Chapter 10

Advanced Interfaces

10.1 Introduction

In Chapter 5 you learned how to use interfaces to connect the design and testbench. These physical interfaces represent real signals, similar to the wires that connected ports in Verilog-1995. The testbench uses these interfaces by statically connecting to them through ports. However, for many designs, the testbench needs to connect dynamically to the design.

In a network switch, a single Driver class may connect to many interfaces, one for each input channel of the DUT. You wouldn't want to write a unique Driver for each channel — instead you want to write a generic Driver, instantiate it N times, and have it connect each of the N physical interfaces. You can do this in SystemVerilog by using a virtual interface that is merely a handle to a physical interface.

You may need to write a testbench that attaches to several different configurations of your design. In one configuration, the pins of the chip may drive a USB bus, while in another the same pins may drive an I2C serial bus. Once again, you can use a virtual interface in the testbench so you can decide at runtime which drivers to use.

A SystemVerilog interface is more than just signals — you can put executable code inside. This might include routines to read and write to the interface, initial and always blocks that run code inside the interface, and assertions to constantly check the status of the signals. However, do not put testbench code in an interface. Program blocks have been created expressly for building a testbench, including scheduling their execution in the Reactive region, as described in the LRM.

10.2 Virtual Interfaces with the ATM Router

The most common use for a virtual interface is to allow objects in a testbench to refer to items in a replicated interface using a generic handle rather than the actual name. Virtual interfaces are the only mechanism that can bridge the dynamic world of objects with the static world of modules and interfaces.