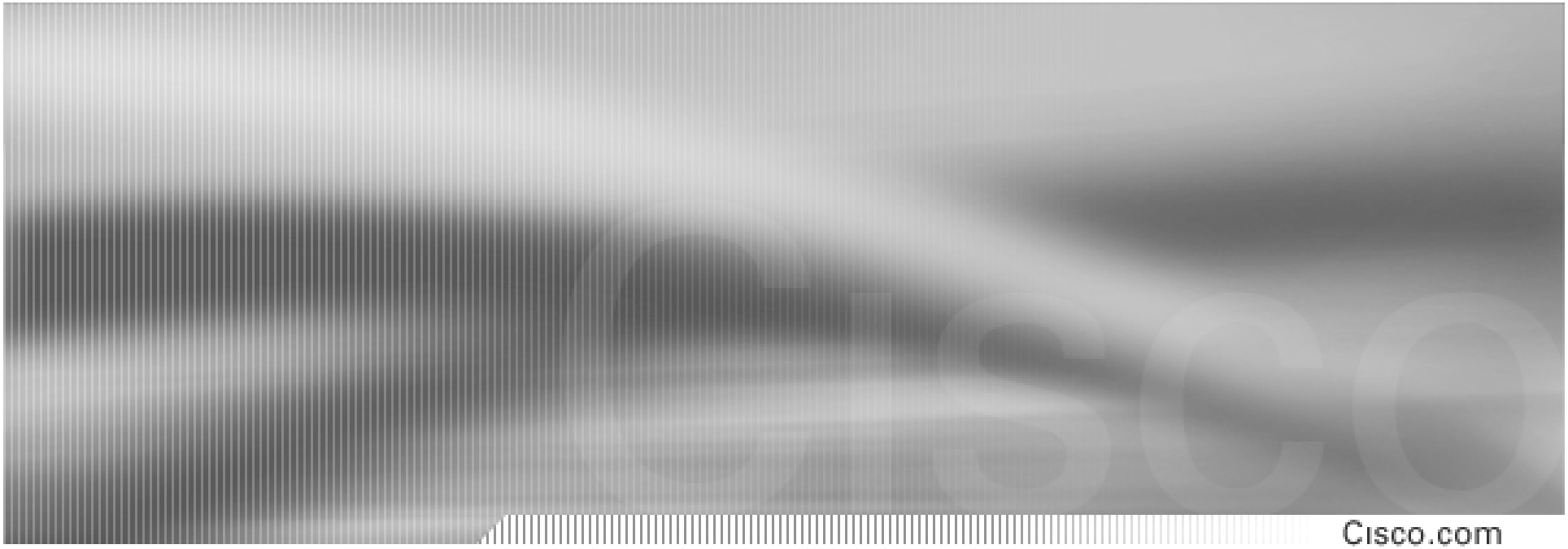


Simulation-Based System Level Fault Insertion Using Co-Verification Tools

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

Prototype and Production DFT and Structural Test

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**

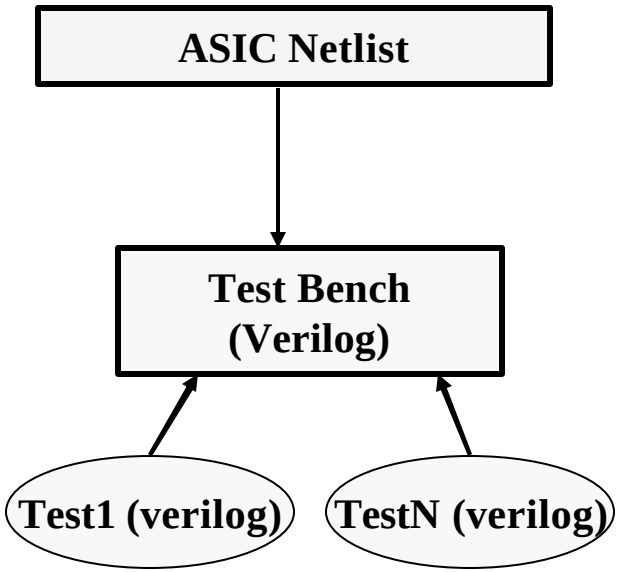
Thanks Toai!!!

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

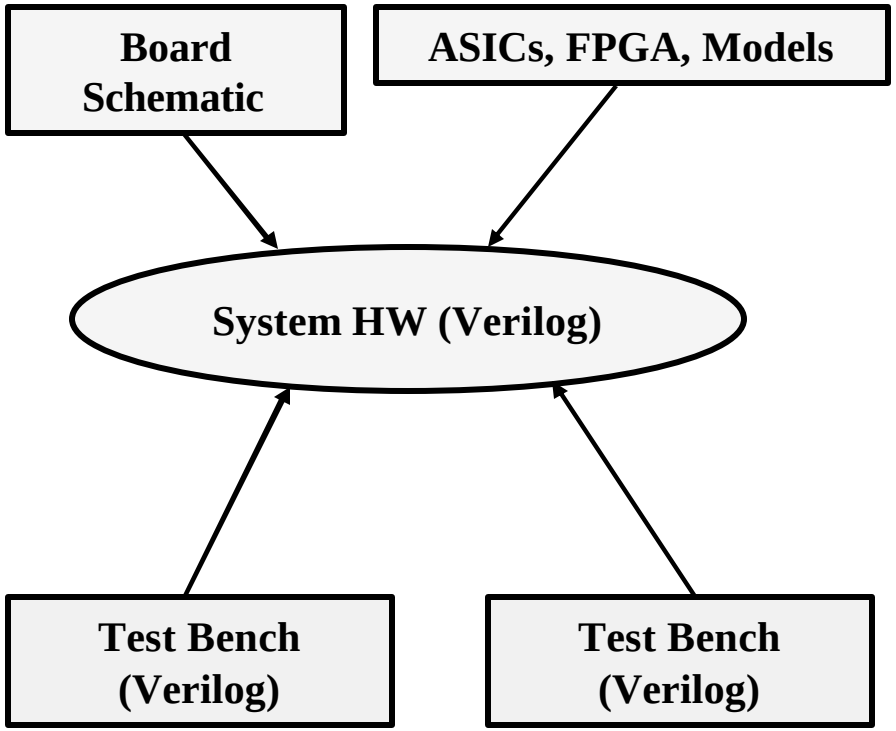
- **Interface between software and hardware verification environments**
  - **Software running on workstation**
  - **Hardware running in Verilog environment**
- **Primary use in the SoC environment**
  - **Embedded processor arrays**
- **Provides early, functional verification of hardware**
  - **Critical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

Current verification environment

ASIC Verification



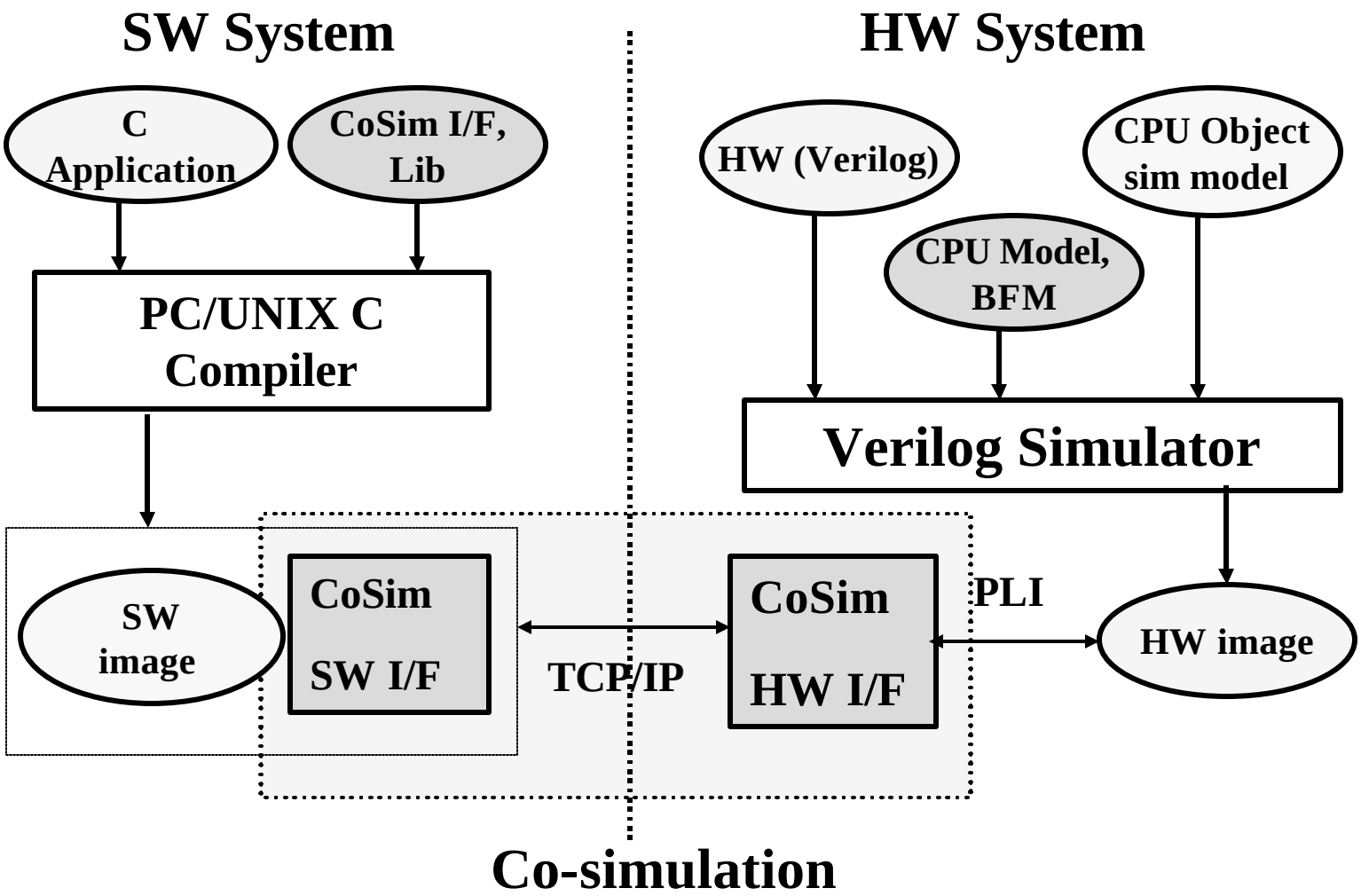
Board/System Verification



Current verification environment - problems

- 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

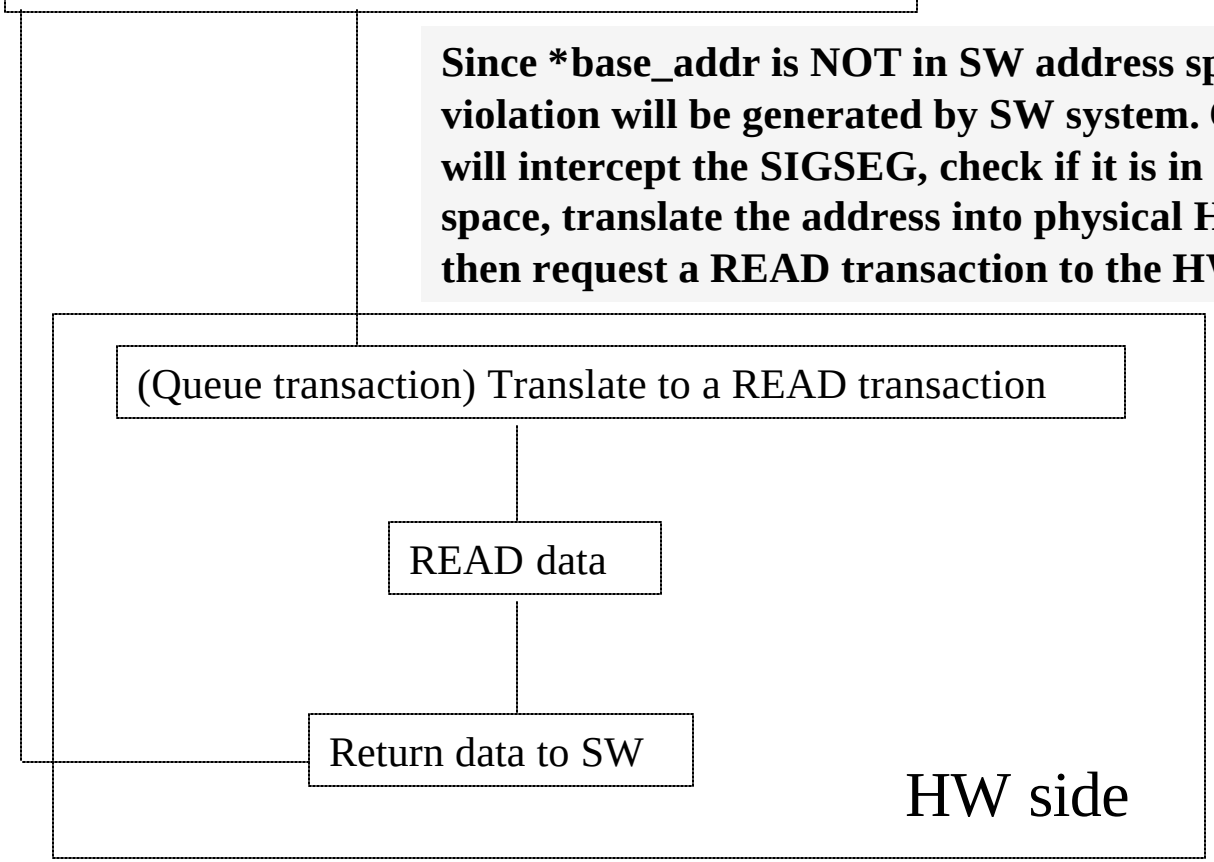
The Co-verification environment



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.



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.

(Queue transaction) WRITE

WRITE data

HW side

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**

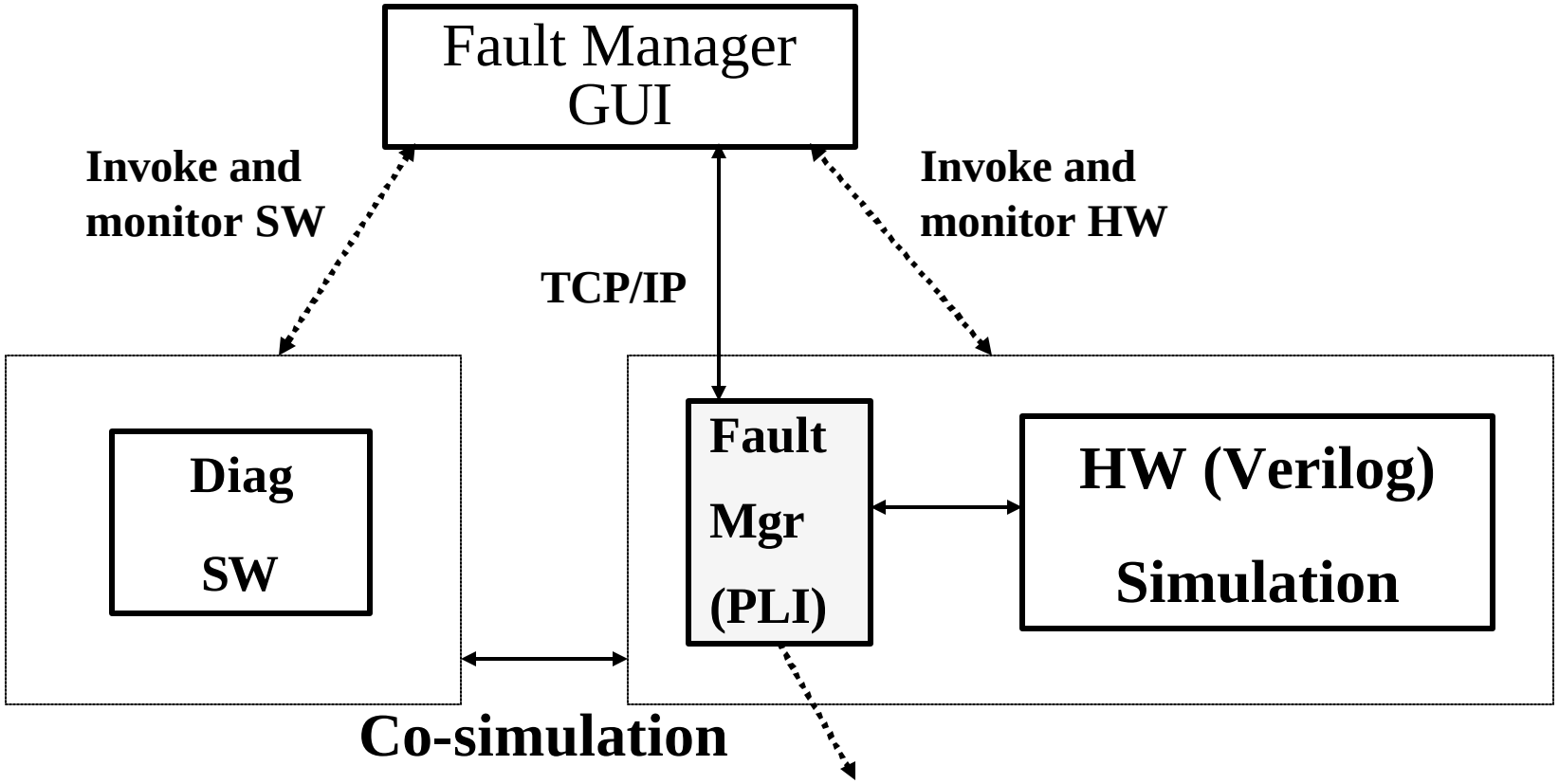
Fault Insertion: Why do it?

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

Fault Insertion: Current approaches

- **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

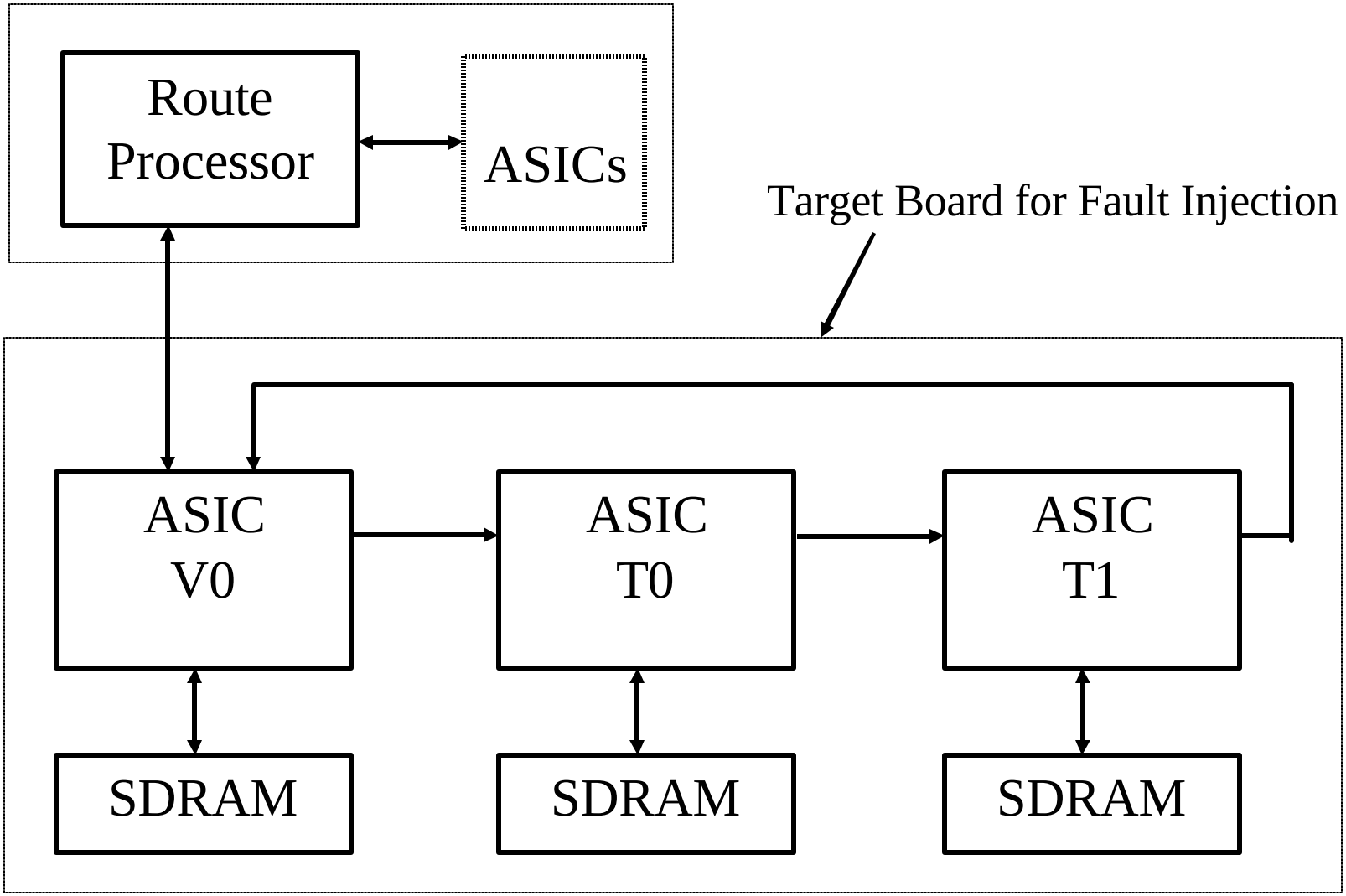


Perform: SA fault injection, generate fault list, board info, fault reports, save and restore HW, user commands, etc.,…

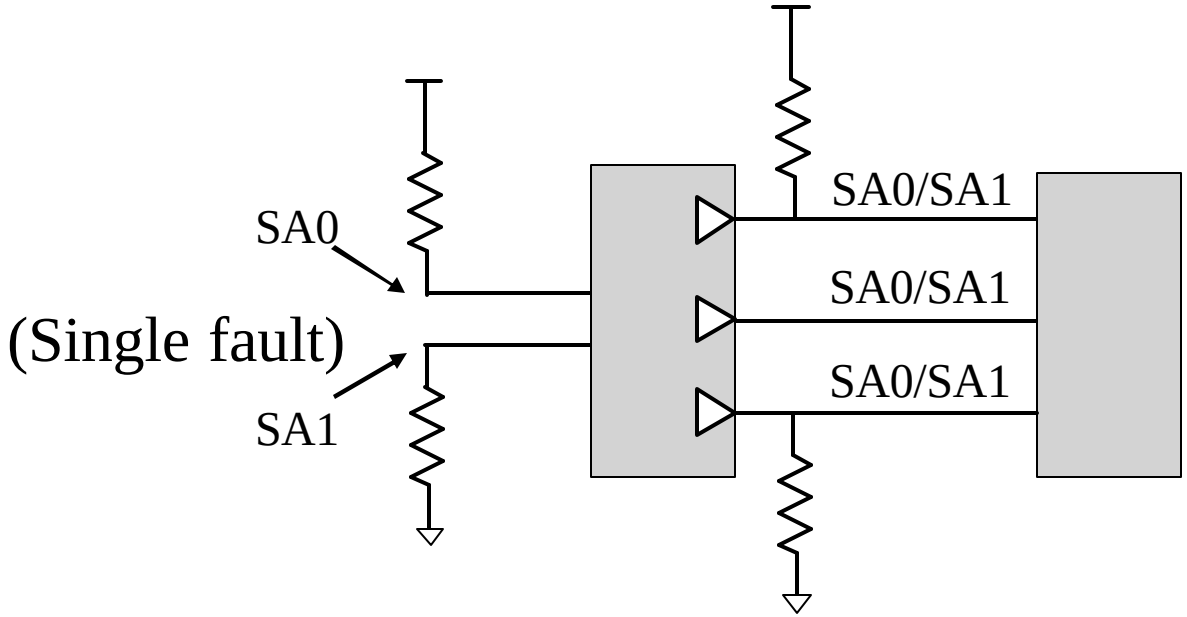
Test case

- **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



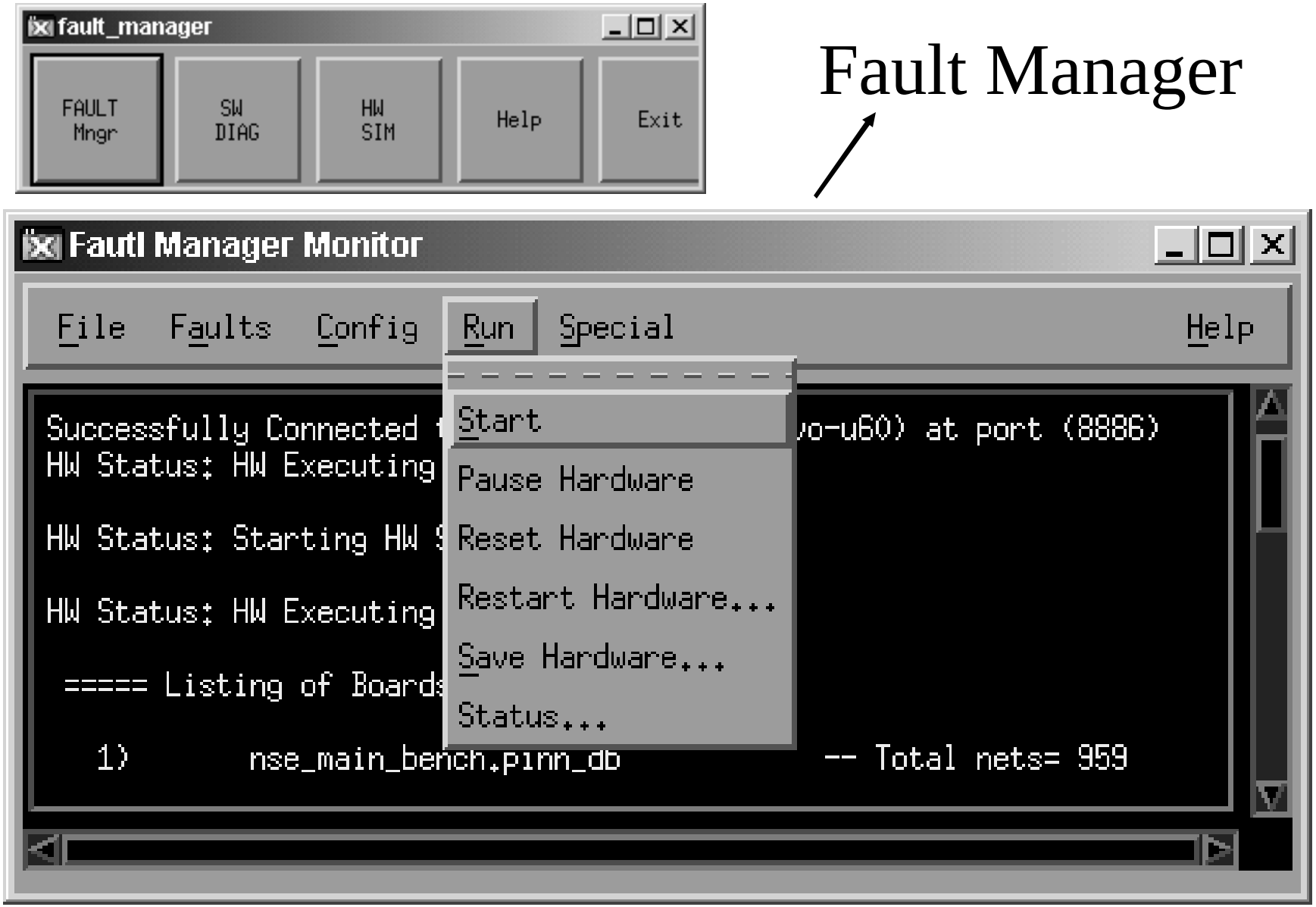
Fault Simulation Result

Total Nets	Single Flt Nets	Tot SA Flts	Det Flts	Flt Cov
1410	**251**	**2569**	**2543**	**99.00%**

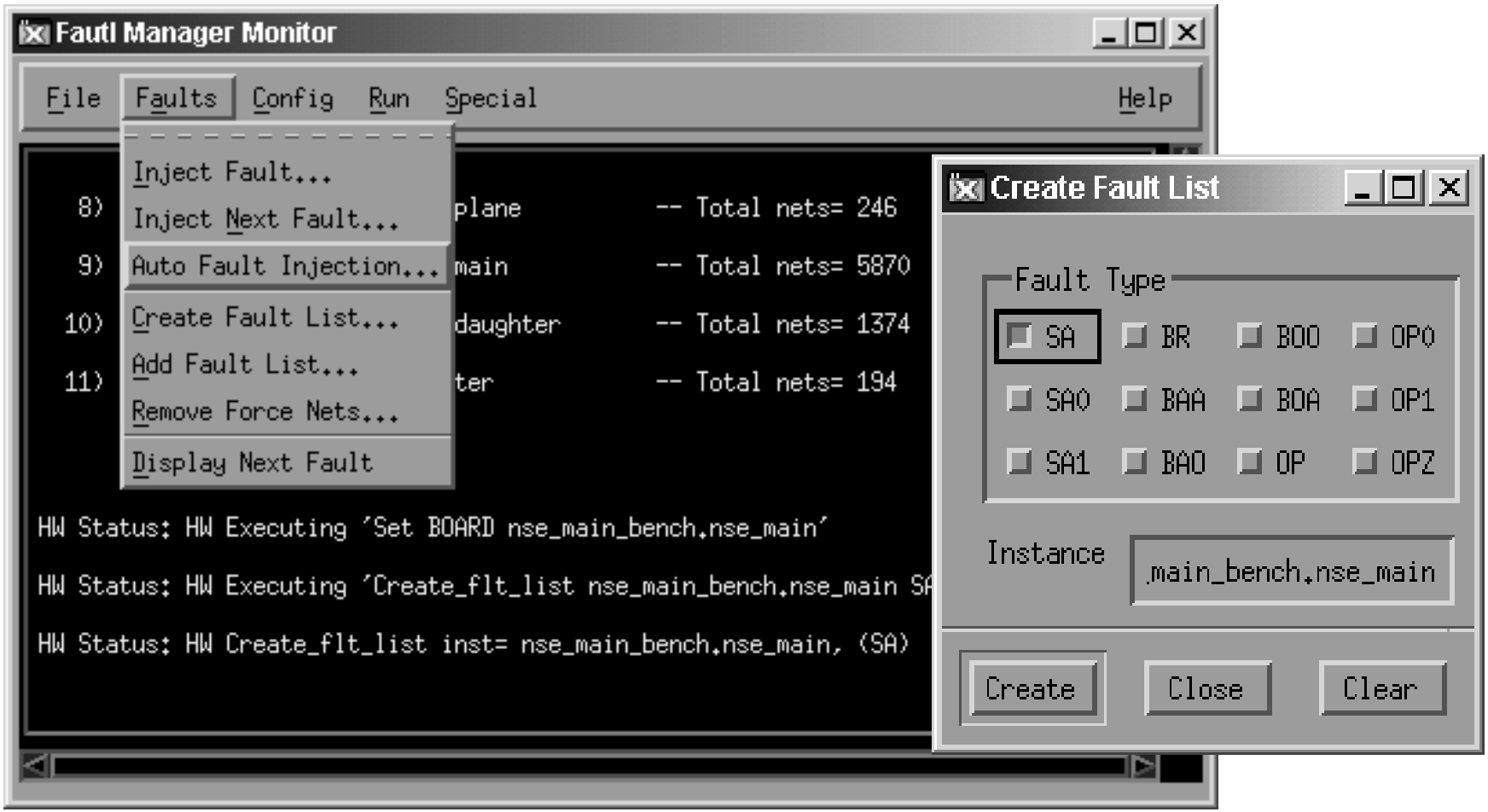
Undetected Faults

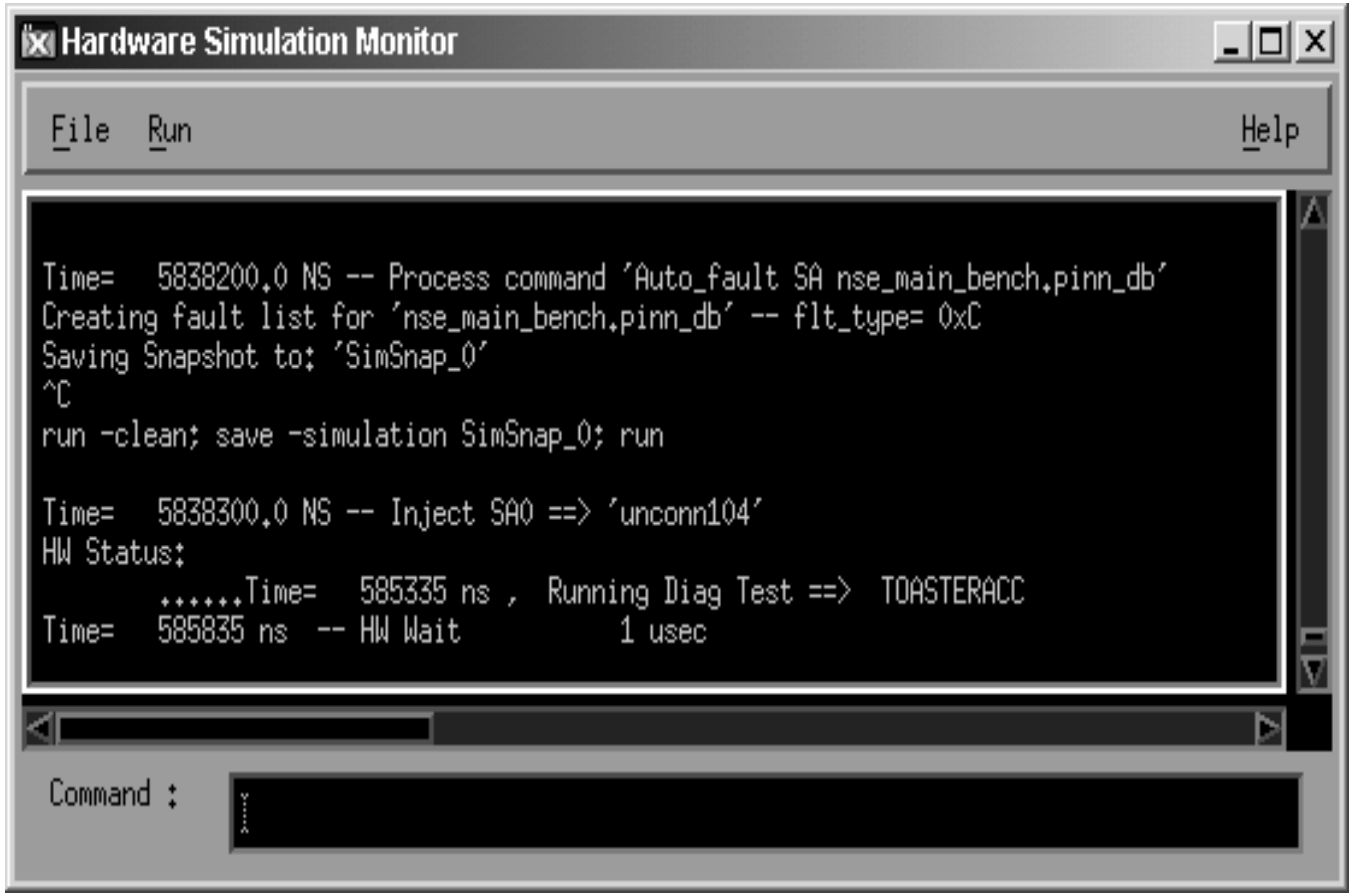
- Traffic control mode signal
- QoS flow control (not enough packets were sent)
- Some status and monitor signals
- Some optional ASIC debug enable/control signals

Fault Manager



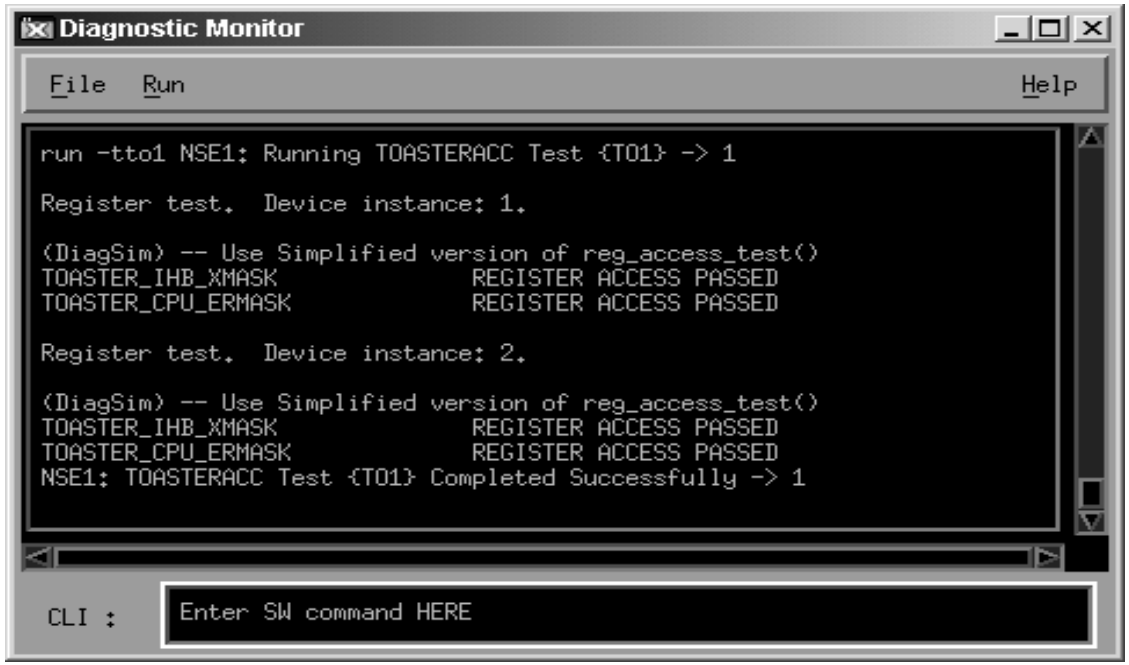
Fault Commands





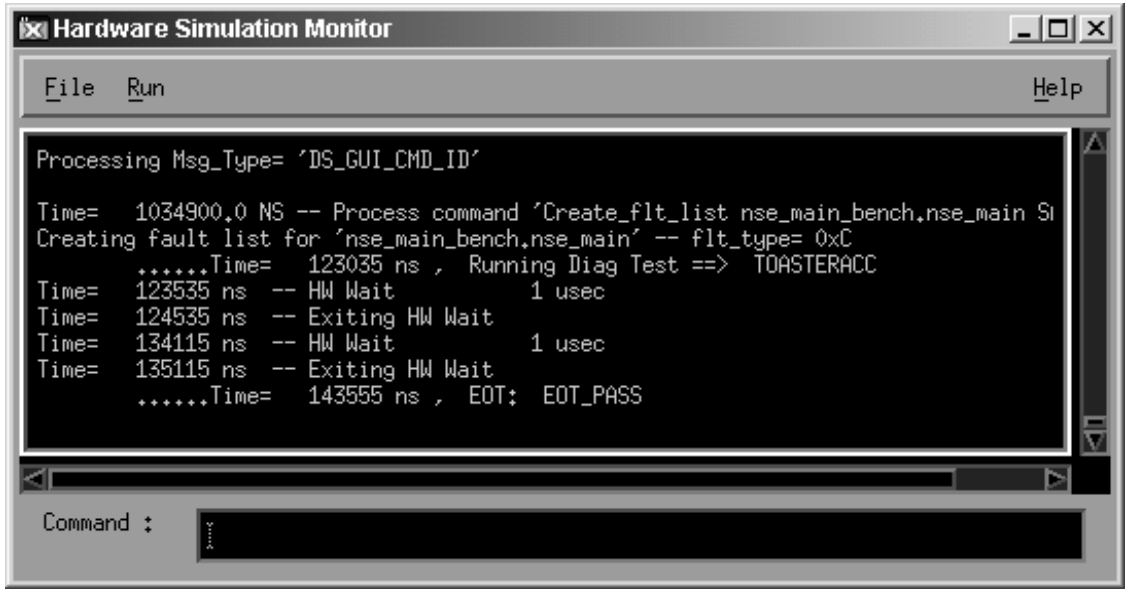
HW Sim:

Injected an Undetected SA0 fault



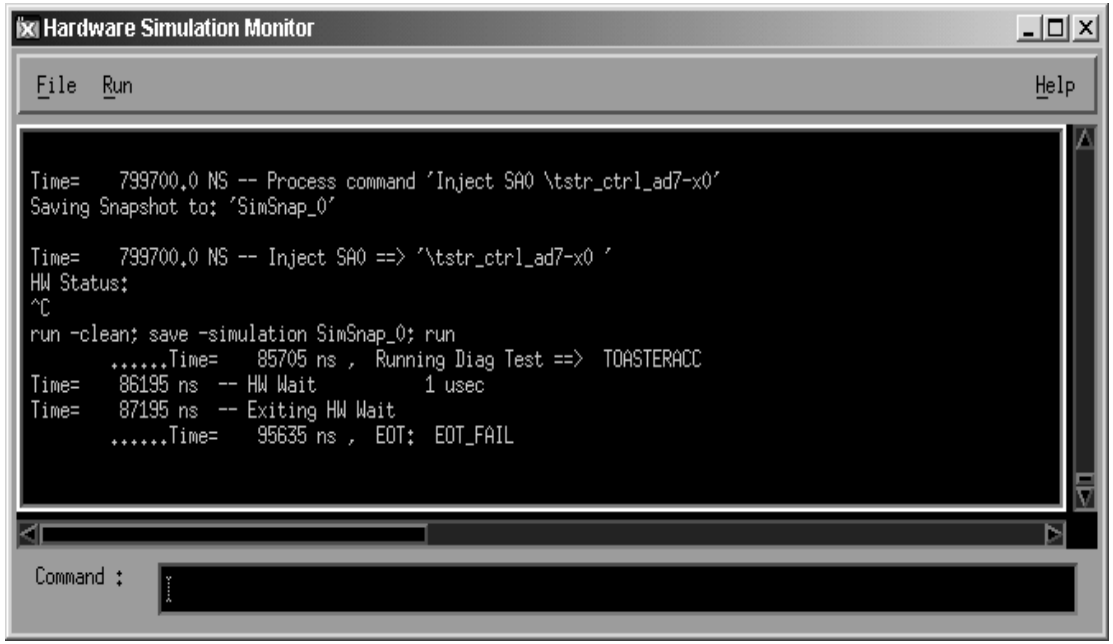
SW Run:

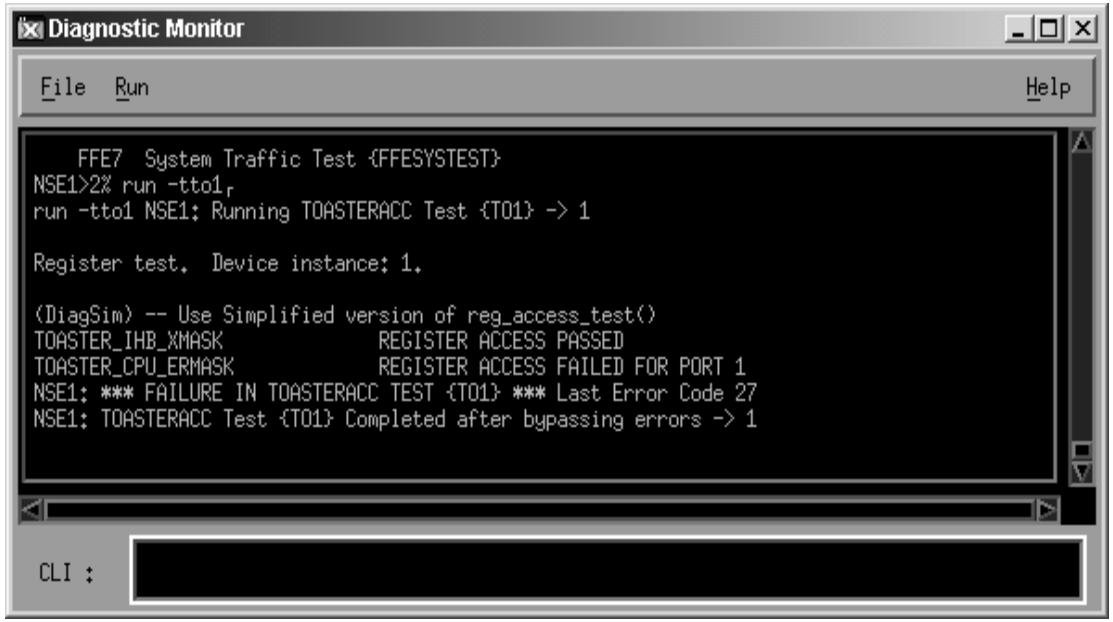
(Test PASS)



HW Sim:

(Test PASS)





<u>HW Sim:</u>

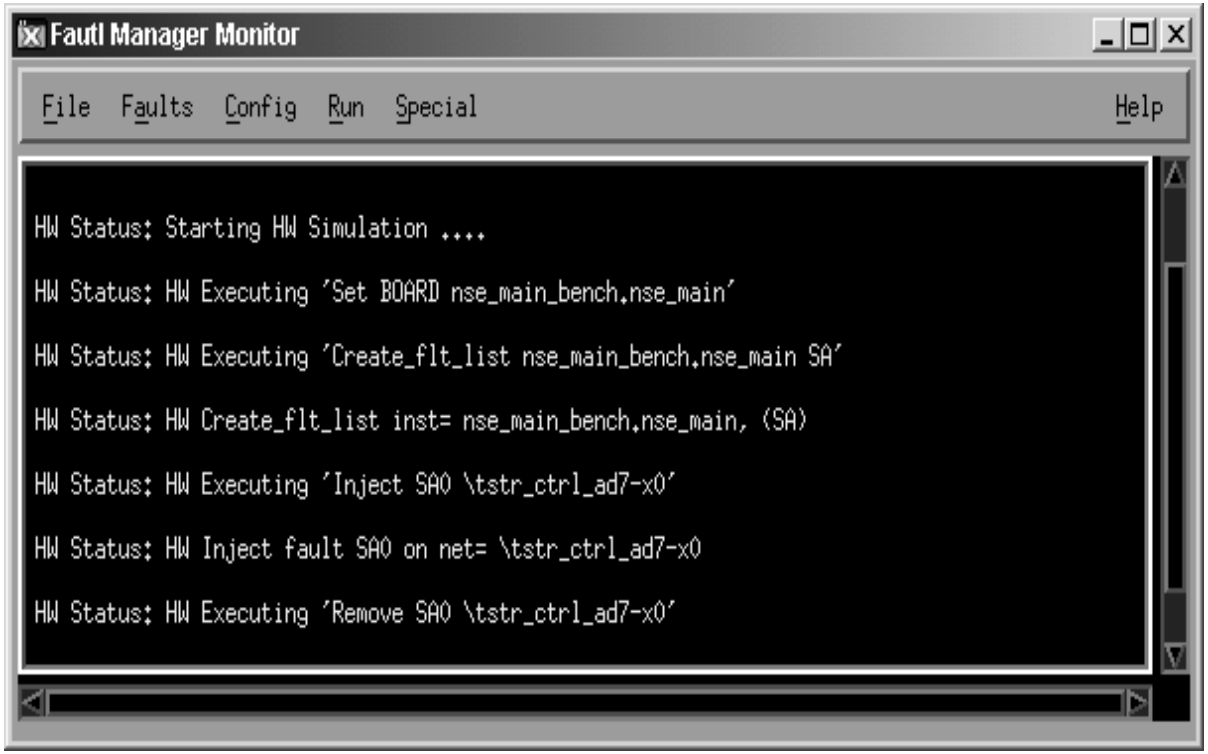
Inject SA0 fault

(Test Failed) --

Fault is detected

<u>SW Run:</u>

(Test Failed)



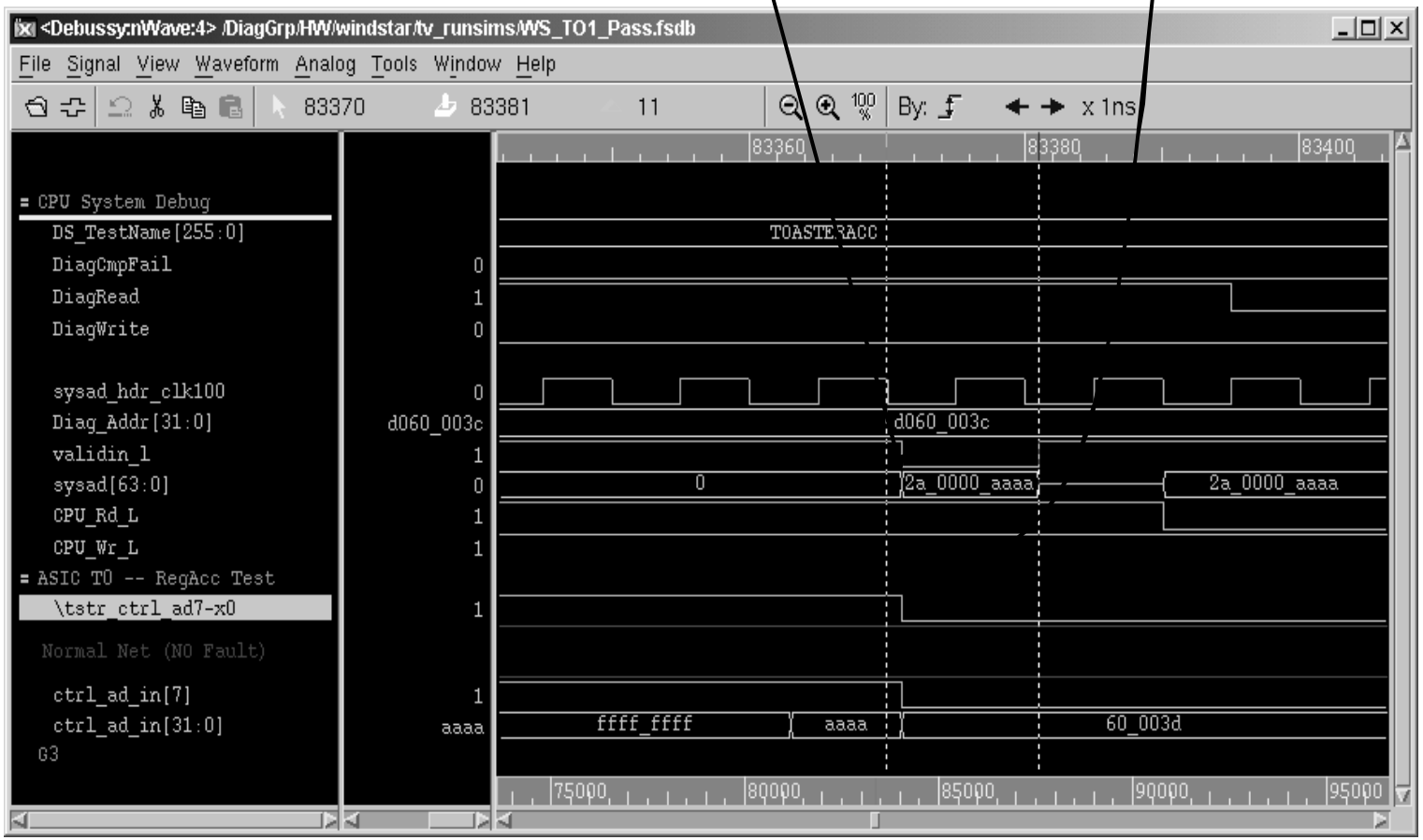
Fault Manager:

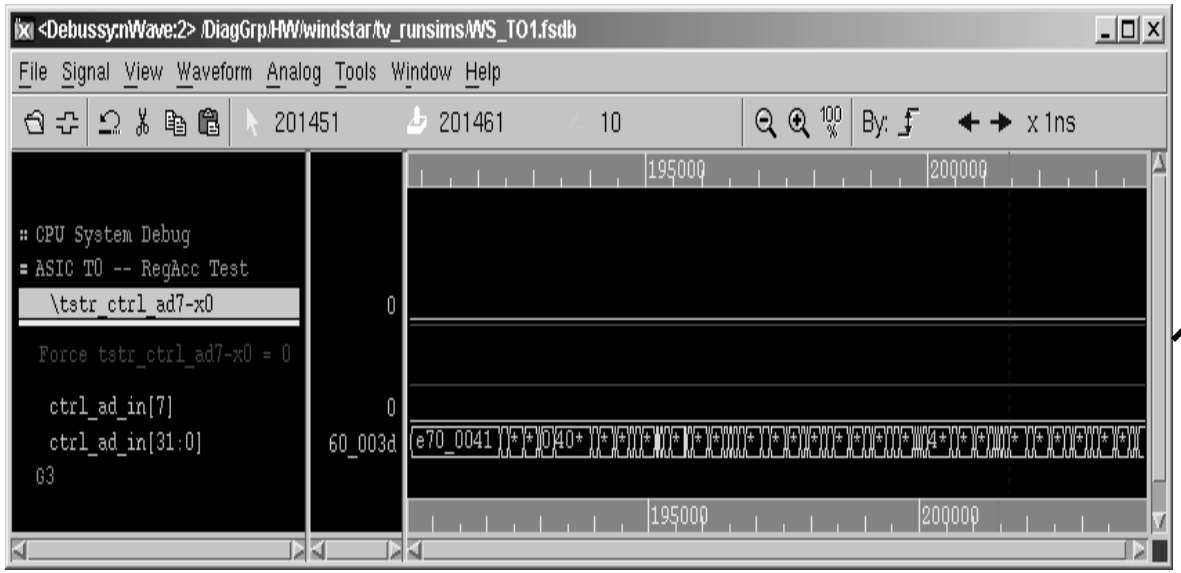
Log Messages

No Fault -- Net \tstr_ctrl_ad7-x0

Read data = 2a_0000_aaaa

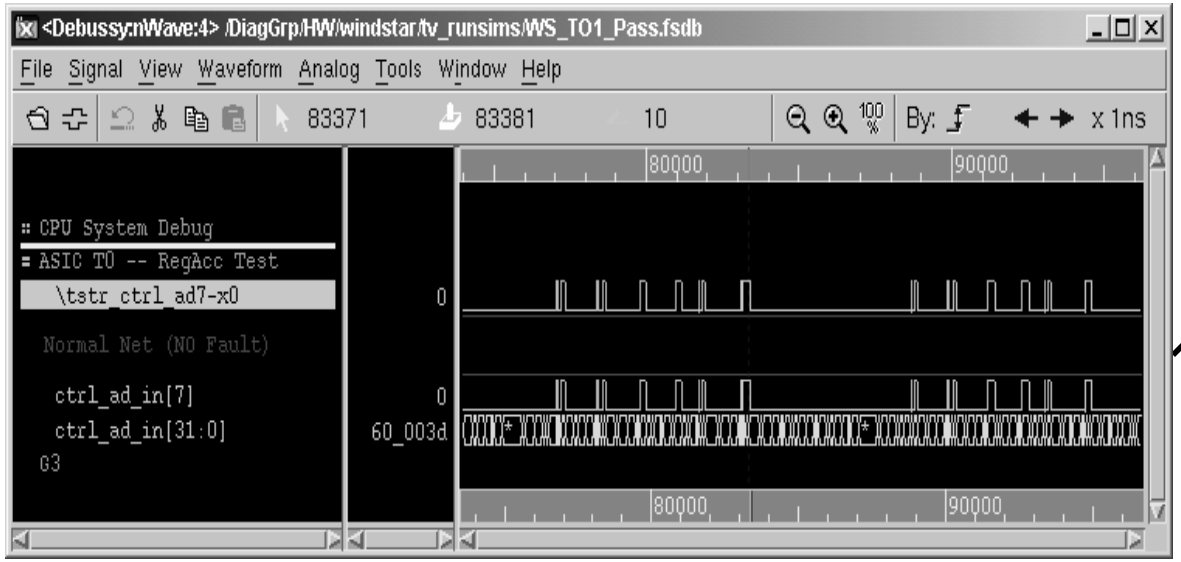
net: \tstr_ctrl_ad7-x0





Fault Sim SA0 net:

\tstr_ctrl_ad7-x0

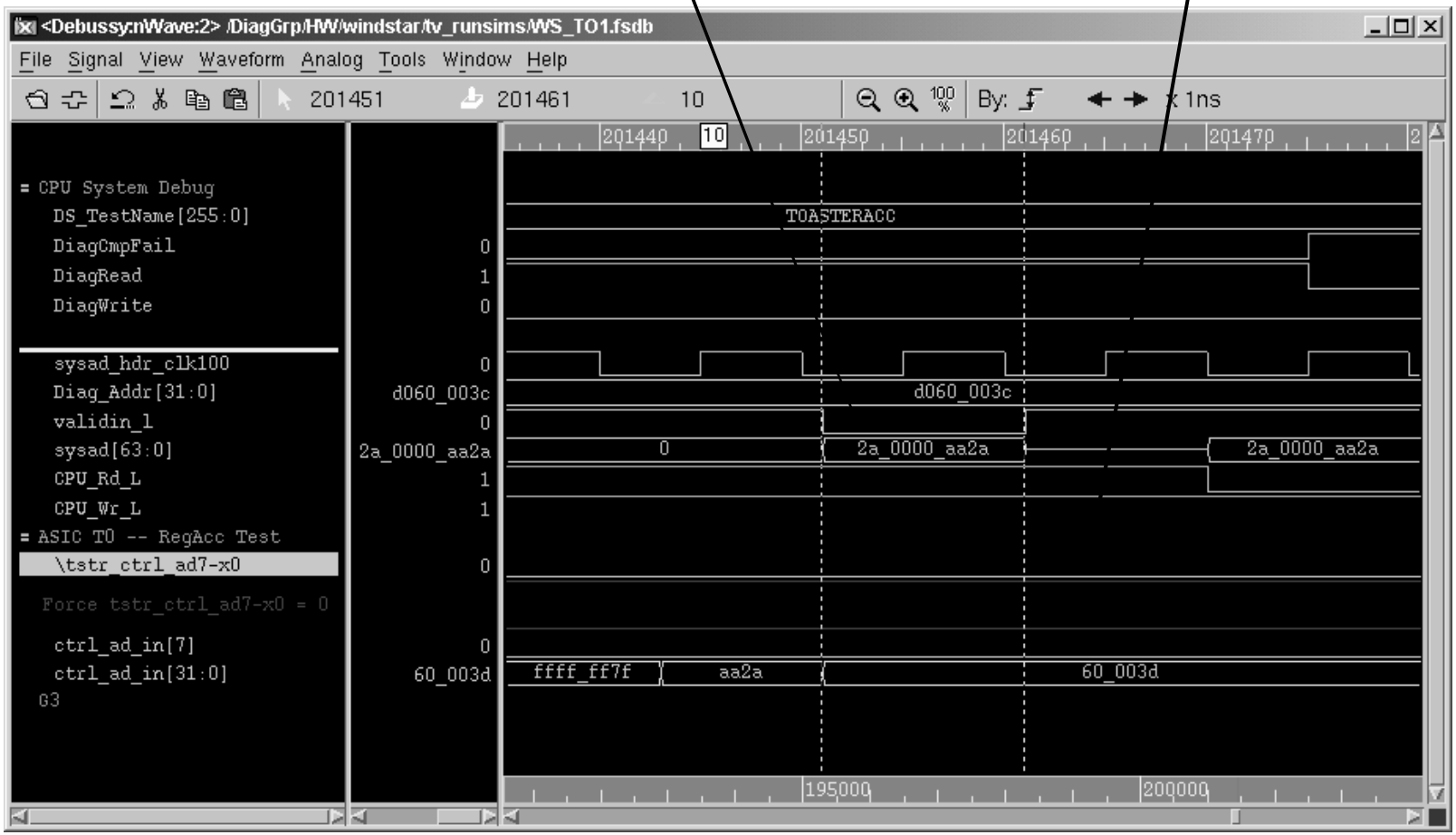


No Fault Sim net:

\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!**