Chapter 6

Randomization

6.1 Introduction

As designs grow larger, it becomes more difficult to create a complete set of stimuli needed to check their functionality. You can write a directed testcase to check a certain set of features, but you cannot write enough directed testcases when the number of features keeps doubling on each project. Worse yet, the interactions between all these features are the source for the most devious bugs and are the least likely to be caught by going through a laundry list of features.

The solution is to create testcases automatically using constrained-random tests (CRT). A directed test finds the bugs you think are there, but a CRT finds bugs you never thought about, by using random stimulus. You restrict the test scenarios to those that are both valid and of interest by using constraints.

Creating a CRT environment takes more work than creating one for directed tests. A simple directed test just applies stimulus, and then you manually check the result. These results are captured as a golden log file and compared with future simulations to see whether the test passes or fails. A CRT environment needs not only to create the stimulus but also to predict the result, using a reference model, transfer function, or other techniques. However, once this environment is in place, you can run hundreds of tests without having to hand-check the results, thereby improving your productivity. This trade-off of test-authoring time (your work) for CPU time (machine work) is what makes CRT so valuable.

A CRT is made of two parts: the test code that uses a stream of random values to create input to the DUT, and a seed to the pseudo-random number generator (PRNG), shown in section 6.15.1. You can make a CRT behave differently just by using a new seed. This feature allows you to leverage each test so each is the functional equivalent of many directed tests, just by changing seeds. You are able to create more equivalent tests using these techniques than with directed testing.

You may feel that these random tests are like throwing darts. How do you know when you have covered all aspects of the design? The stimulus space is too large to generate every possible input by using `for`-loops, so you need to generate a useful subset. In Chapter 9 you will learn how to measure verification progress by using functional coverage.