

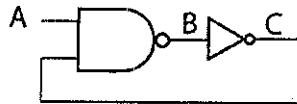
Chapter 2 Solutions

- 2.1 (a) VHDL - VHSIC Hardware Description Language (c) A hardware description language allows a digital system to be designed and debugged at a higher level of abstraction than schematic capture with gates, flip-flops, and standard MSI building blocks. The details of the gates and flip-flops do not need to be handled during early phases of design. Designs are more portable when low-level library-specific details are not included in the model. HDLs allow the creation of such portable high-level behavioral models.
- VHSIC - Very High Speed Integrated Circuit
- (b) VHDL has statements that execute concurrently since it must model real hardware in which the components are all in operation at the same time.

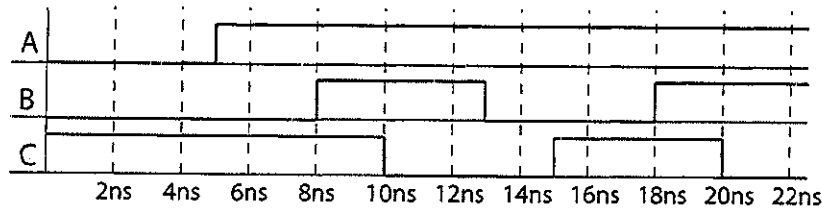
- 2.2 (a) Legal: A_123, and1; Illegal: 123A (Starts with number), _A123 (starts with underscore), A123_ (ends with underscore), c1__c2 (adjacent underscores), and (reserved word).

(b) They all equivalent. VHDL is not case sensitive

- 2.3 (a)



- (b)



```
2.4 entity Comb is
    port(A, B, C, D : in bit;
          Z : out bit);
end Comb;

architecture Comb2_1 of Comb is
    signal E, F, G, H, I : bit;
begin
    H <= (A and B and C) after 5 ns;
    E <= H or D after 5 ns;
    G <= (B nor C) after 5 ns;
    F <= G nand A after 5 ns;
    I <= not F after 2 ns;
    Z <= E xor I after 5 ns;
end Comb2_1;
```

```
2.5 (a) entity one_bit_sub is
    port(x, y, bin: in bit;
          diff, bout: out bit);
end one_bit_sub;

architecture equ of one_bit_sub is
begin
    diff <= x xor y xor bin; -- difference. See problem 1.2 answer
    bout <= (not x and bin) or (not x and y)
              or (bin and y); -- borrow out. See problem 1.2 answer
end equ;
```

```

(b) entity four_bit_sub is
    port(a, b: in bit_vector(3 downto 0);
          bin: in bit;
          d: out bit_vector(3 downto 0);
          bout: out bit);
end four_bit_sub;

architecture test of four_bit_sub is
    signal bo: bit_vector(3 downto 0) := "0000"; -- borrow bits
    component one_bit_sub
        port(x, y, bin: in bit;
              diff, bout: out bit);
    end component;
begin
    FS0: one_bit_sub port map(a(0),b(0),bin,d(0),bo(1));
    FS1: one_bit_sub port map(a(1),b(1),bo(1),d(1),bo(2));
    FS2: one_bit_sub port map(a(2),b(2),bo(2),d(2),bo(3));
    FS3: one_bit_sub port map(a(3),b(3),bo(3),d(3),bout);
end test;

```

2.6 (a) entity circuit is

```

    port(A, B, C, D: in bit;
          G: out bit);
end circuit;

```

```

architecture internal of circuit is
    signal E, F: bit;
begin
    E <= A and B;
    F <= E or C;
    G <= D and F;
end internal;

```

(b) entity circuit is

```

    port(A, B, C, D: in bit;
          G: out bit);
end circuit;

```

```

architecture internal of circuit is
    signal E, F: bit;
begin
    process(A, B, C, D, E, F)
    begin
        E <= A and B;
        F <= E or C;
        G <= D and F;
    end process;
end internal;

```

2.7 A changes to 1 at 25 ns, B changes to 1 at 20 + Δ ns, C does not change

2.8 (a) A falling-edge triggered D flip-flop with asynchronous active high clear and set

(b) Q = '0', because Clr = 1 has priority.

2.9 entity SR_Latch is

```

    port(S, R: in bit;
          Q, Qn: inout bit);
end SR_Latch;

```

```

architecture proc of SR_Latch is
begin
    process(S, R)
    begin
        if S = '1' then Q <= '1'; end if;
        if R = '1' then Q <= '0'; end if;
    end process;
    Qn <= not Q;
end proc;

```

2.10 entity MNFF is
 port(M, N, CLK, CLRn: in bit; Q: inout bit; Qn: out bit);
 end MNFF;

```
architecture MN of MNFF is
begin
  process(CLK, CLRn)
  begin
    if CLRn = '0' then Q <= '0';
    elsif CLK = '0' and CLK'event then
      if M = '0' and N = '0' then Q <= not Q;
      elsif M = '0' and N = '1' then Q <= '1';
      elsif M = '1' and N = '0' then Q <= '0';
      elsif M = '1' and N = '1' then Q <= Q; --optional
      end if;
    end if;
  end process;
  QN <= not Q;
end MN;
```

2.11 entity DDEF is
 port(R, S, D, Clk : in bit;
 Q : out bit);
 end DDEF;

```
architecture Behav of DDEF is
begin
  process(Clk, R, S)
  begin
    if R = '0' then Q <= '0';
    elsif S = '0' then Q <= '1';
    elsif Clk'event then Q <= D;
    end if;
  end process;
end Behav;
```

2.12 (a) entity ITFF is
 port(I0, I1, T, R: in bit;
 Q, QN: inout bit);
 end ITFF;

```
architecture behavior of ITFF is
begin
  process(T, R)
  begin
    if R = '1' then
      Q <= '0' after 5 ns;
    else
      if (I0 = '1' and T = '1' and I'event) or
        (I1 = '1' and T = '0' and T'event) then
        Q <= QN after 8 ns;
      end if;
    end if;
  end process;
  QN <= not Q;
end behavior;
```

(b) add list *
 add wave *
 force I 0 0, 1 100 -repeat
 200
 force I1 0 0, 1 50, 0 450
 force I0 0 0, 1 450
 run 750 ns

2.13

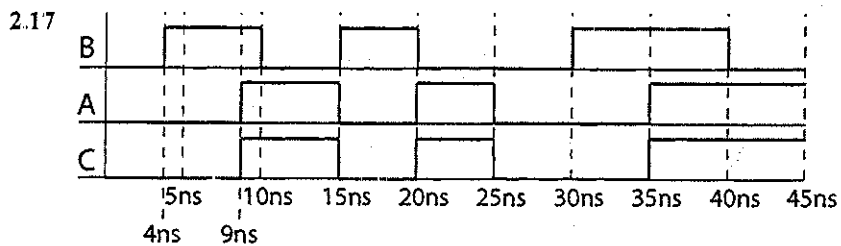
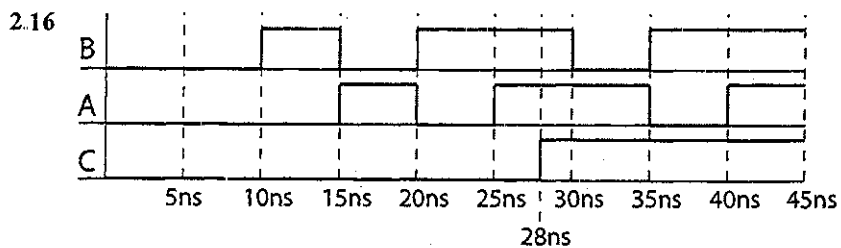
ns	Δ	a	b	c	d	e
0	+0	0	0	0	0	0
20	+0	0	0	0	0	1
20	+1	0	1	0	0	1
20	+2	0	8	0	0	1
23	+0	0	8	0	1	1
25	+0	1	8	0	1	1
35	+0	5	8	0	1	1

2.14

ns	Δ	a	b	c	d	e
10	+0	0	0	0	0	0
20	+0	0	0	0	0	1
20	+1	0	7	0	0	1
25	+0	1	7	0	0	1
35	+0	5	7	0	0	1

2.15

ns	Δ	a	b	c	d	e	f
0	+0	0	0	0	0	0	0
5	+0	0	0	0	1	0	0
5	+1	1	0	0	1	0	0
5	+2	1	1	1	1	0	0
5	+3	1	1	1	0	0	0
10	+0	1	1	1	0	1	0
10	+1	0	1	1	0	1	0
10	+2	0	0	0	0	1	0



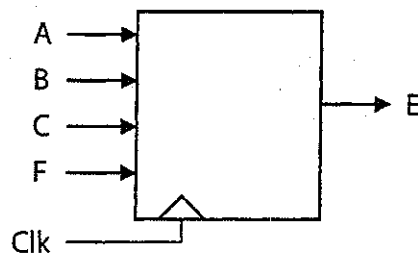
2.18

ns	Δ	a	b	c	d	
4	+0	0	0	0	0	
5	+0	1	0	0	0	P1 executes
10	+0	1	1	0	0	P2 executes
10	+1	0	1	0	1	P1 executes
12	+0	0	1	1	1	
15	+0	0	0	1	1	P2 executes
15	+1	1	0	1	1	P1 executes
17	+0	1	0	0	1	

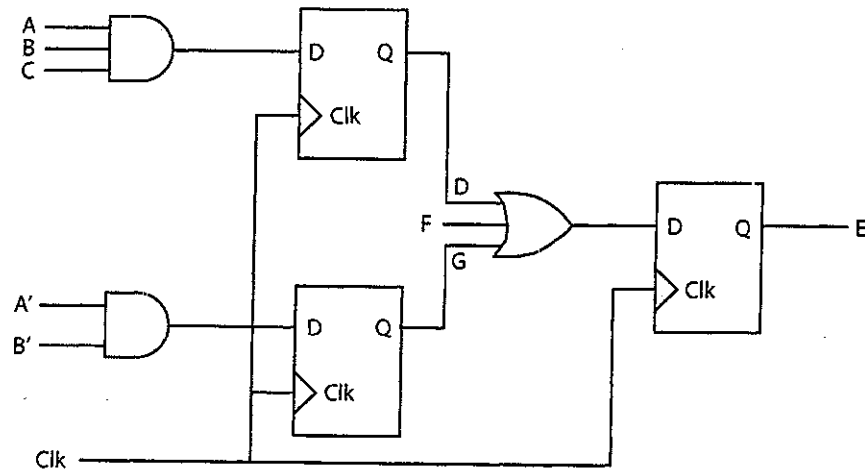
- 2.19 (a) $101011 \text{ or } 011010 = 111011$
 (b) $101 \text{ ror } 2 = 011$
 (c) $101 \text{ sla } 2 = 111$

- (d) $101 \& \text{not } 011 = 101 \& 100 = 101100$
 $101100 \neq 111110$, so evaluates to false
 (e) $101 \text{ or } 011 \text{ and } 010 = 111 \text{ and } 010 = 010$

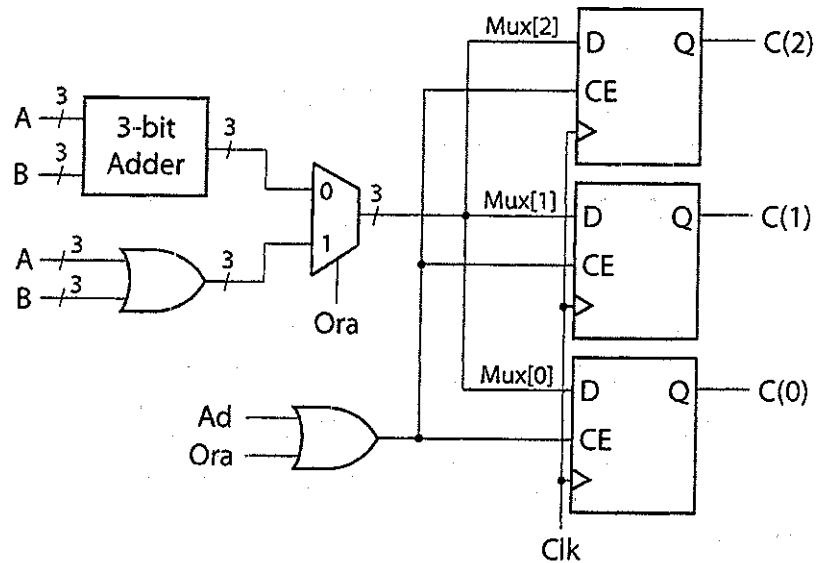
2.20 (a)



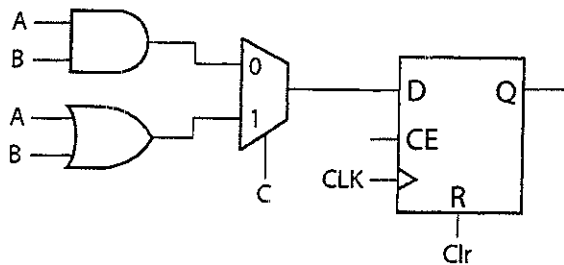
(b)



2.21



2.22



Unlike *Clr*, the output from the mux is only read on falling clock edges; therefore, adding *C* to the sensitivity list is not required for proper operation of the circuit.

2.23 (a) `sel <= C&D;`
`with sel select`
`F <= not A after 10 ns when "00",`
`B after 10 ns when "01",`
`not B after 10 ns when "10",`
`'0' after 10 ns when "11";`

(b) `F <= not A after 10 ns when C&D = "00"`
`else B after 10 ns when C&D = "01"`
`else not B after 10 ns when C&D = "10"`
`else '0' after 10 ns;`

(c) `sel <= C&D;`
`process(sel)`
`begin`
`case sel is`
`when "00" => F <= not A after 10 ns;`
`when "01" => F <= B after 10 ns;`
`when "10" => F <= not B after 10 ns;`
`when "11" => F <= '0' after 10 ns;`
`end case;`
`end process;`

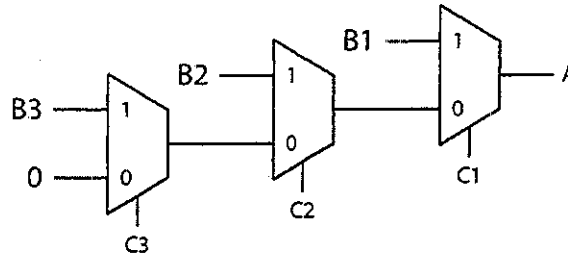
2.24 (a) `process(C, B1, B2, B3)`

```
begin
  case C is
    when 1 => A <= B1;
    when 2 => A <= B2;
    when 3 => A <= B3;
    when others => A <= 0;
  end case;
end process;
```

or

```
process(C, B1, B2, B3)
begin
  if C = 1 then A <= B1;
  elsif C = 2 then A <= B2;
  elsif C = 3 then A <= B3;
  else A <= 0; end if;
end process;
```

(b)



2.25 (a) `entity Latch is`
`port(S, R: in bit;`
`P, Q: inout bit);`
`end Latch;`

`architecture conditional of Latch is`
`begin`

```
  Q <= '1' when S = '1' -- Assume SR = 0
    else '0' when R = '1'
      else Q;
  P <= not Q;
end conditional;
```

(b) `architecture characteristic of Latch is`
`begin`

```
  Q <= S or (not R and Q);
  P <= not Q;
end characteristic;
```

(c) `architecture gate of Latch is`
`begin`

```
  P <= S nor Q;
  Q <= R nor P;
end gate;
```

2.26 `S="0101", Co='1'`

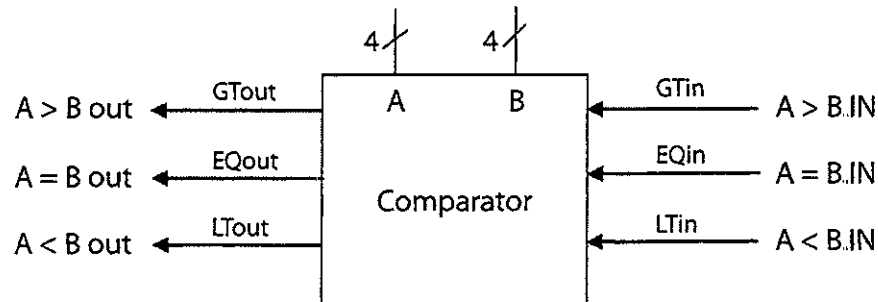
2.27 `library IEEE;`
`use IEEE.numeric_bit.all;`

```
entity Adder is
  port(A: in bit_vector(3 downto 0);
      B: in integer range 0 to 15;
      Sum: out bit_vector(4 downto 0));
end Adder;
```

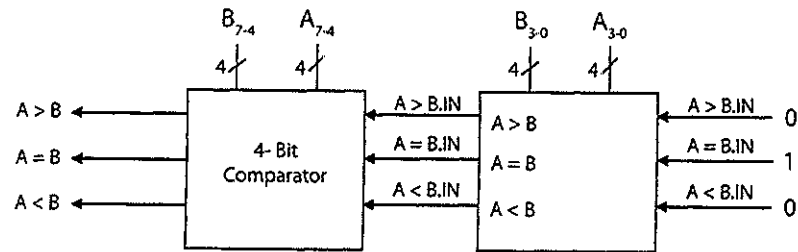
`architecture overloaded of Adder is`
`signal sum5: unsigned(4 downto 0);`
`begin`

```
  sum5 <= '0' & UNSIGNED(A) + TO_UNSIGNED(B, 4); -- conv fns in Appndx B
  Sum <= BIT_VECTOR(sum5);
end overloaded;
```

2.28 (a)



(b)



(c) entity comp4bit is

```
port(A, B: in bit_vector(3 downto 0);
      EQin, GTin, LTin: in bit;
      EQout, GTout, LTout: out bit);
end comp4bit;
```

architecture comparator of comp4bit is
begin

```
process(A, B, EQin, GTin, LTin)
begin
  if A > B then EQout <= '0'; GTout <= '1'; LTout <= '0';
  elsif A < B then EQout <= '0'; GTout <= '0'; LTout <= '1';
  elsif GTin = '1' then
    EQout <= '0'; GTout <= '1'; LTout <= '0';
  elsif (LTin = '1') then
    EQout <= '0'; GTout <= '0'; LTout <= '1';
  else EQout <= '1'; GTout <= '0'; LTout <= '0';
  end if;
end process;
end comparator;
```

(d) entity comp8bit is

```
port(A, B: in bit_vector(7 downto 0);
      EQi, GTi, LTi: in bit;
      EQ, GT, LT: out bit);
end comp8bit;
```

architecture comparator8 of comp8bit is

```
component comp4bit
port(A, B: in bit_vector(3 downto 0);
      EQin, GTin, LTin: in bit;
      EQout, GTout, LTout: out bit);
end component;
signal LowGT, LowEQ, LowLI: bit;
begin
  Comp0: comp4bit port map(A(3 downto 0), B(3 downto 0),
                           EQi, GTi, LTi, LowEQ, LowGT, LowLI);
  Comp1: comp4bit port map(A(7 downto 4), B(7 downto 4),
                           LowEQ, LowGT, LowLI, EQ, GT, LT);
end comparator8;
```

```

2.29 entity shift_reg is
    port(si, en, ck: in bit;
          so: out bit);
end entity;

```

```

architecture behav of shift_reg is
    signal reg : bit_vector(15 downto 0);
begin
    process(ck)
    begin
        if ck'event and ck = '1' then
            if en = '1' then
                reg(15 downto 0) <= si & reg(15 downto 1);
            end if;
        end if;
    end process;
    so <= reg(0);
end behav;

```

```

2.30 (a) entity shift74194 is
    port(d: in bit_vector(3 downto 0);
          s: in bit_vector(1 downto 0);
          sdr, sdl, clrb, clk: in bit;
          q: inout bit_vector(3 downto 0));
end shift74194;

```

```

architecture behav of shift74194 is
begin
    process(clk, clrb)
    begin
        if clrb = '0' then
            q <= "0000"; -- clear
        elsif clk = '1' and clk'event then
            case s is
                when "11" => q <= d; -- load
                when "10" => q <= sdr & q(3 downto 1); -- shift left
                when "01" => q <= q(2 downto 0) & sdl; -- shift right
                when "00" => q <= q; -- no action
            end case;
        end if;
    end process;
end behav;

```

```

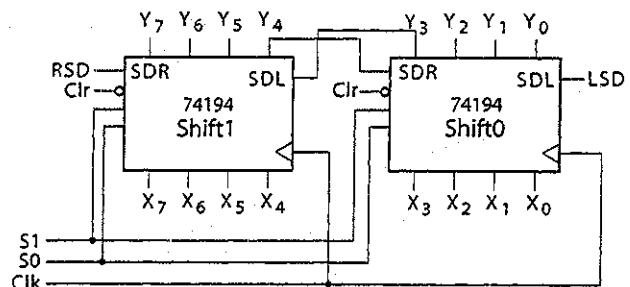
(b) entity bit8Shift is
    port(x: in bit_vector(7 downto 0);
          s: in bit_vector(1 downto 0);
          rsd, lsd, clrb, clk: in bit;
          y: inout bit_vector(7 downto 0));
end bit8Shift;

```

```

architecture struct of bit8Shift is
    component shift74194
        port(d: in bit_vector(3 downto 0);
              s: in bit_vector(1 downto 0);
              sdr, sdl, clrb, clk: in bit;
              q: inout bit_vector(3 downto 0));
    end component;
begin
    shift0: shift74194 port map(x(3 downto 0), s, y(4), lsd, clrb,
                                clk, y(3 downto 0));
    shift1: shift74194 port map(x(7 downto 4), s, rsd, y(3), clrb,
                                clk, y(7 downto 4));
end struct;

```



2.31 (a) library IEEE;
use IEEE.numeric_bit_all;

```
entity counter is
    port(d: in unsigned(3 downto 0);
          clk, clr, ent, enp, up, load: in bit;
          q: inout unsigned(3 downto 0); co: out bit);
end counter;

architecture test of counter is
begin
    process(clk, clr)
    begin
        if clr = '0' then Q <= "0000";
        elsif clk = '1' and clk'event then
            if load = '0' then q <= d; -- load
            elsif (ent = '1' and enp = '1' and (not up) = '1') then
                if q = "0000" then q <= "1001"; -- wrap around
                else q <= q - 1; end if; --decrement
            elsif (ent = '1' and enp = '1' and up = '1') then
                if q = "1001" then q <= "0000"; -- wrap around
                else q <= q + 1; end if; --increment
            end if;
        end if;
    end process;
    co <= '1' when (ent = '1') and ((up = '1' and (Q = "1001")) or
        (up = '0' and (Q = "0000"))) else '0';
end test;
```

(b) The block diagram is similar to Figure 2-47 with an "Up" input added to each counter. The VHDL description similar to Figure 2-48.

2.32 Students should look on the web for 74192 data sheet CLR is active high LOADB is active low. Counting up happens when UP has a rising edge and DOWN=1 Counting down happens when DOWN has a rising edge and UP=1. CARRY indicates terminal count in the up direction, i.e. 9. BORROW indicates terminal count in the down direction, i.e. 0.

Operating Mode	CLR	LOADB	UP	DOWN	D	Q	Borrow	Carry
Clear	1	X	X	0	XXXX	0000	0	1
	1	X	X	1	XXXX	0000	1	1
Load	0	0	X	X	XXXX	Q = D	1*	1*
Count Up	0	1	↑	1	XXXX	Q = Q + 1	1	1**
Count Down	0	1	1	↑	XXXX	Q = Q - 1	1**	1

* when loading, if the input is 0 and down = 0, borrow will be 0. If the input is 9 and up = 0, carry will be 0

** Borrow = 0 when the counter is in state 0 and down = 0 Carry = 0 when the counter is in state 9 and up = 0

```
entity count74192 is
    port(DOWN, UP, CLR, LOADB: in bit;
          BORROW, CARRY: out bit;
          D: in integer range 0 to 15;
          Q: inout integer range 0 to 15);
end count74192;

architecture behav of count74192 is
begin
    process(DOWN, UP, CLR, LOADB)
    begin
        if CLR = '1' then Q <= 0;
        elsif LOADB = '0' then Q <= D;
        elsif UP'event and UP = '1' and DOWN = '1' then
            if Q = 9 then Q <= 0;
            else Q <= Q + 1; end if;
        elsif DOWN'event and DOWN = '1' and UP = '1' then
            if Q = 0 then Q <= 9;

```

```

    else Q <= Q - 1; end if;
  end if;
end process;
-- borrow rises on rising edge of DOWN in state 0
BORROW <= '0' when DOWN = '0' and Q = 0 else '1';
-- carry rises on rising edge of UP in state 9
CARRY <= '0' when UP = '0' and Q = 9 else '1';
end behave;

```

2.33 (a) entity shift8 is

```

  port(Q: inout bit_vector(7 downto 0);
        D: in bit_vector(7 downto 0);
        CLR, CLK, S0, S1, LSI, RSI: in bit);
  end shift8;

```

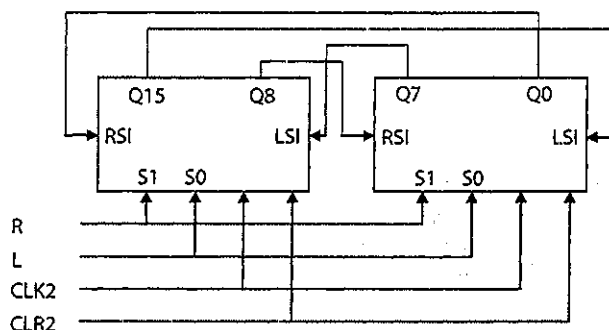
(b) architecture behave of shift8 is

```

begin
  process(CLK, CLR)
  begin
    if CLR = '1' then Q <= "00000000";
    elsif (CLK = '1' and CLK'event) then
      if (S0 = '1' and S1 = '1') then Q <= D;
      elsif (S0 = '0' and S1 = '1') then
        Q <= RSI & Q(7 downto 1);
      elsif (S0 = '1' and S1 = '0') then
        Q <= Q(6 downto 0) & LSI;
      end if;
    end if;
  end process;
end behave;

```

(c)



(d) entity shiftreg is

```

  port(QQ: inout bit_vector(15 downto 0);
        DD: in bit_vector(15 downto 0);
        CLK2, CLR2, L, R: in bit);
  end shiftreg;

```

(e) architecture struct of shiftreg is

```

  component shift8 is
    port(Q: inout bit_vector(7 downto 0);
          D: in bit_vector(7 downto 0);
          CLR, CLK, S0, S1, LSI, RSI: in bit);
  end component;
begin
  SR1: shift8 port map(QQ(15 downto 8), DD(15 downto 8),
    CLR2, CLK2, L, R, QQ(7), QQ(0));
  SR2: shift8 port map(QQ(7 downto 0), DD(7 downto 0),
    CLR2, CLK2, L, R, QQ(15), QQ(8));
end struct;

```

```

2.34 library IEEE;
    use IEEE.numeric_bit.all;

    entity countQ1 is
        port(clk, Ld8, Enable: in bit; S5: out bit;
              Q: out unsigned(3 downto 0));
    end countQ1;

    architecture counter of countQ1 is
        signal Qint: unsigned(3 downto 0);
    begin
        process(clk)
        begin
            if clk'event and clk = '1' then
                if Ld8 = '1' then Qint <= "1000";
                elsif Enable = '1' then
                    if Qint = "0011" then Qint <= "1000";
                    else Qint <= Qint - 1; end if;
                end if;
            end if;
        end process;
        S5 <= '1' when Qint <= "0101" else '0';
        Q <= Qint;
    end counter;

```

```

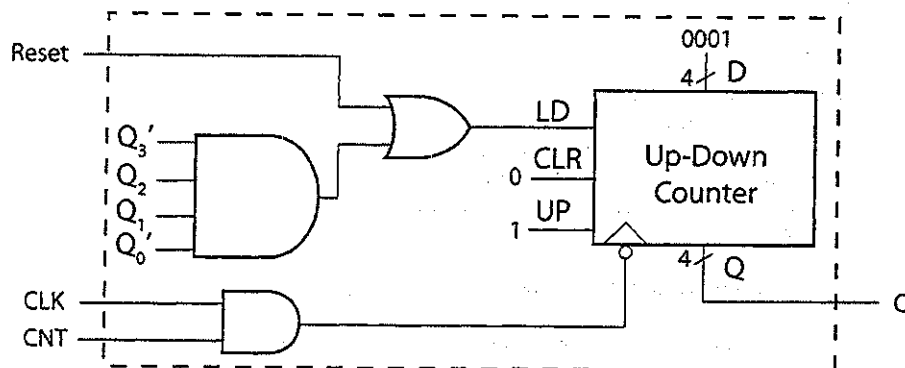
2.35 (a) library IEEE;
    use IEEE.numeric_bit.all;

    entity up_down is
        port(CLK, CLR, LD, UP: in bit; D: in unsigned(3 downto 0);
              Q: inout unsigned(3 downto 0));
    end up_down;

    architecture counter of up_down is
    begin
        process
        begin
            wait until CLK = '0' and CLK'event;
            if CLR = '1' then Q <= "0000";
            elsif LD = '1' then Q <= D;
            elsif UP = '1' then Q <= Q + 1;
            else Q <= Q - 1;
            end if;
        end process;
    end counter;

```

(b)



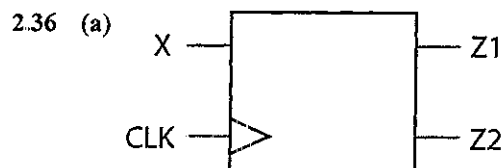
```

(c) library IEEE;
    use IEEE.numeric_bit.all;

    entity modulo6 is
        port(Q: inout unsigned(3 downto 0);
             CLK, Reset, CNI: in bit);
    end entity;

    architecture six of modulo6 is
        component up_down is
            port(CLK, CLR, LD, UP: in bit;
                 D: in unsigned(3 downto 0);
                 Q: inout unsigned(3 downto 0));
        end component;
        signal load, clock: bit;
    begin
        load <= Reset or (not Q(0) and Q(1) and Q(2) and not Q(3));
        clock <= CLK and CNI; --assume CNI changes when CLK is 0
        U0: up_down port map(CLOCK, '0', load, '1', "0001", Q);
    end six;

```



(b)

Present State	Next State		X = 0		X = 1	
	X = 0	X = 1	Z1	Z2	Z1	Z2
S0	S0	S1	1	0	0	0
S1	S1	S2	0	1	0	1
S2	S2	S3	0	1	0	1
S3	S0	S1	0	0	1	0

2.37 The following solutions utilize the solution for 1.13

```

(a) entity P2_37 is
    port(X, CLK: in bit;
         S, V: out bit);
end P2_37;

architecture Table of P2_37 is
    type StateTable is array
        (integer range <>, bit range <>) of integer;
    type OutTable is array
        (integer range <>, bit range <>) of bit_vector(1 downto 0);
    signal State, NextState: integer := 0;
    signal SV: bit_vector(1 downto 0);
    constant ST: StateTable (0 to 5, '0' to '1') :=
        ((1,1), (2,4), (3,3), (0,0), (3,5), (0,0));
    constant OT: OutTable (0 to 5, '0' to '1') :=
        (("00","10"), ("10","00"), ("00","10"), ("00","10"),
        ("10","00"), ("10","01"));
begin
    NextState <= ST(State, X);
    SV <= OT(State, X);
    S <= SV(1); -- Alternate method S <= OT(State,X)(1)
    V <= SV(0); -- V <= OT(State,X)(0)
    process(CLK)
    begin
        if CLK'event and CLK = '0' then
            State <= NextState;
        end if;
    end process;
end Table;

```

```

(b) entity P2_37 is
    port(X, CLK: in bit;
          S, V: out bit);
end P2_37;

architecture Equations of P2_37 is
    signal Q1, Q2, Q3: bit;
begin
    process(CLK)
    begin
        if CLK = '0' and CLK'event then
            Q1 <= not Q1 and Q3;
            Q2 <= (not Q2 and not Q3) or (X and not Q1 and Q2);
            Q3 <= (not Q1 and Q3) or (Q2 and not Q3);
        end if;
    end process;
    S <= (X and not Q2) or (not X and Q2);
    V <= (X and Q1 and Q2);
end Equations;

```

```

(c) entity P2_37 is
    port(X, CLK: in bit;
          S, V: out bit);
end P2_37;

architecture Structural of P2_37 is
    component DFF
        port(D, CLK: in bit; Q: out bit; QN: out bit := '1');
    end component;
    component And2
        port(A1, A2: in bit; Z: out bit);
    end component;
    component Or2
        port(A1, A2: in bit; Z: out bit);
    end component;
    component And3
        port(A1, A2, A3: in bit; Z: out bit);
    end component;
    component Inverter
        port(A: in bit; Z: out bit);
    end component;
    signal Q1, Q2, Q3: bit := '0';
    signal XN, Q1N, Q2N, Q3N: bit := '1';
    signal D1, D2, D3: bit := '0';
    signal A1, A2, A3, A4, A5, A6: bit := '0';
begin
    I1: Inverter port map (X, XN);
    G1: And2 port map (Q1N, Q3, D1);
    G2: And2 port map (Q2N, Q3N, A1);
    G3: And3 port map (X, Q1N, Q2, A2);
    G4: Or2 port map (A1, A2, D2);
    G5: And2 port map (Q1N, Q3, A3);
    G6: And2 port map (Q2, Q3N, A4);
    G7: Or2 port map (A3, A4, D3);
    G8: And2 port map (X, Q2N, A5);
    G9: And2 port map (XN, Q2, A6);
    G10: Or2 port map (A5, A6, S);
    G11: And3 port map (X, Q1, Q2, V);
    DFF1: DFF port map (D1, CLK, Q1, Q1N);
    DFF2: DFF port map (D2, CLK, Q2, Q2N);
    DFF3: DFF port map (D3, CLK, Q3, Q3N);
end Structural;

```

See Section 2.15 for the definition of the DFF component. The And3, And2, Or2, and Inverter components are all similar to the Nand3 component given on pages 109-110 (section 2.15).

2.38 The following solutions utilize the solution for 1.14

(a) entity P2_38a is

```
port(X, CLK: in bit;
      D, B: out bit);
```

end P2_38a;

architecture Table of P2_38a is

```
signal State, NextState: integer range 0 to 5;
```

begin

```
process(State, X)
```

```
begin
```

```
case State is
```

```
when 0 => if X = '0' then D <= '0'; B <= '0'; NextState <= 1;
```

```
else D <= '1'; B <= '0'; NextState <= 1; end if;
```

```
when 1 => if X = '0' then D <= '1'; B <= '0'; NextState <= 2;
```

```
else D <= '0'; B <= '0'; NextState <= 3; end if;
```

```
when 2 => if X = '0' then D <= '1'; B <= '0'; NextState <= 4;
```

```
else D <= '0'; B <= '0'; NextState <= 5; end if;
```

```
when 3 => if X = '0' then D <= '0'; B <= '0'; NextState <= 5;
```

```
else D <= '1'; B <= '0'; NextState <= 5; end if;
```

```
when 4 => if X = '0' then D <= '1'; B <= '1'; NextState <= 0;
```

```
else D <= '0'; B <= '0'; NextState <= 0; end if;
```

```
when 5 => if X = '0' then D <= '0'; B <= '0'; NextState <= 0;
```

```
else D <= '1'; B <= '0'; NextState <= 0; end if;
```

```
end case;
```

```
end process;
```

```
process(CLK)
```

```
begin
```

```
if (CLK = '0' and CLK'event) then
```

```
State <= NextState; end if;
```

```
end process;
```

end Table;

(b) entity P2_38b is

```
port(X, CLK: in bit;
      D, B: out bit);
```

end P2_38b;

architecture Equations of P2_38b is

```
signal Q1, Q2, Q3: bit;
```

begin

```
process(CLK)
```

```
begin
```

```
if CLK = '0' and CLK'event then
```

```
Q1 <= (not Q1 and not Q3) or (not X and Q1 and not Q2);
```

```
Q2 <= (not Q2 and Q3);
```

```
Q3 <= not Q2 and (Q3 or Q1);
```

```
end if;
```

```
end process;
```

```
D <= (not X and Q1) or (X and not Q1 and Q3);
```

```
B <= not X and Q1 and Q2;
```

end Equations;

(c) entity P2_38c is

```
port(X, CLK: in bit;
      D, B: out bit);
```

end P2_38c;

architecture Structure of P2_38c is

```
component JKFF
```

```
port(SN, RN, J, K, CLK: in bit; Q, QN: out bit);
```

```
end component;
```

```
component Nand2
```

```
port(A1, A2: in bit; Z: out bit);
```

```
end component;
```

```
component And3
```

```

    port(A1, A2, A3: in bit; Z: out bit);
end component;
component Nand3
    port(A1, A2, A3: in bit; Z: out bit);
end component;
component Inverter
    port(A: in bit; Z: out bit);
end component;
signal A1, A2, A3: bit := '0';
signal Q1, Q2, Q3: bit := '0';
signal Q1N, Q2N, Q3N, XN, One: bit := '1';
begin
    I1: Inverter port map (X, XN);
    G1: Nand2 port map (XN, Q2N, A1);
    FF1: JKFF port map ('1', '1', Q3N, A1, CLK, Q1, Q1N);
    FF2: JKFF port map ('1', '1', Q3, '1', CLK, Q2, Q2N);
    FF3: JKFF port map ('1', '1', Q1, Q2, CLK, Q3, Q3N);
    G2: Nand2 port map (XN, Q1, A2);
    G3: Nand3 port map (X, Q1N, Q3, A3);
    G4: Nand2 port map (A2, A3, D);
    G5: And3 port map (XN, Q1, Q2, B);
end Structure;

```

The Nand2, And3, and Inverter components are similar to the Nand3 component in Section 2.15. The JKFF component is similar to the DFF component in Section 2.15.

```

2.39 entity moore_mach is
    port(X1, X2: in bit;
        Clk: in bit;
        Z: out bit);
end moore_mach;

architecture moore_mach_bhv of moore_mach is
    signal state: natural := 1;
begin
    process(Clk)
    begin
        if Clk = '0' and Clk'event then
            case state is
                when 1 =>
                    if (X1 xor X2) = '1' then
                        state <= 2 after 10 ns;
                        Z <= '0' after 20 ns; end if;
                when 2 =>
                    if X2 = '1' then
                        state <= 1 after 10 ns;
                        Z <= '1' after 20 ns; end if;
                    when others => null;
                end case;
            end if;
        end process;
    end moore_mach_bhv;

```

```

2.40 entity P_40 is
    port(x1, x2, clk: in bit;
          z1, z2: out bit);
end P_40;

architecture behavioral of P_40 is
    signal state, next_state: integer range 1 to 3;
begin
    process(state, x1, x2)
    begin
        case state is
            when 1 => if ((x1 & x2) = "00") then next_state <= 3 after 10 ns;
                       elsif ((x1 & x2) = "01") then next_state <= 2 after 10 ns;
                       else next_state <= 1 after 10 ns; end if;
            when 2 => if ((x1 & x2) = "00") then next_state <= 2 after 10 ns;
                       elsif ((x1 & x2) = "01") then next_state <= 1 after 10 ns;
                       else next_state <= 3 after 10 ns; end if;
            when 3 => if ((x1 & x2) = "00") then next_state <= 1 after 10 ns;
                       elsif ((x1 & x2) = "01") then next_state <= 2 after 10 ns;
                       else next_state <= 3 after 10 ns; end if;
        end case;
    end process;
    process(clk)
    begin
        if clk = '0' and clk'event then
            state <= next_state after 5 ns; end if;
        end process;
        z1 <= '1' after 10 ns when state = 2 else '0' after 10 ns;
        z2 <= '1' after 10 ns when state = 3 else '0' after 10 ns;
    end behavioral;

```

- 2.41 (a) *nextstate* is not always assigned a new value in the conditional statements, else clauses are not specified so a latch will be created to hold *nextstate* to its old value

- (b) The latch output would have the most recent value of *nextstate*.

```

(c) process(state, X)
begin
    case state is
        when 0 => if X = '1' then nextstate <= 1;
                   else nextstate <= 0; end if;
        when 1 => if X = '0' then nextstate <= 2;
                   else nextstate <= 1; end if;
        when 2 => if X = '1' then nextstate <= 0;
                   else nextstate <= 2; end if;
    end case;
end process;

```

2.42

Signal or Variable	time	new value
A	20 ns	1
F	20 ns	6
A	20 ns	6
B	25 ns	7
C	30 ns	2
D	35 ns	5

Note: The change to A=1 is never visible.

- 2.43 *sel* should be a variable, instead of a signal. Otherwise *sel* will not update for current use. It updates only at the end of a process so the case statement will get the wrong value.

2.44

ns	Δ	A	B	D
0	+0	0	0	0
5	+0	1	0	0
15	+0	1	0	1
15	+1	1	1	1
25	+0	1	1	0
25	+1	1	0	0
35	+0	1	0	1
35	+1	1	1	1
40	+0	0	1	1

2.45 Rising-edge triggered toggle flip-flop (T-flip-flop), with asynchronous active-high clear signal

2.46 (a) library IEEE;

use IEEE.numeric_bit.all;

entity ROM4_3 is

port(ROMin: in unsigned(0 to 3);

ROMout: out unsigned(0 to 2));

end ROM4_3;

architecture Behavioral of ROM4_3 is

type ROM16x3 is array (0 to 15) of unsigned(0 to 2);

constant ROM1: ROM16x3 := ("000", "001", "001", "010", "001", "010",
"010", "011", "001", "010", "010", "011", "010", "011", "011", "100");

signal index: integer range 0 to 15;

begin

index <= to_integer(ROMin);

ROMout <= ROM1(index);

end Behavioral;

(b) library IEEE;

use IEEE.numeric_bit.all;

entity P_46 is

port(A: in unsigned(11 downto 0);

count: out unsigned(3 downto 0));

end P_46;

architecture Behavioral of P_46 is

component ROM4_3

port(ROMin: in unsigned(0 to 3);

ROMout: out unsigned(0 to 2));

end component;

signal B, C, D: unsigned(0 to 2);

begin

RO1: ROM4_3 port map (A(11 downto 8), B);

RO2: ROM4_3 port map (A(7 downto 4), C);

RO3: ROM4_3 port map (A(3 downto 0), D);

count <= '0' & B + C + D;

end Behavioral;

(c) A	Count	D	C	B
1111111111	1100	100	100	100
010110101101	0111	011	010	010
100001011100	0101	010	010	001

2.47 library IEEE;

use IEEE.numeric_bit.all;

entity decoder is

port(A, B, C: in bit;

X: out unsigned(7 downto 0));

end decoder;

a	b	c	y ₇	y ₆	y ₅	y ₄	y ₃	y ₂	y ₁	y ₀
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

architecture LUI of decoder is

type ROM is array (0 to 7) of unsigned(7 downto 0);

signal Table: ROM := ("00000001", "00000010", "00000100", "00001000",
"00010000", "00100000", "01000000", "10000000");

signal index: unsigned(2 downto 0);

begin

index <= A & B & C;

X <= Table(to_integer(index));

end LUI;

```

2.48 (a) process(A)
    variable Max: integer;
begin
    Max := A(1);
    for i in 2 to 20 loop
        if A(i) > Max then
            Max := A(i);
        end if;
    end loop;
end process;

```

```

(b) process(A)
    variable Max, i: integer;
begin
    Max := A(1);
    i := 2;
    while i <= 20 loop
        if A(i) > Max then
            Max := A(i);
        end if;
        i := i + 1;
    end loop;
end process;

```

```

2.49 architecture test1 of tester is
    component Mealy
        port(X, CLK: in bit; Z: out bit);
    end component;
    signal XA: bit_vector(0 to 11) := "011011011100";
    signal ZA: bit_vector(0 to 11) := "100110110110";
    signal X, clk, Z: bit := '0';
begin
    clk <= not clk after 50 ns;
    M1: Mealy port map(X, clk, Z);
    process
    begin
        for i in 0 to 11 loop
            X <= XA(i) after 10 ns; --start first output immediately
            wait until clk = '1' and clk'event;
            assert (Z = ZA(i))
                report "Error"
                severity error;
        end loop;
        report "sequence correct";
    end process;
end test1;

```

```

2.50 entity TestExcess3 is
end TestExcess3;

```

```

architecture test1 of TestExcess3 is
    component Code_Converter is
        port(X, CLK: in bit;
            Z: out bit);
    end component;
    type bv_arr is array(1 to 10) of bit_vector(3 downto 0);
    constant XA: bv_arr := ("0000", "0001", "0010", "0011", "0100",
        "0101", "0110", "0111", "1000", "1001");
    constant ZA: bv_arr := ("0011", "0100", "0101", "0110", "0111",
        "1000", "1001", "1010", "1011", "1100");
    signal X, Z, CLK: bit := '0';
begin
    CLK <= not CLK after 50 ns;
    C1: Code_Converter port map (X, CLK, Z);
    process
    begin
        for i in 1 to 10 loop
            for j in 0 to 3 loop
                X <= XA(i)(j); --start first output immediately
                wait until clk'event and clk = '1';
                wait for 10 ns; --wait for gate delay
                assert (Z = ZA(i)(j))
                    report "sequence incorrect"
                    severity error;
                wait for 15 ns; --input will change 10+15 ns after edge
            end loop;
        end loop;
    end process;
end test1;

```

```

        report "all sequences correct";
    end process;
end test1;

```

2.51 library IEEE;

```

use IEEE.numeric_bit.all;

```

```

entity testbench is
    port(time1: out time);
end testbench;

```

```

architecture test1 of testbench is

```

```

    signal clk, Ld8, Enable, S5: bit;

```

```

    signal Q: unsigned(3 downto 0);

```

```

    component countQ1 is

```

```

        port(clk, Ld8, Enable: in bit; S5: out bit;

```

```

            Q: out unsigned(3 downto 0));

```

```

    end component;

```

```

begin

```

```

    time1 <= now when S5 = '1' else 0 ns;

```

```

    clk <= not clk after 50 ns;

```

```

    Ld8 <= '1', '0' after 100 ns;

```

```

    Enable <= '0', '1' after 100 ns, '0' after 600 ns,

```

```

        '1' after 800 ns, '0' after 1800 ns;

```

```

    cnt1: countQ1 port map (clk, Ld8, Enable, S5, Q);

```

```

end test1;

```

2.52 entity testSMQ1 is

```

    port(correct: inout Boolean);

```

```

end testSMQ1;

```

```

architecture testSM of test SMQ1 is

```

```

    component SMQ1

```

```

        port(X, CLK: in bit; Z: out bit);

```

```

    end component;

```

```

    constant answer: bit_vector(1 to 5) := "11010";

```

```

    signal X, Z, CLK: bit;

```

```

begin

```

```

    clk <= not clk after 50 ns;

```

```

    X <= '1', '0' after 100 ns, '1' after 300 ns;

```

```

    SMQ1_1: SMQ1 port map (X, CLK, Z);

```

```

    process

```

```

    begin

```

```

        wait for 40 ns; --read output 10ns before rising edge of clock

```

```

        for i in 1 to 5 loop

```

```

            correct <= answer(i) = Z;

```

```

            wait for 100 ns;

```

```

            if correct = FALSE then exit; end if;

```

```

        end loop;

```

```

        wait;

```

```

    end process;

```

```

end testSM;

```