A Hybrid Verification Approach for Cache Coherent Systems: Functionality and Performance

Jiang-Tang Xiao, Osmond Yao

MediaTek Building E, No.8, Dusing 1st Rd., Hsinchu Science Park, Hsinchu City 300, Taiwan

Jiang-Tang.Xiao@mediatek.com, Osmond.Yao@mediatek.com

Yung Cheng Chen, Harish Peta
Cadence Design Systems, 2655 Seely Ave, San Jose, CA 95134, USA
vicchen@cadence.com, hpeta@cadence.com

***Abstract*- This paper proposes a comprehensive verification approach for cache coherent systems, addressing the challenges of meaningful stimulus, system coherent checkers, and performance analysis tools. The approach combines Perspec, CPU, CHI VIP, and system verification scoreboard to ensure correct cache coherent protocol operation and identify performance issues. The hybrid verification environment enables functional verification and performance analysis, leveraging inheritable sequences to reuse test stimuli across projects. The approach captures cache coherency and system performance bugs, identifying corner RTL issues that cause system hangs and performance bottlenecks, leading to improved system reliability and performance.**

I. INTRODUCTION

Cache coherent systems are increasingly complex and critical components of modern computing architectures, but their verification remains a significant challenge. The lack of meaningful stimulus, system coherent checkers, and system performance analysis tools hinders the verification of these systems, leading to potential errors and system failures. This paper proposes a comprehensive verification approach that addresses these challenges by combining the strengths of Perspec[1], CPU, CHI VIP[2], System verification scoreboard[3], and System Performance Analyzer[4].

Our approach consists of two main components: cache coherent system verification and system performance verification. In the cache coherent system verification component, This hybrid verification environment is employed in both cache coherent system verification and system performance verification, integrating Perspec, CPU, and CHI VIP to generate stress coherent stimulus, ensuring the correct operation of the cache coherent protocol. The above test scenarios are generated by the Perspec built-in cache coherency library. Furthermore, protocol functional coverage is utilized to ensure the strength of the verification. The hybrid verification environment enables the completion of functional verification, followed by system performance analysis using stress bandwidth stimulus. Additionally, there is another test scenario about system performance where the test sequence is based on the UVM sequences. The inheritable UVM sequence feature of CHI VIP is leveraged to reuse test stimuli across different projects, enabling performance analysis and identification of performance issues specific to the current project.

The proposed approach enables the capture of cache coherency and system performance bugs that may not be detected by traditional verification methods. Our methodology demonstrates the effectiveness of this approach in identifying corner RTL issues (incorrect system cache line status) that cause system bus fabric hang and contribute to performance bottlenecks due to incorrect u-architecture design, leading to improved system reliability and performance.

II. METHODOLOGY

The following steps describe the verification and implementation methods:

1. Hybrid verification overview
2. Stress cache coherent stimulus for function and performance
3. System verification scoreboard connected with DUT
4. System performance analysis
5. Functional coverage development

Step-1: Hybrid verification overview

The hybrid verification environment is introduced, comprising a DUT that consists of multiple CPUs with level 1 and level 2 caches, interconnected through an ARM DynamIQ Shared Unit (DSU) with a level 3 cache. The primary communication protocol between the DUT and the System-on-Chip (SoC) is based on the AMBA CHI (Coherent Hub Interface). The Coherent Mesh Network, a 4x4 mesh-based bus fabric, can be connected to the Cluster using AMBA CHI, and these designs constitute the DUT. The CHI VIP is utilized to interface with 8 interfaces of the Coherent Mesh Network, with the primary objective of generating stress cache coherent operations to verify the cache coherency functionality of the Coherent Mesh Network and Cluster.

In the stimulus section, the Perspec-built cache coherent library is employed to generate embedded C tests and host C tests scenarios, enabling the Coherent Mesh Network to produce corresponding snoop operations in response to requests and perform cache coherent test scenarios. The CHI VIP's built-in protocol coverage is allowed for further review of the verification strength. Additionally, the custom CHI UVM sequence is tested for system performance. More details will be introduced in the Step-2 section.

The checker section utilizes the System Verification Scoreboard to ensure the correctness of the SoC and Cluster CHI bus functions and cache coherent functionality, providing a comprehensive checker. In subsequent sections, it will be discussed how this checker was used to capture an incorrect cache line status in the corner RTL issue. In addition to functional verification, system performance is also a crucial indicator of product competitiveness. The system performance analyzer is introduced to measure system bandwidth and latency across different scenarios, thereby determining whether performance issues arise.

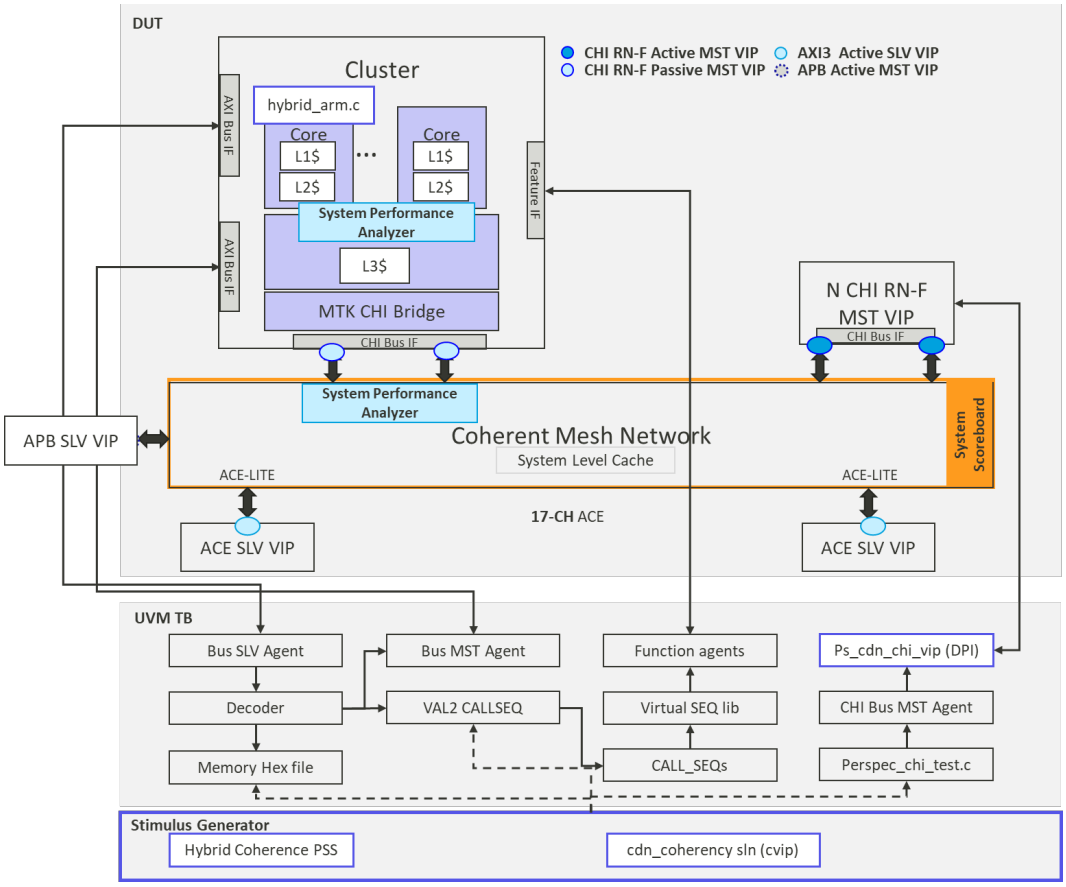

Figure 1: Hybrid Verification Environment Overview.

Step-2: Stress cache coherent stimulus for function and performance

This section presents two stimulus methods, which are the Perspec built-in cache coherent library and the CHI VIP UVM sequence, respectively. Primarily, the cache coherent library includes nine kinds of coherent scenarios, including basic memory access, exclusive, false sharing, true sharing, cache state transition, basic coherency, false-true sharing, atomic, and stress cache operations. Perspec generated embedded C tests and host C tests are executed to activate system cache coherent test scenarios and meaningful snoop transaction of Coherent Mesh Network then access the Core/DSU/Coherent Mesh Network system-level cache, thereby achieving system coherent verification. First, the user needs to determine the number of cores and CHI VIPs to be used based on the system configuration, as shown in Figure 2. It is also necessary to pre-register the memory space to allow Perspec to generate cache line addresses within the correct range. The second step involves using the Perspec built-in library to generate C tests for both the CPU and CHI VIP. Finally, the C tests are run on the system to execute the coherent system simulation.

The verification environment comprising multiple CPUs and a CHI VIP within the same memory space shareable is illustrated in Figure 3. Furthermore, The Perspec built-in cache coherency library provides a diverse range of solutions that users can directly inherit and utilize. This ensures that both core and CHI VIP can allocate the same cache line and execute multiple read/write operations. Subsequently, self-checking is performed in the cache region to guarantee data consistency.

The completeness of cache coherent function verification is crucial, and the bandwidth of the DUT is a key performance indicator. It has been observed that comparing bandwidth of all cache hit scenarios across different projects and bandwidth results can reveal bandwidth drops of DUT due to micro-architecture issues under the same test scenario conditions.

This performance test scenario method will be introduced in the following section. First, a CHI UVM sequence is developed for the level 3 cache read-hit scenario and executed by CHI VIP. Then, the transaction traffic is recorded as the input for the system performance analyzer. Once the testbench is set up, it allows us to take different DUTs to trigger the same CHI traffic and observe the performance raw data from different projects using the System Performance Analyzer (SPA) post-processing, as shown in Figure 4. The CHI UVM sequence is developed based on an understanding of the CHI protocol and cluster level-3 cache. Moreover, this sequence can be easily inherited and reused across different project testbenches. The sequence begins with the VIP sending a MakeUnique command to set the cache line to a unique cache line state. Then, the cache data is pre-filled using the WriteBackFull command. Subsequently, coherent read commands are sent to ensure that the cache lines are read and hit, with no external memory access observed. Furthermore, the outstanding number of the CHI VIP is controlled using plusargs in SystemVerilog to create different bandwidth scenarios, as shown in Figure 5. The CHI VIP records transaction information at runtime, including command, data, and round-trip latency. After the simulation completes, users can obtain a complete transaction trace log file. This log file is then provided to the system performance analyzer for post-processing and analysis. In Step-5, the system performance analyzer is used to analyze the bandwidth of these test scenarios, and a performance drop is identified after bandwidth data comparison.

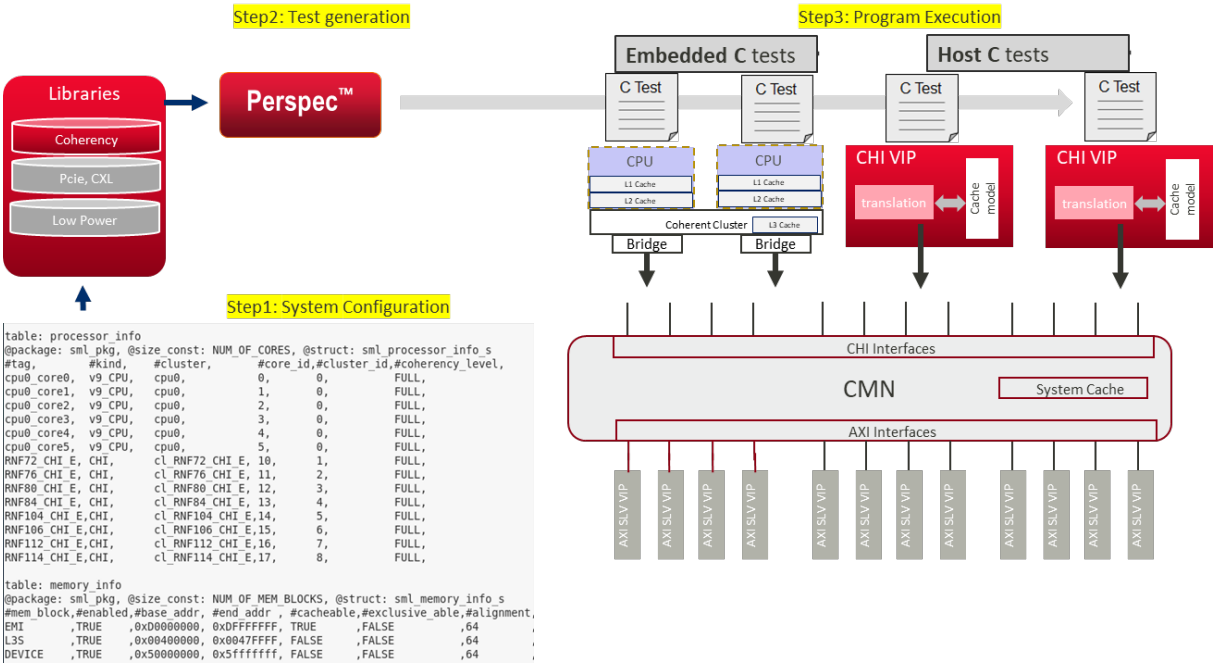

Figure 2: Cache coherent stimulus generation flow.

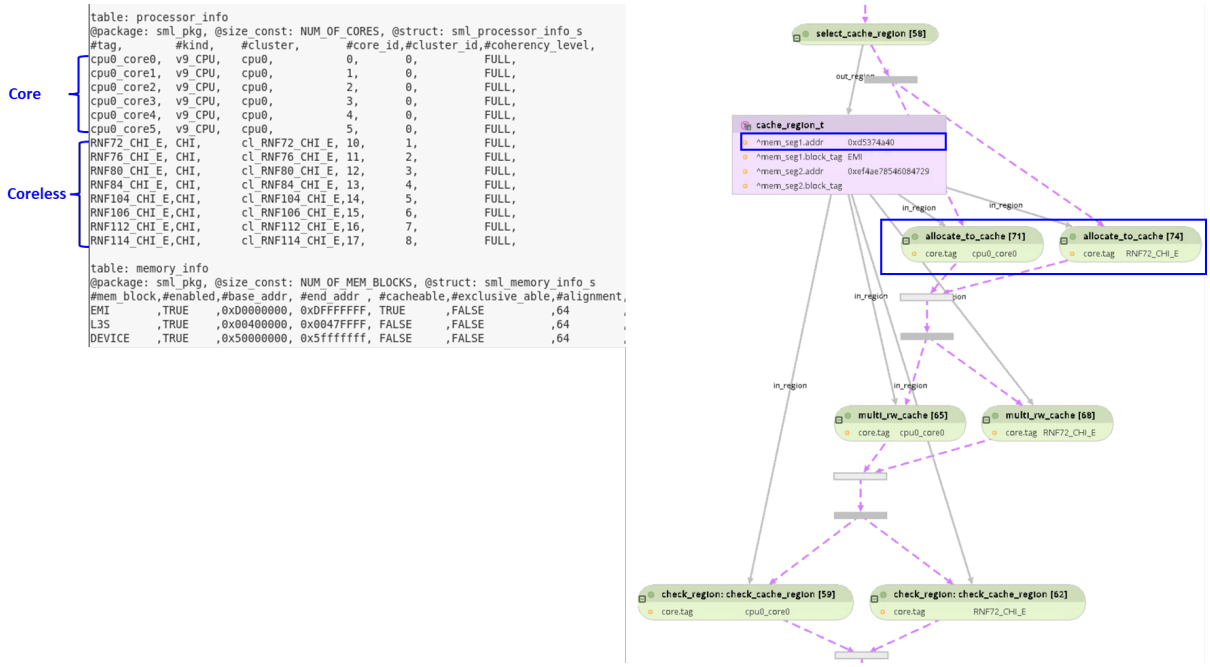

Figure 3: Determining the number of Cores and CHI VIPs through system SPEC. Cores and VIP are accessing the same address simultaneously.

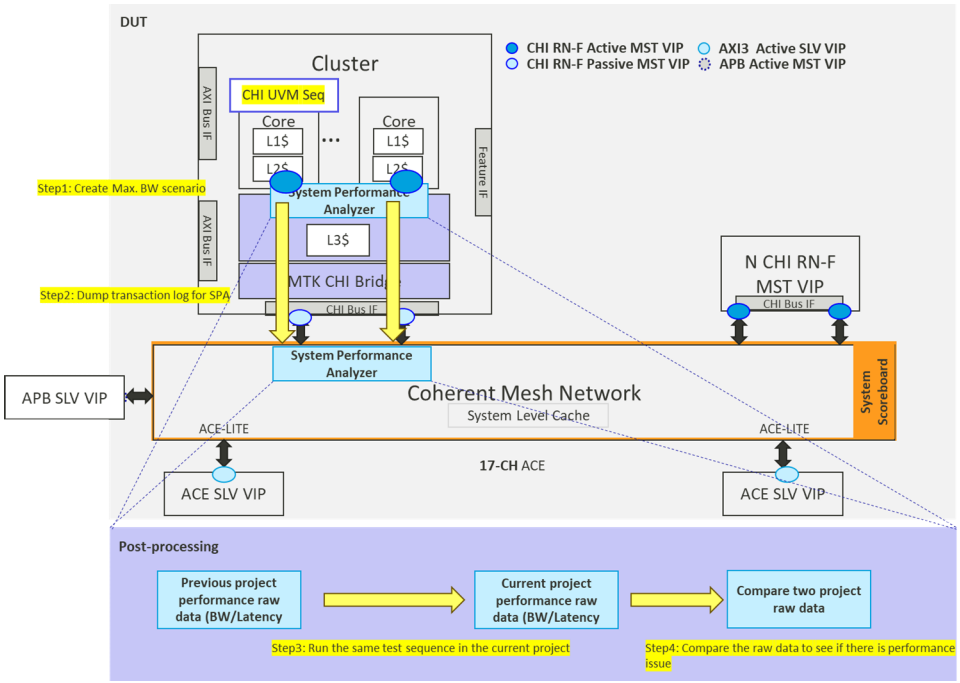

Figure 4: Read cache-hit sequence for performance test scenario.

```
`define DSU_L3_Read_Hit_Transaction (opcode, addr)          \
  `uvm_do_on_with(request, p_sequencer.pAgent.sequencer, { \
    request.ReqOpCode           == ``opcode``;             \
    request.MemAttr             == 'hd;                    \
    request.RSVDC[`INNER_ALLOC] == 1;                      \
    request.Addr                == ``addr``;               \
  });

class performance_seq_scenario extends base_seq_scenario ;
  `uvm_object_utils(performance_seq_scenario)

  function new(string name="performance_seq_scenario");
    super.new(name);
  endfunction

  virtual task body();
    int chi_ostd;
    super.body();

    if($value$plusargs("chi_ostd=%d", chi_ostd))
      p_sequencer.pAgent.cfg.MaxOutStandingTransactions = chi_ostd;
    p_sequencer.pAgent.reconfigure();

    for (int i=0; i<tr_length; i++) begin
      `DSU_L3_Read_Hit_Transaction(DENALI_CHI_REQOPCODE_MakeUnique        , `BASED_ADDR + i*'h40)
      `DSU_L3_Read_Hit_Transaction(DENALI_CHI_REQOPCODE_WriteBackFull     , `BASED_ADDR + i*'h40)
    end

    for (int i=0; i<tr_length; i++) begin
      `DSU_L3_Read_Hit_Transaction(DENALI_CHI_REQOPCODE_ReadNotSharedDirty, `BASED_ADDR + i*'h40)
    end
  endtask
endclass
```

Figure 5: Read cache-hit CHI sequence.

Step-3: System Verification Scoreboard connected with DUT

In cache coherent system verification, the CHI VIP's built-in protocol checker and system verification scoreboard are utilized to ensure bus functionality and cache coherent protocol correctness. This section focuses on the application of the system verification scoreboard.

It is necessary to identify the master agents and slave agents in the system and define the memory regions that master agents can access based on the system memory map. Then, it is required to determine whether each agent belongs to a cache coherent node. The system verification scoreboard features an internal pseudo snoop filter that checks for transactions with the violation of cache protocol. Finally, it is necessary to define each connection between slave agents and master and ensure no bus decode issues in the system.

Through the system verification scoreboard, we have observed that IP assumption mismatches can lead to unexpected cache line status was updated by DUT, resulting in system hangs.

```
virtual function void definePorts();
   inst.definePort("input0", DENALI_SVD_SIDE_INPUT, DENALI_SVD_PROTOCOL_CHI,DENALI_SVD_INTERFACETYPE_COHERENT);
   inst.definePort("input1", DENALI_SVD_SIDE_INPUT, DENALI_SVD_PROTOCOL_CHI,DENALI_SVD_INTERFACETYPE_COHERENT);
   //
   inst.definePort("output0",  DENALI_SVD_SIDE_OUTPUT, DENALI_SVD_PROTOCOL_AXI,DENALI_SVD_INTERFACETYPE_BASIC);
   inst.definePort("output1",  DENALI_SVD_SIDE_OUTPUT, DENALI_SVD_PROTOCOL_AXI,DENALI_SVD_INTERFACETYPE_BASIC);
   inst.definePort("output2",  DENALI_SVD_SIDE_OUTPUT, DENALI_SVD_PROTOCOL_AXI,DENALI_SVD_INTERFACETYPE_BASIC);
   inst.definePort("output3",  DENALI_SVD_SIDE_OUTPUT, DENALI_SVD_PROTOCOL_AXI,DENALI_SVD_INTERFACETYPE_BASIC);
endfunction // definePorts

virtual function void mapMemorySegments();
   inst.mapMemorySegmentToMultipleSlaves("input0",48'h0000_8000_0000,48'h0ff_ffff_ffff,{"output0","output1", "output2", "output3"});
   inst.mapMemorySegmentToMultipleSlaves("input1",48'h0000_8000_0000,48'h0ff_ffff_ffff,{"output0","output1", "output2", "output3"});

   inst.setInnerDomain("Inner", {"input0","input1"});
endfunction // mapMemorySegments
```

Figure 6: Example of system master and slave defined in system verification scoreboard for cache coherent protocol checks.

Step-4: System Performance Analysis

The previous verification methods are effective in detecting functional design issues. However, performance issues are not able to be identified during the early RTL development stage. Therefore, this section describes how the system performance analyzer is introduced to facilitate system verification.

The system performance analyzer can analyze the bandwidth and latency of the VIP node during a single simulation process and utilizing CHI-VIP to dump the simulation trace file as an input for the analyzer, as shown in Figure 7. This analysis result applies the maximum latency to be an outlier index with the corresponding transaction attribute (e.g., address/command type/length) within the current test regression. From Figure 7, the summary provides an analysis of six CHI masters, detailing the total number of transactions issued throughout the entire simulation, along with the corresponding minimum, maximum, and average latency, average bandwidth, and outstanding transactions. Additionally, the trend of average bandwidth over the entire simulation time is illustrated. This graphical representation aids in identifying latency outliers and assists in the design process, as shown in Figure 8. The design team can apply this result to determine whether the latency behavior meets expectations then quickly locate the issue point. On the other hand, the inherited test sequence is utilized to perform bandwidth testing on the system. The current project's bandwidth raw data of the CHI VIP binding location is compared with previous cases to determine whether performance issues occurred. A comparative analysis of the design between the previous project and the current one reveals that the successful utilization of a performance analyzer enabled the identification of performance degradation stemming from u-architecture issues.

Summary Table

Bandwidth Calculation Window (µs) : 2008.248264 - 2047.330104

Interface Name	Protocol	Num of transactions	Bytes	Max Latency (Round Trip), ns	Avg Latency (Round Trip), ns	Min Latency (Round Trip), ns	Max Bandwidth MB/s
Filter Interface Name	Filter Protocol						
CHI_RN_MST_CORE_5	CHI	1651	105664	1390.0	271.162	80.0	4465.894
CHI_RN_MST_CORE_4	CHI	1674	107136	1405.0	270.596	80.0	4017.66
CHI_RN_MST_CORE_3	CHI	1571	100544	1580.0	294.056	100.0	3749.716
CHI_RN_MST_CORE_2	CHI	1543	98752	1425.0	295.669	100.0	4060.189
CHI_RN_MST_CORE_1	CHI	1596	102144	1305.0	264.934	80.0	3719.81
CHI_RN_MST_CORE_0	CHI	1591	101824	1460.0	296.445	80.0	3799.812

Figure 7: Summary of CHI node bandwidth and average latency by a single pattern.

Figure 8: Trend graphs of CHI VIP bandwidth and latency under single pattern conditions.

Step-5: Functional coverage development

In cache coherent system verification, there are two essential and fundamental functional coverages that can be used to measure the quality of verification. Firstly, the protocol coverage built into the CHI VIP can effectively confirm that protocol attributes of each CHI mode are fully verified. The Hybrid verification environment proposed in this paper, combined with user-defined test sequences and achieved 83% single coverpoints and 90% cross coverpoints of the CHI interface protocol, comprising 1460 CHI VIP cover points and 5380 cross-cover points.

The second important indicator is that the scenario coverage of Perspec can be used to measure whether the defined cache coherent scenarios have been correctly generated in embedded C tests and host C tests. As shown in Figure 9, the scenario coverage for false sharing and true sharing has been covered. A functional coverage review can effectively demonstrate the quality of design verification and facilitate discussions on the reasons for uncovered cover bins, determining whether they need to be covered.

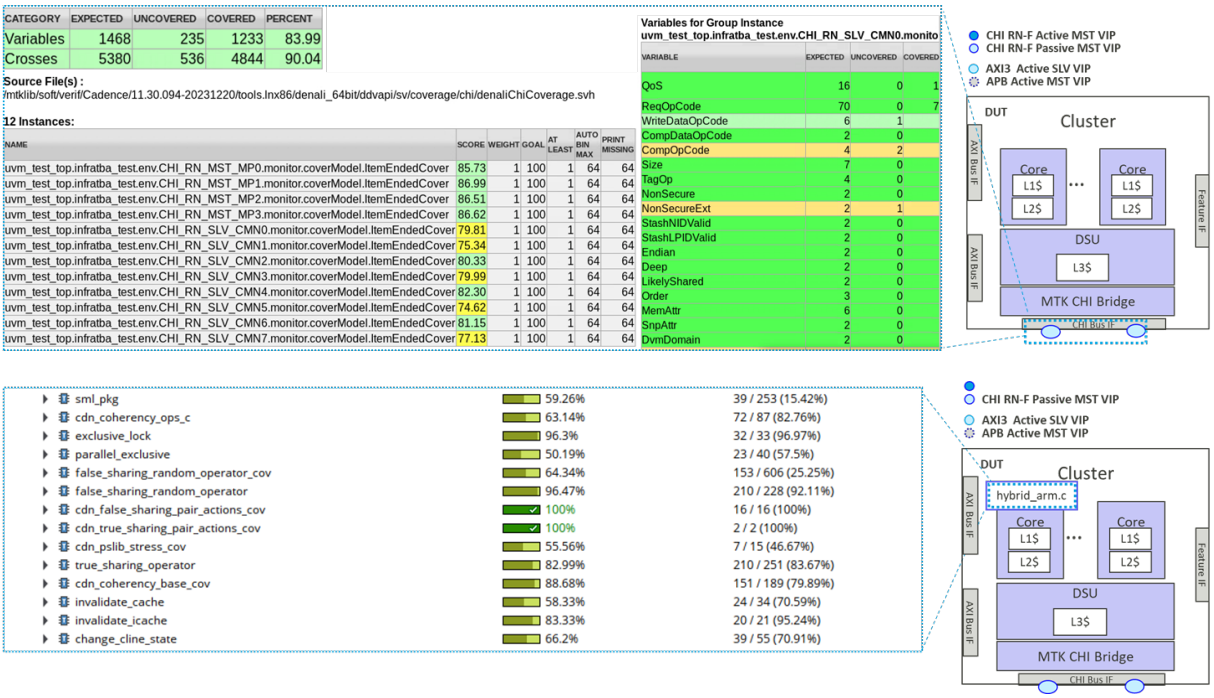

Figure 9: CHI VIP protocol and cache coherent test scenario coverage.

III. CASE STUDY

This section presents two case studies. The case A involves an RTL issue captured by the system verification scoreboard. The case B involves a performance drop observed through the system performance analyzer's bandwidth raw data via cross-project comparison.

A. Capture of Incorrect Cacheline Status by System Verification Scoreboard

The current Hybrid verification environment, where the CHI VIP sends a ReadOnce command, is illustrated in Figure 10. It is expected that the Coherent Mesh Network will subsequently send a SnpOnce command to the Cluster based on the cache line. However, the Coherent Mesh Network sends SnpOnce commands to each CHI coherent node in certain application and each CHI node in the Cluster receives a SnpOnce command. The first core completes SnpOnce and sends the cache line with Unique Clean state to Coherent Mesh Network. When the 2nd core completes SnpOnce, it still sends the cache line with Unique Clean state hence violating the cache protocol. According to the cache protocol, only one cache coherent node can send the cache line with Unique Clean State to the Coherent Mesh Network, while other coherent nodes must update the cache line with invalid state. This issue can be captured by the system verification scoreboard using its built-in checker, as shown in Figure 11.

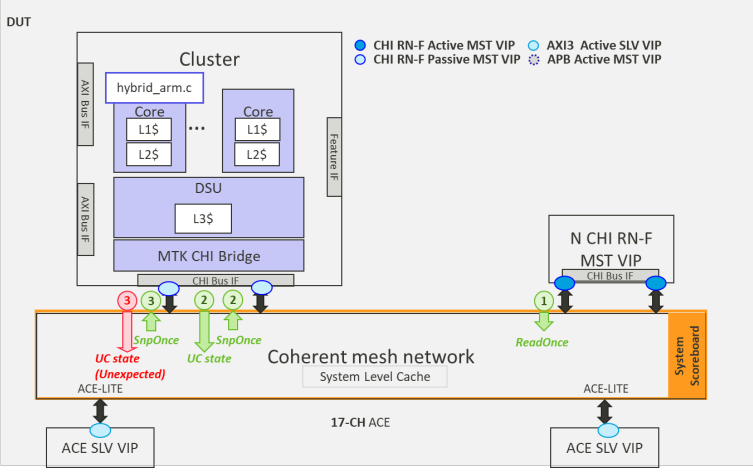

Figure 10: Capture of unexpected cache line status by system verification scoreboard leading to system hang.

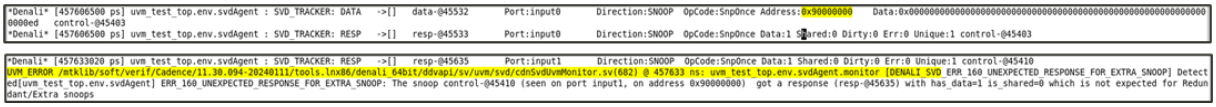

Figure 11: Checker details for capturing unexpected cache line status.

B. Capture of Bandwidth Drop due to u-Architecture Issue Observed by System Performance Analyzer

In each project, verifying the bandwidth and latency of a new cache coherent system is a crucial task of design verification. By comparing the bandwidth raw data obtained from the same CHI UVM sequence in previous projects with the one in current project, performance differences can be easily identified, as shown in Figure 12. In this case, the CHI bus topology change of the cache coherent system contributes to maximum bandwidth was congested with more transactions in performance degradation, as shown in Figure 13. Therefore, DV can challenge the architecture team to improve this issue via the experimental data.

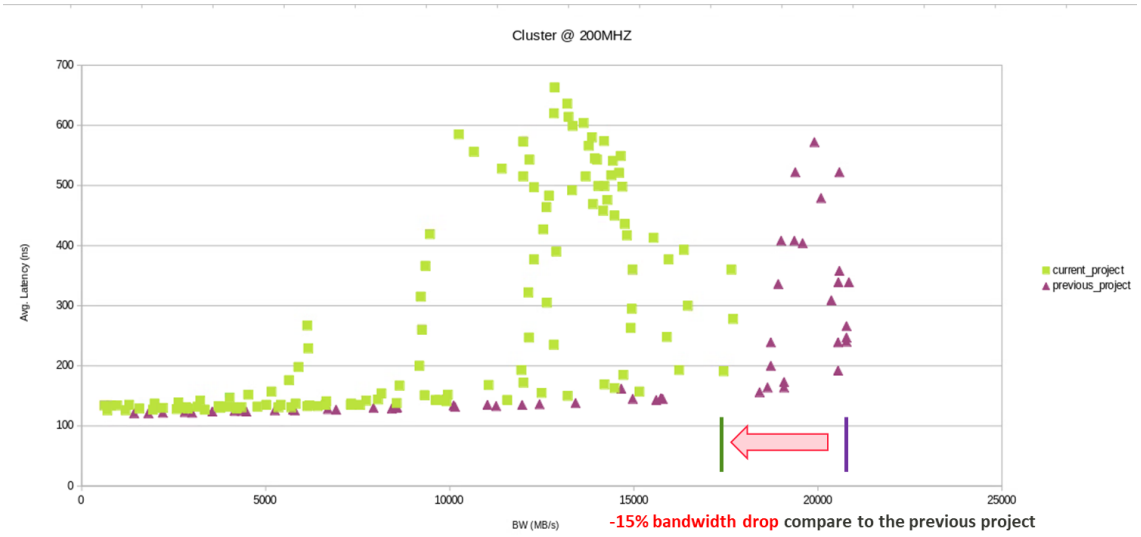

Figure 12: Identification of bandwidth performance drop in the current project using a performance analysis tool.

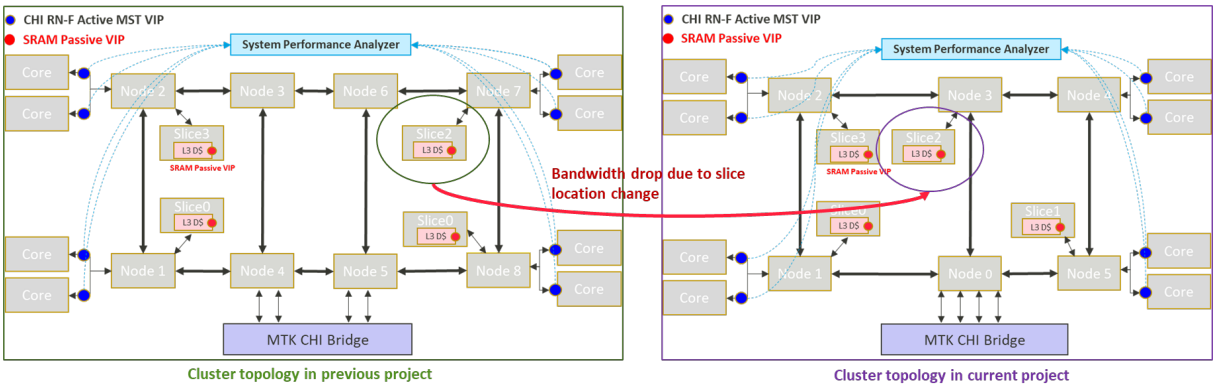

Figure 13: Capture of performance drop issue due to CHI bus topology change.

IV. CONCLUSION

This paper primarily describes how to verify coherency between cores and CHI-VIP by utilizing Perspec. Then, users can perform functional and performance verification based on this platform. The main contributions are:

- Developing 9 scenario groups of cache coherency, consisting of 134 tests for coherent function and performance stress scenarios.
- Introducing system verification scoreboard and performance analyzer to increase the diversity and strength of verification.
- Identifying corner RTL issues related to incorrect system cache line state that causes system bus fabric hang.
- Finding performance bottlenecks due to incorrect u-architecture design

REFERENCES

[1] Portable Test and Stimulus Standard Version 3.0
https://www.accellera.org/downloads/standards/portable-stimulus

[2] CHI VIP:
https://www.cadence.com/zh_TW/home/tools/system-design-and-verification/verification-ip/simulation-vip/amba/amba-chi.html

[3] System Verification Scoreboard:

https://www.cadence.com/en_US/home/resources/technical-briefs/system-vip-tb.html#verification-scoreboard

[4] System Performance Analyzer:
https://www.cadence.com/en_US/home/resources/technical-briefs/system-vip-tb.html#performance-analyzer