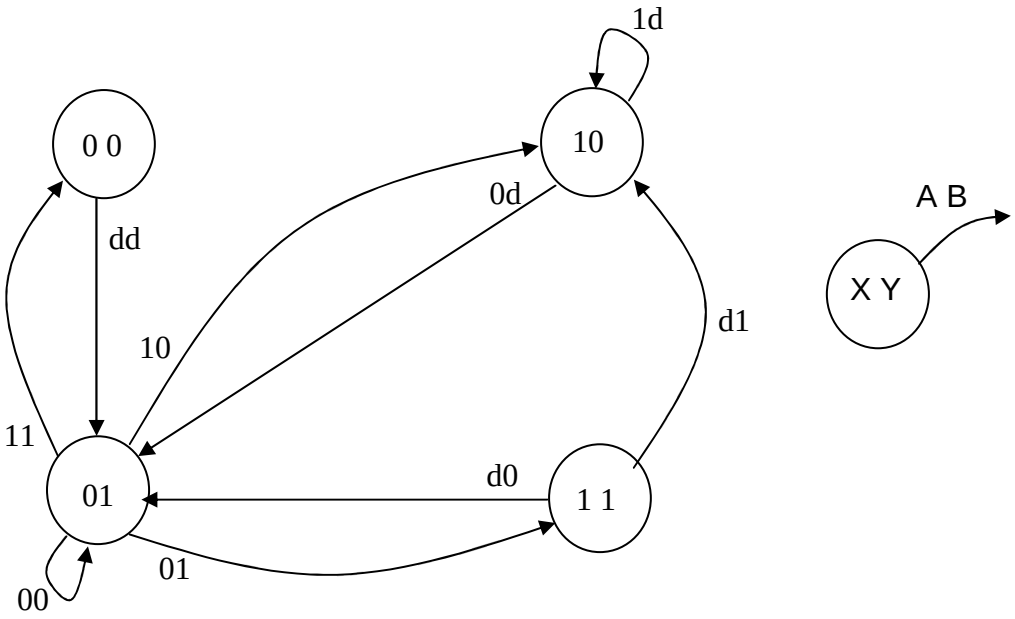
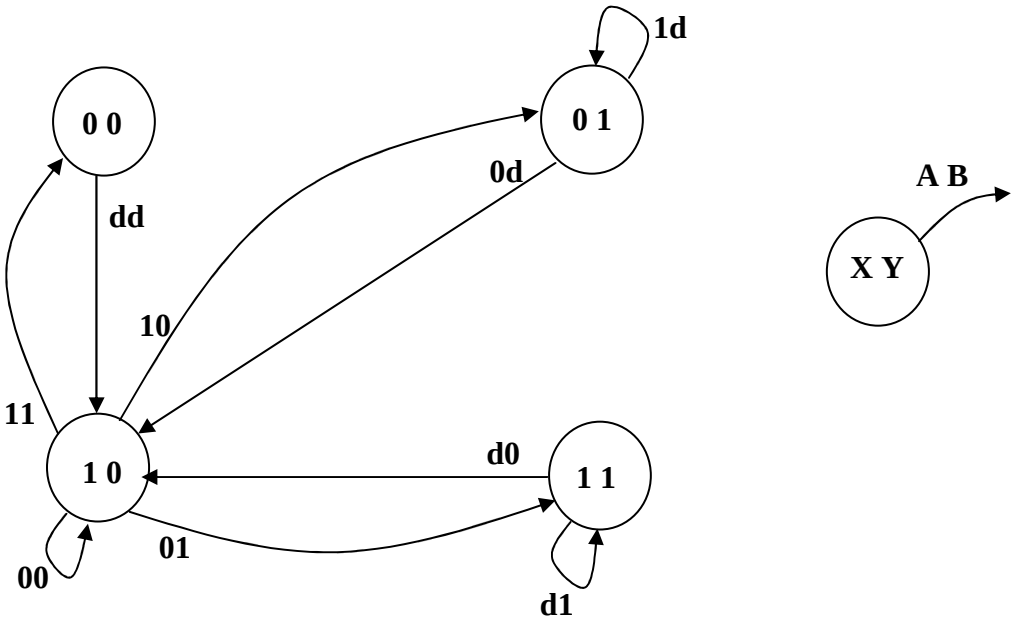
Practice Homework Solution for Module 3

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

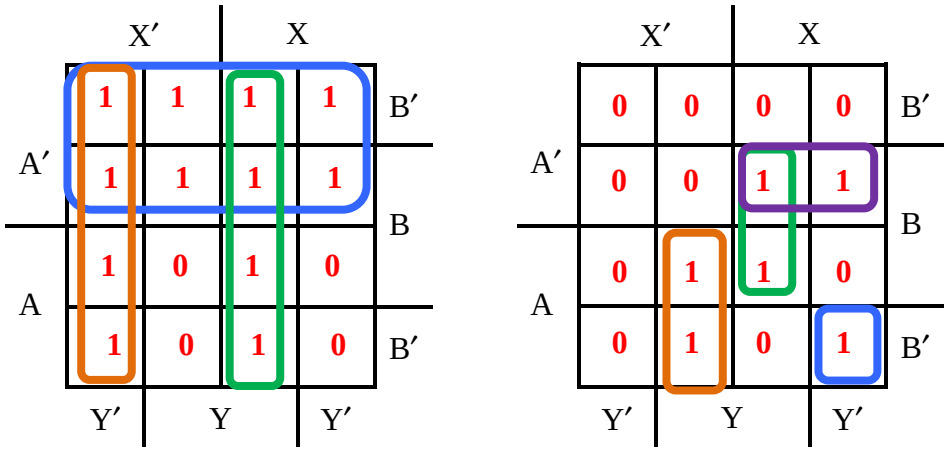


2. Given the following state transition diagram, determine the *next state equations* it represents in *minimum sum-of-products form.*



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

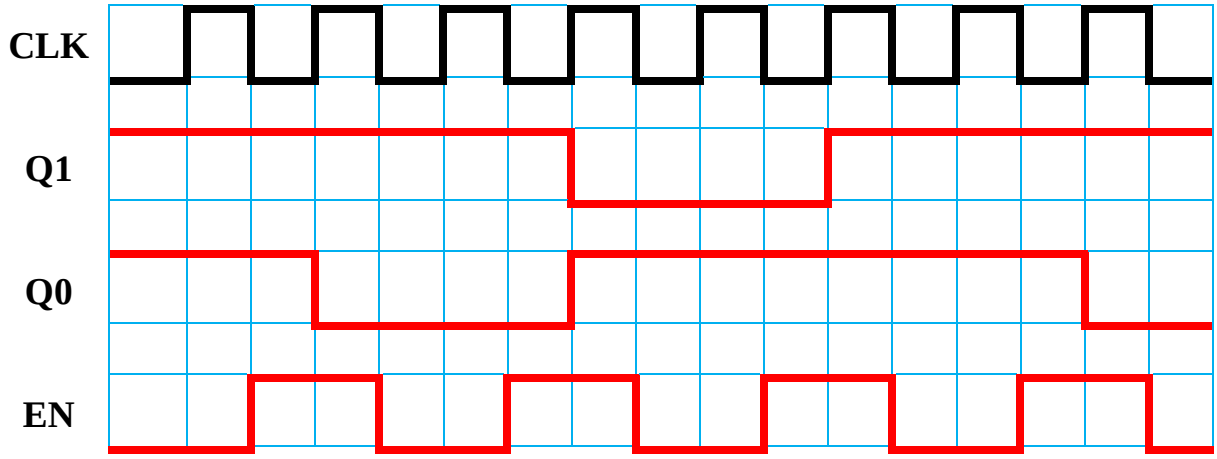
X* and Y* are "shorthand" for the next state of X and Y

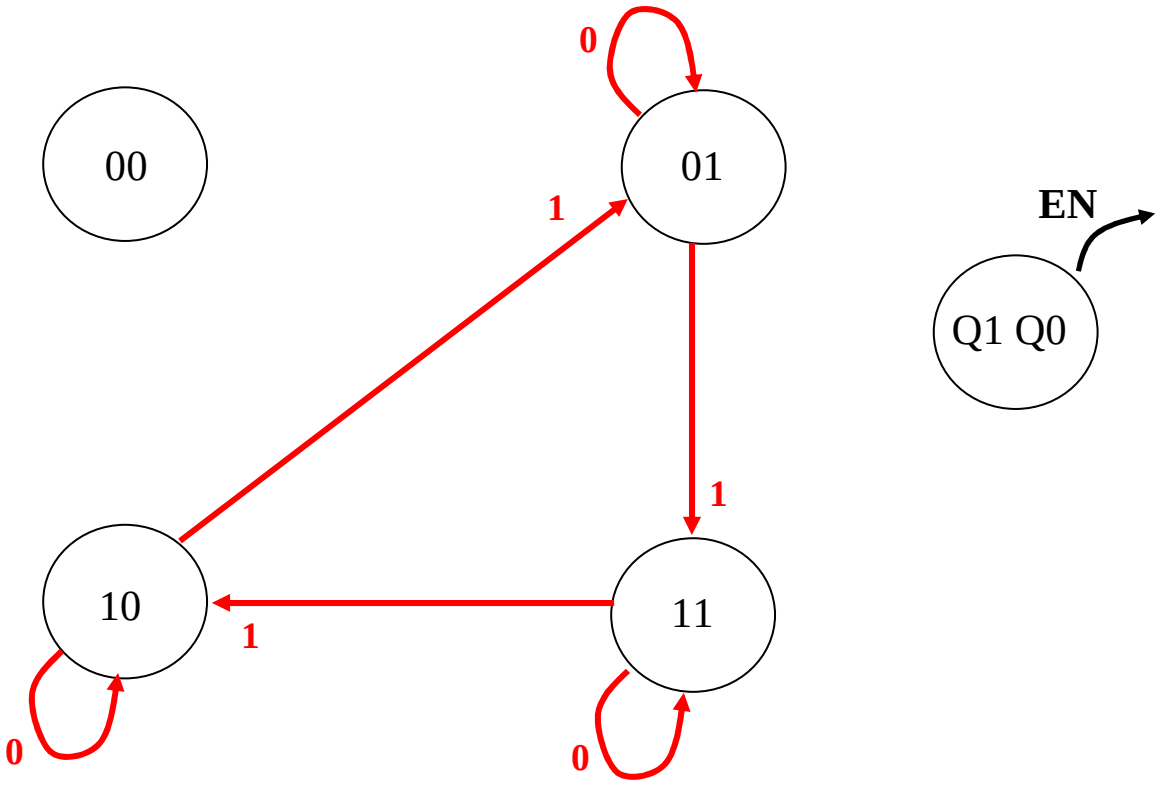


X* = X′·Y′ + X·Y + A′

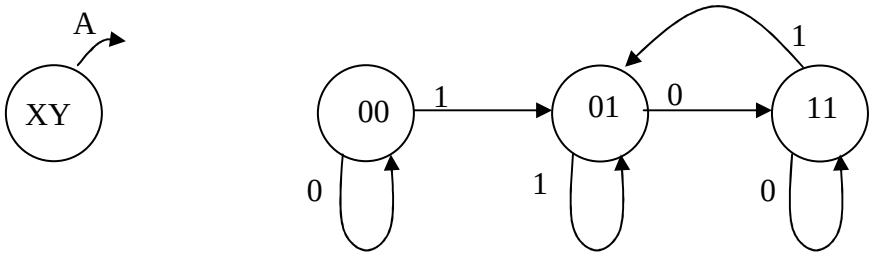
Y* = A·X′·Y + B·X·Y + A′·B·X + A·B′·X ·Y′

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′·Y

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

X* = A′·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·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.

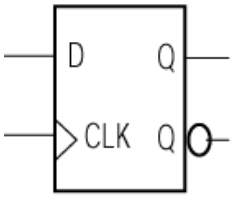
R	G	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

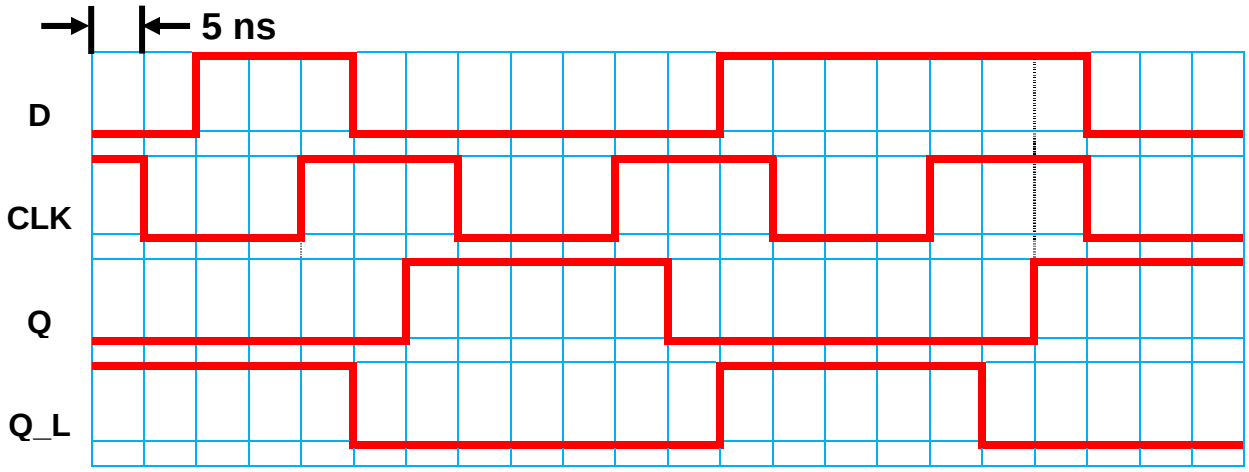
Excitation table:

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

Q* = R′•Q + G•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**

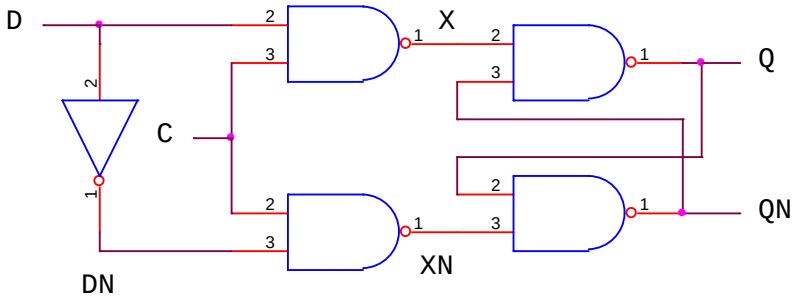
(b) The **nominal hold time** provided for the D flip-flop **5 ns**

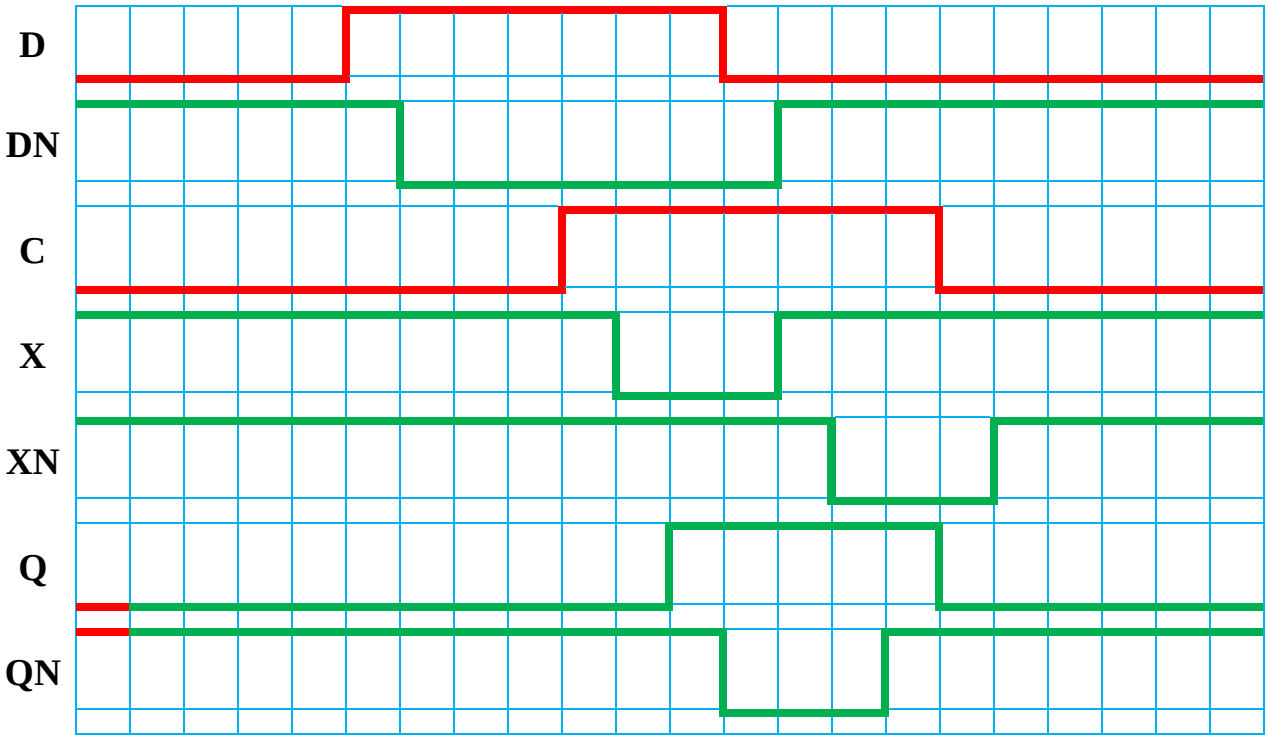
(c) The **nominal clock pulse width** provided for the D flip-flop **15 ns**

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

(e) The $\mathbf{t_{PLH(C\rightarrow Q)}}$ 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**

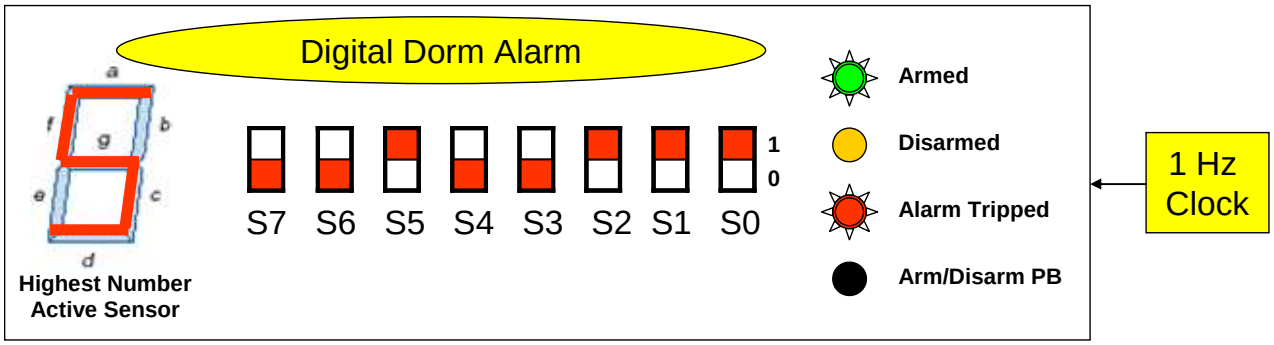
(c) Determine the **nominal hold time** provided for the D latch. **≥ 30 ns**

(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**

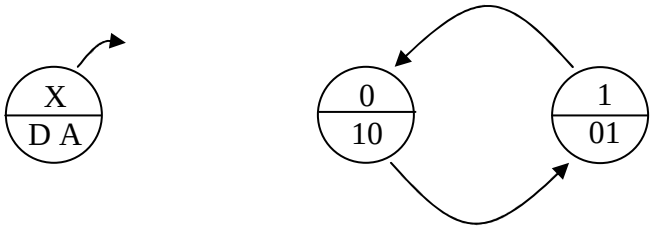
8. Implement a "dorm-room alarm" that accommodates eight sensor inputs, labeled S0 through S7, plus an ARM/DISARM pushbutton than can be used to "toggle" the state of the alarm system (a GREEN LED should be illuminated if the system is armed, and a YELLOW LED should be illuminated if the system is disarmed). 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 an ABEL source file for your design, with all inputs and outputs clearly defined.

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



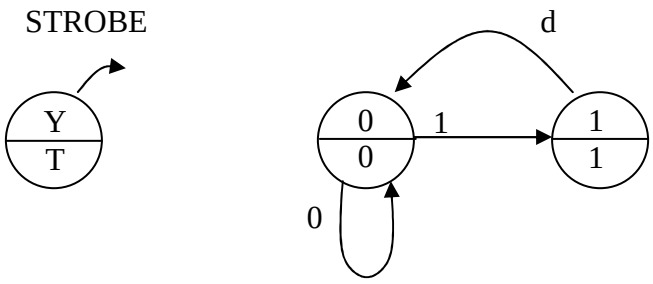
The DDA (Digital Dorm Alarm) in action. The GREEN LED indicates the alarm is in the "armed" state, the blinking RED LED indicates the alarm has been tripped, and the 7-segment display indicates the highest number sensor that is active. The Arm/Disarm pushbutton "toggles" the alarm between the armed and 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:



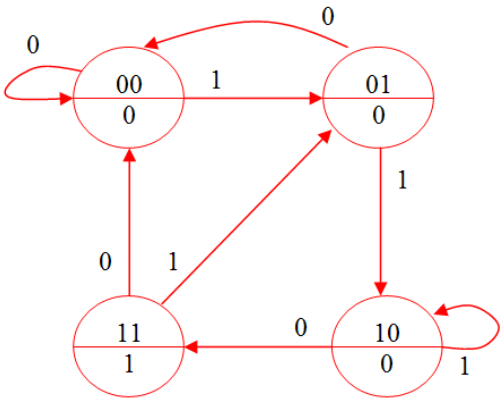
Y is state variable
T is tripped (red) LED
STROBE is input variable

```
MODULE dormalm

TITLE 'Digital Dorm Alarm with 7-segment Display'

DECLARATIONS
" Note – all LED outputs are active low (current sinking configuration)
ACLOCK pin; " arm/disarm bounceless switch clock input (need explicit pin assignment)
TCLOCK pin; " tripped 1 Hz (external) clock input (need explicit pin assignment)
S0..S7 pin;   " sensor inputs
!TRIPPED pin istype 'reg';  " TRIPPED ff/output indicator
!ARMED pin istype 'reg';    " ARMED ff/output indicator (also input to priority encoder)
!DISARMED pin istype 'com'; " DISARMED is just the complement of ARMED
STROBE pin istype 'com'; " STROBE output of encoder - used to enable TRIPPED state machine
!LA,!LB,!LC,!LD,!LE,!LF,!LG pin istype 'com'; " 7-segment display outputs

d = .X.;           " short hand for don't care

TRUTH_TABLE
 ([ARMED,S7, S6, S5, S4, S3, S2, S1, S0] -> [STROBE, LA, LB, LC, LD, LE, LF, LG])

  [  0 ,  d,  d,  d,  d,  d,  d,  d,  d] -> [   0,    0,  0,  0,  0,  0,  0,  0]; " off

  [  1 ,  0,  0,  0,  0,  0,  0,  0,  0] -> [   0,    0,  0,  0,  0,  0,  0,  0]; " off

  [  1 ,  0,  0,  0,  0,  0,  0,  0,  1] -> [   1,    1,  1,  1,  1,  1,  1,  0]; " 0

  [  1 ,  0,  0,  0,  0,  0,  0,  1,  d] -> [   1,    0,  1,  1,  0,  0,  0,  0]; " 1

  [  1 ,  0,  0,  0,  0,  0,  1,  d,  d] -> [   1,    1,  1,  0,  1,  1,  0,  1]; " 2

  [  1 ,  0,  0,  0,  0,  1,  d,  d,  d] -> [   1,    1,  1,  1,  1,  0,  0,  1]; " 3

  [  1 ,  0,  0,  0,  1,  d,  d,  d,  d] -> [   1,    0,  1,  1,  0,  0,  1,  1]; " 4

  [  1 ,  0,  0,  1,  d,  d,  d,  d,  d] -> [   1,    1,  0,  1,  1,  0,  1,  1]; " 5

  [  1 ,  0,  1,  d,  d,  d,  d,  d,  d] -> [   1,    1,  0,  1,  1,  1,  1,  1]; " 6

  [  1 ,  1,  d,  d,  d,  d,  d,  d,  d] -> [   1,    1,  1,  1,  0,  0,  0,  0]; " 7


EQUATIONS

DISARMED = !ARMED;

ARMED := !ARMED;
ARMED.CLK = ACLOCK;

TRIPPED := !TRIPPED & STROBE;
TRIPPED.CLK = TCLOCK;


END
```

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

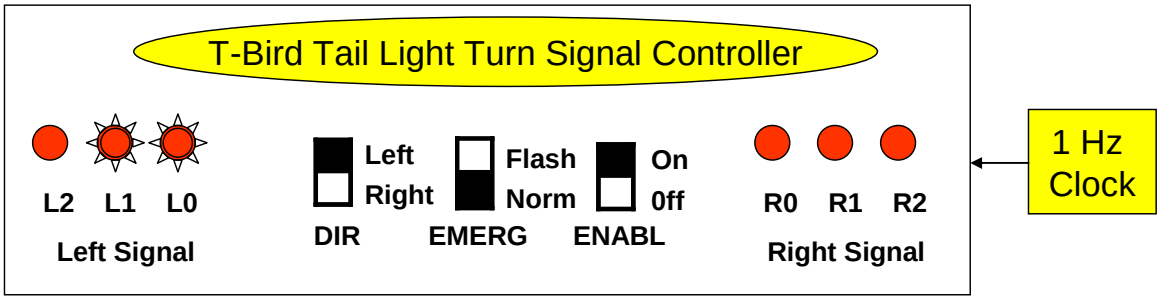


   (b) Determine the **embedded binary sequence** recognized by this state machine

   110

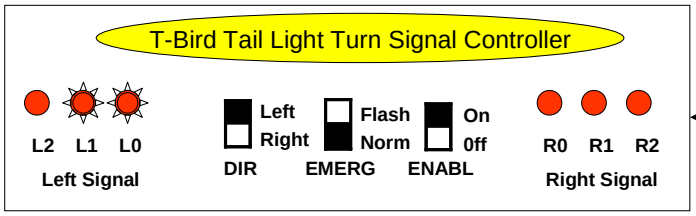
10. Inspired by the (ancient) Beach Boys hit single, *Fun Fun Fun*, you wish to implement a T-Bird Tail Light Turn Signal Controller (TBTLTSC)…hoping, at long last, you've found *something* that your friends can actually relate to (maybe not the Beach Boys, though…). Here, each "tail light" will consist 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 (EN=0), all LEDs should be off.

    Create the following:
    a. Mealy model of state machine
    b. ABEL source file listing

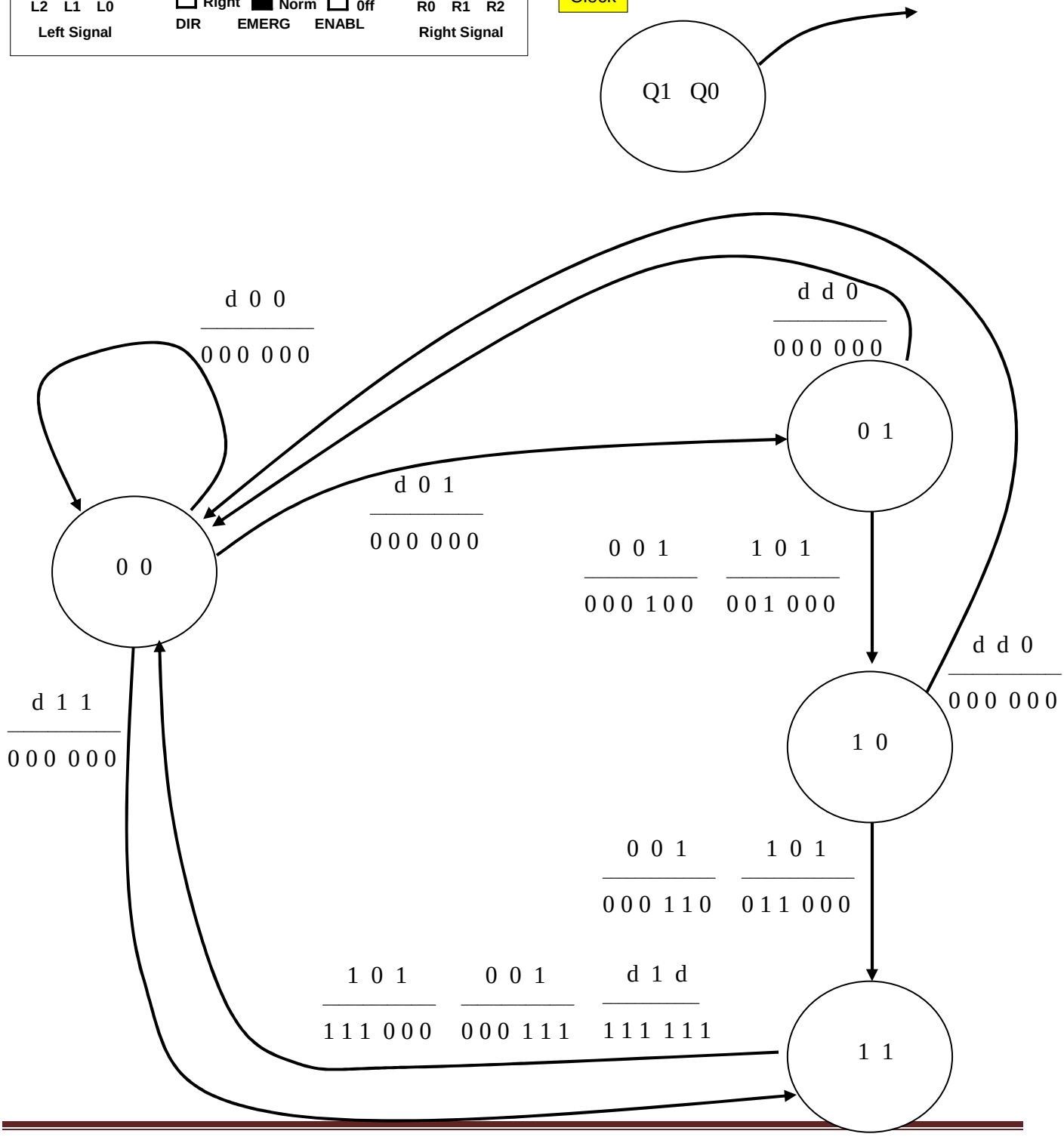


The TBTLTSC taking a *left turn,* for which the output sequence should be: (a) L0, (b) L0 and L1, (c) L0, L1, and L2, (d) all off. This sequence should continuously repeat as long as the enable signal is asserted (ENABL=1). If the "emergency flash mode" is selected, the six lights should alternate between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



DIR EMERG ENABL
L2 L1 L0 R0 R1 R2

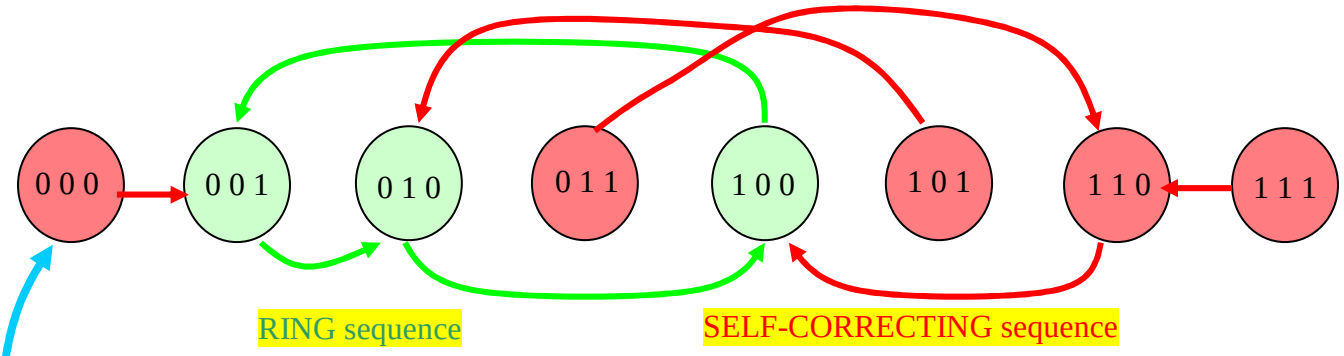


```
MODULE TBTLTSC

TITLE 'T-Bird Tail Light Turn Signal Controller - Mealy Model'

DECLARATIONS

Q1..Q0 pin istype 'reg';
L2..L0 pin istype 'com';
R2..R0 pin istype 'com';
DIR, EMERG, ENABL pin;
CLOCK pin;


TRUTH_TABLE ([Q1,Q0,DIR,EMERG,ENABL] :> [Q1,Q0])
             [ 0, 0, 0,   0,    0  ] :> [ 0, 0];
             [ 0, 0, 0,   0,    1  ] :> [ 0, 1];
             [ 0, 0, 0,   1,    0  ] :> [ 0, 0];
             [ 0, 0, 0,   1,    1  ] :> [ 1, 1];
             [ 0, 0, 1,   0,    0  ] :> [ 0, 0];
             [ 0, 0, 1,   0,    1  ] :> [ 0, 1];
             [ 0, 0, 1,   1,    0  ] :> [ 0, 0];
             [ 0, 0, 1,   1,    1  ] :> [ 1, 1];

             [ 0, 1, 0,   0,    0  ] :> [ 0, 0];
             [ 0, 1, 0,   0,    1  ] :> [ 1, 0];
             [ 0, 1, 0,   1,    0  ] :> [ 0, 0];
             [ 0, 1, 0,   1,    1  ] :> [ 0, 0];
             [ 0, 1, 1,   0,    0  ] :> [ 0, 0];
             [ 0, 1, 1,   0,    1  ] :> [ 1, 0];
             [ 0, 1, 1,   1,    0  ] :> [ 0, 0];
             [ 0, 1, 1,   1,    1  ] :> [ 0, 0];

             [ 1, 0, 0,   0,    0  ] :> [ 0, 0];
             [ 1, 0, 0,   0,    1  ] :> [ 1, 1];
             [ 1, 0, 0,   1,    0  ] :> [ 0, 0];
             [ 1, 0, 0,   1,    1  ] :> [ 0, 0];
             [ 1, 0, 1,   0,    0  ] :> [ 0, 0];
             [ 1, 0, 1,   0,    1  ] :> [ 1, 1];
             [ 1, 0, 1,   1,    0  ] :> [ 0, 0];
             [ 1, 0, 1,   1,    1  ] :> [ 0, 0];

             [ 1, 1, 0,   0,    0  ] :> [ 0, 0];
             [ 1, 1, 0,   0,    1  ] :> [ 0, 0];
             [ 1, 1, 0,   1,    0  ] :> [ 0, 0];
             [ 1, 1, 0,   1,    1  ] :> [ 0, 0];
             [ 1, 1, 1,   0,    0  ] :> [ 0, 0];
             [ 1, 1, 1,   0,    1  ] :> [ 0, 0];
             [ 1, 1, 1,   1,    0  ] :> [ 0, 0];
             [ 1, 1, 1,   1,    1  ] :> [ 0, 0];


TRUTH_TABLE ([Q1,Q0,DIR,EMERG,ENABL] -> [L2,L1,L0,R0,R1,R2])
             [ 0, 0, 0,   0,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 0, 0, 0,   0,    1  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 0, 0, 0,   1,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 0, 0, 0,   1,    1  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 0, 0, 1,   0,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 0, 0, 1,   0,    1  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 0, 0, 1,   1,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 0, 0, 1,   1,    1  ] -> [ 0, 0, 0, 0, 0, 0];

             [ 0, 1, 0,   0,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 0, 1, 0,   0,    1  ] -> [ 0, 0, 0, 1, 0, 0];
             [ 0, 1, 0,   1,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 0, 1, 0,   1,    1  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 0, 1, 1,   0,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 0, 1, 1,   0,    1  ] -> [ 0, 0, 1, 0, 0, 0];
             [ 0, 1, 1,   1,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 0, 1, 1,   1,    1  ] -> [ 0, 0, 0, 0, 0, 0];

             [ 1, 0, 0,   0,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 1, 0, 0,   0,    1  ] -> [ 0, 0, 0, 1, 1, 0];
             [ 1, 0, 0,   1,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 1, 0, 0,   1,    1  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 1, 0, 1,   0,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 1, 0, 1,   0,    1  ] -> [ 0, 1, 1, 0, 0, 0];
             [ 1, 0, 1,   1,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 1, 0, 1,   1,    1  ] -> [ 0, 0, 0, 0, 0, 0];

             [ 1, 1, 0,   0,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 1, 1, 0,   0,    1  ] -> [ 0, 0, 0, 1, 1, 1];
             [ 1, 1, 0,   1,    0  ] -> [ 1, 1, 1, 1, 1, 1];
             [ 1, 1, 0,   1,    1  ] -> [ 1, 1, 1, 1, 1, 1];
             [ 1, 1, 1,   0,    0  ] -> [ 0, 0, 0, 0, 0, 0];
             [ 1, 1, 1,   0,    1  ] -> [ 1, 1, 1, 0, 0, 0];
             [ 1, 1, 1,   1,    0  ] -> [ 1, 1, 1, 1, 1, 1];
             [ 1, 1, 1,   1,    1  ] -> [ 1, 1, 1, 1, 1, 1];

EQUATIONS

[Q1..Q0].CLK = CLOCK;


END
```

```
Q1.D   = (  ENABL & !EMERG & !Q0.PIN & Q1.PIN
          #   ENABL & !EMERG & Q0.PIN & !Q1.PIN
          #   ENABL & EMERG & !Q0.PIN & !Q1.PIN ); " ISTYPE 'INVERT'
Q1.C   = (  CLOCK );

Q0.D   = (  ENABL & !EMERG & !Q0.PIN
          #   ENABL & !Q0.PIN & !Q1.PIN ); " ISTYPE 'INVERT'
Q0.C   = (  CLOCK );

L2     = (  EMERG & Q0.PIN & Q1.PIN
          #   ENABL & DIR & Q0.PIN & Q1.PIN );

L1     = (  ENABL & !EMERG & DIR & Q1.PIN
          #   EMERG & Q0.PIN & Q1.PIN );

L0     = (  ENABL & !EMERG & DIR & Q0.PIN
          #   ENABL & !EMERG & DIR & Q1.PIN
          #   EMERG & Q0.PIN & Q1.PIN );

R0     = (  ENABL & !EMERG & !DIR & Q0.PIN
          #   ENABL & !EMERG & !DIR & Q1.PIN
          #   EMERG & Q0.PIN & Q1.PIN );

R1     = (  ENABL & !EMERG & !DIR & Q1.PIN
          #   EMERG & Q0.PIN & Q1.PIN );

R2     = (  EMERG & Q0.PIN & Q1.PIN
          #   ENABL & !DIR & Q0.PIN & Q1.PIN );
```

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. ABEL source file listing




```
MODULE ring_cnt

TITLE 'Self-Correcting 3-bit RING Counter'

DECLARATIONS
CLOCK pin;

R2..R0 pin istype 'reg';

EQUATIONS

R2 := R1;

R1 := R0;

R0 := !(R1 # R0);

[R0..R2].CLK = CLOCK;

END
```

NOTE: The flip-flop outputs [R0..R2] are the "glitch-free decoded outputs" of interest. By definition, NOTHING needs to be done to "decode" them!

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