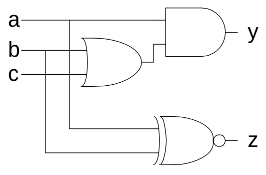
CHAPTER 4

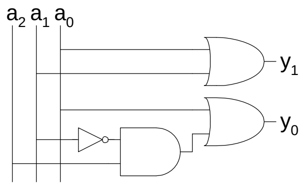
Note: the HDL files given in the following solutions are available on the textbook’s companion website at:

http://textbooks.elsevier.com/9780123704979

Exercise 4.1



Exercise 4.2



© 2015 Elsevier, Inc.

Exercise 4.3

SystemVerilog

```
module xor_4(input logic [3:0] a,
             output logic       y);

   assign y = ^a;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity xor_4 is
  port(a: in  STD_LOGIC_VECTOR(3 downto 0);
       y: out STD_LOGIC);
end;

architecture synth of xor_4 is
begin
  y <= a(3) xor a(2) xor a(1) xor a(0);
end;
```

Exercise 4.4

ex4_4.tv file:

```
0000_0
0001_1
0010_1
0011_0
0100_1
0101_0
0110_0
0111_1
1000_1
1001_0
1010_0
1011_1
1100_0
1101_1
1110_1
1111_0
```

SystemVerilog

```
module ex4_4_testbench();
  logic         clk, reset;
  logic  [3:0]  a;
  logic         yexpected;
  logic         y;
  logic [31:0]  vectornum, errors;
  logic [4:0]   testvectors[10000:0];

  // instantiate device under test
  xor_4 dut(a, y);

  // generate clock
  always
    begin
      clk = 1; #5; clk = 0; #5;
    end

  // at start of test, load vectors
  // and pulse reset
  initial
    begin
      $readmemb("ex4_4.tv", testvectors);
      vectornum = 0; errors = 0;
      reset = 1; #27; reset = 0;
    end

  // apply test vectors on rising edge of clk
  always @(posedge clk)
    begin
      #1; {a, yexpected} =
            testvectors[vectornum];
    end

  // check results on falling edge of clk
   always @(negedge clk)
    if (~reset) begin // skip during reset
      if (y !== yexpected) begin
        $display("Error: inputs = %h", a);
        $display("  outputs = %b (%b expected)",
                 y, yexpected);
        errors = errors + 1;
      end
      vectornum = vectornum + 1;
      if (testvectors[vectornum] === 5'bx) begin
       $display("%d tests completed with %d errors",
                vectornum, errors);
        $finish;
      end
    end
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use STD.TEXTIO.all;
use work.txt_util.all

entity ex4_4_testbench is -- no inputs or outputs
end;

architecture sim of ex4_4_testbench is
  component sillyfunction
    port(a: in  STD_LOGIC_VECTOR(3 downto 0);
         y: out STD_LOGIC);
  end component;
  signal a: STD_LOGIC_VECTOR(3 downto 0);
  signal y, clk, reset: STD_LOGIC;
  signal yexpected: STD_LOGIC;
  constant MEMSIZE: integer := 10000;
  type tvarray is array(MEMSIZE downto 0) of
    STD_LOGIC_VECTOR(4 downto 0);
  signal testvectors: tvarray;
  shared variable vectornum, errors: integer;
begin
  -- instantiate device under test
  dut: xor_4 port map(a, y);

  -- generate clock
  process begin
    clk <= '1'; wait for 5 ns;
    clk <= '0'; wait for 5 ns;
  end process;

  -- at start of test, load vectors
  -- and pulse reset
  process is
    file tv: TEXT;
    variable i, j: integer;
    variable L: line;
    variable ch: character;
  begin
    -- read file of test vectors
    i := 0;
    FILE_OPEN(tv, "ex4_4.tv", READ_MODE);
    while not endfile(tv) loop
      readline(tv, L);
      for j in 4 downto 0 loop
        read(L, ch);
        if (ch = '_') then read(L, ch);
        end if;
        if (ch = '0') then
          testvectors(i)(j) <= '0';
        else testvectors(i)(j) <= '1';
        end if;
      end loop;
      i := i + 1;
    end loop;
    vectornum := 0; errors := 0;
    reset <= '1'; wait for 27 ns; reset <= '0';
    wait;
  end process;
```

(*VHDL continued on next page*)

© 2015 Elsevier, Inc.

(*continued from previous page*)

VHDL

```
  -- apply test vectors on rising edge of clk
  process (clk) begin
    if (clk'event and clk = '1') then

      a <= testvectors(vectornum)(4 downto 1)
        after 1 ns;
      yexpected <= testvectors(vectornum)(0)
        after 1 ns;
    end if;
  end process;

  -- check results on falling edge of clk
  process (clk) begin
    if (clk'event and clk = '0' and reset = '0') then
      assert y = yexpected
        report "Error: y = " & STD_LOGIC'image(y);
      if (y /= yexpected) then
        errors := errors + 1;
      end if;
      vectornum := vectornum + 1;
      if (is_x(testvectors(vectornum))) then
        if (errors = 0) then
          report "Just kidding -- " &
                 integer'image(vectornum) &
                 " tests completed successfully."
                 severity failure;
        else
          report integer'image(vectornum) &
                 " tests completed, errors = " &
                 integer'image(errors)
                 severity failure;
        end if;
      end if;
    end if;
  end process;
end;
```

Exercise 4.5

SystemVerilog

```
module minority(input  logic a, b, c
                output logic y);

   assign y = ~a & ~b | ~a & ~c | ~b & ~c;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity minority is
  port(a, b, c:   in  STD_LOGIC;
       y:         out STD_LOGIC);
end;

architecture synth of minority is
begin
  y <= ((not a) and (not b)) or ((not a) and (not c))
      or ((not b) and (not c));
end;
```

Exercise 4.6

SystemVerilog

```
module sevenseg(input  logic [3:0] data,
                output logic [6:0] segments);

  always_comb
    case (data)
      //                   abc_defg
      4'h0: segments = 7'b111_1110;
      4'h1: segments = 7'b011_0000;
      4'h2: segments = 7'b110_1101;
      4'h3: segments = 7'b111_1001;
      4'h4: segments = 7'b011_0011;
      4'h5: segments = 7'b101_1011;
      4'h6: segments = 7'b101_1111;
      4'h7: segments = 7'b111_0000;
      4'h8: segments = 7'b111_1111;
      4'h9: segments = 7'b111_0011;
      4'ha: segments = 7'b111_0111;
      4'hb: segments = 7'b001_1111;
      4'hc: segments = 7'b000_1101;
      4'hd: segments = 7'b011_1101;
      4'he: segments = 7'b100_1111;
      4'hf: segments = 7'b100_0111;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity seven_seg_decoder is
  port(data:     in  STD_LOGIC_VECTOR(3 downto 0);
       segments: out STD_LOGIC_VECTOR(6 downto 0));
end;

architecture synth of seven_seg_decoder is
begin
  process(all) begin
    case data is
--                                abcdefg
      when X"0"   => segments <= "1111110";
      when X"1"   => segments <= "0110000";
      when X"2"   => segments <= "1101101";
      when X"3"   => segments <= "1111001";
      when X"4"   => segments <= "0110011";
      when X"5"   => segments <= "1011011";
      when X"6"   => segments <= "1011111";
      when X"7"   => segments <= "1110000";
      when X"8"   => segments <= "1111111";
      when X"9"   => segments <= "1110011";
      when X"A"   => segments <= "1110111";
      when X"B"   => segments <= "0011111";
      when X"C"   => segments <= "0001101";
      when X"D"   => segments <= "0111101";
      when X"E"   => segments <= "1001111";
      when X"F"   => segments <= "1000111";
      when others => segments <= "0000000";
    end case;
  end process;
end;
```

Exercise 4.7

ex4_7.tv file:

```
0000_111_1110
0001_011_0000
0010_110_1101
0011_111_1001
0100_011_0011
0101_101_1011
0110_101_1111
0111_111_0000
1000_111_1111
1001_111_1011
1010_111_0111
1011_001_1111
1100_000_1101
1101_011_1101
1110_100_1111
1111_100_0111
```

© 2015 Elsevier, Inc.

Option 1:

SystemVerilog

```
module ex4_7_testbench();
  logic        clk, reset;
  logic [3:0]  data;
  logic [6:0]  s_expected;
  logic [6:0]  s;
  logic [31:0] vectornum, errors;
  logic [10:0] testvectors[10000:0];

  // instantiate device under test
  sevenseg dut(data, s);

  // generate clock
  always
    begin
      clk = 1; #5; clk = 0; #5;
    end

  // at start of test, load vectors
  // and pulse reset
  initial
    begin
      $readmemb("ex4_7.tv", testvectors);
      vectornum = 0; errors = 0;
      reset = 1; #27; reset = 0;
    end

  // apply test vectors on rising edge of clk
  always @(posedge clk)
    begin
      #1; {data, s_expected} =
            testvectors[vectornum];
    end

  // check results on falling edge of clk
   always @(negedge clk)
    if (~reset) begin // skip during reset
      if (s !== s_expected) begin
        $display("Error: inputs = %h", data);
        $display("  outputs = %b (%b expected)",
                 s, s_expected);
        errors = errors + 1;
      end
      vectornum = vectornum + 1;
      if (testvectors[vectornum] === 11'bx) begin
       $display("%d tests completed with %d errors",
                vectornum, errors);
        $finish;
      end
    end
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use STD.TEXTIO.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.STD_LOGIC_ARITH.all;

entity ex4_7_testbench is -- no inputs or outputs
end;

architecture sim of ex4_7_testbench is
  component seven_seg_decoder
  port(data:     in  STD_LOGIC_VECTOR(3 downto 0);
       segments: out STD_LOGIC_VECTOR(6 downto 0));
  end component;
  signal data: STD_LOGIC_VECTOR(3 downto 0);
  signal s:    STD_LOGIC_VECTOR(6 downto 0);
  signal clk, reset: STD_LOGIC;
  signal s_expected: STD_LOGIC_VECTOR(6 downto 0);
  constant MEMSIZE: integer := 10000;
  type tvarray is array(MEMSIZE downto 0) of
    STD_LOGIC_VECTOR(10 downto 0);
  signal testvectors: tvarray;
  shared variable vectornum, errors: integer;
begin
  -- instantiate device under test
  dut: seven_seg_decoder port map(data, s);

  -- generate clock
  process begin
    clk <= '1'; wait for 5 ns;
    clk <= '0'; wait for 5 ns;
  end process;

  -- at start of test, load vectors
  -- and pulse reset
  process is
    file tv: TEXT;
    variable i, j: integer;
    variable L: line;
    variable ch: character;
  begin
    -- read file of test vectors
    i := 0;
    FILE_OPEN(tv, "ex4_7.tv", READ_MODE);
    while not endfile(tv) loop
      readline(tv, L);
      for j in 10 downto 0 loop
        read(L, ch);
        if (ch = '_') then read(L, ch);
        end if;
        if (ch = '0') then
          testvectors(i)(j) <= '0';
        else testvectors(i)(j) <= '1';
        end if;
      end loop;
      i := i + 1;
    end loop;
```

(*VHDL continued on next page*)

© 2015 Elsevier, Inc.

(*continued from previous page*)

VHDL

```
    vectornum := 0; errors := 0;
    reset <= '1'; wait for 27 ns; reset <= '0';
    wait;
  end process;

  -- apply test vectors on rising edge of clk
  process (clk) begin
    if (clk'event and clk = '1') then

      data <= testvectors(vectornum)(10 downto 7)
        after 1 ns;
    s_expected <= testvectors(vectornum)(6 downto 0)
        after 1 ns;
    end if;
  end process;

  -- check results on falling edge of clk
  process (clk) begin
    if (clk'event and clk = '0' and reset = '0') then
      assert s = s_expected
        report "data = " &
          integer'image(CONV_INTEGER(data)) &
          "; s = " &
          integer'image(CONV_INTEGER(s)) &
          "; s_expected = " &
           integer'image(CONV_INTEGER(s_expected));
      if (s /= s_expected) then
        errors := errors + 1;
      end if;
      vectornum := vectornum + 1;
      if (is_x(testvectors(vectornum))) then
        if (errors = 0) then
          report "Just kidding -- " &
                 integer'image(vectornum) &
                 " tests completed successfully."
                 severity failure;
        else
          report integer'image(vectornum) &
                 " tests completed, errors = " &
                 integer'image(errors)
                 severity failure;
        end if;
      end if;
    end if;
  end process;
end;
```

Option 2 (VHDL only):

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
use STD.TEXTIO.all;
use work.txt_util.all;

entity ex4_7_testbench is -- no inputs or outputs
end;

architecture sim of ex4_7_testbench is
  component seven_seg_decoder
  port(data:     in  STD_LOGIC_VECTOR(3 downto 0);
       segments: out STD_LOGIC_VECTOR(6 downto 0));
  end component;
  signal data: STD_LOGIC_VECTOR(3 downto 0);
  signal s:    STD_LOGIC_VECTOR(6 downto 0);
  signal clk, reset: STD_LOGIC;
  signal s_expected: STD_LOGIC_VECTOR(6 downto 0);
  constant MEMSIZE: integer := 10000;
  type tvarray is array(MEMSIZE downto 0) of
    STD_LOGIC_VECTOR(10 downto 0);
  signal testvectors: tvarray;
  shared variable vectornum, errors: integer;
begin
  -- instantiate device under test
  dut: seven_seg_decoder port map(data, s);

  -- generate clock
  process begin
    clk <= '1'; wait for 5 ns;
    clk <= '0'; wait for 5 ns;
  end process;

  -- at start of test, load vectors
  -- and pulse reset
  process is
    file tv: TEXT;
    variable i, j: integer;
    variable L: line;
    variable ch: character;
  begin
    -- read file of test vectors
    i := 0;
    FILE_OPEN(tv, "ex4_7.tv", READ_MODE);
    while not endfile(tv) loop
      readline(tv, L);
      for j in 10 downto 0 loop
        read(L, ch);
        if (ch = '_') then read(L, ch);
        end if;
        if (ch = '0') then
          testvectors(i)(j) <= '0';
        else testvectors(i)(j) <= '1';
        end if;
      end loop;
      i := i + 1;
    end loop;

    vectornum := 0; errors := 0;
    reset <= '1'; wait for 27 ns; reset <= '0';
    wait;
  end process;

  -- apply test vectors on rising edge of clk
  process (clk) begin
    if (clk'event and clk = '1') then

      data <= testvectors(vectornum)(10 downto 7)
        after 1 ns;
     s_expected <= testvectors(vectornum)(6 downto 0)
        after 1 ns;
    end if;
  end process;

  -- check results on falling edge of clk
  process (clk) begin
   if (clk'event and clk = '0' and reset = '0') then
      assert s = s_expected
        report "data = " & str(data) &
          "; s = " & str(s) &
          "; s_expected = " & str(s_expected);
      if (s /= s_expected) then
        errors := errors + 1;
      end if;
      vectornum := vectornum + 1;
      if (is_x(testvectors(vectornum))) then
        if (errors = 0) then
          report "Just kidding -- " &
                 integer'image(vectornum) &
                 " tests completed successfully."
                 severity failure;
        else
          report integer'image(vectornum) &
                 " tests completed, errors = " &
                 integer'image(errors)
                 severity failure;
        end if;
      end if;
    end if;
  end process;
end;
```

(*see Web site for file: txt_util.vhd*)

© 2015 Elsevier, Inc.

Exercise 4.8

SystemVerilog

```
module mux8
  #(parameter width = 4)
   (input logic  [width-1:0] d0, d1, d2, d3,
                             d4, d5, d6, d7,
    input logic  [2:0]       s,
    output logic [width-1:0] y);

   always_comb
     case (s)
       0: y = d0;
       1: y = d1;
       2: y = d2;
       3: y = d3;
       4: y = d4;
       5: y = d5;
       6: y = d6;
       7: y = d7;
     endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux8 is
  generic(width: integer := 4);
  port(d0,
       d1,
       d2,
       d3,
       d4,
       d5,
       d6,
       d7: in  STD_LOGIC_VECTOR(width-1 downto 0);
       s:  in  STD_LOGIC_VECTOR(2 downto 0);
        y:  out STD_LOGIC_VECTOR(width-1 downto 0));
end;

architecture synth of mux8 is
begin
  with s select y <=
    d0 when "000",
    d1 when "001",
    d2 when "010",
    d3 when "011",
    d4 when "100",
    d5 when "101",
    d6 when "110",
    d7 when others;
end;
```

Exercise 4.9

SystemVerilog

```
module ex4_9
   (input  logic a, b, c,
    output logic y);

   mux8 #(1) mux8_1(1'b1, 1'b0, 1'b0, 1'b1,
                    1'b1, 1'b1, 1'b0, 1'b0,
                    {a,b,c}, y);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_9 is
  port(a,
       b,
       c: in  STD_LOGIC;
       y: out STD_LOGIC_VECTOR(0 downto 0));
end;

architecture struct of ex4_9 is
  component mux8
    generic(width: integer);
  port(d0, d1, d2, d3, d4, d5, d6,
       d7:  in  STD_LOGIC_VECTOR(width-1 downto 0);
       s:   in  STD_LOGIC_VECTOR(2 downto 0);
       y:   out STD_LOGIC_VECTOR(width-1 downto 0));
  end component;
  signal sel: STD_LOGIC_VECTOR(2 downto 0);
begin
  sel <= a & b & c;

  mux8_1: mux8 generic map(1)
               port map("1", "0", "0", "1",
                        "1", "1", "0", "0",
                        sel, y);
end;
```

© 2015 Elsevier, Inc.

Exercise 4.10

SystemVerilog

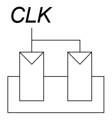
```
module ex4_10
   (input  logic a, b, c,
    output logic y);

   mux4 #(1) mux4_1( ~c, c, 1'b1, 1'b0, {a, b}, y);
endmodule

module mux4
  #(parameter width = 4)
   (input  logic [width-1:0] d0, d1, d2, d3,
    input  logic [1:0]       s,
    output logic [width-1:0] y);

   always_comb
     case (s)
       0: y = d0;
       1: y = d1;
       2: y = d2;
       3: y = d3;
     endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_10 is
  port(a,
       b,
       c: in  STD_LOGIC;
       y: out STD_LOGIC_VECTOR(0 downto 0));
end;

architecture struct of ex4_10 is
  component mux4
    generic(width: integer);
    port(d0, d1, d2,
       d3: in  STD_LOGIC_VECTOR(width-1 downto 0);
          s: in  STD_LOGIC_VECTOR(1 downto 0);
         y: out STD_LOGIC_VECTOR(width-1 downto 0));
  end component;
  signal cb:      STD_LOGIC_VECTOR(0 downto 0);
  signal c_vect:  STD_LOGIC_VECTOR(0 downto 0);
  signal sel:     STD_LOGIC_VECTOR(1 downto 0);
begin
  c_vect(0) <= c;
  cb(0) <= not c;
  sel <= (a & b);
  mux4_1: mux4 generic map(1)
              port map(cb, c_vect, "1", "0", sel, y);
end;

library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux4 is
  generic(width: integer := 4);
  port(d0,
       d1,
       d2,
       d3: in  STD_LOGIC_VECTOR(width-1 downto 0);
       s:  in  STD_LOGIC_VECTOR(1 downto 0);
       y:  out STD_LOGIC_VECTOR(width-1 downto 0));
end;

architecture synth of mux4 is
begin
  with s select y <=
    d0 when "00",
    d1 when "01",
    d2 when "10",
    d3 when others;
end;
```

Exercise 4.11

A shift register with feedback, shown below, cannot be correctly described with blocking assignments.



Exercise 4.12

SystemVerilog

```
module priority(input  logic [7:0] a,
                output logic [7:0] y);

  always_comb
    casez (a)
      8'b1???????: y = 8'b10000000;
      8'b01??????: y = 8'b01000000;
      8'b001?????: y = 8'b00100000;
      8'b0001????: y = 8'b00010000;
      8'b00001???: y = 8'b00001000;
      8'b000001??: y = 8'b00000100;
      8'b0000001?: y = 8'b00000010;
      8'b00000001: y = 8'b00000001;
      default:     y = 8'b00000000;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity priority is
  port(a:      in  STD_LOGIC_VECTOR(7 downto 0);
       y: out STD_LOGIC_VECTOR(7 downto 0));
end;

architecture synth of priority is
begin
  process(all) begin
    if    a(7) = '1' then y <= "10000000";
    elsif a(6) = '1' then y <= "01000000";
    elsif a(5) = '1' then y <= "00100000";
    elsif a(4) = '1' then y <= "00010000";
    elsif a(3) = '1' then y <= "00001000";
    elsif a(2) = '1' then y <= "00000100";
    elsif a(1) = '1' then y <= "00000010";
    elsif a(0) = '1' then y <= "00000001";
    else                  y <= "00000000";
    end if;
  end process;
end;
```

Exercise 4.13

© 2015 Elsevier, Inc.

SystemVerilog

```
module decoder2_4(input  logic [1:0] a,
                  output logic [3:0] y);
  always_comb
    case (a)
      2'b00: y = 4'b0001;
      2'b01: y = 4'b0010;
      2'b10: y = 4'b0100;
      2'b11: y = 4'b1000;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity decoder2_4 is
  port(a: in  STD_LOGIC_VECTOR(1 downto 0);
       y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of decoder2_4 is
begin
  process(all) begin
    case a is
      when "00"   => y <= "0001";
      when "01"   => y <= "0010";
      when "10"   => y <= "0100";
      when "11"   => y <= "1000";
      when others => y <= "0000";
    end case;
  end process;
end;
```

Exercise 4.14

SystemVerilog

```
module decoder6_64(input  logic [5:0]  a,
                   output logic [63:0] y);

  logic [11:0] y2_4;

  decoder2_4 dec0(a[1:0], y2_4[3:0]);
  decoder2_4 dec1(a[3:2], y2_4[7:4]);
  decoder2_4 dec2(a[5:4], y2_4[11:8]);

  assign y[0] = y2_4[0] & y2_4[4] & y2_4[8];
  assign y[1] = y2_4[1] & y2_4[4] & y2_4[8];
  assign y[2] = y2_4[2] & y2_4[4] & y2_4[8];
  assign y[3] = y2_4[3] & y2_4[4] & y2_4[8];
  assign y[4] = y2_4[0] & y2_4[5] & y2_4[8];
  assign y[5] = y2_4[1] & y2_4[5] & y2_4[8];
  assign y[6] = y2_4[2] & y2_4[5] & y2_4[8];
  assign y[7] = y2_4[3] & y2_4[5] & y2_4[8];
  assign y[8] = y2_4[0] & y2_4[6] & y2_4[8];
  assign y[9] = y2_4[1] & y2_4[6] & y2_4[8];
  assign y[10] = y2_4[2] & y2_4[6] & y2_4[8];
  assign y[11] = y2_4[3] & y2_4[6] & y2_4[8];
  assign y[12] = y2_4[0] & y2_4[7] & y2_4[8];
  assign y[13] = y2_4[1] & y2_4[7] & y2_4[8];
  assign y[14] = y2_4[2] & y2_4[7] & y2_4[8];
  assign y[15] = y2_4[3] & y2_4[7] & y2_4[8];
  assign y[16] = y2_4[0] & y2_4[4] & y2_4[9];
  assign y[17] = y2_4[1] & y2_4[4] & y2_4[9];
  assign y[18] = y2_4[2] & y2_4[4] & y2_4[9];
  assign y[19] = y2_4[3] & y2_4[4] & y2_4[9];
  assign y[20] = y2_4[0] & y2_4[5] & y2_4[9];
  assign y[21] = y2_4[1] & y2_4[5] & y2_4[9];
  assign y[22] = y2_4[2] & y2_4[5] & y2_4[9];
  assign y[23] = y2_4[3] & y2_4[5] & y2_4[9];
  assign y[24] = y2_4[0] & y2_4[6] & y2_4[9];
  assign y[25] = y2_4[1] & y2_4[6] & y2_4[9];
  assign y[26] = y2_4[2] & y2_4[6] & y2_4[9];
  assign y[27] = y2_4[3] & y2_4[6] & y2_4[9];
  assign y[28] = y2_4[0] & y2_4[7] & y2_4[9];
  assign y[29] = y2_4[1] & y2_4[7] & y2_4[9];
  assign y[30] = y2_4[2] & y2_4[7] & y2_4[9];
  assign y[31] = y2_4[3] & y2_4[7] & y2_4[9];
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity decoder6_64 is
  port(a: in  STD_LOGIC_VECTOR(5 downto 0);
       y: out STD_LOGIC_VECTOR(63 downto 0));
end;

architecture struct of decoder6_64 is
  component decoder2_4
    port(a:  in  STD_LOGIC_VECTOR(1 downto 0);
         y:  out STD_LOGIC_VECTOR(3 downto 0));
  end component;
  signal y2_4: STD_LOGIC_VECTOR(11 downto 0);
begin
  dec0: decoder2_4 port map(a(1 downto 0),
                            y2_4(3 downto 0));
  dec1: decoder2_4 port map(a(3 downto 2),
                            y2_4(7 downto 4));
  dec2: decoder2_4 port map(a(5 downto 4),
                            y2_4(11 downto 8));

  y(0)  <= y2_4(0) and y2_4(4) and y2_4(8);
  y(1)  <= y2_4(1) and y2_4(4) and y2_4(8);
  y(2)  <= y2_4(2) and y2_4(4) and y2_4(8);
  y(3)  <= y2_4(3) and y2_4(4) and y2_4(8);
  y(4)  <= y2_4(0) and y2_4(5) and y2_4(8);
  y(5)  <= y2_4(1) and y2_4(5) and y2_4(8);
  y(6)  <= y2_4(2) and y2_4(5) and y2_4(8);
  y(7)  <= y2_4(3) and y2_4(5) and y2_4(8);
  y(8)  <= y2_4(0) and y2_4(6) and y2_4(8);
  y(9)  <= y2_4(1) and y2_4(6) and y2_4(8);
  y(10)  <= y2_4(2) and y2_4(6) and y2_4(8);
  y(11)  <= y2_4(3) and y2_4(6) and y2_4(8);
  y(12)  <= y2_4(0) and y2_4(7) and y2_4(8);
  y(13)  <= y2_4(1) and y2_4(7) and y2_4(8);
  y(14)  <= y2_4(2) and y2_4(7) and y2_4(8);
  y(15)  <= y2_4(3) and y2_4(7) and y2_4(8);
  y(16)  <= y2_4(0) and y2_4(4) and y2_4(9);
  y(17)  <= y2_4(1) and y2_4(4) and y2_4(9);
  y(18)  <= y2_4(2) and y2_4(4) and y2_4(9);
  y(19)  <= y2_4(3) and y2_4(4) and y2_4(9);
  y(20)  <= y2_4(0) and y2_4(5) and y2_4(9);
  y(21)  <= y2_4(1) and y2_4(5) and y2_4(9);
  y(22)  <= y2_4(2) and y2_4(5) and y2_4(9);
  y(23)  <= y2_4(3) and y2_4(5) and y2_4(9);
  y(24)  <= y2_4(0) and y2_4(6) and y2_4(9);
  y(25)  <= y2_4(1) and y2_4(6) and y2_4(9);
  y(26)  <= y2_4(2) and y2_4(6) and y2_4(9);
  y(27)  <= y2_4(3) and y2_4(6) and y2_4(9);
  y(28)  <= y2_4(0) and y2_4(7) and y2_4(9);
  y(29)  <= y2_4(1) and y2_4(7) and y2_4(9);
  y(30)  <= y2_4(2) and y2_4(7) and y2_4(9);
  y(31)  <= y2_4(3) and y2_4(7) and y2_4(9);
```

(continued on next page)

© 2015 Elsevier, Inc.

(continued from previous page)

SystemVerilog

```
  assign y[32] = y2_4[0] & y2_4[4] & y2_4[10];
  assign y[33] = y2_4[1] & y2_4[4] & y2_4[10];
  assign y[34] = y2_4[2] & y2_4[4] & y2_4[10];
  assign y[35] = y2_4[3] & y2_4[4] & y2_4[10];
  assign y[36] = y2_4[0] & y2_4[5] & y2_4[10];
  assign y[37] = y2_4[1] & y2_4[5] & y2_4[10];
  assign y[38] = y2_4[2] & y2_4[5] & y2_4[10];
  assign y[39] = y2_4[3] & y2_4[5] & y2_4[10];
  assign y[40] = y2_4[0] & y2_4[6] & y2_4[10];
  assign y[41] = y2_4[1] & y2_4[6] & y2_4[10];
  assign y[42] = y2_4[2] & y2_4[6] & y2_4[10];
  assign y[43] = y2_4[3] & y2_4[6] & y2_4[10];
  assign y[44] = y2_4[0] & y2_4[7] & y2_4[10];
  assign y[45] = y2_4[1] & y2_4[7] & y2_4[10];
  assign y[46] = y2_4[2] & y2_4[7] & y2_4[10];
  assign y[47] = y2_4[3] & y2_4[7] & y2_4[10];
  assign y[48] = y2_4[0] & y2_4[4] & y2_4[11];
  assign y[49] = y2_4[1] & y2_4[4] & y2_4[11];
  assign y[50] = y2_4[2] & y2_4[4] & y2_4[11];
  assign y[51] = y2_4[3] & y2_4[4] & y2_4[11];
  assign y[52] = y2_4[0] & y2_4[5] & y2_4[11];
  assign y[53] = y2_4[1] & y2_4[5] & y2_4[11];
  assign y[54] = y2_4[2] & y2_4[5] & y2_4[11];
  assign y[55] = y2_4[3] & y2_4[5] & y2_4[11];
  assign y[56] = y2_4[0] & y2_4[6] & y2_4[11];
  assign y[57] = y2_4[1] & y2_4[6] & y2_4[11];
  assign y[58] = y2_4[2] & y2_4[6] & y2_4[11];
  assign y[59] = y2_4[3] & y2_4[6] & y2_4[11];
  assign y[60] = y2_4[0] & y2_4[7] & y2_4[11];
  assign y[61] = y2_4[1] & y2_4[7] & y2_4[11];
  assign y[62] = y2_4[2] & y2_4[7] & y2_4[11];
  assign y[63] = y2_4[3] & y2_4[7] & y2_4[11];
endmodule
```

VHDL

```
  y(32) <= y2_4(0) and y2_4(4) and y2_4(10);
  y(33) <= y2_4(1) and y2_4(4) and y2_4(10);
  y(34) <= y2_4(2) and y2_4(4) and y2_4(10);
  y(35) <= y2_4(3) and y2_4(4) and y2_4(10);
  y(36) <= y2_4(0) and y2_4(5) and y2_4(10);
  y(37) <= y2_4(1) and y2_4(5) and y2_4(10);
  y(38) <= y2_4(2) and y2_4(5) and y2_4(10);
  y(39) <= y2_4(3) and y2_4(5) and y2_4(10);
  y(40) <= y2_4(0) and y2_4(6) and y2_4(10);
  y(41) <= y2_4(1) and y2_4(6) and y2_4(10);
  y(42) <= y2_4(2) and y2_4(6) and y2_4(10);
  y(43) <= y2_4(3) and y2_4(6) and y2_4(10);
  y(44) <= y2_4(0) and y2_4(7) and y2_4(10);
  y(45) <= y2_4(1) and y2_4(7) and y2_4(10);
  y(46) <= y2_4(2) and y2_4(7) and y2_4(10);
  y(47) <= y2_4(3) and y2_4(7) and y2_4(10);
  y(48) <= y2_4(0) and y2_4(4) and y2_4(11);
  y(49) <= y2_4(1) and y2_4(4) and y2_4(11);
  y(50) <= y2_4(2) and y2_4(4) and y2_4(11);
  y(51) <= y2_4(3) and y2_4(4) and y2_4(11);
  y(52) <= y2_4(0) and y2_4(5) and y2_4(11);
  y(53) <= y2_4(1) and y2_4(5) and y2_4(11);
  y(54) <= y2_4(2) and y2_4(5) and y2_4(11);
  y(55) <= y2_4(3) and y2_4(5) and y2_4(11);
  y(56) <= y2_4(0) and y2_4(6) and y2_4(11);
  y(57) <= y2_4(1) and y2_4(6) and y2_4(11);
  y(58) <= y2_4(2) and y2_4(6) and y2_4(11);
  y(59) <= y2_4(3) and y2_4(6) and y2_4(11);
  y(60) <= y2_4(0) and y2_4(7) and y2_4(11);
  y(61) <= y2_4(1) and y2_4(7) and y2_4(11);
  y(62) <= y2_4(2) and y2_4(7) and y2_4(11);
  y(63) <= y2_4(3) and y2_4(7) and y2_4(11);
end;
```

Exercise 4.15

(a) $Y = AC + \overline{A}\overline{B}C$

SystemVerilog

```
module ex4_15a(input  logic a, b, c,
               output logic y);

  assign y = (a & c) | (~a & ~b & c);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_15a is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;

architecture behave of ex4_15a is
begin
  y <= (not a and not b and c) or (not b and c);
end;
```

(b) $Y = \overline{A}\overline{B} + \overline{A}B\overline{C} + \overline{(A + \overline{C})}$

SystemVerilog

```
module ex4_15b(input  logic a, b, c,
               output logic y);

  assign y = (~a & ~b) | (~a & b & ~c) | ~(a | ~c);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_15b is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
end;

architecture behave of ex4_15b is
begin
  y <= ((not a) and (not b)) or ((not a) and b and
         (not c)) or (not(a or (not c)));
end;
```

(c) $Y = \overline{A}\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C} + A\overline{B}C\overline{D} + ABD + \overline{A}\overline{B}C\overline{D} + B\overline{C}D + \overline{A}$

SystemVerilog

```
module ex4_15c(input  logic a, b, c, d,
               output logic y);

  assign y = (~a & ~b & ~c & ~d) | (a & ~b & ~c) |
             (a & ~b & c & ~d) | (a & b & d) |
             (~a & ~b & c & ~d) | (b & ~c & d) | ~a;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_15c is
  port(a, b, c, d: in  STD_LOGIC;
       y:          out STD_LOGIC);
end;

architecture behave of ex4_15c is
begin
 y <= ((not a) and (not b) and (not c) and (not d)) or
        (a and (not b) and (not c)) or
        (a and (not b) and c and (not d)) or
        (a and b and d) or
        ((not a) and (not b) and c and (not d)) or
        (b and (not c) and d) or (not a);
end;
```

© 2015 Elsevier, Inc.

Exercise 4.16

SystemVerilog

```
module ex4_16(input  logic a, b, c, d, e,
              output logic y);

  assign y = ~(~(~(a & b) & ~(c & d)) & e);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_16 is
  port(a, b, c, d, e: in  STD_LOGIC;
       y:             out STD_LOGIC);
end;

architecture behave of ex4_16 is
begin
  y <= not((not((not(a and b)) and
                (not(c and d)))) and e);

end;
```

Exercise 4.17

SystemVerilog

```
module ex4_17(input  logic a, b, c, d, e, f, g
              output logic y);

  logic n1, n2, n3, n4, n5;

  assign n1 = ~(a & b & c);
  assign n2 = ~(n1 & d);
  assign n3 = ~(f & g);
  assign n4 = ~(n3 | e);
  assign n5 = ~(n2 | n4);
  assign y = ~(n5 & n5);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_17 is
  port(a, b, c, d, e, f, g: in  STD_LOGIC;
       y:            out STD_LOGIC);
end;

architecture synth of ex4_17 is
  signal n1, n2, n3, n4, n5: STD_LOGIC;
begin
  n1 <= not(a and b and c);
  n2 <= not(n1 and d);
  n3 <= not(f and g);
  n4 <= not(n3 or e);
  n5 <= not(n2 or n4);
  y <= not (n5 or n5);
end;
```

Exercise 4.18

© 2015 Elsevier, Inc.

Verilog

```
module ex4_18(input  logic a, b, c, d,
              output logic y);

  always_comb
    casez ({a, b, c, d})
      // note: outputs cannot be assigned don't care
       0: y = 1'b0;
       1: y = 1'b0;
       2: y = 1'b0;
       3: y = 1'b0;
       4: y = 1'b0;
       5: y = 1'b0;
       6: y = 1'b0;
       7: y = 1'b0;
       8: y = 1'b1;
       9: y = 1'b0;
      10: y = 1'b0;
      11: y = 1'b1;
      12: y = 1'b1;
      13: y = 1'b1;
      14: y = 1'b0;
      15: y = 1'b1;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_18 is
  port(a, b, c, d: in  STD_LOGIC;
       y:          out STD_LOGIC);
end;

architecture synth of ex4_17 is
signal vars: STD_LOGIC_VECTOR(3 downto 0);
begin
  vars <= (a & b & c & d);
  process(all) begin
    case vars is
      -- note: outputs cannot be assigned don't care
      when X"0" => y <= '0';
      when X"1" => y <= '0';
      when X"2" => y <= '0';
      when X"3" => y <= '0';
      when X"4" => y <= '0';
      when X"5" => y <= '0';
      when X"6" => y <= '0';
      when X"7" => y <= '0';
      when X"8" => y <= '1';
      when X"9" => y <= '0';
      when X"A" => y <= '0';
      when X"B" => y <= '1';
      when X"C" => y <= '1';
      when X"D" => y <= '1';
      when X"E" => y <= '0';
      when X"F" => y <= '1';
      when others => y <= '0';--should never happen
    end case;
  end process;
end;
```

Exercise 4.19

SystemVerilog

```
module ex4_18(input  logic [3:0] a,
              output logic       p, d);

  always_comb
    case (a)
       0: {p, d} = 2'b00;
       1: {p, d} = 2'b00;
       2: {p, d} = 2'b10;
       3: {p, d} = 2'b11;
       4: {p, d} = 2'b00;
       5: {p, d} = 2'b10;
       6: {p, d} = 2'b01;
       7: {p, d} = 2'b10;
       8: {p, d} = 2'b00;
       9: {p, d} = 2'b01;
      10: {p, d} = 2'b00;
      11: {p, d} = 2'b10;
      12: {p, d} = 2'b01;
      13: {p, d} = 2'b10;
      14: {p, d} = 2'b00;
      15: {p, d} = 2'b01;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_18 is
  port(a:    in  STD_LOGIC_VECTOR(3 downto 0);
       p, d: out STD_LOGIC);
end;

architecture synth of ex4_18 is
signal vars: STD_LOGIC_VECTOR(1 downto 0);
begin
  p <= vars(1);
  d <= vars(0);
  process(all) begin
    case a is
      when X"0"   => vars <= "00";
      when X"1"   => vars <= "00";
      when X"2"   => vars <= "10";
      when X"3"   => vars <= "11";
      when X"4"   => vars <= "00";
      when X"5"   => vars <= "10";
      when X"6"   => vars <= "01";
      when X"7"   => vars <= "10";
      when X"8"   => vars <= "00";
      when X"9"   => vars <= "01";
      when X"A"   => vars <= "00";
      when X"B"   => vars <= "10";
      when X"C"   => vars <= "01";
      when X"D"   => vars <= "10";
      when X"E"   => vars <= "00";
      when X"F"   => vars <= "01";
      when others => vars <= "00";
    end case;
  end process;
end;
```

© 2015 Elsevier, Inc.

Exercise 4.20

SystemVerilog

```
module priority_encoder(input  logic [7:0] a,
                        output logic [2:0] y,
                        output logic       none);

  always_comb
    casez (a)
      8'b00000000: begin y = 3'd0;  none = 1'b1; end
      8'b00000001: begin y = 3'd0;  none = 1'b0; end
      8'b0000001?: begin y = 3'd1;  none = 1'b0; end
      8'b000001??: begin y = 3'd2;  none = 1'b0; end
      8'b00001???: begin y = 3'd3;  none = 1'b0; end
      8'b0001????: begin y = 3'd4;  none = 1'b0; end
      8'b001?????: begin y = 3'd5;  none = 1'b0; end
      8'b01??????: begin y = 3'd6;  none = 1'b0; end
      8'b1???????: begin y = 3'd7;  none = 1'b0; end
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity priority_encoder is
  port(a:    in  STD_LOGIC_VECTOR(7 downto 0);
       y:    out STD_LOGIC_VECTOR(2 downto 0);
       none: out STD_LOGIC);
end;

architecture synth of priority_encoder is
begin
  process(all) begin
    case? a is
      when "00000000" => y <= "000"; none <= '1';
      when "00000001" => y <= "000"; none <= '0';
      when "0000001-" => y <= "001"; none <= '0';
      when "000001--" => y <= "010"; none <= '0';
      when "00001---" => y <= "011"; none <= '0';
      when "0001----" => y <= "100"; none <= '0';
      when "001-----" => y <= "101"; none <= '0';
      when "01------" => y <= "110"; none <= '0';
      when "1-------" => y <= "111"; none <= '0';
      when others     => y <= "000"; none <= '0';
    end case?;
  end process;
end;
```

Exercise 4.21

SystemVerilog

```
module priority_encoder2(input  logic [7:0] a,
                         output logic [2:0] y, z,
                         output logic       none);

  always_comb
  begin
    casez (a)
       8'b00000000: begin y = 3'd0; none = 1'b1; end
       8'b00000001: begin y = 3'd0; none = 1'b0; end
       8'b0000001?: begin y = 3'd1; none = 1'b0; end
       8'b000001??: begin y = 3'd2; none = 1'b0; end
       8'b00001???: begin y = 3'd3; none = 1'b0; end
       8'b0001????: begin y = 3'd4; none = 1'b0; end
       8'b001?????: begin y = 3'd5; none = 1'b0; end
       8'b01??????: begin y = 3'd6; none = 1'b0; end
       8'b1???????: begin y = 3'd7; none = 1'b0; end
    endcase

    casez (a)
      8'b00000011: z = 3'b000;
      8'b00000101: z = 3'b000;
      8'b00001001: z = 3'b000;
      8'b00010001: z = 3'b000;
      8'b00100001: z = 3'b000;
      8'b01000001: z = 3'b000;
      8'b10000001: z = 3'b000;
      8'b0000011?: z = 3'b001;
      8'b0000101?: z = 3'b001;
      8'b0001001?: z = 3'b001;
      8'b0010001?: z = 3'b001;
      8'b0100001?: z = 3'b001;
      8'b1000001?: z = 3'b001;
      8'b000011??: z = 3'b010;
      8'b000101??: z = 3'b010;
      8'b001001??: z = 3'b010;
      8'b010001??: z = 3'b010;
      8'b100001??: z = 3'b010;
      8'b00011???: z = 3'b011;
      8'b00101???: z = 3'b011;
      8'b01001???: z = 3'b011;
      8'b10001???: z = 3'b011;
      8'b0011????: z = 3'b100;
      8'b0101????: z = 3'b100;
      8'b1001????: z = 3'b100;
      8'b011?????: z = 3'b101;
      8'b101?????: z = 3'b101;
      8'b11??????: z = 3'b110;
      default:     z = 3'b000;
  end
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity priority_encoder2 is
  port(a:    in  STD_LOGIC_VECTOR(7 downto 0);
       y, z: out STD_LOGIC_VECTOR(2 downto 0);
       none: out STD_LOGIC);
end;

architecture synth of priority_encoder is
begin
  process(all) begin
    case? a is
      when "00000000" => y <= "000"; none <= '1';
      when "00000001" => y <= "000"; none <= '0';
      when "0000001-" => y <= "001"; none <= '0';
      when "000001--" => y <= "010"; none <= '0';
      when "00001---" => y <= "011"; none <= '0';
      when "0001----" => y <= "100"; none <= '0';
      when "001-----" => y <= "101"; none <= '0';
      when "01------" => y <= "110"; none <= '0';
      when "1-------" => y <= "111"; none <= '0';
      when others     => y <= "000"; none <= '0';
    end case?;
    case? a is
      when "00000011" => z <= "000";
      when "00000101" => z <= "000";
      when "00001001" => z <= "000";
      when "00001001" => z <= "000";
      when "00010001" => z <= "000";
      when "00100001" => z <= "000";
      when "01000001" => z <= "000";
      when "10000001" => z <= "000";
      when "0000011-" => z <= "001";
      when "0000101-" => z <= "001";
      when "0001001-" => z <= "001";
      when "0010001-" => z <= "001";
      when "0100001-" => z <= "001";
      when "1000001-" => z <= "001";
      when "000011--" => z <= "010";
      when "000101--" => z <= "010";
      when "001001--" => z <= "010";
      when "010001--" => z <= "010";
      when "100001--" => z <= "010";
      when "00011---" => z <= "011";
      when "00101---" => z <= "011";
      when "01001---" => z <= "011";
      when "10001---" => z <= "011";
      when "0011----" => z <= "100";
      when "0101----" => z <= "100";
      when "1001----" => z <= "100";
      when "011-----" => z <= "101";
      when "101-----" => z <= "101";
      when "11------" => z <= "110";
      when others     => z <= "000";
    end case?;
  end process;
end;
```

© 2015 Elsevier, Inc.

Exercise 4.22

SystemVerilog

```
module thermometer(input  logic [2:0] a,
                   output logic [6:0] y);

  always_comb
    case (a)
       0: y = 7'b0000000;
       1: y = 7'b0000001;
       2: y = 7'b0000011;
       3: y = 7'b0000111;
       4: y = 7'b0001111;
       5: y = 7'b0011111;
       6: y = 7'b0111111;
       7: y = 7'b1111111;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity thermometer is
  port(a:    in  STD_LOGIC_VECTOR(2 downto 0);
       y:    out STD_LOGIC_VECTOR(6 downto 0));
end;

architecture synth of thermometer is
begin
  process(all) begin
    case a is
      when "000"  => y <= "0000000";
      when "001"  => y <= "0000001";
      when "010"  => y <= "0000011";
      when "011"  => y <= "0000111";
      when "100"  => y <= "0001111";
      when "101"  => y <= "0011111";
      when "110"  => y <= "0111111";
      when "111"  => y <= "1111111";
      when others => y <= "0000000";
    end case;
  end process;
end;
```

Exercise 4.23

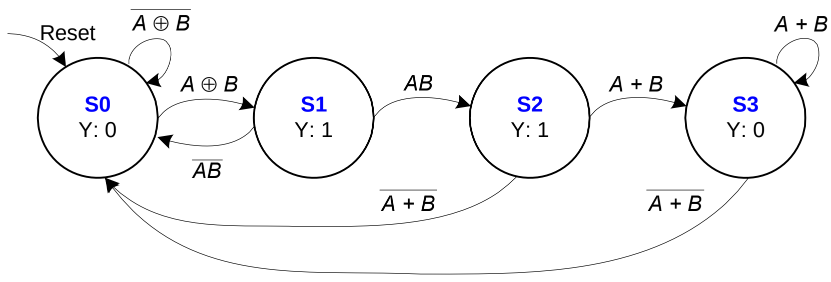
SystemVerilog

```
module month31days(input  logic [3:0] month,
                   output logic       y);

  always_comb
    casez (month)
       1:       y = 1'b1;
       2:       y = 1'b0;
       3:       y = 1'b1;
       4:       y = 1'b0;
       5:       y = 1'b1;
       6:       y = 1'b0;
       7:       y = 1'b1;
       8:       y = 1'b1;
       9:       y = 1'b0;
       10:      y = 1'b1;
       11:      y = 1'b0;
       12:      y = 1'b1;
       default: y = 1'b0;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity month31days is
  port(a:    in  STD_LOGIC_VECTOR(3 downto 0);
       y:    out STD_LOGIC);
end;

architecture synth of month31days is
begin
  process(all) begin
    case a is
      when X"1"    => y <= '1';
      when X"2"    => y <= '0';
      when X"3"    => y <= '1';
      when X"4"    => y <= '0';
      when X"5"    => y <= '1';
      when X"6"    => y <= '0';
      when X"7"    => y <= '1';
      when X"8"    => y <= '1';
      when X"9"    => y <= '0';
      when X"A"    => y <= '1';
      when X"B"    => y <= '0';
      when X"C"    => y <= '1';
      when others => y <= '0';
    end case;
  end process;
end;
```

Exercise 4.24



Exercise 4.25

© 2015 Elsevier, Inc.

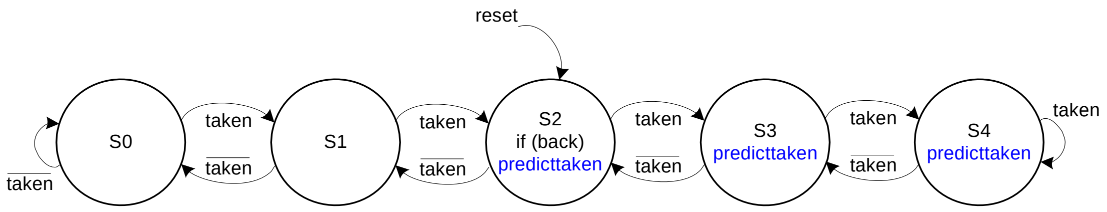


FIGURE 4.1 State transition diagram for Exercise 4.25

Exercise 4.26

SystemVerilog

```
module srlatch(input  logic s, r,
               output logic q, qbar);

  always_comb
    case ({s,r})
      2'b01: {q, qbar} = 2'b01;
      2'b10: {q, qbar} = 2'b10;
      2'b11: {q, qbar} = 2'b00;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity srlatch is
  port(s, r:      in  STD_LOGIC;
       q, qbar:   out STD_LOGIC);
end;

architecture synth of srlatch is
signal qqbar: STD_LOGIC_VECTOR(1 downto 0);
signal sr: STD_LOGIC_VECTOR(1 downto 0);
begin
  q <= qqbar(1);
  qbar <= qqbar(0);
  sr <= s & r;
  process(all) begin
    if s = '1' and r = '0'
      then qqbar <= "10";
    elsif s = '0' and r = '1'
      then qqbar <= "01";
    elsif s = '1' and r = '1'
      then qqbar <= "00";
    end if;
  end process;
end;
```

Exercise 4.27

SystemVerilog

```
module jkflop(input  logic j, k, clk,
              output logic q);

  always @(posedge clk)
    case ({j,k})
      2'b01: q <= 1'b0;
      2'b10: q <= 1'b1;
      2'b11: q <= ~q;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity jkflop is
  port(j, k, clk: in    STD_LOGIC;
       q:         inout STD_LOGIC);
end;

architecture synth of jkflop is
signal jk: STD_LOGIC_VECTOR(1 downto 0);
begin
  jk <= j & k;
  process(clk) begin
    if rising_edge(clk) then
      if j = '1' and k = '0'
        then q <= '1';
      elsif j = '0' and k = '1'
        then q <= '0';
      elsif j = '1' and k = '1'
        then q <= not q;
      end if;
    end if;
  end process;
end;
```

Exercise 4.28

SystemVerilog

```
module latch3_18(input  logic d, clk,
                 output logic q);

  logic n1, n2, clk_b;

  assign #1 n1 = clk & d;
  assign    clk_b = ~clk;
  assign #1 n2 = clk_b & q;
  assign #1 q = n1 | n2;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity latch3_18 is
  port(d, clk: in    STD_LOGIC;
       q:      inout STD_LOGIC);
end;

architecture synth of latch3_18 is
signal n1, clk_b, n2: STD_LOGIC;
begin
  n1 <= (clk and d) after 1 ns;
  clk_b <= (not clk);
  n2 <= (clk_b and q) after 1 ns;
  q <= (n1 or n2) after 1 ns;
end;
```

This circuit is in error with any delay in the inverter.

Exercise 4.29

© 2015 Elsevier, Inc.

SystemVerilog

```
module trafficFSM(input  logic clk, reset, ta, tb,
                  output logic [1:0] la, lb);

  typedef enum logic [1:0] {S0, S1, S2, S3}
    statetype;
  statetype [1:0] state, nextstate;

  parameter green  = 2'b00;
  parameter yellow = 2'b01;
  parameter red    = 2'b10;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: if (ta) nextstate = S0;
          else    nextstate = S1;
      S1:         nextstate = S2;
      S2: if (tb) nextstate = S2;
          else    nextstate = S3;
      S3:         nextstate = S0;
    endcase

  // Output Logic
  always_comb
    case (state)
      S0: {la, lb} = {green, red};
      S1: {la, lb} = {yellow, red};
      S2: {la, lb} = {red, green};
      S3: {la, lb} = {red, yellow};
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity trafficFSM is
  port(clk, reset, ta, tb: in  STD_LOGIC;
       la, lb: inout STD_LOGIC_VECTOR(1 downto 0));
end;

architecture behave of trafficFSM is
  type statetype is (S0, S1, S2, S3);
  signal state, nextstate: statetype;
  signal lalb: STD_LOGIC_VECTOR(3 downto 0);
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S0 => if ta then
                   nextstate <= S0;
                 else nextstate <= S1;
                 end if;
      when S1 => nextstate <= S2;
      when S2 => if tb then
                   nextstate <= S2;
                 else nextstate <= S3;
                 end if;
      when S3 => nextstate <= S0;
      when others =>  nextstate <= S0;
    end case;
  end process;

  -- output logic
  la <= lalb(3 downto 2);
  lb <= lalb(1 downto 0);
  process(all) begin
    case state is
      when S0 =>     lalb <= "0010";
      when S1 =>     lalb <= "0110";
      when S2 =>     lalb <= "1000";
      when S3 =>     lalb <= "1001";
      when others => lalb <= "1010";
    end case;
  end process;
end;
```

Exercise 4.30

Mode Module

SystemVerilog

```
module mode(input  logic clk, reset, p, r,
            output logic m);

  typedef enum logic {S0, S1} statetype;
  statetype state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: if (p) nextstate = S1;
          else   nextstate = S0;
      S1: if (r) nextstate = S0;
          else   nextstate = S1;
    endcase

  // Output Logic
  assign m = state;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mode is
  port(clk, reset, p, r: in  STD_LOGIC;
       m:                out STD_LOGIC);
end;

architecture synth of mode is
  type statetype is (S0, S1);
  signal state, nextstate: statetype;
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S0 => if p then
                        nextstate <= S1;
                 else nextstate <= S0;
                 end if;
      when S1 => if r then
                        nextstate <= S0;
                 else nextstate <= S1;
                 end if;
      when others =>  nextstate <= S0;
    end case;
  end process;

  -- output logic
  m <= '1' when state = S1 else '0';
end;
```

(continued on next page)

© 2015 Elsevier, Inc.

Lights Module

SystemVerilog

```
module lights(input  logic clk, reset, ta, tb, m,
              output logic [1:0] la, lb);

  typedef enum logic [1:0] {S0, S1, S2, S3}
    statetype;

  statetype [1:0] state, nextstate;

  parameter green  = 2'b00;
  parameter yellow = 2'b01;
  parameter red    = 2'b10;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: if (ta)     nextstate = S0;
          else        nextstate = S1;
      S1:             nextstate = S2;
      S2: if (tb | m) nextstate = S2;
          else        nextstate = S3;
      S3:             nextstate = S0;
    endcase

  // Output Logic
  always_comb
    case (state)
      S0: {la, lb} = {green, red};
      S1: {la, lb} = {yellow, red};
      S2: {la, lb} = {red, green};
      S3: {la, lb} = {red, yellow};
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity lights is
  port(clk, reset, ta, tb, m: in  STD_LOGIC;
       la, lb: out STD_LOGIC_VECTOR(1 downto 0));
end;

architecture synth of lights is
  type statetype is (S0, S1, S2, S3);
  signal state, nextstate: statetype;
  signal lalb: STD_LOGIC_VECTOR(3 downto 0);
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S0 => if ta then
                      nextstate <= S0;
                 else nextstate <= S1;
                 end if;
      when S1 =>      nextstate <= S2;
      when S2 => if ((tb or m) = '1') then
                      nextstate <= S2;
                 else nextstate <= S3;
                 end if;
      when S3 =>      nextstate <= S0;
      when others =>  nextstate <= S0;
    end case;
  end process;

  -- output logic
  la <= lalb(3 downto 2);
  lb <= lalb(1 downto 0);
  process(all) begin
    case state is
      when S0 =>     lalb <= "0010";
      when S1 =>     lalb <= "0110";
      when S2 =>     lalb <= "1000";
      when S3 =>     lalb <= "1001";
      when others => lalb <= "1010";
    end case;
  end process;
end;
```

(continued on next page)

Controller Module

SystemVerilog

```
module controller(input  logic clk, reset, p,
                                r, ta, tb,
                  output logic [1:0] la, lb);

  mode modefsm(clk, reset, p, r, m);
  lights lightsfsm(clk, reset, ta, tb, m, la, lb);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity controller is
  port(clk, reset: in  STD_LOGIC;
       p, r, ta:   in  STD_LOGIC;
       tb:         in  STD_LOGIC;
       la, lb: out STD_LOGIC_VECTOR(1 downto 0));
end;

architecture struct of controller is
  component mode
    port(clk, reset, p, r: in  STD_LOGIC;
         m:                out STD_LOGIC);
  end component;
  component lights
    port(clk, reset, ta, tb, m: in  STD_LOGIC;
         la, lb: out STD_LOGIC_VECTOR(1 downto 0));
  end component;

begin
  modefsm:   mode   port map(clk, reset, p, r, m);
  lightsfsm: lights port map(clk, reset, ta, tb,
                             m, la, lb);
end;
```

Exercise 4.31

© 2015 Elsevier, Inc.

SystemVerilog

```
module fig3_42(input  logic clk, a, b, c, d,
               output logic x, y);

  logic n1, n2;
  logic areg, breg, creg, dreg;

  always_ff @(posedge clk) begin
    areg <= a;
    breg <= b;
    creg <= c;
    dreg <= d;
    x <= n2;
    y <= ~(dreg | n2);
  end

  assign n1 = areg & breg;
  assign n2 = n1 | creg;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fig3_42 is
  port(clk, a, b, c, d: in  STD_LOGIC;
       x, y:            out STD_LOGIC);
end;

architecture synth of fig3_40 is
  signal n1, n2, areg, breg, creg, dreg: STD_LOGIC;
begin
  process(clk) begin
    if rising_edge(clk) then
      areg <= a;
      breg <= b;
      creg <= c;
      dreg <= d;
      x <= n2;
      y <= not (dreg or n2);
    end if;
  end process;

  n1 <= areg and breg;
  n2 <= n1 or creg;
end;
```

Exercise 4.32

SystemVerilog

```
module fig3_69(input  logic clk, reset, a, b,
               output logic q);
  typedef enum logic [1:0] {S0, S1, S2} statetype;
  statetype [1:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: if (a) nextstate = S1;
          else   nextstate = S0;
      S1: if (b) nextstate = S2;
          else   nextstate = S0;
      S2:        nextstate = S0;
      default:   nextstate = S0;
    endcase

  // Output Logic
  assign q = state[1];
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fig3_69 is
  port(clk, reset, a, b: in  STD_LOGIC;
       q:                out STD_LOGIC);
end;

architecture synth of fig3_69 is
  type statetype is (S0, S1, S2);
  signal state, nextstate: statetype;
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S0 => if a then
                      nextstate <= S1;
                 else nextstate <= S0;
                 end if;
      when S1 => if b then
                      nextstate <= S2;
                 else nextstate <= S0;
                 end if;
      when S2 =>      nextstate <= S0;
      when others =>  nextstate <= S0;
    end case;
  end process;

  -- output logic
  q <= '1' when state = S2 else '0';
end;
```

Exercise 4.33

© 2015 Elsevier, Inc.

SystemVerilog

```
module fig3_70(input  logic clk, reset, a, b,
               output logic q);
  typedef enum logic [1:0] {S0, S1, S2} statetype;
  statetype [1:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: if (a)     nextstate = S1;
          else       nextstate = S0;
      S1: if (b)     nextstate = S2;
          else       nextstate = S0;
      S2: if (a & b) nextstate = S2;
          else       nextstate = S0;
      default:       nextstate = S0;
    endcase

  // Output Logic
  always_comb
    case (state)
      S0:            q = 0;
      S1:            q = 0;
      S2: if (a & b) q = 1;
          else       q = 0;
      default:       q = 0;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fig3_70 is
  port(clk, reset, a, b: in  STD_LOGIC;
       q:                out STD_LOGIC);
end;

architecture synth of fig3_70 is
  type statetype is (S0, S1, S2);
  signal state, nextstate: statetype;
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S0 => if a then
                   nextstate <= S1;
                 else nextstate <= S0;
                 end if;
      when S1 => if b then
                   nextstate <= S2;
                 else nextstate <= S0;
                 end if;
      when S2 => if (a = '1' and b = '1') then
                   nextstate <= S2;
                 else nextstate <= S0;
                 end if;
      when others =>  nextstate <= S0;
    end case;
  end process;

  -- output logic
  q <= '1' when ( (state = S2) and
                  (a = '1' and b = '1'))
           else '0';
end;
```

Exercise 4.34

SystemVerilog

```
module ex4_34(input  logic clk, reset, ta, tb,
              output logic [1:0] la, lb);
  typedef enum logic [2:0] {S0, S1, S2, S3, S4, S5}
    statetype;
  statetype [2:0] state, nextstate;

  parameter green  = 2'b00;
  parameter yellow = 2'b01;
  parameter red    = 2'b10;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: if (ta) nextstate = S0;
          else    nextstate = S1;
      S1:         nextstate = S2;
      S2:         nextstate = S3;
      S3: if (tb) nextstate = S3;
          else    nextstate = S4;
      S4:         nextstate = S5;
      S5:         nextstate = S0;
    endcase

  // Output Logic
  always_comb
    case (state)
      S0: {la, lb} = {green, red};
      S1: {la, lb} = {yellow, red};
      S2: {la, lb} = {red, red};
      S3: {la, lb} = {red, green};
      S4: {la, lb} = {red, yellow};
      S5: {la, lb} = {red, red};
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_34 is
  port(clk, reset, ta, tb: in  STD_LOGIC;
       la, lb: out STD_LOGIC_VECTOR(1 downto 0));
end;

architecture synth of ex4_34 is
  type statetype is (S0, S1, S2, S3, S4, S5);
  signal state, nextstate: statetype;
  signal lalb: STD_LOGIC_VECTOR(3 downto 0);
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S0 => if ta = '1' then
                      nextstate <= S0;
                 else nextstate <= S1;
                 end if;
      when S1 =>      nextstate <= S2;
      when S2 =>      nextstate <= S3;
      when S3 => if tb = '1' then
                      nextstate <= S3;
                 else nextstate <= S4;
                 end if;
      when S4 =>      nextstate <= S5;
      when S5 =>      nextstate <= S0;
      when others =>  nextstate <= S0;
    end case;
  end process;

  -- output logic
  la <= lalb(3 downto 2);
  lb <= lalb(1 downto 0);
  process(all) begin
    case state is
      when S0 =>     lalb <= "0010";
      when S1 =>     lalb <= "0110";
      when S2 =>     lalb <= "1010";
      when S3 =>     lalb <= "1000";
      when S4 =>     lalb <= "1001";
      when S5 =>     lalb <= "1010";
      when others => lalb <= "1010";
    end case;
  end process;
end;
```

© 2015 Elsevier, Inc.

Exercise 4.35

SystemVerilog

```
module daughterfsm(input  logic clk, reset, a,
                   output logic smile);
  typedef enum logic [1:0] {S0, S1, S2, S3, S4}
    statetype;
  statetype [2:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: if (a) nextstate = S1;
          else   nextstate = S0;
      S1: if (a) nextstate = S2;
          else   nextstate = S0;
      S2: if (a) nextstate = S4;
          else   nextstate = S3;
      S3: if (a) nextstate = S1;
          else   nextstate = S0;
      S4: if (a) nextstate = S4;
          else   nextstate = S3;
      default:   nextstate = S0;
     endcase

  // Output Logic
  assign smile = ((state == S3) & a) |
                 ((state == S4) & ~a);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity daughterfsm is
  port(clk, reset, a: in  STD_LOGIC;
       smile:         out STD_LOGIC);
end;

architecture synth of daughterfsm is
  type statetype is (S0, S1, S2, S3, S4);
  signal state, nextstate: statetype;
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S0 => if a then
                     nextstate <= S1;
                 else nextstate <= S0;
                 end if;
      when S1 => if a then
                     nextstate <= S2;
                 else nextstate <= S0;
                 end if;
      when S2 => if a then
                     nextstate <= S4;
                 else nextstate <= S3;
                 end if;
      when S3 => if a then
                     nextstate <= S1;
                 else nextstate <= S0;
                 end if;
      when S4 => if a then
                     nextstate <= S4;
                 else nextstate <= S3;
                 end if;
      when others =>  nextstate <= S0;
    end case;
  end process;

  -- output logic
  smile <= '1' when ( ((state = S3) and (a = '1')) or
                      ((state = S4) and (a = '0')) )
           else '0';
end;
```

© 2015 Elsevier, Inc.

Exercise 4.36

(starting on next page)

© 2015 Elsevier, Inc.

SystemVerilog

```
module ex4_36(input  logic clk, reset, n, d, q,
              output logic dispense,
                           return5, return10,
                           return2_10);
  typedef enum logic [3:0] {S0  = 4'b0000,
                            S5  = 4'b0001,
                            S10 = 4'b0010,
                            S25 = 4'b0011,
                            S30 = 4'b0100,
                            S15 = 4'b0101,
                            S20 = 4'b0110,
                            S35 = 4'b0111,
                            S40 = 4'b1000,
                            S45 = 4'b1001}
  statetype;
  statetype [3:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0:       if (n) nextstate = S5;
           else if (d) nextstate = S10;
           else if (q) nextstate = S25;
           else        nextstate = S0;
      S5:       if (n) nextstate = S10;
           else if (d) nextstate = S15;
           else if (q) nextstate = S30;
           else        nextstate = S5;
      S10:      if (n) nextstate = S15;
           else if (d) nextstate = S20;
           else if (q) nextstate = S35;
           else        nextstate = S10;
      S25:             nextstate = S0;
      S30:             nextstate = S0;
      S15:      if (n) nextstate = S20;
           else if (d) nextstate = S25;
           else if (q) nextstate = S40;
           else        nextstate = S15;
      S20:      if (n) nextstate = S25;
           else if (d) nextstate = S30;
           else if (q) nextstate = S45;
           else        nextstate = S20;
      S35:             nextstate = S0;
      S40:             nextstate = S0;
      S45:             nextstate = S0;
      default:         nextstate = S0;
    endcase
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_36 is
  port(clk, reset, n, d, q: in  STD_LOGIC;
       dispense, return5, return10: out STD_LOGIC;
       return2_10:                  out STD_LOGIC);
end;

architecture synth of ex4_36 is
  type statetype is (S0, S5, S10, S25, S30, S15, S20,
                     S35, S40, S45);
  signal state, nextstate: statetype;
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S0  =>
        if n    then nextstate <= S5;
        elsif d then nextstate <= S10;
        elsif q then nextstate <= S25;
        else         nextstate <= S0;
        end if;
      when S5  =>
        if n    then nextstate <= S10;
        elsif d then nextstate <= S15;
        elsif q then nextstate <= S30;
        else         nextstate <= S5;
        end if;
      when S10 =>
        if n    then nextstate <= S15;
        elsif d then nextstate <= S20;
        elsif q then nextstate <= S35;
        else         nextstate <= S10;
        end if;
      when S25 =>    nextstate <= S0;
      when S30 =>    nextstate <= S0;
      when S15 =>
        if n    then nextstate <= S20;
        elsif d then nextstate <= S25;
        elsif q then nextstate <= S40;
        else         nextstate <= S15;
        end if;
      when S20 =>
        if n    then nextstate <= S25;
        elsif d then nextstate <= S30;
        elsif q then nextstate <= S45;
        else         nextstate <= S20;
        end if;
      when S35 =>    nextstate <= S0;
      when S40 =>    nextstate <= S0;
      when S45 =>    nextstate <= S0;
      when others => nextstate <= S0;
    end case;
  end process;
```

(continued from previous page)

SystemVerilog

```
  // Output Logic
  assign dispense   = (state == S25) |
                      (state == S30) |
                      (state == S35) |
                      (state == S40) |
                      (state == S45);
  assign return5    = (state == S30) |
                      (state == S40);
  assign return10   = (state == S35) |
                      (state == S40);
  assign return2_10 = (state == S45);
endmodule
```

VHDL

```
  -- output logic
  dispense   <= '1' when ((state = S25) or
                          (state = S30) or
                          (state = S35) or
                          (state = S40) or
                          (state = S45))
              else '0';
  return5    <= '1' when ((state = S30) or
                          (state = S40))
              else '0';
  return10   <= '1' when ((state = S35) or
                          (state = S40))
              else '0';
  return2_10 <= '1' when (state = S45)
              else '0';
end;
```

Exercise 4.37

© 2015 Elsevier, Inc.

SystemVerilog

```
module ex4_37(input  logic       clk, reset,
              output logic [2:0] q);
  typedef enum logic [2:0] {S0 = 3'b000,
                            S1 = 3'b001,
                            S2 = 3'b011,
                            S3 = 3'b010,
                            S4 = 3'b110,
                            S5 = 3'b111,
                            S6 = 3'b101,
                            S7 = 3'b100}
    statetype;

  statetype [2:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: nextstate = S1;
      S1: nextstate = S2;
      S2: nextstate = S3;
      S3: nextstate = S4;
      S4: nextstate = S5;
      S5: nextstate = S6;
      S6: nextstate = S7;
      S7: nextstate = S0;
    endcase

  // Output Logic
  assign q = state;
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_37 is
  port(clk:   in  STD_LOGIC;
       reset: in  STD_LOGIC;
       q:     out STD_LOGIC_VECTOR(2 downto 0));
end;

architecture synth of ex4_37 is
  signal state:     STD_LOGIC_VECTOR(2 downto 0);
  signal nextstate: STD_LOGIC_VECTOR(2 downto 0);
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= "000";
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

  -- next state logic
  process(all) begin
    case state is
      when "000"  => nextstate <= "001";
      when "001"  => nextstate <= "011";
      when "011"  => nextstate <= "010";
      when "010"  => nextstate <= "110";
      when "110"  => nextstate <= "111";
      when "111"  => nextstate <= "101";
      when "101"  => nextstate <= "100";
      when "100"  => nextstate <= "000";
      when others => nextstate <= "000";
    end case;
  end process;

  -- output logic
  q <= state;
end;
```

Exercise 4.38

SystemVerilog

```
module ex4_38(input  logic       clk, reset, up,
              output logic [2:0] q);

  typedef enum logic [2:0] {
    S0 = 3'b000,
    S1 = 3'b001,
    S2 = 3'b011,
    S3 = 3'b010,
    S4 = 3'b110,
    S5 = 3'b111,
    S6 = 3'b101,
    S7 = 3'b100} statetype;
  statetype [2:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: if (up) nextstate = S1;
          else    nextstate = S7;
      S1: if (up) nextstate = S2;
          else    nextstate = S0;
      S2: if (up) nextstate = S3;
          else    nextstate = S1;
      S3: if (up) nextstate = S4;
          else    nextstate = S2;
      S4: if (up) nextstate = S5;
          else    nextstate = S3;
      S5: if (up) nextstate = S6;
          else    nextstate = S4;
      S6: if (up) nextstate = S7;
          else    nextstate = S5;
      S7: if (up) nextstate = S0;
          else    nextstate = S6;
    endcase

  // Output Logic
  assign q = state;
endmodule
```

(continued on next page)

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_38 is
  port(clk:   in  STD_LOGIC;
       reset: in  STD_LOGIC;
       up:    in  STD_LOGIC;
       q:     out STD_LOGIC_VECTOR(2 downto 0));
end;

architecture synth of ex4_38 is
  signal state:     STD_LOGIC_VECTOR(2 downto 0);
  signal nextstate: STD_LOGIC_VECTOR(2 downto 0);
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= "000";
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

  -- next state logic
  process(all) begin
    case state is
      when "000"  => if up then
                       nextstate <= "001";
                     else
                       nextstate <= "100";
                     end if;
      when "001"  => if up then
                       nextstate <= "011";
                     else
                       nextstate <= "000";
                     end if;
      when "011"  => if up then
                       nextstate <= "010";
                     else
                       nextstate <= "001";
                     end if;
      when "010"  => if up then
                       nextstate <= "110";
                     else
                       nextstate <= "011";
                     end if;
```

© 2015 Elsevier, Inc.

(continued from previous page)

VHDL

```
      when "110"  => if up then
                       nextstate <= "111";
                     else
                       nextstate <= "010";
                     end if;
      when "111"  => if up then
                       nextstate <= "101";
                     else
                       nextstate <= "110";
                     end if;
      when "101"  => if up then
                       nextstate <= "100";
                     else
                       nextstate <= "111";
                     end if;
      when "100"  => if up then
                       nextstate <= "000";
                     else
                       nextstate <= "101";
                     end if;
      when others => nextstate <= "000";
    end case;
  end process;

  -- output logic
  q <= state;
end;
```

Exercise 4.39

Option 1

SystemVerilog

```
module ex4_39(input  logic clk, reset, a, b,
              output logic z);
  typedef enum logic [1:0] {S0, S1, S2, S3}
    statetype;
  statetype [1:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: case ({b,a})
            2'b00: nextstate = S0;
            2'b01: nextstate = S3;
            2'b10: nextstate = S0;
            2'b11: nextstate = S1;
          endcase
      S1: case ({b,a})
            2'b00: nextstate = S0;
            2'b01: nextstate = S3;
            2'b10: nextstate = S2;
            2'b11: nextstate = S1;
          endcase
      S2: case ({b,a})
            2'b00: nextstate = S0;
            2'b01: nextstate = S3;
            2'b10: nextstate = S2;
            2'b11: nextstate = S1;
          endcase
      S3: case ({b,a})
            2'b00: nextstate = S0;
            2'b01: nextstate = S3;
            2'b10: nextstate = S2;
            2'b11: nextstate = S1;
          endcase
      default:     nextstate = S0;
    endcase

  // Output Logic
  always_comb
    case (state)
      S0:      z = a & b;
      S1:      z = a | b;
      S2:      z = a & b;
      S3:      z = a | b;
      default: z = 1'b0;
    endcase
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_39 is
  port(clk:   in  STD_LOGIC;
       reset: in  STD_LOGIC;
       a, b:  in  STD_LOGIC;
       z:     out STD_LOGIC);
end;

architecture synth of ex4_39 is
  type statetype is (S0, S1, S2, S3);
  signal state, nextstate: statetype;
  signal ba: STD_LOGIC_VECTOR(1 downto 0);
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

  -- next state logic
  ba <= b & a;
  process(all) begin
    case state is
      when S0  =>
        case (ba) is
          when "00"   => nextstate <= S0;
          when "01"   => nextstate <= S3;
          when "10"   => nextstate <= S0;
          when "11"   => nextstate <= S1;
          when others => nextstate <= S0;
        end case;
      when S1  =>
        case (ba) is
          when "00"   => nextstate <= S0;
          when "01"   => nextstate <= S3;
          when "10"   => nextstate <= S2;
          when "11"   => nextstate <= S1;
          when others => nextstate <= S0;
        end case;
      when S2  =>
        case (ba) is
          when "00"   => nextstate <= S0;
          when "01"   => nextstate <= S3;
          when "10"   => nextstate <= S2;
          when "11"   => nextstate <= S1;
          when others => nextstate <= S0;
        end case;
      when S3  =>
        case (ba) is
          when "00"   => nextstate <= S0;
          when "01"   => nextstate <= S3;
          when "10"   => nextstate <= S2;
          when "11"   => nextstate <= S1;
          when others => nextstate <= S0;
        end case;
      when others     =>   nextstate <= S0;
    end case;
  end process;
```

© 2015 Elsevier, Inc.

(continued from previous page)

VHDL

```
  -- output logic
  process(all) begin
    case state is
      when S0     => if (a = '1' and b = '1')
                     then z <= '1';
                     else z <= '0';
                     end if;
      when S1     => if (a = '1' or b = '1')
                     then z <= '1';
                     else z <= '0';
                     end if;
      when S2     => if (a = '1' and b = '1')
                     then z <= '1';
                     else z <= '0';
                     end if;
      when S3     => if (a = '1' or b = '1')
                     then z <= '1';
                     else z <= '0';
                     end if;
      when others => z <= '0';
    end case;
  end process;
end;
```

Option 2

SystemVerilog

```
module ex4_37(input  logic clk, a, b,
              output logic z);

  logic aprev;

  // State Register
  always_ff @(posedge clk)
    aprev <= a;

  assign z = b ? (aprev | a) : (aprev & a);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_37 is
  port(clk:   in  STD_LOGIC;
       a, b:  in  STD_LOGIC;
       z:     out STD_LOGIC);
end;

architecture synth of ex4_37 is
  signal aprev, n1and, n2or: STD_LOGIC;
begin
  -- state register
  process(clk) begin
    if rising_edge(clk) then
      aprev <= a;
    end if;
  end process;


  z <= (a or aprev) when b = '1' else
       (a and aprev);
end;
```

Exercise 4.40

SystemVerilog

```
module fsm_y(input  clk, reset, a,
             output y);
  typedef enum logic [1:0] {S0=2'b00, S1=2'b01,
    S11=2'b11} statetype;
  statetype [1:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0:  if (a) nextstate = S1;
           else   nextstate = S0;
      S1:  if (a) nextstate = S11;
           else   nextstate = S0;
      S11:        nextstate = S11;
      default:    nextstate = S0;
    endcase

  // Output Logic
  assign y = state[1];
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fsm_y is
  port(clk, reset, a: in  STD_LOGIC;
       y:             out STD_LOGIC);
end;

architecture synth of fsm_y is
  type statetype is (S0, S1, S11);
  signal state, nextstate: statetype;
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S0 =>  if a then
                       nextstate <= S1;
                  else nextstate <= S0;
                  end if;
      when S1 =>  if a then
                       nextstate <= S11;
                  else nextstate <= S0;
                  end if;
      when S11 =>      nextstate <= S11;
      when others =>   nextstate <= S0;
    end case;
  end process;

  -- output logic
  y <= '1' when (state = S11) else '0';
end;
```

(continued on next page)

© 2015 Elsevier, Inc.

(continued from previous page)

SystemVerilog

```
module fsm_x(input  logic clk, reset, a,
             output logic x);
  typedef enum logic [1:0] {S0, S1, S2, S3}
    statetype;
  statetype [1:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: if   (a) nextstate = S1;
          else     nextstate = S0;
      S1: if   (a) nextstate = S2;
          else     nextstate = S1;
      S2: if   (a) nextstate = S3;
          else     nextstate = S2;
      S3:          nextstate = S3;
    endcase

  // Output Logic
  assign x = (state == S3);
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fsm_x is
  port(clk, reset, a: in  STD_LOGIC;
       x:             out STD_LOGIC);
end;

architecture synth of fsm_x is
  type statetype is (S0, S1, S2, S3);
  signal state, nextstate: statetype;
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S0 =>  if a then
                       nextstate <= S1;
                  else nextstate <= S2;
                  end if;
      when S1 =>  if a then
                       nextstate <= S2;
                  else nextstate <= S1;
                  end if;
      when S2 =>  if a then
                       nextstate <= S3;
                  else nextstate <= S2;
                  end if;
      when S3 =>       nextstate <= S3;
      when others =>   nextstate <= S0;
    end case;
  end process;

  -- output logic
  x <= '1' when (state = S3) else '0';
end;
```

Exercise 4.41

SystemVerilog

```
module ex4_41(input  logic clk, start, a,
              output logic q);
  typedef enum logic [1:0] {S0, S1, S2, S3}
    statetype;
  statetype [1:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge start)
    if (start) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: if   (a) nextstate = S1;
          else     nextstate = S0;
      S1: if   (a) nextstate = S2;
          else     nextstate = S3;
      S2: if   (a) nextstate = S2;
          else     nextstate = S3;
      S3: if   (a) nextstate = S2;
          else     nextstate = S3;
    endcase

  // Output Logic
  assign q = state[0];
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_41 is
  port(clk, start, a: in  STD_LOGIC;
       q:             out STD_LOGIC);
end;

architecture synth of ex4_41 is
  type statetype is (S0, S1, S2, S3);
  signal state, nextstate: statetype;
begin
  -- state register
  process(clk, start) begin
    if start then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S0 =>  if a then
                       nextstate <= S1;
                  else nextstate <= S0;
                  end if;
      when S1 =>  if a then
                       nextstate <= S2;
                  else nextstate <= S3;
                  end if;
      when S2 =>  if a then
                       nextstate <= S2;
                  else nextstate <= S3;
                  end if;
      when S3 =>  if a then
                       nextstate <= S2;
                  else nextstate <= S3;
                  end if;
      when others =>   nextstate <= S0;
    end case;
  end process;

  -- output logic
  q <= '1' when ((state = S1) or (state = S3))
       else '0';
end;
```

Exercise 4.42

© 2015 Elsevier, Inc.

SystemVerilog

```
module ex4_42(input  logic clk, reset, x,
              output logic q);
  typedef enum logic [1:0] {S0, S1, S2, S3}
    statetype;
  statetype [1:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S00;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S00: if (x) nextstate = S11;
           else   nextstate = S01;
      S01: if (x) nextstate = S10;
           else   nextstate = S00;
      S10:        nextstate = S01;
      S11:        nextstate = S01;
    endcase

  // Output Logic
  assign q = state[0] | state[1];
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_42 is
  port(clk, reset, x: in  STD_LOGIC;
       q:             out STD_LOGIC);
end;

architecture synth of ex4_42 is
  type statetype is (S00, S01, S10, S11);
  signal state, nextstate: statetype;
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S00;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S00 =>  if x then
                       nextstate <= S11;
                  else nextstate <= S01;
                  end if;
      when S01 =>  if x then
                       nextstate <= S10;
                  else nextstate <= S00;
                  end if;
      when S10 =>      nextstate <= S01;
      when S11 =>      nextstate <= S01;
      when others =>   nextstate <= S00;
    end case;
  end process;

  -- output logic
  q <= '0' when (state = S00) else '1';
end;
```

Exercise 4.43

SystemVerilog

```
module ex4_43(input  clk, reset, a,
              output q);
  typedef enum logic [1:0] {S0, S1, S2} statetype;
  statetype [1:0] state, nextstate;

  // State Register
  always_ff @(posedge clk, posedge reset)
    if (reset) state <= S0;
    else       state <= nextstate;

  // Next State Logic
  always_comb
    case (state)
      S0: if (a) nextstate = S1;
          else   nextstate = S0;
      S1: if (a) nextstate = S2;
          else   nextstate = S0;
      S2: if (a) nextstate = S2;
          else   nextstate = S0;
      default:   nextstate = S0;
    endcase

  // Output Logic
  assign q = state[1];
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_43 is
  port(clk, reset, a: in  STD_LOGIC;
       q:             out STD_LOGIC);
end;

architecture synth of ex4_43 is
  type statetype is (S0, S1, S2);
  signal state, nextstate: statetype;
begin
  -- state register
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

 -- next state logic
  process(all) begin
    case state is
      when S0 =>  if a then
                       nextstate <= S1;
                  else nextstate <= S0;
                  end if;
      when S1 =>  if a then
                       nextstate <= S2;
                  else nextstate <= S0;
                  end if;
      when S2 =>  if a then
                       nextstate <= S2;
                  else nextstate <= S0;
                  end if;
      when others =>   nextstate <= S0;
    end case;
  end process;

  -- output logic
  q <= '1' when (state = S2) else '0';
end;
```

Exercise 4.44

© 2015 Elsevier, Inc.

(a)

SystemVerilog

```
module ex4_44a(input logic clk, a, b, c, d,
               output logic q);

  logic areg, breg, creg, dreg;

  always_ff @(posedge clk)
    begin
      areg <= a;
      breg <= b;
      creg <= c;
      dreg <= d;
      q    <= ((areg ^ breg) ^ creg) ^ dreg;
    end
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_44a is
  port(clk, a, b, c, d: in  STD_LOGIC;
       q:               out STD_LOGIC);
end;

architecture synth of ex4_44a is
  signal areg, breg, creg, dreg: STD_LOGIC;
begin
  process(clk) begin
    if rising_edge(clk) then
      areg <= a;
      breg <= b;
      creg <= c;
      dreg <= d;
      q <= ((areg xor breg) xor creg) xor dreg;
    end if;
  end process;
end;
```

(d)

SystemVerilog

```
module ex4_44d(input logic clk, a, b, c, d,
               output logic q);

  logic areg, breg, creg, dreg;

  always_ff @(posedge clk)
    begin
      areg <= a;
      breg <= b;
      creg <= c;
      dreg <= d;
      q    <= (areg ^ breg) ^ (creg ^ dreg);
    end
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_44d is
  port(clk, a, b, c, d: in  STD_LOGIC;
       q:               out STD_LOGIC);
end;

architecture synth of ex4_44d is
  signal areg, breg, creg, dreg: STD_LOGIC;
begin
  process(clk) begin
    if rising_edge(clk) then
      areg <= a;
      breg <= b;
      creg <= c;
      dreg <= d;
      q <= (areg xor breg) xor (creg xor dreg);
    end if;
  end process;
end;
```

Exercise 4.45

SystemVerilog

```
module ex4_45(input  logic       clk, c,
              input  logic [1:0] a, b,
              output logic [1:0] s);

  logic  [1:0] areg, breg;
  logic        creg;
  logic [1:0]  sum;
  logic        cout;

  always_ff @(posedge clk)
    {areg, breg, creg, s} <= {a, b, c, sum};

  fulladder fulladd1(areg[0], breg[0], creg,
                     sum[0], cout);
  fulladder fulladd2(areg[1], breg[1], cout,
                     sum[1], );
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity ex4_45 is
  port(clk, c: in  STD_LOGIC;
       a, b:   in  STD_LOGIC_VECTOR(1 downto 0);
       s:      out STD_LOGIC_VECTOR(1 downto 0));
end;

architecture synth of ex4_45 is
  component fulladder is
    port(a, b, cin:  in  STD_LOGIC;
         s, cout:    out STD_LOGIC);
  end component;
  signal creg: STD_LOGIC;
  signal areg, breg, cout: STD_LOGIC_VECTOR(1 downto
0);
  signal sum:        STD_LOGIC_VECTOR(1 downto 0);
begin
  process(clk) begin
    if rising_edge(clk) then
      areg <= a;
      breg <= b;
      creg <= c;
      s <= sum;
    end if;
  end process;

  fulladd1: fulladder
   port map(areg(0), breg(0), creg, sum(0), cout(0));
  fulladd2: fulladder
      port  map(areg(1),  breg(1),  cout(0),  sum(1),
cout(1));
end;
```

Exercise 4.46

A signal declared as `tri` can have multiple drivers.

Exercise 4.47

© 2015 Elsevier, Inc.

SystemVerilog

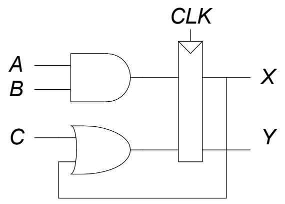
```
module syncbad(input  logic clk,
               input  logic d,
               output logic q);

  logic n1;

  always_ff @(posedge clk)
    begin
       q  <= n1;// nonblocking
       n1 <= d; // nonblocking
   end
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity syncbad is
  port(clk: in  STD_LOGIC;
       d:   in  STD_LOGIC;
       q:   out STD_LOGIC);
end;

architecture bad of syncbad is
begin
  process(clk)
    variable n1: STD_LOGIC;
  begin
    if rising_edge(clk) then
      q  <= n1; -- nonblocking
      n1 <= d;  -- nonblocking
    end if;
  end process;
end;
```

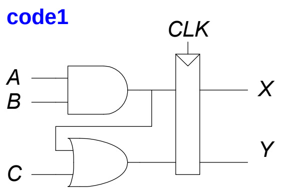
Exercise 4.48

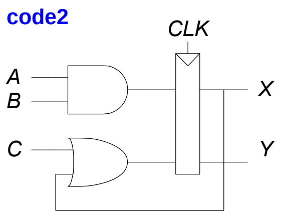
They have the same function.



Exercise 4.49

They do not have the same function.





Exercise 4.50

(a) Problem: Signal `d` is not included in the sensitivity list of the always statement. Correction shown below (changes are in bold).

```
module latch(input  logic     clk,
             input  logic [3:0] d,
             output logic [3:0] q);

   always_latch
      if (clk) q <= d;
endmodule
```

(b) Problem: Signal `b` is not included in the sensitivity list of the always statement. Correction shown below (changes are in bold).

```
module gates(input  logic [3:0] a, b,
             output logic [3:0] y1, y2, y3, y4, y5);

   always_comb
      begin
         y1 = a & b;
         y2 = a | b;
         y3 = a ^ b;
         y4 = ~(a & b);
         y5 = ~(a | b);
      end
endmodule
```

(c) Problem: The sensitivity list should not include the word “`posedge`”. The `always` statement needs to respond to any changes in `s`, not just the positive edge. Signals `d0` and `d1` need to be added to the sensitivity list. Also, the always statement implies combinational logic, so blocking assignments should be used.

© 2015 Elsevier, Inc.

```
module mux2(input  logic [3:0] d0, d1,
            input  logic       s,
            output logic [3:0] y);

   always_comb
      if (s) y = d1;
      else   y = d0;
endmodule
```

(d) Problem: This module will actually work in this case, but it's good practice to use nonblocking assignments in `always` statements that describe sequential logic. Because the `always` block has more than one statement in it, it requires a begin and end.

```
module twoflops(input  logic clk,
                input  logic d0, d1,
                output logic q0, q1);

   always_ff @(posedge clk)
   begin
      q1 <= d1;           // nonblocking assignment
      q0 <= d0;           // nonblocking assignment
   end
endmodule
```

(e) Problem: out1 and out2 are not assigned for all cases. Also, it would be best to separate the next state logic from the state register. reset is also missing in the input declaration.

```
module FSM(input  logic clk,
           input  logic reset,
           input  logic a,
           output logic out1, out2);

   logic  state, nextstate;

   // state register
   always_ff @(posedge clk, posedge reset)
     if (reset)
       state <= 1'b0;
     else
       state <= nextstate;

   // next state logic
   always_comb
     case (state)
       1'b0: if (a) nextstate = 1'b1;
             else nextstate = 1'b0;
       1'b1: if (~a) nextstate = 1'b0;
             else nextstate = 1'b1;
     endcase

   // output logic (combinational)
   always_comb
      if (state == 0) {out1, out2} = {1'b1, 1'b0};
      else            {out1, out2} = {1'b0, 1'b1};
endmodule
```

(f) Problem: A priority encoder is made from combinational logic, so the HDL must completely define what the outputs are for all possible input combinations. So, we must add an else statement at the end of the `always` block.

```
module priority(input  logic [3:0] a,
                 output logic [3:0] y);

   always_comb
      if      (a[3]) y = 4'b1000;
      else if (a[2]) y = 4'b0100;
      else if (a[1]) y = 4'b0010;
      else if (a[0]) y = 4'b0001;
      else           y = 4'b0000;
endmodule
```

(g) Problem: the next state logic block has no default statement. Also, state S2 is missing the S.

```
module divideby3FSM(input  logic clk,
                    input  logic reset,
                    output logic out);

   logic  [1:0] state, nextstate;

   parameter S0 = 2'b00;
   parameter S1 = 2'b01;
   parameter S2 = 2'b10;

   // State Register
   always_ff @(posedge clk, posedge reset)
      if (reset) state <= S0;
      else       state <= nextstate;

   // Next State Logic
   always_comb
      case (state)
         S0:      nextstate = S1;
         S1:      nextstate = S2;
         S2:      nextstate = S0;
         default: nextstate = S0;
      endcase

   // Output Logic
   assign out = (state == S2);
endmodule
```

(h) Problem: the ~ is missing on the first tristate.

```
module mux2tri(input  logic [3:0] d0, d1,
               input  logic       s,
               output logic [3:0] y);

   tristate t0(d0, ~s, y);
   tristate t1(d1, s, y);

endmodule
```

(i) Problem: an output, in this case, `q`, cannot be assigned in multiple always or assignment statements. Also, the flip-flop does not include an enable, so it should not be named `floprsen`.

```
module floprs(input  logic       clk,
              input  logic       reset,
              input  logic       set,
              input  logic [3:0] d,
              output logic [3:0] q);

   always_ff @(posedge clk, posedge reset, posedge set)
```

© 2015 Elsevier, Inc.

```
      if (reset)    q <= 0;
      else if (set) q <= 1;
      else          q <= d;
endmodule
```

(j) Problem: this is a combinational module, so nonconcurrent (blocking) assignment statements (=) should be used in the always statement, not concurrent assignment statements (<=). Also, it's safer to use always @(*) for combinational logic to make sure all the inputs are covered.

```
module and3(input  logic a, b, c,
            output logic y);

   logic tmp;

   always_comb
     begin
      tmp = a & b;
      y   = tmp & c;
     end
endmodule
```

Exercise 4.51

It is necessary to write

```
q <= '1' when state = S0 else '0';
```

rather than simply

```
q <= (state = S0);
```

because the result of the comparison `(state = S0)` is of type `Boolean` (`true` and `false`) and q must be assigned a value of type STD_LOGIC (`'1'` and `'0'`).

Exercise 4.52

(a) **Problem:** both clk and d must be in the process statement.

```
architecture synth of latch is
begin
  process(clk, d) begin
    if clk = '1' then q <= d;
    end if;
  end process;
end;
```

(b) **Problem:** both a and b must be in the process statement.

```
architecture proc of gates is
begin
  process(all) begin
    y1 <= a and b;
    y2 <= a or b;
    y3 <= a xor b;
    y4 <= a nand b;
    y5 <= a nor b;
  end process;
end;
```

(c) **Problem:** The end if and end process statements are missing.

```
architecture synth of flop is
begin
  process(clk)
    if clk'event and clk = '1' then
      q <= d;
    end if;
  end process;
end;
```

(d) **Problem:** The final else statement is missing. Also, it’s better to use “process(all)” instead of “process(a)”

```
architecture synth of priority is
begin
  process(all) begin
    if    a(3) = '1' then y <= "1000";
    elsif a(2) = '1' then y <= "0100";
    elsif a(1) = '1' then y <= "0010";
    elsif a(0) = '1' then y <= "0001";
    else                  y <= "0000";
    end if;
  end process;
end;
```

(e) **Problem:** The `default` statement is missing in the nextstate case statement. Also, it’s better to use the updated statements: “if reset”, “rising_edge(clk)”, and “process(all)”.

```
architecture synth of divideby3FSM is
  type statetype is (S0, S1, S2);
  signal state, nextstate: statetype;
begin
  process(clk, reset) begin
    if reset then state <= S0;
    elsif rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

   process(all) begin
    case state is
      when S0 =>      nextstate <= S1;
      when S1 =>      nextstate <= S2;
      when S2 =>      nextstate <= S0;
      when others =>  nextstate <= S0;
    end case;
  end process;

  q <= '1' when state = S0 else '0';
end;
```

(f) **Problem:** The select signal on tristate instance t0 must be inverted. However, VHDL does not allow logic to be performed within an instance declaration. Thus, an internal signal, `sbar`, must be declared.

```
architecture struct of mux2 is
  component tristate
```

© 2015 Elsevier, Inc.

```
    port(a:  in  STD_LOGIC_VECTOR(3 downto 0);
         en: in  STD_LOGIC;
         y:  out STD_LOGIC_VECTOR(3 downto 0));
  end component;
  signal sbar: STD_LOGIC;
begin
  sbar <= not s;
  t0: tristate port map(d0, sbar, y);
  t1: tristate port map(d1, s, y);
end;
```

(g) **Problem:** The q output cannot be assigned in two process or assignment statements. Also, it's better to use the updated statements: "if reset", and "rising_edge(clk)".

```
architecture asynchronous of flopr is
begin
  process(clk, reset, set) begin
    if reset then
      q <= '0';
    elsif set then
      q <= '1';
    elsif rising_edge(clk) then
      q <= d;
    end if;
  end process;
end;
```

Question 4.1

SystemVerilog

```
assign result = sel ? data : 32'b0;
```

VHDL

```
result <= data when sel = '1' else X"00000000";
```

Question 4.2

HDLs support *blocking* and *nonblocking assignments* in an `always` / `process` statement. A group of blocking assignments are evaluated in the order they appear in the code, just as one would expect in a standard programming language. A group of nonblocking assignments are evaluated concurrently; all of the statements are evaluated before any of the left hand sides are updated.

SystemVerilog

In a SystemVerilog `always` statement, = indicates a blocking assignment and <= indicates a nonblocking assignment.

Do not confuse either type with continuous assignment using the `assign` statement. `assign` statements are normally used outside `always` statements and are also evaluated concurrently.

VHDL

In a VHDL `process` statement, `:=` indicates a blocking assignment and <= indicates a nonblocking assignment (also called a concurrent assignment). This is the first section where `:=` is introduced.

Nonblocking assignments are made to outputs and to signals. Blocking assignments are made to variables, which are declared in `process` statements (see the next example).

<= can also appear outside `process` statements, where it is also evaluated concurrently.

See HDL Examples 4.24 and 4.29 for comparisons of blocking and nonblocking assignments. Blocking and nonblocking assignment guidelines are given on page 206.

Question 4.3

The SystemVerilog statement performs the bit-wise AND of the 16 least significant bits of data with 0xC820. It then ORs these 16 bits to produce the 1-bit result.