VLSI Design
Practice Sheet: Verilog

M.P. Samartha (2023102038)

October 8, 2024

Q9: Sequential Circuits

a) Sequential circuit with given state and output equations

We are given a sequential circuit with 2 flip-flops, two inputs and one output. Here we have two states A and B. Two inputs x and y, and one output z. The state equations linking all of them is given as follows.

$$A(t+1) = xy' + xB$$
$$B(t+1) = xA + xB'$$
$$z = A$$

The following is the circuit diagram for the given set of state equations. Here notice that a combinational circuit as a function of A, B and the inputs x *and* y has been constructed to feed in the input to the two D flipflops for the next clock cycle.

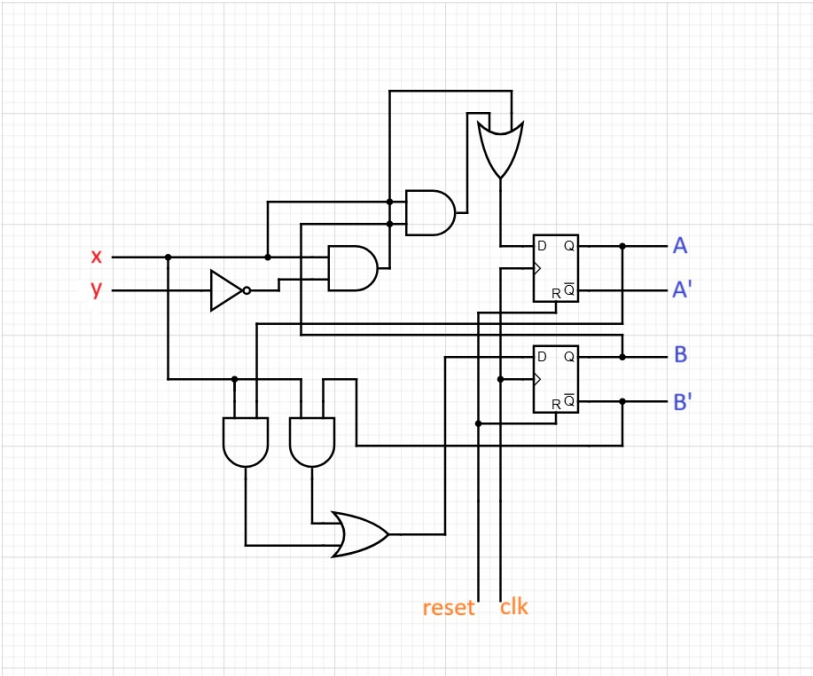

Figure 1: Circuit Diagram for 9(a)

The following is the state diagram for this circuit. Here we have assumed $\mathbf{S_0 = 00}, \mathbf{S_1 = 01}, \mathbf{S_2 = 10}, \mathbf{S_3 = 11}$.

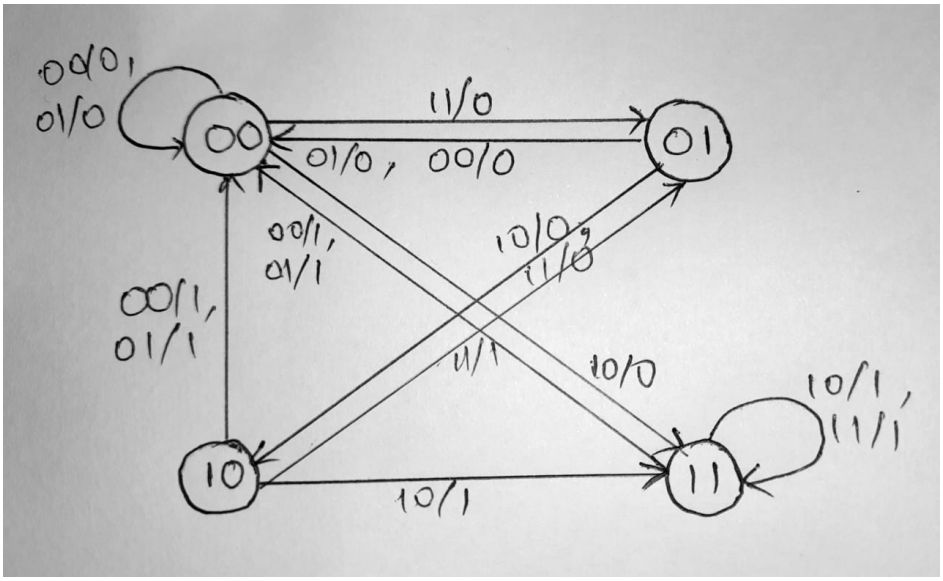

Figure 2: State diagram for 9(a)

The following is the state table for this circuit.

Present State		Input		Next State		Output
A	B	x	y	A	B	z
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	1	1
1	0	1	1	0	1	1
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

Table 1: State Table for 9(a)

```verilog
`timescale 1ns/1ns

// Reset is active high
module D_flipflop(input d, input clk, input reset, output reg q, output reg qbar);
    always @(posedge clk or posedge reset)
    begin
        if (reset) begin
            q <= 1'b0;
            qbar <= 1'b1;
        end
        else begin
            q <= d;
            qbar <= ~d;
        end
    end
endmodule

// A(t+1) = x.y' + x.B
// B(t+1) = x.A + x.B'
// z = A
module q9a(input clk, reset, x, y, output A, B, z);
    wire A, B, Abar, Bbar;
    // reg in1, in2;
    not(noty, y);
    and(temp1, x, noty);
    and(temp2, x, B);
    and(temp3, x, A);
    and(temp4, x, Bbar);

    or(in1, temp1, temp2);
    or(in2, temp3, temp4);

    D_flipflop dff1(in1, clk, reset, A, Abar);
    D_flipflop dff2(in2, clk, reset, B, Bbar);

    assign z = A;
endmodule
```

Figure 3: Verilog Design code for 9(a)

Above is the Verilog design code for this subsection. The design contains two modules. Here we have followed structural modelling, and hence are describing the components of the circuit too, not just its behaviour. The first one is a module for a *D flipflop*. It has inputs *clk, reset* and outputs *q, qbar*. *qbar* is the complement of the output q. The reset is set to be active high and the flipflop is designed to operate on the positive edge of the clock. The second module, which is the one of interest has the description for the circuit and the first few lines implement the combinational logic required for the inputs to the flipflop. The following lines instantiate the two flipflops required in this question by passing respective inputs as given by the combinational circuit. The last line finally assigns the value to the output which by the given equations is simply the state A.

Below is the verilog testbench code for 9(a). First we initialize the required input and output elements for the circuit. Then we use the *forever* statement to set the clock alternation every 5ns. Then as seen in lines 20-24, we set the initial conditions for the circuit. Without this the output of the flipflops would be in *Don't care condition/ High Impedence*. We initially set both the inputs

to zero. Following this, we run a second order nested for-loop to iterate over all possible input combinations for this circuit. Finally we end the code with the monitor statement which prints the values of interest in the terminal as shown below.

```
1  `timescale 1ns/1ns
2  `include "q9a_design.v"
3
4  module q9a_test();
5      reg clk, reset, x, y;
6      wire A, B, z;
7
8      q9a uut(clk, reset, x, y, A, B, z);
9
10     initial begin
11         clk = 0;
12         forever begin
13             #5 clk = ~clk;
14         end
15     end
16
17     initial begin
18         $dumpfile("q9a.vcd");
19         $dumpvars(0, q9a_test);
20         reset =1;
21         x = 0;
22         y = 0;
23         #10;
24         reset = 0;
25
26         for (integer i = 0;i< 2;i = i+1)
27         begin
28             for (integer j = 0;j< 2;j = j+1)
29             begin
30                 x = i;
31                 y = j;
32                 #10;
33             end
34         end
35         $finish;
36     end
37
38     initial begin
39         $monitor("Time = %0t: clk = %b, reset = %b, A = %b, B = %b, x = %b, y = %b, z = %b",
40         $time, clk, reset, A, B, x, y, z);
41     end
42 endmodule
43
```

Figure 4: Verilog Testbench code for 9(a)

```
VCD info: dumpfile q9a.vcd opened for output.
Time = 0: clk = 0, reset = 1, A = 0, B = 0, x = 0, y = 0, z = 0
Time = 5: clk = 1, reset = 1, A = 0, B = 0, x = 0, y = 0, z = 0
Time = 10: clk = 0, reset = 0, A = 0, B = 0, x = 0, y = 0, z = 0
Time = 15: clk = 1, reset = 0, A = 0, B = 0, x = 0, y = 0, z = 0
Time = 20: clk = 0, reset = 0, A = 0, B = 0, x = 0, y = 1, z = 0
Time = 25: clk = 1, reset = 0, A = 0, B = 0, x = 0, y = 1, z = 0
Time = 30: clk = 0, reset = 0, A = 0, B = 0, x = 1, y = 0, z = 0
Time = 35: clk = 1, reset = 0, A = 1, B = 1, x = 1, y = 0, z = 1
Time = 40: clk = 0, reset = 0, A = 1, B = 1, x = 1, y = 1, z = 1
Time = 45: clk = 1, reset = 0, A = 1, B = 1, x = 1, y = 1, z = 1
Time = 50: clk = 0, reset = 0, A = 1, B = 1, x = 1, y = 1, z = 1
mpsamartha@Samartha:~/Verilog/Practice_Sheet/Q9_Sequential_Circuits$
```

Figure 5: Terminal output of 9(a)

The following is the Timing diagram for this subsection featuring the clk, reset, x, y, A, B and z. The validity of the Verilog code can be verified from this diagram.

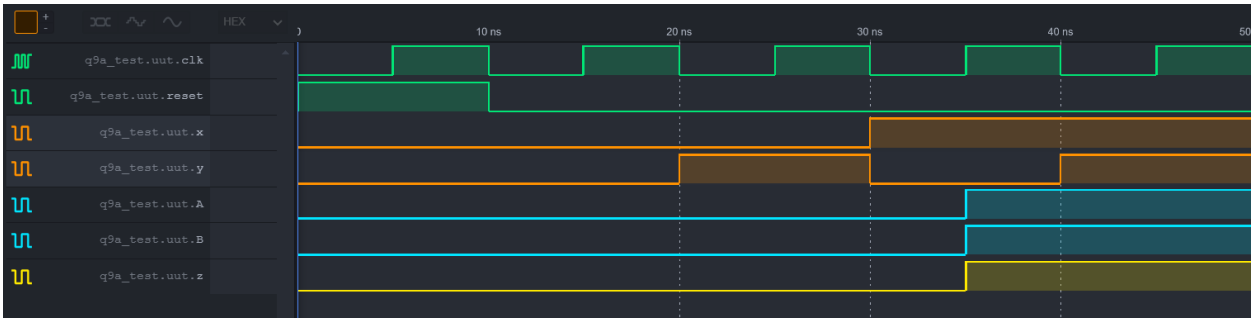

Figure 6: Timing diagram for 9(a)

b) Analysing the given Sequential Circuit

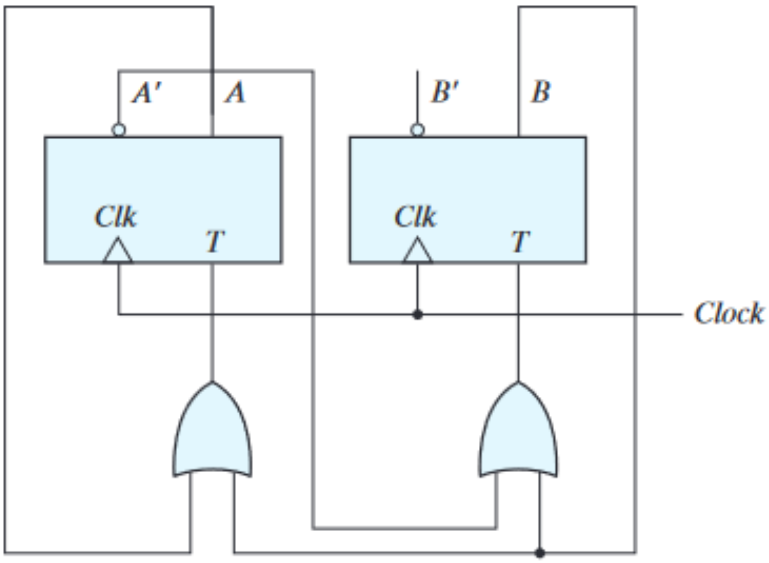

Figure 7: Given Circuit diagram for 9(b)

Above is the circuit diagram given in this subsection. Here since nothing is mentioned about the output for the circuit, we simply declare the outputs of the T-flipflops viz. A and B as outputs and handle the same in the testbench. By the combinational part of the circuit, we make the following state equations:

$$A(t+1) = (A+B)' = A'.B'$$
$$B(t+1) = (A'+B)' = A.B'$$

The following is the state table for the given circuit. The equations derived above are used for deriving the table.

Present State		Inputs to FF		Next State	
A	B	x	y	A	B
0	0	0	1	0	1
0	1	1	1	1	0
1	0	1	0	0	0
1	1	1	1	0	0

Table 2: State Table for 9(b)

```
1  `timescale 1ns/1ns
2
3  // Reset is Active High
4  module q9b(input clk, input reset, output reg A, B);
5      always @(posedge clk or posedge reset)
6      begin
7          if (reset) begin
8              A <= 1'b0;
9              B <= 1'b0;
10         end
11         else begin
12             if ((~A) & (~B)) begin
13                 A <= 1'b0;
14                 B <= 1'b1;
15             end
16             else if ((~A) & B) begin
17                 A <= 1'b1;
18                 B <= 1'b0;
19             end
20             else if (A & (~B)) begin
21                 A <= 1'b0;
22                 B <= 1'b0;
23             end
24             else if (A & B) begin
25                 A <= 1'b0;
26                 B <= 1'b0;
27             end
28         end
29     end
30 endmodule
31
```

Figure 8: Verilog Design code for 9(b)

Above is the design code for this subsection. In this module, we have inputs of clk, reset and outputs A and B, the outputs of the corresponding flipflops. Again the circuit is designed to operate on the positive edge of the clock and the reset is active high. Here we follow the behavioural modelling style, thus we brute force the next states according to the current states by the *if-else if-else* blocks.

```
1  `timescale 1ns/1ns
2  `include "q9b_design.v"
3
4  module q9b_test();
5      reg clk, reset;
6      wire A, B;
7
8      q9b uut(clk, reset, A, B);
9
10     initial begin
11         clk = 0;
12         forever begin
13             #5 clk = ~clk;
14         end
15     end
16
17     initial begin
18         $dumpfile("q9b.vcd");
19         $dumpvars(0, q9b_test);
20         reset =1;
21         #10;
22         reset = 0;
23
24         for (integer i = 0;i< 6;i = i+1)
25         begin
26             #10;
27         end
28         $finish;
29     end
30
31     initial begin
32         $monitor("Time = %0t: clk = %b, reset = %b, A = %b, B = %b",
33         $time, clk, reset, A, B);
34     end
35
36 endmodule
```

Figure 9: Verilog testbench code for 9(b)

Above is the testbench for this subsection. Similar to the previous subsection, we initialise the clock inside the forever statement and set the time delay between the rise/fall as 5ns. As explained in the previous subsection, we initially assert *reset* high to set the flipflops' outputs to some initial condition (0,0). This ensures that the feedback needed for calculating the inputs to the FF is not indeterminate. As will be discussed in the next page, this circuit goes through states *00, 01 and 10* and then starts all over again at state 00. Thus to show this behaviour we run the for loop for 6 iterations with a delay of 10ns between each iteration. Then using the *$monitor* statement we print the desired values on the terminal at each time step.

The following is the state diagram for this subsection.

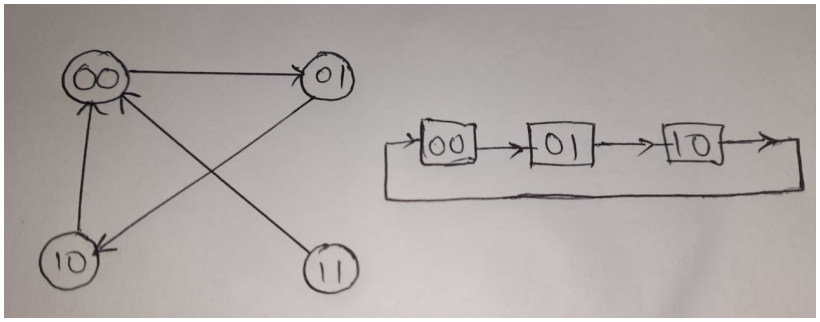

Figure 10: State diagram for 9(b)

The following is the output obtained on the terminal at respective time step.

```
VCD info: dumpfile q9b.vcd opened for output.
Time = 0: clk = 0, reset = 1, A = 0, B = 0
Time = 5: clk = 1, reset = 1, A = 0, B = 0
Time = 10: clk = 0, reset = 0, A = 0, B = 0
Time = 15: clk = 1, reset = 0, A = 0, B = 1
Time = 20: clk = 0, reset = 0, A = 0, B = 1
Time = 25: clk = 1, reset = 0, A = 1, B = 0
Time = 30: clk = 0, reset = 0, A = 1, B = 0
Time = 35: clk = 1, reset = 0, A = 0, B = 0
Time = 40: clk = 0, reset = 0, A = 0, B = 0
Time = 45: clk = 1, reset = 0, A = 0, B = 1
Time = 50: clk = 0, reset = 0, A = 0, B = 1
Time = 55: clk = 1, reset = 0, A = 1, B = 0
Time = 60: clk = 0, reset = 0, A = 1, B = 0
Time = 65: clk = 1, reset = 0, A = 0, B = 0
Time = 70: clk = 0, reset = 0, A = 0, B = 0
mpsamartha@Samartha:~/Verilog/Practice_Sheet/Q9_Sequential_Circuits$
```

Figure 11: Terminal output for 9(b)

[H] The following is the Timing diagram for this subsection featuring the clk, reset, A, B. The validity of the Verilog code can be verified from this diagram.

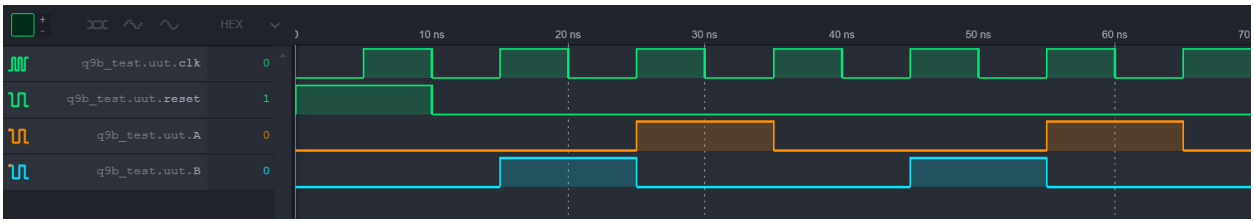

Figure 12: Timing diagram for 9(b)

Q10: Finite State Machines

In this section, we implement the two different machines possible for sequential circuits viz. Moore FSM and Mealy FSM. Both are often implemented in digital circuits but each one of them have their own positives-negatives. The fundamental difference between lies in the dependency of the output on the inputs. This fundamental difference gives rise to different properties among the two FSMs. The following table gives a concise tabulation of the differences between the two FSM in various aspects of operation such as speed, timing, glitch sensitivity and so on.

Characteristic	**Moore FSM**	**Mealy FSM**
Output Dependency	Depends only on current state	Depends on current state and input
Output Timing	Synchronous with state transitions	Can change asynchronously with inputs
State Diagram Representation	Outputs associated with states	Outputs associated with transitions
Reaction Speed	Generally slower	Can be faster
Number of States	Often requires more states	Typically requires fewer states
Complexity	Generally simpler to design and analyze	Can be more complex but more flexible
Glitch Sensitivity	Less sensitive to input glitches	More sensitive to input glitches

Table 3: Comparison of Moore and Mealy Finite State Machines

In the following bullet points, we discuss each aspect of the difference between the FSM in detail.

- The output dependency is the fundamental difference between the FSM and is solely because of the way the FSMs are designed.
- The Moore FSM is synchronous with state transitions, whereas Mealy FSM isn't. This is again a consequence of the fact that the output in a Mealy machine is dependent also on the inputs. Thus the outputs immediately change, without requiring any clock edge. But in the Moore FSM, since the outputs are only dependent on the current state, they can only change when a positive/negative clock edge (depending on the design) comes. This make Moore FSM more **reliable** than its counterpart because the change in outputs is deterministic in the sense that they change only during one of the edges of the clock.
- This again is because of the fundamental difference between them. In the state diagram of a Moore FSM, the outputs are mentioned inside the state bubble, since they are a characteristic of a respective state. But in the state diagram of a Mealy FSM, the outputs are mentioned along with the corresponding inputs on the transition arrow between the states.
- Since the outputs in Mealy are dependent on inputs too, they can be carefully designed to have the correct outputs for some input combination. Since the inputs have to go through the flipflops to cause a change in the outputs and wait for a clock edge, they are relatively slower than the former.
- Generally Moore FSM require more states because to represent any unique output combination requires a distinct state, as the output is tied solely to the state. But in the case of

Mealy FSM, different outputs can be produced from the same state based on different inputs. This allows one state to serve multiple purposes depending on the input.

- Since the output is just a function of the states, the designing is simple and straightforward compared to the Mealy FSM.
- This is an important point to consider. Since the output of the Mealy FSM depend asynchronously on the inputs, any variations or fluctuations in the input can cause the equivalent disturbance in the output signal. Thus they are more sensitive to input *glitches*.

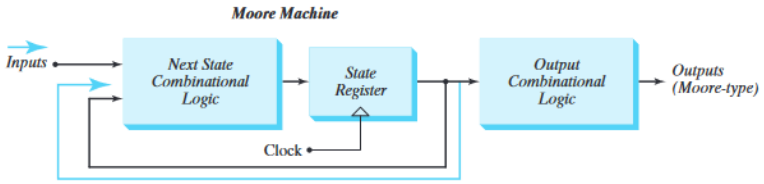

Figure 13: Illustration for Moore FSM (*Credits: D.D. by M. Mano*)

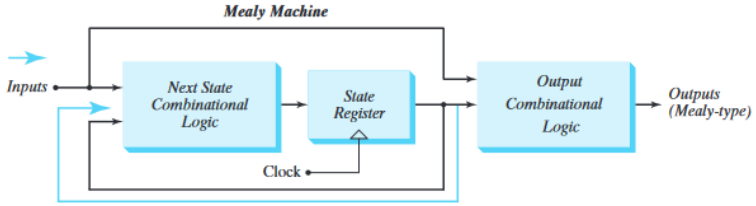

Figure 14: Illustration for Mealy FSM (*Credits: D.D. by M. Mano*)

a) Moore FSM

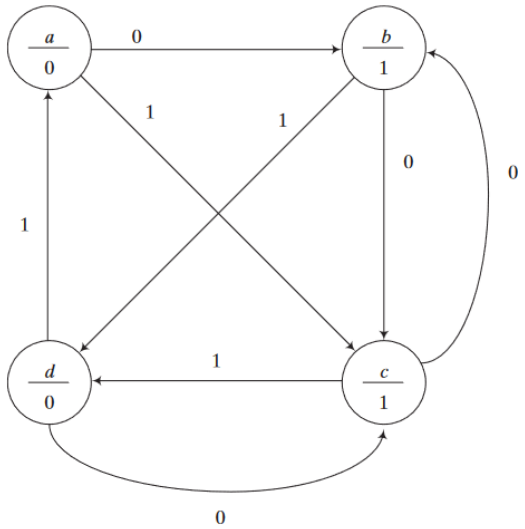

Figure 15: State diagram for Moore FSM

```verilog
`timescale 1ns/1ns

// Reset is Active High
module moore_fsm(input clk, reset, input x, output reg y,
output reg [1:0] state);
    reg [1:0] next_state;

    parameter S_a = 2'b00, S_b = 2'b01, S_c = 2'b10, S_d = 2'b11;

    always @(posedge clk or posedge reset)
    begin
        if (reset) begin
            state <= S_a;
            next_state <= S_a;
            y <= 1'b0;
        end
        else begin
            state <= next_state;
            case (state)
                S_a: begin
                    if (x==1)
                        next_state <= S_c;
                    else
                        next_state <= S_b;
                    y <= 1'b0;
                end
                S_b: begin
                    if (x==1)
                        next_state <= S_d;
                    else
                        next_state <= S_c;
                    y <= 1'b1;
                end
                S_c: begin
                    if (x==1)
                        next_state <= S_d;
                    else
                        next_state <= S_b;
                    y <= 1'b1;
                end
                S_d: begin
                    if (x==1)
                        next_state <= S_a;
                    else
                        next_state <= S_c;
                    y <= 1'b0;
                end
            endcase
        end
    end
endmodule
```

Figure 16: Verilog design code for Moore FSM

Above is the Verilog design code for Moore FSM. The module contains inputs clk, reset and x. The

outputs are y and register state. Here we follow the behavioural modelling for the design. the logic uses another register called *next_state*, which is used to store the value for the next state depending upon the current states and inputs. We use *parameter* to define the four states as *00, 01, 10, and 11*. Then at the positive edge of a clock or reset(used to reset the high impedence condition at the beginning), the state is assigned the value of next_state. Then the next_state is updated depending on the current state and the input x. This is done efficiently using a *case* block. Thus corresponding to each of the 4 states and the input x, we set the next_state its value. In each case statement, we correspondingly update the output value as given by the state diagram.

```
`timescale 1ns/1ns
`include "q10_moore_fsm_design.v"

module moore_fsm_tb();
    reg clk, reset, x;
    wire [1:0] state;
    wire y;

    moore_fsm uut(
        .clk(clk),
        .reset(reset),
        .x(x),
        .state(state),
        .y(y)
    );

    initial begin
        clk = 0;
        forever begin
            #5 clk = ~clk;
        end
    end

    initial begin
        $dumpfile("q10_moore_fsm.vcd");
        $dumpvars(0, moore_fsm_tb);

        reset =1;
        x = 0;
        #10
        reset = 0;
        x = 0; #20;
        x = 0; #20;
        x = 1; #20;
        x = 1; #20; // back to state 0
        $display("Back to state a");
        x = 1; #20;
        x = 1; #20;
        x = 1; #20; // back to state 0
        $display("Back to state a");
        x = 0; #20;
        x = 1; #20;
        x = 1; #30;
        $display("Back to state a");
        $finish;
    end

    initial begin
        $monitor("Time = %0t: clk = %b, reset = %b, x = %b, state = %b , y = %b",
        $time, clk, reset, x, state , y);
    end
endmodule
```

Figure 17: Verilog Testbench code for Moore FSM

Above is the Verilog testbench code for the Moore FSM. Initially we use a *forever* statement to set the switching of the clock throughout the testing. The we run three sets of inputs with 20ns delay between each. Here instead of 10ns, we have chosen 20ns because of the logic used in the design code. The next_state is calculated at the positive edge of clock, thus we hold the state of the FSM for one extra clock cycle so that in the following clock edge the corresponding value for the state is set correctly. Then as can be observed in the testbench code above, we are testing our design for three set of values. First we input $x = 0, 0, 1, 1$. As can be seen in the state diagram, these set of inputs makes the states to go through *S_a, S_b, S_c, S_d, S_a.* Thus in a circular fashion we come back to the initial state, S_a. Similarly we run for input values' set, $x = 1, 1, 1$ and $x = 0, 1, 1$. To see the validity of the design and the testbench codes, we print *"Back to state a"* after completion of each cycle. Finally we print the truth table on the terminal using the *$monitor* statement.

```
VCD info: dumpfile q10_moore_fsm.vcd opened for output.
Time = 0: clk = 0, reset = 1, x = 0, state = 00 , y = 0
Time = 5: clk = 1, reset = 1, x = 0, state = 00 , y = 0
Time = 10: clk = 0, reset = 0, x = 0, state = 00 , y = 0
Time = 15: clk = 1, reset = 0, x = 0, state = 00 , y = 0
Time = 20: clk = 0, reset = 0, x = 0, state = 00 , y = 0
Time = 25: clk = 1, reset = 0, x = 0, state = 01 , y = 0
Time = 30: clk = 0, reset = 0, x = 0, state = 01 , y = 0
Time = 35: clk = 1, reset = 0, x = 0, state = 01 , y = 1
Time = 40: clk = 0, reset = 0, x = 0, state = 01 , y = 1
Time = 45: clk = 1, reset = 0, x = 0, state = 10 , y = 1
Time = 50: clk = 0, reset = 0, x = 1, state = 10 , y = 1
Time = 55: clk = 1, reset = 0, x = 1, state = 10 , y = 1
Time = 60: clk = 0, reset = 0, x = 1, state = 10 , y = 1
Time = 65: clk = 1, reset = 0, x = 1, state = 11 , y = 1
Time = 70: clk = 0, reset = 0, x = 1, state = 11 , y = 1
Time = 75: clk = 1, reset = 0, x = 1, state = 11 , y = 0
Time = 80: clk = 0, reset = 0, x = 1, state = 11 , y = 0
Time = 85: clk = 1, reset = 0, x = 1, state = 00 , y = 0
Back to state a
Time = 90: clk = 0, reset = 0, x = 1, state = 00 , y = 0
Time = 95: clk = 1, reset = 0, x = 1, state = 00 , y = 0
Time = 100: clk = 0, reset = 0, x = 1, state = 00 , y = 0
Time = 105: clk = 1, reset = 0, x = 1, state = 10 , y = 0
Time = 110: clk = 0, reset = 0, x = 1, state = 10 , y = 0
Time = 115: clk = 1, reset = 0, x = 1, state = 10 , y = 1
Time = 120: clk = 0, reset = 0, x = 1, state = 10 , y = 1
Time = 125: clk = 1, reset = 0, x = 1, state = 11 , y = 1
Time = 130: clk = 0, reset = 0, x = 1, state = 11 , y = 1
Time = 135: clk = 1, reset = 0, x = 1, state = 11 , y = 0
Time = 140: clk = 0, reset = 0, x = 1, state = 11 , y = 0
Time = 145: clk = 1, reset = 0, x = 1, state = 00 , y = 0
Back to state a
Time = 150: clk = 0, reset = 0, x = 0, state = 00 , y = 0
Time = 155: clk = 1, reset = 0, x = 0, state = 00 , y = 0
Time = 160: clk = 0, reset = 0, x = 0, state = 00 , y = 0
Time = 165: clk = 1, reset = 0, x = 0, state = 01 , y = 0
Time = 170: clk = 0, reset = 0, x = 1, state = 01 , y = 0
Time = 175: clk = 1, reset = 0, x = 1, state = 01 , y = 1
Time = 180: clk = 0, reset = 0, x = 1, state = 01 , y = 1
Time = 185: clk = 1, reset = 0, x = 1, state = 11 , y = 1
Time = 190: clk = 0, reset = 0, x = 1, state = 11 , y = 1
Time = 195: clk = 1, reset = 0, x = 1, state = 11 , y = 0
Time = 200: clk = 0, reset = 0, x = 1, state = 11 , y = 0
Time = 205: clk = 1, reset = 0, x = 1, state = 00 , y = 0
Time = 210: clk = 0, reset = 0, x = 1, state = 00 , y = 0
Time = 215: clk = 1, reset = 0, x = 1, state = 00 , y = 0
Back to state a
Time = 220: clk = 0, reset = 0, x = 1, state = 00 , y = 0
mpsamartha@Samartha:~/Verilog/Practice_Sheet/Q10_Finite_State_Machines$
```

Figure 18: Terminal output for Moore FSM

As seen above in the terminal output, we go through 3 different cycles of states as explained above. Thus we can see the print statements of being at state a and the corresponding state in the row above it. It matches (00) with the expected results.

The following is the state table derived from the state diagram 15.

Current State	Input	Next State	Output
a	0	b	0
a	1	c	0
b	0	c	1
b	1	d	1
c	0	b	1
c	1	d	1
d	0	c	0
d	1	a	0

Table 4: State Transition Table for Moore FSM

the following is the timing diagram for the Moore FSM. The explanation above about the three cycles can also be verified from the value of the state below in the diagram. The state goes through the three cycles, starting and ending at the initial state $S\_a = 00$.

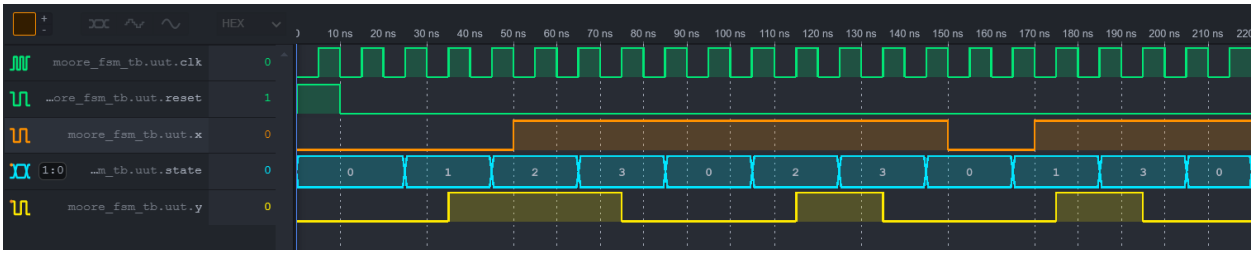

Figure 19: Timing Diagram for Moore FSM

b) Mealy FSM

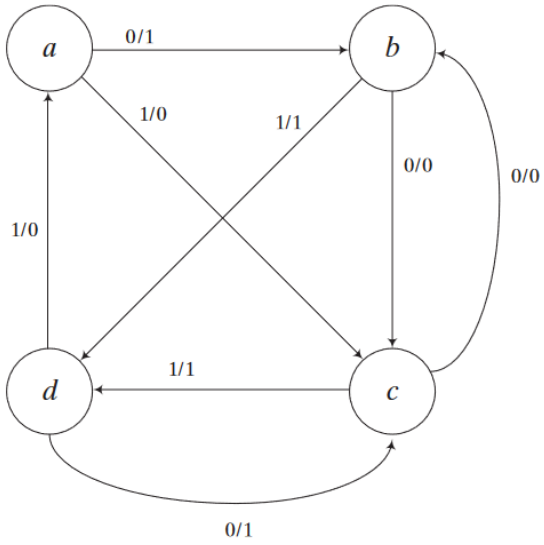

Figure 20: State diagram for Mealy FSM

```verilog
`timescale 1ns/1ns

// Reset is Active High
module mealy_fsm(input clk, reset, input x, output reg y, output reg [1:0] state);
    reg [1:0] next_state;

    parameter S_a = 2'b00, S_b = 2'b01, S_c = 2'b10, S_d = 2'b11;

    always @(posedge clk or posedge reset)
    begin
        if (reset) begin
            state <= S_a;
            next_state <= S_a;
            y <= 1'b0;
        end
        else begin
            state <= next_state;
            case (state)
                S_a: begin
                    if (x==1)
                    begin
                        next_state <= S_c;
                        y <= 1'b0;
                    end
                    else
                    begin
                        next_state <= S_b;
                        y <= 1'b1;
                    end
                end
                S_b: begin
                    if (x==1)
                    begin
                        next_state <= S_d;
                        y <= 1'b1;
                    end
                    else
                    begin
                        next_state <= S_c;
                        y <= 1'b0;
                    end
                end
                S_c: begin
                    if (x==1)
                    begin
                        next_state <= S_d;
                        y <= 1'b1;
                    end
                    else
                    begin
                        next_state <= S_b;
                        y <= 1'b0;
                    end
                end
                S_d: begin
                    if (x==1)
                    begin
                        next_state <= S_a;
                        y <= 1'b0;
                    end
                    else
                    begin
                        next_state <= S_c;
                        y <= 1'b1;
                    end
                end
            endcase
        end
    end
endmodule
```

Figure 21: Verilog design code for Mealy FSM

Above is the Verilog design code for Mealy FSM. The module contains inputs clk, reset and x. The outputs are y and register state. Here we follow the behavioural modelling for the design. the logic uses another register called *next_state*, which is used to store the value for the next state depending upon the current states and inputs. We use *parameter* to define the four states as *00, 01, 10, and 11*. Then at the positive edge of a clock or reset(used to reset the high impedence condition at the beginning), the state is assigned the value of next_state. Then the next_state is updated depending on the current state and the input x. This is done efficiently using a *case* block. Thus corresponding to each of the 4 states and the input x, we set the next_state its value. In each case statement, we correspondingly update the output value as given by the state diagram. Here is output is dependent on both input value and the current state.

```
`timescale 1ns/1ns
`include "q10_mealy_fsm_design.v"

module mealy_fsm_tb();
    reg clk, reset, x;
    wire [1:0] state;
    wire y;

    mealy_fsm uut(
        .clk(clk),
        .reset(reset),
        .x(x),
        .y(y),
        .state(state)
    );

    initial begin
        clk = 0;
        forever begin
            #5 clk = ~clk;
        end
    end

    initial begin
        $dumpfile("q10_mealy_fsm.vcd");
        $dumpvars(0, mealy_fsm_tb);

        reset =1;
        x = 0;
        #10
        reset = 0;

        x = 0; #20;
        x = 0; #20;
        x = 1; #20;
        x = 1; #20; // back to state 0
        $display("Back to state a");
        x = 1; #20;
        x = 1; #20;
        x = 1; #20; // back to state 0
        $display("Back to state a");
        x = 0; #20;
        x = 1; #20;
        x = 1; #30;
        $display("Back to state a");
        $finish;
    end

    initial begin
        $monitor("Time = %0t: clk = %b, reset = %b, x = %b, state = %b , y = %b", $time, clk, reset, x, state , y);
    end
endmodule
```

Figure 22: Verilog Testbench code for Mealy FSM

Above is the Verilog testbench code for the Mealy FSM. Initially we use a *forever* statement to set the switching of the clock throughout the testing. The we run three sets of inputs with 20ns delay between each. Here instead of 10ns, we have chosen 20ns because of the logic used in the design code. The next_state is calculated at the positive edge of clock, thus we hold the state of the FSM for one extra clock cycle so that in the following clock edge the corresponding value for the state is set correctly. Then as can be observed in the testbench code above, we are testing our

design for three set of values. First we input *x = 0, 0, 1, 1.* As can be seen in the state diagram, these set of inputs makes the states to go through *S_a, S_b, S_c, S_d, S_a.* Thus in a circular fashion we come back to the initial state, S_a. Similarly we run for input values' set, *x = 1, 1, 1* and *x = 0, 1, 1.* To see the validity of the design and the testbench codes, we print *"Back to state a"* after completion of each cycle. Finally we print the truth table on the terminal using the *$monitor* statement.

```
VCD info: dumpfile q10_mealy_fsm.vcd opened for output.
Time = 0: clk = 0, reset = 1, x = 0, state = 00 , y = 0
Time = 5: clk = 1, reset = 1, x = 0, state = 00 , y = 0
Time = 10: clk = 0, reset = 0, x = 0, state = 00 , y = 0
Time = 15: clk = 1, reset = 0, x = 0, state = 00 , y = 1
Time = 20: clk = 0, reset = 0, x = 0, state = 00 , y = 1
Time = 25: clk = 1, reset = 0, x = 0, state = 01 , y = 1
Time = 30: clk = 0, reset = 0, x = 0, state = 01 , y = 1
Time = 35: clk = 1, reset = 0, x = 0, state = 01 , y = 0
Time = 40: clk = 0, reset = 0, x = 0, state = 01 , y = 0
Time = 45: clk = 1, reset = 0, x = 0, state = 10 , y = 0
Time = 50: clk = 0, reset = 0, x = 1, state = 10 , y = 0
Time = 55: clk = 1, reset = 0, x = 1, state = 10 , y = 1
Time = 60: clk = 0, reset = 0, x = 1, state = 10 , y = 1
Time = 65: clk = 1, reset = 0, x = 1, state = 11 , y = 1
Time = 70: clk = 0, reset = 0, x = 1, state = 11 , y = 1
Time = 75: clk = 1, reset = 0, x = 1, state = 11 , y = 0
Time = 80: clk = 0, reset = 0, x = 1, state = 11 , y = 0
Time = 85: clk = 1, reset = 0, x = 1, state = 00 , y = 0
Back to state a
Time = 90: clk = 0, reset = 0, x = 1, state = 00 , y = 0
Time = 95: clk = 1, reset = 0, x = 1, state = 00 , y = 0
Time = 100: clk = 0, reset = 0, x = 1, state = 00 , y = 0
Time = 105: clk = 1, reset = 0, x = 1, state = 10 , y = 0
Time = 110: clk = 0, reset = 0, x = 1, state = 10 , y = 0
Time = 115: clk = 1, reset = 0, x = 1, state = 10 , y = 1
Time = 120: clk = 0, reset = 0, x = 1, state = 10 , y = 1
Time = 125: clk = 1, reset = 0, x = 1, state = 11 , y = 1
Time = 130: clk = 0, reset = 0, x = 1, state = 11 , y = 1
Time = 135: clk = 1, reset = 0, x = 1, state = 11 , y = 0
Time = 140: clk = 0, reset = 0, x = 1, state = 11 , y = 0
Time = 145: clk = 1, reset = 0, x = 1, state = 00 , y = 0
Back to state a
Time = 150: clk = 0, reset = 0, x = 0, state = 00 , y = 0
Time = 155: clk = 1, reset = 0, x = 0, state = 00 , y = 1
Time = 160: clk = 0, reset = 0, x = 0, state = 00 , y = 1
Time = 165: clk = 1, reset = 0, x = 0, state = 01 , y = 1
Time = 170: clk = 0, reset = 0, x = 1, state = 01 , y = 1
Time = 175: clk = 1, reset = 0, x = 1, state = 01 , y = 1
Time = 180: clk = 0, reset = 0, x = 1, state = 01 , y = 1
Time = 185: clk = 1, reset = 0, x = 1, state = 11 , y = 1
Time = 190: clk = 0, reset = 0, x = 1, state = 11 , y = 1
Time = 195: clk = 1, reset = 0, x = 1, state = 11 , y = 0
Time = 200: clk = 0, reset = 0, x = 1, state = 11 , y = 0
Time = 205: clk = 1, reset = 0, x = 1, state = 00 , y = 0
Time = 210: clk = 0, reset = 0, x = 1, state = 00 , y = 0
Time = 215: clk = 1, reset = 0, x = 1, state = 00 , y = 0
Back to state a
Time = 220: clk = 0, reset = 0, x = 1, state = 00 , y = 0
mpsamartha@Samartha:~/Verilog/Practice_Sheet/Q10_Finite_State_Machines$
```

Figure 23: Terminal output for Mealy FSM

As seen above in the terminal output, we go through 3 different cycles of states as explained above. Thus we can see the print statements of being at state a and the corresponding state in the row above it. It matches (00) with the expected results.

The following is the state table derived from the state diagram 20.

Current State	Input	Next State	Output
a	0	b	1
a	1	c	0
b	0	c	0
b	1	d	1
c	0	b	0
c	1	d	1
d	0	c	1
d	1	a	1

Table 5: State Transition Table for Mealy FSM

the following is the timing diagram for the Mealy FSM. The explanation above about the three cycles can also be verified from the value of the state below in the diagram. The state goes through the three cycles, starting and ending at the initial state $S\_a = 00$.

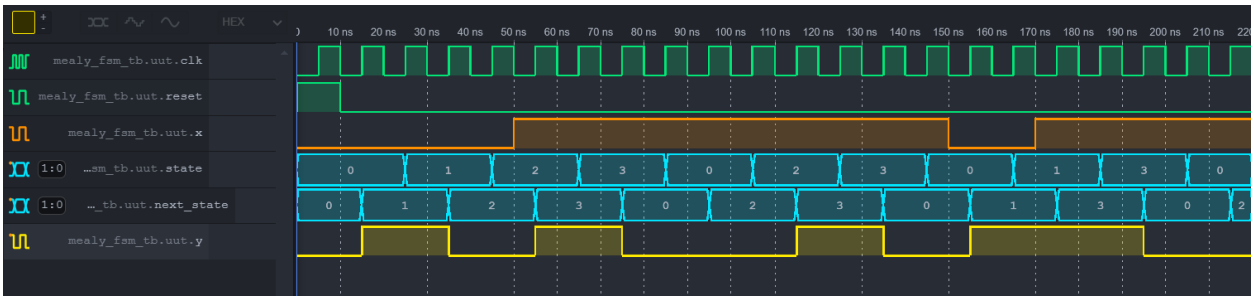

Figure 24: Timing Diagram for Mealy FSM