Meeting the Challenges of Formal Verification

Doug Fisher – Synopsys
Jean-Marc Forey - Synopsys
23rd May 2013
In the next 30 minutes . . .

• Benefits and Challenges of Formal Verification
• Meeting the Challenges of Formal Verification
  – Increasing Capacity
  – Increasing Automation
  – Raising the Level of Abstraction
  – Ensuring Completeness
• Summary & Questions
Synopsys Completes Acquisition of SpringSoft

MOUNTAIN VIEW, Calif., December 2, 2012—Synopsys, Inc. (Nasdaq:SNPS), a global leader providing software, IP and services used to accelerate innovation in chips and electronic systems, has completed the acquisition of SpringSoft (TAIEX:2473), a global supplier of specialized IC design software headquartered in Hsinchu, Taiwan, through a follow-on merger to acquire all of the remaining outstanding shares of SpringSoft. Previously, on October 1, 2012, Synopsys completed a tender offer for approximately 91.64 percent of SpringSoft’s outstanding Shares. As of November 30, 2012, Synopsys is the 100 percent owner of SpringSoft and SpringSoft stock is no longer trading.

The acquisition increases Synopsys’ investment in Taiwan by growing local engineering expertise, technology development capabilities and customer support. The combination of SpringSoft’s and Synopsys’ technology portfolios will help accelerate delivery of a unified, powerful system-on-chip (SoC) debug platform, and a higher level of automation in custom implementation tools to customers.
Expanding Static Verification Solution

- Magellan™
- Verdi³®

- Hybrid Formal Property Checking
- Advanced Debug

- Formal Consistency Checker
- Mutation Analysis

- HECTOR™
- Certitude™

- Some bugs not detected
- ALL bugs detected

© Synopsys 2013
Why Formal Verification?

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Quality</td>
<td>• Adoption</td>
</tr>
<tr>
<td>– Exhaustive Verification</td>
<td>• Realizing the benefits</td>
</tr>
<tr>
<td>– Mandated for mission</td>
<td>• Scaling</td>
</tr>
<tr>
<td>critical applications</td>
<td>• Simulation Integration</td>
</tr>
<tr>
<td>(e.g. medical, automotive)</td>
<td>• Completeness</td>
</tr>
<tr>
<td>– IP reuse</td>
<td></td>
</tr>
<tr>
<td>• Time to results</td>
<td></td>
</tr>
<tr>
<td>– Formal is faster for</td>
<td></td>
</tr>
<tr>
<td>some verification tasks</td>
<td></td>
</tr>
<tr>
<td>– Bug hunting</td>
<td></td>
</tr>
</tbody>
</table>
Assertion Development

- Easier to develop assertions in an interactive tool
  - Example: Verdi
Obstacles to Realizing FV Benefits

**Obstacles**

Additional verification effort
- “As much effort as simulation”
  - Extra resource
  - Extra cost

“Your mileage may vary”
- Did the proofs complete?
- Were there enough properties?
- Were the properties effective?

**Recommendations**

- Use early in order to show bug hunting efficiency
- Use structural assertions
- Use Checker libraries (e.g. OVL)
- Use assertion IP
- Target specific applications
  - Simulation coverage reachability analysis
  - SoC connectivity
Application: Reachability Analysis

- Identifies unreachable coverage points, automatically
- Generates tests for reachable coverage points
- Determine realistic 100% coverage goal
Scaling with Design Complexity

• Increase Tool Capacity

• Use a Hybrid Formal Architecture
  – Augments FV with dynamic bug-hunting engines.
  – Enables formal methods to bug hunt over a wider state space

• Divide Tasks Between Simulation and Static Approaches
  – Will you be able to merge the results?

• Use the Right Formal Engines for the Job
  – e.g. Formal proof of datapath
FV Capacity Limitations

Initial State

Capacity Gap

Bug
Magellan – Hybrid Formal RTL Verification

- Exhaustive proofs of correctness
  - User-defined properties
  - Automatically extracted properties
- Detection of deep corner-case design bugs
- Safety and Liveness property proofs and counter-examples
- Assertion IP and Library support - Boost verification productivity

Harness Combined Power of Formal and Simulation
Magellan’s Hybrid Architecture
Harness Combined Power Of Formal And Simulation

- Initial State
- VCS simulation of random stimulus
- Promising state
- Bug
- Unreachables
- Formal Search 1
- Formal Search 2
- Formal Search 3
The Right Engine For Datapath: HECTOR

Formal Block-Level Consistency Checker

- Proves consistency of independently developed models
- Exhaustively verifies successive design refinements
- Does not require testbenches, assertions, coverage
- In production use at 4 major graphics and CPU companies

Leading Graphics Company’s Experience on Selected Blocks

10+ post silicon bugs found prior to and **NONE found after** HECTOR deployment
**HECTOR Example: FPU Arithmetic IP Verification**

- **Design characteristics**
  - Floating point unit with many rounding modes
  - Two 64-bit inputs

- **Verification challenge**
  - Establishing consistency between C and RTL using simulation

- **HECTOR solution**
  - All FPU operations exhaustively verified
  - Exhaustively proved correctness in minutes/hours
  - Only 1 out of $2^{128}$ input combinations would have exposed the design bug.

---

### HECTOR Example: FPU Arithmetic IP Verification

<table>
<thead>
<tr>
<th>Design Version</th>
<th>Status</th>
<th>HECTOR Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode 1</td>
<td>Proven</td>
<td>100s</td>
</tr>
<tr>
<td>Opcode 2</td>
<td>Proven</td>
<td>78s</td>
</tr>
<tr>
<td>Opcode 3</td>
<td>Proven</td>
<td>82s</td>
</tr>
<tr>
<td>Opcode 4</td>
<td>Proven</td>
<td>178s</td>
</tr>
<tr>
<td>Opcode 5</td>
<td>Proven</td>
<td>215s</td>
</tr>
<tr>
<td>Opcode 6, ver. 1</td>
<td>Bug found</td>
<td>24631s</td>
</tr>
<tr>
<td>Opcode 6, ver. 2</td>
<td>Bug fix Proven</td>
<td>6213s</td>
</tr>
</tbody>
</table>

“NEAR ZERO probability for simulation to catch the bug”
Completeness of Formal Verification

- Properties
- No (explicit) stimuli
- Execution traces / waveforms
- How much of the design is used to get the proofs?
- Which aspects of the design are “verified”?
- White box verification remains a misleading friend
Completeness of Formal Verification

• not (FifoFull && FifoEmpty)

• In the above example
  – Inputs ReadEn/WriteEn can influence the FifoEmpty output
  – Only 1 gate is needed to prove the property
    – The property is necessary but not sufficient

• Cone of influence !≡ What is verified ≅ Reduced cone
Completeness

• Assertion density
  – Assertions per line of code
  – Doesn’t account for complexity
  – Doesn’t measure effectiveness

• Sequential depth
  – Number of clock cycles in path from register to assertion
  – Accounts for complexity
  – Doesn’t measure effectiveness

• Mutation Analysis
  – Accounts for complexity
  – Measures effectiveness of assertions
What is Mutation Analysis?

- Measures ability of verification environment to **activate**, **propagate**, and **detect** RTL bugs
- Identifies specific verification holes that would let bugs escape
- Provides objective measure of overall verification robustness

The Synopsys Mutation Analysis Tool is called **Certitude**
Certitude Principle

• Inject faults (artificial bugs)

• Run the verification
  – At least 1 Fail => fault is detected (good)
  – All Pass => fault is not detected (bad)
How it Works

• **Automatically Modifies RTL code to insert faults**

  \[ \text{out1} = f(\text{in1}) \rightarrow \text{out1} = 1'\text{b0} \quad // \text{disconnect output and tie to constant} \]

  \[ \begin{align*}
    \text{if (a)} & \rightarrow \text{if (TRUE)} \quad // \text{remove “else” branch} \\
    & \rightarrow \quad f1(); \quad \rightarrow \quad f1(); \\
    \text{else} & \rightarrow \quad \text{else} \quad \rightarrow \quad \text{f2();} \\
  \end{align*} \]

  \[ \text{a} = \text{b} \; | \; \text{c} \quad \rightarrow \quad \text{a} = \text{b} \; & \; \text{c} \quad // \text{change operator} \]

• **Simulate Modified RTL code in your environment**
  
  – Does at least one test fail? *Great!*
    – Environment is robust enough to detect that the RTL is broken
  
  – Do all tests pass? *Help!*
    – Original and broken RTL both compliant with environment
How Faults Can Be Missed

**NA**
- Not Activated;
- RTL Fault is not reached

**NP**
- Non Propagated;
- No effect at the boundary of the design and tests pass

**ND**
- Not Detected;
- Output behavior is different, but tests pass
Initial Results: Properties Incomplete

Original Code & Properties Results

Fault Detected

Fault Not Detected

(More Properties needed)
Improved Results

```verilog
class always @ (negedge Reset_ or posedge Clock) begin
    if (Reset_ == 1'b0) begin
        ReadPtr <= ((ADDR_WIDTH+1) [1'b0]);
        WritePtr <= ((ADDR_WIDTH+1) [1'b0]);
        FifoEmpty <= 1'b1;
        FifoFull <= 1'b0;
    end else begin
        if (ReadEn == 1'b1) && (FifoEmpty == 1'b0) begin
            ReadPtr <= ReadPtr + 1'b0;
        end
        if (WriteEn == 1'b1) && (FifoFull == 1'b0) begin
            WritePtr <= WritePtr + 1'b0;
        end
        if (WriteEn == 1'b0) &&
            (FifoEmpty == 1'b1) &&
            (WritePtr + 1'b0 == ReadPtr) &&
            (ReadEn == 1'b0) begin
            FifoEmpty <= 1'b0;
        end else begin
            FifoEmpty <= 1'b1;
        end
        if (ReadEn == 1'b0) &&
            (FifoFull == 1'b1) &&
            (WritePtr + 1'b0 == ReadPtr) &&
            (ReadEn == 1'b0) begin
            FifoFull <= 1'b0;
        end else begin
            FifoFull <= 1'b1;
        end
    end
end
```
Combining FV and Simulation Results

**Dynamic Verification** i.e. Simulation
Focusses on Data-Dependencies

Certitude Qualification of Testbench
5 NA, 9 NP, 11 ND out of 83 faults
Improving testbench finds 2 bugs in verification environment
2 NA, 2 NP, 2 ND out of 95 faults

**Static Verification** i.e. Formal
Focusses on Control Path & Protocols

Certitude Qualification of Properties
28 NA, 0NP, 14 ND out of 83 faults
Adding 2 properties
34 NA, 0NP, 2 ND out of 93 faults
(NDs in redundant code)

**Merge Common Metrics from Dynamic and Static Environments**

- 0 NA, 2 NP, 0 ND out of 95 faults
  - **Two Redundancies detected**
- Obtain the global metric of the verification quality
- Address the issues where best suited
- Optimize the verification effort
Completeness Measured

• Certitude helps Measure Completeness
  – Understand what parts of the design are verified
  – Even more important with formal verification

• Addressing verification issues pointed out by Certitude allows verifier to discover design bugs
  – The bugs discovered with formal would probably not have been found with dynamic verification – too corner case

• Certitude Provides a Strong Unified Metric for both Simulation and Formal Verification Quality
Summary

• Static Methods are Essential for Complete Verification
  – Complementing simulation and emulation methods
  – A key part of the Synopsys Verification Solution
• Formal Verification Tools Need to Meet New Challenges
  – Increasing Capacity
  – Increasing Automation
  – Raising the Level of Abstraction
  – Ensuring Completeness
Scalability, Completeness, Automation and Debug

HECTOR™

Certitude™

Formal Consistency Checker

Mutation Analysis

Unreachable States
Formal (Magellan) Stimulus
Dynamic (VCS) Coverage

Realistic 100% Coverage

Magellan™

Hybrid Formal Property Checking

Advanced Debug

Verdi³®

Some bugs not detected
All bugs detected
Thank You