ASIC Verification
ECE745

Verification Environment

Fall 2009
Dr. Meeta Yadav
Outline

1. Verification Plan
2. Verification Environment
3. Verification Guidelines
Strategy of Verification?

How do I verify my design?

• What resources will I need?
• Am I driving all possible input scenarios?
• How will I know a failure has occurred?
• How do I measure my progress?
• When will I be done?

Verification Plan
Evolution of the Verification Plan

Design and Verification follow the “Waterfall” flow

The Verification Plan

- **Contents of a Verification Plan**
  - Description of verification levels
  - Required tools
  - Risks and dependencies
  - Functions to be verified
  - Specific tests and methods
  - Coverage requirements
  - Test case scenarios
  - Resource requirements
  - Schedule details
Contents of Verification Plan

• Description of Verification Levels
  ◆ Articulating multiple levels of design hierarchy
  ◆ Group levels together into functional components
    ► Complexity of individual components
      – Proper verification on complex functions requires a high level of control and observability
      – Simple functions which do not require high level of control and observability can be combined with other levels
    ► Existence of clean interface and specification
      – “A moving target” interface requires the function should be verified individually
      – Stable interfaces with simple functions can be combined with other levels
Contents of Verification Plan

- **Required Tools**
  - Event simulation tools for units
  - Cycle simulation tools for chip
  - Formal verification tools
  - Assertion-based tools
  - Debuggers
  - Emulation hardware
  - Acceleration hardware
  - Co-simulation hardware
  - High-level verification languages
  - Libraries of functions
Contents of Verification Plan

- **Risks and Dependencies**
  - Tool based risks
    - Delivery and startup delays
    - Integration with established tools
    - Educational challenges
  - On-time HDL delivery dependency: HDL delivery might be scheduled such that simple functions are delivered first and complex functions are delivered later
  - Reliance on a separate verification team
  - Architecture closure
    - Unresolved specification issues
  - Sufficient resources
Contents of Verification Plan

• Functions to be verified
  ◆ Critical functions: Functions the team needs to verify before using the design elsewhere
  ◆ Secondary functions
    ▶ Non-critical to tapeout
      – Performance related functions
      – Functions to be enabled in the later version of the chip
      – Functions with software backup
    ▶ Non-critical to the next level of verification
      – Functions that can be verified in parallel with the next level of verification
      – Corner case conditions
  ◆ Pervasive functions: Operations that do not occur during normal running conditions
    ▶ System resets
    ▶ Error handling
    ▶ System debug
  ◆ Non-verified functions at this level
    ▶ Team fully verified the function at a lower level and the function will be verified again at a higher level through simulation
    ▶ Function is not applicable to this level of verification
Contents of Verification Plan

• **Specific tests and methods**
  ➤ What type of verification?
    ➤ The functions to be verified
    ➤ Exercising the internal structures
    ➤ Error manifestation
    ➤ Availability of resources
  ➤ Verification strategy
    ➤ Deterministic simulation
      ➤ Used for simple designs
    ➤ Random based simulation
      ➤ Complex functions
    ➤ Formal verification
      ➤ Small, complex blocks of design for which many permutations exist
  ➤ Random aspects
    ➤ Hangs due to looping
    ➤ Low activity scenarios
    ➤ Specific directed tests
  ➤ Abstraction level
  ➤ Checking Strategy
    ➤ White box testing
    ➤ Grey box testing
    ➤ Black box testing
Contents of Verification Plan

• **Coverage Requirements**
  ♦ Define coverage goals: A feedback mechanism that evaluates the quality of the verification environment’s stimulus generation components
    ► The environment has exercised all types of commands and transactions
    ► The stimulus has created a specific or varying range of data types
    ► The environment has driven varying degrees of legal concurrent stimulus
    ► The initiator and responder components have driven errors into the DUV
  ♦ Measure coverage progress
  ♦ Fill coverage holes
  ♦ Write directed test cases if necessary
Contents of Verification Plan

• **Test Case Scenarios: Matrix**
  - List of interesting test scenarios
    - Configurations to verify
    - Variations of the data items in the environment
    - Important attributes of data items
    - Interesting sequences for every DUV input port
    - Error conditions
    - Corner cases
Contents of Verification Plan

• Resource Requirements
  ◆ Manpower
    ► Type of environment
      ─ Reference model checking environment requires more people
      ─ Transaction based environments require less people
    ► Experience of individuals
  ◆ Computation resources
    ► (Length of one test scenario X Number of tests to be run) determines the compute as well as license resources
Contents of Verification Plan

- **Schedule Details**
  - Time-line for different verification activities should list
    - Deliveries of verification team
    - Verification work items
  - Schedule should contain
    - Specification delivery
    - Verification environment development
    - First HDL delivery
    - Verification update
    - Regression run
    - Release to manufacturing
  - Schedule should account for each level of hierarchy
  - Verification should be moved to the next level when the bug rate in a particular level begins to drop (estimate this based on history of other projects)

*Lower levels of verification tend to uncover more bugs since they occur earlier in the design cycle and because verification of each designer or unit level occurs in parallel with the others. It is a good practice to wait until the bug rate begins to drop in the low levels before moving to the next level.*

Figure Courtesy: Will, Goss, Roesner: Comprehensive Functional Verification: Elsevier
Verification Environment

**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 modify and control
  - Layered testbenches separate code into smaller pieces that can be developed separately and combine common actions together
- Catches bus and achieves coverage quickly
  - Randomizable!!!
Verification Environment: Layered Testbench

- **Signal layer**
  - DUT and its connections

- **Command Layer**
  - Convert from commands send(), read(), write() to signal lines into the DUT
  - Convert signals from output to commands
  - Assertions written and monitored: assertions encode expected behavior of the system.

```task Driver:: send_instr()
  ...
  in_box1.get(instr2send);
  lc3.cb.complete <= 1;
  lc3.cb.dout <= instr2send.op;
  ...
endtask

task Driver:: send_address()
  ...
  in_box2.get(src1send);
  if(src1send.op == MEM)
    lc3.cb.src1_addr = src1send.addr1;
endtask

task Receiver::get_output()
  ...
  pkt_to_chk.val1 = lc3.cb.out1;
  pkt_to_chk.din  = lc3.cb.din;
  pkt_to_chk.dest = lc3.cb.addr;
endtask```
Verification Environment: Layered Testbench

- **Functional Layer**
  - Convert from high level transactions (TXor) to commands to input to DUT for example DMA read operation
  - TXor to a temporarily storage to keep tabs on input to DUT and the order of inputs
  - Receive data from output of DUT and check with expected result

```
task send_input()
    ...
    ...
    send_instr();
    send_address();
    send_control();
endtask

task receive()
    ...
    get_output();
    check();
    ...
endtask

task check()
    ...
    get_output();
    ...
    if(pkt_to_chk.data!=expected)
        print("ERROR .. ..");
endtask
```
Verification Environment: Layered Testbench

- **Scenario Layer**
  - Generate inputs of interest
  - Directed tests or constrained random

```plaintext
task gen();
// generate an arbitrary number of payloads
// and create random types of inputs
repeat($urandom_range(9,12))
begin
  payload_src1.push_back($random);
  payload_src2.push_back($random);
  payload_imm.push_back($random);
end
endtask
```

*push_back() is an inbuilt function for queues that inserts the object specified at the end of the queue. More on this as the lectures progress.*
Verification Environment: Layered Testbench

- **Test Layer and Functional Coverage**
  - Test controls all that goes on in environment
  - Sets constraints for the stimulus to be sent in
  - Creates multiple combination of tests
  - Functional coverage results used to determine constrains for next set of inputs
Verification Environment: Layered Testbench

- **Benefits of a layered testbench environment**
  - Environment updating takes less time
  - Testbench is easy to constrain from the top level file
  - All Legal Device Configurations are tested
    - Regression can select different DUT configurations
    - Configuration object is randomized and constrained
  - Enables reuse