Challenges of Design Verification

Sridhar Narayanan
srider_narayanan@apple.com

April 11th 2019

Agenda

• Design flow and Verification methodologies
  – Design Process
  – Verification Environments
  – Test Generation

• Required Background for Verification
  – Software
  – Logic and Formal
  – Architecture and Microarchitecture
  – Understanding Bug Probabilities

• Verification Challenges
  – Design complexity
  – Test plan creation
  – Functional verification
  – Performance and Power verification
  – When is verification complete?
  – Design for verification
  – Silicon bring up and debug
Design Flow & Verification Methodologies

Design Process Flowchart

- Project Start
- Architecture Definition
- Performance Modeling
- RTL Implementation
- Verification
- Tape-Out(s)
- Production
- Fab
- Silicon Bring up & Validation
Verification Environments

Formal
- Assumptions
- DUT
- Assertions
- Modeling Code

Simulation
- Test Bench
- Xactors
- DUT
- Assertions
- Monitors
- Checkers

Emulation
- FPGA

Verification Infrastructure

Approaches and Platforms

- Test Bench
  - HDL (Verilog, VHDL)
  - VMM, UVM
  - System Verilog
  - C/C++

- Validation Platform
  - Simulation
  - Emulation
  - FPGA
  - Formal

- Checkers & Assertions
  - Verilog or VHDL
  - PSL or SVA
  - C/C++
  - UVM

EE 382M II: Verification of Digital Systems, April 11, 2019
Models for Verification
Fidelity of the Design Models

- 4-state vs 2-state simulation
  - X-propagation
  - Impact of reset sequence and uninitialized memories
- Structural vs Abstract (behavior) models
- Zero delay vs unit delay vs back-annotated delay models
- RTL vs Gate-level model
- ASYNC domain crossings for clock and reset
- Multi-cycle paths
Test Generation
Quality vs Quantity

- Generation approaches
  - Hand written: Can take hours or days per test
  - Active random: tens of tests generated per second
  - Constrained random
  - Template driven random
- Ideally, want to generate **HIGH quality tests** **FAST**
Background 1

Software

- Modeling of the designs
  - Functional and behavior models in C/C++
  - Performance models
  - RTL models
  - Gate level models

- Test bench
- Use cases by software
- Random test generation
- Hand written tests
- Checkers

Background 2

Logic Design and Formal Methods

- RTL model - **GOLDEN** model for verification
  - AND, OR, NOT, temporal, and other logic primitives
  - Verilog, VHDL, or System Verilog
  - Assertions (SVA)
  - UPF or CPF (power aware)

- Formal
  - Equivalency checking - combinational and sequential
  - Property model checking
  - Theorem proving
Background 3
Processor Architecture and Microarchitecture

- What is the design?
  - Instruction set / functional spec
  - Performance goals
    - IPC, memory latency and bandwidth
  - Power spec
    - Power target for various workloads and operating modes
  - Frequency spec
    - Operating range and desirable frequencies
- Specs are the foundation for verification
  - Need to verify design meets requirements of spec

Background 4
Understanding Bug Probabilities

- Design complexity
- Resource and Schedule
  - Compute and human
- Construction of test bench to target bugs
- Generation of tests: Random or Directed
- How to flush out bugs
- How to prune the search trees
- Coverage
  - Cannot exercise every possible case
  - Must execute complete test plan
  - Explain all uncovered/unreachable cases
Verification Challenges

Verification Challenge
Design Complexity - Example: Memory Request Queue

<table>
<thead>
<tr>
<th>Table Size</th>
<th>Valid subsets</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>16</td>
<td>65</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>109 K</td>
</tr>
<tr>
<td>16</td>
<td>65536</td>
<td>56 Trillion</td>
</tr>
</tbody>
</table>
Verification Challenge
Exploded Verification Space

- Performance features
  - Super-scalar, OOO, multi-core, multi-threading, heterogeneous computing, SOC
  - VLIW, data speculation, control speculation, register window, register stack engine
- Large capacity of structures and buffers
- Deep sequence of events such as ordering and age
- Adaptive mechanisms in design
- Clock gating and power management
- Multiple clock and power domains
- System bus protocols and link protocols
- Asynchronous events such as errors, interrupts, etc.

Bug Space
Spatial and Temporal

- Spatial
  - Address space
  - Entries
  - Bus widths
  - Pipes
  - Channels
  - Data patterns

- Temporal
  - Pipeline depths
  - Long latency protocols
  - Interaction between structures
  - Number of cycles
Overlay Different Verification Methodologies

Spatial
Address Space Entries
Pipes
Channels

Temporal
Number of Cycles

Verification Challenge

Execute Test Plan

- Design specification
  - Market, features, IP availability, Learning from previous generations
- Continuous design changes
  - Timing, power, performance and logic bug fixes
- Resource planning: human and machines
  - The design should be verified orthogonal to the designer
- A Good Test Plan: What, How, and When
  - Break down into pieces
  - Verify from bottom up and have multi-level approaches
  - Focus and elaborate on the uArch cases and make sure they are covered
  - Redundancy: It's good to have overlapping coverage
Verification Challenge

Functional Verification

- Flushing out the bugs
- Where are assertions and checkers needed?
  - Scope and location
- Problematic areas
  - Undefined cases and backward compatibility
  - Data coherency and ordering
  - Starvation and deadlock
  - Real time constraints
  - Clock and power domain crossings
  - Multiple levels of clock gating
  - Power management

---

Case Study

Hang cases

- Hard hang
  - Deadlock: Resource not available to proceed
  - In a state that has no exit transition
- Soft hang - starvation
  - Livelock: cannot make forward progress
  - Cannot get needed tasks done in expected time
Case Study

Data Corruption

- Wrong data being bypassed
- Extremely difficult to debug in silicon
- Common data corruption failures
  - Unexpected translation fault
  - Access violation
  - Lock starvation
  - Conflict in critical session
- Simulation
  - Use data integrity checkers

<table>
<thead>
<tr>
<th>Valid</th>
<th>Dependency</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0110</td>
<td>Barrier</td>
</tr>
<tr>
<td>1</td>
<td>0000</td>
<td>0f10</td>
</tr>
<tr>
<td>1</td>
<td>0000</td>
<td>0f20</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0f30</td>
</tr>
</tbody>
</table>

Producer Consumer Sequence
* Data located in 2:0f20
* Flag located in 3:0f30

Set data

Set flag

EE 382M-11: Verification of Digital Systems, April 11, 2019
Case Study
Data Corruption

<table>
<thead>
<tr>
<th>Valid</th>
<th>Dependency</th>
<th>Addr</th>
<th>-------</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0110</td>
<td>DSB</td>
<td>.......</td>
</tr>
<tr>
<td>1</td>
<td>0000</td>
<td>0f10</td>
<td>.......</td>
</tr>
<tr>
<td>2</td>
<td>0000</td>
<td>0f20</td>
<td>.......</td>
</tr>
<tr>
<td>3</td>
<td>0000</td>
<td>0f30</td>
<td>.......</td>
</tr>
</tbody>
</table>

Set data

Flag could be set before data is!!

Verification Challenge
Performance Features

- IPC, Bandwidth, and Latency
- Quality of service and real-time requirements
- Tests to exercise the design in meaningful ways
  - Benchmarks: too long
  - DV tests: too small
  - Micro benchmarks: need to ensure they are representative of real workloads
- How to check and debug
  - Performance model: too coarse
  - Latency and bandwidth monitors
- How to define and collect coverage?
Case Study
Prefetch Effectiveness

- Under-prefetch
  - Don't prefetch the data in time
  - Insufficient amount of data prefetched
- Over-prefetch
  - Data prefetched but not used, resulting in waste of power and thrashing
- An adaptive scheme could be implemented to keep prefetch at an “ideal” distance

Stride Prefetcher

![Stride Prefetcher Diagram](image)

Verification Challenge
Power Features

- Clock gating
  - How do you gauge that clocks are gated **ONLY** for the required cycles?
  - Turned off Early or turned on Late  => Functional failures
  - Turned off Late or turned on Early  => Waste of power
  - Difficult to establish necessary and sufficient conditions
- Power gating
  - Similar to clock gating constraints
- Power-aware simulations
- Power estimates for use cases
  - How to generate tests that represent real applications?
Case Study
Clock Gating Corner Case

• Design with multiple levels of clock gating
  – Complicated logic cones into the enable logic
  – Does higher level clock gating subsume the lower level ones?
    • Receiving logic could still be clocked \( \rightarrow \) duplicated requests
    • Sending logic is clocked **BUT** receiving logic stops \( \rightarrow \) lost requests
  – How to exercise all transition cases?

Clock Gating Bug
A Simple Example

• In what situations, could there be a bug?
  – \( \text{On}(B) \&\& \text{On}(C1) \&\& \neg \text{On}(C2) \)
  – \( \text{On}(B) \&\& \neg \text{On}(C1) \&\& \text{On}(C2) \)
  – \( \neg \text{On}(B) \&\& \text{On}(C1) \)
  – \( \neg \text{On}(B) \&\& \text{On}(C2) \)
• How would you check?
• What should be the design guidelines?
Verification Challenge
Other Areas

- Behaviors not observable in RTL model
  - Electrical and process-related features
- Features that don’t always result in functional failures
  - Debug, performance and power features
- Correctness of Transactors, Checkers, and Generators
  - Negative testing: inject bugs in design
  - Check if your test bench catches them

---

When is Verification DONE?
Criteria for Completeness

- All test plans have been executed
- Coverage has reached the target
- Recent bug rates and cousin bugs
- Quality of the bugs
  - Areas not covered
  - Corner cases
  - Complexity of the bugs
Coverage
Find all clues - Check under every single rock

- Architecture coverage
  - Features, ISA, instruction sequences, data pattern for arithmetic operations, and error detection
- uArch coverage
  - Structure, pipeline, full/empty/bypass, event sequences
  - Permutation of parallel events, asynchronous events such as interrupt, stalls, flushes, and replays
  - Clock gating, power management
- RTL coverage
  - Code, toggle, expression, state machine
- Coverage collection
  - Tedious, time-consuming and compute intensive
  - Design keeps changing
  - How to merge coverage across different models: what’s valid to merge and what’s not?

Verification Challenge
Design for Verification

- Quotes from Rindert Schutten and Tom Fitzpatrick's article in EE Times: DFV is a cohesive methodology for design and verification of today's (and tomorrow's) complex SoCs to effectively do the following:
  - Leverage a designer's intent and knowledge to strengthen the verification effort.
  - Specify and refine design modules on multiple levels of abstraction.
  - Maximize correctness of functionality of individual modules.
  - Ensure correctness in the integration of these modules.
- Reduce unnecessary design complexity
- Increase the chance of hitting bugs
  - Perturb event timing
  - Increase the probability of certain corner cases
  - Inject interesting events
  - Build assertions into the designs
Silicon Bring-up and Debug
Tough detective work with very little clues.

- Start with anecdotal evidence => ‘something seems wrong’
  - Reproduce consistently and eliminate non-causal factors
  - Reduce failing test case, collect data and analyze
- Reproduction is vital to perform data collection and experiments
  - Early experiments and data collection help define the landscape
- Silicon debug is a team effort — RTL, DV, Software, Systems, PE/TE
  - Important to record and communicate goals & results of each experiment

Silicon Debug Landscape
Bugs come in all forms and shapes

- Gotcha hardware “features”
  - Software bug but needs HW analysis to figure out what went wrong
- Operating point or bad-part issue
  - May be user error, marginal part or SW bug
  - Incorrect voltage settings, PLL issue, power supply issue, di/dt, voltage droop, thermal
- Physical/Electrical hardware bug
- Logical hardware bug (immune to voltage, not part specific)
  - Hang (look at postmortem state of design)
  - Data corruption (multiple flavors, tricky)
Summary

• Design never stops progressing
• Verification is full of challenges.
• Use the best methods and techniques
  – Test generators
  – Environments and Tools
  – Coverage
  – Design for verification
• Development goals
  – Thoroughness: flush out all the bugs
  – Efficiency: flush out maximum number of bugs given the effort
  – Speed: flush out bugs as soon as possible

Thank You!