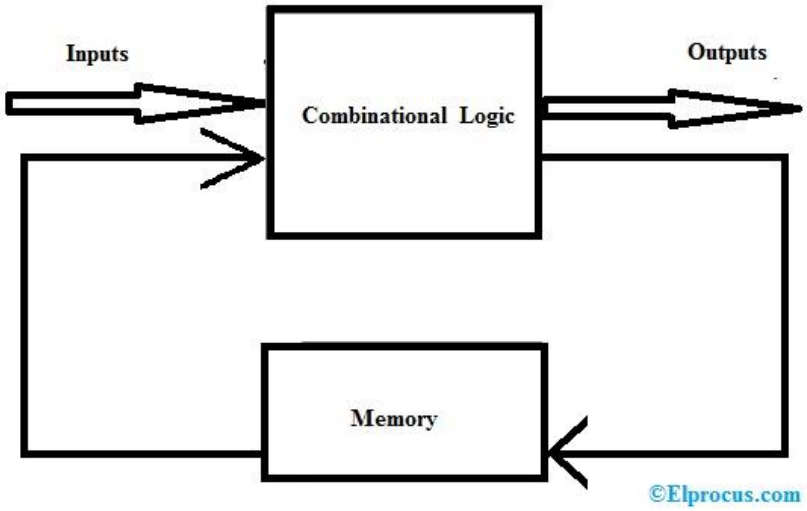
LAB 14
OPEN ENDED LAB

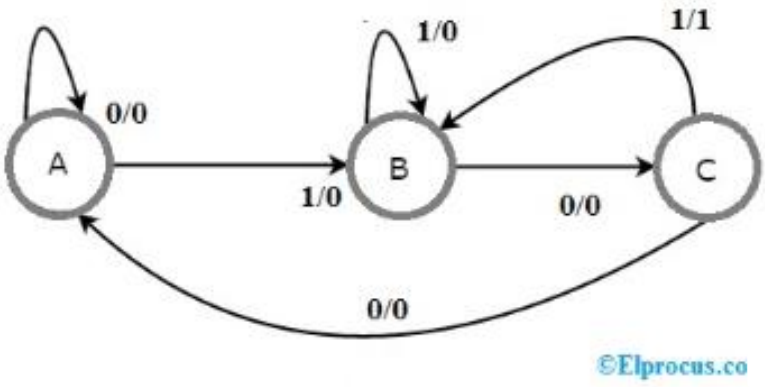
OBJECTIVE:
To design, implement and simulate different finite state diagrams (Combinations) using mealy machine concept.

HARDWARE / SOFTWARE REQUIRED:
HARDWARE: The mealy state machine block diagram consists of two parts namely combinational logic as well as memory. The memory in the machine can be used to provide some of the previous outputs as combinational logic inputs.



Mealy State Machine Block Diagram
SOFTWARE: We have used a software Xilinx 14.7 for performing this task.

DIAGRAM 01:



State Diagram of Mealy State Machine

METHODOLOGY:
Based on the current inputs as well as states, this machine can produce outputs. Thus, the outputs can be suitable only at positive otherwise negative of the CLK signal.
The above state diagram of mealy state machine mainly includes three states namely A, B, and C. These three states are tagged within the circles as well as every circle communicates with one state. Conversions

among these three states are signified by directed lines. In the above diagram, the inputs and outputs are denoted with 0/0, 1/0, and 1/1. Based on the input value, there are two conversions from every state.

How can we choose between Mealy state machine & Moore state machine :

Generally, the amount of required states in the mealy machine is below or equivalent to the number of required states in Moore state machine. There is an equal Moore state machine for every Mealy state machine. As a result, based on the necessity we can employ one of them.

SOURCE CODE:

Verilog Module Code:

```
//Umer(CE-118-2020)
module VendingMealyMachine(open,Clk,Reset,b);
    output open;
    input Clk;
    input Reset;
    input b;

    reg open,next_open;
    reg [1:0]state;
    reg [1:0]next_state;

    parameter  [1:0]A = 2'b00;
    parameter [1:0]B =2'b01;
    parameter [1:0]C =2'b10;


    always@(b or state)
    begin
    case(state)

            A:begin
               if(b) begin
                        next_state=B;
                        next_open=0;
                     end
               else  begin
                        next_state=A;
                        next_open=0;
                     end

               end

            B:begin
               if(b) begin
                        next_state=B;
                        next_open=0;
                     end
               else  begin
                        next_state=C;
                        next_open=0;
                     end
                end

            C:begin
               if(b) begin
                        next_state=B;
                        next_open=1;
                     end
               else  begin
                        next_state=0;
                        next_open=0;
                     end
                end
   endcase
   end
```

```
58      always @(posedge Clk)
59      begin
60          if(Reset )          //|| (!N && !D)) s
61          begin
62              state <= A;
63              open<=0;
64          end
65
66          else begin
67              state<=next_state;
68              open<=next_open;
69          end
70      end
71  endmodule
```

Verilog Test Fixture Code:

```
1   module SD_tf;
2
3       // Inputs
4       reg Clk;
5       reg Reset;
6       reg b;
7
8       // Outputs
9       wire open;
10
11      // Instantiate the Unit Under Test (UUT)
12      VendingMealyMachine uut (
13          .open(open),
14          .Clk(Clk),
15          .Reset(Reset),
16          .b(b)
17      );
18
19  parameter PERIOD=100;
20
21      always
22      begin
23
24          Clk=1;
25          #(PERIOD/2);
26
27          Clk=0;
28          #(PERIOD/2);
```

```
29      end
30
31      initial begin
32          // Initialize Inputs
33
34          Reset = 1;
35          b = 0;
36
37          // Wait 100 ns for global reset to finish
38          // Add stimulus here
39
40          #100;
41
42          Reset = 0;
43          b = 1;
44          #100;
45
46
47          b = 1;
48          #100;
49
50
51          b = 1;
52          #100;
53
54          b = 0;
55          #100;
56
57
58           b = 1;
59           #100;
60
61           b = 1;
62           #100;
63
64           b = 0;
65           #100;
66
67           b = 1;
68           #100;
69
70           b = 1;
71           #100;
72
73       end
74
75  endmodule
```

Result / Wave Form:

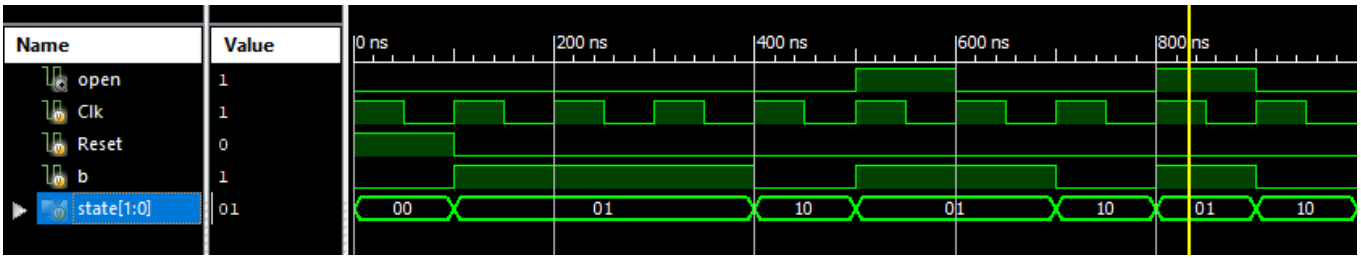
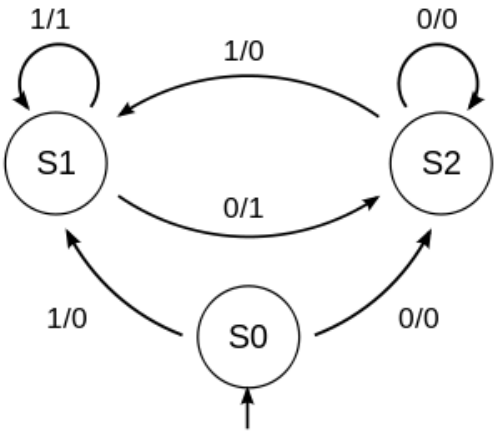


DIAGRAM 02:



METHODOLOGY:

Based on the current inputs as well as states, this machine can produce outputs. Thus, the outputs can be suitable only at positive otherwise negative of the CLK signal.

The above state diagram of mealy state machine mainly includes three states namely S0, S1, and S2. These three states are tagged within the circles as well as every circle communicates with one state. Conversions among these three states are signified by directed lines. In the above diagram, the inputs and outputs are denoted with 0/0, 0/1,1/0, and 1/1. Based on the input value, there are two conversions from every state.

SOURCE CODE:

Verilog Module Code:

```
2  //Umer(CE-118-2020)
3  module VendingMealyMachine(open,Clk,Reset,b);
4      output open;
5      input Clk;
6      input Reset;
7      input b;
8
9      reg open,next_open;
10     reg [1:0]state;
11     reg [1:0]next_state;
12
13     parameter  [1:0]s0 = 2'b00;
14     parameter [1:0]s1 =2'b01;
15     parameter [1:0]s2 =2'b10;
16
17
18     always@(b or state)
19     begin
```

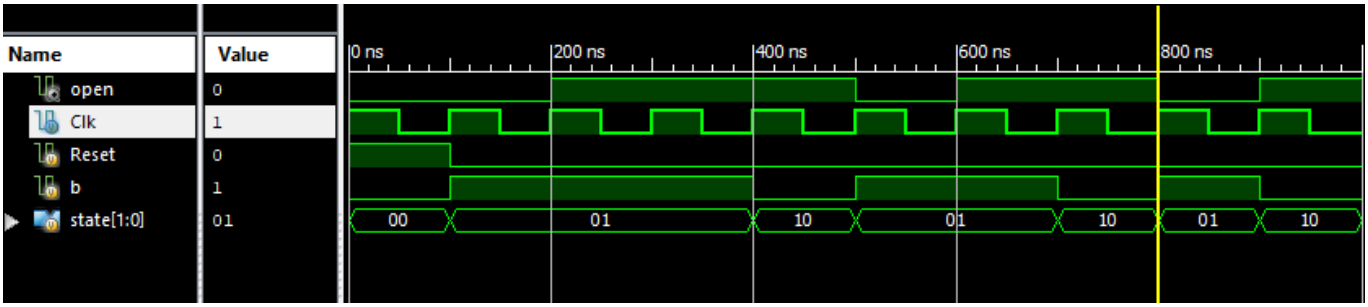
```
20      case(state)
21
22              s0:begin
23                 if(b) begin
24                          next_state=s1;
25                          next_open=0;
26                        end
27                 else  begin
28                          next_state=s2;
29                          next_open=0;
30                        end
31
32                  end
33
34              s1:begin
35                 if(b) begin
36                          next_state=s1;
37                          next_open=1;
38                        end
39                 else  begin
40                          next_state=s2;
41                          next_open=1;
42                        end
43                  end
44
45              s2:begin
46                 if(b) begin
47                          next_state=s1;
48                          next_open=0;
49                        end
50                 else  begin
51                          next_state=s2;
52                          next_open=0;
53                        end
54                  end
55     endcase
56     end
57
58     always @(posedge Clk)
59     begin
60        if(Reset )          //|| (!N && !D)) s
61        begin
62           state <=s0;
63           open<=0;
64        end
65
66        else begin
67           state<=next_state;
68           open<=next_open;
69        end
70     end
71  endmodule
```

Verilog Test Fixture Code:

```
module SD_tf;

    // Inputs
    reg Clk;
    reg Reset;
    reg b;

    // Outputs
    wire open;

    // Instantiate the Unit Under Test (UUT)
    VendingMealyMachine uut (
        .open(open),
        .Clk(Clk),
        .Reset(Reset),
        .b(b)
    );

parameter PERIOD=100;

   always
   begin

      Clk=1;
      #(PERIOD/2);

      Clk=0;
      #(PERIOD/2);
   end

   initial begin
      // Initialize Inputs

      Reset = 1;
      b = 0;

      // Wait 100 ns for global reset to finish
      // Add stimulus here

      #100;

      Reset = 0;
      b = 1;
      #100;

      b = 1;
      #100;
```

```
49          b = 1;
50          #100;
51
52          b = 0;
53          #100;
54
55
56          b = 1;
57          #100;
58
59          b = 1;
60          #100;
61
62          b = 0;
63          #100;
64
65          b = 1;
66          #100;
67
68          b = 0;
69          #100;
70
71      end
72
73  endmodule
```

Result / Wave Form:



CONCLUSION:

From this open ended lab we learned how to design, implement and simulate different finite state diagrams (combinations) using mealy machine concept.

Using the ISE Design Suite, We also produced a wave output in which we can see the characteristics of the output after performing these tasks.

Implementing a Mealy machine can be a useful tool in designing and implementing a finite state machine for a specific application. It allows for the output of the machine to be dependent on both the current input and current state, which can be beneficial in certain situations where the output needs to be generated in real-time based on the current input. However, it is important to carefully consider the specific requirements and constraints of the application before deciding whether a Mealy machine is the appropriate solution. It may be necessary to evaluate the trade-offs and limitations of using a Mealy machine, such as the potential for increased complexity in the design and implementation, compared to other types of finite state machines. Overall, the decision to use a Mealy machine should be based on a thorough analysis of the specific needs of the application and the resources available for implementation.