

# Simulation-Based System Level Fault Insertion Using Co- Verification Tools

**Bill Eklow, Toai, Vo, Chi Khuong, Shyam, Pullela, Anoosh, Hosseini,  
Hien Chau**

# Purpose

- **Present an overview of co-verification**
- **Discuss how co-verification is used at Cisco to verify diagnostics, hardware and fault tolerance**
- **Describe the fault insertion environment that was set up to work in conjunction with Cisco's co-verification platform**
- **Provide some examples and discuss pro's and con's**

# Outline

- **Co-verification: what is it and what are the benefits**
- **Background on Fault Insertion (why it is used and current techniques)**
- **How do you set up a co-verification environment**
- **Adding Fault insertion to the environment**
- **Preliminary results of “virtual fault insertion” on a Cisco product**

# Thanks Toai!!!

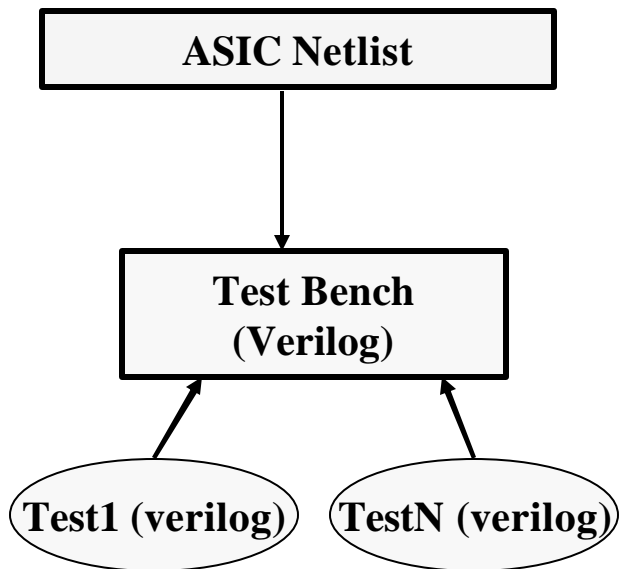
# Co-verification: What is it? Why use it?

Cisco.com

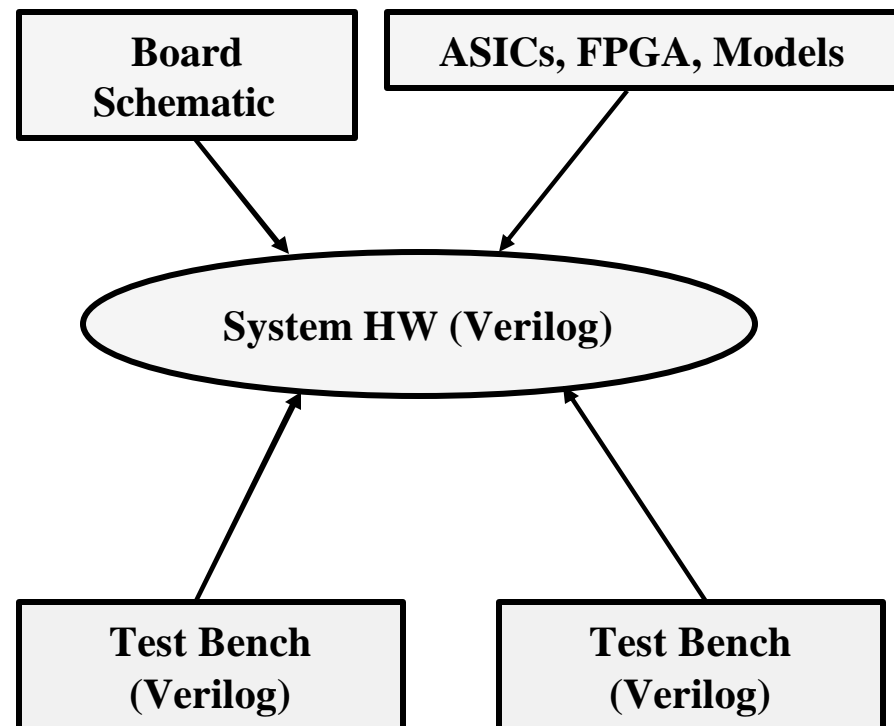
- **Interface between software and hardware verification environments**
  - Software running on workstation
  - Hardware running in Verilog environment
- **Primary use in the SoC environment**
  - Embedded processor arrays
- **Provides early, functional verification of hardware**
  - Critical to time to market
  - Quicker debug of hardware in simulation environment
- **Allows code debug without real hardware**

# Current verification environment

## ASIC Verification



## Board/System Verification

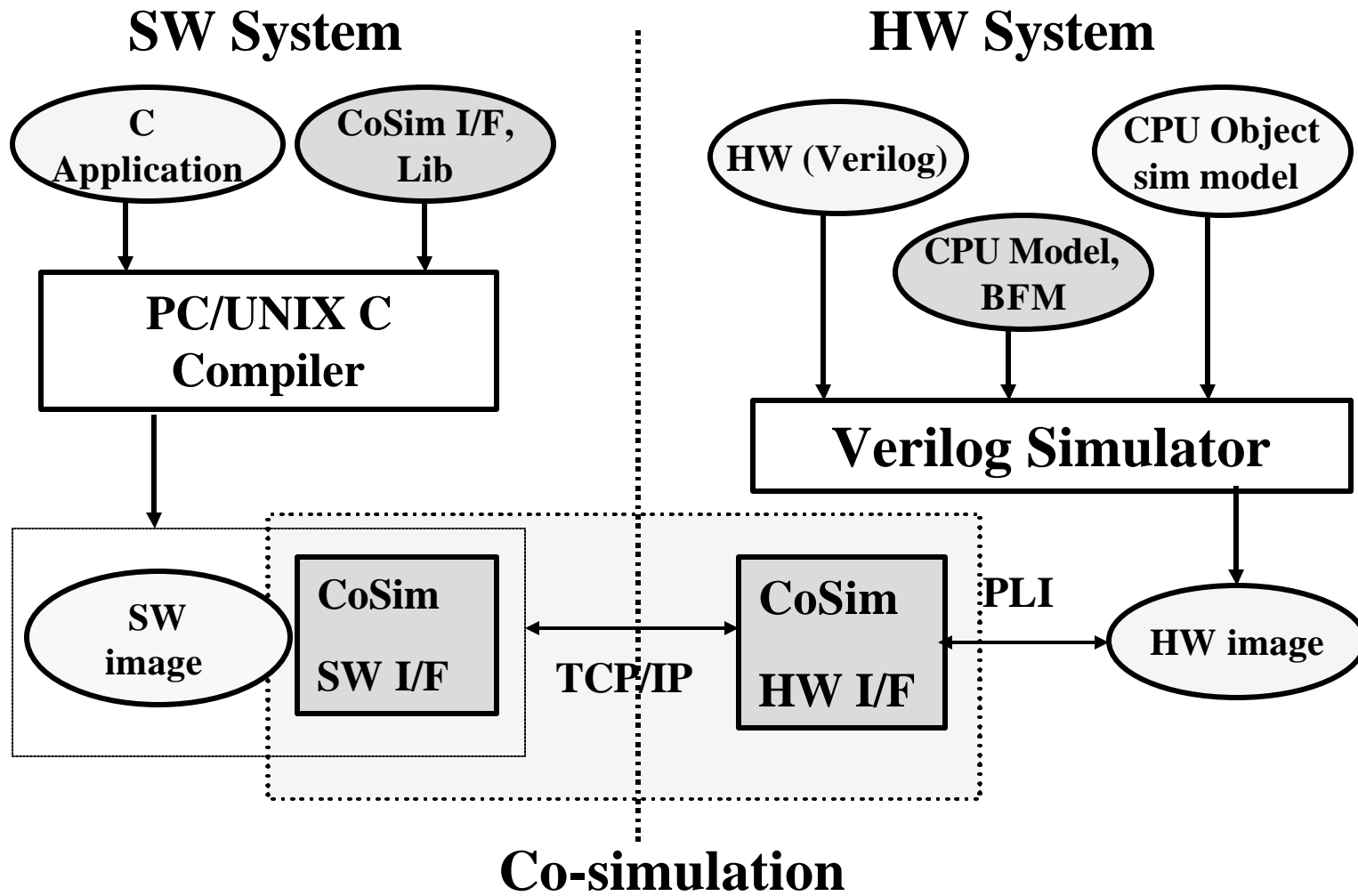


# Current verification environment - problems

Cisco.com

- **ASIC Verification – focus on functions**
- **Board Verification – focus on interfaces**
- **What gets missed**
  - **Physical placement of the part on the board**
  - **Simple logic (decoders)**
  - **Software**

# The Co-verification environment



# Co-verification: Read instruction

```
// SW SIDE: example of C code -- READ HW  
Uint32 *base_addr = (Uint32 *) 0xB0000000;  
Uint32 data = *base_addr; // READ
```

**Since `*base_addr` is NOT in SW address space, SIGSEG violation will be generated by SW system. CoSim SW I/F will intercept the SIGSEG, check if it is in HW address space, translate the address into physical HW address, then request a READ transaction to the HW side.**

(Queue transaction) Translate to a READ transaction

READ data

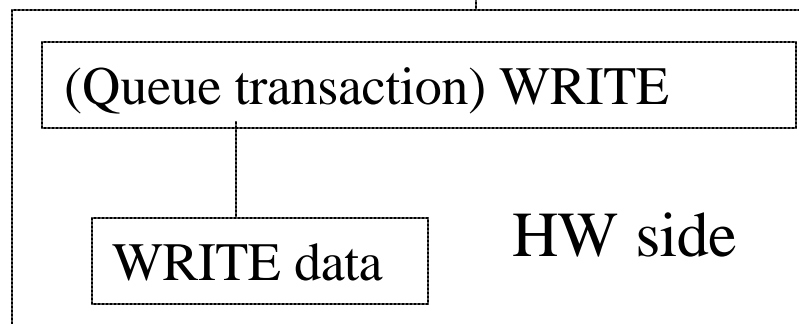
Return data to SW

HW side

# Co-verification: Write instruction

```
// SW SIDE: example of C code -- WRITE TO HW  
  
Uint32 *base_addr = (Uint32 *) 0xB0000000;  
  
Uint32 *asic_addr = 0x20 ; // Write x20 to asic_addr  
  
// Next command  
if ( A == B) then ...
```

Since `asic_addr` is a HW address, CoSim SW I/F will translate the address into physical HW address, then request a WRITE transaction to the HW side. SW will continue to the next execution without waiting for HW WRITE complete.



# Co-verification results

- **High end router product**
  - 5 ASIC's (15 million gates total)
  - 2, high end FPGA's
  - MIPS processor
  - 5000 – 7000 nets
- **Approximately 340 cycles/second with accelerator**
- **Full pass of all diagnostics – 2 days**
- **Optimization done to minimize clock cycles – broadside loading**

# Fault Insertion: Why do it?

Cisco.com

- **Our customers require it (OS level)**
- **Can potentially expose diagnostic issues**
  - Coverage
  - Error handling

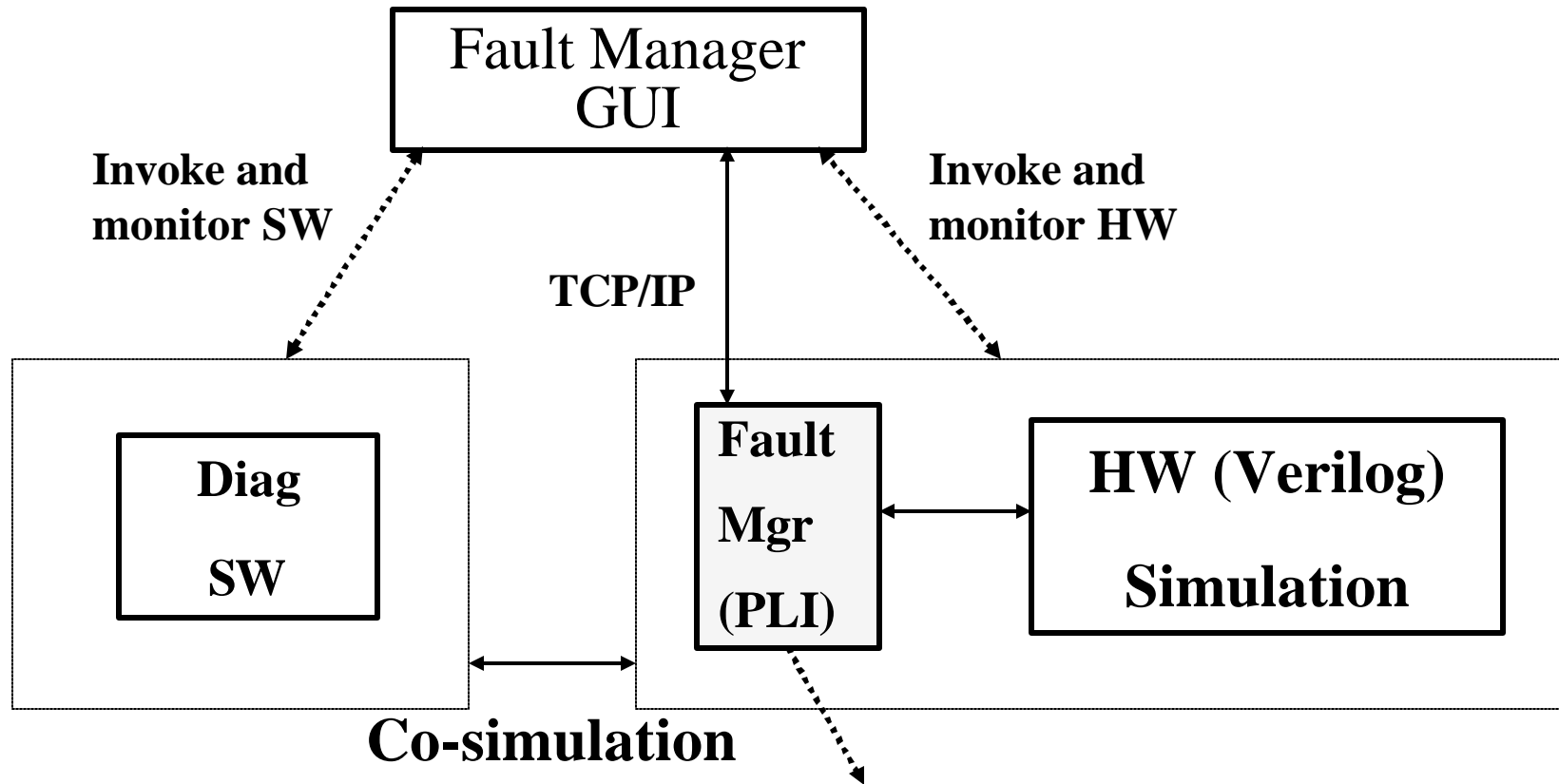
# Fault Insertion: Current approaches

Cisco.com

- **Hardwire**
  - Very time intensive
  - Difficult to debug problems
  - Post silicon
- **Boundary scan insertion**
  - Automated insertion, built into chip
  - Difficult to debug
  - Post Silicon
- **Virtual Fault Insertion**
  - Very time intensive (compute intensive)
  - Easy to debug
  - Pre-silicon (can be done in parallel with silicon verification)

# The Fault Insertion Environment

Cisco.com

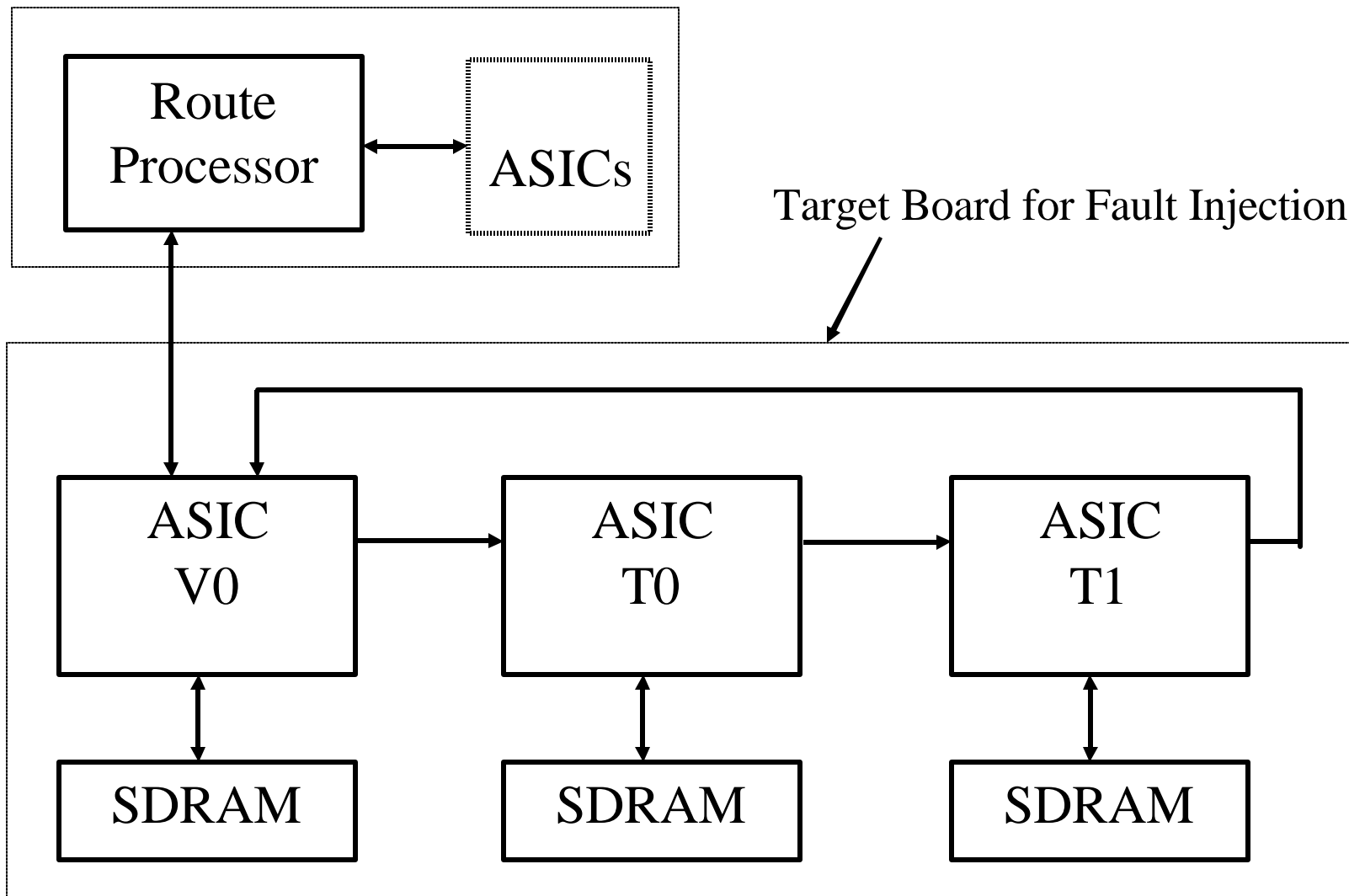


Perform: SA fault injection, generate fault list, board info, fault reports, save and restore HW, user commands, etc.,...

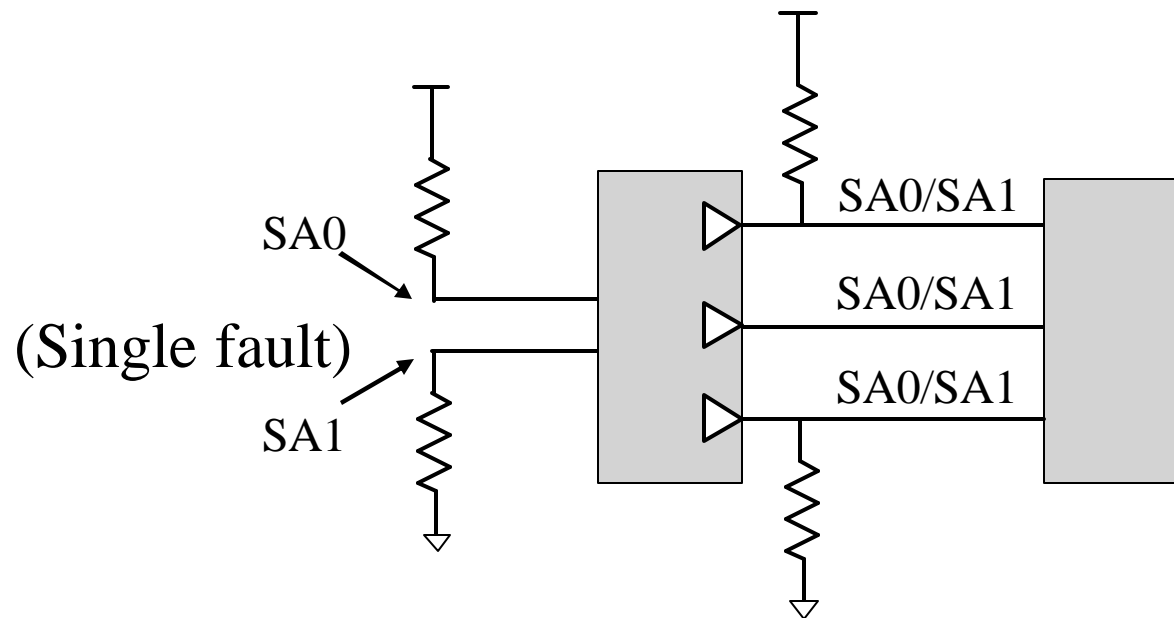
# Test case

- **Route processor for mid/high end router**
- **Selected a subsection of the board**
  - **Packet processing engine**
  - **2 ASIC's consist of 4x4 array of ARM-7 processors**
  - **1 ASIC provides support and packet forwarding**
  - **Both ASIC types >5M gates**
  - **Other on board logic involved in simulation but not fault insertion**
  - **Only register and traffic tests were run**

# Fault Insertion Test Case



# Fault Insertion Test Case



## Fault Simulation Result

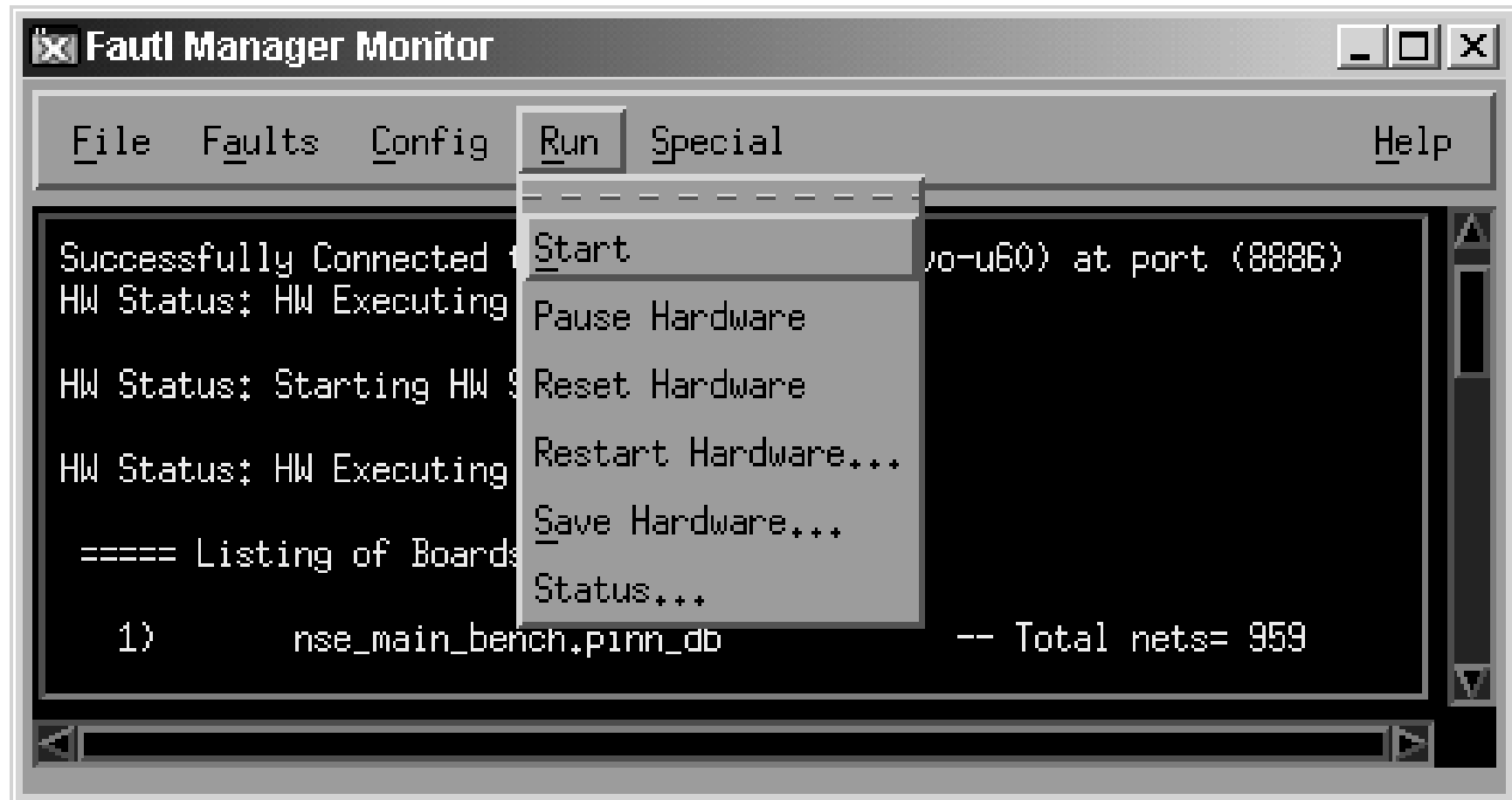
Total Nets	Single Flt Nets	Tot SA Flts	Det Flts	Flt Cov
<b>1410</b>	<b>251</b>	<b>2569</b>	<b>2543</b>	<b>99.00%</b>

# Undetected Faults

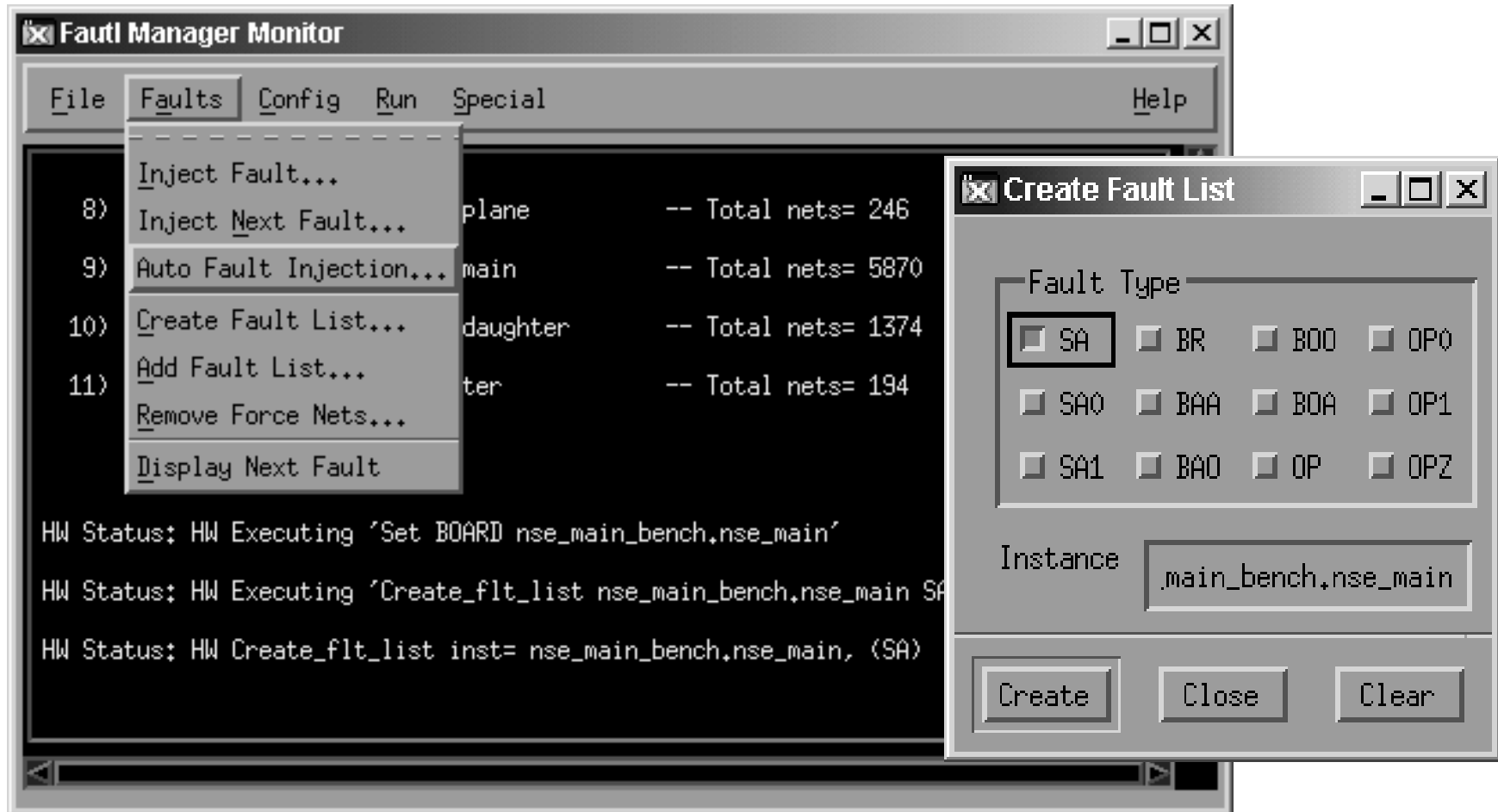
- **Traffic control mode signal**
- **QoS flow control (not enough packets were sent)**
- **Some status and monitor signals**
- **Some optional ASIC debug enable/control signals**

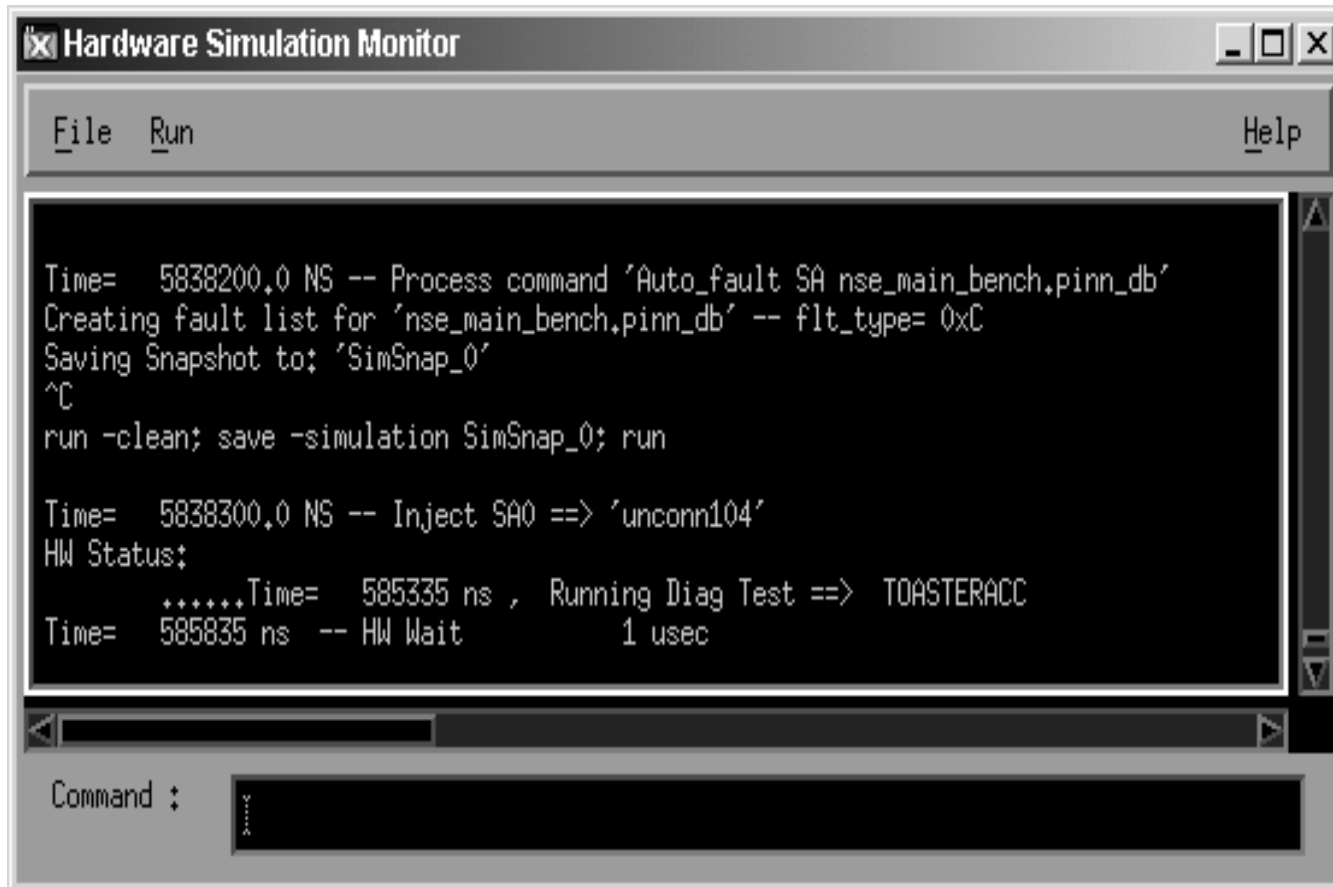


Fault Manager



# Fault Commands





```
Hardware Simulation Monitor
File Run Help

Time= 5838200,0 NS -- Process command 'Auto_fault SA nse_main_bench,pinn_db'
Creating fault list for 'nse_main_bench,pinn_db' -- flt_type= 0xC
Saving Snapshot to: 'SimSnap_0'
^C
run -clean; save -simulation SimSnap_0; run

Time= 5838300,0 NS -- Inject SA0 ==> 'unconn104'
HW Status:
.....Time= 585335 ns , Running Diag Test ==> TOASTERACC
Time= 585835 ns -- HW Wait 1 usec

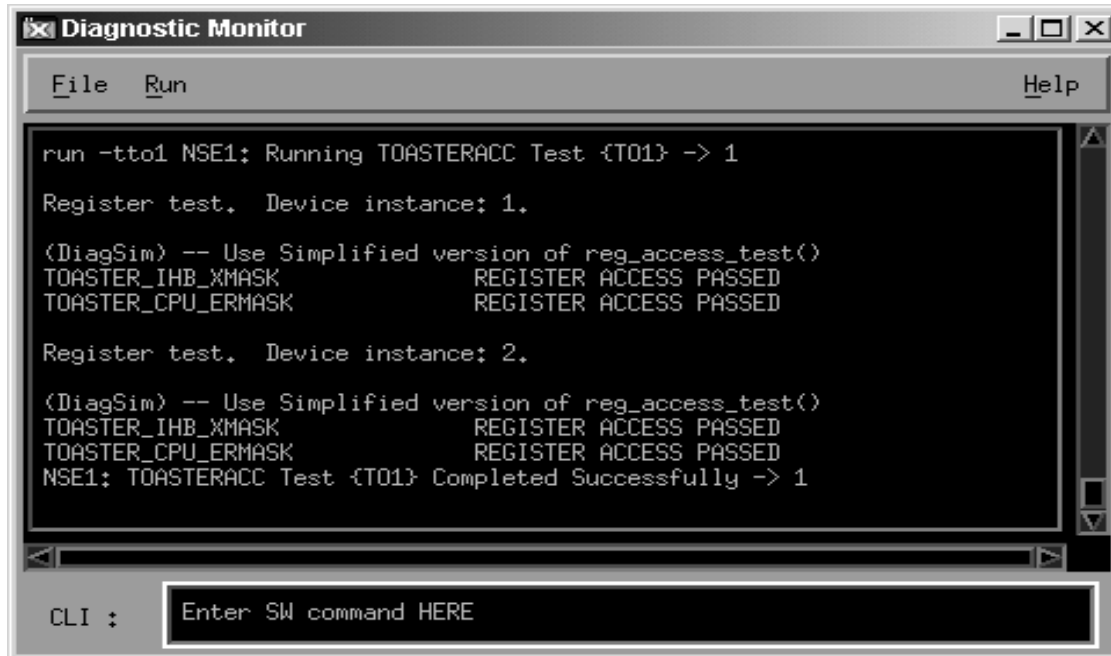
Command :
```

HW Sim:

Injected an

Undetected

SA0 fault



```
run -tto1 NSE1: Running TOASTERACC Test {T01} -> 1

Register test. Device instance: 1.

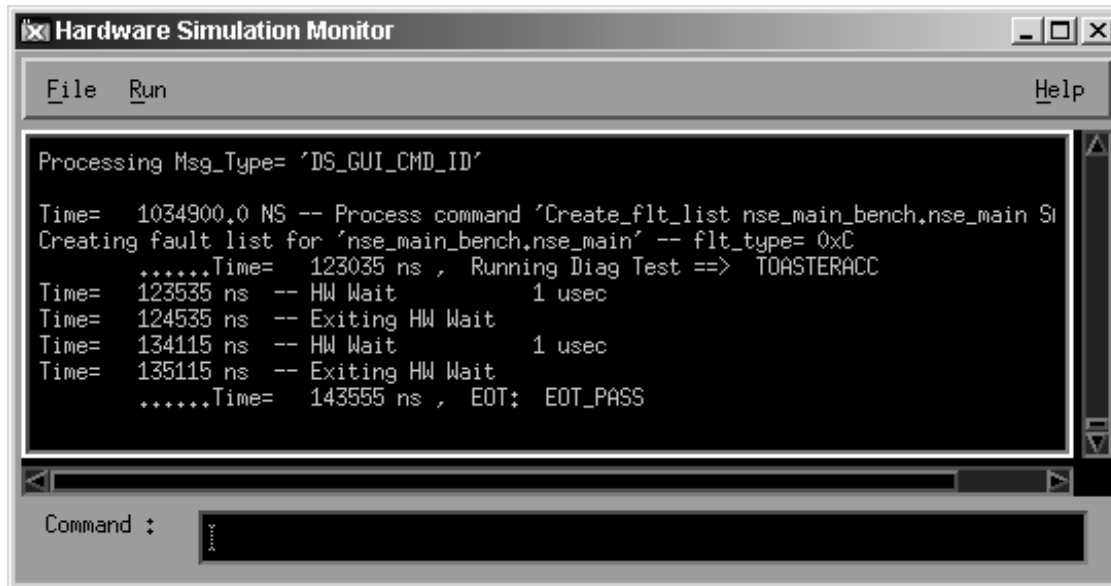
(DiagSim) -- Use Simplified version of reg_access_test()
TOASTER_IHB_XMASK      REGISTER ACCESS PASSED
TOASTER_CPU_ERMASK     REGISTER ACCESS PASSED

Register test. Device instance: 2.

(DiagSim) -- Use Simplified version of reg_access_test()
TOASTER_IHB_XMASK      REGISTER ACCESS PASSED
TOASTER_CPU_ERMASK     REGISTER ACCESS PASSED
NSE1: TOASTERACC Test {T01} Completed Successfully -> 1

CLI : Enter SW command HERE
```

SW Run:  
(Test PASS)



```
Processing Msg_Type= 'DS_GUI_CMD_ID'

Time= 1034900,0 NS -- Process command 'Createflt_list nse_main_bench.nse_main Si
Creating fault list for 'nse_main_bench.nse_main' -- flt_type= 0xC
.....Time= 123035 ns , Running Diag Test ==> TOASTERACC
Time= 123535 ns -- HW Wait 1 usec
Time= 124535 ns -- Exiting HW Wait
Time= 134115 ns -- HW Wait 1 usec
Time= 135115 ns -- Exiting HW Wait
.....Time= 143555 ns , EOT: EOT_PASS

Command :
```

HW Sim:  
(Test PASS)



```
Hardware Simulation Monitor
File Run Help

Time= 799700,0 NS -- Process command 'Inject SA0 \tstr_ctrl_ad7-x0'
Saving Snapshot to: 'SimSnap_0'

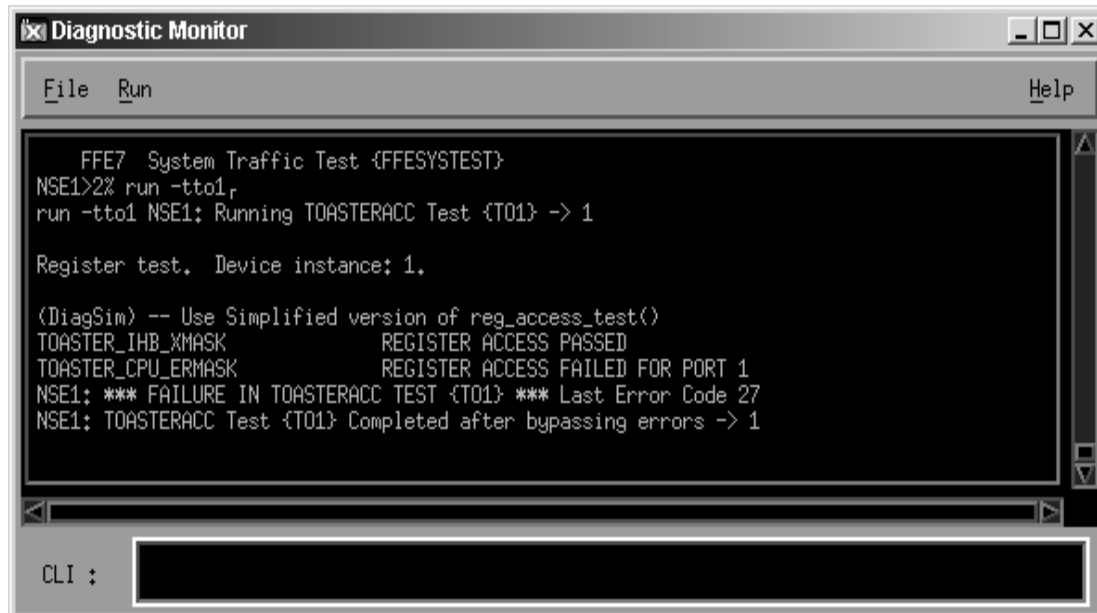
Time= 799700,0 NS -- Inject SA0 ==> '\tstr_ctrl_ad7-x0 '
HW Status:
^C
run -clean; save -simulation SimSnap_0; run
.....Time= 85705 ns , Running Diag Test ==> TOASTERACC
Time= 86195 ns -- HW Wait 1 usec
Time= 87195 ns -- Exiting HW Wait
.....Time= 95635 ns , EOT; EOT_FAIL

Command :
```

HW Sim:

Inject SA0 fault  
(Test Failed) --

Fault is detected



```
Diagnostic Monitor
File Run Help

FFE7 System Traffic Test {FFESYSTEST}
NSE1>2% run -tto1,
run -tto1 NSE1: Running TOASTERACC Test {T01} -> 1

Register test. Device instance: 1.

(DiagSim) -- Use Simplified version of reg_access_test()
TOASTER_IHB_XMASK REGISTER ACCESS PASSED
TOASTER_CPU_ERMASK REGISTER ACCESS FAILED FOR PORT 1
NSE1: *** FAILURE IN TOASTERACC TEST {T01} *** Last Error Code 27
NSE1: TOASTERACC Test {T01} Completed after bypassing errors -> 1

CLI :
```

SW Run:

(Test Failed)



The screenshot shows a window titled "Fault Manager Monitor" with a menu bar containing "File", "Faults", "Config", "Run", "Special", and "Help". The main area is a black terminal window with white text displaying the following log messages:

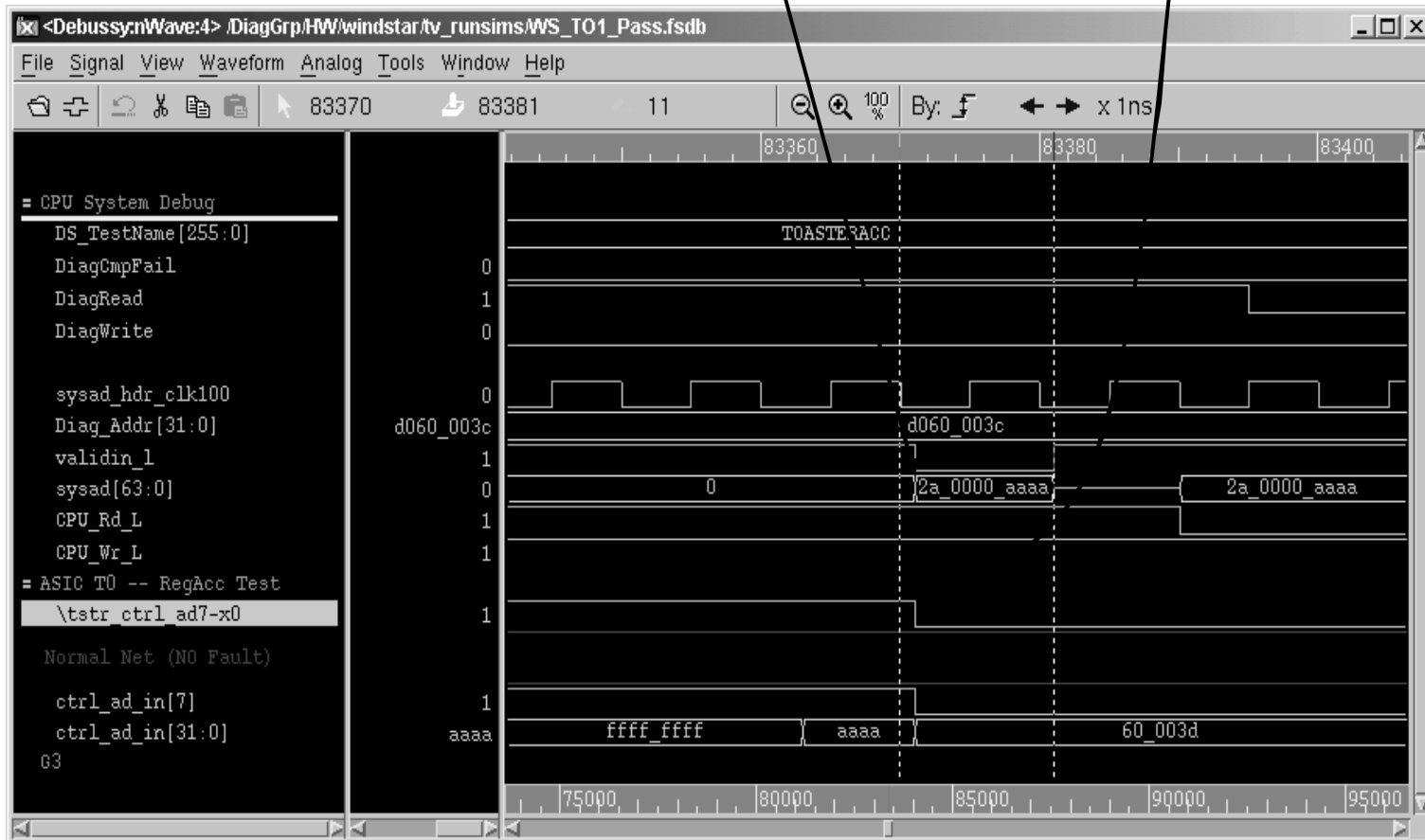
```
HW Status: Starting HW Simulation ....  
HW Status: HW Executing 'Set BOARD nse_main_bench,nse_main'  
HW Status: HW Executing 'Create_flt_list nse_main_bench,nse_main SA'  
HW Status: HW Create_flt_list inst= nse_main_bench,nse_main, (SA)  
HW Status: HW Executing 'Inject SA0 \tstr_ctrl_ad7-x0'  
HW Status: HW Inject fault SA0 on net= \tstr_ctrl_ad7-x0  
HW Status: HW Executing 'Remove SA0 \tstr_ctrl_ad7-x0'
```

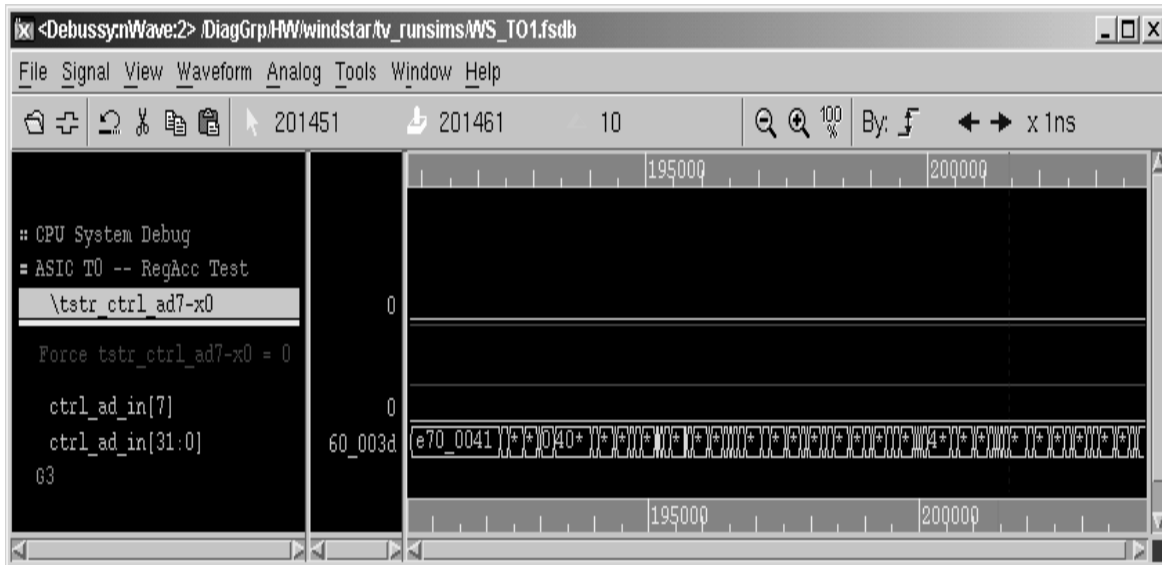
## Fault Manager: Log Messages

# No Fault -- Net \tstr\_ctrl\_ad7-x0

Read data = 2a\_0000\_aaaa

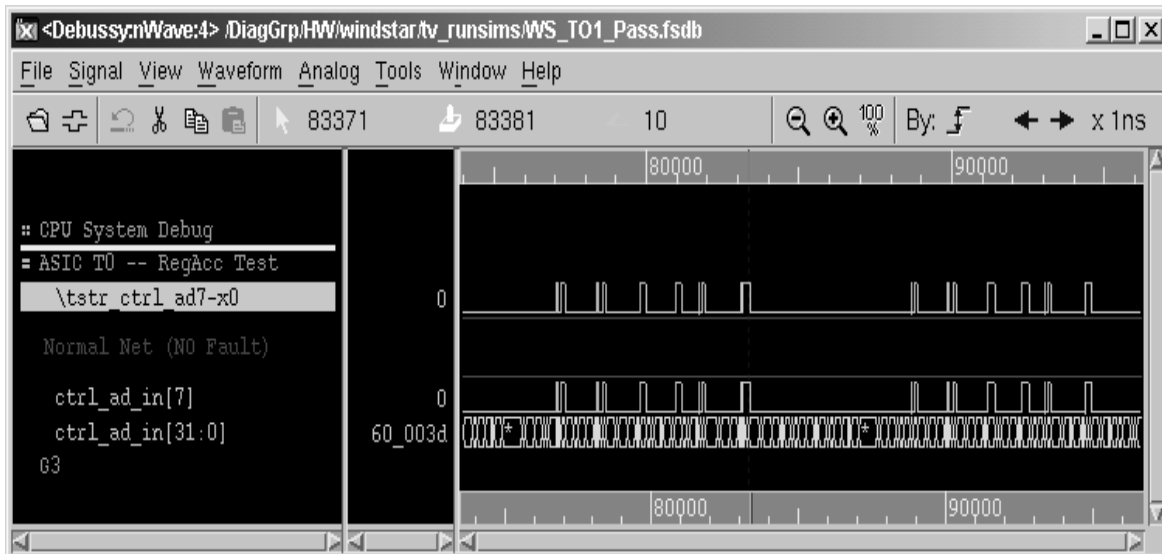
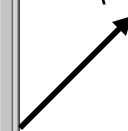
net: \tstr\_ctrl\_ad7-x0





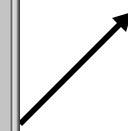
Fault Sim SA0 net:

\tstr\_ctrl\_ad7-x0



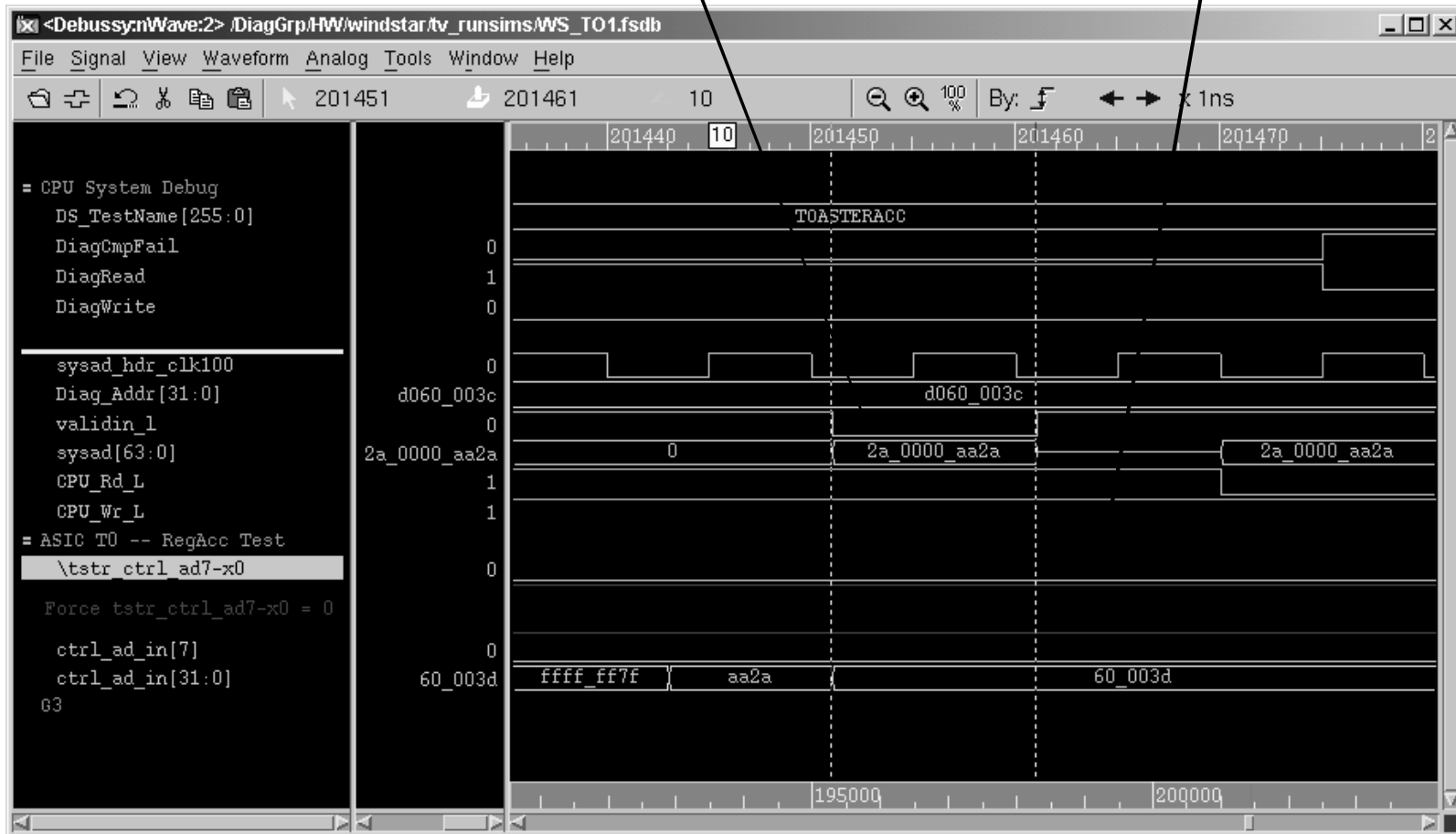
No Fault Sim net:

\tstr\_ctrl\_ad7-x0



# Inject SA0 Fault -- Net \tstr\_ctrl\_ad7-x0

Read data = 2a\_0000\_aa2a      Force \tstr\_ctrl\_ad7-x0 = 0



# Conclusion

- **Co-verification is VERY difficult and time consuming**
- **BUT – it can have great value on high complexity and fast time to market products**
- **If your engineers use it – the environment can be easily modified to include fault insertion**
- **Let's talk more about Functional Test in BTW!**