Traffic Light Controller Finite State Machine

By Ganesh Manjunath

Net ID: dal542146

EECT/CE 6325 - VLSI Design

Project Partner

Aryan Verma

Net ID: dal820508

Instructor

Prof. Carl Sechen

Date

09-10-2025

Introduction

This project design implements a finite state machine (FSM) for a traffic light controller that manages a four-way intersection, alternating traffic flow between North-South (NS) and East-West (EW) directions to ensure safety and efficiency. The system utilizes synchronous flip-flops for state registration and a 6-bit counter for timing control, cycling through four encoded states: NS Green (lasting 30 clock cycles), NS Yellow (5 cycles), EW Green (30 cycles), and EW Yellow (5 cycles). Inputs include a clock signal (clk) for synchronization, a synchronous reset (rst) that initializes the FSM to the NS Green state, and an unused sensor input for future extensions like adaptive traffic detection. Outputs consist of control signals for the red, yellow, and green lights in both NS and EW directions, with combinational logic ensuring no simultaneous green lights across perpendicular paths to prevent conflicts. Implemented in behavioral Verilog, the design is scalable, with the RTL schematic illustrating key components such as multiplexers for state selection and output decoding, comparators for timing thresholds, an adder for counter increment, and registers for state and counter storage, all interconnected to form a robust, synchronous digital circuit.

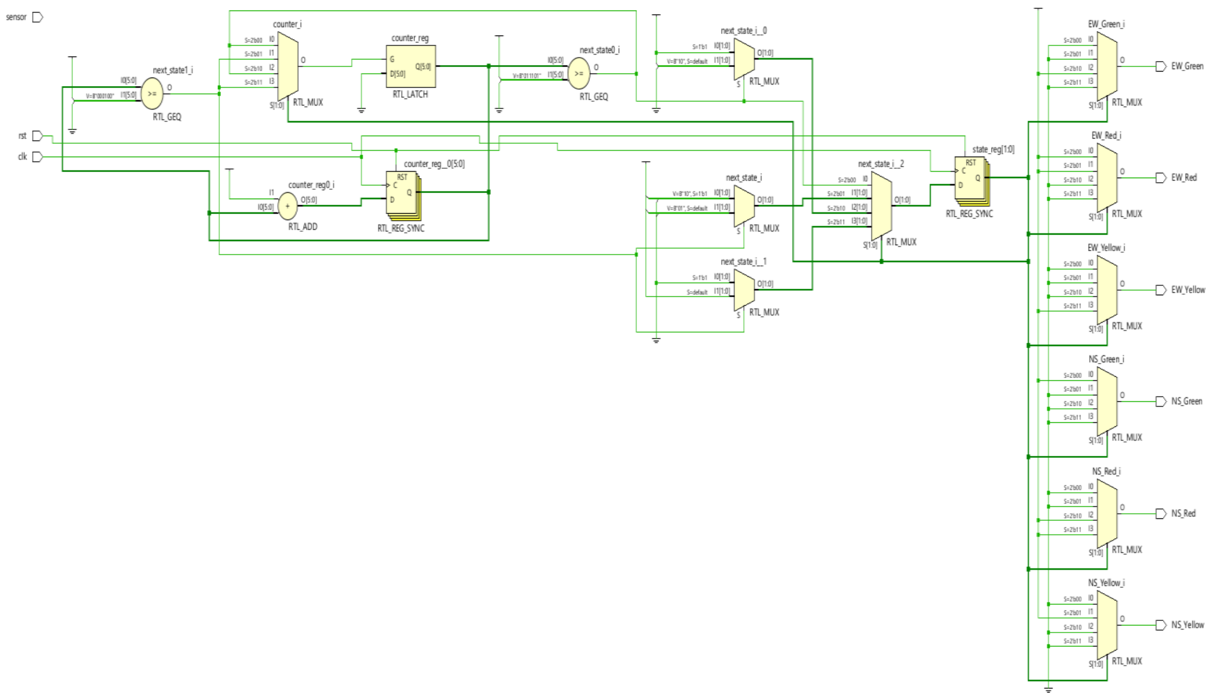
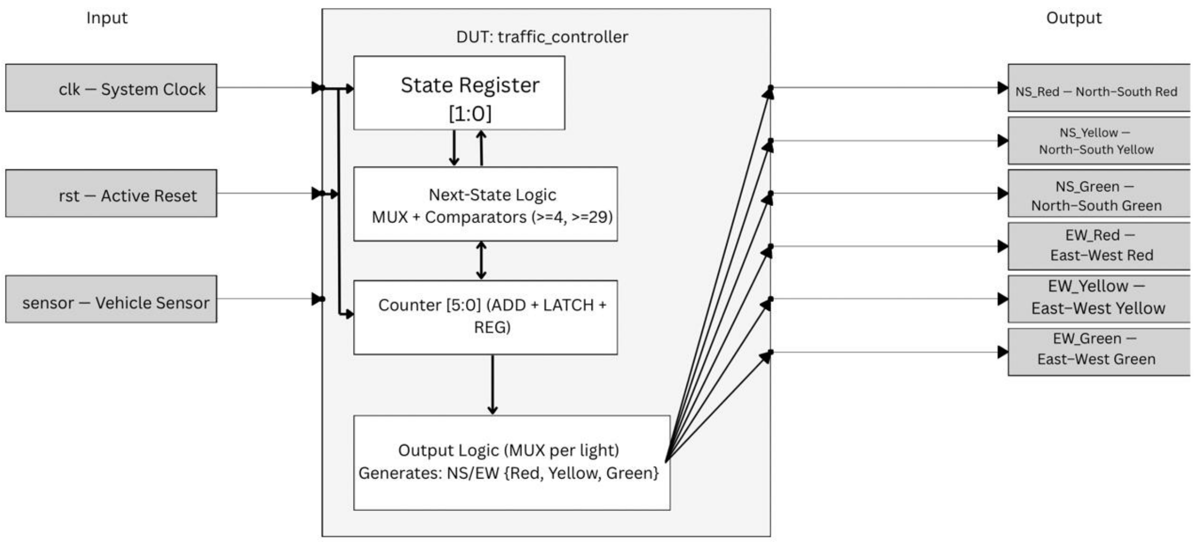


Figure 1: RTL Design

Design Block



Input
clk – System Clock
rst – Active Reset
sensor – Vehicle Sensor
DUT: traffic_controller
State Register [1:0]
Next-State Logic MUX + Comparators (>=4, >=29)
Counter [5:0] (ADD + LATCH + REG)
Output Logic (MUX per light) Generates: NS/EW {Red, Yellow, Green}
Output
NS_Red – North–South Red
NS_Yellow – North–South Yellow
NS_Green – North–South Green
EW_Red – East–West Red
EW_Yellow – East–West Yellow
EW_Green – East–West Green

Figure 2: Traffic Controller Design Block

The block diagram of the traffic light controller begins with three key inputs: the system clock, the reset signal, and the sensor. The clock is the driving force that synchronizes all operations inside the circuit, ensuring the state changes and counters update in a regular rhythm. The reset signal is used to initialize the system, forcing the controller into its starting condition, which is North–South green. The sensor input is included for scalability and represents adaptive detection, though in the base version of the design it is not actively used.

At the heart of the diagram lies the finite state machine, represented by the state register, the counter, the comparators, and the next-state logic. The state register stores the current state of the traffic lights, whether it is North–South green, North–South yellow, East–West green, or East–West yellow. A six-bit counter runs alongside the state register, increasing on every clock cycle to measure how long each state should last. The comparators connected to the counter check if the required time has been reached—for example, whether the green period has lasted thirty cycles or the yellow period has lasted five cycles. This timing information is fed into the next-state logic, implemented with multiplexers, which decides the upcoming state and ensures a smooth sequence from green to yellow and then to the opposite road's green.

Finally, the output logic in the block diagram converts the current state into actual control signals for the traffic lights. Multiplexers associated with each lamp decide which ones should be active, ensuring that only the correct lights are on at any given time. For instance, when the system is in the North–South green state, the North–South green lamp and the East–West red lamp is activated, while all others remain off. In the East–West yellow state, the East–West yellow and North–South red lights are turned on. The result of this arrangement is six output signals—red, yellow, and green for both directions—that coordinate the safe and timed flow of traffic at the intersection.

<u>Test Bench Block</u>

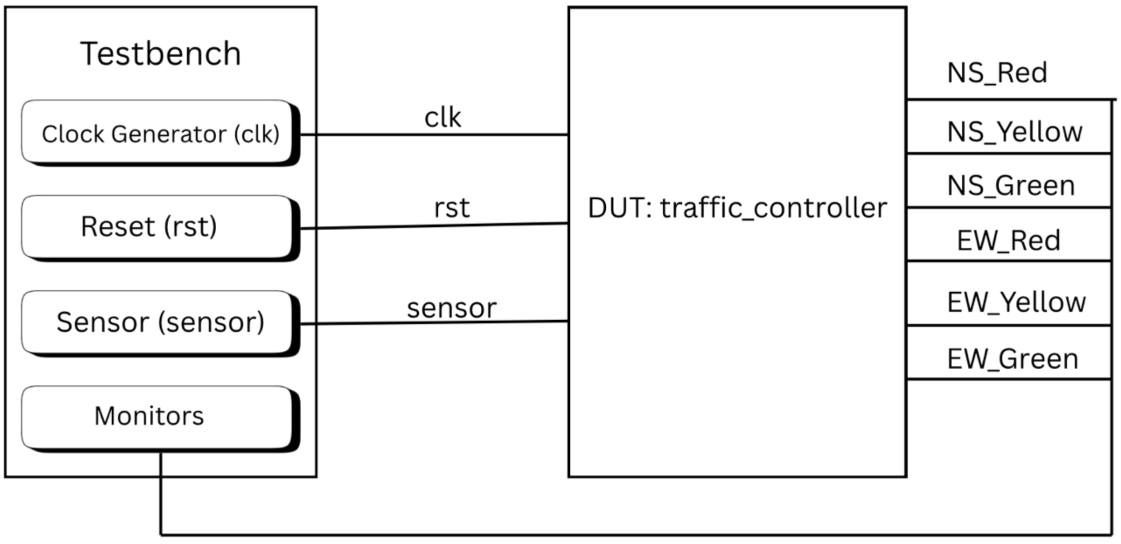


Figure 3: Test Bench Block along with the Design Block

This block diagram shows how the testbench communicates with the design under test (DUT), which in this case is the traffic light controller module. The testbench acts as a virtual environment that supplies all necessary inputs to the DUT and monitors its outputs to verify correct behavior. On the left side, the testbench has four key components: a clock generator, a reset generator, a sensor input, and monitors.

The clock generator produces the clk signal that drives the sequential logic inside the DUT, such as the state register and counter. The reset block controls the rst signal, which initializes the DUT into a known starting condition (North–South green). The sensor input is included for completeness and is connected to the DUT, even though it may not actively affect the base design. These three signals (clk, rst, and sensor) are passed from the testbench into the DUT as inputs.

On the right-hand side of the DUT, the outputs are the six traffic light signals: NS_Red, NS_Yellow, NS_Green, EW_Red, EW_Yellow, and EW_Green. These outputs are fed back to the monitors in the testbench, which record and display the signals during simulation. In this way, the testbench provides stimulus to the traffic controller and observes its responses, ensuring that the design cycles through the expected light patterns correctly.

Results

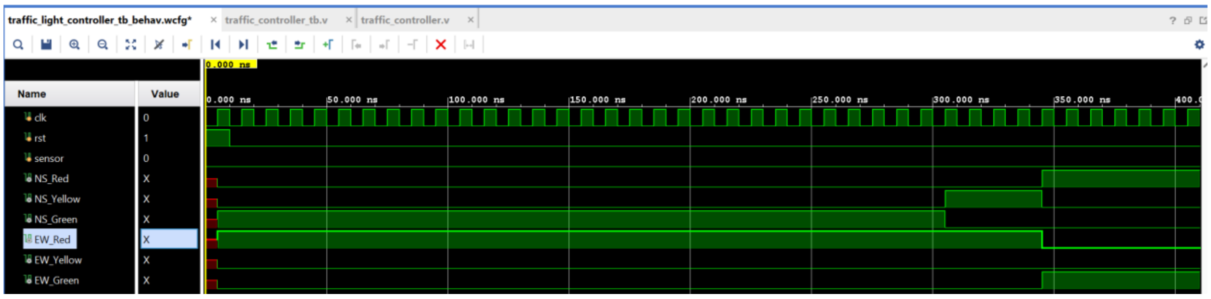


Figure 4: Simulation Waveform

The following is a textual representation and analysis of the simulation waveform generated from the testbench (traffic_light_controller_tb.v) running the design (traffic_light_controller.v) in a tool like ModelSim or Vivado Simulator. The waveform spans from 0 ns to approximately 400 ns, capturing the initial reset and part of the FSM cycle (NS Green → NS Yellow → EW Green → partial EW Yellow).

- Initialization (0-10 ns): Signals start undefined (X) due to uninitialized registers. On the first posedge clk (~5 ns) with rst high, the system resets to NS_GREEN state, setting NS_Green and EW_Red high, others low. This confirms synchronous reset functionality as per the sequential always block.
- NS_GREEN Phase (10-310 ns, ~300 ns = 30 clock cycles): NS_Green and EW_Red remain high, allowing North-South traffic while stopping East-West. Counter increments from 0 to 29 without transition, verifying the combinational logic's if-condition (counter < GREEN_TIME-1) keeps next_state as NS_GREEN. No overlapping greens, ensuring safety.
- Transition to NS_YELLOW (~310 ns): When counter >=29, next_state updates to NS_YELLOW on the next posedge clk, and counter resets to 0. NS_Green drops low, NS_Yellow goes high, EW_Red stays high. This demonstrates correct timing and state transition logic in the case statement.
- NS_YELLOW Phase (310-360 ns, ~50 ns = 5 clock cycles): NS_Yellow high for caution, with counter incrementing to 4. Outputs match the NS_YELLOW case in the output logic, confirming combinational decoding from state 2'b01.
- Transition to EW_GREEN (~360 ns): Counter >=4 triggers next_state to EW_GREEN, counter reset. NS_Yellow low, EW_Red low, EW_Green high, NS_Red high. This shows the FSM advancing correctly without glitches.
- EW_GREEN Phase (360-400 ns+, partial): EW_Green and NS_Red high, allowing East-West flow. The partial view aligns with the testbench's #400 ns duration, but if extended, it would last 300 ns before yellow. Sensor remains low and unused, as designed.
- Overall Functionality: The waveform proves the design works: Timed state cycles (30/5 cycles), safe light controls (opposing reds during greens), and reset to initial state. No errors like stuck states or invalid outputs, validating the FSM for the four-way intersection control.

Code

Verilog Design Code :

```
module traffic_light_controller (
    input clk,            // Clock input
    input rst,            // Synchronous reset
    input sensor,         // Traffic sensor (for future scalability, unused in base design)
    output reg NS_Red,     // North-South Red light
    output reg NS_Yellow,   // North-South Yellow light
    output reg NS_Green,    // North-South Green light
    output reg EW_Red,      // East-West Red light
    output reg EW_Yellow,   // East-West Yellow light
    output reg EW_Green     // East-West Green light
);

    // State encoding
    localparam NS_GREEN  = 2'b00;
    localparam NS_YELLOW = 2'b01;
    localparam EW_GREEN  = 2'b10;
    localparam EW_YELLOW = 2'b11;

    // Timing parameters (scalable)
    localparam GREEN_TIME  = 30; // Green light duration (clock cycles)
    localparam YELLOW_TIME = 5;  // Yellow light duration (clock cycles)

    // Registers (using flip-flops)
    reg [1:0] state, next_state;
    reg [5:0] counter; // Counter for timing (6 bits for up to 64 cycles)

    // Sequential logic: State and counter update (flip-flops)
    always @(posedge clk) begin
    if (rst) begin
        state <= NS_GREEN; // Reset to NS Green state
        counter <= 0;
    end else begin
        state <= next_state; // Update state
        counter <= counter + 1; // Increment counter
    end
end

// Combinational logic: Next state and counter reset
always @(*) begin
    case (state)
        NS_GREEN: begin
            if (counter >= GREEN_TIME - 1) begin
                next_state = NS_YELLOW;
                counter = 0; // Reset counter
            end else begin
                next_state = NS_GREEN;
            end
        end
        NS_YELLOW: begin
            if (counter >= YELLOW_TIME - 1) begin
                next_state = EW_GREEN;
                counter = 0;
            end else begin
                next_state = NS_YELLOW;
            end
        end
        EW_GREEN: begin
            if (counter >= GREEN_TIME - 1) begin
                next_state = EW_YELLOW;
            counter = 0;
         end else begin
            next_state = EW_GREEN;
         end
      end
      EW_YELLOW: begin
         if (counter >= YELLOW_TIME - 1) begin
            next_state = NS_GREEN;
            counter = 0;
         end else begin
            next_state = EW_YELLOW;
         end
      end
      default: next_state = NS_GREEN;
   endcase
end

// Output logic: Control traffic lights
always @(*) begin
   // Default: All lights off
   NS_Red = 0; NS_Yellow = 0; NS_Green = 0;
   EW_Red = 0; EW_Yellow = 0; EW_Green = 0;

   case (state)
      NS_GREEN: begin
         NS_Green = 1;
         EW_Red = 1;
      end
      NS_YELLOW: begin
         NS_Yellow = 1;
         EW_Red = 1;
```

```
            end
            EW_GREEN: begin
                EW_Green = 1;
                NS_Red = 1;
            end
            EW_YELLOW: begin
                EW_Yellow = 1;
                NS_Red = 1;
            end
        endcase
    end

endmodule
```

<u>Test Bench Code :</u>

```
module traffic_light_controller_tb;

    // Testbench signals
    reg clk, rst, sensor;
    wire NS_Red, NS_Yellow, NS_Green;
    wire EW_Red, EW_Yellow, EW_Green;

    // Instantiate the controller
    traffic_light_controller uut (
        .clk(clk),
        .rst(rst),
        .sensor(sensor),
        .NS_Red(NS_Red),
        .NS_Yellow(NS_Yellow),
    .NS_Green(NS_Green),
    .EW_Red(EW_Red),
    .EW_Yellow(EW_Yellow),
    .EW_Green(EW_Green)
);

// Clock generation: 10ns period
initial begin
    clk = 0;
    forever #5 clk = ~clk;
end

// Test procedure
initial begin
    // Initialize inputs
    rst = 1;
    sensor = 0; // Sensor unused in base design
    $display("Starting Traffic Light Controller Test");

    // Apply reset
    #10 rst = 0;
    $display("Reset applied");

    // Simulate for enough cycles to observe all states (2 full cycles)
    #400; // 2 * (30 + 5 + 30 + 5) * 10ns = 280ns + extra for observation
    $display("Test completed");

    // Finish simulation
    $finish;
end
```

```
    // Monitor state transitions and outputs
    always @(posedge clk) begin
        $display("Time=%0t: State=%0b, NS(R,Y,G)=%b,%b,%b, EW(R,Y,G)=%b,%b,%b",
                 $time, uut.state, NS_Red, NS_Yellow, NS_Green, EW_Red, EW_Yellow, EW_Green);
    end

    // Generate VCD file for waveform viewing
    initial begin
        $dumpfile("traffic_light.vcd");
        $dumpvars(0, traffic_light_controller_tb);
    end

endmodule
```