TRAFIC LIGHT USING ARDUINO UNO

MINI PROJECT REPORT

Submitted by

Adhithiyan S(23205003)

Azmal Babu B(23205008)

Mohamed Irfan J(23205030)

Prakalathan G(23205036)

in partial fulfilment for the award of the degree

of

Postgraduate

in

Master of Computer Applications

Rathinam Technical Campus

Eachanari – 641021

An Autonomous Institution

Affiliated to Anna University, Chennai – 600 025

DEC - 2024

ANNA UNIVERSITY, CHENNAI

BONAFIDE CERTIFICATE

Certified that this Report titled **TRAFFIC LIGHT USING ARDUINO UNO** is the bonafide work of **Adhithiyan S (23205003), Azmal Babu B (23205008), Mohamed Irfan J (23205030), Prakalathan G (23205036)** who carried out the work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other Mini project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Varsha JJ

Assistant Professor

Master of Computer Applications
Rathinam Technical Campus

Echanari-641 021

Coimbatore.

Anna university, Chennai

Dr.A.B.Arockia Christopher

Supervisor and Professor Head

Master of Computer Applications
Rathinam Technical Campus

Echanari-641 021

Coimbatore.

Anna university, Chennai

Submitted for the Mini project report Viva – Voice examination held on ……

Internal Examiner

External Examiner

ABSTRACT

The purpose of this paper is to design and implement a low cost system intended in terms of hardware and software to make a street traffic light for cars. This system will be built using the Arduino Uno development platform, and programming will be done using the LabVIEW graphical programming. Hardware resources that will be used in the paper are: Arduino Uno, a red, yellow and green LED, a breadboard, 3 x suitable resistors for the LEDs you have (probably 220 Ohms is fine) and connecting wires. The results will be displayed through the serial interface on the computer in the LabVIEW program, and also on the three LEDs on the breadboard.

The **Traffic Light System using Arduino Uno** is a project designed to simulate and implement the functionality of a real-world traffic signal. This system utilizes an **Arduino Uno microcontroller** to control a set of LEDs (Red, Yellow, and Green) that mimic the operation of traffic lights at an intersection. The project focuses on demonstrating how microcontrollers can be programmed to manage sequential processes efficiently.

This project integrates **basic electronic components** such as resistors, LEDs, and Arduino programming to create a simple, cost-effective traffic light model. The system is programmed to operate in a loop, cycling through predefined time intervals for each light (e.g., red for stop, yellow for caution, and green for go), ensuring smooth traffic flow simulation.

ACKNOWLEDGEMENT

Apart from the efforts of us, the success of this project depends largely on the encouragement and guidelines of many others. We take this opportunity to praise the **almighty** and express our gratitude to the **people** who have been instrumental in the successful completion of our project

We wish to acknowledge with thanks for the excellent encouragement given by the management of our college and we thank **Dr. B.Nagaraj, M.E., Ph.D., PDF (Italy) , CBO** for providing us with a plethora of facilities in the campus to complete our project successfully.

We wish to express our hearty thanks to **Dr. K.Geetha, M.E., Ph.D., Principal** of our college, for her constant motivation regarding our internship towards project.

We extend my heartfelt gratitude to **Dr.A.B.Arockia Christopher, M.E., Ph.D., Professor & HoD– MCA** for his tremendous support and assistance in the completion of our project.

It is our primary duty to thank our **Project guide, Ms. Varsha J J, MCA., Assistant Professor, MCA** who is the backbone of all our project activities, It's her enthusiasm and patience that guided us through the right path.

Finally, we extend our heartfelt thanks to the **parents, friends, and faculty members** for their constant support throughout this project. The guidance and support received from all the members who contributed to the success of the project.

ADHITHIYAN S

AZMAL BABU B

MOHAMED IRFAN J

PRAKALATHAN G

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

IOT	Internet of Things
IDE	Integrated Development Environment
LED	Light Emiting Diode
IR	Infra Red
GPS	Global Positioning System
I/O	Input /Output

CHAPTER - I

INTRODUCTION

1.INTRODUCTION

The traffic light system proves to be very vital in controlling vehicle flow at the interchanges in towns, prevents accidents, and reduces congestion as well, giving a clear indication whether to stop, get ready, or go. We will hereby simplify the making of a simple traffic light using an Arduino Uno, LEDs, and other basic components simulating the natural and real traffic lights. This provides for a teaching opportunity in the control of electronic circuits as well as programming basics with Arduino.

What is Arduino Uno?

The **Arduino Uno** is a microcontroller board based on the ATmega328P. It is one of the most popular boards in the Arduino ecosystem and is widely used for prototyping and learning about embedded systems and electronics.

Key Features of Arduino Uno:

- **Microcontroller:** ATmega328P.
- **Digital I/O Pins:** 14 (6 of which can be used as PWM outputs).
- **Analog Input Pins:** 6.
- **Clock Speed:** 16 MHz.
- **Operating Voltage:** 5V.
- **Programming Interface:** USB.

Why Use Arduino Uno?

- Beginner-friendly with a vast community.
- Extensive libraries and support for various components (sensors, actuators).
- Can be programmed using the **Arduino IDE**, a simple-to-use environment for writing, compiling, and uploading code.

What is IoT?

IoT (Internet of Things) refers to the network of physical devices that communicate and exchange data via the internet. Examples include smart thermostats, connected security systems, and wearable fitness trackers.

Key Components of IoT:

1. **Sensors and Actuators:** Collect data from the environment (e.g., temperature, humidity).
2. **Connectivity:** Transmit data using Wi-Fi, Bluetooth, or cellular networks.
3. **Microcontrollers or Microprocessors:** Process and manage data (e.g., Arduino, Raspberry Pi).
4. **Cloud Platforms:** Store and analyze data (e.g., AWS IoT, Firebase).
5. **User Interfaces:** Mobile apps or dashboards for monitoring and control.

Arduino Uno in IoT Projects

The Arduino Uno can serve as a central device to collect data from sensors and transmit it to the cloud or other devices using additional hardware like **Wi-Fi modules (ESP8266/ESP32)**, **Bluetooth modules (HC-05/HC-06)**, or **Ethernet shields**.

How to Get Started with Arduino and IoT

1. Set Up Arduino IDE

- **Download Arduino IDE** from Arduino's official website.
- Install and launch the IDE.
- Select the correct board and port under **Tools > Board** and **Tools > Port**.

2. Basic IoT Example: Collect and Send Data to the Cloud

Here’s an example using an **Arduino Uno**, a **DHT11 sensor (temperature and humidity)**, and an **ESP8266 Wi-Fi module**.

1.2 SCOPE OF THE PROJECT

- Develop an IoT-enabled system using Arduino Uno to collect data, process it, and transmit it to a cloud platform or a mobile application.
- Utilize sensors, actuators, and communication modules (e.g., Wi-Fi, Bluetooth) to enable real-time monitoring and control.
- **Scalability:** Expand the project to use more advanced boards (e.g., Arduino MKR1000 or ESP32) for added features like built-in Wi-Fi and Bluetooth.
- **Cloud Integration:** Use platforms like AWS IoT, Firebase, or Thingspeak for enhanced data storage and processing.
- **Advanced Applications:** Incorporate AI/ML for predictive analytics (e.g., predicting weather or user behavior).
- **Data Security Enhancements:** Implement encryption for secure communication.

1.3 OBJECTIVE OF THE PROJECT

1.Primary Goal

Implement a functional IoT system for a specific use case, such as smart home automation, weather monitoring, or health tracking.

2.Secondary Goals:

- Demonstrate how sensors interact with Arduino Uno to collect data.
- Utilize a communication module (e.g., ESP8266) to send data to the internet.
- Build a basic dashboard or mobile app for visualization and control (optional).

3.Deliverables

- Functional Prototype: A working model of the IoT system.
- Documentation:
  - Circuit diagrams.
  - Source code with comments.

    - User manual or setup guide.
- Optional: A basic interface or cloud integration for real-time monitoring and control.

4.Features

- Data Collection: Sensors like temperature, humidity, or motion sensors.
- Data Transmission: Send data to the cloud via Wi-Fi (e.g., using ESP8266).
- Data Visualization: Display the collected data on a web dashboard or mobile app.
- Real-Time Alerts: Send notifications or trigger actions based on specific conditions.
- Control: Enable remote operation of actuators like LEDs, relays, or motors.

5. Constraints

- Hardware Limitations: Arduino Uno has limited memory (32 KB) and processing power, so advanced tasks like machine learning may require more capable microcontrollers.
- Connectivity: Use of external Wi-Fi modules, as Arduino Uno lacks built-in Wi-Fi.
- Power Supply: Ensuring a stable power source for the project.
- Data Security: Minimal or no encryption, which could pose risks for sensitive IoT projects.

6. Use Cases

- Smart Home Automation:
    - Remotely control lights, fans, or appliances.
    - Monitor room temperature and humidity.
- Weather Monitoring System:
    - Track environmental conditions and send updates to a web platform.
- Agriculture IoT:
    - Measure soil moisture and temperature for smart irrigation systems.

- Health Monitoring:
    - Track heart rate, body temperature, or other vitals remotely.

7. Timeline

A typical timeline for an Arduino IoT project:

- Week 1: Define project requirements and gather components.
- Week 2: Build and test circuits with Arduino Uno and sensors.
- Week 3: Integrate a Wi-Fi module and test connectivity.
- Week 4: Develop code for data transmission and visualization.
- Week 5: Finalize and document the project.

1.4 Methodology

The methodology for an Arduino Uno and IoT project outlines the systematic approach to achieving the project's objectives. It ensures clarity, efficiency, and structure in development. Below is a general methodology for such a project.

1. Define the Problem Statement

- Clearly outline the purpose of the project.
  Example: "Develop a smart weather monitoring system to measure temperature and humidity and send real-time data to a cloud platform."

2. Requirements Gathering

- Hardware Requirements:
    - Arduino Uno.
    - Sensors (e.g., DHT11 for temperature and humidity).
    - Communication modules (e.g., ESP8266 for Wi-Fi).
    - Power source (USB, battery, or adapter).
    - Actuators (optional, e.g., LED, relay, or buzzer).
- Software Requirements:

- Arduino IDE.
- Required libraries (e.g., DHT for sensor data, ESP8266WiFi for connectivity).
- Cloud platform or data visualization tools (e.g., Thingspeak, Firebase, or a custom web dashboard).

3. System Design

- Block Diagram:
  - Divide the system into modules:
    - Sensor module: Collects data.
    - Microcontroller module: Processes data.
    - Communication module: Sends data to the cloud.
    - Cloud/Interface module: Visualizes data and sends commands.
- Circuit Design:
  - Use a tool like Fritzing or Tinkercad to design the hardware connections.
- Data Flow:
  - Define how data flows through the system:
    Sensor → Arduino → Wi-Fi Module → Cloud → User Interface.

4. Development Stages

Stage 1: Setup and Testing of Arduino Uno

- Install Arduino IDE and required drivers.
- Test basic functionality by running a sample code like blinking an LED.

Stage 2: Sensor Integration

- Connect the sensors (e.g., DHT11).
- Write code to read data from the sensors and print it to the Serial Monitor.
- Validate sensor readings against real-world values.

Stage 3: Communication Module Setup

- Configure the ESP8266 or another Wi-Fi module.
- Write code to connect the module to a Wi-Fi network.
- Test basic connectivity by pinging a server.

Stage 4: Cloud Integration

- Choose a cloud platform (e.g.,Thingspeak, Firebase).
- Write code to send sensor data to the cloud.
- Test data reception and visualization on the cloud.

Stage 5: User Interface Development (Optional)

- Develop a simple web dashboard or mobile app to display data.
- Use tools like HTML/CSS/JavaScript for web dashboards or mobile frameworks like Flutter.

Stage 6: Real-Time Monitoring and Alerts

- Implement logic for alerts (e.g., send notifications if temperature exceeds a threshold).

Stage 7: Final Integration

- Assemble all components into a working prototype.
- Test the system as a whole, ensuring smooth operation.

5. Testing and Debugging

- Validate each module independently (e.g., sensor accuracy, connectivity).
- Test the system under real-world conditions.
- Debug issues like data loss, connectivity errors, or power fluctuations.

6. Deployment

- Deploy the system at the intended location.
- Ensure reliable power and internet connectivity.

- Monitor system performance and refine as needed.

7. Documentation

- Create a detailed report including:
    - Circuit diagrams and block diagrams.
    - Source code with comments.
    - Instructions for setup and operation.
    - Troubleshooting tips.

8. Project Evaluation

- Performance Metrics:
    - Accuracy of sensor readings.
    - Reliability of data transmission to the cloud.
    - User interface responsiveness.
- Improvements:
    - Identify limitations and plan for future enhancements (e.g., better hardware, additional features).

Flow Summary

1. Research & Design → Define problem, gather requirements, and design the system.
2. Development → Build modules (sensors, communication, cloud, UI) and integrate them.
3. Testing → Validate each module and the complete system.
4. Deployment → Install and monitor performance in the intended environment.
5. Evaluation → Analyze results and propose improvements.

CHAPTER - II

LITERATURE SURVEY

LITERATURE SURVEY

A literature survey provides a review of existing studies, projects, and resources relevant to a topic. For a project involving a traffic light system using Arduino Uno, the literature survey focuses on prior work in traffic signal automation, methodologies used, challenges, and the role of microcontrollers like Arduino in addressing traffic management problems.

1. Introduction

Traffic light systems are essential for regulating vehicular and pedestrian traffic at intersections. Traditional systems often follow fixed timing cycles, leading to inefficiencies such as traffic congestion during peak hours or unnecessary delays during low traffic periods. The use of Arduino Uno in traffic light systems has gained popularity due to its simplicity, cost-effectiveness, and versatility in implementing both fixed and adaptive timing mechanisms.

2. Existing Systems

2.1 Traditional Traffic Light Systems

- Operate on fixed timing intervals irrespective of traffic density.
- Limited scope for adaptability or real-time decision-making.
- Require complex and expensive controllers for implementation.

2.2 Modern Traffic Light Systems

- Adaptive systems using real-time data from sensors like IR sensors, ultrasonic sensors, or cameras.
- Integration with Internet of Things (IoT) for remote monitoring and control.
- Focus on energy efficiency using LED lights and solar power.

3. Use of Arduino Uno in Traffic Light Systems

Arduino Uno has emerged as a popular platform for prototyping traffic light systems due to its features:

- Cost-Effective Microcontroller: Affordable compared to industrial-grade controllers.
- Ease of Programming: Simple coding environment using Arduino IDE.
- Integration with Sensors: Allows connection with IR sensors, ultrasonic sensors, and other modules for adaptive systems.
- Scalability: Supports additional components like Bluetooth modules, Wi-Fi modules, or GSM modules for IoT integration.

4. Key Studies and Projects

4.1 Basic Traffic Light System

- Description: Uses Arduino Uno to control three LEDs (Red, Yellow, Green) that simulate a traffic light.
- Features: Fixed timing intervals with simple logic.
- Limitations: Lacks adaptability to real-time traffic conditions.

4.2 Adaptive Traffic Light System

- Description: Incorporates sensors to detect traffic density and adjust signal timing dynamically.
- Example: IR sensors count vehicles at an intersection, and the Arduino adjusts the green light duration based on queue length.
- Benefits: Reduces congestion and improves traffic flow.

4.3 IoT-Based Traffic Light System

- Description: Combines Arduino Uno with a Wi-Fi module (e.g., ESP8266) to connect traffic lights to a cloud platform.
- Applications: Enables remote monitoring, real-time updates, and control from a central system.
- Challenges: Requires stable internet connectivity and additional hardware.

4.4 Emergency Vehicle Priority System

- Description: Uses RF modules or GPS to detect approaching emergency vehicles (e.g., ambulances) and prioritizes green lights on their path.
- Impact: Reduces delays for emergency services, potentially saving lives.

5. Methodologies

5.1 Hardware Components Used

- Arduino Uno: The central controller for signal logic.
- LEDs: Represent red, yellow, and green lights.
- Sensors: IR or ultrasonic sensors for vehicle detection.
- Power Supply: USB or external power source for the Arduino.
- Modules (Optional): Wi-Fi or Bluetooth modules for IoT-enabled systems.

5.2 Software

- Arduino IDE for writing and uploading code.
- Libraries for interfacing with sensors and modules.

5.3 Logic Implementation

- Fixed Timing Logic: Predefined durations for each light.
- Dynamic Timing Logic: Adjusted based on real-time traffic data from sensors.

6. Challenges

- Traffic Density Measurement: Ensuring accurate detection of vehicles, especially in poor weather or low-light conditions.
- Scalability: Expanding a simple Arduino-based model for real-world intersections.
- Power Management: Managing power consumption for energy efficiency.
- Real-Time Adaptation: Ensuring that the system responds quickly and reliably to traffic conditions.

7. Applications

- Educational projects to demonstrate the principles of traffic control systems.
- Small-scale implementations in gated communities or private parking lots.
- Prototypes for adaptive traffic management systems.

8. Conclusion

Arduino Uno offers a robust and accessible platform for implementing traffic light systems, ranging from basic fixed-cycle models to advanced adaptive or IoT-enabled solutions. While the simplicity and affordability of Arduino Uno make it ideal for educational and small-scale projects, integrating sensors and communication modules can significantly enhance its utility for real-world applications

CHAPTER - III

SYSTEM DEVELOPMENT

3.SYSTEM DEVELOPMENT

3.1 HARDWARE REQUIREMENTS

In this project, we will design a simple traffic light system using an Arduino Uno. This system will simulate the traffic light functioning (Red, Yellow, Green) for a crossroad and can be expanded to a full-fledged system with additional features.

1. Arduino Uno
    - The microcontroller board that will control the logic of the traffic light system.
2. LEDs (3 LEDs for each traffic light)
    - Red LED – to represent the stop signal.
    - Yellow LED – to represent the caution or prepare-to-stop signal.
    - Green LED – to represent the go signal.

      For a simple crossroad system, you will need:
    - 3 LEDs for the East-West direction.
    - 3 LEDs for the North-South direction.
3. Resistors
    - 220 ohms resistors (x6) – to limit current through the LEDs and prevent them from burning out.
4. Breadboard
    - A board to connect all the components together.
5. Jumper Wires
    - To connect the components (Arduino to LEDs and breadboard).
6. Push Button (optional)
    - If you want to implement a pedestrian button to control the traffic lights for pedestrians (optional).

7. Power Source
    - USB cable for powering Arduino from your computer or an external 9V battery.

3.2 SOFTWARE REQUIREMENTS

Arduino IDE

- Description: The Arduino IDE is the primary development environment for programming the Arduino Uno board.
- Download: Arduino IDE
- Purpose:
    - Writing and compiling the code.
    - Uploading the code to the Arduino Uno.
    - Debugging with the Serial Monitor.

3.3 SYSTEM DESIGN:

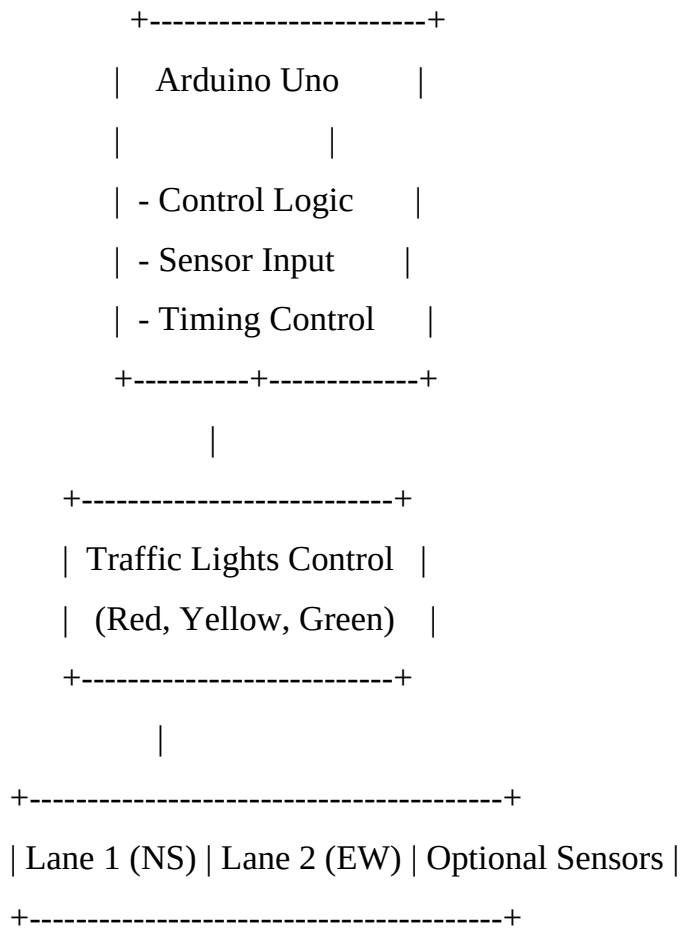
Designing a Traffic Light System using Arduino Uno involves several steps: from hardware design (connecting components) to writing the program (control logic). Below is a comprehensive system design that can help you build the project.

System Overview

A basic Traffic Light System consists of three main components:

1. Traffic Lights (Red, Yellow, Green) for each lane (North-South, East-West).
2. Arduino Uno as the control unit for the system.
3. Sensors (optional) for detecting cars (e.g., infrared or ultrasonic sensors).
4. Timing Logic to control light switching intervals.
5. The system should mimic real-world traffic light behavior, where traffic lights change based on preset timing or sensor input (for more advanced setups).

Block Diagram



- Arduino Uno controls the traffic lights and their transitions.
- Traffic lights are connected to pins on the Arduino and toggled to display Red, Yellow, or Green.
- Optional sensors may be used to detect the presence of vehicles and adjust the traffic light cycles accordingly.

Hardware Components

- Arduino Uno – The microcontroller that runs the logic.
- LEDs – Used to represent the Red, Yellow, and Green lights for each lane.
    - 6 LEDs in total (2 for each direction: North-South and East-West).
- Resistors – 220Ω to limit the current through the LEDs.
- Push Buttons (optional) – For manually controlling traffic lights or for pedestrian crossing signals.
- Wires & Breadboard – For connections and prototyping.

Circuit Design

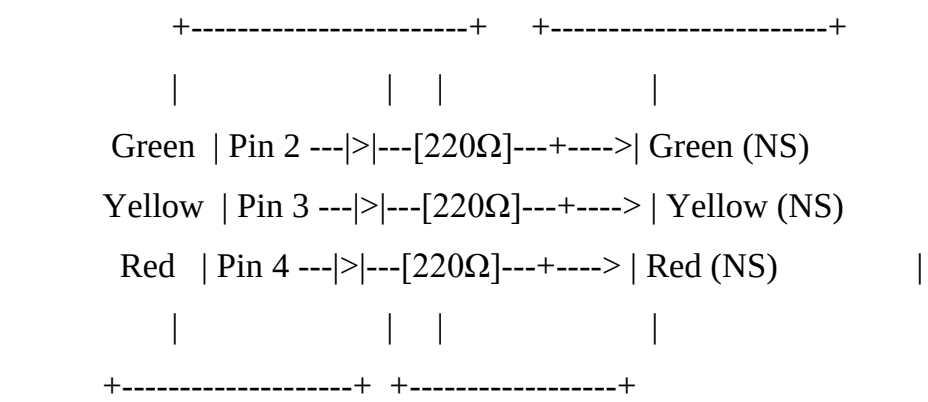
The following is a simple Traffic Light System Circuit where we use 6 LEDs (two for each direction: North-South and East-West) to represent traffic lights:

1. Traffic Light LED Setup:
    - NS Lane: 1 Green LED, 1 Yellow LED, 1 Red LED.
    - EW Lane: 1 Green LED, 1 Yellow LED, 1 Red LED.
2. Connections:
    - Connect each LED to the digital I/O pins of the Arduino via a 220Ω resistor to limit current.
    - LEDs for the NS and EW lanes can be connected to digital pins on the Arduino (e.g., pins 2 to 7).
3. Basic Wiring Diagram:
    - Green, Yellow, Red LEDs for NS lane connected to pins 2, 3, and 4.
    - Green, Yellow, Red LEDs for EW lane connected to pins 5, 6, and 7.
4. Optional Sensor Setup:
    - If you plan to use sensors (e.g., ultrasonic or infrared), connect them to analog or digital pins on the Arduino (e.g., pins 8, 9).

Diagram Example:

plaintext

Copy code



5. Control Logic Design

The traffic light system can be controlled using a basic timing algorithm to toggle the lights in a cycle, or based on sensor input (advanced version).

A. Without Sensor Input (Simple Timed Traffic Light)

1. Green Light:
    - North-South gets Green for 10 seconds, while East-West gets Red.
2. Yellow Light:
    - After Green for 10 seconds, North-South gets Yellow for 2 seconds, while East-West stays Red.
3. Red Light:
    - Then, switch to Red for North-South, and Green for East-West for 10 seconds.
4. Yellow Light (EW):
    - After the Green light for East-West, it switches to Yellow for 2 seconds.
5. Repeat:
    - The process continues to alternate between North-South and East-West.

B. With Sensor Input (Advanced)

For advanced traffic lights with sensors (e.g., infrared or ultrasonic sensors), we can detect vehicles at intersections. Based on the number of vehicles detected, we can change the light cycle dynamically:

- Sensors detect cars approaching the traffic light.
- If the NS lane sensor detects a vehicle, the green light will remain on for a longer duration.
- The EW lane sensor will work similarly, ensuring that traffic flow is managed based on real-time conditions.

6. Future Enhancements

- Sensor Integration: Use sensors to detect cars, triggering longer green lights when necessary. Pedestrian Crossing: Add pedestrian signal control with a button or sensor.
- Traffic Density Management: Integrate an advanced algorithm to change the light cycle based on vehicle density.

7. Testing and Debugging

- Testing: Test the system in different conditions (e.g., no vehicles, heavy traffic) and ensure proper timing and light switching.
- Debugging: Ensure correct wiring and verify that the Arduino is correctly controlling the LEDs as per the logic.

3.4 MODEL DEVELOPMENT

A traffic light system is a great beginner project for learning how to use Arduino Uno for controlling traffic flow at intersections. In this project, we will simulate a basic traffic light with red, yellow, and green LEDs. The system will cycle through the lights to simulate a real-world traffic signal.

Working of the Model:

1. **Normal Operation:**
    - **Red Light** stays on for 5 seconds, then changes to **Yellow** for 2 seconds, followed by **Green** for 5 seconds.
    - This cycle repeats continuously.
2. **Pedestrian Mode (optional):**
    - If the button is pressed, the traffic lights will be turned to red for all vehicles, and the pedestrian crossing mode will simulate a pedestrian light (you could extend the logic to trigger an actual pedestrian light LED).

Possible Enhancements:

1. **Add Multiple Traffic Lights:**
    - You can add more traffic light sets for multi-lane intersections and manage timing for each set.
2. **Add Real-Time Clock:**
    - Use a **real-time clock (RTC)** module to control traffic lights based on the time of day (e.g., longer green light duration during peak hours).
    -

3. **IoT Integration:**
    - You can use an **ESP8266 Wi-Fi module** or an **Arduino with Wi-Fi** to send traffic light data to a cloud platform or a mobile app, enabling remote monitoring and control.
4. **Sensor Integration:**
    - Add **infrared or ultrasonic sensors** to detect vehicles at the intersection and dynamically adjust the traffic light cycle to improve traffic flow.

3.5 DATA FLOW

In a Traffic Light Control System using Arduino Uno, the system controls the traffic lights (red, yellow, green) based on timing or sensor input (e.g., vehicle presence). The goal is to automate the switching of traffic lights to ensure smooth traffic flow.

Below is the data flow for such a system:

System Overview

The system consists of three main components:

- Arduino Uno (acts as the controller).
- Traffic Light Signals (Red, Yellow, Green LEDs).
- Sensors (Optional) – Used for detecting vehicle presence at intersections (inductive sensors, ultrasonic sensors, or IR sensors).

Data Flow Diagram (DFD)

Level 1: High-Level Data Flow

- Input:
    - Time-based control (fixed timing for each light cycle).
    - Optional: Sensor data for vehicle detection (if smart traffic system is implemented).
- Output:
    - Control signals to traffic lights (LEDs - Red, Yellow, Green).

Basic Flow Without Sensors (Timed Control):

css

Copy code

[System Start] --> [Start Timer] --> [Determine Light Cycle]

|

[Control Light Signals]

|

[Traffic Light System] --> [Red Light], [Yellow Light], [Green Light] --> [Repeat Cycle]

Flow with Vehicle Sensors (Smart Traffic):

css

Copy code

[Vehicle Detected] --> [Sensor Input] --> [Arduino Controller]

|

[Decision Based on Vehicle Presence] --> [Change Traffic Lights]

|

[Traffic Light Signals] --> [Red, Yellow, Green Lights] --> [Repeat Cycle]

Detailed Data Flow Explanation

A. System Initialization

1. The Arduino Uno is powered on and initialized.
2. The system begins with all traffic lights turned off or set to the initial state (e.g., red light on for all directions).

B. Timer-Based Control (Fixed Timing)

1. The system sets timers for each of the lights (e.g., red for 30 seconds, green for 45 seconds, yellow for 5 seconds).
2. Based on the timer values, the Arduino sends the appropriate signals to the traffic lights:
    - Green Light: Activates for a fixed duration (e.g., 45 seconds).
    - Yellow Light: Activates for a fixed duration (e.g., 5 seconds).
    - Red Light: Activates for the remaining duration after green and yellow phases.
3. The cycle repeats indefinitely, switching between the red, yellow, and green lights.

C. Sensor-Based Control (Optional)

1. Vehicle Presence Detection:
    - Sensors like IR, Ultrasonic, or Inductive Loop detect vehicles at the intersection or at the traffic light.

    - For example, an IR sensor detects when a car is approaching the red light or waiting at a green light.
2. Sensor Data Input:
    - If a vehicle is detected by the sensor, the Arduino Uno receives this input.
3. Decision Making:
    - If a vehicle is detected on a red light, the Arduino can switch the light to green earlier than scheduled to accommodate the vehicle.
    - Similarly, if no vehicle is detected at a red light, the Arduino may skip or shorten the green light to optimize traffic flow.
4. Output:
    - The Arduino sends control signals to the traffic lights, activating the red, yellow, or green lights based on sensor data.

Example Data Flow - Timer-based Traffic Light System (Without Sensors)

1. Arduino Uno Initialization:
    - The system powers on.
    - The system starts by turning all traffic lights to red (initial state).
    - The Arduino begins the timer for the green light duration (say 30 seconds).
2. Green Light Phase:
    - After 30 seconds, the Arduino switches the traffic light to yellow for 5 seconds.
3. Yellow Light Phase:
    - After 5 seconds, the Arduino switches the traffic light to red for the next 30 seconds, and the cycle continues.

Data Flow with Example Code

Here is a basic example where timed control is implemented (without sensors):

Hardware Setup:

- LEDs: Red, Yellow, and Green connected to Arduino Uno.
- Resistors: Appropriate value (e.g., 220Ω) for each LED.

Data Flow Summary for Timed Traffic Light Control

1. System Start:
    - Initialize Arduino and set traffic lights to red.

2. Start Timer:
    - Start a fixed timer for the green light.
3. Determine Light Cycle:
    - Switch traffic lights to green, then yellow, then red based on the timer.
    - Repeat the cycle indefinitely.
4. Output to Traffic Lights:
    - Red, Yellow, or Green lights are activated for specified durations.

Data Flow Summary for Sensor-based Control

1. System Start:
    - Initialize Arduino and set traffic lights to red.
2. Sensor Input:
    - Sensors detect vehicles at the intersection.
3. Decision Based on Sensor Data:
    - If a vehicle is detected, change the traffic light accordingly.
    - If no vehicles are detected, maintain the cycle.
4. Control Traffic Lights:
    - Output signals to change the state of the red, yellow, and green lights.

3.6 SOFTWARE TESTING

Software testing is an essential phase in the development process of any project, including an **Arduino-based traffic light system**. The goal of software testing in this context is to ensure that the code functions as expected and the traffic light system operates correctly.

Below is a detailed approach to **software testing** for the **Arduino Uno Traffic Light System** project:

1. Test Plan Overview

In the traffic light system, the objective is to manage the light transitions (Green → Yellow → Red) for each direction (East-West, North-South) in a simulated traffic environment.

Key Components:

- **Arduino Uno:** Controls the traffic lights.
- **LEDs:** Represent the Red, Yellow, and Green lights for each direction (e.g., East-West and North-South).
- **Timing:** Ensures the lights change at the correct intervals.

- **State Machine:** Manages the different states of the traffic light (e.g., Green, Yellow, Red).

3. **Test Cases for Traffic Light System**

Test Case 1: Correct Light Transitions

- **Objective:** Ensure that the traffic light transitions from one color to another correctly in a set sequence (Green → Yellow → Red).
- **Steps:**
    1. Power the system on.
    2. Observe if the **East-West** direction starts with a **Green LED**.
    3. After a set time (e.g., 10 seconds), ensure it changes to **Yellow LED**.
    4. After another set time (e.g., 2 seconds), ensure it changes to **Red LED**.
    5. Ensure that the **North-South** direction follows the same sequence with its own LEDs.
- **Expected Result:** The traffic lights should follow the sequence: **Green → Yellow → Red** for each direction at the correct intervals.

Test Case 2: Synchronization of Lights

- **Objective:** Ensure the traffic lights for **East-West** and **North-South** directions do not turn green simultaneously (to avoid collisions).
- **Steps:**
    1. Power the system on.
    2. Observe the **East-West** and **North-South** lights.
    3. Check if when **East-West** is Green, **North-South** is Red, and vice versa.
- **Expected Result:** One direction should be **Green** at a time, while the other is **Red**.

Test Case 3: Timing Accuracy

- **Objective:** Ensure that the timing for each light (Green, Yellow, Red) is accurate.
- **Steps:**
    1. Set the timing for each light: Green for 10 seconds, Yellow for 2 seconds, and Red for 10 seconds.
    2. Power on the system and use a stopwatch to measure how long each light stays on.
    3. Ensure the transitions occur at the correct times.
- **Expected Result:** The Green light should stay on for 10 seconds, Yellow for 2 seconds, and Red for 10 seconds.

Test Case 4: Power Failure Recovery

- **Objective:** Ensure that the system handles power failure or resets without losing the state.

- **Steps:**
    1. Power on the system.
    2. Let the system run for a while.
    3. Disconnect the power supply for 2-3 seconds and then reconnect it.
    4. Observe if the traffic light system resumes correctly from the previous state.
- **Expected Result:** The system should resume operation, either from the start of a cycle or based on the last known state.

Test Case 5: Debugging with Serial Monitor

Objective: Ensure that the system's state can be monitored via the **Arduino**

Serial Monitor for debugging purposes.

- **Steps:**
    1. Modify the code to output state changes to the Serial Monitor (e.g., "Green Light", "Yellow Light", "Red Light").
    2. Open the Serial Monitor in Arduino IDE.
    3. Observe if the Serial Monitor logs the light transitions correctly.
- **Expected Result:** The Serial Monitor should print the appropriate messages as the lights change.

Test Execution

Execute the test cases described above.

- Use a stopwatch to verify the timing.
- Use **Serial Monitor** for debugging and to verify state transitions.
- Manually observe the LED behavior to ensure proper color transitions.

Reporting and Fixing Issues

- **Logs and Findings:** After performing the tests, record any issues or deviations from the expected results.
- **Bug Fixes:** If the lights do not transition properly or if there is a timing issue, review the code logic and the hardware setup.
    - Verify correct connections.
    - Ensure timing (delay) is implemented correctly.
    - Test with different timing intervals to validate the accuracy.

CHAPTER - IV

PROPOSED SYSTEM

4.1 ALGORITHM

Order Placement Algorithm:

Step 1: User selects food items from the menu and adds them to the cart.

Step 2: User proceeds to checkout where the total price is calculated.

Step 3: User logs in (or creates an account) and enters payment details.

Step 4: Payment is processed through Stripe API.

Step 5: If payment is successful, create a new order in the database and update the order status to Pending.

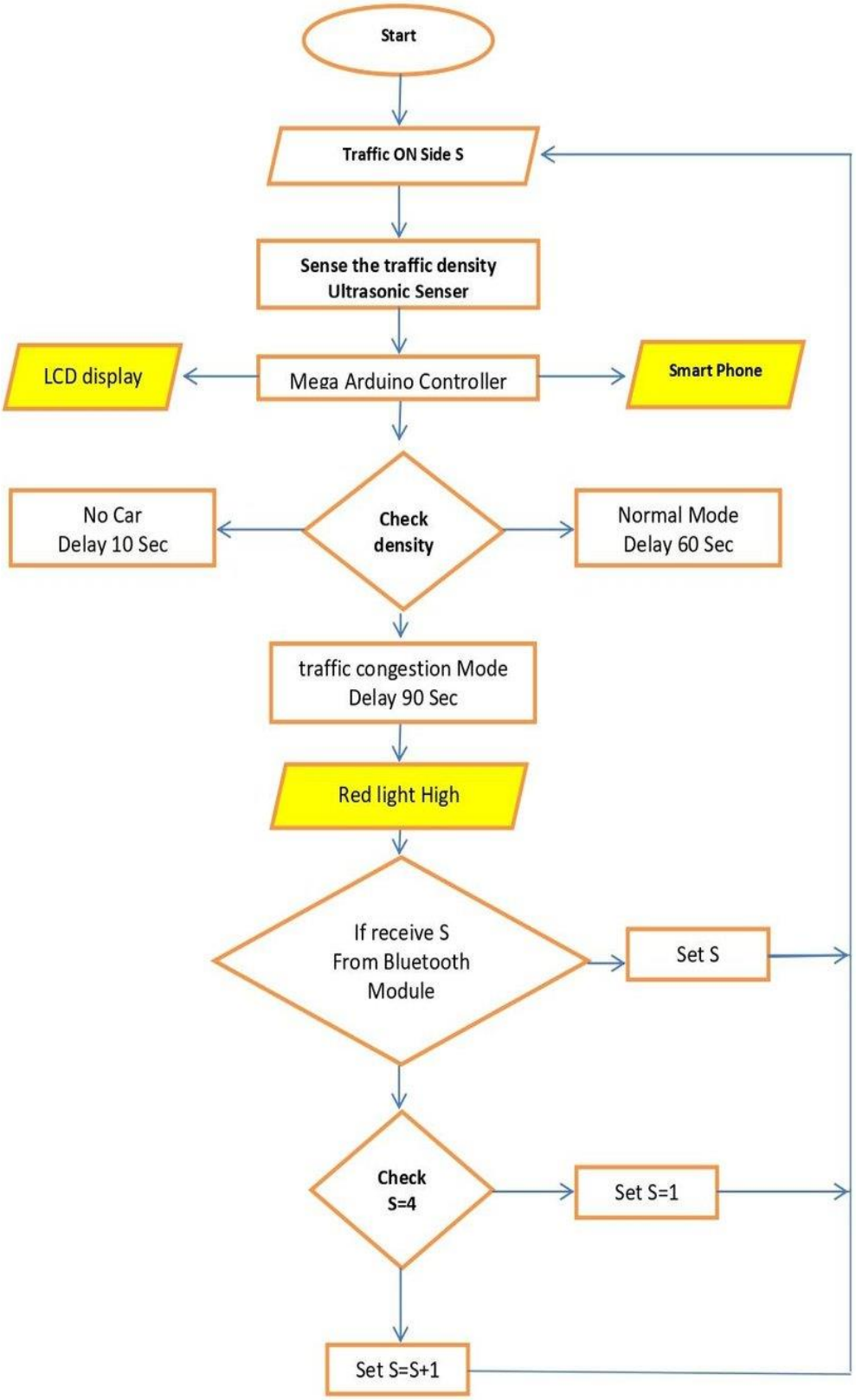
Step 6: Notify the admin that a new order has been placed.

Step7: Admin processes the order (e.g., changes status to "Preparing").

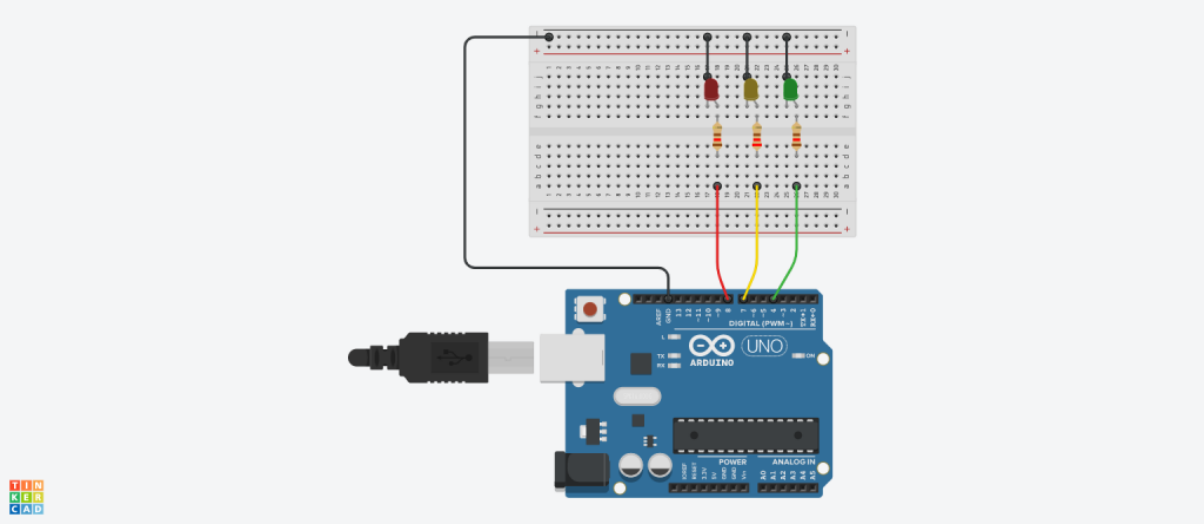
Step 8: User can track the order status.

Step 9: Admin updates the status (e.g., "Shipped" or "Delivered") and notifies the user.

4.2 FLOW OF PROJECT



4.3 ER-Diagram



CHAPTER - V

RESULT AND DISCUSION

5.RESULT AND DISCUSSION

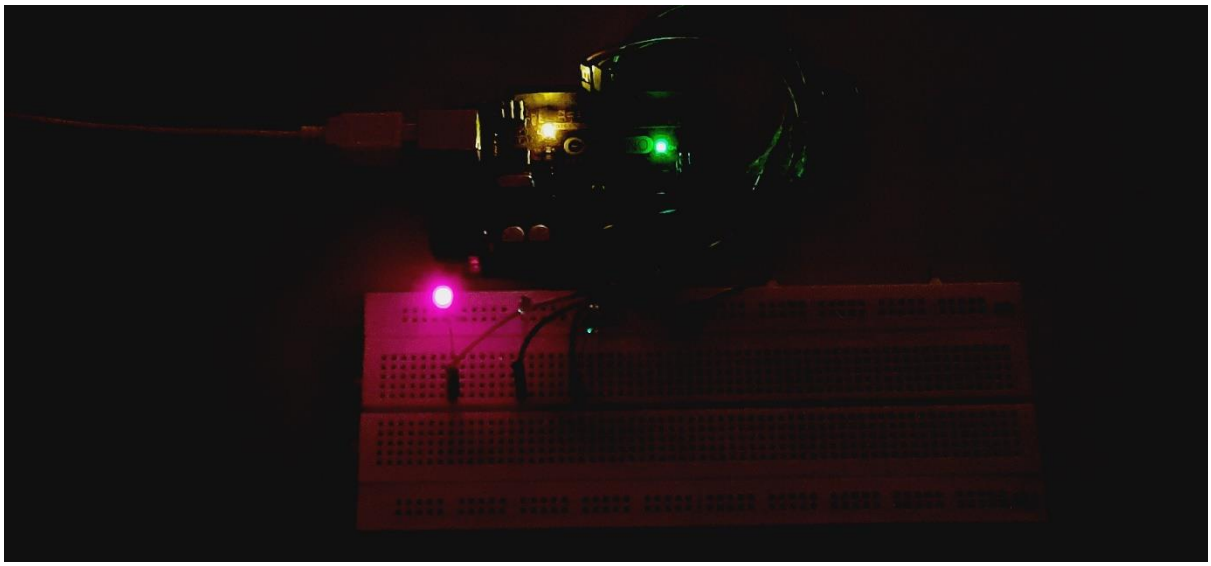


Figure 5.1: RED Led Blink

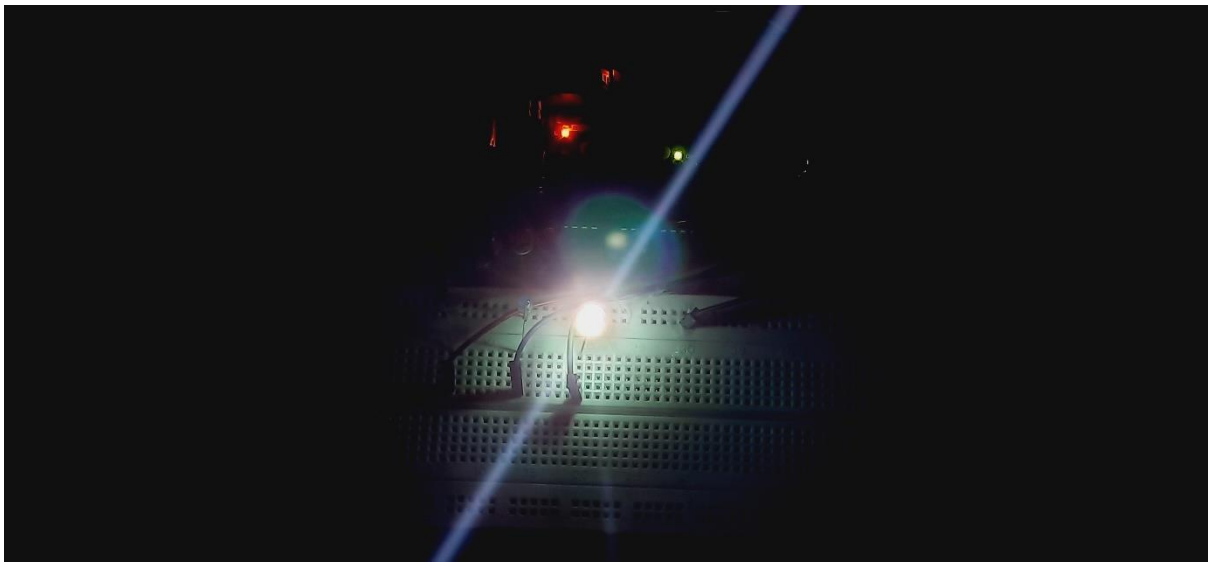


Figure 5.2: YELLOW Led Blink

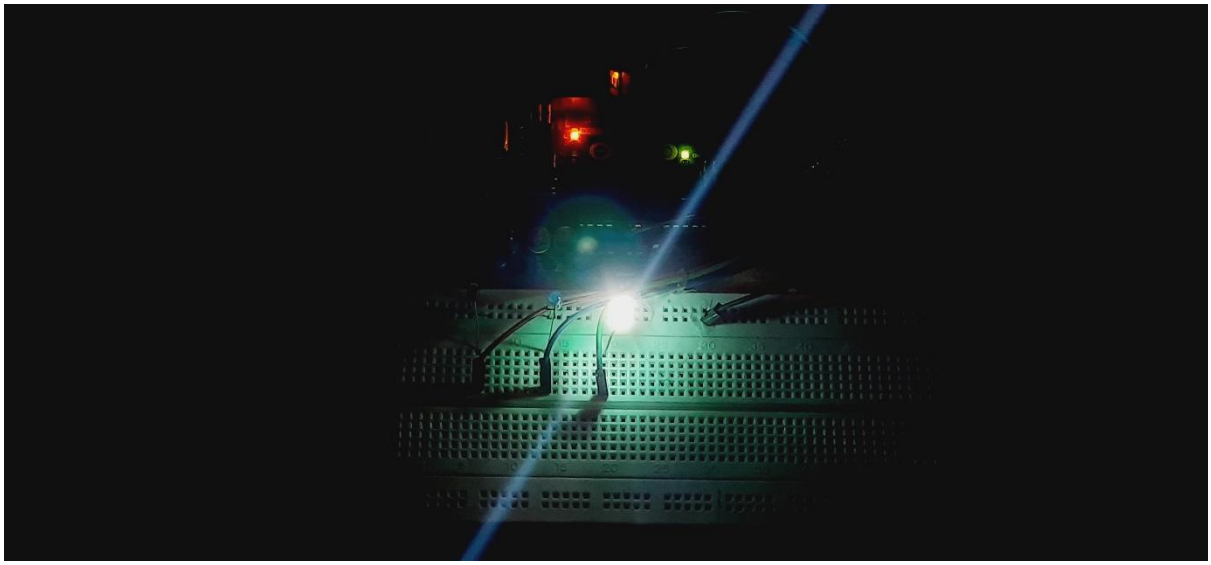


Figure 5.3: GREEN Led Blink

APPENDIX:

CODING:

```
int red = 9;
int yellow = 8;
int green = 7;

void setup(){

  pinMode(red, OUTPUT);
  pinMode(yellow, OUTPUT);
  pinMode(green,  OUTPUT);

}
void loop(){
digitalWrite(red, HIGH);
 delay(15000);
digitalWrite(red,  LOW);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow,  LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow,  LOW);
```

```
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow,  LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow, LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow, LOW);
delay(500);

digitalWrite(green, HIGH);
delay(20000);
digitalWrite(green,  LOW);
digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow,  LOW);
delay(500);

  digitalWrite(yellow, HIGH);
```

```
delay(1000);
  digitalWrite(yellow,  LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow, LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow, LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow, LOW);
delay(500);
}
```

5.1 Comparison Table

Feature	Existing system	Proposed System
Hardware Complexity	High	Low
Cost	High	Low
Ease of Implementation	Requires specialized skills	Beginner-friendly with basic knowledge
Scalability	High	Limited to small-scale demonstrations
Customizability	Limited by hardware	Easily customizable through code

CHAPTER -VI

CONCLUSION

6. CONCLUSION

The Traffic Light Control System using Arduino Uno successfully demonstrated the principles of traffic signal automation using microcontrollers. The system simulated a real-world traffic light, efficiently managing the flow of traffic by implementing predefined time intervals for red, yellow, and green lights.

This project showcased:

1. **Effective Use of Arduino Uno**: The Arduino Uno provided a cost-effective and user-friendly platform to implement and control traffic light sequencing.
2. **Basic Programming Skills**: The project reinforced fundamental skills in programming using the Arduino IDE and introduced concepts like timers and conditional logic.
3. **Hardware Integration**: LEDs, resistors, and jumper wires were effectively integrated to simulate a functioning traffic light system.

The system is scalable and can be enhanced further by:

1. **Adding Sensors**: Incorporating IR or ultrasonic sensors for dynamic traffic management based on real-time traffic density.
2. **Integration with IoT**: Enabling remote monitoring and control using IoT devices.
3. **Pedestrian Crosswalk Integration**: Adding push-button controls for pedestrian crossings

Reference:

- 1. Arduino Official Website:
- The official Arduino website (https://www.arduino.cc/) provides comprehensive documentation, tutorials, and examples for getting started with Arduino boards and projects.
- 2. Arduino Workshop: A Hands-On Introduction with 65 Projects" by John Boxall