Assertion-Based Verification

Harry D. Foster
Chief Scientist Verification
IC Verification Solutions Division
February 2020

Outline

- Industry Pulse
- How Verification is Done Today
- What Makes Verification Difficult
- Observability and Controllability Challenge
- Assertion-Based Verification
- Industry Case Studies
- Summary

Worldwide IC Market Growth


Largest IC Product Categories

Overall World Wide Semiconductor Market

Global FPGA Market

FPGA Applications

HOW VERIFICATION IS DONE TODAY
What is Verification?

Verification is a process of ensuring that a design implementation meets its specification.

Simulation-Based Techniques

—Fundamental verification technique in use today
—Generally scales well
—Testing all possible states is generally incomplete

Simulation Traversal Through the State Space

Time Explosion Problem

How long would it take to exhaustively simulate this example?

2^{34} vectors x 1 vector every micro-second = 584,941 years

An extremely fast simulator by today’s standards!
Simulation and the Time Explosion Problem

- 2ⁿ vectors × 1 vector every micro-second = 584,941 years

Formal-Based Techniques

- Does not require a testbench or input stimulus!
- Automatically uses algorithms to verify the functionality
- Verification can be complete
- Complements simulation-based techniques

Conceptual Formal Tool

- \( T(x, y) \) // next state

How is formal different than simulation?

- Property: \( \text{property } p_{\text{comp}}; \)
- \(@\text{posedge clk}\)
- \( E[1:0: A=B] \)
- \( \text{assert property } p_{\text{comp}}; \)
State Space Explosion

How many states exist in a typical design today?

Time and State Space Explosion

- Simulation suffers from time explosion
- Formal suffers from state explosion
- Together they complement each other

WHAT MAKES VERIFICATION DIFFICULT

INDUSTRY DRIVERS

Rising Design Complexity
INDUSTRY DRIVERS

Rising Verification Complexity

The Emergence of New Layers of Verification

Software
Performance
Security & Safety
Power
Clocking
Functionality
What Makes Verification Difficult?

- Single, sequential data streams
  - Floating point unit
  - Graphics shading unit
  - DSP convolution unit
  - MPEG decode
  - ...

- Multiple, concurrent data streams
  - Cross bar
  - Bus traffic controller
  - DMA controller
  - Standard I/F (e.g., PCIe)
  - ...

Channel

Compressed Audio

Encoder

Decoder

Data Link Layer

TX

RX

PHY

Sequential data streams

1x number of bugs

Concurrent data streams

5x number of bugs

— Ted Scardamalia, internal IBM study

Directed-Test Approach

- Imagine verifying a car using a directed-test approach
  - Requirement: Fuse will not blow under any normal operation
  - Scenario 1: accelerate to 37 mph, pop in the new Billie Eilish CD, and turn on the windshield wipers

A FEW WEEKS LATER...

- Imagine verifying a car using a directed-test approach
  - Requirement: Fuse will not blow under any normal operation
  - Scenario 714: accelerate to 48 mph, roll down the window, and turn on the left-turn signal
The Concurrency Challenge

- A purely directed-test methodology does not scale
  - Imagine writing a directed test for this scenario!
  - Truly heroic effort—but not practical

Finding Corner Case Bugs Due to Concurrency

Directed-test-based simulation finds the bugs you can think of...

Constrained-random simulation finds the bugs you never anticipated!

Concurrency is Complicated to Verify

A Maturing Industry to Address Growing Complexity

Packet-Based Design

Transaction Layer Packet Reformatter

Data Link Layer Packet Reformatter

Retry Buffer

Arbiter

From Fabric

Tx

Rx

From Rx Channel

To PHY

Design Projects

Observability vs. Controllability

Test didn’t set up the condition to propagate the bug

Assertions improve observability and reduce the need to propagate bugs

Controllability and Observability

In Context of Code Coverage

input in;
output out;
output [7:0] counter;
reg [7:0] counter;
wire counter_overflow = (counter == 8'hff);
wire counter_underflow = (counter == 8'h00);

```
if (reset == 0)
  counter <= 0;
else if (counterOverflow) // counter value > 8'hff
  counter <= counter - 1;
else if (counter < 8'h00) // counter value < 8'h00
  counter <= counter + 1;
else
  counter <= counter;
```
Poor Observability Misses Bugs

- Code coverage measures controllability
- 100% code coverage does not mean all bugs are detected [S. Devadas, A. Ghosh, and K. Keutzer. DAC 1996]
- DAC paper study found cases where:

<table>
<thead>
<tr>
<th>Code Coverage Achieved</th>
<th>% of covered lines observable</th>
</tr>
</thead>
<tbody>
<tr>
<td>90% Covered</td>
<td>Only 54% Observable</td>
</tr>
<tr>
<td>100% Covered</td>
<td>Only 70% Observable</td>
</tr>
</tbody>
</table>

Assertions Improve Observability

- Reduce debugging up to 50% [CAV 2000, IBM FoCs paper]
- Bugs detected closer to their source due to improved observability

Where Verification Engineers Spend Their Time

- Test Planning
- Testbench Development
- Creating Test and Running Simulation
- Debug
- Other

Mean % Time Design Engineer is Doing Design vs Verification

- 2014: Design 53%, Verification 47%
- 2016: Design 47%, Verification 53%
- 2018: Design 46%, Verification 54%

Presentation Title

**Assertion-Based Verification**

“How can one check a large routine in the sense of making sure that it’s right? In order that the man who checks may not have too difficult a task, the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole program easily flows.”

Alan Turing, 1949

---

**Property**

- **Property**
  - a statement of design intent
  - used to specify behavior

**Assertion**

- **Property**
  - a statement of design intent
  - used to specify behavior
- **Assertion**
  - A verification directive
High-Level Assertion

- Property
  - a statement of design intent
  - used to specify behavior
- Assertion
  - a verification directive
- High-level
  - Architectural focused
  - Can be part of testbench

Low-Level Assertion

- Property
  - a statement of design intent
  - used to specify behavior
- Assertion
  - A verification directive
- High-level
  - Architectural focused
  - Can be part of testbench
- Low-level
  - Implementation focused
  - Embedded in or bind to the RTL

// Assert that the FIFO controller
// cannot overflow nor underflow

How Assertions Are Used Today

- Formal Verification
- Simulation
- O/S Trials

Who should create the assertions?

Verification Engineer
- High-Level Assertions
- Requirement focused
- Black-box assertions
- Accounted for in testplan
- Compliance traceability
- Create reusable ABV IP

Design Engineer
- Low-Level Assertions
- Implementation focused
- White-box assertions
- Not accounted for in testplan
- Improve observability
- Reduce debugging time
Who should create high-level assertions?

- **Verification Engineer**
  - High-Level Assertions
    - Requirement focused
    - Black-box assertions
    - Accounted for in testplan
    - Compliance traceability
    - Create reusable ABV IP

- **Design Engineer**
  - Low-Level Assertions
    - Implementation focused
    - White-box assertions
    - Not accounted for in testplan
    - Improve observability
    - Reduce debugging time

Who should create low-level assertions?

- **Verification Engineer**
  - High-Level Assertions
    - Requirement focused
    - Black-box assertions
    - Accounted for in testplan
    - Compliance traceability
    - Create reusable ABV IP

- **Design Engineer**
  - Low-Level Assertions
    - Implementation focused
    - White-box assertions
    - Not accounted for in testplan
    - Improve observability
    - Reduce debugging time

Specifying Design Intent

Assertions allow us to specify design intent in a way that lends itself to automation

```
// Assert that the grants for our simple arbiter are mutually exclusive
```

Identifying the Error Condition

For our arbiter example, we can write a Boolean expression for the error condition, as follows:

```
(grant0 & grant1) // error condition
```
Checking the Error Condition before Assertions

- Doesn't lend itself to automation.

```verilog
module arbiter (clk, rst_n, req0, req1, grant0, grant1);

always @(posedge clk or negedge rst_n)
  begin
    if (rst_n != 1'b0) // active low reset
      if (grant0 & grant1)
        $display("ERROR: Grants not mutex");
  endmodule
```

ASIC: Assertion Language Adoption

- **IEEE 1800 SystemVerilog Mutex Example**

```verilog
grant0 and grant1 must be mutually exclusive
```

```verilog
assert property (@(posedge clk) disable iff (~rst_n) !(grant0 & grant1));
```
### IEEE 1850 PSL Mutex Example

grant0 and grant1 must be mutually exclusive

```vhdl
assert always (!grant0 & grant1) abort ~rst_n) @(posedge clk);
```

### Accellera OVL Mutex Example

grant0 and grant1 must be mutually exclusive

```vhdl
ovl_never a_mutex (clk, rst_n, (grant0 & grant1));
```

### Industry Case Studies

- Kantrowitz and Noack
  - DAC 1996
- Taylor et al.
  - DAC 1998

#### Published Data on Assertions Use

<table>
<thead>
<tr>
<th>Percentage bugs found by various techniques</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertion Monitors</td>
<td>34%</td>
</tr>
<tr>
<td>Cache Coherency Checkers</td>
<td>8%</td>
</tr>
<tr>
<td>Register File Trace Compare</td>
<td>1%</td>
</tr>
<tr>
<td>End-of-Run State Compare</td>
<td>6%</td>
</tr>
<tr>
<td>Memory State Compare</td>
<td>7%</td>
</tr>
<tr>
<td>Manual Verification</td>
<td>4%</td>
</tr>
<tr>
<td>PC Trace Compare</td>
<td>25%</td>
</tr>
<tr>
<td>Self-Checking Test</td>
<td>11%</td>
</tr>
<tr>
<td>Simulation Output Inspection</td>
<td>7%</td>
</tr>
<tr>
<td>Simulation Hang</td>
<td>6%</td>
</tr>
<tr>
<td>Other</td>
<td>8%</td>
</tr>
</tbody>
</table>

- 17% of bugs found by assertions on Cyrix M3(p1) project
  - Krolnik '98
- 50% of bugs found by assertions on Cyrix M3(p2) project
  - Krolnik '98
- 85% of bugs found using over 4000 assertions on an HP server chipset project
  - Foster and Coelho HDLCon 2001
- Thousands of assertions in Intel Pentium project
  - Bentley 2001
- 10,000 OVL assertion in Cisco project
  - Sean Smith 2002
DAC 2008 Sun paper with lots of metrics

<table>
<thead>
<tr>
<th>Category</th>
<th>Unique</th>
<th>Instantiated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-Level</td>
<td>3912</td>
<td>132773</td>
</tr>
<tr>
<td>Interface</td>
<td>5004</td>
<td>44756</td>
</tr>
<tr>
<td>High-Level</td>
<td>1930</td>
<td>18618</td>
</tr>
</tbody>
</table>

Bugs Found by Type of Assertion

<table>
<thead>
<tr>
<th>Category</th>
<th>Unique</th>
<th>Instantiated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-Level</td>
<td>3912</td>
<td>132773</td>
</tr>
<tr>
<td>Interface</td>
<td>5004</td>
<td>44756</td>
</tr>
<tr>
<td>High-Level</td>
<td>1930</td>
<td>18618</td>
</tr>
</tbody>
</table>

SUMMARY

Assertion-Based Verification

- The process of creating assertions forces the engineer to think... and in this incredible world of automation, there is no substitute for thinking.
For Additional Info on Industry Trends

- Latest Wilson Research Group Functional Verification Study
  - Verification Horizon Blog
  - http://go.mentor.com/558dr