Let's be Formal While Talking About Verification Quality: A Novel Approach Of Qualifying Assertion Based IPs

Surinder Sood, Sachin Scaria, Erik Seligman
AGENDA

• Introduction: VIP & The 3 Cs
• Checking Consistency : Self-FPV
• Checking Completeness : Fault Injection + FPV
• Conclusions and Future Work
AGENDA

• Introduction: VIP & The 3 Cs
• Checking Consistency: Self-FPV
• Checking Completeness: Fault Injection + FPV
• Conclusions and Future Work
Introduction: VIP & The 3 Cs

• Verification IP (VIP) is increasingly critical
  – Ideally supports both formal & simulation
  – Our emphasis in on formal

• Major requirements for VIP:
  – Correctness: VIP == spec?
    » Beyond the scope of this talk
  – Consistency: VIP fits together & allows good behaviors?
  – Completeness: VIP flags all bad behaviors?

• Consistency & Completeness: Use Formal Property Verification (FPV)
Assertion-Based VIP Structure

- **Assumptions/Constraints**: limit allowable input activity
- **Assertions**: conditions that must be true
  - Failing assertion flags error (in simulation or formal)
- **Covers**: conditions that must be tested
  - Missed cover in simulation == need more testing
  - Missed cover in formal == overconstrained

... + modeling code (queues, scoreboards, …)
AGENDA

• Introduction: VIP & The 3 Cs
• Checking Consistency: Self-FPV
• Checking Completeness: Fault Injection + FPV
• Conclusions and Future Work
Checking Consistency: Self-FPV

Assumptions/Constraints must allow known legal behaviors
Covers == core concepts, spec waveforms, known corner cases
AGENDA

• Introduction: VIP & The 3 Cs
• Checking Consistency: Self-FPV
• Checking Completeness: Fault Injection + FPV
• Conclusions and Future Work
Fault Injection + Formal Property Verification

– Core intuition: *Test the testbench*
– **Fault injection**: insert common faults in RTL + verify
  • Stuck-at, inversion, etc.
  • Does testbench detect the fault?

– Commonly used in simulation
  • Well-known solutions on market for years
  • Insert faults, check if simulation detects

– Use with Formal Property Verification less mature
  • But it’s the same concept! (Sim checkers == FPV assertions)
The Fault Injection Flow

1. Get RTL
2. Inject Faults
3. FPV
4. Review Fails
5. Debug Scenarios
6. Property Missing
7. Add Property
8. Consistency Check
9. Fix Property
10. Property Wrong
AGENDA

• Introduction: VIP & The 3 Cs
• Checking Consistency : Self-FPV
• Checking Completeness : Fault Injection + FPV
• Conclusions and Future Work
Completeness & Consistency: Do Both With FPV!

<table>
<thead>
<tr>
<th>Completeness: Fault Injection + FPV</th>
<th>Consistency: FPV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need one good RTL model</td>
<td>No RTL needed, just VIP</td>
</tr>
<tr>
<td>Focus == unchecked conditions</td>
<td>Focus == overconstraint</td>
</tr>
<tr>
<td>Discover new properties</td>
<td>Focus on written properties</td>
</tr>
<tr>
<td>Not available at early stages of VIP</td>
<td>Always available</td>
</tr>
<tr>
<td>Useful coverage measurement</td>
<td>Relies on hand-coded coverage</td>
</tr>
<tr>
<td>Very general— all classes of VIPs</td>
<td>Best for protocols/bridges</td>
</tr>
</tbody>
</table>
Issues Found

• Consistency: 2 bogus waveforms in protocol spec
  – Text not consistent with pictures, due to various edits since 1.0
  – Potentially major source of designer confusion

• Completeness: 1500+ faults injected
  – 140 “non-activated” or “non-detected”
  – Numerous behaviors not monitored in VIP: added assertions!
  – Some reset-related assertions not quite correct
Conclusions

• **Consistency thru FPV:  great for early VIP checking**
  – Very low cost since just requires light layer on VIP
  – Cover points (not just asserts/assumes) are important enabler
  – Can’t address correctness or completeness
    • Don’t be over-exuberant about ‘FPV passing’
• **Completeness thru fault injection + FPV:  powerful followup**
  – Can’t do early:  need at least one RTL customer model
  – Finds critical omissions in VIP design
  – Finds holes in your user-written coverage
  – More usable than ‘real’ formal completeness
Future Work

• Flow Improvements
  – Injection of faults followed by FPV: a bit clunky
  – Opportunities for more integrated tools?
• Comparison of various forms of “Completeness”
  – Fault-injection: intuitively easy, straightforward debug
  – Formal coverage: possibly more powerful, usability improving
    • In practice, will these be redundant, or complimentary?
    • What about new, advanced tools?