

Managing Deployment of SVA in your Project

Raj Dayal

DVClub: RTP, North Carolina

June 20th, 2007

Raj Dayal



- Raj is currently working for Qualcomm Corp. Processor Solutions Group in Cary , NC where he is responsible for Processor Verification, tools and Methodologies development.
- Prior to joining Qualcomm, Raj held a number of Design and Verification Team Lead positions at
 - Ivivity (Storage Processor startup in Atlanta), Verification Manager.
 - Vitesse Semiconductor – (Internet Traffic Management QoS - startup Orologic) Verification Lead.
 - Broadband Technologies – (Fiber to Curb SONET Frammer), Lead Design Engineer
 - Avant! (now Synopsys) – CAD Tools Developer.
 - IBM PC Company in RTP Area – Device Driver Development.
- Raj worked as an ASIC Design Engineer in IBM Interconnect Division in Mid Hudson Valley, upstate New York.
- Raj has lectured a few courses in C Programming and LAN in Wake County Community College.
- Raj received his MSEE from Syracuse University and BSEE from NJ Institute of Technology.

Agenda

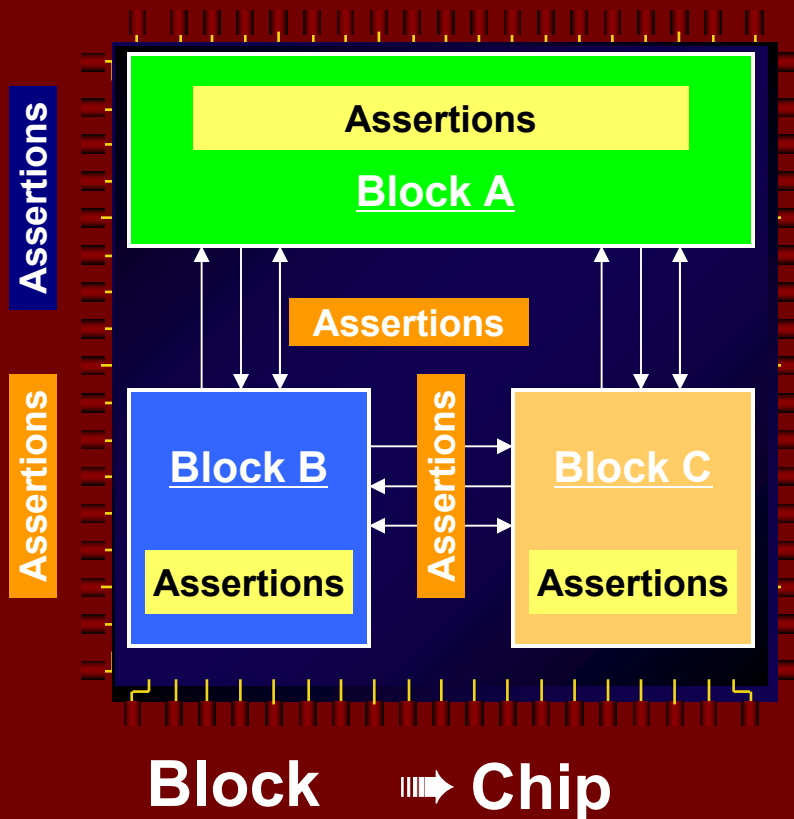
- Why Assertions ?
 - Effectiveness
 - Why it is not used ?
- Deployment Recommendations
- Effectiveness
- Key Issues Addressed
- Summary

SVA Assertions

- A piece of verification code used to check a property
 - correct/illegal behavior
 - assumptions/constraints
 - coverage goals
- Examples:
 - Interface A must follow the AMBA AXI protocol
 - Bus B must be one-hot
 - The FIFO must never overflow
 - I want to see back-to-back reads/writes
 - Write will follow read by 3 cycles

Assertions for Reuse

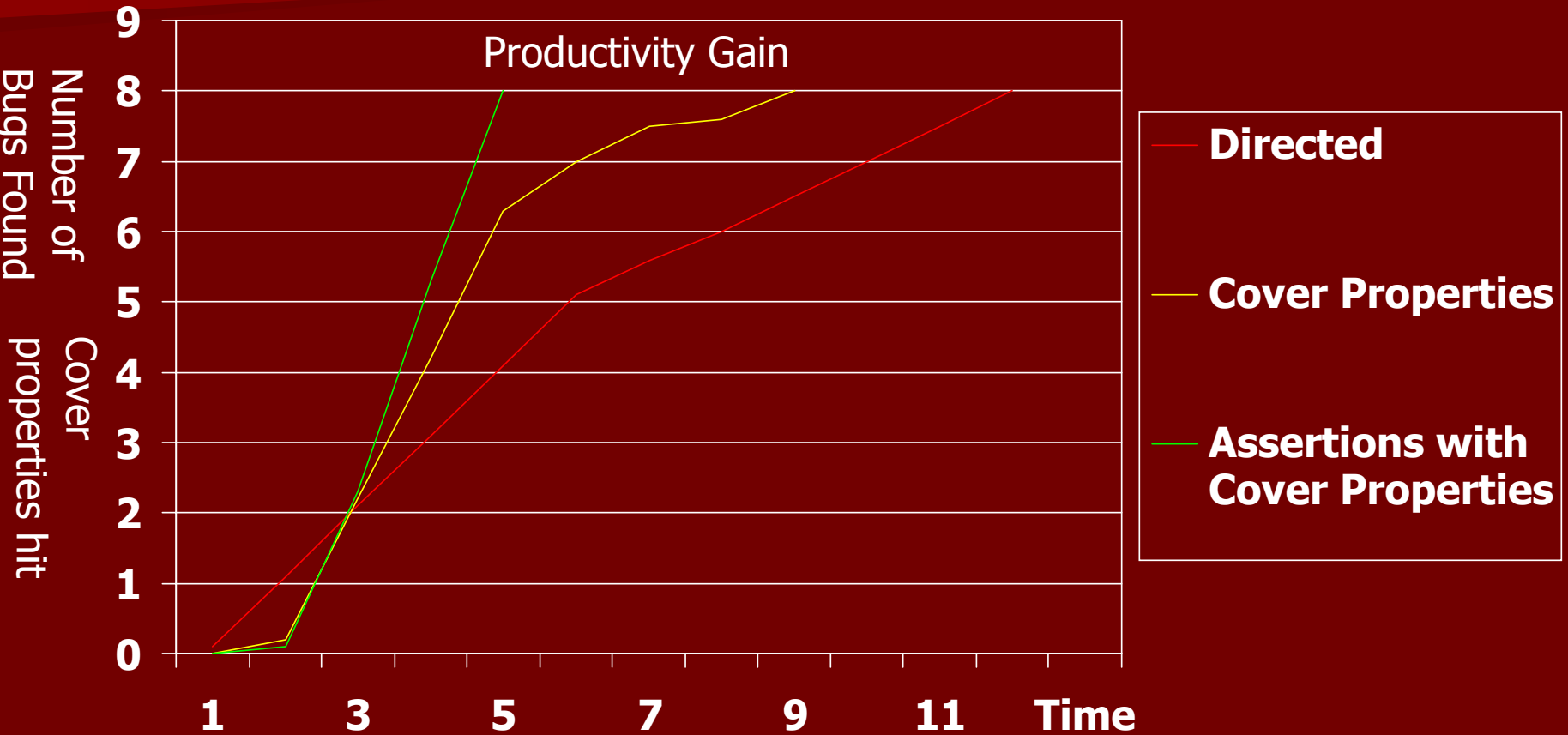
Use Assertions as a focal point of Specification



Reuse

- Add assertions at interfaces between blocks and at chip interface. Those assertions are useful at block level and are reused at chip level and from one project to another.
- Dynamically check the validity in simulation
- Analyze/prove the validity formally
- Utilize coverage metrics linking back to specification
- It is a white box method of verification.

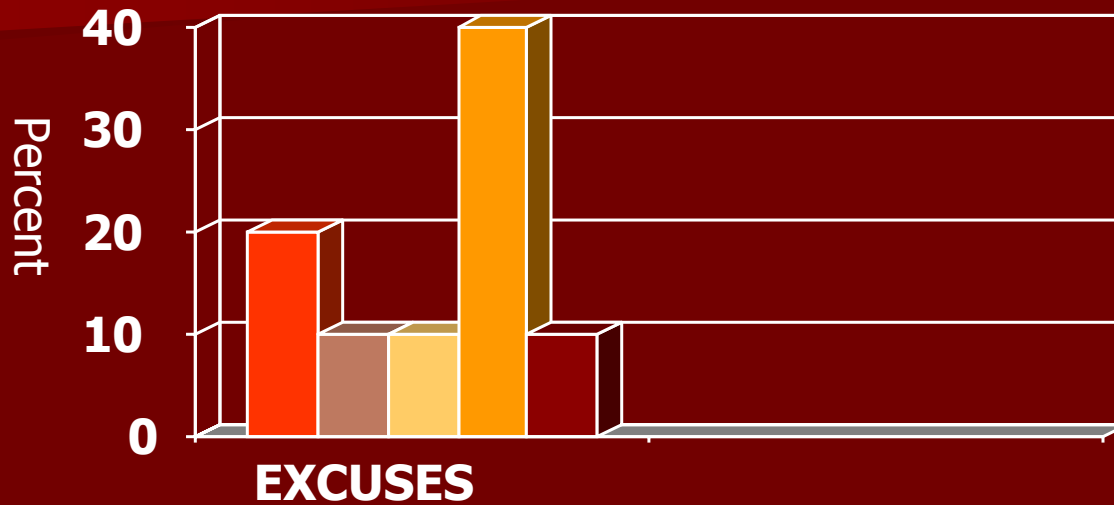
Productivity Gain of Using Assertions



Assertions Benefits

- Assertions describe intended specification unambiguously
- Assertions reused across a broad range of verification tools
 - **Simulation -- Coverage – Testbench -- Formal**
- Makes debug more efficient
 - **Flags errors immediately**
- Major Benefits of SVA we observed in our project.
 - 1. error detection and reduced debug time due to observability**
 - 2. improved integration of modules, units, cores because of built-in self-checking**
 - 3. improved communication between Designers, Verification Engineers and architects.**
 - 4. Better Documentation - helped clarify and define all assumptions and various ways to interpret specifications up front.**
 - 5. Use of SVA covers helped us achieve functional coverage goals.**
 - 6. Use of SVA covers guided us to find holes early on and direct our random generation tool to hit those areas.**

Why SVA is not used



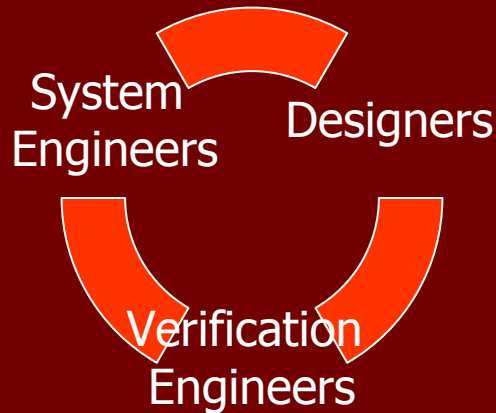
- Not Enough TIME**
- RTL Hard to Read**
- SVA not Synthesizable**
- Slow Down Simulation**
- It's not my Job**

Management of Assertions

- Testbench includes –
 - Scorpion Design includes inline assertions imbedded in the design as well as assertions in an external file.
 - All core design has inline SVA assertions imbedded in the RTL.
 - All interface SVA assertions are coded in external file for maximum re-use between block to unit to core to chip.
 - Ability to select part of Assertions in the design during compile time.
 - Ability to turn off All Assertions at run-time.
 - Ability to turn off an individual Assertion at run-time.
 - Ability to control how the test should end when an assertion fires, number of cycles to continue running so enough waveform dump is captured for further debugging.
 - Ability to control the message generated by Functional coverage.
 - One can get a report stating which functional scenarios were hit and how many times.
 - One can get cycle count when the last assertion was hit so simulation time is not wasted during regression time.
 - Reactive testbench features are utilized to automatically turn-off coverage when threshold is reached for scenarios.
 - For random environment, coveragehole script will analyze (hundreds of tests) and report which scenarios are not covered.

Deployment Recommendation 1

- ?????? Who should Write Assertions ??????



- ANSWER - All of them.
- Training is a Must
- Invest in your design/verification Engineers to have proper training in Assertion language usage that will increase productivity throughout.

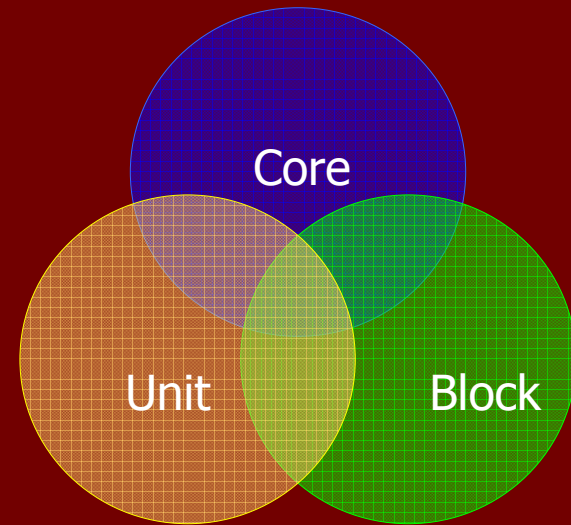
Deployment Recommendation 2

```
Use `ifdef project_core_sva
    SVA Code
`endif
```

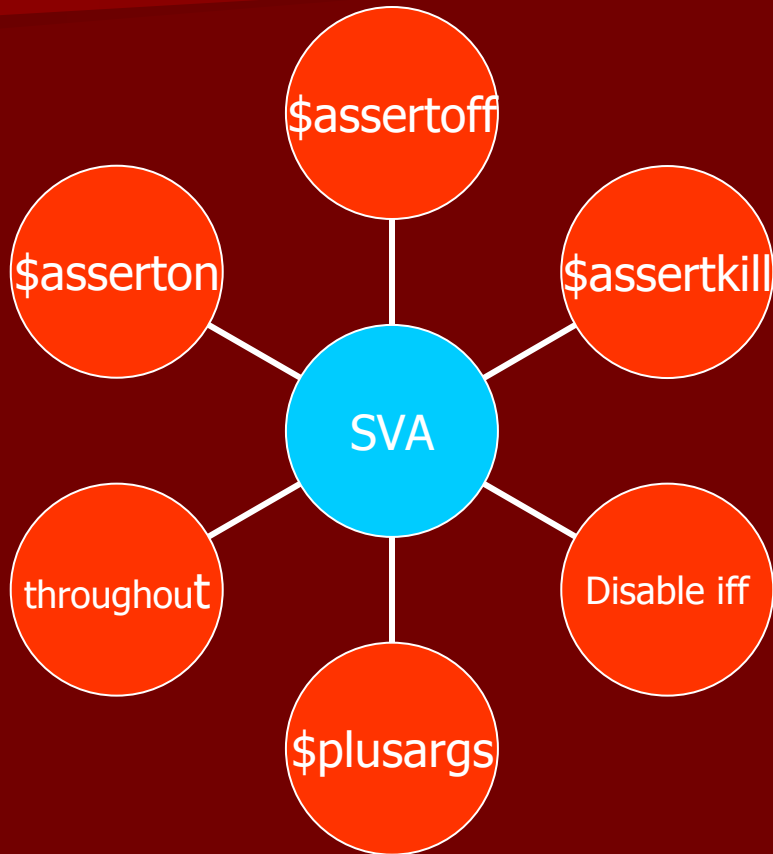
```
Use `ifdef project_Interfacename_sva
    SVA Code
`endif
```

```
Use `ifdef project_blockname_sva
    SVA Code
`endif
```

```
Use +define+project_blockname_sva+
with your favorite Simulator
```



Controlling Assertions



– Use of `$assertkill` or `$assertoff(0)`

```
always @(rst_n)  
if (!rst_n) $assertkill;  
else $asserton;
```

Deployment Recommendation 4

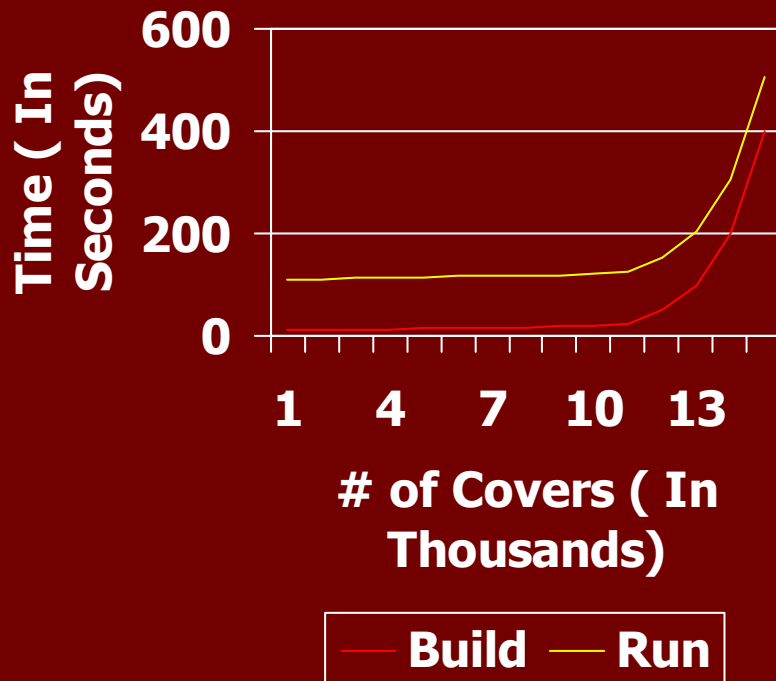
- Allow for change
 - Assertion not valid or out of date.
 - Testbench should allow disabling an assertion at run-time by name
 - `$assertoff(hierarchical.name.assertname);`
 - Use a `$plusarg` to pass the string and if simulator supports `$assertoff (string)`

Deployment Recommendation 5

- Naming conventions and Guidelines
 - Always give a meaningful name to the assertion label. It will help identify exactly what is wrong and can be assigned quickly to the right person for debugging.
 - Write cover properties for all the functional scenarios you want to cover.
 - Use *project_name_assert_or_cover_label* to avoid name pollution with other groups.
 - Always try to constrain unbounded.

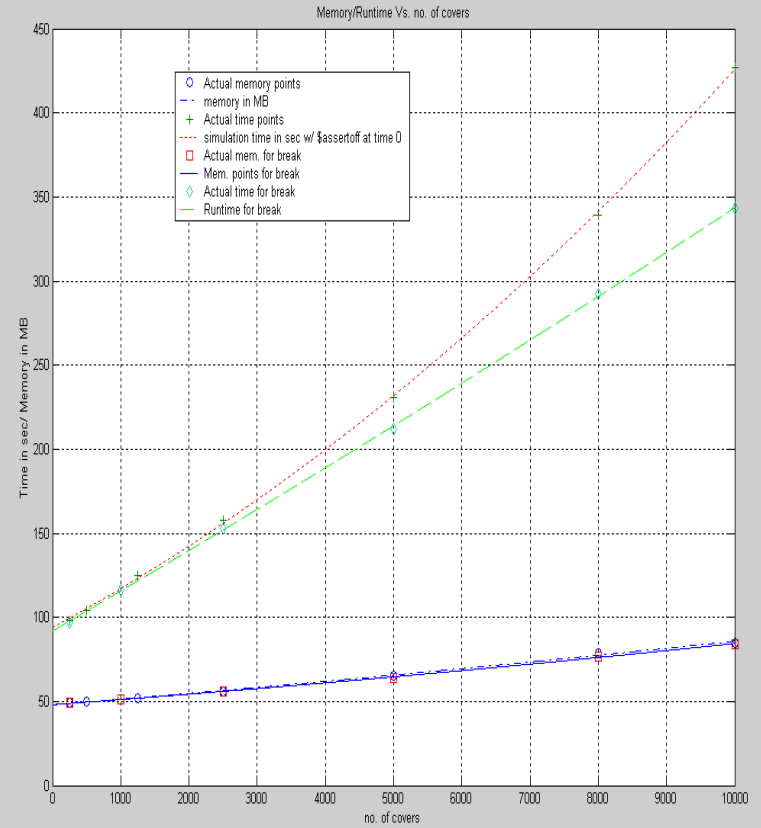
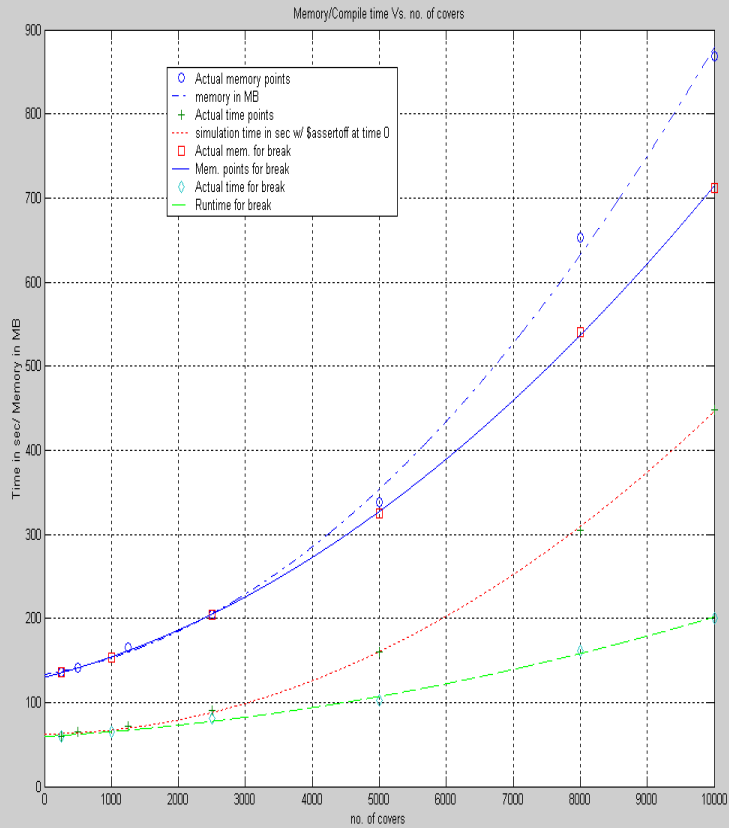
Deployment Recommendation 6

SIM time Vs Number of Assertions



- Assertion Density (assert and cover)
 - In Scorpion we have about 1000 asserts embedded in RTL and about 3700 cover properties that cover the design. For more depth we have about 400,000 cover properties that go into more details.
 - Using directed tests and some random tests we can hit these cover properties which gives us a good feedback about the assertion density.
 - The guideline here is to limit the number of assertions to less than 5000 that are compiled in the simulation model and the performance hit on simulation is minimum.

Actual compile and run-times



Deployment Recommendation 7

■ Managing the test finish when assertion fires.

- SVA allows calling a task when assertion fires. Always call your own task where you can add how many cycles it should continue running after the assertion fires for proper waveform dumping etc.
- In the Example we call `assertFailDisplayMsg(assertName, AssertMsg)` as action task where we pass the name of the assertion and any comment we want to display on the standard output.
- In the common task called by all negative assertions, one can have control over what to do which includes
 - How many cycles to wait before ending the sim.
 - Ignore the assertion firings
 - Generate other database information for post processing tools such as regression management.

- ```
string mymsg;
l2i_a23a: assert property(@(posedge Clk)
 !reset && RLDNxtCycle |-> ##1 !l2iRLDFirstCycle_Q)
else
 begin
 $swrite(mymsg, " %m ASSERT::At time of failure,
 RLDNxtCycle is %d, l2iRLDFirstCycle_Q
 s %d", $sampled(RLDNxtCycle),
 $sampled(l2iRLDFirstCycle_Q));

 assertFailDisplayMsg("l2i_a23a", mymsg);
 end
```
- `$swrite` is Verilog 2000 built-in function which behaves exactly same as `sprintf` in C and it prints the message in a string.
- `%m -à` will get expanded to the full hierarchical name of the module.  
( `scorpion_tb.scorpion_top.cpu.iu.iu_l2i` )
- `$sampled()` will give you the value of signal when the assertion fired and not the current one.

# Deployment Recommendation 8

- Managing the log files.
  - Running in a random environment when coverage hits are dumped can generate a lot of large log files and overflow your storage space. Look to control the ways assertion firings are reported and learn the simulator options to minimize the size.
  - Print the simulation build information, seed and cycles run in the log file for easier debugging and replay.

# Deployment Recommendation 9

- Validating Assertions by inserting faults
  - Add a capability to inject errors to verify that your checkers and assertions are active and catching faults.
  - This will help build quality in your testbench.

# Deployment Recommendation 10

- Create a library of Assertions commonly used on your project.
- Use SVA checker library by adding your company wrappers around so it works with different simulators.

# Deployment cost of Assertions

## ■ Low effort

- Write all checks for X's and Z's.
- protocol operation (relationships, order, response, one-hot, etc)
- unknown values for control (at any time) and data (during transfers)
- over/underflow (FIFOs, stacks, buffers, shift registers, etc)
- legal state acquisition and transitions (Finite State Machines)

## ■ Medium effort

- data integrity (FIFOs, shared memories )

## ■ High effort

- full functionality of complex bus protocol (e.g. AMBA, PCI, AXI)

## ■ Not Worthwhile

- Computation like CRC, checksum etc
- Data order of instructions or packets

# Key Issues Addressed

- Simulator Support
- Industry Standard
- Mixed Language – Verilog, VHDL
- Training – Easy to Learn
- Debuggers Support
- Developing Verification Environment
  - Managing Assertions
  - Managing Coverage
  - Simulation Tool Capacity, Performance and maturity.

# Summary

- SystemVerilog Assertions enable true assertion-based verification and reduce overall verification schedule.
- For more effective deployment of SVA use the following techniques.
  - Training for the whole team as SVA is a team effort.
  - Naming convention and coding guidelines.
  - Control to enable and disable assertions at compile/run time.
  - Partitioning the SVA asserts and covers as wide and deep to cope with the capacity of the simulators.
  - Having a good unified coverage database.
  - Having enough hooks in the testbench to generate data for post processing tools and replay, reporting, and coverage closure.
  - Validating your SVA by having capability of inserting faults.

# References

- IEEE 1800-2005 System Verilog Language Reference Manual
  - IEEE 2005 ISBN 0-7381-4811-3
- System Verilog Assertion Handbook
  - Ben Cohen 2005 ISBN:0-9705394-7-9