OOP Connections

CHAPTER 12

Oh what a tangled web we weave,
When first we practice to deceive.

Sir Walter Scott, Marmion. *Canto vi, Stanza 17*

Connecting classes together is more important than the classes themselves. How can this be? It is so because, by definition, the connecting of classes involves jumping around in the code base. Managing this is mentally more difficult than simply managing the code within a class. For example, when you see a pointer to a class in a method call, you have to think about why the method needs that class. The answer depends upon whether the system is a tangled web or a well-architected series of connections.

Often you have to find the header file for the class and go look at the implementation of the method. In the worst case, the code make no sense whatsoever, even after you stare at the implementation. In the best case, the connections are obvious, such as when a test gets a pointer to a testbench.

Overview

In hardware design we connect modules together and worry about clock domain crossings. With verification, we connect instances of classes and methods together and worry about crossing the threads of execution. In addition, the connections in verification may be temporary (for example, they are used only within a method), or they may be permanent (for example, when a constructor takes in a pointer to a logger and stores it in a data member.

Connections in your code can either form a spider web of complicated and confusing relationships, or they can be a highway, with well-defined points that connect to other roads. Recall that one person's web is another person's highway, so picking the right connection technique may not be universally appreciated. There are many connection techniques and, as a result, trade-offs to be made, as this chapter shows.

We first discuss the various types of connections, then look at implementations of these connection types. We then present the simpler connection types first, increasing the complexity of the connections as the chapter progresses.

We first look at how to classify connections. The type of connection is evaluated according to how much information one class has about another. At one end of the information scale, classes have no mechanism to determine whether any other class instances are connected. At the other end of the scale, a class has a pure virtual method that must be implemented to make the connection.

The idea is to build the appropriate type of connection for the problem at hand. Too loose a connection makes code unnecessarily complicated. This because loose connections make few assumptions about the other side, which in-turn makes tracking events harder. Connections that are too tight, on the other hand, may make code harder to adapt.

While these connection techniques are general, some of them can be used between verification components operating in different threads. Why bring threading into the discussion? In verification systems many events need to happen in parallel. This is normally done through threading. With threading, however, comes a set of problems related to accessing common data. How can a thread be given sole access to a common resource? How