SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE

Constituent of Symbiosis International (Deemed University), Pune

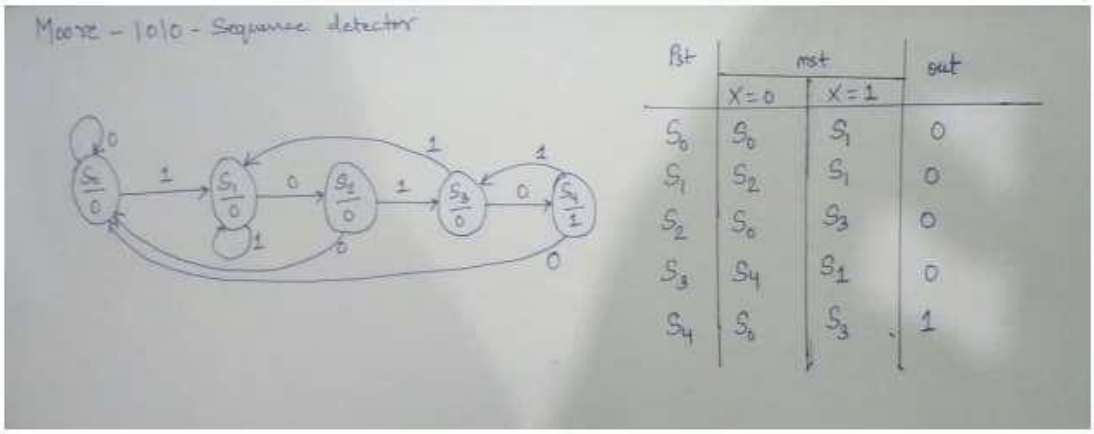
Name:Arjunsingh Gautam **PRN:22070123043** **Date:22/10/24**

Experiment 10

Aim: Implementation of Moore FSM in Verilog.

Software: Xilinx Vivado Design Suite

State Diagram and State Table



Procedure:

1) Open the Xilinx Vivado Design Suite
2) Go to file and click new project
3) Enter the project name and click next
4) Select the family name of the FPGA Device, parameter setting and HDL is verilog click next and click finish.
5) Click new source.
6) Select Verilog module and type file name and click next.
7) Assign input and output port and click next.
8) Finally, the report is shown click finish.
9) Type the program save and check syntax error.
10) To see the output waveform select ISim simulator
11) Give values to the input variables using force clock or force constant and then click run
12) In wave window, click run icon and you can see corresponding output.
13) For synthesis of the design, open XST synthesis tool and run the design for synthesis.
14) Open RTL schematic and Technology schematic and understand implemented design

on FPGA

15) Open synthesis result to know resource utilization of the design.

Code:

```
`timescale 1ns / 1ps
// Arjunsingh Gautam (22070123043)
module moore1010(
    input clk,
    input reset,
    input in,
    output reg out
);
    // State encoding
    reg [2:0] pst, nst; // Present and next state

    // State definitions
    parameter
        s0 = 3'b000, // Initial state
        s1 = 3'b001, // Detected '1'
        s2 = 3'b010, // Detected '10'
        s3 = 3'b011, // Detected '101'
        s4 = 3'b100; // Detected '1010'

    // State transition on clock edge or reset
    always @(posedge clk or posedge reset) begin
        if (reset)
            pst <= s0; // Reset to initial state
        else
            pst <= nst; // Update present state
    end

    // Next state logic
    always @(pst or in) begin
        case (pst)
            s0: begin
                if (in)
                    nst <= s1; // Move to state s1 on '1'
                else
                    nst <= s0; // Stay in s0
            end

            s1: begin
                if (!in)
                    nst <= s2; // Move to state s2 on '0'
                else
                    nst <= s1; // Stay in s1
            end
```

```
            s2: begin
                if (in)
                    nst <= s3; // Move to state s3 on '1'
                else
                    nst <= s0; // Move back to s0 on '0'
            end

            s3: begin
                if (!in)
                    nst <= s4; // Move to state s4 on '0'
                else
                    nst <= s1; // Go back to s1 on '1'
            end

            s4: begin
                if (in)
                    nst <= s3; // Move back to s3 on '1'
                else
                    nst <= s0; // Move back to s0 on '0'
            end

            default:
                nst <= s0; // Default to initial state
        endcase
    end

    // Output logic (Moore)
    always @(pst) begin
        if (pst == s4)
            out <= 1; // Output high when in state s4
        else
            out <= 0; // Output low otherwise
    end
endmodule
```

Output:

Test bench:

```
1   `timescale 1ns / 1ps
2   ////////////////////////////////////////////////
3   // Arjunsingh Gautam (22070123043)
4   module tt_moore_1010;
5   // Inputs
6   reg clk;
7   reg reset;
8   reg in;
9   // Outputs
10  wire out;
11  // Instantiate the Unit Under Test (UUT)
12  moore1010 uut (
13  .clk(clk),
14  .reset(reset),
15  .in(in),
16  .out(out)
17  );
18  initial
19  begin
20  reset=0;clk=0;in=0;
21  #5 in=0;
22  #5 in=0;
23  #5 in=1;
24  #10 in=0;
25  #10 in=1;
26  #10 in=0;
27  #10 in=1;
28  #10 in=0;
29  #10 in=0;
30  #5 in=0;
31  #5 in=1;
32  end
33  always
34  #5 clk=~clk;
35  endmodule
36
```

Result:

a) Simulate the design using the simulation tool to verify its functionality.
b) Synthesis the design using synthesis tool to find out the resource utilization.
c) Draw RTL and Technology schematics of the design.

Conclusion:

The Moore state machine effectively detects the sequence "1010," generating an output signal only when the entire sequence is recognized. This implementation demonstrates the advantages of Moore machines, particularly their stability in output behavior during state transitions.

Questions for Reflection:

1) What is the difference between Mealy and Moore FSM?
2) Which FSM is more stable and why?
3) Draw FSM diagram for sequence detector 1100.
4) What do you mean by overlapping and non overlapping FSM?

- The difference between Mealy and Moore FSMs lies in their output behavior: Mealy outputs depend on both the current state and input, while Moore outputs depend solely on the current state.

- Moore FSMs are more stable because their outputs change only on state transitions, minimizing the chance of glitches from input variations.

- (Diagram description) The FSM for the sequence "1100" includes states: s0 (start), s1 (first '1'), s2 (second '1'), s3 (first '0'), and s4 (second '0'), with an output generated at s4.

- Overlapping FSMs can detect sequences within other sequences, while non-overlapping FSMs identify distinct sequences without recognizing any embedded sequences.