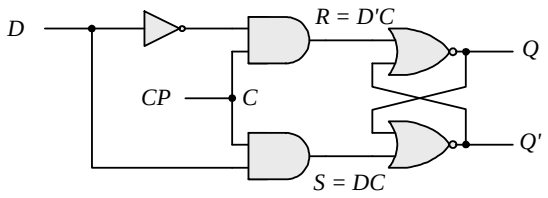
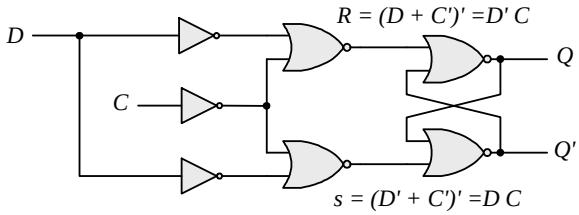
CHAPTER 5

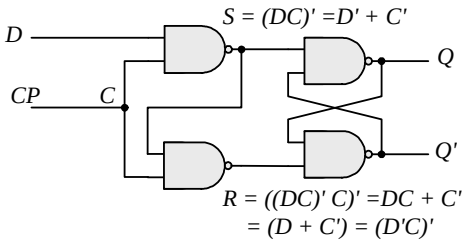
5.1 **(a)**



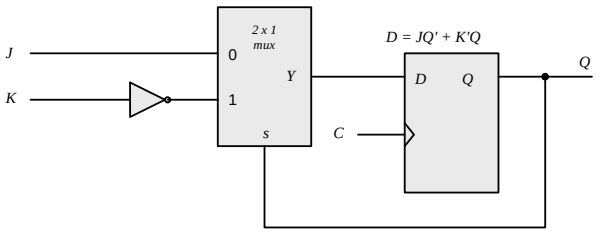
(b)



(c)



5.2



5.3 For T – Flip-Flop

$$Q(t+1) = TQ' + T'Q = T \oplus Q$$

$$Q'(t+1) = [T \oplus Q]'$$

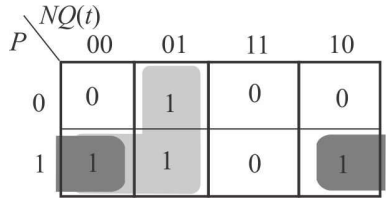
$$= T'Q' + TQ$$

5.4 **(a)**

PN	$Q(t+1)$
0 0	$Q(t)$
0 1	0
1 0	1

1 1 $Q'(t)$

(b)

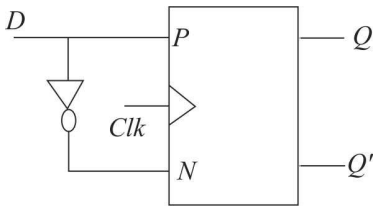


$Q(t + 1) = PN' + PQ(t)' + N'Q(t)$

(c)

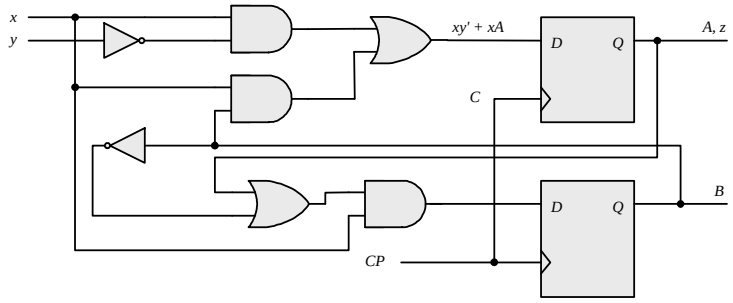
$Q(t)$	$Q(t + 1)$	P	N
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

(d)



5.5 State table is also called as transition table.
The truth table describes a combinational circuit.
The state table describes a sequential circuit.
The characteristic table describes the operation of a flip-flop.
The excitation table gives the values of flip-flop inputs for a given state transition.
The four equations correspond to the algebraic expression of the four tables.

5.6

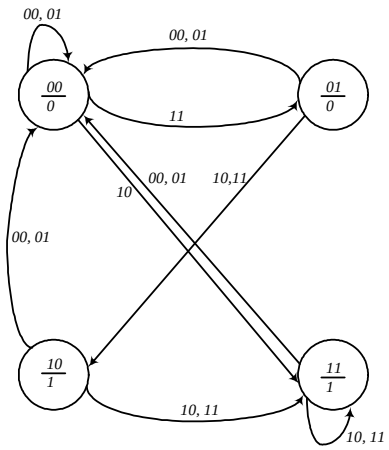


(b)

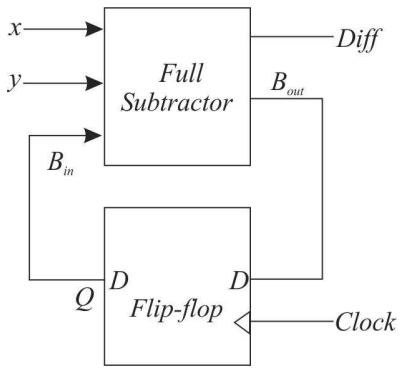
$A(t+1) = xy' + xB$
$B(t+1) = xA + xB'$
$z = A$

Present state		Inputs		Next state		Output
A	B	x	y	A	B	z
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

(c)



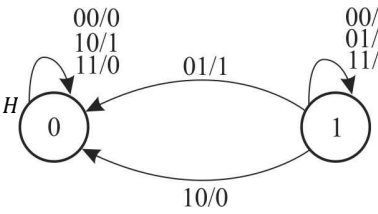
5.7



State Table:

Q(t)	x	y	Q(t + 1)	Diff.
0	0	0	0	0

State Diagram:



0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

$Diff. = Q(t)'(x \oplus y) + Q(t)(x \oplus y)'$

$= Q(t) \oplus x \oplus y$

5.8 A counter with a repeated sequence of 00, 01, 10.

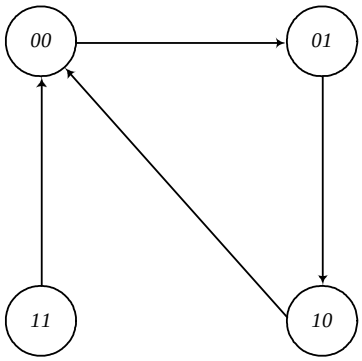
Present state		*Next state*		*FF Inputs*	
A	*B*	*A*	*B*	T_A	T_B
0	0	0	0	0	1
0	1	1	0	1	1
1	0	0	0	1	0
1	1	0	0	1	1

$T_A = A + B$

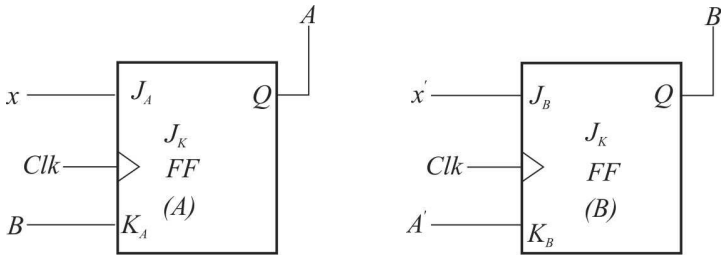
$T_B = A' + B$

Repeated sequence:

00 → 01 → 10



5.9 (a)



$A(t)$	$B(t)$	x	$A(t+1)$	$B(t+1)$

0	0	0	0	1
0	0	1	1	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

$A(t + 1)$:

A \ Bx	00	01	11	10
0	0	1	1	0
1	1	1	0	0

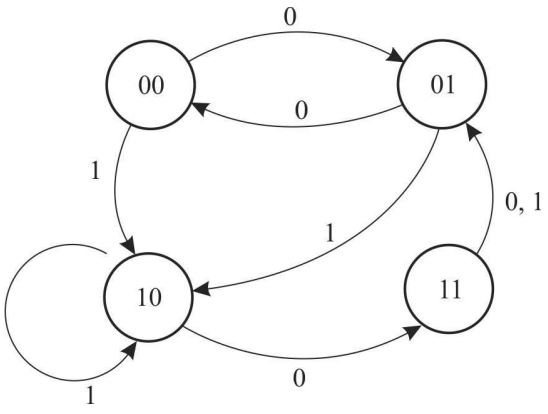
$A(t + 1) = A'x + AB'$

$B(t + 1)$:

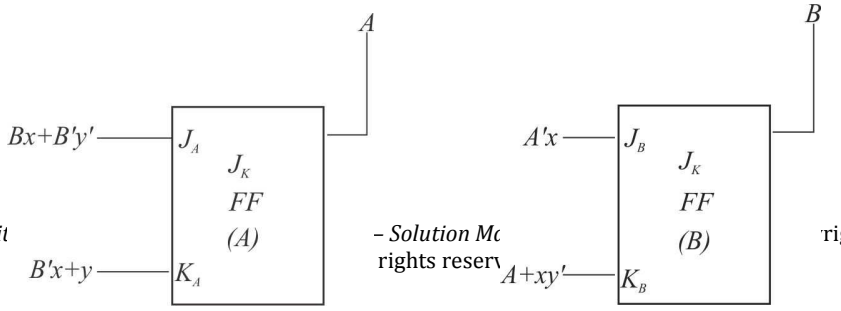
A \ Bx	00	01	11	10
0	1	0	0	0
1	1	0	1	1

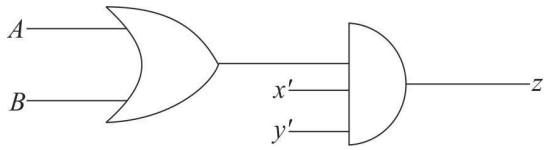
$B(t + 1) = B'x' + AB$

(b)



5.10 **(a)**





(b) State Table:

$A(t)$	$B(t)$	x	y	$A(t+1)$	$B(t+1)$	z
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	1	0	1	1	0
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
0	1	1	1	1	1	0
1	0	0	0	1	0	1
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	1	0	1
1	1	0	1	0	0	0
1	1	1	0	1	0	0
1	1	1	1	0	0	0

(c) $A(t+1)$: $\Sigma(0, 2, 6, 7, 8, 12, 14)$

AB \ xy	00	01	11	10
00	1	0	0	1
01	0	0	1	1
11	1	0	0	1
10	1	0	0	0

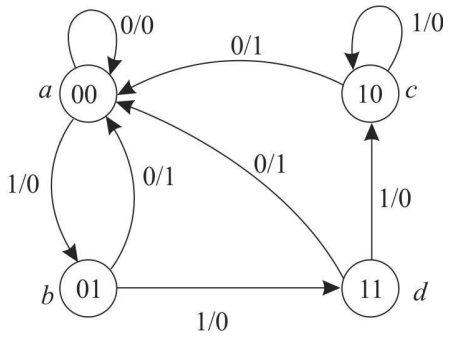
$= A\,x'y' + Bxy' + A'Bx + A'B'y'$

$B(t+1)$: $\Sigma(2, 3, 4, 5, 7)$

AB \ xy	00	01	11	10
00	0	0	1	1
01	1	1	1	0
11	0	0	0	0
10	0	0	0	0

$= A'Bx' + Axy + A'B'x$

5.11 **(a)**



$00 \rightarrow a$, $01 \rightarrow b$, $10 \rightarrow c$, $11 \rightarrow d$

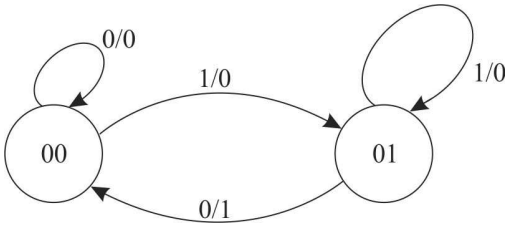
State	*a*	*b*	*d*	*a*	*b*	*d*	*c*	*a*	*d*
Input	1	1	0	1	1	1	0	0	1
Output	0	0	1	0	0	0	1	0	0

(b)

Present State		Input	Next State		Output
A(*t*)	*B*(*t*)	*x*	*A*(*t* + 1)	*B*(*t* + 1)	*z*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

OR

Present	Next		O/p	
	x = 0	*x* = 1	*x* = 0	*x* = 1
00	00	01	0	0
01	00	01	1	0
01	00	10	1	0
10	00	10	1	0



(c)

Present State		Input	Next State		z					
$A(t)$	$B(t)$	x	$A(t+1)$	$B(t+1)$		J_A	K_A	J_B	K_B	
0	0	0	0	0	0	0	0	-	0	-
0	0	1	0	1	1	0	0	-	1	-
0	1	0	0	0	0	1	0	-	-	1
0	1	1	1	0	1	1	0	0	-	-
1	0	0	-	-	-	-	-	-	-	-
1	0	1	-	-	-	-	-	-	-	-
1	1	0	-	-	-	-	-	-	-	-
1	1	1	-	-	-	-	-	-	-	-

$J_A = A$ $K_A = 1$

$J_B =$

A \ Bx	00	01	11	10
0	0	1	-	-
1	-	-	-	-

$= x$

K_B:

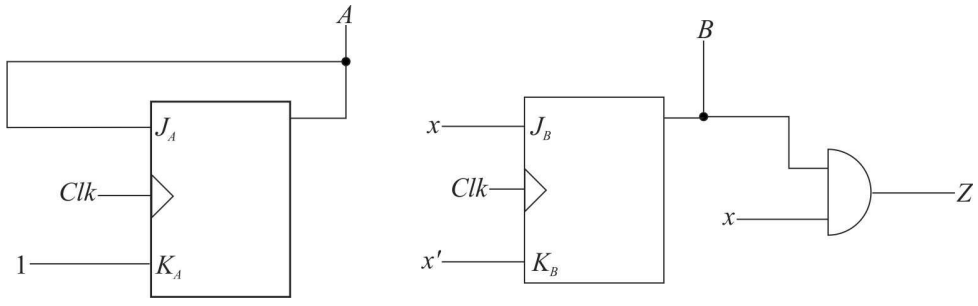
A \ Bx	00	01	11	10
0	-	-	0	1
1	-	-	-	-

$K_B = x'$

z:

A \ Bx	00	01	11	10
0	0	0	0	1
1	-	-	-	-

$= Bx'$



5.12

Present state	*Next state* 0	1	*Output* 0	1
a	*f*	*b*	0	0

b	*d*	*a*	0 0
D	*g*	*a*	1 0
F	*f*	*b*	1 1
G	*g*	*d*	0 1

5.13 **(a)**

State	*a*	*f*	*a*	*f*	*a*	*f*	*g*	*b*	*g*	*b*	*a*
Input	0	1	0	1	0	0	1	0	1	1	1
Output	0	0	0	0	0	1	0	1	0	0	0

(b)

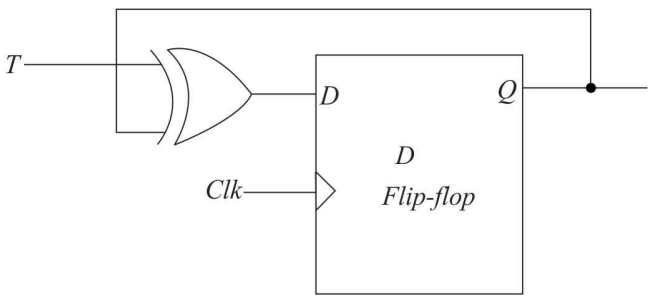
State	*a*	*b*	*a*	*b*	*a*	*b*	*g*	*b*	*g*	*b*	*a*
Input	0	1	0	1	0	0	1	0	1	1	1
Output	0	0	0	0	0	1	0	1	0	0	0

5.14

State	Assignment3
a	00001
b	00010
c	00100
d	01000
e	10000

Present State					*Next State*		*Output*	
	A	*B C D E*			*x* = 0	*x* = 1	*x* = 0	*x* = 1
a	0	0 0	0	1	00001	00010	0	0
b	0	0 0	1	0	00100	01000	0	0
c	0	0 1	0	0	00001	01000	0	0
d	0	1 0	0	0	10000	01000	0	1
e	1	0 0	0	0	00001	01000	0	1

5.15



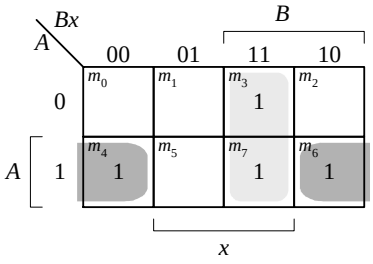
Present State (Q)	Input (T)	Next State (Q)
0	0	0
0	1	1
1	0	1
1	1	0

$Q(t + 1) = T \oplus Q(t)$

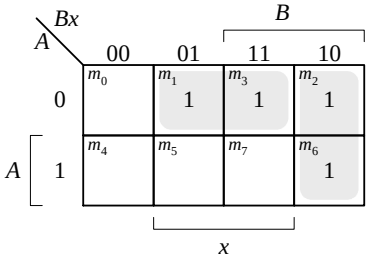
5.16 **(a)** $D_A = Ax' + Bx$

$D_B = A'x + Bx'$

Present state		*Input*	*Next state*	
A	B	x	A	B
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0



$D_A = Ax' + Bx$

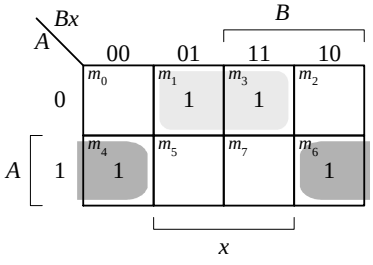


$D_B = A'x + Bx'$

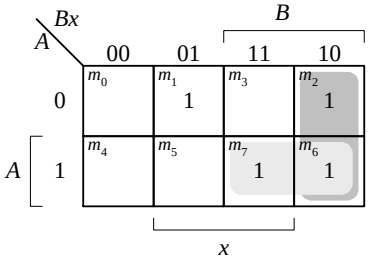
(b) $D_A = A'x + Ax'$

$D_B = AB + Bx'$

Present state		*Input*	*Next state*	
A	B	x	A	B
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	1

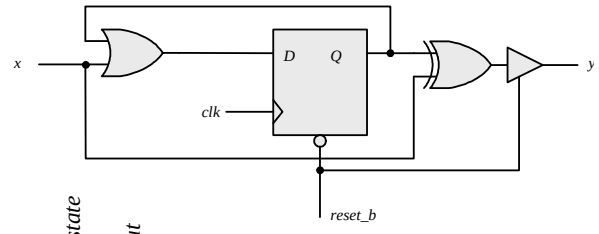


$D_A = A'x + Ax'$

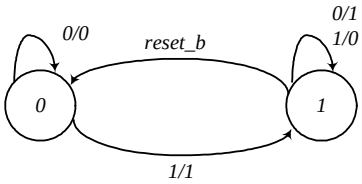


$D_B = AB + Bx'$

5.17 The output is 0 for all 0 inputs until the first 1 occurs, at which time the output is 1. Thereafter, the output is the complement of the input. The state diagram has two states. In state 0: output = input; in state 1: output = input'.



Present state	*Input*	*Next state*	*Output*
A	*x*	*A*	*y*
0	*0*	*0*	*0*
0	*1*	*1*	*1*
1	*0*	*1*	*1*
1	*1*	*1*	*0*

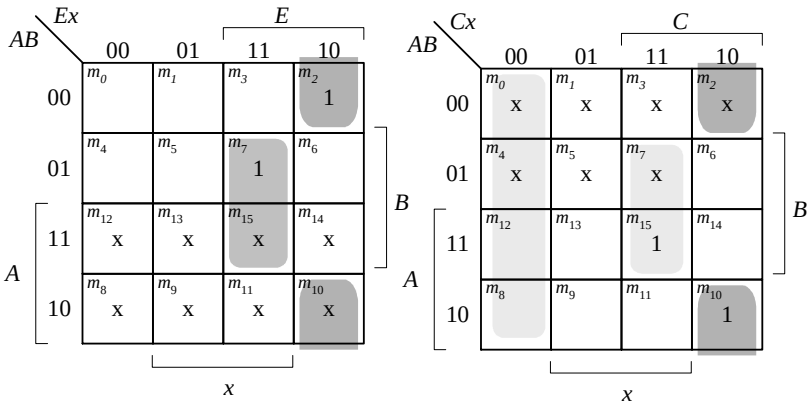


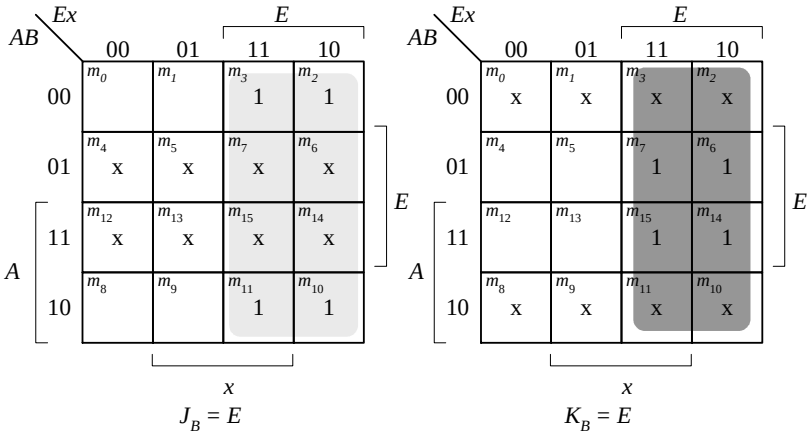
$D_A = A + x$

$y = Ax' + A'x$

5.18 Binary up-down counter with enable *E*.

Present state A B	*Input* x	*Next state* A B	*Flip-flop inputs* J_A K_A	J_B K_B
0 0	0 1	0 0	0 x	0 x
0 0	0 1	0 0	0 x	0 x
0 0	1 0	1 1	1 x	1 x
0 0	1 1	0 1	0 x	1 x
0 1	0 0	0 1	0 x	x 0
0 1	0 1	0 1	0 x	x 0
0 1	1 0	0 1	0 x	x 1
0 1	1 1	1 0	1 x	x 1
1 0	0 0	1 0	x 0	1 0
1 0	0 1	1 0	x 0	1 0
1 0	1 0	0 1	x 1	x 1
1 0	1 1	1 1	x 0	x 1
1 1	0 0	1 1	x 0	x 0
1 1	0 1	1 1	x 0	x 0
1 1	1 0	1 1	1 0	x 1
1 1	1 1	1 1	x 1	x 1

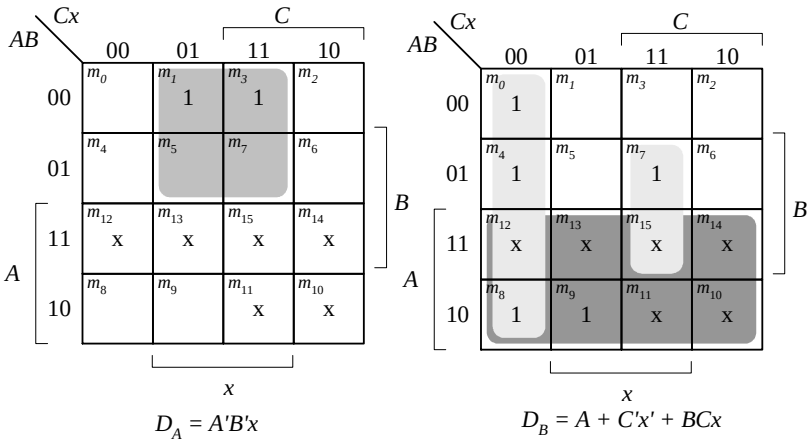


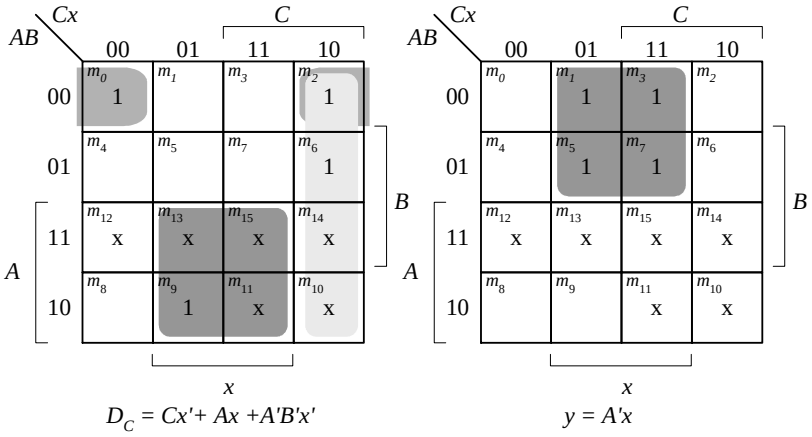


5.19 **(a)** Unused states (see Fig. P5.19): 101, 110, 111.

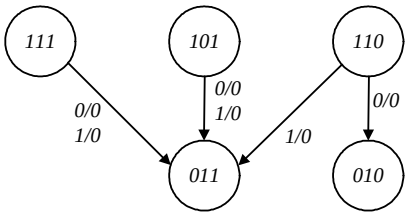
Present state ABC	*Input x*	*Next state ABC*	*Output y*
000	0	011	0
000	1	100	1
001	0	001	0
001	1	100	1
010	0	010	0
010	1	000	1
011	0	001	0
011	1	010	1
100	0	010	0
100	1	011	1

$d(A, B, C, x) = \Sigma\,(10, 11, 12, 13, 14, 15)$





The machine is self-correcting, i.e., the unused states transition to known states.



(b) With JK flip=flops, the state table is the same as in (a).

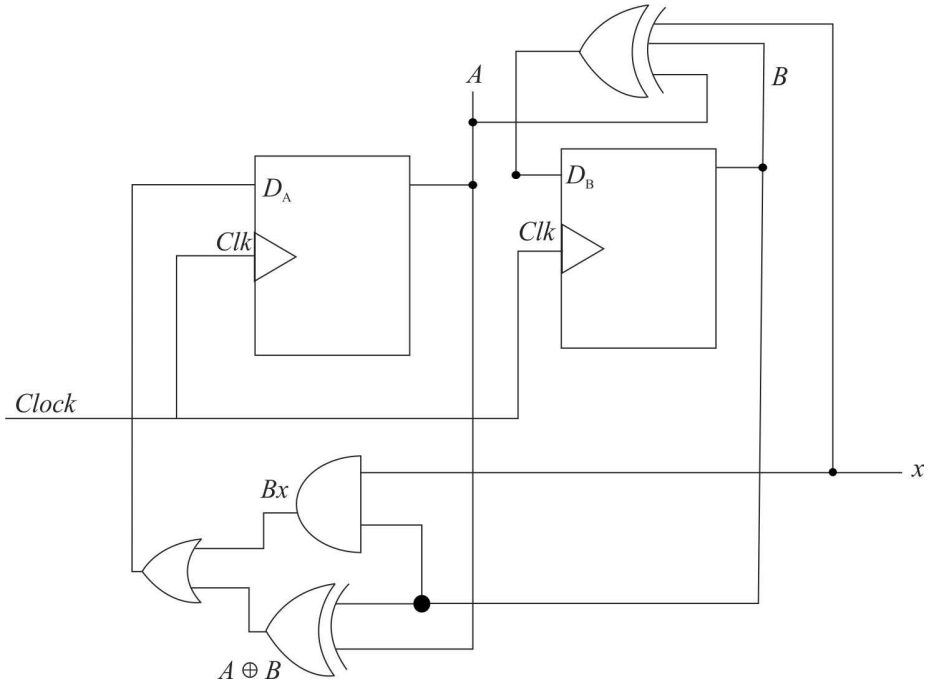
Flip-flop inputs					
J_A	K_A	J_B	K_B	J_C	K_C
0	x	1	x	1	x
1	x	0	x	0	x
0	x	0	x	x	0
1	x	0	x	x	1
0	x	x	0	0	x
0	x	x	1	0	x
0	x	x	1	x	0
0	x	x	0	x	1
x	1	1	x	0	x
x	1	1	x	1	x

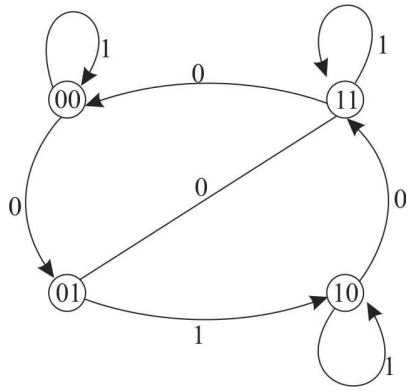
$J_A = B'x$ $K_A = 1$
$J_B = A + C'x'$ $K_B = C' x + Cx'$
$J_C = Ax + A'B'x'$ $K_C = x$
$y = A'x$

The machine is self-correcting because $K_A = 1$.

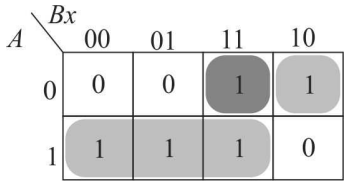
5.20 $D_A = (A \oplus B) + Bx$

$D_B = (x \oplus A \oplus B)$





Present State		Input	Next State			
A(t)	B(t)	x	A(t + 1)	B(t + 1)	D_A	D_B
0	0	0	0	1	0	1
0	0	1	0	0	0	0
0	1	0	1	1	1	1
0	1	1	1	0	1	0
1	0	0	1	1	1	1
1	0	1	1	0	1	0
1	1	0	0	0	0	0
1	1	1	1	1	1	1



$D_A = \Sigma(2, 3, 4, 5, 7)$
$= AB' + Bx + A'B$

A \ Bx	00	01	11	10
0	0	0	0	1
1	1	0	1	0

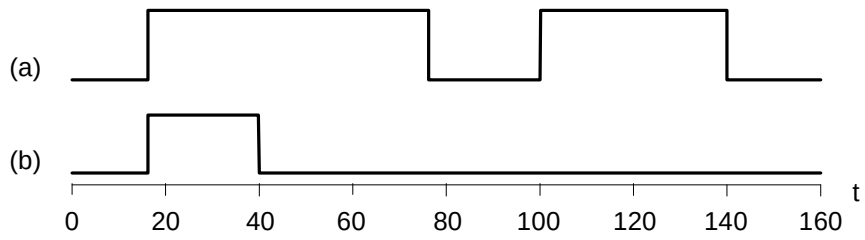
$= (A \oplus B) + Bx$

$D_B = \Sigma(0, 2, 4, 7)$
$= B'x' + A'x' + ABx$

$= x \oplus A \oplus B$

5.21 The statements associated with an **initial** keyword execute once, in sequence, with the activity expiring after the last statment competes execution; the statements assocated with the **always** keyword execute repeatedly, subject to timing control (e.g, #10).

5.22

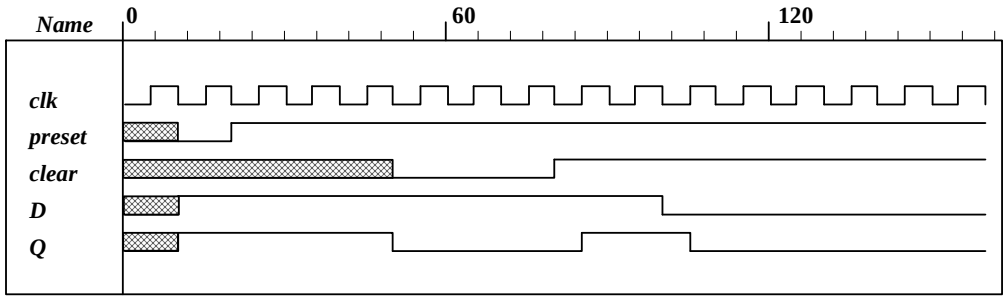


5.23 **(a)** *RegA* = 125, *RegB* = 125
(b) *RegA* = 125, *RegB* = 50 Note: Text has error, with RegB = 30 at page 526).

5.24 **(a)**

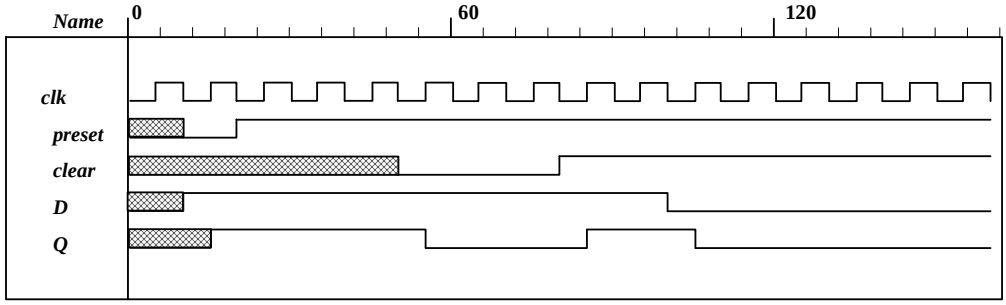
```
module DFF (output reg Q, input D, clk, preset, clear);
   always @ (posedge clk, negedge preset, negedge clear )
      if (preset   == 0) Q <= 1'b1;
      else if (clear == 0) Q <= 1'b0;
      else Q <= D;
endmodule

module t_DFF ();
   wire     Q;
   reg  clk, preset, clear;
   reg  D;
```

```
DFF M0 (Q, D, clk, preset, clear);

initial #160 $finish;
initial begin clk = 0; forever #5 clk = ~clk; end
initial fork
   #10 preset = 0;
   #20 preset = 1;
   #50 clear = 0;
   #80 clear = 1;
   #10 D = 1;
   #100 D = 0;
   #200 D = 1;
join
endmodule
```



(b)

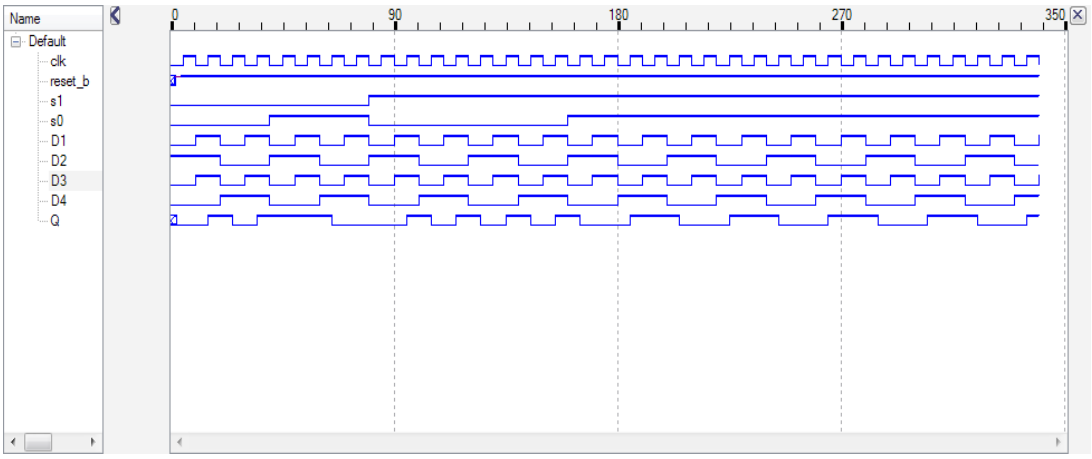
```
module DFF (output reg Q, input D, clk, preset, clear);
always @ (posedge clk)
   if (preset   == 0) Q <= 1'b1;
   else if (clear == 0) Q <= 1'b0;
   else Q <= D;
endmodule
```



5.25

```
module Quad_Input_DFF (output reg Q, input D1, D2, D3, D4, s1, s0, clk, reset_b);
  always @ (posedge clk, negedge reset_b)
    if (reset_b == 1'b0) Q <= 0;
    else case ({s1, s0})
      2'b00: Q <= D1;
      2'b01: Q <= D2;
      2'b10: Q <= D3;
      2'b11: Q <= D4;
    endcase
endmodule

module t_Quad_Input_DFF ();
  wire Q;
```

```
    reg D1, D2, D3, D4, s1, s0, clk, reset_b;
     Quad_Input_DFF M0 (Q, D1, D2, D3, D4, s1, s0, clk, reset_b);
    initial #350 $finish;
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial fork
       begin s1 = 0; s0 = 0; end
       #40 begin s1 = 0; s0 = 1; end
       #80 begin s1 =1; s0 = 0; end
       #120 begin s1 = 1; s0 = 0; end
       #160 begin s1 = 1; s0 = 1; end
    join
    initial fork
       begin D1 = 0; forever #10 D1 = ~D1; end
       begin D2 = 1; forever #20 D2 = ~D2; end
       begin D3 = 0; forever #10 D3 = ~D3; end
       begin D4 = 0; forever #20 D4 = ~D4; end
  join
    initial fork
       #2 reset_b = 1;
       #3 reset_b = 0;
       #4 reset_b = 1;
    join
  endmodule
```



5.26 **(a)**

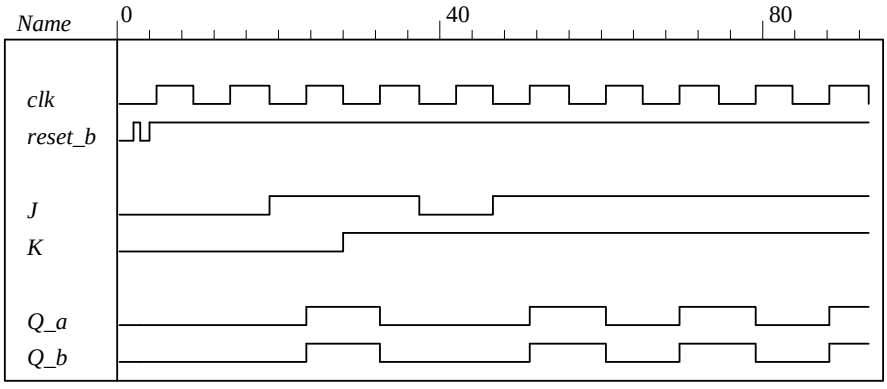
$Q(t + 1) = JQ' + K'Q$

When $Q = 0$, $Q(t + 1) = J$
When $Q = 1$, $Q(t + 1) = K'$

```
module JK_Behavior_a (output reg Q, input J, K, CLK, reset_b);
  always @ (posedge CLK, negedge reset_b)
     if (reset_b == 0) Q <= 0; else
       if (Q == 0)       Q <= J;
       else          Q <= ~K;
endmodule
```

(b)

```
module JK_Behavior_b (output reg Q, input J, K, CLK, reset_b);
   always @ (posedge CLK, negedge reset_b)
       if (reset_b == 0) Q <= 0;
       else
      case ({J, K})
         2'b00:  Q <= Q;
         2'b01:  Q <= 0;
         2'b10:  Q <= 1;
         2'b11:  Q <= ~Q;
      endcase
endmodule

module t_Prob_5_26    ();
   wire Q_a, Q_b;
   reg J, K, clk, reset_b;
    JK_Behavior_a M0 (Q_a, J, K, clk, reset_b);
    JK_Behavior_b M1 (Q_b, J, K, clk, reset_b);

     initial #100 $finish;
     initial begin clk = 0; forever #5 clk = ~clk; end
     initial fork
        #2 reset_b = 1;
        #3 reset_b = 0;       // Initialize to s0
        #4 reset_b = 1;
      J =0; K = 0;
      #20 begin J= 1; K = 0;    end
      #30 begin J = 1; K = 1; end
      #40 begin J = 0; K = 1; end
      #50 begin J = 1; K = 1; end
   join
 endmodule
```

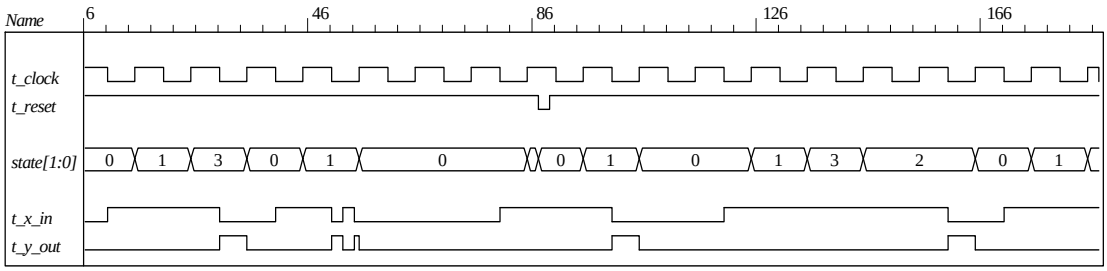


5.27

```
// Mealy FSM zero detector (See Fig. 5.16)
module Mealy_Zero_Detector (
   output reg y_out,
   input    x_in, clock, reset
);
   reg [1: 0]    state, next_state;
   parameter  S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
```

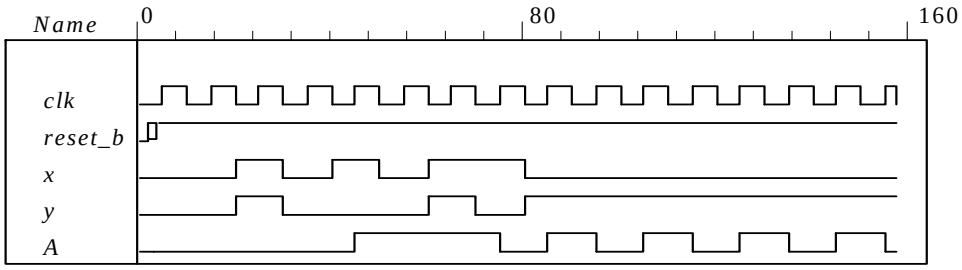
```
   always @ (posedge clock, negedge reset) // state transition
      if (reset == 0) state <= S0;
      else state <= next_state;

   always @ (state, x_in) // Form the next state
      case (state)
         S0: begin y_out = 0;     if (x_in)     next_state = S1; else next_state = S0; end
         S1:      begin y_out = ~x_in; if (x_in)         next_state = S3; else next_state = S0; end
         S2: begin y_out = ~x_in; if (~x_in) next_state = S0; else next_state = S2; end
         S3:      begin y_out = ~x_in; if (x_in)     next_state = S2; else next_state = S0; end
      endcase

endmodule

module t_Mealy_Zero_Detector;
   wire   t_y_out;
   reg  t_x_in, t_clock, t_reset;

Mealy_Zero_Detector M0 (t_y_out, t_x_in, t_clock, t_reset);


initial #200 $finish;
initial begin t_clock = 0; forever #5 t_clock = ~t_clock; end

initial fork
            t_reset = 0;
   #2 t_reset = 1;
   #87 t_reset = 0;
   #89 t_reset = 1;
   #10 t_x_in = 1;
   #30 t_x_in = 0;
   #40 t_x_in = 1;
   #50 t_x_in = 0;
   #52 t_x_in = 1;
   #54 t_x_in = 0;
   #70 t_x_in = 1;
   #80 t_x_in = 1;
   #70 t_x_in = 0;
   #90 t_x_in = 1;
   #100 t_x_in = 0;
   #120 t_x_in = 1;
   #160 t_x_in = 0;
   #170 t_x_in = 1;

   join
endmodule
```

Note: Simulation results match Fig. 5.22.



5.28 (a)

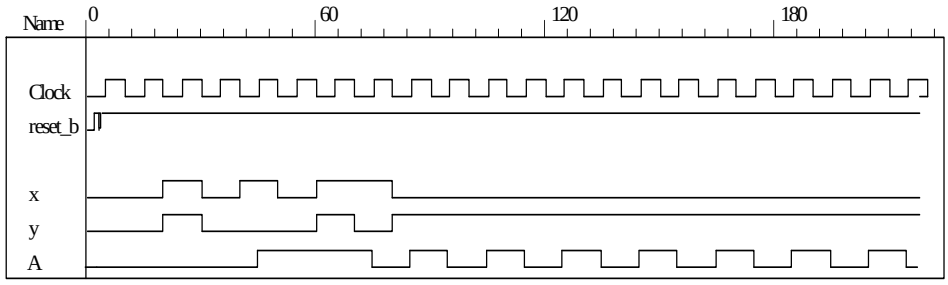
```
module Prob_5_28a   (output A, input x, y, clk, reset_b);
  parameter s0 = 0, s1 = 1;
  reg state, next_state;
  assign A = state;

  always @ (posedge clk, negedge reset_b)
    if (reset_b == 0) state <= s0; else state <= next_state;

  always @ (state, x, y) begin
    next_state = s0;
    case (state)
      s0:     case ({x, y})
                2'b00, 2'b11:   next_state = s0;
                2'b01, 2'b10:   next_state = s1;
              endcase
      s1:     case ({x, y})
                2'b00, 2'b11:   next_state = s1;
                2'b01, 2'b10:   next_state = s0;
              endcase
    endcase
  end
endmodule
module t_Prob_5_28a   ();
  wire A;
  reg x, y, clk, reset_b;
   Prob_5_28a M0 (A, x, y, clk, reset_b);
    initial #350 $finish;
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial fork
      #2 reset_b = 1;
      #3 reset_b = 0;       // Initialize to s0
      #4 reset_b = 1;
     x =0; y = 0;
     #20 begin x= 1; y = 1;   end
     #30 begin x = 0; y = 0; end
     #40 begin x = 1; y = 0; end
     #50 begin x = 0; y = 0; end
     #60 begin x = 1; y = 1; end
     #70 begin x = 1; y = 0; end
     #80 begin x = 0; y = 1; end
  join
endmodule
```



(b)

```
module Prob_5_28b   (output A, input x, y, Clock, reset_b);
  xor (w1, x, y);
  xor (w2, w1, A);
  DFF M0 (A, w2, Clock, reset_b);
```

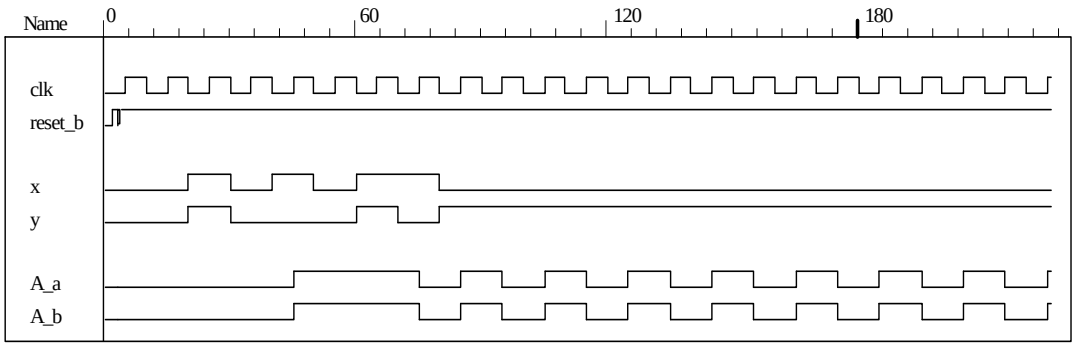
```
 endmodule

module DFF (output reg Q, input D, Clock, reset_b);
   always @ (posedge Clock, negedge reset_b)
      if (reset_b == 0) Q <= 0;
      else Q <= D;
endmodule

module t_Prob_5_28b    ();
   wire A;
   reg x, y, clk, reset_b;
    Prob_5_28b M0 (A, x, y, clk, reset_b);
     initial #350 $finish;
     initial begin clk = 0; forever #5 clk = ~clk; end
     initial fork
        #2 reset_b = 1;
        #3 reset_b = 0;       // Initialize to s0
        #4 reset_b = 1;
      x =0; y = 0;
      #20 begin x= 1; y = 1;    end
      #30 begin x = 0; y = 0; end
      #40 begin x = 1; y = 0; end
      #50 begin x = 0; y = 0; end
      #60 begin x = 1; y = 1; end
      #70 begin x = 1; y = 0; end
      #80 begin x = 0; y = 1; end
   join
 endmodule
```



(c) See results of (b) and (c).

```
module t_Prob_5_28c    ();
   wire A_a, A_b;
   reg x, y, clk, reset_b;
    Prob_5_28a M0 (A_a, x, y, clk, reset_b);
    Prob_5_28b M1 (A_b, x, y, clk, reset_b);

     initial #350 $finish;
     initial begin clk = 0; forever #5 clk = ~clk; end
     initial fork
        #2 reset_b = 1;
        #3 reset_b = 0;       // Initialize to s0
        #4 reset_b = 1;
      x =0; y = 0;
      #20 begin x= 1; y = 1;    end
      #30 begin x = 0; y = 0; end
      #40 begin x = 1; y = 0; end
```

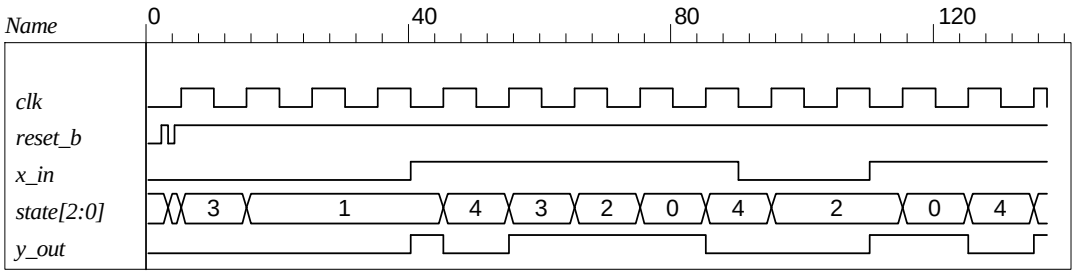
```
        #50 begin x = 0; y = 0; end
        #60 begin x = 1; y = 1; end
        #70 begin x = 1; y = 0; end
        #80 begin x = 0; y = 1; end
      join
    endmodule
```



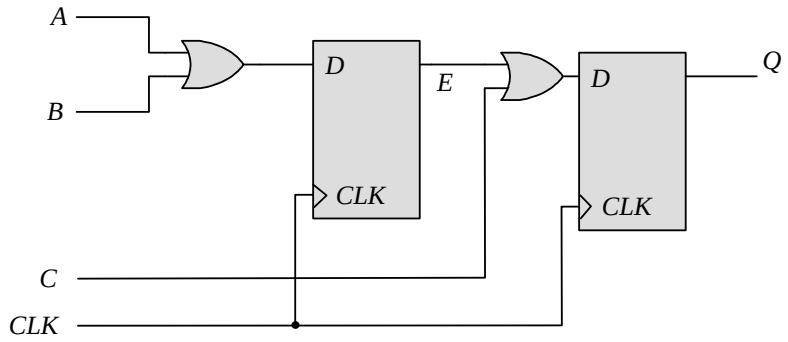
5.29

```
module Prob_5_29 (output reg y_out, input x_in, clock, reset_b);
  parameter s0 = 3'b000, s1 = 3'b001, s2 = 3'b010, s3 = 3'b011, s4 = 3'b100;
  reg [2: 0] state, next_state;

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= s0;
    else state <= next_state;

  always @ (state, x_in) begin
    y_out = 0;
    next_state = s0;
    case (state)
      s0:    if (x_in) begin next_state = s4; y_out = 1; end else begin next_state = s3; y_out = 0;
end
      s1:    if (x_in) begin next_state = s4; y_out = 1; end else begin next_state = s1; y_out = 0;
end
      s2:    if (x_in) begin next_state = s0; y_out = 1; end else begin next_state = s2; y_out = 0;
end
      s3:    if (x_in) begin next_state = s2; y_out = 1; end else begin next_state = s1; y_out = 0;
end
      s4:    if (x_in) begin next_state = s3; y_out = 0; end else begin next_state = s2; y_out = 0;
end
      default:   next_state = 3'bxxx;
    endcase
  end
endmodule

module t_Prob_5_29 ();
  wire y_out;
  reg x_in, clk, reset_b;

  Prob_5_29 M0 (y_out, x_in, clk, reset_b);

  initial #350$finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork
    #2 reset_b = 1;
    #3 reset_b = 0;       // Initialize to s0
    #4 reset_b = 1;
                    // Trace the state diagram and monitor y_out
```

```
    x_in = 0;          // Drive from s0 to s3 to    S1 and park
   #40 x_in = 1;       // Drive to s4 to s3 to s2 to s0 to s4 and loop
   #90 x_in = 0;       // Drive from s0 to s3 to s2 and part
   #110   x_in = 1;       // Drive s0 to s4    etc
  join
endmodule
```



5.30



5.31

```
module Seq_Ckt (input A, B, C, CLK, output reg Q);
  reg E;
  always @ (posedge CLK)
  begin
    Q = E && C;
    E = A || B;
  end
endmodule
```

Note: The statements must be written in an order than produces the effect of concurrent assignments.

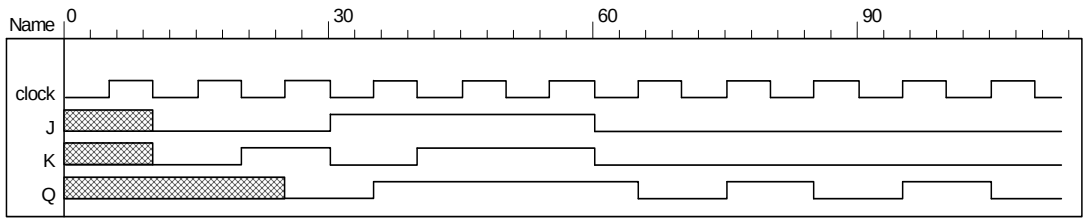
5.32

```
initial begin
  enable = 0; A = 1; B = 0; C = 0; D = 1; E = 1; F = 1;
  #10  A = 0; B = 1; C = 1;
  #10  A = 1; B = 0; D = 1;    E = 0;
  #10  B = 1; E = 1; F = 0;
  #10  enable = 1;
               B = 0; D= 0; F =1;
  #10  B = 1;
  #10  B = 0; D = 1;
  #10  B = 1;
end

initial fork
  enable = 0; A = 1; B = 0; C = 0; D = 1; E = 1; F = 1;
#10 begin A = 0; B = 1; end
#20 begin   A = 1; B = 0; D = 1; E = 0; end
#30 begin   B = 1; E = 1; F = 0; end
#40 begin   B = 0; D = 0; F = 1; end
#50 begin   B = 1; end
#60 begin B = 0; D = 1; end
#70 begin   B = 1; end
join
```

5.33 Signal transitions that are caused by input signals that change on the active edge of the clock race with the clock itself to reach the affected flip-flops, and the outcome is indeterminate (unpredictable). Conversely, changes caused by inputs that are synchronized to the inactive edge of the clock reach stability before the active edge, with predictable outputs of the flip-flops that are affected by the inputs.

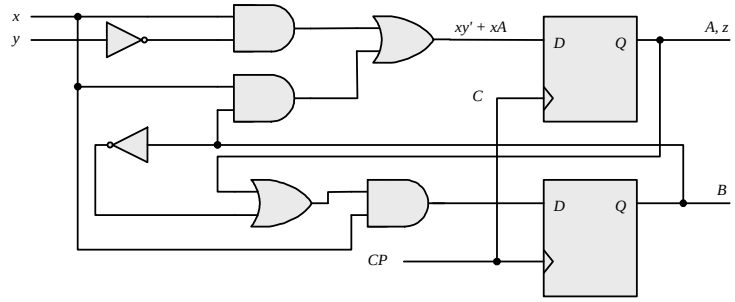
5.34 Note: Problem statement should refer to Problem 5.2 instead of Fig 5.5.

```
module JK_flop_Prob_5_34 (output Q, input J, K, clk);
   wire K_bar;
   D_flop M0 (Q, D, clk);
   Mux M1 (D, J, K_bar, Q);
   Inverter M2 (K_bar, K);
endmodule

module D_flop (output reg Q, input D, clk);
   always @ (posedge clk) Q <= D;
endmodule

module Inverter (output y_bar, input y);
   assign y_bar = ~y;
endmodule

module Mux (output y, input a, b, select);
   assign y = select ? a: b;
endmodule

module t_JK_flop_Prob_5_34 ();
   wire     Q;
   reg  J, K, clock;
   JK_flop_Prob_5_34 M0 (Q, J, K, clock);
   initial #500 $finish;
   initial begin clock = 0; forever #5 clock = ~clock; end
   initial fork
   #10 begin J = 0; K = 0; end        // toggle Q unknown
   #20 begin J = 0; K = 1; end        // set Q to 0
   #30 begin J = 1; K = 0; end        // set q to 1
   #40 begin J = 1; K = 1; end        // no change
   #60 begin J = 0; K = 0; end        // toggle Q
   join
endmodule
```



5.35

From Problem 5.6:

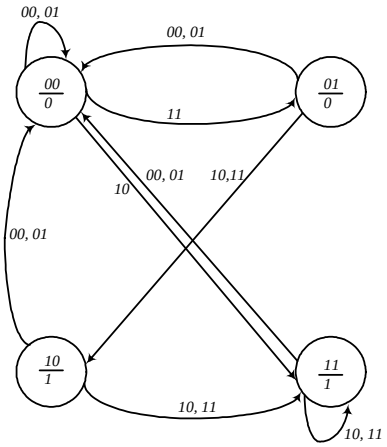


(b)

$A(t+1) = xy' + xB$
$B(t+1) = xA + xB'$
$z = A$

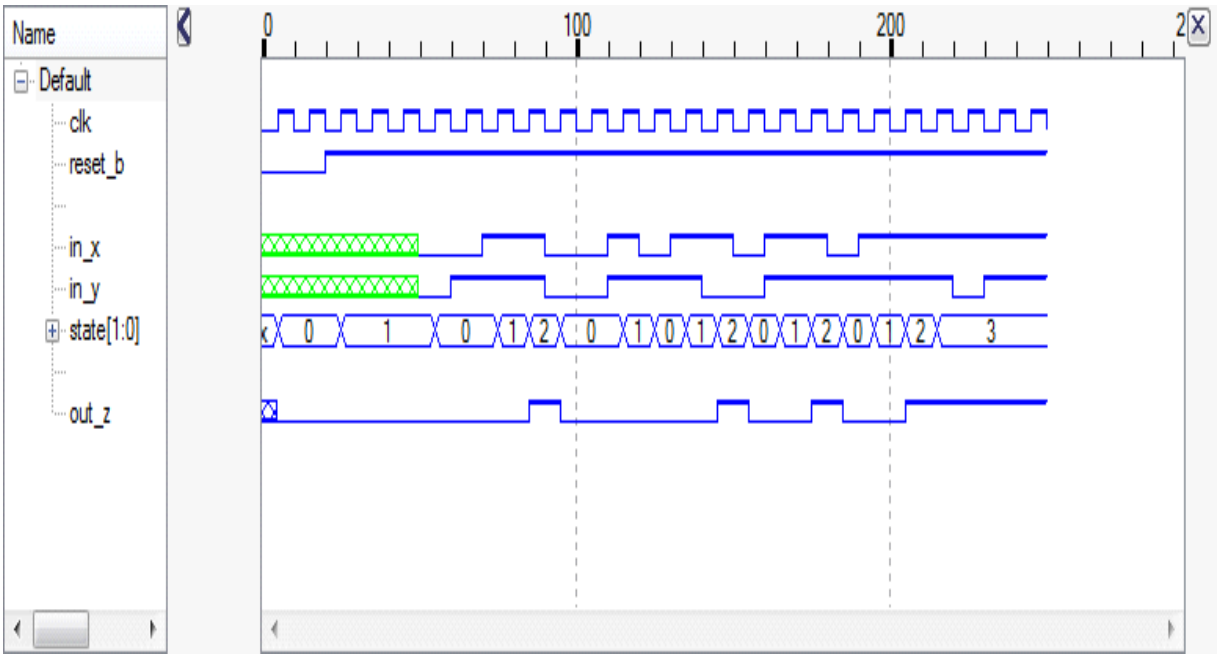
Present state		Inputs		Next state		Output
A	B	x	y	A	B	z
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

(c)



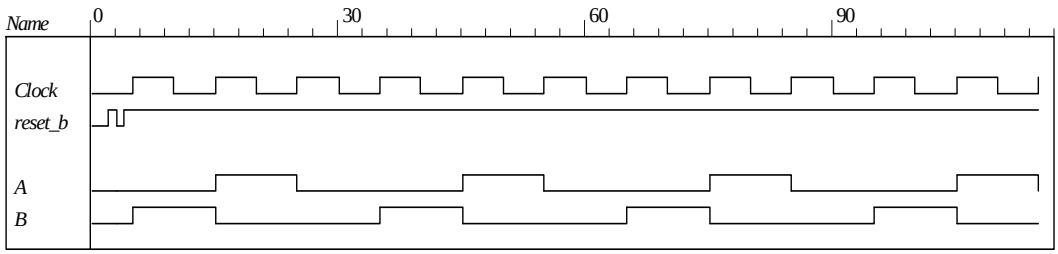

```
module Prob_5_35 (output out_z, input in_x, in, in_y, clk, reset_b);
   reg [1:0] state, next_state;
   assign out_z = ((state == 2'b10) || (state == 2'b11));

   always @ (posedge clk) if (reset_b == 1'b0) state <= 2'b00; else state <= next_state;

   always @ (state, in_x, in_y)
      case (state)
         2'b00:  if (({in_x, in_y} == 2'b00) || ({in_x, in_y} == 2'b01)) next_state = 2'b00;
               else if ({in_x, in_y} == 2'b10) next_state = 2'b11;
               else next_state = 2'b01;

         2'b01:  if (({in_x, in_y} == 2'b00) || ({in_x, in_y} == 2'b01)) next_state = 2'b00;
               else if (({in_x, in_y} == 2'b10) || ({in_x, in_y} == 2'b11)) next_state = 2'b10;

         2'b10:  if (({in_x, in_y} == 2'b00) || ({in_x, in_y} == 2'b01)) next_state = 2'b00;
               else if (({in_x, in_y} == 2'b10) || ({in_x, in_y} == 2'b11)) next_state = 2'b11;

         2'b11:  if (({in_x, in_y} == 2'b00) || ({in_x, in_y} == 2'b01)) next_state = 2'b00;
               else if (({in_x, in_y} == 2'b10) || ({in_x, in_y} == 2'b11)) next_state = 2'b11;
      endcase
endmodule
```

```
module t_Prob_5_35 ();
   wire out_z;
   reg in_x, in, in_y, clk, reset_b;

    Prob_5_35 M0 (out_z, in_x, in, in_y, clk, reset_b);

   initial #250 $finish;
   initial begin clk = 0; forever #5 clk = ~clk; end
   initial fork
      reset_b = 0;
      #20 reset_b = 1;

      #50 {in_x, in_y} = 2'b00;          // Remain in 2'b00
      #60 {in_x, in_y} = 2'b01;          // Remain in 2'b00
      #70 {in_x, in_y} = 2'b11;          // Transition to 2'b01
      #90 {in_x, in_y} = 2'b00;          // Transition to 2'b00
      #110 {in_x, in_y} = 2'b11;             // Transition to 2'b01
      #120 {in_x, in_y} = 2'b01;             // Transition to 2'b00

      #130 {in_x, in_y} = 2'b11;             // Transition to 2'b01
      #140 {in_x, in_y} = 2'b10;             // Transition to 2'b10
      #150 {in_x, in_y} = 2'b00;             // Transition to 2'b00
      #160 {in_x, in_y} = 2'b11;             // Transition to 2'b01

      #170 {in_x, in_y} = 2'b11;             // Transition to 2'b10
      #180 {in_x, in_y} = 2'b01;             // Transition to 2'b00

      #190 {in_x, in_y} = 2'b11;             // Transition to 2'b01
      #200 {in_x, in_y} = 2'b11;             // Transition to 2'b10
      #210 {in_x, in_y} = 2'b11;             // Transition to 2'b11

      #220 {in_x, in_y} = 2'b10;             // Remain in 2'b11
      #230 {in_x, in_y} = 2'b11;             // Remain in 2'b11
   join
endmodule
```



5.36 Note: See Problem 5.8 (counter with repeated sequence: (A, B) = 00, 01, 10, 00 ....

```
// See Fig. P5.8
module Problem_5_36 (output A, B, input Clock, reset_b);
   or (T_A, A, B);
   or (T_B, A_b, B);
   T_flop M0 (A, A_b, T_A, Clock, reset_b);
   T_flop M1 (B, B_b, T_B, Clock, reset_b);
endmodule


module T_flop (output reg Q, output QB, input T, Clock, reset_b);
   assign QB = ~ Q;
   always @ (posedge Clock, negedge reset_b)
      if (reset_b == 0) Q <= 0;
      else if (T) Q <= ~Q;
endmodule

module t_Problem_5_36 ();
   wire A, B;
   reg   Clock, reset_b;

   Problem_5_36 M0 (A, B, Clock, reset_b);

   initial #350$finish;
   initial begin Clock = 0; forever #5 Clock = ~Clock; end
   initial fork
      #2 reset_b = 1;
      #3 reset_b = 0;
      #4 reset_b = 1;
   join
endmodule
```

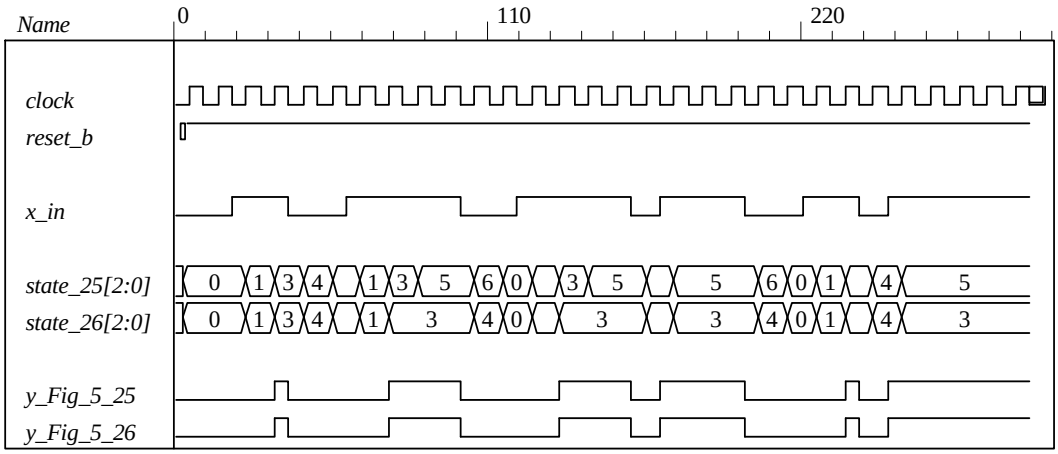


5.37

```
module Problem_5_37_Fig_5_25 (output reg y, input x_in, clock, reset_b);

   parameter a = 3'b000, b = 3'b001, c = 3'b010, d = 3'b011, e = 3'b100, f = 3'b101, g = 3'b110;
   reg [2: 0] state, next_state;

   always @ (posedge clock, negedge reset_b)
      if (reset_b == 0) state <= a;
      else state <= next_state;

   always @ (state, x_in) begin
      y = 0;
      next_state = a;
```

```
    case (state)
      a:      begin y = 0; if (x_in == 0) next_state = a; else next_state = b; end

      b:      begin y = 0; if (x_in == 0) next_state = c; else next_state = d; end

      c:      begin y = 0; if (x_in == 0) next_state = a; else next_state = d; end

      d:      if (x_in == 0) begin y = 0; next_state = e; end
          else begin y = 1; next_state = f; end

      e:      if (x_in == 0) begin y = 0;    next_state = a; end
          else begin y = 1; next_state = f; end

      f:      if (x_in == 0) begin y = 0;    next_state = g; end
          else begin y = 1; next_state = f; end

      g:      if (x_in == 0) begin y = 0; next_state = a; end
          else begin y = 1; next_state = f; end

      default:    next_state = a;
    endcase
  end
endmodule
module Problem_5_37_Fig_5_26 (output reg y, input x_in, clock, reset_b);
  parameter a = 3'b000, b = 3'b001, c = 3'b010, d = 3'b011, e = 3'b100;
  reg [2: 0] state, next_state;

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= a;
    else state <= next_state;
```

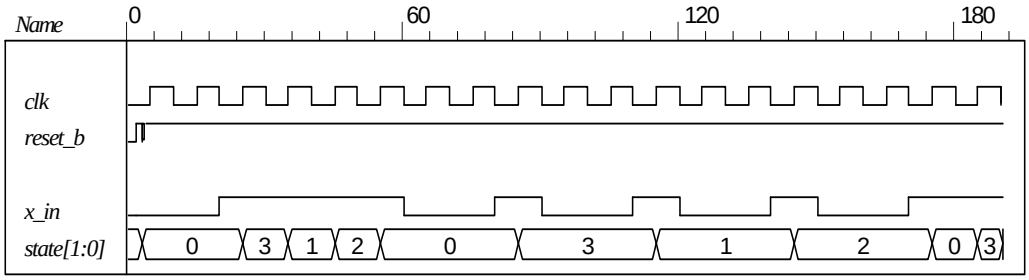
```
  always @ (state, x_in) begin
    y = 0;
    next_state = a;
    case (state)
      a:      begin y = 0; if (x_in == 0) next_state = a; else next_state = b; end

      b:      begin y = 0; if (x_in == 0) next_state = c; else next_state = d; end

      c:      begin y = 0; if (x_in == 0) next_state = a; else next_state = d; end

      d:      if (x_in == 0) begin y = 0; next_state = e; end
          else begin y = 1; next_state = d; end

      e:      if (x_in == 0) begin y = 0;    next_state = a; end
          else begin y = 1; next_state = d; end

      default:    next_state = a;
    endcase
  end
endmodule

module t_Problem_5_37 ();
  wire y_Fig_5_25, y_Fig_5_26;
  reg x_in, clock, reset_b;

  Problem_5_37_Fig_5_25 M0 (y_Fig_5_25, x_in, clock, reset_b);
  Problem_5_37_Fig_5_26 M1 (y_Fig_5_26, x_in, clock, reset_b);

  wire [2: 0] state_25 = M0.state;
  wire [2: 0] state_26 = M1.state;

  initial #350 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    x_in = 0;
    #2 reset_b = 1;
    #3 reset_b = 0;
    #4 reset_b = 1;
    #20 x_in = 1;
    #40 x_in = 0; // abdea, abdea

    #60 x_in = 1;
    #100 x_in = 0;    // abdf....fga, abd ... dea

    #120 x_in = 1;
    #160 x_in = 0;
    #170 x_in = 1;
    #200 x_in = 0; // abdf....fgf...fga, abd ...ded...ea

    #220 x_in = 1;
    #240 x_in = 0;
    #250 x_in = 1;    // abdef...  // abded...
  join
endmodule
```



5.38 (a)

```
module Prob_5_38a (input x_in, clock, reset_b);
  parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;
  reg [1: 0] state, next_state;

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= s0;
    else state <= next_state;

  always @ (state, x_in) begin
      next_state = s0;
    case (state)
      s0:     if (x_in == 0) next_state = s0;
          else if (x_in == 1) next_state = s3;

      s1:     if (x_in == 0) next_state = s1;
          else if (x_in == 1) next_state = s2;

      s2:     if (x_in == 0) next_state = s2;
          else if (x_in == 1) next_state = s0;

      s3:     if (x_in == 0) next_state = s3;
          else if (x_in == 1) next_state = s1;
      default:        next_state = s0;
    endcase
  end
endmodule

module t_Prob_5_38a ();
  reg x_in, clk, reset_b;

  Prob_5_38a M0 ( x_in, clk, reset_b);

  initial #350$finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork
    #2 reset_b = 1;
    #3 reset_b = 0;       // Initialize to s0
    #4 reset_b = 1;
    #2 x_in = 0;
    #20 x_in = 1;
    #60 x_in = 0;
    #80 x_in = 1;
    #90 x_in = 0;
```

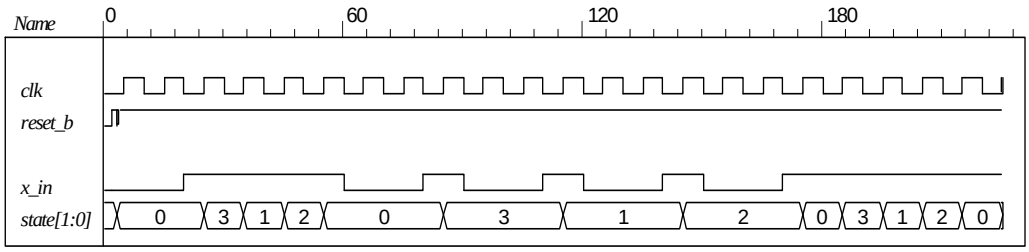
```
    #110 x_in = 1;
    #120 x_in = 0;
    #140 x_in = 1;
    #150 x_in = 0;
    #170 x_in= 1;
  join
endmodule
```



(b)

```
module Prob_5_38b (input x_in, clock, reset_b);
  parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;
  reg [1: 0] state, next_state;

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= s0;
    else state <= next_state;

  always @ (state, x_in) begin
     next_state = s0;
    case (state)
      s0:     if (x_in == 0) next_state = s0;
              else if (x_in == 1) next_state = s3;

      s1:     if (x_in == 0) next_state = s1;
              else if (x_in == 1) next_state = s2;

      s2:     if (x_in == 0) next_state = s2;
              else if (x_in == 1) next_state = s0;

      s3:     if (x_in == 0) next_state = s3;
              else if (x_in == 1) next_state = s1;
      default:        next_state = s0;
    endcase
  end
endmodule

module t_Prob_5_38b ();

  reg x_in, clk, reset_b;

  Prob_5_38b M0 ( x_in, clk, reset_b);

  initial #350$finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork
    #2 reset_b = 1;
    #3 reset_b = 0;       // Initialize to s0
    #4 reset_b = 1;
    #2 x_in = 0;
    #20 x_in = 1;
```

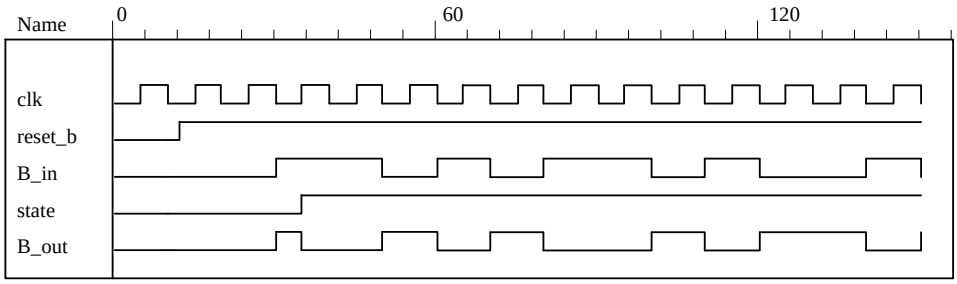
```
    #60 x_in = 0;
    #80 x_in = 1;
    #90 x_in = 0;
    #110 x_in = 1;
    #120 x_in = 0;
    #140 x_in = 1;
    #150 x_in = 0;
    #170 x_in= 1;
  join
endmodule
```



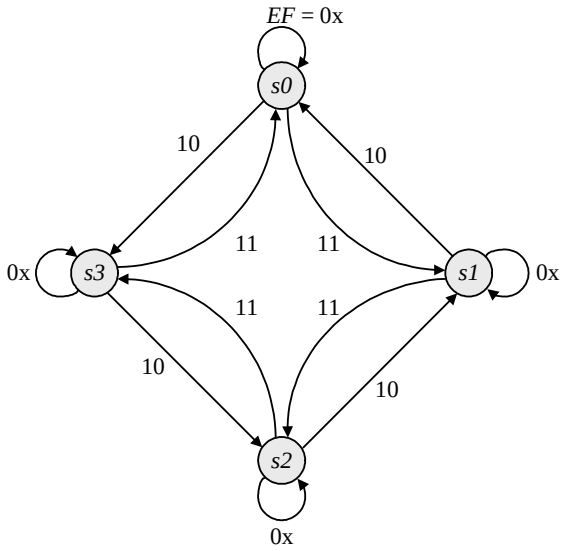
5.39

```
module Serial_2s_Comp (output reg B_out, input B_in, clk, reset_b);
// See problem 5.17
  parameter S_0 = 1'b0, S_1 = 1'b1;
  reg state, next_state;
  always @ (posedge clk, negedge reset_b) begin
    if (reset_b == 0) state <= S_0;
    else state <= next_state;
  end

  always @ (state, B_in) begin
    B_out = 0;
    case (state)
      S_0:    if (B_in == 0) begin next_state = S_0; B_out = 0; end
       else if (B_in == 1) begin next_state = S_1;   B_out = 1; end

      S_1:    begin next_state = S_1; B_out = ~B_in; end
     default:      next_state = S_0;

    endcase
  end
endmodule

module t_Serial_2s_Comp ();
  wire B_in, B_out;
  reg clk, reset_b;
  reg [15: 0] data;
  assign B_in = data[0];

  always @ (negedge clk, negedge reset_b)
    if (reset_b == 0) data <= 16'ha5ac; else data <= data >> 1;    // Sample bit stream

  Serial_2s_Comp M0 (B_out, B_in, clk, reset_b);

  initial #150 $finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork
    #10 reset_b = 0;
```

```
    #12 reset_b = 1;
  join
endmodule
```

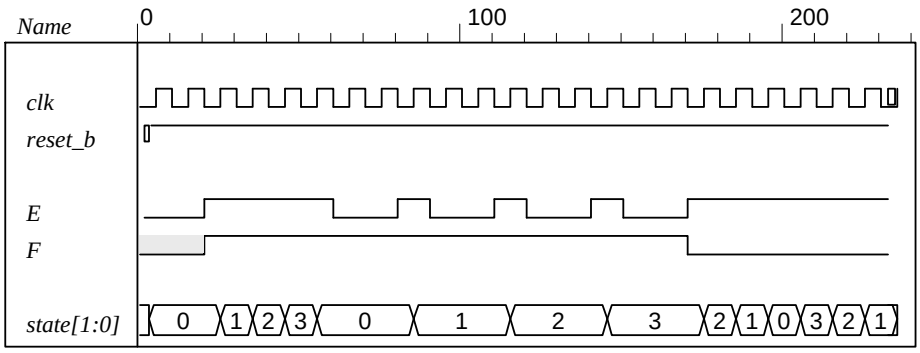


5.40




```
module Prob_5_40 (input E, F, clock, reset_b);
  parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;
  reg [1: 0] state, next_state;

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= s0;
    else state <= next_state;

  always @ (state, E, F) begin
     next_state = s0;
    case (state)
      s0:    if (E == 0) next_state = s0;
          else if (F == 1) next_state = s1; else next_state = s3;

      s1:    if (E == 0) next_state = s1;
          else if (F == 1) next_state = s2; else next_state = s0;

      s2:    if (E == 0) next_state = s2;
          else if (F == 1) next_state = s3; else next_state = s1;

      s3:    if (E == 0) next_state = s3;
          else if (F == 1) next_state = s0; else next_state = s2;
```

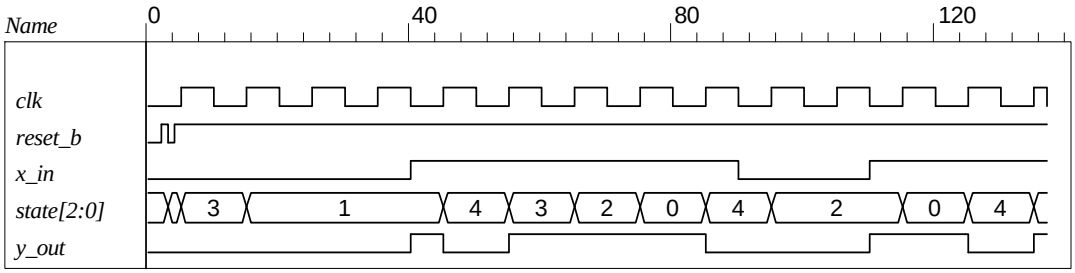
```
        default:    next_state = s0;
      endcase
    end
endmodule

module t_Prob_5_40 ();

  reg E, F, clk, reset_b;

  Prob_5_40 M0 ( E, F, clk, reset_b);

  initial #350$finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork
    #2 reset_b = 1;
    #3 reset_b = 0;        // Initialize to s0
    #4 reset_b = 1;
    #2 E = 0;
    #20 begin E = 1; F = 1; end
    #60 E = 0;
    #80 E = 1;
    #90 E = 0;
    #110 E = 1;
    #120 E = 0;
    #140 E = 1;
    #150 E = 0;
    #170 E= 1;
    #170 F = 0;
   join
endmodule
```



5.41

```
module Prob_5_41 (output reg y_out, input x_in, clock, reset_b);
  parameter s0 = 3'b000, s1 = 3'b001, s2 = 3'b010, s3 = 3'b011, s4 = 3'b100;
  reg [2: 0] state, next_state;

  always @ (posedge clock, negedge reset_b)
    if (reset_b == 0) state <= s0;
    else state <= next_state;

  always @ (state, x_in) begin
    y_out = 0;
    next_state = s0;
    case (state)
      s0:     if (x_in) begin next_state = s4; y_out = 1; end else begin next_state = s3; y_out = 0;
end
```

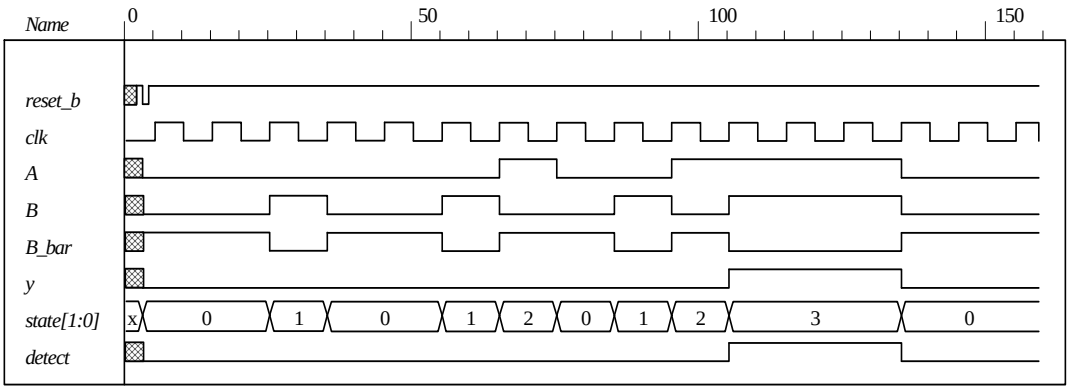
```
        s1:     if (x_in) begin next_state = s4; y_out = 1; end else begin next_state = s1; y_out = 0;
end
        s2:     if (x_in) begin next_state = s0; y_out = 1; end else begin next_state = s2; y_out = 0;
end
        s3:     if (x_in) begin next_state = s2; y_out = 1; end else begin next_state = s1; y_out = 0;
end
        s4:     if (x_in) begin next_state = s3; y_out = 0; end else begin next_state = s2; y_out = 0;
end
        default:    next_state = 3'bxxx;
      endcase
   end
endmodule

module t_Prob_5_41 ();
   wire y_out;
   reg x_in, clk, reset_b;

   Prob_5_41 M0 (y_out, x_in, clk, reset_b);

   initial #350$finish;
   initial begin clk = 0; forever #5 clk = ~clk; end
   initial fork
      #2 reset_b = 1;
      #3 reset_b = 0;        // Initialize to s0
      #4 reset_b = 1;
            // Trace the state diagram and monitor y_out
       x_in = 0;     // Drive from s0 to s3 to   S1 and park
      #40 x_in = 1;      // Drive to s4 to s3 to s2 to s0 to s4 and loop
      #90 x_in = 0;      // Drive from s0 to s3 to s2 and part
      #110   x_in = 1;       // Drive s0 to s4   etc
    join
endmodule
```



5.42

```
module Prob_5_42 (output A, B, B_bar, y, input x, clk, reset_b);
// See Fig. 5.29
   wire w1, w2, w3, D1, D2;
   and (w1, A, x);
   and (w2, B, x);
   or (D_A, w1, w2);

   and (w3, B_bar, x);
   and (y, A, B);
   or (D_B, w1, w3);
   DFF M0_A (A, D_A, clk, reset_b);
   DFF M0_B (B, D_B, clk, reset_b);
   not (B_bar, B);
endmodule

module DFF (output reg Q, input data, clk, reset_b);
```

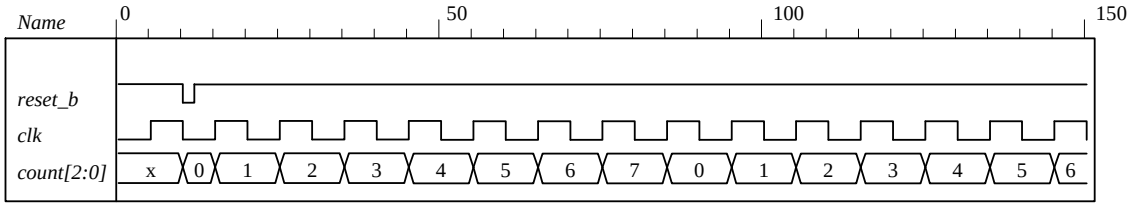
```
   always @ (posedge clk, negedge reset_b)
   if (reset_b == 0) Q <= 0; else Q <= data;
endmodule

module t_Prob_5_42 ();
   wire A, B, B_bar, y;
   reg bit_in, clk, reset_b;
   wire [1:0] state;
   assign state = {A, B};
   wire detect = y;

   Prob_5_42 M0 (A, B, B_bar, y, bit_in, clk, reset_b);

   // Patterns from Problem 5.45.

   initial #350$finish;
   initial begin clk = 0; forever #5 clk = ~clk; end
   initial fork
      #2 reset_b = 1;
      #3 reset_b = 0;
      #4reset_b = 1;
                        // Trace the state diagram and monitor detect (assert in S3)
              bit_in = 0;         // Park in S0
      #20 bit_in = 1;          // Drive to S0
      #30 bit_in = 0;          // Drive to S1 and back to S0 (2 clocks)
      #50 bit_in = 1;
      #70 bit_in = 0;          // Drive to S2 and back to S0 (3 clocks)
      #80 bit_in = 1;
      #130 bit_in = 0;  // Drive to S3, park, then    and back to S0
    join
endmodule
```



5.43

```
module Binary_Counter_3_bit (output [2: 0] count, input clk, reset_b)
   always @ (posedge clk) if (reset_b == 0) count <= 0; else count <= next_count;
   always @ (count) begin
      case (state)
         3'b000: count = 3'b001;
         3'b001: count = 3'b010;
         3'b010: count = 3'b011;
         3'b011: count = 3'b100;
         3'b100: count = 3'b001;
         3'b101: count = 3'b010;
         3'b110: count = 3'b011;
         3'b111: count = 3'b100;
```

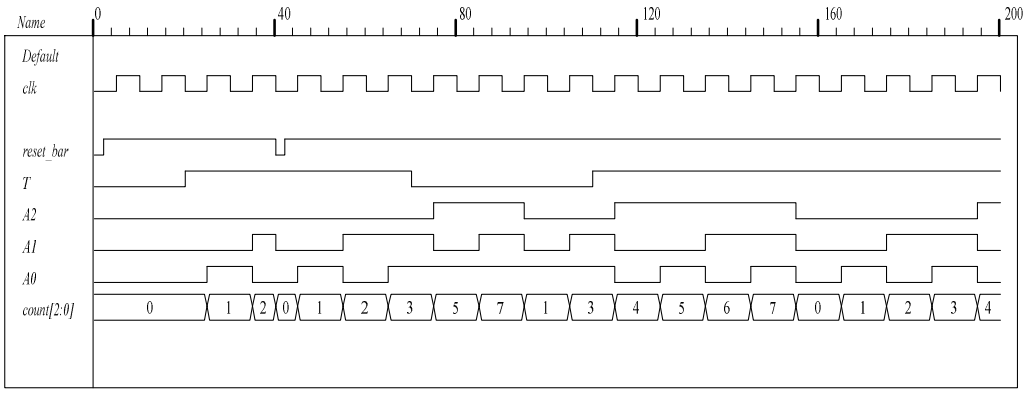
```
        default: count = 3'b000;
      endcase
    end
endmodule

module t_Binary_Counter_3_bit ()
  wire [2: 0] count;
  reg clk, reset_b;
  Binary_Counter_3_bit M0 ( count, clk, reset_b)

  initial #150 $finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork
    reset = 1;
    #10 reset = 0;
    #12 reset = 1;
endmodule
```



Alternative: structural model.

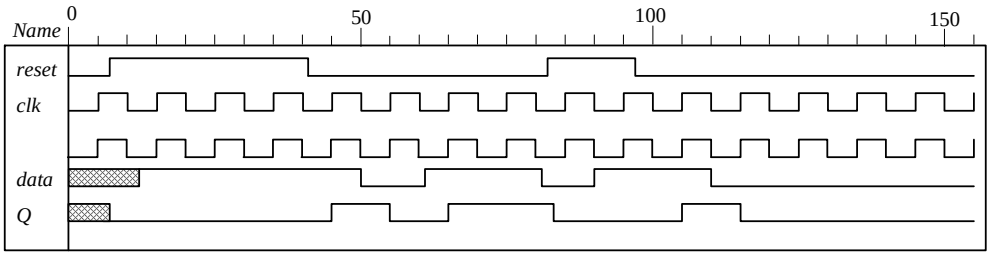
```
module Prob_5_41 (output A2, A1, A0, input T, clk, reset_bar);
  wire  toggle_A2;

  T_flop M0 (A0, T, clk, reset_bar);
  T_flop M1 (A1, A0, clk, reset_bar);
  T_flop M2 (A2, toggle_A2, clk, reset_bar);
  and (toggle_A2, A0, A1);
endmodule

module T_flop (output reg Q, input T, clk, reset_bar);
  always @ (posedge clk, negedge reset_bar)
    if (!reset_bar) Q <= 0; else if (T) Q <= ~Q; else Q <= Q;
endmodule

module t_Prob_5_41;
  wire A2, A1, A0;
  wire [2: 0] count = {A2, A1, A0};
  reg T, clk, reset_bar;
  Prob_5_41 M0 (A2, A1, A0, T, clk, reset_bar);

  initial #200 $finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork reset_bar = 0; #2 reset_bar = 1; #40 reset_bar = 0; #42 reset_bar = 1; join
  initial fork T = 0; #20 T = 1; #70 T = 0; #110 T = 1; join
endmodule
```

If the input to *A*0 is changed to 0 the counter counts incorrectly. It resumes a correct counting sequence when *T* is changed back to 1.



5.44

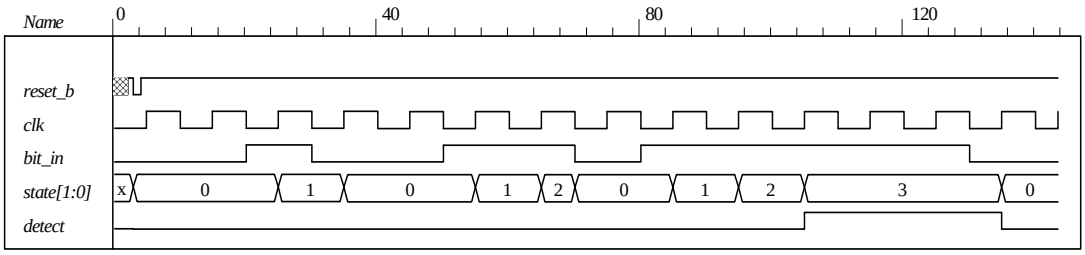
```
module DFF_asynch_reset (output reg Q, input data, clk, reset);
   always @ (posedge clk, posedge reset)          // Asynchronous reset
   if (reset) Q <= 0; else Q <= data;
endmodule

module t_DFF_asynch_reset ();
   reg data, clk, reset;
   wire Q;

   DFF_asynch_reset M0 (Q, data, clk, reset);

   initial #150 $finish;
   initial begin clk = 0; forever #5 clk = ~clk; end
   initial fork
      reset = 0;
      #7 reset = 1;
      #41 reset = 0;
      #82 reset = 1;
      #97 reset = 0;

      #12 data = 1;
      #50 data = 0;
      #60 data = 1;
      #80 data = 0;
      #90 data = 1;
      #110 data = 0;
   join
endmodule
```



5.45

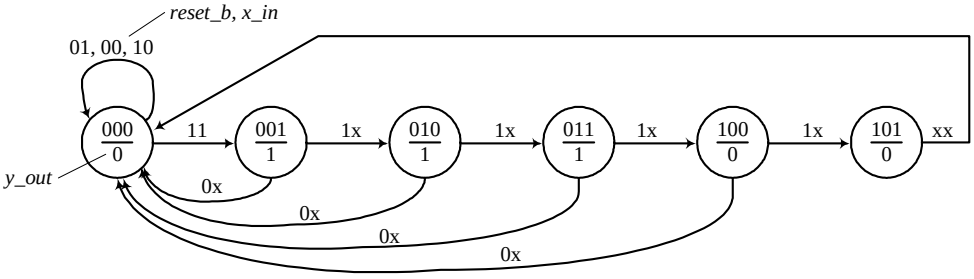
```
module Seq_Detector_Prob_5_45 (output detect, input bit_in, clk, reset_b);
parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3;
```

```
reg [1: 0] state, next_state;

assign detect = (state == S3);
always @ (posedge clk, negedge reset_b)
if (reset_b == 0) state <= S0; else state <= next_state;

   always @ (state, bit_in) begin
      next_state = S0;
      case (state)
         0:       if (bit_in) next_state = S1; else state = S0;
         1:       if (bit_in) next_state = S2; else next_state = S0;
         2:       if (bit_in) next_state = S3; else state = S0;
         3:       if (bit_in) next_state = S3; else next_state = S0;
         default:    next_state = S0;
      endcase
   end
endmodule

module t_Seq_Detector_Prob_5_45 ();
   wire detect;
   reg bit_in, clk, reset_b;

   Seq_Detector_Prob_5_45 M0 (detect, bit_in, clk, reset_b);

   initial #350$finish;
   initial begin clk = 0; forever #5 clk = ~clk; end
   initial fork
      #2 reset_b = 1;
      #3 reset_b = 0;
      #4reset_b = 1;
                        // Trace the state diagram and monitor detect (assert in S3)
               bit_in = 0;         // Park in S0
      #20 bit_in = 1;          // Drive to S0
      #30 bit_in = 0;          // Drive to S1 and back to S0 (2 clocks)
      #50 bit_in = 1;
      #70 bit_in = 0;          // Drive to S2 and back to S0 (3 clocks)
      #80 bit_in = 1;
      #130 bit_in = 0;         // Drive to S3, park, then    and back to S0
    join
endmodule
```



5.46 Pending simulation results

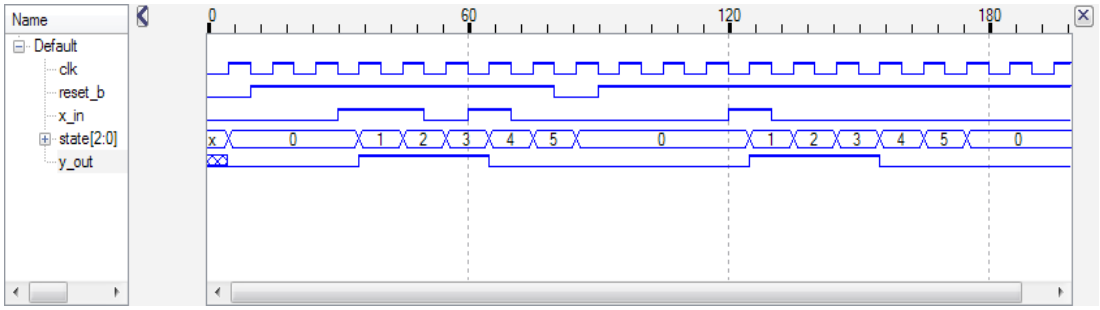
Assumption: Synchronous active-low reset
Moore machine



Verify that machine remains in state 000 while reset_b is asserted, independently of x_in.
Verify that machine makes transition from 000 to 001 if not reset_b and if x_in is asserted.
Verify that state transitions from 000 through 101 are correct.
Verify reset_b "on the fly."
Verify that y_out is asserted correctly.

```
module Prob_5_46 (output y_out, input x_in, clk, reset_b);
  reg [2:0] state, next_state;

  assign y_out = (state == 3'b001)||(state == 3'b010) || (state == 3'b011);
  always @ (posedge clk)
    if (reset_b == 1'b0) state <= 3'b000; else state <= next_state;

  always @ (x_in, state) begin
    next_state = 3'b000;
    case (state)
      3'b000: if (x_in) next_state = 3'b001; else next_state = 3'b000;
      3'b001: next_state = 3'b010;
      3'b010: next_state = 3'b011;
      3'b011: next_state = 3'b100;
      3'b100: next_state = 3'b101;
      3'b101: next_state = 3'b000;
      default:    next_state = 3'b000;
    endcase
  end
endmodule

module t_Prob_5_46 ();
  reg x_in, clk, reset_b;
  wire y_out;

  Prob_5_46 M0 (y_out, x_in, clk, reset_b);

  initial #200 $finish;
  initial begin clk = 0; forever #5 clk = !clk; end
  initial fork
    reset_b = 0;
    #10 reset_b   = 1;
    #80 reset_b = 0;
    #90 reset_b = 1;

    x_in = 0;
    #30 x_in = 1;
    #40 x_in = 1;
```

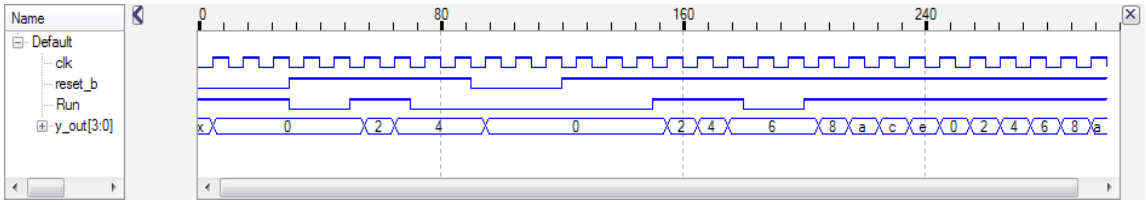
```
      #50 x_in = 0;
      #60 x_in = 1;
      #70 x_in = 0;
      #120 x_in = 1;
      #130 x_in = 0;
    join
endmodule
```



5.47

Assume synchronous active-low reset.

```
module Prob_5_47 (output reg [3:0] y_out, input Run, clk, reset_b);
  always @ (posedge clk)
    if (reset_b == 1'b0) y_out <= 4'b000;
    else if (Run && (y_out < 4'b1110)) y_out <= y_out + 2'b10;
    else if (Run && (y_out == 4'b1110)) y_out <= 4'b0000;
    else y_out <= y_out;      // redundant statement and may be omitted
endmodule

// Verify that counting is prevented while reset_b is asserted, independently of Run
// Verify that counting is initiated by Run if reset_b is de-asserted
// Verify reset on-the-fly
// Verify that deasserting Run suspends counting
// Verify wrap-around of counter.

module t_Prob_5_47 ();
  reg Run, clk, reset_b;
  wire [3:0] y_out;

  Prob_5_47 M0 (y_out, Run, clk, reset_b);

  initial #300 $finish;
  initial begin clk = 0; forever #5 clk = !clk; end
  initial fork
    reset_b = 0;
    #30 reset_b = 1;

    Run = 1;            // Attempt to run is overridden by reset_b
    #30 Run = 0;
    #50 Run = 1;      // Initiate counting
    #70 Run = 0;      // Pause
    #90 reset_b = 0; // reset on-the-fly
    #120 reset_b = 1;   // De-assert reset_b
    #150 Run = 1;         // Resume counting
    #180 Run = 0;         // Pause counting
    #200 Run = 1;         // Resume counting
  join
endmodule
```



5.48

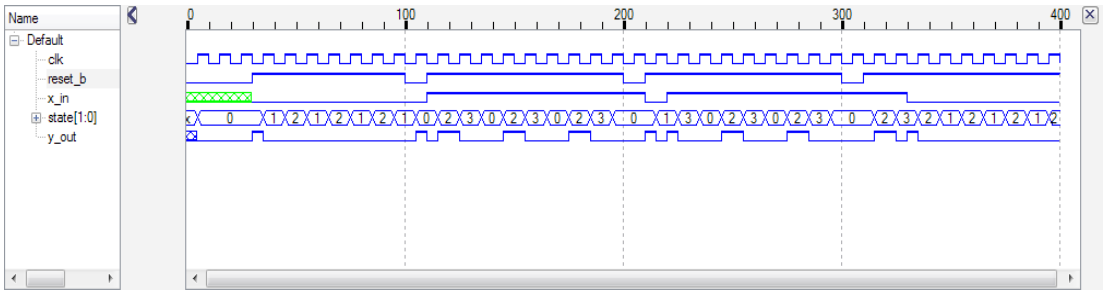
Assume "a" is the reset state.

```
module Prob_5_48 (output reg y_out, input x_in, clk, reset_b);
  parameter s_a = 2'd0;
  parameter s_b = 2'd1;
  parameter s_c = 2'd2;
  parameter s_d = 2'd3;
  reg [1: 0] state, next_state;

  always @ (posedge clk)
    if (reset_b == 1'b0) state <= s_a;
    else state <= next_state;

  always @ (state, x_in) begin
    next_state = s_a;
    y_out = 0;
    case (state)
      s_a:    if (x_in == 1'b0) begin next_state = s_b; y_out = 1; end
              else begin next_state = s_c; y_out = 0; end
      s_b:    if (x_in == 1'b0) begin next_state = s_c; y_out = 0; end
              else begin next_state = s_d; y_out = 1; end
      s_c:    if (x_in == 1'b0) begin next_state = s_b; y_out = 0; end
              else begin next_state = s_d; y_out = 1; end
      s_d:    if (x_in == 1'b0) begin next_state = s_c; y_out = 1; end
              else begin next_state = s_a; y_out = 0; end
      default:    begin next_state = s_a; y_out = 0; end
    endcase
  end
endmodule
```

Verify reset action.
Verify state transitions.
    Transition to a; hold x_in = 0 and get loop bc…
    Transition to a; hold x_in = 1 and get loop acda…
    Transitons to b; hold x_in = 1 and get loop bdacd…
    Transition to d; hold x_in = 0 and get loop dcbc…
Confirm Mealy outputs at each state/input pair
Verify reset on-the-fly.

```
module t_Prob_5_48 ();
  reg x_in, clk, reset_b;
  wire y_out;

  Prob_5_48 M0 (y_out, x_in, clk, reset_b);

  initial #400 $finish;
  initial begin clk = 0; forever #5 clk = !clk; end
  initial fork
    reset_b = 0;
```

```
        #30 reset_b = 1;
        #30 x_in = 0;          // loop abcbcbc…

        #100 reset_b = 0;
        #110 reset_b = 1;
        #110 x_in = 1;             // loop acdacda…

        #200 reset_b = 0;
        #210 reset_b = 1;
        #210 x_in = 0;
        #220 x_in = 1;             // loop bdacdacd…

        #300 reset_b = 0;
        #310 reset_b = 1;
        #310 x_in = 1;
        #330 x_in = 0;             // loop acdcbcbc….
    join
    endmodule
```

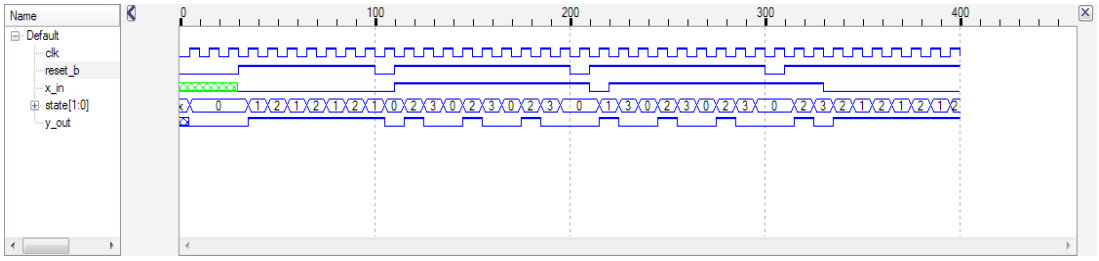


5.49

Assume "a" is the reset state.

```
module Prob_5_49 (output reg y_out, input x_in, clk, reset_b);
  parameter s_a = 2'd0;
  parameter s_b = 2'd1;
  parameter s_c = 2'd2;
  parameter s_ =d 2'd3;
  reg [1: 0] state, next_state;

  always @ (posedge clk)
    if (reset_b == 1'b0) state <= s_a;
    else state <= next_state;

  always @ (state, x_in) begin
    next_state = s_a;
    y_out = 1'b0;
    case (state)
      s_a:    if (x_in == 1'b0) next_state = s_b;
              else next_state = s_c;
      s_b:    begin y_out = 1'b1; if (x_in == 1'b0) next_state = s_c;
              else   next_state = s_d; end
      s_c:    begin y_out = 1'b1; if (x_in == 1'b0) next_state = s_b;
              else next_state = s_d; end
      s_d:    if (x_in == 1'b0) next_state = s_c;
              else next_state = s_a;
      default:    next_state = s_a;
    endcase
  end
endmodule
```

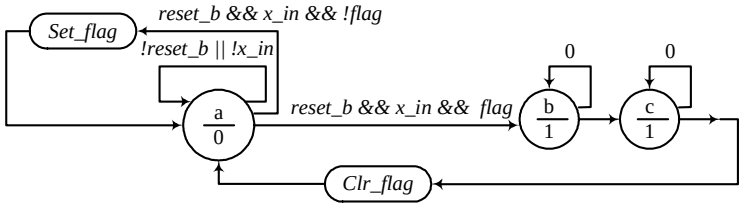
```
// Verify reset action.
// Verify state transitions.
     // Transition to a; hold x_in = 0 and get loop abcbc…
     // Transition to a; hold x_in = 1 and get loop acda…
     // Transitons to b; hold x_in = 1 and get loop bdacd…
     // Transition to d; hold x_in = 0 and get loop dcbc…
// Confirm Moore outputs at each state
// Verify reset on-the-fly.

          module t_Prob_5_49 ();
            reg x_in, Run, clk, reset_b;
            wire y_out;

            Prob_5_49 M0 (y_out, x_in, clk, reset_b);

            initial #400 $finish;
            initial begin clk = 0; forever #5 clk = !clk; end
            initial fork
               reset_b = 0;
               #30 reset_b = 1;
               #30 x_in = 0;          // loop abcbcbc…

               #100 reset_b = 0;
               #110 reset_b = 1;
               #110 x_in = 1;              // loop acdacda…

               #200 reset_b = 0;
               #210 reset_b = 1;
               #210 x_in = 0;
               #220 x_in = 1;              // loop bdacdacd…

               #300 reset_b = 0;
               #310 reset_b = 1;
               #310 x_in = 1;
               #330 x_in = 0;              // loop acdcbcbc….
          join
          endmodule
```



5.50

The machine is to remain in its initial state until a second sample of the input is detected to be 1. A flag will be set when the first sample is obtained. This will enable the machine to detect the presence of the second sample while being in the initial state. The machine is to assert its output upon detection of the second sample and to continue asserting the output until the fourth sample is detected.

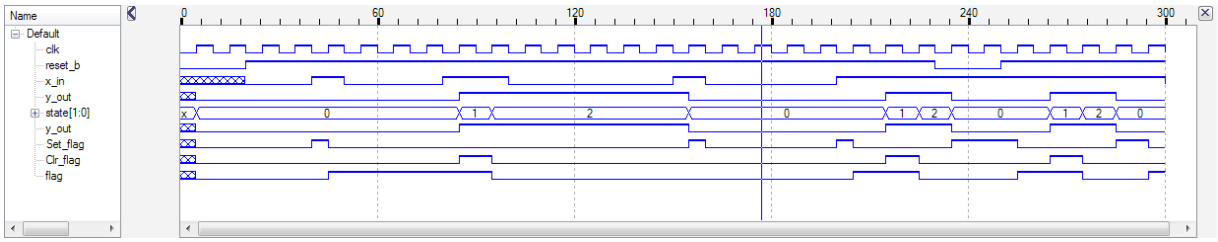
Assumption: Synchronous active-low reset
Moore machine, links for reset on-the-fly are implicit and not shown



Note: the output signal *y_out* is a Moore-type output. The control signals *Set_flag* and *Clr_flag* are not.

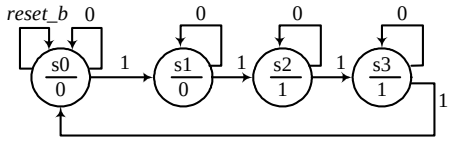
```
module Prob_5_50 (output y_out, input x_in, clk, reset_b);
   parameter s_a = 2'd0;
   parameter s_b = 2'd1;
   parameter s_c = 2'd2;

   reg Set_flag;
   reg Clr_flag;
   reg [1:0] state, next_state;
   assign y_out = (state == s_b) || (state == s_c) ;
   always @ (posedge clk)
      if (reset_b == 1'b0) state <= s_a;
      else state <= next_state;

   always @ (state, x_in, flag) begin
      next_state = s_a;
      Set_flag = 0;
      Clr_flag = 0;
      case (state)
         s_a:     if ((x_in == 1'b1) && (flag == 1'b0))
                     begin next_state = s_a;    Set_flag = 1; end
                     else if ((x_in == 1'b1) && (flag == 1'b1))
                     begin next_state = s_b; Set_flag = 0; end
                     else if (x_in == 1'b0) next_state = s_a;
         s_b:     if (x_in == 1'b0) next_state = s_b;
                     else begin next_state = s_c; Clr_flag = 1; end
         s_c:     if (x_in == 1'b0) next_state = s_c;
                     else next_state = s_a;
         default: begin next_state = s_a; Clr_flag = 1'b0; Set_flag = 1'b0; end
      endcase
   end

   always @ (posedge clk)
      if (reset_b == 1'b0) flag <= 0;
      else if (Set_flag) flag <= 1'b1;
      else if (Clr_flag) flag <= 1'b0;
endmodule
```

```
// Verify reset action
// Verify detection of first input
// Verify wait for second input
// Verify transition at detection of second input
// Verify with between detection of input
// Verify transition to s_d at fourth detection of input
// Verify return to s_a and clearing of flag after fourth input
// Verify reset on-the-fly

module t_Prob_5_50 ();
   wire y_out;
   reg x_in, clk, reset_b;

   Prob_5_50 M0 (y_out, x_in, clk, reset_b);

   initial #500 $finish;
   initial begin clk = 0; forever #5 clk = !clk; end
   initial fork
      reset_b = 1'b0;
      #20 reset_b = 1;
      #20 x_in = 1'b0;
      #40 x_in = 1'b1;
      #50 x_in = 1'b0;
      #80 x_in = 1'b1;
      #100 x_in = 0;
      #150 x_in = 1'b1;
      #160 x_in = 1'b0;
      #200 x_in = 1'b1;
      #230 reset_b = 1'b0;
      #250 reset_b = 1'b1;
      #300 x_in = 1'b0;
   join
endmodule
```



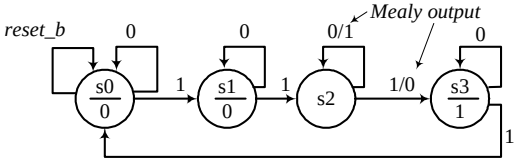
5.51

Assumption: Synchronous active-low reset
Moore machine, links for reset on-the-fly are implicit and not shown



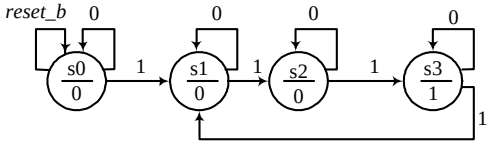
5.52

Assumption: Synchronous active-low reset
Moore/Mealy machine, links for reset on-the-fly are implicit and not shown



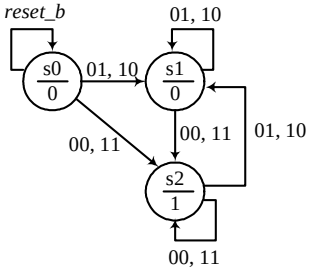
5.53

Assumption: Synchronous active-low reset
Moore machine, links for reset on-the-fly are implicit and not shown



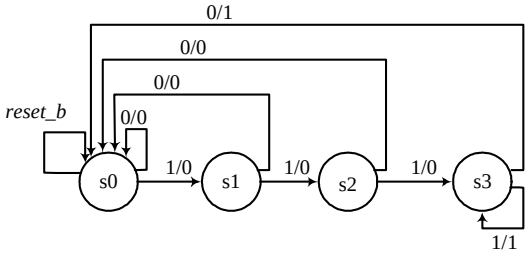
5.54

Assumption: Synchronous active-low reset
Moore machine, links for reset on-the-fly are implicit and not shown



5.55

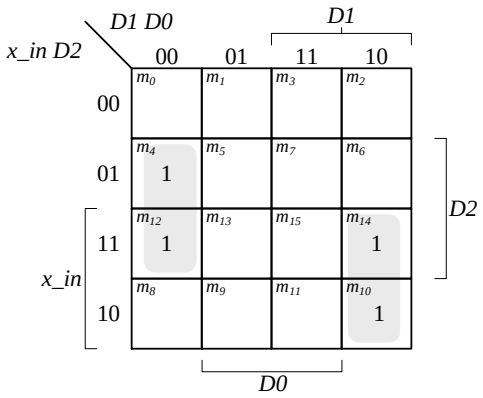
Assumption: Synchronous active-low reset
Mealy machine, links for reset on-the-fly are implicit and not shown



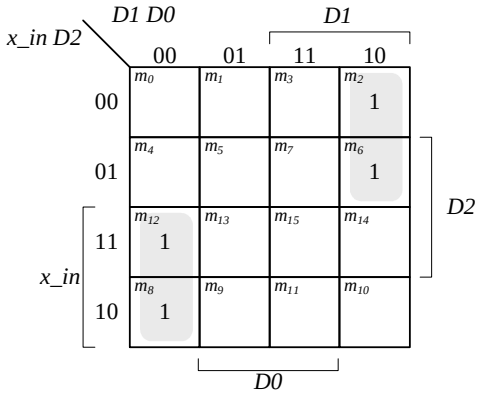
5.56

reset_b	x_in	D2	D1	D0	nD2	nD1	nD0		
0	x	0	0	0		0	0	0	
0	x	0	0	1		0	0	0	
0	x	0	1	0		0	0	0	
0	x	0	1	1		0	0	0	
0	x	1	0	0		0	0	0	
0	x	1	0	1		0	0	0	
0	x	1	1	0		0	0	0	
0	x	1	1	1		0	0	0	
1	0	0	0	0		0	0	0	
1	1	0	0	0		0	1	0	
1	x	0	0	1		0	0	0	
1	0	0	1	0		0	1	0	
1	1	0	1	0		1	0	0	
1	x	0	1	1		0	0	0	
1	0	1	0	0		1	0	0	
1	1	1	0	0		1	1	0	
1	x	1	0	1		0	0	0	
1	0	1	1	0		1	1	0	
1	1	1	1	0		0	0	0	
1	x	1	1	1		0	0	0	

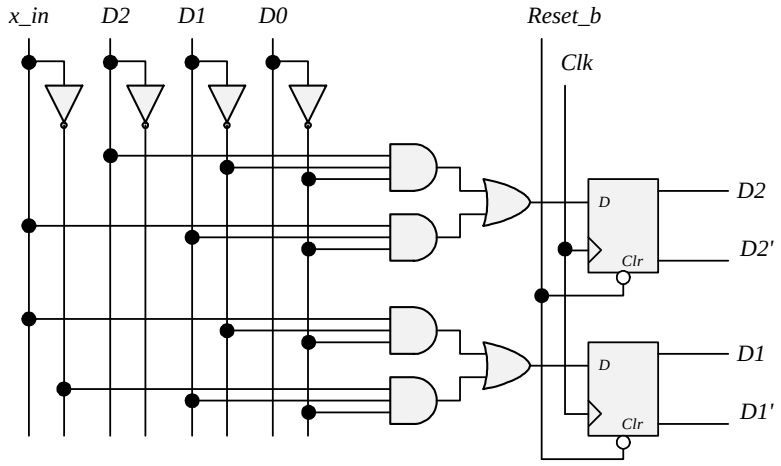
For reset_b = 1:
nD2 = (x_in D2'D1D0') || (x_in' D2 D1' D0') || (x_in D2 D1' D0') || (x_in D2 D1 D0')
nD1 = (x_in D2' D1' D0') || (x_in' D2' D1 D0') || (x_in D2 D1' D0') || (x_in' D2 D1 D0')



nD2 = D2 D1'D0' + x_in D1 D0'



nD1 = x_inD1' D0' + x_in' D1 D0'

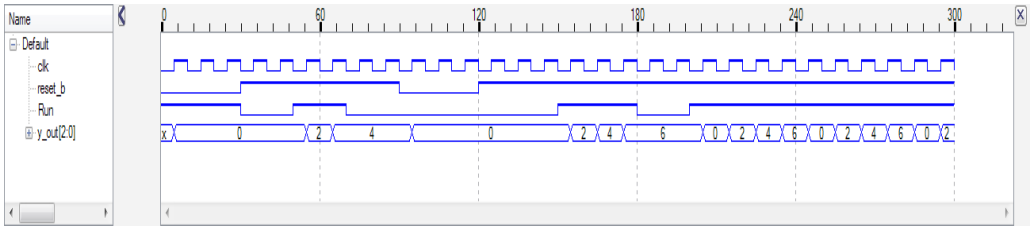


5.57

Assume synchronous active-low reset. Assume that the counter is controlled by assertion of *Run*.

```
module Prob_5_57 (output reg [2:0] y_out, input Run, clk, reset_b);
   always @ (posedge clk)
      if (reset_b == 1'b0) y_out <= 3'b000;
      else if (Run && (y_out < 3'b110)) y_out <= y_out + 3'b010;
      else if (Run && (y_out == 3'b110)) y_out <= 3'b000;
      else y_out <= y_out;      // redundant statement and may be omitted
endmodule

// Verify that counting is prevented while reset_b is asserted, independently of Run
// Verify that counting is initiated by Run if reset_b is de-asserted
// Verify reset on-the-fly
// Verify that deasserting Run suspends counting
// Verify wrap-around of counter.

module t_Prob_5_57 ();
   reg Run, clk, reset_b;
   wire [2:0] y_out;

   Prob_5_57 M0 (y_out, Run, clk, reset_b);
```

```
  initial #300 $finish;
  initial begin clk = 0; forever #5 clk = !clk; end
  initial fork
    reset_b = 0;
    #30 reset_b = 1;

    Run = 1;            // Attempt to run is overridden by reset_b
    #30 Run = 0;
    #50 Run = 1;      // Initiate counting
    #70 Run = 0;      // Pause
    #90 reset_b = 0; // reset on-the-fly
    #120 reset_b = 1;   // De-assert reset_b
    #150 Run = 1;         // Resume counting
    #180 Run = 0;         // Pause counting
    #200 Run = 1;         // Resume counting
  join
endmodule
```



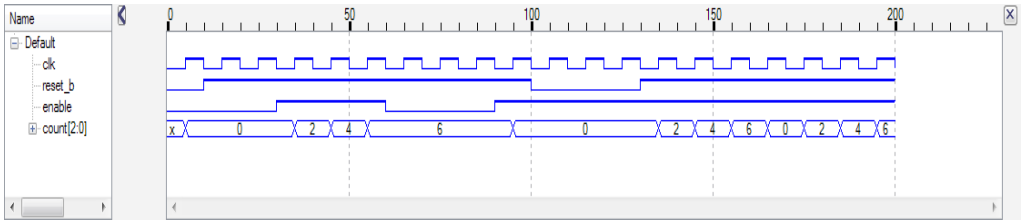
5.58

```
module Prob_5_58 (output reg y_out, input x_in, clk, reset_b)
  parameter s0 = 2'b00;
  parameter s1 = 2'b01;
  parameter s2= 2'b10;
  parameter s3 = 2'b11;
  reg [1:0] state, next_state;

  always @ (posedge clk, negedge reset_b)
    if (reset_b == 1'b0) state <= s0;
    else state <= next_state;

  always @(state, x_in) begin
    y_out = 0;
    next_state = s0;
    case(state)
      s0: if (x_in == 1'b0) next_state = s0; else if (x_in = 1'b1) next_state = s1;
      s1: if (x_in == 1'b0) next_state = s0; else if (x_in = 1'b1) next_state = s2;
      s2: if (x_in == 1'b0) next_state = s0; else if (x_in = 1'b1) next_state = s3;
      s3: if (x_in == 1'b0) next_state = s0; else if (x_in = 1'b1) begin    next_state = s3; y_out = 1;
end
      default:begin next_state = s0; y_out = 0; end
    endcase
  end
endmodule

module t_Prob_5_58 ();
  wire y_out;
  reg x_in, clk, reset_b;

  Prob_5_58 M0 (y_out, x_in, clk, reset_b)
```

```
  initial begin clk = 0; forever #5 clk = !clk; end
  initial fork
    reset_b = 0;
    x_in = 0;
    #20 reset_b = 1;
    #40 reset_b = 1;
    #50 x_in = 1;
    #60 x_in = 0;
    #80 x_in = 1;
    #90 x+in = 0;
    #110 x_in = 1;
    #120 x_in = 1;
    #150 x_in = 0;
    #200 x_in = 1;
    #210 reset_b = 0;
    #240 reset_b = 1;
  join
endmodule
```

5.59

```
module Prob_5_59 (output reg [2: 0] count, input enable, clk, reset_b);
  always @ (posedge clk)
    if (reset_b == 1'b0) count <= 3'b000;
    else if (enable) case (count)
      3'b000: count <= 3'b010;
      3'b010: count <= 3'b100;
      3'b100: count <= 3'b110;
      3'b110: count <= 3'b000;
      default:    count <= 3'b111;       // Use for error detection
    endcase
endmodule

module t_Prob_5_59 ();
  wire [2:0] count;
  reg enable, clk, reset_b;

  Prob_5_59 M0 (count, enable, clk, reset_b);

  initial #200 $finish;
  initial begin clk = 0; forever #5 clk = ~clk; end
  initial fork
    reset_b   = 0;
    #10 reset_b = 1;
    #100 reset_b = 0;
    #130 reset_b = 1;
    enable = 0;
    #30 enable = 1;
    #60 enable = 0;
    #90 enable = 1;
  join
endmodule
```



5.60

Assume synchronous active-low reset. Assume that counting is controlled by Run.

```
module Prob_5_60 (output reg [3:0] y_out, input Run, clk, reset_b);
   always @ (posedge clk)
      if (reset_b == 1'b0) y_out <= 4'b000;
      else if (Run && (y_out < 4'b1001)) y_out <= y_out + 4'b0001;
      else if (Run && (y_out == 4'b1001)) y_out <= 4'b0000;500
      else y_out <= y_out;      // redundant statement and may be omitted
endmodule

// Verify that counting is prevented while reset_b is asserted, independently of Run
// Verify that counting is initiated by Run if reset_b is de-asserted
// Verify reset on-the-fly
// Verify that deasserting Run suspends counting
// Verify wrap-around of counter.

module t_Prob_5_60 ();
   reg Run, clk, reset_b;
   wire [3:0] y_out;

   Prob_5_60 M0 (y_out, Run, clk, reset_b);
   initial #500 $finish;
   initial begin clk = 0; forever #5 clk = !clk; end
   initial fork
      reset_b = 0;
      #30 reset_b = 1;

      Run = 1;            // Attempt to run is overridden by reset_b
      #30 Run = 0;
      #50 Run = 1;      // Initiate counting
      #70 Run = 0;      // Pause
      #90 reset_b = 0; // reset on-the-fly
      #120 reset_b = 1;   // De-assert reset_b
      #150 Run = 1;         // Resume counting
      #180 Run = 0;         // Pause counting
      #200 Run = 1;         // Resume counting
   join
endmodule
```