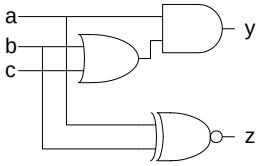
CHAPTER 4

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 4.3

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 4.5

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 4.7

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
```

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*)

(*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*)

Exercise 4.9

SystemVerilog

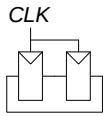
```
module ex4_9
   (input  logic a, b, c,
    output logic y);

   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;
```

Exercise 4.11

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



Exercise 4.13

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 4.15

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

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

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) $Y = \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C} + A\bar{B}C\bar{D} + ABD + \bar{A}\bar{B}C\bar{D} + B\bar{C}D + \bar{A}$

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;
```

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 4.19

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;
```

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;
```

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 4.25

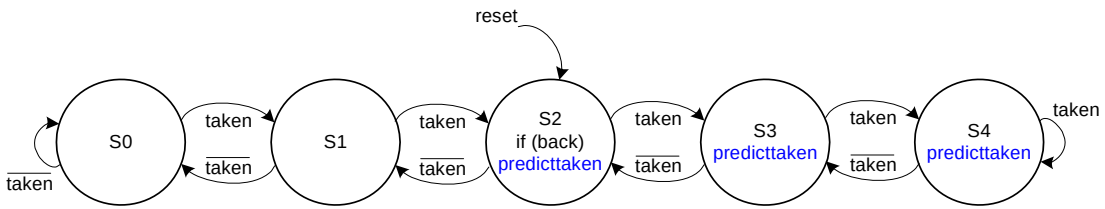


FIGURE 4.1 State transition diagram for Exercise 4.25

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 4.29

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 4.31

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 4.33

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 4.35

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;
```

Exercise 4.37

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 4.39

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;
```

(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 4.41

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 4.43

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 4.45

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 4.47

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
```

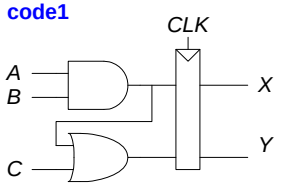
VHDL

```
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;
```

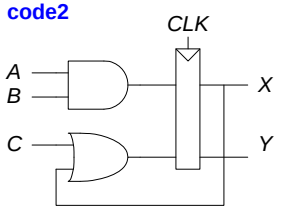
Exercise 4.49

They do not have the same function.

code1



code2



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'`).

Question 4.1

SystemVerilog

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

VHDL

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

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.