Soft Copy

Test Development Software

Edited by
Colleen Purtell-Tappen
Technical Editor

Finally, a Verification GUI!

by Donna Mitchell, SynaptiCAD, Inc.

Verification design complexity will force many of you to reconsider your current methods of creating test benches. But the available choices are difficult to evaluate. Will you stick with a modeling language like VHDL, Verilog, or SystemC? Or will you switch to one of the new verification languages such as OpenVera or the "e" language? Whatever choice you make, you are still faced with the problem of developing complex test benches, and maintaining that code over the course of several projects and possibly several verification engineers. Graphical code generation offers a language independent solution to test bench development that enables engineers to quickly describe test benches in a manner that is clear and precise.

Advanced Modeling

Graphical test bench generation is the process of automatically creating test bench source code from an interface description based on graphical timing diagrams. SynaptiCAD has developed TestBencher Pro, a simple and intuitive graphical test bench generator that can be used to model very advanced bus transactions and interface descriptions. Pipelining, split-phase transactions, data structures and memories, and sequence recognition can all be modeled. TestBencher uses timing diagrams to generate bus transactor source code in the user's preferred verification language. By using timing diagrams, the engineer can work at a higher level of abstraction, free from the coding details that arise solely from the choice of a particular verification language.

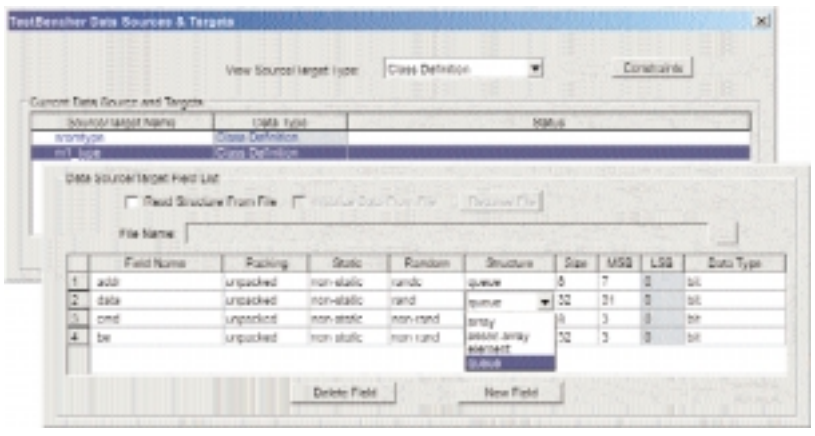

Figure 1: Complex Data Structure using Constrained Random Data

Race Conditions

Test benches are frequently plagued by concurrency and synchronization issues that result in race conditions and non-deterministic execution. Race conditions are caused by subtle design choices that cannot be automatically found in the same way that syntactical errors are. Race conditions only manifest at run time, and often they are not even detectable at run time because the race conditions may "go the right way" on a particular simulator, only to fail when run on another simulator which made a different decision on the order in which to execute the theoretically parallel operations.

Using a graphical code generator, circumstances that can cause race conditions are handled automatically. For example, race conditions are common in clocked functional test benches where signals are frequently driven and sampled at the same clock edge, but on different zero-time delta cycles. Below is a code fragment that shows a simplified example of one way these races can occur. When hand coding, the engineer must be sure to write the code in such a way that the sampling occurs as the first thing that happens during each clock cycle. This is not an easy task in hardware description languages where signals frequently change state several times during a single clock cycle. However, the concurrent events that cause these race conditions are visually obvious in a timing diagram, leading to the easy avoidance of these types of race conditions.

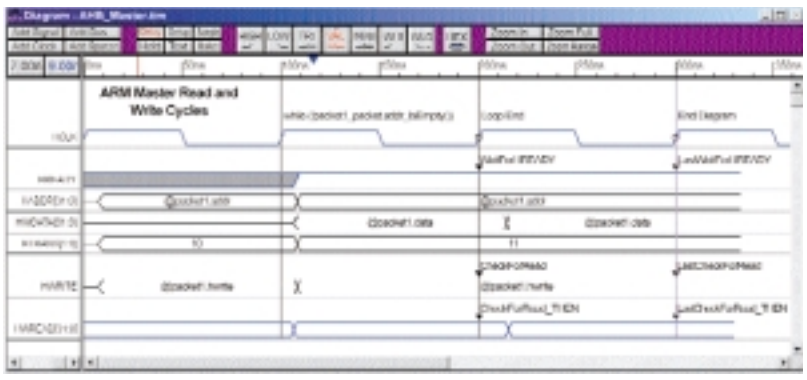

Figure 2: AMBA pipelined bus master driven by queue data structure

Driving Process:
always @ (clock)
data = 'h42;
Monitoring Process:
always @ (clock)
if (data == 'h42)
$display(data);

Constrained Random Data

During the implementation of a hand-coded test bench it is very easy to mismatch port sizes or make other similar small errors. Automatic code generation from graphical descriptions makes it easier for an engineer to define the information in one location and use it throughout the test bench. This is especially true for creating and maintaining complex data structures that are used to supply or store state information for different transactions. A small change to a data structure often causes code changes in many different locations in the test bench. With a graphical interface, once a data structure has been defined for a project, changes to that data structure are automatically propagated throughout the test bench.

The newer verification languages also support advanced data structure features such as constrained randomization of test stimulus during a simulation. With a graphical interface it is very easy to experiment with different randomization options because data structure changes are made in only one location. The example in Figure 1 shows the definition for a data structure that will provide constrained randomized data to a transaction. The fields specified for a data structure can be elements, arrays, or queues. Each field can have different properties defined, depending on the test bench language and the data type being used. These data structures also allow data to be read from or written to a file using a spreadsheet format and can store data internally in memory.

Pipelined Transactions

Another challenge in verification of designs is the implementation of pipelined bus transactions. To speed up the bus, pipelined transactions begin to execute before the end of the previous transaction. This parallel activity makes it more difficult to visualize what the test bench is doing. Figure 2 shows how a single timing diagram can be used to model both the read and write transactions of an AMBA (Advanced High-Performance Bus) master device.

Test Bench Maintenance

Test bench code is often difficult to understand even when written using modular programming techniques because of the large amount of parallel activity occurring in the test bench. Timing diagrams allow a much clearer and concise description of the interaction of parallel processes and signal activity. A graphical representation also facilitates the collaboration of many engineers on a single test bench by removing the need to interpret source code. Any engineer familiar with the design specifications is able to look at a given timing diagram and have an immediate understanding of what the transactor does, dramatically simplifying test bench maintenance.

Summary

Graphical automatic test bench generation provides a good solution to many of the problems faced during functional design verification. In this solution, timing diagrams are used to describe bus transactions in the test bench. Engineers are familiar with timing diagrams, so they are easily understood and the graphical representation enables the engineer to quickly visualize the test bench at a higher level of abstraction. The transactions depicted in the timing diagrams form modular components of a test bench that can quickly be modified and reused in other designs, paving the way for faster verification of future designs.

Donna Mitchell is Vice President of Strategic Marketing at SynaptiCAD, Inc. She received her BS and MS degrees in electrical engineering from Virginia Tech. Mitchell is one of the two founders of SynaptiCAD Inc. She can be reached via email at donna@syncad.com. SynaptiCAD Inc. is located at 520 Prices Fork Rd. #C4, Blacksburg, VA 24060, (540) 953-3390; Fax: (540) 953-3078; www.syncad.com