

Test technology of FPGA-based safety I&C system of nuclear power plant

Xu Zhan
2015.10.13

Outline

- FPGA test background
- FPGA test process
- FPGA verification methodology
- FPGA test benches
- FPGA test application

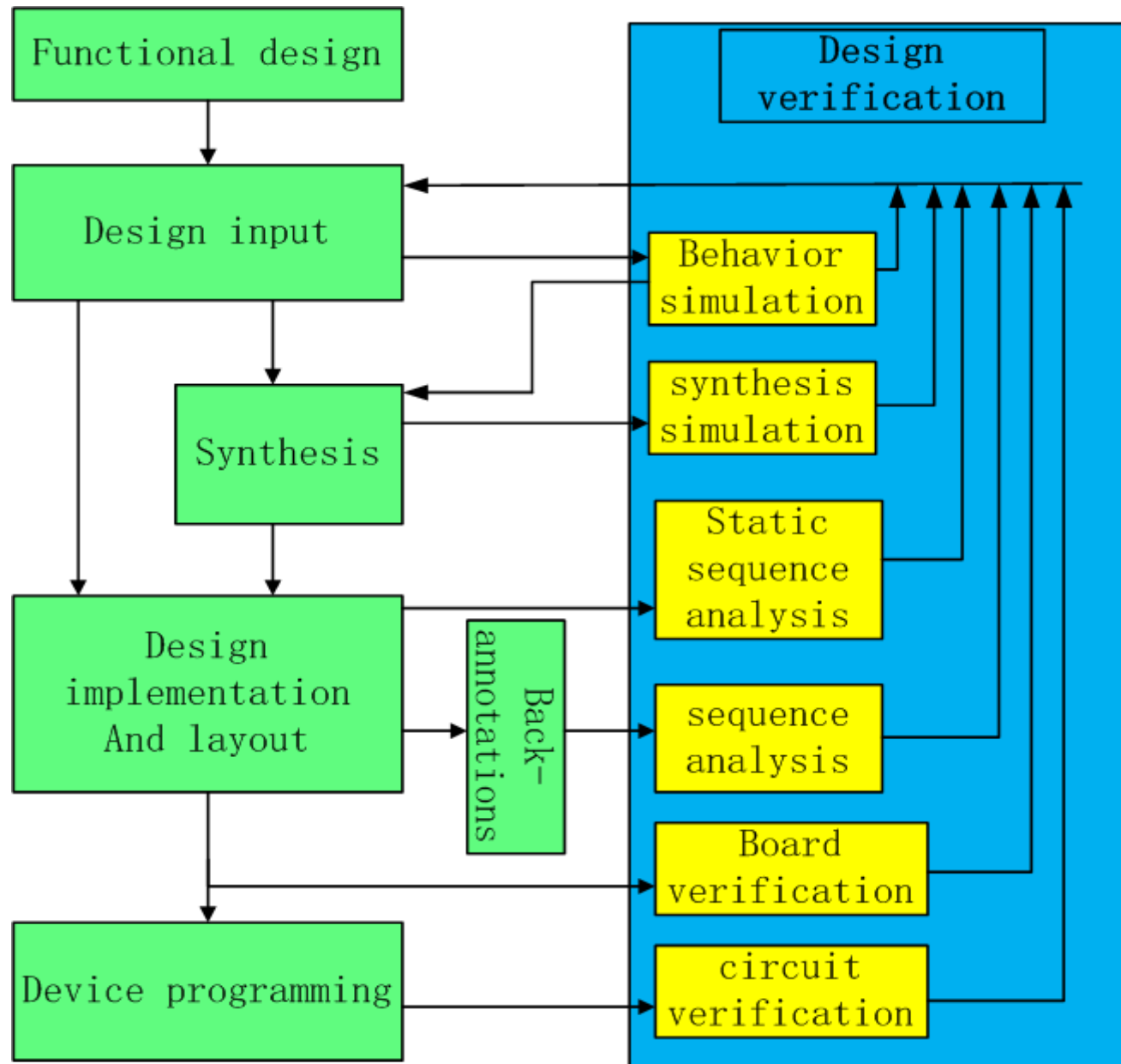


FPGA test background

- Application on FPGA-based safety I &C system of NPP
- For FPGA-based I&C systems, the testing activity needed satisfy the verification objectives of IEC62566 or NUREG CR7006 for safety application.
- With the complexity of FPGA design increasing, the FPGA testing confront significant challenges :
 - Verification consumes more than 70% of resources
 - Time to market affected
 - Bugs remain undetected
 - Conventional simulation inadequate
 - Better approaches needed



FPGA test process



Test approach

Directed test :

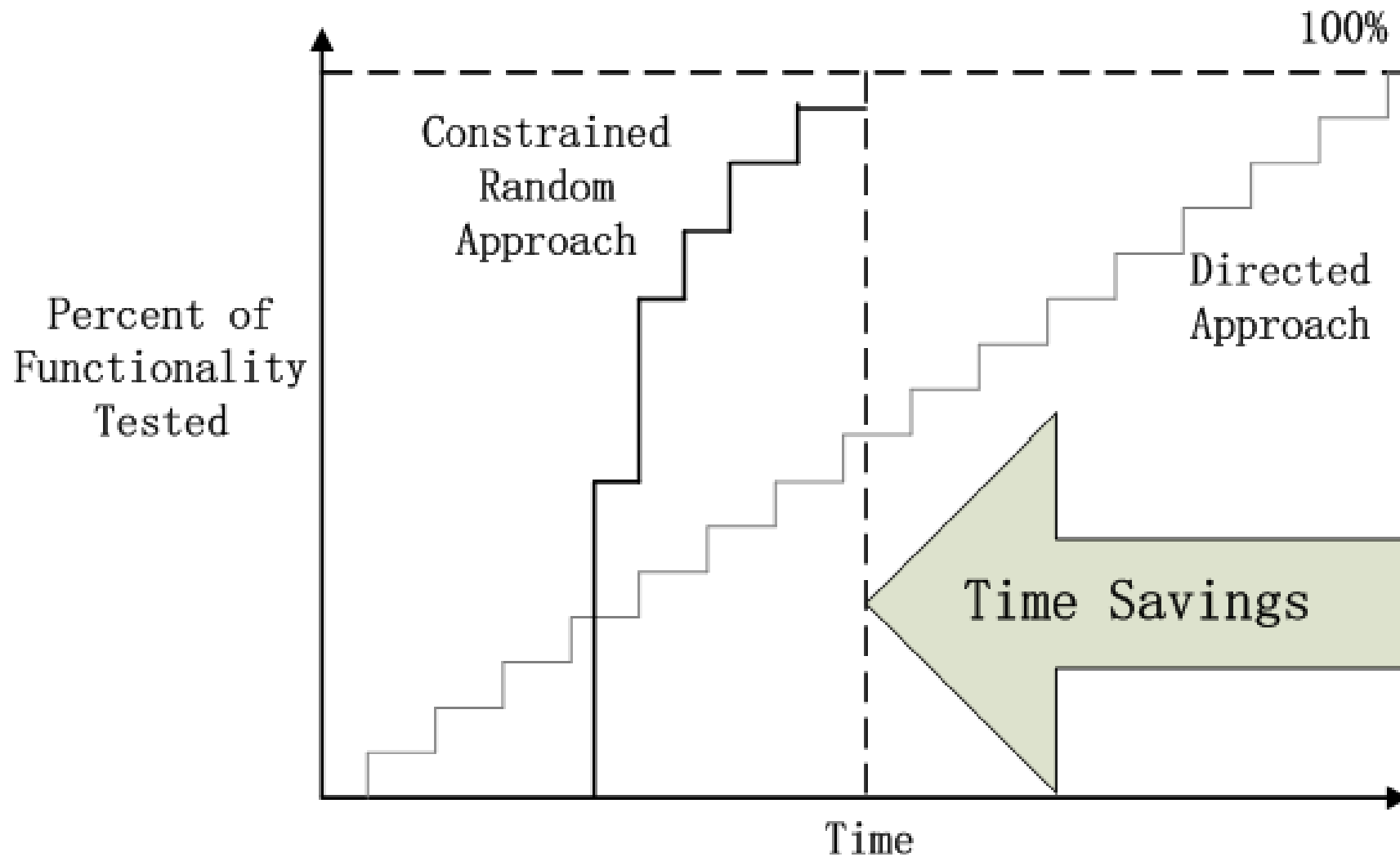
- Testbenches without randomness, targeting a specific item in the verification plan.
- If the design is complex enough, it is impossible to cover all features with directed testbenches.

Random verification:

- 1) Generate random tests using random constrained stimuli generation.
- 2) Check for bugs and correct them if there are
- 3) Check for the coverage values. If not satisfying, add constraints and repeat from 1.



Efficient test approach



What is verification methodology?

The methodology for constructing a software verification platform and providing the reusable verification platform.

Why verification methodology can be used?

- 1) Not to re-develop new solutions
- 2) Not to spend time writing new code
- 3) To increase testing quality, more stable and reliable

Focus on

- Random constrained stimuli generation
- Assertions
- Functional coverage



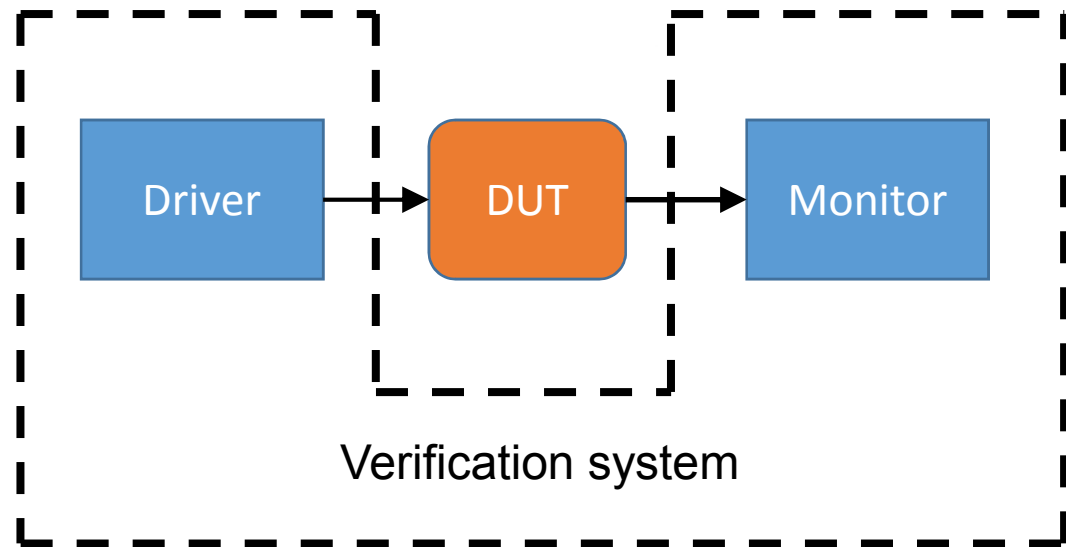
Conventional verification system

DUT: Design Under Test

- Time based simulation
- Cycle based simulation
- Simple design

Defect:

- Not scalability
- Not reusability



Layered verification system

From bottom to top

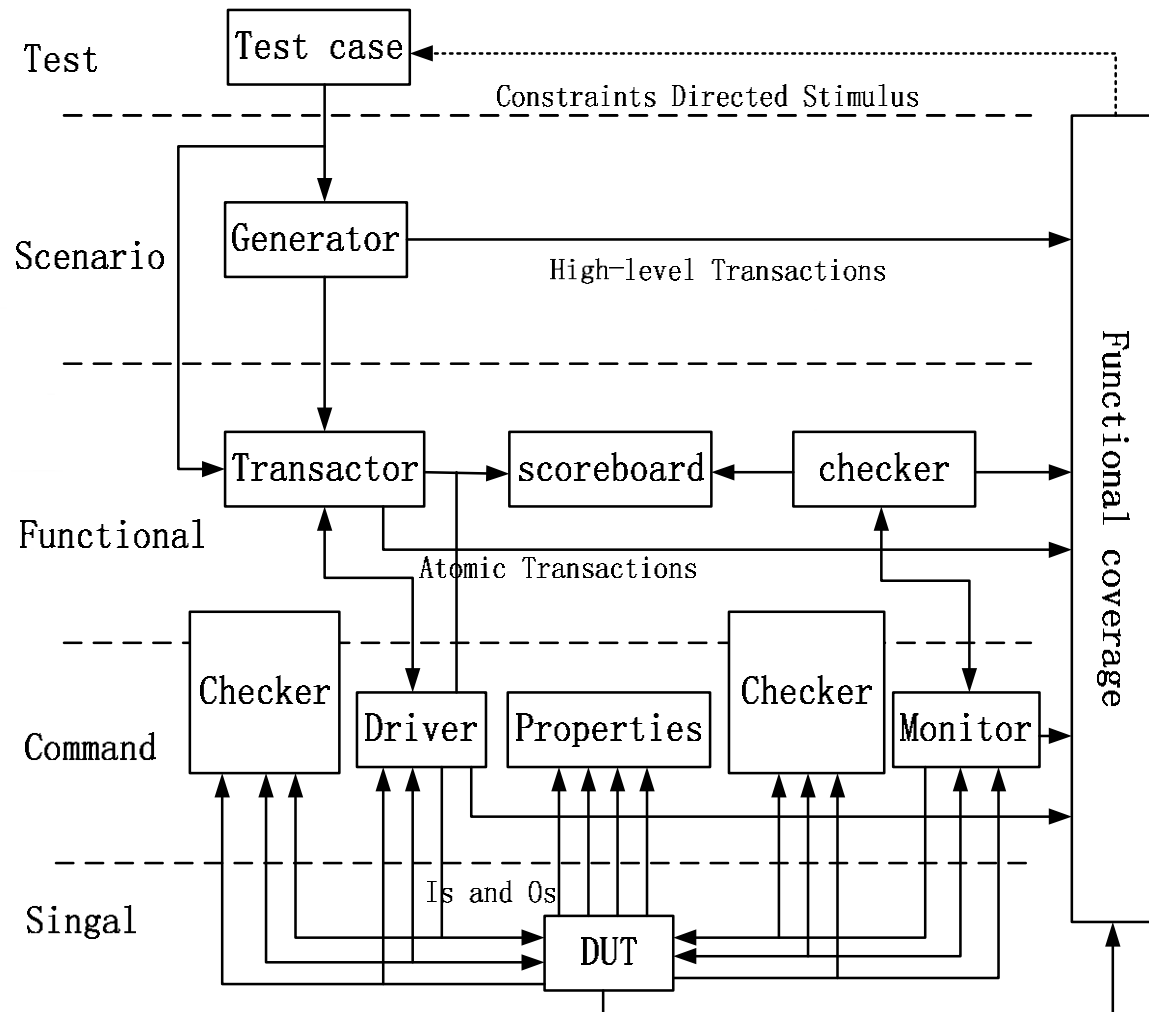
1) Signal layer

2) Command layer

3) Functional layer

4) Scenario layer

5) Test layer



Random constrained stimuli generation

- Define random variables and constraints
- Ask the random solver to find a random set of variables that satisfies the constraints
- Constraints can be added, disabled to create different tests.
- It would be hard to mark a routine to randomly generate one of the legal combinations using only direct randomization of variables



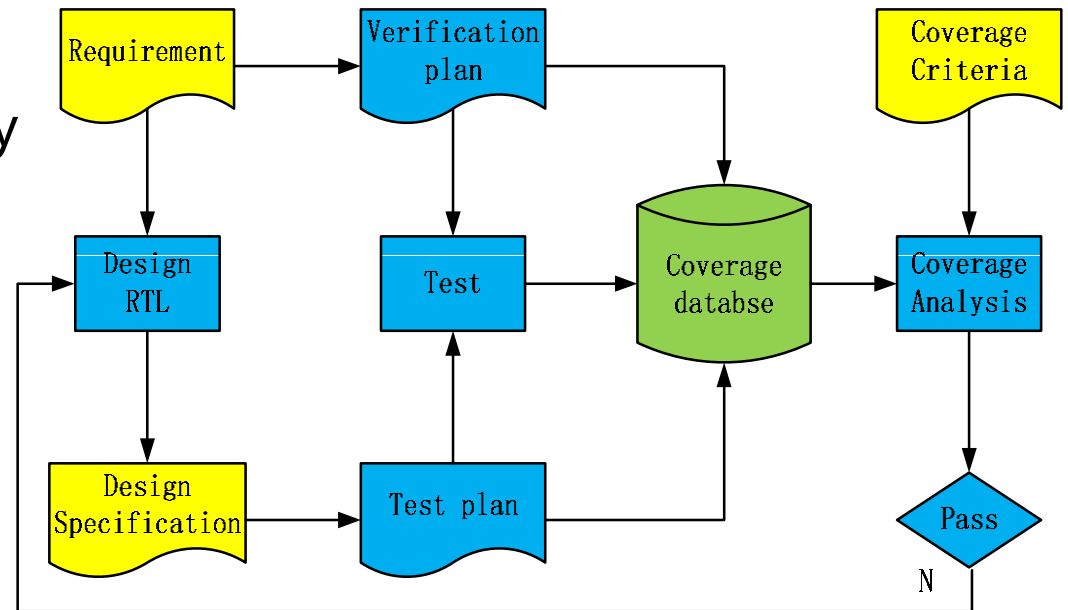
Assertions

- Tools for automatic checking of properties
- prove how the design behavior can meet the requirements
- Assertions are instantiated similarly to other design blocks and are active for the entire simulation
- The simulator keeps track of what assertions have triggered, and so you can gather functional coverage data on them

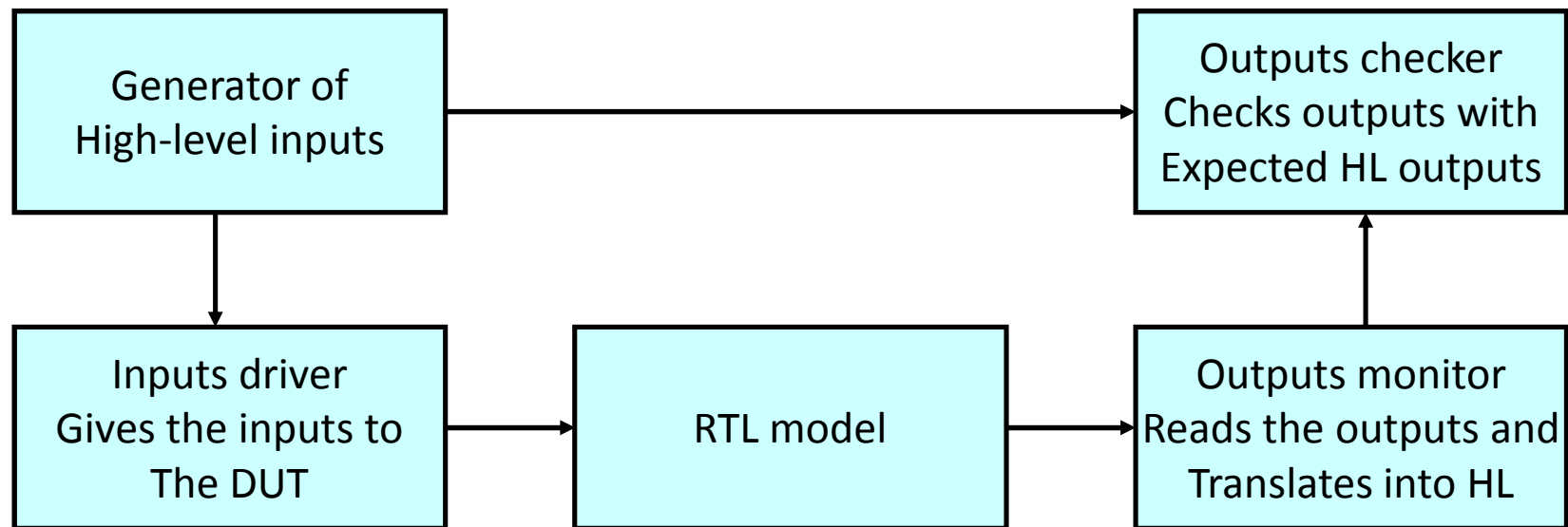


Test coverage

- RTL code coverage
 - Statement, decision, condition, path, toggle, triggering, FSM
 - 100% of statement, decision and condition, path
- Quality metrics
 - Bug detection frequency
 - Length of simulation
 - Simulation Minimize
- Functional coverage
- Requirement coverage
- Regression test
 - Quantitative stopping criterion
 - Test more but simulate less



Test benches structure



Test benches

It is important to conceptually divide testbenches into blocks, depending on the function:

- Generator of high-level input data
- Driver: read the high-level input data and drive the DUT input ports.
- Output monitor: read data from the DUT's output ports.
- Output checking: check correctness of the output

This allows easier readability and reuse. If the DUT input protocol change, only the driver must change.

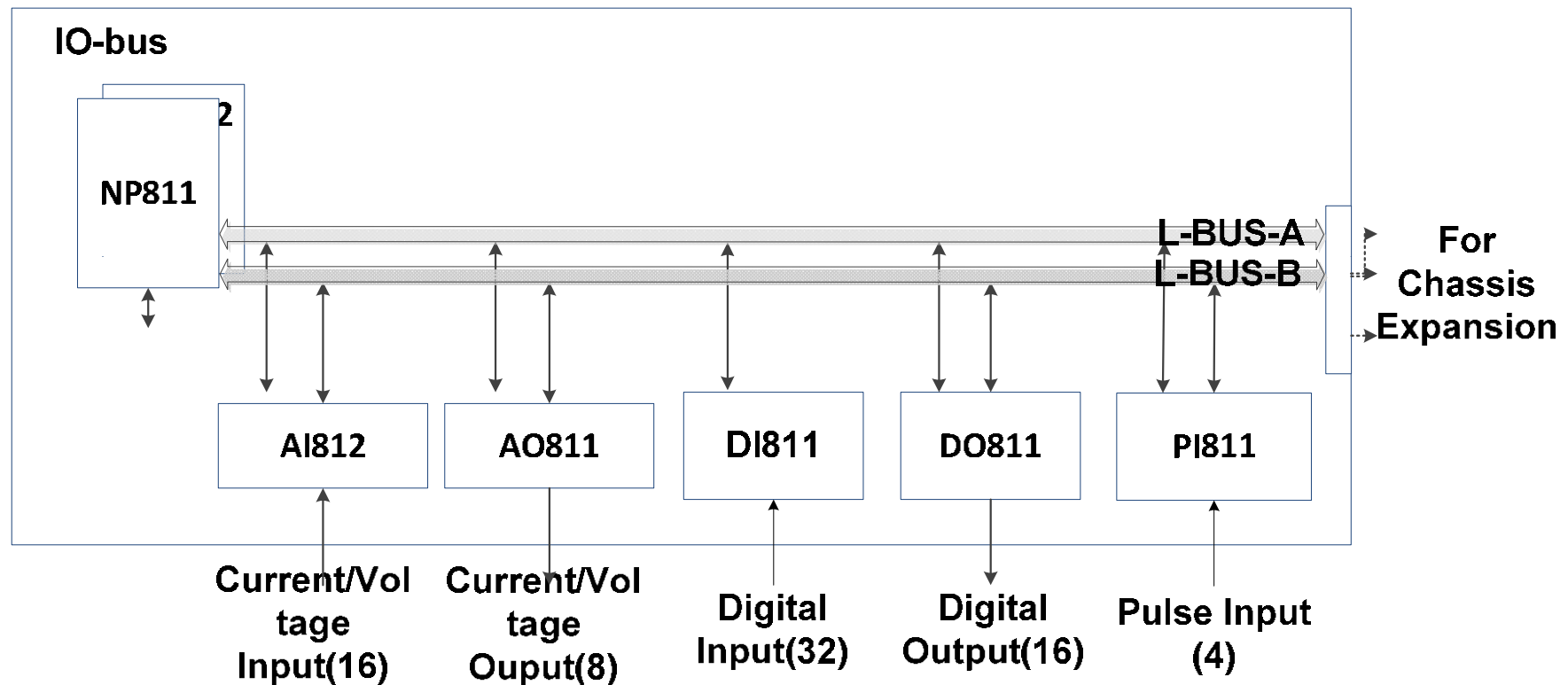
This requirement is most important for big systems, with a lot of reuse and efficiency.



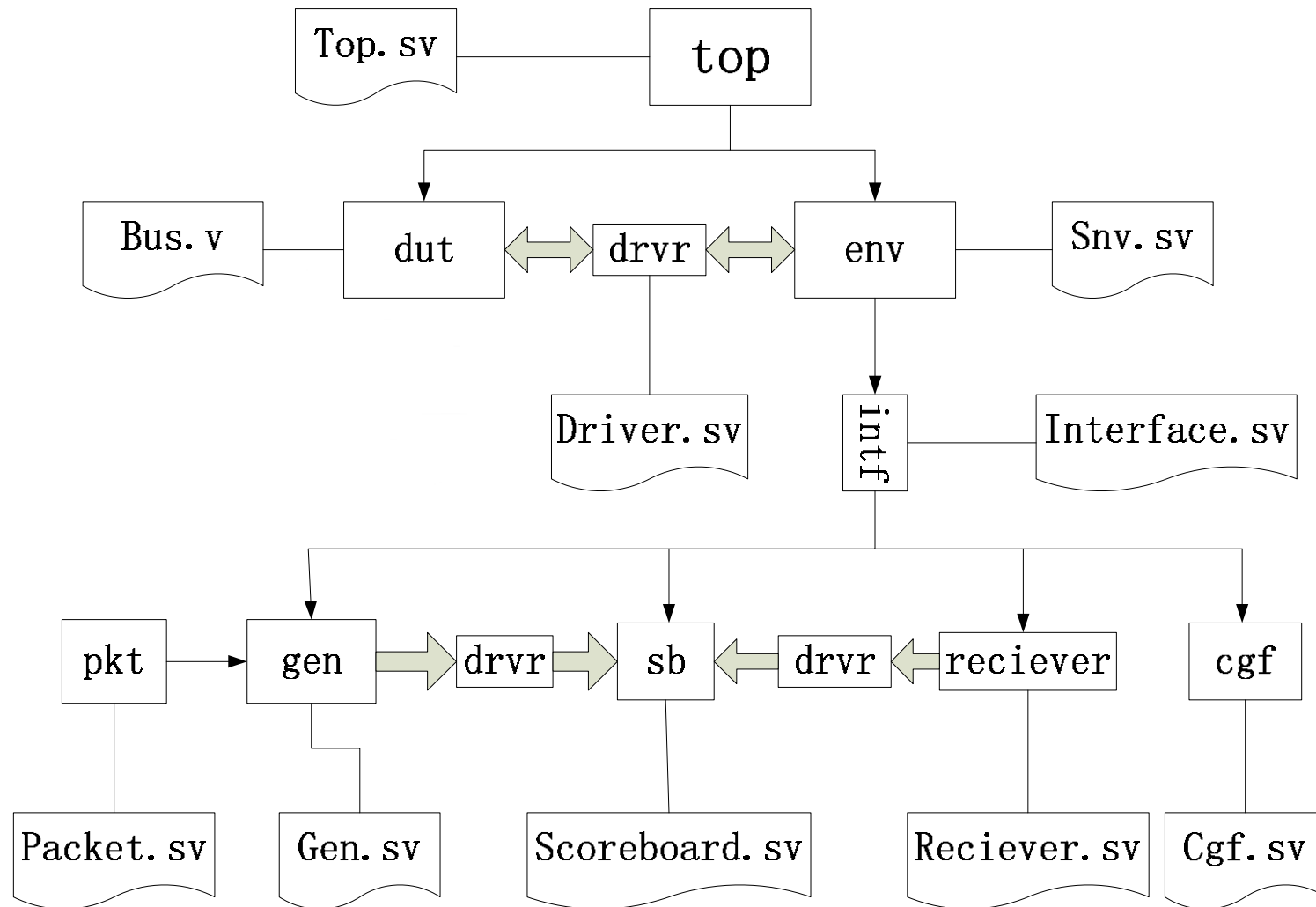
FPGA Test application

Simulation verification tool: Medolsim10.0b

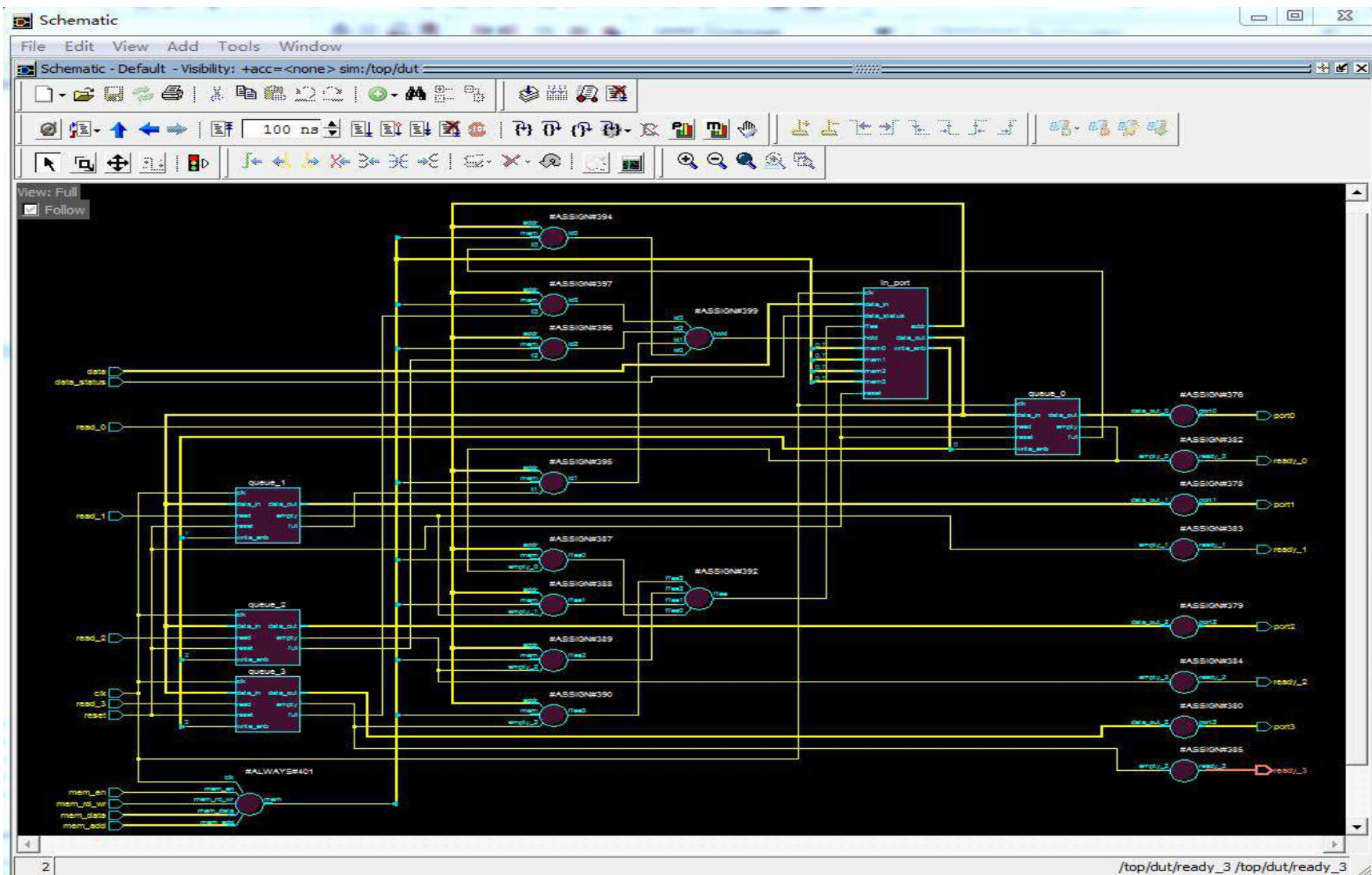
DUT: IO-bus



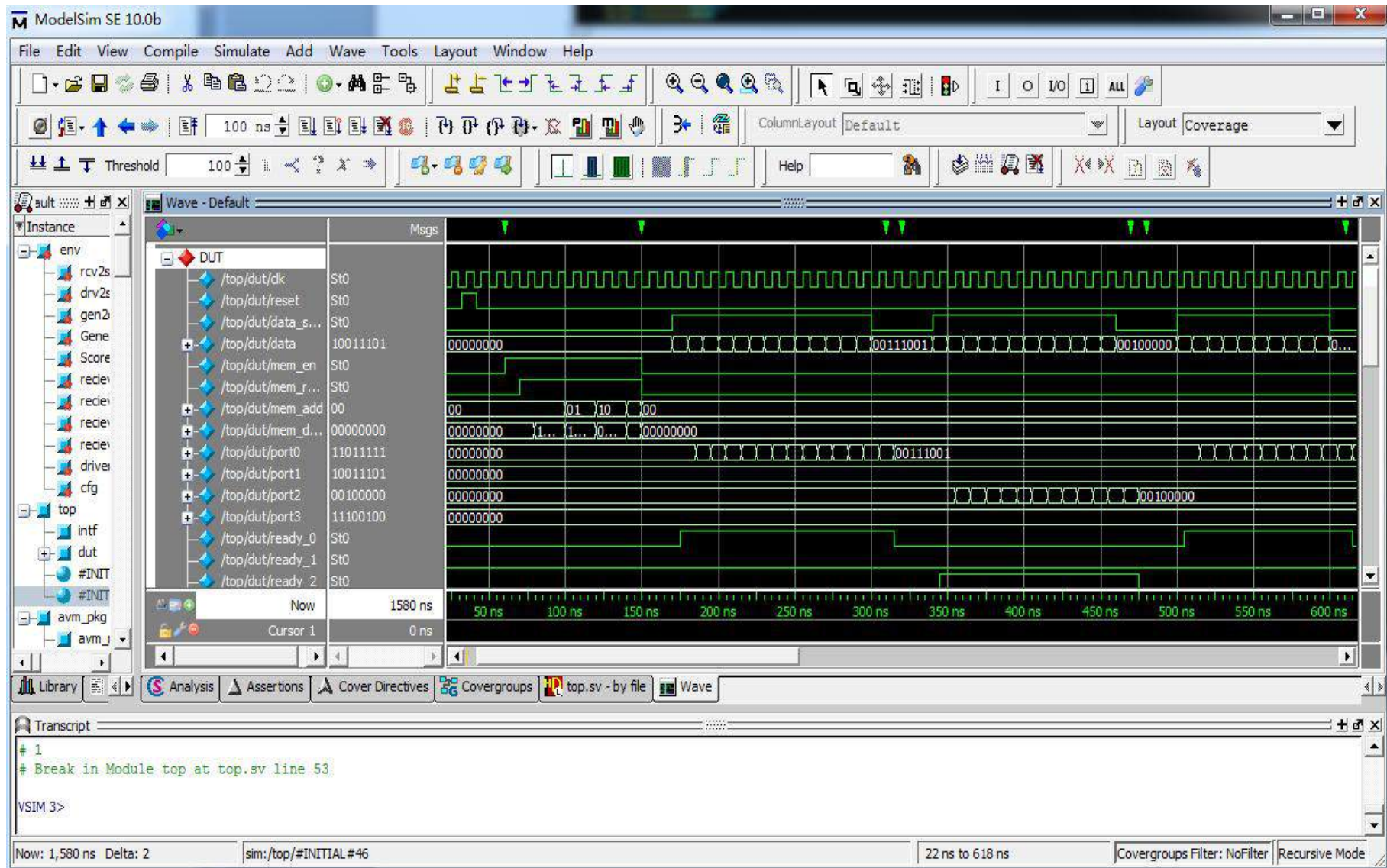
Verification platform



Schematic diagram



Simulation wave



Data flow

The screenshot displays the ModelSim SE 10.0b interface with a dataflow diagram for a Verilog module. The diagram shows the following components and their data flow:

- Instance List (Left):** env, rcv2sb, drv2sb, gen2drv, Generat, Scoreboz, reciever, reciever, reciever, driver, cfg, top, intf, dut, #INITIAL, #INITIAL, avm_pkg.
- Dataflow Diagram (Center):**
 - #ALWAYS#154(fsm_core):** Receives state_r (0000), data_status (S0), and fsm (S1). It outputs state (0000), sus_data_in (0), and addr (10100110).
 - #ALWAYS#150(fsm_state):** Receives clk (S1) and state (0000). It outputs state_r (0000).
 - #ALWAYS#117(addr_mux):** Receives data_status (S0), data_in (1001101), and fsm_write_end (0). It outputs write_end_r (0010).
 - #ASSIGN#251:** Receives write_end_r (0010) and fsm_write_end (0). It outputs write_end (0000).
 - Memory Blocks (mem0-mem3):** mem0 (01100101), mem1 (00100011), mem2 (10100110), mem3 (11011011).
 - Other Signals:** data_out (10011101), read (0010), and various bus signals.
- Transcript (Bottom):**

```
# Break in Module top at top.sv line 53
add dataflow -r /*
VSIM 4>
```
- Status Bar (Bottom):** Now: 1,580 ns Delta: 2 sim:/top Keep 0 Covergroups Filter: NoFilter Recursive Mode

Coverage

The screenshot displays the ModelSim SE 10.0b interface with the Instance Coverage window open. The coverage table shows the following data:

Instance	Design unit	Design unit type	Stmt count	Stmts hit	Stmts missed	Stmt %	Stmt graph
/top/dut	switch	Module	12	12	0	100%	
/top/dut/queue_2	fifo	Module	15	14	1	93.3%	
/top/dut/queue_3	fifo	Module	15	14	1	93.3%	
/top/dut/queue_1	fifo	Module	15	14	1	93.3%	
/top/dut/queue_0	fifo	Module	15	14	1	93.3%	
/top/dut/in_port	port_fsm	Module	66	49	17	74.2%	

The Wave - Default window shows a hierarchical view of the DUT signals and their corresponding waveforms. The signals listed include: /top/dut/clk, /top/dut/reset, /top/dut/data_status, /top/dut/data, /top/dut/mem_en, /top/dut/mem_rd_wr, /top/dut/mem_add, /top/dut/mem_data, /top/dut/port0, /top/dut/port1, /top/dut/port2, /top/dut/port3, /top/dut/ready_0, /top/dut/ready_1, /top/dut/ready_2, /top/dut/ready_3, /top/dut/read_0, and /top/dut/read_1.

The Transcript window shows the following warnings:

```
# Warning: Signal '/top/intf' either has no drivers or driver(s) not recognized by causality analysis
# Warning: Signal '/top/dut' either has no drivers or driver(s) not recognized by causality analysis
```

VSIM 55>

Now: 62,231,135 ns Delta: 0 sim:/top/dut Recursive Mo

Thank you for your attention!
Question & Comments?

