Coherency Verification & Deadlock Detection Using Perspec/Portable Stimulus

Moonki Jang – Samsung Electronics Co., Ltd.
Phu Huynh – Cadence Design Systems, Inc
Agenda

• Why coherency and deadlock detection verification
• Requirements & description of our verification environment
• Using Perspec and PSS to create reusable test suite
Trends

• Latest automotive & mobile SoCs need higher performance and incorporation of additional functionality

High-End Octa core SoC (Samsung, 2013)

Latest ADAS reference platform (ARM, 2018)
Side effects

• The need for higher performance and additional functionality has increased the complexity of coherency networks every year
  – Increased complexity always causes unexpected problems like resource conflicts between multiple coherent masters when working concurrently
Agenda

• Why coherency and deadlock detection verification
• Requirements & description of our verification environment
• Using Perspec and PSS to create reusable test suite
Requirements

• Reasons why it is important to detect deadlock-related coherency issues at the pre-silicon level:
  – Hard to reproducing deadlock on silicon environment
  – Transaction flow tracking is impossible
  – Root cause analysis will be extremely difficult on silicon
Considerations

• Reasons why coherency issue detection is difficult at the pre-silicon level:
  – Slow simulation speeds make it difficult to perform complex scenarios
  – For reproducing the specific target conditions, pre-silicon scenarios should narrow down the scenario scope.
  – It requires much effort and time to create such a complex scenarios

• PSS(Portable Stimulus Standard) verification environment was introduced to solve these issues
PCIe deadlock verification

• The following two blocking conditions cause a PCIe deadlock
  – ACE (AXI Coherency Extensions) master blocking condition
    • ACE master might have to complete a WriteBack transaction of a similar operation before it can respond to a snoop request
  – PCIe blocking condition
    • A PCIe bridge can stall reads and PCIe configuration writes on its AXI slave interface when write transactions from its AXI master interface is stalled
PCIe deadlock verification

- Reproducing scenario should make below conditions
  1. Fill up the PCIe RC request queue using a non-posted read from CPU
  2. After RC request queue is full, additional non-posted write should block the write channel
  3. CPU will generate WriteBack for address A
  4. PCIe EP will generate WLU for address A
Overview of Our Verification Env

• Our test suite can detect deadlock and gather system information

- **System Monitor**
  System status check using ‘heartbeat response’

- **Transaction Latency Monitor**
  Records the latency of all transactions originating from the CPU(s)

- **System Tracker**
  Gathers the system information for root cause analysis and generates log files

- **Simulation Manager**
  Manages the whole process of creating and executing the Perspec scenarios for simulation and analyzes the results
Overview of Our Verification Env

- System Monitor
  - Each core generates a heartbeat to the system monitor
  - If system monitor does not detect the heartbeat in the expected period, it assumes that the system has fallen into a deadlock state.
Overview of Our Verification Env

• System Tracker & Transaction Latency Monitor
  – When deadlock is detected, system tracker will collect the system information from the latency monitor and system monitor.

  ![Snapshot of system register](image)
  Deadlock detection time
  List of the incomplete transactions
Overview of Our Verification Env

- Simulation Manager
  - If deadlock is reported in the log, simulation will be re-launched to get a dump file through regression manager’s post processing
  - At this time, the system information result that is generated in the system tracker can be used to specify the dump period

```bash
# postrun
if(! -e ./dump.gen ) then
  if(! -e ./sim.log ) then
    if(`found_match.pl -s ./sim.log -m 'TEST FAILED'` == 'found_match') then
      run-with_fsdb
    endif
  endif
endif
```
Agenda

- Why coherency and deadlock detection verification
- Requirements & description of our verification environment
- Using Perspec and PSS to create reusable test suite
Coherency Verification

• Perspec and PSS simplified and shortened our verification tasks
  – Basic (PSS) actions are provided by Perspec libraries
  – Coherency test suite is also provided
  – Coverage models are part of the Perspec libraries

• Overall verification process:
  – Create model of the compute (processor-memory) subsystem
  – Run sanity memory R/W tests
  – Run coherency test suite provided by Perspec library
  – Develop additional corner cases & stress test scenarios
Create Processor-Memory model

• Processor Info table

<table>
<thead>
<tr>
<th>Processor Info</th>
<th>#tag</th>
<th>#kind</th>
<th>#cluster</th>
<th>#cluster_id</th>
<th>#core_id</th>
<th>#coherency_level</th>
</tr>
</thead>
<tbody>
<tr>
<td>M0</td>
<td>MO</td>
<td>MO</td>
<td>MO</td>
<td>0</td>
<td>0</td>
<td>FULL</td>
</tr>
<tr>
<td>M1</td>
<td>MO</td>
<td>MO</td>
<td>MO</td>
<td>0</td>
<td>1</td>
<td>FULL</td>
</tr>
<tr>
<td>M2</td>
<td>MO</td>
<td>MO</td>
<td>MO</td>
<td>0</td>
<td>2</td>
<td>FULL</td>
</tr>
<tr>
<td>M3</td>
<td>MO</td>
<td>MO</td>
<td>MO</td>
<td>0</td>
<td>3</td>
<td>FULL</td>
</tr>
<tr>
<td>A0</td>
<td>AP</td>
<td>AP</td>
<td>AP</td>
<td>1</td>
<td>4</td>
<td>FULL</td>
</tr>
<tr>
<td>A1</td>
<td>AP</td>
<td>AP</td>
<td>AP</td>
<td>1</td>
<td>5</td>
<td>FULL</td>
</tr>
<tr>
<td>A2</td>
<td>AP</td>
<td>AP</td>
<td>AP</td>
<td>1</td>
<td>6</td>
<td>FULL</td>
</tr>
<tr>
<td>A3</td>
<td>AP</td>
<td>AP</td>
<td>AP</td>
<td>1</td>
<td>7</td>
<td>FULL</td>
</tr>
<tr>
<td>B0</td>
<td>A72</td>
<td>A72</td>
<td>A72</td>
<td>2</td>
<td>8</td>
<td>FULL</td>
</tr>
<tr>
<td>B1</td>
<td>A72</td>
<td>A72</td>
<td>A72</td>
<td>2</td>
<td>9</td>
<td>FULL</td>
</tr>
<tr>
<td>B2</td>
<td>A72</td>
<td>A72</td>
<td>A72</td>
<td>2</td>
<td>10</td>
<td>FULL</td>
</tr>
<tr>
<td>B3</td>
<td>A72</td>
<td>A72</td>
<td>A72</td>
<td>2</td>
<td>11</td>
<td>FULL</td>
</tr>
</tbody>
</table>
Create Processor-Memory model

• Memory Info table

<table>
<thead>
<tr>
<th>Memory Info</th>
<th>#mem_block</th>
<th>#enabled</th>
<th>#base_addr</th>
<th>#end_addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR0</td>
<td>TRUE</td>
<td>0x80000000</td>
<td>0x800000000</td>
<td>0x9FFFFFFF</td>
</tr>
<tr>
<td>DDR1</td>
<td>TRUE</td>
<td>0xA0000000</td>
<td>0xA00000000</td>
<td>0xBFFFFFFF</td>
</tr>
<tr>
<td>DDR2</td>
<td>TRUE</td>
<td>0xC0000000</td>
<td>0xC00000000</td>
<td>0xFFFFFFF</td>
</tr>
</tbody>
</table>

• Page Table

<table>
<thead>
<tr>
<th>#va</th>
<th>#pa</th>
<th>#size</th>
<th>#secure</th>
<th>#shareable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000000</td>
<td>0x80000000</td>
<td>0x2000000000</td>
<td>TRUE</td>
<td>outer_shareable</td>
</tr>
<tr>
<td>0xA0000000</td>
<td>0xA0000000</td>
<td>0x2000000000</td>
<td>TRUE</td>
<td>outer_shareable</td>
</tr>
<tr>
<td>0xC0000000</td>
<td>0xC0000000</td>
<td>0x4000000000</td>
<td>TRUE</td>
<td>non_shareable</td>
</tr>
</tbody>
</table>
Memory R/W Test - PSS

- Sanity test to ensure that basic processor-memory paths are working.
- Atomic actions provided by Perspec library:
  - `write_data`
  - `copy_data`
  - `read_check_data`
- Built-in data type:
  - `sml_processor_tag_e`
Memory R/W Test - Solutions

• Tests generated from previous PSS code
  – Solution 1: uses core A2
  – Solution 2: use core M2

• Memory blocks are also selected randomly by Perspec

Solution 1

Solution 2
Multi-core Memory R/W - PSS

- Three processor cores doing memory R/W test in parallel
  - Verify inter-processor (mailbox) communication

- Show how previous compound action (pss_wr_cp_rc) is used to build more complex PSS scenario

```plaintext
action pss_wr_cp_rc_3cores {
  pss_wr_cp_rc chain1, chain2, chain3;
  activity {
    parallel {
      chain1;
      chain2 with {
        proc_tag_l != chain1.proc_tag_l;
      };
      chain3 with {
        proc_tag_l != chain1.proc_tag_l;
        proc_tag_l != chain2.proc_tag_l;
      };
    }
  }
}
```
Multi-core Memory R/W - Solution

- In this solution, cores M2, B0, A2 were selected.

- Different memory blocks (DDR0, DDR1, DDR2) were also selected randomly.
Coherency Test Suite

• Perspec library includes a verification plan and a suite of tests
  – Focus on memory access, coherency, and low-power

• Verification Plan Outline:
  – Basic Memory Access
  – Exclusive Accesses
  – Coherency: basic coherency actions, false-sharing, true-sharing, cache states
  – Coherency with IO
  – DVM
  – others
Coherency Test Suite

- Scenarios coverage of memory access from different processor at different data sizes, alignments
- Coverage on lock type like Spin Lock, ticket lock etc, accessed in parallel from all processors
- False sharing and True sharing coverage across cores/clusters
- DVM coverage for ASID, VMID and VA-PA mapping
PCIe Library – RC & EP Actions
//Setup step: setup PCIe RC and PCIe EP ...

//...

**parallel** {  //randomly pick a core to do the following actions

  **do** `pcie_rc_mem_read` multiple times in sequence;  //core A3 in Fig 7
  **do** `pcie_rc_mem_read` multiple times in sequence;  //core A0
  **do** `pcie_rc_mem_read` and `invalidate_cache`;  //core A2
  **do** `pcie_rc_mem_read` and `pcie_config_write`;  //core A1

  //PCIe EP

  **sequence** {

    **do** `pcie_ep_vip_reg_write`;

    **do** `pcie_ep_mem_read` multiple times;

  };

};
Deadlock Verif Scenario – Solution

Core A3:
- pcie_rc_mem_read

Core A0:
- pcie_rc_mem_read

Core A1:
- pcie_rc_mem_read & pcie_rc_mem_read
- invalidate_cache

Core A2:
- pcie_rc_config_write
- pcie_rc_mem_read & pcie_rc_mem_read

EP VIP:
- pcie_vip_reg_write & pcie_ep_mem_read
Conclusions & Lessons Learned

• Writing PSS scenarios were much simpler than writing C tests manually
  – Library actions allow us to work at a higher-level of abstraction
  – Inter-processor communication is handled automatically by Perspec
  – Resource allocation is also handled automatically

• Project-to-project reuse requires more than PSS; need:
  – Libraries
  – Reuse methodology
Conclusions & Lessons Learned

- PSS and Perspec helped shorten our design verification
  - Below chart shows our coherency coverage closure status of recent project.

Through the scenarios generated in the PSS environment, the time taken to increase the coverage from 90% to 100% can be shortened by 50% compared to the previous project.
BACK-UP Slides
DVCon Slide Guidelines

• Use Arial or Helvetica font for slide text
• Use Courier-new or Courier font for code
• First-order bullets should be 24 to 28 point
  – Second-order bullets should be 24 to 26 point
    • Third-order bullets should be 22 to 24 point
    • Code should be at least 18 point
• Your presentation will be shown in a very large room
  – These font guidelines will help ensure everyone can read you slides!

No Company Logo except on title slide!
Code should be enclosed in text boxes (using a background color is optional)

Code should be 18pt Courier-bold, or larger

module example
(input logic foo,
 output logic bar
);

 initial begin
  $display ("Hello World!");

 endmodule

Informational boxes should be 18pt Arial-bold, or larger (using a background color is optional)