This chapter presents Verilog from the point of view of a designer wanting to describe a design, perform pre-synthesis simulation, and synthesize his or her design for programming an FPGA or generating a layout. Many of the complex Verilog constructs related to timing and fine modeling features of this language will not be covered here. The chapter first describes Verilog with emphasis on design using simple examples. We will cover the basics, just enough to describe our examples. In a later section after a general familiarity with the language is gained, more complex features of the Verilog language with emphasis on testbench development will be described.

3.1 Design with Verilog

Verilog syntax and language constructs are designed to facilitate description of hardware components for simulation and synthesis. In addition, Verilog can be used to describe testbenches, specify test data and monitor circuit responses. Figure 3.1 shows a simulation model that consists of a design and its testbench in Verilog. Simulation output is generated in form of a waveform for visual inspection or data files for machine readability.

After a design passes basic functional validations, it must be synthesized into a netlist of components of a target library. Constructs used for verification of a design, or timing checks and timing specifications are not synthesizable. A Verilog design that is to be synthesized must use language constructs that have a clear hardware correspondence. Figure 3.2 shows a block diagram specifying the synthesis process.
The output of synthesis is a netlist of components of the target library. Often synthesis tools have an option to generate this netlist in Verilog. In this case, the same testbench prepared for pre-synthesis simulation can be used with the netlist generated by the synthesis tool.

**3.1.1 Modules**

The entity used in Verilog for description of hardware components is a module. A module can describe a hardware component as simple as a transistor or a network of complex digital systems. As shown in Figure 3.3, modules begin with the module keyword and end with endmodule.

```
module
...
...
endmodule
```