# Using Portable Stimulus to Verify Cache Coherency in a Many-Core SoC

Adnan Hamid
Breker Verification Systems, Inc.
San Jose, CA, USA
adnan@brekersystems.com

David Koogler
Cavium, Inc.
Marlborough, MA, USA
david.koogler@cavium.com

Thomas L. Anderson
Breker Verification Systems, Inc.
San Jose, CA, USA
toma@brekersystems.com

## I. INTRODUCTION

Cache coherency has historically been regarded as one of the most challenging verification problems. Design errors typically lead to data corruption in memory that can be very hard to debug. This paper presents a new approach to this verification: the automatic generation of self-checking test cases for system-on-chip (SoC) designs and multi-SoC systems that stress the design and exercise all variations of the cache coherency protocol. These test cases can be generated for all verification platforms, matching the vision for the "portable stimulus" standard currently being developed by Accellera Systems Initiative. This paper starts by discussing in Sections II and III why modern SoC architectures make cache coherency a system-level problem rather than one confined to the CPU team. Some of the key verification challenges are discussed in Section IV, including the types of test cases that must be run in order to ensure proper cache functionality. Sections V and VI present the new approach in detail. The paper concludes in Sections VII and VIII with results from a real-world project automatically generating test cases for 144 cores in a heterogeneous multi-SoC system and running on silicon using a portable stimulus approach.

## II. MOTIVATION

For many years, cache coherency verification remained an issue only for central processing unit (CPU) teams [1]. Whenever multiple processors using caches were combined, either as part of a semiconductor design or as intellectual property (IP) incorporated into others' chips, the vendor was responsible for verifying that data remained coherent across all caches in use by the processors. A system-on-chip (SoC) designer may license IP for one or more CPU cores, but multiple cores were typically delivered in a self-contained "cluster" that included all caches. Thus, verifying cache coherency was the responsibility of the cluster provider. The SoC and system verification teams assumed that the caches just worked and spent little or no effort re-verifying them at the full-chip or system level.

This situation has changed dramatically with recent and current generations of large chips. There is a clear trend in the industry of the most complex designs moving to SoC architectures with multiple embedded processors. Many of these chips, especially in server and networking applications, contain dozens or even hundreds of processors linked by a common multi-level on-chip bus, chip fabric, or a network-on-chip (NoC) interconnect. As the number of processors grows beyond a single cluster of four or eight, a new era of "many-core" SoCs is beginning. This type

of design introduces new verification challenges, including the need to stimulate the design with multiple simultaneous transactions from processors to memory while verifying cache coherency at the SoC level.

As shown in Figure 1, there are multiple categories of designs evolving to become SoCs with many processors. The factors driving this trend include:

- Large chips adding embedded processors to implement complex functionality while retaining flexibility
- Single-processor chips adding multiprocessor clusters to get better performance at a given process node
- Multiprocessor chips using shared memory for effective data transfer and interprocess communication
- Neighbor-connected processors moving to shared memory to reduce cross-chip latency

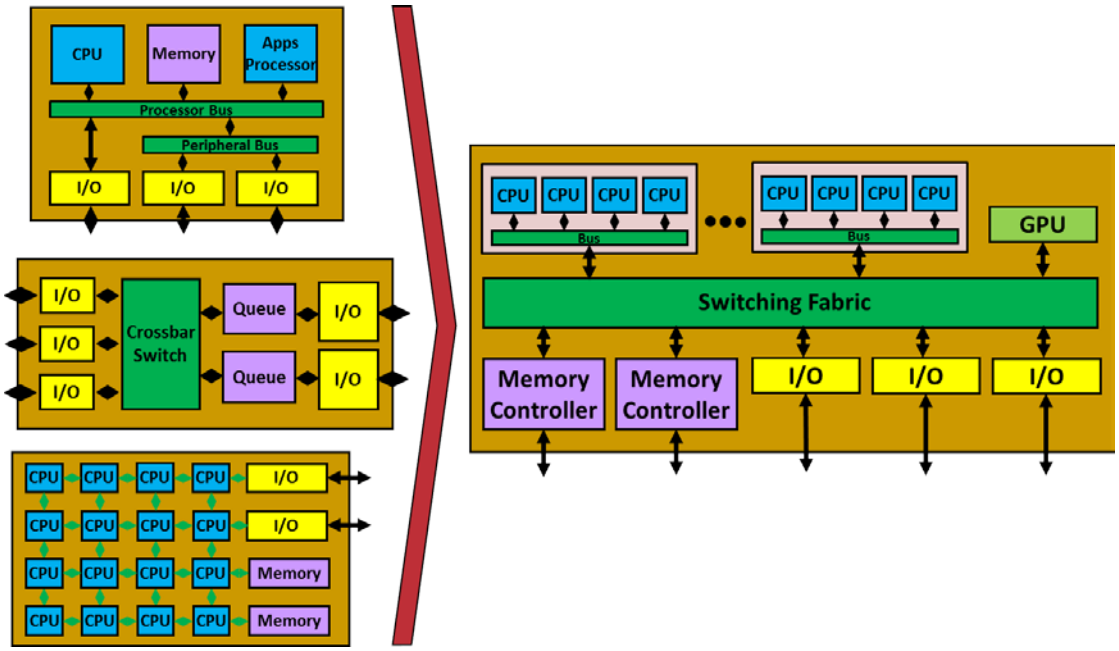


Figure 1. Evolution of Large Chips to Multiprocessor SoCs

## III. CACHE CONFIGURATIONS

There are common themes in the industry as chips move toward cache-coherent SoCs. The general architecture shown in Figure 1 includes multiple levels of caches and multiple kinds of coherency. Within a single processor cluster, as shown in Figure 2, each CPU typically has its own dedicated Level 1 (L1) cache, and a Level 2 (L2) cache spanning all the CPUs in the cluster. Although there may be security rules that restrict certain processors from accessing certain regions of memory, or certain I/O devices, all the processors share a common memory space and may at times be accessing the same memory locations. Thus coherency is required, even within the cluster, to ensure that the right data is read from a shared memory location and that all writes to memory either update or invalidate all copies of the memory location in the caches. Section IV discusses related verification requirements in more detail.

As previously noted, if the only agents that need to be coherent are contained within a single cluster received from the processor vendor, then the SoC team may not need to think about the problem at all. However, SoC teams are increasingly connecting multiple clusters together, as shown in Figure 3. This architecture, an elaboration of the general design shown in Figure 1, includes multiple processor clusters that must remain fully coherent. It also adds

additional agents, including a graphics processing unit (GPU), a digital signal processor (DSP), and input/output (I/O) controllers that also may have varying degrees of coherency requirements. For example, an I/O write driven by direct memory access (DMA) might be required to ensure that data read from memory reflects the latest update to the memory location across all agents and caches.

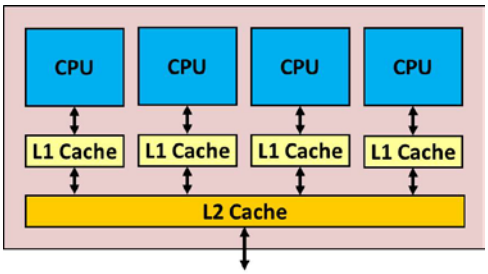


Figure 2. Typical Cache-Coherent Multiprocessor Cluster

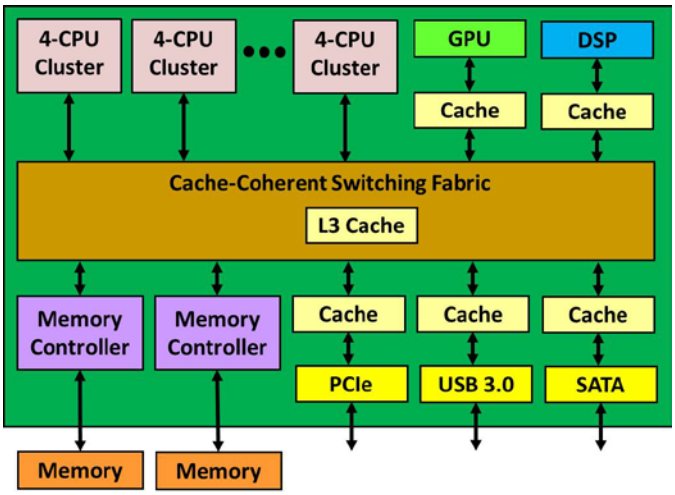


Figure 3. New Class of SoC with Multiple Coherent Agents

This design has two additional elements of note. First, there is a Level 3 (L3) cache that spans all CPUs and other coherent agents. Further, this L3 cache is contained within, and supported by, a cache-coherent switching fabric that interconnects all major agents and memories in the chip. This figure is representative of a whole class of emerging designs, and a number of the chips unveiled at the Hot Chips Symposium and the Linley Group Processor Conference have block diagrams that look quite similar [2][3]. While the approach may be common, a single solution for verifying SoC-level cache coherency is not. Processor vendors may provide no assistance verifying coherency beyond a single cluster as shipped. Interconnect IP vendors may deliver cache-coherent switching fabrics but not provide the necessary test cases to verify full-SoC coherency in simulation or on a hardware platform.

## IV. CACHE COHERENCY VERIFICATION CHALLENGES

There are many reasons why coherency is fundamentally hard to verify, and these reasons apply even more at the SoC level than at the processor level [4]. Different caches in different CPU clusters or other agents and at different levels (L1, L2, L3) may have different cache line widths and different address maps. Referenced data not crossing a

cache line in one cache may cross a line in another. Different instructions using different data widths access caches differently. There any typically many possible variations on what happens when a memory access is attempted by an agent, and these may involve the local cache, other caches, and memory. The actions that occur are determined by a protocol that underlies every cache coherency scheme. That protocol can be described in the form of a state machine, and it is feasible for CPU developers to completely verify all single-processor state transitions, but expanding cache coherency to the system level adds much more complexity.

MOESI (modified-owned-exclusive-shared-invalid) is an example of a widely used full cache coherency protocol. Its chief distinguishing features over other protocols is that it avoids the need to write modified data back to memory before sharing it and can defer the write-back operation. The protocol is so named because at any given point each cache line can be in one of five possible states:

- Modified – the current cache has the only valid copy of the cache line, and has made changes to that copy.
- Owned – the current cache is one of several with a valid copy of the cache line, but has the exclusive right to make changes to it. It must broadcast those changes to all other caches sharing the line.
- Exclusive – The current cache has the only copy of the line, but the line is unmodified.
- Shared – The current line is one of several copies in the system. This cache does not have permission to modify the copy. Other processors in the system may hold copies of the data in the Shared state, as well.
- Invalid – No cache holds the line, so it must be fetched to satisfy any attempted access.

Describing all the actions and transitions that can occur is beyond the scope of this paper, but there are excellent sources available [5][6][7][8]. As noted earlier, the key aspects of MOESI (or any other cache coherency protocol) can be captured in a state diagram, as shown in Figure 4. Exercising the transitions in this diagram, and the transitions when multiple state machines are interacting, is one of the challenges of coherency verification.

Given all the transitions that need to be exercised, one of the biggest issues in cache coherency verification is non-determinism. It is impossible to predict the exact cache and memory transitions that will occur, especially under heavy system load with multiple agents accessing multiple memories through multi-level caches. The SoC team must verify all of the following variations and many crosses of the different variables:

- Single processor coherency state transitions
- Multiprocessor coherency state transitions
- Crossing cache line boundaries
- Cache line evictions
- Page table properties
- Physical memory types
- Processor instructions and data widths
- Single versus block operations
- Memory ordering tests
- Ordered versus unordered tests

Even if the SoC verification team has enough detailed knowledge of caches to tackle this list, the mechanics involved in writing the tests are staggering. For each individual test, a different but related program, most likely in

C, must be hand-written for each processor. If the processors are multi-threaded, then these programs should contain multiple threads. If a cached GPU or DSP is present, code must be hand-written for them as well. If I/O controllers have caches, code also must be written and run on the CPUs to set up each interface and program it to access caches and memories. All tests must be self-checking, with the data from each completed read verified against the expected value. Because of the uncertainty in the timing of cache and memory transitions, interlocks must be built in to ensure that memory accesses happen in the specified order regardless of system load or memory timing variations.

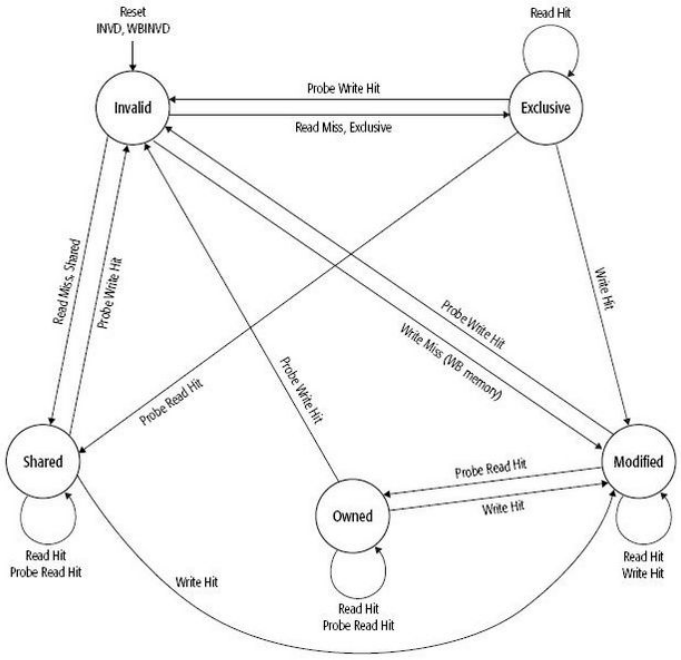


Figure 4. Example State Diagram for the MOESI Protocol

All these programs must be coordinated so that the targeted state transitions are verified. For example, tests should be written to ensure that a CPU attempts a read for data that is its own L1 cache, the L1 cache of another CPU in the cluster, and the cache for another agent outside the cluster. The bottom line is that hand-writing a test case that spans multiple processors and other coherent agents with all required coordination and synchronization is extremely difficult, if not impossible. Humans are not good at thinking in parallel. One might hope that the widely adopted Accellera Universal Verification Methodology (UVM) standard would provide some help. Unfortunately, the UVM deals strictly with automating testbenches that can read and write a design's I/O ports. It provides no way to generate code to run on the SoC's processors, or any way to link hand-written processor tests with the testbench.

## V. AUTOMATED CACHE COHERENCY VERIFICATION

The only way to perform thorough system-level cache coherency verification is to fully automate the process of generating coordinated, self-checking test cases that run on all the processors and other agents (CPU, DSP) simultaneously [9]. This approach can provide the high level of stress on the design necessary to verify all of the aspects of cache coherency described in the previous section, with every processor, memory, and cache active. The key to making this work is abstracting the test case requirements into a graph-based scenario model that captures all

intended cache behavior, including MOESI or another underlying protocol. The process for generating and running the test cases in register transfer level (RTL) simulation is shown in Figure 5.

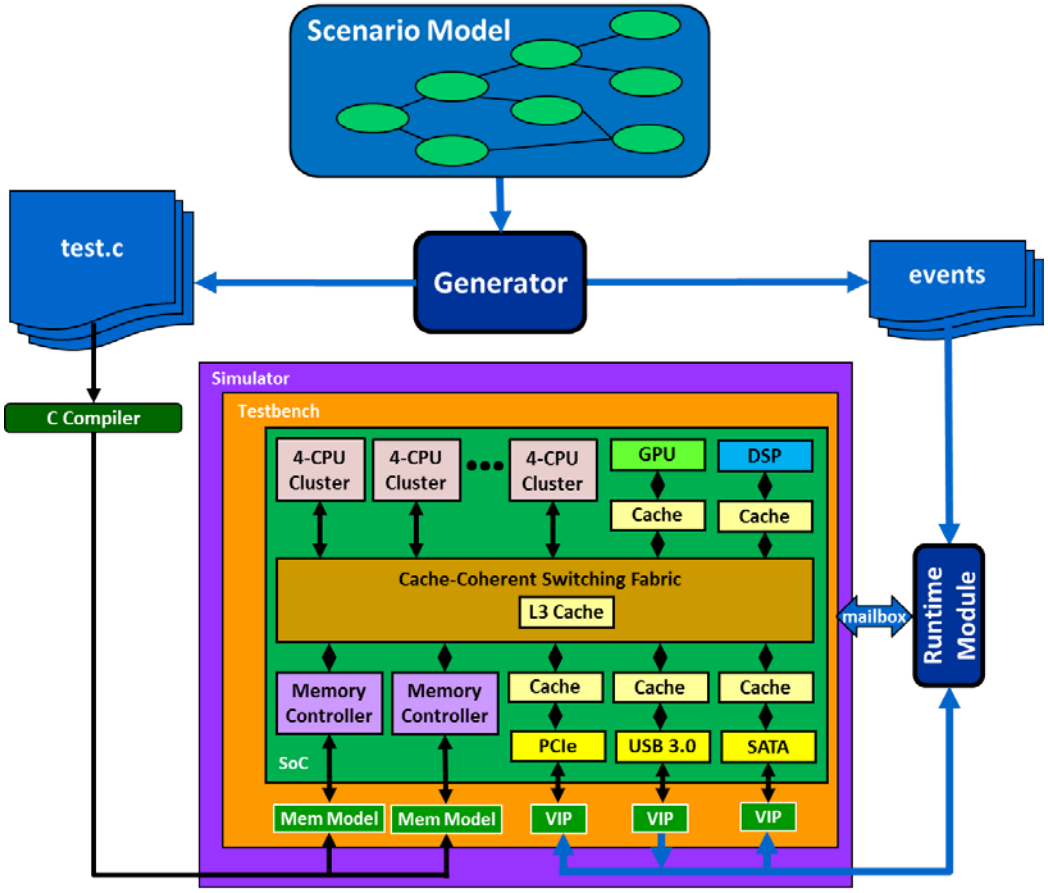


Figure 5. Running Automatically Generated Cache Coherency Test Cases

The test case generator shown in Figure 5 reads the scenario model to determine the verification space for the design. This is a general approach to functional verification of large designs, and the graph for the scenario model is typically a combination of hierarchical models for IP blocks and system-level information such as multi-IP flows captured by the SoC design or verification team. One of the specific advantages of using this approach for cache coherency verification is that a custom scenario model does not have to be developed. Designs using a standard protocol such MOESI can use an off-the-shelf scenario model, with the state diagram shown in Figure 4 already converted directly into a graph. Custom protocols may require some small modifications to the underlying graph.

Given an off-the-shelf scenario model, the user needs to specify only the number of CPUs and other agents, the organization of the memory, and cache details such as line size. From this information, the generator creates a unique C file for every agent and loads the programs into memory. When these programs run on the SoC's agents, they coordinate among themselves. If the processors support multiple threads, then the generated C programs are multi-threaded. The test cases generate a high amount of traffic that stresses all processors, memories, buses, and fabric. All read operations from memory locations check for the expected results so all test cases are self-checking.

In some cases, the verification team may also want to generate test cases that involve the SoC's I/O ports in order to verify I/O coherency. In addition to the C code, the generator can produce testbench transactions that interface directly to standard commercial UVM testbench verification IP (VIP) models. The generated test cases leverage and complement UVM verification activity. When the generated test cases execute in simulation, the programs running in the processors carefully coordinate with the activity in the testbench by means of a runtime module. This module takes care of system tasks such as logging messages, recording coverage metrics, and checking results. The runtime module also supports "back-door" access for the system memories if this feature is available in the testbench.

The processors control the overall test case and use a simple memory-mapped mailbox to send an ID to signal when system tasks must occur or when activity is needed on the VIP interfaces. The runtime module looks up the ID in an "events" file generated as part of the test case to determine the specific action to trigger. The resulting test cases are highly complex, far more than humans could ever write. As an example, Figure 6 shows a segment of a test case for an eight-processor SoC, displaying the high degree of interleave and synchronization among the threads on the multiple processors. This display can be updated in real time by the runtime module, helpful for the verification team to understand what the complex test case is doing, and to debug when design errors are detected.

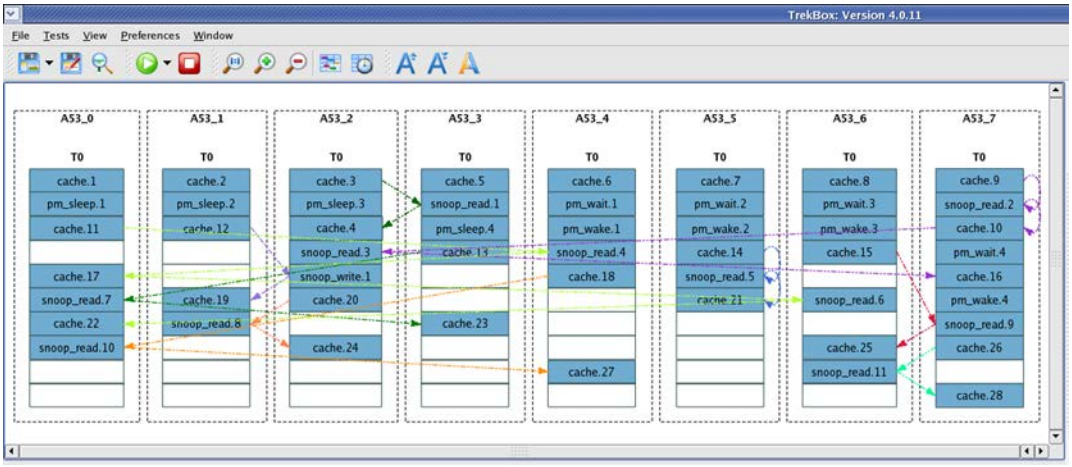


Figure 6. Segment of an Eight-Processor Cache Coherency Test Case

Graph-based scenario models are a powerful method for capturing verification intent, useful for all kinds of IP blocks and chips. They are especially effective for automatic generation of cache coherency test cases because the cache protocol state transitions can be directly and naturally modeled. Figure 7 shows an example of how this works. As previously noted, the MOESI state diagram has been translated to a pre-built graph that shows all possible ways to exercise the coherency protocol by reading and writing memory locations. The test case generator "walks" the graph multiple times, generating enough individual cache scenarios for every thread on every agent. It then schedules the memory operations, interleaves them across the agents and threads, packs them together as tightly as possible to get maximum system stress, and resolve any dependencies.

As stated earlier, the resulting test cases are far more complex and far more effective at verification than any hand-written test cases could ever be. Figure 8 shows an example of this, using an actual SoC design with eight ARM Cortex-A53 cores. The graphs represent a measure of system activity across the processors, memories, and interconnects of the design. The left-hand side shows the results for the original set of hand-written tests before

application of automatically generated cache coherency verification. Results for the generated test cases are on the right-hand side, and the difference between the two graphs clearly shows that much more of the system is being exercised than was the case with the hand-written tests. In fact, the automatically generated test cases produce such a high level of activity in the design that they are often used to gather realistic performance measurements long before production software is available to run on the SoC.

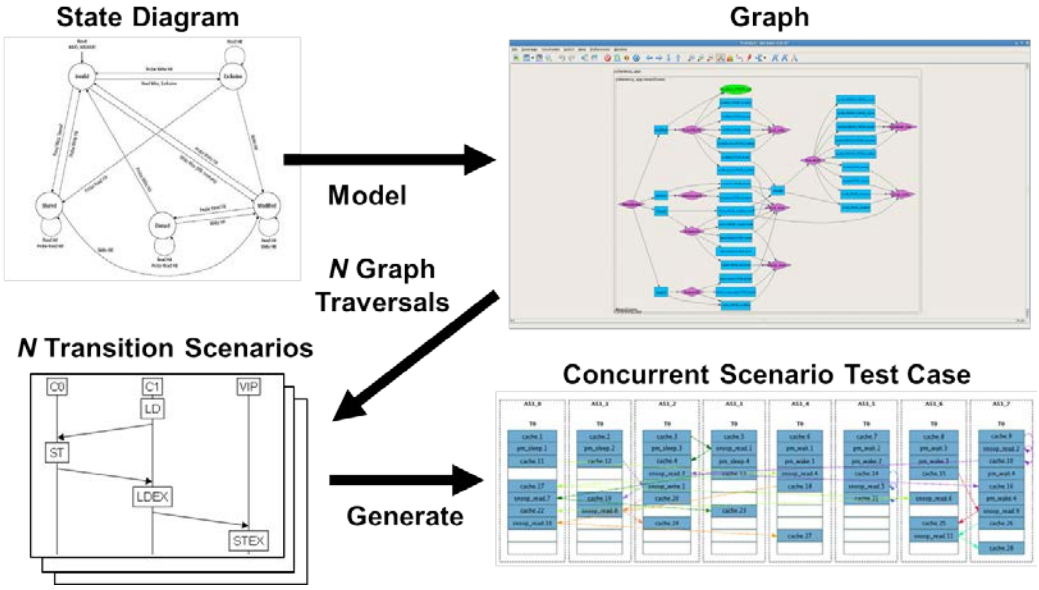


Figure 7. Generation of Multiprocessor Concurrent Cache Coherency Test Cases

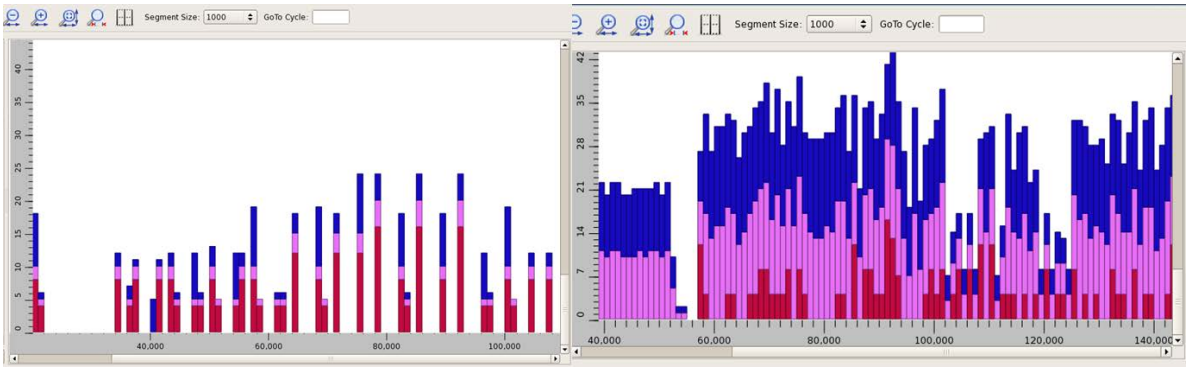


Figure 8. Comparison of Manual Tests and Automatically Generated Test Cases

## VI. TEST CASE PORTABILITY

Figure 5 shows the generated test cases running on an RTL model of the SoC in simulation. Separating the verification intent in the scenario model from the generation engine enables portability of the test cases beyond simulation [10]. This is increasingly important, since some large SoCs cannot be simulated with all processors running compiled code because the simulation image is too large and too slow. In such a case, the verification teams turn to simulation acceleration, in-circuit emulation, and field-programmable gate array (FPGA) prototypes as alternatives. In extreme cases, the entire SoC may not run until initial silicon is available in the bring-up lab.

Fortunately, as shown in Figure 9, the process of generating test cases for cache coherency verification works much the same for all platforms. The same scenario model is used by the same generator, except that the resulting test cases are tuned for the best possible performance in the target platform. For example, a simulator is most efficient running a series of short test cases, but a chip is best at running long iterative test cases since the execution speed is so fast relative to the time to load a new program.

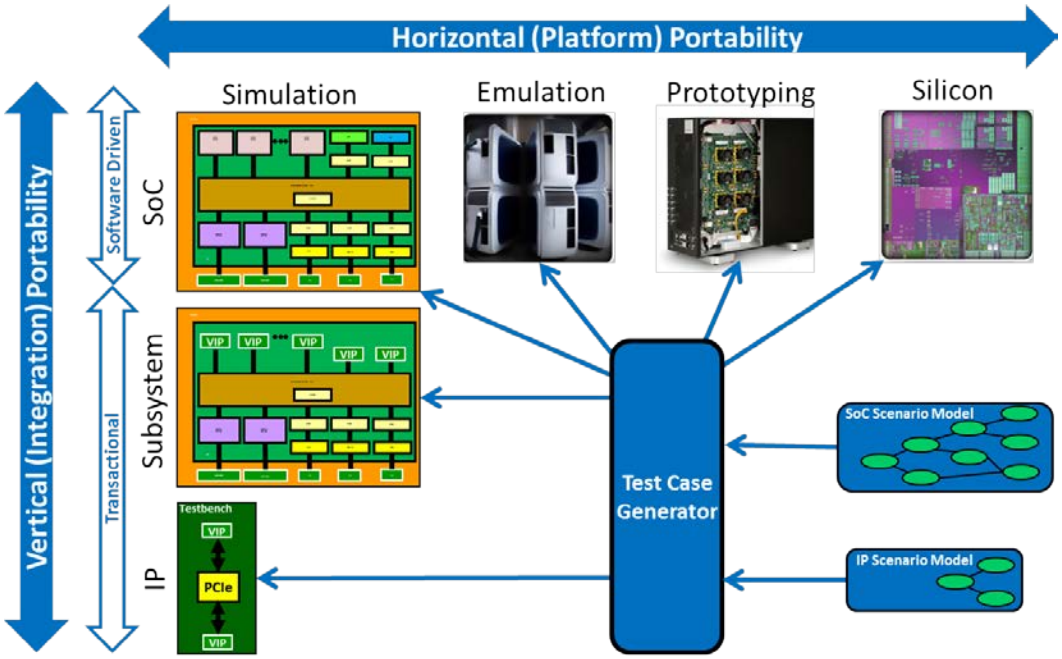


Figure 9. Vertical and Horizontal Portability of Test Cases

Two dimensions of portability are shown in Figure 9. The first is horizontal, moving from simulation to hardware platforms. The other is vertical, which means that scenario models used to verify individual IP blocks can be easily combined together to start building models for subsystems or entire chips. Further, the generator can produce C test cases than run on the SoC's processors and agents, or transaction-based tests that interact with VIP models for the CPU bus. If the full-chip model is too slow for simulation, a reasonable degree of coherency verification can still be performed by using bus models. Of course this will not be as complete as with all agents and caches in the system, but it does offer the opportunity for some testing prior to moving to a hardware platform.

One question that may arise is whether it is valuable to run coherency-specific test cases on a hardware platform if production software (operating system and applications) is available. In fact, it is not only valuable but essential. Production code is not designed to find hardware bugs and so diagnosis of failures is extremely difficult. Further, production software provides only modest exercise for the design, typically somewhere toward the middle of the two extremes shown in Figure 8.

It is worth noting that Figure 9 looks very much like the vision for Accellera's Portable Stimulus Working Group (PSWG), which is developing a standard to support vertical and horizontal reuse using graph-based models and automatic test case generation [11]. The term "portable stimulus" is now widely known in the industry, and used in the title of this paper, but it shortchanges the breadth of the described solution and the proposed standard. Any useful test case must include results checking and coverage metrics in addition to stimulus. Further, it is not the generated

stimulus or the test cases that are portable. The scenario model is reusable or portable in that it can generate test cases for multiple target platforms.

## VII. REAL-WORLD APPLICATION OF CACHE COHERENCY TEST CASES

The application of cache coherency test cases automatically generated from scenario models is relatively new in the industry, although the approach has been used on quite a few SoC projects already. One of those projects was the development of the ThunderX product family from Cavium [12][13]. As shown in Figure 10, each member of this family is a highly complex and powerful SoC providing leading-edge 64-bit ARMv8 data center and cloud process applications. The chips have up to 48 custom cores, fully compliant with ARMv8 architecture specifications as well as ARM's Server Base System Architecture (SBSA).

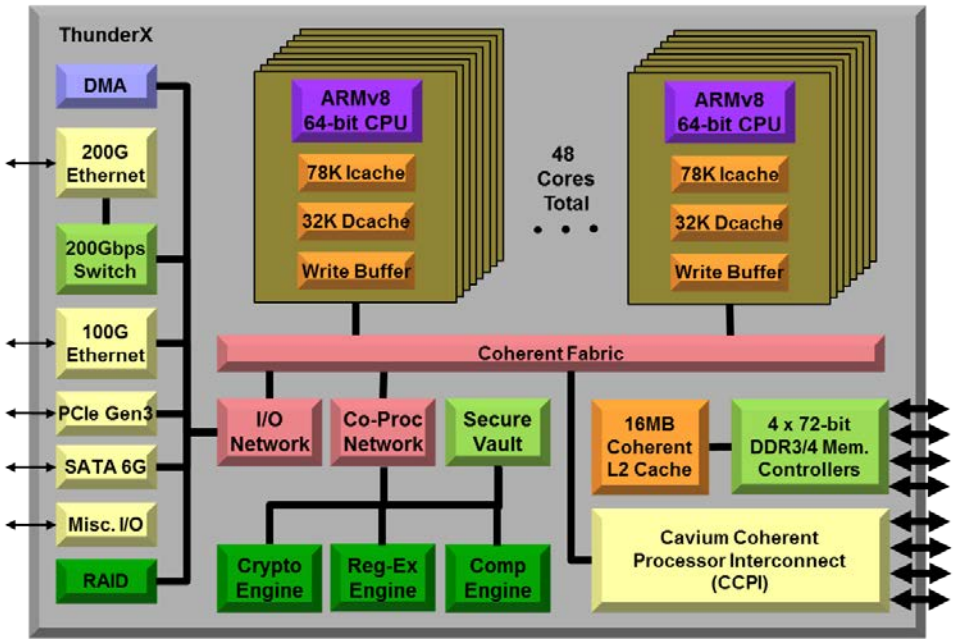


Figure 10. Block Diagram for the Cavium ThunderX Family

Some of the key features of this design include:

- The first ARM-based SoC that scales up to 48 cores with up to 2.5 GHz core frequency
- The first ARM-based SoC to be fully cache coherent across dual sockets
- The largest integrated I/O capacity with 100s of Gigabits of I/O bandwidth
- Four DDR3/4 72-bit memory controllers capable of supporting 2400 MHz memories with 1TB of memory

Verification of this enormous design was a major challenge for the Cavium team. Given the speed limitations of simulation and the capacity limits in hardware platforms, they were unable to ever run all 48 cores together during the pre-silicon development process. Thus, when they received the first chip samples from the foundry they developed a list of tasks that they needed to perform on silicon in the bring-up lab. These included:

- Running code on all 48 cores at the same time
- Verifying cache coherency across all 48 cores and all memory channels
- Adding IP-focused system randomization tests to the cache coherency runs

- Extending cache coherency verification to multi-SoC configurations

Because of the difficulty of hand-writing concurrent system tests, especially across 48 processor cores, the Cavium team decided to use automatically generated test cases as a way to accomplish these verification tasks with a minimum of manual effort. They selected the Trek family of products from Breker Verification Systems as their solution [14]. They were able to use the off-the-shelf C++ scenario model in the Cache Coherency TrekApp to generate test cases tuned for silicon execution. The tools generated 48 C files, one for each ARM processor, which the team compiled and downloaded into the chip using the same host system and process used to load hand-written tests and production software. The chips ran the test cases at full speed in the bring-up lab, reporting results and coverage via Ethernet links to a version of the TrekBox runtime module executing on the host system.

After some short test cases to ensure that the flow worked, the team generated many different scenarios to verify cache coherency with all the variations allowed by the protocol, L2 cache controls, and main memory configurations. They then added some additional system randomization test cases designed to focus on particular IP blocks in the design. Test cases involving I/O ports used loopbacks to fully exercise the interconnect IP blocks.

As mentioned earlier, one unique feature of the ThunderX family is the ability to gang two SoCs together in a 96-core dual-socket configuration that remains fully coherent. This is made possible by the Cavium Coherent Processor Interconnect (CCPI), a novel high-speed interface that connects the two chips. The validation team used the test case generator to produce 96 C files, all interconnected, with cache coherency scenarios interleaved across all the processors. This provided a level of design exercise and coverage unachievable by any other means and was a crucial part of the final validation of the ThunderX product line prior to shipment.

Most recently, the Cavium team extended the verification to an even larger lab setup in which the 96-core dual-socket configuration is connected to an independent 48-core Octeon III CN7800 SoC over a PCI Express (PCIe) link [15]. As shown in Figure 11, this configuration entails automatically generating and downloads 144 interacting C files. They verify full cache coherency across all 96 cores in the two ThunderX chips, while concurrently running PCIe traffic generation using all 48 cores in the CN7800 chip. This shows that test cases can be generated automatically from scenario models for complete systems consisting of multiple SoCs. Since the ThunderX family contains ARM cores and the CN7800 includes MIPS cores, this configuration also demonstrates that tests can be generated seamlessly for systems with heterogeneous processors.

The Cavium project and the experience of numerous other companies have shown the value of graph-based scenario models and automatic test case generation [16]. This approach is being deployed more widely every day in the industry, and when the Accellera standard is released it is likely that adoption will accelerate. When it comes to cache coherency verification, long regarded as one of the toughest challenges in chip development, the “portable stimulus” method offers unparalleled efficiency, efficacy, and reuse across all verification and validation platforms. It should be part of the toolkit for every SoC engineer.

## VIII. RESULTS AND CONCLUSIONS

The Cavium team found issues at multiple levels during their validation of the ThunderX silicon. Some of these were related to the setup in the bring-up lab and the process for downloaded and executing C programs, others in

their driver-level software. The displays provided by the runtime module, especially the thread view shown in Figure 6, gave the team insight into what was going wrong when tests failed and made bug diagnosis less painful than with production software. The information pointed directly to the point when a particular scenario failed on a particular core, giving them the information they needed to efficiently use the ARM DS-5 debug environment.

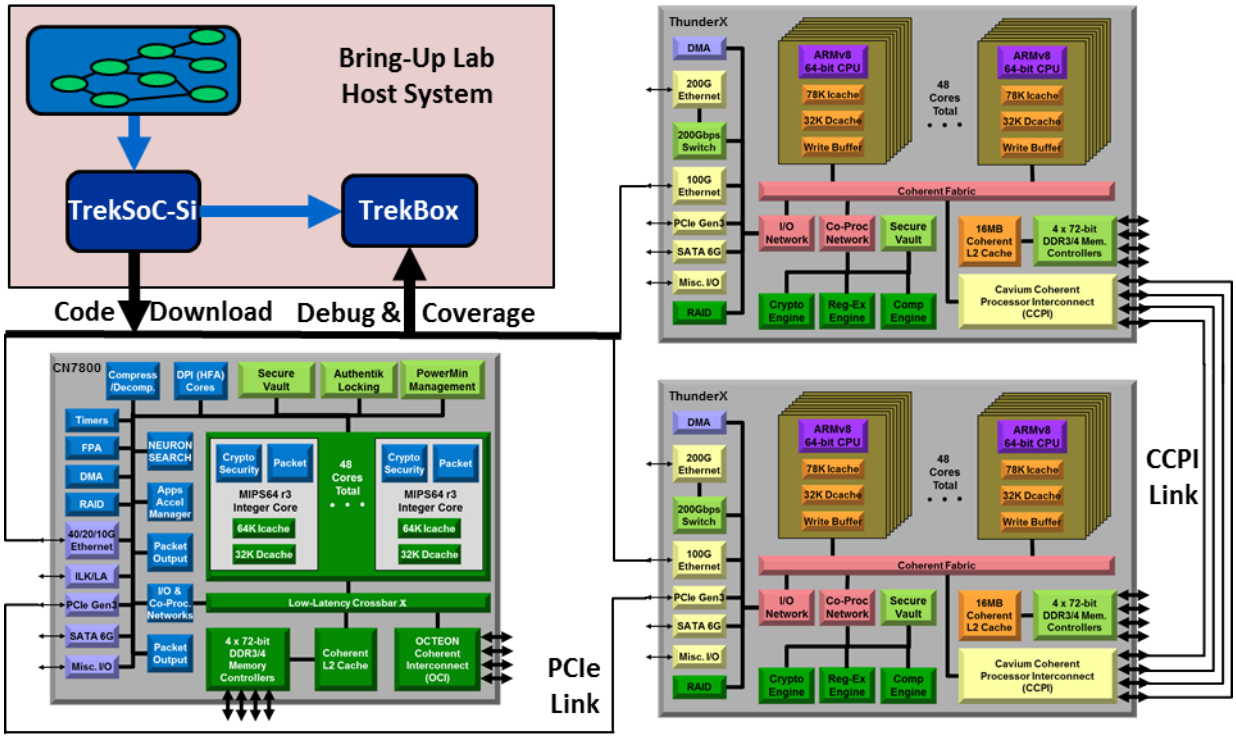


Figure 11. Multi-SoC Validation Configuration for ThunderX

## REFERENCES


[1] D. Patterson and J. Hennessy, *Computer Organization and Design (4th ed.)*, Morgan Kaufmann, ISBN 978-0-12-374493-7, 2009.

[2] "Hot Chips: A Symposium on High Performance Chips," http://www.hotchips.org/archives/2010s/hc27/, 2015.

[3] "Linley Processor Conference," http://www.linleygroup.com/events/event.php?num=39, 2016.

[4] R. Ranjan, "Verifying Cache Coherence," *Electronic Engineering Times*, http://www.eetimes.com/document.asp?doc_id=1279017, August 17, 2011.

[5] North Carolina State University, "CSC/ECE 506 Spring 2011/ch8 cl," http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_506_Spring_2011/ch8_cl, 2012.

[6] Wikipedia, "MOESI protocol," https://en.wikipedia.org/wiki/MOESI_protocol, 2016.

[7] S. Nakshatra, "Computer Architecture (EECC551): cache coherence protocols," Rochester Institute of Technology, http://meseec.ce.rit.edu/551-projects/fall2010/1-3.pdf, 2016.

[8] D. Sorin, M. Hill and D. Wood, *A Primer on Memory Consistency and Cache Coherence*, Morgan & Claypool Publishers, ISBN 978-1-60-845564-5, 2011.

[9] A. Hamid and B. Neifert, "Fast, thorough verification of multiprocessor SoC cache coherency," *Electronic Engineering Times*, http://www.eetimes.com/author.asp?section_id=36&doc_id=1324855&, December 3, 2014.

[10] A. Hamid, "Cache coherency verification with portable stimulus," *DVClub Europe*, https://www.youtube.com/watch?v=ftl7eIdB5iw&feature=youtu.be, April 2015.

[11] Accellera Systems Initiative, "Portable Stimulus Specification Working Group," http://www.accellera.org/activities/working-groups/portable-stimulus, 2016.

[12] Cavium, Inc., "ThunderX™ ARM processors," http://www.cavium.com/ThunderX_ARM_Processors.html, 2016.

[13] L. Gwenapp, "ThunderX rattles server market," http://linleygroup.com/mpr/article.php?id=11223, 2014.

[14] Breker Verification Systems, Inc., "Portable stimulus solution," DVClub Europe, https://youtu.be/ftl7eIdB5iw, April 2015. http://www.brekersystems.com/wp-content/uploads/2015/06/Trek-Family-of-Products-DAC-Final.pdf, 2016.

[15] Cavium, Inc., "OCTEON III CN78XX multi-core MIPS64 processors", http://www.cavium.com/OCTEON-III_CN78XX.html, 2016.

[16] P. Wobil, "Verification of a cache coherent system with an A53 cluster using ACE VIP with graph based stimulus," *16th Annual Workshop on Microprocessor and SOC Test and Verification*, December 2015.