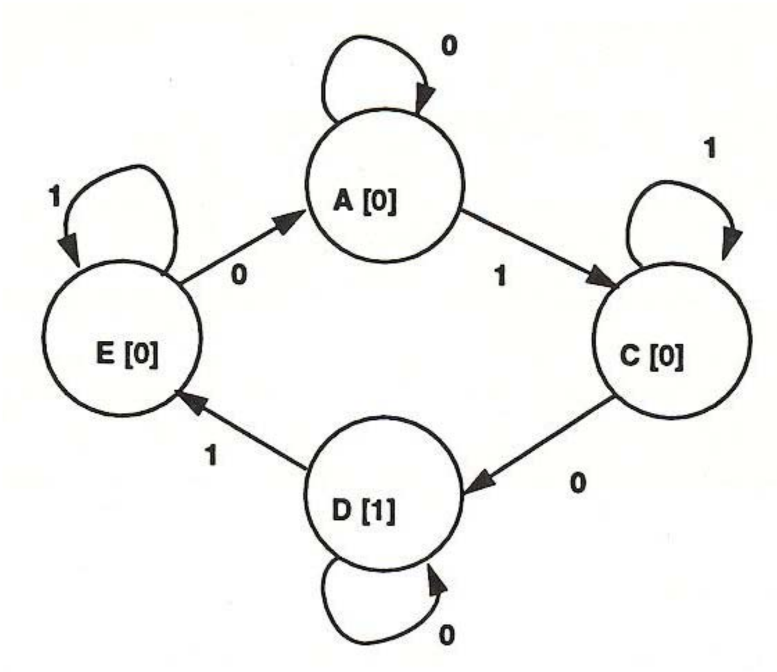
Exercise 8.1

B	B - A C - C					
C	B - D C - C	A - D C - C				
D	X	X	X			
E	B - A C - F	A - A C - F	D - A C - F	X		
F	B - B C - G	A - B C - G	D - B C - G	X	A - B F - G	
G	B - A C - E	A - A C - E	D - A C - E	X	A - A F - E	B - A G - E
	A	B	C	D	E	F

B	B - A C - G					
C	X	X				
D	X	X	X			
E	X	X	X	X		
F	X	X	X	X	A - B F - G	
G	X'	X	X	X	A - A F - E	B - A G - E
	A	B	C	D	E	F

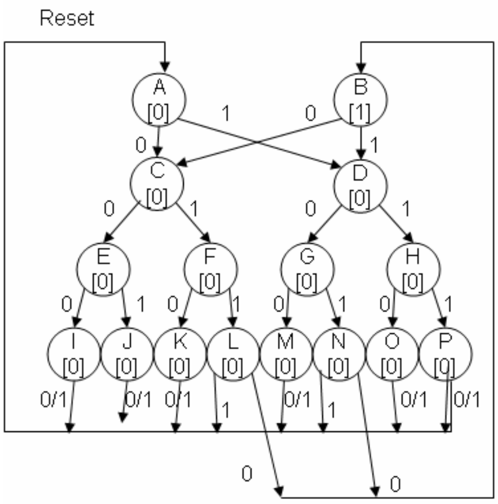
Exercise 8.2

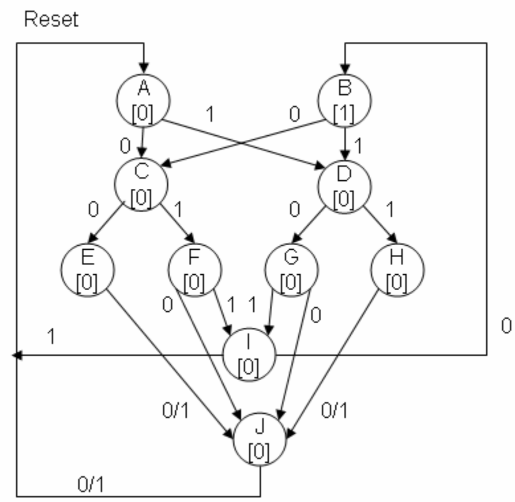
States A and B (only A is shown below) are equivalent as are E, F, and G (only E is shown below).



Exercise 8.3

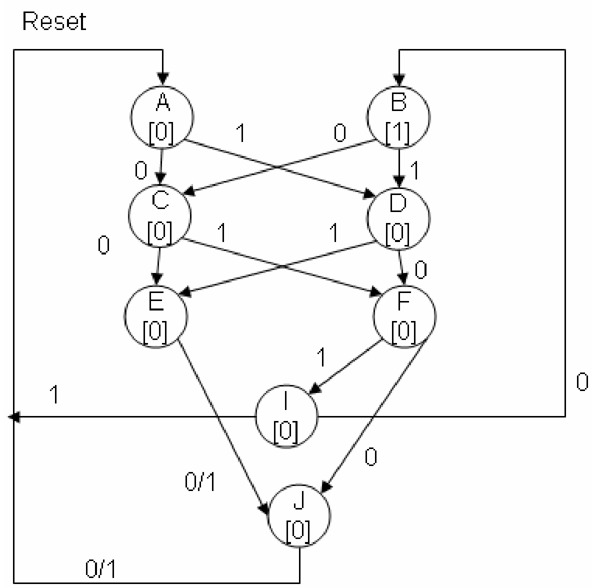
Since there are a lot of states in this example, some pre-reduction has been done before the implication chart to make the implication chart smaller. The image on the left, is the state diagram before any reduction, and the image on the right is the state diagram being used in the implication chart.





	A	B	C	D	E	F	G	H	I
B									
C	~~C – E, D – F~~								
D	~~C – G, D – H~~		~~E – G, F – H~~						
E	~~C – J, D – J~~		~~E – J, F – J~~	~~G – J, H – J~~					
F	~~C – J, D – I~~		~~E – J, F – I~~	~~G – J, H – I~~	~~J – J, J – I~~				
G	~~C – J, D – I~~		~~E – J, F – I~~	~~G – J, H – I~~	~~J – J, J – I~~	J – J, I – I			
H	~~C – J, D – J~~		~~E – J, F – J~~	~~G – J, H – J~~	J – J, J – J	~~J – J, I – J~~	~~J – J, I – J~~		
I	~~C – A, D – B~~		~~E – A, F – B~~	~~G – A, H – B~~	~~J – A, J – B~~	~~J – A, I – B~~	~~J – A, I – B~~	~~J – A, J – B~~	
J	~~C – A, D – A~~		~~E – A, F – A~~	~~G – A, H – A~~	~~J – A, J – A~~	~~J – A, I – A~~	~~J – A, I – A~~	~~J – A, J – A~~	~~A – A, A – B~~

The final state diagram comes out as follows:



Exercise 8.4

S1	S1 - S2 S1 - S4					
S2	S1 - S1 S4 - S6	S1 - S2 S1 - S6				
S3	S1 - S1 S3 - S4	S1 - S2 S1 - S3	S1 - S1 S3 - S6			
S4	S1 - S5 S4 - S4	S2 - S5 S1 - S4	S1 - S5 S4 - S6	S1 -S5 S3 - S4		
S5	S1 - S2 S1 - S4	S2 -S2 S1 - S1	S1 - S2 S1 - S6	S2 - S1 S1 - S3	S2 - S5 S1 - S4	
S6	X	X	X	X	X	X
	S0'	S1	S2	S3	S4	S5

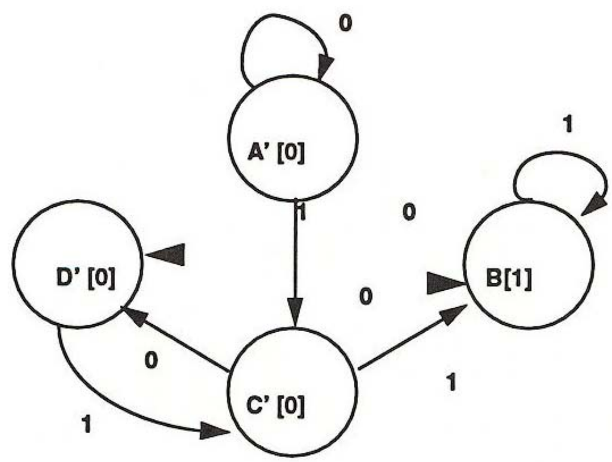
S0 = S4, S1 = S5.

Exercise 8.5

	A	B	C	D	E	F
B	X					
C	A - D C - B	X				
D	A - B C - C	X	D - B B - C			
E	A - E C - F	X	D - E G - F	B - E C - F		
F	A - G C - B	X	D - G B - B	B - G C - B	E - G F - B	
G	A - B C - F	X	D - B B - F	B - B C - F	E - B F - F	G - B B - F

A = E, C = F, D = G.

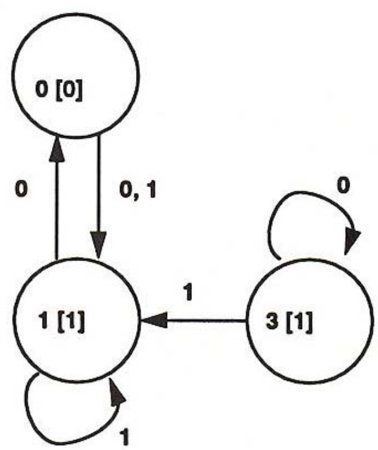
Any string of 2 or more 1s or a double 0 will cause a 1 output.



Exercise 8.6

	0	1	2	3	4	5
1	X					
2	x	0 - 0 1 - 5				
3	x	0 - 3 1 - 1	0 - 3 1 - 5			
4	x	0 - 0 1 - 5	0 - 0 5 - 5	0 - 3 1 - 5		
5	x	0 - 0 1 - 5	0 - 0 5 - 5	0 - 3 1 - 5	0 - 0 5 - 5	
6	5 - 5 2 - 4	X	x	x	x	x

1 = 2 = 4 = 5



Exercise 8.7

a.) Minimum bit change heuristic

7 states, 3 variable K-map

Assign S0 = 000 = (Q2, Q1, Q0)
S0 adjacent to S1, S2
Either S4 adjacent to S1, S2, or S3 adjacent to S1, try both cases.

Q2 \ Q1Q0	00	01	11	10
0	S0	S1	S4	S2
1	S5	S3	S6	

b.) State assignment guidelines

Highest priority: 2x (S5, S6), 2x (S1, S2) (S3, S4)
Medium priority: 2x (S3, S4), (S1, S2), (S5, S6)
Lowest priority: 0/0: (S4, S0, S6, S5)
1/0: (S0, S2, S1, S4, S5)
0/1: (S1, S2, S3)
1/1: (S6, S3)

Q2 \ Q1Q0	00	01	11	10
0	S0	S5	S4	S1
1		S6	S3	S2

Satisfy all high and medium priority.
Try to satisfy as many lowest priority as possible.

Exercise 8.8

High Priority: (B, C), (E, A)
Medium Priority: (A, D), (D, C)
Lowest Priority: 0/0: (E, D, C, B)
1/0: (B, C, A)

Q2 \ Q1Q0	00	01	11	10
0	A	E	B	
1	D	C		

Exercise 8.9

Let Q_1, Q_0 be the current state bits and P_1, P_0, be the next state bits with C, T_L, T_S as inputs and S_T, H_1, H_0, F_1, F_0.

Suppose the random 2-bit assignment is:

$HG = Q_0'Q_1'$

$HY = Q_0Q_1$

$FG = Q_0'Q_1$

$FY = Q_0Q_1'$

Produces the following function:

$P_0 = T_S' Q_0$

$P_1 = Q_1 W$

$ST = T_S Q_0 + Q_0' Z$

$H_1 = Q_1 \text{ xor } Q_0$

$H_0 = Q_0Q_1$

$F_1 = Q_1 \text{ xnor } Q_0$

$F_0 = Q_0Q_1'$

$W = Q_0 + Q_0'CT_L'$

$X = Q_0$

$Y = C' + T_L$

$Z = CT_L Q_1' + Y Q_1$

This encoding has 28 literals.

Using the random 3-bit assignment:

$HG = Q_0Q_1'Q_2'$

$HY = Q_0Q_1Q_2'$

$FG = Q_0'Q_1'Q_2$

$FY = Q_0'Q_1Q_2$

Generates the encoding:

$P_0 = Q_0Q_1' + T_S\ F_0 + T_S' H_0$

$P_1 = T_S' Q_1$

$P_2 = FG + T_S H_0 + T_S' F_0$

$ST = Q_1' W + T_S Q_1$

$H_1 = Q_1$

$H_0 = Q_0Q_1$

$F_1 = Q_2$

$F_0 = Q_0'Q_1$

$Y = C' + T_L$

$W = CT_L\, Q_0 + Y\, Q_2$

This encoding has only 30 literals.

Finally trying a four-bit encoding:

$HG = Q_0Q_1'Q_2'Q_3'$

$HY = Q_0Q_1Q_2'Q_3'$

$FG = Q_0'Q_1'Q_2Q_3$

$FY = Q_0'Q_1'Q_2'Q_3$

Generates:

$P_0 = P_1 + HG$

$P_1 = T_S'\, Q_1$

$P_2 = T_S'\, Q_2$

$P_3 = FG + P_2$

$ST = CT_L\, HG + T_S\, X + Y\, FG$

$H_1 = Q_3$

$H_0 = Q_1$

$F_1 = Q_0$

$F_0 = Q_2$

$FG = Q_2'Q_3$

$HG = Q_0Q_1'$

$X = Q_1 + Q_2$

$Y = C + T_L$

This encoding also gives 27 literals.

Exercise 8.10

Using the solution for 8.9, shows that varying the number of bits used to encode the states, and choosing different bit encodings can provide more or less optimal solutions. For example, the 4-bit encoding used has fewer literals than both the 2 and 3 bit encoding, and the 3-bit encoding has more literals than the 2-bit encoding. Finding the most optimal encoding when varying both the number of bits and the different encodings of each state is an extremely difficult problem. This is because it is not inherently easy to determine whether or not one encoding will be better than all others, unless all the other encodings are tried. For *m* states and *n* bits, $m! / (m - 2^n)!$ where $n \leq \log_2 m$ different possible encodings.

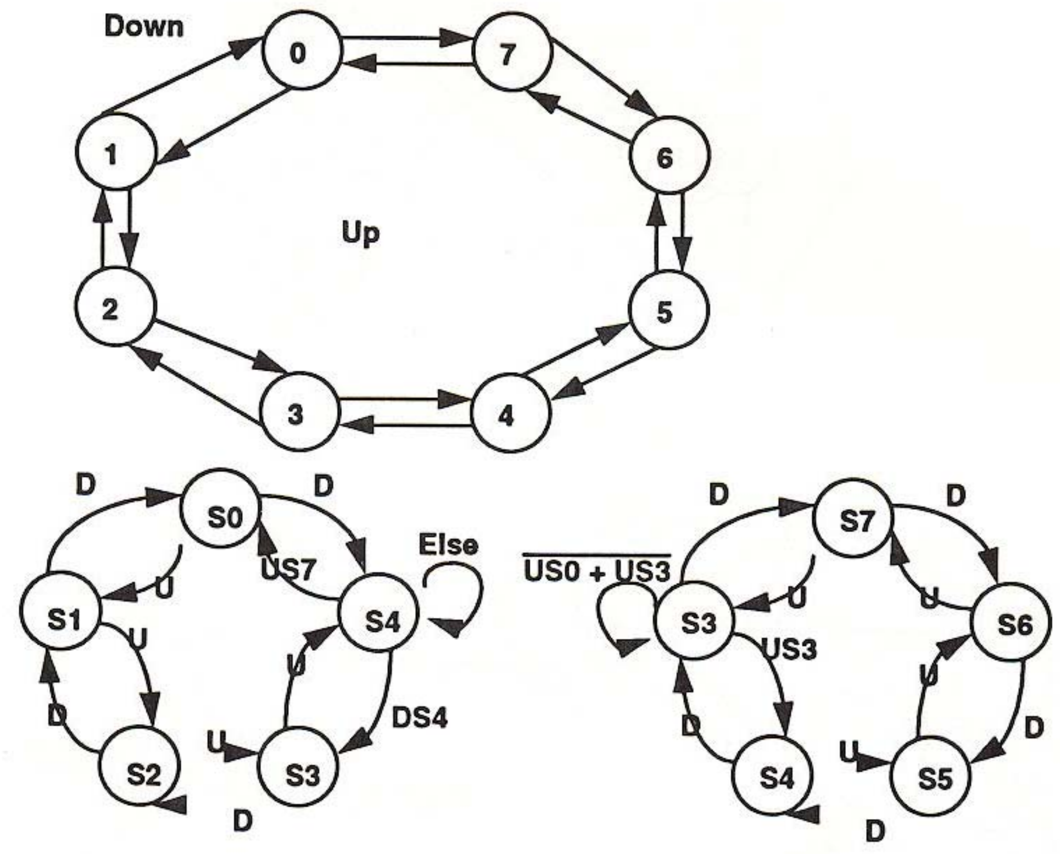
Exercise 8.11

This problem was not completely specified and should be ignored until the text is corrected.

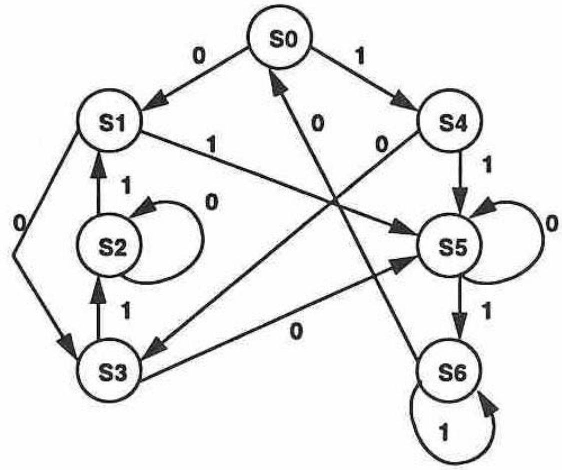
Exercise 8.12

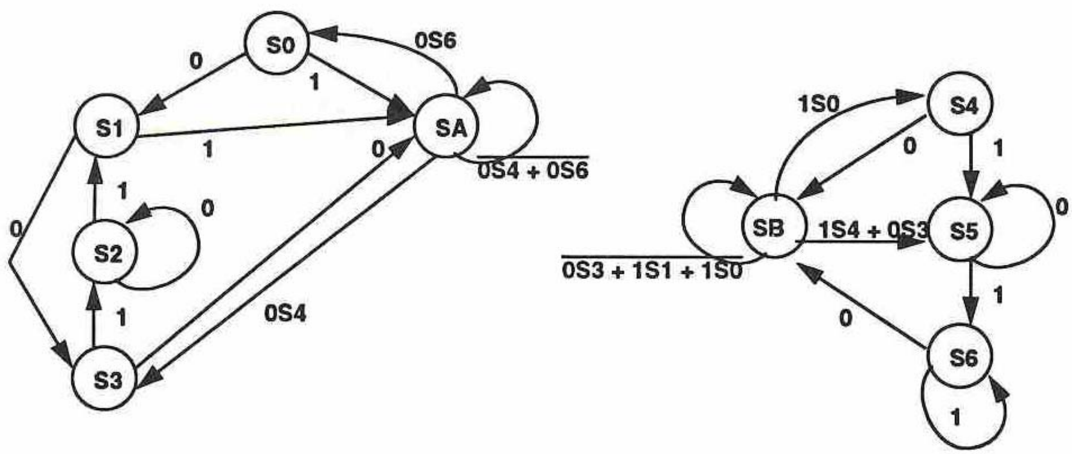
This problem was not completely specified and should be ignored until the text is corrected.

Exercise 8.13

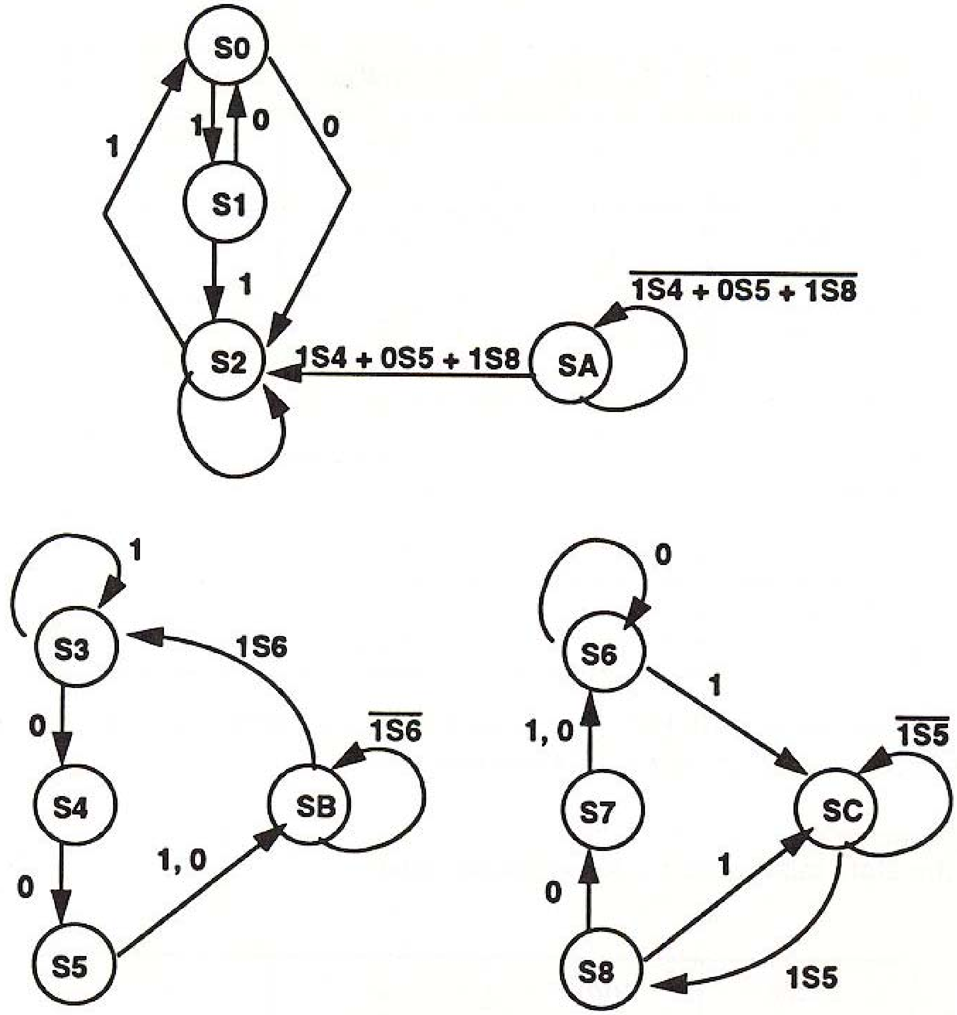


Exercise 8.14





Exercise 8.15



Exercise 8.16

a.) Mealy partitioning rule modification

Each branch that enters the idle state must have an "idle" output, and transfers out of the idle state must be associated with the proper output.

b.) Moore partitioning rule modification

The idle state must have an "idle" output. Combine them by tri-stating one section when in the idle state.

Exercise 8.17

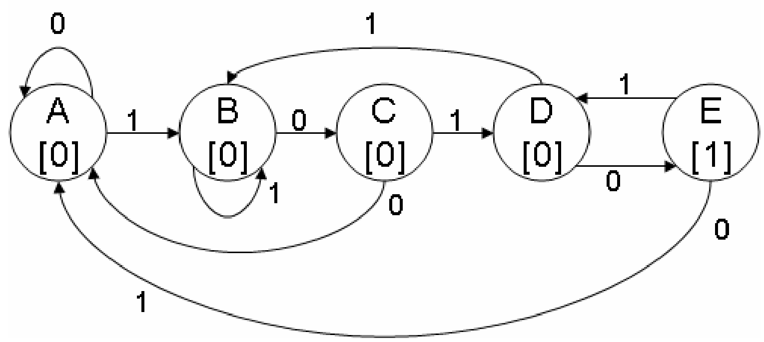
State	Input	Next State	Output
000	0 1	000 010	0 0
001	0 1	100 000	1 0
010	0 1	001 100	0 1
011	0 1	001 010	1 0
100	0 1	101 011	1 0
101	0 1	011 101	0 1

For implementation with D-FFs, minimize the above table with espresso or K-maps. The resulting equations are the next state and output functions.

Exercise 8.18

At the writing of this solution manual, there was a mistake in the printing of Figure Ex 8.18. This is due to the non-deterministic behavior illustrated by the two 0 arcs leaving state D. The arc from D to A on 0 should actually be from E to A.

The actual figure should look like:



The next-state function for this FSM looks like the following:

Current State	Encoding	Input	Reset	Next State	Output
Any	XXX	X	1	000	0
A	000	0	0	000	0
	000	1	0	001	0
B	001	0	0	011	0
	001	1	0	001	0
Invalid	010	X	0	000	0
C	011	0	0	000	0
	011	1	0	111	0
Invalid	100	X	0	000	0
E	101	0	0	000	1
	101	1	0	111	1
Invalid	110	X	0	000	0
D	111	0	0	101	0
	111	1	0	001	0

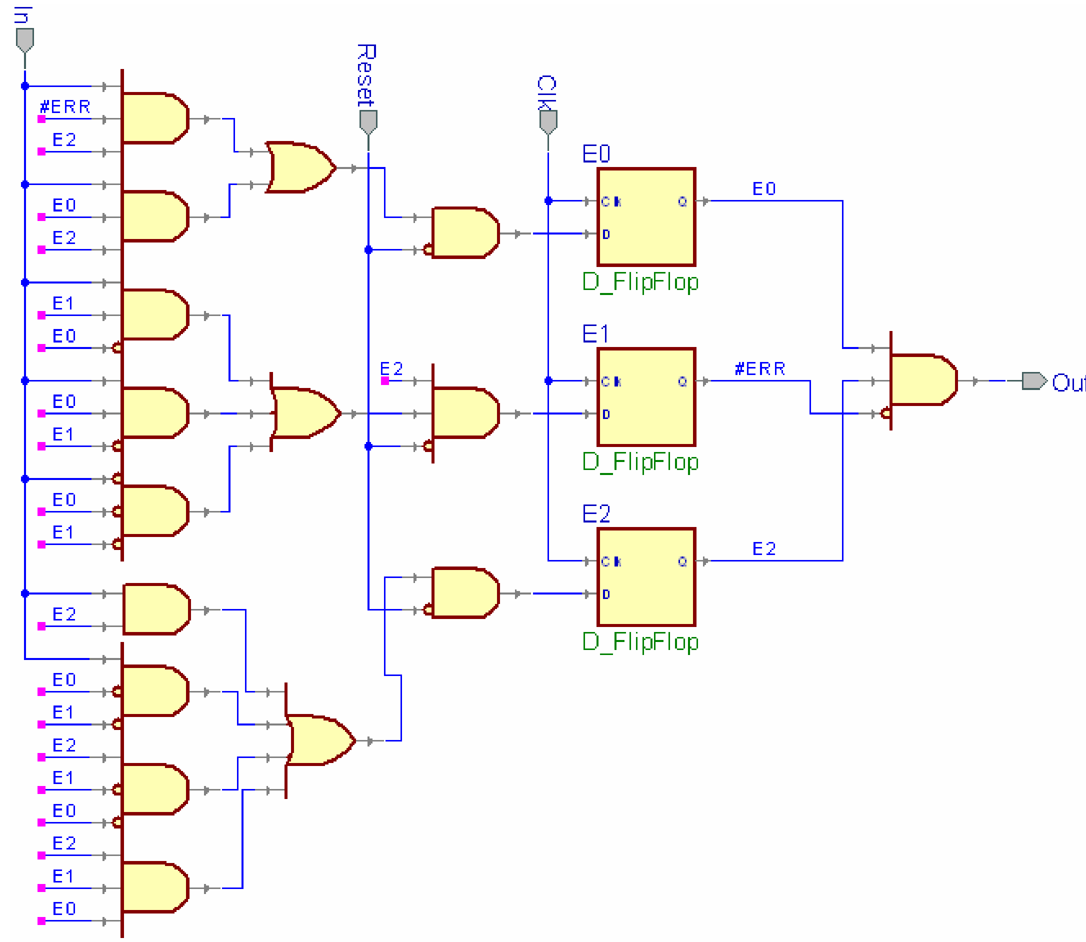
Using K-maps, the Encoding bits E_0, E_1, E_2 and input I translate into N_0, N_1, N_2 and O in the following reduced equations:

$N_0 = \text{Reset}' \, (I\, E_1\, E_2 + I\, E_0\, E_2)$

$N_1 = \text{Reset}' \, (I\, E_0\, E_1'\, E_2 + I\, E_0'\, E_1\, E_2 + I'\, E_0'\, E_1\, E_2)$

$N_2 = \text{Reset}' \, (I\, E_2 + E_0\, E_1\, E_2 + E_0'\, E_1'\, E_2 + I\, E_0'\, E_1' + E_0'\, E_1'\, E_2)$

$O = E_0\, E_1'\, E_2$



In
Reset
Clk
#ERR
E2
E0
E2
E1
E0
E0
E1
E0
E1
E2
E0
E1
E2
E1
E0
E2
E1
E0
E2
E0
E1
E2
D_FlipFlop
D_FlipFlop
D_FlipFlop
E0
#ERR
E2
Out

Exercise 8.19

TS	TL	C	Reset	State	HR	HY	HG	FR	FY	FG
X	X	X	1	6’bxxxxxx	0	0	1	1	0	0
X	0	X	0	Highwaygreen	0	0	1	1	0	0
X	X	0	0	Highwaygreen	0	0	1	1	0	0
X	1	1	0	Highwaygreen	0	1	0	1	0	0
0	X	X	0	Highwayyellow	0	1	0	1	0	0
1	X	X	0	Highwayyellow	1	0	0	0	0	1
X	0	1	0	Farmroadgreen	1	0	0	0	0	1
X	1	X	0	Farmroadgreen	1	0	0	0	1	0
X	0	0	0	Farmroadgreen	1	0	0	0	1	0
0	X	X	0	Farmroadyellow	1	0	0	0	1	0
1	X	X	0	Farmroadyellow	0	0	1	1	0	0

HR = (Highwayyellow)(TS) + (FarmroadGreen) + (Farmroadyellow)(TS)’

HY = (Highwaygreen)(C)(TL) + (Highwayyellow)(TS)’

HG = (Highwaygreen)(C)’ + (Highwaygreen)(TL)’ + (Farmroadyellow)(TS)

FR = (Farmroadyellow)(TS) + (Highwaygreen) + (Highwayyellow)(TS)’

FY = (Farmroadgreen)(C) + (Farmroadgreen)(TL) + (Farmroadyellow)(TS)’

FG = (Farmroadgreen)(C)(TL)’ + (Highwayyellow)(TS)

Exercise 8.20

```
module prob8_20 ( Out ,Clk ,Reset ,In );
input Clk, Reset, In ;
wire Clk, Reset, In ;
output Out ;

parameter S0 = 3'b000;
parameter S1 = 3'b001;
parameter S2 = 3'b010;
parameter S3 = 3'b011;
parameter S4 = 3'b100;
parameter S7 = 3'b101;
parameter S10 = 3'b110;
parameter Sinv = 3'b111; // invalid state

reg Out;
reg [3:1] state ;

always @(posedge Clk) begin
      Out = 0;
      if (Reset)
            state = S0;
      else case (state)
            S0:
                  if (In) state = S2;
                  else state = S0;
            S1:
                  if (In) state = S4;
                  else state = S3;
            S2:
                  if (In) state = S3;
                  else state = S4;
            S3:
                  state = S7;
            S4:
                  if (In) state = S10;
                  else state = S7;
            S7:
                  state = S0;
            S10:
                  begin
                        state = S0;
                        if (!In) Out = 1;
                  end
            Sinv: // Does the same as reset
                  state = S0;
      endcase

end

endmodule
```

Exercise 8.21

Note that there is an error in the diagram for figure 8.13, state S_3 has two outputs marked 11 and is missing an output for 01. This solution assumes that the arc from S_3 to S_0 should be 01 instead of 11.

```
module prob8_21 ( Out ,In ,Clk);

input [2:0] In;
wire [2:0] In ;
input Clk;
wire Clk;

output Out ;
wire Out ;

parameter I0 = 2'b00;
parameter I1 = 2'b01;
parameter I2 = 2'b10;
parameter I3 = 2'b11;

parameter S0 = 3'b000;
parameter S1 = 3'b001;
parameter S2 = 3'b010;
parameter S3 = 3'b011;
parameter S4 = 3'b100;
parameter S5 = 3'b101;
parameter S6 = 3'b110;
parameter S7 = 3'b111;

reg [3:1] state;
assign Out = !state[1]; // the last state bit determines if the
               // state is even or odd, even states
                           // are the only ones that output a 1

always @(posedge Clk) begin
      case (state)
            S0:
                  case (In)
                        I1: state = S1;
                        I2:   state = S2;
                        I3:   state = S3;
                  endcase
            S1:
                  case (In)
                        I0:   state = S0;
                        I1:   state = S3;
                        I3:   state = S5;
                  endcase
            S2:
                  case (In)
                        I0: state = S1;
                        I1:   state = S3;
                        I3:   state = S4;
                  endcase
```

```
        S3:
            case (In)
                I0:   state = S1;
                I1:   state = S0;
                I2:   state = S4;
                I3:   state = S5;
            endcase
        S4:
            case (In)
                I0:   state = S0;
                I1:   state = S1;
                I2:   state = S2;
                I3:   state = S5;
            endcase
        S5:
            case (In)
                I0:   state = S1;
                I1:   state = S4;
                I2:   state = S0;
            endcase
        S6:
            state = S0;
        S7:
            state = S0;
    endcase
end

endmodule
```

Exercise 8.22

This solution divides the partition work into three modules. The prob8_22 module acts as a connecting block between the two partitions. Partition A handles the left side of Figure 8.38 and Partition B handles the right side.

```
module prob8_22 ( Reset ,Clk ,U ,D );
input Reset, Clk, U, D;
wire Reset, Clk, U, D, S0, S2, S3, S5;
PartitionA partA( Reset, Clk, U, D, S3, S5, S0, S2);
PartitionB partB( Reset, Clk, U, D, S0, S2, S3, S5);
endmodule

module partA ( Reset ,Clk ,U ,D ,S3 ,S5 ,S0 ,S2);
      input Reset, Clk, U, D, S5, S3;
      output S0, S2;
      wire Clk, U, D, S5, S3, S0, S2;

      parameter state0 = 2'b00;
      parameter state1 = 2'b01;
      parameter state2 = 2'b10;
      parameter stateA = 2'b11;

      reg [2:1] state;

      assign S0 = !state[1] && !state[2];
      assign S2 = state[1] && state[2];

      always @(posedge Clk) begin
            if (Reset) state = state0;
            else case (state)
                  state0:
                        begin
                              if (U) state = state1;
                              if (D) state = stateA;
                        end
                  state1:
                        begin
                              if (U) state = state2;
                              if (D) state = state0;
                        end
                  state2:
                        begin
                              if (U) state = stateA;
                              if (D) state = state1;
                        end
                  stateA:
                        begin
                              if (U & S5) state = state0;
                              if (D & S3) state = state2;
                        end
            endcase
      end
endmodule
```

```
module partB ( Reset ,Clk ,U ,D ,S0 ,S2 ,S3 ,S5 );
      input Reset, Clk, U, D, S0, S2;
      output S3, S5;
      wire Reset, Clk, U, D, S0, S2;

      parameter state3 = 3'b00;
      parameter state4 = 3'b01;
      parameter state5 = 3'b10;
      parameter stateB = 3'b11;

      reg [2:1] state;

      assign S3 = !state[1] & !state[2];
      assign S5 = state[1] & state[2];

      always @(posedge Clk) begin
            if (Reset) state = stateB;
            else case (state)
                  state3:
                        begin
                              if (U) state = state4;
                              if (D) state = stateB;
                        end
                  state4:
                        begin
                              if (U) state = state5;
                              if (D) state = state3;
                        end
                  state5:
                        begin
                              if (U) state = stateB;
                              if (D) state = state4;
                        end
                  stateB:
                        begin
                              if (U & S2) state = state3;
                              if (D & S0) state = state5;
                        end
            endcase
      end
endmodule
```

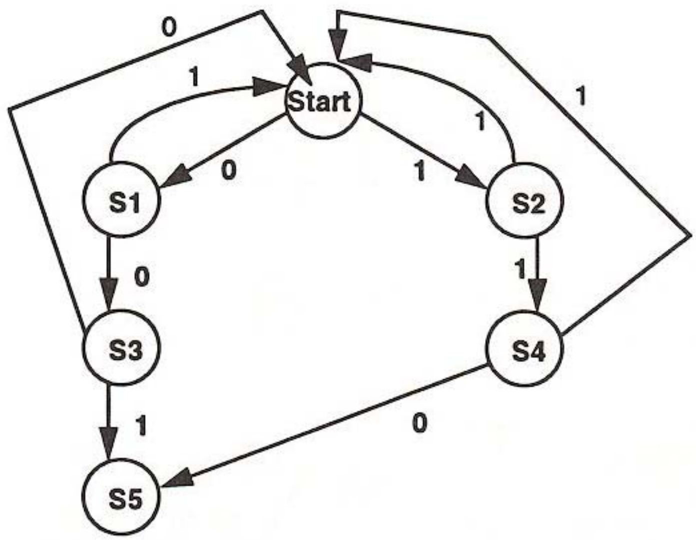
Exercise 8.23

It is possible to rectify this problem without adding any additional states to either state machine. The trick is to have every state between T_{04} and T_{19} output TS. This way if the timer gets past T_{04} before the other state machine can react, it will eventually catch the TS signal from a later state. This does not harm the other state machine, because once the other state machine recognizes TS, the controller is in the FG or HG states and does not care about TS again until it transitions to FY or HY and resets the timer to T_{00} with ST.

Exercise 8.24

0 is asserted in states Start and S1 to S4. 1 is asserted in S5. Various state assignments may be used, including

State	Coding
Start	000
S1	001
S2	010
S3	011
S4	110
S5	111



Exercise 8.25

This solution uses T flip-flops. For an implementation using D flip-flops, simply use K-maps for Q1+ and Q0+ directly.

a.) State table

Q1 Q0	I1 I0	Q1+ Q0+	T1 T0
00	0 0	0 1	0 1
	0 1	1 0	1 0
	1 0	1 1	1 1
	1 1	0 0	0 0
0 1	0 0	1 1	1 0
	0 1	0 0	0 1
	1 0	1 0	1 1
	1 1	0 1	0 0
1 0	0 0	0 0	1 0
	0 1	1 1	0 1
	1 0	0 1	1 1
	1 1	1 0	0 0
1 1	0 0	1 0	0 1
	0 1	0 1	1 0
	1 0	0 0	1 1
	1 1	1 1	0 0

b.) Inputs to T1 and T0

$$T1 = \overline{Q}_1\overline{Q}_0\overline{I}_1 I_0 + I_1\overline{I}_0 + \overline{Q}_1 Q_0\overline{I}_0 + Q_1 Q_0\overline{I}_1 I_0 + Q_1\overline{Q}_0\overline{I}_0$$

$$T2 = \overline{Q}_1 Q_0\overline{I}_1 I_0 + I_1\overline{I}_0 + Q_1 Q_0\overline{I}_0 + Q_1\overline{Q}_0\overline{I}_1 I_0 + \overline{Q}_1\overline{Q}_0\overline{I}_0$$

T_1

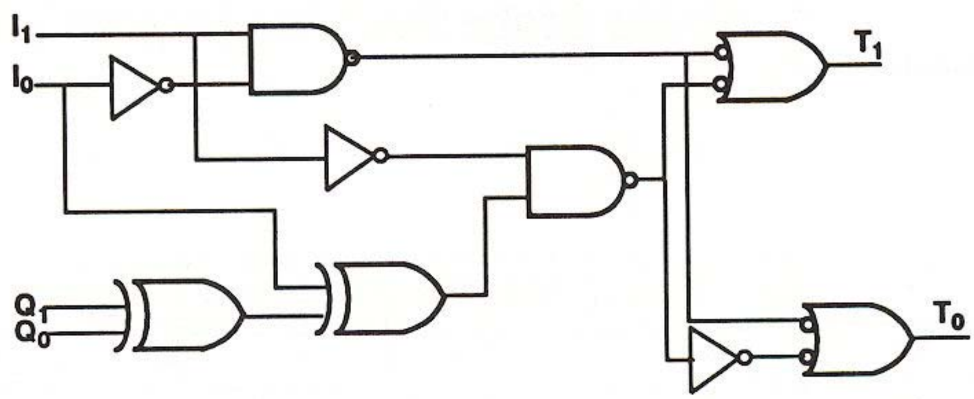
I_1I_0 \ Q_1Q_0	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	0	0	0
10	1	1	1	1

T_0

I_1I_0 \ Q_1Q_0	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	0	0	0	0
10	1	1	1	1

c.) Schematic implementation

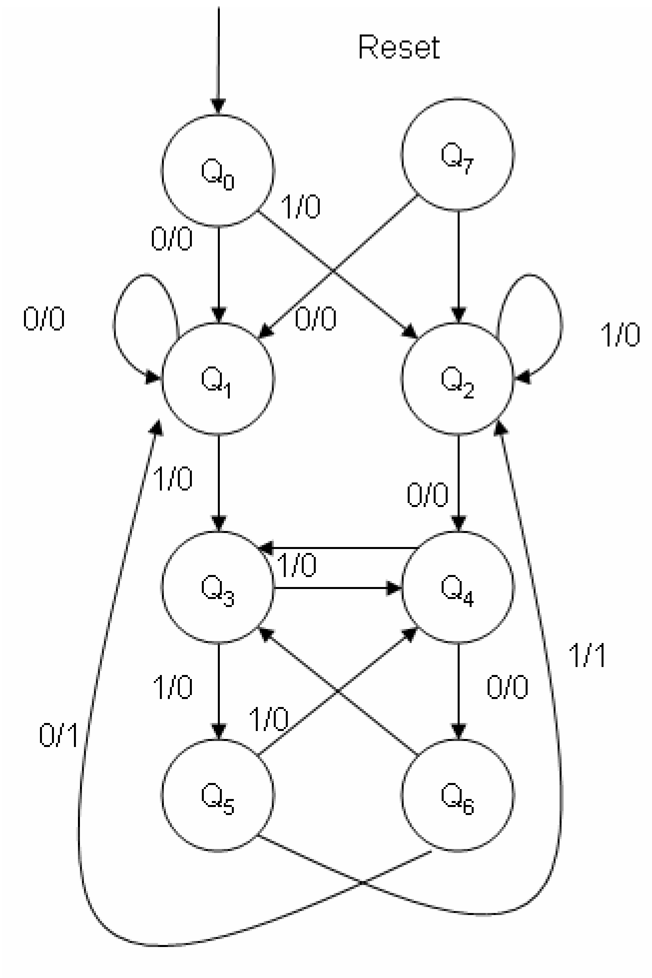
$$T_1 = I_1\bar{I}_0 + (Q_1 \oplus Q_0 \oplus I_0)\bar{I}_1$$
$$T_0 = I_1\bar{I}_0 + (Q_1 \oplus Q_0 \oplus \bar{I}_0)\bar{I}_1$$



Other schematics may be possible.

Exercise 8.26

(a) The state diagram below will implement the function:



(b) The next-state function looks like

Input	Current State	Next State	Output
0	Q_0	Q_1	0
1	Q_0	Q_2	0
0	Q_1	Q_1	0
1	Q_1	Q_3	0
0	Q_2	Q_4	0
1	Q_2	Q_2	0
0	Q_3	Q_4	0
1	Q_3	Q_5	0
0	Q_4	Q_6	0
1	Q_4	Q_3	0
0	Q_5	Q_1	0
1	Q_5	Q_4	1
0	Q_6	Q_1	1
1	Q_6	Q_3	0
0	Q_7	Q_1	0
1	Q_7	Q_2	0

(c) By row-matching, state seven can be left out of the minimized state diagram.

(d) The table below shows the encoding assignment:

State	Encoding
Q_0	010
Q_1	110
Q_2	000
Q_3	011
Q_4	101
Q_5	001
Q_6	100

Q_0 = Reset
$Q_1 = I'\,(Q_0 + Q_1 + Q_6)$
$Q_2 = I\,(Q_0 + Q_2 + Q_5)$
$Q_3 = I\,(Q_1 + Q_4 + Q_6)$
$Q_4 = I'\,(Q_2 + Q_3 + Q_5)$
$Q_5 = IQ_3$
$Q_6 = I'Q_4$

Suppose the Encoding bits are E_0, E_1, E_2 where 0 is the high order bit. Using the state equivalent of a K-map, the equations for each state become:

Q_0 = Reset
$Q_1 = I'E_2'(E_0 + E_1)$
$Q_2 = I\,E_0'(E_1' + E_0')$
$Q_3 = I\,E_0$
$Q_4 = I'E_0'(E_1' + E_2)$
$Q_5 = I\,E_0'E_1E_2$
$Q_6 = I'E_0E_1'E_2$
Output $= I\,E_0'E_1'E_2 + I'\,E_0E_1'E_2'$

E_2 \ E_0E_1	00	01	11	10
0	Q_2	Q_0	Q_1	Q_6
1	Q_5	Q_3	x	Q_4

This uses the high-medium-low priority heuristic fairly well.

For the highest priority, on I = 0, the states Q_2, Q_3, and Q_5 have the same output, Q_0 and Q_1 have the same output, and each of these are adjacent. On I = 1, the states Q_1, Q_4, and Q_6 have the same output, and Q_0 and Q_2 have the same output, and each of these is adjacent. Thus for all of the states, the high-priority condition is met.

For medium priority, $Q_3 \rightarrow Q_5$ or Q_4 and both of these are adjacent.

For low priority, on input 0, Q_0, Q_2, Q_3, and Q_5 all output 0, and on input 1, Q_1, Q_6, Q_4 and x all output 0, meeting the low priority conditions.

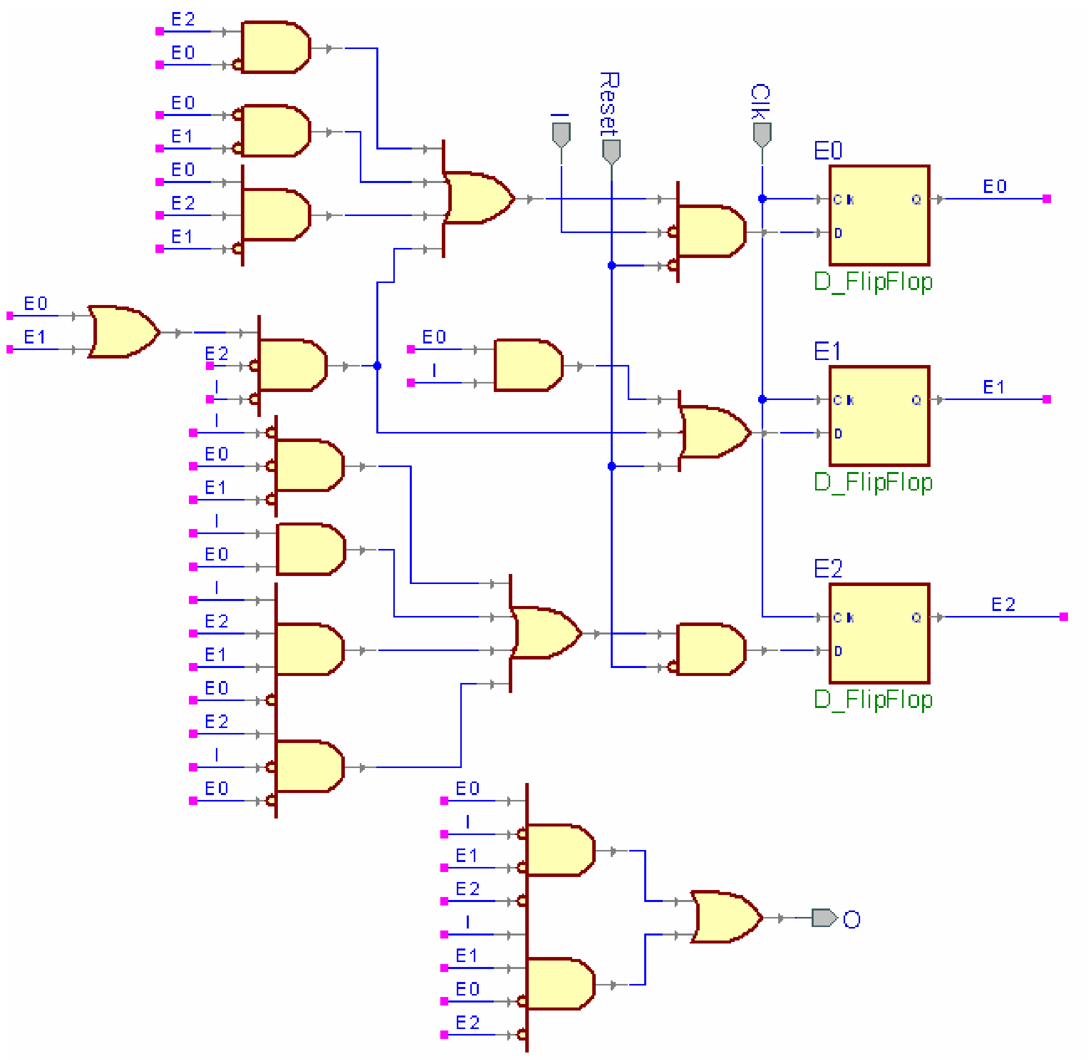
(e) The equations are:

$E_0^* = \text{Reset}'\,I'\,[\,E_0E_1'E_2 + E_0'E_1' + E_0'E_2 + E_2'(E_0 + E_1)\,]$

$E_1^* = \text{Reset} + I'E_2'(E_0 + E_1) + I\,E_0$

$E_2^* = \text{Reset}'\,[\,I\,E_0'E_1E_2 + I\,E_0 + I'E_0'E_1' + I'E_0'E_2\,]$

Output $= I\,E_0'E_1'E_2 + I'\,E_0E_1'E_2'$



(f) Here is the coded implementation:

```
module prob8_26f ( I ,Clk ,Reset ,O );

input I, Clk, Reset;
wire I, Clk, Reset;

output O ;
reg O ;

parameter Q0 = 3'b010;
parameter Q1 = 3'b110;
parameter Q2 = 3'b000;
parameter Q3 = 3'b011;
parameter Q4 = 3'b101;
parameter Q5 = 3'b001;
parameter Q6 = 3'b100;

reg [3:1] state;
```

```
always @(posedge Clk) begin
   O = (!I & (state == Q5)) | (I & (state == Q6));
   case (state)
         Q0:
               begin
                     if (I) state = Q2;
                     else state = Q1;
               end
         Q1:
               begin
                     if (I) state = Q3;
                     else state = Q1;
               end
         Q2:
               begin
                     if (I) state = Q2;
                     else state = Q4;
               end
         Q3:
               begin
                     if (I) state = Q5;
                     else state = Q4;
               end
         Q4:
               begin
                     if (I) state = Q3;
                     else state = Q6;
               end
         Q5:
               begin
                     if (I) state = Q2;
                     else state = Q4;
               end
         Q6:
               begin
                     if (I) state = Q3;
                     else state = Q1;
               end
   endcase

end

endmodule
```