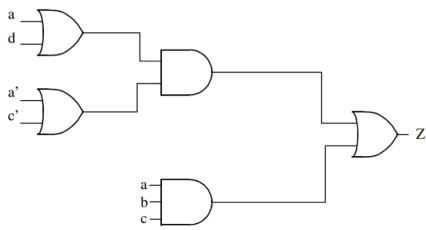


1.

1.1

For the network shown below, find any/all static 1-hazards. For any 1-hazard found specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and the variable which is changing)



$$(a+d)(a'+c') + (abc)$$

cd	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	1	1	1	0
10	0	0	1	0

$$\text{i) } 0101 \leftrightarrow 110$$

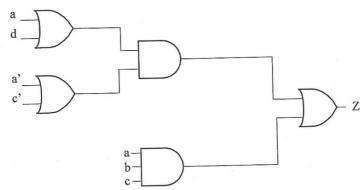
$$\text{ii) } 0111 \leftrightarrow 1111$$

$$\text{iii) } 1001 \leftrightarrow 0001$$

2.

1.2

For the network shown below, find any/all static 0-hazards. For any 0-hazard found specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and the variable which is changing) [10 points].



$$(a+d)(a'+c') + (abc)$$

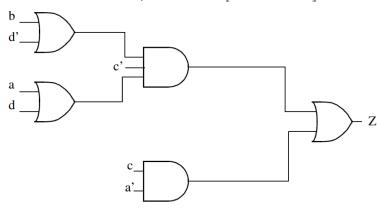
cd	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	1	1	1	0
10	0	0	1	0

$$\begin{array}{l|l} A = 0 & \leftrightarrow 1 \\ B = 0 & 0 \\ C = 1 & 1 \\ D = 1 & 1 \end{array}$$

4.

1.4

For the network shown below, find all static 0-hazards. For each 0-hazard found specify the conditions which will cause the hazard to appear at the output (i.e. specify the logic values of the variables which are constant and clearly specify the variable that is assumed to be changing). If there are no 0-hazards found, use a K' map to show why this is the case.



$$(b+d')(a+d)(c') + (ca')$$

cd	00	01	11	10
00	0	0	1	1
01	0	1	1	0
11	1	1	0	0
10	1	1	0	0

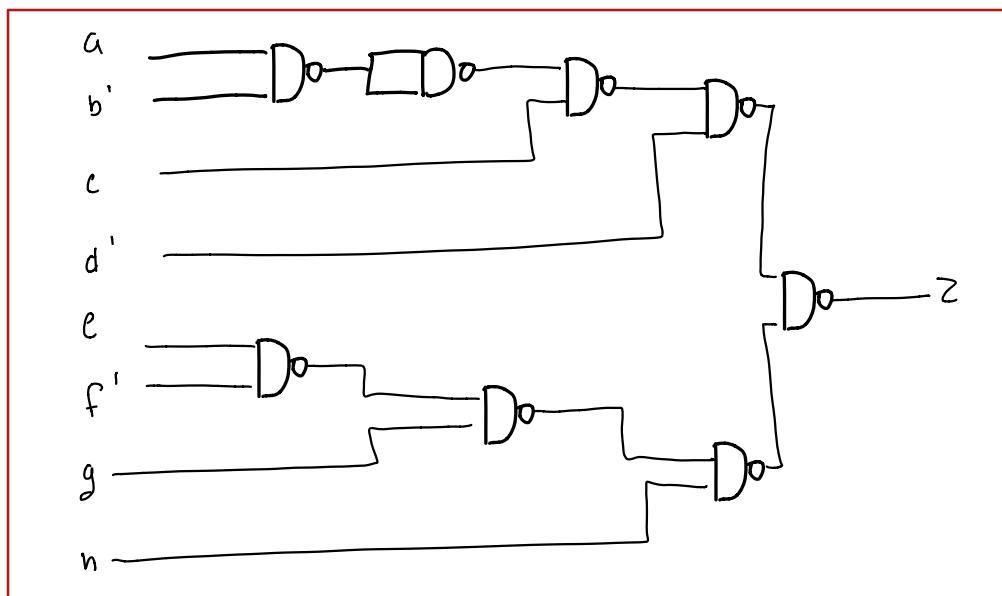
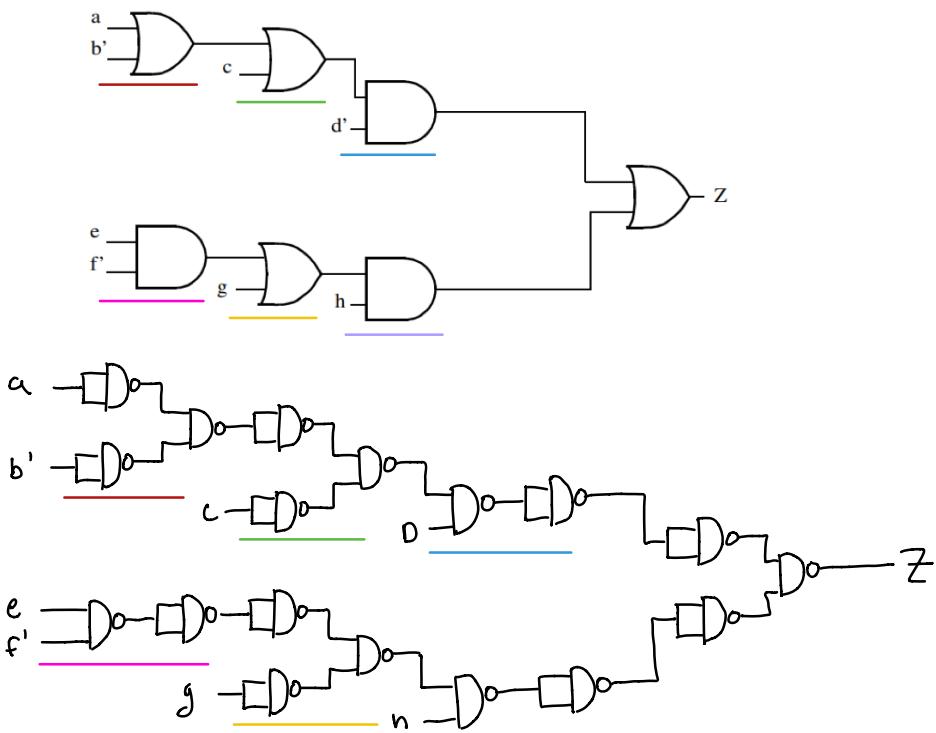
$$\begin{array}{l|l} A = 0 & 0 \\ B = 0 & 0 \\ C = 0 & 0 \\ D = 0 & 1 \end{array} \leftrightarrow \begin{array}{l} 0 \\ 0 \\ 0 \\ 1 \end{array}$$

$$\begin{array}{l|l} A = 1 & 1 \\ B = 0 & 0 \\ C = 0 & 1 \\ D = 1 & 1 \end{array} \leftrightarrow \begin{array}{l} 1 \\ 0 \\ 1 \\ 1 \end{array}$$

Nolan Anderson  
CPE 322  
Midterm Corrections

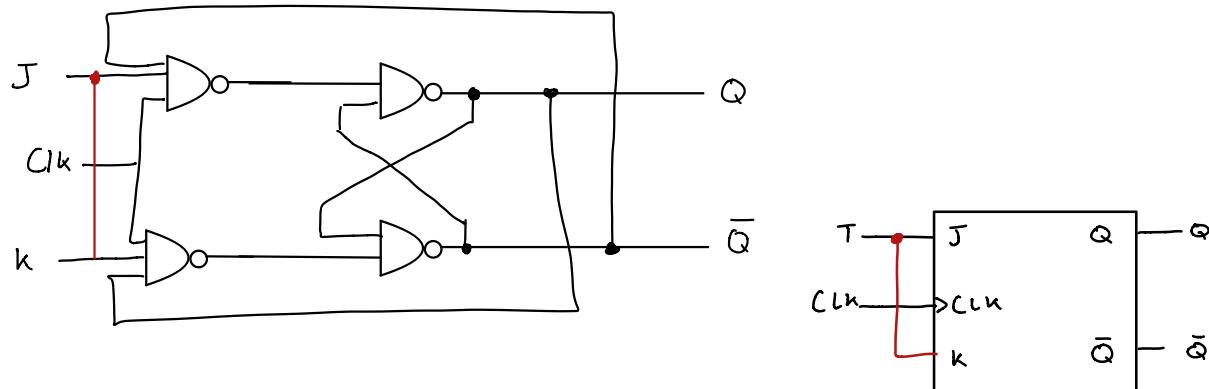
### 3.1

Draw an equivalent multi-level network that is composed entirely of multiple two-input NAND gates. Assume that all input variables (a through h) are available in their true and complemented form.



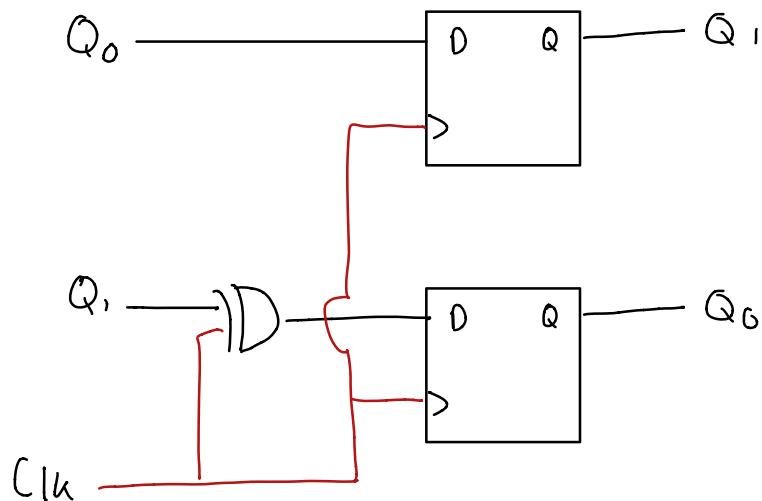
3.2

Use a single rising edge triggered JK Flip-Flop with no additional components (other than connecting wires) to create a rising edge triggered T flip flop (see Fig. 1-13 of the textbook for an example).



3.3

A 2-bit Gray Code accumulator counts  $00 \Rightarrow 01 \Rightarrow 11 \Rightarrow 10 \Rightarrow 00$  and continues repeatedly. Using 2 rising-edge triggered DQ Flip Flops and a single 2-input XOR gate, draw a circuit that continually generates this pattern.



1. Write a short Verilog description of a transparent D Latch with D and G inputs and Q output.

4.1

**module DLATCH (input D, G, output Q)**

```

1 module DLATCH(input D, G, output Q)
2 input D, G;           // D is data, G is clock
3 output Q;            // output is Q
4 always @ (D or G)
5   if(G)
6     |   Q <= D;
7   end
8   else
9     |   Q <= Q;
10  end
11 end
12 endmodule

```

2. Write a short Verilog description of a rising-edge triggered T Flip-Flop (see Fig. 1-13 on pp. 16 of the textbook for definition), with both Q and QN outputs (QN is the inverse of Q). In addition to its normal T and CLK inputs, add an active-low asynchronous SETN input that sets Q high when SETN is low (regardless of clock edge).

4.2

**module TFF (input T, CLK, SETN, output Q, QN)**

```

1 module TFF(input T, CLK, SETN, output Q, QN)
2 always @ (posedge CLK)
3   if(!SETN)
4     |   Q <= 1;          // SETN input that sets Q high when SETN is low.
5   else
6     |   if(T)
7       |     |   QN <= !Q;    // QN gets the inverse of Q
8     |   else
9       |     |   Q <= Q;    // If Q is low it gets Q
10    end
11 end
12 endmodule

```

3. Write a short Verilog description of a falling-edge triggered JK Flip-Flop, with J and K inputs, and an active-high asynchronous clear input CLR that sets the output Q immediately to 0.

4.3

**module JKFF (input J, K, CLK, CLR, output Q)**

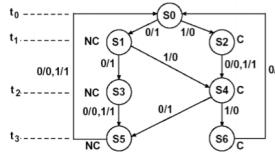
```

1 module JKFF(input J, K, CLK, CLR, output Q)
2 always @ (posedge CLR or negedge CLR)
3 begin
4   if(CLR)
5     |   Q <= 0;
6   else
7     |   if(J & K)
8       |     |   Q <= !Q;
9     |   else if(J & !K)
10      |     |   Q <= 1;
11    else if(!J & K)
12      |     |   Q <= 0;
13    else
14      |     |   Q <= Q;
15   end
16 end
17 end
18 endmodule

```

Problem 5 Retry:

1. Using two always @() procedures, write a Verilog model of the following state graph shown below.  
For state encoding, use 3'd0 for S0, 3'd1 for S1, ... 3'd6 for S6.



5.1

```

1 module fsm_prob5_1 (input X, CLK, output Z)begin
2 reg [2:0] state;
3 reg [2:0] next_state;
4 parameter S0 = 3'd0, S1 = 3'd1, S2 = 3'd2, S3 = 3'd3, S4 = 3'd4, S5 = 3'd5;
5 always @ (X, state)
6 begin
7     case(state)
8         S0:
9             if(X)
10                 Z = 2b'01;
11             else
12                 Z = 2b'00;
13         S1:
14             if(X)
15                 Z = 2b'00;
16             else
17                 Z = 2b'01;
18         end
19         S2:
20             if(X)
21                 Z = 2b'01;
22             else
23                 Z = 2b'00;
24         end
25         S3:
26             if(X)
27                 Z = 2b'01;
28             else
29                 Z = 2b'00;
30         end
31         S4:
32             if(X)
33                 Z = 2b'00;
34             else
35                 Z = 2b'01;
36         end
37         S5:
38             if(X)
39                 Z = 2b'01;
40             else
41                 Z = 2b'00;
42         end
43         S6:
44             if(X == 0)
45                 Z = 2b'01;
46             end
47         default:
48             next_state == S0;
49     endcase
50 end

```

```

53     always @(posedge CLK)
54     begin
55         case(state)
56             S0:
57                 if(X)
58                     next_state == S2;
59                 else
60                     next_state == S1;
61             S1:
62                 if(X)
63                     next_state == S4;
64                 else
65                     next_state == S3;
66             S2:
67                 if(X)
68                     next_state == S4;
69                 else
70                     next_state == S4;
71             S3:
72                 if(X)
73                     next_state == S5;
74                 else
75                     next_state == S5;
76             S4:
77                 if(X)
78                     next_state == S5;
79                 else
80                     next_state == S5;
81             S5:
82                 if(X)
83                     next_state == S6;
84                 else
85                     next_state == S5;
86             S6:
87                 if(X)
88                     next_state == S0;
89                 else
90                     next_state == S0;
91             S0:
92                 if(X == 0)
93                     next_state == S0;
94                 else
95                     next_state == S0;
96             default: next_state == S0;
97         endcase
98     end
99 endmodule

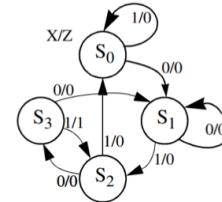
```

5.2

```

1 module fsm_prob5_2(input X, CLK, output Z)begin
2 reg [3:0] state;
3 reg [3:0] next_state;
4 parameter S0 = 2'd0, S1 = 2'd1, S2 = 2'd2, S3 = 2'd3;
5 always @ (X, state)
6 begin
7     case(state)
8         S0:
9             if(X)
10                 Z = 2b'00;
11             else
12                 Z = 2b'00;
13         S1:
14             if(X)
15                 Z = 2b'00;
16             else
17                 Z = 2b'00;
18         S2:
19             if(X)
20                 Z = 2b'00;
21             else
22                 Z = 2b'00;
23         S3:
24             if(X)
25                 Z = 2b'01;
26             else
27                 Z = 2b'00;
28     endcase
29 end
30 always @(posedge CLK)
31 begin
32     case(state)
33         S0:
34             if(X)
35                 next_state == S0;
36             else
37                 next_state == S1;
38         S1:
39             if(X)
40                 next_state == S2;
41             else
42                 next_state == S1;
43         S2:
44             if(X)
45                 next_state == S0;
46             else
47                 next_state == S3;
48         S3:
49             if(X)
50                 next_state == S2;
51             else
52                 next_state == S1;
53     endcase
54 end
55 endmodule

```



2. Using two always @() procedures, write a Verilog model of the following state graph shown below.  
For state encoding, use 2'd0 for S0, 2'd1 for S1, 2'd2 for S2, and 2'd3 for S3.

## 5.3

3. Create a valid Verilog model for a 3-to-8 decoder, with a 3-bit input SEL and an 8-bit output ONEHOT. The 3-bit input SEL enables a 1 on the output bit in the ONEHOT output vector at bit position SEL, with all other bits set to 0.

```
1 module three2eight_dec (input [2:0] SEL, output[7:0]ONEHOT)begin
2     always @(SEL)
3         case (SEL)
4             3'b000 : ONEHOT = 8'b00000001;
5             3'b001 : ONEHOT = 8'b00000010;
6             3'b010 : ONEHOT = 8'b00000100;
7             3'b011 : ONEHOT = 8'b00001000;
8             3'b100 : ONEHOT = 8'b00010000;
9             3'b101 : ONEHOT = 8'b00100000;
10            3'b110 : ONEHOT = 8'b01000000;
11            3'b111 : ONEHOT = 8'b10000000;
12            default : ONEHOT = 8'b00000000;
13        endcase
14    end
15 endmodule
16
```

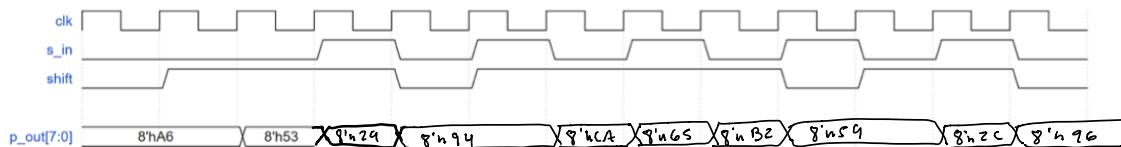
4. Create a valid Verilog 5-bit priority encoder, where the highest bit position on the 5-bit input VEC[4:0] that contains a 1 is identified as the 3-bit output SEL[2:0]. For example, under the input condition VEC[4:0] = 5'b01011, the SEL output would be 3'd3 (since bit position 3 is the highest one containing a 1).

```
1 module prio_enc (input [4:0] VEC, output[2:0]SEL)begin
2     assign SEL =
3         (input[4] == 1'b1) ? 3'b100;
4         (input[3] == 1'b1) ? 3'b011;
5         (input[2] == 1'b1) ? 3'b010;
6         (input[1] == 1'b1) ? 3'b001;
7         (input[0] == 1'b1) ? 3'b000: 3'bxxx;
8     endmodule
9
```

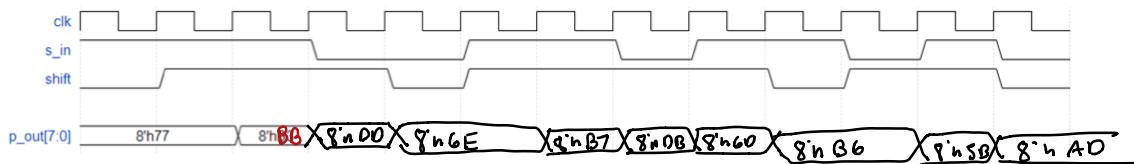
**INSTRUCTIONS:** In this problem set, all state machines are intended to behave as Moore state machines; the **p\_out[7:0]** and **accum[7:0]** outputs must change in value one clock cycle after the inputs change, as shown in the first 3 provided values in the timing diagrams.

6.1

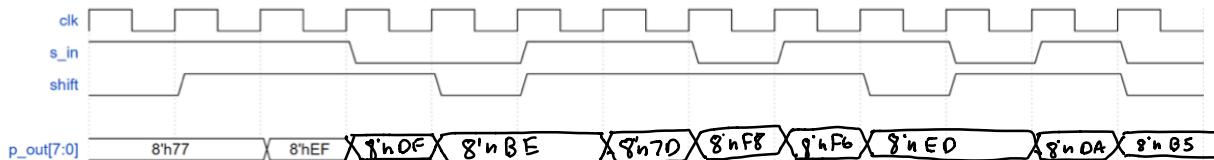
1. Complete the following timing diagram of a serial-in/parallel-out shift register, where serial data shifts in bit-by-bit from the **s\_in** input into the most-significant bit side (bit 7; newest/most-recent) towards the least-significant bit side (bit 0; oldest/least-recent). Data only shifts when the **shift** input is high; when shift is low **p\_out[7:0]** does not change in value. You can either complete the **p\_out[7:0]** signal values in hexadecimal or binary.



2. Complete the following timing diagram of a serial-in/parallel-out shift register, where serial data shifts in bit-by-bit from the **s\_in** input into the most-significant bit side (bit 7; newest/most-recent) towards the least-significant bit side (bit 0; oldest/least-recent). Data only shifts when the **shift** input is high; when shift is low **p\_out[7:0]** does not change in value. You can either complete the **p\_out[7:0]** signal values in hexadecimal or binary. 6.2



3. NOTE: THIS PROBLEM STATEMENT DIFFERS FROM #1 & #2... Complete the following timing diagram of a serial-in/parallel-out shift register, where serial data shifts in bit-by-bit from the **s\_in** input into the **least-significant** bit side (bit 0; newest/most-recent) towards the **most-significant** bit side (bit 0; oldest/least-recent). Data only shifts when the **shift** input is high; when shift is low **p\_out[7:0]** does not change in value. You can either complete the **p\_out[7:0]** signal values in hexadecimal or binary. 6.3



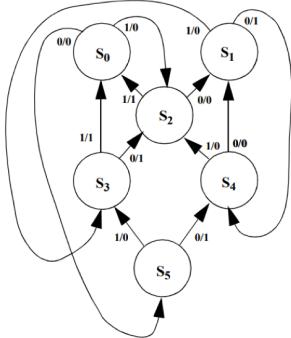
**INSTRUCTIONS: FOR THE FOLLOWING 4 STATE MACHINES, DO THE FOLLOWING:**

- 1) Create an equivalent Algorithmic State Chart representation, using rectangular boxes to show states, diamonds to show input/decision blocks, and ovals to represent conditional outputs. See the Module/Slides for Algorithmic State Machine Representation for examples.
- 2) Write a behavioral Verilog HDL model that implements the state transition graph. In your model include a synchronous active high reset input signal that will always place the design in state S0 on the active edge of the next clock pulse regardless of the current state of the network.

Present State	Next State $X=0$	Output $X=0$	Output $X=1$
S0	S6 S1	S2	0 0
S1	S4 S3	S3	1 0
S2	S1 S0	S0	0 1
S3	S2 S6	S1	1 1
S4	S1 S2	S0	0 0
S5	S4 S3	S3	1 0

A:

7A

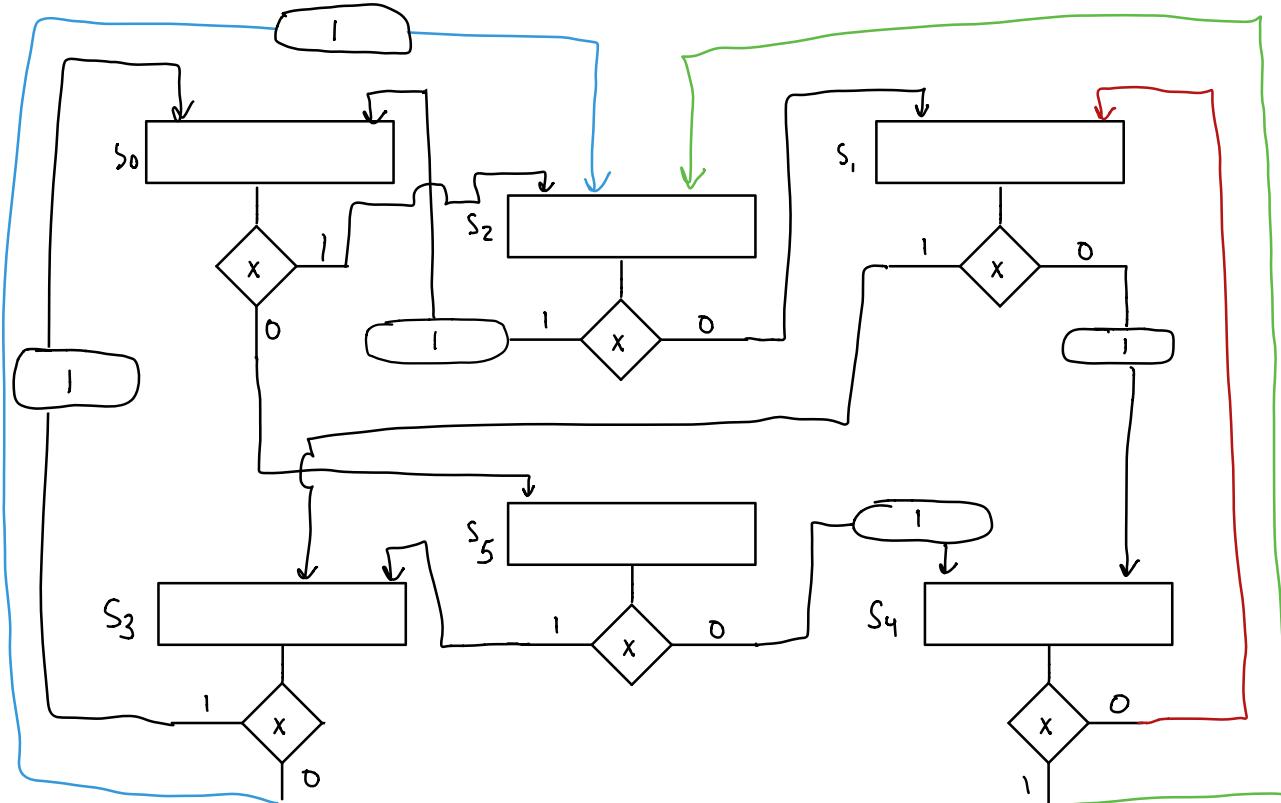


```

1 module SevenA(input data_in, clk, reset, output data_out)
2   input clk;
3   input [1:0] data_in;
4   reg [1:0] state;
5   parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4, S5 = 5;
6   always @(posedge clk or posedge reset)
7     begin
8       if(reset)
9         state <= S0;
10      else
11        case(state)
12          S0:
13            if(data_in)
14              state <= S2;
15            else
16              state <= S5;
17            end
18            S1:
19            if(data_in)
20              state <= S3;
21            else
22              state <= S4;
23            end
24            S2:
25            if(data_in)
26              state <= S0;
27            else
28              state <= S1;
29            end
30            S3:
31            if(data_in)
32              state <= S0;
33            else
34              state <= S2;
35            end
36            S4:
37            if(data_in)
38              state <= S0;
39            else
40              state <= S1;
41            end
42            S5:
43            if(data_in)
44              state <= S3;
45            else
46              state <= S4;
47            end
48          default: state <= S0;
49        endcase
50      end
51    endmodule
  
```

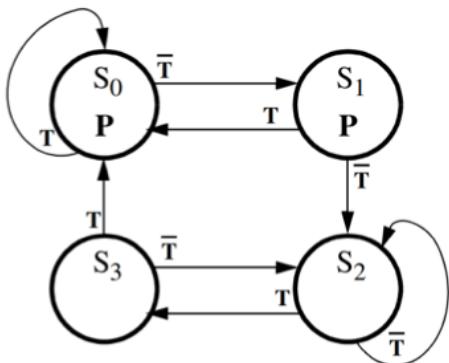
```

1 always @ (state)
2   begin
3     case (state)
4       S0:
5         if (data_in)
6           data_out = 2b'00;
7         else
8           data_out = 2b'00;
9         end
10      S1:
11        if (data_in)
12          data_out = 2b'00;
13        else
14          data_out = 2b'01;
15        end
16      S2:
17        if (data_in)
18          data_out = 2b'01;
19        else
20          data_out = 2b'00;
21        end
22      S3:
23        if (data_in)
24          data_out = 2b'01;
25        else
26          data_out = 2b'00;
27        end
28      S4:
29        if (data_in)
30          data_out = 2b'00;
31        else
32          data_out = 2b'01;
33        end
34      S5:
35        if (data_in)
36          data_out = 2b'00;
37        else
38          data_out = 2b'01;
39        end
40      endcase
41    end
42  endmodule
  
```



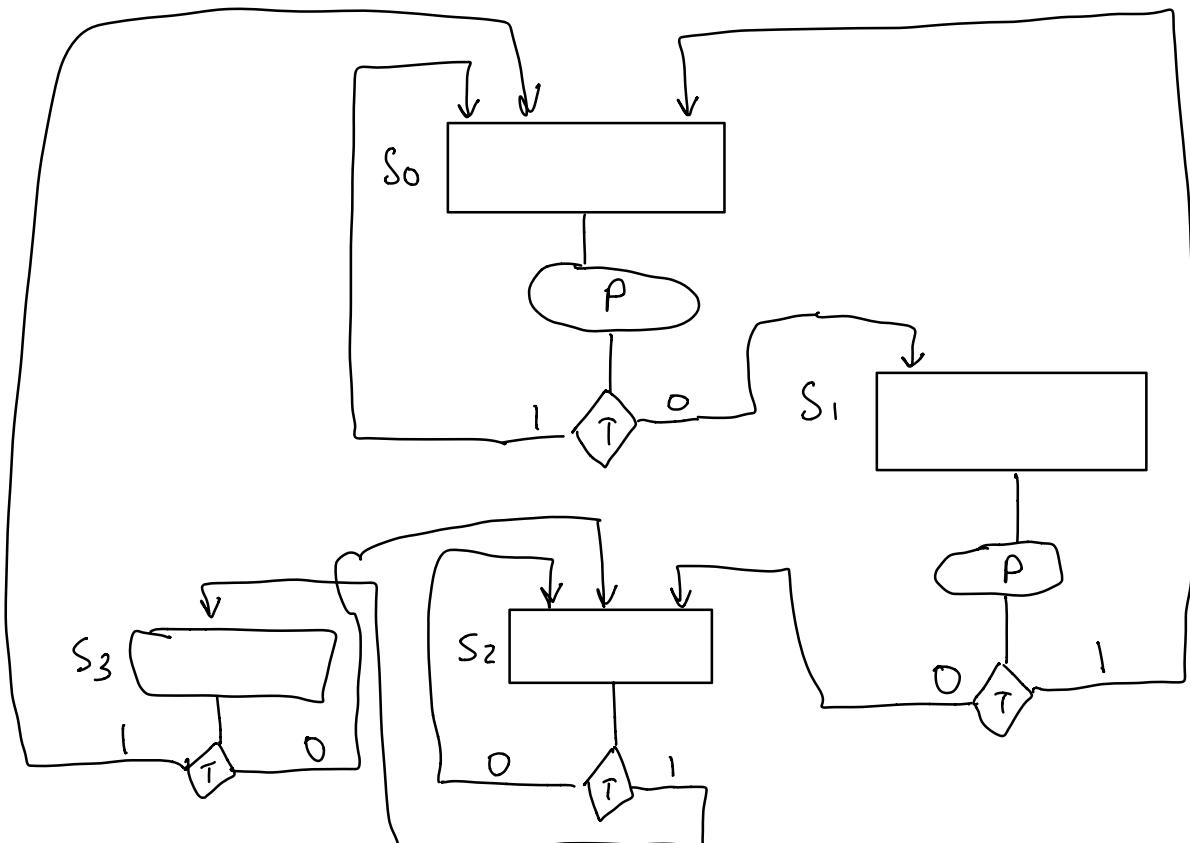
7B

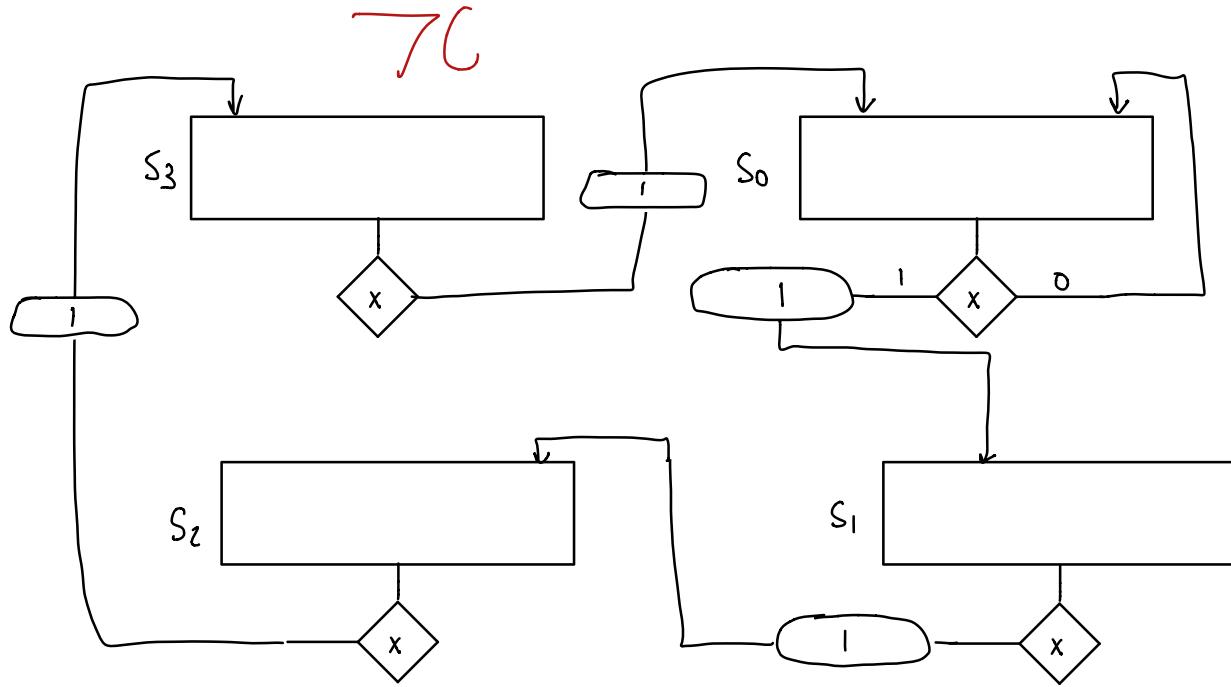
B:



```

1 module SevenB(input T, CLK, output Z)begin
2   reg [1:8] state;
3   parameter S0 = 2'd0, S1 = 2'd1, S2 = 2'd2, S3 = 2'd3;
4
5   always @(posedge CLK)
6   begin
7     case(state)
8       S0:
9         if(T) begin
10           state = S0;
11           Z = 2'b01;
12         end
13       S1:
14         if(T) begin
15           state = S1;
16           Z = 2'b'01;
17         end
18       S2:
19         if(T) begin
20           state = S2;
21           Z = 2'b00;
22         end
23       S3:
24         if(T) begin
25           state = S3;
26           Z = 2'b00;
27         end
28       else
29         state = S2;
30         Z = 2'b'00;
31     end
32   endcase
33 end
  
```

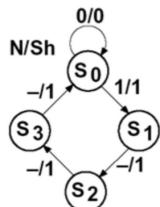




```

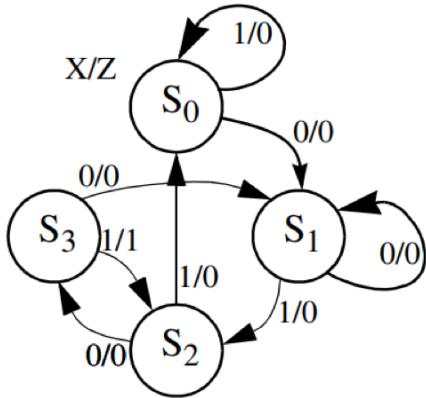
C:
1 module SevenC(input data_in, clk, reset, output data_out)
2 input clk, data_in, reset;
3 output reg [1:0] data_out;
4 reg [1:0] state;
5 parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3;
6
7 always @(posedge clk or posedge reset)
8 begin
9     if(reset)
10         state <= S0;
11     else
12         case(state)
13             S0:
14                 if(data_in)
15                     state <= S1;
16                 else
17                     state <= S0;
18             end
19             S1:
20                 if(data_in || !data_in)
21                     state <= S2;
22             end
23             S2:
24                 if(data_in || !data_in)
25                     state <= S3;
26             end
27             S3:
28                 if(data_in || !data_in)
29                     state <= S0;
30             end
31             default: state <= S0;
32         endcase
33     end
34 end
35 always @ (state)
36 begin
37     case(state)
38         S0:
39             if(data_in)
40                 data_out = 2b'01;
41             else
42                 data_out = 2b'00;
43             end
44         S1:
45             if(data_in || !data_in)
46                 data_out = 2b'01;
47             end
48         S2:
49             if(data_in || !data_in)
50                 data_out = 2b'01;
51             end
52         S3:
53             if(data_in || !data_in)
54                 data_out = 2b'01;
55             end
56         endcase
57     end
58 endmodule

```



7D

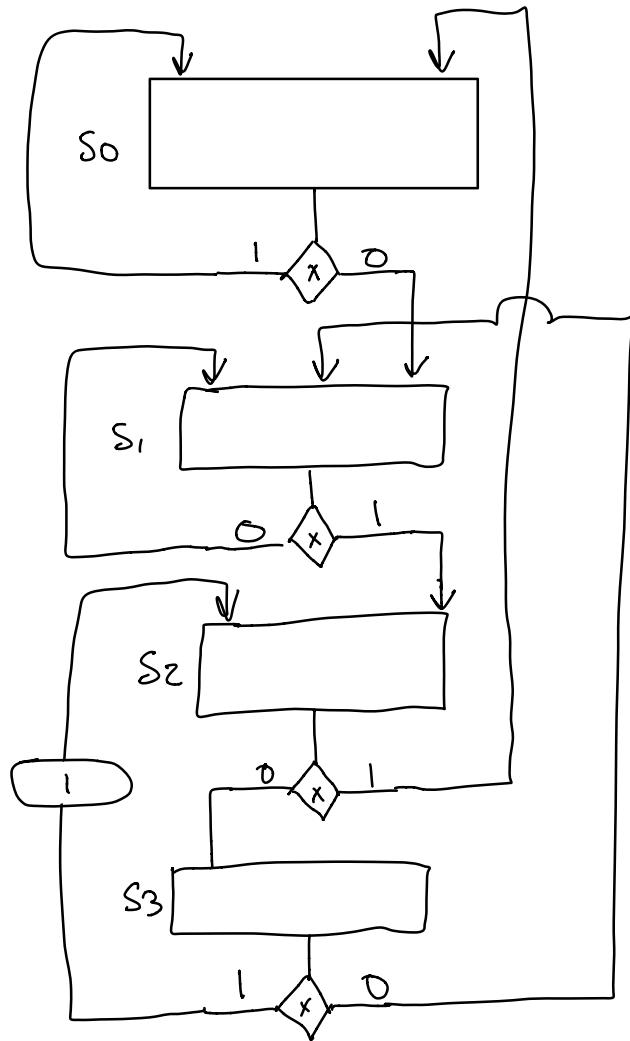
D:



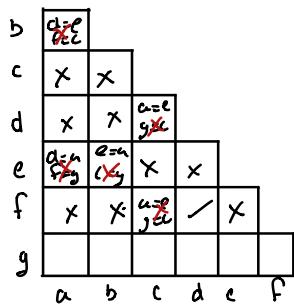
```

1 module SevenD(input X, CLK, output Z)begin
2 reg [1:0] state;
3 reg [1:0] next_state;
4 parameter S0 = 2'd0, S1 = 2'd1, S2 = 2'd2, S3 = 2'd3;
5
6 always @ (X, state)
7 begin
8     case(state)
9         S0:
10             if(X)
11                 Z = 2b'00;
12             else
13                 Z = 2b'00;
14             end
15         S1:
16             if(X)
17                 Z = 2b'00;
18             else
19                 Z = 2b'00;
20             end
21         S2:
22             if(X)
23                 Z = 2b'00;
24             else
25                 Z = 2b'00;
26             end
27         S3:
28             if(X)
29                 Z = 2b'01;
30             else
31                 Z = 2b'00;
32             end
33     endcase
34 end
35
36 always @(posedge CLK)
37 begin
38     case(state)
39         S0:
40             if(X)
41                 next_state = S0;
42             else
43                 next_state = S1;
44             end
45         S1:
46             if(X)
47                 next_state = S2;
48             else
49                 next_state = S1;
50             end
51         S2:
52             if(X)
53                 next_state = S0;
54             else
55                 next_state = S3;
56             end
57         S3:
58             if(X)
59                 next_state == S2;
60             else
61                 next_state == S1;
62             end
63     endcase
64 end
65 endmodule

```



present state	next state		present output	
	X=0	1	X=0	1
a	d	f	1	0
b	e	c	1	0
c	a	g	0	1
d	e	c	0	1
e	a	g	1	1
f	e	c	0	1



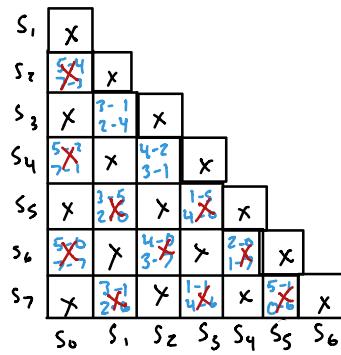
8.1

Present	Next		Present output	
	0	1		
a	d	d	1	0
b	e	v	1	0
c	a	u	0	1
d	e	c	0	1
e	a	a	1	1

2.

8.2

present state	next state		present output
	X=0	1	
A S <sub>0</sub>	S <sub>5</sub>	S <sub>7</sub>	0
B S <sub>1</sub>	S <sub>3</sub>	S <sub>2</sub>	1
C S <sub>2</sub>	S <sub>4</sub>	S <sub>3</sub>	0
D S <sub>3</sub>	S <sub>1</sub>	S <sub>4</sub>	1
E S <sub>4</sub>	S <sub>2</sub>	S <sub>1</sub>	0
F S <sub>5</sub>	S <sub>5</sub>	S <sub>0</sub>	1
G S <sub>6</sub>	S <sub>0</sub>	S <sub>7</sub>	0
H S <sub>7</sub>	S <sub>1</sub>	S <sub>6</sub>	1



Present State	Next State		Present output
S <sub>0</sub>	S <sub>5</sub>	S <sub>7</sub>	0
S <sub>1</sub>	S <sub>1</sub>	S <sub>2</sub>	1
S <sub>2</sub>	S <sub>2</sub>	S <sub>1</sub>	0
S <sub>3</sub>	S <sub>5</sub>	S <sub>0</sub>	1
S <sub>4</sub>	S <sub>0</sub>	S <sub>7</sub>	0
S <sub>5</sub>	S <sub>1</sub>	S <sub>6</sub>	1

$S_3 \rightarrow S_1$

$S_4 \rightarrow S_2$

## 10.2

2. Develop a Verilog model for a 6-to-1 multiplexer where the general inputs represent a 6-bit bus and the output is a single signal. When S[2:0] is 6, choose I[2] as the output, and when S[2:0] is 7, choose I[3] as the output. If using a case statement, please use a default clause; if using an if/else statement, assign O in all clauses including a final "else" clause to avoid creating a latch. Label the inputs I[5:0], S[2:0], and the output O.

```
1 module MUX6T01(input [5:0] I, [2:0] S, output O)
2 reg [5:0] O;
3 always @ (inp or S)
4     case(S)
5         3'b000 : O = inp[0];
6         3'b001 : O = inp[1];
7         3'b010 : O = inp[2];
8         3'b011 : O = inp[3];
9         3'b100 : O = inp[4];
10        3'b101 : O = inp[5];
11        default : O = 8'bx;
12    endcase
13 endmodule
```

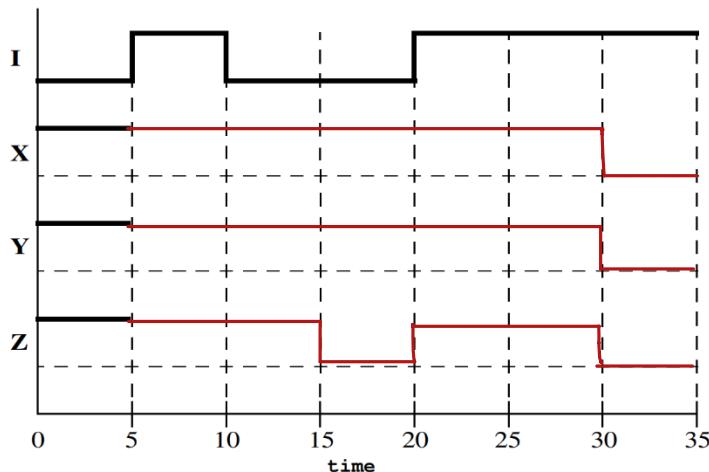
Problem 11 Retry (first part only):

1. 1

1.

Complete the Timing Diagram for the simulation of the following Verilog Model where input I is driven in the manner that is shown on the diagram.

```
module v_model(input I, output X, Y, output reg Z);
    assign #10 X = ~I;
    not #(10) G1 (Y, I);
    always @(I) Z <= #10 ~I;
endmodule;
```



2.

Complete the following timing diagram for the output signal, Z and the internal input signal, state. Assume that a basic functional RTL Simulation is to be performed.

```
module fsm_network (input clk, X, Reset, output reg Z);
    reg state, next_state;

    always @ (state, X)
        case (state)
            0 : if (X) begin Z=1; next_state=1; end
            else begin Z=0; next_state=0; end
            1 : if (X) begin Z=0; next_state=1; end
            else begin Z=1; next_state=0; end
        endcase

    always @ (posedge clk)
        if (Reset) state=0;
        else state=next_state;
endmodule
```

1. 2

State = Next State

