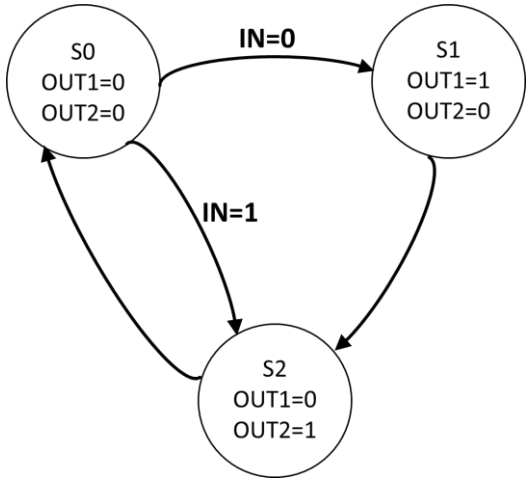
Tutorial 10 (SOLUTIONS)

NOTE THAT THERE ARE MANY OTHER POSSIBLE SOLUTIONS TOO…

SS1

1) State Transition Diagram



2) Next State Table

Current State	Inputs	Next State
S	**IN**	**S+**
S0	0	S1
S0	1	S2
S1	X	S2
S2	X	S0

By using two bits to represent the state, S_1S_0 :

State Assignment

	S_1	S_0
S0	0	0
S1	0	1
S2	1	0
S3	1	1

Current State		Inputs	Next State	
S_1	**S_0**	**IN**	**S_1+**	**S_0+**
0	0	0	0	1
0	0	1	1	0
0	1	X	1	0
1	0	X	0	0
1	1	X	0	0

The next state after S3 has been set to S0 instead of don't cares. This is a minimum-risk design choice because there is no external reset signal to this FSM.

A two bit counter can be designed using D Flip-flops for the state memory :

S_1S_0 \ IN	0	1
00	0	1
01	1	1
11	0	0
10	0	0

S_1S_0 \ IN	0	1
00	1	0
01	0	0
11	0	0
10	0	0

$$D_1 = S_1^+ = \overline{S_1}\,IN + \overline{S_1}S_0 \qquad D_0 = S_0^+ = \overline{IN}\,\overline{S_1}\,\overline{S_0}$$

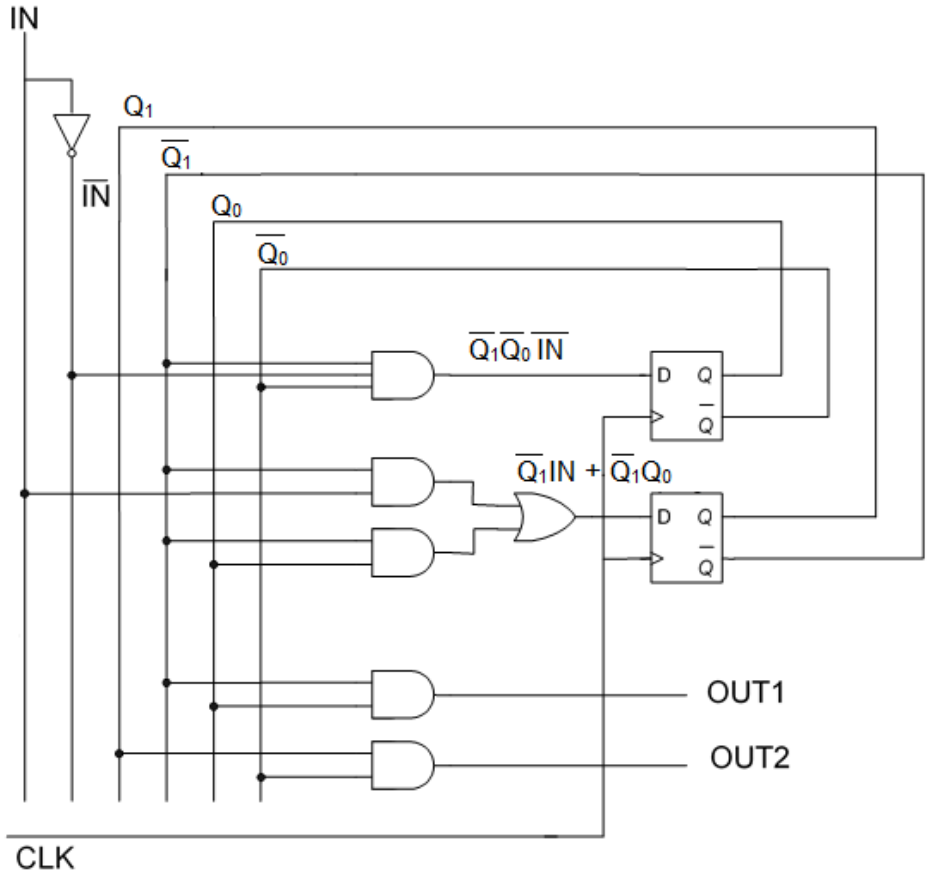
3) Output Logic Table

Current State	Outputs	
S	**OUT1**	**OUT2**
S0	0	0
S1	1	0
S2	0	1

Current State		Outputs	
S_1	**S_0**	**OUT1**	**OUT2**
0	0	0	0
0	1	1	0
1	0	0	1
1	1	X	X

$$OUT_1 = S_1 = \overline{Q_1}Q_0, \quad OUT_2 = S_2 = Q_1\overline{Q_0}$$

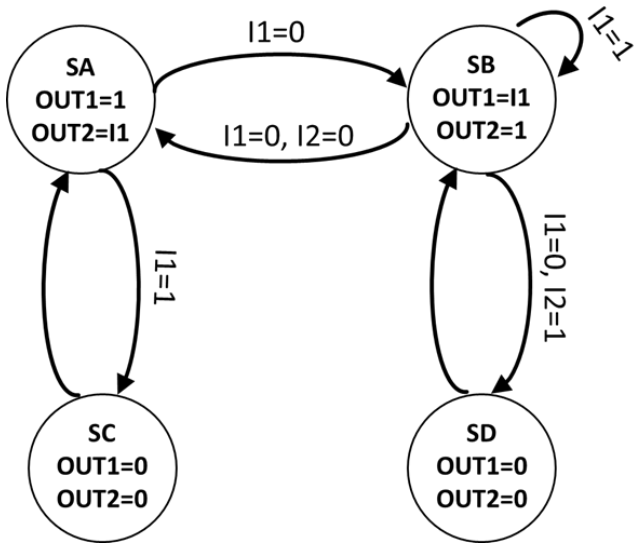
4) Circuit Implementation



Notice that this is a Moore machine as the outputs (OUT1, OUT2) are a function of the current state (Q_1, Q_0) only.

SS2

1) State Transition Diagram



2) Next State Table

Current State	Inputs		Next State
S	**I1**	**I2**	**S+**
SA	0	X	SB
SA	1	X	SC
SB	0	0	SA
SB	0	1	SD
SB	1	X	SB
SC	X	X	SA
SD	X	X	SB

You can arrange the next state table like a KMAP to save yourself one step!

	I_1, I_2			
	00	01	11	10
SA	SB	SB	SC	SC
SB	SA	SD	SB	SB
SD	SB	SB	SB	SB
SC	SA	SA	SA	SA

A two bit counter can be designed using D Flip-flops for the state memory.

Applying the following State Assignment

	Q_1	Q_0
SA	0	0
SB	0	1
SC	1	0
SD	1	1

Q_1Q_0 \ I_1I_2	00	01	11	10
00	0	0	1	1
01	0	1	0	0
11	0	0	0	0
10	0	0	0	0

Q_1Q_0 \ I_1I_2	00	01	11	10
00	1	1	0	0
01	0	1	1	1
11	1	1	1	1
10	0	0	0	0

$$D_1 = Q_1^+ = \overline{Q_1}\,\overline{Q_0} I_1 + \overline{Q_1} Q_0 \overline{I_1} I_2$$

$$D_0 = Q_0^+ = \overline{Q_1}\,\overline{Q_0}\,\overline{I_1} + Q_1 Q_0 + Q_0 I_2 + Q_0 I_1$$

3) Output Logic Table

Current State	Outputs	
S	**OUT1**	**OUT2**
SA	1	I1
SB	I1	1
SC	0	0
SD	0	0

By inspection,

$$OUT1 = SA + SB.I_1$$
$$= \overline{Q_1}\,\overline{Q_0} + \overline{Q_1}Q_0I_1 = \overline{Q_1}\,\overline{Q_0} + \overline{Q_1}I_1$$

$$OUT2 = SB + SA.I_1$$
$$= \overline{Q_1}\,\overline{Q_0}I_1 + \overline{Q_1}Q_0 = \overline{Q_1}\,Q_0 + \overline{Q_1}I_1$$

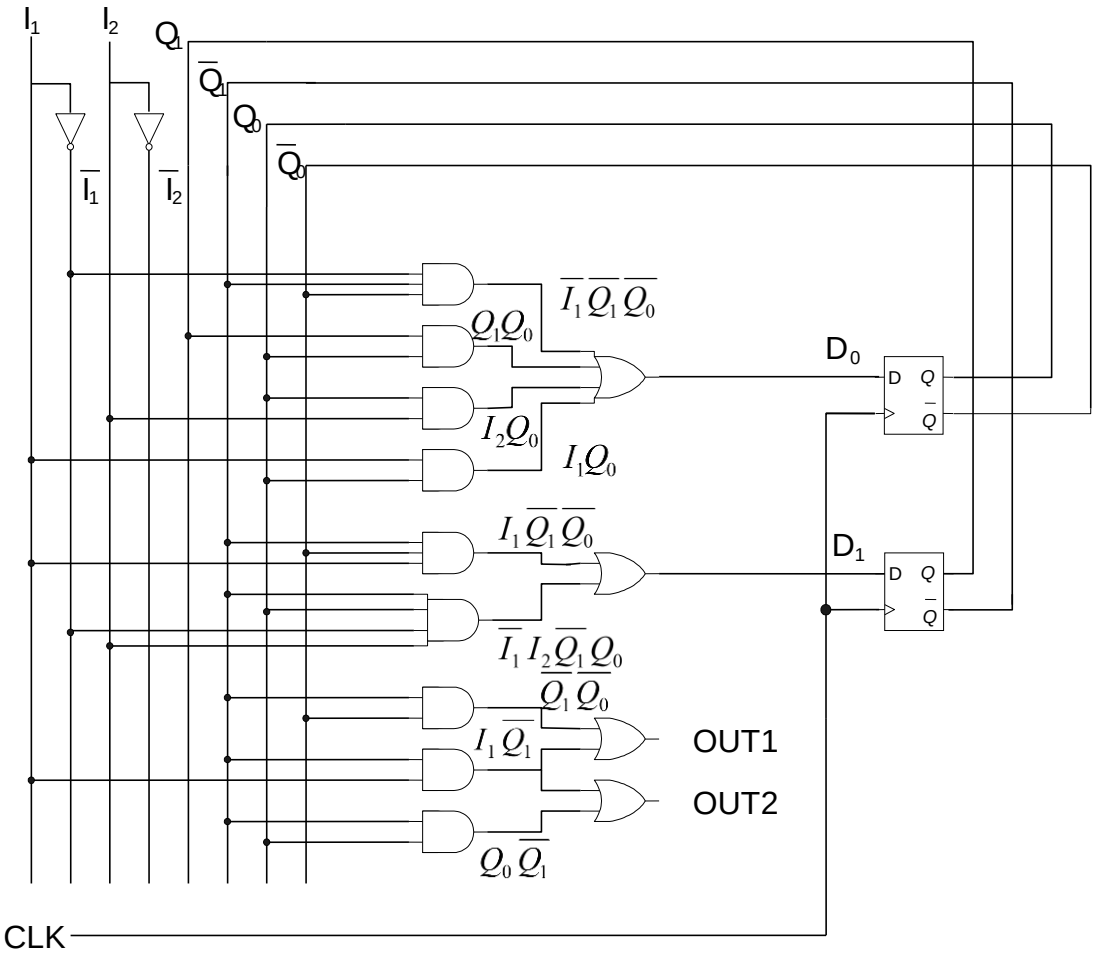
Current State	Inputs	Outputs	
Q1Q0	**I1**	**OUT1**	**OUT2**
00	0	1	0
00	1	1	1
01	0	0	1
01	1	1	1
10	X	0	0
11	X	0	0

Solving (KMAP),

$$OUT1 = \overline{Q_1}.\overline{Q_0} + \overline{Q_1}.I_1$$

$$OUT2 = \overline{Q_1}.Q_0 + \overline{Q_1}.I_1$$

4) Circuit Implementation



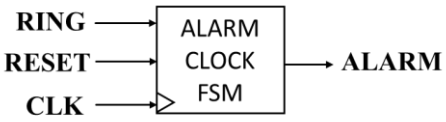
As the outputs, OUT1 and OUT2, are a function of both current state (Q_1, Q_0) as well as the current inputs (I1, I2), this is a Mealy machine.

Question 1

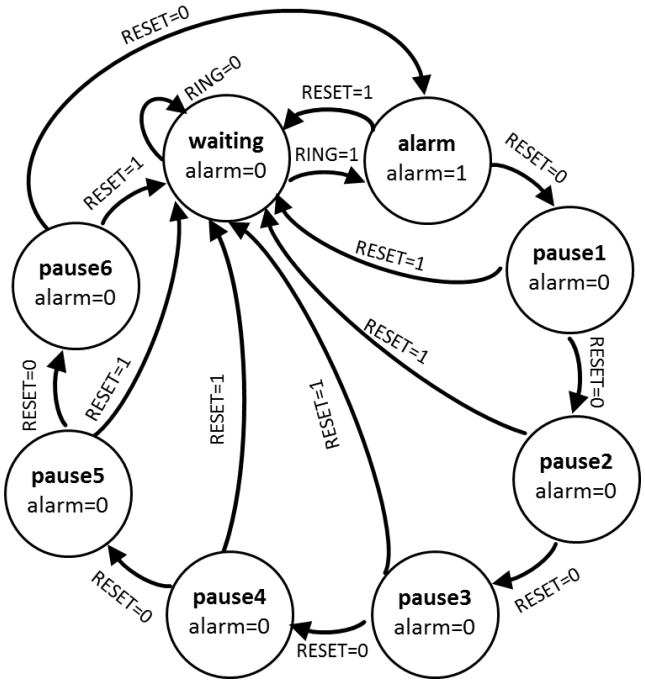
Selecting a clock period of 10 seconds, the snooze can be implemented by having 1 state of alarm (10 seconds) followed by 6 states of waiting (60 seconds).

Solution 1 :

Overall Block Diagram:

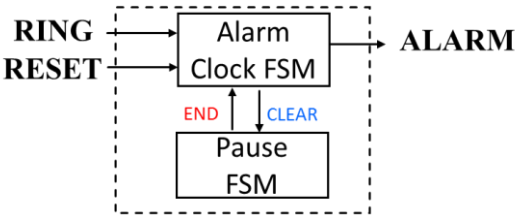


Alarm Clock FSM :

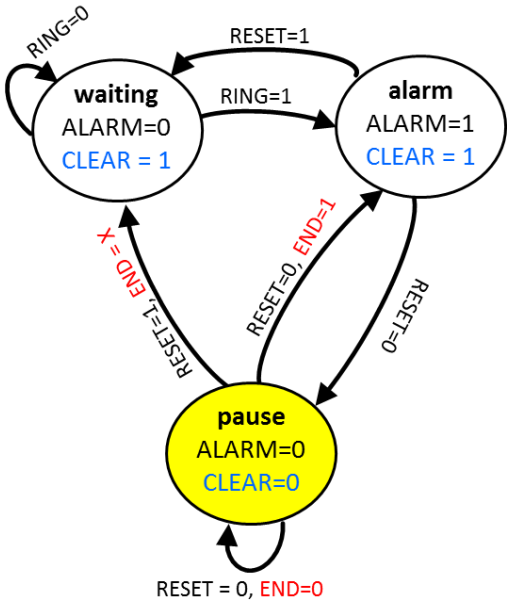


Solution 2 :

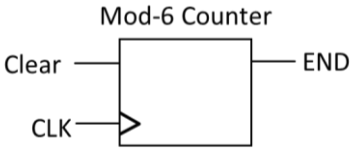
Modular FSM Block Diagram:



Alarm Clock FSM



Pause FSM

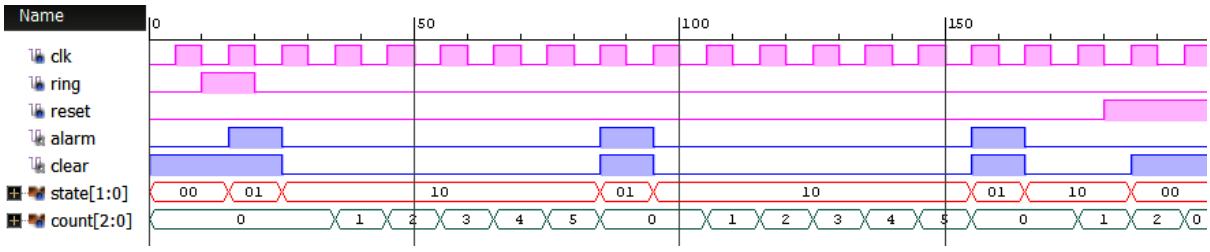


END= 1 *when* count = 5 (101)

Verilog Code (For Reference Only)

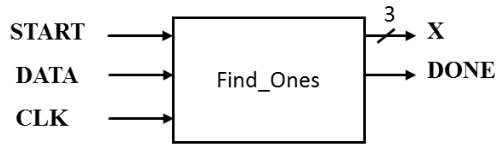
```
// This is a Verilog realization of the Alarm Clock state machine

module alarm_clock (input clk, ring, reset, output alarm);

reg [1:0] state = 2'b00; reg [1:0] nextstate = 2'b00;

// A temporary register is used to keep count of 1 minute
reg [2:0] count=3'b000;

wire done;

// wait is a Verilog reserved word and shouldn't be used as a state name
parameter WAITING = 2'b00, ALARM = 2'b01, PAUSE = 2'b10;

//Next State Logic
always @ (*) begin
   case (state)
      WAITING :  nextstate = ring ? ALARM : WAITING;
      ALARM :    nextstate = reset ? WAITING : PAUSE;
      PAUSE :    if (reset) nextstate = WAITING;
      else if (done) nextstate = ALARM;
      default :     nextstate = WAITING;
   endcase
end

//State Memory (2 x D Flip-flops)
always @(posedge clk) begin
      state <= nextstate;
end

//Output Logic
assign alarm = (state == ALARM);
assign clear = (state == WAITING) || (state == ALARM);

//Mod-6 counter for Sleep FSM
always @(posedge clk) begin
   if (clear) count <= 0;
   else begin
      if (count == 3'b101) begin
         count <= 0;
      end
      else begin
         count <= count +1;
      end
   end
end

assign done = count[2] & count[0];

endmodule
```

Verilog Simulation Results

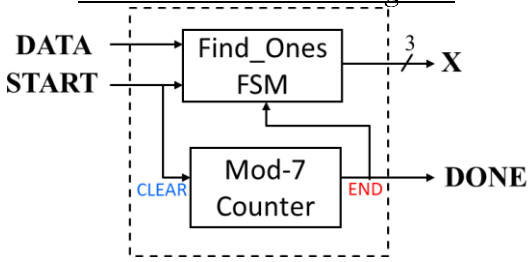


Question 2

Overall Block Diagram :



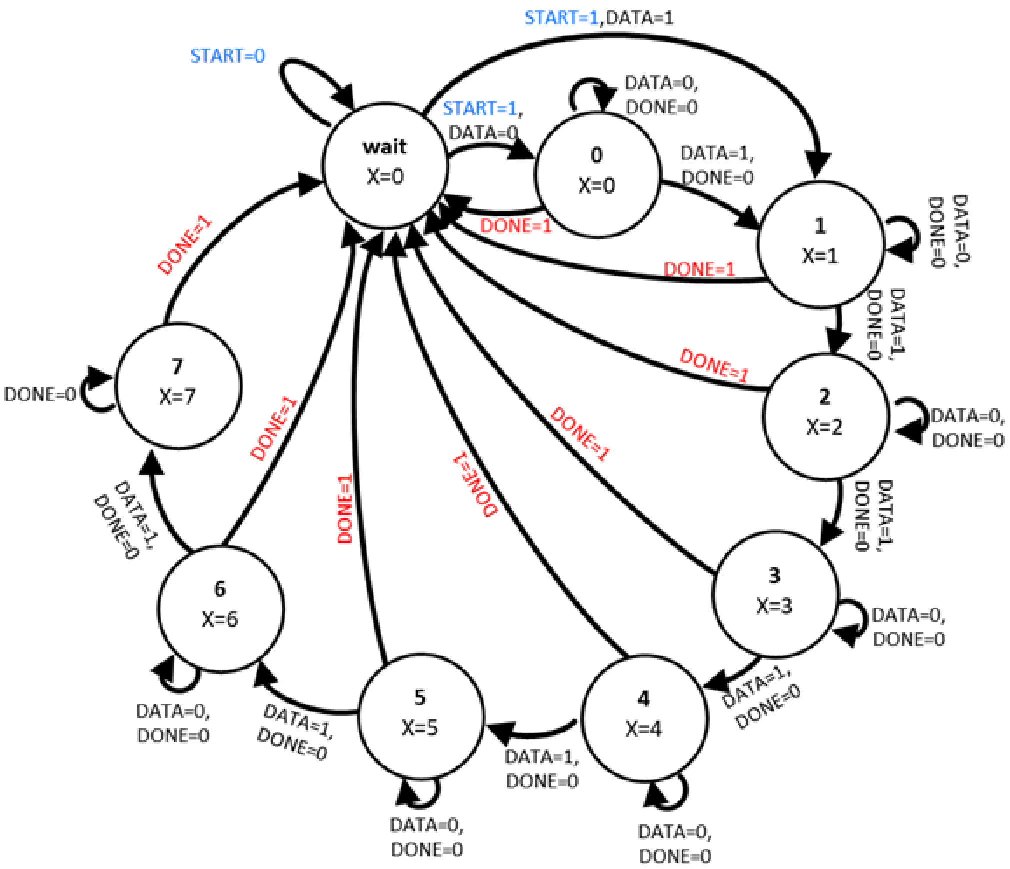
Modular FSM Block Diagram:



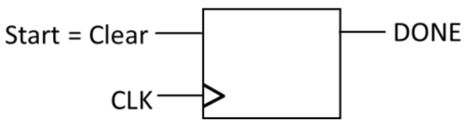
START = CLEAR & DONE = END

By reusing the START signal as the CLEAR signal, and the DONE signal as the END signal, we save two signals.

Find_Ones FSM :



Mod-7 Counter

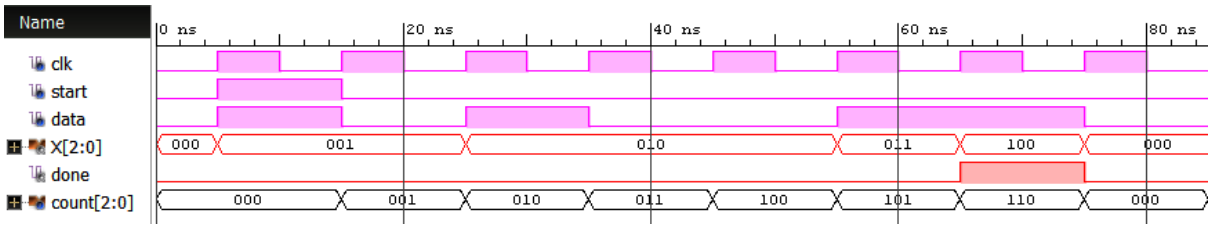


DONE= 1 *when* count = 6 (110)

Verilog Code (For Reference Only)

```
module find_ones(input clk, start, data, output done, output [2:0] X);

reg [3:0] state = 4'b1000; reg [3:0] nextstate = 0;

// A temporary register is used to keep count of 7 cycles.
reg [2:0] count = 3'b000;

parameter WAITING = 4'b1000, ZERO = 4'b0000, ONE = 4'b0001, TWO = 4'b0010,
THREE = 4'b0011, FOUR = 4'b0100, FIVE = 4'b0101, SIX = 4'b0110, SEVEN =
4'b0111;

//Next State Logic
always @ (*) begin
  case (state)
      WAITING:    nextstate = start ? (data ? ONE : ZERO) : WAITING;
      ZERO:       nextstate = done ? WAITING: (data ? ONE : ZERO);
      ONE:        nextstate = done ? WAITING: (data ? TWO : ONE);
      TWO:        nextstate = done ? WAITING: (data ? THREE : TWO);
      THREE:      nextstate = done ? WAITING: (data ? FOUR : THREE);
      FOUR:       nextstate = done ? WAITING: (data ? FIVE : FOUR);
      FIVE:       nextstate = done ? WAITING: (data ? SIX : FIVE);
      SIX:        nextstate = done ? WAITING: (data ? SEVEN : SIX);
      SEVEN:      nextstate = done ? WAITING: SEVEN;
      default :   nextstate = WAITING;
  endcase
end

//State Memory
always @ (posedge clk) begin
      state <= nextstate;
end

//Output Logic
assign X = state[2:0];

//Mod-7 counter
always @(posedge clk) begin
   if (start) count <= 0;
   else begin
      if (count == 3'b110)
            count <= 0;
      else
            count <= count +1;
   end
end

assign done = count[2] & count[1];

endmodule
```

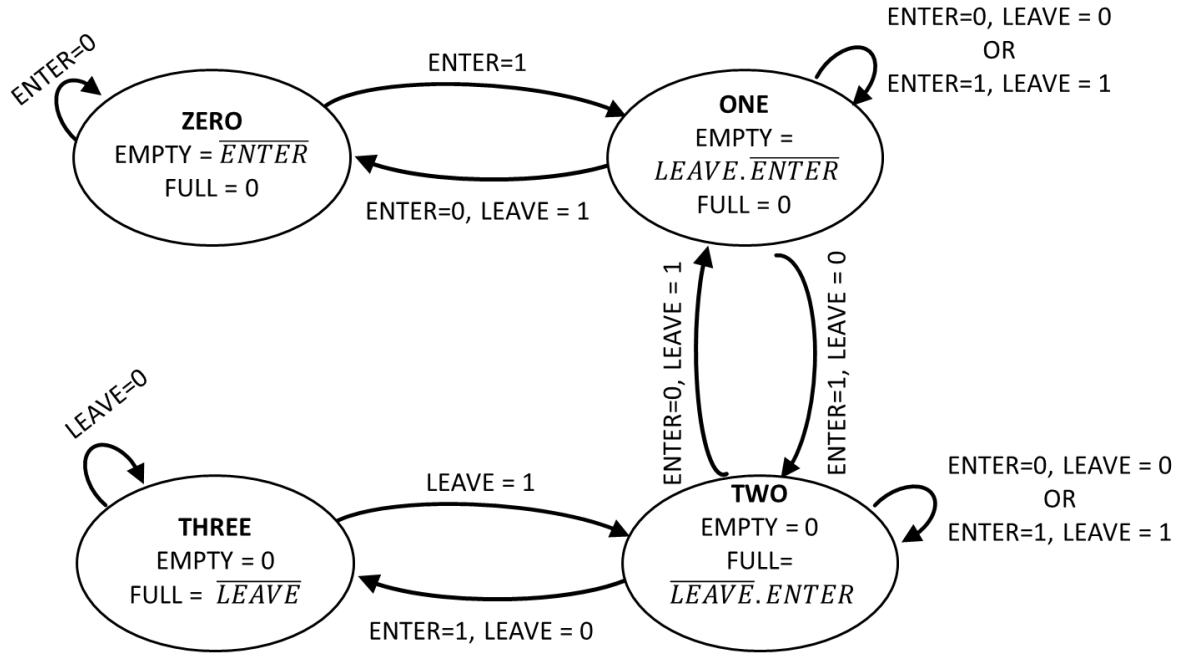
Verilog Simulation Results



Question 3

Overall Block Diagram :

State Transition Diagram of PPL_COUNTER FSM :



Verilog Code (For Reference Only)

```
module PPL_CNT (input clk, enter, leave, output full, empty);
reg [1:0] state, nextstate;

parameter ZERO = 2'b00, ONE = 2'b01, TWO = 2'b10; THREE = 2'b11;

always @ (*) begin
  case (state)
      ZERO:       nextstate = enter ? ONE : ZERO;
      ONE:        nextstate = enter ? (leave ? : ONE : TWO)
                                    : (leave ? : ZERO : ONE);
      TWO:        nextstate = enter ? (leave ? : TWO : THREE)
                                    : (leave ? : ONE : TWO);
      THREE:      nextstate = leave ? TWO : THREE;
      default :   nextstate = ZERO;
  endcase
end

always@(posedge clk)
      state <= nextstate;

assign empty = ((state == ZERO) & ~enter) | (state == ONE) & (~enter &
leave));
assign full  = ((state == TWO) & (enter & ~leave)) | (state == THREE) &
~leave);

endmodule
```