International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 14, Issue 1, 2026

Traffic Light Controller Using Verilog HDL

Dr Pradeep Kumar Goud N[1], Chegu SaiHarshitha[2],Kola Sowjanya[3],Revanala Yamuna[4]

[1]Associate Professor; Department Of Electronics And Communication Engineering, Bhoj Reddy Engineering College For Women, Hyderabad, India.

[2,3,4]B.Tech Students; Department Of Electronics And Communication Engineering, Bhoj Reddy Engineering College For Women, Hyderabad, India.

saiharshithachegu@gmail.com

Abstract

Urban transportation systems are becoming increasingly complex due to rapid growth in vehicle density and expanding road infrastructure. Conventional traffic control approaches, including manual operation and fixed-time signal scheduling, are often unable to respond effectively to fluctuating traffic conditions, leading to congestion, longer waiting times, and reduced roadway efficiency. To address these limitations, this work proposes the design and hardware implementation of a Traffic Light Controller tailored for a T-junction using Verilog Hardware Description Language and synthesized on an FPGA platform through AMD Vivado. The proposed controller is based on a Moore-model Finite State Machine (FSM), enabling deterministic and well-defined transitions among traffic signal phases—green, yellow, and red. The control strategy assigns priority to the main road while ensuring systematic signal allocation to the side road through a structured timing sequence. The overall architecture is modular, comprising dedicated Verilog components for clock division, state transition logic, and output signal generation. This modular approach enhances readability, maintainability, and potential reuse in related designs.Functional validation was performed using behavioral simulation and waveform analysis within the Vivado environment to confirm timing accuracy and logical correctness. The results demonstrate reliable sequencing of traffic signals with synchronous state transitions driven by clock-defined intervals. The design maintains low latency and efficient hardware resource utilization, making it suitable for implementation on low-power FPGA development boards.The developed system provides a scalable and dependable solution for hardware-based traffic management. Its FSM-driven architecture supports future extensions, including sensor-based adaptive control, coordination among multiple intersections, and integration into intelligent transportation frameworks. Consequently, the proposed implementation serves as a practical prototype for real-time, embedded traffic control applications in modern urban environments.

Keywords— *Traffic Light Controller, Finite State Machine (FSM), Verilog HDL, FPGA Implementation, T-Junction Traffic Control, AMD Vivado*

INTRODUCTION

Traffic control systems are fundamental to maintaining safe and orderly movement of vehicles and pedestrians at road intersections, highways, and urban transportation networks. With the rapid growth of urban populations and increasing vehicle density, efficient traffic management has become an essential requirement for modern cities. Inefficient signal coordination often leads to congestion, increased travel time, fuel wastage, and elevated accident risks.Traditional traffic control mechanisms commonly rely on fixed-time signal operations or manual regulation by traffic personnel. These approaches lack the flexibility required to respond to real-time variations in traffic flow. Fixed timers may allocate unnecessary green time to empty roads while congested lanes experience delays. Additionally, manual operation introduces variability due to human fatigue, delayed reactions, and inconsistent decision-making, especially at complex junctions.Contemporary traffic control solutions incorporate digital electronics, embedded systems, and automated decision logic to improve reliability and responsiveness. These systems operate through predefined sequences of signal states—Red, Yellow, and Green—which regulate vehicular movement. The timing of these states may be static or dynamically adjusted based on environmental inputs, sensors, or programmed logic.Hardware-oriented implementations using Field Programmable Gate Arrays (FPGAs) and Hardware Description Languages (HDLs) such as Verilog have emerged as efficient alternatives for traffic control design. FPGA-based controllers offer deterministic execution, parallel processing capability, and minimal latency, making them suitable for safety-critical applications. Moreover, HDL-based designs support modular development, easy verification, and reconfiguration, which are advantageous for both prototyping and deployment.

Challenges in Manual Traffic Management

Although manual traffic control has historically served as a basic solution, it presents several limitations in modern transportation systems. These challenges affect operational efficiency, safety, and overall commuter experience.

One major drawback is inconsistency in signal management. Human operators may experience fatigue or distraction, leading to irregular timing

decisions and delayed responses. Such inconsistencies can cause unnecessary congestion, uneven traffic distribution, and increased accident probability at intersections.

Another limitation is the inability to adapt effectively to fluctuating traffic volumes. Manual systems generally depend on fixed schedules or operator judgment, which may not reflect real-time conditions such as peak-hour demand, special events, or emergency situations. As a result, busy roads may suffer from prolonged delays while less congested routes remain underutilized.

Safety issues also arise due to poor synchronization between multiple traffic points. Coordinating signal transitions across interconnected intersections manually is difficult, which increases the likelihood of conflicting signals and unsafe traffic movements. These shortcomings highlight the need for automated and deterministic traffic control mechanisms.

Motivation for FSM-Based Automation

The complexity of traffic management systems requires structured and reliable control strategies. Finite State Machines (FSMs) provide an effective methodology for modeling sequential control systems, making them well suited for traffic signal automation.FSM-based design ensures deterministic behavior, where each traffic condition corresponds to a predefined state with clearly defined transitions. This structured approach eliminates ambiguity in signal sequencing and enhances operational safety. By representing traffic phases as discrete states, designers can precisely control transitions based on timing or external inputs.Another advantage of FSM implementation is modularity. Traffic phases such as green, yellow, and red can be represented as individual states, simplifying design and verification. The modular nature of FSMs also supports easier debugging, scalability, and future system expansion. These characteristics make FSM-based automation an ideal approach for implementing reliable traffic controllers.

Problem Statement: T-Intersection Traffic Flow

Traffic management at T-shaped intersections presents unique operational challenges due to the asymmetrical road configuration. Typically, a major road intersects with a minor road, requiring priority handling to maintain efficient traffic flow. Unlike four-way intersections, T-junctions demand customized signal sequencing to prevent delays and ensure safety.

The traffic controller must address the following requirements:

Provide sufficient green time for the main road to maintain continuous traffic flow.

Allow timely access for vehicles on the side road without excessive waiting.

Prevent simultaneous green signals that may cause collisions.

Maintain consistent and synchronized timing intervals to ensure predictable signal transitions.

These constraints necessitate a carefully designed control system capable of balancing efficiency and safety.

Methodology

The project follows a structured methodology to design and implement the traffic light controller:

RequirementAnalysis

Traffic flow at a T-intersection was studied to determine signal phases and timing requirements, with priority given to the main road.

FSMDesign

A Moore-type FSM was selected to represent signal states, with transitions governed by clock-based timing logic.

ModularVerilogCoding

The system was divided into independent modules, including:

Clock divider for generating timing intervals

FSM controller for state transitions

Output logic for driving traffic signals

FUNDAMENTALS OF FSM AND DIGITAL TRAFFIC CONTROL

Finite State Machines (FSMs) are widely used models for designing sequential logic systems in digital electronics, embedded control, and automation applications. An FSM represents system behavior using a limited number of states, where transitions between states occur according to defined rules and input conditions. Depending on the design methodology, the output of an FSM may depend only on the present state or on both the current state and external inputs. This structured approach allows designers to implement predictable and well-organized control logic, which is particularly important in applications requiring reliability and timing accuracy such as traffic light controllers.

FSMs are primarily categorized into two models: Moore machines and Mealy machines. In a Moore-type FSM, the output signals are determined exclusively by the current state of the system. As a result, output changes occur only when the state transitions, typically synchronized with a clock edge. This characteristic provides stable and glitch-free outputs, which simplifies timing analysis and enhances system reliability. In contrast, a Mealy-type FSM produces outputs based on both the current state and the input signals. This enables faster response to input changes because outputs can update immediately without waiting for a state transition. However, this flexibility can introduce timing complexity and potential output glitches, making verification more challenging in hardware implementations.

The Moore and Mealy models differ significantly in their operational behavior and design trade-offs. Moore FSMs offer improved output stability

because signals remain constant throughout each state, reducing the risk of transient errors. They also simplify synchronization with clock-driven systems, making them suitable for safety-critical applications. On the other hand, Mealy FSMs generally require fewer states since outputs react directly to input conditions, which can reduce hardware resources. Nevertheless, the asynchronous nature of output changes in Mealy machines increases the possibility of glitches and complicates timing analysis. Due to these considerations, traffic signal controllers often adopt the Moore model to ensure predictable operation and consistent signal transitions. In this work, a Moore-based FSM is selected to guarantee stable and deterministic traffic light behavior.

Traffic signal control strategies are generally categorized as time-driven or event-driven approaches. In time-driven systems, signal changes occur at fixed intervals determined during system design. These systems operate on predefined cycles and do not respond dynamically to real-time traffic conditions. The simplicity of this approach makes it easy to implement using FSM-based designs, and the behavior remains predictable under stable traffic patterns. However, fixed-time control may lead to inefficiencies when traffic volume fluctuates, causing unnecessary waiting times or underutilized road segments. Event-driven systems, in contrast, adjust signal timing based on sensor inputs such as vehicle detectors or traffic density measurements. Although event-driven control improves adaptability, it increases system complexity. The present work focuses on a time-driven FSM-based design to maintain simplicity and deterministic behavior while ensuring safe traffic flow at a T-intersection.

Finite State Machines offer several advantages in control applications, particularly in traffic signal systems. One major benefit is deterministic operation, where clearly defined states and transitions ensure predictable system responses. This characteristic is essential for maintaining safety in traffic control, where incorrect signal sequencing can lead to hazardous situations. FSMs also simplify system design by decomposing complex behavior into manageable states, making implementation straightforward using hardware description languages such as Verilog. Furthermore, FSMs enable comprehensive verification because each state and transition can be simulated and validated independently. Another advantage is scalability; additional states or transitions can be incorporated to extend functionality without redesigning the entire system. These properties make FSMs highly suitable for embedded control applications, including FPGA-based traffic light controllers.

MODIFIED TRAFFIC LIGHT CONTROLLER DESIGN

The proposed traffic light controller is developed to regulate vehicle movement at a T-shaped intersection using a deterministic Finite State Machine (FSM) implemented on an FPGA platform. The design focuses on predictable signal transitions, safe operation, and configurable timing control. The following specifications and assumptions define the system framework.

System Specifications

1. **IntersectionConfiguration**
   The controller targets a T-shaped junction consisting of one primary road and one secondary road intersecting at a right angle. Traffic from both roads must be coordinated to prevent conflicting movements.
2. **TrafficSignalArrangement**
   Each road is equipped with standard three-color traffic signals: Green, Yellow, and Red. These signals operate independently for the main and side roads to regulate vehicle movement.
3. **Signal Timing Parameters**

- **Green Phase:** Allocated to allow vehicles to pass safely on the active road.
- **Yellow Phase:** A short transition interval indicating an upcoming stop condition.
- **Red Phase:** Applied to halt traffic while the opposing road receives the right of way.

4. **ControlLogic**
   The FSM ensures sequential operation of traffic lights, allowing only one road to display a green signal at any given time. Yellow transitions are inserted between green and red states to maintain safe switching.
5. **ClockSource**
   The design operates using the onboard FPGA clock. A clock divider is used to derive lower-frequency timing signals from the high-frequency system clock (e.g., 50 MHz or 100 MHz) to control signal durations accurately.

Functional Requirements

The system must satisfy the following operational requirements:

1. Signal Sequencing

The controller cycles through traffic phases in a fixed repeating order:

Main Road Green → Main Road Yellow → Side Road Green → Side Road Yellow → Repeat

This sequence ensures continuous and orderly traffic flow.

2. Exclusive Green Condition

Only one direction is permitted to display a green signal at any time. The opposing direction must remain red to prevent vehicle conflicts.

3. Yellow Transition Interval

Each transition from green to red must include a yellow phase. This interval provides drivers with sufficient warning before stopping.

State Diagram for Traffic Flow

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

The FSM governing the controller consists of multiple states representing distinct traffic signal combinations. Each state corresponds to a specific configuration of main and side road signals. Transitions between states occur based on timer completion events driven by the system clock.
(Figure 3.4 illustrates the FSM state diagram representing signal transitions.)

SIMULATION AND VERIFICATION

Importance of Simulation in Digital Design

Simulation is an essential phase in the development of digital hardware systems, particularly for Field Programmable Gate Array (FPGA) and Application-Specific Integrated Circuit (ASIC) implementations. It enables designers to validate functionality, analyze timing behavior, and verify logical correctness prior to synthesis and hardware deployment. By examining the design at the Register Transfer Level (RTL), potential errors can be detected early, reducing development time and avoiding costly hardware-level debugging.
In this project, simulation is used to evaluate the behavior of the FSM-based traffic light controller and ensure that the design operates according to the specified requirements.

FunctionalValidation

Simulation confirms that the traffic controller transitions through the defined states correctly. It verifies the timing intervals, output signal combinations, and sequencing of traffic lights for both main and side roads.

ErrorIdentificationandDebugging

The simulation environment helps detect logical mistakes such as incorrect state transitions, improper signal assignments, or timing mismatches. Resolving these issues at the RTL stage simplifies the design process and improves reliability.

VisualizationofFSMOperation

Waveform viewers provide a graphical representation of signals over time, allowing observation of state changes and output behavior. This visualization ensures that each state persists for the intended duration and transitions occur as expected.

Testbench Design Methodology

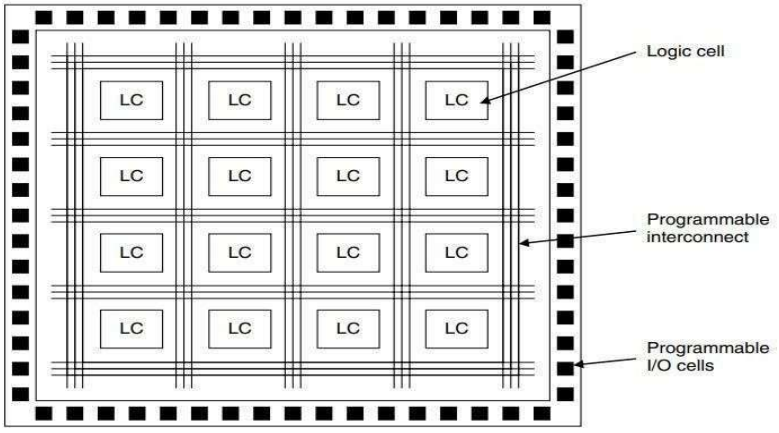
A testbench provides a controlled simulation environment for evaluating the design under test (DUT). It generates input signals, applies stimulus, and observes output responses. For sequential circuits such as an FSM-based traffic controller, a structured testbench is critical for comprehensive verification.

Purpose of the Testbench

The testbench developed in this project generates the clock signal, applies reset conditions, and monitors traffic light outputs. Its objective is to confirm that the controller cycles through all FSM states with correct timing and signal combinations.

FPGA

Field-Programmable Gate Arrays (FPGAs) are reconfigurable semiconductor devices that allow designers to implement digital logic after manufacturing. Unlike fixed-function integrated circuits, FPGAs provide programmable logic resources and routing networks that can be configured to perform a wide variety of digital functions. This flexibility makes FPGAs suitable for prototyping, rapid hardware development, and real-time embedded applications.
An FPGA consists of an array of configurable hardware blocks interconnected through programmable routing resources. By using Hardware Description Languages (HDLs) such as Verilog or VHDL, designers can define custom logic that is synthesized and mapped onto these resources. The ability to reprogram FPGAs multiple times enables iterative design improvements and system upgrades without modifying physical hardware.



.

Figure 1 Internal Structure of an FPGA

Components of an FPGA

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

An FPGA is composed of several fundamental elements that collectively enable configurable digital circuit implementation. The primary components are described below.

Configurable Logic Blocks (CLBs)

Configurable Logic Blocks form the core computational units of an FPGA. Each CLB typically contains Look-Up Tables (LUTs), flip-flops, multiplexers, and sometimes arithmetic logic resources. LUTs implement combinational logic, while flip-flops provide sequential storage. By configuring these elements, CLBs can realize various digital functions.

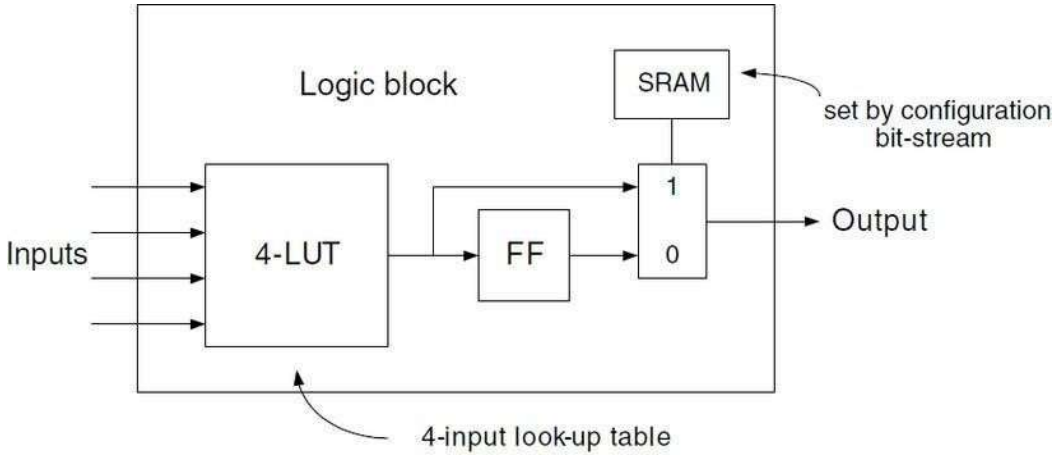


Figure 2 Block diagram of a configurable logic block

Programmable Interconnect Points (PIPs)

Programmable Interconnect Points provide routing paths between CLBs and other FPGA resources. These programmable switches allow signals to be connected flexibly across the device. The routing fabric is implemented using transistor-based switching elements that enable customizable signal paths.

Input/Output Blocks (IOBs)

Input/Output Blocks act as interfaces between the FPGA and external hardware components. These blocks support multiple voltage levels and signaling standards, allowing communication with sensors, memory devices, and other peripherals.

Block RAM (BRAM)

Block RAM refers to embedded memory resources available within the FPGA. These on-chip memories offer fast access and low latency compared to external memory devices. BRAMs are commonly used for buffering data, implementing FIFO structures, and storing lookup tables.

Digital Signal Processing (DSP) Blocks

DSP slices are specialized hardware units optimized for arithmetic operations such as multiplication, addition, and accumulation. These blocks improve performance in applications requiring high-speed numerical processing, including filtering and signal processing tasks.

FPGA Design Flow The FPGA design process follows a structured workflow that transforms a conceptual design into a hardware implementation. The major stages of this flow are outlined below.

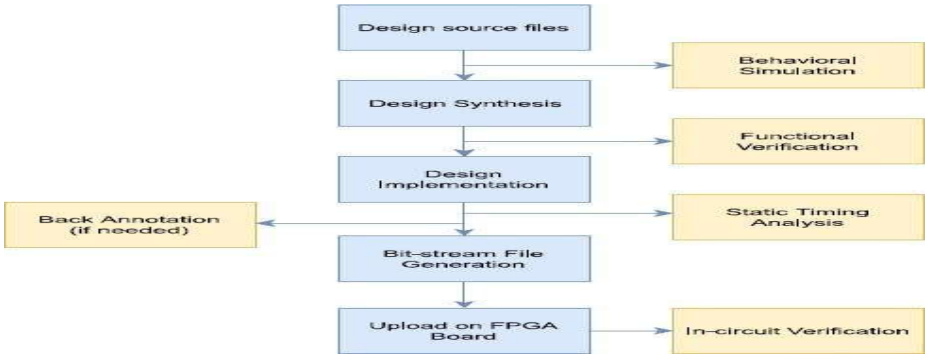


Figure 5.3 Typical FPGA Design Flow

International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

1. Design Entry
The design process begins with creating a hardware description using an HDL such as Verilog or VHDL. This stage defines system behavior and logical structure.

2. RTL Design and Simulation
The HDL description is represented at the Register Transfer Level (RTL), where data flow between registers and logical operations is modeled. Simulation is performed to verify functional correctness before proceeding further.

3. Synthesis
During synthesis, the RTL description is converted into a gate-level netlist consisting of logic gates and flip-flops. The synthesis tool also performs optimizations for speed, area, and power consumption. Common synthesis tools include AMD Vivado, Intel Quartus, and Synopsys Design Compiler.

4. Implementation
The implementation stage maps the synthesized netlist onto the physical FPGA resources. This process includes placement, routing, and bitstream generation. Placement assigns logic elements to specific locations, while routing establishes signal connections. The resulting bitstream file is used to configure the FPGA device.

5. Testing and Debugging
After implementation, the design is tested either through simulation or on actual hardware. Debugging tools such as logic analyzers and waveform viewers are used to verify functionality and identify potential issues. This stage ensures that the design meets performance and timing requirements.

Advantages of FPGA

FPGAs provide several benefits compared to fixed hardware solutions:

- **Flexibility:** Logic functionality can be modified through reprogramming.
- **Rapid Prototyping:** Designers can quickly test and refine hardware designs.
- **Parallel Processing:** Multiple operations can execute simultaneously, improving performance.
- **Customization:** Hardware can be tailored for specific applications.
- **Reconfigurability:** Designs can be updated without replacing hardware.
- **Lower Development Cost:** No expensive fabrication process is required compared to ASICs.
- **Scalability:** Designs can be adjusted according to system requirements.

Applications of FPGA

Due to their versatility, FPGAs are used in numerous domains, including:

- Communication systems and network infrastructure
- Aerospace, defense, and radar applications
- Software-defined radio and digital signal processing
- Automotive electronics and infotainment systems
- High-resolution video and audio processing
- Data centers, servers, and security systems
- Consumer electronics such as cameras and set-top boxes
- Industrial automation and embedded control systems

RESULTS AND COMPARATIVE STUDY

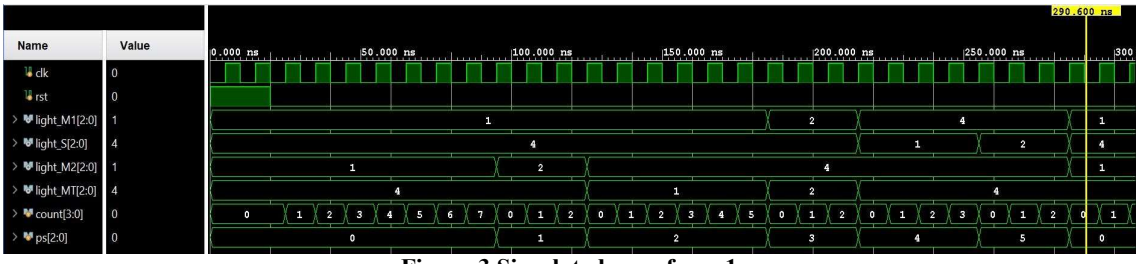
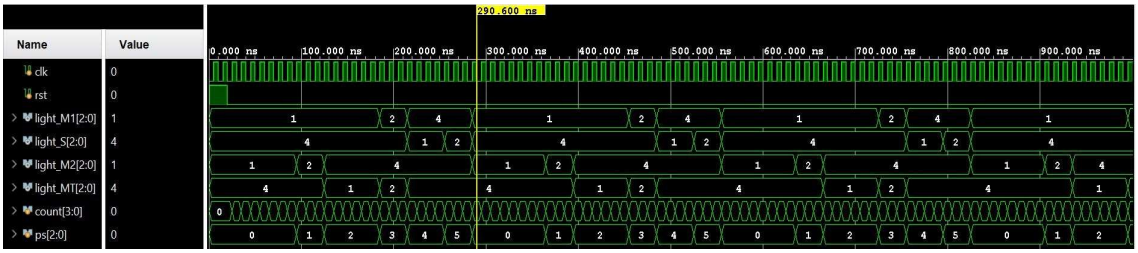


Figure 3 Simulated waveform 1



International Journal of Information Technology & Computer Engineering
ISSN 2347–3657
Volume 14, Issue 1, 2026

Figure 4 Simulated waveform 2

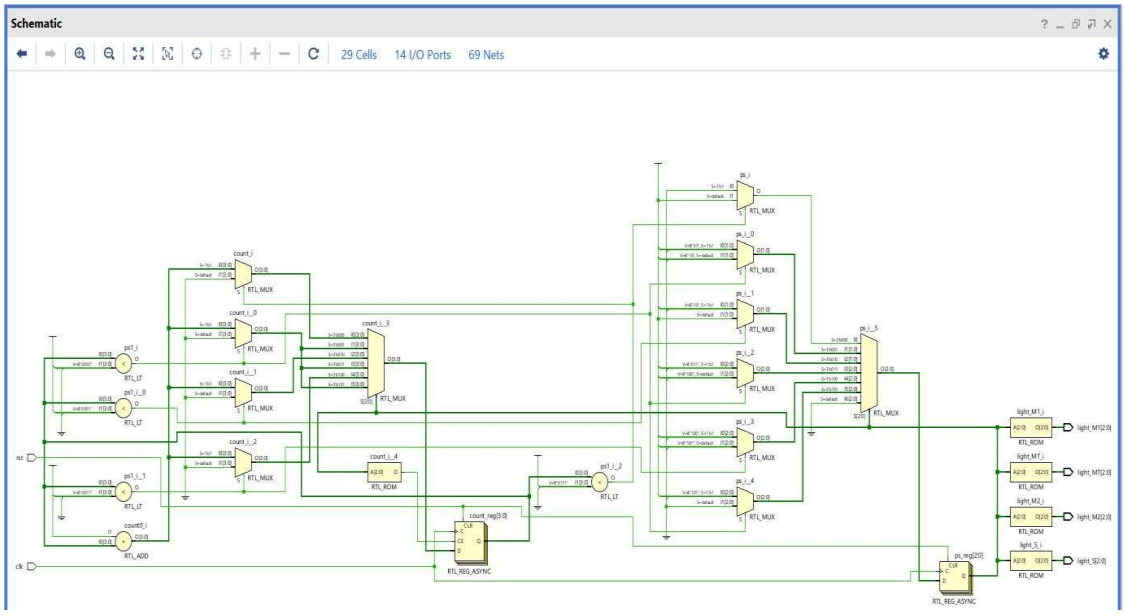


Figure 5 RTL-Schematic

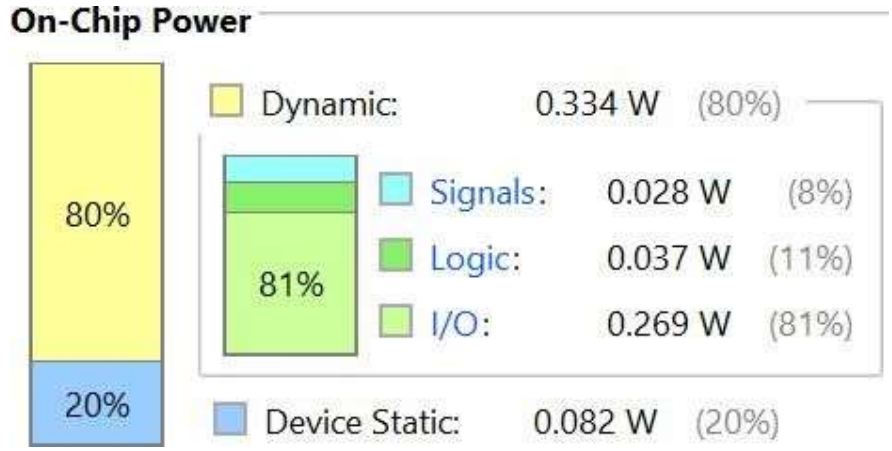


Figure 6 On-Chip Power Breakdown

```
Slack:                  inf
  Source:               rst
                          (input port)
  Destination:          FSM_onehot_ps_reg[2]/CLR
  Path Group:           (none)
  Path Type:            Min at Fast Process Corner
  Data Path Delay:      0.490ns  (logic 0.079ns (16.121%)  route 0.411ns (83.879%))
  Logic Levels:         1  (IBUF=1)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
    T18                                               0.000     0.000 f  rst (IN)
                         net (fo=0)                   0.000     0.000    rst
    T18                  IBUF (Prop_ibuf_I_O)         0.079     0.079 f  rst_IBUF_inst/O
                         net (fo=9, routed)           0.411     0.490    rst_IBUF
    SLICE_X1Y7           FDCE                                        f  FSM_onehot_ps_reg[2]/CLR
  -------------------------------------------------------------------    -------------------
```

Figure 6.5 Timing report of FPGA design tool

The simulation and synthesis results demonstrate the correct functional behavior of the proposed FSM-based traffic light controller. Waveform outputs confirm that the system transitions sequentially through all defined states with appropriate timing intervals. The RTL schematic verifies that the synthesized design corresponds to the intended hardware architecture. Power analysis and timing reports further confirm that the controller meets design constraints and operates efficiently on the FPGA platform.

Traditional Timer-Based Traffic Light Systems

Conventional traffic control systems have historically relied on fixed timer-based mechanisms. These systems operate by assigning predefined time intervals to each traffic direction without considering real-time traffic conditions. Although such systems are simple to implement, they often lead to inefficient utilization of road capacity.

Operational Overview

Timer-based controllers switch traffic signals using constant timing cycles. Typical configurations include fixed green, yellow, and red durations that repeat continuously. The hardware implementation usually involves:

- Clock oscillator or timing circuit
- Relay-based switching or simple logic controller
- Absence of real-time traffic sensing

Because these systems operate independently of actual traffic demand, congestion may occur during peak hours while empty roads receive unnecessary green time during low-traffic conditions.

FSM vs Microcontroller vs HDL Models

Different architectural approaches can be used to design traffic controllers. The three common approaches include:

1. FSM-based design using Verilog HDL
2. Microcontroller-based implementation
3. HDL-based modeling using VHDL

Comparison Metrics

Power Consumption

Power efficiency is an important consideration for embedded hardware systems.

- **FSM(VerilogHDL):** Low power consumption due to synchronous operation and minimal switching activity.
- **Microcontroller:** Higher power usage due to continuous instruction execution, even in idle conditions.
- **VHDLDesign:** Comparable to Verilog since both rely on the same FPGA hardware resources.

CONCLUSION

The FSM-based traffic light controller designed in this project demonstrates an effective solution for managing traffic at a T-junction intersection. By applying digital design principles and FPGA-based implementation, the controller achieves predictable timing behavior and reliable signal sequencing.

The design process involved developing a structured FSM model, defining state transitions, and implementing timing control using Verilog HDL. Simulation results verified correct functional behavior, while synthesis confirmed efficient hardware utilization. The FPGA implementation provides advantages such as deterministic execution, fast response time, and reconfigurability.

Despite its effectiveness, the current design operates using fixed timing intervals and does not adapt to real-time traffic conditions. Future enhancements can improve system intelligence by integrating sensors, adaptive algorithms, and communication interfaces. These improvements would enable dynamic traffic management and better utilization of road infrastructure.

Overall, the proposed design provides a strong foundation for advanced traffic control systems and demonstrates the practical applicability of FSM-based hardware solutions.

FUTURE SCOPE

Adaptive Traffic Control Using Sensors

Future systems can incorporate sensors such as infrared detectors, inductive loops, or cameras to monitor traffic density. Real-time data can be used to dynamically adjust signal durations, improving traffic flow efficiency.

Emergency Vehicle Priority

Emergency vehicle detection using GPS, RF communication, or acoustic sensors can enable automatic priority signaling, reducing response time for emergency services.

Vehicle Density Monitoring

Image processing or radar-based detection can estimate queue length at intersections. This information allows adaptive allocation of green time based on traffic demand.

Multi-Intersection Expansion

The controller can be extended to support four-way intersections and complex junctions by increasing FSM states and introducing hierarchical control strategies.

REFERENCES

1) A. Bidwai, A. Hodge, and H. Humnabakar, “Traffic Light Control System Using Verilog,” Department of Electronics and Telecommunication Engineering, Vishwakarma Institute of Technology, Pune, India.
2) K. Bhulakshmi, “Smart Traffic Light Controller Using Verilog,” Teegala Krishna Reddy Engineering College, Hyderabad, India.

ISSN 2347–3657
Volume 14, Issue 1, 2026

3) A. Shaheen, A. Supriya, B. Keerthana, B. Srivani, and C. Ajay, "Design of Smart Traffic Signal Using Verilog," Christu Jyothi Institute of Technology and Science, Telangana, India.
4) S. R. Pawar and P. K. Kawitkar, "FPGA Based Traffic Light Controller Using Finite State Machine," International Journal of Computer Applications, vol. 116, no. 17, pp. 25–29, 2015.
5) S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design*, McGraw-Hill Education, 2014.
6) M. Mano and M. Ciletti, *Digital Design with an Introduction to Verilog HDL*, Pearson Education, 2017.
7) S. Trimberger, *Field-Programmable Gate Array Technology*, Springer, 2012.
8) Xilinx Inc., "Vivado Design Suite User Guide," AMD Xilinx Documentation, 2023.
9) P. Chu, *FPGA Prototyping by Verilog Examples*, Wiley-Interscience, 2018.
10) R. Tessier and W. Burleson, "Reconfigurable Computing for Digital Signal Processing," IEEE Signal Processing Magazine, vol. 18, no. 3, pp. 45–54, 2011.