Teal Basics

CHAPTER 5

Coming together is a beginning. Keeping together is progress. Working together is success.

Henry Ford

Building a verification system is a daunting task, but build we must. That is why we use the technique of layering, to break the problem down. By starting with the lowest layer—that is, the one that directly drives and senses the wires—we can start to get some real work done. Still, because C++ is not what most hardware engineers use for their HDL, we'll need an interface layer to connect the HDL with C++. Teal is just such an interface. Teal tries to be as unobtrusive as possible, using terms borrowed from the HDL domain, such as *posedge* and *reg*.

This chapter introduces Teal and shows how to use it. We'll talk a bit about the main parts of Teal—for example, how you can (fairly seamlessly) get and set values in the HDL, and how you can pause execution until HDL signals change.

Overview

Teal is a C++ class library for functional verification. Teal is tiny, consisting of only a handful of source files, yet it provides the necessary minimum features for verification. (A version of Teal is on the companion CD.)

Teal, like Vera, SystemVerilog, and "e," provides the illusion that the verification system is in control of the chip. In Teal, you write a `verification_top()` function, and create tests, generators, checkers, drivers, and monitors. Each of these objects can appear to be running independently of the chip, with each in its own thread of execution. Of course, in reality these threads only execute in response to a chip wire or register change. However, by driving wires and registers, the threads do, in some sense, control the chip.

Teal is unobtrusive; it does not get in the way of your C or C++ structure. You don't put Teal calls everywhere you want to sample or drive a signal, so Teal is also unobtrusive in the HDL code.

The authors realize that many companies have developed their own version of an HDL-to-C/C++ interconnect. We encourage those companies to contact us and share their experiences, so Teal can be made better. This is one of the reasons why Teal is open source.

What Teal provides

Teal enables functional verification by providing connections to HDL signals and allowing actions based on changes in the HDL simulation. It encourages the development of independent generators, checkers, drivers, and monitors by providing management for user-created threads that execute concurrently with the HDL simulation. Teal provides repeatability and constrained random-number generation, as well as a simple interface to pass in runtime arguments, either through the code or command line, or through "scenario" text files. Teal also provides flexible message printing.

This functionality provides the basis for functional verification, but it serves as only a small part of a verification project. You must still write code that stimulates the design, checks the output, and controls the