BREKER™

TrekUVM OVERVIEW

Enhancing UVM Testbenches with Graph-Based Scenario Models

Introduction

For the past 15 years, the semiconductor industry has relied on constrained-random testbenches as the primary method for verifying its largest and most complex designs. This technique was a huge leap over manually-written directed tests, automating the verification process and making it possible to generate an arbitrary number of tests from an initial investment in testbench elements and constraints on the chip inputs. In fact, the constrained-random approach became so popular that it was standardized by Accellera as the Universal Verification Methodology (UVM).

For all its benefits, many chip design teams are finding that the UVM has severe limitations when it comes to verifying large, complex chips with significant sequential depth between inputs and outputs. As is the nature of the methodology, exercising deep state or producing a desired output requires the UVM-generated stimulus to propagate through a chain of IP blocks that might have very specific rules for what can pass. The verification team can use constraints and biasing to try to "steer" the stimulus along a particular path, but fundamentally the UVM sequencers are "pushing on a rope" and trying to maneuver that rope around corners and through tight spaces. Having control only at the pushing end is highly likely to result in inadequate verification.

Supplementing UVM testbenches with graph-based scenario models that describe the chip's verification intent provides a major enhancement to the verification process. Because graphs are automatically analyzed from results to inputs, TrekUVM can generate additional UVM sequences that predictably exercise desired behavior and achieve system-level coverage goals. TrekUVM generates stimulus, results, and coverage metrics directly from the scenario models. This exercises the design more thoroughly than a UVM testbench alone, increasing the change of first-silicon success and reducing the effort needed for tape-out.

Target Designs

Many large chips contain one or more embedded processors, which immediately takes them outside the domain of the UVM. These system-on-chip (SoC) designs require a supplemental methodology that encompasses embedded code as well as testbenches. TrekUVM provides a solution for large, complex chips that do not contain embedded processors that must be programmed. The chip may contain micro-coded applications processors, but these run as part of the hardware and do not require any special treatment. Thus, TrekUVM is ideal for chips that fit within standard UVM testbenches but need enhancements to meet verification goals.

The general area of networking is the most common source of chip designs that fit this description. Many routers, switches, bridges, and modems are essentially bidirectional datapaths between two (possibly different) protocols. A great deal of complexity can exist along these paths: field packing/unpacking, data manipulation, address translation, queuing, arbitration, and more. These functions are precisely the reason that constrained-random stimulus may have trouble reaching all portions of the design. Figure 1 shows a small but representative example of a complex chip design with no embedded processors. It connects multiple PCI Express (PCIe) buses with multiple USB 3.0 buses, with complex switching and queuing in between.

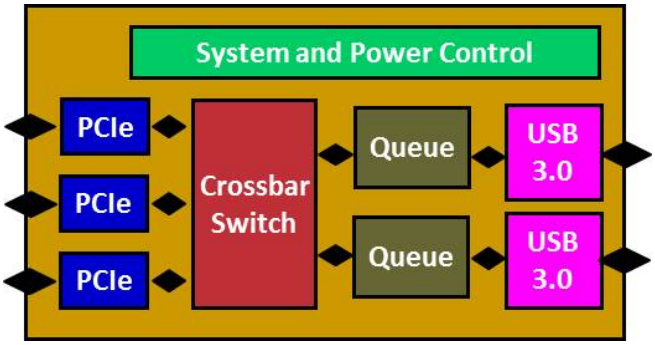

Figure 1: Example Chip Suitable for TrekUVM

The UVM testbench is built by connecting every chip I/O port to a UVM virtual component (UVC) that contains the sequencer to generate stimulus, some results checking, and possibly some coverage. As the IP blocks are composed together to build the complete chip, both the stimulus and the expected results must be coordinated across multiple blocks. These tasks fall to the virtual sequencer and the scoreboard, as shown in Figure 2. The scoreboard may also collect some chip-level coverage metrics beyond those for individual IP blocks or interfaces.

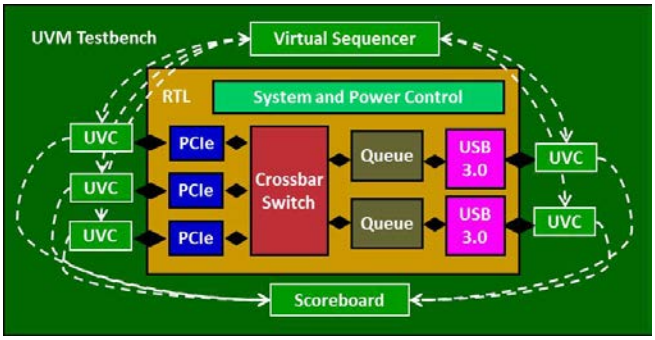

Figure 2: Traditional UVM Testbench

In this testbench, the burden is on the verification team to set up the input sequences with appropriate constraints to reach all parts of the design and to exercise all the operational cases. Even if there is enough time and expertise to achieve full functional coverage, which is highly unlikely, the verification team gets limited insight into the real extent of verification. Code coverage and user-specified functional coverage should be supplemented by system-level use case coverage. This form of coverage shows which of the possible paths through the design as shown in the block diagram have been exercised.

Graph-Based Scenario Models

Graphs have several advantages when used as the source for functional verification. They document the data flow through the design, define the verification space, and yield clear system-level coverage metrics. At the root of the graph concept is “beginning with the end in mind” by starting with possible outcomes and showing all possible paths through the chip to achieve those outcomes. This enables the TrekUVM test case generator to “pull on the rope” by starting with a particular result to be verified and tracing arbitrarily many ways to get there. The generator randomizes at all decision points in the graph in addition to randomization of data and address values.

Figure 3 shows a possible top-level scenario model for the chip shown in Figure 1. It represents the data flow from the PCIe side to the USB 3.0 side, thus it starts with the USB 3.0 outcomes on the left. Each time that the test case generator walks the graph, it selects an outcome and randomly chooses which PCIe port supplies the input data. The graph provides all the information that TrekUVM needs to provide the input UVM transaction (stimulus), output UVM transaction (result), and coverage. All generated test cases are self-checking to ensure that the results match expectations and that achieved coverage is legitimate (no errors).

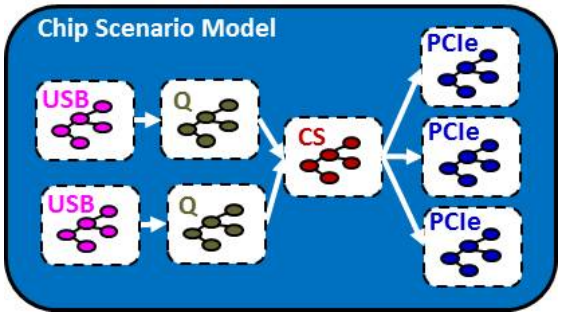

Figure 3: Scenario Model for Example Chip

One unique feature of scenario models is automatic coverage closure. Coverage is reported in terms of use cases relevant for the chip, such as cross-coverage of which PCIe ports have driven data to which USB ports. When portions of the graph have not yet been covered, the verification team can explicitly request TrekUVM to generate a test case to achieve that coverage. Because of the "begin with the end in mind" and "pulling on a rope" nature of graphs, the generated test case is guaranteed to hit the target. This is a major enhancements over constrained-random testbenches, whether UVM or otherwise, in which the verification team can only "push on the rope" and try to lead the stimulus generated in the right direction.

In addition, TrekUVM can statically analyze the graph and report portions of the scenario model that cannot be reached. This may be intentional, for example, blocking verification for parts of the design not yet coded. However, it can also indicate an error in the graph or a conflict in the constraints on the graph. Constraining and biasing allow the verification team to customize many aspects of test case generation. In the case of the chip in Figure 1, the team might want to have 75% of the test cases use the first PCIe bus rather than the second (3:1 bias).

TrekUVM Operation

The easiest way to think of TrekUVM is that it replaces both the virtual sequencer and the scoreboard in a UVM testbench. It is responsible for generating transactions and feeding them to the UVCs, collecting results from the UVCs, and gathering and reporting system coverage results. As shown in Figure 4, a runtime module called TrekBox ties together the UVCs during a simulation run. It is responsible for coordinating the application of stimulus and alignment of stimulus with results. It also provides a runtime display of how the test case is proceeding.

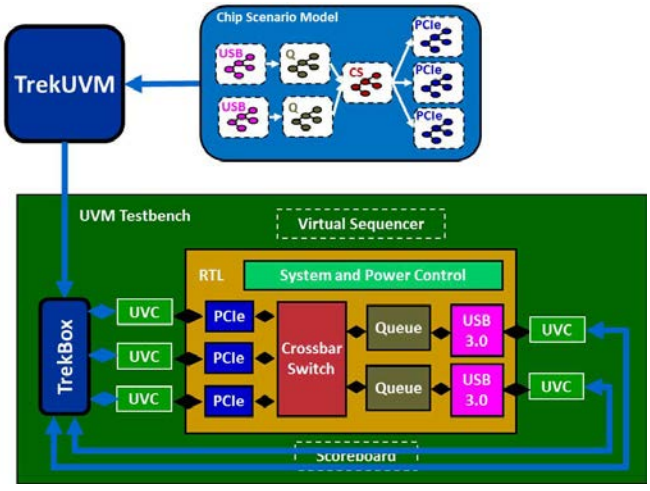

Figure 4: TrekUVM in Simulation

The test cases generated may be quite complicated. For example, in some chips the data on the outputs may appear in a different order than it was received on the chip inputs. TrekUVM can generate multi-threaded test cases, in which each thread corresponds to a different active path through the design. In the Figure 1 chip, the crossbar switch may support multiple simultaneous connections. Thus, it might be possible for a transaction to be active from the first PCIe port to the second USB port at the time that a transaction is flowing from the second PCIe port to the first USB port.

As another example, the queues might allow multiple serial transactions to be in progress but staged sequentially. In all these examples, TrekUVM will determine the chip's capabilities from the scenario model and generate the most aggressive test case possible. TrekBox will coordinate everything during runtime and will generate a multi-thread test case display. This unique form of visualization is essential for debug when a test fails due to a bug in the design or an error in the graph specification.

IP to Cluster to Chip Reuse

Another unique capability of graphs is that they can be composed together. As the scenario model in Figure 3 shows, the graphs from the individual IP blocks can be combined to form the graph for a multi-block cluster or a complete chip. This is a sharp contrast to the elements in a typical UVM testbench. The UVCs on the external ports can generally be reused, but only the monitor portions are relevant once interfaces are buried in the chip. Further, the virtual sequencer needs significant rework at every new level.

Some project teams start developing scenario models at the very beginning of the project, verifying individual IP blocks with UVM testbenches and then enhancing the results with TrekUVM. This gives a boost to cluster-level and chip-level verification since the lower-level graphs can be combined with no effort. Since most teams do not want to repeat the verification of every IP feature at higher levels, they can use graph constraints to suppress test cases that are considered unnecessary.

Conversely, a verification team may start using graphs only at the full-chip level. This does not mean that they need to develop a complete scenario model for every IP block. They typically specify only the subset of functionality needed to verify that the IP blocks interact properly in use case scenarios.

Summary

As chips grow every larger and more complex, verification teams are finding that it is hard to exercise deep behavior and achieve coverage goals with UVM testbenches alone. TrekUVM enhances UVM results by automatically generating test cases with complex transactions reflecting real use cases for the design. The input to TrekUVM is a graph-based scenario model that looks much like a chip dataflow diagram. There is no new language to learn; scenario models are specified using the C/C++ and Backus-Naur Form (BNF) standards. Scenario models are reusable from IP to chip level and from project to project. They are also fully compatible with all of Breker's products, including TrekSoC and TrekSoC-Si.

For a small investment, verification teams find more bugs and hit more coverage in less time.