CSE 45493 - 3

Functional Verification of Hardware Design

Offered jointly by SUNY- New Paltz and IBM
Today’s agenda

- Class Syllabus
- Grading policy
- Verification Introduction
- Industry Perspective (Bruce Wile)
Instructors

SUNY – New Paltz:
• Dr. Baback Izadi
• Jixiong Yang (TA)

IBM:
• Bruce Wile
• Dean Bair
• Kurt Schweiger
• Jim Schafe
• Rebecca Gott
• Steve Mittermaier
• John Aylward
Text


• Additional information on verification can be found at:
  – http://janick.bergeron.com/wtb
Prior Knowledge

- This class focuses on functional verification of hardware design using VHDL
- Expect students to have a basic knowledge of VHDL
- Expect students to have basic understanding of digital hardware design
Course Objectives

• The Course objectives are:
  
  – Students will learn to plan and carry out an effective functional verification of a design.
  – Students will learn to use verification tools and experiment on actual circuits designed in industry.
  – Student will learn to work in teams to debug designs.

• NOTE:  At the end of the semester, students will evaluate how these course objectives have been met.
What is Verification?

- Verification is a process used to demonstrate the functional correctness of a design.
  - Balancing a checkbook
- Also called logic verification or simulation.
Verification VS. Testing

- Two often confused
- Purpose of *test* is to verify that the design was manufactured properly
- *Verification* is to ensure that the design meets the intended functionality
What this course is about?

- To teach necessary concepts and tools for verification
- Describe a process for carrying out effective functional as well as formal verification
- Present techniques for applying stimulus and monitoring the response of a design utilizing bus functional models (BFM)
- Present the importance of behavioral modeling
Importance of verification

• Most books focus on syntax, semantics and register transfer level (RTL) subset (for more information on RTL see: http://www.doulos.com/fi/desguidevhdl/dhuv14_rtl_coding.htm)
  – Given the amount of literature on writing synthesizeable code vs. writing verification testbenches, one would think that the former is a more daunting task. Experience proves otherwise.

• 70% of design effort goes to verification
  – Properly staffed design teams have dedicated verification engineers.
  – Verification Engineers usually outweigh designers 2-1

• 80% of all written code is in the verification environment
Why is verification so important?

- 60% - 80% time spent in verification – WHY??
  - Economics
  - Product time-to-market
    - Hardware turn-around time
    - Volume of "bugs"
  - Development costs
    - "Early User Hardware" (EUH)
Why hardware verification?

- Objective: deliver products and make money
- Cost of bugs over time
  - Longer a bug goes undetected, the more expensive it is
    - Bug found early (designer sim) has little cost
    - Finding a bug in chip/system has moderate cost
      - Requires more debug time and isolation time
      - Could require new algorithm, which could effect schedule and cause board rework
    - Finding a bug in System Test (test floor) requires new ‘spin’ of a chip
    - Finding bug in customer’s environment can cost hundreds of millions and worst of all - Reputation
Conclusion

- Verification is on critical path
- Want to minimize verification time!
  - Using new tools and methodologies
Ways to reduce verification time

- Verification can be reduced through:
  - Parallelism: Add more resources
  - Abstraction: Higher level of abstraction (i.e. C vs. Assembly)
    - Beware though – this means a reduction in control
  - Automation: Tools to automate standard processes
    - Requires standard processes with well defined inputs and outputs
    - Not all processes can be automated
Reconvergence model

- Conceptual representation of the verification process
- Most important question
  - What are you verifying?
Human factor in verification process

• An individual (or group of individuals) must interpret specification and transform into correct function.
Ways to reduce human-introduced errors

• Automation
  – Take human intervention out of the process
• Poka-Yoka
  – Make human intervention fool-proof
• Redundancy
  – Have two individuals (or groups) check each others work
Automation

- Obvious way to eliminate human-introduced errors – take the human out.
  - Good in concept, not always possible
  - Reality dictates that this is not feasible
    - Processes are not defined well enough
    - Processes require human ingenuity and creativity
Poka-Yoka

- Term coined in Total Quality Management circles
- Means to “mistake-proof” the human intervention
- Typically the last step in complete automation
- Same pitfalls as automation – verification remains an art, it does not yield itself to well-defined steps.
Redundancy

- Duplicate every transformation
  - Every transformation made by a human is either:
    - Verified by another individual
    - Two complete and separate transformations are performed with each outcome compared to verify that both produced the same or equivalent result
- Simplest
- Most costly, but still cheaper than redesign and replacement of a defective product
- Designer should NOT be in charge of verification!
What is being verified?

- Choosing a common origin and reconvergence points determines what is being verified and what type of method to use.
- Following types of verification all have different origin and reconvergence points & verify different things:
  - Formal Verification
  - Functional Verification
  - Testbench Generators
Formal verification

• Once the end points of formal verification reconvergence paths are understood, then you know exactly what is being verified.

• Two types of formal verification:
  – Equivalence checking
  – Model checking
Equivalence checking

- Compares two models to see if equivalent
- Proves mathematically that the origin and output are logically equivalent & transformation preserves its functionality
- Only interested in comparing Boolean functions & not their used technology
- Can detect bug in synthesis software
- Examples:
  - Scan chain connections
  - RTL to gates (post synthesis)
  - Post synthesis gates to post PD gates
Equivalence reconvergence model

RTL Gates

 Checks to show mathematically the same

Synthesis

RTL

Gates
Model checking

• Form of formal verification
• Characteristics of a design are formally proven or disproved
• Looks for generic problems or violations of user defined rules about the behavior of the design
  – Example:
    • If ALE is asserted, either RD or WR signal would eventually be asserted.
Model checking reconvergence model

- **Challenge**: knowing which assertion to prove and expressing it correctly.
Functional verification

• Verifies design intent
  – Without, one must trust that the transformation of a specification to RTL was performed correctly

• Prove presence of bugs, but cannot prove their absence
What is a testbench?

- A “testbench” usually refers to the code used to create a pre-determined input sequence to a design, then optionally observe the response.
  - Generic term used differently across industry
  - Always refers to a testcase
  - Most commonly (and appropriately), a testbench refers to code written (VHDL, Verilog, etc) at the top level of the hierarchy. The testbench is often simple, but may have some elements of randomness

- Completely closed system
  - No inputs or outputs out of the whole system
  - Effectively a model of the universe as far as the design is concerned.

- Verification challenge:
  - What input patterns to supply to the Design Under Verification (DUV) and what is expected for the output for a properly working design
Testbench Generators

- Tool to generate stimulus to exercise code or expose bugs
- Designer input is still required
- RTL code is the origin and there is no reconvergence point
- Verification engineer is left to determine if the testbench applies valid stimulus
- If used with parameters, can control the generator in order to focus the testbenches on more specific scenarios
Testbench generation reconvergence model

- RTL code
- Code Coverage/Proof
- Metrics
- Testbench Generation
- Testbench
Functional verification approaches

- Black-Box approach
- White-Box approach
- Grey-Box approach
The black box has inputs, outputs, and performs some function.

The function may be well documented...or not.

To verify a black box, you need to understand the function and be able to predict the outputs based on the inputs.

The black box can be a full system, a chip, a unit of a chip, or a single macro.
White-Box

- White box verification means that the internal facilities are visible and utilized by the testbench stimulus.
- Examples: Unit/Module level verification
Grey-Box

- Grey box verification means that a limited number of facilities are utilized in a mostly black-box environment.
- Example: Most environments! Prediction of correct results on the interface is occasionally impossible without viewing an internal signal.
Perfect verification

- To fully verify a black-box, you must show that the logic works correctly for all combinations of inputs.
  
  This entails:
  - Driving all permutations on the input lines
  - Checking for proper results in all cases

- Full verification is not practical on large pieces of designs, but the principles are valid across all verification.
Verification and test reconvergence model
Verification vs testing

Verification vs testing

Functional Verification

Janick Bergeron, June, 2002
Copyright © 2002 Qualis Design Corporation
Verification and design reuse

- Won’t use what you don’t trust.
- How to trust it?
  - Verify it.
- For reuse, designs must be verified with more strict requirements
  - All claims, possible combinations and uses must be verified.
  - Not just how it is used in a specific environment.
Cost of verification

• Necessary Evil
  – Always takes too long and costs too much
  – Verification does not generate revenue

• Yet indispensable
  – To create revenue, design must be functionally correct and provide benefits to customer
  – Proper functional verification demonstrates trustworthiness of the design
When is verification done?

- Never truly done on complex designs
- Verification can only show presence of errors, not their absence
- Given enough time, errors will be uncovered
- Question – Is the error likely to be severe enough to warrant the effort spent to find the error?
- Verification is similar to statistical hypothesis.
  - Hypothesis – Is the design functionally correct?
## Hypothesis matrix

<table>
<thead>
<tr>
<th></th>
<th>Errors</th>
<th>No Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bad Design</strong></td>
<td></td>
<td>Type II</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(False Positive)</td>
</tr>
<tr>
<td><strong>Good Design</strong></td>
<td>Type I</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(False Negative)</td>
<td></td>
</tr>
</tbody>
</table>
What is verification?

CPI

Performance Verification

Timing Verification

Logic Equival. Verification

Tape-Out (Fabrication)

Architecture

High Level Design

Implementation in VHDL

Functional Verification

Cycle Time
Verification challenge

- How do we know that a design is correct?
- How do we know that the design behaves as expected?
- How do we know we have checked everything?
- How do we deal with size increases of designs faster than tools performance?
- How do we get correct hardware for the first RIT?
Answer: functional verification

- Verification is based on
  - Testpattern generation
  - Reference model development
  - Result checking
Verification process

- Involves

People

Methods

Tools
Logic Verification
Industry Perspective

Bruce Wile
IBM Server Group Verification Lead
8/26/02
What a great time to be an engineer!

- Exciting work
- Major effect on culture
- Compensation
  - Industry's word for "money, morale, and benefits"
Why all the big bucks?

- Basic business principle:
  - Company that gets a product to the market first gets an inordinate share of the market revenue
Triple Constraints

- Schedule
- Costs
- Quality
Why is verification so important to the chip industry?

- Verification is the single biggest lever to positively effect the triple constraints
  - Fewer revs through the fabrication process means lower costs and faster time-to-market
  - Re-spinning a chip costs:
    - Hundreds of thousands of dollars
    - 6-8 weeks
- So if you can get it right in fewer "passes", you WIN!!!
Server Design Process

High Level Design/ System Structure

Customer

High Level Chip Design

Substrate/MCM Design

VHDL Chip Implementation

Microcode Design

Service Code Design

Logic Verification

Chip Design Team

Chip Timing, Circuit Design, Integration

Manufacturing

Hardware Fabrication

Systems Test

You are here
EE career choices

EE

- Verification
- Design
- Circuits
Biggest challenges are in Verification

- Circuit design process has been "fixed"
- Industry-wide shortage of "good" verification engineers
Famous Examples of Failed Verification

- Intel FP Divide bug
  - Problem uncovered on Pentium chips in 1994
  - Bug discovered by math professor investigating mathematical theory
  - FP Unit returns erroneous values for certain digits beyond the 8th significant digit
  - Despite 1 in a million users ever being affected, confidence in Intel dropped
  - Opened the door for AMD market growth
  - NASA's Jet Propulsion Lab was worried that the Pentium chip would cause errors.....
Famous Examples of Failed Verification

- NASA.....
  - 1999 Mars Climate Orbiter
  - System level verification failure caused Orbiter to fly too close to Mars atmosphere and burn up
  - Problem was a mix up between metric and english units of measurement causing a miscalculation in trajectory
So why is verification such a challenge?
Options for hitting performance

MHz-GHz

Instructions per cycle

[Graph showing a downward curve from higher MHz-GHz to lower Instructions per cycle]
So where are we now?

- Hundreds of processors
- Increasing complexity
- Multi-GHz designs
- Speculative execution
- Speed Demon & Braniac!
- Superscaler (multiple path) execution
- Complex recovery
Options for hitting performance

Bottom line: Complexity will continue to increase.
The Art of Verification

- Two simple questions; One huge task
  - Am I driving all possible input scenarios?
  - How will I know when it fails?
Three Simulation Commandments

1. Thou shalt stress thine logic harder than it will ever be stressed again.
2. Thou shalt place checking upon all things.
3. Thou shalt not move onto a higher platform until the bug rate has dropped off.