SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE

Constituent of Symbiosis International (Deemed University), Pune

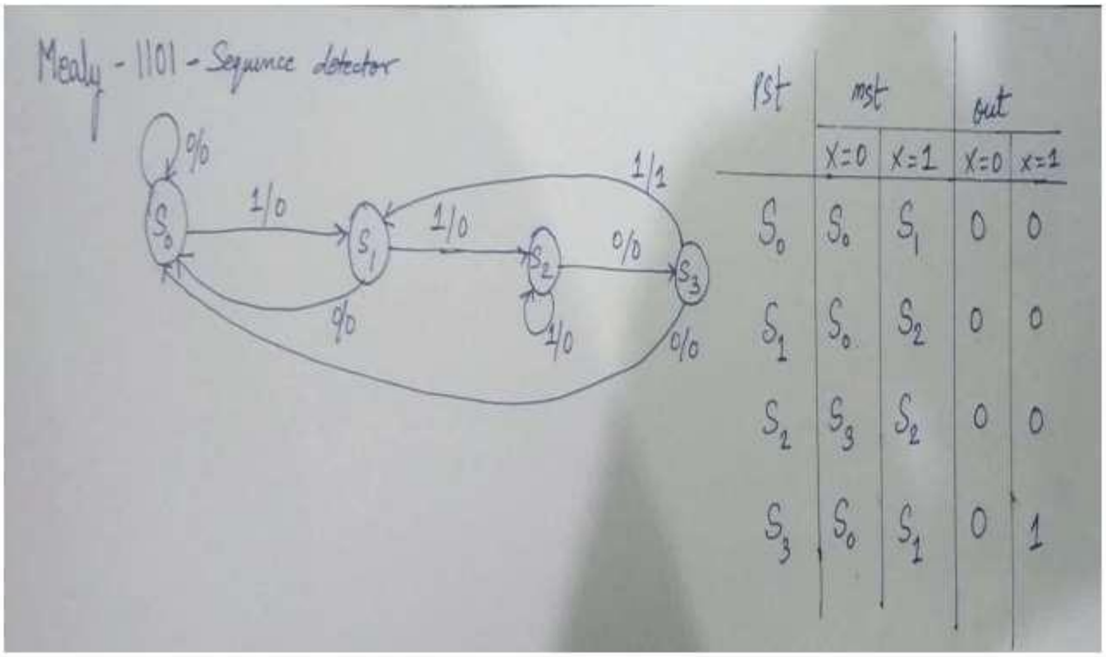
Name:Arjunsingh Gautam **PRN:22070123043** **Date:22/10/24**

Experiment 9

Aim: Implementation of Mealy FSM in Verilog.

Software: Xilinx Vivado Design Suite

State Diagram and State Table



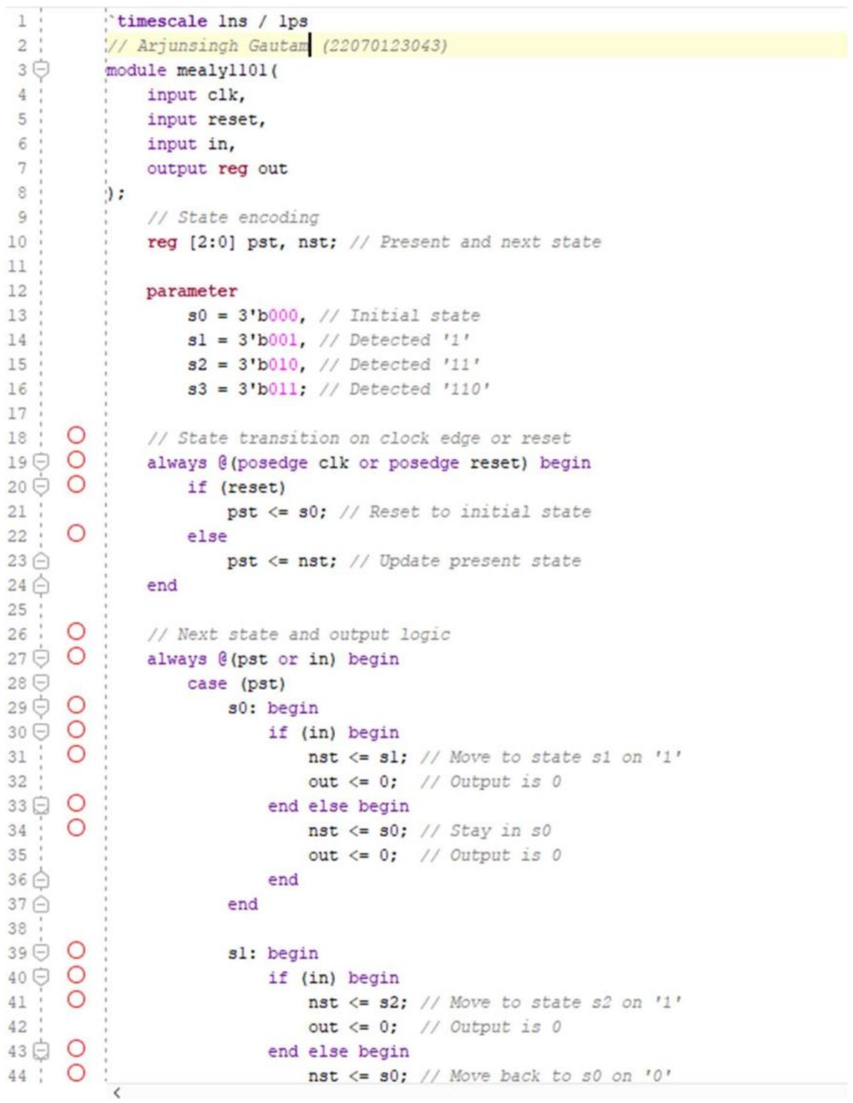
ps	ns x=0	ns x=1	out x=0	out x=1
S_0	S_0	S_1	0	0
S_1	S_0	S_2	0	0
S_2	S_3	S_2	0	0
S_3	S_0	S_1	0	1

Procedure:

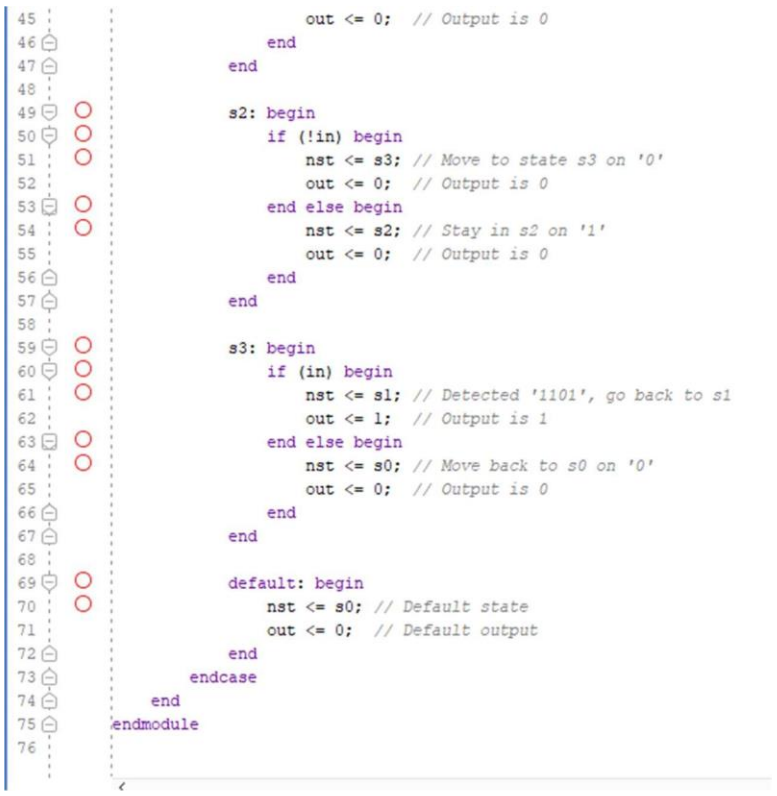
1) Open the Xilinx Vivado Design Suite
2) Go to file and click new project
3) Enter the project name and click next
4) Select the family name of the FPGA Device, parameter setting and HDL is verilog click next and click finish.
5) Click new source.
6) Select Verilog module and type file name and click 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:

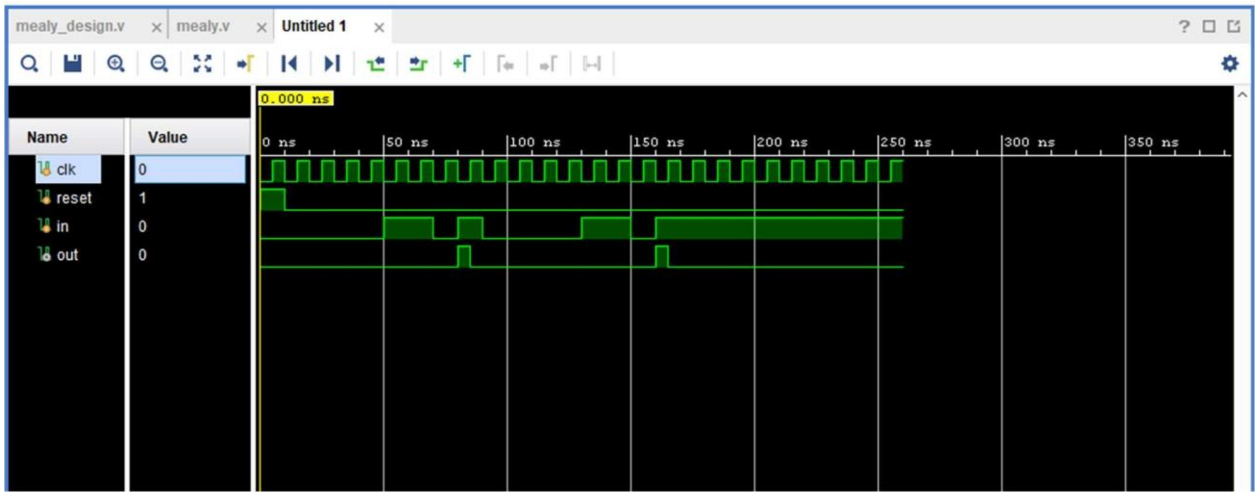


```
`timescale 1ns / 1ps
// Arjunsingh Gautam (22070123043)
module mealy1101(
    input clk,
    input reset,
    input in,
    output reg out
);
    // State encoding
    reg [2:0] pst, nst; // Present and next state

    parameter
        s0 = 3'b000, // Initial state
        s1 = 3'b001, // Detected '1'
        s2 = 3'b010, // Detected '11'
        s3 = 3'b011; // Detected '110'

    // 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 and output logic
    always @(pst or in) begin
        case (pst)
            s0: begin
                if (in) begin
                    nst <= s1; // Move to state s1 on '1'
                    out <= 0;  // Output is 0
                end else begin
                    nst <= s0; // Stay in s0
                    out <= 0;  // Output is 0
                end
            end

            s1: begin
                if (in) begin
                    nst <= s2; // Move to state s2 on '1'
                    out <= 0;  // Output is 0
                end else begin
                    nst <= s0; // Move back to s0 on '0'
```

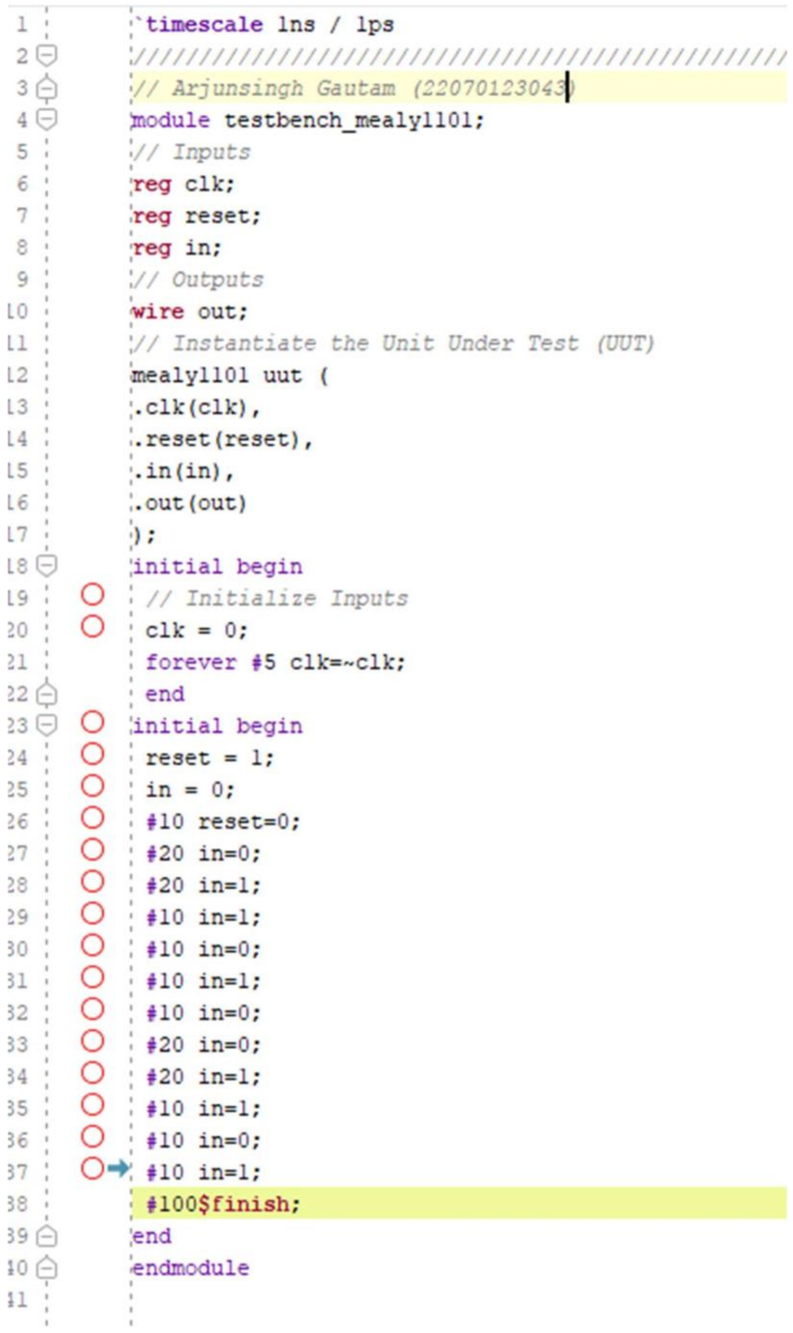


```
                    out <= 0;  // Output is 0
                end
            end

            s2: begin
                if (!in) begin
                    nst <= s3; // Move to state s3 on '0'
                    out <= 0;  // Output is 0
                end else begin
                    nst <= s2; // Stay in s2 on '1'
                    out <= 0;  // Output is 0
                end
            end

            s3: begin
                if (in) begin
                    nst <= s1; // Detected '1101', go back to s1
                    out <= 1;  // Output is 1
                end else begin
                    nst <= s0; // Move back to s0 on '0'
                    out <= 0;  // Output is 0
                end
            end

            default: begin
                nst <= s0; // Default state
                out <= 0;  // Default output
            end
        endcase
    end
endmodule
```

Output:



TestBench:



```
`timescale 1ns / 1ps
////////////////////////////////////////////////////
// Arjunsingh Gautam (22070123043)
module testbench_mealy1101;
// Inputs
reg clk;
reg reset;
reg in;
// Outputs
wire out;
// Instantiate the Unit Under Test (UUT)
mealy1101 uut (
.clk(clk),
.reset(reset),
.in(in),
.out(out)
);
initial begin
 // Initialize Inputs
 clk = 0;
 forever #5 clk=~clk;
 end
initial begin
 reset = 1;
 in = 0;
 #10 reset=0;
 #20 in=0;
 #20 in=1;
 #10 in=1;
 #10 in=0;
 #10 in=1;
 #10 in=0;
 #20 in=0;
 #20 in=1;
 #10 in=1;
 #10 in=0;
 #10 in=1;
 #100$finish;
end
endmodule
```

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 Mealy state machine successfully detects the sequence "1101," producing the correct output based on the defined state transitions. This implementation demonstrates the effectiveness of using state machines in recognizing specific input patterns in digital systems.

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?

1. The main difference between Mealy and Moore FSMs is that Mealy outputs depend on both the current state and the input, while Moore outputs depend only on the current state.
2. Moore FSMs are generally considered more stable because their outputs change only on state transitions, reducing the risk of output glitches due to input changes.
3. (Diagram description) The FSM for detecting the sequence "1100" has states: s0 (start), s1 (first '1'), s2 (second '1'), s3 (first '0' after '11'), and s4 (second '0' after '11'); an output is generated when reaching s4.
4. Overlapping FSMs allow for sequences to be detected within other sequences, while non-overlapping FSMs only detect distinct sequences without considering any potential overlaps.