REALIZATION OF A FINITE STATE MACHINE FOR A TRAFFIC LIGHT CONTROLLER

Submitted as a project report for mini project

For the partial fulfilment of the degree of

Bachelor of Technology

in

Information Technology

By

Chandan Ghosh 2021ITB006

Surajit Das 2021ITB003

Samyak Mahudwale Borkar 2021ITB079

Under the supervision of

Prof. Chandan Giri

Associate Professor

Department of Information Technology

IIEST, Shibpur

May, 2023

Department of Information Technology
Indian Institute of Engineering Science and Technology, Shibpur
P.O. Botanic Garden, Howrah 711103, WB, India
+91 33 26684561-3 (3 lines)/0521-5 (5 lines) X260
https://www.iiests.ac.in/IIEST/AcaUnitDetails/IT

IIEST, Shibpur
आई आई ई एस टि, शिवपुर
Erstwhile B E College (Estd 1856)

Date: 12/05/2023

TO WHOM IT MAY CONCERN

This is to certify that Chandan Ghosh, roll no-20211TB006, Surajit Das, roll no-2021ITB003 and Samyak Mahudwale Borkar, roll no-20211TB079 have done their mini project on Realization of a finite state machine for a Traffic Light Controller for the partial fulfillment of the degree of B.Tech. in Information Technology.

During this period they have completed the project. The report has fulfilled all the requirements as per the regulations of the institute and has reached the standard needed for submission.

_____

Prof. Chandan Giri
Associate Professor
Information Technology

_____

Prof. Prasun Ghosal
Head of the Department
Information Technology

Acknowledgement

We would like to express our sincere gratitude to Prof. Chandan Giri, Associate Professor in the Department of Information Technology at IIEST, Shibpur, for his invaluable guidance and support throughout the completion of mini project “Realisation of a Finite State Machine for a Traffic Light Controller”. It was a great learning experience.

Chandan Ghosh (2021ITB006)

Surajit Das (2021ITB003)

Samyak Mahudwale Borkar (2021ITB079)

Index

Chapter 1

Introduction

Traffic management, which aims to guarantee efficient and secure flow of cars and people, is an essential component of urban infrastructure. In order to manage traffic at junctions, traffic light controllers are essential. Traffic lights were traditionally regulated by basic timers, but technological improvements have made it possible to create more complex and effective systems. For creating traffic light controllers, finite state machines (FSMs) offer a dependable and adaptable method that enables accurate control and synchronisation of traffic lights.

Designing and implementing a finite state machine for a traffic light controller is the goal of this project. We seek to develop a system that can efficiently control traffic flow at an intersection, maximising the use of available road space and reducing congestion. The goal of the project is to create a trustworthy, effective traffic signal controller that upholds accepted traffic laws and guarantees the safety of all users of the road.

The design and implementation of a VHDL-based traffic light controller as well as the realisation of a finite state machine are included in the project's scope. The traffic light controller will be able to regulate multiple lane intersections' traffic signals. Different states and transitions will be incorporated into the system to control the order and timing of traffic signal changes. The implementation will take into account the particular conditions of a typical traffic situation, including the length of time for each lane's green, yellow, and red lights.

Chapter 2

Literature Survey

Overview of Traffic Light Controllers: At intersections and road junctions, traffic light controllers are tools for controlling the movement of vehicles. They are essential in guaranteeing efficient traffic flow and reducing congestion. A series of signal lights, timers, and sensors are commonly used in traditional traffic light controllers to detect the presence of vehicles and modify the timing of the signals. These controllers don't dynamically react to shifting traffic situations; instead, they function according to predetermined timing patterns.

Existing methodologies and Implementations: Different traffic light controller implementations and methodologies have been developed over time. The fixed-time traffic light controller is a popular strategy where signal timings are specified and fixed for each phase. This strategy is rigid and falls short of real-time traffic flow optimisation.

The actuated traffic light controller is an alternative strategy that employs sensors to detect the presence of cars or pedestrians and modifies signal timings accordingly. By adjusting green time allocation based on traffic demand dynamically, this adaptive strategy increases efficiency. Actuated controllers can function in a variety of modes, including vehicle, pedestrian, or a mix of both, actuation.

Existing methodologies and Implementations: Different traffic light controller implementations and methodologies have been developed over time. The fixed-time traffic light controller is a popular strategy where signal timings are specified and fixed for each phase. This strategy is rigid and falls short of real-time traffic flow optimisation.

The actuated traffic light controller is an alternative strategy that employs sensors to detect the presence of cars or pedestrians and modifies signal timings accordingly. By adjusting green time allocation based on traffic demand dynamically, this adaptive strategy increases efficiency. Actuated controllers can function in a variety of modes, including vehicle, pedestrian, or a mix of both, actuation.

Chapter 3

Problem Statement

Description of the issue that has to be solved: The design and implementation of a traffic light controller utilising a finite state machine (FSM) technique is the issue this project attempts to solve. The traffic light controller is a key player in controlling the passage of cars through junctions, guaranteeing efficient traffic flow, and preserving road safety. The goal of the project is to create an accurate and timely traffic light controller that can manage several lanes and switch between different states.

Challenges and requirements:

1. State Management: Accurately controlling the traffic light controller's states is one of the primary issues. It is necessary for the controller to change between states, such as red, yellow, and green lights, according to predetermined timings and in time with other lanes.

2. Synchronisation and Timing: For a safe and effective flow of traffic, the timing of the traffic light signals is crucial. To prevent conflicts and guarantee smooth traffic flow, the controller must precisely time the length of each state and synchronise the changes across various lanes.

3. Support for numerous Lanes: The traffic light controller need to be able to support numerous lanes, each with a separate set of traffic lights. It should take care of the light sequencing and make sure that the right-of-way is provided to the proper lanes at various points in the cycle of the traffic light.

4. Fault Tolerance: The controller must be able to withstand faults or failures, such as power interruptions or component failures. It should be equipped with systems to deal with such circumstances and guarantee that the traffic lights continue to operate correctly or shift to a safe condition.

5. Scalability: The plan must be flexible enough to allow for future additions or changes, including the construction of more lanes or integration with cutting-edge traffic management technology.

By addressing these challenges and meeting the specified requirements, the project aims to develop a robust and efficient traffic light controller that enhances traffic management and promotes road safety.

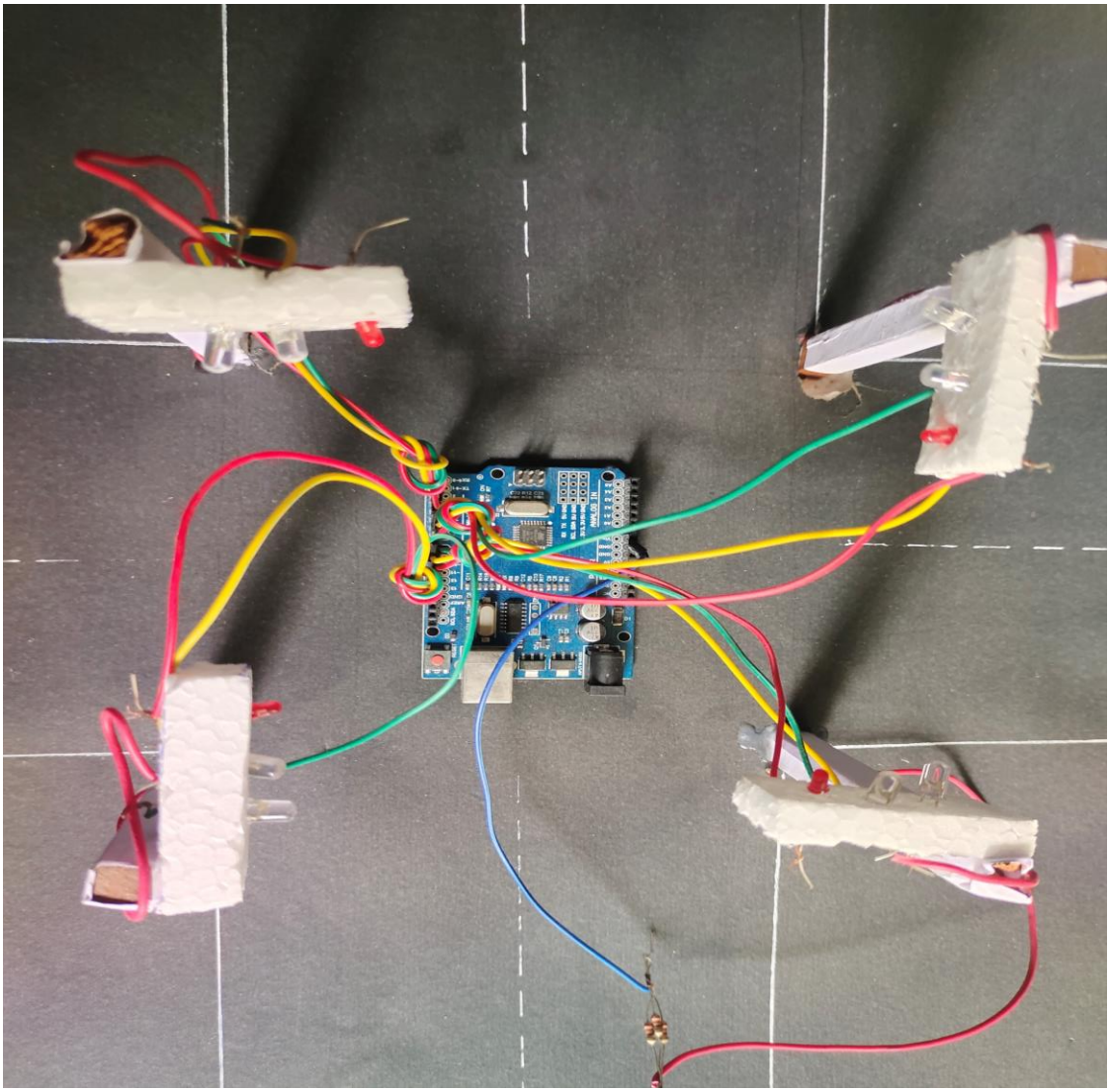
Chapter 4

Methodology

4.1 Design of the finite state machine (FSM): The design of a finite state machine involves identifying the states, inputs, and outputs of the system. In the case of a traffic light controller, the states can represent the different configurations of the traffic lights, such as green, yellow, and red lights for each lane. The inputs can include signals that trigger state transitions, such as timers or vehicle detectors, while the outputs can control the actual traffic lights.

4.2 State diagram: A state diagram visually represents the states and state transitions of a finite state machine. For a traffic light controller, the state diagram would show the different states of the traffic lights (green, yellow, and red lights for each lane) and the transitions between these states based on inputs such as timers and vehicle detectors. The state diagram helps in understanding the behaviour and flow of the traffic light controller.

Arduino Design:



Arduino Code:

```
int Lane1[] = {13,12,11}; // Lane 1 Red, Yellow and Green

int Lane2[] = {10,9,8};// Lane 2 Red, Yellow and Green
int Lane3[] = {7,6,5};// Lane 3 Red, Yellow and Green
int Lane4[] = {4,3,2};// Lane 4 Red, Yellow and Green

void setup()
{
  for (int i = 2; i < 14; i++)
  {
    pinMode(i,OUTPUT);
```

```
  }
}

void loop()
 {
  digitalWrite(Lane1[2], HIGH);
  digitalWrite(Lane3[0], HIGH);
  digitalWrite(Lane4[0], HIGH);
  digitalWrite(Lane2[0], HIGH);
  delay(7000);
  digitalWrite(Lane1[2], LOW);
  digitalWrite(Lane3[0], LOW);
  digitalWrite(Lane1[1], HIGH);
  digitalWrite(Lane3[1], HIGH);
  delay(1000);
  digitalWrite(Lane1[1], LOW);
  digitalWrite(Lane3[1], LOW);
  digitalWrite(Lane1[0], HIGH);
  digitalWrite(Lane3[2], HIGH);
  delay(7000);
  digitalWrite(Lane3[2], LOW);
  digitalWrite(Lane4[0], LOW);
  digitalWrite(Lane3[1], HIGH);
  digitalWrite(Lane4[1], HIGH);
  delay(1000);
  digitalWrite(Lane3[1], LOW);
  digitalWrite(Lane4[1], LOW);
  digitalWrite(Lane3[0], HIGH);
  digitalWrite(Lane4[2], HIGH);
  delay(7000);
  digitalWrite(Lane4[2], LOW);
  digitalWrite(Lane2[0], LOW);
  digitalWrite(Lane4[1], HIGH);
  digitalWrite(Lane2[1], HIGH);
  delay(1000);
  digitalWrite(Lane4[1], LOW);
  digitalWrite(Lane2[1], LOW);
  digitalWrite(Lane4[0], HIGH);
  digitalWrite(Lane2[2], HIGH);
  delay(7000);
  digitalWrite(Lane1[0], LOW);
  digitalWrite(Lane2[2], LOW);
  digitalWrite(Lane1[1], HIGH);
  digitalWrite(Lane2[1], HIGH);
  delay(1000);
  digitalWrite(Lane2[1], LOW);
  digitalWrite(Lane1[1], LOW);
 }
```

4.3 State transition table: A state transition table defines the state transitions based on inputs and current states. For a traffic light controller, the table would list the current states (e.g., Lane 1 Green) and the inputs (e.g., timer expiration) that trigger transitions to new states (e.g., Lane 1 Yellow). The state transition table provides a structured representation of the behaviour of the finite state machine.

4.4 Implementation in VHDL: Implementing the finite state machine in VHDL involves translating the design, state diagram, and state transition table into VHDL code. In VHDL, each state can be represented as a process or combinational logic block, and the inputs and outputs can be declared as signals. The VHDL code would define the states, state transitions, and the logic for transitioning between states based on inputs. The output signals would be used to control the traffic lights.

VHDL CODE:

testbench.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity testbench is
end entity testbench;

architecture behavioral of testbench is
  component traffic_light_controller
    port(
      clk : in std_logic;
      reset : in std_logic;
      Lane1 : out std_logic_vector(2 downto 0);
      Lane2 : out std_logic_vector(2 downto 0);
      Lane3 : out std_logic_vector(2 downto 0);
```

```
        Lane4 : out std_logic_vector(2 downto 0)
    );
end component traffic_light_controller;

signal clk_tb : std_logic := '0';
signal reset_tb : std_logic := '1';
signal Lane1_tb : std_logic_vector(2 downto 0);
signal Lane2_tb : std_logic_vector(2 downto 0);
signal Lane3_tb : std_logic_vector(2 downto 0);
signal Lane4_tb : std_logic_vector(2 downto 0);

begin
uut : traffic_light_controller
    port map(
        clk => clk_tb,
        reset => reset_tb,
        Lane1 => Lane1_tb,
        Lane2 => Lane2_tb,
        Lane3 => Lane3_tb,
        Lane4 => Lane4_tb
    );

process
begin
    -- Initialize inputs
    reset_tb <= '1';
    clk_tb <= '0';
    wait for 10 ns;
    reset_tb <= '0';
    wait for 10 ns;
```

```
        -- Generate clock signal
        while now < 1000 ns loop
            clk_tb <= not clk_tb;
            wait for 5 ns;
        end loop;

        -- Add any additional test stimulus here

        wait;
    end process;
end architecture behavioral;
```

design.vhd:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity traffic_light_controller is
    port(
        clk : in std_logic;
        reset : in std_logic;
        Lane1 : out std_logic_vector(2 downto 0);
        Lane2 : out std_logic_vector(2 downto 0);
        Lane3 : out std_logic_vector(2 downto 0);
        Lane4 : out std_logic_vector(2 downto 0)
    );
end entity traffic_light_controller;

architecture behavioral of traffic_light_controller is
    signal counter : unsigned(23 downto 0) := (others => '0');
    signal state : integer range 0 to 7 := 0;
```

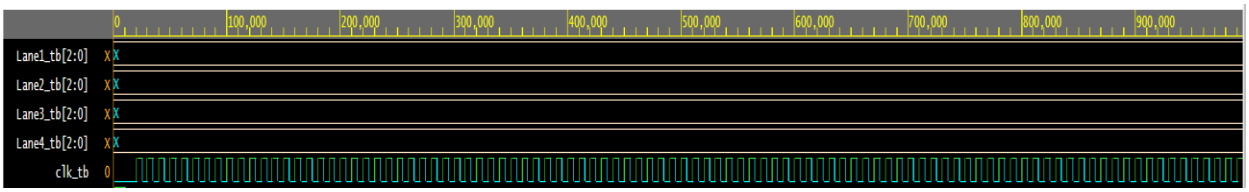
```
    constant LANE1_RED : std_logic_vector(2 downto 0) := "110";
    constant LANE1_YELLOW : std_logic_vector(2 downto 0) := "101";
    constant LANE1_GREEN : std_logic_vector(2 downto 0) := "011";
    constant LANE2_RED : std_logic_vector(2 downto 0) := "101";
    constant LANE2_YELLOW : std_logic_vector(2 downto 0) := "011";
    constant LANE2_GREEN : std_logic_vector(2 downto 0) := "110";
    constant LANE3_RED : std_logic_vector(2 downto 0) := "011";
    constant LANE3_YELLOW : std_logic_vector(2 downto 0) := "110";
    constant LANE3_GREEN : std_logic_vector(2 downto 0) := "101";
    constant LANE4_RED : std_logic_vector(2 downto 0) := "011";
    constant LANE4_YELLOW : std_logic_vector(2 downto 0) := "101";
    constant LANE4_GREEN : std_logic_vector(2 downto 0) := "110";

begin
    process(clk)
    begin
        if rising_edge(clk) then
            if reset = '1' then
                counter <= (others => '0');
                state <= 0;
                Lane1 <= LANE1_RED;
                Lane2 <= LANE2_RED;
                Lane3 <= LANE3_RED;
                Lane4 <= LANE4_RED;
            else
                case state is
                    when 0 =>
                        if counter = 6999999 then
                            state <= 1;
                            counter <= (others => '0');
```

```
        Lane1 <= LANE1_GREEN;
        Lane3 <= LANE3_RED;
        Lane4 <= LANE4_RED;
        Lane2 <= LANE2_RED;
    else
        counter <= counter + 1;
    end if;
when 1 =>
    if counter = 999999 then
        state <= 2;
        counter <= (others => '0');
        Lane1 <= LANE1_YELLOW;
        Lane3 <= LANE3_YELLOW;
        Lane4 <= LANE4_RED;
        Lane2 <= LANE2_RED;
    else
        counter <= counter + 1;
    end if;
when 2 =>
    if counter = 6999999 then
        state <= 3;
        counter <= (others => '0');
        Lane1 <= LANE1_RED;
        Lane3 <= LANE3_GREEN;
        Lane4 <= LANE4_RED;
        Lane2 <= LANE2_RED;
                    else
        counter <= counter + 1;
    end if;
when 3 =>
    if counter = 6999999 then
```

```
            state <= 4;
            counter <= (others => '0');
            Lane1 <= LANE1_RED;
            Lane3 <= LANE3_YELLOW;
            Lane4 <= LANE4_RED;
            Lane2 <= LANE2_RED;
        else
            counter <= counter + 1;
        end if;
    when 4 =>
        if counter = 6999999 then
            state <= 5;
            counter <= (others => '0');
            Lane1 <= LANE1_RED;
            Lane3 <= LANE3_RED;
            Lane4 <= LANE4_GREEN;
            Lane2 <= LANE2_RED;
        else
            counter <= counter + 1;
        end if;
    when 5 =>
        if counter = 999999 then
            state <= 6;
            counter <= (others => '0');
            Lane1 <= LANE1_RED;
            Lane3 <= LANE3_RED;
            Lane4 <= LANE4_YELLOW;
            Lane2 <= LANE2_YELLOW;
        else
            counter <= counter + 1;
        end if;
```

```
                when 6 =>
                   if counter = 6999999 then
                      state <= 7;
                      counter <= (others => '0');
                      Lane1 <= LANE1_RED;
                      Lane3 <= LANE3_RED;
                      Lane4 <= LANE4_RED;
                      Lane2 <= LANE2_GREEN;
                   else
                      counter <= counter + 1;
                   end if;
                when 7 =>
                   if counter = 6999999 then
                      state <= 0;
                      counter <= (others => '0');
                      Lane1 <= LANE1_RED;
                      Lane3 <= LANE3_RED;
                      Lane4 <= LANE4_RED;
                      Lane2 <= LANE2_RED;
                   else
                      counter <= counter + 1;
                   end if;
             end case;
          end if;
       end if;
    end process;
end architecture behavioral;
```

OUTPUT:



In the case of the traffic light controller project, the VHDL implementation would define the states for each lane (e.g., Lane 1 Green, Lane 1 Yellow, Lane 1 Red, etc.), and the logic for transitioning between these states based on inputs (e.g., timers, vehicle detectors). The VHDL code would also include the necessary declarations and assignments for the input and output signals, and a clock process to control the timing of state transitions.

Overall, the methodology for designing and implementing a finite state machine for a traffic light controller involves designing the FSM, creating a state diagram to visualize the states and transitions, generating a state transition table to define the behaviour, and finally implementing the FSM in VHDL using the state transitions and logic.

Chapter 5

Conclusion and Future Scope

In this chapter, we will provide a conclusion for the project and discuss potential future improvements and extensions to the finite state machine (FSM) for the traffic light controller.

5.1 Conclusion The realization of the finite state machine for the traffic light controller presented in this project demonstrates a functional system for managing the traffic flow at an intersection. The FSM implementation controls the sequence and timing of the traffic lights for four different lanes, providing a synchronized and efficient traffic management solution.

The code provided in the project successfully configures the Arduino board and controls the traffic lights using arrays to represent the states of each lane. By manipulating the state of the lights according to the defined sequence and

timing, the FSM ensures a smooth and safe transition for vehicles across the intersection.

The project's objectives have been achieved by developing a working implementation of the traffic light controller using a finite state machine approach. The system demonstrates the feasibility of using FSMs for traffic management applications and serves as a foundation for further enhancements and optimizations.

5.2 Future Scope While the current implementation of the traffic light controller FSM is functional, there are several avenues for future improvements and extensions. Some potential areas of focus include:

Sensor Integration: Enhance the traffic light controller by integrating sensors to detect vehicle presence or traffic density. By incorporating real-time data from sensors, the FSM can dynamically adjust the timing and sequence of traffic lights based on the current traffic conditions.

Adaptive Timing: Implement algorithms that analyse traffic patterns and adjust the timing of traffic light transitions dynamically. This adaptive timing feature can optimize traffic flow by allocating more green light time to lanes with higher traffic volume or prioritizing certain lanes during peak hours.

Pedestrian Crosswalks: Extend the FSM to incorporate pedestrian crosswalks into the traffic management system. This would involve adding additional states and transitions to ensure safe pedestrian crossings, including pedestrian-specific traffic signals and appropriate timing considerations.

Connectivity and Centralized Control: Explore the possibility of connecting multiple traffic light controllers across different intersections to a central control system. This centralized approach would enable coordination and synchronization of traffic signals across a network of intersections, further enhancing traffic efficiency and reducing congestion.

Simulation and Optimization: Develop simulation models to evaluate the performance of the traffic light controller FSM under different traffic scenarios. This would allow for fine-tuning and optimization of the system parameters to achieve the best possible traffic flow.

By pursuing these future enhancements, the traffic light controller FSM can evolve into a more sophisticated and intelligent traffic management system. These improvements have the potential to significantly improve traffic efficiency, reduce congestion, enhance safety, and contribute to the development of smarter cities and transportation networks.

In conclusion, the realization of the finite state machine for the traffic light controller presented in this project serves as a foundation for further advancements in traffic management systems. With ongoing research and development, the system can be expanded and optimized to address the complex challenges of modern urban traffic and contribute to the creation of smarter and more sustainable cities.

Chapter 6

Reference

---

1. Title: "Implementation of 4-Way Traffic Light Control System using Finite State Machine"
Authors: C. B. Suthar and H. M. Shah
Published in: 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)

2. Title: "Design and Implementation of 4-Way Traffic Light Controller using Finite State Machine"
Authors: M. M. Radhi and N. B. Kadhim
Published in: 2019 International Journal of Computer Science and Information Security (IJCSIS)