Portable Stimulus vs Formal vs UVM

A Comparative Analysis of Verification Methodologies Throughout the Life of an IP Block

Gaurav Bhatnagar
12/12/2017
Introduction

• DV Challenges
  – Creating all possible scenarios to test a design
  – Achieving a 100% Coverage
  – Shrinking time to market
  – Under Pressure to Do More, Faster, and with Less!
• Proven Methodologies UVM based and Formal Verification
• Upcoming PS based Verification Methodology
• Choice of Verification Methodology based on requirements
• Comparative Analysis
Life cycle of an IP Block

• Block Level Verification
  – Focus on micro-architecture
  – Exhaustive test conditions using UVM or Formal techniques

• SoC Level Verification
  – Focus on connectivity checks
  – Integration and Directed testing using C and SV based tests

• Silicon Validation
  – Focus on Customer Scenario
  – Use Case testing using C/C++ based platforms
Life cycle of an IP Block

1. **Block Design**
   - Block Verification
     - UVM
     - Formal
     - PS

2. **SoC/System Integration**
   - SoC Verification
     - UVM
     - Formal
     - PS

3. **Post Silicon Process**
   - Silicon Validation
     - UVM
     - Formal
     - PS

4. **Reuse Possibility**
   - **100% Reuse Possible**
   - **Partial Reuse Possible**
   - **No Reuse Possible**
UVM Based Verification Flow

1. Verification Plan
2. Create Directed/Random Test
3. Run Simulation and Collect Coverage
4. Functional Coverage
5. Coverage Analysis

Decision:
- Is Code/Functional Coverage 100%?
  - Yes: FINISH
  - No: Go back to Verification Plan
AHB2APB UVM based Verification
AHB2APB UVM Verification

• Pre-verified IP block upgraded with a glue logic
• AHB to APB conversion with additional control and debug logic
• AHB Master UVC, APB Slave UVC and Glue Interface UVC
• Directed and Random tests
• Regressions are run and reports are generated
• Functional and Code Coverage
• No Bugs Found
### AHB2APB UVM Verification

<table>
<thead>
<tr>
<th>Initial Setup</th>
<th>Directed Tests</th>
<th>Random Test</th>
<th>Coverage Closure</th>
<th>Overall Development Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 week</td>
<td>2 weeks</td>
<td>1 weeks</td>
<td>2 weeks</td>
<td>6 weeks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tests Run</th>
<th>Passed</th>
<th>Failed</th>
<th>Not Run</th>
<th>Overall Code Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>106</td>
<td>106</td>
<td>0</td>
<td>0</td>
<td>2634/2864</td>
</tr>
</tbody>
</table>
Formal Verification: Coverage Types
Formal Verification Flow

1. Formal Verification Plan
2. Create Assumption and Assertion
3. Analyze and Elaborate
4. Prove and Collect Coverage

- **Is Stimuli/COI Coverage Complete?**
  - Yes
    - FINISH
  - No
    - Unreachability Analysis

- **Is Proof/Bounded Coverage Complete?**
  - Yes
    - FINISH
  - No
    - Coverage Analysis
AHB2APB Formal Verification Setup

AHB Master VIP

AHB2APB DUT

AHB Slave If

APB Master If

Glue If

Glue If Assertions

APB Master VIP
AHB2APB Formal Verification Results

- AHB and APB ABVIP for interface assertions
- Increased Verification Quality due to standard components
- User defined assertions specific to Glue Logic
- **BUG!!** found related to Glue Logic
- Unwanted switching of logic found
- Glue logic driver limitation in UVM environment found
- The UVM based environment also finds the error after the fix
- Random test and code coverage analysis in UVM environment
# AHB2APB Formal Verification Results

<table>
<thead>
<tr>
<th>Initial Setup</th>
<th>Unreachability Analysis, ABVIP Assertion and Debug</th>
<th>Manual Assertion Coding and Debug</th>
<th>Analysis and Coverage Closure</th>
<th>Overall Development Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 week</td>
<td>1 weeks</td>
<td>1 weeks</td>
<td>2 weeks</td>
<td>5 weeks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total number of COI Items</th>
<th>Unreachable Waived Items</th>
<th>Undetermined</th>
<th>Bounded and Waived</th>
<th>Proof Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>106</td>
<td>106</td>
<td>0</td>
<td>0</td>
<td>2634/2864</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proven</th>
<th>Unprocessed</th>
<th>Processing</th>
<th>Failed</th>
<th>Passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>483</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>483</td>
</tr>
</tbody>
</table>
What is Portable Stimulus?

• **New standard** defining new test writing language for Portable Tests
• Vertical Reuse
  – Tests developed at the IP level easily integrated/reused at SOC level
• Horizontal Reuse
  – Simulation, Emulation, board level, tester etc.
• Bi-Directional Re-Use
  – Evaluation Board Failure to IP Test
  – Reuse S/W drivers in simulation

• Supported by Breker Trek, Cadence Perspec and Questa Infact
PS based Development Process

- **Architect**
  - Use Cases, PSS Models

- **IP Design/DV**
  - IP DUT, PSS Models

- **System Design/DV**
  - SUT, PSS Models

- **Apps Eval**
  - Scenarios

- **Software**
  - Device Drivers

- **Test**
  - Scenarios

**PSS Framework**

- **TLM Model**
  - SystemC

- **Simulation**
  - SV, UVM
    - C/C++

- **Evaluation, Customer Boards**
  - C/C++

- **Virtual, Emulator, Silicon**
  - C/C++

- **Tester**
  - Multiple
Generating tests from PS Flow

- PS Model Creation
- Tool Configuration
- Constraints Creation

PS Tool Interface
- PS based Compiler
- PS based Graph Generator

Generate Tests and Graph based Coverage

Setup environment, run tests and debug
PS based Verification Flow

Verification Plan → Create Constraints and PS model → Generate Tests → Collect Graph based Coverage

- Is Graph based Coverage Complete?
  - Yes: FINISH
  - No: Coverage Analysis

- Is Code/Functional Coverage 100%?
  - Yes
  - No: Collect Coverage

- Run Simulation and Collect Coverage
AHB2APB PS based Verification Setup

PS doesn’t replace UVM!!

AHB2APB UVM Environment

VIRTUAL SEQUENCER

AHB Master UVC

AHB2APB DUT

SCOREBOARD

GLUE IF UVC

PORTABLE STIMULUS MODEL

TX Gen

C Gen

PS Test Sequence

UVM SV Setup

DPI CALLS

VIDEO INTERFACE

GLUE IF

UVM SV Setup

IP model is reusable

Works at higher layer of abstraction

APB SLAVE UVC
TEST GENERATION AND DEBUG

Parallel execution of tests

Each test block corresponds to C/SV sequence

PS TEST

Pipeline 1
- Seq 1
- Seq 4
- Seq 5
- Seq 8

Pipeline 2
- Seq 2
- Seq 3
- Seq 6
- Seq 7

Color Coding to represent state of test

Debugging with the help of linked log file, field values and memory
GRAPH COVERAGE
REGRESSION ANALYSIS

- SV Random vs PS Ransom comparison
- AHB2APB Gasket Regression with 10 seeds
- Exact same number of sequences run
AHB2APB PS based Verification Results

• PS Models generated tests covering more conditions
• The number of tests in a regression required to achieve same coverage results is reduced by 50%!!
• Time required to bring up the verification environment, code the models and generate tests is 6 weeks
• The bug found in case of Formal Verification took less number of regressions to hit
• Coverage Closure is easier due to high quality tests
### AHB2APB PS based Verification Results

<table>
<thead>
<tr>
<th>Initial Setup</th>
<th>Initial Model Development</th>
<th>Test Generation, Graph Coverage and Run</th>
<th>Coverage Closure</th>
<th>Overall Development Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 week</td>
<td>2 weeks</td>
<td>1 weeks</td>
<td>2 weeks</td>
<td>6 weeks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tests Run</th>
<th>Passed</th>
<th>Failed</th>
<th>Not Run</th>
<th>Overall Code Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>2634/2864</td>
</tr>
</tbody>
</table>
AHB2APB SoC PS Verification Setup

Generated C Tests based on Model

SoC Verification Environment

SoC Infrastructure

Processor

AHB2APB

AHB Monitor

Peripheral Set 1

Peripheral Set 2

AHB2APB

AHB Monitor

APB Monitor

AHB2APB

AHB Monitor

APB Monitor

PS Test Generation

TX Gen

C Gen

PS Test Sequence

PS Model from IP level reused at SoC level
AHB2APB SoC PS Verification Results

- PS Models can be used to generate C tests
- The C tests can be compiled and run on the processor for SoC simulation
- The tests can be integrated with Post Si Evaluation board setup to be run with different debuggers and tools
- The tests can be mapped to different instances of the AHB2APB VIP
- The tests inherit the same test quality as seen at the IP level
- Integration **BUG!** was uncovered using the IP based models at the SoC level
Post Si Evaluation with PS

- UVM and Formal can’t be used
- PS Models can be targeted for C test generation for Post Si Validation
- Integration with common debug platforms
- High quality, self checking constrained random tests
- Debug Interface for generated tests
- Failures on Post Si Evaluation Board can be ported back to simulation
## Comparative Analysis

<table>
<thead>
<tr>
<th>Point of Comparison</th>
<th>PS vs UVM</th>
<th>PS vs Formal</th>
</tr>
</thead>
</table>
| **Verification Planning and Development Time** | • Planning and Development time is very similar  
  • Overhead of learning new language but it is very similar to C++  
  • More detailed verification plan required for Formal  
  • Lesser development time for Formal  
  • Availability of ABVIP considerably reduces the verification complexity |                                                                                       |
| **Infrastructure Development and Ease of integration** | • Integration logic is required both for model and SV code  
  • The integration logic is reusable across common interface  
  • Setup time will vary depending on the design complexity for Formal  
  • Easier for Formal due to lesser complexity  
 |                                                                                       |
| **Effective test generation**        | • Visual representation of tests  
  • High Quality tests covering more conditions  
  • Self Checking tests  
  • Visual representation of tests in PS  
  • Tool generated stimulus in Formal covers most conditions easily |                                                                                       |
## Comparative Analysis

<table>
<thead>
<tr>
<th>Point of Comparison</th>
<th>PS vs UVM</th>
<th>PS vs Formal</th>
</tr>
</thead>
</table>
| **Coverage Closure and Report Analysis** | • Graph based Coverage   
• Easy to identify uncovered graph conditions   
• More coverage with lesser number of runs | • Code, Functional and Graph based Coverage in PS  
• COI, Stimuli, Proof and Bounded Coverage in Formal |
| **Reuse and Portability**            | • Greater reuse possible especially when switching target platforms     | Greater reuse possible especially when switching target platforms           |
Conclusion

• PS based Verification has an edge over UVM based verification
  – Better test quality
  – Reuse across different target platforms

• Formal Verification offers an effective verification approach and a clear winner at the IP level

• All the three verification methodologies are self sufficient and can provide a High Quality Verification
THANK YOU