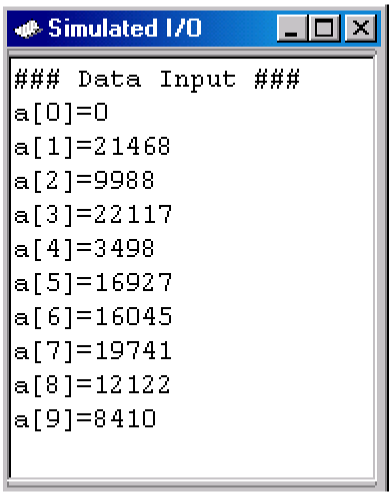


Figure 7.17 Simulated I/O Window

- Do not close the [Simulated I/O] window.

7.3.5 Checking the Breakpoints

A list of all the breakpoints that are set in the program can be checked in the [Event] window.

- Select [Eventpoints] from the [View -> Code] menu.

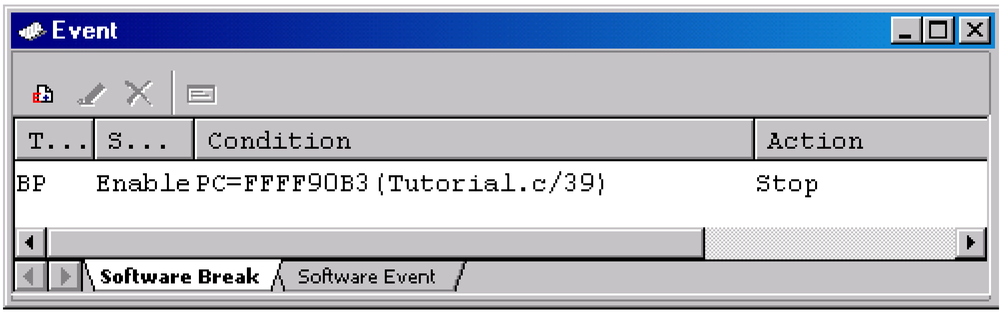


Figure 7.18 Event Window

A breakpoint can be set, a new breakpoint can be defined, and a breakpoint can be deleted using the [Event] window.

- Close the [Event] window.

7.3.6 Watching Variables

It is possible to watch the values of variables used in your program and to verify that they change in the way that you expected. For example, set a watch on the long-type array “a” declared at the beginning of the program, by using the following procedure:

- Select [Watch] from the [View -> Symbol] menu to open the [Watch] window. And click the right mouse button on the [Watch] window and choose [Add Watch...] from the pop-up menu.

The following dialog box will be displayed.

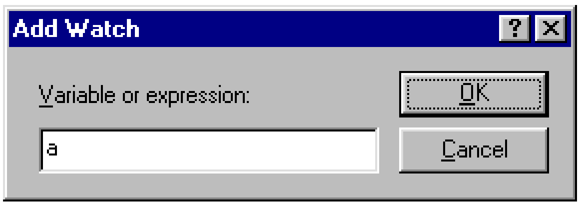


Figure 7.19 Add Watch Dialog Box

- Type array “a” and click the [OK] button.

The [Watch] window will show the long-type array “a”.
You can double-click the + symbol to the left of array “a” in the [Watch] window to expand the variable and show the individual elements in the array.

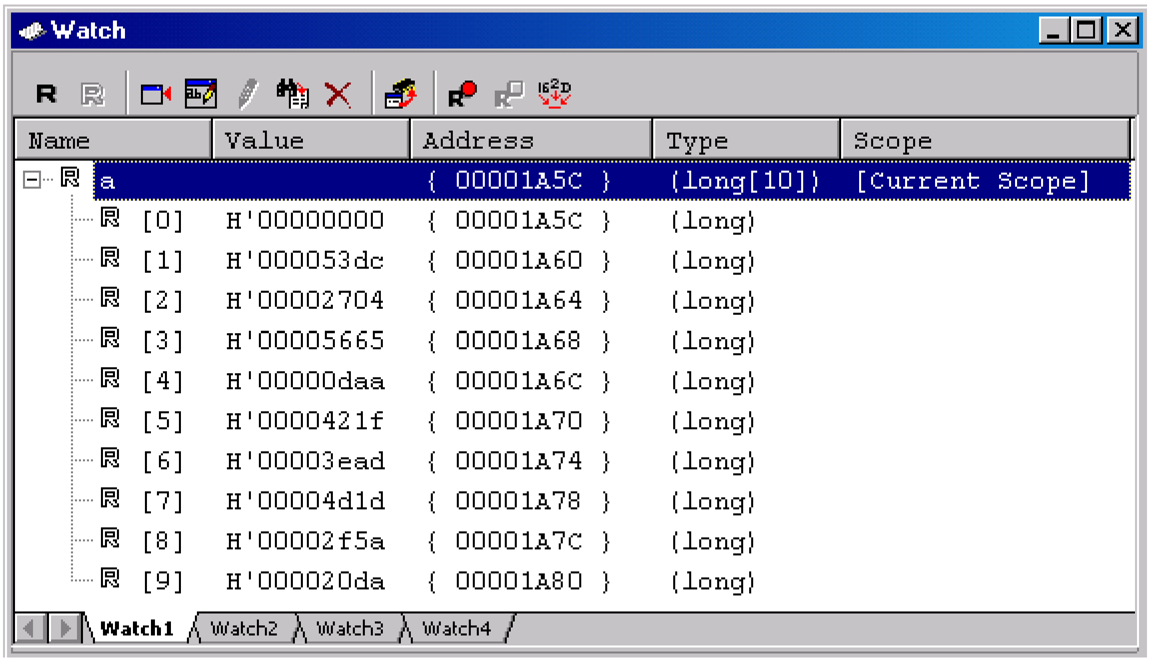


Figure 7.20 Watch Window

- Close the [Watch] window.

7.3.7 Executing the Program in Single Steps

The simulator/debugger has various stepping menus that are useful in debugging the program.

Menu	Description
Step In	Executes each statement (includes statements within the function)
Step Over	Executes a function call in a single step
Step Out	Steps out of a function, and stops at the next statement of the program that called the function
Step...	Executes the specified number of steps at the specified speed

[Step In]: Enters the called function and stops at the statement at the start of the called function.

- To step in the sort function, select [Step In] from the [Debug] menu, or click the [Step In] button on the toolbar.

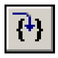


Figure 7.21 Step In Button

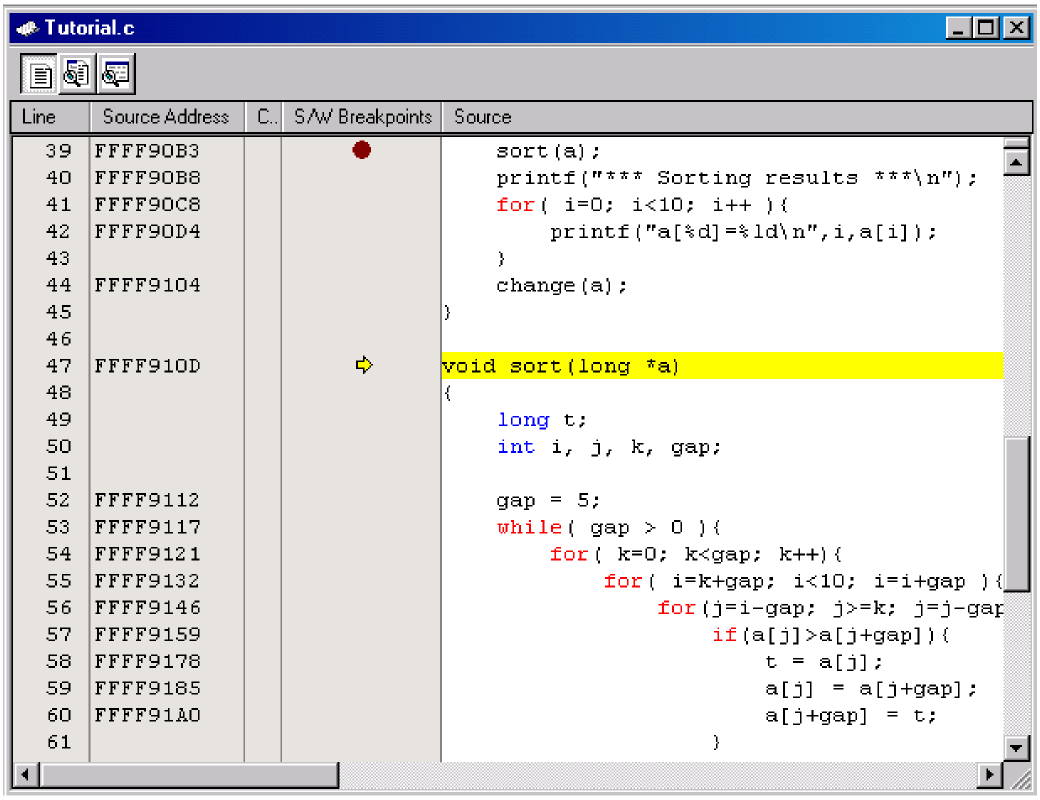


Figure 7.22 Source Window (Step In)

- The PC location display (=>) in the [Source] window moves to the statement at the start of the sort function.

[Step Out]: Steps out of the called function and stops at the next statement in the called program.

- Select [Step Out] from the [Debug] menu to exit the sort function, or click the [Step Out] button on the toolbar.

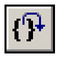


Figure 7.23 Step Out Button

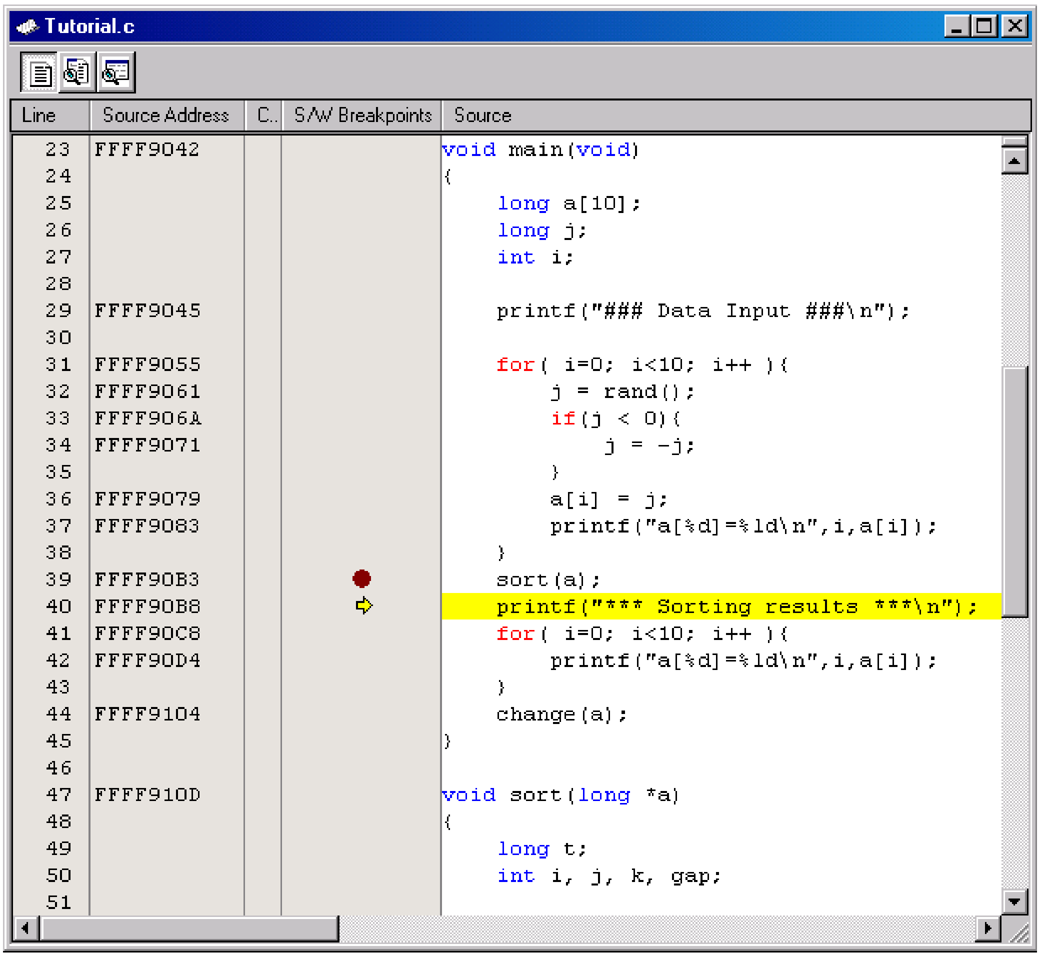


Figure 7.24 Source Window (Step Out)

[Step Over]: Executes a function call in a single step, and stops at the next statement in the main program.

Select [Step Over] from the [Debug] menu or click the [Step Over] button on the toolbar to step over the statements in the printf function.

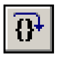


Figure 7.25 Step Over Button

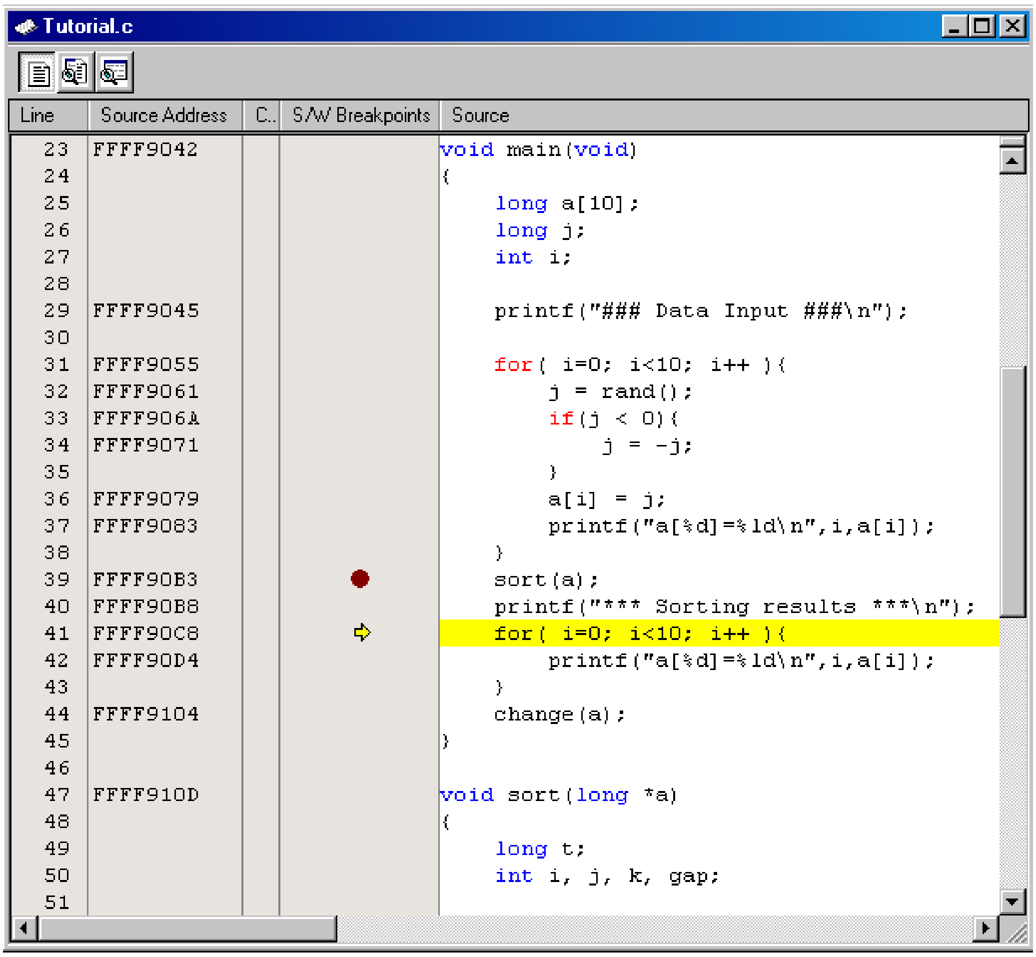


Figure 7.26 Source Window (Step Over)

When the printf function has been executed, *** Sorting results *** will be displayed in the [Simulated I/O] window.

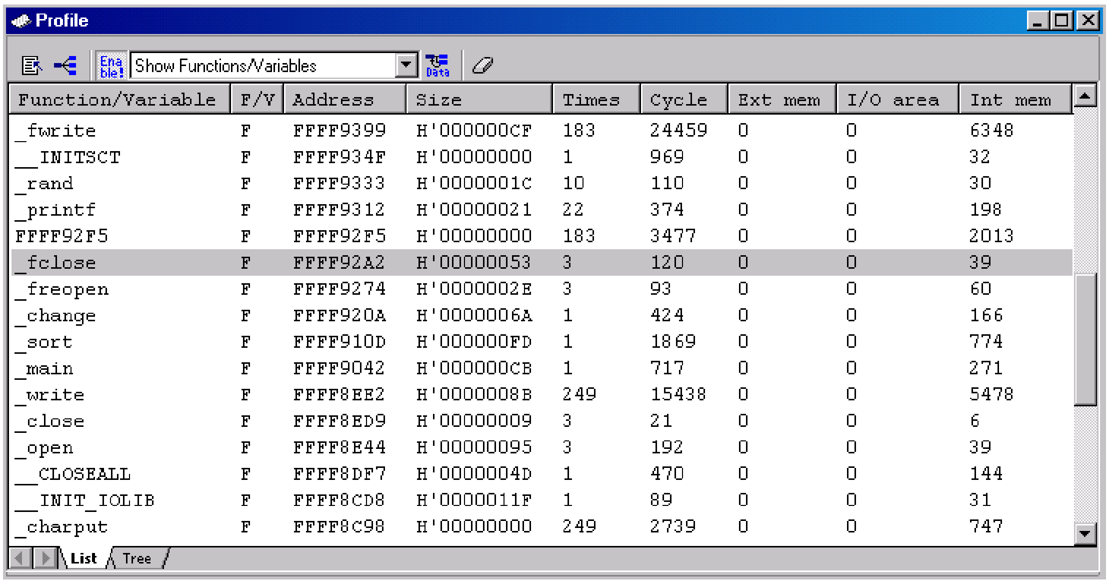
7.3.8 Checking Profile Information

The profile information can be checked in the [Profile] window.

- Clicking the [Go] button and continuing execution from the current PC executes the BRK instruction and then stops.

[List] Sheet: Displays the profile information as a list.

- Open the [Profile] window by selecting [Profile] from the [View->Performance] menu. The [List] sheet will be displayed.



Function/Variable	F/V	Address	Size	Times	Cycle	Ext mem	I/O area	Int mem
_fwrite	F	FFFF9399	H'000000CF	183	24459	0	0	6348
__INITSCT	F	FFFF934F	H'00000000	1	969	0	0	32
_rand	F	FFFF9333	H'0000001C	10	110	0	0	30
_printf	F	FFFF9312	H'00000021	22	374	0	0	198
FFFF92F5	F	FFFF92F5	H'00000000	183	3477	0	0	2013
_fclose	F	FFFF92A2	H'00000053	3	120	0	0	39
_freopen	F	FFFF9274	H'0000002E	3	93	0	0	60
_change	F	FFFF920A	H'0000006A	1	424	0	0	166
_sort	F	FFFF910D	H'000000FD	1	1869	0	0	774
_main	F	FFFF9042	H'000000CB	1	717	0	0	271
_write	F	FFFF8EE2	H'0000008B	249	15438	0	0	5478
_close	F	FFFF8ED9	H'00000009	3	21	0	0	6
_open	F	FFFF8E44	H'00000095	3	192	0	0	39
__CLOSEALL	F	FFFF8DF7	H'0000004D	1	470	0	0	144
__INIT_IOLIB	F	FFFF8CD8	H'0000011F	1	89	0	0	31
_charput	F	FFFF8C98	H'00000000	249	2739	0	0	747

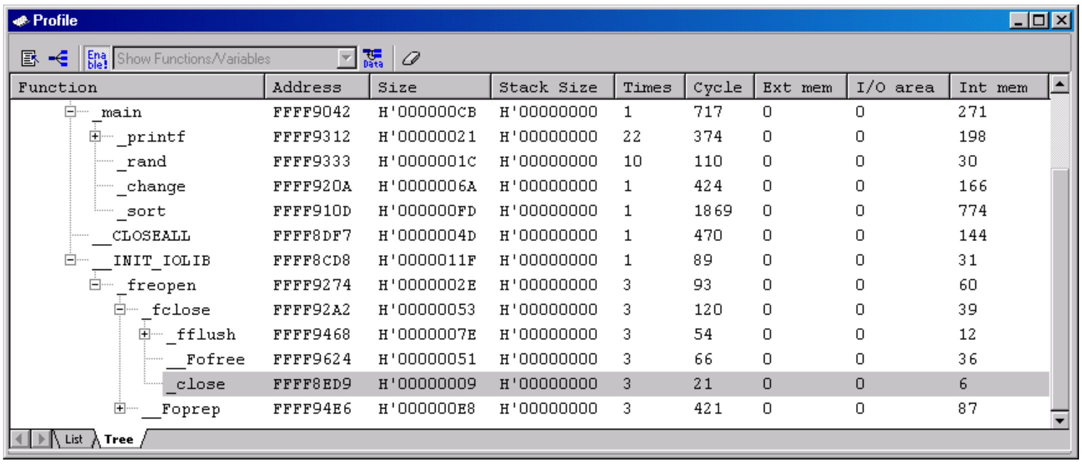
Figure 7.27 Profile Window (List Sheet)

In the above figure, it can be found that the __fclose function was called three times, the execution cycle was 120, and the internal memory was accessed 39 times.

It is possible to search for the critical path, such as a function that is called or accesses the memory many times, for the program performance.

[Tree] Sheet: Displays the profile information as a tree diagram.

- Select the [Tree] sheet. Double-clicking the function name in the [Profile] window expands or minimizes the tree structure.



Function	Address	Size	Stack Size	Times	Cycle	Ext mem	I/O area	Int mem
_main	FFFF9042	H'000000CB	H'00000000	1	717	0	0	271
_printf	FFFF9312	H'00000021	H'00000000	22	374	0	0	198
_rand	FFFF9333	H'0000001C	H'00000000	10	110	0	0	30
_change	FFFF920A	H'0000006A	H'00000000	1	424	0	0	166
_sort	FFFF910D	H'000000FD	H'00000000	1	1869	0	0	774
__CLOSEALL	FFFF8DF7	H'0000004D	H'00000000	1	470	0	0	144
__INIT_IOLIB	FFFF8CD8	H'0000011F	H'00000000	1	89	0	0	31
_freopen	FFFF9274	H'0000002E	H'00000000	3	93	0	0	60
_fclose	FFFF92A2	H'00000053	H'00000000	3	120	0	0	39
_fflush	FFFF9468	H'0000007E	H'00000000	3	54	0	0	12
__Fofree	FFFF9624	H'00000051	H'00000000	3	66	0	0	36
__close	FFFF8ED9	H'00000009	H'00000000	3	21	0	0	6
__Foprep	FFFF94E6	H'000000E8	H'00000000	3	421	0	0	87

Figure 7.28 Profile Window (Tree Sheet)

In above figure, it can be found that the __close function was called three times from the _fclose function, the execution cycle was 21, and the internal memory was accessed six times.

[Profile-Chart] Window: Displays the relation of calls for a specific function.

- Select the __flclose function on the [Profile] window. Open the pop-up menu by right clicking the mouse on the [Profile] window, and select [View Profile-Chart] to display the [Profile-Chart] window.

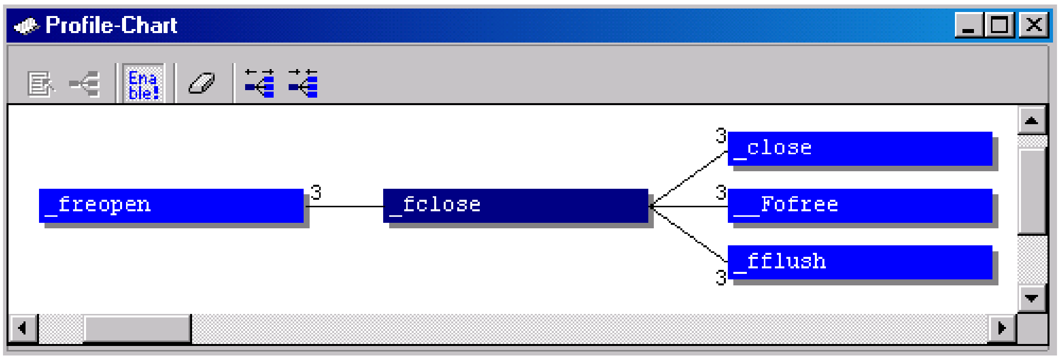


Figure 7.29 Profile-Chart Window

In the above figure, it can be found that the __flclose function was called three times from the __freopen functions, and the _close function was called three times.

This is the end of the tutorial using the simulator/debugger.

RX Family Simulator/Debugger V.1.01
User's Manual

Publication Date: Apr. 1, 2010 Rev.1.00

Published by: Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi,
Kanagawa 211-8668 Japan

Edited by: Renesas Solutions Corp.

RX Family Simulator/Debugger V.1.01
User's Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ10J2162-0100