2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Formal Architectural Specification and Verification of A Complex SOC

Shahid Ikram, Isam Akkawi, David Asher, Jim Ellis

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

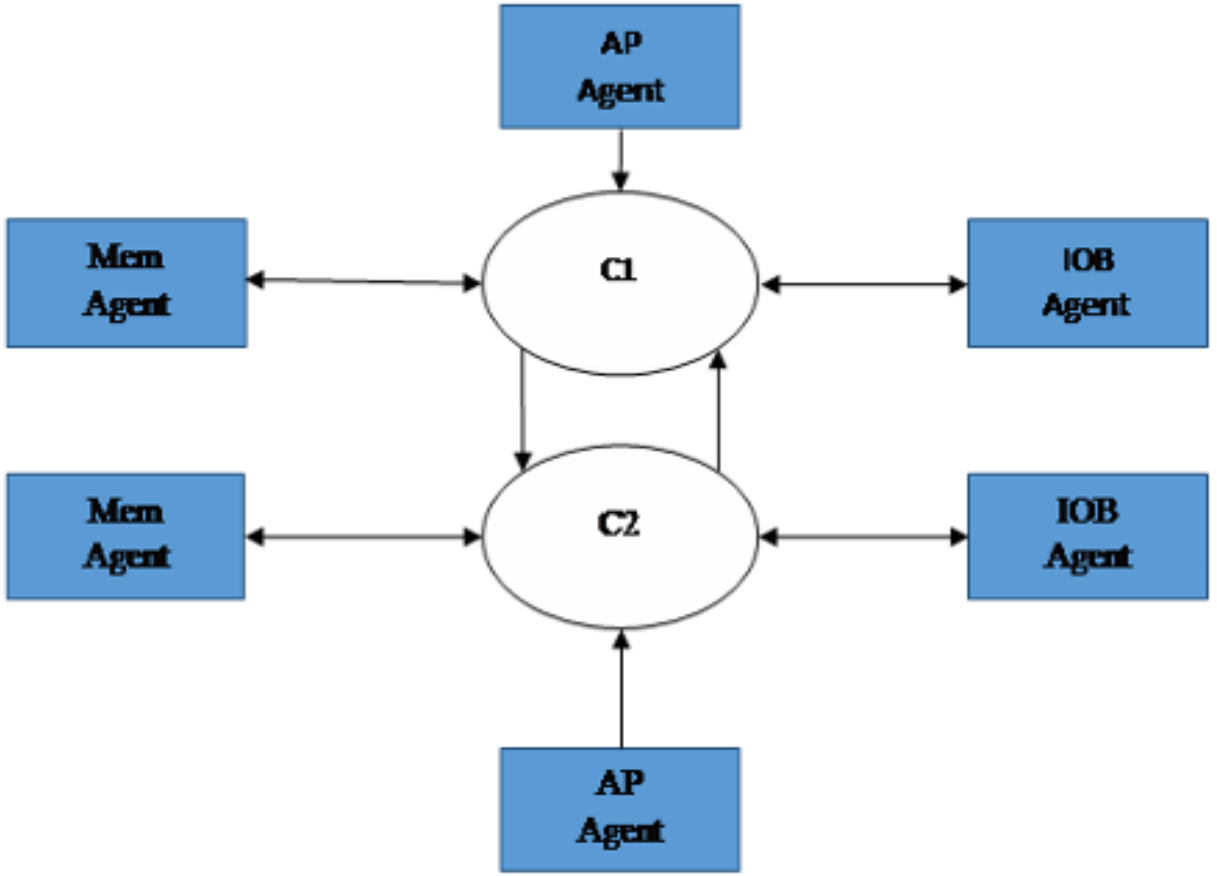
Outline

- 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

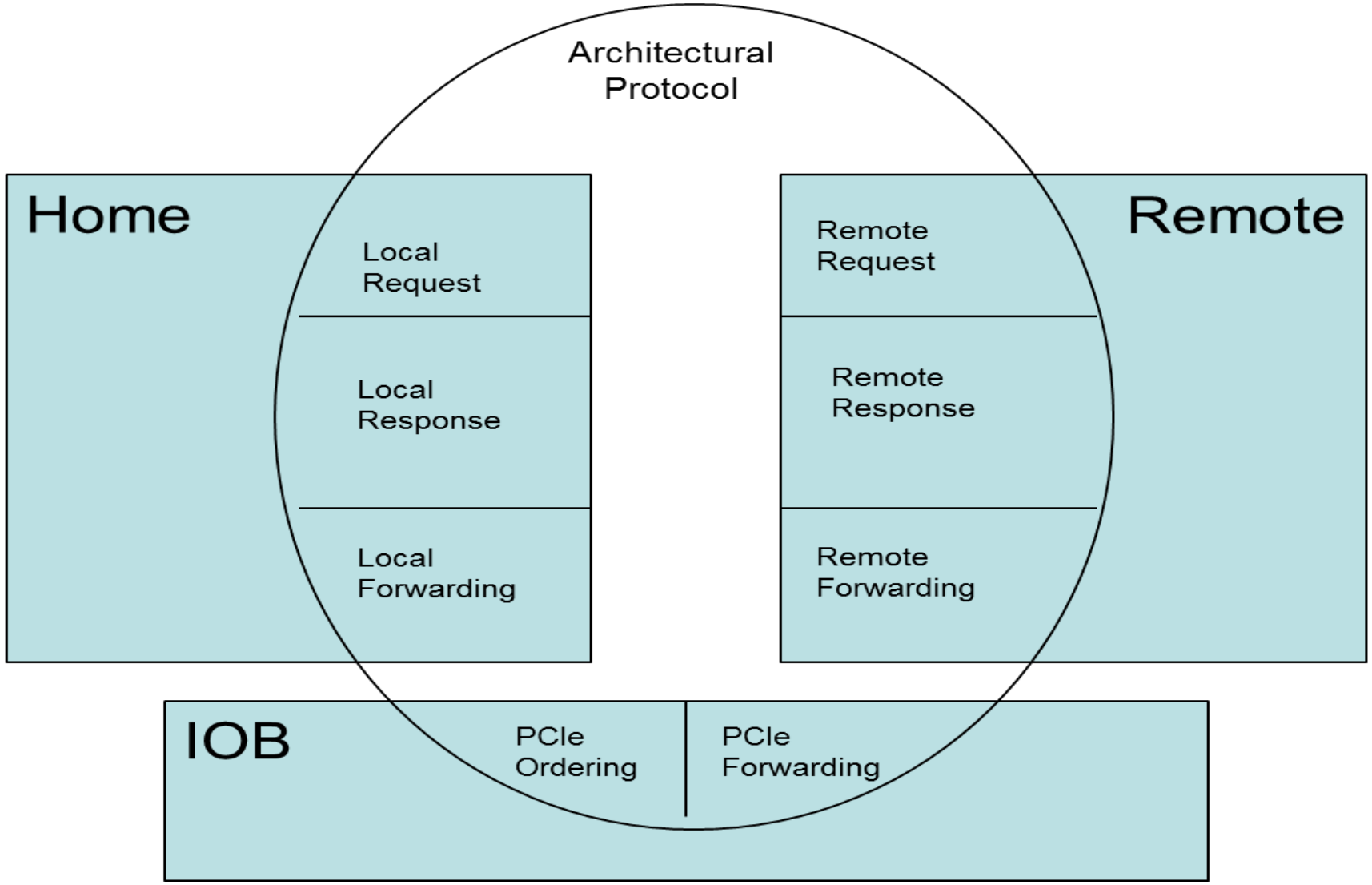
Extended Table-based Specification

Current State			Next State			Outputs	Inputs
Cmd	**H**	**N1**	**Cmd**	**H**	**N1**	**N1**	**Req/Resp/Frwd**
none	E	I	none	S	S	Read_Response	RD_R (Read from remote node)
none	S	S	Local_Write	S	S->I	INV (Invalidate)	LCL_WR_LCL_A (local write to home address)
Local_Write	S	S->I	none	M	I	-	INV_RSP (Invalidate response)

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



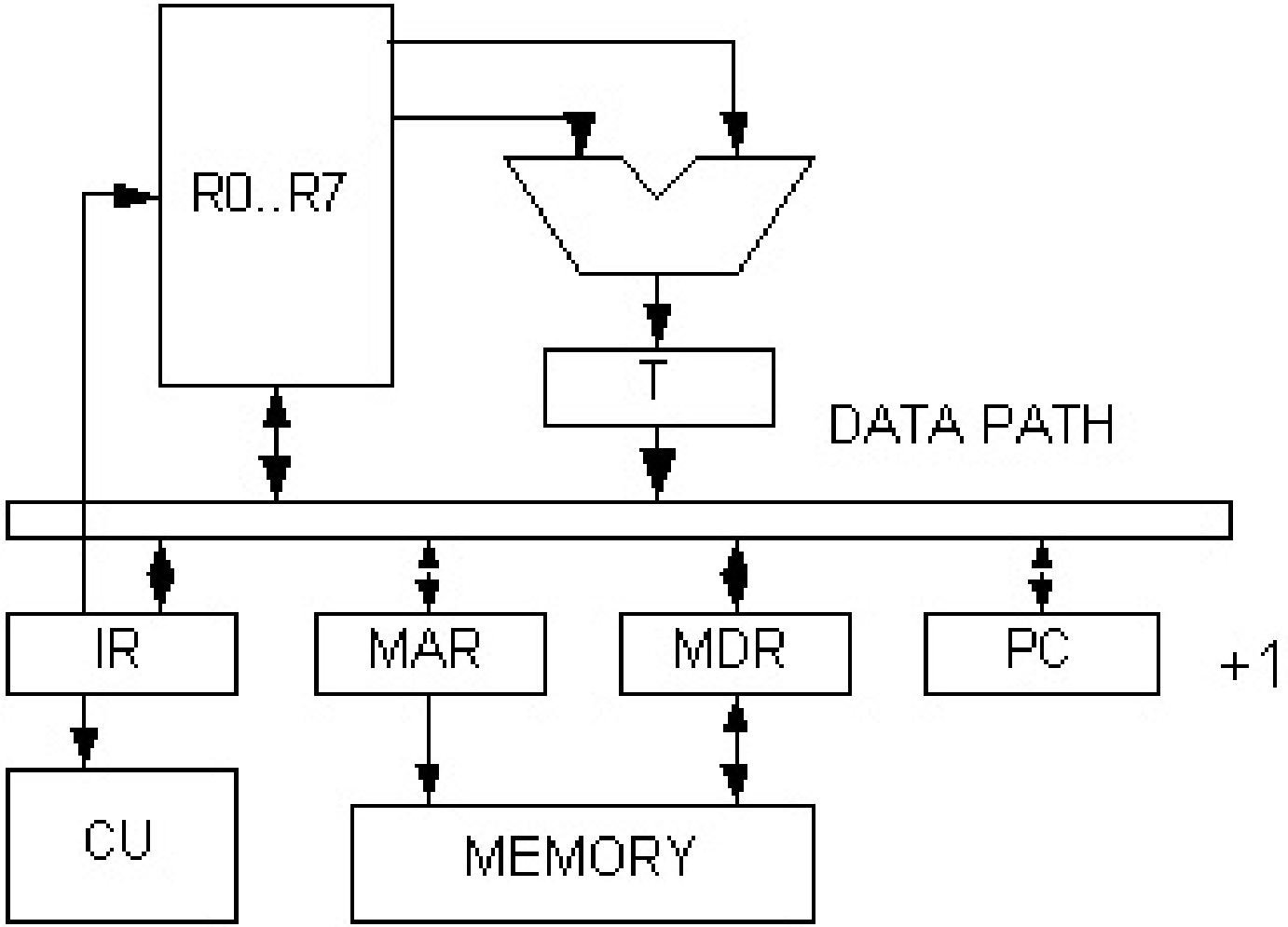
Input Columns of An Agent Table

Current State			New Request Cmd			New Ack Cmd		New Data Cmd	
Transient State	Home	Remote	Requester	Server	Cmd	Requester	Cmd	Requester	Cmd
NONE	I	I	Local	Local	Read	none	none	none	none
WI00	I	K	Remote	Local	Forwa rd	none	none	none	none
CSFH	S->I	K->E	none	none	none	Remote	PRE	none	none
ED00	I	D->E	none	none	none	none	none	Remote	PRD

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

R0..R7	Mem Reads	Mem Writes	Comment
000xx	0x000	W00S0	R0..R7->T, T->Bus, Bus-> (MDR,MAR) MDR-> Memory
I00xx	0x000	0V000	IR -> Bus, Bus->MAR, MAR->Memory,…
0W0xV	0x000	0V0S0	R0..R7->ALU,…,(MAR,MDR)->Memory)
0W0xF	RF000	00000	Memory->MDR, MDR->Bus, Bus->R0..R7

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

Micro-Operation	MEMORY	R0..R7	MAR	MDR	Bus
T->Bus					T
Bus-> (MDR,MAR)	(MAR,MDR)				
Memory->MDR			Address	Memory	
Bus->R0..R7		Bus			
IR->Bus					IR

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

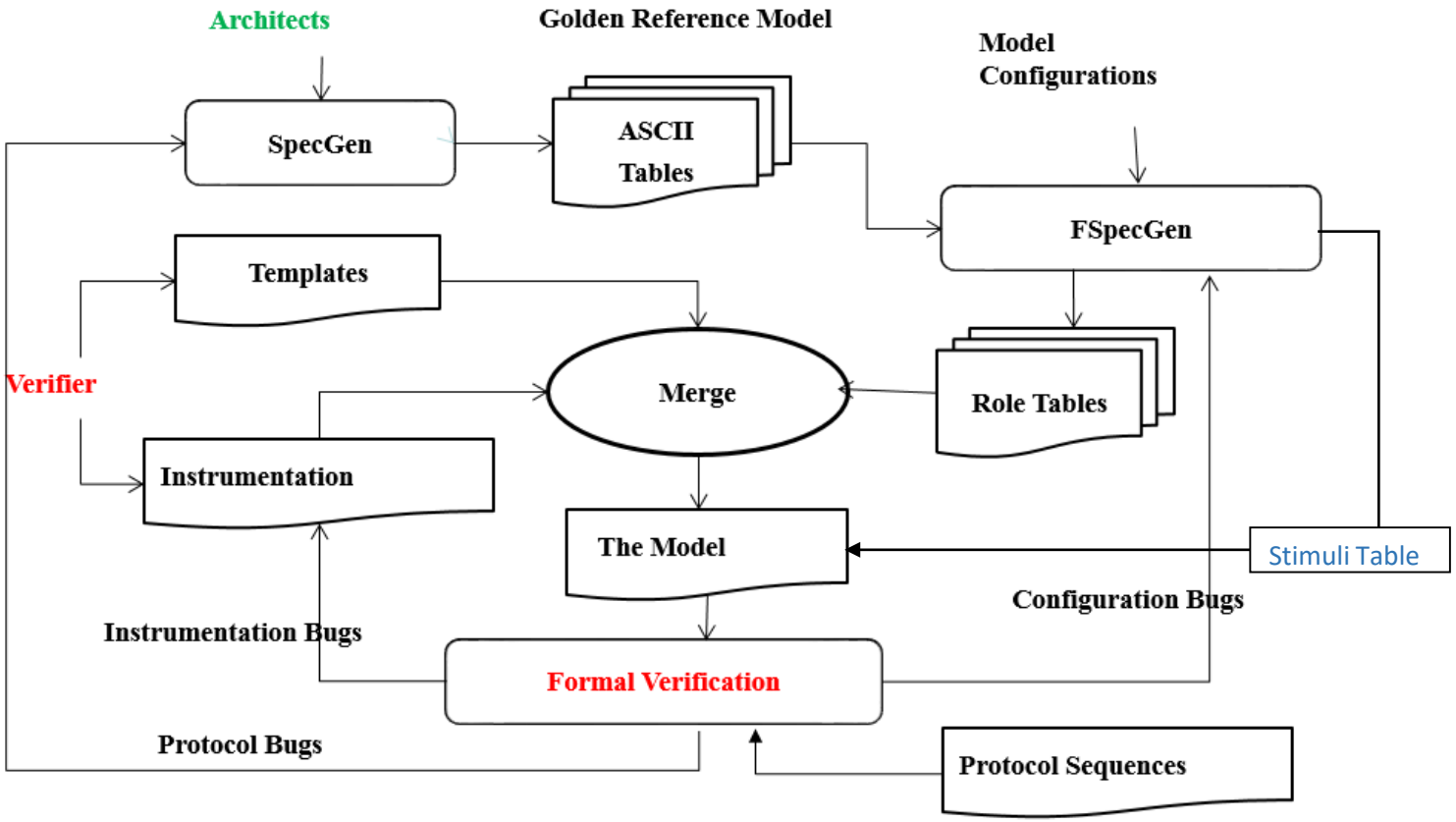
Instruction	Requester	Server	CMD	Exclusive	Partial
12'b000000000000	Local	Local	Read	1'b0	1'b1
12'b000000000001	Local	Local	Read	1'b0	1'b0
12'b000000000010	Remote	Local	Write	1'b1	1'b1
12'b000000000011	Remote	Local	Write	1'b0	1'b0

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



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

Start_event captures the moment when agent receives a write request.

```
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
```

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.

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

```
sequence tnxm(xnm);  (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:$]##1 tnxm(20'h20019)[*1:$]));
```

Auto generated sequences, while assuming default clocking and disable blocks are defined.

Results

Results and Conclusions

- 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.

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

References

- Weber, Ross. (2011) “Modeling and Verifying Cache-Coherent Protocols, VIP, and Designs”. Jasper Design Automation, June 2011.
- [2] Ikram, Shahid et al. (2014) “A Framework for Specifying, Modeling, Implementation, and Verification of SOC Protocols”, September 2014, IEEE-SOCC, Las Vegas, Nevada.
- [3] Ikram, Shahid et al. (2015) “Table-based Functional Coverage Management for SOC Protocols”, March 2015, DVCON, San Jose, CA.
- [4] Ikram, Shahid et al. (2016) “Formal verification of the microarchitectural features in the context of an architectural model”, November 2016, Jasper User Group Workshop 2017, Cupertino, CA.
- [5] Ikram, Shahid et al. (2015) “Design and Verification of a Multichip Coherence Protocol”, March 2015, DVCON, San Jose, CA.
- [6] Cerny, Eduard et al. (2015) “SVA: The Power of Assertions in System Verilog”, Second Edition, Springer, ISBN 978-3-319-07138-1.
- [7] Seligman, Erik et al. (2015) “Formal Verification”, Morgan Kaufmann, ISBN 978-0-12-800727-3.

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Thank You.