# Advanced Techniques for ARM L2 Cache Verification in an Accelerated Hardware and Software environment

Rob Pelt
Altera Corporation
San Jose, CA

Jay O'Donnell
Mentor Graphics
Seattle, WA

**ABSTRACT:**

High-end ARM-based SOC designs typically implement an L2 cache controller to improve system performance and manage memory access. Such systems also provide external peripheral access to L2 and L3 main memory using the accelerator coherency port (ACP) on the processor. System performance can be optimized in the design of both hardware and software that configures and manages memory access. Verifying correct operation and performance in such a highly configurable system is a key goal.

This paper presents techniques to verify operation and performance of L2 cache in this type of system. Methods were developed supporting concurrent software and ACP hardware L2 accesses in a coordinated systematic fashion for various system configurations. Some configurations include software-configurable cache "ways" combined with different types of cache-able accesses coming from both ACP and software targeting cached memory in various states.

The requirement for dual access to L2/L3 by both software and hardware presents significant challenges due to the need to systematically manage two distinctly different processes using a common verification framework. Hardware accesses use AXI read and write transfers to drive the ACP port. Software accesses utilize a software test framework and operating system routines. A centralized testbench managing both types of accesses is needed to provide control and synchronization.

This work utilized intelligent testbench (iTBA) techniques to systematically manage both accesses, enumerate the verification state space (which had well in excess of 10 million cases) and manage the test scenario generation to meet coverage goals. Traditional approaches using constrained-random or directed testing were inadequate due to the extremely large number of test scenarios and the requirement, incompatible with random generation, to precisely manage the scenarios. Traditional functional coverage approaches to measure verification effectiveness were considered but ultimately rejected due to the complexity of instrumenting the environment. Also, such approaches were found to be unnecessary since the iTBA tool could automatically enumerate and target the stimulus state space.

A hardware-accelerated simulation environment was used to efficiently simulate the large number of scenarios. Results of the work and lessons learned are presented.

## 1. Introduction

Verifying the L2 cache from both the CPU and external AXI masters through the ACP presents a number of challenges:

- Implementing a flexible verification architecture supporting both software and hardware access managed by one central process
- Developing a flexible software framework and communication mechanism to coordinate software and hardware activity
- Developing a hardware test environment to drive ACP
- Developing a top-level control testbench capable of describing the overall test scenario and controlling hardware and software interfaces to generate the tests

## 2. Environment

The testbench environment is based on System Verilog and the OVM library. A number of Verification IP components (VIP) based on OVM form the testbench foundation. The testbench is also tightly bound to the CPU in the SOC, which can be used to coordinate and check testbench activity. Although the VIP components can be driven by

OVM sequences implemented using directed and constrained random approaches, additional advanced approaches were considered.

One testbench architectural challenge is coming up with a scheme for ACP access. This is because ACP is normally accessed via multiple AMBA fabric masters. Accessing ACP using these AXI masters could be very difficult to coordinate and control since the actual peripherals would need to generate the AXI traffic targeting ACP and rely on the fabric to deliver the transactions deterministically.

In order to simplify ACP access a single AXI fabric port having ACP access is configured to generate all ACP transactions directly with other ACP capable ports configured to be inactive.

Additionally, the OVM testbench must be able to communicate with the CPU's embedded software. This is done to coordinate the L2 cache state before issuing transactions to the ACP across the fabric, and to instruct the embedded software to initiate various L2 cache accesses while concurrently initiating cache accesses via ACP.

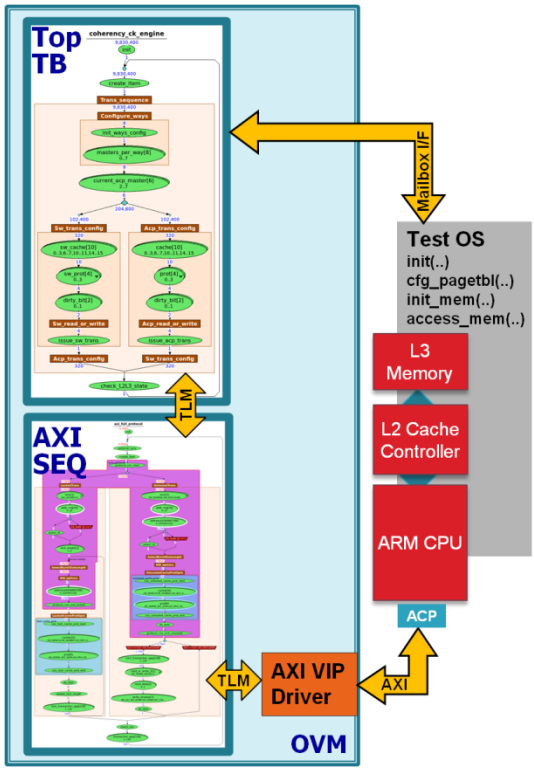


**Figure 1: Verification environment**

Figure 1 shows the main elements of the verification environment. Top TB manages generation of two streams of L2/L3 traffic:

- ACP
- CPU

A graph in Top TB defines the total stimulus state space and manages the generation of both stimulus streams during simulation.

ACP traffic implemented using TLM is passed to the AXI SEQ block, which in turn assembles an OVM sequence and passes the (TLM) sequence to the OVM AXI VIP driver for delivery on the ACP port interface. The AXI SEQ block is highly configurable and can implement all required ACP accesses.

CPU software accesses are also managed by Top TB, but use a custom mailbox communication scheme that lets Top TB call pre-configured software routines in the test operating system to initiate L2 accesses and monitor L2 and L3 states.

### 3. Beyond constrained random

To improve coverage and efficiency, a graph based approach was chosen to test the complete scope of the protocol, specifically in the space of the AXI protocol relative to cache-based transactions to the ACP port. This approach assured complete coverage of the ACP-related protocol space and efficient coverage of corner cases. The AXI SEQ graph gives a comprehensive view of the functional space we intend to cover, thus giving critical feedback on the parameters that are covered and those that are intentionally left out. Careful reviews of the graph can also help identify ACP-related parameters of the AXI protocol that we might have missed. This graph-based approach is much easier to visualize and review for accuracy than a scattered list of random variables and provides direct tabulation of the size of the stimulus state space. This lets the user identify which parts of the graph are important to target during simulation (colored highlights) and which run randomly. The approach also supports targeting multiple stimulus generation regions with different priorities, which improves verification efficiency and early bug detection.

The inFact graph-based intelligent testbench tool was selected for this application because of its ability to precisely generate all test combinations non-redundantly as the graph is traversed during simulation. This differs from a traditional constrained-random testing (CRT) approach which requires the addition of external coverage measurement code to validate when test combinations are hit. Such CRT flows typically require re-running of the tests using different seeds to cause different random variable combinations to be hit. On average at least 10x more CRT tests are required to achieve coverage due to the probabilities of random variable selection.

These advantages of the graph-based approach eliminate requirements to write coverage code and iteratively run tests with different seeds, resulting in significant savings in testbench development time and overall time savings in the verification process.

**4. Top TB implementation**

Figure 2 shows the top testbench graph responsible for coordinating the overall testbench execution controlling ACP and CPU accesses of L2/L3 cache. Sizing information for the different graph segments is also shown.

This graph specifies a loop of transaction sequences that could be a CPU ->ACP (Sw_trans->Acp_*) or an ACP->CPU for different "ways" configurations targeting different address regions and cache scenarios. The details of address region selection and cache scenarios are contained within the brown graph symbols that implement sub-graphs.

Some of the more important scenarios which this graph implements include:

- Cache initialization calling Test OS routines
- Access ordering: Software or ACP first
- Ways configurations
- Cache control and protection bit setting combinations, both accesses
- Cache line state in L2 and L3 and cache access scenarios based on state
- Backdoor access using Test OS routine to verify L2 and L3 states during different accesses

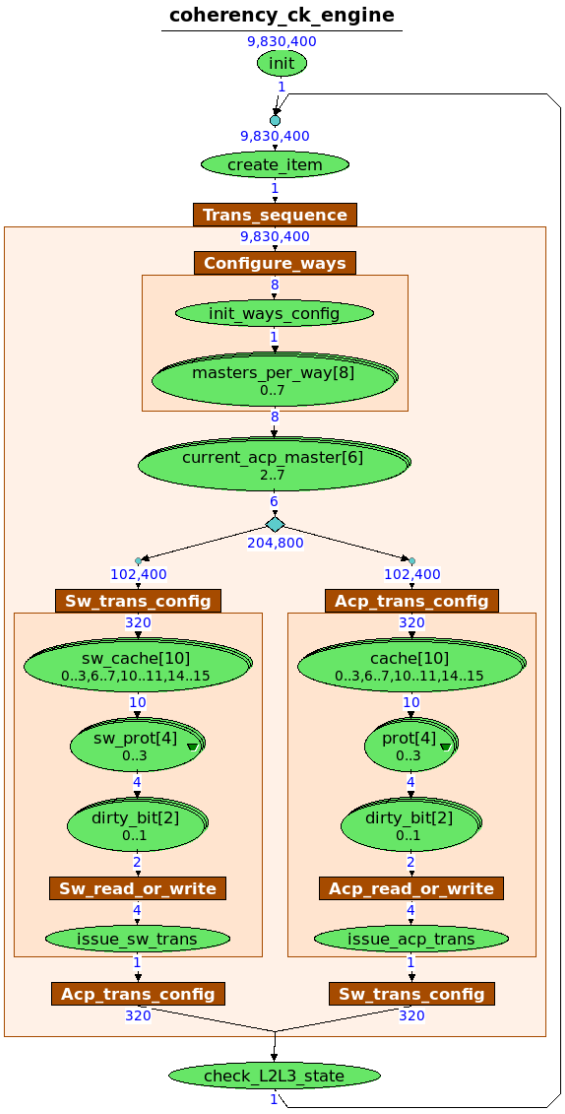


**Figure 2: TOP TB generation graph with sizing**

Modeling L2 and L3 states under different configurations is important to assure proper cache operation. To do this, the initial states need to be established and then accesses must be precisely managed by careful selection of addresses. This helps to simplify predictive model of cache state, which will be implemented as a side object in the top level testbench.

**5. AXI SEQ implementation**

Figure 3 shows the graph construction for the AXI SEQ block that manages AXI accesses. A close inspection of the graph shows sizing for different graph segments that gives visibility into stimulus size

and helps plan which stimulus combinations are important and practical to target in combination.

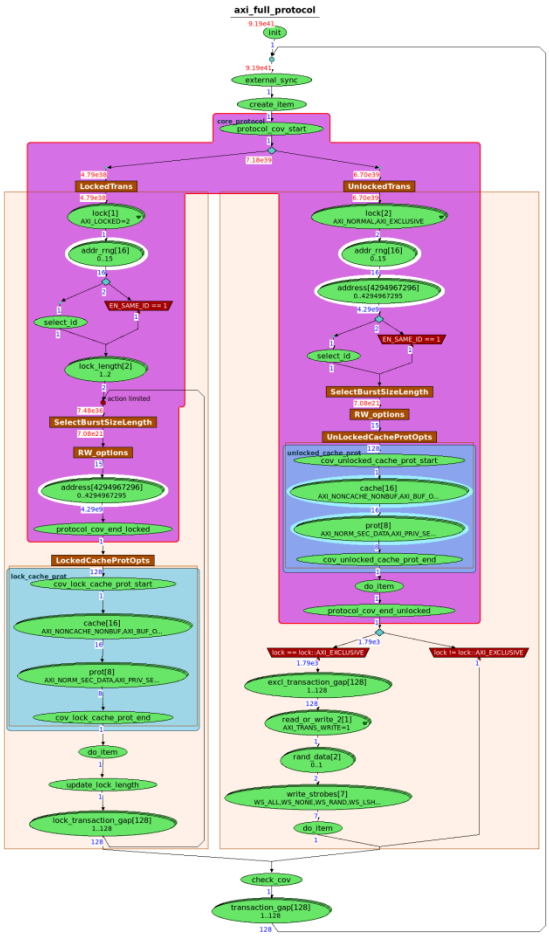


**Figure 3: AXI SEQ generation graph with sizing**

## 6. Test initiation from the embedded software

Testing cache operations from the CPU's embedded software is fairly easy. But when ACP transactions are submitted from external AXI masters, the embedded software must be used to set up the cache into known states. This requires communications between the graph-based sequence and the embedded software.

The mailbox communication scheme shown in Figure 1 provides a mechanism for calling functions in the Test OS and verifying OS state calling status functions. A collection of functions were developed that support memory initialization and configurable memory accesses. The number of such functions is fairly small, though the number of possible accesses is quite large due to the combinations of function arguments and calling order. The Top TB graph manages the order of function calls and pass argument values when constructing the various test scenarios.

## 7. Accelerating the test

Cache operations and the sheer number of required tests will quickly exceed practical limits of simulation. Therefore the OVM testbench, the VIP components and the overall model of the design must also be compatible with emulation-based simulation acceleration. This will permit the full spectrum of cache operations to be tested.

## 8. Findings and conclusion

The initial work verifying correct L2 operation relied on a purely software-based scheme. In this scheme, graph-generated static software routines created a block of loadable C code that was executed on the Test OS to verify basic cache operation in different ways scenarios. The graph was run once to generate the test code block, which was later run on the CPU during emulation.

This technique validated basic cache subsystem operation but lacked support to verify ACP accesses.

Various ideas were considered to add ACP, including both pre-generation of a purely software-based scheme that included calls to external OVM sequences to initiate ACP accesses. This scheme, while feasible, had a number of drawbacks:

- The generated test code would be huge, requiring some sort of paging to break up the generated tests into manageable blocks
- Dynamic control of test execution was not possible since everything was pre-generated
- Varying the access order was not feasible without re-generating the test code running the generator graph with different software seeds
- Time synchronization might present problems since it had to be done from the CPU

We concluded that the simpler approach would locate the top-level testbench control in the simulation environment running under OVM, where the CPU and Test OS would be slaved to a single simulation-based OVM process. Among this scheme's benefits:

- Easier to develop
- Eliminated the complexity of paging in different tests
- Test ordering could be changed on the fly during simulation
- Easier to make the testbench reactive to simulation and CPU state since the graph traversal engine is actively managing processes and can dynamically adapt to state changes

This work is currently underway and results will be available for discussion during the DVcon 2012 session.