Fault Injection & Formal – Made for Each Other

Iain Singleton
June 2018
Introduction

• Formal usage has been growing rapidly in recent years

• Formal Technology has evolved
  – Lots of new metrics

• Question: Which should we be using and what for?
Cone of Influence Based Structural Analysis

- Simplest Formal Metric
  - Quick and easy to run
  - Identifies holes in verification
  - Does not prove anything inside COI

- First pass sanity check

- Verification signoff
Overconstraint Analysis

- Identifies unreachable code
  - Easy to run
  - Determines proofs are genuine
  - Does not measure what is being checked

- Important part of signoff ✓
- Good enough to signoff ❌
Formal Core Based Analysis

- Reports logic that has been checked by Formal
  - More accurate than COI metric
  - Identified holes in verification
  - Reports logic truly involved in a given proof

- Robust metric – ok to use for signoff

- Ensures everything has been checked
Fault Injection

- Injects the RTL faults, checks if assertions can catch all faults
  - Highest level of verification
  - If no assertion fails – RED FLAG

- Formal Signoff ✓
Fault Injection – In a Nutshell

- **Automatically** inserts “artificial bugs” called **faults** into the design
- Runs verification process on “broken” design
- Measures the ability of the environment to activate, propagate, and detect faults
Fault Injection with Formal Verification

• Modifies the design code to insert faults

```plaintext
o1 = f(i1) → o1 = 1'b0  // tie to constant
```

```plaintext
if (a) → if (TRUE)  // force execution of “if” branch
  f1();
else
  f2();
```

```plaintext
a = b | c → a = b & c  // change operator
```

• Passes the broken design to formal verification
  – Does at least one assertion fail? **OK!**
    – There are good quality assertions to detect that the design is broken
  – Do all assertions pass? **Issue!**
    – Problem: At least one assertion should have failed
Verification Made Easier with VC Formal Apps

**Auto Checks**
Formal Aware Structural Design Analysis
Easy setup and comprehensive checks

**Formal Coverage Analyzer**
Native integration in VCS
Common coverage debug with Verdi

**Register Verification**
Validate correctness of configuration registers against specifications

**Connectivity Checking**
Highest capacity for largest SoC’s
Up to 8x faster than competition

**Sequential Equivalence**
Catches bugs missed by other tools
Faster setup with auto helper discovery

**Property Verification**
High Performance and High Capacity
Property Convergence

**Security Verification**
Identify Security Vulnerabilities

**Assertion IP**
Validate correctness of standard protocols

**Navigator**
Design and Property Exploration in GUI

**Formal Testbench Analyzer**
Formal testbench completeness
High performance fault injection & analysis

**AEP**
**FCA**
**FRV**
**FPV**
**SEQ**
**NAV**
**FTA**
**AIP**

© 2018 Synopsys, Inc.
Formal Testbench Analyzer (FTA)

VC Formal

RTL -> Instrument Fault (Certitude) -> Formal Model with faults

Property & Constraints

Prove

At least one proof fails?

All proofs pass?

OK

VE HOLE

Property & Constraints

Prove

At least one proof fails?

All proofs pass?

OK

VE HOLE

Property & Constraints

Prove

At least one proof fails?

All proofs pass?

OK

VE HOLE
Performance Considerations

• Lots of ways that runtime can be optimized
  – Native integration of Certitude into FTA app allows for faster turnaround time
  – Pruning of COI ensures only relevant set of assertions run per fault
  – Native server farm support

• Last stage of Formal signoff
  – Using other metrics should have covered other holes
  – Most checks therefore expected to fail and fail quickly
A Match Made in Heaven (...Well, Mountain View)

- Formal is exhaustive
  - All we need to do is ensure the assertions we have are asking the right question

- Any bug that can't be caught indicates we aren't asking the right questions
  - Assertions need adding/changing

- VC Formal FTA optimized the numbers of faults without any loss of efficacy

- VC Formal Native FTA provides ~5X speedup
# Signoff Summary

<table>
<thead>
<tr>
<th>Metric</th>
<th>Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>COI based</td>
<td>First pass analysis of holes in verification</td>
</tr>
<tr>
<td></td>
<td>- Quick results</td>
</tr>
<tr>
<td></td>
<td>- Coarse analysis</td>
</tr>
<tr>
<td>Overconstraint analysis</td>
<td>Ensure no part of design is overconstrained</td>
</tr>
<tr>
<td></td>
<td>- Checks quality of <strong>results</strong></td>
</tr>
<tr>
<td></td>
<td>- Doesn’t check quality of <strong>checks</strong></td>
</tr>
<tr>
<td>Formal core</td>
<td>Robust formal analysis</td>
</tr>
<tr>
<td></td>
<td>- Checks that all logic is covered by checks</td>
</tr>
<tr>
<td></td>
<td>- Not full analysis of check quality</td>
</tr>
<tr>
<td>Formal Testbench Analyser</td>
<td>Fault injection to ensure check quality</td>
</tr>
<tr>
<td></td>
<td>- Ensures checks can catch bugs</td>
</tr>
<tr>
<td></td>
<td>- Checks quality of exhaustive checks</td>
</tr>
</tbody>
</table>
Thank You