BREKER™

TrekApp OVERVIEW

Pushbutton Verification of Cache Coherency

Introduction

There is a clear trend in the semiconductor industry of the most complex designs moving to system-on-chip (SoC) architectures with multiple embedded processors, multi-level caches, and cache-coherent interconnects. Many of these chips, especially in server and networking applications, contain dozens or even hundreds of processors linked by a common cache-coherent multi-level on-chip bus, chip fabric, or a network-on-chip (NoC) interconnect. This type of design introduces new verification challenges. The Breker Cache Coherency TrekApp is designed specifically to address these challenges in a pushbutton process requiring minimal user input.

Industry Trend

As shown in Figure 1, there are multiple categories of designs moving to SoCs with many processors and multi-level caches. 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
- Multiprocessor designs adding caches to reduce memory access time and bypass memory bottlenecks
- Multiprocessor designs with caches requiring coherency to maintain data ordering

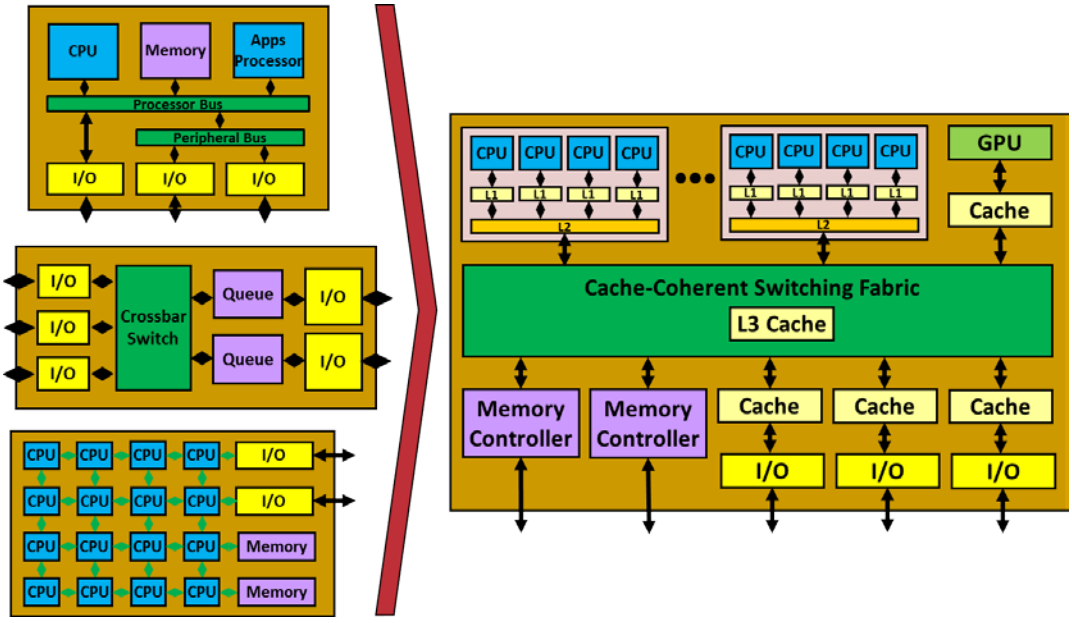

Figure 1: Industry Transition to Cache-Coherent Multiprocessor SoCs

SoC Verification Challenges

One effect of this SoC architecture is that the switching fabric typically allows a number of simultaneous transactions to be in progress at one time. Also, with a shared memory space, all processors may access all memories and all memory regions. These attributes make every aspect of SoC verification more difficult, since code must be run on the embedded processors in order to effectively exercise all the data paths inside the chip.

Most chip testbenches conform to the Accellera Universal Verification Methodology (UVM) standard. For non-SoC designs, the UVM may be able to generate constrained-random stimulus to exercise all key parts of the chip. In SoC designs, the embedded processors must be leveraged for effective verification.

Unfortunately, the UVM does not prescribe any way to include processors. The verification team must find some way to write and run embedded code while coordinating with activity in the testbench. This is difficult, if sometimes feasible, for a single processor. But humans are not good at thinking in parallel. In any SoC with multiple processors, the verification team simply cannot hand-write multi-threaded, multiprocessor tests that are scheduled and synchronized with each other. Manual effort alone is not enough to verify such designs.

Cache Verification Challenges

SoCs containing caches are even more difficult to verify using manual tests. In fact, cache coherency is widely known to be one of the most challenging tasks in all of chip verification. With the addition of multi-level caches there is a major shift in the need for cache coherency verification. Historically caches were found only within the clusters provided by the processor vendor, such as the four-CPU clusters in Figure 1. In this case, coherency was regarded as the CPU designer's problem. But the full SoC in Figure 1 contains multiple clusters interconnected by a fabric that also must be cache-coherent. Further, non-CPU elements such as the GPU may also have caches to speed their access to memory. Thus, cache coherency is now the SoC team's problem.

Cache coherency has several aspects that make it difficult to verify. Different caches in different clusters and at different levels (L1, L2, and 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. Proper updates of all caches are essential to preserve data ordering. For example, two CPUs may pass data as follows:

1. CPU0 writes data A to a memory location
2. CPU0 writes flag B to a different memory location
3. CPU1 monitors flag B, and sees it set
4. CPU1 reads data A and resets flag B

In this scenario, CPU1 must get the updated value of data A as written by CPU0, not a stale copy of an older value from its cache or from a higher-level (L2 or L3) cache.

The third major issue in cache coherency verification is non-determinism. Fully verifying all the individual cache policies is very hard since it is impossible to predict the exact transitions that will occur, especially under heavy system load. Thus, multiple processors with multi-level caches are impossible to verify with hand-written tests. The better approach is a method for automatically generating self-checking, multi-

threaded, multiprocessor C test cases that exercise all the processor-to-memory paths and stress all the corner cases for data moving in and out of the caches. The Cache Coherency TrekApp harnesses the power of Breker's Trek engines in a self-contained pushbutton application.

The TrekApp Solution

As with the other Breker products, the Cache Coherency TrekApp automatically generates test cases from graph-based scenario models that represent the chip's behavior and its intended verification space. The test case generator creates a unique C file for every embedded processor and loads the programs into memory.

When these programs run on the SoC's processors, they coordinate among themselves. All read operations from memory check for the expected results. The test cases generate a high amount of traffic that stresses all processors, caches, memories, buses, and fabric.

As shown in Figure 2, this level of verification can be performed with a pre-built scenario model that ships with the TrekApp; the user need only specify the processor and memory configuration and then the test cases can be generated automatically. No knowledge of graphs or scenario models is needed for cache coherency verification, and in fact the user does not even have to be an expert in caches. If the verification team wants to generate test cases that involve the SoC's I/O ports, they can provide scenario models for their I/O blocks and move seamlessly from the Cache Coherency TrekApp to TrekSoC.

When the generated test cases execute in simulation, the programs running in the processors carefully coordinate with each other and with the TrekBox runtime module. The processors control the overall test case and use an "events" file to trigger status and debug messages and debug via TrekBox in the testbench

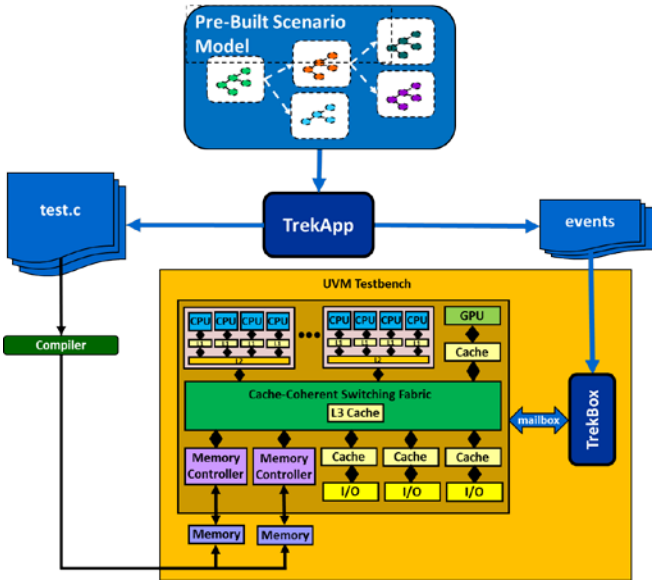

Figure 2: TrekApp Test Case Generation Flow

The generated test cases are highly complex, far more than humans could ever write. Figure 3 is a screenshot from the runtime TrekBox display shows an actual design with eight processors running in parallel. The TrekApp-generated test cases include many different varieties of memory reads and write as well as cache snoops across the multi-level cache architecture. The display shows the high degree of interleave and synchronization among the threads on the multiple processors.

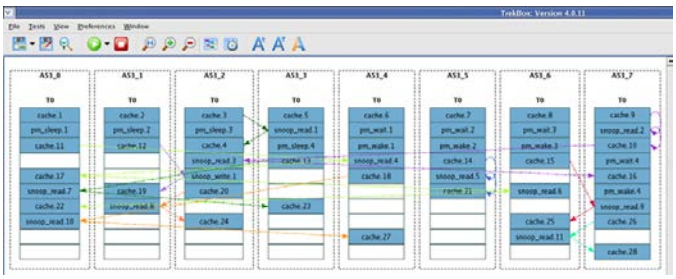

Figure 3: Multi-Threaded Cache Test Cases

This thread display helps the verification team to understand what the complex test case is doing, and to debug when design errors are detected. The display can be updated in real time by TrekBox not only in simulation, but also when the test cases are running on a hardware platform such as an emulator, an FPGA prototype, or the actual SoC silicon in the lab. This provides great insight into the test cases running deep inside the silicon. These test cases are ideal for validation of the silicon when it first arrives from the foundry.

Stressing Cache Coherency

The Cache Coherency TrekApp can generate sophisticated processor and memory workload test cases with no configuration. These perform dense, multi-threaded memory access and, just by themselves, go a long way toward cache verification. However, they are not sufficient to verify all aspects of coherency. More effective verification is possible when the user provides the generator with parameters for cache organization and cache lines widths

The generated test cases can verify many specific challenges such as:

- Exercising single-processor and multi-processor cache state transitions
- Crossing cache line boundaries with misaligned multi-byte operations
- Forcing multi-level (to L2, L3, and main memory) cache line evictions
- Creating different page tables with various security and coherency rules
- Accessing all memory types to stress different timing characteristics
- Exercising processor instructions for different sizes of data and bursts

The graphs in Figure 4 show the tremendous increase in cache verification provided by the test cases from the Cache Coherency TrekApp. The left-hand side shows the results from a set of hand-written tests for an actual design. Three different coherency-related metrics are displayed; clearly the design is not being well stressed. The right-hand side shows the results from a group of TrekApp test cases. Much more of the verification space is being covered, with more exercise of the SoC design throughout that space.

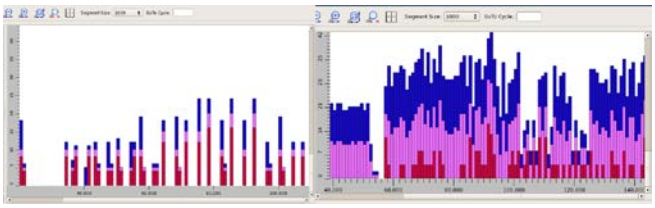

Figure 4: Benefits of Using TrekApp

Summary

The industry trend of large chips becoming SoCs with many embedded processors and multiple levels of caches puts a big strain on verification teams. Neither traditional UVM-based constrained-random testbenches nor hand-written C tests suffice to verify such designs. In particular, cache coherency verification is now everyone's problem, not just the CPU designer's. Test cases automatically generated from the Cache Coherency TrekApp solve this problem. They use a built-in graph-based scenario model hidden from the user to allow pushbutton operation. These test cases can be run on any verification platform, from simulation and acceleration to emulation and FPGA prototyping, and on actual silicon in the lab. They provide a portable test case solution for even the most complex SoC designs.