Formal Architectural Specification and Verification of A Complex SOC

Shahid Ikram, Isam Akkawi, David Asher, Jim Ellis
Outline

• The Problem Definition.
• Architecture.
• Formal Specification.
• Formal Verification.
• Results.
The Problem Definition

• Ever increasing complexity of SOCs.
• Moving to higher levels of abstraction helps.
• At higher levels of abstraction,
  – We may think in terms of subsystems.
  – Each subsystem have its own protocols.
    • These protocols need to be validated.
  – The subsystems interact with each other.
    • Protocol interaction verification.
  – The subsystem are implemented using micro-architecture.
    • Micro-architecture verification.
The Architecture
Formal Specification
## Extended Table-based Specification

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Outputs</th>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cmd</td>
<td>H</td>
<td>N1</td>
<td>Cmd</td>
</tr>
<tr>
<td>none</td>
<td>E</td>
<td>I</td>
<td>none</td>
</tr>
<tr>
<td>none</td>
<td>S</td>
<td>S</td>
<td>Local_Write</td>
</tr>
<tr>
<td>Local_Write</td>
<td>S</td>
<td>S-&gt;I</td>
<td>none</td>
</tr>
</tbody>
</table>
Agents, States, Roles and Messages

- A hardware architecture is defined with reference to an instruction set.
- An instruction’s definition may involve multiple subsystems’ execution.
- This execution is captured as protocol tables for each of the subsystems.
  - We may think of these subsystems as agents.
  - Agents can have different states and play different roles depending on the state.
  - Agents may need to send messages to each other.
  - We need to identify these messages:
  - Response to a message depends upon the current state of the agent.
Architectural Agents and Roles

Architectural Protocol

Home
- Local Request
- Local Response
- Local Forwarding

Remote
- Remote Request
- Remote Response
- Remote Forwarding

IOB
- PCIe Ordering
- PCIe Forwarding
## Input Column Combinations of An Agent Table

<table>
<thead>
<tr>
<th>Transient State</th>
<th>Current State</th>
<th>New Request Cmd</th>
<th>New Ack Cmd</th>
<th>New Data Cmd</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>I</td>
<td>I</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>WI00</td>
<td>I</td>
<td>K</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>CSFH</td>
<td>S-&gt;I</td>
<td>K-&gt;E</td>
<td>none</td>
<td>Remote PRE</td>
</tr>
<tr>
<td>ED00</td>
<td>I</td>
<td>D-&gt;E</td>
<td>none</td>
<td>Remote PRD</td>
</tr>
</tbody>
</table>
Micro-architecture

• A Micro-architecture represents a possible implementation of an architecture.
• Each architectural instruction is executed through a set of micro-operations.
• These micro-operations are executed inside the time-frame of one architectural steps and hence are combinational in nature.
• The micro-operations’ execution has certain ordering requirements.
• We extended our architectural tables with micro-architectural steps.
A Generic Architecture
# Micro-Architecture Steps

<table>
<thead>
<tr>
<th>R0..R7</th>
<th>Mem Reads</th>
<th>Mem Writes</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>000xx</td>
<td>0x000</td>
<td>W00S0</td>
<td>R0..R7-&gt;T, T-&gt;Bus, Bus-&gt; (MDR,MAR) MDR-&gt;Memory</td>
</tr>
<tr>
<td>100xx</td>
<td>0x000</td>
<td>0V000</td>
<td>IR -&gt; Bus, Bus-&gt;MAR, MAR-&gt;Memory,…</td>
</tr>
<tr>
<td>0W0xV</td>
<td>0x000</td>
<td>0V0S0</td>
<td>R0..R7-&gt;ALU,….,(MAR,MDR)-&gt;Memory</td>
</tr>
<tr>
<td>0W0xF</td>
<td>RF000</td>
<td>00000</td>
<td>Memory-&gt;MDR, MDR-&gt;Bus, Bus-&gt;R0..R7</td>
</tr>
</tbody>
</table>
Modeling micro-operations

- Using multiple micro-transitions inside one architectural transition.
  - Identify all the possible micro-operations.
  - Create a table mapping architectural action bits to the micro-operations list.
  - For each architectural role:
    - Identify the minimal list of micro-operations possible at the first micro-step.
    - Create a table linking the drivers and receivers of micro-operations.
    - Repeat the same process for the second and third micro-steps if needed.
- Create tables for each of the possible micro-steps for each of the architectural roles.
# Micro-Transitions

<table>
<thead>
<tr>
<th>Micro-Operation</th>
<th>MEMORY</th>
<th>R0..R7</th>
<th>MAR</th>
<th>MDR</th>
<th>Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-&gt;Bus</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>Bus-&gt; (MDR,MAR)</td>
<td>(MAR,MDR)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory-&gt;MDR</td>
<td></td>
<td>Address</td>
<td></td>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td>Bus-&gt;R0..R7</td>
<td></td>
<td>Bus</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IR-&gt;Bus</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IR</td>
</tr>
</tbody>
</table>
Macro-Operations

• Multiple Protocol interaction.
• Concurrent versus interleaved modeling.
• One protocol may have to idle/wait while other is in the middle of processing. Semaphore is one solution.
• Example:
  – An incoming forwarding message.
  – Need to compare with all the outstanding requests.
    • Multiple requests can share the same datum.
  – May takes multiple cycles for an interleaved model.
  – The IOB will not honor any other messages during this.
Modeling Out-of-Order Interconnection

• An artificial architectural step is defined to mimic this transition.
• A FIFO of limited size modeling the interconnection.
• When buffer is not full, sender can put a message in it.
• When buffer is not empty, receiver can get a message from a random valid entry in the buffer.
  – Randomness is achieved through usage a free variable as selector.
Constraining Inputs

• Need to create a legal environment.
• Instruction set is the main input.
• Each instruction consists of 20+ subfields and 100+ bits.
• Creating constraints for all these cases is a:
  – Huge challenge.
  – Error prone.
  – Hard to manage to accommodate consistent changes.
• Solution is the automation:
  – Generate from the protocol tables
  – DV also picked our solution for the random stimuli generation.
### Stimuli Generation

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Requester</th>
<th>Server</th>
<th>CMD</th>
<th>Exclusive</th>
<th>Partial</th>
</tr>
</thead>
<tbody>
<tr>
<td>12'b000000000000</td>
<td>Local</td>
<td>Local</td>
<td>Read</td>
<td>1'b0</td>
<td>1'b1</td>
</tr>
<tr>
<td>12'b000000000001</td>
<td>Local</td>
<td>Local</td>
<td>Read</td>
<td>1'b0</td>
<td>1'b0</td>
</tr>
<tr>
<td>12'b000000000010</td>
<td>Remote</td>
<td>Local</td>
<td>Write</td>
<td>1'b1</td>
<td>1'b1</td>
</tr>
<tr>
<td>12'b000000000011</td>
<td>Remote</td>
<td>Local</td>
<td>Write</td>
<td>1'b0</td>
<td>1'b0</td>
</tr>
</tbody>
</table>
Unique Transition Identifiers

• Each transition have been assigned a unique identifier.
• Any protocol failure trace only need to print these identifiers.
• A Tcl script can match these identifiers with the protocol tables to generate failing interaction in terms of the table transitions.
• Great help in debugging at architectural level.
• Also very useful to transaction coverage.
Formal Verification
Formal Verification Flow

1. **Architects**
   - SpecGen
   - Templates

2. **Verifiers**
   - Instrumentation
     - Instrumentation Bugs
   - Protocol Bugs

3. **Golden Reference Model**
   - ASCII Tables

4. **Model Configurations**
   - FSpecGen
     - Role Tables
     - Configuration Bugs
     - Stimuli Table

5. **Merge**
   - The Model
     - Protocol Sequences
Assumptions

• Around 10 assumptions because of automatic stimuli generation.
• Here is a selected list.
  – Assuming address space is of size 2.
  – Assuming agents’ FIFO depth is 4.
  – Assuming if an address is faulty, it can or cannot become fault-free.
  – Assuming the depth of different channels.
  – Assuming only a subset of instructions is available by constraining the “instruction” counter.
Path Clearing

• Verifying that model is:
  – neither over-constrained
  – nor under-constrained.
• No conflicting assumptions.
• All transitions are reachable.
• All legal states are reachable.
• The legal transitions of the abstract state machine representing roles of different agents are reachable.
Functional Correctness

• When there are no outstanding transactions the system only can be in one of the legal states.
• The completeness of each architectural role.
• Roles’ abstract state machine verification.
• Cache coherence is maintained.
• Agents’ messages’ constraints are observed.
• There are no deadlocks because of the interaction of the architectural components.
• A dirty block in cache will always be written back to the memory before its invalidation.
A Sample Property

property agents_constraints(start_event, start_data, end_event, end_data, Outstanding, clk, rst);
  logic [$bits(start_data)-1:0] local_data;
  logic [$bits(Outstanding)-1:0] numAhead;
  (start_event, local_data = start_data,numAhead = Outstanding)
    ##1 (numAhead > 0 ##0 end_event[-1], numAhead--)[*]
    ##1 (numAhead == 0 ## 0 end_event[-1]) |-> end_data == local_data;
endproperty

Start_event captures the moment when agent receives a write request.

End_event captures the moment when the transaction is finished.

Agents’ message constraints are observed.
Functional Coverage

• All possible paths between all the starting and ending states.
• Covered complete instruction set.
• Couple of weeks to finish the run.

sequence tnxm(xhm); (tad_nxm == xnm); Endsequence

cover property((tnxm(20'h00000)[*1:$]##1 tnxm(20'h100a9)[*1:$]));
cover property((tnxm(20'h00000)[*1:$]##1 tnxm(20'h100aa)[*1:$]##1 tnxm(20'h2001a)[*1:$]));
cover property((tnxm(20'h00000)[*1:$]##1 tnxm(20'h1011b)[*1:$]));
cover property((tnxm(20'h00000)[*1:$]##1 tnxm(20'h101f3)[*1:$]##1 tnxm(20'h20019)[*1:$]));

A simple sequence to capture the current value of tad_nxm, the register that records Agents’ transitions.

Auto generated sequences, while assuming default clocking and disable blocks are defined.
Results
• The first and biggest challenge is the completeness of the tables.
  – There were missing transitions in home, remote, and IOB tables.
    • The missing transitions cause dead-ends (a benign form of deadlocks).
• The next biggest set of bugs originated from the bad transitions.
  – The architect inadvertently inserted wrong transition of a few operations.
    There were missing or wrong micro-operations.
• There were few unreachable transitions as well.
• The functional coverage using sequence coverage is a powerful mechanism to prove the completeness of the effort.
  • We found that there were few sequences that were not reachable because of the architectural constraints.
References


Thank You.