Table-based Functional Coverage Management for SOC Protocols

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

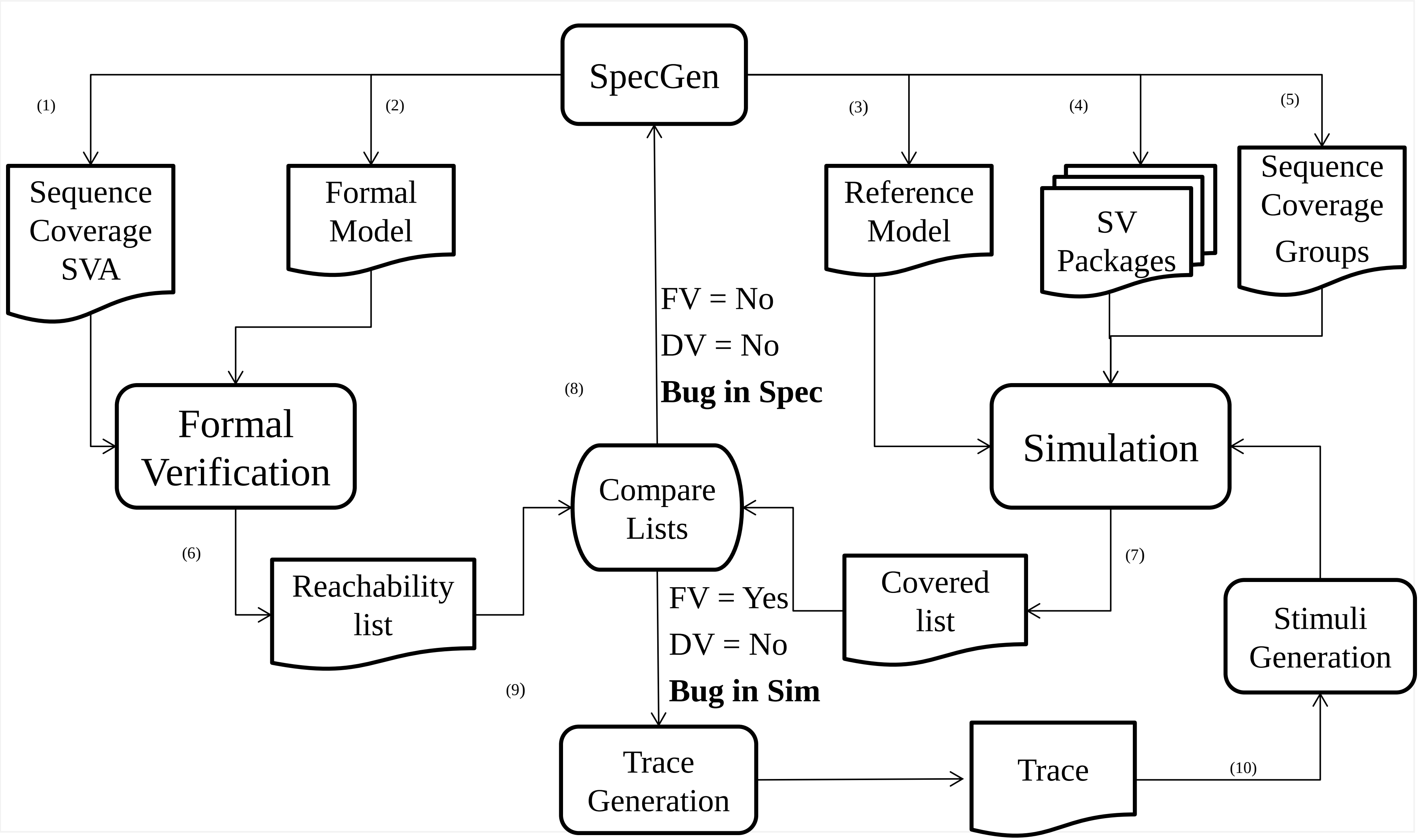
Outline

- 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.
  - Manual-maintenance challenges.
  - Completeness check.
- Protocol Functional Coverage.
  - Effectively a protocol is a set of interacting FSMs.

FCMS


SpecGen
(1)
(2)
(3)
(4)
(5)
Sequence Coverage SVA
Formal Model
Reference Model
SV Packages
Sequence Coverage Groups
FV = No
DV = No
Bug in Spec
(8)
Formal Verification
Simulation
Compare Lists
(6)
(7)
Reachability list
Covered list
Stimuli Generation
FV = Yes
DV = No
Bug in Sim
(9)
Trace Generation
Trace
(10)

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.
  - Tables for Formal and Reference Models.
  - 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.

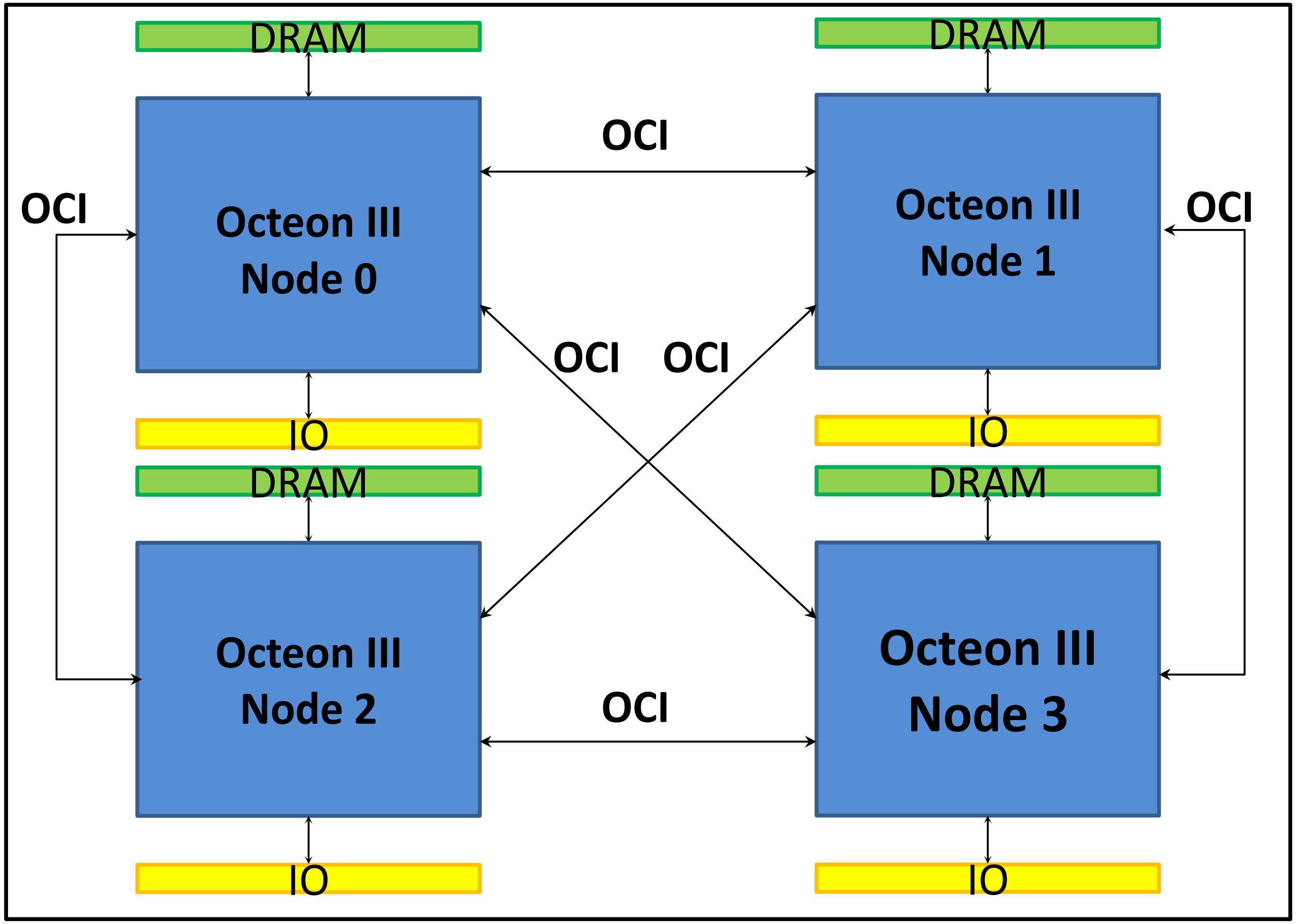
For a Given Protocol Transaction		
Simulations	**Formal Verification**	**Conclusions**
Unreachable	Reachable	A hole in stimuli generation or a bug in RTL
Unreachable	Unreachable	A bug in architectural specification.
Reachable	Reachable	Done with this transaction.
Reachable	Unreachable	Formal model is over-constrained or a bug in

Deploying FCMS

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

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

The OCI Protocol



Specification Tables(1)

Home Table													
Current State					Next State					Outputs			Inputs
Cmd	**H**	**N1**	**N2**	**N3**	**Cmd**	**H**	**N1**	**N2**	**N3**	**N1**	**N2**	**N3**	**Req/Resp/Frwd**
none	E	I	I	I	none	S	S	I	I	RD_RSP	-	-	OCI_RD (Read from remote node)
none	S	S	I	I	LCL_WR	S	**S->I**	I	I	INVAL	-	-	LCL_WR_LCL_A (local write to home address)
LCL_WR	S	**S->I**	I	I	none	M	I	I	I	-	-	-	INV_RSP (Invalidate response)

Specification Tables(2)

Remote Table for Node 1							
Current State		Next State		Outputs			Inputs
Cmd	**St**	**Cmd**	**St**	**H**	**N2**	**N3**	**Req/Resp/Frwd**
none	I	OCI_RD	I	OCI_RD (Read from remote node – i.e. home)	-	-	LCL_RD_RMT_A (local read – remote address)
OCI_RD	I	none	S	-	-	-	RD_RSP (read response)
none	S	none	I	INV_RSP (Invalidate Response)	-	-	INV (Invalidate)

OCI Functional Coverage

An OCI Home Transaction for a 2-node Configuration							
Current State			Next State			Outputs	Inputs
Cmd	**H**	**N1**	**Cmd**	**H**	**N1**	**N1**	**Req/Resp/Fwd**
none	I	E	E2S	S	S	FWDH	OCI_LD
E2S	S	S	E2S	S	S->I	none	REM_INV
E2S	S	S->I	none	M	I	-	VDATA

SVA Model

```
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 ));
endsequence

cover_homeTrans_OCI_LD_1:cover property(
(Request == OCI_LD) ##1 homeTrans_111(…));
```

Initial state

Transition state

Waiting state

Transition state

Final state

Covering an OCI_LD

SV Covergroup

```
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

Auto generated Transitions cover points

HOME_TRANS_COV.sv

```
……. …..
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,INVL,NONE }=>
{OCI_LD,E2S,SHR,S_I,NONE,NONE } ) [*1:SDELAY]=>
{OCI_LD,E2S,SHR,S_I,VDATA,NONE }=>{NONE,NONE,SHR,INV,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.

OCI RTL Function

```
//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 end
  endfunction:oci_vab_tbl_pipe_tbl_f
```

A table function

A table transition

Push the label for transition

Somewhere else in the RTL

```
assign oci_vab_tbl_out  = oci_vab_tbl_pipe_tbl_f(oci_vab_tbl_in_9a);
```

Results

		Architectural Spec		RTL	
	Tool	Bugs	Holes	Bugs	Holes
Chip-1	Formal	15	20	0	0
	Simulation	2	3	45	20
Chip-2	Formal	5	7	0	0
	Simulation	0	0	35	8

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.