Coherency Verification & Deadlock Detection Using Perspec/Portable Stimulus

Moonki Jang – Samsung Electronics Co.,Ltd.

Phu Huynh – Cadence Design Systems, Inc

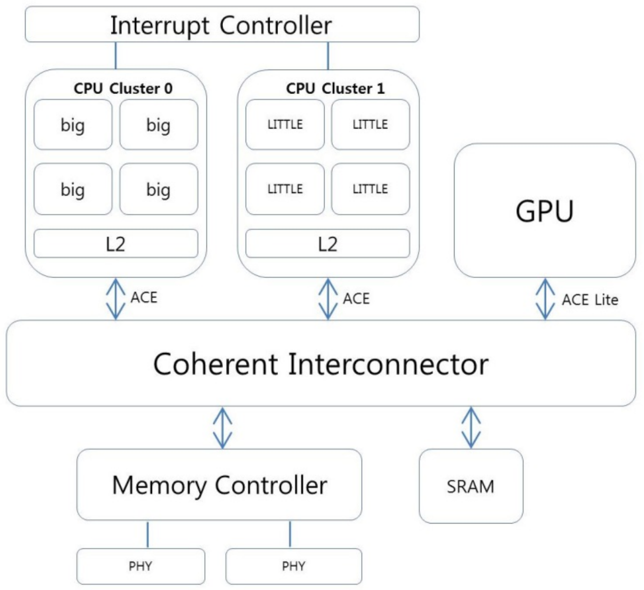
Agenda

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

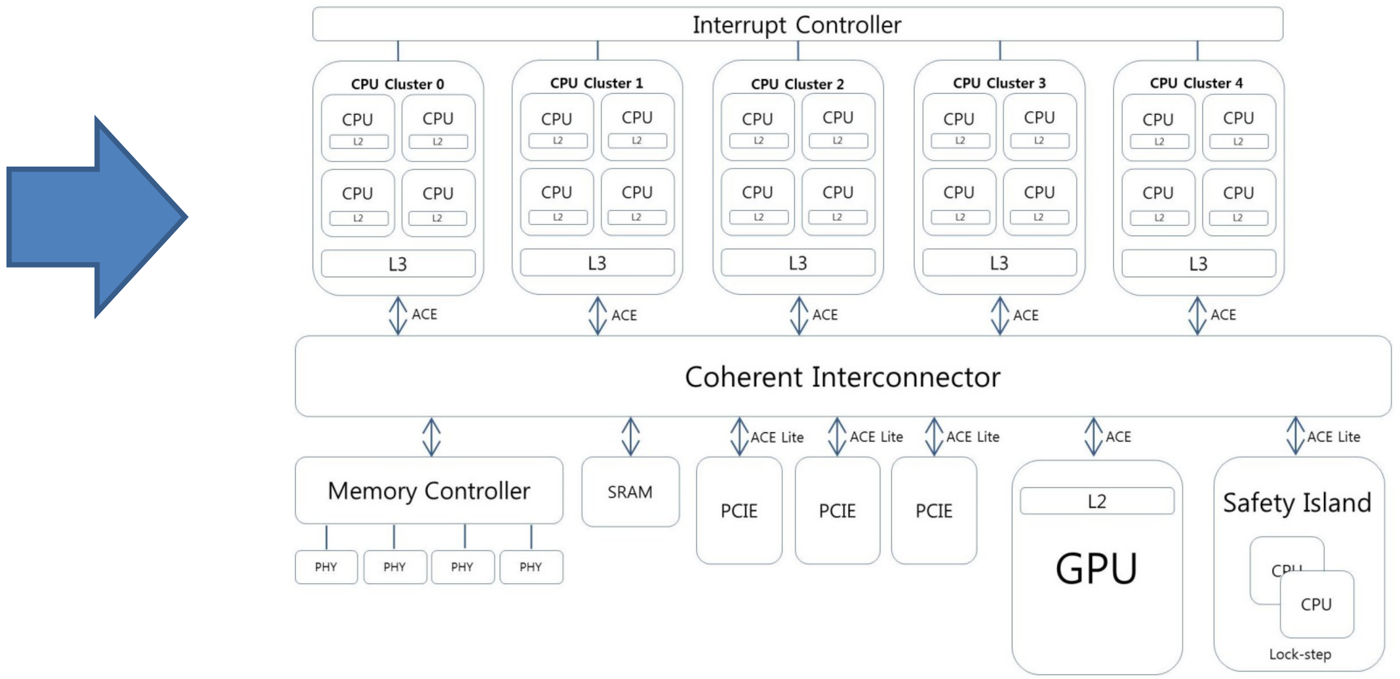
Trends

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

High-End Octa core SoC (Samsung, 2013)

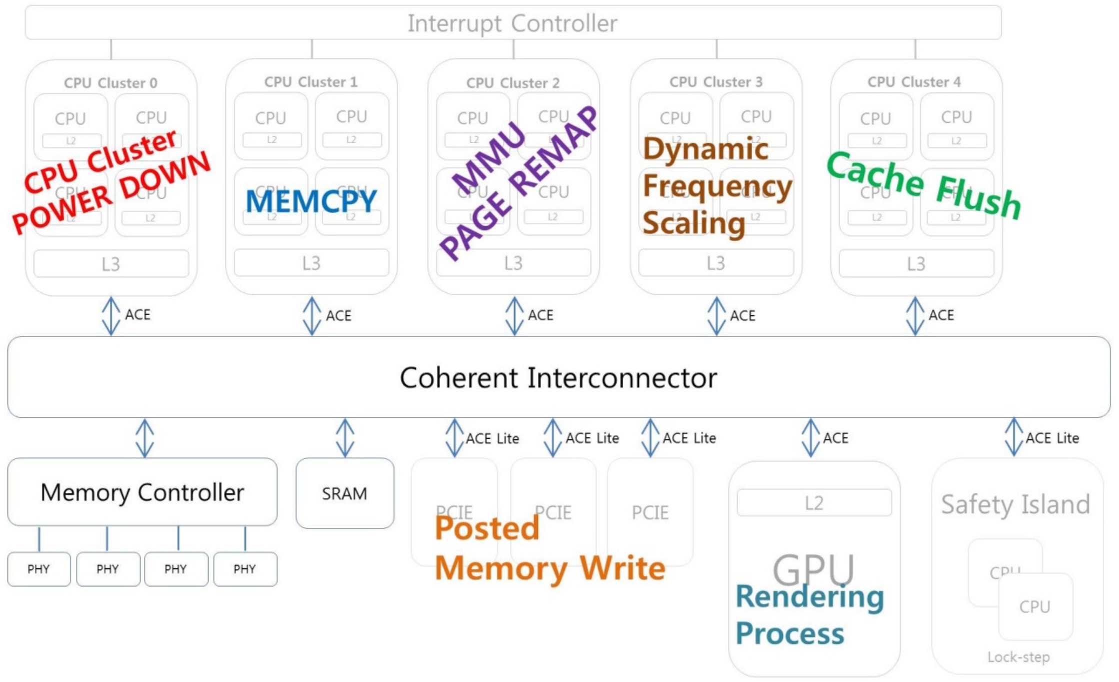


Latest ADAS reference platform (ARM,2018)



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

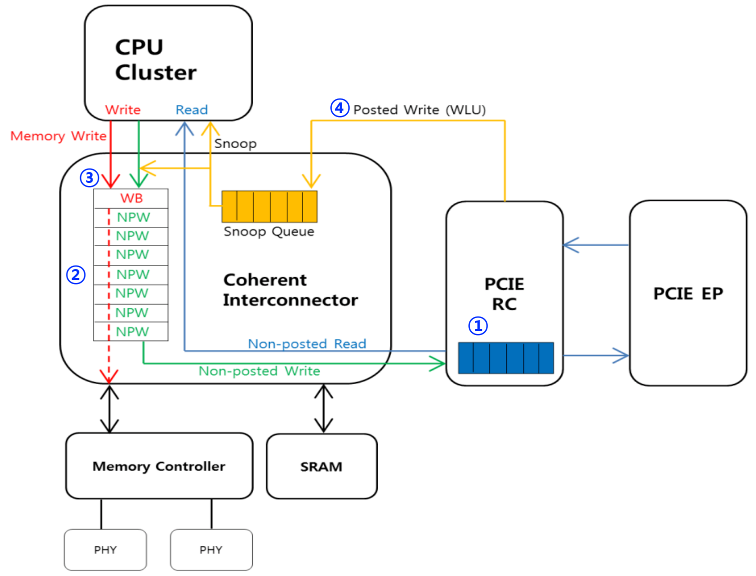
- PSS(Portable Stimulus Standard) verification environment was introduced to solve these issues

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

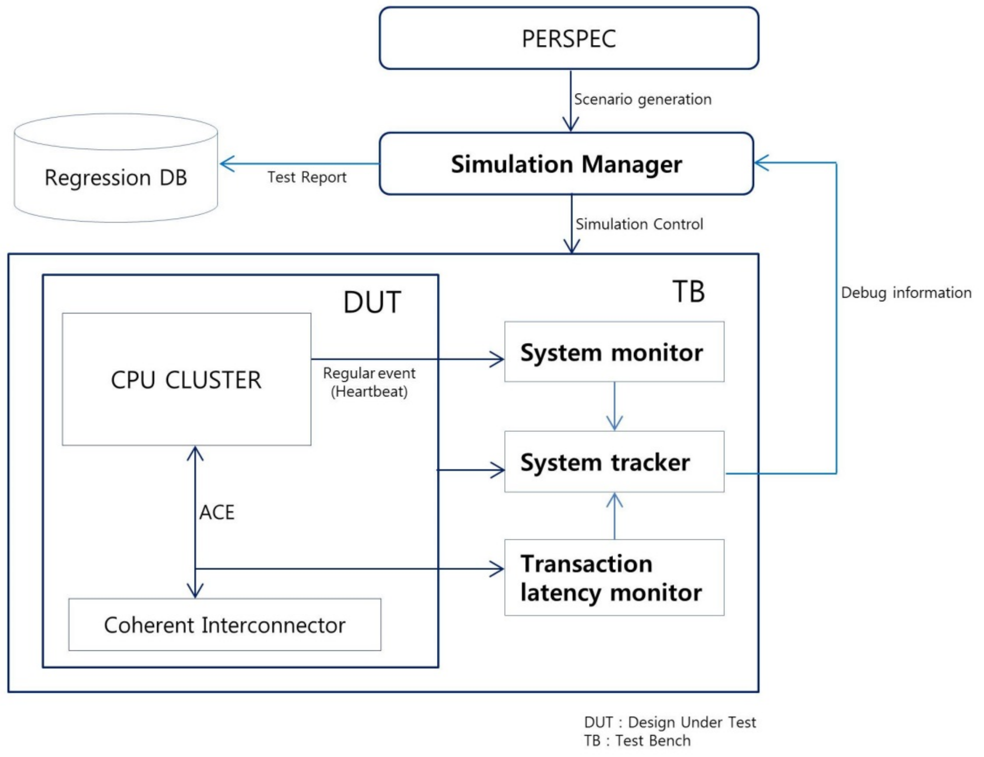
PCIe deadlock verification

- 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'

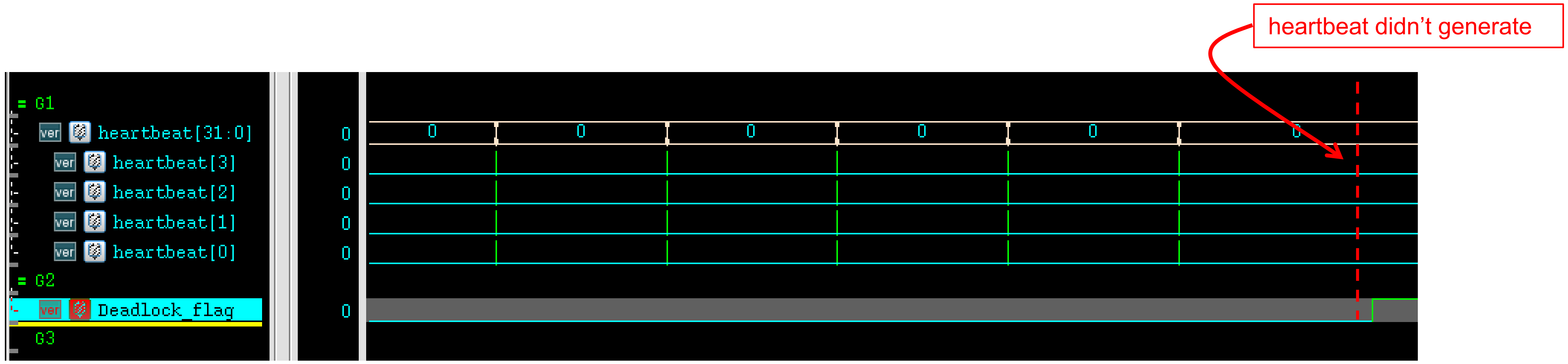
- Transaction Latency Monitor
Records the latency of all transactions originating from the CPU(s)

- 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

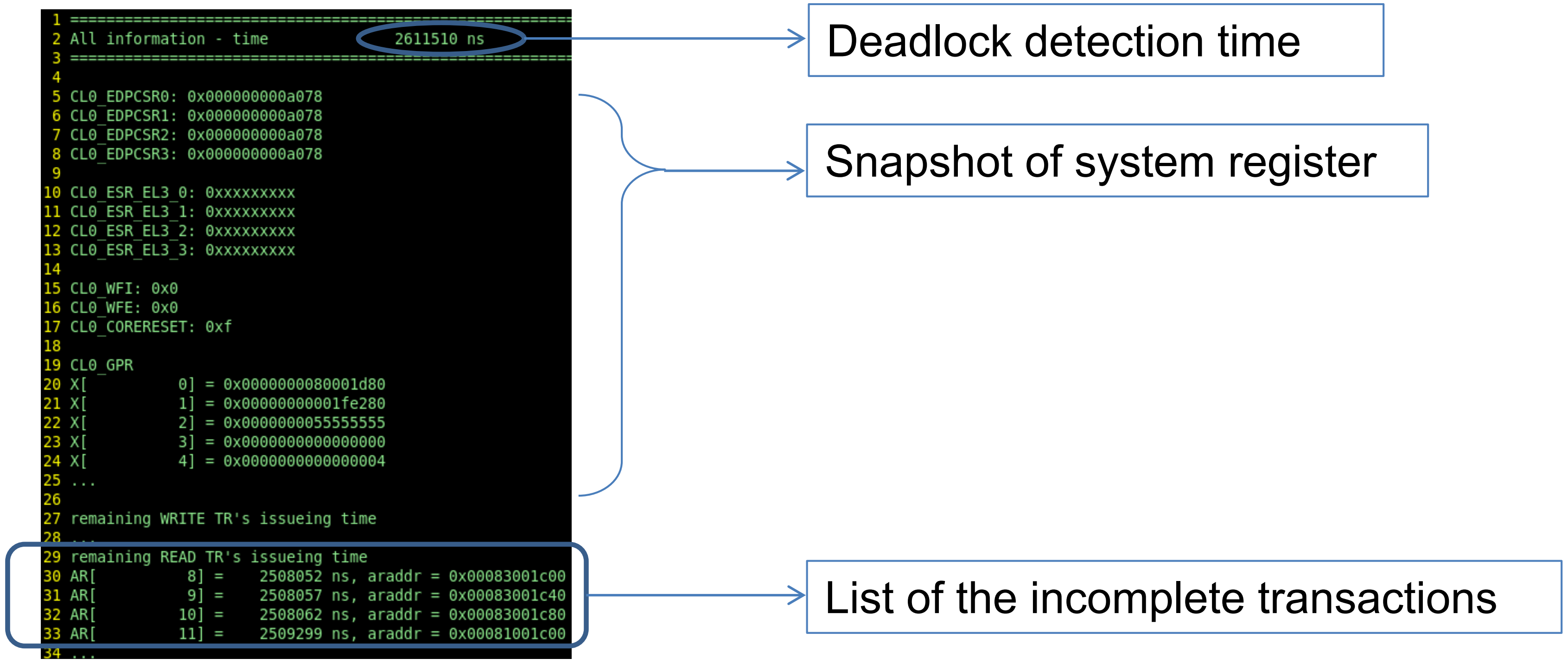
Overview of Our Verification Env

- 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.

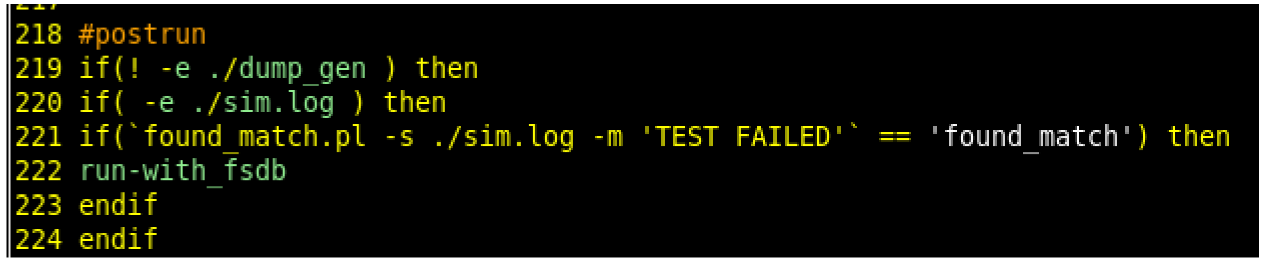


```
==========================================================
All information - time             2611510 ns
==========================================================

CL0_EDPCSR0: 0x000000000a078
CL0_EDPCSR1: 0x000000000a078
CL0_EDPCSR2: 0x000000000a078
CL0_EDPCSR3: 0x000000000a078

CL0_ESR_EL3_0: 0xxxxxxxxx
CL0_ESR_EL3_1: 0xxxxxxxxx
CL0_ESR_EL3_2: 0xxxxxxxxx
CL0_ESR_EL3_3: 0xxxxxxxxx

CL0_WFI: 0x0
CL0_WFE: 0x0
CL0_CORERESET: 0xf

CL0_GPR
X[          0] = 0x0000000080001d80
X[          1] = 0x000000000001fe280
X[          2] = 0x0000000055555555
X[          3] = 0x0000000000000000
X[          4] = 0x0000000000000004
...

remaining WRITE TR's issueing time
...
remaining READ TR's issueing time
AR[          8] =    2508052 ns, araddr = 0x00083001c00
AR[          9] =    2508057 ns, araddr = 0x00083001c40
AR[         10] =    2508062 ns, araddr = 0x00083001c80
AR[         11] =    2509299 ns, araddr = 0x00081001c00
...
```

Deadlock detection time
Snapshot of system register
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



```
#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
```

Agenda

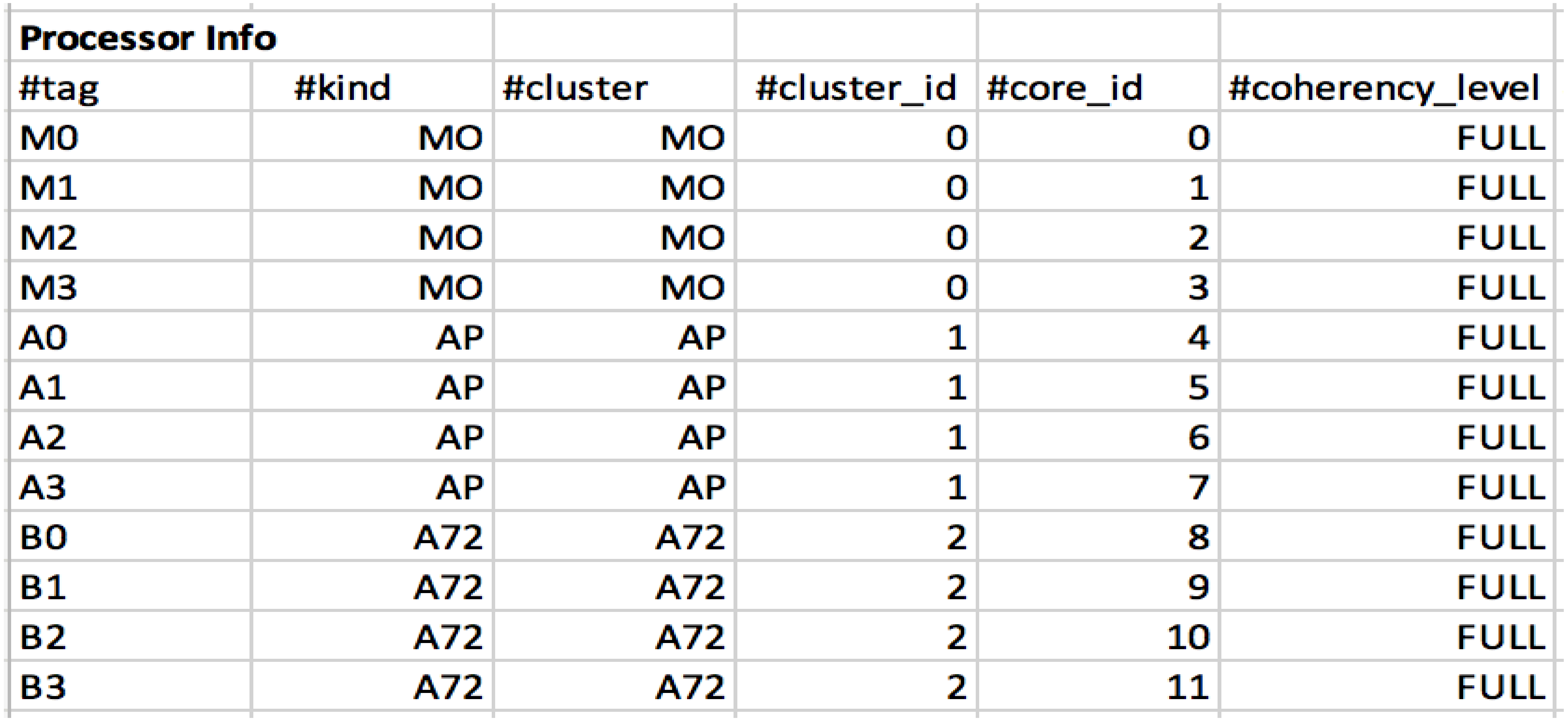
- Why coherency and deadlock detection verification
- Requirements & description of our verification environment
- **Using Perspec and PSS to create reusable test suite**

Coherency Verification

- Perspec and PSS simplifed 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

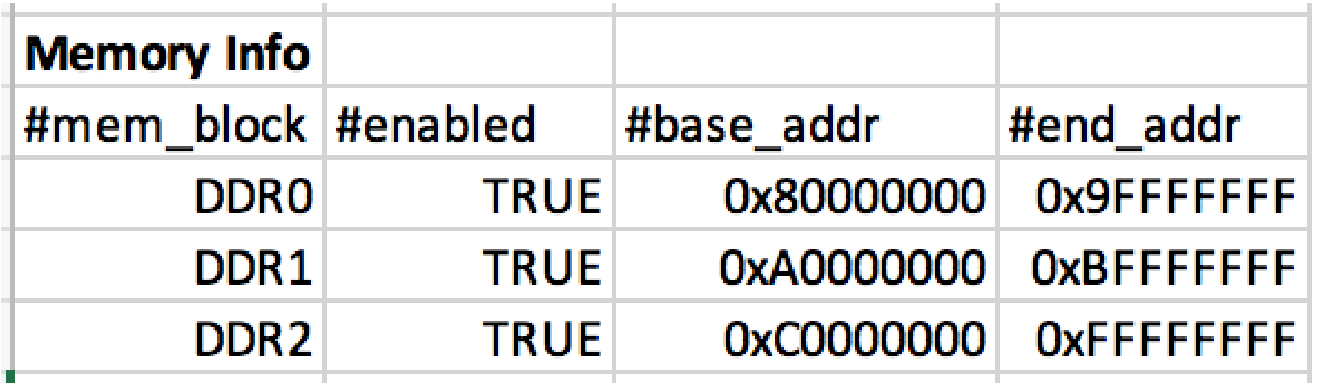
- Processor Info table



Processor Info					
#tag	#kind	#cluster	#cluster_id	#core_id	#coherency_level
M0	MO	MO	0	0	FULL
M1	MO	MO	0	1	FULL
M2	MO	MO	0	2	FULL
M3	MO	MO	0	3	FULL
A0	AP	AP	1	4	FULL
A1	AP	AP	1	5	FULL
A2	AP	AP	1	6	FULL
A3	AP	AP	1	7	FULL
B0	A72	A72	2	8	FULL
B1	A72	A72	2	9	FULL
B2	A72	A72	2	10	FULL
B3	A72	A72	2	11	FULL

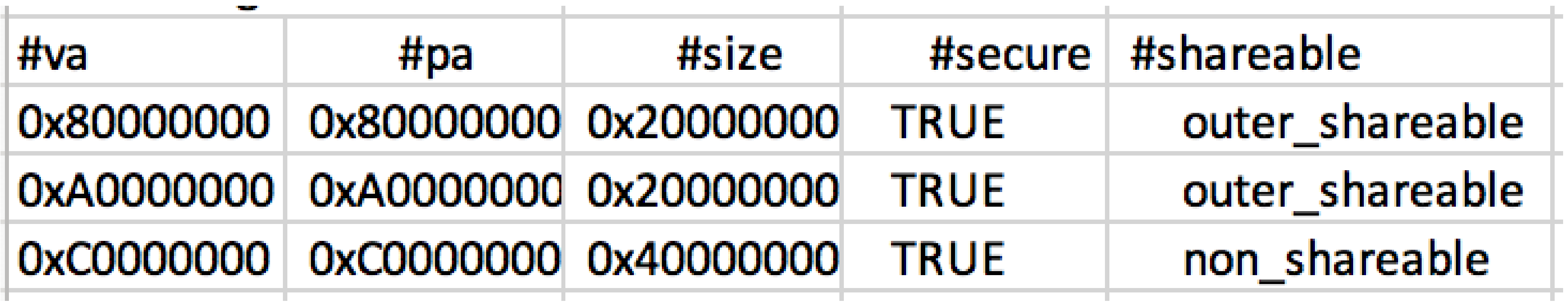
Create Processor-Memory model

- Memory Info table



Memory Info			
#mem_block	#enabled	#base_addr	#end_addr
DDR0	TRUE	0x80000000	0x9FFFFFFF
DDR1	TRUE	0xA0000000	0xBFFFFFFF
DDR2	TRUE	0xC0000000	0xFFFFFFFF

- Page Table



#va	#pa	#size	#secure	#shareable
0x80000000	0x80000000	0x20000000	TRUE	outer_shareable
0xA0000000	0xA0000000	0x20000000	TRUE	outer_shareable
0xC0000000	0xC0000000	0x40000000	TRUE	non_shareable

Memory R/W Test - PSS

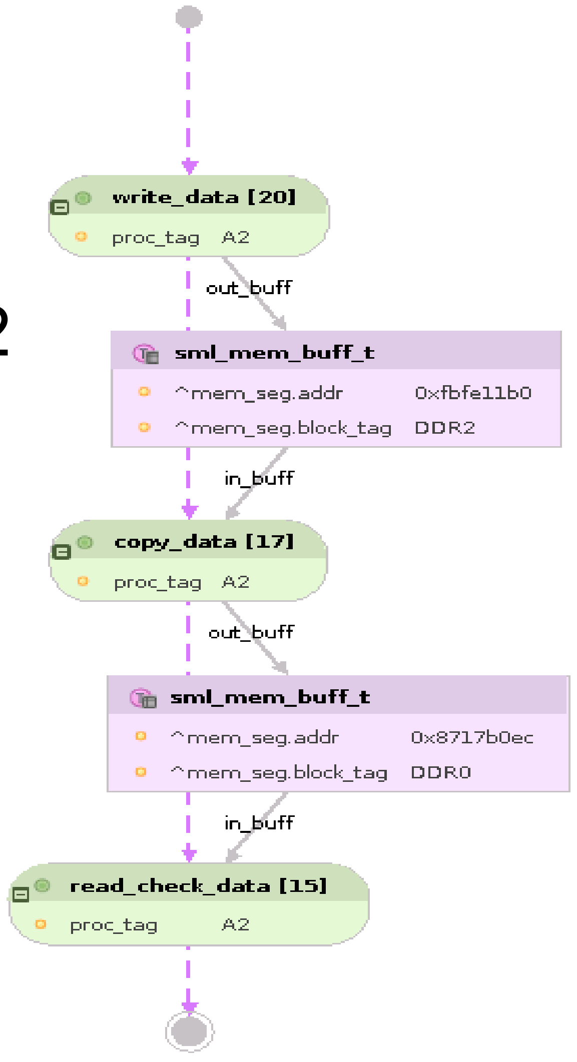
- Sanity test to ensure that basic processor-memory paths are working.
- Atomic actions provided by Perspec library:
  - write_data
  - copy_data
  - read_check_data
- Built-in data type:
  - sml_processor_tag_e

```
action pss_wr_cp_rc {
  rand sml_processor_tag_e proc_tag_l;

  activity {
     sequence {
        do sml_sw_ops_c::write_data with {
           proc_tag == proc_tag_l;
        };
        do sml_sw_ops_c::copy_data with {
           proc_tag == proc_tag_l;
        };
        do sml_sw_ops_c::read_check_data with {
           proc_tag == proc_tag_l;
        };
     };
  };
};
```

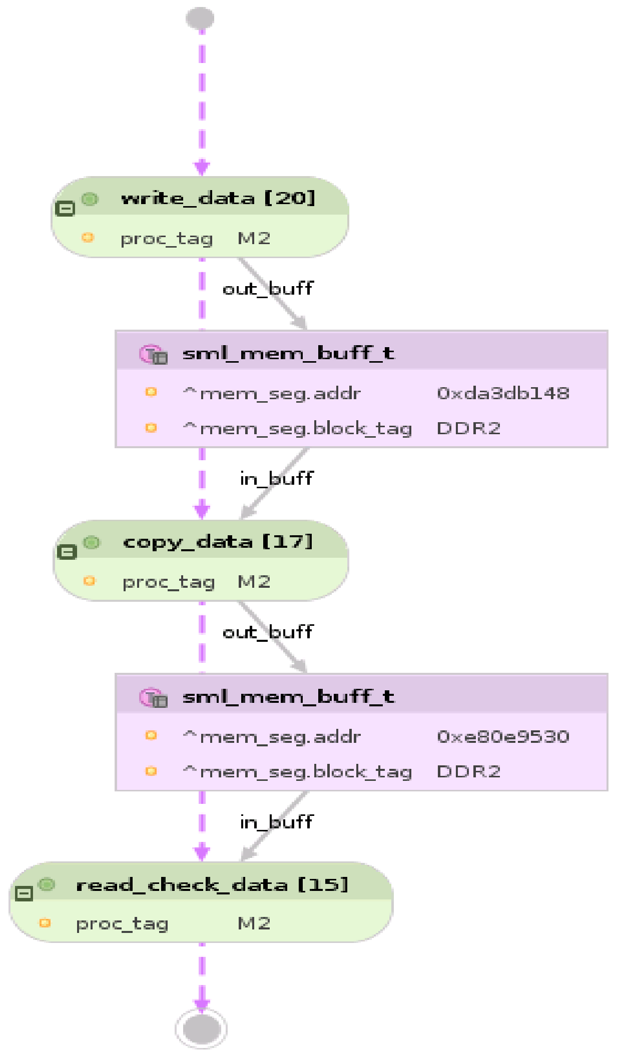
Memory R/W Test - Solutions

- 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

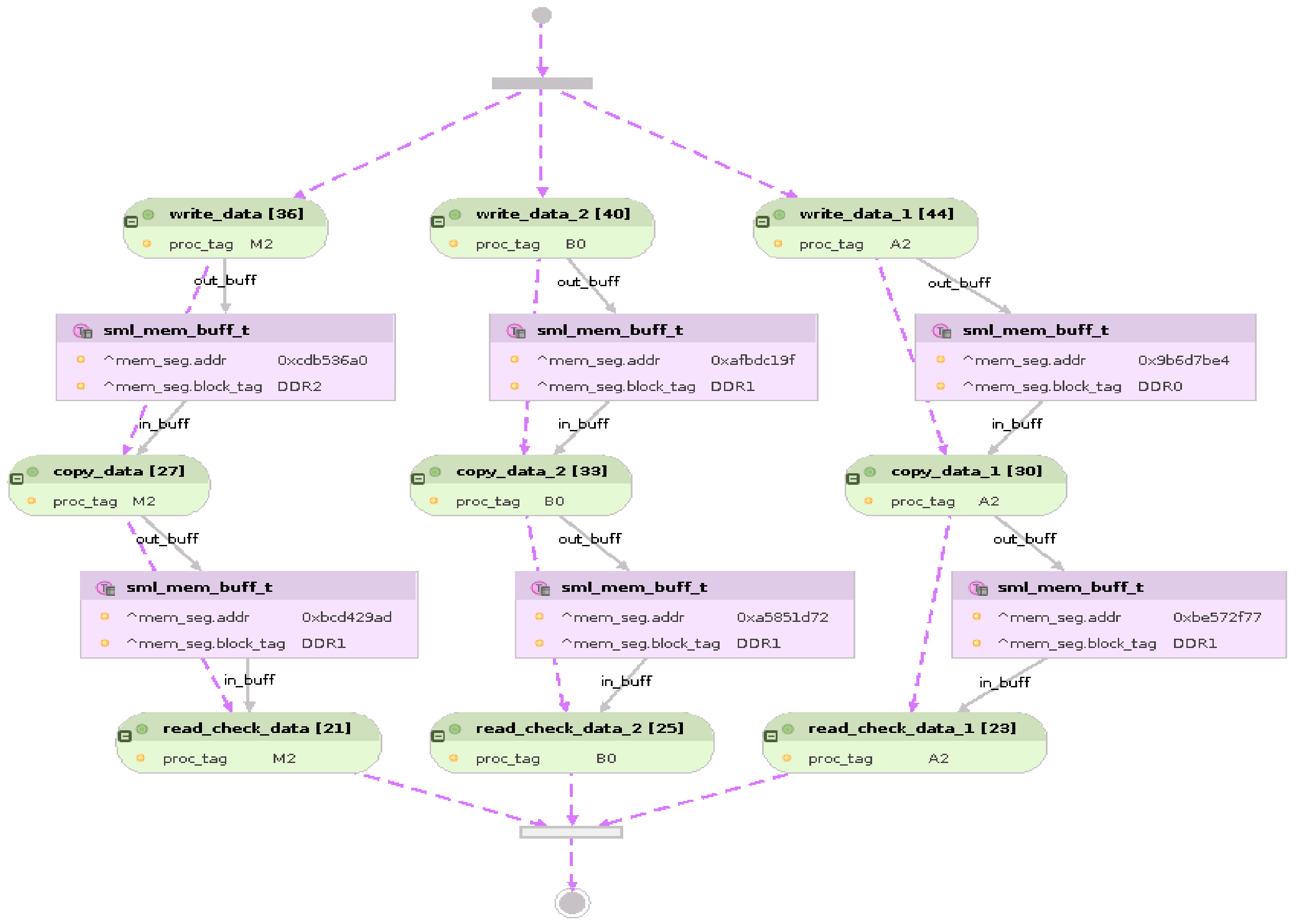
- Show how previous compound action (`pss_wr_cp_rc`) is used to build more complex PSS scenario

```
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

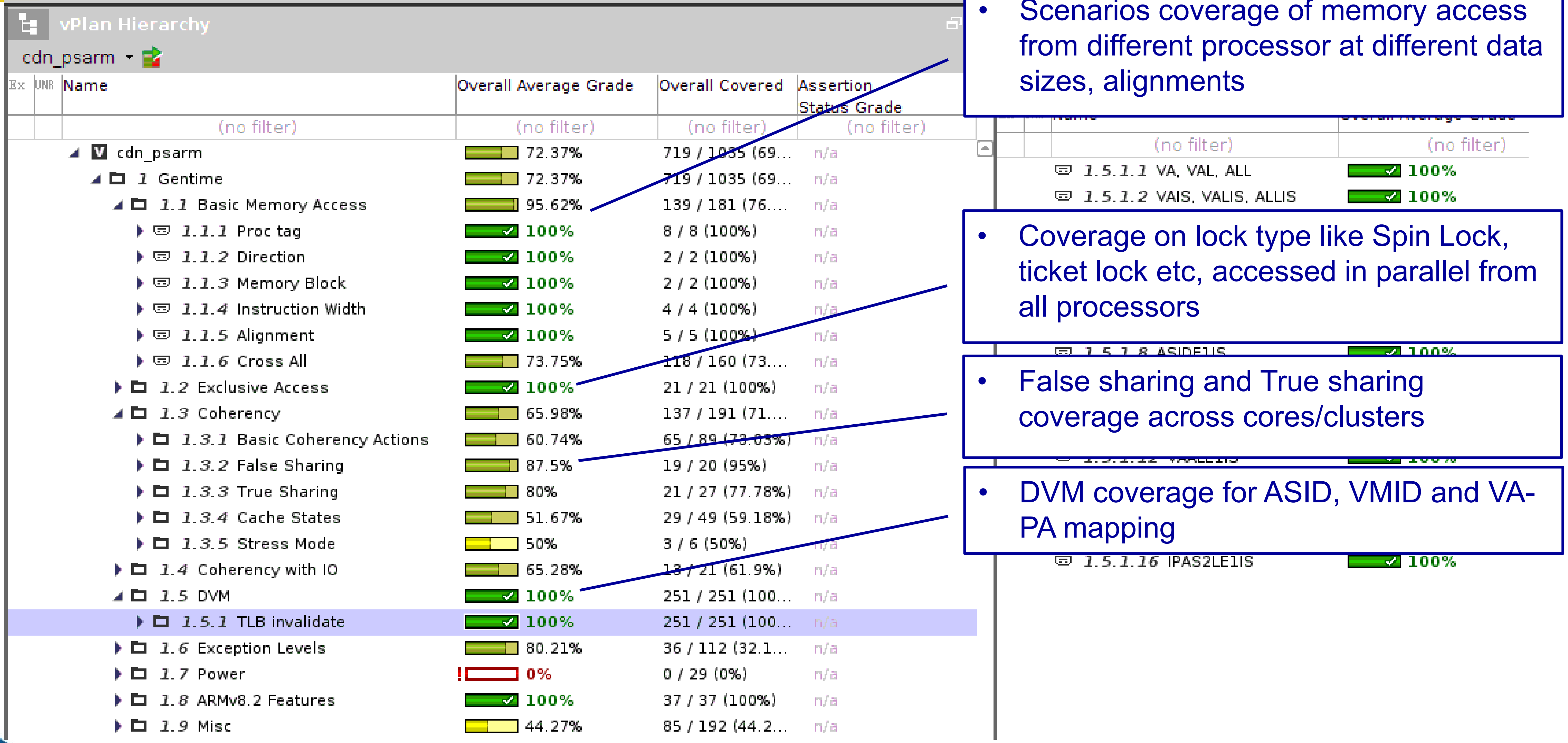


Coherency Test Suite

- 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



vPlan Hierarchy

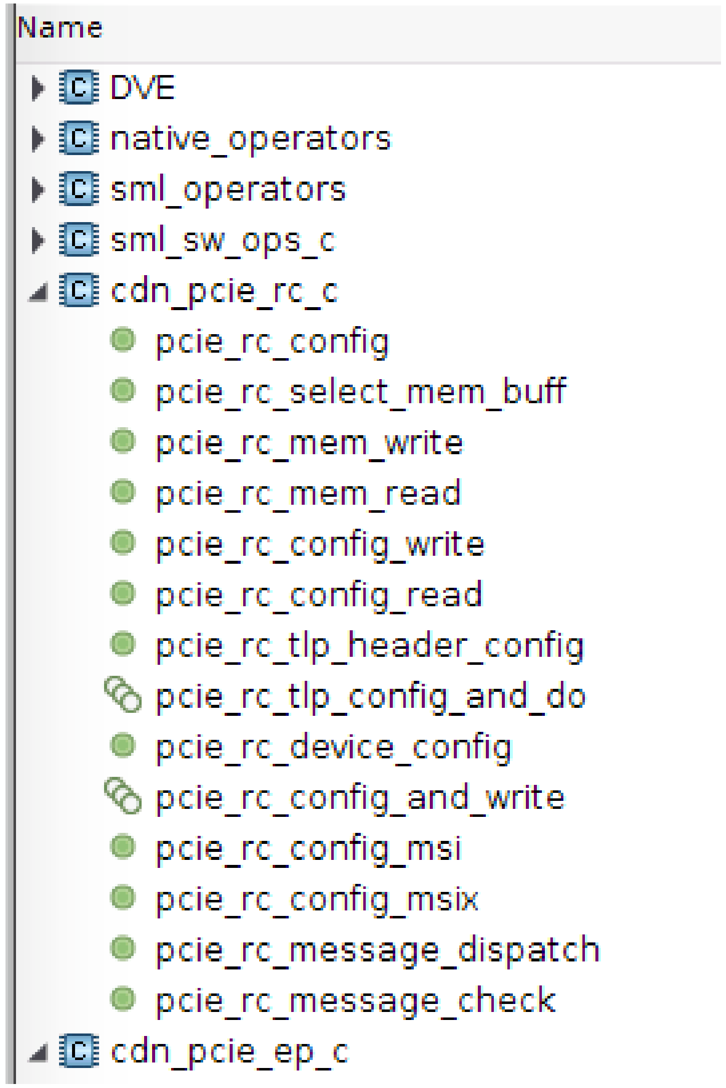
cdn_psarm

Name	Overall Average Grade	Overall Covered	Assertion Status Grade
(no filter)	(no filter)	(no filter)	(no filter)
cdn_psarm	72.37%	719 / 1035 (69...	n/a
1 Gentime	72.37%	719 / 1035 (69...	n/a
1.1 Basic Memory Access	95.62%	139 / 181 (76....	n/a
1.1.1 Proc tag	✓ 100%	8 / 8 (100%)	n/a
1.1.2 Direction	✓ 100%	2 / 2 (100%)	n/a
1.1.3 Memory Block	✓ 100%	2 / 2 (100%)	n/a
1.1.4 Instruction Width	✓ 100%	4 / 4 (100%)	n/a
1.1.5 Alignment	✓ 100%	5 / 5 (100%)	n/a
1.1.6 Cross All	73.75%	118 / 160 (73....	n/a
1.2 Exclusive Access	✓ 100%	21 / 21 (100%)	n/a
1.3 Coherency	65.98%	137 / 191 (71....	n/a
1.3.1 Basic Coherency Actions	60.74%	65 / 89 (73.03%)	n/a
1.3.2 False Sharing	87.5%	19 / 20 (95%)	n/a
1.3.3 True Sharing	80%	21 / 27 (77.78%)	n/a
1.3.4 Cache States	51.67%	29 / 49 (59.18%)	n/a
1.3.5 Stress Mode	50%	3 / 6 (50%)	n/a
1.4 Coherency with IO	65.28%	13 / 21 (61.9%)	n/a
1.5 DVM	✓ 100%	251 / 251 (100...	n/a
1.5.1 TLB invalidate	✓ 100%	251 / 251 (100...	n/a
1.6 Exception Levels	80.21%	36 / 112 (32.1...	n/a
1.7 Power	! 0%	0 / 29 (0%)	n/a
1.8 ARMv8.2 Features	✓ 100%	37 / 37 (100%)	n/a
1.9 Misc	44.27%	85 / 192 (44.2...	n/a

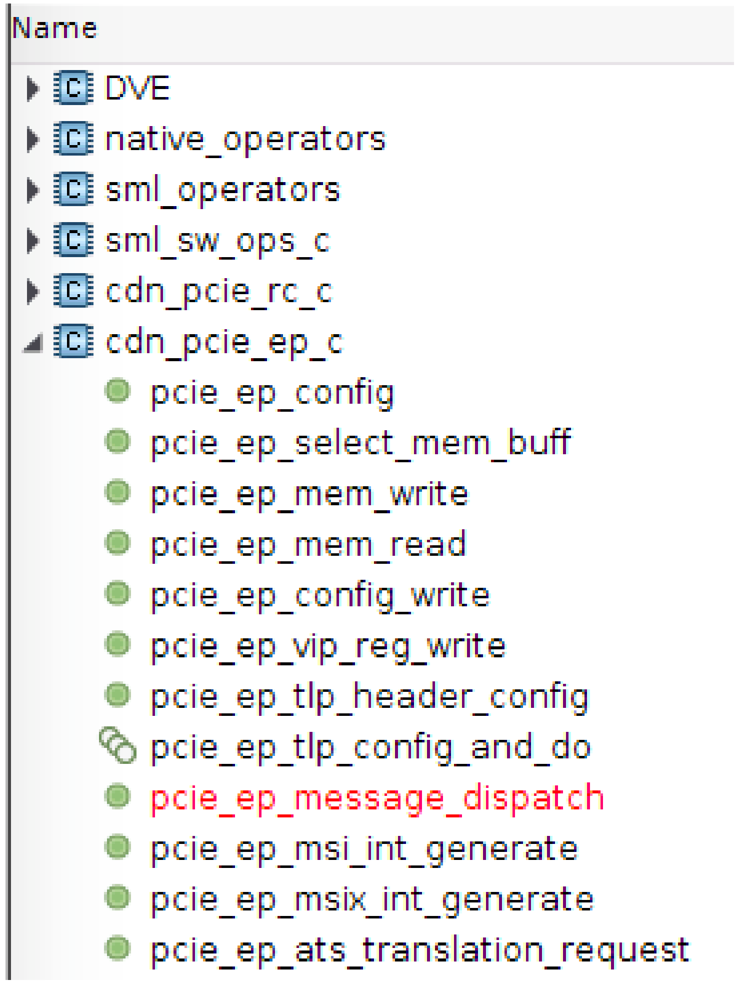
Name	Overall Average Grade
(no filter)	(no filter)
1.5.1.1 VA, VAL, ALL	✓ 100%
1.5.1.2 VAIS, VALIS, ALLIS	✓ 100%
1.5.1.8 ASIDE1IS	✓ 100%
1.5.1.16 IPAS2LE1IS	✓ 100%

- 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



```
Name
▸ DVE
▸ native_operators
▸ sml_operators
▸ sml_sw_ops_c
◢ cdn_pcie_rc_c
    pcie_rc_config
    pcie_rc_select_mem_buff
    pcie_rc_mem_write
    pcie_rc_mem_read
    pcie_rc_config_write
    pcie_rc_config_read
    pcie_rc_tlp_header_config
    pcie_rc_tlp_config_and_do
    pcie_rc_device_config
    pcie_rc_config_and_write
    pcie_rc_config_msi
    pcie_rc_config_msix
    pcie_rc_message_dispatch
    pcie_rc_message_check
◢ cdn_pcie_ep_c
```

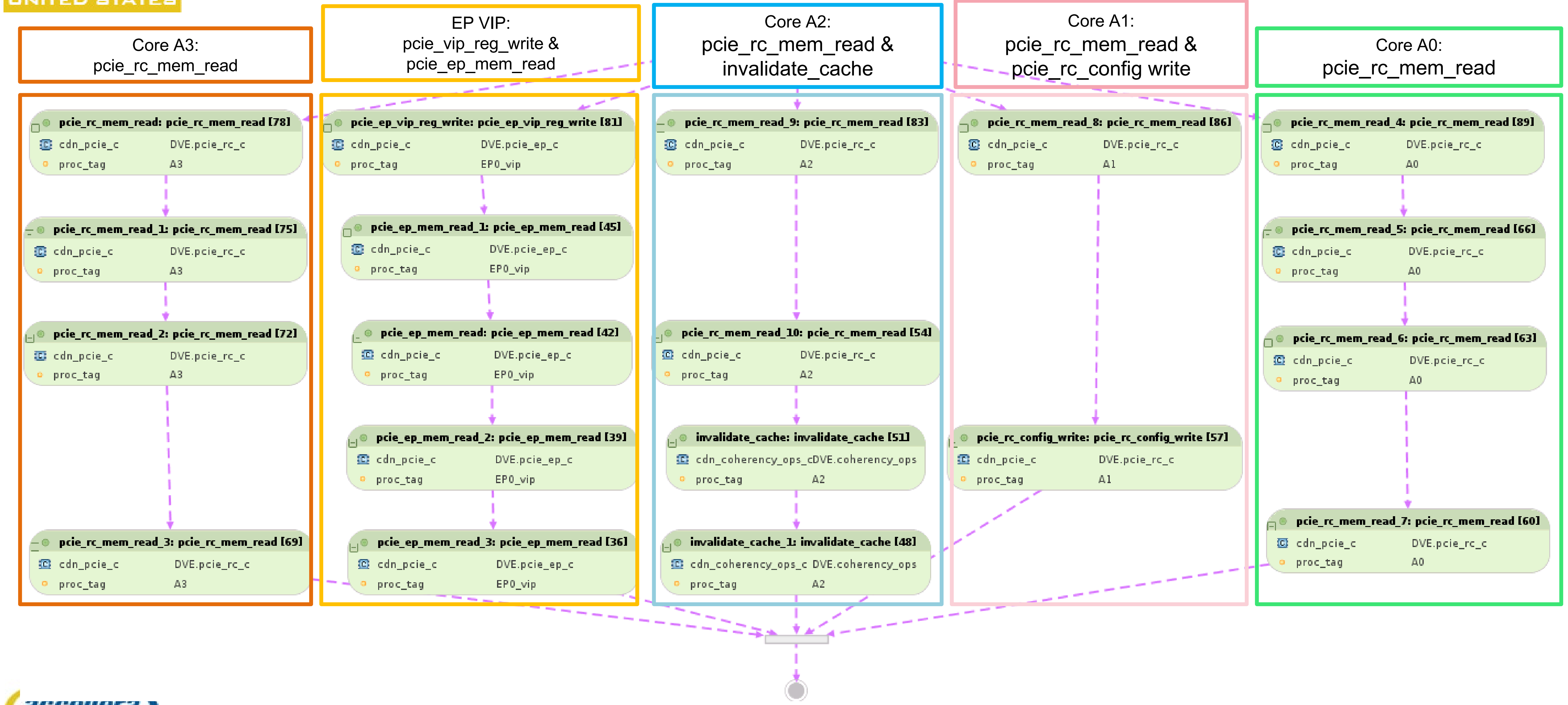


```
Name
▸ DVE
▸ native_operators
▸ sml_operators
▸ sml_sw_ops_c
▸ cdn_pcie_rc_c
◢ cdn_pcie_ep_c
    pcie_ep_config
    pcie_ep_select_mem_buff
    pcie_ep_mem_write
    pcie_ep_mem_read
    pcie_ep_config_write
    pcie_ep_vip_reg_write
    pcie_ep_tlp_header_config
    pcie_ep_tlp_config_and_do
    pcie_ep_message_dispatch
    pcie_ep_msi_int_generate
    pcie_ep_msix_int_generate
    pcie_ep_ats_translation_request
```

Deadlock Verif Scenario – Pseudo Code

```
//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

- pcie_rc_mem_read: pcie_rc_mem_read [78] — cdn_pcie_c DVE.pcie_rc_c; proc_tag A3
- pcie_rc_mem_read_1: pcie_rc_mem_read [75] — cdn_pcie_c DVE.pcie_rc_c; proc_tag A3
- pcie_rc_mem_read_2: pcie_rc_mem_read [72] — cdn_pcie_c DVE.pcie_rc_c; proc_tag A3
- pcie_rc_mem_read_3: pcie_rc_mem_read [69] — cdn_pcie_c DVE.pcie_rc_c; proc_tag A3

EP VIP: pcie_vip_reg_write & pcie_ep_mem_read

- pcie_ep_vip_reg_write: pcie_ep_vip_reg_write [81] — cdn_pcie_c DVE.pcie_ep_c; proc_tag EP0_vip
- pcie_ep_mem_read_1: pcie_ep_mem_read [45] — cdn_pcie_c DVE.pcie_ep_c; proc_tag EP0_vip
- pcie_ep_mem_read: pcie_ep_mem_read [42] — cdn_pcie_c DVE.pcie_ep_c; proc_tag EP0_vip
- pcie_ep_mem_read_2: pcie_ep_mem_read [39] — cdn_pcie_c DVE.pcie_ep_c; proc_tag EP0_vip
- pcie_ep_mem_read_3: pcie_ep_mem_read [36] — cdn_pcie_c DVE.pcie_ep_c; proc_tag EP0_vip

Core A2: pcie_rc_mem_read & invalidate_cache

- pcie_rc_mem_read_9: pcie_rc_mem_read [83] — cdn_pcie_c DVE.pcie_rc_c; proc_tag A2
- pcie_rc_mem_read_10: pcie_rc_mem_read [54] — cdn_pcie_c DVE.pcie_rc_c; proc_tag A2
- invalidate_cache: invalidate_cache [51] — cdn_coherency_ops_cDVE.coherency_ops; proc_tag A2
- invalidate_cache_1: invalidate_cache [48] — cdn_coherency_ops_c DVE.coherency_ops; proc_tag A2

Core A1: pcie_rc_mem_read & pcie_rc_config write

- pcie_rc_mem_read_8: pcie_rc_mem_read [86] — cdn_pcie_c DVE.pcie_rc_c; proc_tag A1
- pcie_rc_config_write: pcie_rc_config_write [57] — cdn_pcie_c DVE.pcie_rc_c; proc_tag A1

Core A0: pcie_rc_mem_read

- pcie_rc_mem_read_4: pcie_rc_mem_read [89] — cdn_pcie_c DVE.pcie_rc_c; proc_tag A0
- pcie_rc_mem_read_5: pcie_rc_mem_read [66] — cdn_pcie_c DVE.pcie_rc_c; proc_tag A0
- pcie_rc_mem_read_6: pcie_rc_mem_read [63] — cdn_pcie_c DVE.pcie_rc_c; proc_tag A0
- pcie_rc_mem_read_7: pcie_rc_mem_read [60] — cdn_pcie_c DVE.pcie_rc_c; proc_tag A0

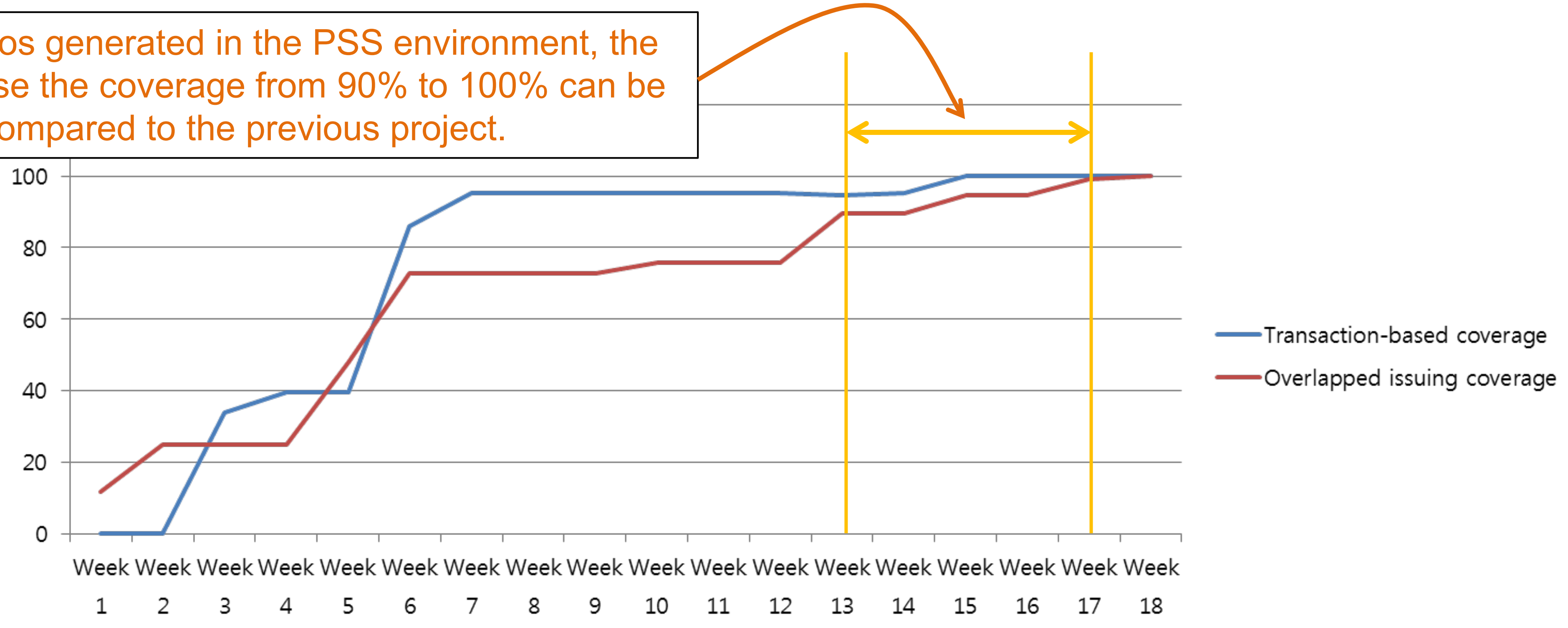
Conclusions & Lessons Learned

- 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 and Notes

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)