Back to Basics: Doing Formal “The Right Way”

Mark Handover
Digital Design & Verification Solutions
Mentor Graphics Corporation

May 21, 2015
Formal Fact #1: Design Size

Larger Designs Use More Formal

Formal Property Checking Adoption by Design Size
(Gate Count Excluding Memories)

Formal Fact #2: Formal Usage

**Bug Hunting**
- Many RTL assertions
- Success: # bugs found
- Productivity focus

**Assurance**
- A few spec focused assertions
- Success: Design meet spec
- Quality focus

**Useful for both Bug Hunting and Assurance**
Formal Fact #3: Who writes 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

**Both Design and Verification Engineers do**
"The Right Way" Rests on the 4 Pillars of a Successful Formal Test Plan

- Design Style
- Assertions
- Formal Engines
- Coverage

Verification Infrastructure
Design Style: The Wrong Way

- Running formal on the whole SoC design
  — Formal capacity is limited

- Running formal on a block you know nothing about
  — Won’t be able to debug

- Running formal on a block without a proper design spec
  — Reading the RTL code too much

- Data transform block is typically beyond formal capacity
  — Other verification approaches are more efficient for this
Design Style: The Right Way

- Design and interface specifications are important
- Control logic is typically the sweet-spot
- Verification of data transportation is good for formal

Arbiters of many different kinds
- Interrupt controller
- Clock programming unit
- Power management unit
- DMA controller
- Host bus interface unit
- Scheduler, multiple channels for QoS

On-chip bus bridge
- Memory controller
- Token generator
- Credit manager block
- Standard interface (AMBA, PCI Express)
- Proprietary interfaces

Recommendation

Run automatic formal or coverage checking to understand the controllability and observability of the design
Assertions: The Wrong Way

- Putting assertions on a lot of low-level structures (counters, FSMs, etc) of the design.
  — Subject too much to design changes

- Trying to describe everything in the assertion language
  — Properties need to be synthesized for formal verification
  — Lead to overly complex representation

- Creating assertions based too much on the internal signals of the design.
  — They may not behave correctly.

Recommendation

A mix of RTL + assertions may be the best approach to capture complex behavior.
Assertions: The Right Way

- Capture high-level functionality
  - Based on specification
  - Consist of
    - Functional properties
    - Interface constraints
    - Coverage goals

- Decomposition
  - Based on functional blocks
  - Generally hierarchical in nature

- Refinement
  - Assume-guarantee proof status
  - Simulate the constraints
  - Status of lower-level properties may affect upper-level properties
Formal Engines: The Wrong Way

- Assuming formal works the same way as simulation
  - Simulation: exercises events sequentially
  - Simulation: the same engine on multiple servers
  - Formal: examines events concurrently
  - Formal: multiple engines on multiple servers

- Throwing all (too many) properties to the tool
  - Insufficient formal time was spent with each property
  - Using the right formal engine on a few properties

- Assuming the best engine(s) for one design is the best for all
  - Sometimes true, but most often not

Recommendation

*Understanding the formal complexity of the properties is the best way to lead the formal verification process.*
By limiting the amount of time formal spent on the difficult assertions P2, P3, and P4, the simple assertions P1, P5 and P6 were verified in the first pass.

Verified assertions are used for fan-in cone pruning and to aid formal in the next pass.
Coverage: Formal-Specific Issues

- **Assertion Density**
  - Number of assertions per 100 lines of code
  - Anywhere from 1-10 based on design complexity
  - Reported early by formal tool

- **Minimum Sequential Distance (MSD)**
  - Number of flops between assertions
  - Identifies registers that are not covered by any assertion
  - Reported after design elaboration by formal tool

- **Cone of Influence**
  - The logic that is functionally covered by the assertion
  - Examine the fan-in cone of logic for each assertion
  - Reported after formal analysis
Formal Coverage: Minimum Sequential Distance

Measures how well you are implementing ABV

- Sequential distance between observation points
- Heuristic measure of functional vs. assertion complexity
- Blind spots due to inadequate assertion coverage

Register coverage aggregated for every module instance
Formal ABV Capability Maturity Model

The ABV Capability Maturity Model is a process reference model

Used by organizations to assess:
  — Current capability to implement an integrated ABV methodology
    - Formal Verification
    - Assertions
    - Advanced coverage
  — Current capability for predictability in terms of schedule and quality

The model helps the organization answer the following questions:
  — What **value** will be realized by increasing the organization’s maturity level?
  — What **infrastructure** is required to move to the next level of maturity?
  — What people **skills** and process management **resources** are required?
Formal ABV Capability Maturity Model

Capability levels represent the ABV maturity of engineering organizations.

**Level 1**
- Ad-hoc assertions
- Some interest
- Not coordinated

**Level 2**
- Planned & tracked
- ABV within design
- Used in simulation
- Automatic Formal Applications
- Improved debugging time
- Improved structural quality

**Level 3**
- Assertions
- Bug-hunting of assertions at block level
- Focus on high-value low-effort blocks *(bounded formal!)*
- Improved functional quality and schedule

**Level 4**
- Targeted proofs
- High-level requirements proved on targeted blocks
- Unbounded formal proofs (high-value and higher-effort)
- Improved quality

**Level 5**
- Optimizing
- Higher-level assertions proved on blocks
- Eliminate block-level simulation when possible
- Fully proved blocks integrated with system-simulation
- Improved schedule *(focus is on optimizing processes)*
Oracle: Evolution of Formal Plan: 1.0

“What can formal model checking do for us?”

- Static Clock Domain Checking
- SoC Interconnect Verification
  - Interrupts
  - DFT signals
**Oracle: Evolution of Formal Plan: 2.0**

"We think formal model checking can help us!"

- IP Assurance

- Replacing simulation for unit verification
  - SRAM Controller
  - Event Monitors
  - System Interrupt Controller
  - Arbiters

**CMM Level 3**
Oracle: Evolution of Formal Plan: 3.0

"Where else can we leverage formal model checking?"

- Bug Hunting
- Augmenting simulation to accelerate IP verification closure
  - Caches
  - Fuse Ctrl <-> MBIST Interface
  - Clock Control Unit
Oracle: Today’s Formal Plan of Record

- A key pillar for RAPID SoC Verification
  — More aggressive, and yet, practical plans
- Design for Formal Verification
- Optimal mix of Simulation + Formal
- Broader deployment with more users
For the Full Story …

- Verification Horizons, March 2015: *Evolving the Use of Formal Model Checking in SoC Design Verification*, By Ram Narayan, Oracle
  

- Verification Academy *Maturing a Project’s Formal ABV Process Capabilities*
  
  [https://verificationacademy.com/sessions/maturing-abv-process-capabilities](https://verificationacademy.com/sessions/maturing-abv-process-capabilities)
# Questa Formal Solutions & Apps

**Automated, Exhaustive Verification For Complex Challenges**

<table>
<thead>
<tr>
<th>Design</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CDC Solutions</strong></td>
<td><strong>Verification Apps</strong></td>
</tr>
<tr>
<td>Power-Aware CDC</td>
<td>CoverCheck</td>
</tr>
<tr>
<td>CDC-FX</td>
<td>X-Check</td>
</tr>
<tr>
<td>Reset Check</td>
<td>Security Check</td>
</tr>
<tr>
<td>CDC</td>
<td>AutoCheck</td>
</tr>
</tbody>
</table>

**Questa Formal Engines**
Summary

- It’s critical to provide clear guidance to formal novices so they will not get too frustrated and “revert” back to sims.

- The methods reviewed in this presentation aim to start novices off on the right foot with formal.

- The Oracle case study shows complete novices can “cross the chasm” starting w/apps and solid fundamentals.

- Share your favorite “righteous” formal tips! [LinkedIn] > [Experts in Assertion-Based Verification] https://www.linkedin.com/grp/home?gid=3925032