VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

FALL SEMESTER 2024-25

Course Code : BECE 102P
Course Name : Digital Systems Design Lab

PROJECT REPORT

AUTOMATED TRAFFIC LIGHT CONTROL

Team Members & Registration Numbers :

Aarti Chhabaria (23BML0045), Soham Patra (23BEC 0193), Ishita Mohanta (23BML0086), Srijita Bhattacharjee (23BEC0120)

LAB SLOT : L43+L44

Faculty Incharge : Dr. Nithish Kumar V

AIM OF THE PROJECT :

The goal of this experiment is to develop and construct a traffic light management system using a Moore finite state machine (FSM) model coupled with counters that will allow the system to control delay timing. In particular, the objective is to control the timing of an intersection traffic light sequence comprising yellow-red-green lights in a logical, effective, and safe way with a view of enhancing traffic management.

WORKING PRINCIPLE OF THE PROJECT :

A Moore Machine is a Finite State Machine whose outputs depend only on the present state and are not influenced by any input events. There is no output observed during state transition. The outputs of a Moore Machine FSM change synchronously at the beginning of each state which shows that the output is associated with the states.

The purpose of using a Moore model FSM in automated traffic light control is to ensure that the system's outputs depend solely on its current state, rather than on the inputs. This approach offers several key advantages :

1. Predictability and Simplicity : in a Moore FSM, outputs (like the yellow, red and green lights) are linked to states, which simplifies its design and understanding of the traffic light sequence. The lights change only during state transitions, resulting in a predictable control flow.

2. Reliability and Safety : since the outputs are not directly influenced by inputs, the system is less susceptible to glitches or errors that might arise from temporary input changes. This stability is vital for traffic control, as it helps prevent erratic light signal changes that could compromise safety.

3. Clear Timing Control : the Moore model distinguishes timing from state transitions by using counters for delays. Each state is maintained for a specific duration, managed by these counters, before transitioning to the next state, which allows precise automated control over the light phases.

4. Ease of Debugging and Testing : with outputs linked solely to the states, troubleshooting and testing each individual state and transitions becomes easier, ensuring correct operation before the system is deployed in real-world scenarios.

In the Moore FSM model, each state is associated with a specific output configuration (which traffic lights should be on).

For a typical intersection, you might define states such as:

a. **State 1:** Green for North-South, Red for East-West
b. **State 2:** Yellow for North-South, Red for East-West
c. **State 3:** Red for North-South, Green for East-West
d. **State 4:** Red for North-South, Yellow for East-West

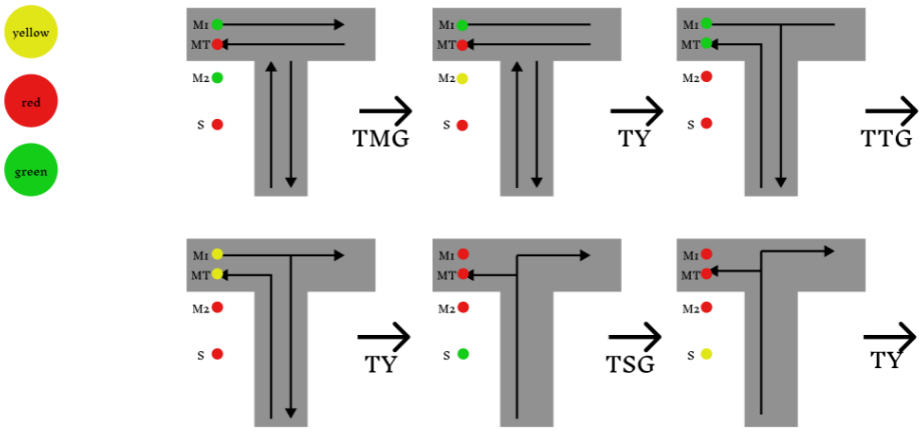
Each state has fixed outputs (light signals) tied to it, independent of any inputs.
Transitions between states are determined by counters, which introduce delays between state changes.

- For instance, in **State 1 (Green for North-South)**, a counter could be set to a specific value to hold the state for a fixed time (e.g., 10 seconds). Once the counter reaches zero, the FSM transitions to **State 2 (Yellow for North-South)**.
- Similarly, after holding in **State 2** for a shorter period (e.g., 3 seconds), the FSM moves to **State 3 (Red for North-South, Green for East-West)**.

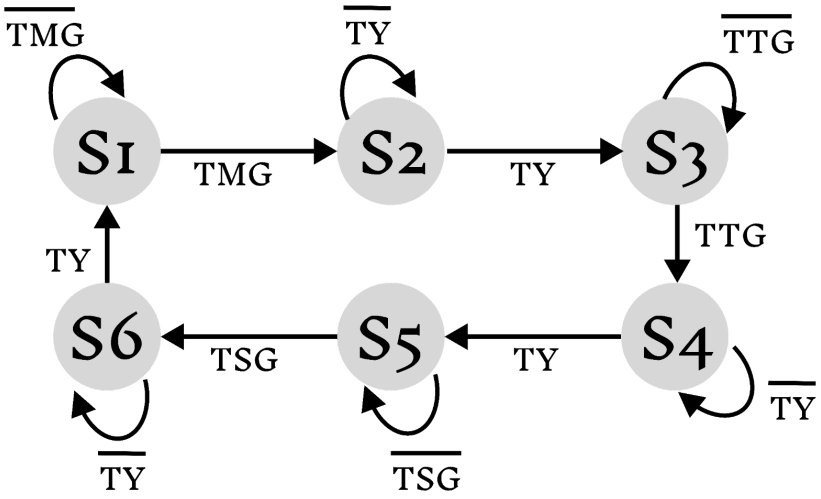
Each state transition only occurs when the timer expires, ensuring a predictable and orderly light sequence.

After the FSM reaches the final state in the sequence (State 4), it loops back to the first state, starting a new cycle.

This repetitive cycle mimics real-world traffic lights, continually alternating the green and red lights for each direction.



FORMULATION OF THE STATE TRANSITION DIAGRAM :



FORMULATION OF THE STATE TRANSITION TABLE :

Present State			Input			Next State			ST	M1			M2			MT			S		
0	0	0	-			0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	¬T	¬M	¬G	0	0	1	0	0	0	1	0	0	1	1	0	0	1	0	0
			T	M	G	0	1	0	1												
0	1	0	¬T	¬Y		0	1	0	0	0	0	1	0	1	0	1	0	0	1	0	0
			T	Y		0	1	1	1												
0	1	1	¬T	¬T	¬G	0	1	1	0	0	0	1	1	0	0	0	0	1	1	0	0
			T	T	G	1	0	0	1												
1	0	0	¬T	¬Y		1	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0
			T	Y		1	0	1	1												
1	0	1	¬T	¬S	¬G	1	0	1	0	1	0	0	1	0	0	1	1	0	0	0	1
			T	S	G	1	1	0	1												
1	1	0	¬T	¬Y		1	1	0	0	1	0	0	1	1	0	1	0	0	0	1	0
			T	Y		0	0	1	1												
1	1	1	-			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

VERILOG CODE :

```
module traffic_light(
  input clk,
  input rst,
  output reg [2:0] light_M1,
  output reg [2:0] light_S,
  output reg [2:0] light_MT,
  output reg [2:0] light_M2,
  output reg [3:0] count,   // Added count as an output
  output reg [2:0] ps       // Added ps as an output
);

  // State parameters
  parameter s1 = 0, s2 = 1, s3 = 2, s4 = 3, s5 = 4, s6 = 5;

  // Adjustable delay parameters (can be modified as needed)
  parameter delay_s1 = 4'd7; // Replace sec7
  parameter delay_s2 = 4'd2; // Replace sec2
  parameter delay_s3 = 4'd5; // Replace sec5
  parameter delay_s4 = 4'd2; // Replace sec2
  parameter delay_s5 = 4'd3; // Replace sec3
  parameter delay_s6 = 4'd2; // Replace sec2

  reg [3:0] delay; // Variable to hold the current delay count for each state

  always @(posedge clk or posedge rst) begin
    if (rst) begin
      ps <= s1;
      count <= 0;
      delay <= delay_s1; // Set the initial delay based on state s1
    end else begin
      if (count < delay) begin
        count <= count + 1;
      end else begin
```

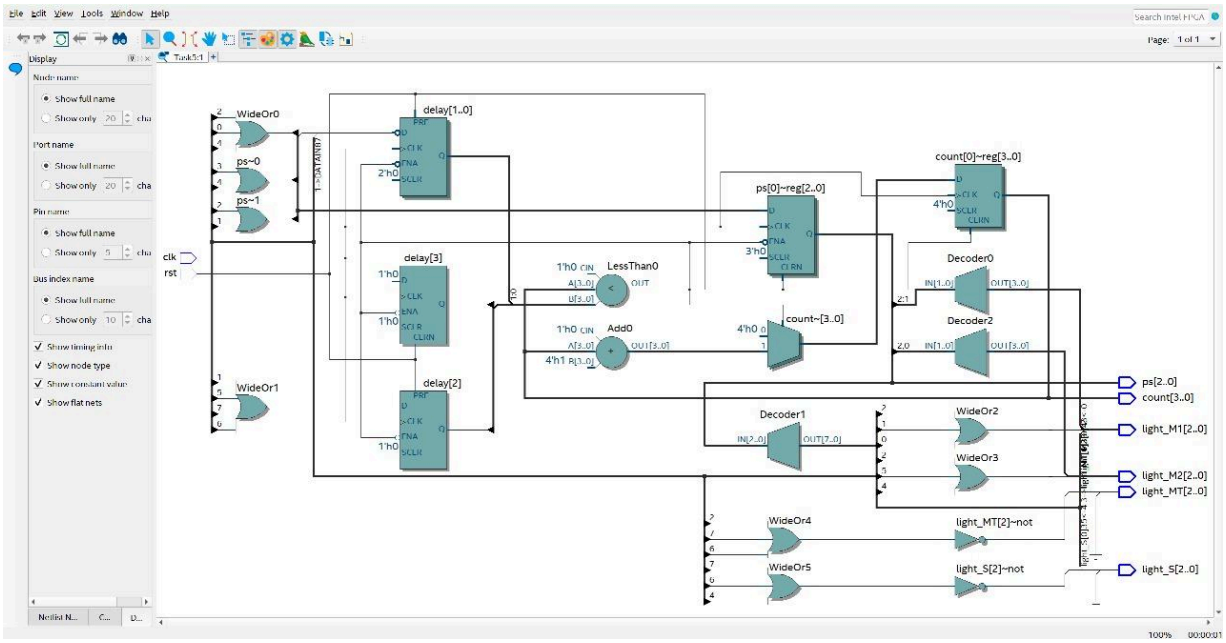
```
      count <= 0;
      case (ps)
        s1: begin
          ps <= s2;
          delay <= delay_s2; // Update delay for the next state
        end
        s2: begin
          ps <= s3;
          delay <= delay_s3;
        end
        s3: begin
          ps <= s4;
          delay <= delay_s4;
        end
        s4: begin
          ps <= s5;
          delay <= delay_s5;
        end
        s5: begin
          ps <= s6;
          delay <= delay_s6;
        end
        s6: begin
          ps <= s1;
          delay <= delay_s1;
        end
        default: begin
          ps <= s1;
          delay <= delay_s1;
        end
      endcase
    end
  end
end

// State-based light control logic
```

```
always @(ps) begin
  case (ps)
    s1: begin
      light_M1 <= 3'b001;
      light_M2 <= 3'b001;
      light_MT <= 3'b100;
      light_S  <= 3'b100;
    end
    s2: begin
      light_M1 <= 3'b001;
      light_M2 <= 3'b010;
      light_MT <= 3'b100;
      light_S  <= 3'b100;
    end
    s3: begin
      light_M1 <= 3'b001;
      light_M2 <= 3'b100;
      light_MT <= 3'b001;
      light_S  <= 3'b100;
    end
    s4: begin
      light_M1 <= 3'b010;
      light_M2 <= 3'b010;
      light_MT <= 3'b100;
      light_S  <= 3'b100;
    end
    s5: begin
      light_M1 <= 3'b100;
      light_M2 <= 3'b100;
      light_MT <= 3'b100;
      light_S  <= 3'b001;
    end
    s6: begin
      light_M1 <= 3'b100;
      light_M2 <= 3'b100;
      light_MT <= 3'b100;
```

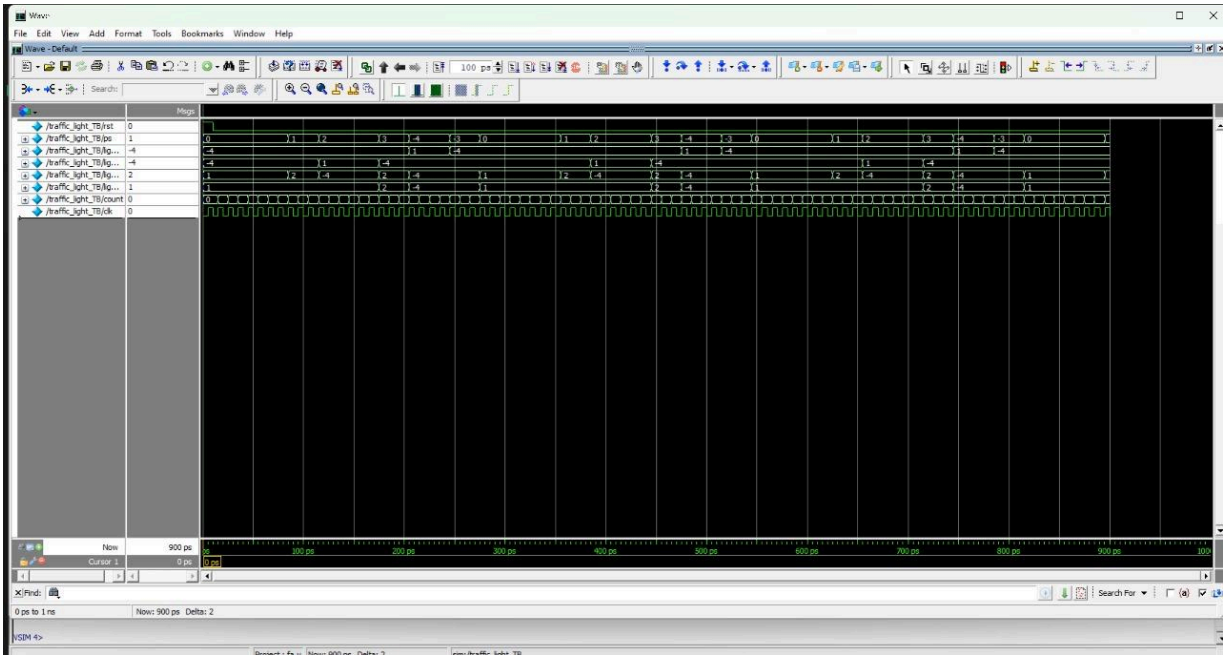
```
      light_S  <= 3'b100;
    end
    default: begin
      light_M1 <= 3'b000;
      light_M2 <= 3'b000;
      light_MT <= 3'b000;
      light_S  <= 3'b000;
    end
  endcase
 end

endmodule

module traffic_light_TB;
 reg clk, rst;
 wire [2:0] light_M1;
 wire [2:0] light_S;
 wire [2:0] light_MT;
 wire [2:0] light_M2;
 wire [3:0] count;
 wire [2:0] ps;

 // Instantiate traffic_light
 traffic_light dut (
  .clk(clk),
  .rst(rst),
  .light_M1(light_M1),
  .light_S(light_S),
  .light_MT(light_MT),
  .light_M2(light_M2),
  .count(count),    // Connect count to the testbench
  .ps(ps)           // Connect ps to the testbench
 );

 // Clock generation
 initial begin
```

```
    clk = 1'b0;
    forever #5 clk = ~clk; // Toggle clock every 5 time units for simulation
  end

  // Test stimulus
  initial begin
    rst = 1;        // Start with reset high
    #10 rst = 0;    // Release reset after 10 time units
    #2000 $finish;  // End simulation after a set time (adjust as needed)
  end
endmodule
```

RTL VIEW OF THE PROJECT :



STIMULATED WAVEFORM :



REAL LIFE PROJECT STIMULATION IDEA :

To implement this traffic light control simulation in a real-life system, you can combine microcontrollers, LEDs, sensors, and power management to create a functional model that controls traffic lights at a small-scale intersection. Here is a suggested approach :

1. **Hardware Components :**
    a. *Microcontroller :* Choose a microcontroller like Arduino, STM32, or a similar platform to handle the FSM logic and timing controls. This microcontroller will execute the FSM logic developed in your software simulation.
    b. *LED Lights :* Utilize red, yellow, and green LEDs to represent the traffic lights for each direction. For larger implementations,

consider using LED traffic light modules that are commonly found in real-world traffic systems.

c. *Counter and Timing Modules :* Use the microcontroller's internal clock or external timer modules to manage timing. This will enable precise control over the light durations as outlined in the FSM model.

d. *Power Supply :* Ensure you have a reliable power source, such as a DC adapter or battery pack, to maintain continuous operation.

2. Sensors and Inputs :

a. *Vehicle and Pedestrian Sensors (Optional) :* To enhance the system, consider adding infrared or ultrasonic sensors to detect the presence of vehicles. This feature would enable adaptive traffic control, allowing the system to adjust signal timings based on real-time traffic conditions.

b. *Push Buttons for Pedestrian Crossings :* Incorporate push-button inputs to simulate pedestrian crossing signals. When a pedestrian presses the button, it initiates a state change that temporarily halts vehicle movement.

3. Software and Logic Implementation :

a. *FSM Coding in Microcontroller :* Write the Moore FSM model in C or another compatible language for the microcontroller. Each state should be represented as a case in a switch statement, with specific outputs (like LED signals) and a delay counter linked to each case.

b. *State Transitions :* Set the duration for each state using counters, similar to your simulation. For instance, the FSM will remain in the "Green NS, Red EW" state for 10 seconds, counting down before moving to the next state.

c. *Testing and Debugging :* Execute the program on the microcontroller and check that the LEDs change states as intended, following the correct timing sequence without any issues.

4. Deploying a Test Intersection :

a. *Prototype Intersection Setup :* Create a small model of an intersection, either as a scaled-down version on a tabletop or an outdoor setup, to illustrate various traffic directions.

b. *Mount LEDs and Sensors :* Position the LEDs at the "traffic lights" on the model intersection, and if applicable, place sensors along the lanes where vehicles would travel.

c. *Run the System :* Turn on the microcontroller and watch the traffic lights cycle through the states according to your Moore FSM and delay counters.

5. Expanding to a Real-Time Adaptive Control :

a. *Traffic Density Sensing :* Consider adding more FSM states to manage changes in traffic density. By using sensors, you can create conditional state transitions that respond to real-time data. For instance, if no vehicles are detected on a particular road, the FSM could bypass certain states or modify the green light duration to enhance traffic flow efficiency.

b. *Networking and Central Control :* In more extensive systems, you might connect several microcontrollers to manage multiple intersections, effectively simulating an urban traffic control network.