

FS2 Trigger-Trace Windows for GDB/Insight for MIPS Processors

User Guide

January 29, 2004



First Silicon Solutions, Inc.
4000 SW Kruse Way Place
Bldg 3, Suite 210
Lake Oswego, OR 97035
voice: +1-503-489-0311
fax: +1-503-489-0315
info@fs2.com
support@fs2.com

Copyright © 2004 First Silicon Solutions, Inc.

Table of Contents

1	Introduction	3
2	Installation	3
3	Building the Application.....	3
4	Starting up Insight and FS2 windows	4
5	GDB/Insight and FS2 windows Tutorial using the installed fs2_ex example program	7
5.1	Accessing on-line Help	7
5.2	Starting Insight and FS2 windows	8
5.3	Loading Symbols.....	9
5.4	Setting Trace Modes.....	11
5.5	Interleaved Source lines and disassembly lines	12
5.6	Source-only view	13
5.7	Including load/store cycles.....	14
5.8	Full path for source lines.....	15
5.9	MDI view	16
5.10	Trace Message view.....	16
5.11	Turning Trace on and off	17
5.12	Data Triggers	19
5.13	Example of measuring the duration of the program loop.....	23
5.14	Saving the trace to a file	24
6	Conclusion	24

1 Introduction

This document describes how to install and use the FS2 HW Triggers, HW Trace, and Trace Mode windows with the GDB/Insight GUI environment. The HW Triggers window provides all the setup fields necessary to program the MIPS hardware execution triggers and data triggers which can be used to break the processor or control the turning of trace on and off.

The Trace Mode window provides all the setup choices to control what cycle types go into the trace buffer. It also provides the setting of the trace clock ratio and the trace buffer.

The HW Trace window provides various views of trace results, including source line insertion and variable name lookup.

These FS2 Trigger/Trace windows are tcl/tk-based and are part of the FS2 Console code. All the features are also available through the console commands (except symbolic trace insertions). If you are writing scripts that involve setting up triggers or trace modes, you can make the setups through the windows, display the results with the console command, then copy and edit the output into a script file.

2 Installation

Details about the installation of files, setup information, and procedures for the GDB/Insight environment are in Appendix F of the “Getting Started: ISA-MIPS Debugger” manual (MIPS-Getting-Started.pdf). This is available through Start > Programs > FS2 > ISA-MIPS Getting Started Manual menu.

3 Building the Application

A MIPS target application can be built using the make facility included in the MIPS SDE installation. The example program provided by the FS2 software installation in the fs2_ex directory that is subsequently used in this document can be built with the steps and explanations that follow.

```
open a bash shell
cd to ../examples/fs2_ex directory
```

The Makefile in the fs2_ex directory contains the following lines:

```
PROG          =fs2_ex
OBJS          =fs2func1.o fs2func2.o fs2func3.o fs2_ex.o
CFLAGS        =-O0 -g
SBD           =MALTA32LJ
include ../make.mk
```

Note that this example builds an image for the MALTA32LJ target board. The “L” stands for “little endian” and the “J” stands for the FS2 JTAG control box.

The next step is to run this makefile. The sde-make defaults to running the Makefile script:

```
sde-make fs2
```

The “fs2” switch directs the make files to create a symbol file that will be used by the FS2 HW Trace window. The specific make file that includes the command to make the symbol file for the MALTA board is the file:

```
../sde/kit/malta/fs2.mk
```

One can override the default compiler flags defined in Makefile (and any other .mk file). An example is:

FS2 Trigger-Trace Windows for GDB/Insight for MIPS: User Guide

```
sde-make fs2 CFLAGS='-O1 -g -ffast-math'
```

The resulting load file for this example is:

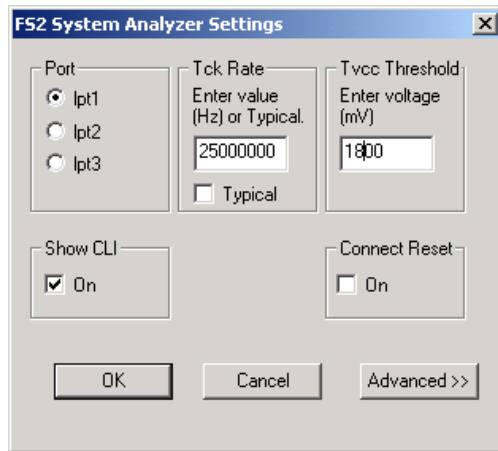
```
fs2_exram
```

4 Starting up Insight and FS2 windows

From the prompt of the Bash shell, type:

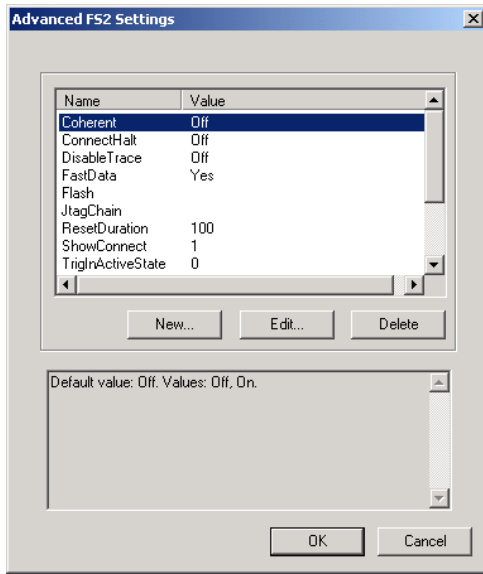
```
sde-gdb fs2_exram
```

By installation default, the first dialog that comes up is the FS2 System Analyzer Settings dialog.¹ It allows you to modify the FS2 Hardware Probe settings like the port connection, Tck Rate, and Tvcc Threshold voltage. Here is an example:



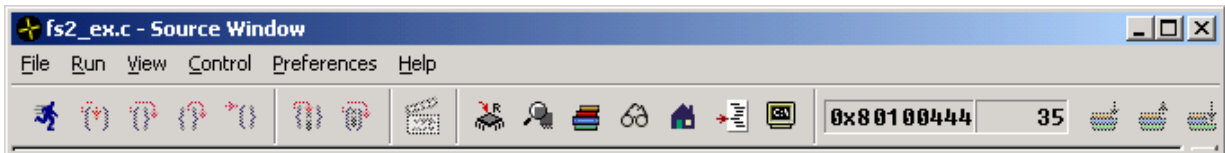
The Advanced button brings up a list of name-value pairs that are stored in the [MIPS] section of the fs2.ini file (located in your system directory – e.g. Windows or WINNT) that determine the operation of the FS2 probe. Example:

¹ If you get an error message that the file fs2mips.dll cannot be found, the likely problem is that the path to this file has not been set in the cygwin environment. Add this path to the cygwin environment. The default (Windows) path of this .dll is c:\program files\fs2\mips\bin. The cygwin path would be something like /cygpath/progra~1/fs2/mips/bin.

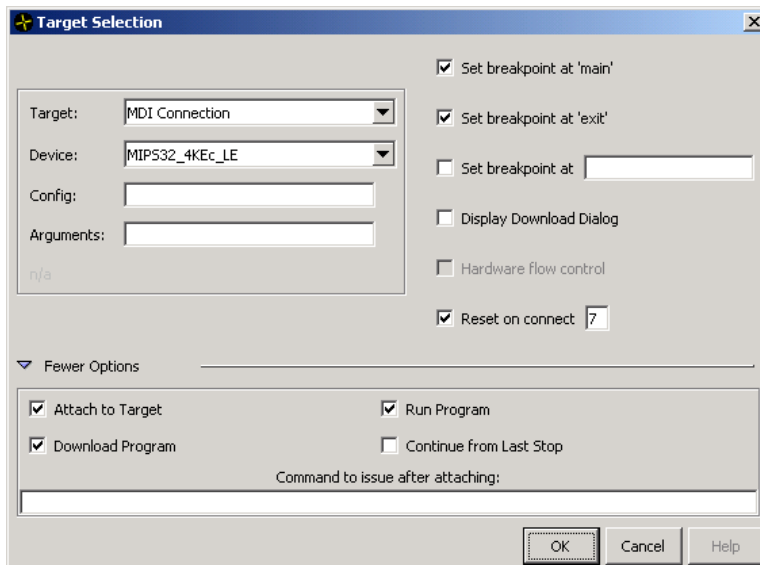


Once these values are set and the target connection is known to work, you can turn off this pop-up dialog by changing ShowConnect value to 0. To have the dialog appear again when starting Insight, edit fs2.ini and change the value back to 1.

The next step is to click on the far left “Run (R)” icon (or the “Run > Connect to target” menu):



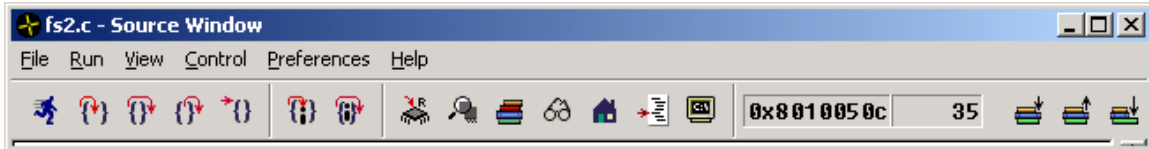
This brings up the Target Selection dialog. The first time it comes up, make sure the Target: field is “MDI Connection” and the other fields and check boxes are the same as the screen capture below (note that the “More Options” area has already been expanded). Note that the Device: field may be different depending on the target processor and whether LE (Little Endian) or BE (Big Endian) is used. Make sure the “Reset on connect” field has the value 7 in it; this is the number of seconds after the target is reset and started before Insight starts to download the program; it allows the target to boot and configure itself. The last checkbox should be unchecked – i.e. Continue from Last Stop.



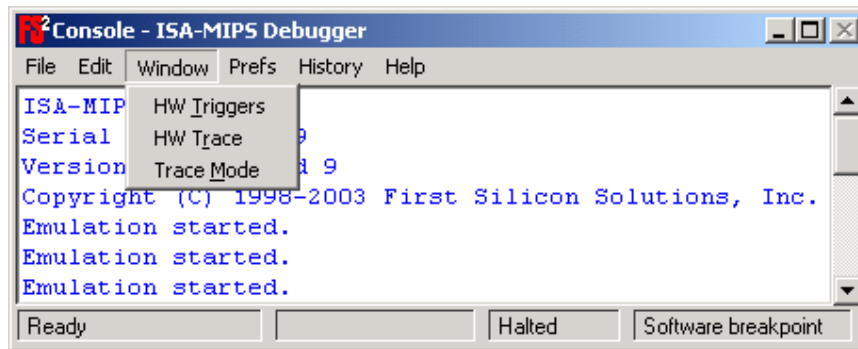
FS2 Trigger-Trace Windows for GDB/Insight for MIPS: User Guide

If the Device: field has the text "Unknown_LE" in it, this means that the FS2 probe could not interrogate the target to determine its type. This is usually a result of both ConnectReset and ConnectHalt set to "off". Type "config" in the FS2 console to see what the settings are. They can be changed with the commands "config ConnectReset on" and "config ConnectHalt on". These parameters are stored in the fs2.ini file in your WINDOWS or WINNT directory. It can also be edited directly or use the Advanced FS2 Settings dialog pictured previously.

Once the dialog is configured, clicking the OK button will reset and start the target executing, wait 7 seconds with the above settings of "Reset on connect 7", then start the download. When it has completed, all the Control icons are available to use as shown below:



To bring up the HW Triggers, HW Trace, or Trace Mode windows, click on the appropriate menu item under FS2 Console Window menu:

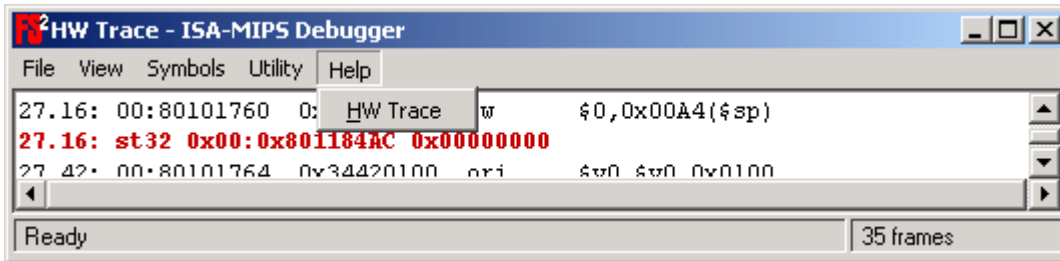


5 GDB/Insight and FS2 windows Tutorial using the installed fs2_ex example program

This tutorial is based on the fs2_ex program; the description of how to build and load it is in the previous section. The source files have been installed either in the cygwin path ending in `../sde/examples/fs2_ex` or, if the cygwin path did not exist, in the FS2 directory path `\progra~1\fs2\mips\tutorial`. If the latter, move the tutorial directory to your cygwin-based `../sde/examples` directory (since it is required for the make files to work) then rename it "fs2_ex".

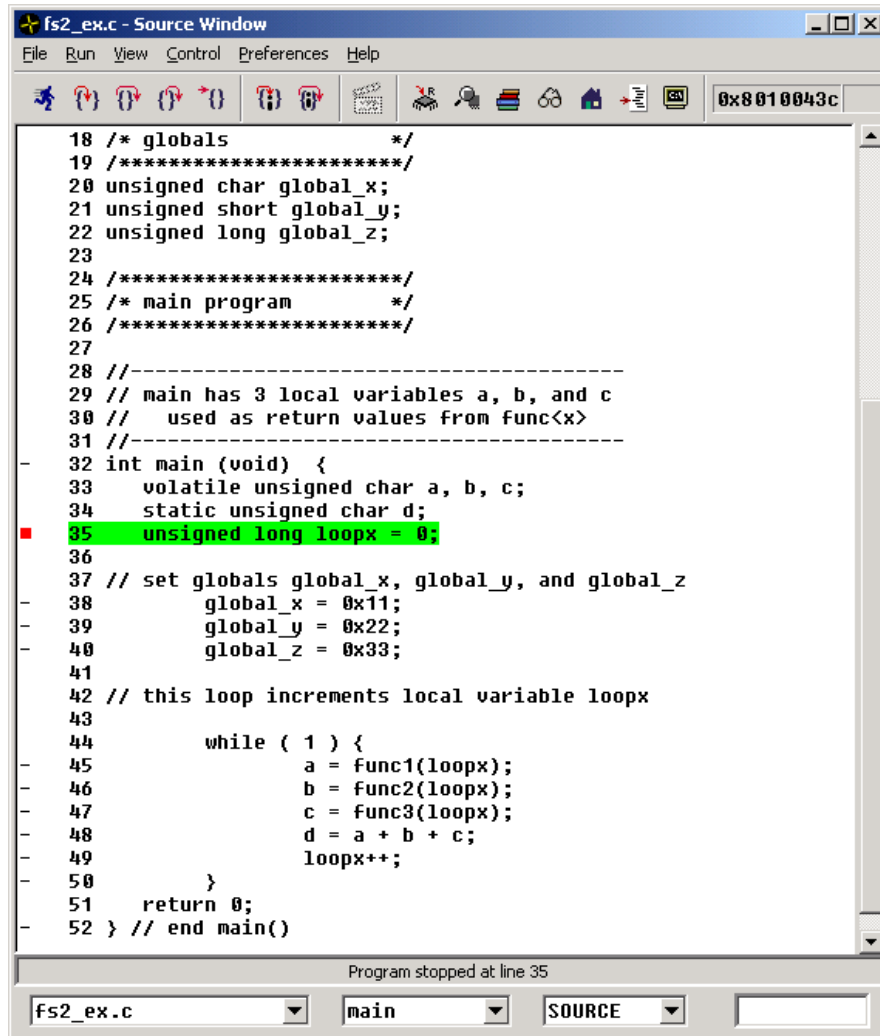
5.1 Accessing on-line Help

The windows operation, menus, and setup fields are described in detail in the FS2 on-line help. Each of these windows has a `Help > <window name>` menu (e.g. `Help > HW Trace` shown below) that will bring up the help text for that window. Because on-line help covers the menus and field entries, this tutorial will not duplicate these detailed descriptions.



5.2 Starting Insight and FS2 windows

When Insight has started along with the FS2 console², and the fs2_exram program has loaded, the Insight Source Window should look like the following:

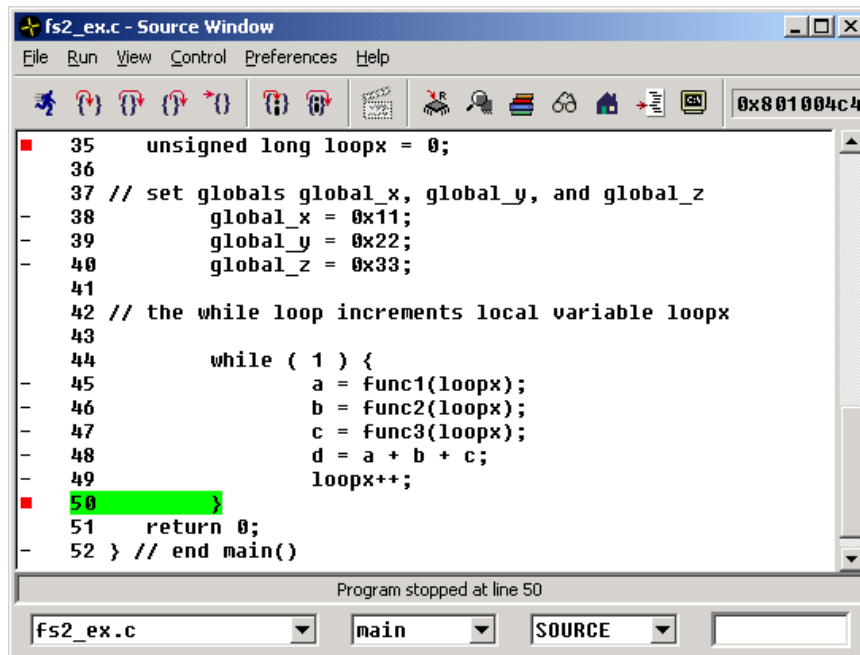


```
18 /* globals */
19 /*****/
20 unsigned char global_x;
21 unsigned short global_y;
22 unsigned long global_z;
23
24 /*****/
25 /* main program */
26 /*****/
27
28 //-----
29 // main has 3 local variables a, b, and c
30 // used as return values from func<x>
31 //-----
32 int main (void) {
33     volatile unsigned char a, b, c;
34     static unsigned char d;
35     unsigned long loopx = 0;
36
37     // set globals global_x, global_y, and global_z
38     global_x = 0x11;
39     global_y = 0x22;
40     global_z = 0x33;
41
42     // this loop increments local variable loopx
43
44     while ( 1 ) {
45         a = func1(loopx);
46         b = func2(loopx);
47         c = func3(loopx);
48         d = a + b + c;
49         loopx++;
50     }
51     return 0;
52 } // end main()
```

² If the FS2 console is not visible or in your windows taskbar, insert "ShowConsole=1" in the fs2.ini file (located in your windows or winnt directory) in the [mips] section and restart GDB/Insight.

FS2 Trigger-Trace Windows for GDB/Insight for MIPS: User Guide

Next, set a breakpoint on line 50 then click on Continue. The Source Window is updated as follows:



```
fs2_ex.c - Source Window
File Run View Control Preferences Help
0x801004c4
35 unsigned long loopx = 0;
36
37 // set globals global_x, global_y, and global_z
38     global_x = 0x11;
39     global_y = 0x22;
40     global_z = 0x33;
41
42 // the while loop increments local variable loopx
43
44     while ( 1 ) {
45         a = func1(loopx);
46         b = func2(loopx);
47         c = func3(loopx);
48         d = a + b + c;
49         loopx++;
50     }
51     return 0;
52 } // end main()
Program stopped at line 50
fs2_ex.c main SOURCE
```

To view the HW Trace window, top the FS2 Console then click on Windows > HW Trace. To see all the disassembled instructions and source line insertions, first check the "View > Dasm view" menu.

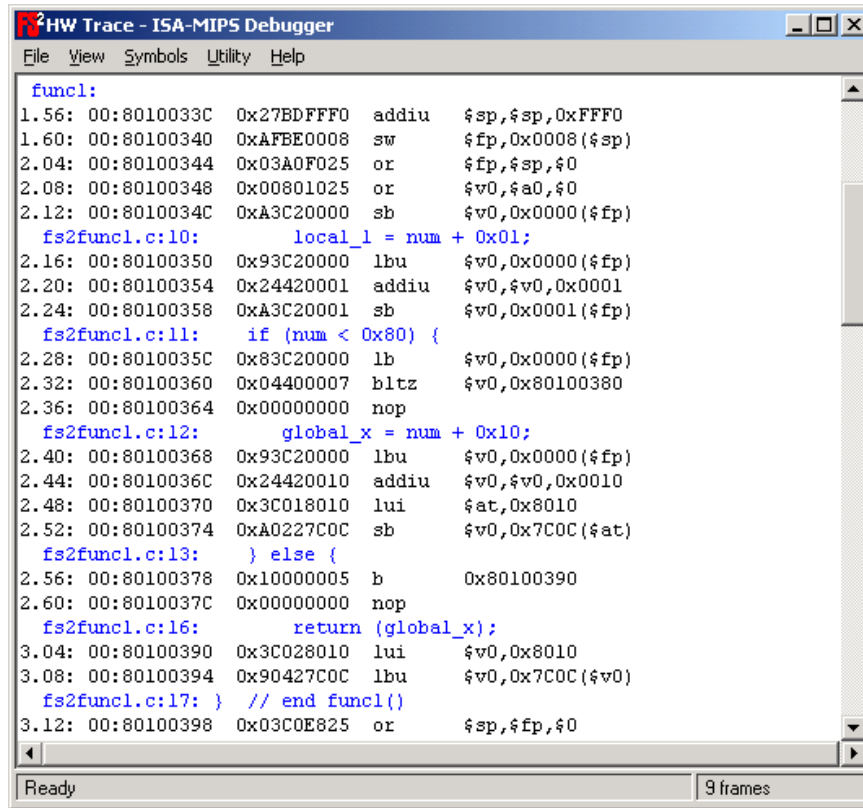
5.3 Loading Symbols

Next, if this is the first time bringing up symbols for this program with the HW Trace window, load the symbol file using the "Symbols > Load Symbol File..." menu. A standard file load dialog opens up with a "File of type:" field of "objdump files (*.fs2)". Navigate to the "../examples/fs2_ex" directory and double click on "fs2_exram.fs2". A pop-up message reports the completion of symbol loading with the number of symbols loaded.

If you recompile your program (while Insight is still running) and want to reload symbols, use the "Symbols->Reload Symbol File" menu item. Alternatively, if you close the trace window then open it again, the last symbol file that was loaded will be automatically reloaded.

Loading multiple symbol files merges all of the symbols into one internal table. If you need to remove symbols, use the "Symbols->Unload Symbol File" menu item. Only one code symbol or one data symbol are allowed per address; additional symbols with the same address will be ignored.

Getting back to hardware trace, your trace display should now look something like the next screen. When symbols are loaded and enabled (Symbols > Show Symbols is checked) the filename, line number, and source line are inserted in a trace line above the code whose address exactly matches the line number address.



You can view the beginning of the trace frames in several ways – the Ctrl-Home shortcut key or the menu “Utility > Go to frame...” and enter 0. To turn on the single function:line number and source line on one line, check “Symbols > Dasm options... > Short file paths”.

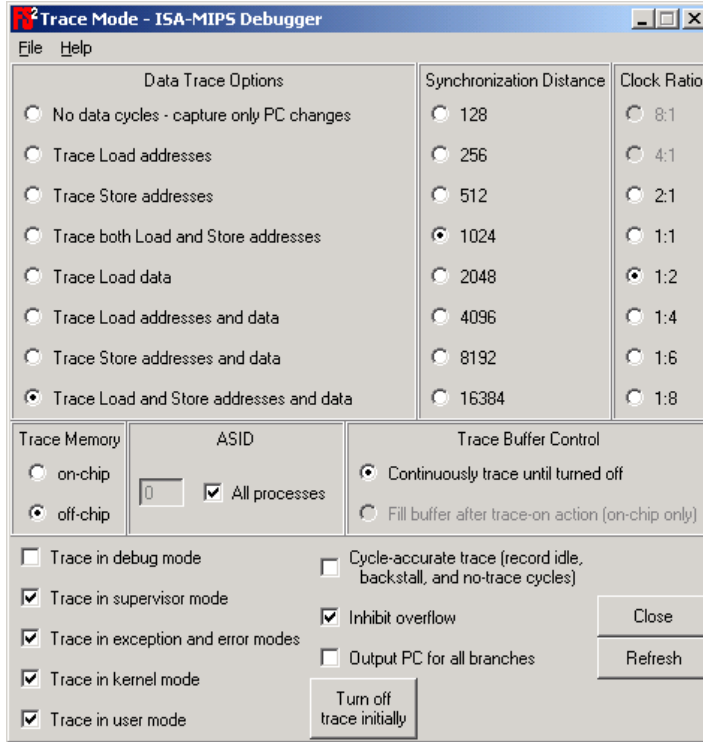
When using the Insight Source Window with the FS2 probe and its setup of hardware triggers, you may get the following Warning when the processor breaks:



This indicates that a hardware breakpoint or hardware trigger has occurred that Insight was not specifically coded to handle it. This is normal; simply click on “Don't show this warning again”.

5.4 Setting Trace Modes

To add load and store cycles into the trace, bring up the Trace Mode window from the FS2 console Windows > Trace Mode window menu. Click on the “Trace Load and Store addresses and data” item under Data Trace Options.



5.5 Interleaved Source lines and disassembly lines

“Continue” again in the Insight Source Window. Go to the top of the trace and it should look like the following screen with “func1:” near the top. In this example, store instruction lines are displayed in red and loads are in green to distinguish data cycles from disassembled code. These colors, fonts, and styles can be changed with the View > Highlighting menus.

```

func1:
2.16: 00:8010033C 0x27BDFFF0 addiu  $sp,$sp,0xFFFF
2.45: 00:80100340 0xAFBE0008 sw    $fp,0x0008($sp)
2.45: st32 0x00:0x801184D0 0x801184D8
3.12: 00:80100344 0x03A0F025 or    $fp,$sp,$0
3.38: 00:80100348 0x00801025 or    $v0,$a0,$0
4.04: 00:8010034C 0xA3C20000 sb    $v0,0x0000($fp)
4.04: st8 0x00:0x801184C8 0x01
fs2func1.c:10:      local_l = num + 0x01;
4.33: 00:80100350 0x93C20000 lbu   $v0,0x0000($fp)
4.33: ld8 0x00:0x801184C8 0x01
5.28: 00:80100354 0x24420001 addiu  $v0,$v0,0x0001
5.54: 00:80100358 0xA3C20001 sb    $v0,0x0001($fp)
5.54: st8 0x00:0x801184C9 0x02
fs2func1.c:11:      if (num < 0x80) {
6.24: 00:8010035C 0x83C20000 lb    $v0,0x0000($fp)
6.24: ld8 0x00:0x801184C8 0x01
7.20: 00:80100360 0x04400007 bltz  $v0,0x80100380
7.46: 00:80100364 0x00000000 nop
fs2func1.c:12:      global_x = num + 0x10;
8.16: 00:80100368 0x93C20000 lbu   $v0,0x0000($fp)
8.16: ld8 0x00:0x801184C8 0x01
9.12: 00:8010036C 0x24420010 addiu  $v0,$v0,0x0010
9.16: 00:80100370 0x3C018010 lui   $at,0x8010
9.20: 00:80100374 0xA0227C0C sb    $v0,0x7C0C($at)
9.20: st8 0x00:0x80107C0C <global_x> 0x11
fs2func1.c:13:      } else {
9.46: 00:80100378 0x10000005 b     0x80100390
10.12: 00:8010037C 0x00000000 nop
fs2func1.c:16:      return (global_x);
10.41: 00:80100390 0x3C028010 lui   $v0,0x8010
10.45: 00:80100394 0x90427C0C lbu   $v0,0x7C0C($v0)
10.45: ld8 0x00:0x80107C0C <global_x> 0x11
fs2func1.c:17: } // end func1()
    
```

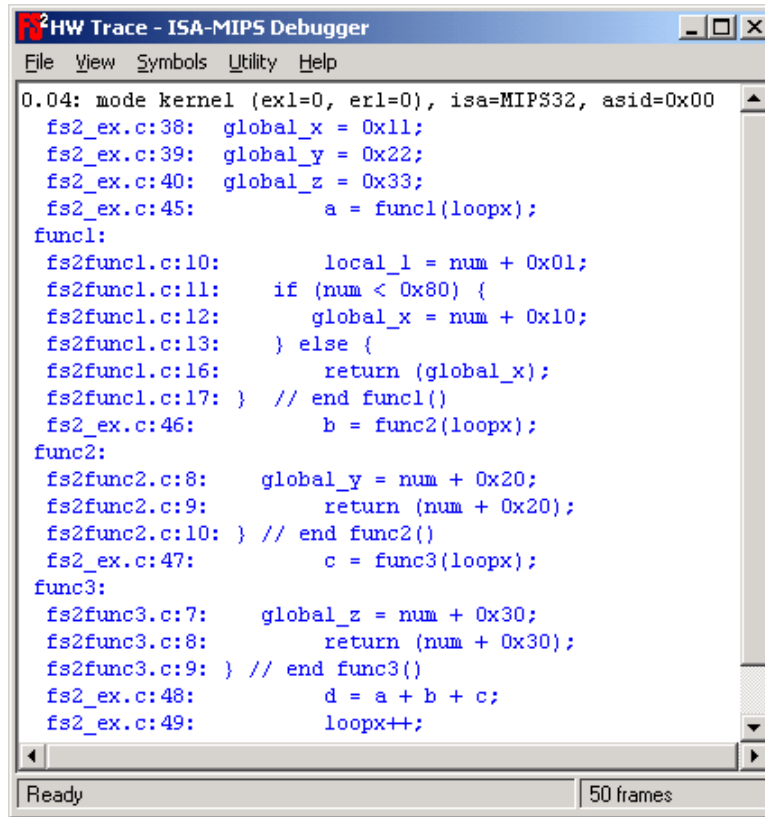
In the screen above, the first display line is “func1:”. Whenever an executed line exactly matches the entry to a function, that function name is inserted on a separate line.

Notice on frame number 10.45 load cycle (in green) that the name of the variable is “global_x” (surrounded by <>). The loaded symbols also include variable names, and when a load/store address matches a symbol, the name is inserted.

There are no idle cycles in the above display. The “View->No idle cycles” option hides “notrace cycles” trace messages. These messages are generated if the cycle-accurate trace mode is enabled, i.e. “Cycle-accurate trace” is checked in the Trace Mode window (previous page) or the CPU has been stalled to prevent the FIFO from filling up, i.e. “Inhibit overflow” is checked.

5.6 Source-only view

The next screen capture has instruction disassembly turned off (check View > Dasm Option... > No instructions) as well as load/store cycles (check View > No data cycles). This view shows just the execution flow at the source code level.



The screenshot shows a window titled "HW Trace - ISA-MIPS Debugger" with a menu bar containing "File", "View", "Symbols", "Utility", and "Help". The main area displays source code with line numbers and file names. The code includes a main function and three sub-functions: func1, func2, and func3. The execution flow is shown as it moves through these functions. The status bar at the bottom indicates "Ready" and "50 frames".

```
0.04: mode kernel (exl=0, erl=0), isa=MIPS32, asid=0x00
fs2_ex.c:38: global_x = 0x11;
fs2_ex.c:39: global_y = 0x22;
fs2_ex.c:40: global_z = 0x33;
fs2_ex.c:45:     a = func1(loopx);
func1:
fs2func1.c:10:     local_l = num + 0x01;
fs2func1.c:11:     if (num < 0x80) {
fs2func1.c:12:         global_x = num + 0x10;
fs2func1.c:13:     } else {
fs2func1.c:16:         return (global_x);
fs2func1.c:17: } // end func1()
fs2_ex.c:46:     b = func2(loopx);
func2:
fs2func2.c:8:     global_y = num + 0x20;
fs2func2.c:9:     return (num + 0x20);
fs2func2.c:10: } // end func2()
fs2_ex.c:47:     c = func3(loopx);
func3:
fs2func3.c:7:     global_z = num + 0x30;
fs2func3.c:8:     return (num + 0x30);
fs2func3.c:9: } // end func3()
fs2_ex.c:48:     d = a + b + c;
fs2_ex.c:49:     loopx++;
```

5.7 Including load/store cycles

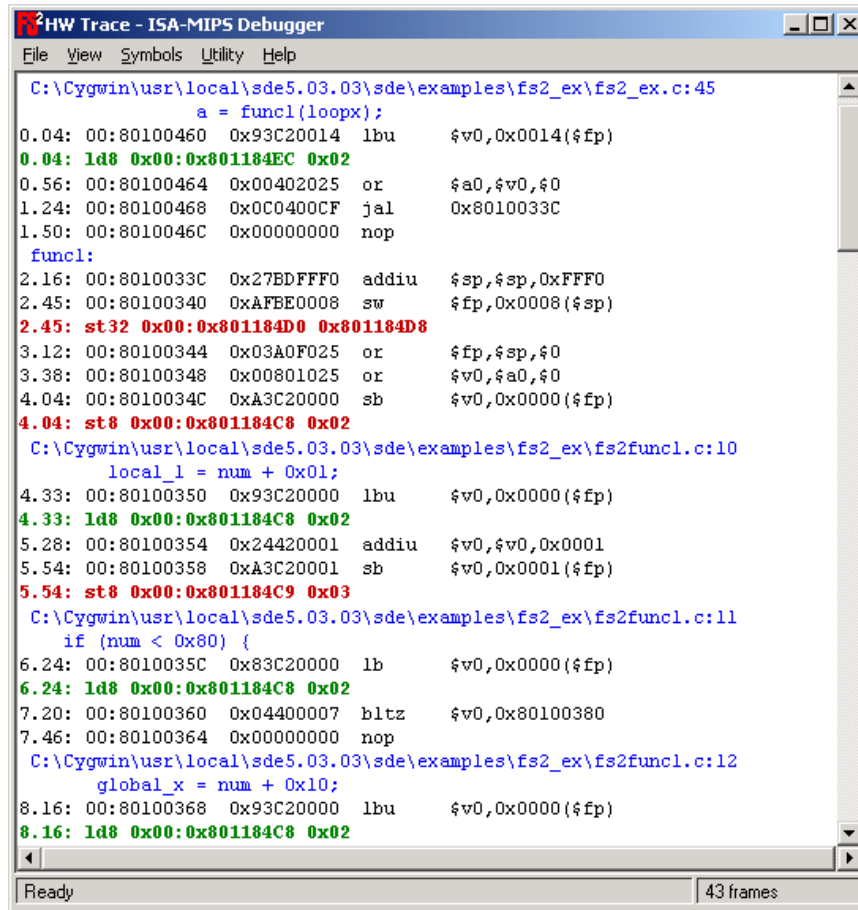
The next view has the load/store cycles turned on (uncheck View > No data cycles). This is useful to see the code and data flow as the code executed, and what conditional code and variable values caused it. For example, the trace includes line 11 “if (num < 0x80) {” and the next trace frame (6.24) shows a ld8 value of 0x02 indicating that the “if” clause is true. The next source line is 12 meaning the code executed into the “if” clause.

```

HW Trace - ISA-MIPS Debugger
File View Symbols Utility Help
fs2_ex.c:45:      a = func1(loopx);
0.04: ld8 0x00:0x801184EC 0x02
func1:
2.45: st32 0x00:0x801184D0 0x801184D8
4.04: st8 0x00:0x801184C8 0x02
fs2func1.c:10:   local_l = num + 0x01;
4.33: ld8 0x00:0x801184C8 0x02
5.54: st8 0x00:0x801184C9 0x03
fs2func1.c:11:   if (num < 0x80) {
6.24: ld8 0x00:0x801184C8 0x02
fs2func1.c:12:   global_x = num + 0x10;
8.16: ld8 0x00:0x801184C8 0x02
9.20: st8 0x00:0x80107C0C <global_x> 0x12
fs2func1.c:13:   } else {
fs2func1.c:16:   return (global_x);
10.45: ld8 0x00:0x80107C0C <global_x> 0x12
fs2func1.c:17: } // end func1()
11.38: ld32 0x00:0x801184D0 0x801184D8
13.28: st8 0x00:0x801184E8 0x12
fs2_ex.c:46:      b = func2(loopx);
13.57: ld8 0x00:0x801184EC 0x02
func2:
16.41: st32 0x00:0x801184D0 0x801184D8
17.63: st8 0x00:0x801184C8 0x02
fs2func2.c:8:    global_y = num + 0x20;
18.32: ld8 0x00:0x801184C8 0x02
19.61: st16 0x00:0x80107C04 <global_y> 0x0022
fs2func2.c:9:    return (num + 0x20);
20.28: ld8 0x00:0x801184C8 0x02
fs2func2.c:10: } // end func2()
22.24: ld32 0x00:0x801184D0 0x801184D8
Ready 43 frames
    
```

5.8 Full path for source lines

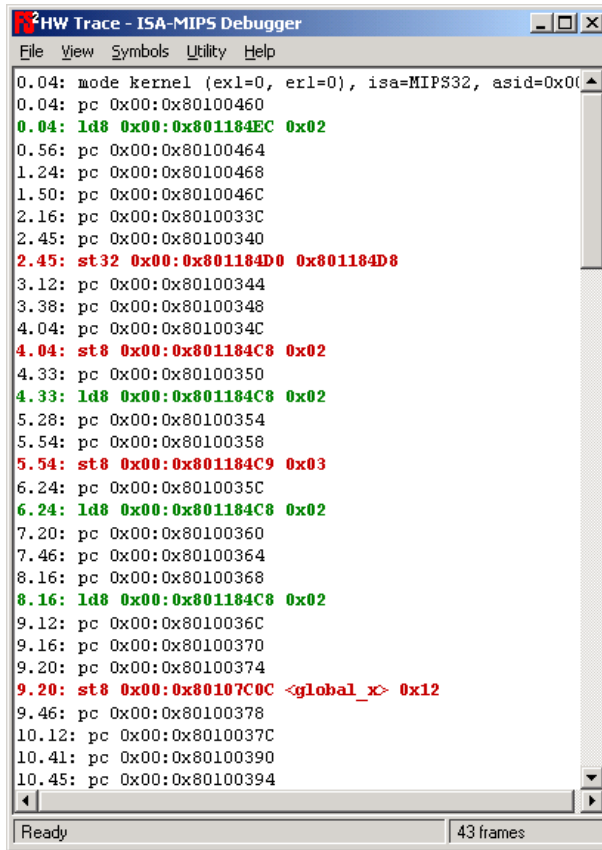
This screen turns on instructions, load/stores, and full path and filenames (uncheck View > Dasm options... > Short file paths). This is useful for knowing where the actual source file is located, or to distinguish which source file executed if there were duplicate filenames in different directories.



```
HW Trace - ISA-MIPS Debugger
File View Symbols Utility Help
C:\Cygwin\usr\local\sde5.03.03\sde\examples\fs2_ex\fs2_ex.c:45
    a = func1(loopx);
0.04: 00:80100460 0x93C20014 lbu    $v0,0x0014($fp)
0.04: ld8 0x00:0x801184EC 0x02
0.56: 00:80100464 0x00402025 or     $a0,$v0,$0
1.24: 00:80100468 0x0C0400CF jal   0x8010033C
1.50: 00:8010046C 0x00000000 nop
func1:
2.16: 00:8010033C 0x27BDFFF0 addiu  $sp,$sp,0xFFFF0
2.45: 00:80100340 0xAFBE0008 sw    $fp,0x0008($sp)
2.45: st32 0x00:0x801184D0 0x801184D8
3.12: 00:80100344 0x03A0F025 or     $fp,$sp,$0
3.38: 00:80100348 0x00801025 or     $v0,$a0,$0
4.04: 00:8010034C 0xA3C20000 sb    $v0,0x0000($fp)
4.04: st8 0x00:0x801184C8 0x02
C:\Cygwin\usr\local\sde5.03.03\sde\examples\fs2_ex\fs2func1.c:10
    local_l = num + 0x01;
4.33: 00:80100350 0x93C20000 lbu    $v0,0x0000($fp)
4.33: ld8 0x00:0x801184C8 0x02
5.28: 00:80100354 0x24420001 addiu  $v0,$v0,0x0001
5.54: 00:80100358 0xA3C20001 sb    $v0,0x0001($fp)
5.54: st8 0x00:0x801184C9 0x03
C:\Cygwin\usr\local\sde5.03.03\sde\examples\fs2_ex\fs2func1.c:11
    if (num < 0x80) {
6.24: 00:8010035C 0x83C20000 lb     $v0,0x0000($fp)
6.24: ld8 0x00:0x801184C8 0x02
7.20: 00:80100360 0x04400007 bltz   $v0,0x80100380
7.46: 00:80100364 0x00000000 nop
C:\Cygwin\usr\local\sde5.03.03\sde\examples\fs2_ex\fs2func1.c:12
    global_x = num + 0x10;
8.16: 00:80100368 0x93C20000 lbu    $v0,0x0000($fp)
8.16: ld8 0x00:0x801184C8 0x02
Ready 43 frames
```

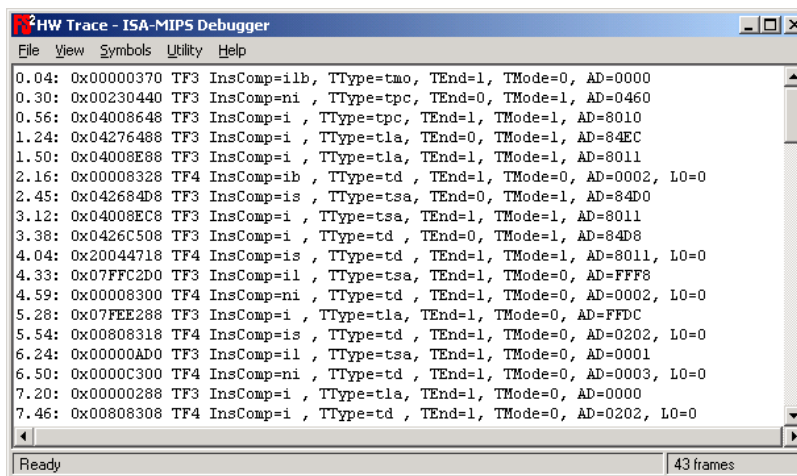
5.9 MDI view

This view shows the pc values where the executed branches took place. It does not include disassembly between these branches or source line insertions.



5.10 Trace Message view

Trace Message view displays the trace frames in a parsed format annotated with the labels defined in the TCB specification.



5.11 Turning Trace on and off

For the next example, switch back to Dasm view and turn off viewing of disassembly and load/store cycles.

Each of the hardware triggers can turn the trace on or off. In the following example, the trace is turned on at the entry to function func1() and turned off at the exit of the same function, thus tracing only the execution in this function. The trigger window requires entering the code address in hex or decimal. To determine the start and end address of func1(), bring up the function in the Source window. Then switch to Mixed mode to show both source and assembly. Select func1() from the function list menu, the second pull-down list from the left, at the bottom of the window.

```

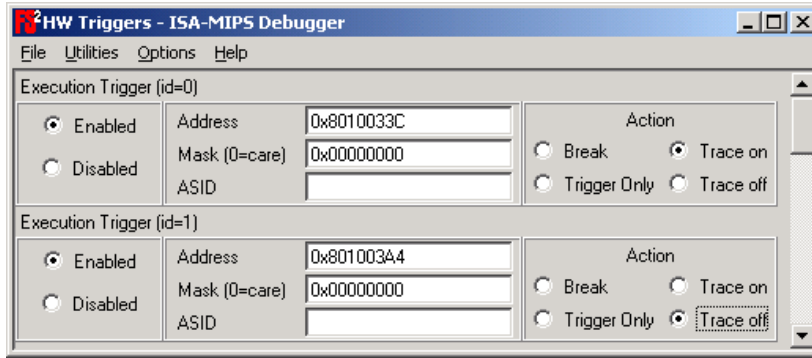
7  unsigned char func1 (unsigned char num) {
- 0x8010033c <Func1>:      addiu   $sp,$sp,-16
- 0x80100340 <Func1+4>:    sw      $s8,8($sp)
- 0x80100344 <Func1+8>:    move   $s8,$sp
- 0x80100348 <Func1+12>:   move   $v0,$a0
- 0x8010034c <Func1+16>:   sb      $v0,0($s8)
8
9      unsigned char local_1;
10     local_1 = num + 0x01;
- 0x80100350 <Func1+20>:   lbu    $v0,0($s8)
- 0x80100354 <Func1+24>:   addiu  $v0,$v0,1
- 0x80100358 <Func1+28>:   sb      $v0,1($s8)
11     if (num < 0x80) {
- 0x8010035c <Func1+32>:   lb      $v0,0($s8)
- 0x80100360 <Func1+36>:   bltz   $v0,0x80100380 <Func1+68>
- 0x80100364 <Func1+40>:   nop
12     global_x = num + 0x10;
- 0x80100368 <Func1+44>:   lbu    $v0,0($s8)
- 0x8010036c <Func1+48>:   addiu  $v0,$v0,16
- 0x80100370 <Func1+52>:   lui    $at,0x8010
- 0x80100374 <Func1+56>:   sb      $v0,31756($at)
13     } else {
- 0x80100378 <Func1+60>:   b      0x80100390 <Func1+84>
- 0x8010037c <Func1+64>:   nop
14     global_x = num - 0x10;
- 0x80100380 <Func1+68>:   lbu    $v0,0($s8)
- 0x80100384 <Func1+72>:   addiu  $v0,$v0,-16
- 0x80100388 <Func1+76>:   lui    $at,0x8010
- 0x8010038c <Func1+80>:   sb      $v0,31756($at)
15     }
16     return (global_x);
- 0x80100390 <Func1+84>:   lui    $v0,0x8010
- 0x80100394 <Func1+88>:   lbu    $v0,31756($v0)
- 0x80100398 <Func1+92>:   move   $sp,$s8
- 0x8010039c <Func1+96>:   lw      $s8,8($sp)
- 0x801003a0 <Func1+100>:  addiu  $sp,$sp,16
- 0x801003a4 <Func1+104>:  jr      $ra
- 0x801003a8 <Func1+108>:  nop
    
```

Program stopped at line 10, 0x80100350

Fs2func1.c func1 MIXED

As shown below, the function entry address (0x8010033C) is entered into the Execution Trigger (id=0), the Mask set to 0 (cares), and the Action set to "Trace on". The function exit address (0x801003A4) is entered into Execution Trigger (id=1) with the Action set to "Trace off".

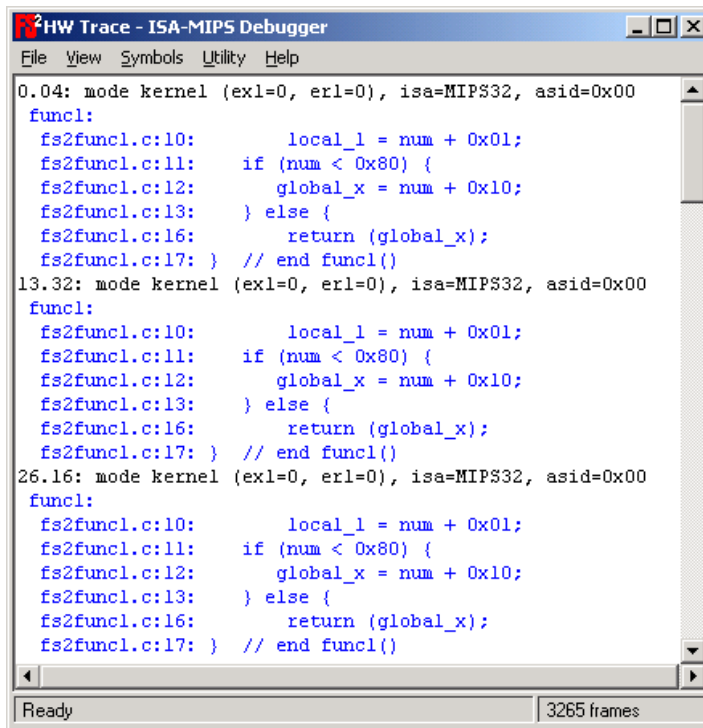
FS2 Trigger-Trace Windows for GDB/Insight for MIPS: User Guide



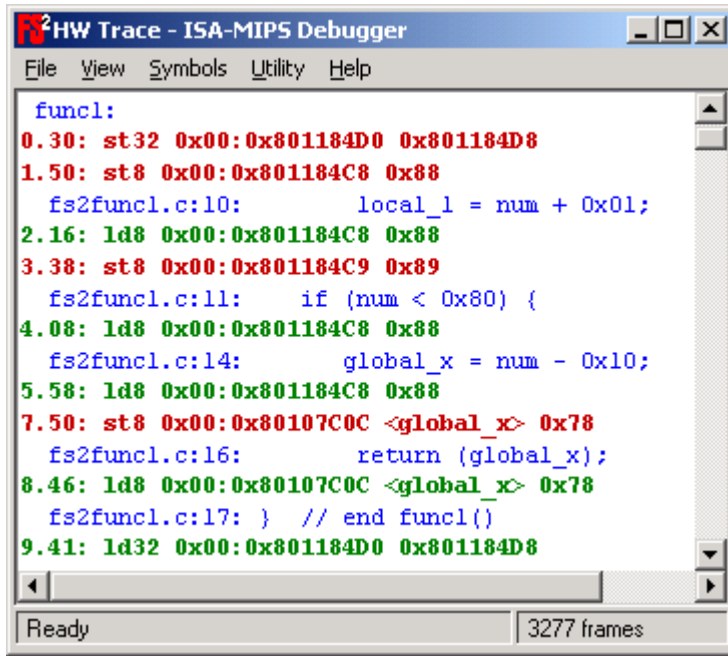
In order for trace on-off to work, the initial state of trace needs to be turned off. This is done by clicking the “Turn off trace initially” button at the bottom of the Trace Mode window which unchecks “Trace in exceptions and error modes”, “Trace in kernel mode”, and “Trace in user mode”.

Before starting the program again with the “Continue” icon, remove the software breakpoint on line 50 in fs2_ex.c (function main()). Start the program then stop it manually with the far left “Stop” icon.

The resulting trace is the screen capture below which shows three executions of func1() and no other code in the trace:



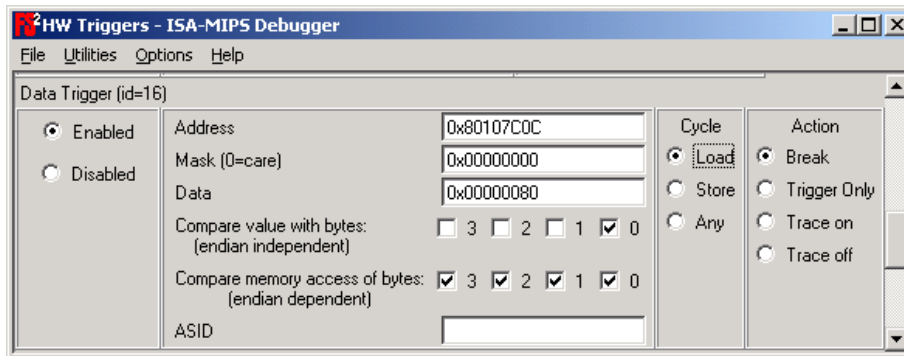
Turn on load/store cycles. The following screen shows two executions of func1() including the variable accesses. Notice that the “if” condition of source line 11 is false because the variable “num” has a value of 0x88 (frame 4.08) so the else portion of the if-else is executed (source line 14).



5.12 Data Triggers

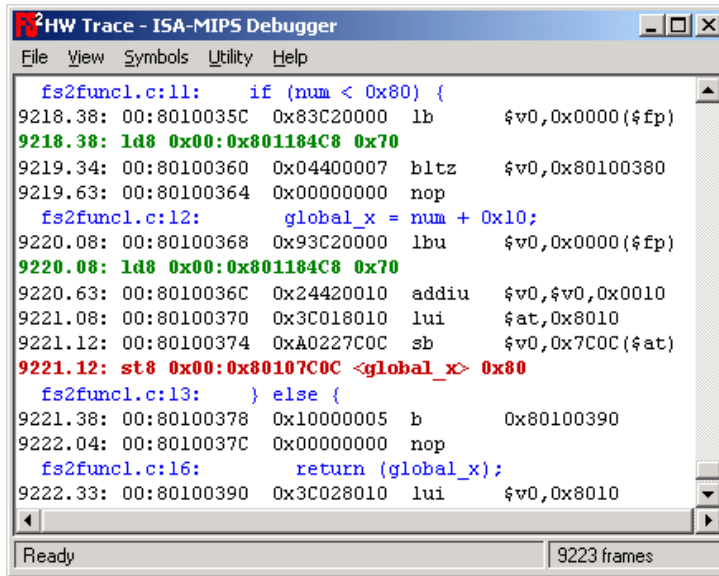
Data triggers (id >= 16) provide triggering on the address of a variable, bit-wise mask of the address, data value compare, byte-lane masking of the data value, byte-lane comparison of memory access, optional ASID comparison, and load, store, or any cycle type access. This section illustrates how to set up and trigger on a load of data value 0x80 to the "global_x" variable.

Look in the previous trace and extract the address of "global_x", which is 0x80107C0C. Enter it in the Address field, then set the mask to all cares (0). Set the Data field to 0x80 then enable (check) the 0 byte lane (least significant byte). Set Cycle type to "Load" and Action to "Break". The Data Trigger should look like the screen below:



Start execution. The program stops when global_x is loaded as a return value in fs2func1.c (line 16).

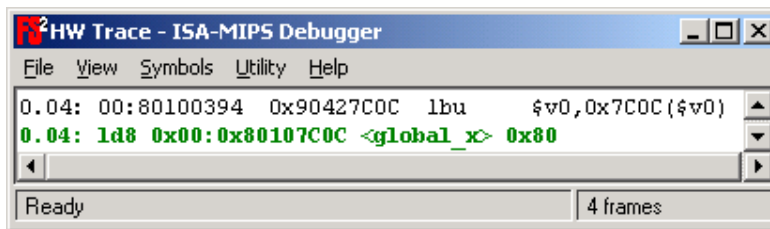
FS2 Trigger-Trace Windows for GDB/Insight for MIPS: User Guide



```
fs2func1.c:11:  if (num < 0x80) {
9218.38: 00:8010035C 0x83C20000 lb      $v0,0x0000($fp)
9218.38: ld8 0x00:0x801184C8 0x70
9219.34: 00:80100360 0x04400007 bltz   $v0,0x80100380
9219.63: 00:80100364 0x00000000 nop
fs2func1.c:12:  global_x = num + 0x10;
9220.08: 00:80100368 0x93C20000 lbu    $v0,0x0000($fp)
9220.08: ld8 0x00:0x801184C8 0x70
9220.63: 00:8010036C 0x24420010 addiu  $v0,$v0,0x0010
9221.08: 00:80100370 0x3C018010 lui    $at,0x8010
9221.12: 00:80100374 0xA0227C0C sb     $v0,0x7C0C($at)
9221.12: st8 0x00:0x80107C0C <global_x> 0x80
fs2func1.c:13:  } else {
9221.38: 00:80100378 0x10000005 b      0x80100390
9222.04: 00:8010037C 0x00000000 nop
fs2func1.c:16:  return (global_x);
9222.33: 00:80100390 0x3C028010 lui    $v0,0x8010
```

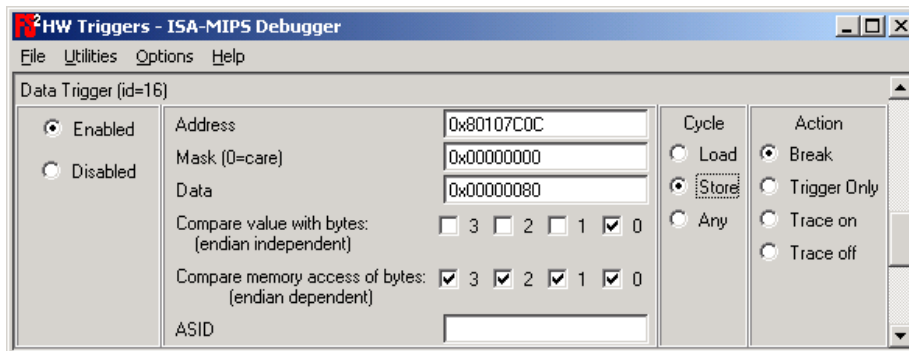
An important point here is that the trace won't have the actual load 0x80 cycle in it because the on-chip Data Trigger stops the processor **before** it completes the write.

When the processor is single stepped - which is required to get past this hardware trigger - the trace now contains the global_x load cycle with the value of 0x80:



```
0.04: 00:80100394 0x90427C0C lbu    $v0,0x7C0C($v0)
0.04: ld8 0x00:0x80107C0C <global_x> 0x80
```

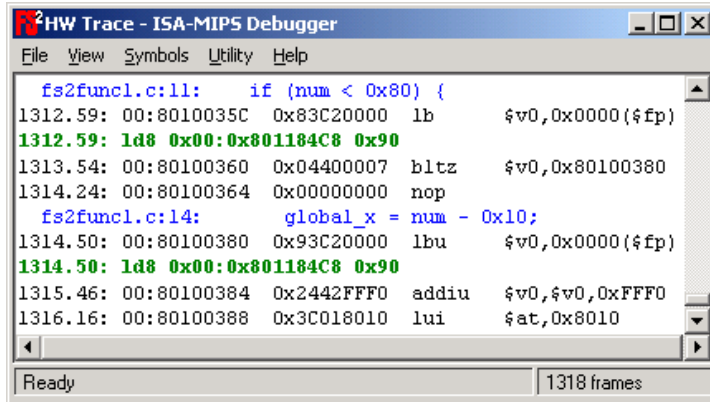
Now change the Data Trigger to Break the processor when global_x is **written** with a value of 0x80:



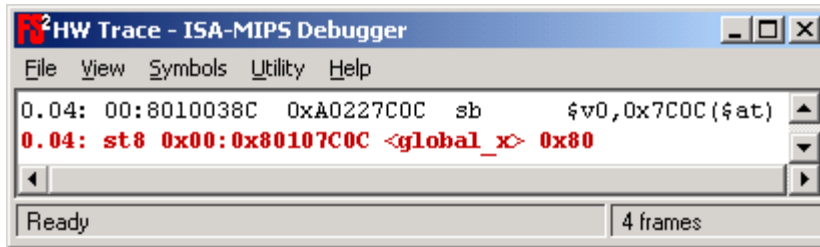
Data Trigger (id=16)			
<input checked="" type="radio"/> Enabled	Address	0x80107C0C	Cycle
<input type="radio"/> Disabled	Mask (0=care)	0x00000000	<input type="radio"/> Load
	Data	0x00000080	<input checked="" type="radio"/> Store
	Compare value with bytes: (endian independent)	<input type="checkbox"/> 3 <input type="checkbox"/> 2 <input type="checkbox"/> 1 <input checked="" type="checkbox"/> 0	<input type="radio"/> Any
	Compare memory access of bytes: (endian dependent)	<input checked="" type="checkbox"/> 3 <input checked="" type="checkbox"/> 2 <input checked="" type="checkbox"/> 1 <input checked="" type="checkbox"/> 0	<input type="radio"/> Break
	ASID		<input type="radio"/> Trigger Only
			<input type="radio"/> Trace on
			<input type="radio"/> Trace off

FS2 Trigger-Trace Windows for GDB/Insight for MIPS: User Guide

The code stops on line 14 of func1() when the processor attempts to write to global_x with the value of 0x80:



Again, the processor hasn't completed the write cycle until the debugger is stepped one source line. The resulting trace shows the store to global_x:

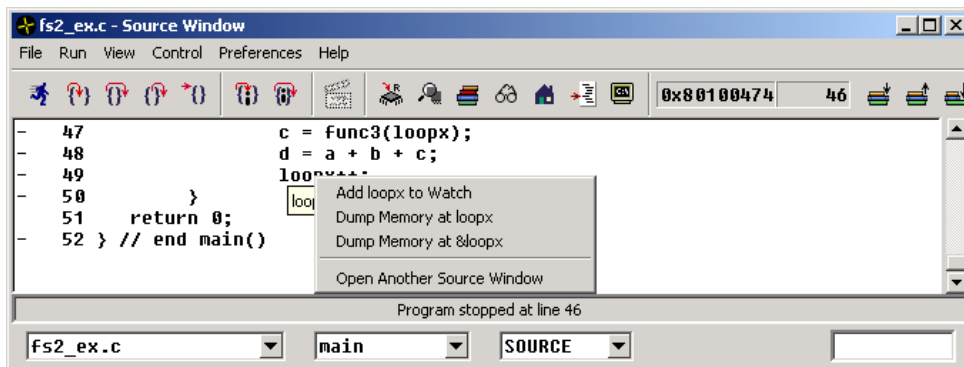


global_x was an 8 bit variable. Now we will set a data breakpoint on a 32 bit variable, namely "loopx" in main(). First, disable the previous data trigger (id=16). Note that you can re-enable it and the values are returned. (However when you close the window or close down Insight, the trigger values are not retained).

To get the address of loopx, you can use the console:

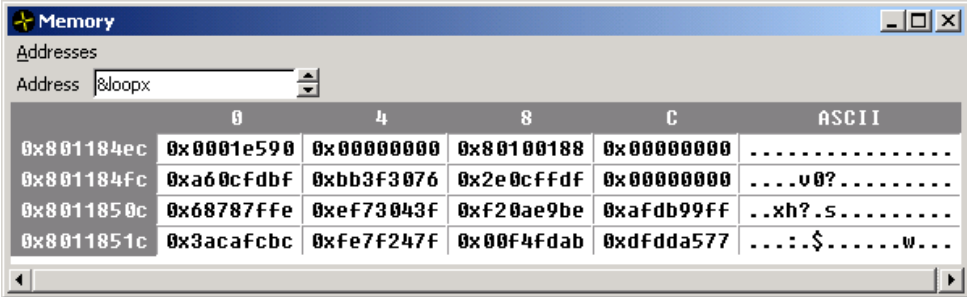
```
(gdb) print &loopx
$1 = (long unsigned int *) 0x801184ec
```

or use the right-click pop-up menu and dump memory at &loopx:

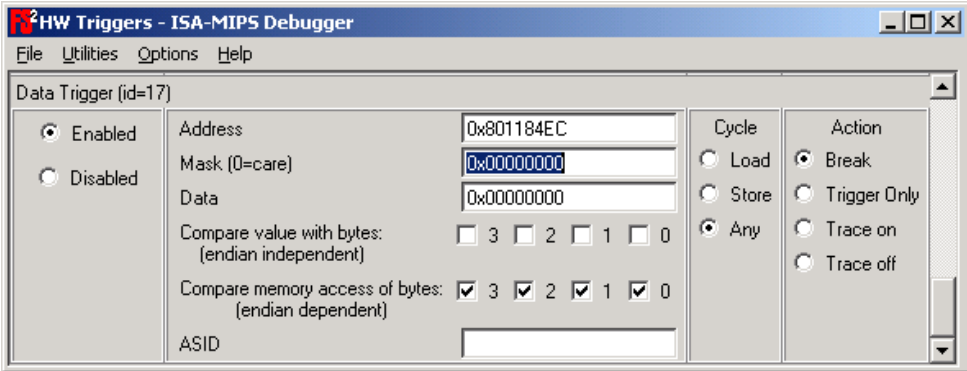


FS2 Trigger-Trace Windows for GDB/Insight for MIPS: User Guide

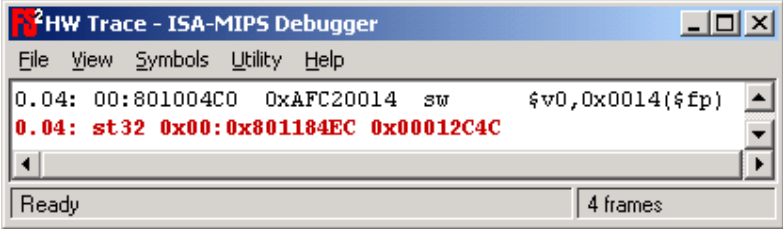
The Memory window shows the address of loopx to be the hex value on the first line (0x801184ec):



Now enter the address into the Data Trigger (id=17) Address field to trigger on “Any cycle” (load or store) to loopx, regardless of the data value; i.e. “Compare value with bytes” are all unchecked:

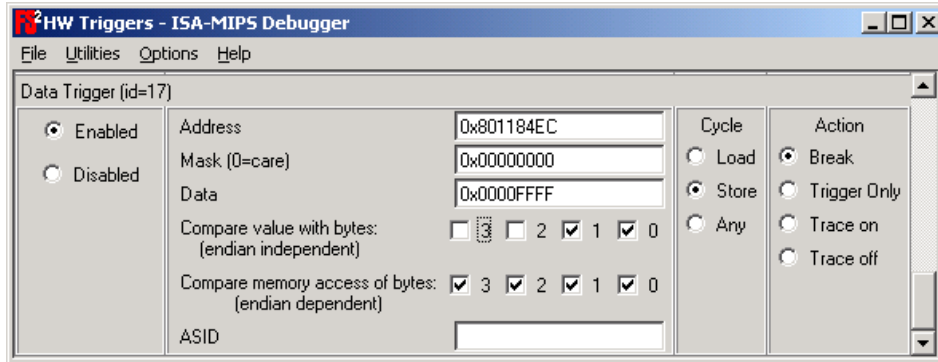


Start execution. Below is an example of a write of the value 0x00012C4C to loopx:

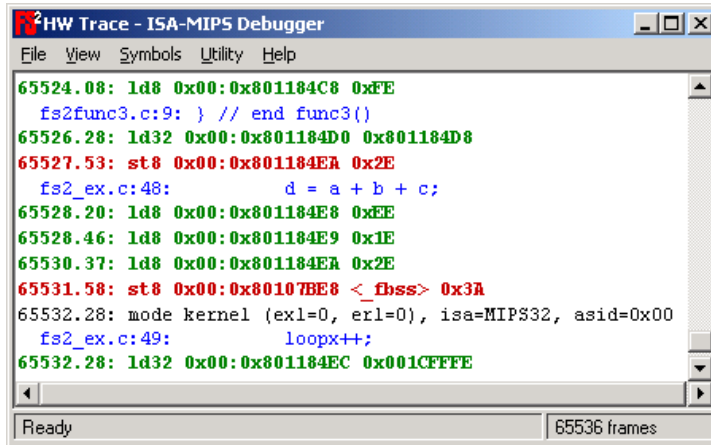


FS2 Trigger-Trace Windows for GDB/Insight for MIPS: User Guide

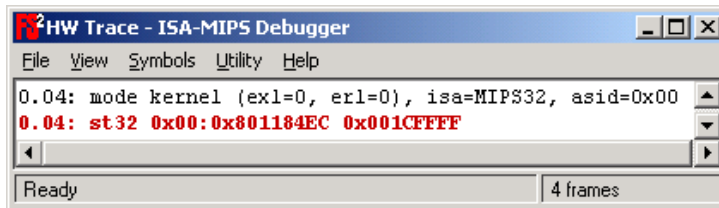
To make a more narrowly defined trigger pattern, the following example sets up the trigger on loopx when a write (store) value of all ones (0xFFFF) occurs on the lower 16 bits (and don't cares on the upper 2 bytes):



Below shows the 32 bit load value (0x001CFFFE) just prior to the store:



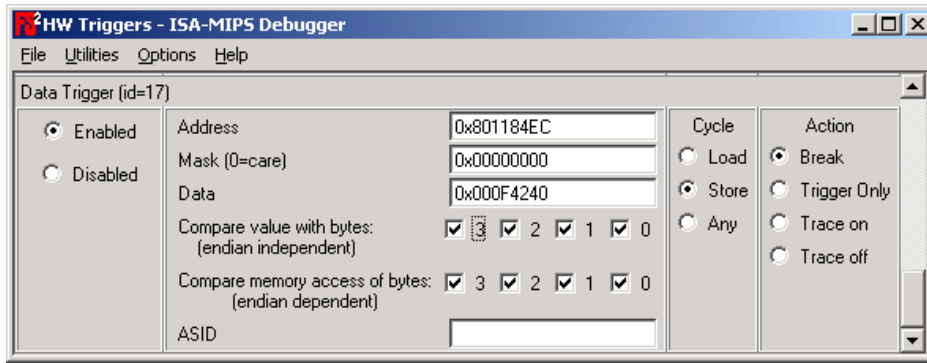
Single step and the trace captures the write to loopx with a value of 0xFFFF on the lower two bytes:



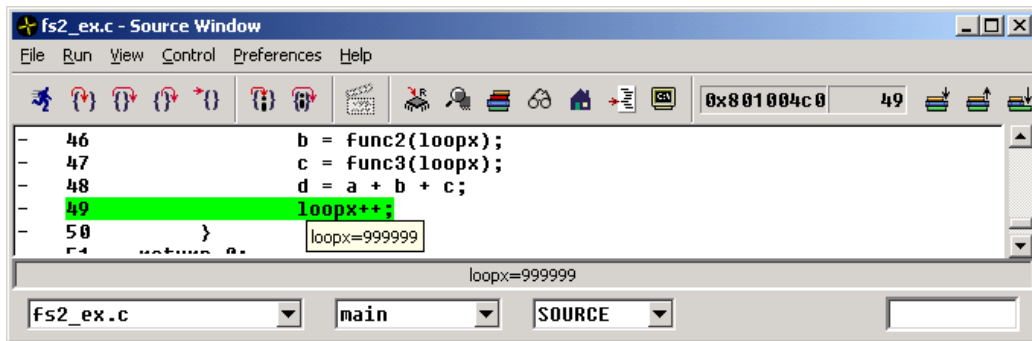
5.13 Example of measuring the duration of the program loop

The ability to trigger on a specific 32 bit value in loopx provides a means of measuring the average time of the main() while loop. First, set loopx to 0. This can be done by editing the value in the memory window. Next, set the Data Trigger Data value to 1000000 by entering this decimal value in the Data field. When leaving the field with a Tab key, it is converted to the hex equivalent of 1,000,000 (0x000F4240). Check all four "Compare value with bytes:" which means all 4 bytes are included in the value comparison.

FS2 Trigger-Trace Windows for GDB/Insight for MIPS: User Guide



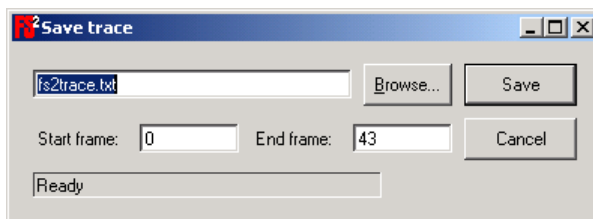
Click the run icon and start a stop watch simultaneously. When the program breaks and halts, stop the stopwatch. Put the mouse pointer over the loopx variable. The “value tip” of 999999 indicates that the program broke when loopx was about to be written with a value of 1,000,000:



Reading out the stopwatch, the result is 11.9 seconds. Divided by 1,000,000 results in a per-loop time of 11.9 μ s (microseconds).

5.14 Saving the trace to a file

The File > Save as... provides a dialog to save all or part of the Trace window formatted data to a file, whatever the current display mode is.



This can be useful to go back with a text editor and search for specific execution or data value occurrences. It could also be grep'd to find specific lines of interest.

6 Conclusion

The FS2 HW Trigger window provides easy setup of hardware execution triggers and hardware data address-value triggers. The HW Trace window is also an easy-to-use scrolling window that automatically updates with the latest trace data when the processor breaks execution. It implements source line insertions for matching function and line number addresses and variable name insertions for load/store addresses that match a symbols generated from the gcc toolchain.

FS2 Trigger-Trace Windows for GDB/Insight for MIPS: User Guide

The Trace Mode window makes it easy to change the trace data types included in the trace buffer and hardware filters that remove specific trace data from the buffer.

Be sure to access the on-line help for further details on operating these window menus and the programmable fields in them. Each of the three windows has a Help > <window name> link to the on-line help text.