BREKER™

THE LEADER IN PORTABLE STIMULUS

TrekSoC

Verifying Many-Core SoC Designs

White Paper

Contents

Summary

Industry trend of building many-core SoC designs has led to unique verification needs, solved by TrekSoC:

- SoC's embedded processors leveraged for effective verification of multiple IP blocks strung together
- Effectively stress all processors, memories, buses, and fabric
- Automatic generation of test cases that involve the SoC's I/O ports
- Automatic generation of cache coherency test cases needed for SoCs with multi-level caches
- Automatic generation of highly complex system-level test cases not easily recreated manually
- Real time visualization of test cases as they run, enabling DV engineers to quickly understand and debug errors

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. 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.

Industry Trend

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

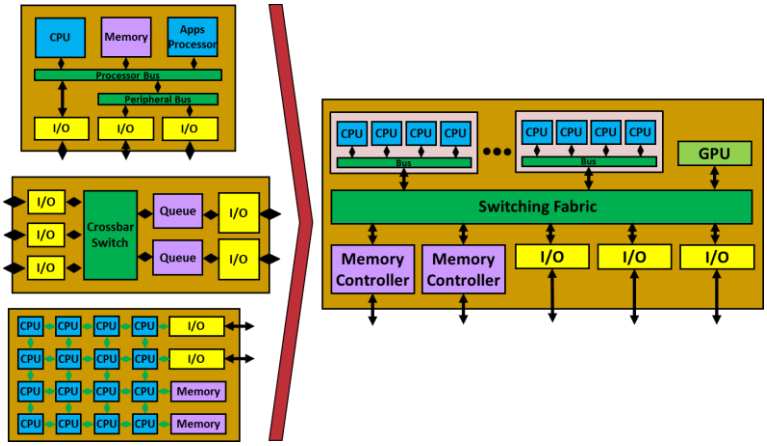

Figure 1: Industry Transition to Many-Core SoCs

- 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

One effect of this SoC architecture is that the switching fabric typically allows a number of simultaneous transactions to be in operation at one time. Also, with a shared memory space, all processors may access all memories and all memory regions.

Verification Challenges

Most complex chips are verified bottom-up, with initial focus on thorough verification of custom blocks in stand-alone testbenches. As the SoC is assembled, focus shifts to ensuring that custom blocks and any commercial IP blocks are integrated properly.

Traditional verification entails connecting all inputs and outputs of a chip to a simulation testbench. Most often, the testbench and its verification IP (VIP) components will 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. This becomes more difficult for SoC designs where multiple IP blocks must be strung together to create realistic use cases. The SoC's 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. In a many-core chip with dozens or hundreds of processors, the situation is even worse. Manual effort alone is not enough to verify such designs.

Automatic Test Cases

The better approach is a method for automatically generating self-checking, multi-threaded, multiprocessor test cases that both exercise all the processor-to-memory paths and string IP blocks together into realistic user scenarios. As shown in Figure 2, these test cases can be generated 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, memories, buses, and fabric. This level of verification can be performed with a pre-built scenario model; 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.

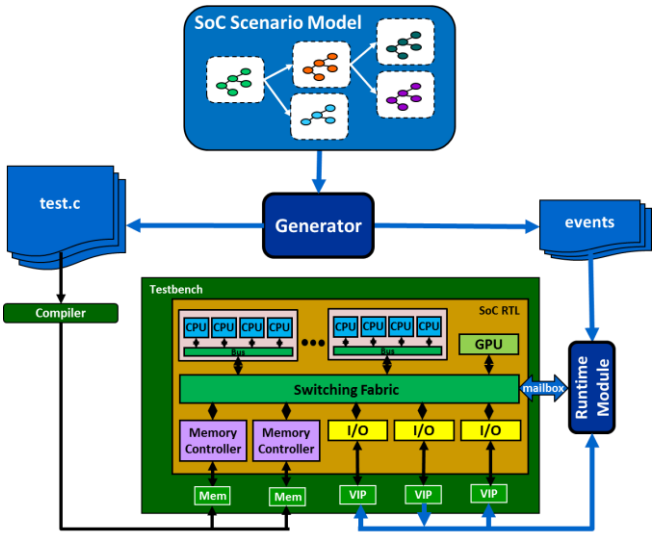

Figure 2: Test Case Generation Flow

In some cases, the verification team will also want to generate test cases that involve the SoC's I/O ports. They can start by using a VIP model for the fabric as a stand-in for the I/O blocks. If commercial IP blocks are used, the vendor may provide detailed scenario models that show how to program the I/O interfaces to send data on and off chip. If these models are not available from the vendors, and for any custom blocks, the verification team can easily create the scenario models. These look much like data-flow diagrams through an IP or SoC and are natural for both design and verification engineers. If an existing C "bare metal" driver is available for a block, only a minimal scenario model is required.

When the generated test cases execute in simulation, the programs running in the processors carefully coordinate with the activity in the testbench. The existing testbench VIP models (most likely UVM-compliant) are used to send data into the chip as well as to receive data from the chip and check for expected results. The processors control the overall test case and use an "events" file to trigger desired actions (reads, writes, and status messages) in the testbench.

The test cases are highly complex, far more than humans could ever write. Figure 3 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 same runtime module that provides testbench coordination. These thread displays are helpful for the verification team to understand what the complex test case is doing, and to de bug when design errors are detected.

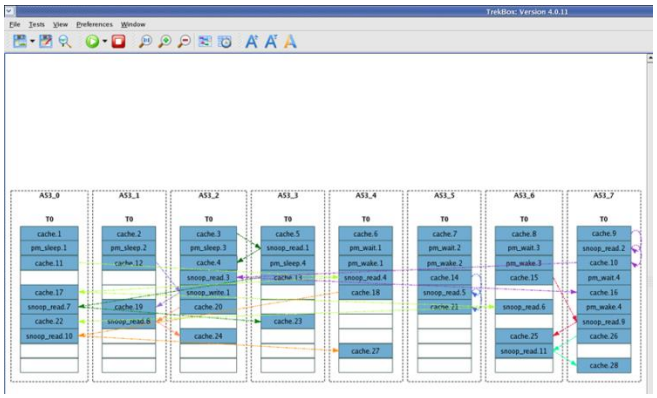

Figure 3: Multi-Processor Test Case Threads

Adding Caches to the SoC

The diagram in Figure 1 represents a specific type of SoC in which all the processors, plus a GPU and I/O interfaces, share a common memory space without any caches. In addition to the four factors cited earlier driving the transition to many-core designs, two more are often in play:

- Multiprocessor designs adding caches to reduce memory access time and bypass memory bottlenecks
- Multiprocessor designs with caches requiring coherency in order to ensure that the right data is always accessed

Caches store local copies of memory contents so that they can be accessed more quickly. This creates multiple versions of the same memory location, the source of increased complexity in both the SoC design itself and in the verification of the design. When a location is updated in main memory, any cached copies must be updated as well, or flagged as outdated. If the copy of memory in a cache is updated, then main memory must also be updated and any other cached copies must be updated or invalidated.

With the addition of multi-level caches, a typical many-core design looks much like the chip shown in Figure 4. This architecture represents 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 4. In this case, coherency was regarded as the CPU designer's problem. But the SoC in Figure 4 contains multiple clusters interconnected by a fabric that also must be cache-coherent. Further, non-CPU elements such as the GPU and the I/O interfaces may also have caches to speed their access to memory. Thus, cache coherency is now the SoC team's problem. The team must be able to run tests that thoroughly verify the entire SoC, stressing the caches as well as the processors, memories, and I/O interfaces.

Cache Coherency Test Cases

As noted earlier, humans are not very good at thinking or programming in parallel. Since the verification team cannot hand-write effective multi-threaded, multiprocessor programs there is no hope for writing cache coherency tests. Fortunately, the same graph-based scenario models referenced earlier can be used to automatically generate high-quality test cases that will stress the caches and the coherency logic that keeps them synchronized.

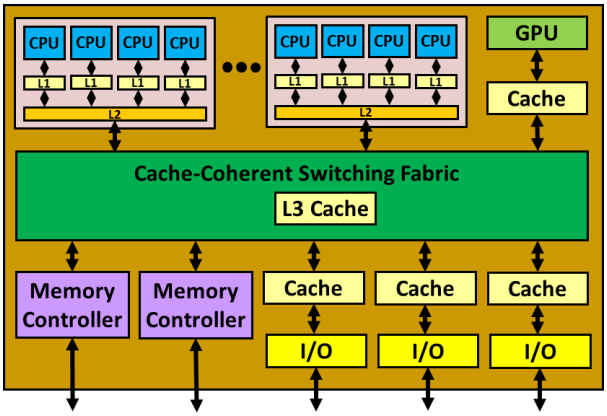

Figure 4: SoC with Multi-Level Caches

For a start, the processor and memory test cases described earlier perform dense, multi-threaded memory access. Just by themselves, these go a long way toward cache verification. However, they are not sufficient to verify all aspects of coherency. By providing the generator with additional information on cache organization and cache lines widths, the generated test cases can verify many specific challenges such as:

- Different cache line widths
- Different address maps
- Page tables with differing coherency rules
- Multiple memory types
- Instructions with multiple data sizes

As before, no knowledge of graphs or scenario models is required.

The effectiveness of this approach is shown in Figure 5. The left-hand graph shows the results from a set of hand-written tests for an actual multiprocessor design. Three different coherency-related metrics are displayed; clearly the design is not being well stressed. The right-hand side shows the results from test cases automatically generated from scenario models. Much more of the verification space is being covered, with more exercise of the design throughout that space.

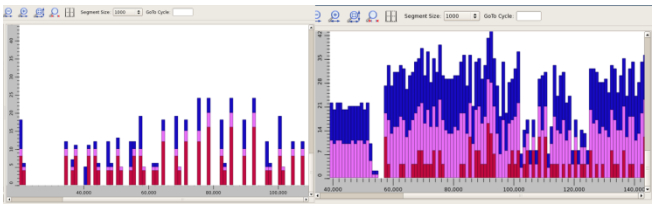

Figure 5: Better Cache Coherency Verification

Summary

The industry trend of large chips becoming SoCs with many embedded processors, and often 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. Test cases automatically generated from graph-based scenario models solve this problem. IP-level graphs can be directly instantiated in higher-level scenario models for reusability. The generated 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 solution for even the most complex SoC designs across the entire duration of the project.

Challenge

- Many-Core SoCs require the SoC's processors to be leveraged – a challenge not addressable with UVM
- Hand-writing multi-threaded, multiprocessor tests that are scheduled and synchronized with each other, is infeasible
- Limited coverage achieved with existing methodologies

Solution

- Automatically generate self-checking, multi-threaded, multiprocessor test cases that both exercise all the processor-to-memory paths and string IP blocks together into realistic user scenarios
- Port, reuse and run generated content across multiple platforms, from simulation and acceleration to emulation and FPGA prototyping, and on actual silicon in the lab

Why TrekSoC?

- Industry leader in automated system-level test case generation
- TrekSoC takes as input easily created graph-based scenario models capturing chip's behaviors and its intended verification space, and gives as output a unique C file for every embedded processor
- TrekSoC verifies SoCs "from the inside out" by generating software driven tests in C that control the testbench via the TrekBox
- TrekSoC automates many aspects of SoC verification such as memory management, memory initialization, mutli-core scheduling, I/O management

Results

- Portability of stimulus, checks, coverage, debug and tests across multiple platforms
- Greater coverage at least 10x faster
- Faster debug due to annotated visualization of runtime tests