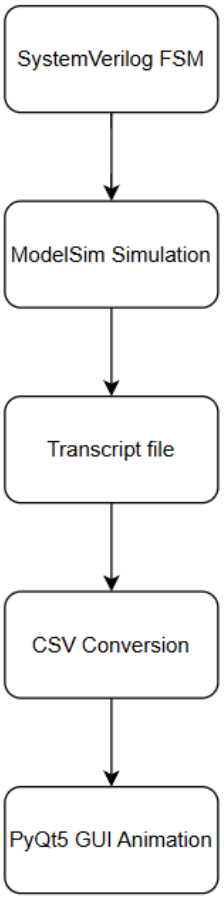
4-Way Traffic Light Controller 🚦

1. Project Overview

This project implements a **4-way traffic light controller** using a **Finite State Machine (FSM)** designed in **SystemVerilog** and synthesized using **Quartus Prime**.

A **Python (PyQt5) animated GUI** visualizes the controller behavior using simulation data exported from the VLSI flow.

2. System Architecture



3. Traffic Controller Working

3.1 Traffic Directions

- **North & South (NS)** → same signal
- **East & West (EW)** → same signal

Opposite directions always share the same color to prevent collisions.

3.2 Signal Colors

Color	Meaning
Red (R)	Stop
Yellow (Y)	Transition
Green (G)	Go

4. Finite State Machine (FSM) Design

4.1 Introduction to Finite State Machine (FSM)

A **Finite State Machine (FSM)** is a mathematical model of computation used to design **sequential digital systems**. An FSM operates by transitioning between a **finite number of states**, based on:

- The **current state**
- The **inputs**
- The **clock**
- The **reset condition**

FSMs are widely used in **control-dominated VLSI systems** such as:

- Traffic controllers
- Communication protocols
- CPU control units
- Industrial automation systems

An FSM provides a **structured and predictable way** to model time-dependent behavior.

4.2 Core Components of an FSM

An FSM consists of the following components:

1. **States**

   Represent distinct modes of operation of the system.

2. **State Register**

   Stores the current state and updates on the clock edge.

3. **Next-State Logic**

   Determines the next state based on the current state and conditions.

4. **Output Logic**

   Generates outputs depending on the state (and possibly inputs).

5. **Clock and Reset**

   Synchronize state transitions and initialize the system.

4.3 Types of FSMs

FSMs are broadly classified into two types:

1. Moore Machine

- Outputs depend **only on the current state**
- Outputs change **only on clock edges**
- Glitch-free and timing-safe

2. Mealy Machine

- Outputs depend on **current state and inputs**
- Faster response but more prone to glitches

4.4 FSM Selection for Traffic Controller

For the traffic light controller, a **Moore FSM** is selected because:

- Traffic signals must be **stable and glitch-free**

- Outputs should change **only at defined time boundaries**
- Safety-critical systems prefer deterministic behavior

Thus, signal colors depend **only on the active state**, not on asynchronous inputs.

4.5 FSM Application in Traffic Light Control

In a traffic controller:

- Each **state represents a traffic phase**
- State transitions occur after a **fixed time duration**
- Output signals (Red, Yellow, Green) are mapped directly to states

This makes FSMs an **ideal abstraction** for traffic signal control.

4.6 FSM State Definition for 4-Way Traffic Controller

The 4-way intersection is divided into two orthogonal directions:

- **North-South (NS)**
- **East-West (EW)**

Opposite directions share the same signal to prevent conflicts.

4.7 FSM States Description

State	NS Signal	EW Signal	Description
S0	Green	Red	NS traffic allowed
S1	Yellow	Red	NS transition
S2	Red	Green	EW traffic allowed
S3	Red	Yellow	EW transition

4.8 State Transition Logic

The FSM transitions sequentially as follows:

```
S0 → S1 → S2 → S3 → S0
```

- Transitions occur when the **state timer expires**
- Timer is driven by a **1 Hz clock**
- Ensures real-time traffic behavior

4.9 Timing Control Using FSM

Each state is associated with a **time duration**:

- Green states: longer duration
- Yellow states: shorter duration

Timing is implemented using a **state timer counter** that resets on every state transition.

4.10 Reset Behavior

An **active-low reset (`rst_n`)** initializes the FSM:

- Forces the system into a known safe state (S0)
- Ensures NS starts with Green and EW with Red
- Prevents undefined startup behavior

4.11 FSM Design Advantages in This Project

- Clear state separation
- Safe signal transitions
- Easy parameterization
- Synthesizable and scalable
- Industry-aligned design approach

🛠️ Tools & Technologies

Category	Tool
HDL	SystemVerilog
Simulator	ModelSim
Synthesis Tool	**Intel Quartus Prime (20.1 Lite)**
Target FPGA	Cyclone V
GUI	PyQt5
Language	Python 3
Data Format	CSV

🧩 Quartus Prime – Synthesis Details

The design was synthesized using **Intel Quartus Prime Lite Edition** with the following configuration:

- **Top-Level Entity:** `traffic_controller_4way`
- **FPGA Family:** Cyclone V
- **Registers Used:** 8
- **Memory Blocks:** 0
- **DSP Blocks:** 0
- **PLLs Used:** 0
- **Synthesis Status:** ✅ Successful

📂 Project Structure

```
Traffic-Light-Controller/
│
├── traffic_controller_4way.sv     # RTL FSM Design
├── tb_traffic_controller_4way.sv  # Verification Testbench
│
├── sim_data.csv                   # Auto-generated simulat
ion data
│
├── traffic_gui.py                 # PyQt5 animated GUI
```

```
│
├── README.md                          # Documentation
```

▶ How to Run the Project

1️⃣ RTL Simulation (ModelSim)

```
vlog *.sv
vsim tb_traffic_controller_4way
run -all
```

Simulation prints structured logs which are converted to CSV.

2️⃣ FPGA Synthesis (Quartus Prime)

- Open Quartus Prime
- Set `traffic_controller_4way` as top module
- Select **Cyclone V**
- Run:
  - Analysis & Synthesis

✔ Confirms synthesizability and hardware correctness.

3️⃣ GUI Execution

```
Traffic_controller_GUI.py
```

- **START** → Begin real-time animation
- **STOP** → Pause simulation
- GUI reads **CSV automatically**

🖥️ GUI Features

- Animated **4-direction traffic lights**

- Real-time timer display
- Start / Stop simulation control
- Table view synced with FSM states
- Designed for **non-technical clarity**
- GUI-

	TIME	STATE	TIMER	NS	EW
1	3000000000000	0	0	G	R
2	4000000000000	0	1	G	R
3	5000000000000	0	2	G	R
4	6000000000000	0	3	G	R
5	7000000000000	0	4	G	R
6	8000000000000	0	5	G	R
7	9000000000000	0	6	G	R
8	10000000000000	0	7	G	R
9	11000000000000	0	8	G	R
10	12000000000000	0	9	G	R

📘 Academic & VLSI Relevance

This project demonstrates:

- FSM modeling best practices
- Clean RTL coding style
- Proper testbench methodology
- Successful FPGA synthesis
- End-to-end hardware verification

Ideal for:

- VLSI Lab evaluations
- FPGA coursework
- Final-year engineering projects

- Viva demonstrations

🔮 Future Enhancements

- Pedestrian signals
- Emergency vehicle override
- Adaptive AI-based timing
- FPGA board deployment
- Live waveform-to-GUI linking

👤 Author

Vishal Prakash Shinde

Electronics & Telecommunication Engineering

FSM • VLSI • RTL Design • FPGA Synthesis