Table-based Functional Coverage Management for SOC Protocols

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

• The Problem.
• A Functional Coverage Management System.
• The Components of FCMS:
  – Specification, Coverage Validation, Coverage Collection, Coverage Analysis.
• Deploying FCMS:
  – OCI, Specification Tables, Functional Coverage, SVA Properties, SV Covergroups, RTL Packages.
• Results.
• Conclusions.
The Problem

• Why and What of Coverage?
  – Verification quality, Code coverage, FSM coverage.

• Functional Coverage.
  – Why code coverage is not enough?
  – Scaling issues.
  – Completeness check.

• Protocol Functional Coverage.
  – Effectively a protocol is a set of interacting FSMs.
SpecGen

FV = No
DV = No

Bug in Spec

FV = Yes
DV = No

Bug in Sim

Trace Generation

Trace

Compare Lists

Reachability list

(a)

Sequence Coverage SVA

Formal Model

Formal Verification

(2)

(6)

(8)

(1)

(3)

(4)

(5)

Reference Model

SV Packages

Sequence Coverage Groups

Simulation

Covered list

Stimuli Generation

(10)

(7)

(9)
Components of FCMS

Specification Generation
Validation
Data Collection
Analysis
Specification Generation

• SpecGen:
  – A Perl-based architectural specification tool.

• Inputs:
  – Valid states set, Commands, Constraints.

• Outputs:
  – ASCII specification tables.
  – RTL in form of SV packages.
  – List of the sequences/transactions of the protocol.
  – One for home node and one for remote node.
Validation

• Two requirements:
  – Correctness and completeness.
• Correctness: Are the auto-generated sequence reachable?
  – Formal model + coverage sequence in SVA.
  – 16 Processor/384G machine.
  – About a week time.
• Completeness: Are we listing all the sequences?
  – Simple directed graph algorithm to find all possible paths between two points.
Data Collection

• RTL is what will be implemented and should be covered during simulations.
  – Need to cover protocol states, transitions, transactions and home-remote interactions.

• Two sources of data collection:
  – Auto generated coverpoints, covergroups, cross coverage in reference model.
  – SV-packages with annotated coverage information flushed during simulations.

• Data = RTL protocol coverage + Reference model protocol coverage.
Analysis

- All the coverage properties, coverpoints, covergroups, RTL-packages are back-annotated and come from the same source.
- Simulations’ data is compared to FV’s data.

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Formal Verification</th>
<th>Conclusions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unreachable</td>
<td>Reachable</td>
<td>A hole in stimuli generation or a bug in RTL</td>
</tr>
<tr>
<td>Unreachable</td>
<td>Unreachable</td>
<td>A bug in architectural specification.</td>
</tr>
<tr>
<td>Reachable</td>
<td>Reachable</td>
<td>Done with this transaction.</td>
</tr>
<tr>
<td>Reachable</td>
<td>Unreachable</td>
<td>Formal model is over-constrained or a bug in RTL/reference</td>
</tr>
</tbody>
</table>
Deploying FCMS

OCI Protocol
Specification Tables
Functional Coverage Transactions
SVA Properties
SV Covergroups
Results
The OCI Protocol

Octeon III Node 0

Octeon III Node 1

Octeon III Node 2

Octeon III Node 3

OCI

OCI

OCI

OCI

IO

IO

IO

IO

DRAM

DRAM

DRAM

DRAM

OCI

OCI

OCI

OCI
### Specification Tables(1)

#### Home Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Outputs</th>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cmd</strong></td>
<td><strong>H</strong></td>
<td><strong>N1</strong></td>
<td><strong>N2</strong></td>
</tr>
<tr>
<td>none</td>
<td>E</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>none</td>
<td>S</td>
<td>S</td>
<td>I</td>
</tr>
<tr>
<td>LCL_WR</td>
<td>S</td>
<td>S-&gt;I</td>
<td>I</td>
</tr>
</tbody>
</table>
## Remote Table for Node 1

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Outputs</th>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cmd</strong></td>
<td><strong>St</strong></td>
<td><strong>Cmd</strong></td>
<td><strong>St</strong></td>
</tr>
<tr>
<td>none</td>
<td>I</td>
<td>OCI_RD</td>
<td>I</td>
</tr>
<tr>
<td>OCI_RD</td>
<td>I</td>
<td>none</td>
<td>S</td>
</tr>
<tr>
<td>none</td>
<td>S</td>
<td>none</td>
<td>I</td>
</tr>
</tbody>
</table>
### OCI Functional Coverage

**An OCI Home Transaction for a 2-node Configuration**

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Outputs</th>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cmd</strong></td>
<td><strong>H</strong></td>
<td><strong>N1</strong></td>
<td><strong>Cmd</strong></td>
</tr>
<tr>
<td>none</td>
<td>I</td>
<td>E</td>
<td>E2S</td>
</tr>
<tr>
<td>E2S</td>
<td>S</td>
<td>S</td>
<td>E2S</td>
</tr>
<tr>
<td>E2S</td>
<td>S</td>
<td>S-&gt;I</td>
<td>none</td>
</tr>
</tbody>
</table>
sequence homeTrans_111(logic [A-1:0][3:0]Cmd, ...);

((Cmd == NONE)&&(H_state == I)&&(N1_state == E) &&
(Outputs == NONE)&&(Inputs == NONE ))[1:$])

###1 ((Cmd == E2S)&&(H_state == S)&&(N1_state == S) &&
(Outputs == FWDH)&&(Inputs == NONE ))

###1 ((Cmd == E2S)&&(H_state == S)&&(N1_state == S) &&
(Outputs == NONE)&&(Inputs == NONE ))[*1:$])

###1 ((Cmd == E2S)&&(H_state == S)&&(N1_state == S) &&
(Outputs == NONE)&&(Inputs == REM_INV ))

............

###1 ((Cmd == E2S)&&(H_state == S)&&(N1_state == I) &&
(Outputs == NONE)&&(Inputs == NONE ));

dense homeTrans_111(....);

cover_homeTrans_OCI_LD_1:cover property(
(Request == OCI_LD) ###1 homeTrans_111(...));
covergroup Home_Monitor;
requests: coverpoint request_type {
  bins cov_LD[] = {OCI_LD};
  ....
}
transitions: coverpoint {request_type,cmd,h_state,r_state,output,input } {
  `include "HOME_TRANS_COV.sv"
}
endgroup

A bin for OCI_LD
Other Core requests and their bins

HOME_TRANS_COV.sv
Auto generated Transitions cover points

bins HomeTrans_OCI_LD_1 = ({OCI_LD,NONE,INV,EXL,NONE,NONE} =>
{OCI_LD,E2S,SHR,SHR,FWDE,NONE } =>{OCI_LD,E2S,SHR,SHR,NONE,NONE } ) [*1:SDELAY] => {OCI_LD,E2S,SHR,SHR,INV,SHR,NONE } =>
{OCI_LD,E2S,SHR,S_I,NONE,NONE } ) [*1:SDELAY] =>
{OCI_LD,E2S,SHR,S_I,VDATA,NONE } =>{NONE,NONE,NONE,NONE,NONE,NONE } ) ;
Annotated SV Packages

• Tables are part of the RTL as SV functions.
• Each table transition has a unique ID.
• The execution of a transition push this ID into an instrumentation buffer.
• When a protocol transaction completes, the sequence of ID in the instrumentation buffer is dumped into a text-file.
• At the end of the simulations these text-files are parsed to see what transactions are covered.
//Protocol table function
    function automatic oci_vab_tbl_out_t oci_vab_tbl_pipe_tbl_f;
    input oci_vab_tbl_in_t oci_vab_tbl_in;
    begin
        unique casez (oci_vab_tbl_in)
        {1'b1,XMC_WBIL2},OCI_INAD_L,...} : oci_vab_tbl_pipe_tbl_f =
        {OCI_VABC_R2I_FV,OCI_L_ST_FV,OCI_L_HS_I,...}; Push;
        ........
        default: begin
            oci_vab_tbl_pipe_tbl_f.oerr = OCI_BIT_1; //Set error bit to 1
        end
    endcase
endfunction

assign oci_vab_tbl_out = oci_vab_tbl_pipe_tbl_f(oci_vab_tbl_in_9a);
## Results

<table>
<thead>
<tr>
<th>Tool</th>
<th>Architectural Spec</th>
<th>RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bugs</td>
<td>Holes</td>
</tr>
<tr>
<td><strong>Chip-1</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formal</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Simulation</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td><strong>Chip-2</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formal</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Simulation</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Conclusions

• Correctness:
  – Proved no coverage holes were left uncovered and all transactions were reachable.

• Completeness:
  – All protocol transactions were covered.

• Scalability
  – 2, 3, 4 nodes (4000 to 16000+ transitions)

• Manageability
  – Protocol iterations.
  – Changes incorporated into the regression flow in a day time and validated in a week time.