Project 84: Traffic Light Controller

A Comprehensive Study of Advanced Digital Circuits

By: Abhishek Sharma , Ayush Jain , Gati Goyal, Nikunj Agrawal

Documentation Specialist: Dhruv Patel & Nandini Maheshwari

Contents

1 Project Overview

A Traffic Light Controller (TLC) serves as an integral component in traffic management systems, utilizing digital circuits to automate light signals at intersections. This reduces human dependency and ensures systematic traffic regulation. The provided elevator control system design offers valuable insights into state-based logic, timing, and synchronization that can be leveraged in TLC development. Similar to the FSM-based operation in the elevator control system, a TLC manages transitions between states (e.g., RED, YELLOW, GREEN) and adjusts the output based on real-time or predefined conditions.

2 Traffic Light Cotroller

2.1 Key Components of Traffic Light Controller

1. Finite State Machines (FSMs):

Definition: FSMs are models of computation with a finite number of states. Each state represents a unique configuration of the system, transitioning based on inputs or conditions.

Relevance: The elevator control system transitions between IDLE, MOVING_UP, MOVING_DOWN, and OPEN_DOOR. Similarly, the TLC cycles through RED, GREEN, and YELLOW.

2. Inputs and Outputs:

Inputs: Sensors for traffic density, pedestrian crossing buttons, and timing signals act as the primary control points for the TLC.

Outputs: Signals to control light bulbs for various directions or pedestrian crossings.

3. Synchronous Timing:
The TLC synchronizes state transitions using a clock signal, ensuring precise timing of light changes. This approach is also evident in the elevator design's clock-driven state transitions.

4. Hierarchy of States:
Just as the elevator system has primary states and sub-behaviors, the TLC can include advanced states, such as emergency vehicle overrides or adaptive timing for traffic surges.

2.2 Working of Traffic Light Controller

1. Finite State Machine (FSM):
The system uses FSM to manage states: RED, YELLOW, and GREEN for traffic lights. States transition based on timers or sensor inputs.

2. Clock-Driven State Transitions:
A clock signal synchronizes the system, ensuring precise timing for state transitions.

3. Inputs and Outputs:
Inputs: Clock, reset signal, and optional sensors for traffic density or pedestrian crossings.
Outputs: Signals to control RED, YELLOW, and GREEN lights for each traffic direction.

4. State Logic:
RED: Stops vehicles while other directions operate or pedestrians cross.
GREEN: Allows vehicles to proceed.
YELLOW: Alerts vehicles to prepare for stopping.

5. Transition Conditions:

Timer expiration or sensor input triggers transitions between states.
Example: From RED to GREEN after a predefined time.

6. Synchronous Operation:
All actions, such as light changes, are synchronized with the clock signal for smooth operation.

7. Initialization:
On reset, the system starts in a predefined safe state, usually all directions on RED.

8. Simulation and Testing:
Functional correctness is verified through a SystemVerilog testbench in Vivado.
Outputs like timing diagrams and state transitions are analyzed.

9. Synthesis and Hardware Implementation:
SystemVerilog code is synthesized into a hardware design for FPGA or ASIC implementation.

10. Scalability:
The design is modular, allowing easy expansion for more traffic directions, pedestrian crossings, or adaptive traffic control features.

2.3 RTL Code

Listing 1: Traffic Light Controller

```
1
2
3  module traffic_light_controller (
4      input logic clk, reset,
5      output logic [2:0] lights // {Red, Yellow, Green}
6  );
7
8      typedef enum logic [1:0] {
9          RED = 2'b00,
10         GREEN = 2'b01,
11         YELLOW = 2'b10
12     } state_t;
13
14     state_t state, next_state;
15     logic [3:0] counter;
16
17     // State transition logic
18     always_ff @(posedge clk or posedge reset) begin
19         if (reset) begin
20             state <= RED;
21             counter <= 0;
22         end else begin
23             if (counter == 4'd9) begin
24                 state <= next_state;
25                 counter <= 0;
26             end else
27                 counter <= counter + 1;
28         end
29     end
30
31     // Next state logic
32     always_comb begin
33         case (state)
```

```
            RED:    next_state = GREEN;
            GREEN:  next_state = YELLOW;
            YELLOW: next_state = RED;
            default: next_state = RED;
        endcase
    end

    // Output logic
    always_comb begin
        case (state)
            RED:    lights = 3'b100;
            GREEN:  lights = 3'b001;
            YELLOW: lights = 3'b010;
            default: lights = 3'b100;
        endcase
    end

endmodule
```

2.4 Testbench

Listing 2: Traffic Light Controller

```

module tb_traffic_light_controller();
    logic clk, reset;
    logic [2:0] lights;

    traffic_light_controller uut (
        .clk(clk),
        .reset(reset),
        .lights(lights)
    );

    // Clock generator
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    // Test sequence
    initial begin
        reset = 1;
        #10 reset = 0;
        #200 $stop;
    end

    // Monitor outputs
    initial begin
        $monitor("Time: %0t | Reset: %b | Lights: %b", $time, reset,
           lights);
    end
endmodule
```

3 Results

3.1 Simulation

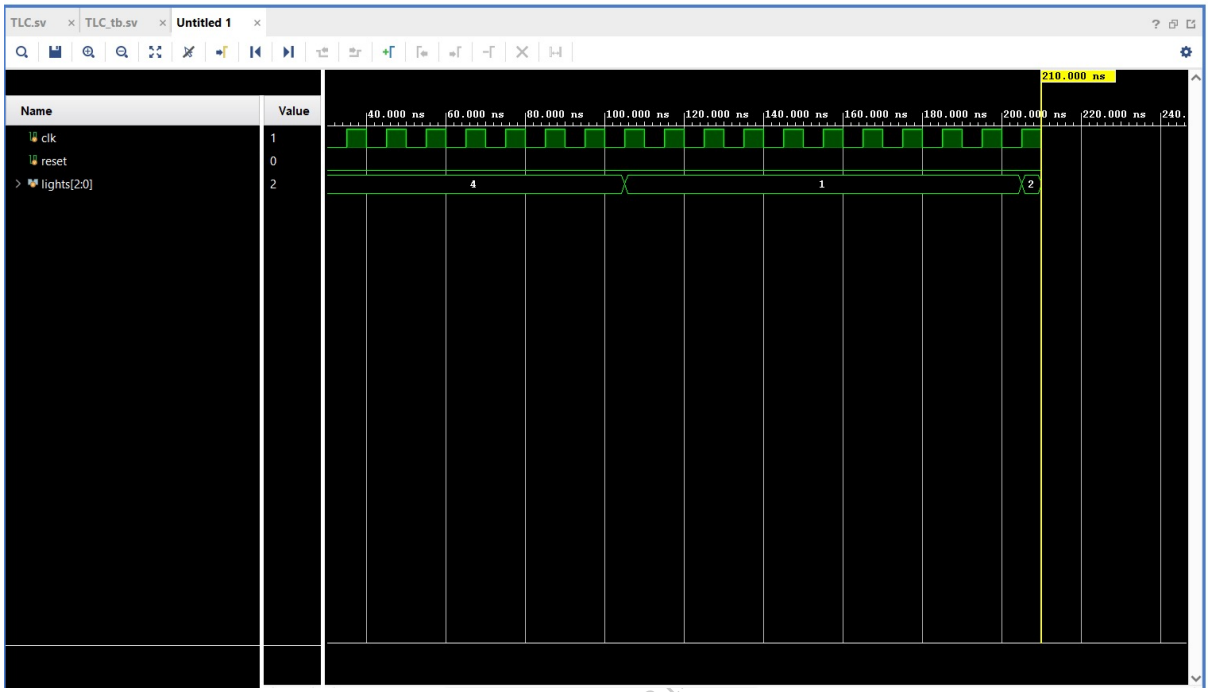

Figure 1: Simulation of Traffic Light Controller

3.2 Schematic

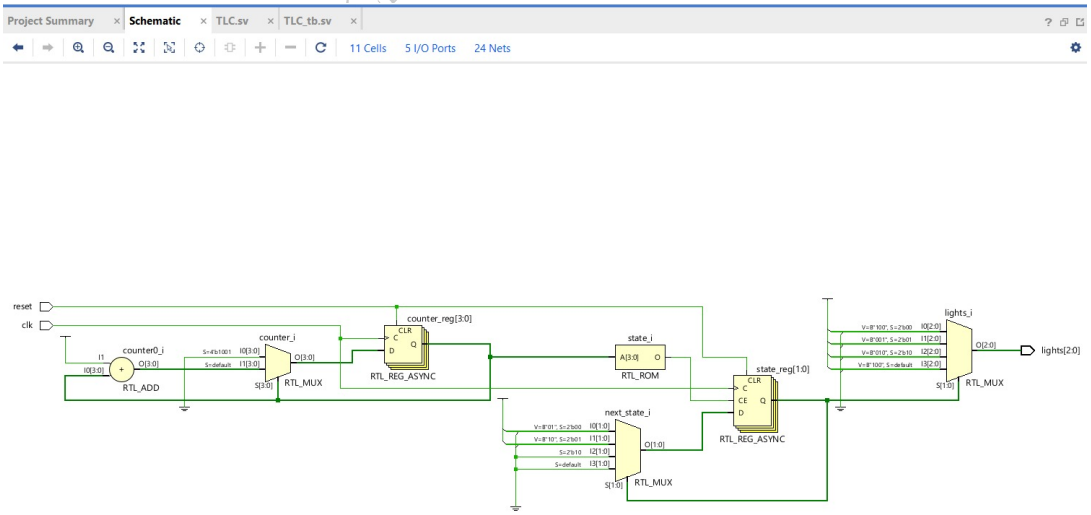

Figure 2: Schematic of Traffic Light Controller

3.3 Synthesis Design

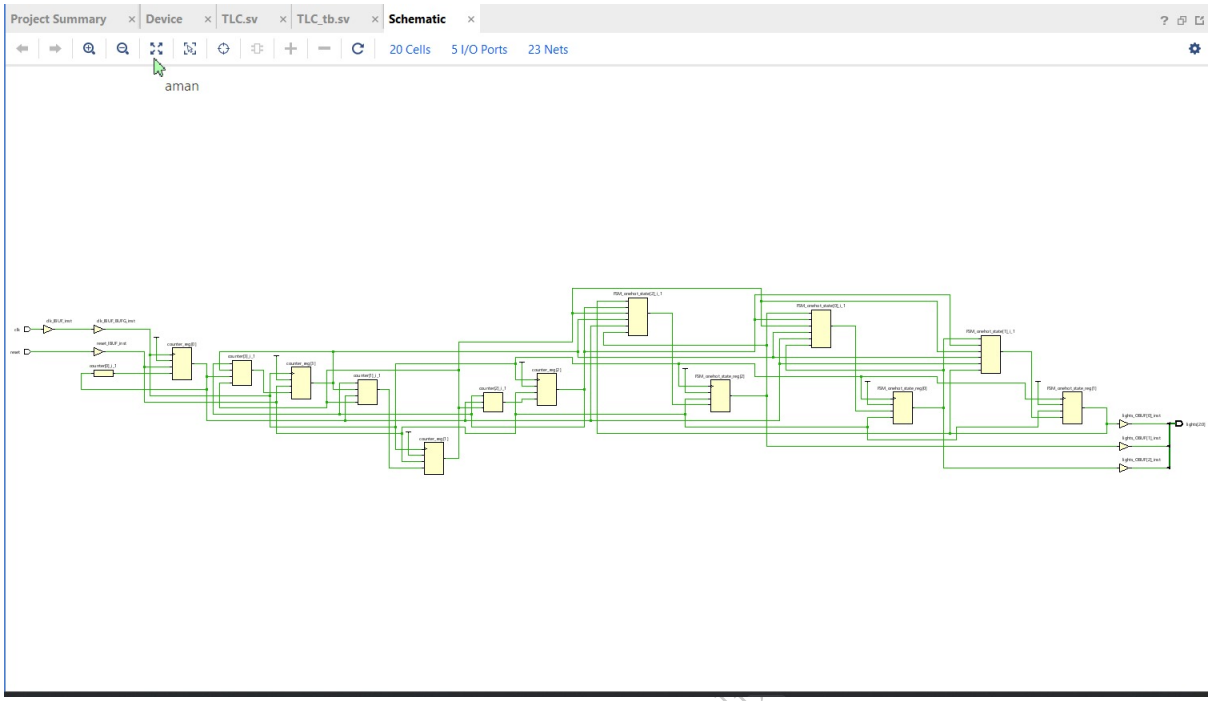

Figure 3: Synthesis Design of Traffic Light Controller

4 Advantages of Traffic Light Controller

Improves Traffic Flow: Ensures orderly movement at intersections.

Reduces Accidents: Minimizes human error with automated control.

Customizable: Easily adjustable for intersection complexity or traffic patterns.

Energy Efficiency: Optimized systems can save power by turning off lights when no vehicles are present.

Reliability: FSM-based designs ensure predictable and error-free operation.

Scalability: New features, such as pedestrian crossings or smart sensors, can be easily integrated.

Real-Time Operation: Synchronous systems guarantee timely response to inputs.

5 Disadvantages of Traffic Light Controller

Cost: Initial setup and maintenance can be expensive.

Sensor Dependency: Faulty sensors may disrupt the system.

Delays: Fixed timing may not account for varying traffic density.

Complexity in Large Intersections: As the number of roads and sensors increases, the state diagram and logic complexity grow.

Dependency on External Systems: Malfunctioning sensors or timers can disrupt traffic flow.

6 Applications of Traffic Light Controller

Urban Intersections: To manage complex, high-density traffic.

Pedestrian Crossings: Enhances safety for pedestrians.

Smart Cities: Integrated with IoT for adaptive traffic control.

Emergency Routing: Can prioritize emergency vehicles.//
Automated Toll Booths: Adjust signals based on vehicle detection.

Emergency Vehicle Routes: Priority green lights for ambulances and fire trucks.

Airports and Railway Crossings: Traffic control for service vehicles.

7 Conclusion

The Traffic Light Controller is a practical and essential application of digital logic and SystemVerilog. By utilizing concepts such as finite state machines (FSMs), synchronous timing, and modular design, the system ensures efficient and safe traffic management. The project demonstrates the power of digital design in automating real-world tasks while providing flexibility for enhancements like adaptive timing and IoT integration.

8 FAQs

1. What is the role of FSMs in a traffic light controller?

FSMs organize states (RED, YELLOW, GREEN) and define transitions based on timers or inputs.

How does the clock signal influence system operation?

The clock synchronizes state transitions, ensuring timing precision.

2. What are the similarities between your elevator control code and the traffic light controller?

Both use FSMs, state transition logic, and synchronous outputs for control.

3. How would you handle pedestrian crossing signals in the design?

Add states for pedestrian crossings, triggered by button inputs or sensors.

4. What additional inputs can enhance traffic light functionality?

Traffic density sensors, emergency vehicle detection, and adaptive timing inputs.

5. Explain the importance of reset in synchronous designs.

Initializes the system to a known state, preventing undefined behavior.

6. How can traffic light controllers adapt to real-time traffic conditions?

Using sensors and IoT integration for dynamic timing adjustments.

7. What optimization techniques can you apply to a traffic light controller?

Minimize state transitions, use energy-efficient lighting, and simplify logic.

8. What is the role of synthesis in your SystemVerilog project?

Converts high-level designs into hardware implementation for FPGA or ASIC.

9. How do you test the functionality of your traffic light controller?

Using a testbench to simulate real-world scenarios and validate outputs.

10. What are the key differences between sequential and combinational designs?

Sequential systems rely on state memory; combinational systems respond solely to inputs.

11. How do you ensure fault tolerance in your design?

Add redundancy in critical sensors and implement default safe states.

12. What debugging techniques would you use for your project?

Waveform analysis, step-by-step simulation, and assertions in the testbench.

13. Can your traffic light controller design be extended for smart city applications?

Yes, by integrating IoT and vehicle communication systems for adaptive traffic control.

14. What lessons did you learn from implementing the elevator control system that are applicable to this project?

FSM modularity, importance of initialization, and state-driven output logic are crucial for reliable operation.