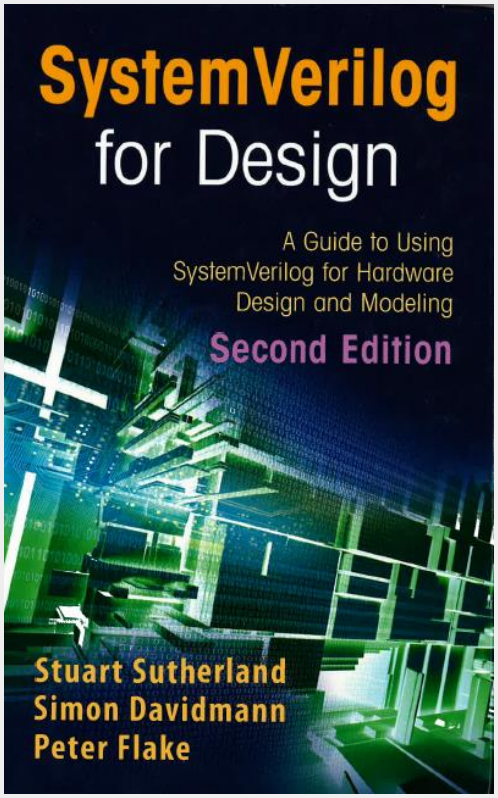
INF5430

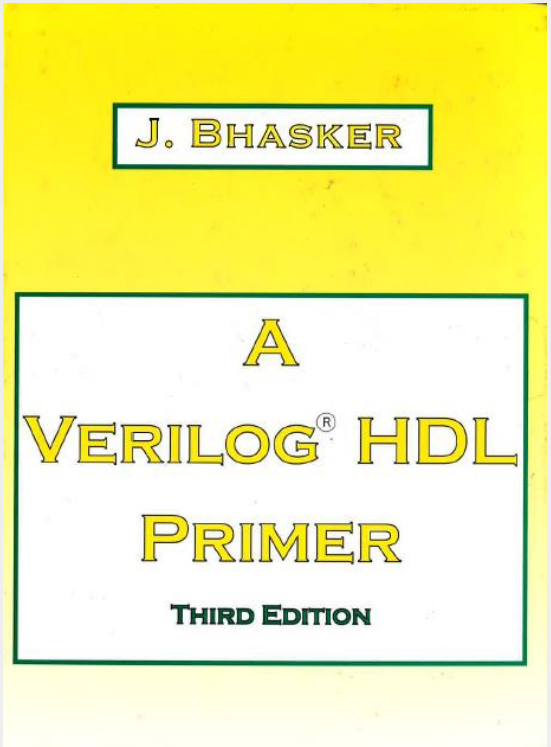
Introduction to Verification with SystemVerilog

References Verilog and SystemVerilog Design

Sutherland, Davidmann & Flake — *SystemVerilog for Design, 2nd Edition*

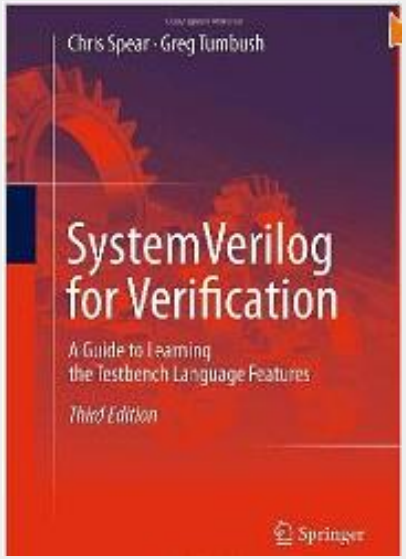


Bhasker — A Verilog HDL Primer, *3rd Edition*

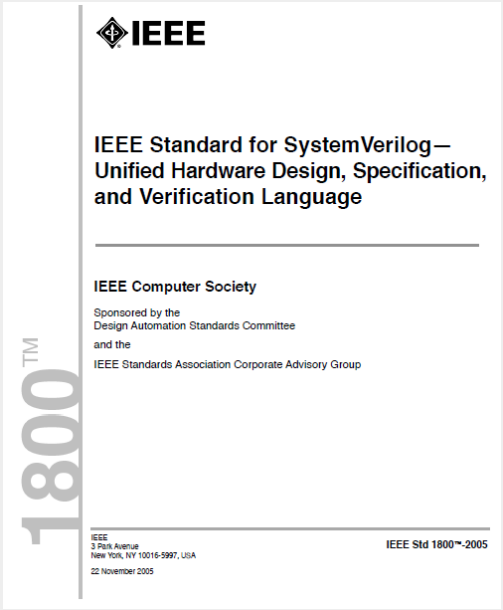


References SystemVerilog Testbenches

Spear and Tumbush , *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features, 3rd Edition*

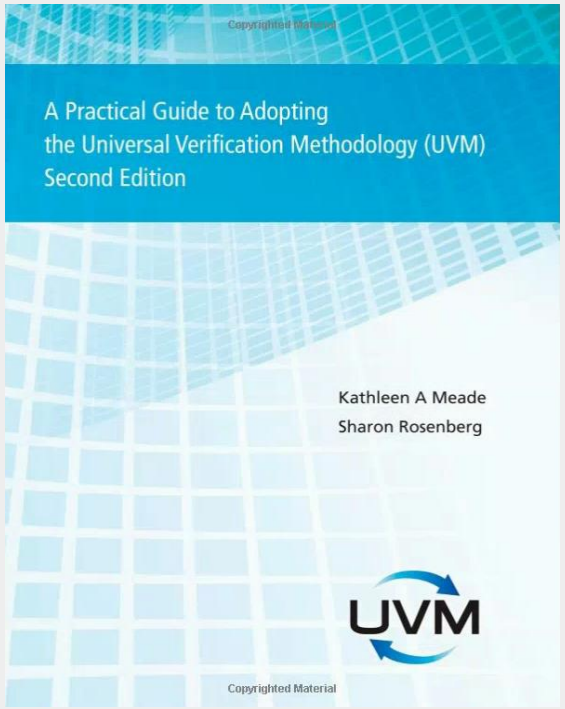


IEEE, *IEEE Standard for SystemVerilog— Unified Hardware Design, Specification, and Verification Language, 2009*

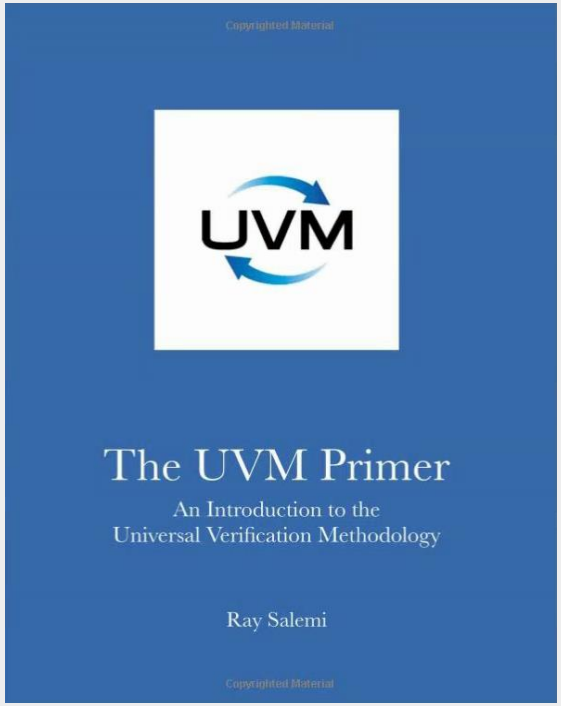


References SystemVerilog UVM

K. A. Mead & S. Rosenberg — A Practical Guide to Adopting the Universal Verification Methodology (UVM), 2nd Edition



Ray Salemi — *The UVM Primer*

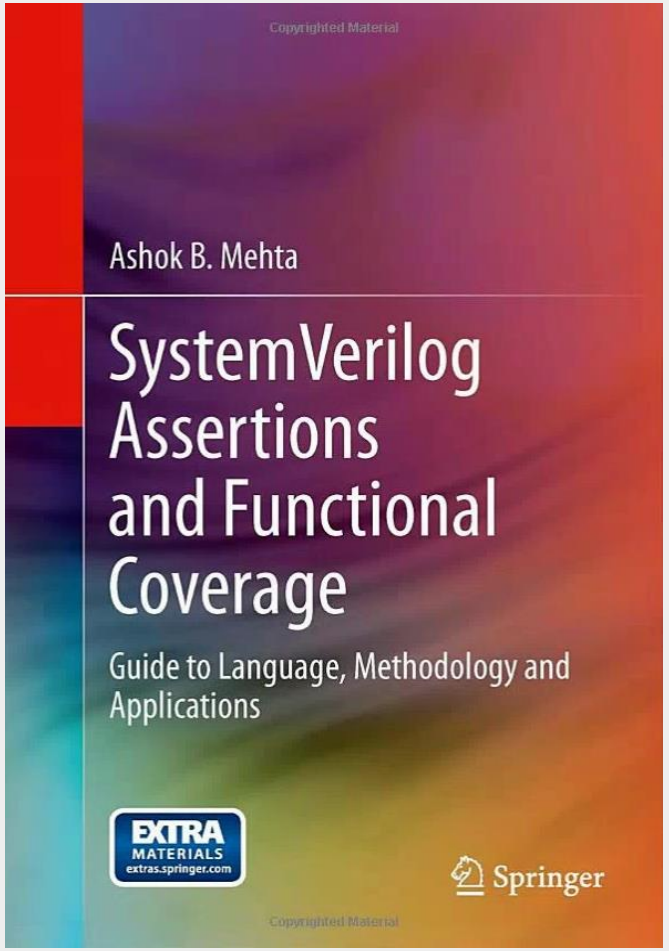


Reference SystemVerilog Assertions

Ashok B. Mehta

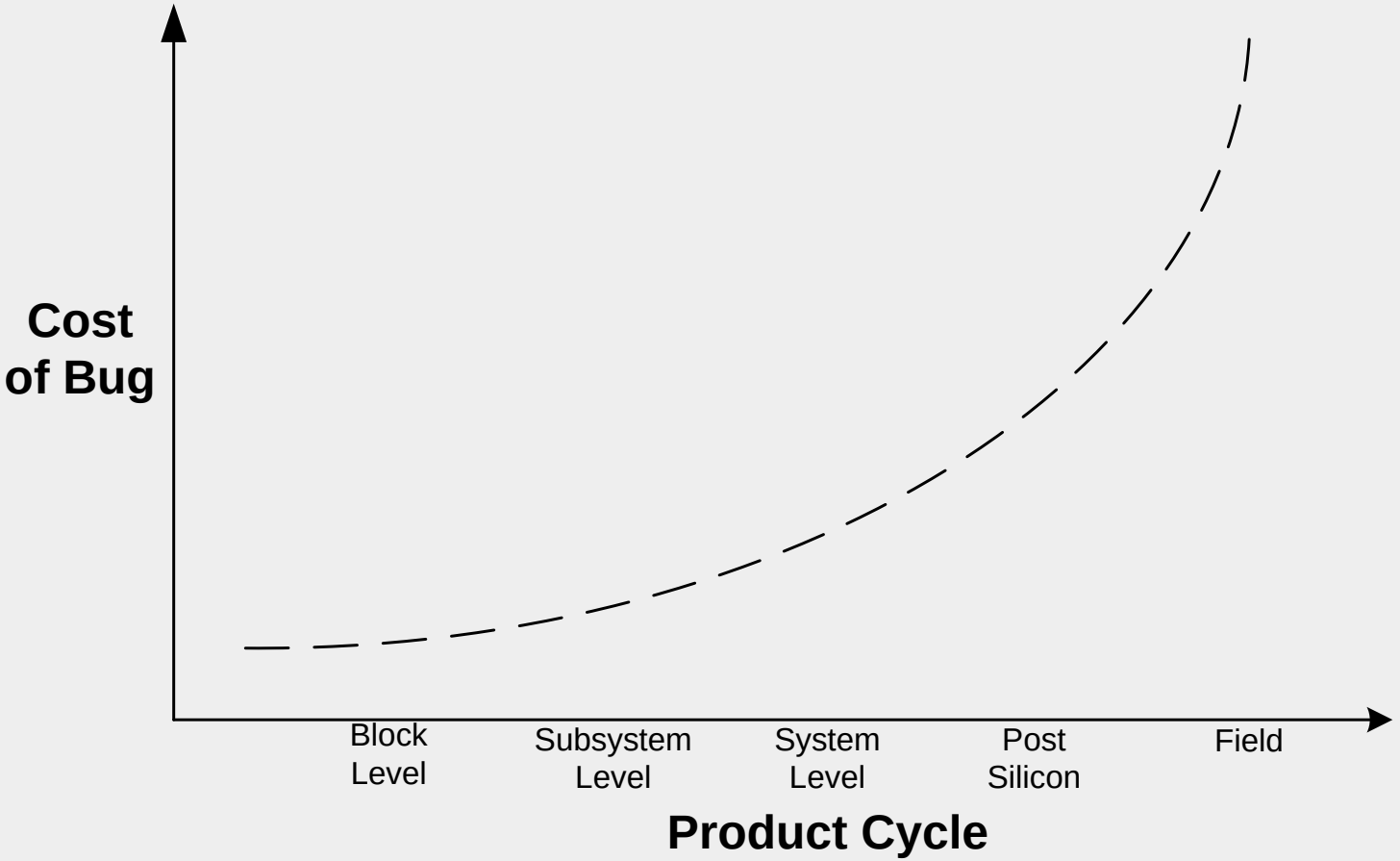
SystemVerilog Assertions and Functional Coverage;

Guide to Language, Methodology and Applications.



Why Verify?

The later in the product cycle a bug is found the more costly it is.



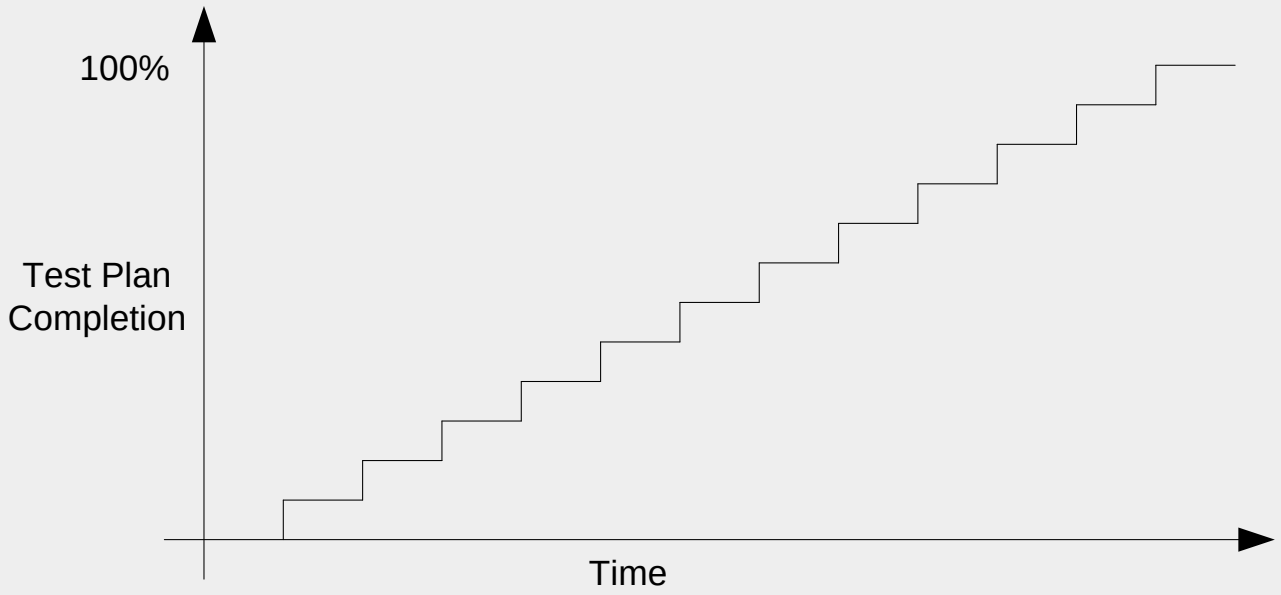
1.3 Basic Testbench Functionality

1. Generate stimulus
2. Apply stimulus to DUT
3. Capture the responses
4. Check for correctness
5. Measure progress

1.4 Directed Testing

Most (all) probably specified directed testing in their test plan

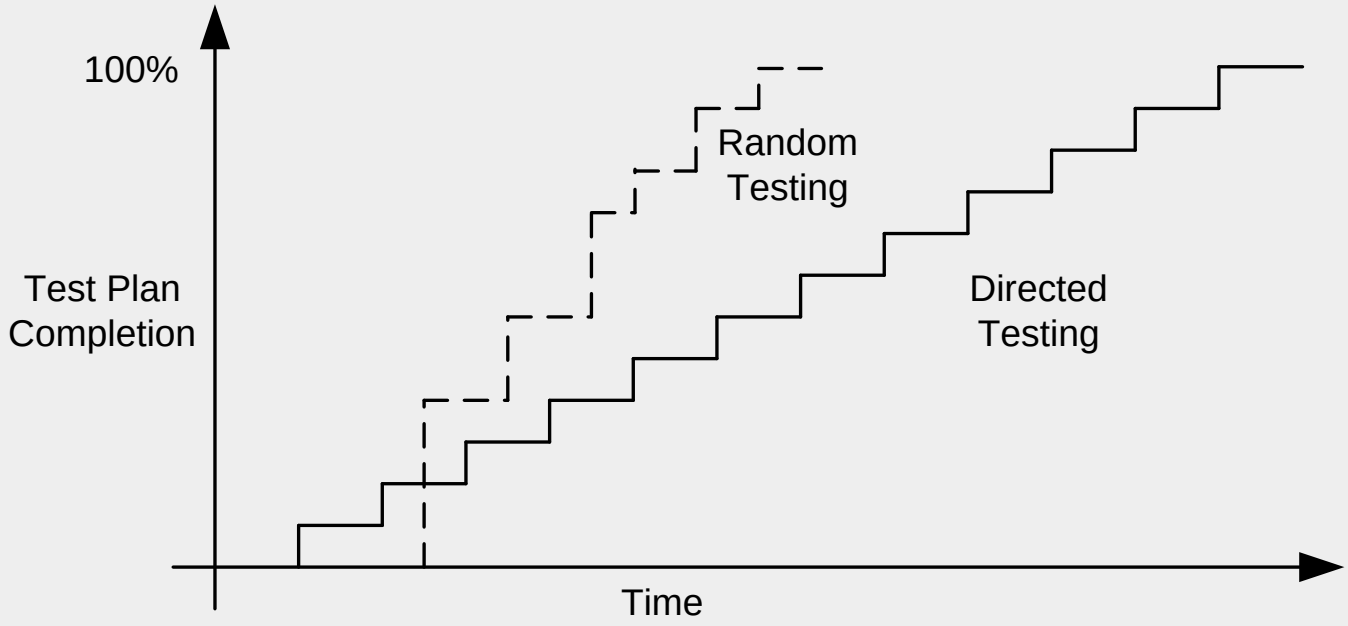
- Steady progress
- Little up-front infrastructure development
- Small design changes could mean massive test changes



1.5 Methodology Basics

Our verification environments will use the following principles

1. Constrained random stimulus
2. Functional coverage
3. Layered testbench using transactors
4. Common testbench for all tests
5. Test-specific code kept separate from testbench



1.7 What should you randomize?

Much more than data

1. Device configuration
2. Environment configuration
3. Protocol exceptions
4. Errors and violations
5. Delays
6. Test order
7. Seed for the random test

1.8 Functional Coverage

How do you know your random testbench is doing anything useful?

Functional coverage measures how many items in your test plan have been tested.

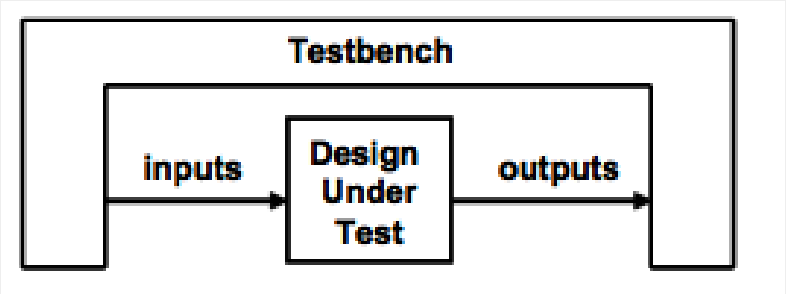
For ALU:

1. Have all opcodes been exercised?
2. Have operands taken values of max pos, max neg, 0?
3. Have all permutation of operands been exercised?

Functional coverage can be collected manually or by writing SystemVerilog coverage statements.

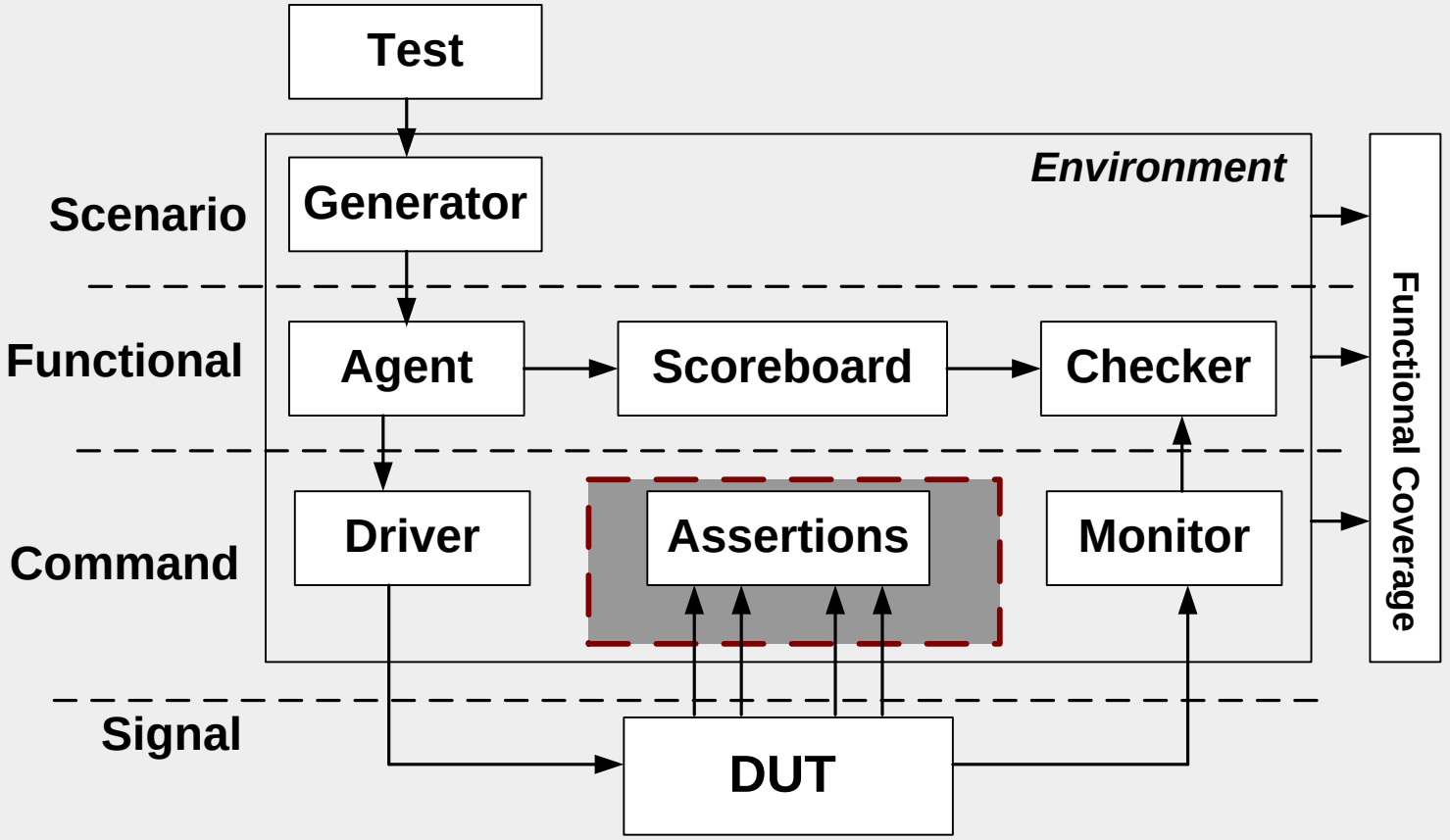
1.9 Testbench Components

- Testbench wraps around the Design Under Test
  - Generate stimulus
  - Capture response
  - Check for correctness
  - Measure progress through coverage numbers
- Features of an effective testbench
  - Reusable and easy to modify for different DUTs <- Object oriented
  - Testbench should be layered to enable reuse
    - Flat testbenches are hard to expand.
    - Separate code into smaller pieces that can be developed separately and combine common actions together
  - Catches bugs and achieves coverage quickly <- Randomize!



1.10.5 Test Layer and Functional Coverage

- Test block determines:
  - What scenarios to run
  - Timing of scenarios
  - Random constraints
- Functional Coverage
  - Measures progress of tests
  - Changes throughout project



1.13 Maximum Code Reuse

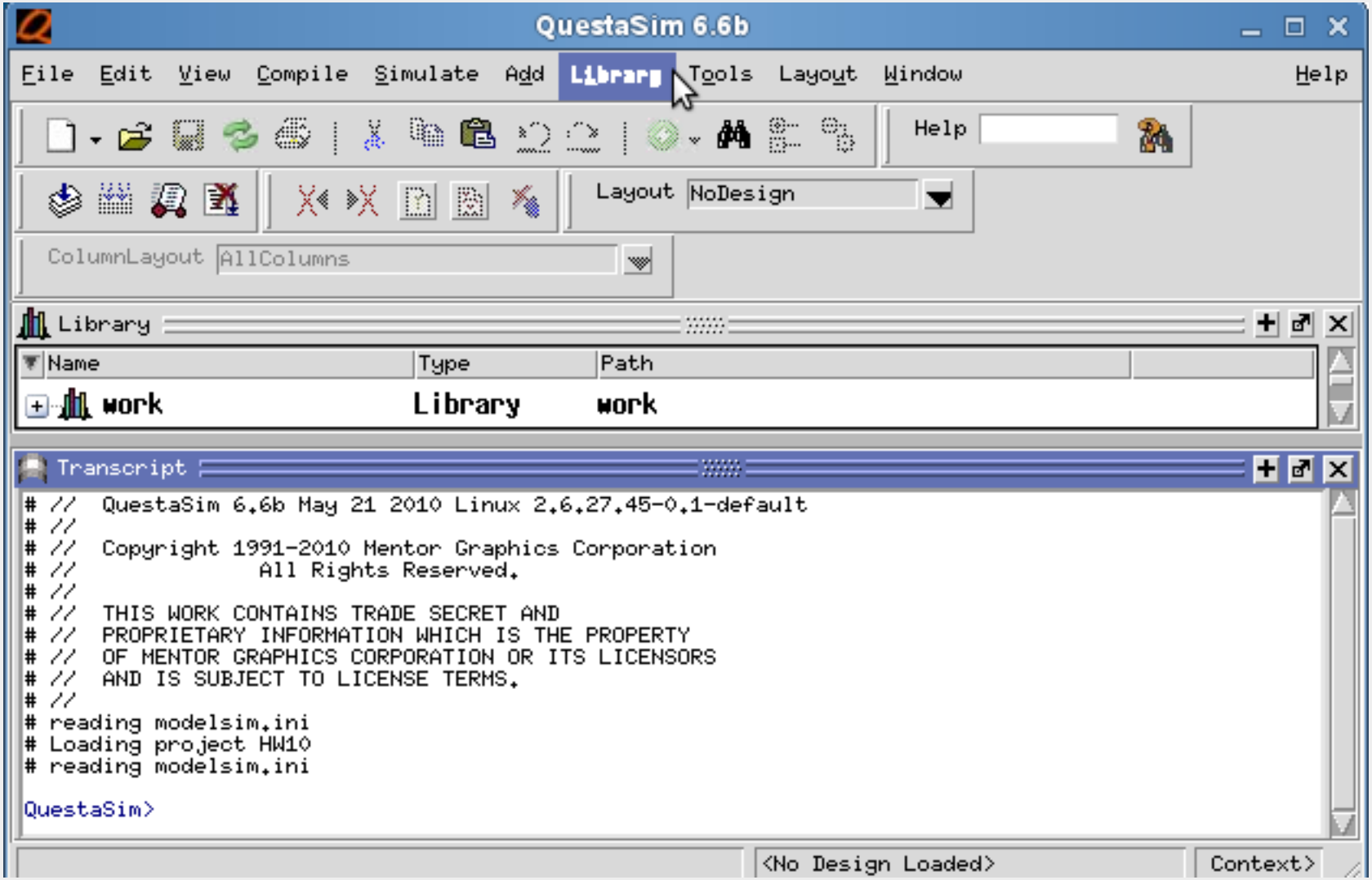
- Put your effort into your testbench not into your tests.
- Write 100's or 1000's of directed tests or far fewer random tests.
- Use directed test to cover missed functional coverage

1.14 Testbench Performance

- Directed tests run quicker than random tests
- Random testing explores a wide range of the state space
- Simulation time is cheap
- Your time is not
- Avoid visually verified tests; i.e. create a self checking testbench
- Test maintenance can be a huge effort

Simulation Tool

Mentor Graphics Questa Prime



Questa Verification IP 10.4 Protocol support

Large Library of Protocols and Interfaces Continues to Expand

- AMBA
  - ACE
  - ACE-lite
  - AXI4
  - AXI3
  - AHB
  - APB
  - AMBA 5 CHI
- DDRx
  - LPDDR4
  - LPDDR3
  - LPDDR2
  - DDR4
  - DDR3
  - DDR2
  - DFI 3.1
  - Wide IO 2
- Ethernet
  - 100G
  - 50G
  - 40G
  - 25G
  - 10G
  - 1G
  - 100M
  - 10M
- Ethernet (Continued)
  - PTP
  - MDIO
  - EEE
  - MII
  - RMII
  - GMII
  - TBI
  - RTBI
  - SGMII
  - RGMII
  - QSGMII
  - BASE-X
  - BASE-T
  - BASE-R
  - BASE-W
  - CAUI
  - XGMII
  - XAUI
  - XLAUI
  - RXAUI
  - XSBI
  - XLGMII
  - CGMII
  - HiGig2
  - FEC
  - Auto-Neg
- HDMI
  - HDMI 2.0
  - HDMI 1.4
  - HDCP 1.4
- MIPI
  - MPHY 3.0
  - LLI 2.0
  - DSI 1.1
  - CSI-2 1.1, CSI-3
  - DigRFv4 1.2
  - HSI 1.0.1
  - Unipro 1.6
  - UFS 2.0
- PCIe
  - PCIe 4.0
  - PCIe 3.0
  - PCIe 2.0
  - PCIe 1.1
  - PIPE
  - PIE-8
  - SR-IOV
  - MR-IOV
  - NVMe, AHCI
- USB
  - USB 3.1
  - USB 3.0 + OTG
  - USB PD
  - PIPE
  - xHCI
  - SSIC
  - USB 2.0 + OTG
  - UTMI+
  - UTMI
  - ULPI
  - oHCI
  - eHCI
- Serial
  - SmartCard
  - SPI – TI, Moto
  - SPI 4.2
  - UART
  - I2C 5.0
  - I2S – Philips, TI
  - JTAG