Outline

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

// SystemVerilog Assertion
property p_comp;
@ (posedge clk)
E |-> (A==B);
endproperty
assert property (p_comp);

Assertions can be used to check results and measure coverage
Time Explosion Problem

- How long would it take to exhaustively simulate this example?

1000000011101011011011110111
101010001000110101110100101

\[ 2^{64} \text{ vectors} \times 1 \text{ vector every micro-second} = 584,941 \text{ years} \]

An extremely fast simulator by today’s standards!

Simulation and the Time Explosion Problem

\[ 2^{64} \text{ vectors} \times 1 \text{ vector every micro-second} = 584,941 \text{ years} \]
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

\[ T_x(a, x, y) \text{ // next state} \]
How is formal different than simulation?

How many states exist in a typical design today?
WHAT MAKES VERIFICATION DIFFICULT

INDUSTRY DRIVERS

Rising Design Complexity
Rise in the Average Number of IP Blocks

Closing the Design Productivity Gap!

Source: Semico Research Corp.

<table>
<thead>
<tr>
<th>Year</th>
<th>Avg. Number of 'Other' SIP Blocks</th>
<th>Avg. Number of CPU / DSP / Controllers</th>
<th>Avg. Number of Embedded Memory Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2017*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2018*</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: Mentor Graphics Corporation, all rights reserved.

Design Engineers are Being Productive

Transistors produced per electronic engineer more than 5-orders of magnitude since 1985

Design Engineers are Being Productive!

Growth of Transistor Volume Leads to Sustained ~ 30% per Year Cost Reduction

We are Keeping Up with Design Complexity!

Thanks to Automation and Reuse!
Another View of Moore’s Law

EDA Cost per Transistor and Total IC Revenue per Transistor Both Decrease About 30% per Year

EDA Cost/transistor
IC Revenue/transistor

Cumulative Transistors Shipped

EDA Cost/Transistor ($)

1.00E-09 1.00E-08 1.00E-07 1.00E-06 1.00E-05 1.00E-04

1.00E+13 1.00E+14 1.00E+15 1.00E+16 1.00E+17 1.00E+18 1.00E+19 1.00E+20

Note: EDA Cost Consists of EDA License and Maintenance revenue adjusted for Inflation…
Source: SIA, VLSI Research, Federal Reserve

EDA Cost per Transistor and Total IC Revenue per Transistor Both Decrease About 30% per Year

Demand for Design Engineers Grows Slowly

CAGR Designers 3.6%

<table>
<thead>
<tr>
<th>Year</th>
<th>Design Engineers</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>7.80</td>
</tr>
<tr>
<td>2010</td>
<td>8.10</td>
</tr>
<tr>
<td>2012</td>
<td>8.53</td>
</tr>
<tr>
<td>2014</td>
<td>10.05</td>
</tr>
<tr>
<td>2016</td>
<td>10.48</td>
</tr>
</tbody>
</table>

CAGR Designers 3.6%
But what about Verification Productivity?

- CAGR Designers 3.6%
- CAGR Verifiers 10.4%

INDUSTRY DRIVERS

Rising Verification Complexity
The Emergence of New Layers of Verification

- Software
- Security Domains
- Power Domains
- Clock Domains
- Functional

What Makes Verification Difficult?

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

  Sequential data streams
  1x number of bugs

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

  Concurrent data streams
  5x number of bugs

-Ted Scardamalia, internal IBM study
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
  *Lady GaGa 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

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

Packet-Based Design

Transaction Layer Packet Reformatter

Data Link Layer Packet Reformatter

Retry Buffer

Arbiter

From Fabric

Tx

To PHY

Rx

From Rx Channel

© Mentor Graphics Corporation
Adoption Trends in Verification Techniques

- Code coverage
- Assertions
- Functional coverage
- Constrained-Random Simulation


OBSERVABILITY & CONTROLLABILITY
Fundamental Challenge of Verification

1. Activate
2. Propagate
3. Detect

- Stimulus
- DUT
- Checkers

= Assertions

Observability vs. Controllability

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

- Assertions improve observability and reduce the need to propagate bugs
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
### 2014 Where Verification Engineers Spend Their Time

- **Test Planning**: 37%
- **Testbench Development**: 3%
- **Creating Test and Running Simulation**: 24%
- **Debug**: 14%
- **Other**: 22%


### Designers Spend a Lot of Time in Verification & Debug

- **Doing Design**
  - 2007: 54%
  - 2010: 46%
  - 2012: 49%
  - 2014: 53%

- **Doing Verification**
  - 2007: 40%
  - 2010: 51%
  - 2012: 47%
  - 2014: 47%

“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

- State Search
  - Formal Prop's
  - Passing tests
- Testbench
  - RTL
  - Improved bug rate
- Formal Verification
- Simulation
- O/S Trials
- FPGA or Emulation

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

[Foster, Larsen, Turpin - DVCon 2006]
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

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)
            if (grant0 & grant1)
                $display (“ERROR: Grants not mutex”);
    endmodule
```

Assertion Language Adoption

![Assertion Language Adoption Chart](chart.png)


* Multiple answers possible
assert property ( @(posedge clk) disable iff (~rst_n) !(grant0 & grant1));

grant0 and grant1 must be mutually exclusive

assert always ( !(grant1 & grant2) abort ~rst_n) @(posedge clk);

grant0 and grant1 must be mutually exclusive
Accellera OVL Memory Address Example

grant0 and grant1 must be mutually exclusive

clk

grant0

grant1

error

rtl_never_a_mutex (clk, rst_n, (grant1 & grant2));

INDUSTRY CASE STUDIES
Published Data on Assertions Use

Percentage bugs found by various techniques:

<table>
<thead>
<tr>
<th>Assertion Monitors</th>
<th>34%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Coherency Checkers</td>
<td>9%</td>
</tr>
<tr>
<td>Register File Trace Compare</td>
<td>8%</td>
</tr>
<tr>
<td>Memory State Compare</td>
<td>7%</td>
</tr>
<tr>
<td>End-of-Run State Compare</td>
<td>6%</td>
</tr>
<tr>
<td>PC Trace Compare</td>
<td>4%</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>

Kantrowitz and Noack [DAC 1996]

Assertion Monitors: 25%

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Register Miscompare</td>
<td>22%</td>
</tr>
<tr>
<td>Simulation &quot;No Progress&quot;</td>
<td>15%</td>
</tr>
<tr>
<td>PC Miscompare</td>
<td>14%</td>
</tr>
<tr>
<td>Memory State Miscompare</td>
<td>8%</td>
</tr>
<tr>
<td>Manual Inspection</td>
<td>6%</td>
</tr>
<tr>
<td>Self-Checking Test</td>
<td>5%</td>
</tr>
<tr>
<td>Cache Coherency Check</td>
<td>3%</td>
</tr>
<tr>
<td>SAVES Check</td>
<td>2%</td>
</tr>
</tbody>
</table>

Taylor et al. [DAC 1998]

Metrics from DAC 2008 Sun paper:

<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>

DAC 2008 Sun paper with lots of metrics

Assertion-Based Verification of a 32 thread SPARC™ CMT Processor [Turumella, Sharma, DAC 2008]

Bugs Found Using Assertions

Bugs Found by Type of Assertion
Significant reduction in debugging time

Assertion-Based Verification of a 32 thread SPARC™ CMT Processor
[Turumella, Sharma, DAC 2008]

<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.

Verification Academy

Assertion-Based Verification

Harry Foster
Chief Scientist Verification

info@verificationacademy.com | www.verificationacademy.com