INTRODUCTION
Here we present an architecture for verifying proper operation and performance of a complex AXI bus fabric in a dual-core ARM processor system using a combination of SystemVerilog and C software-driven test techniques.

Specifically, we describe deployment of an advanced graph-based solution that provides the capability for checking full protocol compliance, an engine for continuous traffic generation, precise control and configurability for shaping the form and type of traffic needed to test the fabric. These characteristics were easier to construct, easier to analyze and review, and were more efficient in terms of achieving coverage using a graph-based approach than constrained-random or directed OVM sequences. For us, this architecture yielded successful completion of the verification process ahead of schedule.

The system we verified consisted of a dual core ARM Cortex A9 processor connected to an AXI bus fabric. Master peripherals connect to the fabric using both AXI and AHB bus interfaces. Slave peripherals respond using AXI, AHB or APB ports. Although the fabric is predominately AXI, some older peripherals still use AHB which is bridged to AXI within the fabric. APB is used for most peripheral register access and data transport for the slow peripherals.

Verifying each master-to-slave connection on an AMBA fabric is a reasonably straightforward task, but verifying that each port complies with both the standard defined protocol and any user-defined conditions at every master and at every reachable slave increases the complexity of verification. Moreover, verifying that the fabric will continue to function and maintain acceptable performance under normal and heavily loaded traffic conditions introduces even more challenges. Normal fabric operations include bus transactions from multiple masters being sent to multiple slaves. Some masters may also have multiple transactions in flight. Defining and controlling transactions on the fabric from each of the master and slave ports in a real system using the particular protocol for each of those ports can be an intractable problem particularly when seeking a high level of synchronization and control of that traffic.

For example, it is not easy to send a packet to an ethernet peripheral block and then predict with some precision exactly which types of AXI transactions might result, much less predict on which clocks they will occur. There are several dependencies on the state of the block such as buffer conditions and packets already in flight, and this can be further obscured by design-specific implementation choices and register settings that are found in a third-party IP. Start by taking into account just this one peripheral example and its protocol, and now imagine multiplying this across all the protocols found in a system.

To solve this challenge, we replaced peripheral blocks with VIP models for the connected protocol, which gave us much more precise control and dependable operation. Then we leveraged this control to construct tests that mimic the normal flow of data from each peripheral. Normal and heavy traffic scenarios were modeled to match expected system operations. Verifying the functionality and performance under loaded traffic scenarios helped us determine that there were no conditions that could lead to stalls or deadlocks in the fabric, stalled masters, stalled slaves or issues with performance degradation beyond tolerable limits.

ENVIRONMENT
Our testbench environment was fully based on System Verilog and the OVM 2.1.1 library. The environment also included significant embedded C software running on the CPU model that performed chip-level initialization, driver operations and many test control and monitoring operations. A number of OVM-based Verification IP (VIP) components formed the foundation of the testbench. Coupling the embedded C software into the verification environment meant that the testbench was also tightly bound to the CPU. Therefore, the CPU could be used to coordinate and check testbench activity. This was facilitated by a custom OVM-based mailbox system with dynamic message passing. The VIP models received transactions directly from OVM sequences launched by an API in the embedded C software and/or OVM sequences launched by the testbench. A block diagram of the environment is shown in Figure 1 below.
STIMULUS DESCRIPTION AND COVERAGE MODEL

Constrained Random Testing or CRT has been proven to increase productivity and find bugs missed by directed tests. Nevertheless it has several limitations:

- Users are responsible for defining a reasonable set of random variables and constraints. The definition of the variables and constraints is spread across many files. This sprawling structure of data makes it difficult to create, visualize, analyze or refine any of these variable and constraints. In general, it’s hard to assert any level of precise control.
- Constraint solvers are proprietary and users are not assured of consistent results across simulation platforms.
- Discovery of interesting and important corner cases are randomly discovered and are subject to the odds of random convergence of multiple variables. Random coverage of the defined coverage space is not efficient. Some areas may be repeated many times before a new unexplored area is exposed.
- Successful CRT also requires development of coverage models to measure test effectiveness which can be extremely difficult.

To overcome these limitations, we chose a graph-based stimulus description and coverage modeling solution. Specifically we addressed the limitations of CRT in the following ways:

- Random variable and constraint definition were replaced with an efficient and compact grammar in a single file. This file was compiled into a graph that made it easy to visualize and analyze for correct
and complete definition. This comprehensive view of the functional space gave feedback on the parameters that were covered and those that were intentionally left out, as well as out-of-band features that may be selectively enabled and covered.

- Careful reviews of the graph also gave feedback on any features of a protocol that might have been missed. An example of the grammar is shown in Figure 2 and an example of the graph it produced is shown in Figure 3.

- This solution can be ported to any simulation platform ensuring consistent results without any dependency on the constraint solver of the simulator.

- Coverage checking can be built into the graph, and improved coverage closure efficiency by testing the complete scope of the protocol in a minimal number of simulation clock cycles. This approach ensured that the complete protocol space is covered, including corner cases, with high efficiency. Interesting and important cases are dependably covered efficiently without dependencies on random chance.

### Testing Individual Ports and Paths of the Fabric

Testing individual ports with specific protocols must cover all aspects of the protocol. Additionally, each master port must be tested to confirm that it can reach all accessible slave destinations. The slaves themselves may support a subset of a protocol or even be a different protocol altogether from the master. For example, an AXI master could initiate a 64-bit transaction to a 32-bit APB slave. It is the job of the fabric to split the original 64-bit transaction into 2x32-bit transactions and the testbench to track it. In this example the master and slave monitors report transactions based on the master ID and fabric ID using local scoreboards and analysis ports. A subsystem scoreboard subscribed to the local analysis ports for checking. Details of this checker are omitted for brevity.

We used a graph in the form of an OVM sequence compatible with the VIP. This works for most masters, and the graph can also be used to generate calls to the embedded C API, thus giving us the ability to use a consistent approach to test the AXI master port on the CPU connecting it to the fabric. The graph-based sequences can be used for protocol testing, path coverage, and also for generating high volumes of transactions. The graph-based sequences also had numerous parameters used to activate or deactivate supported features of each individual port instance.

In addition to protocol and path tests, the graph can also be used to generate endless streams of traffic that can be controlled by the graph itself with a local perspective matching traffic expected from the normal peripheral the VIP has temporarily replaced. Moreover, the local traffic controls and parameters in the graph can be extended to an external graph with a system perspective where there is awareness of the traffic conditions on all other ports. These controls gave us the ability to define, control and synchronize the traffic conditions across the fabric.
TRAFFIC SYNCHRONIZATION AND CONTROL

The same principles that guide the choice of a graph-based solution for bus protocols also apply to definition and control of traffic conditions. The local controls in each “protocol graph” that give the ability to shape traffic within that graph can be dynamically controlled by a “traffic graph”. Examples of these controls include graph parameters such as the number of idle clocks between transactions, the size of data, and number bursts in a transaction.

Graph parameters can be dynamically controlled to be a fixed number, a random range of numbers, or a weighted random range of the numbers. Additional controls are added for synchronization. For example, a protocol graph can be instructed to conduct a single transaction and stop until instructed to run the next transaction. It can be instructed to run specific numbers of transactions or run continuously until instructed to stop. This gives the traffic graph several different ways to control traffic.

Dynamically controllable graph parameters can be changed between transactions and even during a transaction at certain control points defined in the protocol graph. For example, before a transaction completes, it could check to see if there are any updates to the number of idle clocks between transactions prior to completing the transaction.

Most important, the traffic graph has the ability to simultaneously launch transactions on multiple ports that can be synchronized to start on the same clock. All of these features can be used to produce any number of worst case scenarios to thoroughly examine the capabilities of the Fabric. An example traffic graph is shown in Figure 4.
TRAFFIC MODULATION
Making effective use of the graph parameters described above gave us the ability to control and shape, or modulate, the traffic at each master port. For example, the density of transactions can be adjusted to match the traffic conditions found on a peripheral that only has occasional traffic.

Transactions can be queued and released on multiple ports simultaneously or staggered in a very controlled manner. The size and type of transactions can also be controlled to match expected system operations. For example, the bandwidth of a slow peripheral device will not generate the same amount of traffic as a high speed peripheral device. Some peripherals may have very dense transactions for brief periods of time and then go quiet for a while. Some may have constant high density transactions. The shape of the traffic can be influenced by buffer sizes in the peripherals, the layout and arbitration defined in the fabric, bandwidth limitations at popular slaves like DDR, clock and clock ratio settings and interactions between multiple masters and slaves. The traffic graph modulates traffic in a manner that matches normal system operations described above. The protocol graph that is used to interact with the API in the embedded C also implements the same level of control. An example of modulated traffic control with a normal traffic scenario on three masters is shown in Figure 5. Each box represents a series of nearly continuous bus transactions with very short (not pictured) idle cycles. A “heavy” traffic scenario would have more activity and less idle time between each series of transactions. An example is shown on Figure 6.

CONCLUSION
Using the graph-based approach improves design quality very early in the project. Protocol coverage is reached efficiently and traffic analysis achieved excellent results, improving the design of blocks connected to the fabric, as well as the interaction between blocks at the system-level.

The advantages of graph-based stimulus and coverage versus constrained random were significant. Fabric and system-level coverage goals were more easily defined and achieved. As a result, dual core ARM Cortex A9 processor-based design was successfully verified several months ahead of schedule, achieved all functional and performance verification goals, and yielded a successful first-pass silicon design.

Editor’s note: A version of this article with the title “Verifying an AXI Fabric Using a Graph-Based Approach” was presented as a paper by the authors at ARM TechCon 2013.
Figure 5 – Normal Traffic Modulation Scenario

Figure 6 – Heavy Traffic Modulation Scenario