Chapter 2: Introduction to Verilog®

2.1 **(a)** HDL – Hardware Description Language
FPGA – Field Programmable Logic Array

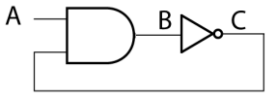
(b) Verilog has statements that execute concurrently since it must model real hardware in which the components are all in operation at the same time.

(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.

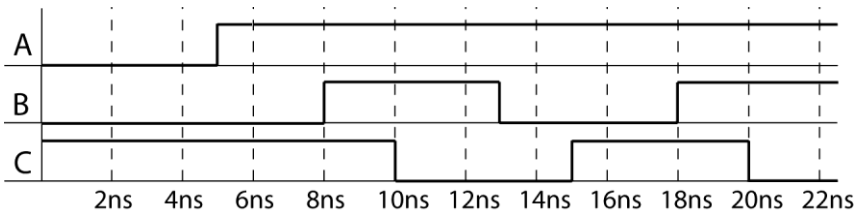
2.2 **(a)** Legal: `A_123`, `_A123`, `c1__c2`, `and1`; Illegal: `123A` (starts with number), `$A123_` (starts with $), `and` (reserved word).

(b) None of the Verilog identifiers are equivalent since Verilog is case sensitive.

2.3 **(a)**



(b)



2.4

```
module Gate(A, B, C, D, Z);
  input A, B, C, D;
  output Z;

  wire E, F, G, H, I;

  assign #5 H = A & B & C;
  assign #5 E = H | D;
  assign #5 G = ~(B | C);
  assign #5 F = ~(G & A);
  assign #2 I = ~F;
  assign #5 Z = E ^ I;
endmodule
```

2.5 **(a)**

```
module one_bit_sub(x, y, bin, diff, bout);
  input x, y, bin;
  output diff, bout;

  assign diff = x ^ y ^ bin;
  assign bout = (~x & bin) | (~x & y) | (bin & y);
endmodule
```

(b)

```
module four_bit_sub(a, b, bin, d, bout);
  input[3:0] a, b;
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Visit TestBankDeal.com to get complete for all chapters

```
    input bin;
    output[3:0] d;
    output bout;

    wire[3:0] bo;

    one_bit_sub s1(a[0], b[0], bin, d[0], bo[1]);
    one_bit_sub s2(a[1], b[1], bo[1], d[1], bo[2]);
    one_bit_sub s3(a[2], b[2], bo[2], d[2], bo[3]);
    one_bit_sub s4(a[3], b[3], bo[3], d[3], bout);
  endmodule
```

2.6 **(a)**

```
module circuit(A, B, C, D, G);
  input A, B, C, D;
  output G;

  wire E, F;

  assign E = A & B;
  assign F = E | C;
  assign G = D & F;
endmodule
```

(b)

```
module circuit(A, B, C, D, G);
  input A, B, C, D;
  output reg G;

  reg E, F;

  initial begin
    E <= 0;
    F <= 0;
    G <= 0;
  end

  always @(*)
  begin
    E <= A & B;
    F <= E | C;
    G <= F & D;
  end
endmodule
```

2.7 **(a)** *A* changes to 1 at 25 ns, *B* changes to 1 at 25 ns, *C* change to 1 at 35 ns
(b) *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

```
module SR_Latch(S, R, Q, Qn);
  input S, R;
  output reg Q;
  output Qn;

  initial begin
    Q <= 0;
  end
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
  always @(S, R)
  begin
    if(S == 1'b1)
      Q <= 1'b1;
    if(R == 1'b1)
      Q <= 1'b0;
  end

  assign Qn = ~Q;
endmodule
```

2.10

```
module MNFF(M, N, CLK, CLRn, Q, Qn);
  input M, N, CLK, CLRn;
  output reg Q;
  output Qn;

  initial begin
    Q <= 0;
  end

  always @(CLK, CLRn)
  begin
    if(CLRn == 1'b0)
    begin
      Q <= 0;
    end
    else if(CLK == 0)
    begin
      if(M == 0 && N == 0)
        Q <= ~Q;
      else if(M == 0 && N == 1)
        Q <= 1;
      else if(M == 1 && N == 0)
        Q <= 0;
      else if(M == 1 && N == 1)
        Q <= Q;
    end
  end

  assign Qn = ~Q;
endmodule
```

2.11

```
module DDFF(R, S, D, Clk, Q);
  input R, S, D, Clk;
  output reg Q;

  initial begin
    Q <= 0;
  end

  always @(Clk, R, S)
  begin
    if(R == 1'b0)
      Q <= 0;
    else if(S == 1'b0)
      Q <= 1;
    else
      Q <= D;
  end
endmodule
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

2.12 **(a)**

```
module ITFF(I0, I1, T, R, Q, QN);
  input I0, I1, T, R;
  output reg Q;
  output QN;

  initial begin
    Q <= 0;
  end

  always @(T, R)
  begin
    if(R == 1'b1)
      #5 Q <= 0;
    else begin
      if((I0 == 1'b1 && T == 1'b1) || (I1 == 1'b1 && T == 1'b0))
        #8 Q <= QN;
    end
  end

  assign QN = ~Q;
endmodule
```

(b)

```
add list *
add wave *
force T 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
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

2.14

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

i. 5'b10101
ii. 8'b11010101
iii. 3'b100

2.16

i. 4'b0000
ii. 8'b00001010
iii. 7'b0000101

© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

2.17

i. 8'h0D
ii. 8'hFD
iii. 8'h50
iv. 8'h50
v. 8'h0D
vi. 8'hFD
vii. 8'h50
viii. 8'h50

2.18

(a)
A >> 4 == 2'h0C
A >>> 4 == 2'hFC
A << 4 == 8'h70
A <<< 4 == 8'h70

(b)
A >> 4 == 2'h0C
A >>> 4 == 2'h0C
A << 4 == 8'h70
A <<< 4 == 8'h70

(c)
A >> 4 == 2'h0C
A >>> 4 == 2'h0C
A << 4 == 8'h70
A <<< 4 == 8'h70

(d)
A >> 4 == 8'h0C
A >>> 4 == 8'h0C
A << 4 == 8'h70
A <<< 4 == 8'h70

(e)
A >> 4 == 8'h0C
A >>> 4 == 8'h0C
A << 4 == 8'h70
A <<< 4 == 8'h70

(f)
A >> 4 == 8'h0C
A >>> 4 == 8'hFC
A << 4 == 8'h70
A <<< 4 == 8'h70

(g)
A >> 4 == 8'h0C
A >>> 4 == 8'hFC
A << 4 == 8'h70
A <<< 4 == 8'h70

(h)
A >> 4 == 32'h0FFFFFC
A >>> 4 == 32'hFFFFFFFC
A << 4 == 32'hFFFFFC70
A <<< 4 == 32'hFFFFFC70

2.19

i. 8'h0D
ii. 8'h0D
iii. 8'h50
iv. 8'h50
v. 8'h0D

© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

vi. 8'h0D
vii. 8'h50
viii. 8'h50

2.20
i. 8'h0D
ii. 8'h0D
iii. 8'h50
iv. 8'h50
v. 8'h0D
vi. 8'h0D
vii. 8'h50
viii. 8'h50

2.21 The synthesized hardware is a 4-bit shift register.

2.22 The synthesized hardware is a single flip-flop.

2.23 Both modules are synthesized to 4-bit shift registers. There are no differences between the two shift registers.

2.24 **(a)** D1 = 5, D2 = 1. The values of D1 and D2 swap.

(b) D1 = 1, D2 = 1. The values of D1 and D2 do not swap.

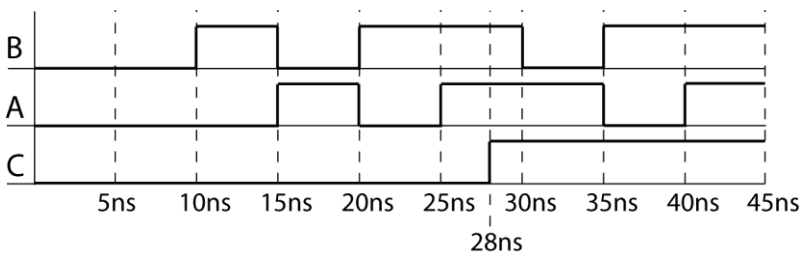
(c) iii

2.25 a; *y* must be in the sensitivity list, otherwise *sum* and *carry* will not update from changes to *y*

2.26

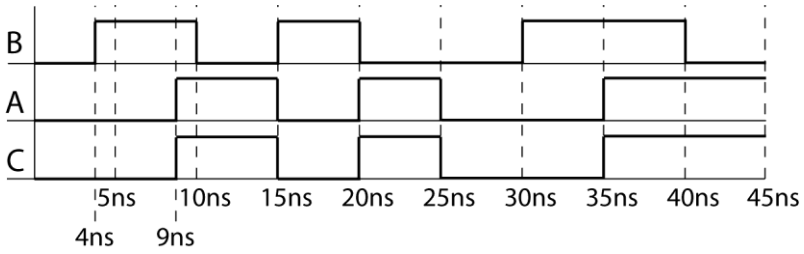
ns	Δ	a	b	c	d	e	f
0	+0	0	0	0	0	0	0
5	+0	0	0	0	1	0	0
5	+1	1	0	0	1	0	0
5	+2	1	1	1	1	0	0
6	+0	1	1	1	0	0	0
10	+0	1	1	1	0	1	0
10	+1	0	1	1	0	1	0
10	+2	0	0	0	0	1	0

2.27



© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

2.28



2.29 **(a)**

ns	a	b	c
0	1	0	0
4	1	1	0
5	1	1	1
10	1	0	1
15	1	1	0
20	1	0	1
25	1	0	0
30	1	1	0
35	1	1	1
40	1	0	1
45	1	0	0

(b)

ns	a	b	c
0	1	0	0
4	1	1	0
5	1	1	1
10	1	0	1
15	1	1	0
20	1	0	1
25	1	0	0
30	1	1	0
35	1	1	1
40	1	0	1
45	1	0	0

(c)

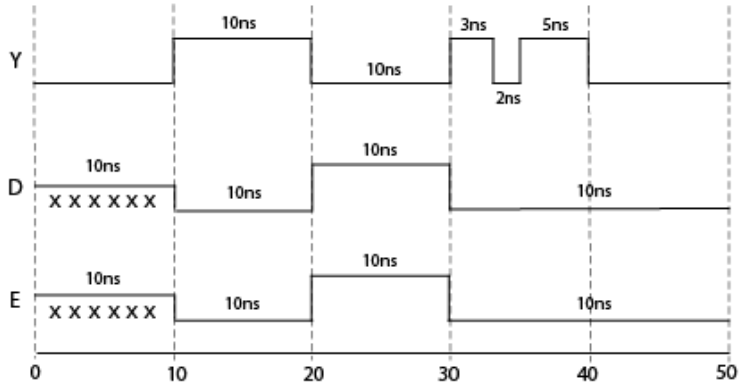
ns	a	b	c
0	1	0	0
4	1	1	0
9	1	1	1
10	1	0	1
15	1	1	0
20	1	0	1
25	1	0	0
30	1	1	0
35	1	1	1
40	1	0	1
45	1	0	0

© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

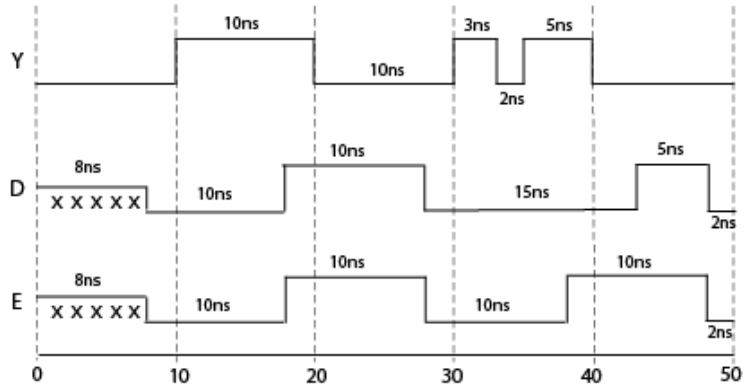
(d)

ns	a	b	c
0	1	0	0
4	1	1	0
10	1	0	0
15	1	1	0
20	1	0	1
25	1	0	0
30	1	1	0
35	1	1	1
40	1	0	1
45	1	0	0

2.30



2.31



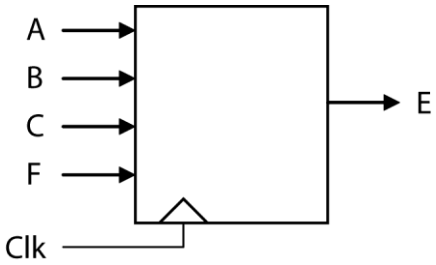
2.32

ns	Δ	a	b	c	d
4	+0	0	0	0	0
5	+0	1	0	0	0
10	+0	1	1	0	0
10	+1	0	1	0	0
11	+0	0	1	0	1
12	+0	0	1	1	1
15	+0	0	0	1	1

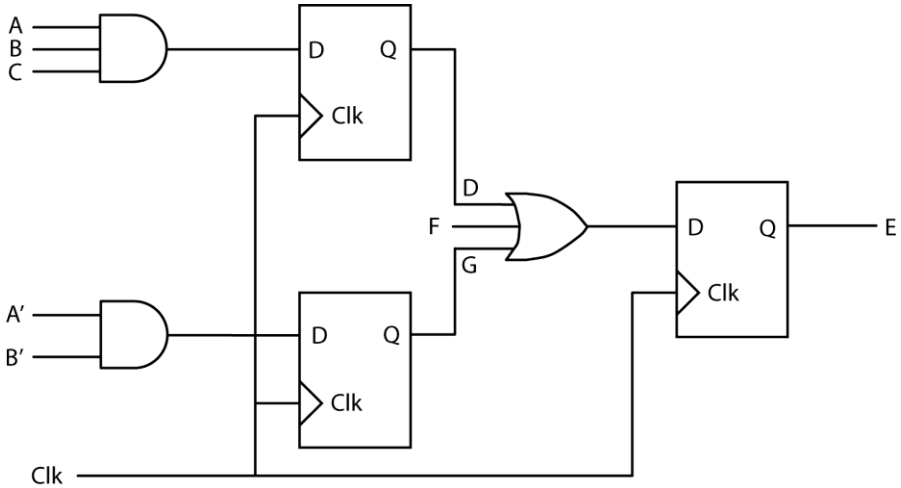
© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

2.33 **(a)** 6’b111011
(b) 3’b001
(c) 3’b001
(d) 1’b0
(e) 3'b111

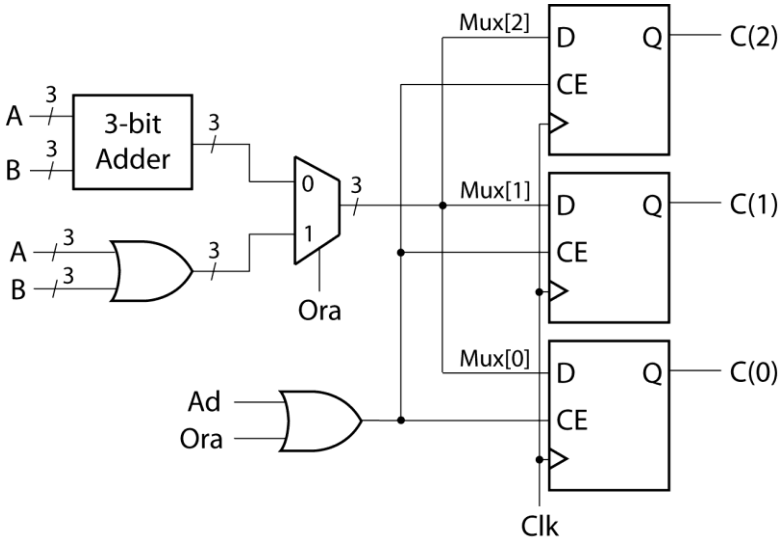
2.34 **(a)**



(b)

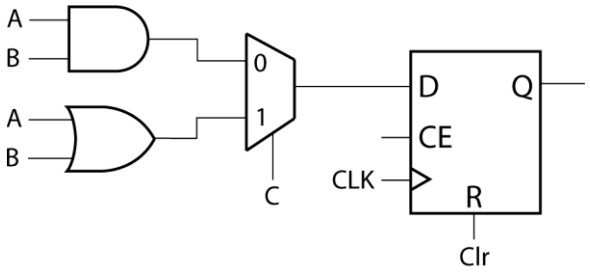


2.35



© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

2.36



Clr is asynchronous, whereas C affects a synchronous input to the D flip-flop.

2.37 **(a)**

```
assign #10 F = (C == 0)? ((D==0)? ~A: B):((D==0)? ~B: 0);
```

(b)

```
always @(*)
begin
  if(C==0 && D==0)
    #10 F = ~A;
  else if(C==0 && D==1)
    #10 F = B;
  else if(C==1 && D==0)
    #10 F = ~B;
  else
    #10 F = 0;
end
```

(c)

```
always @(*)
begin
   case(sel)
     0: #10 F = ~A;
     1: #10 F = B;
     2: #10 F = ~B;
     3: #10 F = 0;
   endcase
end
```

2.38 **(a)** module 1:

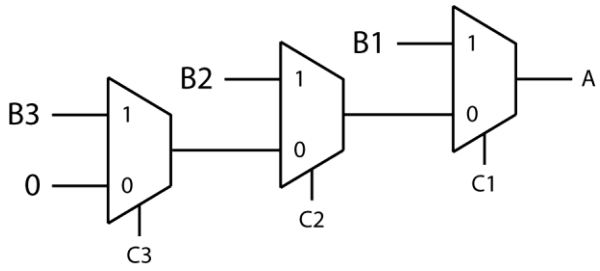
```
always @(C, B1, B2, B3)
begin
  if (C == 1)
    A <= B1;
  else if (C == 2)
    A <= B2;
  else if (C == 3)
    A <= B3;
  else
    A <= 0;
 end
```

module 2:

```
assign A = (C==1)? B1 : ((C==2)? B2 : ((C==3)? B3 : 0 ));
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

(b)



2.39 **(a)**

```
module SR_Latch(S, R, P, Q);
  input S, R;
  output P, Q;

  assign Q = (S)? 1 : ((R)? 0 : Q);
  assign P = ~Q;

endmodule
```

(b)

```
assign Q = S | (~R & Q);
assign P = ~Q;
```

(c)

```
assign Q = ~(R | P);
assign P = ~(S | Q);
```

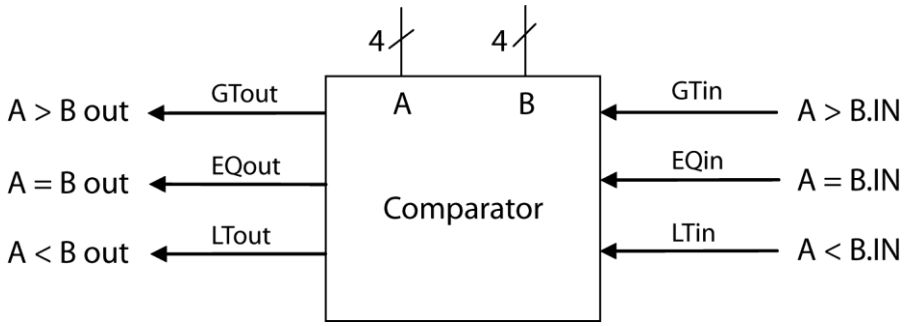
2.40

i. 8'b00010100
ii. 8'b00010100
iii. 8'b11111100
iv. 8'b11111100
v. 8'b00010100
vi. 8'b00010100

2.41

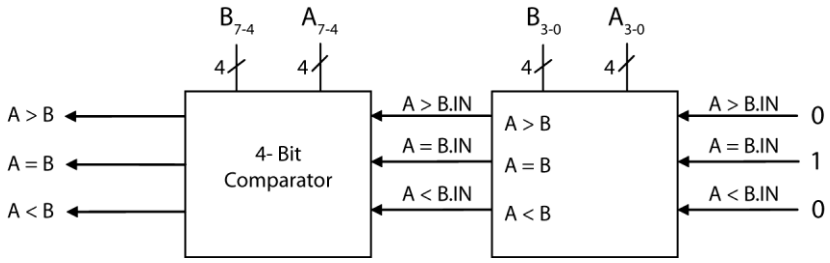
i. 32'hFFFFFF5B
ii. 32'hFFFFFF5B
iii. 32'h0000015B
iv. 32'h0000015B
v. 32'h0000005B
vi. 32'h0000005B

2.42 **(a)**



© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

(b)



(c)

```
module comp4bit(A, B, EQin, GTin, LTin, EQout, GTout, LTout);
  input[3:0] A, B;
  input EQin, GTin, LTin;
  output reg EQout, GTout, LTout;

  initial begin
    EQout = 0;
    GTout = 0;
    LTout = 0;
  end

  always @(A, B, EQin, GTin, LTin)
  begin
    if(A > B) begin
      EQout <= 0;
      GTout <= 1;
      LTout <= 0;
    end
    else if(A < B) begin
       EQout <= 0;
       GTout <= 0;
       LTout <= 1;
    end
    else if(GTin == 1) begin
       EQout <= 0;
       GTout <= 1;
       LTout <= 0;
    end
    else if(LTin == 1) begin
       EQout <= 0;
       GTout <= 0;
       LTout <= 1;
    end
    else begin
       EQout <= 1;
       GTout <= 0;
       LTout <= 0;
    end
  end
endmodule
```

(d)

```
module comp8bit(A, B, EQi, GTi, LTi, EQ, GT, LT);
  input[7:0] A, B;
  input EQi, GTi, LTi;
  output EQ, GT, LT;

  wire LowEQ, LowGT, LowLT;

  comp4bit C1(A[3:0], B[3:0], EQi, GTi, LTi, LowEQ, LowGT, LowLT);
  comp4bit C2(A[7:4], B[7:4], LowEQ, LowGT, LowLT, EQ, GT, LT);
endmodule
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

2.43

```
module shift_reg(SI, EN, CK, SO);
  input SI, EN, CK;
  output SO;
  reg[15:0] register;

  initial begin
    register <= 0;
  end

  always @(posedge CK)
  begin
    if(EN == 1)
      register <= {SI, register[15:1]};
  end

  assign SO = register[0];
endmodule
```

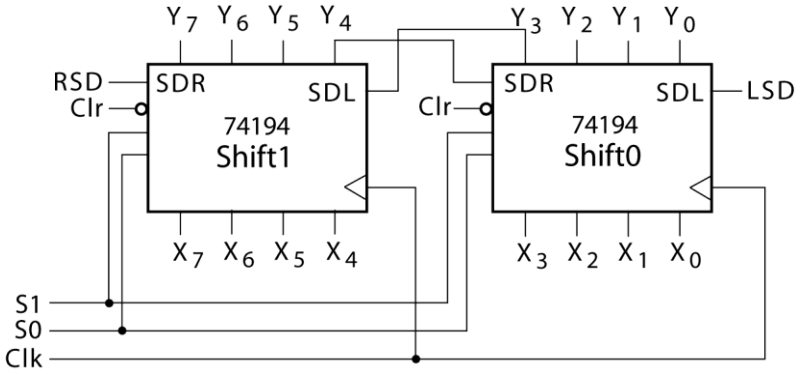
2.44 **(a)**

```
module shift74194(D, S, SDR, SDL, CLRb, CLK, Q);
  input[3:0] D;
  input[1:0] S;
  input SDR, SDL, CLRb, CLK;
  output reg [3:0] Q;

  initial begin
    Q <= 0;
  end

  always @(CLK, CLRb)
  begin
    if(CLRb == 0)
      Q <= 4'b0000;
    else if(CLK == 1)
    begin
      case(S)
      0: Q <= Q;
      1: Q <= {Q[2:0],SDL};
      2: Q <= {SDR, Q[3:1]};
      3: Q <= D;
      endcase
    end
  end
endmodule
```

(b)



© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
module bit8shift(X, S, RSD, LSD, CLRb, CLK, Y);
  input[7:0] X;
  input[1:0] S;
  input RSD, LSD, CLRb, CLK;
  output[7:0] Y;

  shift74194 S1(X[3:0], S, Y[4], LSD, CLRb, CLK, Y[3:0]);
  shift74194 S2(X[7:4], S, RSD, Y[3], CLRb, CLK, Y[7:4]);
endmodule
```

2.45 (a)

```
module Counter(D, CLK, CLR, ENT, ENP, UP, LOAD, Q, CO);
  input[3:0] D;
  input CLK, CLR, ENT, ENP, UP, LOAD;
  output reg[3:0] Q;
  output CO;

  initial begin
    Q = 0;
  end

  assign CO = ((ENT == 1) && ((UP == 1 && (Q == 4'b1001)) ||
                (UP == 0 && (Q == 4'b0000))));

  always @(CLK, CLR)
  begin
    if(CLR == 0)
      Q <= 0;
    else if(CLK == 1)
    begin
      if(LOAD == 0)
        Q <= D;
      else if(ENT == 1 && ENP == 1 && UP == 0) begin
        if(Q == 0)
          Q <= 4'b1001;
        else
          Q <= Q - 1;
      end
      else if(ENT == 1 && ENP == 1 && UP == 1) begin
        if(Q == 4'b1001)
          Q <= 0;
        else
          Q <= Q + 1;
      end
    end
  end
endmodule
```

(b)

```
module Century_Counter(Din1, Din2, CLK, CLR, ENT, ENP, UP, LOAD,
Count, CO);
  input [3:0] Din1, Din2;
  input CLK, CLR, ENT, ENP, UP, LOAD;
  output [7:0] Count;
  output CO;

  wire [3:0] Qout1, Qout2;
  wire Carry1, Carry2;

  Counter ct1(Din1, CLK, CLR, ENT, ENP, UP, LOAD, Qout1, Carry1);
  Counter ct2(Din2, CLK, CLR, ENT, Carry1, UP, LOAD, Qout2,
Carry2);
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
  assign Count = {Qout2, Qout1};
  assign CO = Carry2;
endmodule
```

The block diagram is similar to Figure 2-45 with an "Up" input added to each counter.

(c)

```
add wave *
force Din2 4'b1001
force Din1 4'b1000
force CLK 0 0 ns, 1 50 ns -repeat 100 ns
force CLR 1 0 ns, 0 1000 ns
force LOAD 0 0 ns, 1 100 ns
force ENT 1 0 ns, 0 400 ns, 1 600 ns
force ENP 1
force UP 1 0 ns, 0 500 ns
run 1200 ns
```

2.46 Students should look on the web for 74HC192 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.

```
module count74HC192(DOWN, UP, CLR, LOADB, BORROW, CARRY, D, Q);
  input DOWN, UP, CLR, LOADB;
  input[3:0] D;
  output BORROW, CARRY;
  output reg[3:0] Q;

  initial begin
    Q = 0;
  end

  always @(DOWN, UP, CLR, LOADB)
  begin
    if(CLR == 1)
      Q <= 0;
    else if(LOADB == 0)
      Q <= D;
    else if(DOWN == 1) begin
      @(posedge UP)
      if(Q == 4'b1001)
        Q <= 0;
      else
        Q <= Q + 1;
    end
    else if(UP == 1) begin
      @(posedge DOWN)
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
      if(Q == 0)
        Q <= 4'b1001;
      else
        Q <= Q - 1;
    end
  end

  assign BORROW = (DOWN == 0 && Q == 0)? 0 : 1;
  assign CARRY = (UP == 0 && Q == 4'b1001)? 0 : 1;
endmodule
```

2.47 **(a)**

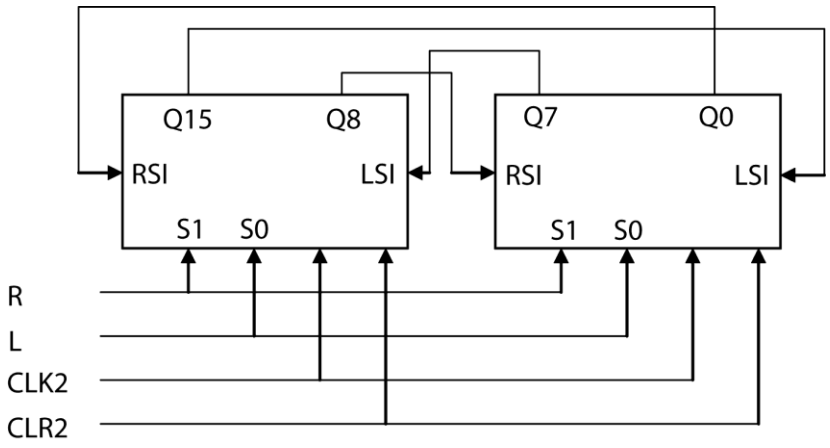
```
module shift8(Q, D, CLR, CLK, S0, S1, LSI, RSI);
  input[7:0] D;
  output reg[7:0] Q;
  input CLR, CLK, S0, S1, LSI, RSI;

  initial begin
    Q = 0;
  end

  always @(CLK, CLR)
  begin
    if(CLR == 1)
      Q <= 0;
    else if(CLK == 1) begin
      if(S0 == 1 && S1 ==1)
        Q <= D;
      else if(S0 == 0 && S1 == 1)
        Q <= {RSI, Q[7:1]};
      else if(S0 == 1 && S1 == 0)
        Q <= {Q[6:0], LSI};
      else
        Q <= Q;
    end
  end

endmodule
```

(b)



Note: D is not shown in the diagram.

(c)

```
module shiftreg(QQ, DD, CLK2, CLR2, L, R);
  input[15:0] DD;
  input CLK2, CLR2, L, R;
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
    output[15:0] QQ;

    shift8 SR1(QQ[15:8], DD[15:8], CLR2, CLK2, L, R, QQ[7], QQ[0]);
    shift8 SR2(QQ[7:0], DD[7:0], CLR2, CLK2, L, R, QQ[15], QQ[8]);
  endmodule
```

2.48

```
module countQ1(clk, Ld8, Enable, S5, Q);
  input clk, Ld8, Enable;
  output S5;
  output[3:0] Q;

  reg[3:0] Qint;

  initial begin
    Qint = 0;
  end

  always @(posedge clk)
  begin
    if(Ld8 == 1)
      Qint <= 4'b1000;
    else if(Enable == 1)
    begin
      if(Qint == 4'b0011)
        Qint <= 4'b1000;
      else
        Qint <= Qint - 1;
    end
  end

  assign S5 = (Qint == 4'b0101)? 1 : 0;
  assign Q = Qint;
endmodule
```

2.49 **(a)**

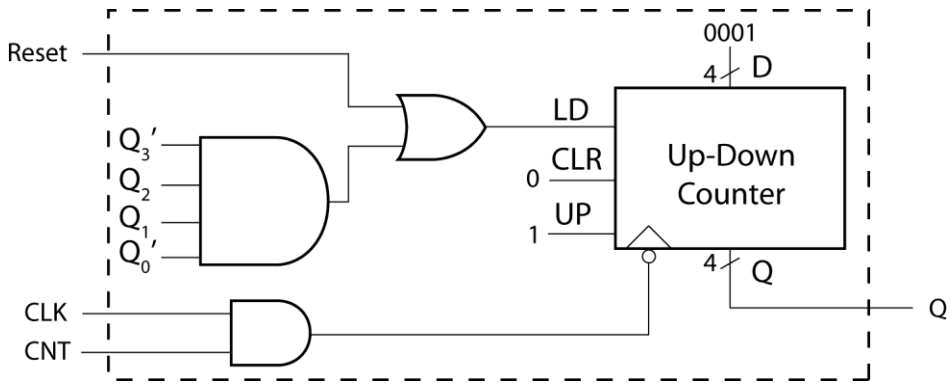
```
module up_down(CLK, CLR, LD, UP, D, Q);
  input CLK, CLR, LD, UP;
  input[3:0] D;
  output reg[3:0] Q;

  initial begin
    Q = 0;
  end

  always @(negedge CLK)
  begin
    if(CLR == 1)
      Q <= 4'b0000;
    else if(LD == 1)
      Q <= D;
    else if(UP == 1)
      Q <= Q + 1;
    else
      Q <= Q - 1;
  end
endmodule
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

(b)



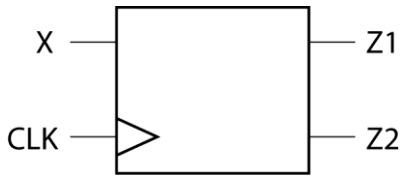
(c)

```
module modulo6(CLK, Reset, CNT, Q);
  input CLK, Reset, CNT;
  output[3:0] Q;
  wire load, clock;

  assign load = Reset | (~Q[0] & Q[1] & Q[2] & ~Q[3]);
  assign clock = CLK & CNT;

  up_down U1(clock, 1'b0, load, 1'b1, 4'b0001, Q);
endmodule
```

2.50 (a)



(b)

	Next State		X = 0		X = 1	
Present State	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

2.51 The following solutions utilize the solution for 1.13.

(a)

```
module P2_51a(X, CLK, S, V);
  input X, CLK;
  output S, V;

  reg [2:0] StateTable0 [5:0];
  reg [2:0] StateTable1 [5:0];
  reg [1:0] OutTable0 [5:0];
  reg [1:0] OutTable1 [5:0];

  reg [2:0] State;
  wire [2:0] NextState;
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
  initial begin
    StateTable0[0] <= 1; StateTable1[0] <= 1;
    StateTable0[1] <= 2; StateTable1[1] <= 4;
    StateTable0[2] <= 3; StateTable1[2] <= 3;
    StateTable0[3] <= 0; StateTable1[3] <= 0;
    StateTable0[4] <= 3; StateTable1[4] <= 5;
    StateTable0[5] <= 0; StateTable1[5] <= 0;
    OutTable0[0] <= 2'b00; OutTable1[0] <= 2'b10;
    OutTable0[1] <= 2'b10; OutTable1[1] <= 2'b00;
    OutTable0[2] <= 2'b00; OutTable1[2] <= 2'b10;
    OutTable0[3] <= 2'b00; OutTable1[3] <= 2'b10;
    OutTable0[4] <= 2'b10; OutTable1[4] <= 2'b00;
    OutTable0[5] <= 2'b10; OutTable1[5] <= 2'b01;
    State <= 0;
  end

  assign NextState = (X==0)? StateTable0[State] :
StateTable1[State];
  assign S = (X==0)? OutTable0[State][1] : OutTable1[State][1];
  assign V = (X==0)? OutTable0[State][0] : OutTable1[State][0];

  always @(negedge CLK)
  begin
    State <= NextState;
  end
endmodule
```

example simulation commands:

```
add wave *
force CLK 1 0 ns, 0 10 ns -repeat 20 ns
force X 1 0 ns, 0 15 ns, 1 35 ns, 0 75 ns, 1 95 ns, 0 175 ns
run 240 ns
```

(b)

```
module P2_51b(X, CLK, S, V);
  input X, CLK;
  output S, V;

  reg Q1, Q2, Q3;

  initial begin
    Q1 <= 0;
    Q2 <= 0;
    Q3 <= 0;
  end

  always @(negedge CLK)
  begin
    Q1 <= (~Q1 & Q3);
    Q2 <= (~Q2 & ~Q3) | (X & ~Q1 & Q2);
    Q3 <= (~Q1 & Q3) | (Q2 & ~Q3);
  end

  assign S = (X & ~Q2) | (~X & Q2);
  assign V = (X & Q1 & Q2);
endmodule
```

Read each set of outputs after 1/4 clock period before the falling edge of the clock but no later than the falling edge of the clock.

© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

(c)

```
module P2_51c(X, CLK, S, V);
  input X, CLK;
  output S, V;

  wire Q1, Q2, Q3;
  wire XN, Q1N, Q2N, Q3N;
  wire D1, D2, D3;
  wire A1, A2, A3, A4, A5, A6;

  Inverter I1(X, XN);
  And2 G1(Q1N, Q3, D1);
  And2 G2(Q2N, Q3N, A1);
  And3 G3(X, Q1N, Q2, A2);
  Or2 G4(A1, A2, D2);
  And2 G5(Q1N, Q3, A3);
  And2 G6(Q2, Q3N, A4);
  Or2 G7(A3, A4, D3);
  And2 G8(X, Q2N, A5);
  And2 G9(XN, Q2, A6);
  Or2 G10(A5, A6, S);
  And3 G11(X, Q1, Q2, V);
  DFF DFF1(D1, CLK, Q1, Q1N);
  DFF DFF2(D2, CLK, Q2, Q2N);
  DFF DFF3(D3, CLK, Q3, Q3N);
endmodule
```

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).

2.52 The following solutions utilize the solution for 1.14.

(a)

```
module P2_52a(X, CLK, D, B);
  input X, CLK;
  output reg D, B;
  reg[2:0] State, NextState;

  initial begin
    State <= 0;
    NextState <= 0;
  end

  always @(State, X)
  begin
    case(State)
    0:begin
      if(X == 0)
      begin
        D <= 0;
        B <= 0;
        NextState <= 1;
      end
      else begin
        D <= 1;
        B <= 0;
        NextState <= 1;
      end
    end
    1:begin
      if(X == 0)
      begin
        D <= 1;
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
        B <= 0;
        NextState <= 2;
      end
      else begin
        D <= 0;
        B <= 0;
        NextState <= 3;
      end
    end
    2: begin
      if(X == 0)
      begin
        D <= 1;
        B <= 0;
        NextState <= 4;
      end
      else begin
        D <= 0;
        B <= 0;
        NextState <= 5;
      end
    end
    3: begin
      if(X == 0)
      begin
        D <= 0;
        B <= 0;
        NextState <= 5;
      end
      else begin
        D <= 1;
        B <= 0;
        NextState <= 5;
      end
    end
    4: begin
      if(X == 0)
      begin
        D <= 1;
        B <= 1;
        NextState <= 0;
      end
      else begin
        D <= 0;
        B <= 0;
        NextState <= 0;
      end
    end
    5: begin
      if(X == 0)
      begin
        D <= 0;
        B <= 0;
        NextState <= 0;
      end
      else begin
        D <= 1;
        B <= 0;
        NextState <= 0;
      end
    end
    endcase
  end
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
  always @(negedge CLK)
  begin
    State <= NextState;
  end
endmodule
```

example simulation commands:

```
add wave *
force CLK 1 0 ns, 0 10 ns -repeat 20 ns
force X 1 0 ns, 0 15 ns, 1 35 ns, 0 75 ns, 1 95 ns, 0 175 ns
run 240 ns
```

(b)

```
module P2_52b(X, CLK, D, B);
  input X, CLK;
  output D, B;

  reg Q1, Q2, Q3;

  initial begin
    Q1 <= 0;
    Q2 <= 0;
    Q3 <= 0;
  end

  always @(negedge CLK)
  begin
    Q1 <= (~Q1 & ~Q3) | (~X & Q1 & ~Q2);
    Q2 <= (~Q2 & Q3);
    Q3 <= ~Q2 & (Q3 | Q1);
  end

  assign D = (~X & Q1) | (X & ~Q1 & Q3);
  assign B = ~X & Q1 & Q2;
endmodule
```

Read each set of outputs after 3/4 clock period before the falling edge of the clock but no later than the falling edge of the clock.

(c)

```
module PC_52c(X, CLK, D, B);
  input X, CLK;
  output D, B;
  wire A1, A2, A3;
  wire Q1, Q2, Q3;
  wire Q1N, Q2N, Q3N, XN, One;
  parameter I = 1;

  Inverter I1(X, XN);
  Nand2 G1(XN, Q2N, A1);
  JKFF FF1(I, I, Q3N, A1, CLK, Q1, Q1N);
  JKFF FF2(I, I, Q3, I, CLK, Q2, Q2N);
  JKFF FF3(I, I, Q1, Q2, CLK, Q3, Q3N);
  Nand2 G2(XN, Q1, A2);
  Nand3 G3(X, Q1N, Q3, A3);
  Nand2 G4(A2, A3, D);
  And3 G5(XN, Q1, Q2, B);
endmodule
```

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.

© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

2.53

```
module moore_mach(X1, X2, Clk, Z);
  input X1, X2;
  input Clk;
  output Z;

  reg[1:0] state;

  initial begin
    state <= 1;
  end

  always @(negedge Clk)
  begin
    case(state)
    1: begin
      if((X1 ^ X2) == 1)begin
        #10 state <= 2;
      end
    end
    2: begin
      if(X2 == 1)begin
        #10 state <= 1;
      end
    end
    default: begin
    end
    endcase
  end

  assign #10 Z = state[1];
endmodule
```

2.54

```
module P_54(x1, x2, clk, z1, z2);
  input x1, x2, clk;
  output z1, z2;

  reg[1:0] state, next_state;

  initial begin
    state <= 1;
    next_state <= 1;
  end

  always @(state, x1, x2)
  begin
    case(state)
    1: begin
      if({x1,x2} == 2'b00)
        #10 next_state <= 3;
      else if({x1,x2} == 2'b01)
        #10 next_state <= 2;
      else
        #10 next_state <= 1;
    end
    2: begin
      if({x1,x2} == 2'b00)
        #10 next_state <= 2;
      else if({x1,x2} == 2'b01)
        #10 next_state <= 1;
      else
        #10 next_state <= 3;
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
    end
    3: begin
      if({x1,x2} == 2'b00)
        #10 next_state <= 1;
      else if({x1,x2} == 2'b01)
        #10 next_state <= 2;
      else
        #10 next_state <= 3;
    end
    endcase
  end

  always @(negedge clk)
  begin
    #5 state <= next_state;
  end

  assign #10 z1 = (state == 2)? 1: 0;
  assign #10 z2 = (state == 3)? 1: 0;
endmodule
```

2.55 **(a)** *nextstate* is not always assigned a new value in the conditional statements, i.e. else clauses are not specified. Therefore, 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)

```
always @(state, X)
begin
  case(state)
  0: begin
    if(X == 1)
      nextstate <= 1;
    else
      nextstate <= 0;
  end
  1: begin
    if(X == 0)
      nextstate <= 2;
    else
      nextstate <= 1;
  end
  2: begin
    if(X == 1)
      nextstate <= 0;
    else
      nextstate <= 2;
  end
  endcase
end
```

(d) Yes, unconditionally set *nextstate* to a default value at the beginning of the always block

2.56 The nonblocking assignments must be changed to blocking assignments. Otherwise *sel* will not update for current use. *sel* updates only at the end of the process so the case statement will get the wrong value.

© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

2.57

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

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

2.59 **(a)**

```
module ROM4_3(ROMin, ROMout);
  input[3:0] ROMin;
  output[2:0] ROMout;

  reg[2:0] ROM16X3 [15:0];

  initial begin
    ROM16X3[0] <= 3'b000;
    ROM16X3[1] <= 3'b001;
    ROM16X3[2] <= 3'b001;
    ROM16X3[3] <= 3'b010;
    ROM16X3[4] <= 3'b001;
    ROM16X3[5] <= 3'b010;
    ROM16X3[6] <= 3'b010;
    ROM16X3[7] <= 3'b011;
    ROM16X3[8] <= 3'b001;
    ROM16X3[9] <= 3'b010;
    ROM16X3[10] <= 3'b010;
    ROM16X3[11] <= 3'b011;
    ROM16X3[12] <= 3'b010;
    ROM16X3[13] <= 3'b011;
    ROM16X3[14] <= 3'b011;
    ROM16X3[15] <= 3'b100;
  end

  assign ROMout = ROM16X3[ROMin];
endmodule
```

(b)

```
module P_59(A, count);
  input[11:0] A;
  output[3:0] count;

  wire[2:0] B,C,D;

  ROM4_3 R1(A[11:8], B);
  ROM4_3 R2(A[7:4], C);
  ROM4_3 R3(A[3:0], D);

  assign count = {1'b0, B} + C + D;
endmodule
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

(c)

A	Count	D	C	B
111111111111	1100	100	100	100
010110101101	0111	011	010	010
100001011100	0101	010	010	001

2.60

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

```
module decoder(A, B, C, Y);
  input A,B,C;
  output[7:0] Y;
  wire[2:0] index;
  reg[7:0] ROM[0:7];

  initial begin
    ROM[0] <= 8'b00000001;
    ROM[1] <= 8'b00000010;
    ROM[2] <= 8'b00000100;
    ROM[3] <= 8'b00001000;
    ROM[4] <= 8'b00010000;
    ROM[5] <= 8'b00100000;
    ROM[6] <= 8'b01000000;
    ROM[7] <= 8'b10000000;
  end

  assign index = {A,B,C};
  assign Y = ROM[index];
endmodule
```

2.61 **(a)**

```
always
begin
  MAX = A[0];
  for (i = 0; i < 20; i = i + 1) begin
    if(A[i] > MAX)
      MAX = A[i];
  end
end
```

(b)

```
always
begin
  MAX = A[0];
  i = 1;
  while (i < 20) begin
    if(A[i] > MAX)
      MAX = A[i];
    i = i + 1;
  end
end
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

2.62

```
module tester;
  reg CLK;
  reg [0:11] X;
  reg [0:11] Z;

  initial begin
    X = 12'b011011011100;
    Z = 12'b100110110110;
    CLK = 1;
  end

  reg Xin;
  wire Zout;
  integer i;

  always
    #50 CLK = ~CLK;

  Mealy M1(Xin, CLK, Zout);

  always
  begin
    for(i = 0; i < 12; i = i + 1) begin
      @(posedge CLK)
        #10 Xin = X[i];
        #80 // wait to read output
      if( Zout != Z[i]) begin
        $display("Error");
        $stop;
      end
    end
    $display("sequence correct");
    $stop;
  end
endmodule
```

2.63

```
module TestExcess3;
  reg[3:0] XA[1:10];
  reg[3:0] ZA[1:10];
  reg X, CLK;
  wire Z;
  integer i;
  integer j;

  initial begin
    X = 0;
    CLK = 0;
    XA[1] = 4'b0000;
    XA[2] = 4'b0001;
    XA[3] = 4'b0010;
    XA[4] = 4'b0011;
    XA[5] = 4'b0100;
    XA[6] = 4'b0101;
    XA[7] = 4'b0110;
    XA[8] = 4'b0111;
    XA[9] = 4'b1000;
    XA[10] = 4'b1001;
    ZA[1] = 4'b0011;
    ZA[2] = 4'b0100;
    ZA[3] = 4'b0101;
    ZA[4] = 4'b0110;
    ZA[5] = 4'b0111;
    ZA[6] = 4'b1000;
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
    ZA[7] = 4'b1001;
    ZA[8] = 4'b1010;
    ZA[9] = 4'b1011;
    ZA[10] = 4'b1100;
  end

  always
    #50 CLK <= ~CLK;

  Code_Converter C1(X, CLK, Z);

  always
  begin
    for(i = 1 ; i <= 10; i = i + 1)
    begin
      for(j= 0; j <= 3; j = j + 1)
      begin
        X = XA[i][j];
        @(posedge CLK);
        #(25);
        if(ZA[i][j] != Z) begin
          $display("sequence incorrect");
          $stop;
        end
      end
    end
    $display("all sequences correct");
    $stop;
  end
endmodule
```

2.64

```
module testbench;
  wire S5;
  reg clk, Ld8, Enable;
  wire[3:0] Q;

  initial begin
    clk = 1;
    Ld8 = 1;
    Enable = 0;
    #100
    Ld8 = 0;
    Enable = 1;
    #500 Enable = 0;
    #200 Enable = 1;
    #1000 Enable = 0;
  end

  always
    #50 clk <= ~clk;

  always @(posedge S5)
      $display($time);

  countQ1 C1(clk, Ld8, Enable, S5, Q);
endmodule
```

2.65

```
module testSMQ1(correct);
  output reg correct;
  reg clk, X;
  integer i;
  wire Z;
  reg[1:5] answer;
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

```
  initial begin
    answer[1] = 1;
    answer[2] = 1;
    answer[3] = 0;
    answer[4] = 1;
    answer[5] = 0;
    clk = 0;
    X = 1;
    #100 X = 0;
    #200 X = 1;
  end

  always
    #50 clk <= ~clk;

  SMQ1 S1(X, clk, Z);

  always
  begin
    for(i = 1; i <= 5; i = i + 1)
    begin
      @(posedge clk);
      correct = (answer[i] == Z);
      if(correct == 0)
        $display($time);
      #10;
    end
    $stop;
  end
endmodule
```


© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

© 2016 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Visit TestBankDeal.com to get complete for all chapters