

# Chapter 10

## Advanced Interfaces

In Chapter 4 you learned how to connect the design and testbench with interfaces. These physical interfaces represent real signals, similar to the wires that connected ports in Verilog-1995. A testbench uses these interfaces by statically connecting to them through ports. However, for many designs, the testbench needs to connect dynamically to the design.

For example, 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 to each of the  $N$  physical interfaces. You can do this in SystemVerilog by using a virtual interface, which is merely a handle or pointer to a physical interface.<sup>18</sup>

You may need to write a testbench that attaches to several different configurations of your design. In another example, a chip may have multiple configurations. In one, the pins might drive a USB bus, whereas in another the same pins may drive an I2C serial bus. Once again, you can use a virtual interface so you can decide at run-time which drivers to run in your testbench.

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 SystemVerilog LRM.

<sup>18</sup>. A better name for a virtual interface would be a “ref interface.”