Accelerated, High Quality SoC Memory Map Verification using Formal Techniques

Cletan Sequeira
Rajesh Kedia
Lokesh Babu Pundreeka
Bijitendra Mittra

1 Texas Instruments India Private Limited
2 Cadence Design Systems Incorporated
Outline

• Introduction
• Prior Work
• Proposed solution
  – Use of Assertion Based VIPs
  – Memory Map verification
    • Features/Flavors
• Consolidation of features being verified
• Advantages
• Results
• Conclusion
Introduction

• Our SoC contains a complex bus-system comprising:
  – 4 AHB masters.
  – Multiple AHB, APB and TI custom protocol based slaves.
  – Different stages of bus-fabric, bridges and decoders.

• Traditional verification flow suffer from below challenges:
  – Dependency on test case flow and chances of corner cases being missed.
  – Exhaustive verification leading to huge test development and runtime.
  – Conditions like “CPU getting stalled when sweeping through the whole memory map” could not be checked easily.

• SoC memory map verification using formal techniques was implemented:
  – To increase the exhaustiveness of verification.
  – To improve the development time.
  – To improve the runtime.
Prior Work

- SoC level checks typically done using C or Assembly based directed test cases.
  - Exact use-case exercised.
  - Not exhaustive in nature.
  - Require significant time to develop, simulate and debug.
- Formal used for standalone systems like IPs and Bridges + directed tests at SoC.
  - Provides a high quality at unit level.
  - Does not ensure the whole system to work correctly.
- Use of emulation platforms as accelerator.
  - Happens late in the development cycle.
  - Still it may not fully guarantee all scenarios being covered.
Proposed Solution – Overview

- Hook up Assertion based Verification IPs (VIPs) to various masters and slaves:
  - AHB assertion VIPs\(^1\) to each AHB master and slave.
  - APB assertion VIPs\(^1\) to each APB slave.
  - TI custom protocol based VIP\(^2\) to each custom slave.

- Properties in these VIPs are controlled precisely:
  - For masters, the master properties are made constraints and slave properties as assertions.
  - For slaves, the slave properties are made constraints and master properties as assertions.

- Develop memory map related functional properties and bind them to above VIPs
  - These will verify SoC memory map related aspects.

\(^1\)Cadence VIPs used
\(^2\)Coded in-house
The same property file can act as checker or BFM via TCL control as shown in the example below:

- Assume property `prop1: (!hresetn_i) -> (hready)`: an AHB protocol check

  - `assert prop1` in TCL → Property becomes an assertion and behaves as AHB slave checker. Whenever hresetn_i is ‘0’, it will check that hready is ‘1’ else flag an error.

- `assume prop1` in TCL → Property becomes a constraint and behaves as AHB slave. Whenever hresetn_i goes ‘0’, it will force hready to ‘1’.
ABVIP Hookup in our System

1. Slave properties as constraints.
2. Master properties as assertions.
## Automation for ABVIP Hookup

<table>
<thead>
<tr>
<th>MASTER</th>
<th>SLAVE</th>
<th>START ADDRESS</th>
<th>END ADDRESS</th>
<th>SIZE</th>
<th>Sel_Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2#M0</td>
<td>S31</td>
<td>SA31</td>
<td>EA31</td>
<td>SZ31</td>
<td>H1.pin1</td>
</tr>
<tr>
<td>M2#M0</td>
<td>S41</td>
<td>SA41</td>
<td>EA41</td>
<td>SZ41</td>
<td>H2.pin2</td>
</tr>
<tr>
<td>M0#M1#M3</td>
<td>S0</td>
<td>SA0</td>
<td>EA0</td>
<td>SZ0</td>
<td>H3.pin3</td>
</tr>
<tr>
<td>M3#M1</td>
<td>S1</td>
<td>SA1</td>
<td>EA1</td>
<td>SZ1</td>
<td>H4.pin4</td>
</tr>
<tr>
<td>M2</td>
<td>S1</td>
<td>SA1m</td>
<td>EA1m</td>
<td>SZ1m</td>
<td>H4.pin4</td>
</tr>
</tbody>
</table>
Write Path Checks

Constraint:
Address = S1 and Write access => HWDATA = WDATA_S1

Assertion: Valid Write access => HWDATA = WDATA_S1
**Read Path Checks**

**ARM CM4 Sub-system**

- AHB VIP(M0)
- AHB VIP(M1)
- AHB VIP(M2)

**DMA**

**Assertion:**

Address = S1 and Read access => HRDATA = RDATA_S1

**Constraint:**

Valid Read access => HRDATA = RDATA_S1
Reserved Space Checks

Assertion:
Address = Reserved and Read Access =>
HRDATA = 0x00000000
Assertion:
Any Illegal Access => HRESP = 1 in next cycle

Illegal Access:
1. Access to invalid addresses
2. Access for higher width data for smaller width slaves
Master Priority Checks

Constraint:
All Masters access the same slave with different HWDATA

Assertion: Valid Write access => HWDATA = WDATA_M0 (highest priority)
Finally what gets verified

- Adherence of interconnect logic (bus fabric, decoders, bridges) to various protocols (AHB, APB, TI custom).
- Integration (Connectivity) of bus fabric with masters and slaves.
- Functionality of bus fabric, bridges and decoders.
- Accessibility and address mapping for each slave from various masters.
- Priority checks for each of the masters.
- Reserved space access behavior.
- Mirroring of Slave at different addresses for 2 different masters.
- Error response checks for all the invalid addresses.
- Error response checks for slaves not supporting 32-bit accesses.
Advantages

• Only Assertions used for SoC memory map verification.
• Flow can work directly on the integrated RTL without changes.
• Can catch bugs early in development phase.
• Easily portable and customizable for various devices.
• Advantages inherent with FV that comes along:
  – Higher confidence on verification quality.
  – No disk-space issues because debug in IFV doesn’t require waveforms to be dumped unlike in simulations.
  – Debug is quite simple – the exact (or sometimes close) cone of signals added to the waveform automatically.
Results

- Few corner case bugs were caught for the bridges.
- Verified access for all slaves with all possible masters very easily unlike in simulation.

<table>
<thead>
<tr>
<th>assertions</th>
<th>protocol checks + memory map checks</th>
<th>only memory map checks</th>
</tr>
</thead>
<tbody>
<tr>
<td>total assertions</td>
<td>1300</td>
<td>500</td>
</tr>
<tr>
<td>assertion passed</td>
<td>1268</td>
<td>500</td>
</tr>
<tr>
<td>assertion failed</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>fv runtime (hrs)</td>
<td>16</td>
<td>5</td>
</tr>
</tbody>
</table>

Flow Deployment

<table>
<thead>
<tr>
<th>device</th>
<th>approximate bring-up effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>first device</td>
<td>~ 6 weeks</td>
</tr>
<tr>
<td>subsequent devices in family</td>
<td>~1 week</td>
</tr>
</tbody>
</table>
Example of bugs caught
Looking Back!

• What helped:
  – Following naming conventions for the design signals helped in automating the flow.
  – Availability of ABVIPs for AMBA™ protocols.
  – Debug features in the tool.

• Challenges faced:
  – Black Boxing unwanted modules.
  – Finding right design constraints.
  – Choosing the right solver.
Conclusion

• FV based Memory Map verification flow has been described, which:
  – Reduces run time by a huge margin compared to simulations.
  – Provides more exhaustiveness and hence higher confidence on the verification quality.
  – Allows easy deployment in subsequent devices due to automated & configurable options in the flow.
  – Being generic, could be used with any protocol.

• Future Scope
  – Enhance the flow by using formal scoreboard for data integrity checks.
  – Replace each slave ABVIP with the corresponding RTL to gain further confidence on the whole system.
References

• Cadence Incisive Formal Verifier User Manual
• Cadence Assertion Based Verification IP User Guide
• AMBA™ 3 Specification Rev 1.0, http://www.arm.com
Thanks