R32C Simulator Debugger V.1.01.00
Release Notes

This document describes the notes of this debugger, and please read before you start to use this debugger.

And also, please refer to the "High-performance Embedded Workshop Release Notes" about the notes of High-performance Embedded Workshop IDE.

Contents

1 Notes

1.1 Line Assembly

Regardless of the Radix setting, the default for line assembly input is decimal. Specify H as the radix for a hexadecimal input.

1.2 Event Setting

1. TAB order in Set Event Status dialog box
   Even when you press [TAB] key, the next input control may not be focused on the Set Event Status dialog boxes opened from Trace Point.
2. In-place-edit mode on event list
   On event list in Trace Point, High-performance Embedded Workshop will not escape from in-place-edit mode even when you press the [ESC] key.
3. Event setting by BIT SYMBOL
   When the specified address is odd numbers, the setting by BIT SYMBOL can not set the correct condition. Use DATA ACCESS setting and specify the compared data with the data mask. For details about how to set the conditions for odd number addresses, refer to the online help.
4. Event detection for BIT SYMBOL
   When the event is set to detect the access to specified bit, it will be detected even if the other bit of the same address as the specified bit is accessed. This is because the access to the bit from MCU is byte access.

1.3 Output Port

1. Use of Symbol
   If you use a symbol to specify an address in the Set Port Dialog box, an error “Can’t find symbol.“ will occur, when you restart the HEW or get back to the session.

1.4 Data Trace

1. Split-bar behavior when double-clicking
   If you double-click the split-bar, which divides view up and down, the horizontal scroll-bar, vertical scroll-bar, and tabs in the upper view will vanish. Drag the split-bar to display them again.

1.5 Trace

1. Specifying function in SRC mode
   In the SRC mode, when you specify a function to display it, if the current displayed source file includes the function, the top of the source file will be displayed.
2. Saving of tracing result in text
   - When you save a tracing result in text with only “BUS” and “DATA” buttons ON, the vertical position of some headers will shifts from the corresponding data. Check “Tab Separated Format” and open it with spreadsheet applications to display them correctly.
   - When you save a tracing result that includes BUS and DATA access information in text, some headers can not be correct. When you check “Tab Separated Format”, the headers will be correct.
3. Time concerned data
   Displayed time concerned data is not measured, though time concerned buttons and items in pop-up menu are active.(Time measurement is not supported.)
4. Loading the trace image file
   Trace window can not load the trace image file saved by PDxx debuggers. And also, trace window can not load the trace image file saved by the different target from the current target.

1.6 RAM Monitor

1. Proportional Fonts
   When a proportional font is selected, a part of the characters in the view may be hidden. Fixed fonts are recommended.

1.7 Memory

1. 8 bytes data operations
   To set, fill, and copy 8 bytes data are not supported.

1.8 Script

1. Result of interactive command
   When you invoke an interactive command, for example, Assemble and setMemoryByte, the running dialog box will appear and may hide the view of the results.
2. SCOPE Command
   When you refer current scope name with SCOPE command after program execution, the scope of the start-up module will be returned even if scope has been changed to the other.

1.9 Real-time OS debugging functions

1. When several labels are allocated to the entry address of the tasks or handlers, the task name or the handler name displayed in the windows may be different from the actual function name.

2. When you use the feature to issue system-calls by the script command (MR SYS), the target program should be built with a specific option. For details, refer to the topic "Prepare the real-time OS debug" in the online help.

1.10 Macro recording function

The debug windows which support the macro recording function are memory, registers, and IO. And also, the debug operations which support this are Reset CPU, Go, Reset Go, Go To Cursor, Step In, Step Over, Step Out, Add/Delete a break point, and Download the target program.

1.11 Test facility function

The contents to be compared by the test facility functions are memory, registers, I/O, Output, and stack trace window.

1.12 Using cast operators for the member variable

When you use cast operators for the member variable to refer to it as the pointer of the structure, you would not refer to it correctly.

1.13 Download module dialog box

This debugger does not support the setting of "Offset", "Memory verify on download", and "Access Size" in the download module setting dialog box. These are always treated as "Offset: 0", "Memory verify: off", and "Access Size: 1".

1.14 The option "Always treat variables of enumerator type with unknown size as 1 byte"

The "Always treat variables of enumerator type with unknown size as 1 byte" option is effective after downloading the program. When the option status is changed, target program should be downloaded again.

And also, this option is effective for all variables of enumerator type in the program, even if the compiling options are different for each file.

1.15 Debugging for assembler macros

When the break points are set at the assembler macro codes, the break points would be set at the different address or not be displayed as the PC line.

1.16 Debugging for inline functions

When stepping the function including the call for a inline function, local variables would not be able to be referred.

1.17 Automatic target connection on changing the session

When the target connection is not performed on changing the session, select the menu [Debug] -> [Connect]. To perform automatic target connection, remove the check from the option "Do not perform automatic target connection" in the Option tab on Debug Setting dialog box which is invoked by the menu [Debug] -> [Debug Settings…].

1.18 Run program option

The "Run Program" dialog box enables to specify several temporary PC breakpoints, but this debugger only supports one breakpoint which is listed first in the “Temporary PC breakpoints” list box.

1.19 Selection of the object format for download module

When the specified file format in the debug setting dialog box is different from the format of the object module file, downloading the file may cause a freeze of the debugger. Please select the correct object format. And also, when selecting the object format for download module file, if there are two or more object format, whose name includes the vender name another ones do not include it, prioritize the file whose name includes vender name leading the object format name.

1.20 Using the Simulator Debugger Customizing Kit (I/O DLL kit)

When you use the I/O DLL Kit, you should specify I/O DLL file name to sim100.exe’s environment setup file “sim100.ini”.
The sim100.ini file is stored in the directory shown below:

“Workspace directory\Project directory“

Please copy the I/O DLL file to the same directory as the sim100.ini file, and specify the I/O DLL file name to the sim100.ini file when you use the I/O DLL kit.

Example: When you specify the Sample.dll.

```
[DLLNAME]        <- Add
IODLL=Sample     <- Add
```

1.21 Notes on Debugging (R32C Simulator Debugger)

1.21.1 Executed time span and executed cycles

The displayed cycles for execution are calculated by the value described in the software manuals of the micro computer. And the displayed time span for execution is calculated from the cycles and the frequency which is specified in the INIT dialog. The external bus width, the state of instruction queues, or the software wait for instructions are not considered, so the value will be different from the value of the actual MCUs.

To calculate the actual cycles, use the emulator.

1.22 Notes on Programmable sound field processor

To execute the program for programmable sound field processor (hereafter abbreviated sound field processor) on simulator debugger, it is necessary to register the DLL file for sound field processor to simulator debugger.

There are the following two kinds of types in DLL files for sound field processor.

1. DLL for execution of sound field processor program. (PSFPGodll.dll)
   Executes the program for sound field processor.
2. DLL for step execution of sound field processor program. (PSFPStepdll.dll)
   Step executes the program for sound field processor.
   Refers the contents of a special register for sound field processor.

1.22.1 Configuration

PSFPxxdll.dll (xx denotes the name such as Go or Step) operates by registering in simulator engine sim100.exe.

The simulator debugger and the PSFPxxdll.dll are configured in the manner shown below.

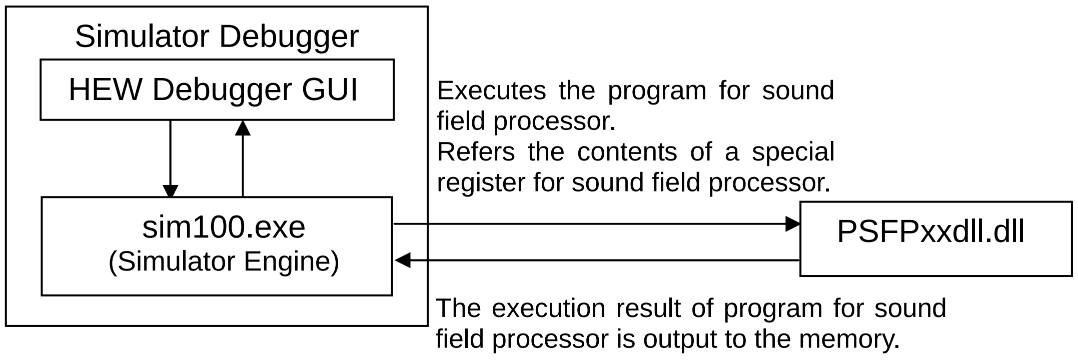
Simulator Debugger
HEW Debugger GUI
sim100.exe
(Simulator Engine)
Executes the program for sound field processor.
Refers the contents of a special register for sound field processor.
PSFPxxdll.dll
The execution result of program for sound field processor is output to the memory.

Figure 1 Configuration of PSFPxxdll.dll

1.22.2 Method for Using

The following explains how to use the PSFPxxdll.dll after registering it to the Simulator Debugger.

1. Register the PSFPxxdll.dll to sim100.exe. To do this, write the PSFPxxdll.dll filename in sim100.exe's environment setup file "sim100.ini".
   - Note that the sim100.ini file and PSFPxxdll.dll are stored in the directory shown below:
     - sim100.ini file
       "Workspace directory\Project directory"
       The sim100.ini file is nonexistent if you have never started Simulator Debugger since you installed it. In that case, this file needs to be newly created using an editor.
     - PSFPxxdll.dll
       "HEW install directory
       \Tools\Renesas\DebugComp\Platform\PDTarget\PD100SIM"
       Please copy PSFPxxdll.dll to the same directory as the sim100.ini file.
   - In the sim100.ini file, create a [DLLNAME] section and write the PSFPxxdll.dll filename after "IODLL=," with the extension ".dll" removed as shown below.

     Example: When you register the PSFPGodll.dll.
```
[DLLNAME]          <- Add
IODLL=PSFPGodll    <- Add
```
2. When you start Simulator Debugger, the PSFPxxdll.dll is loaded. If you do not use the PSFPxxdll.dll, delete the description of the [DLLNAME] section that you created in the sim100.ini file before starting Simulator Debugger.

   Example: When you release the registration of PSFPGodll.dll.
```
[DLLNAME]          <-Delete
IODLL=PSFPGodll    <-Delete
```

1.22.3 Execution processing

The execution processing is different in PSFPGodll.dll and PSFPStepdll.dll. To explains the execution processing of each DLL as follows.

1.22.3.1 Execution processing of PSFPGodll.dll

The sound field processor program starts running when a data write to the address 20000h (a condition for the sound field processor program to start) is executed in the target program.

The target program is kept waiting while the sound field processor program is running. Then when execution of the sound field processor program is completed, the target program restarts from where it left off.

A sound field processor-completed interrupt (audio interface 1 interrupt) is generated when execution of the sound field processor program is completed (FIN instruction executed). Therefore, control branches to the interrupt routine before the target program restarts (i.e., before the next instruction is executed). Note that the sound field processor-completed interrupt is generated with priority level 7.

Figure 2 shows the flow of program execution control.

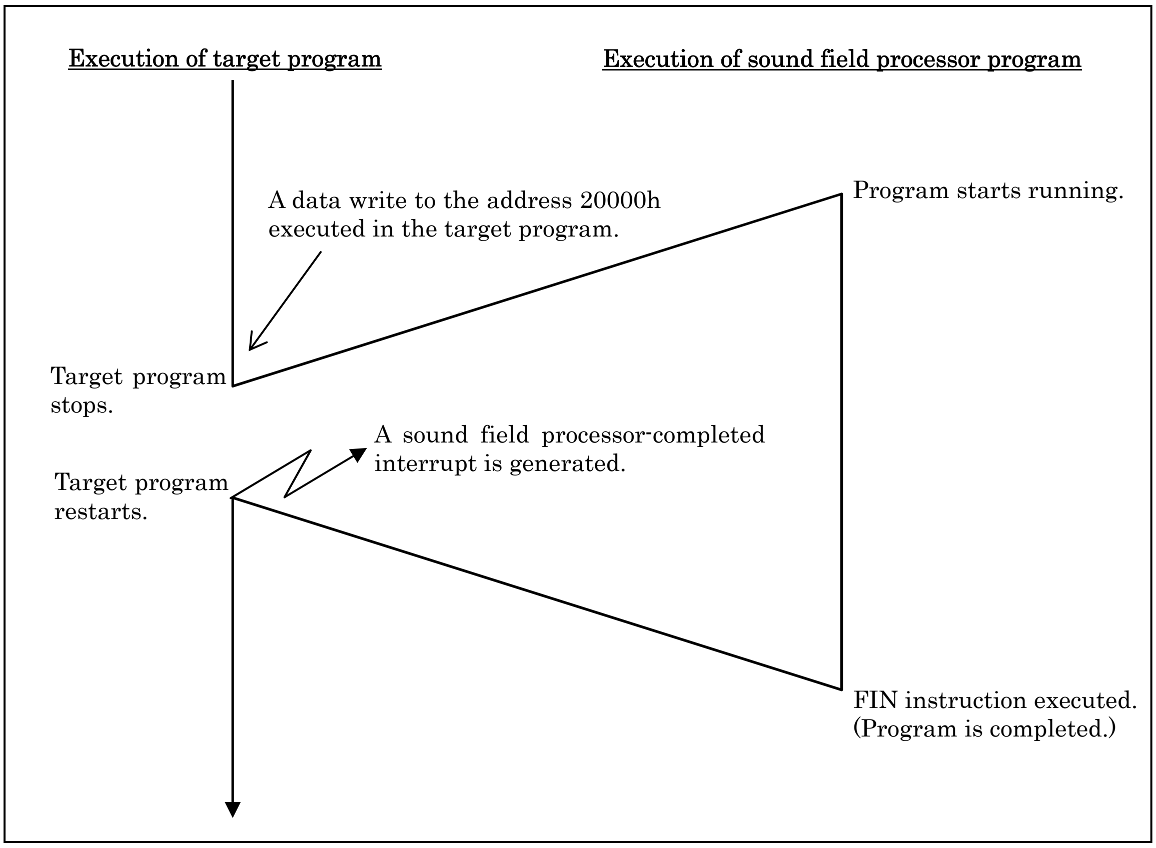
Execution of target program
Execution of sound field processor program
A data write to the address 20000h executed in the target program.
Program starts running.
Target program stops.
A sound field processor-completed interrupt is generated.
Target program restarts.
FIN instruction executed. (Program is completed.)

Figure 2 The flow of program execution control of PSFPGodll.dll.

1.22.3.2 Execution processing of PSFPStepdll.dll

The sound field processor program stops after executing single-stepped instruction when a data write to the address 20000h (a condition for the sound field processor program to start) is executed in the target program.

The target program stops at the next instruction after a data write to the address 20000h is executed. At this point of time, the target program and the sound field processor program have both stopped running.

Use script commands to run or single-step the sound field processor program and to inspect the dedicated registers. For each script command specification, refer to Table 1.

A sound field processor-completed interrupt (audio interface 1 interrupt) is generated when execution of the sound field processor program is completed (FIN instruction executed). Therefore, if the target program is restarted, control immediately branches to the interrupt routine. Note that the sound field processor-completed interrupt is generated with priority level 7.

Figure 3 shows the flow of program execution control.

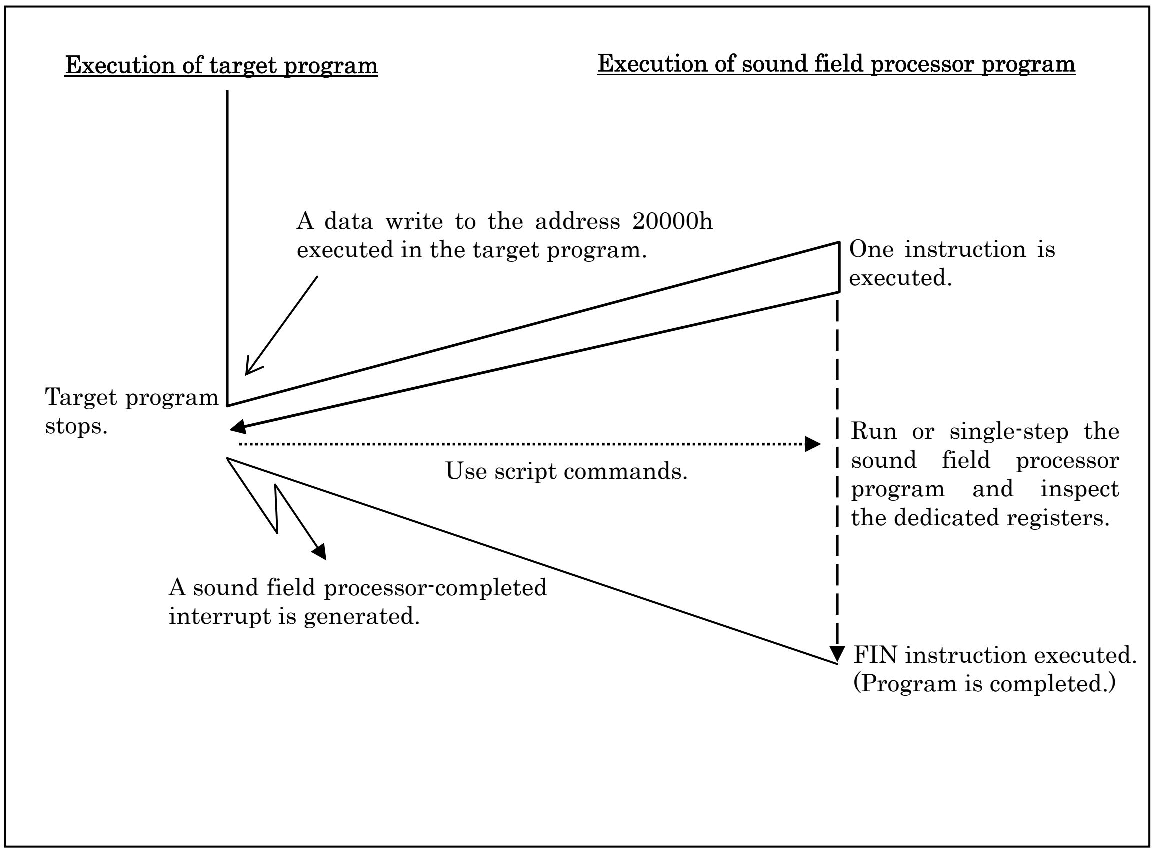
Execution of target program
Execution of sound field processor program
A data write to the address 20000h executed in the target program.
One instruction is executed.
Target program stops.
Use script commands.
Run or single-step the sound field processor program and inspect the dedicated registers.
A sound field processor-completed interrupt is generated.
FIN instruction executed. (Program is completed.)

Figure 3 The flow of program execution control of PSFPStepdll.dll

Table 1 Specification of script command for sound field processor

- Execution of program for sound field processor

Command name (Short name)	PAFPGo(PG)
Input formats	PafpGo
Functions	Executes the program for sound field processor. It stops when the FIN instruction is executed. This command cannot be used while executing the target program.

- Step execution of program for sound field processor

Command name (Short name)	PAFPStep(PS)
Input formats	PafpStep
Functions	Step executes the program for sound field processor. This command displays the PC value of sound field processor as an execution result. When the FIN instruction is executed, the “(FIN)" character string is added to the PC value and displayed. This command cannot be used while executing the target program.

- Reference of special register for sound field processor

Command name (Short name)	PAFPRegister(PR)
Input formats	PafpRegister
Functions	Refers the contents of a special register for sound field processor. This command cannot be used while executing the target program.

1.22.4 Notes

Simulation of the sound field processor operation is subject to some limitations, as described below

- The script commands to run or single-step the sound field processor program and to inspect the dedicated registers can only be used when PSFPStepdll.dll is registered.
- When PSFPStepdll.dll is registered, do not run the sound field processor program from the line in which a data write to the address 20000h as a start condition is executed. After the sound field processor program start condition is executed, the target program cannot be stopped running.
- Once the sound field processor program starts running, it cannot be stopped until the FIN instruction is executed.
- The execution result of the sound field processor program can be checked in only the memory area for the sound field processor program located in the CPU. Use the memory window, RAM monitor window, etc. of the simulator debugger to see the memory area for the sound field processor program. (Displayable with only address locations on the CPU side). In the RAM monitor window, however, the function to display the attribute information (READ/WRITE) on memory accessed in the sound field processor program in separate colors is not supported.
- Although the sound field processor program permits its execution result to be checked in said memory area and the sources to be displayed in SRC mode of the program and source windows, it does not support the function to display sources in MIX/DIS modes, as well as breakpoint setting, trace measurement and other simulation functions.
- If the target program is reset, the sound field processor program is also reset.

2 System Requirements

2.1 Operating Environment (Windows® XP or Windows® 2000)

PC Environment	
PC	IBM PC/AT compatible
OS	32-bit editions of Windows® XP *1 *2 Windows® 2000 *1
CPU	Pentium 4 running at 1.6 GHz or more recommended
Memory	768 Mbytes or larger (more than 10 times the file size of the load module) recommended
Hard disk	Installation of the simulator debugger requires free space of 200 Mbytes or larger. Also keep additional free space that is at least twice the memory capacity (four times or larger recommended) for use as swap space.
Display resolution	1024 × 768 or higher recommended

2.2 Operating Environment (Windows Vista®)

PC Environment	
PC	IBM PC/AT compatible
OS	32-bit editions of Windows Vista® *1 *3
CPU	Pentium 4 running at 3 GHz or Core 2 Duo running at 1 GHz or more recommended
Memory	1.5 Gbytes or larger (more than 10 times the file size of the load module) recommended
Hard disk	Installation of the simulator debugger requires free space of 200 Mbytes or larger. Also keep additional free space that is at least twice the memory capacity (four times or larger recommended) for use as swap space.
Display resolution	1024 × 768 or higher recommended

*1: Windows and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.
*2: The 64-bit editions of Windows® XP is not supported.
*3: The 64-bit edition of Windows Vista® is not supported.

3 Version Report

This section describes the specification of the changed software.

3.1 R32C Simulator Debugger V.1.01.00

In this version, the following specifications were changed from the previous version R32C Simulator Debugger V.1.00.00.

This version supports all of the function extensions and the revisions to the restrictions in the High-performance Embedded Workshop V.4.03.00, V.4.04.00, V.4.04.01, V.4.05.00 and V.4.05.01. For more details, please refer to the RENESAS TOOL NEWS "070701/tn1" issued on July 1, 2007, "071216/tn5" issued on December 16, 2007, "080118/tn1" issued on January 18, 2008, "081125/tn1" issued on November 25, 2008 and "090201/tn3" issued on February 1, 2009.

3.1.1 Revisions to Restrictions

1. A limitation has been corrected: If you perform an operation by applying Bank1 register direct of Extended Instruction Addressing to the dest operand of a multiplication or division instruction with the B flag of the FLG register cleared to "0", the B flag may be set to "1". (For more details, refer to the RENESAS TOOL NEWS "071001/tn4" issued on October 1, 2007).
2. A limitation has been corrected: If the value of an int-type variable of 32 bits long is displayed, the upper 16 bits take always 0s. (For more details, refer to the RENESAS TOOL NEWS "081101/tn4" issued on November 1, 2008).

3.1.2 Functional Extensions and Modifications

1. The 32-bit editions of Widows Vista® is supported. You can run the revised product on Windows Vista® by using your user rights.
2. The number of execution cycles conforms to the specification of the following manuals.
   "R32C/100 SERIES SOFTWARE MANUAL Rev.1.00"
3. The MR window and the MR command were added.
4. The IAR C Compiler is supported.
5. Displaying source files information automatically in the Workspace Window. When a download module file is downloaded, the High-performance Embedded Workshop obtains source files information contained in the download module file through its debug information, and displays it under "download module" node in the Projects tab on the Workspace Window. Note that this function is available only for debugging the debug-only projects.
6. Supports C watch window included in High-performance Embedded Workshop V.4.03. The feature Changing scope of the variable and suppression leading zero are supported.
7. The option to specify the size (1byte or 2byte) of enumerator type is added.
8. Supports "Disconnect target" feature.
9. When the instruction is modified by the line assemble feature, if the old instruction length is longer than the new instruction length, NOP instructions are inserted automatically so as to suit the old instruction length.
10. The instruction format specifier is displayed for each instruction in disassembly. And also, can be switched not to display.
11. Supports to substitute the bit field member variable.
12. The default value of "Reset CPU after download module" is changed to be checked.