

Graph-Based IP Verification in an ARM SoC Environment

by Andreas Meyer, Verification Technologist, Mentor Graphics Corporation

The use of graph-based verification methods for block designs has been shown to provide a significant time and cost reduction when compared to more traditional constrained random techniques. By filling the random constraint space in a controlled random fashion, the full set of constraints can be filled significantly faster than a random constraint solver will. Possibly more importantly, the ability to describe a graph in canonical form is likely to be easier to modify and maintain over the life of an IP block than constraint statements. Combining graphs with a UVM environment, it is possible to extend block-level verification components into a SoC-level test environment with little additional verification development.

GRAPHS

There are a variety of choices for stimulus generation in a verification environment. The possibilities include directed tests, constrained random testing, graphs, or a combination of approaches. Graphs have several advantages that can be important in a reconfigurable verification environment.

A graph defines a single tree of stimulus, where each branch represents one or more legal stimulus sets. The graph itself is a self-contained entity, that may provide an API for higher-level graphs to call, and provides a legal stimulus set for the underlying environment. Because a graph is a self-contained entity, it is possible to define graphs for standard protocols, or specific functions. Once a graph is defined, it can be passed between projects or layered.

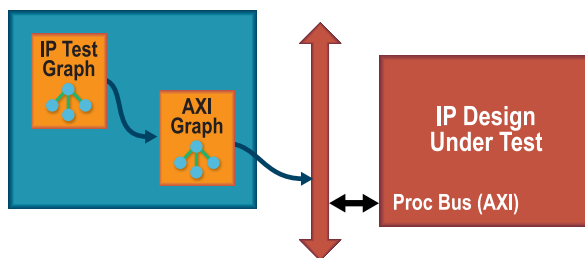


Figure 1: Layered Graphs

A stand-alone IP block tends to have specific functionality that is accessed through one or more standard bus protocols. This can fit nicely with layered stimulus, where a lower-level graph follows the bus protocol, and a higher-level graph provides stimulus for the IP itself. By isolating the protocol knowledge from the IP, the graphs are simpler to develop and maintain, and easier to reuse or acquire from third parties. Because a graph can be expressed as a diagram, it can be significantly easier to understand the stimulus, particularly for team-members who are not familiar with verification.

A graph representation allows the input stimulus space to be expressed as a single tree, with each branch providing one or more legal stimulus sets. Because of this, the full input state space can be enumerated. With a count of all possible legal inputs, input coverage can be detected and reported automatically; in the same way that line coverage is generally provided automatically.

UVM

The ability to migrate from an IP-level to SoC-level verification requires an environment designed for reuse. This is the purpose of the UVM. It provides a standard library to develop modular, encapsulated components with a configurable interconnect to tie them together.

The use of UVM agents allows the details of each protocol to be separated from the stimulus source, checkers, and coverage monitors. Agents may be constructed for proprietary busses, or acquired for standard protocols. This allows the verification environment to be constructed quickly from building-block components that are connected through TLM based on the needs of a particular environment.

STAND-ALONE VERIFICATION ENVIRONMENT

The UVM and graph-based environment shown in figure 2 provides a stand-alone IP verification environment. Through modularity and standard interfaces, a flexible testbench structure can be built with easy access for visibility, debug, and process monitoring.

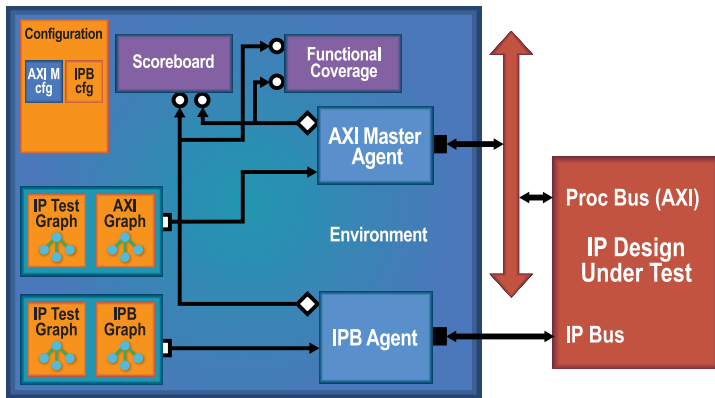


Figure 2: Reconfigurable UVM Environment

However, this environment relies on the processor bus agent (the AXI master agent in figure 2) to directly drive the ARM bus. As a result, this testbench cannot be used unmodified for an SoC integration-level test, since that would result in conflicts on the processor bus.

SOC VERIFICATION

Functional verification requirements at the SoC level are changing. Where there was little interaction or resource contention between IP blocks, an interconnect test was generally sufficient for SoC functional verification. As more complex IP blocks are being integrated into SoCs, system level verification

is required to measure interactions, resource sharing, utilization, or power concerns. The ability to reuse existing, tested block-level IP verification components can significantly reduce the SoC verification effort.

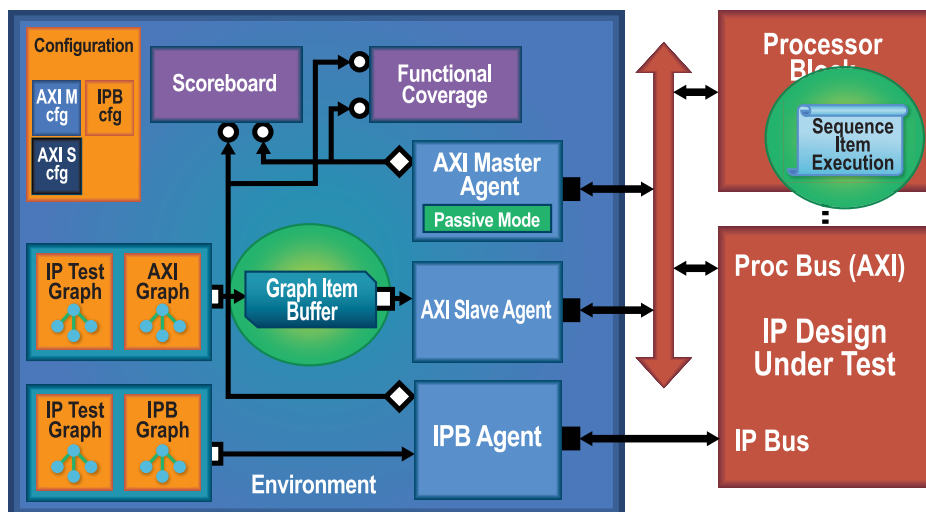
Part of the goal of a UVM-based approach is to ensure that IP-level verification components have the potential to be

instantiated unmodified into an SoC environment. Components such as scoreboards and checkers are good candidates. The IP-level graph stimulus block may also be able to be used unmodified, but the sequence items that were created by the graph can no longer be driven directly onto the processor bus, assuming that the bus only supports one master, and a processor has been instantiated in the SoC environment.

While this changes the connection between the stimulus graph and the IP block, the stimulus itself may not need to change. To connect the

graph, the sequence items need to be accessed by the processor, and then driven on the processor bus. Two new components are needed: a software routine running on the processor that reads sequence items and performs the specified operations, and a graph item buffer. Since sequence items tend to contain basic bus operations: read or write to a specific address with specific data, the software routine performs the operation, and then fetches the next sequence item. The item buffer is a verification component that is addressable by the processor that stores sequence items from the graph and delivers them to the processor when accessed, as shown in figure 3.

Figure 3: Graph Stimulus Driven Through Processor



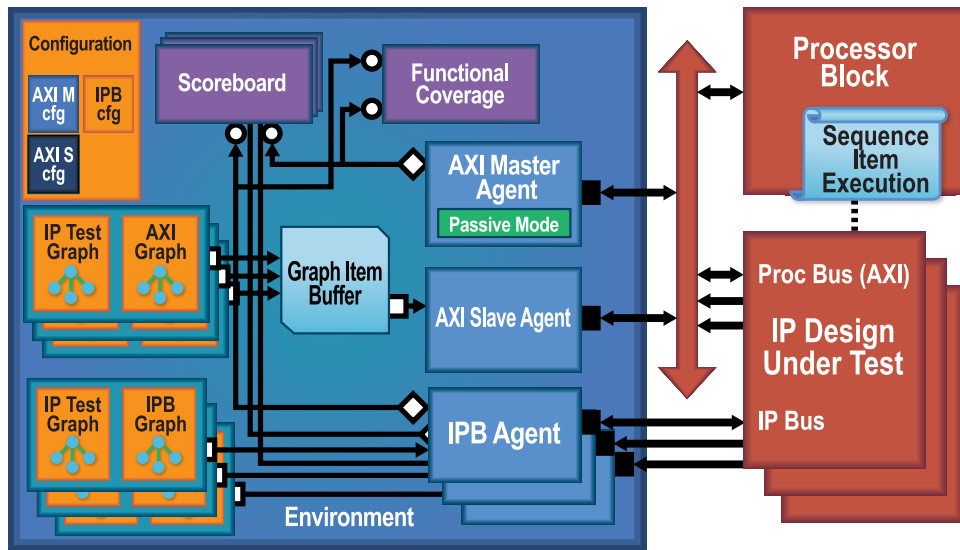


Figure 4: System Exerciser

This approach will allow the IP to be driven from within the SoC integration. The main verification IP: stimulus, check, and coverage may be completely unmodified, while block-level environment is reconfigured. A top layer graph may be used to provide some control and communication between the existing

software and the lower-level API stimulus graph.

The simplest software routine is a loop that waits for an item to be available from the buffer, pulls the item from the graph item buffer, executes it, and then waits for a new item to be available. For more complex operations, this routine may access multiple items. If coordination is required between existing software and stimulus, then the processor may also write commands that provide status or control to a top-level stimulus graph.

The graph item buffer can be a UVM sequencer that accepts items from any number of stimulus graphs, and has a driver to handshake with the software routine. For bi-directional operation, two buffers can be implemented to allow software to control the verification environment.

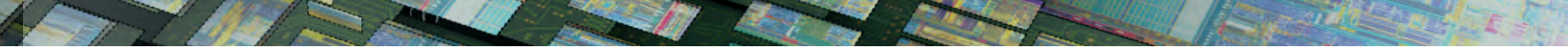
With this method, all items generated by the stimulus graph will be driven onto the processor bus, with the support of the processor. This approach requires that there is an available address space for the graph item buffer to be accessed by the processor. Because the software will execute several operations to fetch items, the timing between items is likely to change. Back-to-back operations may be difficult to reproduce, and some low-level control that was available through a master agent may be lost.

SYSTEM EXERCISER

When multiple IP blocks have been independently tested and then integrated into the SoC, this method can be used as a system exerciser. For IP blocks that do not share resources, multiple stimulus graphs can be connected directly to the graph item buffer, and each IP-level test can run in parallel as shown in figure 4. Block-level scoreboards and functional coverage are used to measure IP functionality. Additional system-level scoreboards can be added to check overall system operation if desired.

Note that this environment will most likely need to include a top-level graph to coordinate and configure each of the lower-level graphs. This may include waiting until the software is running, coordinating operation for shared resources, and determining overall completion of lower-level graphs.

Using just the IP-level stimulus graphs, this simple system exerciser allows the IP blocks to run in parallel, possibly in parallel with system software. Resource sharing, bus utilization and contention, and simple system performance can be observed.

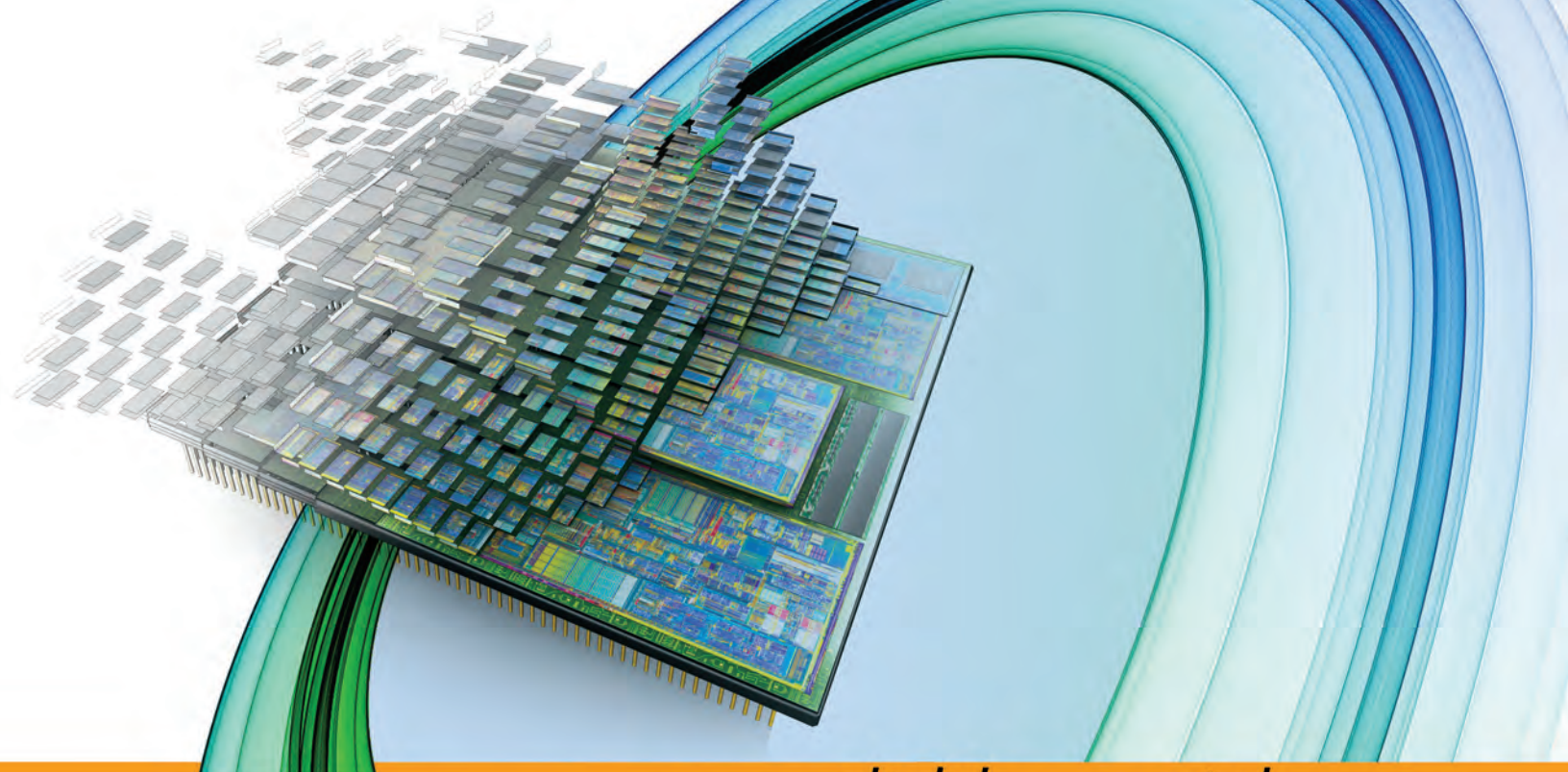


Where the IP tests are run unmodified, only tests that do not cause interference with other blocks can be run. This will likely result in a subset of the IP tests being used. If system-level tests are needed that stress IP interactions, this environment provides all the building blocks to add a new SoC-level graph that drives multiple IP blocks in a coordinated fashion. The graph item buffer can be used to allow coordination between the system-level graph and software if needed.

CONCLUSION

This approach provides a method for functional verification of an IP in a reusable environment. This allows for a low-cost method for a verification engineer to re-verify an IP after it has been integrated into an SoC. Existing, tested, verification components are used to check that the IP is operating correctly within the target SoC environment.

In an SoC environment that contains multiple independent IP blocks, this approach can be extended to run multiple IP verification suites in parallel, providing a simple system-level exerciser. By adding a top-level stimulus graph for coordination, a more complex SoC level verification environment can be constructed from the block-level components.



verification HORIZONS

Editor: Tom Fitzpatrick
Program Manager: Rebecca Granquist

Wilsonville Worldwide Headquarters
8005 SW Boeckman Rd.
Wilsonville, OR 97070-7777
Phone: 503-685-7000

To subscribe visit:
www.mentor.com/horizons

To view our blog visit:
VERIFICATIONHORIZONSBLOG.COM

Mentor
Graphics[®]

www.mentor.com