the *currentstate* to the FSM's initial state, *S0*. Otherwise, on the rising edge of the clock, the process updates the state register with the *nextstate*.

The second process, named *comblogic*, is sensitive to the inputs to the combinational logic, namely, the external inputs *B* and *S*, and the state register output *currentstate*. When either of those signals changes, the process sets the FSM's outputs, in this case *L*, *Dreg_clr*, *Dreg_ld*, *Dctr_clr*, and *Dctr_ld*, with the appropriate value for the current state. In the controller example of Figure 9.24, the FSM's output *x* was defined within the case statement for all possible states. With five outputs that we must define in the *LDM_Controller* and five possible states, assigning the values to all outputs in each state would be cumbersome. Furthermore, finding a mistake and making corrections or modification to the controller would become very difficult in a larger FSM consisting of more states and having many more outputs. Instead, the process uses a different approach, in which a default value for the all outputs is first assigned and only the deviations from the defaults are assigned later. The process first assigns a default value of `0` to all five outputs. The process then evaluates the current state and assigns the values to the outputs only when the output should be `1`. The process also assigns the value `0` to several signals within the ***case*** statements; however, these assignments are included only to clearly indicate the behavior of the controller (they are redundant but help make the description easier to understand).

The process also determines what the next state should be, based on the current state and the values of inputs *B* and *S*. The next state will be loaded into the state register by the state register process on the next positive clock edge.

▶ 9.6 CHAPTER SUMMARY

This chapter stated that hardware description languages (HDLs) are widely used in modern digital design. The chapter introduced three popular HDLs: VHDL, Verilog, and SystemC. The chapter introduced those HDLs primarily through the use of examples, illustrating how each HDL might be used to describe combinational logic, sequential logic, datapath components, and RTL behavior and structure. To become proficient at the use of HDLs, a more thorough study of a particular HDL might be helpful. This chapter also illustrated the point that the three different HDLs have several aspects in common.

▶ 9.7 EXERCISES

The following exercises can be completed using any of the HDLs described in this chapter. The solution to Exercise 9.1 is needed in order to solve many of the remaining exercises.

SECTION 9.2: COMBINATIONAL LOGIC DESCRIPTION USING HARDWARE DESCRIPTION LANGUAGES

9.1 (a) Create combinational behavioral HDL descriptions of *Inv*, *AND2*, *AND3*, *OR2*, and *OR3* gates, such that those gates can then be used as components in another design. Use names *a*, *b*, and *c* for inputs (as needed) and *F* for the output. (b) Create a testbench for the *AND3* gate to test its correctness.

9.2 (a) Create combinational behavioral HDL descriptions for *NAND2* and *XOR2* gates, where each gate has two inputs *a* and *b* and a single output *F*. (b) Create a separate testbench for each.

9.3 Show that the parity generator equation of Example 2.19 correctly generates the desired parity bit, by first capturing the equation as combinational behavior in an HDL, creating a testbench that tests all possible input combinations, and then showing that simulation yields correct output value for each input combination.

9.4 For the seat belt warning light system in Example 2.7: (a) create a combinational HDL description using the given equation, (b) create a testbench to test the description, checking all possible combinations of input values, (c) simulate the system using your testbench, and verify that the output values are correct (d) create a structural HDL description of the given circuit, (e) reuse the testbench to test the structural description.

9.5 Example 2.12 algebraically transformed one equation into another. Show that those two equations represent the same function by using simulation. First, create a combinational behavioral description of the initial equation. Second, create a combinational behavioral description of the final equation. Third, create a testbench that tests all possible combinations of input values and show that simulation yields identical output values for both equations.

9.6 For the 2x4 decoder in Figure 2.62, but extended with an enable input: (a) create a structural HDL description of the given circuit, (b) create a testbench to test the structural description, testing all possible input combinations, (c) simulate to verify correct behavior.

9.7 For the 4x1 mux in Figure 2.67: (a) create a structural HDL description of the given circuit, (b) create a testbench to test the structural description; do not test all possible input combinations, but rather a few input cases for each possible combination of the select control inputs, (c) simulate to verify correct behavior.

9.8 Create a behavioral HDL description of a 2x1 multiplexor described in Figure 2.54. Then, create a structural HDL description that combines three 2x1 multiplexors to create a 4x1 multiplexor as shown in Figure 9.46.

9.9 Create a combinational behavioral HDL description of an 8-bit 4x1 multiplexor. Be sure to specify the input and output ports using a multiple-bit data type.

9.10 Clearly explain the difference between a structural HDL description and a behavioral HDL description. Explain the benefits of using each.

9.11 Explain why a combinational behavioral HDL description must include all the combinational circuit's inputs in a sensitivity list. In particular, explain why omitting an input actually describes a sequential circuit.

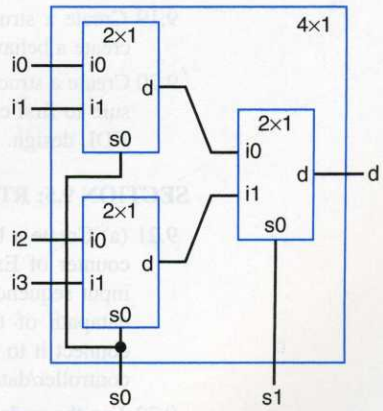

Figure 9.46 4x1 multiplexor composed of three 2x1 multiplexors.

SECTION 9.3: SEQUENTIAL LOGIC DESCRIPTION USING HARDWARE DESCRIPTION LANGUAGES

9.12 (a) Create a behavioral HDL description of a 32-bit basic register (no load or clear control inputs). (b) Create a testbench to test the description for some random data input values. (c) Simulate the system to demonstrate correct behavior.

9.13 (a) Create a behavioral HDL description of the sequence generator of Figure 3.66. (b) Create a testbench; with no inputs, the testbench will not have to set input values and instead will just let the clock run. (c) Simulate the system to show that it generates the correct repeating sequence.

9.14 (a) Create a behavioral HDL description of the flight attendant call button system of Figure 3.53. (b) Create a testbench to test various sequences of the call and cancel buttons being pressed (there are many possibile sequences; you can't test all of them). (c) Simulate the system to show correct behavior.

SECTION 9.4: DATAPATH COMPONENT DESCRIPTION USING HARDWARE DESCRIPTION LANGUAGES

9.15 (a) Create a behavioral HDL description of an 8-bit parallel load register with load and clear control inputs (both synchronous; clear has priority over load). (b) Create a testbench to test the description, testing load, clear, simultaneous load/clear, and holding the register value. (c) Simulate the system to demonstrate correct behavior.

9.16 Create a behavioral HDL description of an 8-bit register with two control inputs *s0* and *s1* having the control behavior described in Figure 9.47.

s1	s0	Operation
0	0	Maintain present value
0	1	Parallel load
1	0	Shift right
1	1	Rotate right

Figure 9.47 Register operation table.

9.17 Create a structural HDL description of a half-adder.

9.18 Create a structural HDL description of a 4-bit carry-ripple adder without a carry input. First create a behavioral description of a full-adder, and then use the full-adder component in your carry-ripple adder description.

9.19 Create a structural HDL description of the counter circuit in Figure 4.69. Be sure to first create a behavioral HDL description of each component used in your structural HDL design.

9.20 Create a structural HDL description of the 4-bit adder/subtractor circuit in Figure 4.54(b). Be sure to first create a behavioral HDL description of each component used in your structural HDL design.

SECTION 9.5: RTL DESIGN USING HARDWARE DESCRIPTION LANGUAGES

9.21 (a) Create a behavioral HDL description of the high-level state machine for the cycles-high counter of Example 5.1. (b) Create a testbench and simulate the system for some sample input sequences and verify correct behavior. (c) Create a structural HDL description for the datapath of the cycles-high counter in Figure 5.16(d). (d) Create a controller FSM and connect it to the datapath as in Figure 5.16(d). (e) Use the earlier testbench to simulate the controller/datapath system to verify correct behavior.

9.22 For the soda dispenser example used throughout Chapter 5: (a) create a behavioral HDL description of the HLSM, (b) create a testbench and simulate the system for some sample input sequences and verify correct behavior, (c) create a structural HDL description for the datapath, (d) create a controller FSM and connect it to the datapath, (e) use the earlier testbench to simulate the controller/datapath system to verify correct behavior.

9.23 Starting from the C description shown in Figure 9.48, create an RTL design of a greatest common divisor (GCD) calculator that takes as input two 16-bit inputs *a* and *b*, an enable input *go*, and a 16-bit output *D*. When *go* is '1', the GCD calculator will compute the greatest common divisor and output the GCD on the output *D*. (a) Convert the C to a high-level state machine captured behaviorally in an HDL, (b) create a testbench and simulate for various

input values, (c) create a controller and datapath HDL description, (d) simulate using the same testbench.

```
uint GCD(uint a, uint b) // not quite C syntax
{
   while ( a ! = b ) {
      if( a > b ) {
         a = a - b;
      } else {
         b = b - a;
      }
   }
   return(a);
}
```

Figure 9.48 C program description of a greatest common divisor calculator.