Chapter 8

Advanced OOP and Guidelines

8.1 Introduction

How would you create a complex class for a bus transaction that also includes performs error injection and variable delays? The first approach is to put everything in a large, flat class. This approach is simple to build, easy to understand (all the code is right there in one class) but can be slow to develop and debug. Additionally, such a large class is a maintenance burden, as anyone who wants to make a new transaction behavior has to edit the same file. Just as you would never create a complex RTL design using just one Verilog module, you should break classes down into smaller, reusable blocks.

The next choice is composition. As you learned in Chapter 4, you can instantiate one class inside another, just as you instantiate modules inside another, building up a hierarchical testbench. You write and debug your classes from the top down or bottom up. Look for natural partitions when deciding what variables and methods go into the various classes.

Sometimes it is difficult to divide the functionality into separate parts. Take the example of a bus transaction with error injection. When you write the original class for the transaction, you may not think of all the possible error cases. Ideally, you would like to make a class for a good transaction, and later add different error injectors. The transaction may have data fields and an error-checking CRC field generated from the data. One form of error injection is corruption of the CRC field. If you use composition, you need separate classes for a good transaction, and an error transaction. Testbench code that used good objects would have to be rewritten to process the new error objects. What you need is something that resembles the original class but adds a few new variables and methods. This result is accomplished through inheritance.

Inheritance allows a new class to be derived from an existing one in order to share its variables and routines. The original class is known as the base or super class, while the new one, since it extends the capability of the base class, is called the extended class. Inheritance provides reusability by adding features, such as error injection, to an existing class, the basic transaction, without modifying the base class.

The real power of OOP is that it gives you the ability to take an existing class, such as a transaction, and selectively change parts of its behavior by replacing routines, but without having to change the surrounding infrastructure. With some planning, you can create a testbench solid enough