Multi Processor SoC Debug with Verdi Hardware Software Debug Tool

A. Mark Jesensky
Andy Sha
Analog Devices, Inc

March 23, 2015
SNUG Silicon Valley

Agenda

Brief review of SoC debug methods – Traditional method

Brief review of SoC debug methods – Verdi HW/SW Debug

Multicore implementation

Debug cases

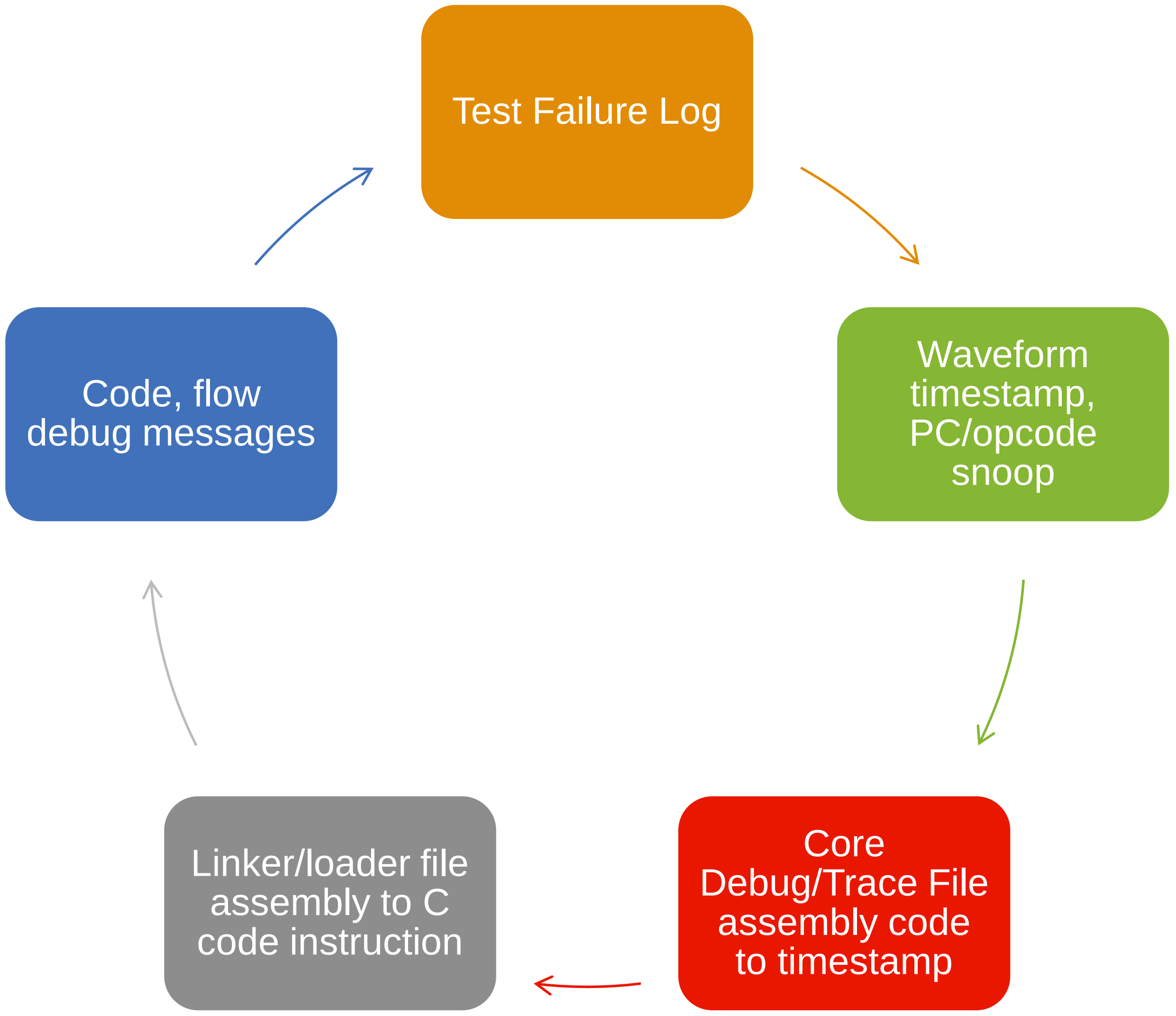
Feature requests

Conclusions

Brief Review of Debug Methods

Traditional Debug

SoC Traditional Code Debug


Test Failure Log
Waveform timestamp, PC/opcode snoop
Core Debug/Trace File assembly code to timestamp
Linker/loader file assembly to C code instruction
Code, flow debug messages

SoC Debug Files
Single Core

Run log:

Timestamps, debug messages

.elf/.out File:

C code, assembled core code, memory segments

Waveform file:

Internal signal waves, timestamps

Debug/Trace File :

Core register and pipeline information, with timestamps

SoC Debug Files

Multi Core

Run log:

Timestamps, debug messages

.elf/.out File n:

ed

ory

.elf/.out File2:

ed

ry

.elf/.out File1:

C code, assembled core code, memory segments

Waveform file:

Internal signal waves, timestamps

Debug/Trace File

Co

pi

wi

Debug/Trace File

C

pi

Debug/Trace File n

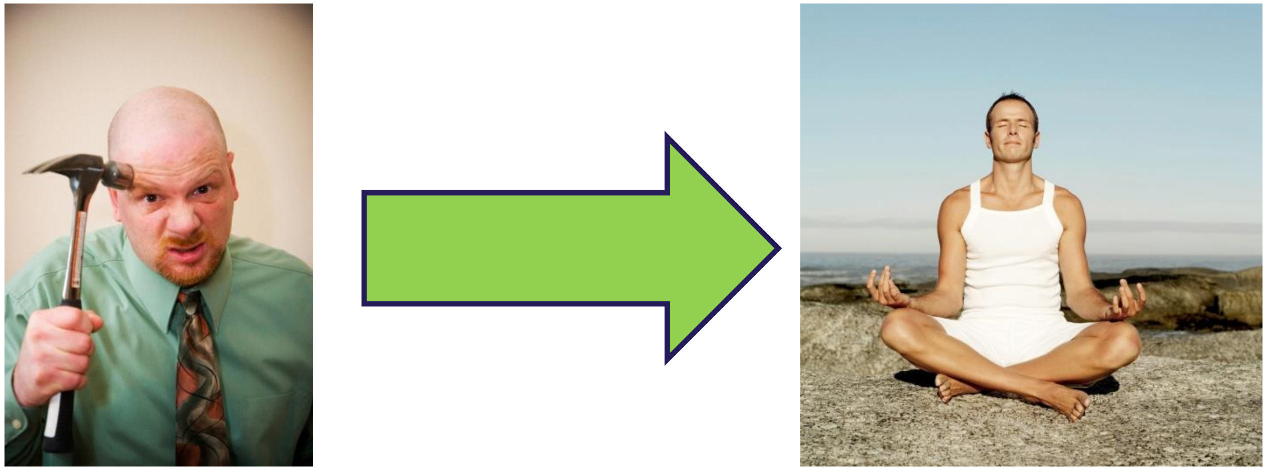
Core register and pipeline information, with timestamps

More Cores, More challenges

- SoCs have become a blend of reuse blocks, IP black boxes, and custom fabrics often coming from unrelated groups
- Complexity drives longer debug times
  - Have you ever heard of a more complex design with a lower debug time?
- Finding the right person for debug
  - Get all verification engineers experts on all aspects
  - Have one expert firefighting problems

More cores, better solution

- Have the knowledgeable verification engineer do one setup of the project in a tool that multiple others can use
- Combine all of the debug files into one interface
- Automate as much of the cross referencing as possible
- More importantly, find a way to compress hours of test debug into minutes

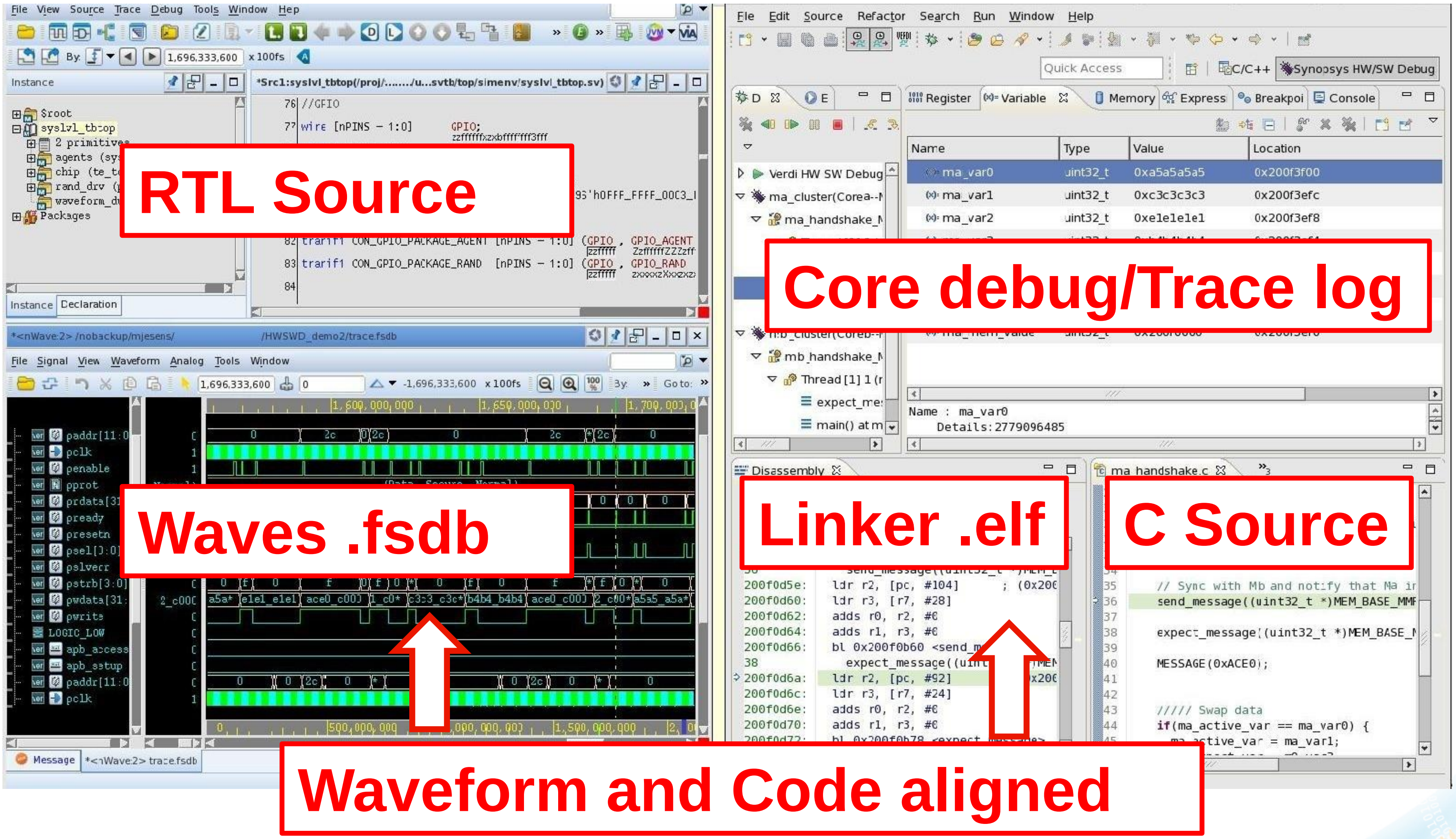


Brief Review of Debug Methods

Verdi HW/SW Debug Tool

Verdi HW/SW Debug Tool

Combines all of these files into linked GUIs



```
76 //GPIO
77 wire [nPINS - 1:0]        GPIO;
82 trarif1 CON_GPIO_PACKAGE_AGENT [nPINS - 1:0] (GPIO , GPIO_AGENT
83 trarif1 CON_GPIO_PACKAGE_RAND  [nPINS - 1:0] (GPIO , GPIO_RAND
```

Name	Type	Value	Location
ma_var0	uint32_t	0xa5a5a5a5	0x200f3f00
ma_var1	uint32_t	0xc3c3c3c3	0x200f3efc
ma_var2	uint32_t	0xe1e1e1e1	0x200f3ef8

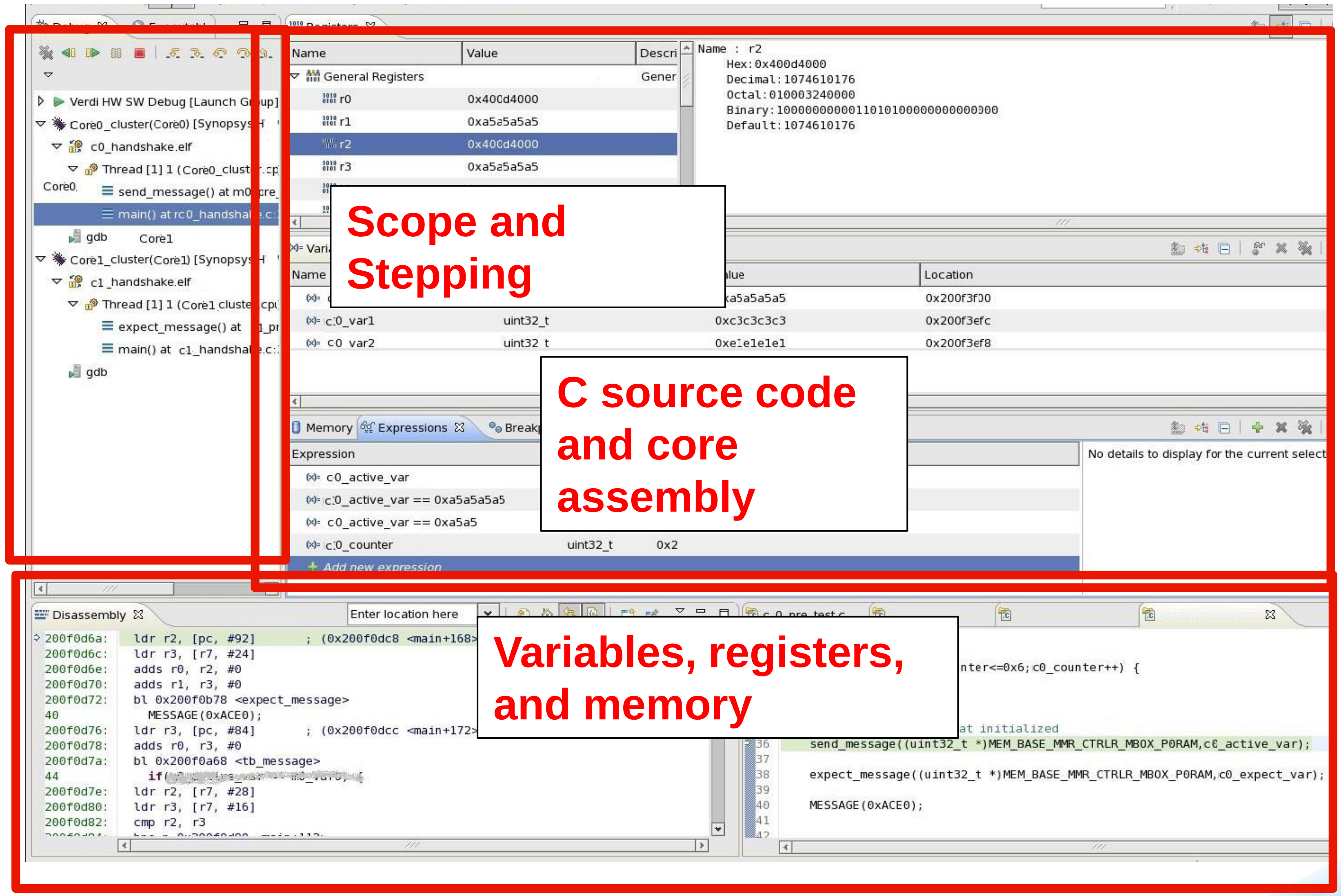
```
Name : ma_var0
  Details:2779096485
```

```
200f0d5e:   ldr r2, [pc, #104]      ; (0x200
200f0d60:   ldr r3, [r7, #28]
200f0d62:   adds r0, r2, #0
200f0d64:   adds r1, r3, #0
200f0d66:   bl 0x200f0b60 <send_m
38            expect_message((uint
200f0d6a:   ldr r2, [pc, #92]
200f0d6c:   ldr r3, [r7, #24]
200f0d6e:   adds r0, r2, #0
200f0d70:   adds r1, r3, #0
```

```c
    // Sync with Mb and notify that Ma i
    send_message((uint32_t *)MEM_BASE_MMF

    expect_message((uint32_t *)MEM_BASE_

    MESSAGE(0xACE0);


    ///// Swap data
    if(ma_active_var == ma_var0) {
      ma_active_var = ma_var1;
```

Verdi HW/SW Debug Tool GUI

Multi Core Debug in a single Eclipse GUI



Scope and Stepping

C source code and core assembly

Variables, registers, and memory

Name	Value	Descri
General Registers		Gener
r0	0x400d4000	
r1	0xa5a5a5a5	
r2	0x400d4000	
r3	0xa5a5a5a5	

```
Name : r2
    Hex:0x400d4000
    Decimal:1074610176
    Octal:010003240000
    Binary:1000000000011010100000000000000
    Default:1074610176
```

Name		lue	Location
		a5a5a5a5	0x200f3f00
c0_var1	uint32_t	0xc3c3c3c3	0x200f3efc
c0_var2	uint32_t	0xe1e1e1e1	0x200f3ef8

Expression		
c0_active_var		
c0_active_var == 0xa5a5a5a5		
c0_active_var == 0xa5a5		
c0_counter	uint32_t	0x2
Add new expression		

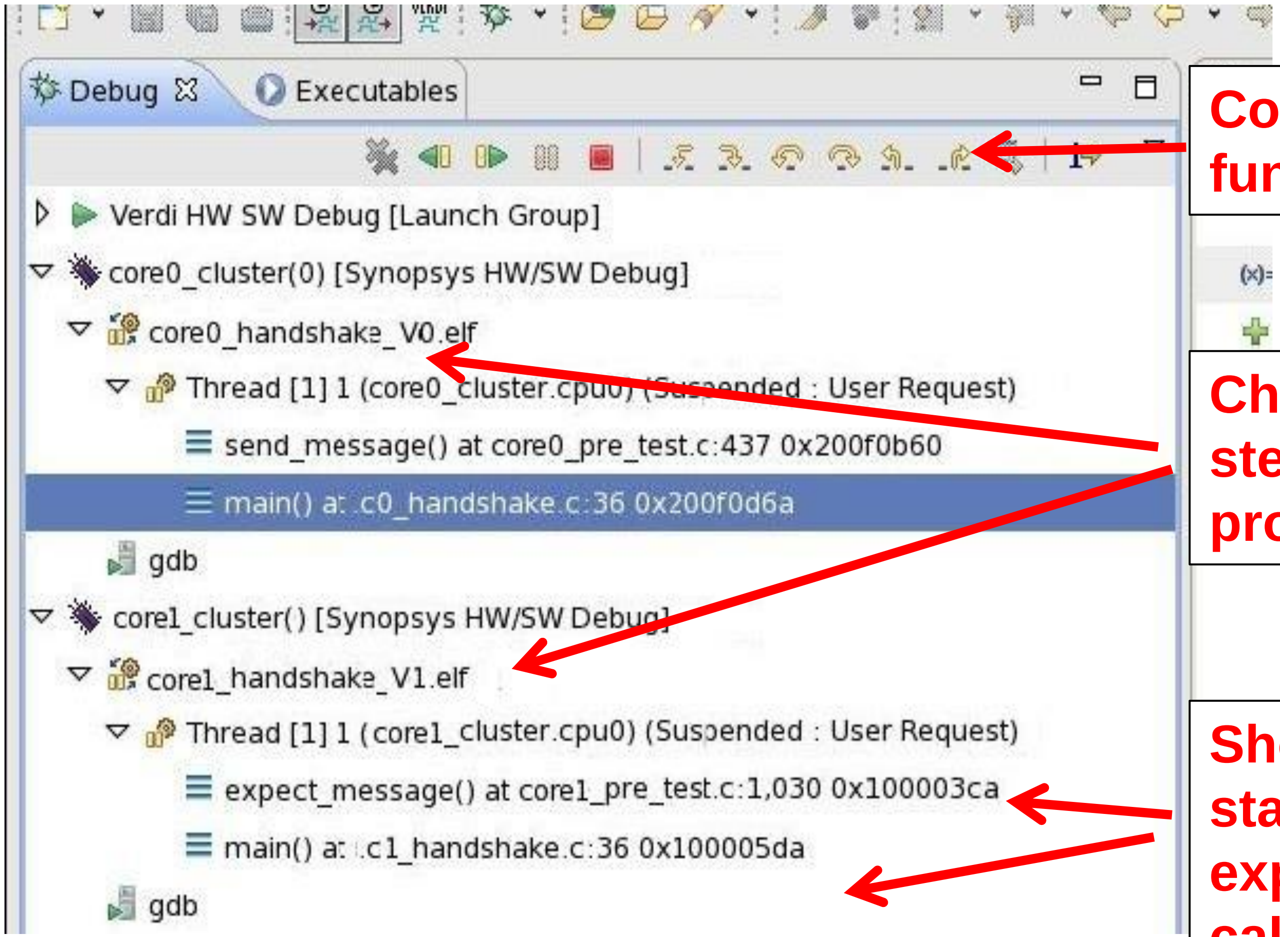
No details to display for the current select

```
200f0d6a:   ldr r2, [pc, #92]       ; (0x200f0dc8 <main+168>
200f0d6c:   ldr r3, [r7, #24]
200f0d6e:   adds r0, r2, #0
200f0d70:   adds r1, r3, #0
200f0d72:   bl 0x200f0b78 <expect_message>
40            MESSAGE(0xACE0);
200f0d76:   ldr r3, [pc, #84]       ; (0x200f0dcc <main+172>
200f0d78:   adds r0, r3, #0
200f0d7a:   bl 0x200f0a68 <tb_message>
44            if( [illegible]
200f0d7e:   ldr r2, [r7, #28]
200f0d80:   ldr r3, [r7, #16]
200f0d82:   cmp r2, r3
```

```
nter<=0x6;c0_counter++) {

at initialized
    send_message((uint32_t *)MEM_BASE_MMR_CTRLR_MBOX_P0RAM,c0_active_var);

    expect_message((uint32_t *)MEM_BASE_MMR_CTRLR_MBOX_P0RAM,c0_expect_var);

    MESSAGE(0xACE0);
```

Verdi HW/SW Debug Tool GUI

Scope and Stepping – choosing a processor



Verdi HW SW Debug [Launch Group]
core0_cluster(0) [Synopsys HW/SW Debug]
core0_handshake_V0.elf
Thread [1] 1 (core0_cluster.cpu0) (Suspended : User Request)
send_message() at core0_pre_test.c:437 0x200f0b60
main() at .c0_handshake.c:36 0x200f0d6a
gdb
core1_cluster() [Synopsys HW/SW Debug]
core1_handshake_V1.elf
Thread [1] 1 (core1_cluster.cpu0) (Suspended : User Request)
expect_message() at core1_pre_test.c:1,030 0x100003ca
main() at .c1_handshake.c:36 0x100005da
gdb

Code stepping functions

Choose scope of stepping (specific processor)

Show current call stack, such as: expect_message(), called from main()

Debugging Hardware or Software

Just like the movie *The Matrix*

Tackling hardware only:
- Hard to trace system setups
- No context of test
- Interrupt and branching difficulties

Tackling software only:
- Lose hardware conflict info
- System reaction latencies
- External accesses unknown

You're destiny is to fail

Debugging Hardware AND Software

The tool allows everyone to be a Neo:

- Step through the code execution, see hardware reactions
- Click hardware events and see the code that got you there
- Go back to trace cause and effect in code or logic

Multicore Implementation

Implementation Overview

- .fsdb waveforms
  - Waveform data
  - References RTL source files
- Linker files
  - compiled assembly, memory segments
  - C source code
- Trace logs
  - Core internal states, registers
- Recorder Module
  - System Verilog in the testbench top level
  - References all files into the Verdi HW/SW interfaces

Waveform, Trace Logs, Linker files

- Standard debug files commonly generated
- .fsdb waveforms
  - Dumpvars with “+all” “+parameter” switches
- Linker files
  - With debug info and source file paths
- Trace logs:
  - If ARM core, trace file code in dev. kit
  - If custom core, requires custom trace logs

Recorder Module

- What is the Recorder Module ?
  - System Verilog module to process the files and execution
  - Placed into the testbench top level
- Synopsys recorder modules exist for many 3rd party IP cores
  - Up and running in a day or two
- Custom core recorder modules can be developed with support from Synopsys
  - Up and running….depends on core/complexity

Recorder Module

```
`include "verdiRecorderCore0.svp"
`include "verdiRecorderCore1.svp‟

module verdiHwswDebugTop;


Core0_cluster Core0_cluster();
Core1_cluster Core1_cluster();


    // FSDB dumper: change FSDB file name if needed.
    initial #0 $fsdbDumpvars(0,"verdiHwswDebugTop","+all","+parameter");
`VERDI_HWSW_INIT_TOP
endmodule // verdiHwswDebugTop
```

16 lines of code !

```
module Core0_cluster();
    verdiRecorderCore0 #(.clusterId(0), .cpuId(0), .traceFileName(“dfile_Core0.log"))
    cpu0(.cpuClock(1'b0));
endmodule
module Core1_cluster();
    verdiRecorderCore1 #(.clusterId(1), .cpuId(0), .traceFileName(“dfile_Core1.log"))
    cpu0(.cpuClock(1'b0));
endmodule
```

Recorder Module

```
`include "verdiRecorderCore0.svp"
`include "verdiRecorderCore1.svp"
```

Include one recorder per core type

```
module verdiHwswDebugTop;

Core0_cluster Core0_cluster();
Core1_cluster Core1_cluster();

    // FSDB dumper: change FSDB file name if needed.
    initial #0
   $fsdbDumpvars(0,"verdiHwswDebugTop","+all","+parameter");
`VERDI_HWSW_INIT_TOP
endmodule // verdiHwswDebugTop
```

Recorder Module

```
`include "verdiRecorderCore0.svp"
`include "verdiRecorderCore1.svp“

module verdiHwswDebugTop;

Core0_cluster Core0_cluster();
Core1_cluster Core1_cluster();

    // FSDB dumper: change FSDB file name if needed.
    initial #0
   $fsdbDumpvars(0,"verdiHwswDebugTop","+all","+parameter");
`VERDI_HWSW_INIT_TOP
endmodule // verdiHwswDebugTop
```

Instantiate one per core in design

Recorder Module

```
`include "verdiRecorderCore0.svp"
`include "verdiRecorderCore1.svp‟

module verdiHwswDebugTop;


Core0_cluster Core0_cluster();
Core1_cluster Core1_cluster();

    // FSDB dumper: change FSDB file name if needed.
    initial #0
   $fsdbDumpvars(0,"verdiHwswDebugTop","+all","+parameter");
VERDI_HWSW_INIT_TOP

endmodule // verdiHwswDebugTop
```

Waves the same as single core

Recorder Module

Define modules for each core

```
module Core0_cluster();
    verdiRecorderCore0 #(.clusterId(0), .cpuId(0),
   .traceFileName("dfile_Core0.log")) cpu0(.cpuClock(1'b0));
endmodule
```

```
module Core1_cluster();
    verdiRecorderCore1 #(.clusterId(1), .cpuId(0),
   .traceFileName("dfile_Core1.log")) cpu0(.cpuClock(1'b0));
endmodule
```

Further Multicore Implementation

```
`include "verdiRecorderCore0.svp"
`include "verdiRecorderCore1.svp‟

module verdiHwswDebugTop;

Core0_cluster Core0_cluster();
Core1_cluster Core1_cluster();

    // FSDB dumper: change FSDB file name if needed.
    initial #0 $fsdbDumpvars(0,"verdiHwswDebugTop","+all","+parameter");
`VERDI_HWSW_INIT_TOP
endmodule // verdiHwswDebugTop

module Core0_cluster();
    verdiRecorderCore0 #(.clusterId(0), .cpuId(0), .traceFileName(‟dfile_Core0.log"))
    cpu0(.cpuClock(1'b0));
endmodule
module Core1_cluster();
    verdiRecorderCore1 #(.clusterId(1), .cpuId(0), .traceFileName(‟dfile_Core1.log"))
    cpu0(.cpuClock(1'b0));
endmodule
```

Include other core types

Instantiate additional cores

Define additional cores

Synopsys Detailed Implementation

More detailed information on the field definitions pertaining to memory layouts and clocking are located in the Synopsys overview here:

http://www.synopsys.com/Tools/Verification/debug/Documents/verdi3-hw-sw-debug-ds.pdf

Debug Cases

Sample Debug Issues

- Issue #1 : Memory Aliasing Issue
  - Tesbench based issue
- Issue #2 : Handshaking protocol lockup
  - RTL logic issue
- Issue #3 : STAT bit miscompare
  - Compiler optimization issue

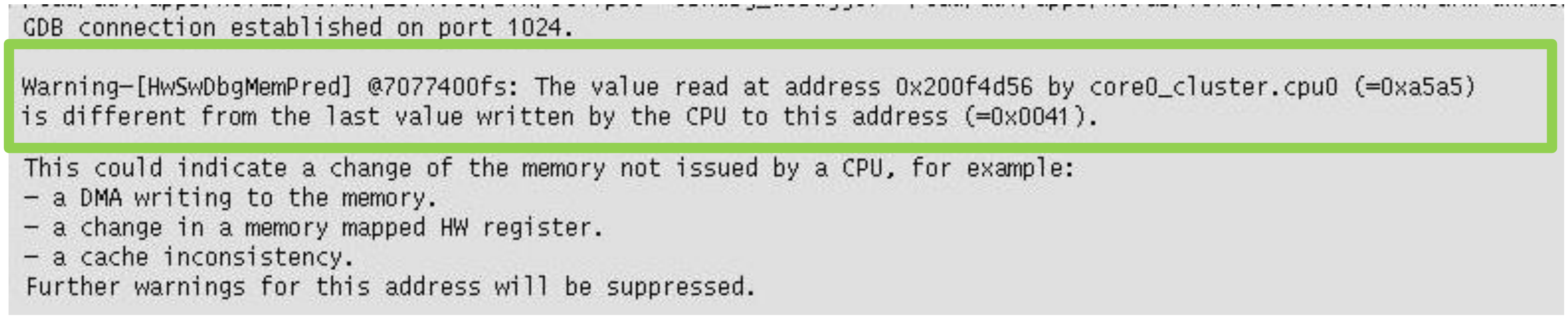
Issue #1 : Memory Aliasing

- The code for Core0 executes an output to some flags on the processor to track execution flow
- Core1 had some randomized local memory write/reads
- In the flow, the Core0 execution became corrupted
- A day was spent on this with traditional debug methods without any positive result
- Depending on additional code added for debug, the test would pass correctly, pass prematurely, or fail as before

Issue #1 : Memory Aliasing

- Using the Verdi HW/SW Debug Tool, Core0 was stepped through and the PC was monitored, but the test just ended on a fault

- The Verdi HW/SW Tool's message window was flagged with an unexpected memory location change, which detailed the time at which an address in the instruction memory was changing vs the Tool's memory model

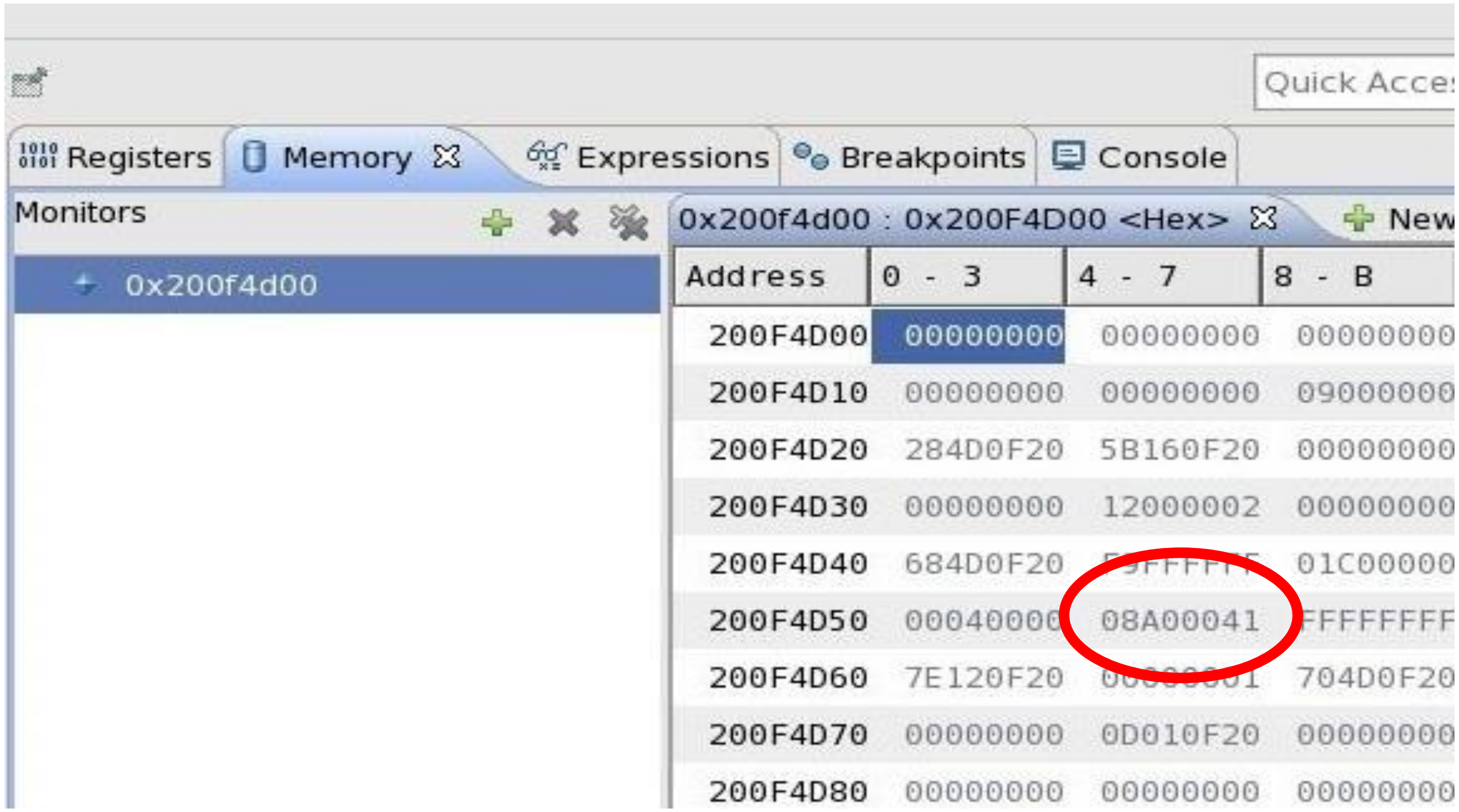


```
GDB connection established on port 1024.

Warning-[HwSwDbgMemPred] @7077400fs: The value read at address 0x200f4d56 by core0_cluster.cpu0 (=0xa5a5)
is different from the last value written by the CPU to this address (=0x0041).

This could indicate a change of the memory not issued by a CPU, for example:
- a DMA writing to the memory.
- a change in a memory mapped HW register.
- a cache inconsistency.
Further warnings for this address will be suppressed.
```

- Using the Verdi waveform, an expression was added to the memory interface address pins to flag when this location was accessed. It happened more often than expected

Issue #1 : Memory Aliasing

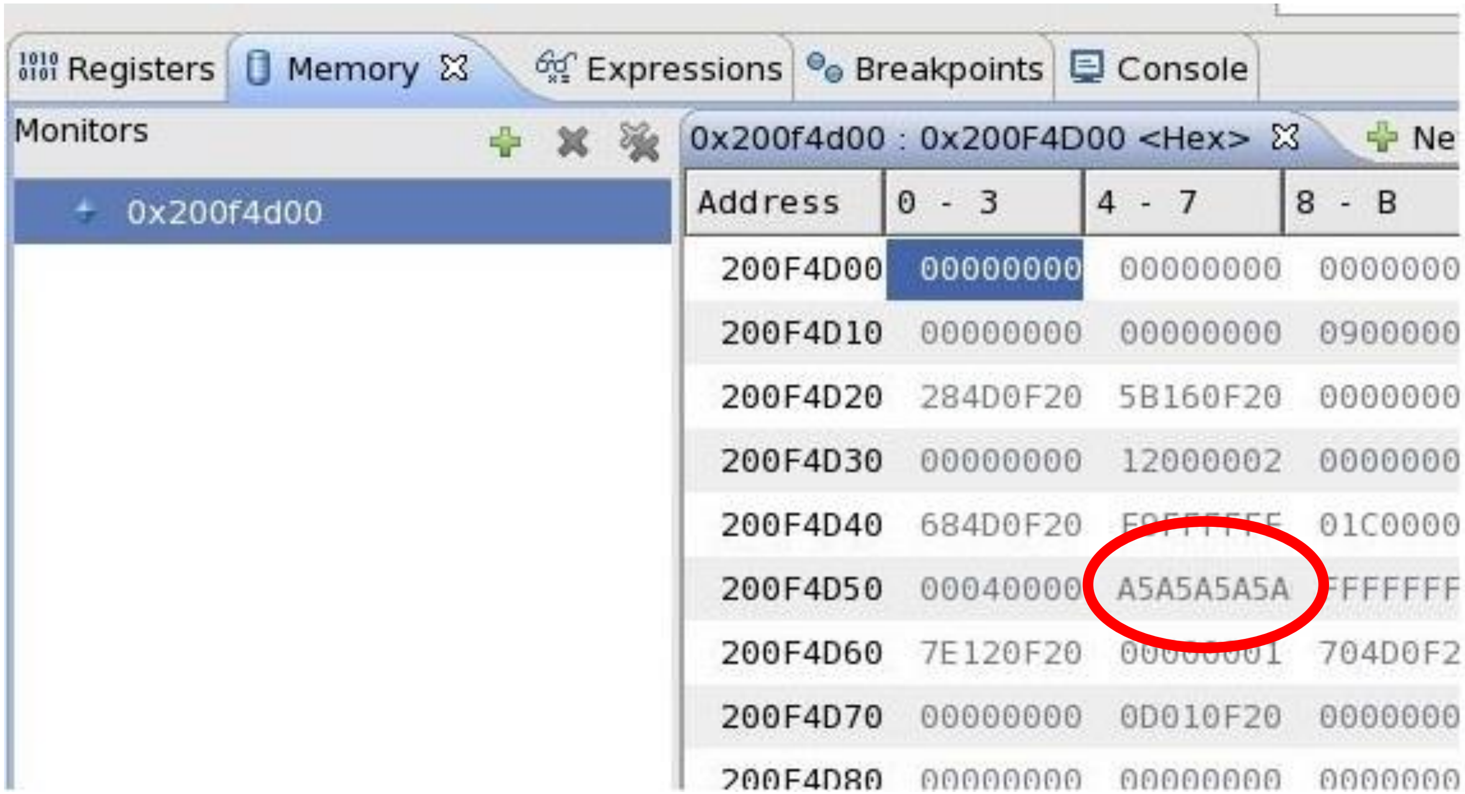
One step before the offending memory access occurs.



```
Address    0 - 3     4 - 7     8 - B
200F4D00   00000000  00000000  00000000
200F4D10   00000000  00000000  09000000
200F4D20   284D0F20  5B160F20  00000000
200F4D30   00000000  12000002  00000000
200F4D40   684D0F20  F9FFFFFF  01C00000
200F4D50   00040000  08A00041  FFFFFFFF
200F4D60   7E120F20  00000001  704D0F20
200F4D70   00000000  0D010F20  00000000
200F4D80   00000000  00000000  00000000
```

Issue #1 : Memory Aliasing

One step after the offending memory access occurs.



Address	0 - 3	4 - 7	8 - B
200F4D00	00000000	00000000	0000000
200F4D10	00000000	00000000	0900000
200F4D20	284D0F20	5B160F20	0000000
200F4D30	00000000	12000002	0000000
200F4D40	684D0F20	F0FFFFFF	01C0000
200F4D50	00040000	A5A5A5A5	FFFFFFF
200F4D60	7E120F20	00000001	704D0F2
200F4D70	00000000	0D010F20	0000000
200F4D80	00000000	00000000	0000000

Memory Aliasing Conclusion

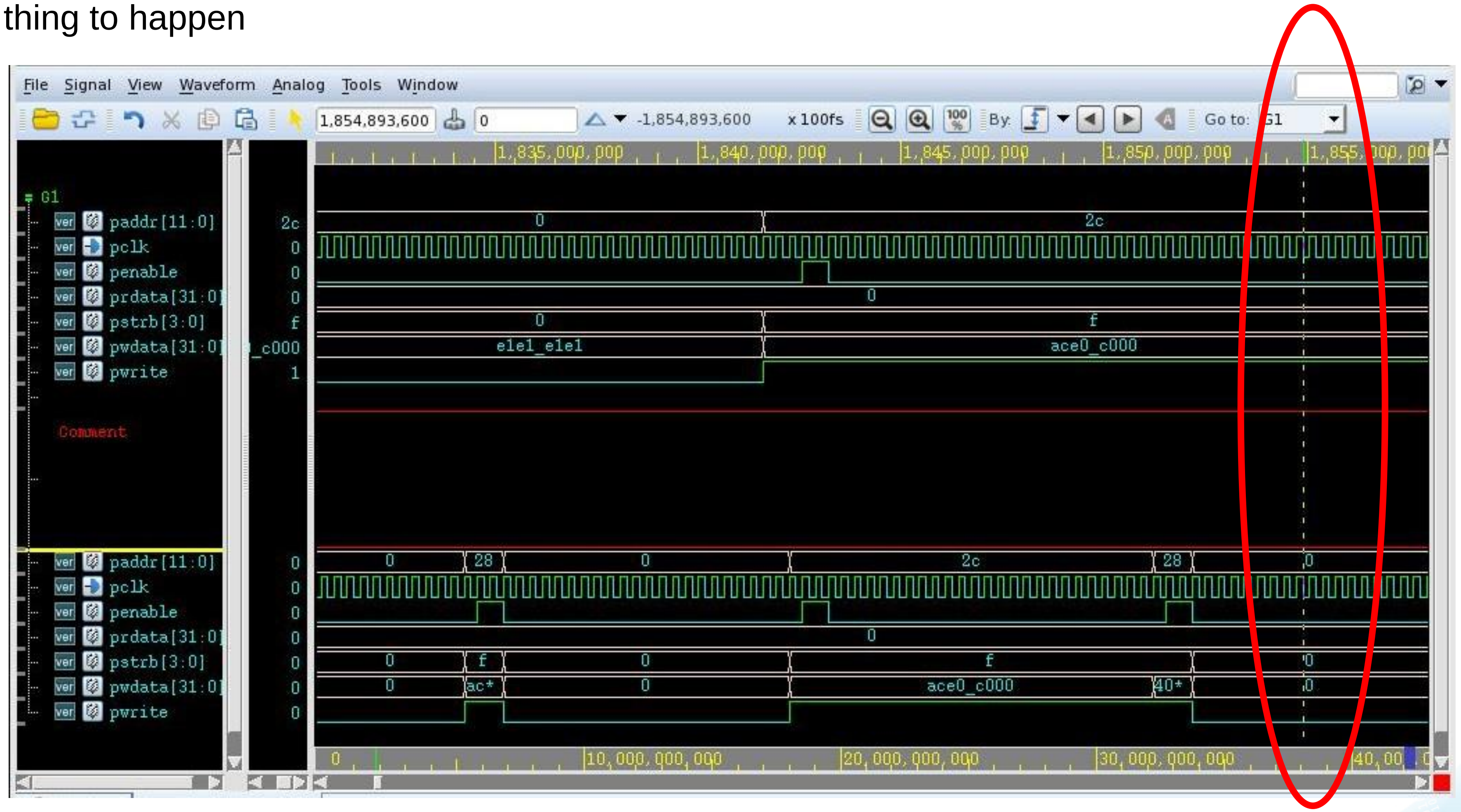
- After stepping through Core1 code, it did not show any access to that memory location, though a local memory access was occurring
- The waveform view showed the corrupted address was seen on the Core1 memory interface
- The address was traced back through the design to an aliasing offset register in Core1 that had been randomized without proper constraints, corrupting the Core0 program stack
- Debug time using the tool was a little over an hour

Issue #2 : Mailbox lockup

- A handshaking protocol between Core0 and Core1 was locking up
- Simple test: send_message, expect_message
- Messages originally printed in the log file showed that traffic in both directions worked properly for a number of transactions and then the execution stalled

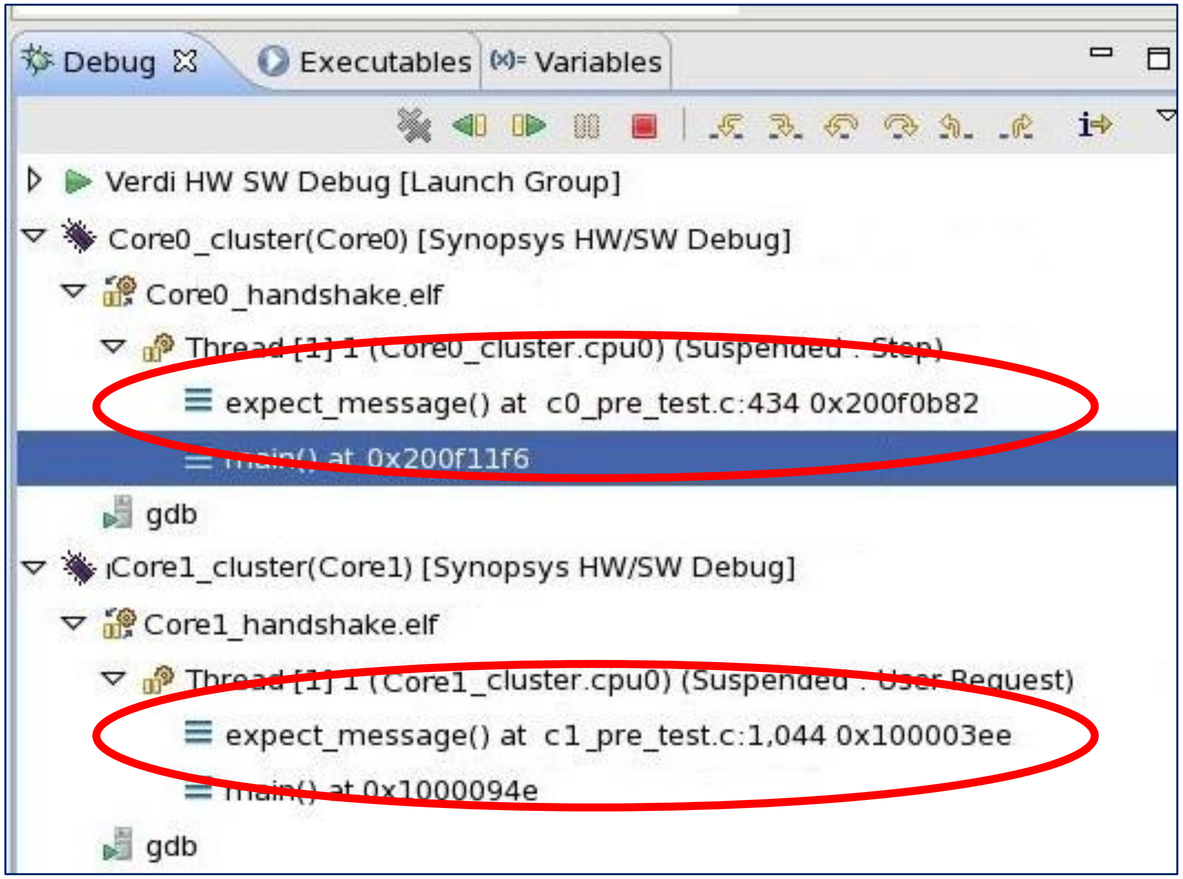
Issue #2 : Mailbox lockup

First step was to open the waveform in Verdi and click cursor in waveform at last interesting thing to happen



Issue #2 : Mailbox lockup

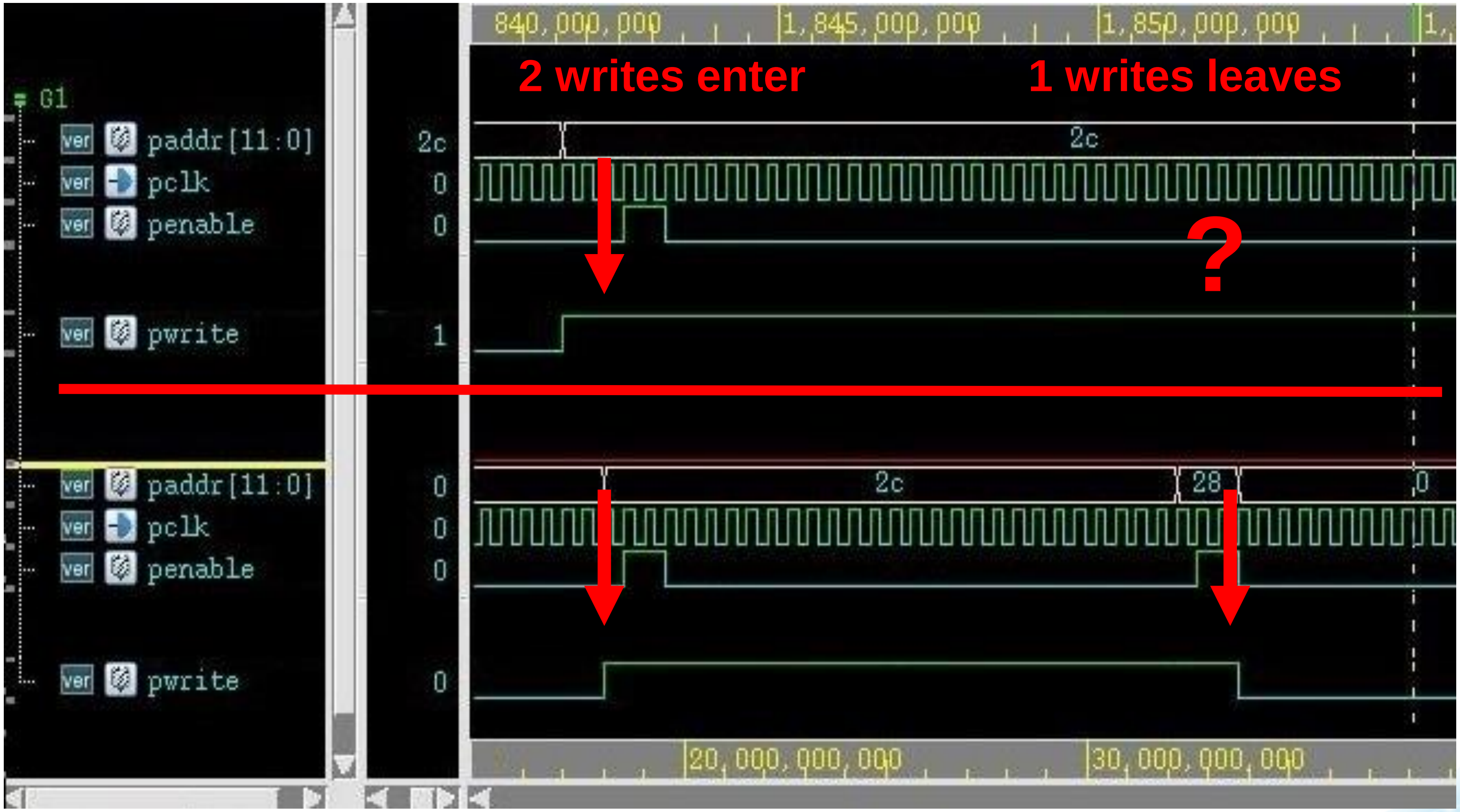
The stack tracing and C code of Core0 and Core1 are below, where both are locked up at the same register access for expect_message()



```
Verdi HW SW Debug [Launch Group]
Core0_cluster(Core0) [Synopsys HW/SW Debug]
  Core0_handshake.elf
    Thread [1] 1 (Core0_cluster.cpu0) (Suspended : Step)
      expect_message() at c0_pre_test.c:434 0x200f0b82
      main() at 0x200f11f6
  gdb
Core1_cluster(Core1) [Synopsys HW/SW Debug]
  Core1_handshake.elf
    Thread [1] 1 (Core1_cluster.cpu0) (Suspended : User Request)
      expect_message() at c1_pre_test.c:1,044 0x100003ee
      main() at 0x1000094e
  gdb
```

Issue #2 : Mailbox lockup

We can see that the write signal never deasserts for Core0, while Core1 functioned as expected. We single stepped through the assembly to see where the Core0 write happens.



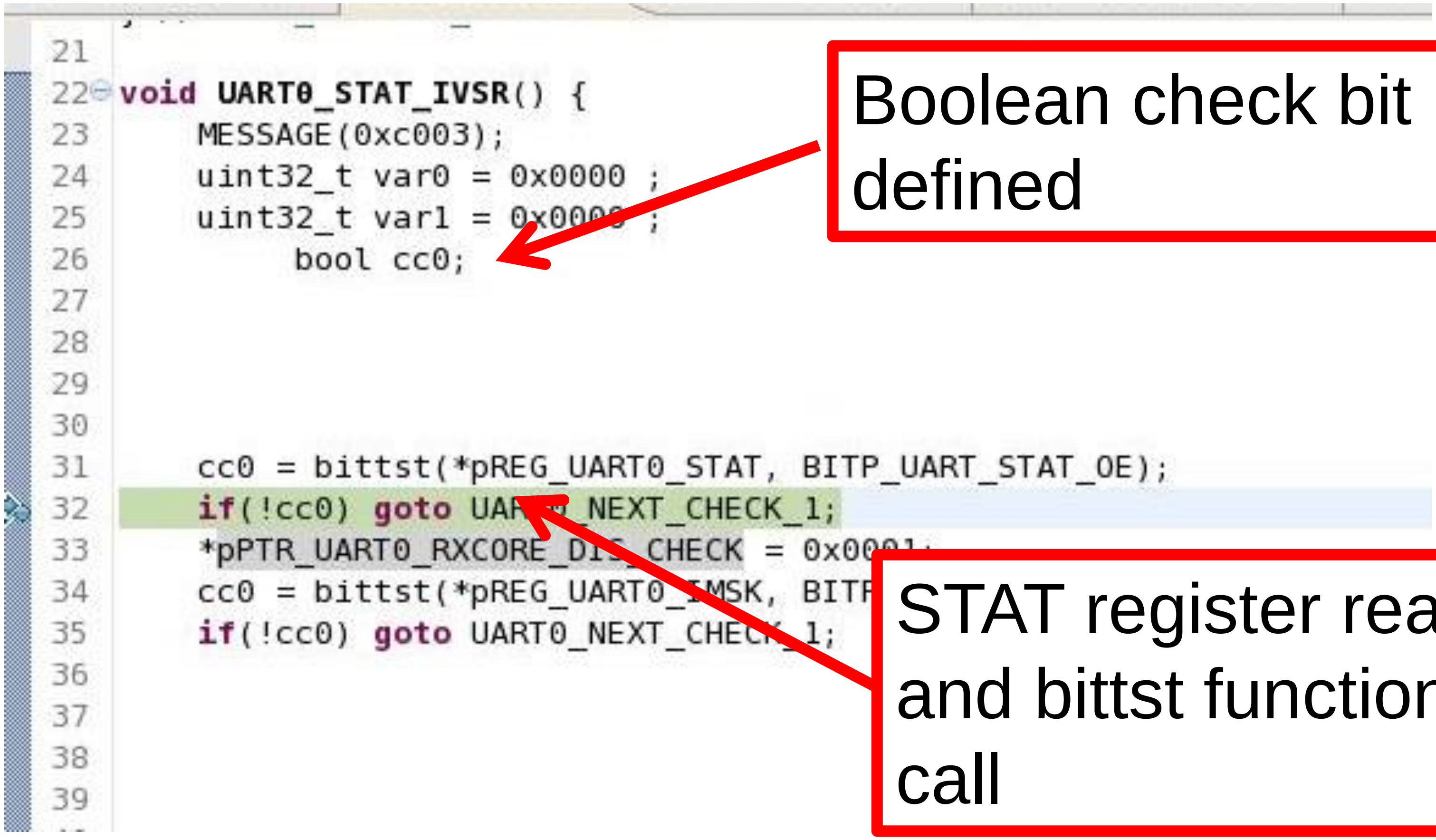
Mailbox lockup Conclusion

- Core1 wrote it's data and then went to the expect wait state while the Core0 RTL fabric was waiting for arbitration from it's own write register access
- Stepping though the code, we could see where Core0 sent out the data and the bus was stalled without feedback to this RTL logic
- The subsequent "expect_data" read caused the bus to lockup
- Debug time from failing test to root cause around 2 hours

Issue #3 : STAT miscompare

- A test was running to timeout on Core0, and messages showed it was stuck in a status check loop
- The interesting note is that executing on Core1, the test **passed** (same code, same peripheral, etc)
- Based on this, first suspect was Core0 fabric logic
- Stepping into the loop with the Verdi HW/SW Debug Tool the status bit was observed being read out but it was not triggering the check

Issue #3 : STAT miscompare

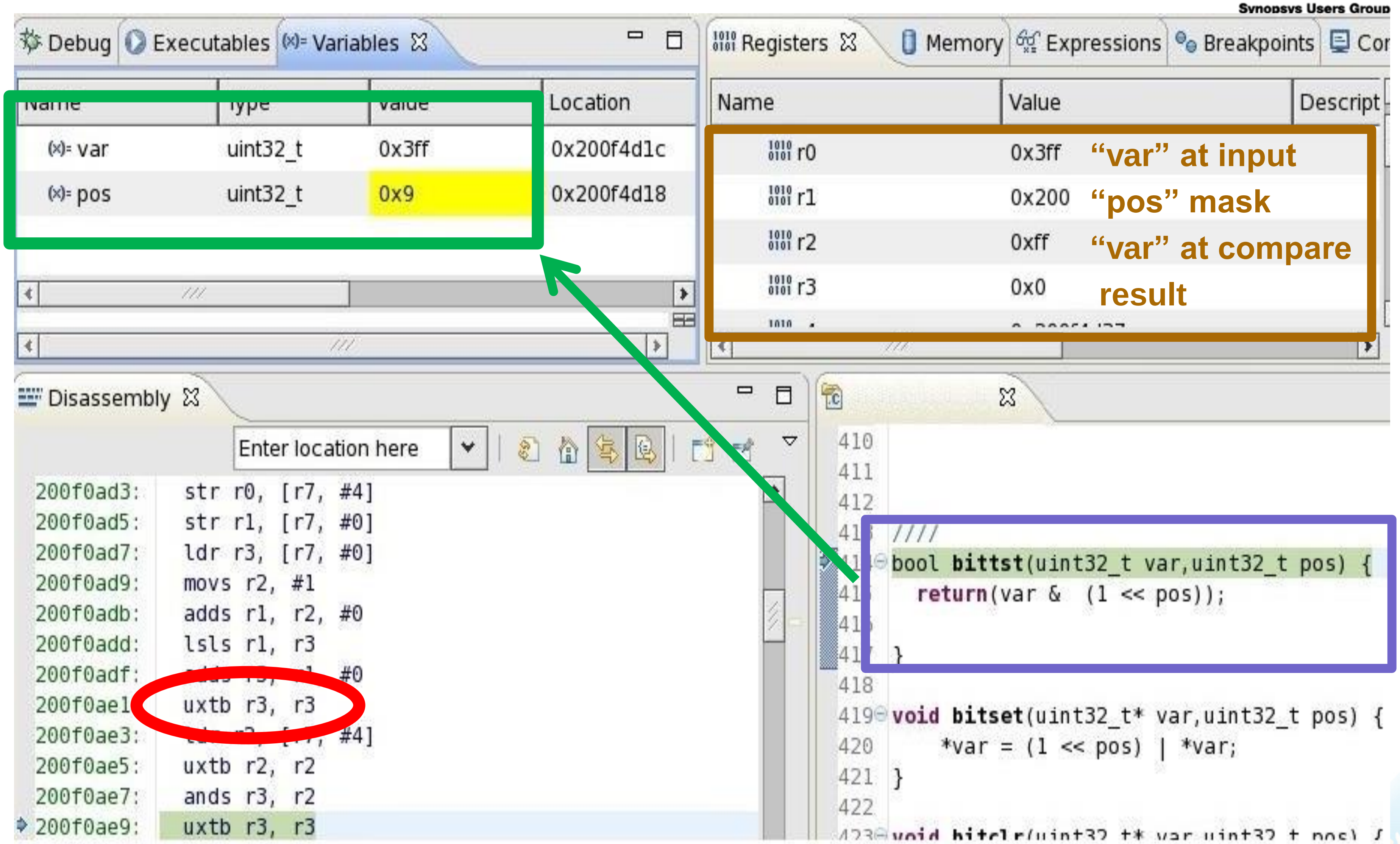


```
void UART0_STAT_IVSR() {
    MESSAGE(0xc003);
    uint32_t var0 = 0x0000 ;
    uint32_t var1 = 0x0000 ;
        bool cc0;




    cc0 = bittst(*pREG_UART0_STAT, BITP_UART_STAT_OE);
    if(!cc0) goto UART0_NEXT_CHECK_1;
    *pPTR_UART0_RXCORE_DIS_CHECK = 0x0001;
    cc0 = bittst(*pREG_UART0_IMSK, BITP
    if(!cc0) goto UART0_NEXT_CHECK_1;
```

Boolean check bit defined

STAT register read and bittst function call

Issue #3 : STAT miscompare



Name	Type	Value	Location
var	uint32_t	0x3ff	0x200f4d1c
pos	uint32_t	0x9	0x200f4d18

Name	Value	Descript
r0	0x3ff	"var" at input
r1	0x200	"pos" mask
r2	0xff	"var" at compare
r3	0x0	result

```
200f0ad3:   str r0, [r7, #4]
200f0ad5:   str r1, [r7, #0]
200f0ad7:   ldr r3, [r7, #0]
200f0ad9:   movs r2, #1
200f0adb:   adds r1, r2, #0
200f0add:   lsls r1, r3
200f0adf:   adds r3, r1, #0
200f0ae1:   uxtb r3, r3
200f0ae3:   ldr r2, [r7, #4]
200f0ae5:   uxtb r2, r2
200f0ae7:   ands r3, r2
200f0ae9:   uxtb r3, r3
```

```
////
bool bittst(uint32_t var,uint32_t pos) {
   return(var &  (1 << pos));

}

void bitset(uint32_t* var,uint32_t pos) {
     *var = (1 << pos) | *var;
}

void bitclr(uint32_t* var,uint32_t pos) {
```

STAT miscompare Conclusion

- Comparing to Core1 there were assembly differences for such a simple function
- Core1 could support 32bit boolean checks, while Core0 could only support 8bit boolean checks

Core0

```
100002da: 683b          ldr r3, [r7, #0]
100002dc: 2201          movs r2, #1
100002de: fa02 f303 lsl.w r3, r2, r3
100002e2: b2da          uxtb r2, r3
100002e4: 687b          ldr r3, [r7, #4]
```

Core1

```
100004b2: 683b          ldr r3, [r7, #0]
100004b4: f04f 0201 mov.w r2, #1
100004b8: fa02 f303 lsl.w r3, r2, r3
100004bc: 461a          mov r2, r3
100004be: 687b          ldr r3, [r7, #4]
```

- The compiler “helpfully” truncated the data to 8 bits for the compare

- Debug time was under an hour

~~Things I whine about like a 2 year old~~
Feature Requests

Main drawback of the tool:

- <u>Name is too clunky:</u>

Verdi Hardware Software Debug Tool

- No cool name: DaVinci, Certitude, Hector
- No Three Letter Acronym: DVE, VCS
- VHWSWDT doesn’t roll off the tongue

- Tried opening a support ticket for this, was unsuccessful

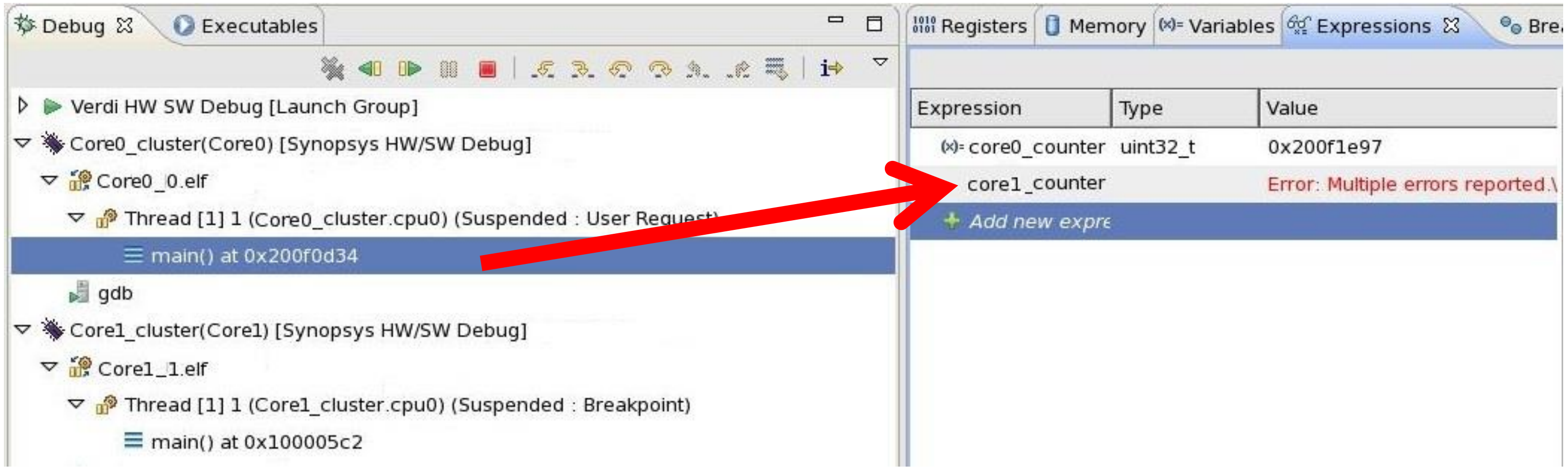
Great tool, warts and all:

- Cannot map RTL register information into the Eclipse GUI for same-window debug of registers, or C variables to the waveform from the GUI

- Logfile is deprecated: cannot pull messages from log into debugger

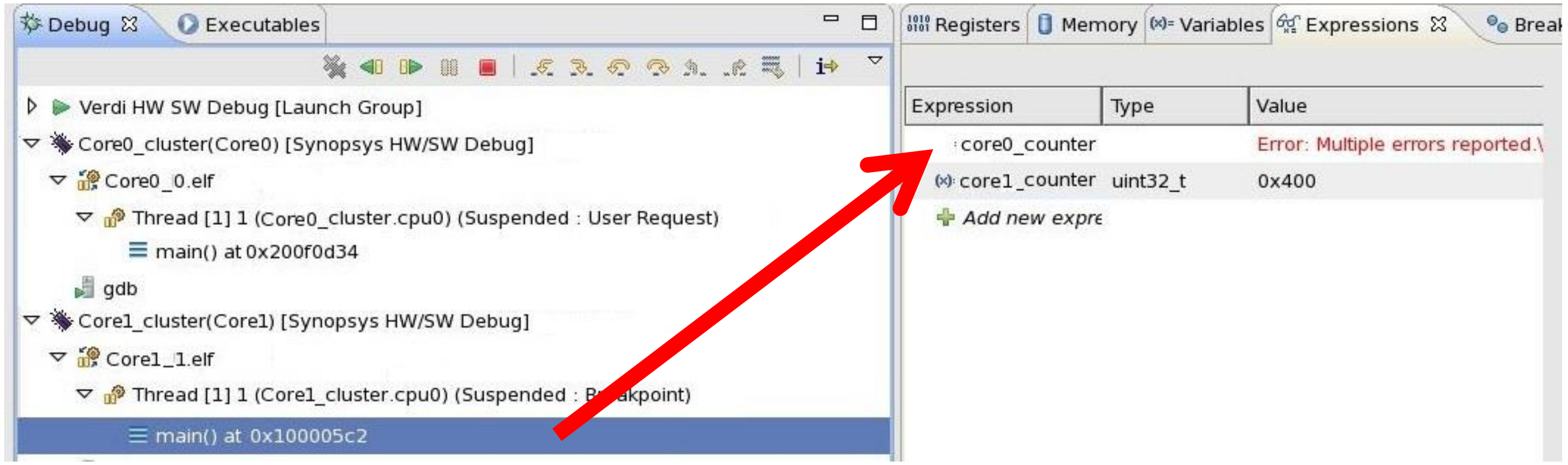
- No scope independence for any Eclipse window, even for user defined expressions

Scope Error Messages



```
▷ Verdi HW SW Debug [Launch Group]
▽ Core0_cluster(Core0) [Synopsys HW/SW Debug]
    ▽ Core0_0.elf
        ▽ Thread [1] 1 (Core0_cluster.cpu0) (Suspended : User Request)
              main() at 0x200f0d34
      gdb
▽ Core1_cluster(Core1) [Synopsys HW/SW Debug]
    ▽ Core1_1.elf
        ▽ Thread [1] 1 (Core1_cluster.cpu0) (Suspended : Breakpoint)
              main() at 0x100005c2
```

Expression	Type	Value
core0_counter	uint32_t	0x200f1e97
core1_counter		Error: Multiple errors reported.\
Add new expre		



```
▷ Verdi HW SW Debug [Launch Group]
▽ Core0_cluster(Core0) [Synopsys HW/SW Debug]
    ▽ Core0_0.elf
        ▽ Thread [1] 1 (Core0_cluster.cpu0) (Suspended : User Request)
              main() at 0x200f0d34
      gdb
▽ Core1_cluster(Core1) [Synopsys HW/SW Debug]
    ▽ Core1_1.elf
        ▽ Thread [1] 1 (Core1_cluster.cpu0) (Suspended : Breakpoint)
              main() at 0x100005c2
```

Expression	Type	Value
core0_counter		Error: Multiple errors reported.\
core1_counter	uint32_t	0x400
Add new expre		

Conclusion

Conclusion

Results

- We've found the fault isolation time using this tool versus traditional methods was reduced by about 80%
  - *Save your time for Coverage Closure!*

- Note: This tool does not do anything that you can't do the "old way", it just does it faster and more efficiently
  - Makes it simple enough for a new hire (or even a manager) to be productive with debug
  - Real world issues are more complex than my NDA sanitized cases
  - Finding interactions in systems with 4, 8, 32 cores, especially a mixed core system, gets tough without it

Conclusion

Who Shouldn't Use the Verdi HW/SW Debug Tool ?

- People who really like print messages for debug
- People who need to do neck exercises by switching views between multiple windows
- People who get paid by the hour
- The tooth fairy

Who Should Use the Verdi HW/SW Debug Tool ?

- People who actually exist

Conclusion

- The options of debug allow a broader range of user to be in a comfort zone while using it:
  - Waveforms and opcodes for RTL and system fabric designers
  - C code for test writers and algorithm developers, legacy test conversion
  - Benchmarkers tracing performance, optimizing code
  - DV engineers get the flexibility to debug both worlds

Acknowledgements

- Andy Sha (co author), ADI : Implementing the first single core Verdi HW/SW Debug Tool at ADI, and standardized a lot of the flows

- Alex Wakefield, Synopsys : Providing the templates and instruction for the implementation

- Dave Brownell, ADI : Assisting with integration into our testbench and sim environment

Thank You