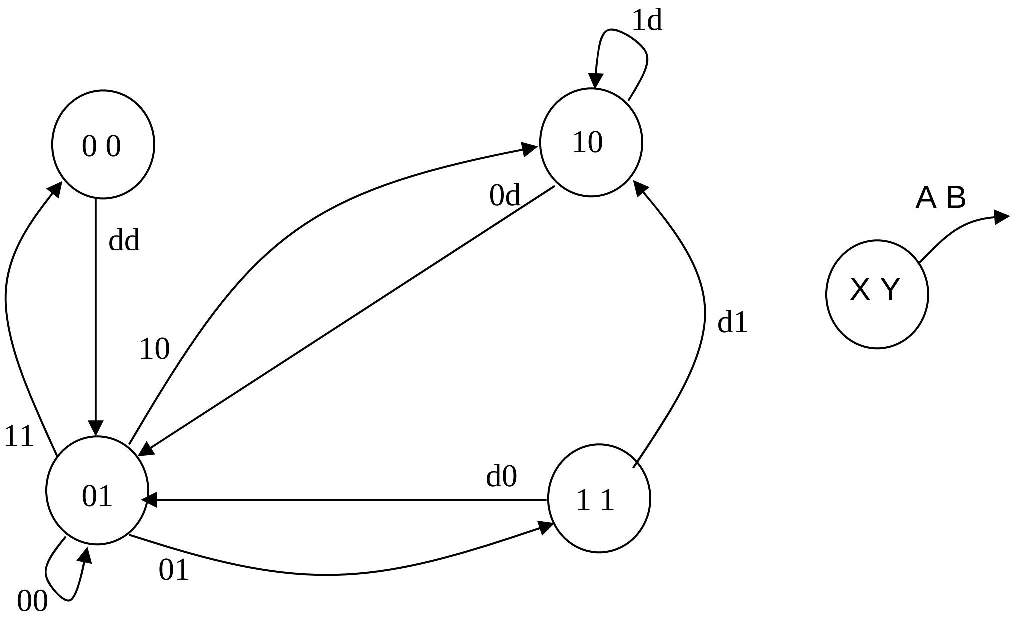
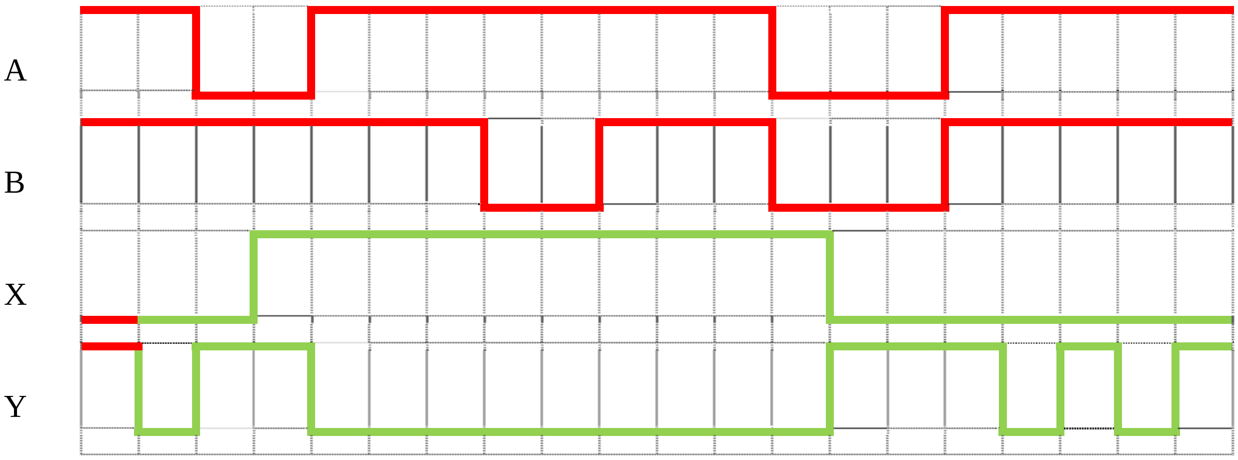
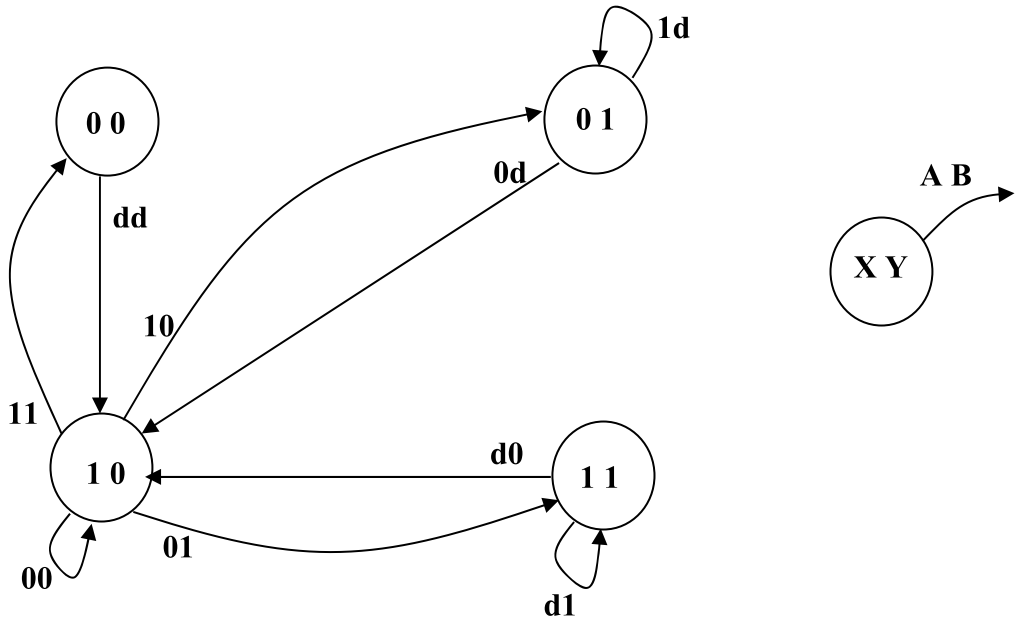
Practice Homework Solution for Module 3

1. Given the following state transition diagram, complete the timing chart below.



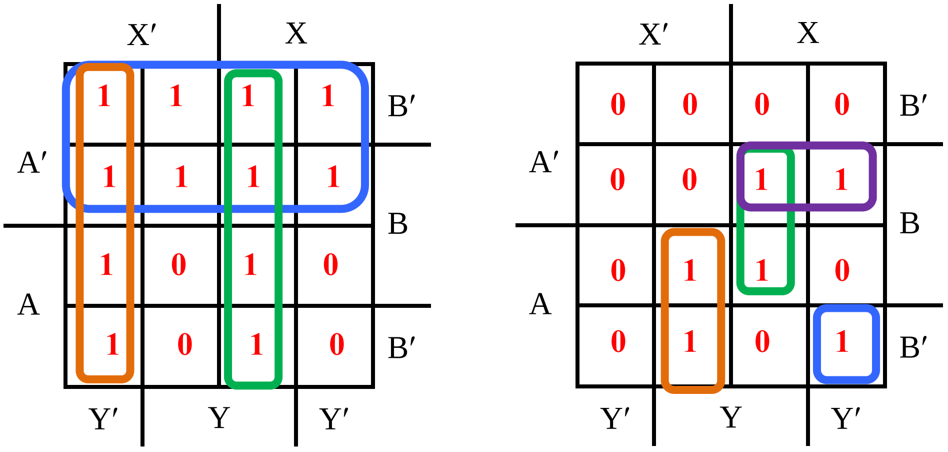


2. Given the following state transition diagram, determine the *next state equations* it represents in *minimum sum-of-products form.*



X	Y	A	B	X*	Y*
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	1	0	0
1	1	0	0	1	0
1	1	0	1	1	1
1	1	1	0	1	0
1	1	1	1	1	1

X* and Y* are "shorthand" for the next state of X and Y



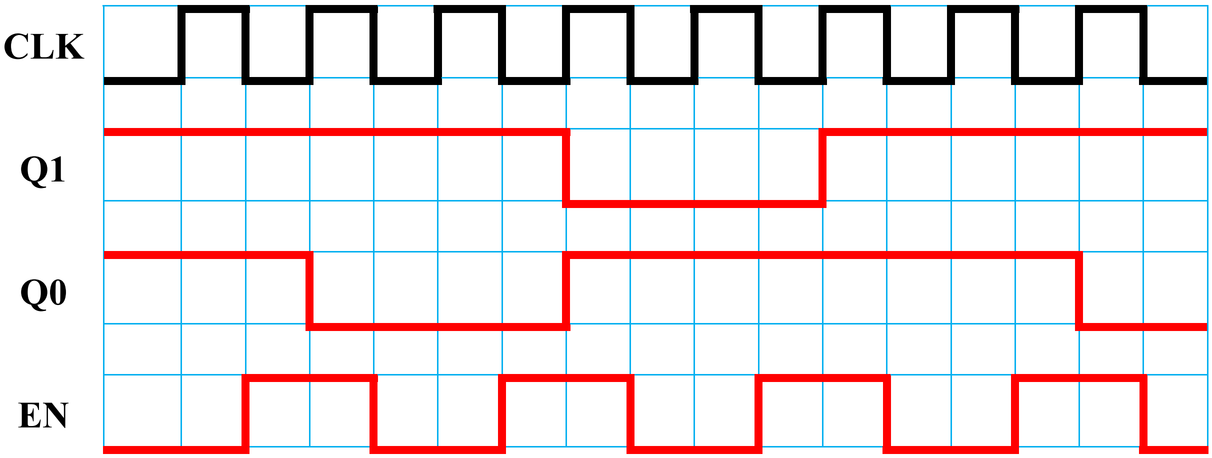
	X′ Y′	X′ Y	X Y	X Y′	
A′	1	1	1	1	B′
A′	1	1	1	1	B
A	1	0	1	0	B
A	1	0	1	0	B′

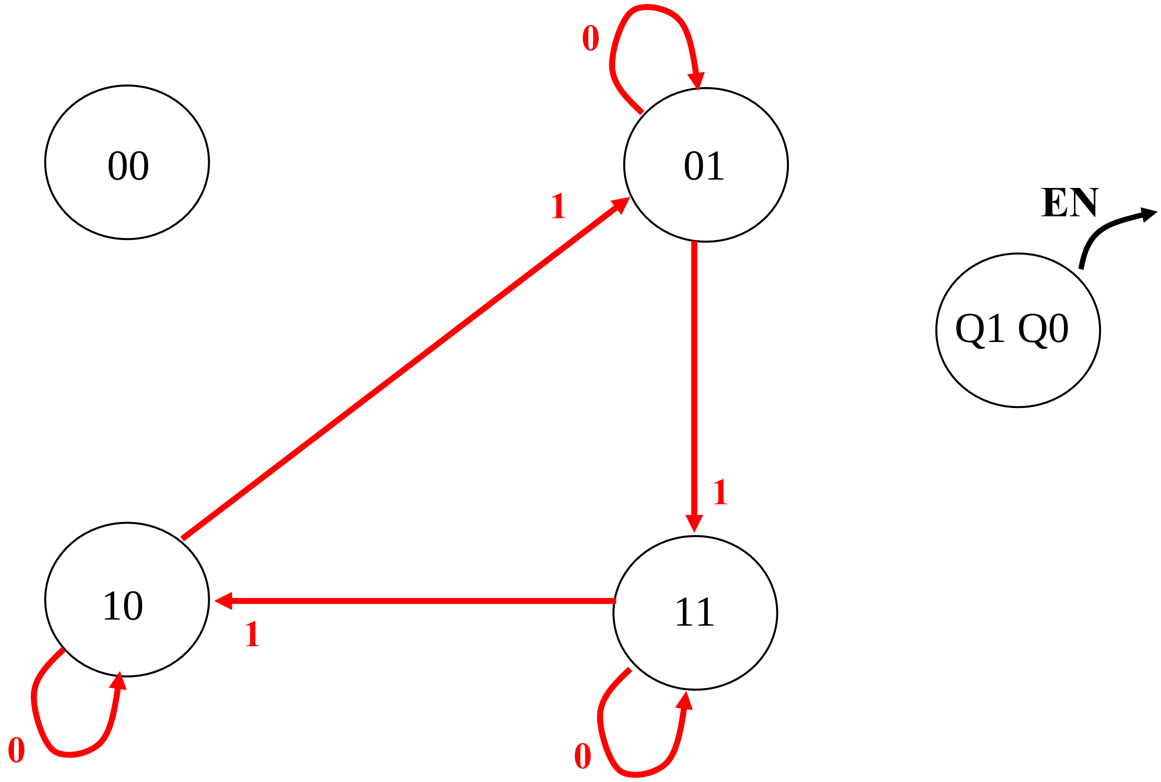
	X′ Y′	X′ Y	X Y	X Y′	
A′	0	0	0	0	B′
A′	0	0	1	1	B
A	0	1	1	0	B
A	0	1	0	1	B′

$X^* = X'\cdot Y' + X\cdot Y + A'$

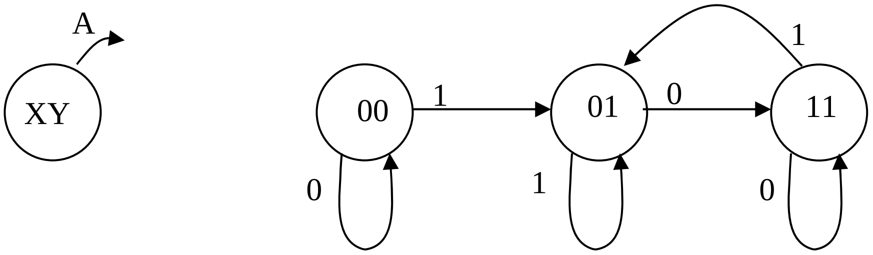
$Y^* = A\cdot X'\cdot Y + B\cdot X\cdot Y + A'\cdot B\cdot X + A\cdot B'\cdot X \cdot Y'$

3. Given the timing diagram, below, for a state machine that has one input (EN) and two state variables (Q1 and Q0), derive a state transition diagram:





4. Given the following state transition diagram, determine:



(a) The next state equation for X if the state machine is designed for **minimum cost**

$$X^* = A' \cdot Y$$

(b) The next state equation for X if the state machine is designed for **minimum risk**

$$X^* = A' \cdot Y$$

(c) The next state equation for Y if the state machine is designed for **minimum cost**

$$Y^* = A + Y$$

(d) The next state equation for Y if the state machine is designed for **minimum risk**

$$Y^* = A \cdot X' + Y$$

5. A “new” type of flip-flop, the RG (“Raul Good”), is described by the following PS-NS table. Derive its next state equation and excitation table.

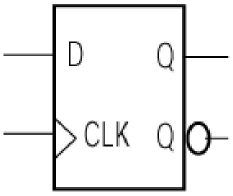
R	G	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

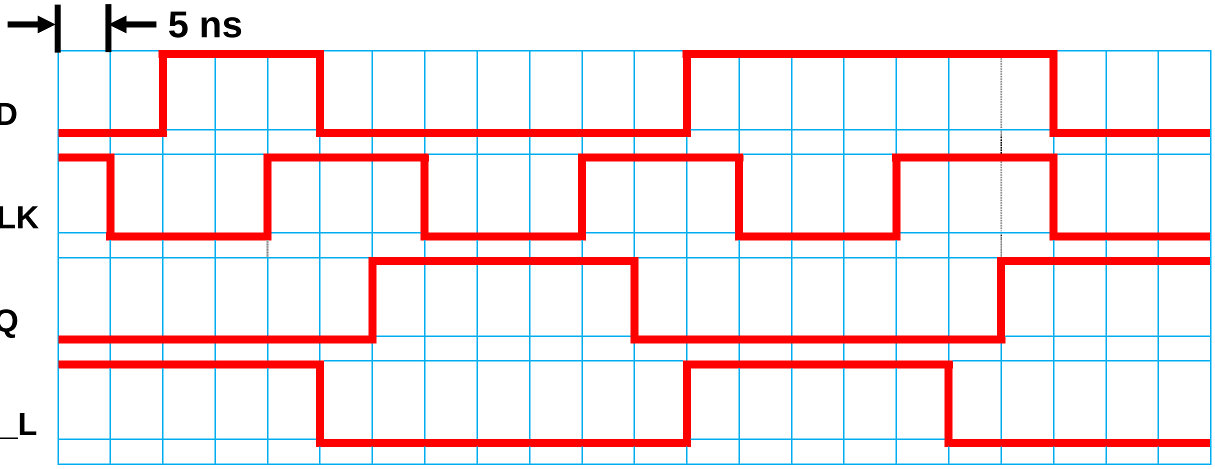
Excitation table:

Q	Q*	R	G
0	0	d	0
0	1	d	1
1	0	1	d
1	1	0	d

$$Q^* = R' \bullet Q + G \bullet Q'$$

6. Given the following timing chart for an edge-triggered D flip-flop, determine the following based on the excitation signals (D and CLK) depicted:





(a) The **nominal setup time** provided for the D flip-flop **10 ns**

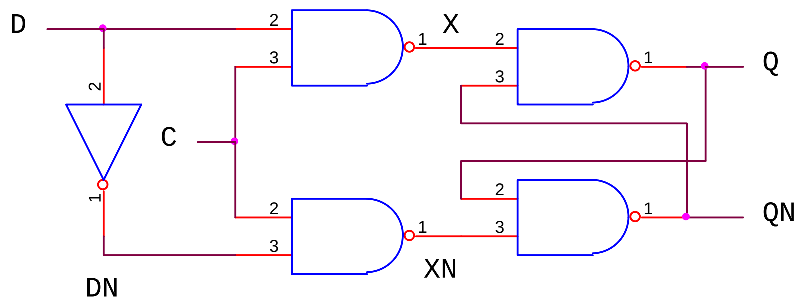
(b) The **nominal hold time** provided for the D flip-flop **5 ns**

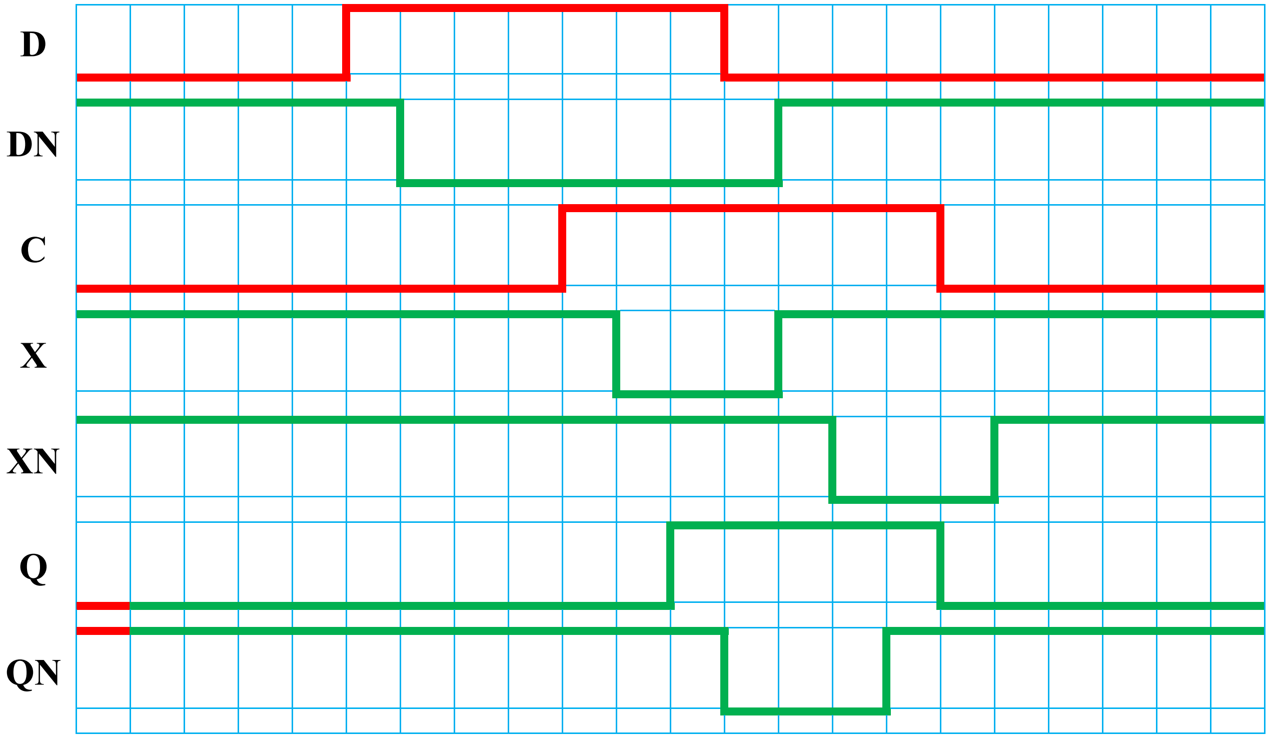
(c) The **nominal clock pulse width** provided for the D flip-flop **15 ns**

(d) The $\mathbf{t}_{\mathbf{PHL(C\rightarrow Q)}}$ of the D flip-flop **5 ns**

(e) The $\mathbf{t}_{\mathbf{PLH(C\rightarrow Q)}}$ of the D flip-flop **10 ns**

7. Complete the timing chart, below, for a D latch, and answer the questions that follow. Assume each gate has 5 ns of delay (t_{PLH} and t_{PHL}), and that each division is 5 ns.





(a) Determine the **minimum time** input C should be asserted (while the D input remains stable) to ensure reliable operation of the latch. **10 ns**

(b) Determine the **nominal setup time** provided for the D latch. **20 ns**

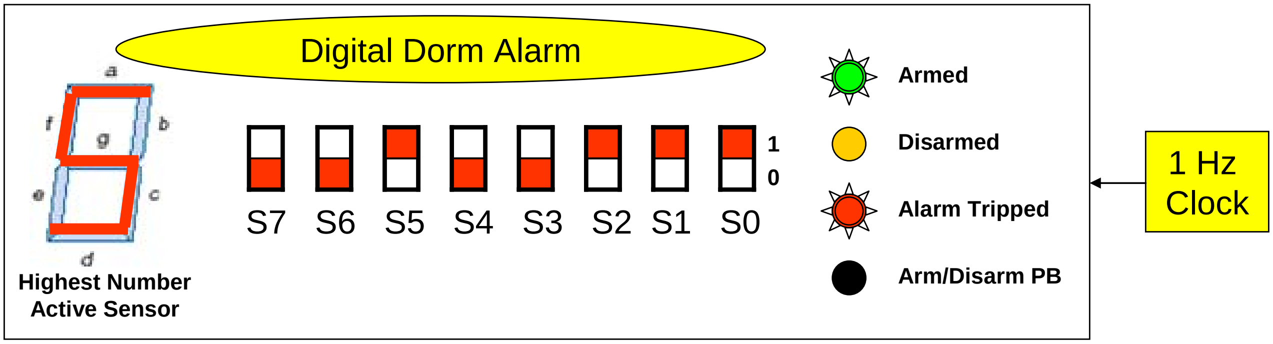
(c) Determine the **nominal hold time** provided for the D latch. **≥ 30 ns**

(d) Determine the $\mathbf{t_{PLH(C \rightarrow Q)}}$ of the D latch. **10 ns**

(e) Determine the $\mathbf{t_{PHL(D \rightarrow Q)}}$ of the D latch. **20 ns**

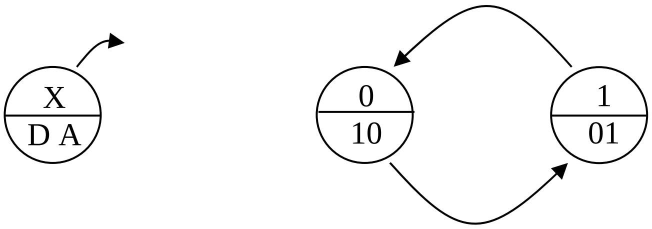
8. Implement a "dorm-room alarm" that accommodates eight sensor inputs, labeled S0 through S7, plus an ARM/DISARM pushbutton than can be used to "toggle" the state of the alarm system (a GREEN LED should be illuminated if the system is armed, and a YELLOW LED should be illuminated if the system is disarmed). If any sensors are asserted while the alarm is armed, the number of the *highest* sensor input asserted should be displayed on a 7-segment LED and a RED LED (that indicates the alarm has been tripped) should start *blinking* (at a 1 Hz rate, based on a clock signal provided by the function generator). The RED LED should stop blinking when the alarm is disarmed, and the 7-segment display should be blank (the 7-segment display should also be blank if the alarm is armed and none of the sensor inputs are asserted). Draw a Moore model for the "arm/disarm" state machine, and a separate Moore model for the "alarm tripped" state machine. Create Verilog program for your design, with all inputs and outputs defined.

   Create the following:
   a. Moore model of "arm/disarm" state machine
   b. Moore model of "alarm tripped" state machine
   c. Verilog source file listing



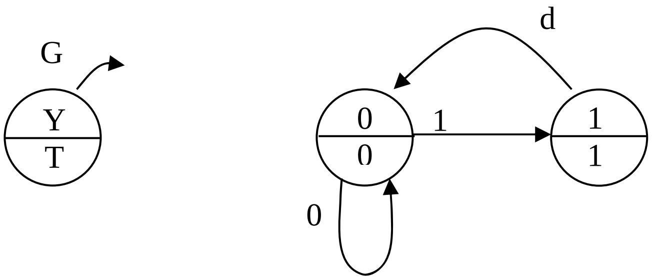
The DDA (Digital Dorm Alarm) in action. The GREEN LED indicates the alarm is in the "armed" state, the blinking RED LED indicates the alarm has been tripped, and the 7-segment display indicates the highest number sensor that is active. The Arm/Disarm pushbutton "toggles" the alarm between the armed and disarmed states. The 7-segment display is *blank* if the alarm is disarmed or, if armed, none of the sensor inputs are asserted.

The “arm/disarm” state machine can be realized with a single flip-flop (configured as a “T”); note that it is clocked by the arm/disarm bounceless switch (this state machine has *no inputs*):



X is state variable
D is disarmed (yellow) LED
A is armed (green) LED
Note – *no input variables*

The “alarm tripped” state machine can also be realized with a single flip-flop (configured as a “T”); it has a single input (STROBE) from the priority encoder and it is clocked by the (external) 1 Hz clock:



Y is state variable
T is tripped (red) LED
G (“strobe”) is input variable

```
module dda (ACLK, TCLK, S, nARMED, nTRIPPED, nL);

input wire ACLK;       // arm/disarm (bounceless switch) clock input
input wire TCLK;       // alarm tripped 1 Hz clock input
input wire [7:0] S;    // sensor inputs
output wire [6:0] nL;  // active low outputs to 7-segment display
output wire nARMED;    // active low ARMED output indicator
output wire nTRIPPED;  // active low TRIPPED output indicator

reg ARMED;             // ARMED flip-flop
reg TRIPPED;           // TRIPPED flip-flop
reg [6:0] L;           // LED segments (A..G), where L[6] = A, etc.
reg G;                 // encoder G output used to enable TRIPPED state machine

reg [7:0] GL;

assign nL = ~L;
assign nARMED = ~ARMED;
assign nTRIPPED = ~TRIPPED;

assign {G, L[6:0]} = GL;

always @ (posedge ACLK) begin
  ARMED <= ~ARMED;
end

always @ (posedge TCLK) begin
  TRIPPED <= ~TRIPPED & G;
end

always @ ({ARMED, S[7:0]}) begin
   casez({ARMED, S[7:0]})
//            7-segment LED   abcdefg
     9’b0????????:   GL = 8’b00000000;   // not armed
     9’b100000000:   GL = 8’b00000000;   // armed, but all sensors off
     9’b100000001:   GL = 8’b11111110;   // display 0
     9’b10000001?:   GL = 8’b10110000;   // display 1
     9’b1000001??:   GL = 8’b11101101;   // display 2
     9’b100001???:   GL = 8’b11111001;   // display 3
     9’b10001????:   GL = 8’b10110011;   // display 4
     9’b1001?????:   GL = 8’b11011011;   // display 5
     9’b101??????:   GL = 8’b11011111;   // display 6
     9’b11???????:   GL = 8’b11110000;   // display 7
   endcase
end

endmodule
```

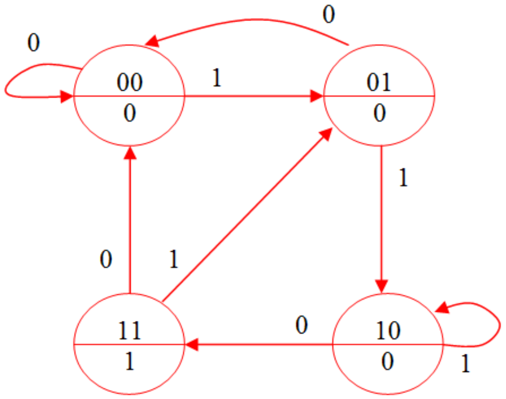
9. Given the following state transition diagram, determine:

   (a) Assuming the state machine depicted is initialized to state **00**, determine the output sequence generated by the input sequence **111000111000**

   000100000100

   (b) Determine the **embedded binary sequence** recognized by this state machine

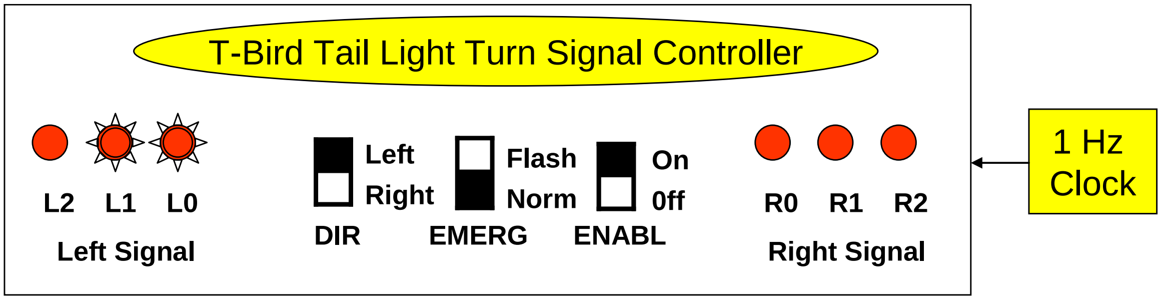
   110



10. Inspired by the (ancient) Beach Boys hit single, *Fun Fun Fun*, you wish to implement a T-Bird Tail Light Turn Signal Controller (TBTLTSC)…hoping, at long last, you've found *something* that your friends can actually relate to (maybe not the Beach Boys, though…). Here, each "tail light" will consist of three LEDs, which will be illuminated in a "building dot" mode to indicate the turn direction (either "left" or "right", selected by a DIP switch). An "emergency flash" mode (in which all the tail lights alternate between the on and off states) will be controlled by a second DIP switch. The overall taillight enable will be controlled by a third DIP switch; if disabled (ENABL=0), all LEDs should be off.

    Create the following:

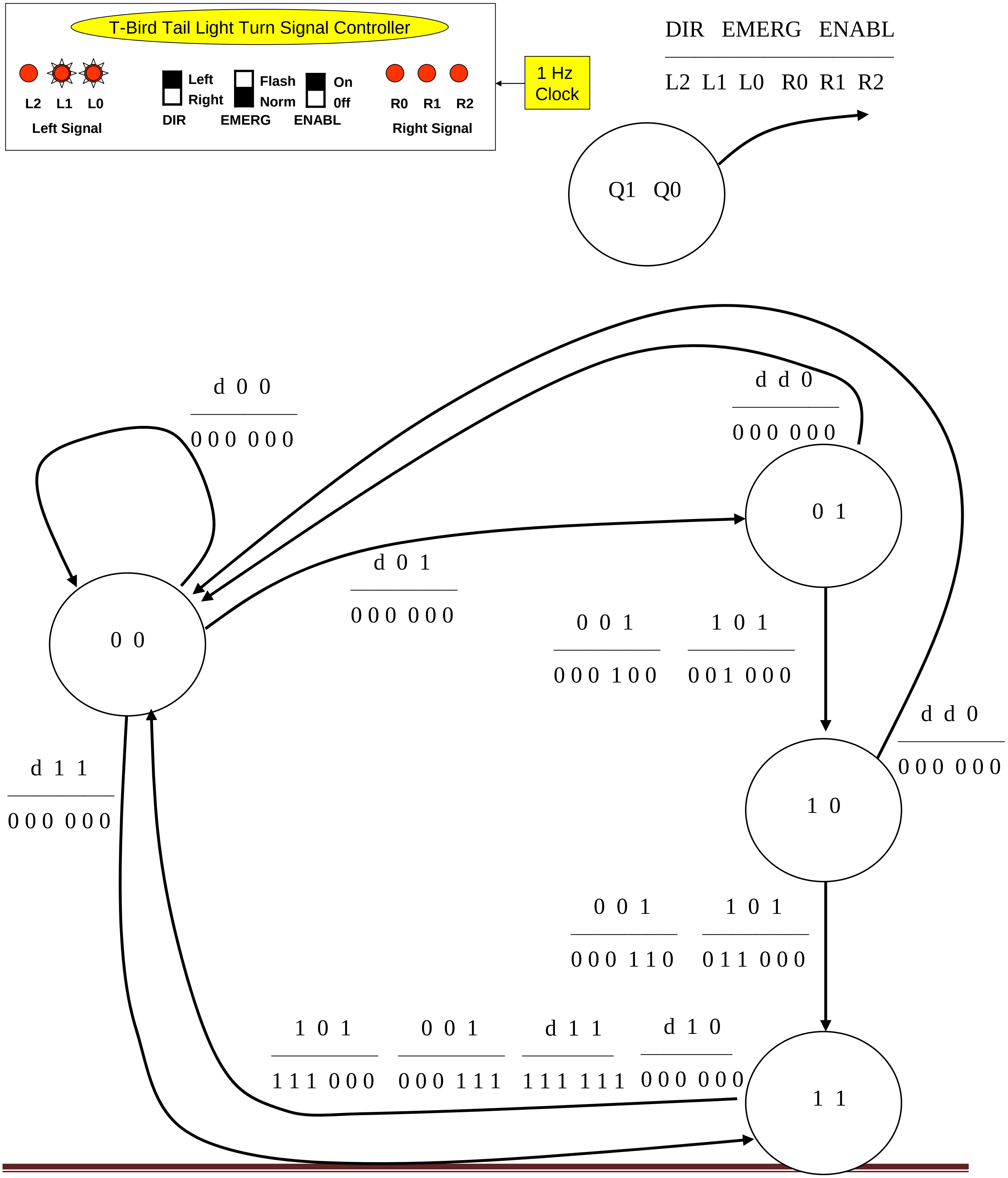
    a. Mealy model of state machine
    b. Verilog source file listing



The TBTLTSC taking a *left turn,* for which the output sequence should be: (a) L0, (b) L0 and L1, (c) L0, L1, and L2, (d) all off. This sequence should continuously repeat as long as the enable signal is asserted (ENABL=1). If the "emergency flash mode" is selected, the six lights should alternate between the "all on" and "all off" states (DIR is ignored). When disabled (ENABL=0), all LEDs should be off.

Assume switch in "down" position = 0, "up" position = 1

Mealy model can be realized with 4 states, that has 3 inputs, 6 outputs, and 2 state variables (many solution variants possible – Moore model would require more flip-flops but would be conceptually simpler) – note the simplification exploited here that a "newly selected" sequence does not necessarily have to start at its beginning

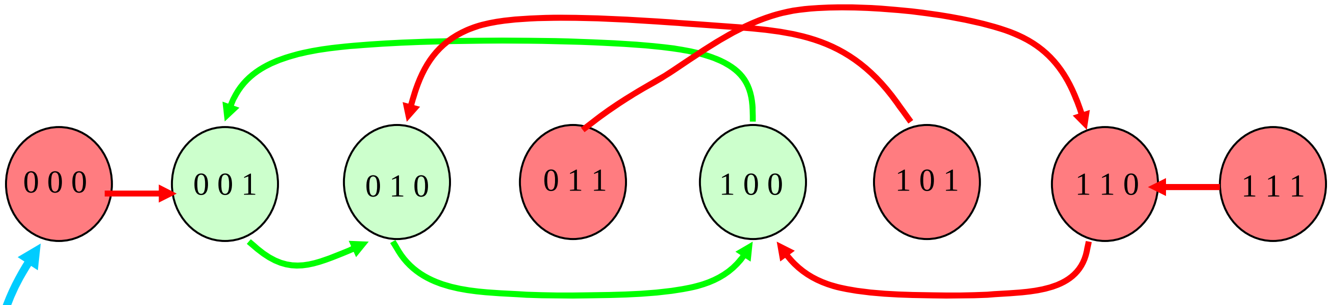


```
/* Mealy Model of TBTLTSC Implemented with State Diagram */

module tbtltsc_sd(CLK, DIR, EMERG, ENABLE, L2, L1, L0, R0, R1, R2);

  input wire CLK;           // clock input
  input wire DIR;            // direction control input
  input wire EMERG;          // emergency flash control input
  input wire ENABL;          // turn enable input
  output wire L2, L1, L0, R0, R1, R2;  // left/right output indicators

  reg [5:0] LR;              // left and right turn signal indicators

  reg [1:0] Q, next_Q;

  // State declarations
  localparam A0 = 2'b00;
  localparam A1 = 2'b01;
  localparam A2 = 2'b10;
  localparam A3 = 2'b11;

  // LR = L2, L1, L0, R0, R1, R2
  assign L2 = LR[5];
  assign L1 = LR[4];
  assign L0 = LR[3];
  assign R0 = LR[2];
  assign R1 = LR[1];
  assign R2 = LR[0];

  always @ (posedge CLK) begin
    Q <= next_Q;
  end

  always @ {DIR, EMERG, ENABL} begin
    case (Q)
      A0:      if ((EMERG==0)&(ENABL==0)) next_Q = A0;
          else if ((EMERG==0)&(ENABL==1)) next_Q = A1;
          else if ((EMERG==1)&(ENABL==1)) next_Q = A3;

      A1:      if (ENABL==0)              next_Q = A0;
          else if ((EMERG==0)&(ENABL==1)) next_Q = A2;

      A2:      if (ENABL==0)              next_Q = A0;
          else if ((EMERG==0)&(ENABL==1)) next_Q = A3;

      A3: next_Q = A0;
    endcase
  end

always @ (Q, DIR, EMERG, ENABL) begin
    casez ({Q,DIR,EMERG,ENABL})
      5'b00???: LR = 6'b000000;

      5’b01001: LR = 6’b000100;
      5’b01101: LR = 6’b001000;
      5’b01??0: LR = 6’b000000;

      5’b10001: LR = 6’b000110;
      5’b10101: LR = 6’b011000;
      5’b10??0: LR = 6’b000000;

      5’b11001: LR = 6’b111000;
      5’b11101: LR = 6’b000111;
      5’b11?11: LR = 6’b111111;
      5’b11?10: LR = 6’b000000;
    endcase
  end
endmodule
```

11. Design a 3-bit, self-correcting RING counter with glitch-free decoded outputs. Draw a state transition diagram to prove your design is self-correcting. NOTE: The initial state should be "001", and the counter should SHIFT LEFT.

    Create the following:

    a. Moore model of state machine, clearly showing the "self-correcting" mechanism
    b. Verilog source file listing



RING sequence

SELF-CORRECTING sequence

```
/* Self-Correcting 3-bit Ring Counter */

module ring3sc(CLK, Q);
  input wire CLK;
  output reg [2:0] Q;
  reg [2:0] next_Q;

  // Uses NOR function to make sure that
  //  the next state after d00 is 001

  always @(posedge CLK) begin
    Q <= next_Q;
  End

  always @ (Q) begin
    next_Q[2] = Q[1];
    next_Q[1] = Q[0];
    next_Q[0] = ~(Q[1] | Q[0]);
  end
endmodule
```

NOTE: The flip-flop outputs Q[2:0] are the "glitch-free decoded outputs" of interest. By definition, **NOTHING** needs to be done to "decode" them (each output = one state variable)!

The SELF-CORRECTING mechanism is of most interest in this problem – make sure that the feedback for self-correction specified in the Verilog program matches the one indicated in the state transition diagram