Lab 12

Implement a Traffic Light controller using FSM.

Spring 2025

Submitted by: **Mohsin Sajjad**

Registration No: **22pwsce2149**

Class Section: **A**

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Student Signature: _____

Submitted to:

Engr. Faheem Jan

Month Day, Year (28 05, 2025)

Department of Computer Systems Engineering

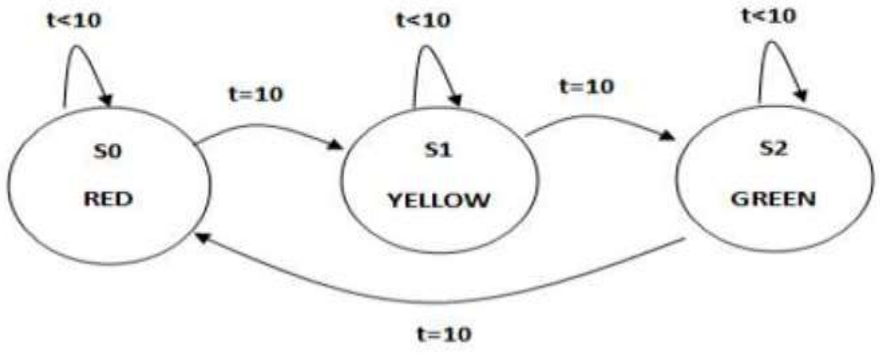
University of Engineering and Technology, Peshawar

Implement a Traffic Light controller using FSM.

Objective:

The lab this week will continue the introduction to FSMs with a simple Traffic Light Controller design project. In the lab, you are required to create a state diagram of a Mealy machine, which implements a traffic light controller, based on provided guidelines. You will then use this state diagram to write the behavioral Verilog description of the traffic light controller. The testing of your traffic light controller will take place during the lab session using the ISE development tools and the Spartan 6 board. As with the previous lab, we will make use of the character LEDs to display the outputs of our digital circuit.

Figure 1. Task1:

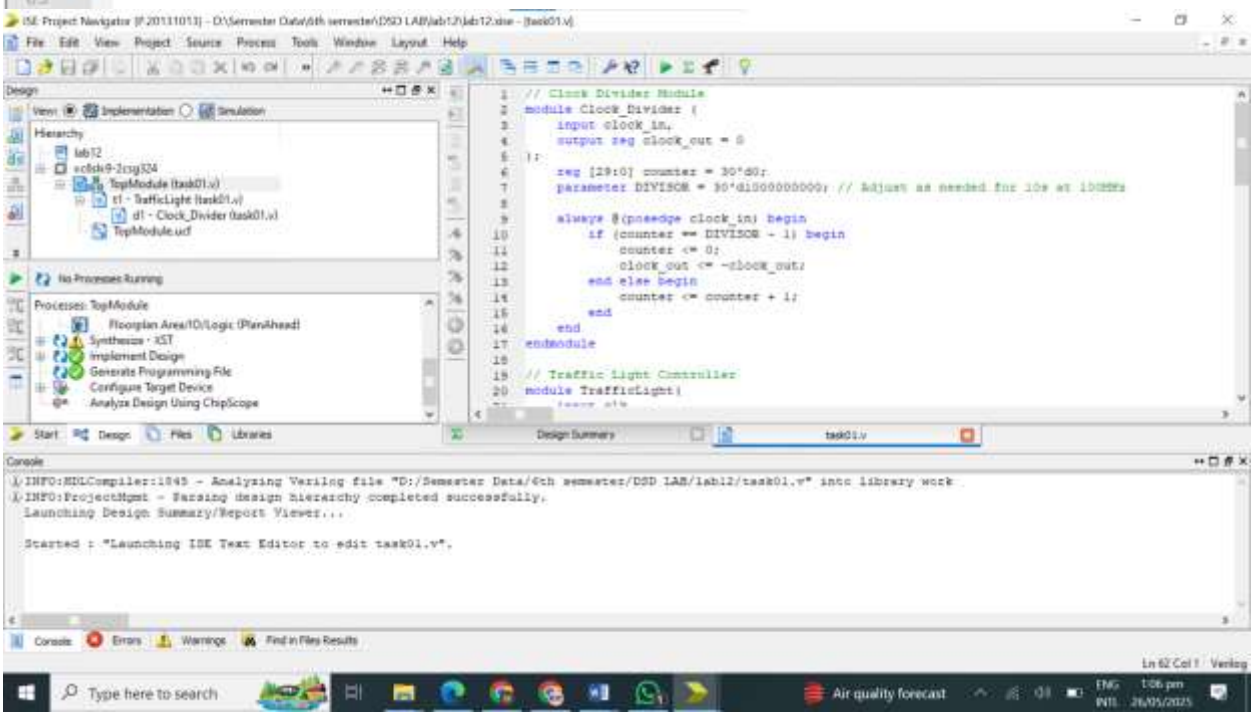


CODE:

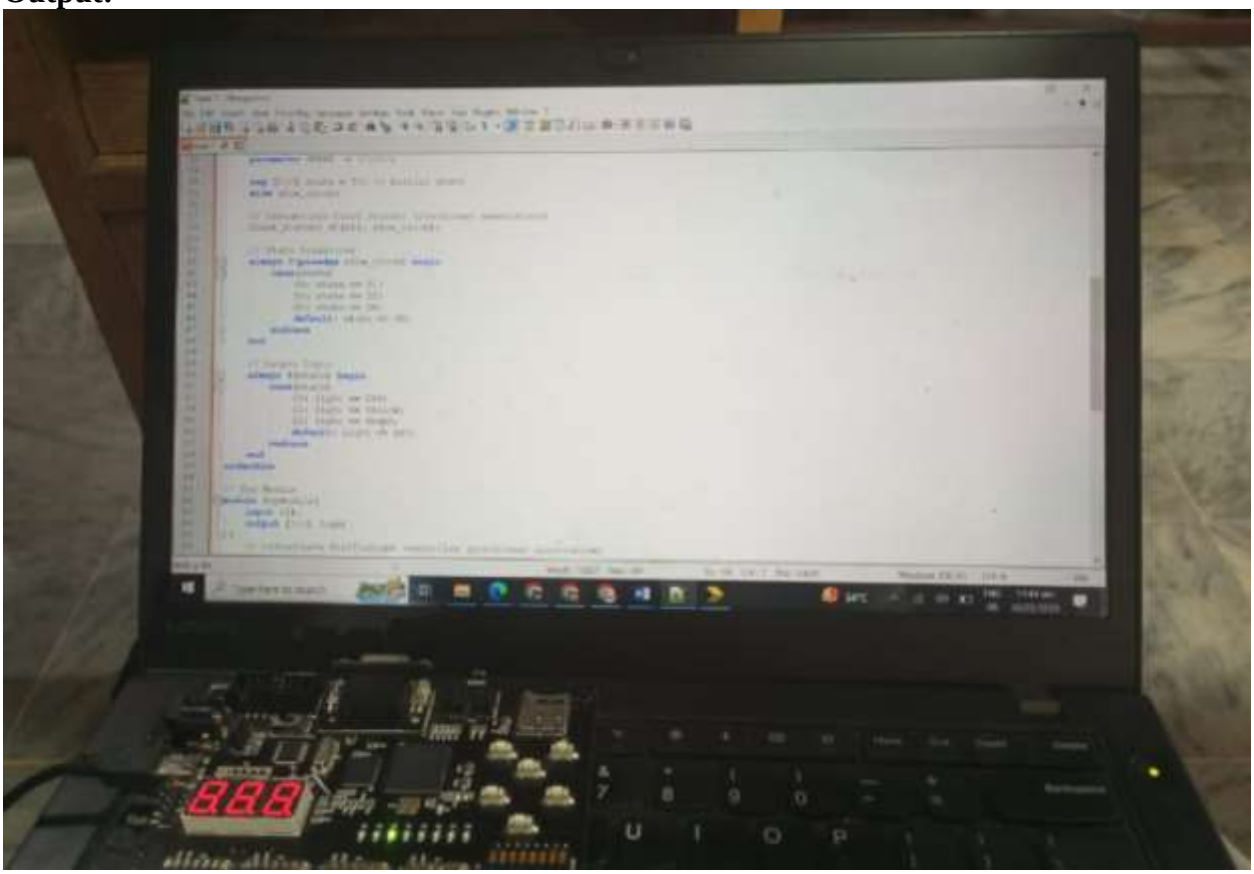
```
module Clock_Divider (
    input clock_in,
    output reg clock_out = 0
);
    reg [29:0] counter = 30'd0;
    parameter DIVISOR = 30'd1000000000; // Adjust as needed for 10s at 100MHz

    always @(posedge clock_in) begin
        if (counter == DIVISOR - 1) begin
            counter <= 0;
            clock_out <= ~clock_out;
        end else begin
            counter <= counter + 1;
        end
    end
endmodule

module TrafficLight(
    input clk,
    output reg [2:0] light
);
    // State encoding
    parameter S0 = 2'b00;
    parameter S1 = 2'b01;
    parameter S2 = 2'b10;

    // Light signals
    parameter RED    = 3'b100;
    parameter YELLOW = 3'b010;
    parameter GREEN  = 3'b001;

    reg [1:0] state = S0; // Initial state
    wire slow_clock;

    Clock_Divider d1(clk, slow_clock);
```

new 2

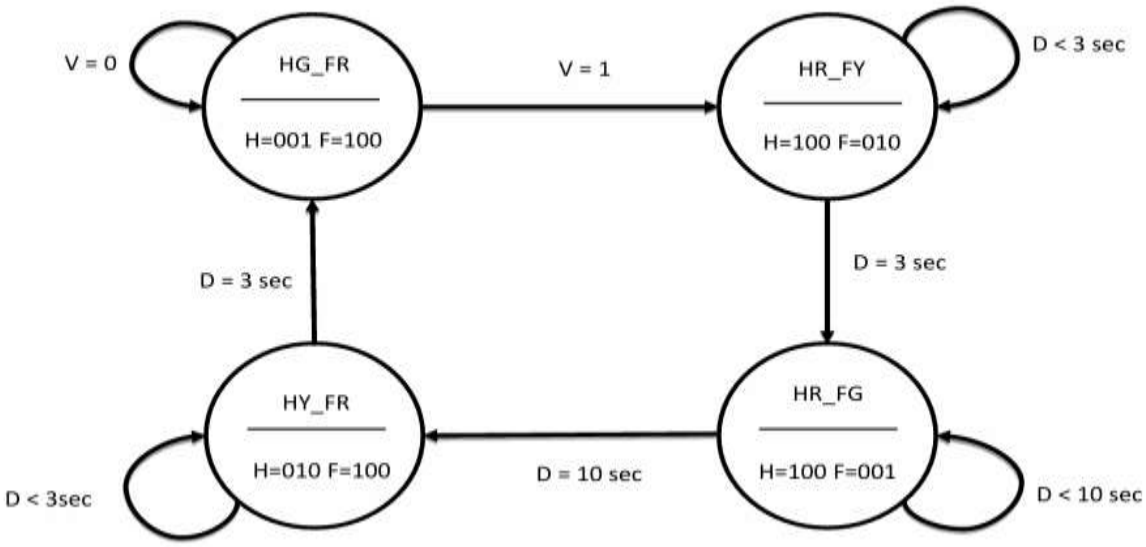
```
36
37      always @(posedge slow_clock) begin
38          case(state)
39              S0: state <= S1;
40              S1: state <= S2;
41              S2: state <= S0;
42              default: state <= S0;
43          endcase
44      end
45
46      // Output logic
47      always @(state) begin
48          case(state)
49              S0: light <= RED;
50              S1: light <= YELLOW;
51              S2: light <= GREEN;
52              default: light <= RED;
53          endcase
54      end
55  endmodule
56
57  module TopModule(
58      input clk,
59      output [2:0] light
60  );
61      TrafficLight t1(clk, light);
62  endmodule
63
```



Output:



Task 02:



CODE:

new 2

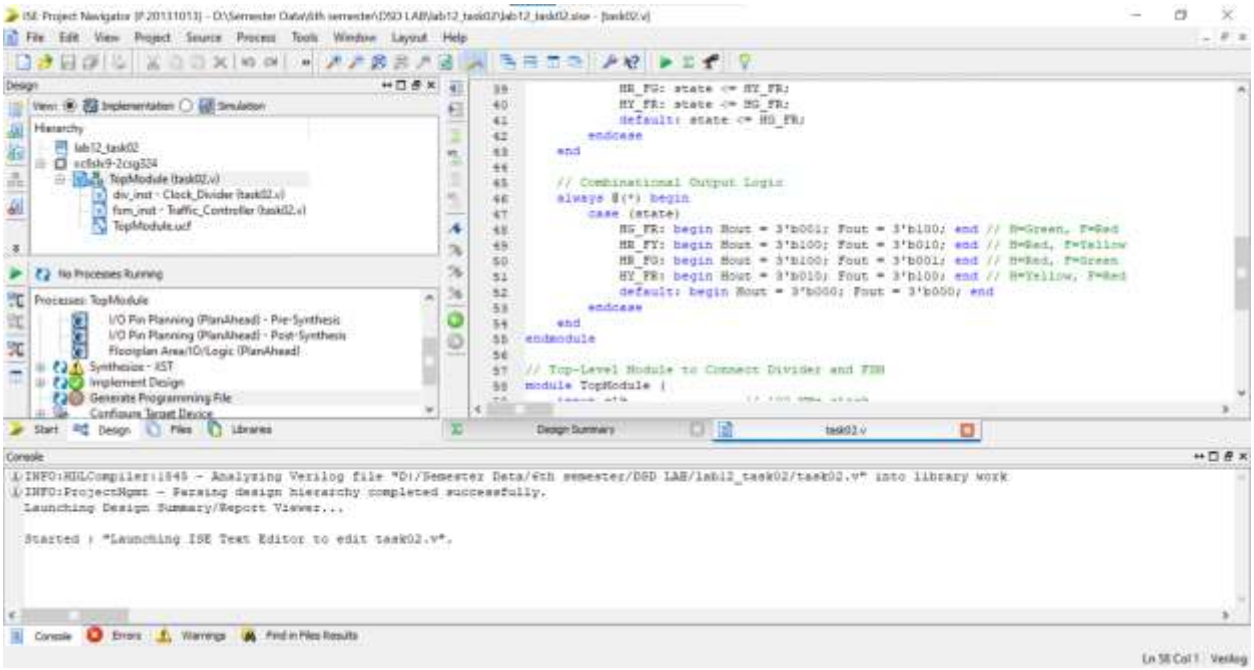
```
module Clock_Divider (
    input clock_in,
    output reg clock_out = 0
);
    reg [29:0] counter = 30'd0;
    parameter DIVISOR = 30'd100000000; // 1 second at 100 MHz

    always @(posedge clock_in) begin
        if (counter == DIVISOR - 1) begin
            counter <= 0;
            clock_out <= ~clock_out;
        end else begin
            counter <= counter + 1;
        end
    end
endmodule

module Traffic_Controller (
    input clk,
    input reset,
    input V,
    output reg [2:0] Hout,
    output reg [2:0] Fout
);
    // State encoding
    parameter HG_FR = 2'b00,  // Highway Green, Farm Red
              HR_FY = 2'b01,  // Highway Red, Farm Yellow
              HR_FG = 2'b10,  // Highway Red, Farm Green
              HY_FR = 2'b11;  // Highway Yellow, Farm Red

    reg [1:0] state = HG_FR;
    reg [3:0] counter = 0;  // 4-bit counter is enough for 10 sec

    always @(posedge clk or posedge reset) begin
        if (reset) begin
```

new 2

```
            state <= HG_FR;
            counter <= 0;
        end else begin
            case (state)
                HG_FR: begin
                    Hout <= 3'b001; // Green
                    Fout <= 3'b100; // Red
                    if (V) begin
                        state <= HR_FY;
                        counter <= 0;
                    end
                end

                HR_FY: begin
                    Hout <= 3'b100; // Red
                    Fout <= 3'b010; // Yellow
                    if (counter < 3)
                        counter <= counter + 1;
                    else begin
                        state <= HR_FG;
                        counter <= 0;
                    end
                end

                HR_FG: begin
                    Hout <= 3'b100; // Red
                    Fout <= 3'b001; // Green
                    if (counter < 10)
                        counter <= counter + 1;
                    else begin
                        state <= HY_FR;
                        counter <= 0;
                    end
                end
```

new 2

```
                    HY_FR: begin
                        Hout <= 3'b010; // Yellow
                        Fout <= 3'b100; // Red
                        if (counter < 3)
                            counter <= counter + 1;
                        else begin
                            state <= HG_FR;
                            counter <= 0;
                        end
                    end
                endcase
            end
        end
    endmodule

    module TopModule (
        input clk,
        input reset,
        input V,
        output [2:0] Hout,
        output [2:0] Fout
    );
        wire slow_clk;
        Clock_Divider divider(clk, slow_clk);
        Traffic_Controller controller(slow_clk, reset, V, Hout, Fout);
    endmodule
```



Output: