Experiment-5 : Sequence Detector

Objective:

To design a Moore FSM that detects the sequence 1001. The test bench should check the generated output with the expected output and prints pass/fail messages.

Tool Used:

Xilinx ISE.

Theory:

A FSM detects the matching of input sequence and triggers the output, a self checking test bench is stimulated using shift registers as 4 bit memory and the output is verified.

DUT Code:

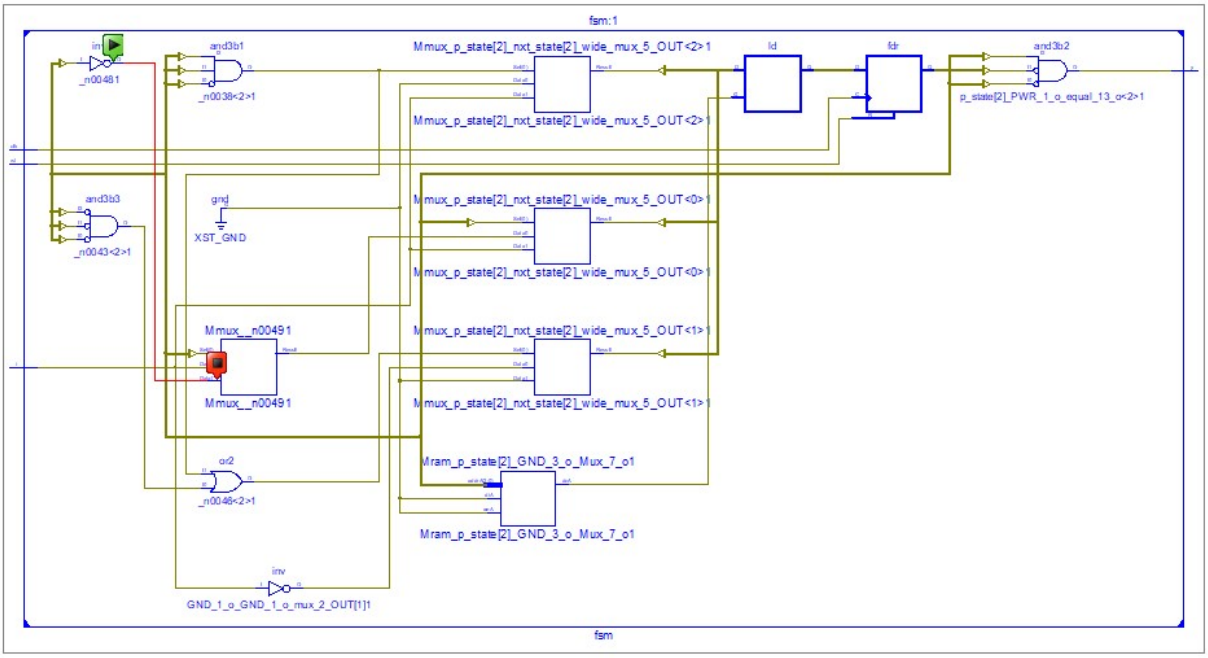
```
module fsm(input i,rst,clk, output y);
    parameter idle=3'b000, s1=3'b001, s10=3'b010, s100=3'b011, s1001=3'b100;
    reg [2:0] p_state,nxt_state;

    always@(*)begin
        case(p_state)
            idle    : nxt_state <= i ? s1   : idle;
            s1      : nxt_state <= i ? s1   : s10   ;
            s10 : nxt_state <= i ? s1   : s100;
            s100    : nxt_state <= i ? s1001: idle;
            s1001 : nxt_state <= i ? s1 : s10   ;
        endcase
    end

    always@(posedge clk)begin
        if(rst) p_state <= idle;
        else p_state <= nxt_state;
    end

    assign y = (p_state == s1001) ? 1 : 0;
endmodule
```

TB Code:

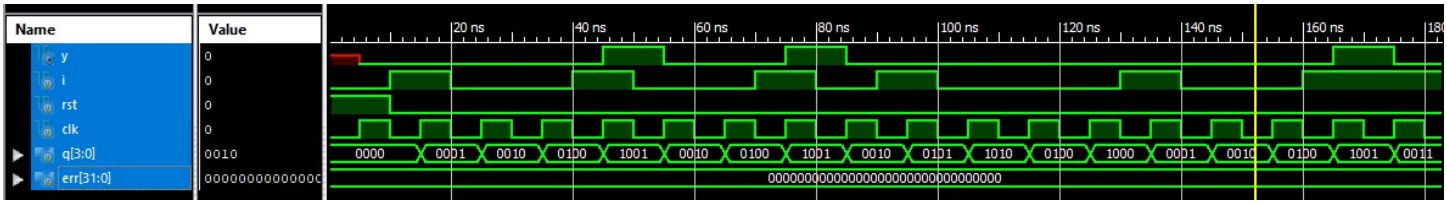
```
module tb;

    reg i=0,rst,clk=0;// Inputs
    wire y;// Output
    fsm uut (i,rst,clk,y);// Instantiate the Unit Under Test (UUT)
    reg [3:0]q=0    ;
    integer err = 0;

    initial forever #5 clk = !clk;
    initial forever @(posedge clk) q <= {q[2:0],i};

    initial begin
        $monitor("%0d : input i = %b state is %d, output y = %b ",$time,i,uut.p_state,y);
```

```
        rst = 1'b1;
        #10;
        rst = 1'b0;
        @(negedge clk) i = 1;
        @(negedge clk) i = 0;
        @(negedge clk) i = 0;
        @(negedge clk) i = 1;
        @(negedge clk) i = 0;
        @(negedge clk) i = 0;
        @(negedge clk) i = 1;
        @(negedge clk) i = 0;
        @(negedge clk) i = 1;
        @(negedge clk) i = 0;
        @(negedge clk) i = 0;
        @(negedge clk) i = 0;
        @(negedge clk) i = 1;
        @(negedge clk) i = 0;
        @(negedge clk) i = 0;
        @(negedge clk) i = 1;
        #100;
        if(err) $display("error"); else $display("success");
        #20;
        $finish;
    end

    always@(q)begin
        if(q == 4'b1001)begin
            @(negedge clk);
            if(!y) err = err+1;
        end
    end

endmodule
```

RTL Diagram:



Output Waveform:



Simulation Output:

```
0 : input i = 0 state is x, output y = x
5 : input i = 0 state is 0, output y = 0
10 : input i = 1 state is 0, output y = 0
15 : input i = 1 state is 1, output y = 0
20 : input i = 0 state is 1, output y = 0
25 : input i = 0 state is 2, output y = 0
35 : input i = 0 state is 3, output y = 0
40 : input i = 1 state is 3, output y = 0
45 : input i = 1 state is 4, output y = 1
50 : input i = 0 state is 4, output y = 1
55 : input i = 0 state is 2, output y = 0
65 : input i = 0 state is 3, output y = 0
70 : input i = 1 state is 3, output y = 0
75 : input i = 1 state is 4, output y = 1
80 : input i = 0 state is 4, output y = 1
85 : input i = 0 state is 2, output y = 0
90 : input i = 1 state is 2, output y = 0
95 : input i = 1 state is 1, output y = 0
100 : input i = 0 state is 1, output y = 0
105 : input i = 0 state is 2, output y = 0
115 : input i = 0 state is 3, output y = 0
125 : input i = 0 state is 0, output y = 0
130 : input i = 1 state is 0, output y = 0
135 : input i = 1 state is 1, output y = 0
140 : input i = 0 state is 1, output y = 0
145 : input i = 0 state is 2, output y = 0
155 : input i = 0 state is 3, output y = 0
160 : input i = 1 state is 3, output y = 0
165 : input i = 1 state is 4, output y = 1
175 : input i = 1 state is 1, output y = 0
success
```

Result:

The simulation output and the RTL diagram is observed and found to be valid.