Block Level Testing

CHAPTER 14

I can give you a six-word formula for success:
Think things through—then follow through.

Sir Walter Scott

In many endeavors, follow-through is everything. From sports to parenting, it's not only what you say, but what you do that is important. This chapter is the first of the "follow-through" chapters.

We use all the tools, tips, and techniques from the rest of the handbook and apply them to something resembling a real-world example. This is the first complete example of what a test system using C++ might look like.

We look at a block-level verification system. Later, we'll adapt this same system to be used at the full-chip level.

Overview

This chapter covers a block-level verification effort as part of a large project. The goal is to verify a UART 16550 RTL block, written in Verilog. To do this, we will build an environment that will not only verify the block but also provide adaptable verification components for later project stages.

The example presented here will show all verification components needed to do UART 16550 verification as well as a fully randomized test. Several points of interest in the code will be highlighted throughout the chapter. (We present code in a slightly different form from the source code on the accompanying CD.) Sometimes, we merge a the interface and implementation of a class together, although they are separated in the source code. Also, we may abbreviate a class interface or some method's code to get straight to the point.

This chapter differs from the Truss tutorial chapter (in Part II) in that it focuses more on the middle layers of a verification system instead of on the flow. The middle layers are where managing the complexity of a verification system comes mostly into play.

If you want to look closer at execution order, it's recommended that you start by referring to the *Truss standard test algorithm* known as the "*dance*" in the Truss Basics chapter. Then, with the "dance" as a reference, divide and conquer by using a ends-in approach. In other words, take a closer look at both the top level `test.cpp` (and its related `test_components`) and the interface aggregrator, `testbench.cpp`. This will help show the overall structure and flow of the environment.

This chapter will talk about a few things. First, we set up the example with a theory of operation. This section highlights the overall environment and the interfaces that are used.

Then we look at several points of interest in the code. These points of interest cover code complexity problems of the middle layers in a verification system. We present these middle-layer techniques in their order of execution, by first looking at power-on reset, then at configuration and traffic generation, and then at checking.