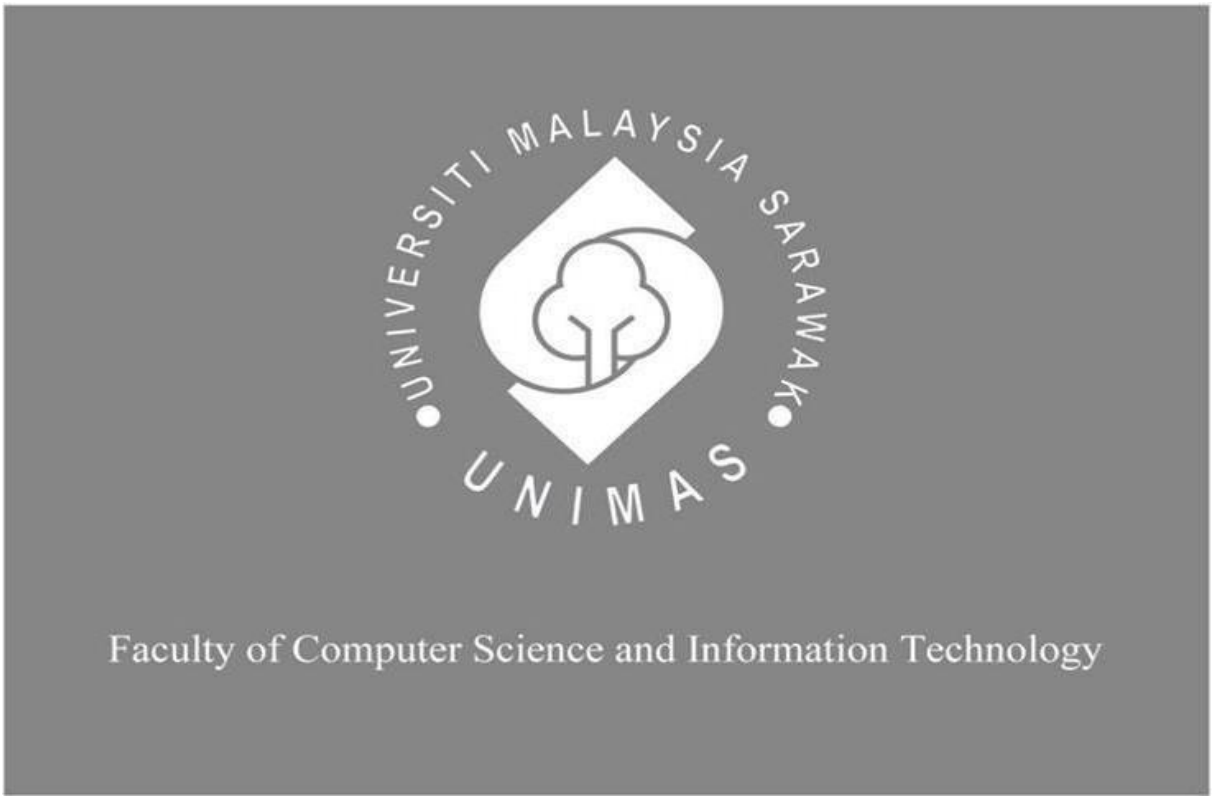

Portable Wireless Traffic Control System for Partial Road Closures

Awang Nabil Muqri bin Awang Selimi

Bachelor of Computer Science with Honours (Software Engineering)

2024

PORTABLE WIRELESS TRAFFIC CONTROL SYSTEM FOR PARTIAL ROAD CLOSURES

AWANG NABIL MUQRI BIN AWANG SELIMI

This project is submitted in partial fulfilment of the
requirements for the degree of
Bachelor of Computer Science with Honours (Software Engineering)

Faculty Of Computer Science and Information Technology
UNIVERSITI MALAYSIA SARAWAK
2024

UNIVERSITI MALAYSIA SARAWAK

THESIS STATUS ENDORSEMENT FORM

TITLE PORTABLE WIRELESS TRAFFIC CONTROL SYSTEM FOR PARTIAL ROAD CLOSURES

ACADEMIC SESSION: 2024/2025

(CAPITAL LETTERS)

hereby agree that this Thesis* shall be kept at the Centre for Academic Information Services, Universiti Malaysia Sarawak, subject to the following terms and conditions:

1. The Thesis is solely owned by Universiti Malaysia Sarawak
2. The Centre for Academic Information Services is given full rights to produce copies for educational purposes only
3. The Centre for Academic Information Services is given full rights to do digitization in order to develop local content database
4. The Centre for Academic Information Services is given full rights to produce copies of this Thesis as part of its exchange item program between Higher Learning Institutions [or for the purpose of interlibrary loan between HLI]
5. ** Please tick (√)

☐	CONFIDENTIAL	(Contains classified information bounded by the OFFICIAL SECRETS ACT 1972)
☐	RESTRICTED	(Contains restricted information as dictated by the body or organization where the research was conducted)
☐	UNRESTRICTED	

(AUTHOR'S SIGNATURE)

Validated by

(SUPERVISOR'S SIGNATURE)

Permanent Address

NO 313, Blok C, Kampung Haji Baki,
Jalan Batu Kitang,
93250, Kuching, Sarawak.

Date: 23 June 2025

Date: _____

Note * Thesis refers to PhD, Master, and Bachelor Degree
** For Confidential or Restricted materials, please attach relevant documents from relevant organizations / authorities

DECLARATION

I hereby declare that this project titled “Portable Wireless Traffic Control System for Partial Road Closures” is the result of my own work, except where explicitly stated otherwise. All data and information in this report were gathered and presented according to the academic standards and ethical guidelines. I affirm that this work has not been submitted in any form for any degree or diploma at any other university or institution. Appropriate credit has been given for any work or sources referenced or used throughout this report.

Date: 17th January 2025

(AWANG NABIL MUQRI BIN AWANG SELIMI)

Faculty Of Computer Science and Information Technology

University Malaysia Sarawak

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to everyone who has supported and guided me throughout my final year project. First and foremost, I would like to thank Dr. Lau Sei Ping, my Final Year Project supervisor, for giving me the opportunity to embark on this project. I deeply appreciate her continuous support, guidance, and insightful feedback, which has been invaluable in helping me overcome challenges and refine my work towards achieving its objectives.

I would also like to express my sincere thanks to the traffic warden who participated in the interview and shared their insights into traffic management and the challenges they face during partial road closures. Their input has been crucial in shaping the direction of this project. Additionally, I am grateful to the individuals who contributed through questionnaires and surveys, whose responses provided essential data for the development of this system.

Finally, I would like to thank my friends, whose constant encouragement and support have been a source of motivation throughout this project. Whether it was lending an ear during stressful times or offering helpful suggestions, they have made this journey easier and more enjoyable. I am also extremely grateful to my family for their unwavering support, patience, and understanding, particularly during challenging moments. Their belief in me has been a constant source of strength and inspiration, and I would not have been able to complete this project without them. Thank you to all for making this journey an enriching, fulfilling, and memorable experience.

ABSTRACT

The management of traffic during partial road closures is a significant challenge, particularly in rural areas with limited infrastructure. The Portable Wireless Traffic Control System for Partial Road Closures aims to address this issue by developing an offline, portable system that enables traffic wardens to efficiently manage traffic signals through real-time synchronization and manual overrides. The system employs a Master-Slave architecture, where two ESP32 microcontrollers communicate wirelessly to control two traffic signals, ensuring seamless operation during road closures. The system also includes a mobile application that allows traffic wardens to monitor traffic signal statuses, adjust countdown timers, and override signal sequences. This project follows an iterative development approach, utilizing Agile methodology to incorporate continuous feedback and refine the system according to user needs. The system is designed to be portable, robust, and easy to use, even in areas with no internet connectivity. Through an interview and surveys with traffic wardens and road users, the project identified key features required for an effective traffic control system. The results emphasize the importance of real-time adaptability, manual control, and synchronization between traffic signals, all of which are addressed in the Portable Wireless Traffic Control System. The findings highlight the system's potential to improve traffic flow, reduce accidents, and provide better control during temporary road closures.

ABSTRAK

Pengurusan lalu lintas semasa penutupan jalan sementara adalah cabaran yang signifikan, terutamanya di kawasan luar bandar yang mempunyai infrastruktur terhad. Sistem Kawalan Lalu Lintas Tanpa Wayar Mudah Alih untuk Penutupan Jalan Sementara bertujuan untuk menangani isu ini dengan membangunkan sistem luar talian dan mudah alih yang membolehkan pengawal trafik menguruskan isyarat trafik dengan cekap melalui penyelarasan masa nyata dan kawalan manual. Sistem ini menggunakan seni bina Master-Slave, di mana dua mikrodenyar ESP32 berkomunikasi tanpa wayar untuk mengawal dua isyarat trafik, memastikan operasi yang lancar semasa penutupan jalan. Sistem ini juga merangkumi aplikasi mudah alih yang membolehkan pengawal trafik memantau status isyarat trafik, melaraskan pemasa undur, dan menggantikan urutan isyarat. Projek ini mengikut pendekatan pembangunan berulang, menggunakan metodologi Agile untuk mengambil kira maklum balas berterusan dan memperhalusi sistem mengikut keperluan pengguna. Sistem ini direka untuk menjadi mudah alih, kukuh, dan mudah digunakan, walaupun di kawasan yang tidak mempunyai sambungan internet. Melalui temubual dan tinjauan bersama pengawal trafik dan pengguna jalan raya, projek ini telah mengenal pasti ciri-ciri utama yang diperlukan untuk sistem kawalan trafik yang berkesan. Hasil kajian menekankan kepentingan penyesuaian masa nyata, kawalan manual, dan penyelarasan antara isyarat trafik, yang semuanya ditangani dalam Sistem Kawalan Lalu Lintas Tanpa Wayar Mudah Alih. Penemuan ini menekankan potensi sistem untuk meningkatkan aliran trafik, mengurangkan kemalangan, dan memberikan kawalan yang lebih baik semasa penutupan jalan sementara..

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ESP32	Espressif Systems 32-bit Microcontroller
MCU	Microcontroller Unit
IR	Infrared
Wi-Fi	Wireless Fidelity
LED	Light Emitting Diode
FYP	Final Year Project
GUI	Graphical User Interface
API	Application Programming Interface
Bluetooth	Wireless Technology for Short-Range Communication
App	Mobile Application
USB	Universal Serial Bus
PWM	Pulse Width Modulation
DB	Database
TCP/IP	Transmission Control Protocol/Internet Protocol
GPS	Global Positioning System
UI	User Interface
FIFO	First In, First Out
HTTP	Hypertext Transfer Protocol
SSID	Service Set Identifier
RF	Radio Frequency
UART	Universal Asynchronous Receiver-Transmitter

CHAPTER 1: INTRODUCTION

1.1 Introduction

This project, titled "Portable Wireless Traffic Control System for Partial Road Closures," is designed to provide a reliable, safe, and adaptable solution for managing traffic flow in scenarios where partial road closures occur. Manual traffic control methods, often used during such closures, expose workers to significant safety risks, including oncoming vehicles and hazardous weather conditions. These methods are also prone to miscommunication and inefficiencies, which can lead to accidents, congestion, and delays. This project aims to replace these traditional approaches with an offline, automated traffic management system that is portable, efficient, and user-friendly, making it especially suitable for rural areas with limited resources

The system consists of two portable LED traffic signals, each controlled by an ESP32 microcontroller, and infrared sensors for detecting vehicle presence. These components communicate wirelessly via a controlled communication protocol, ensuring synchronization between the traffic signals and smooth transitions between green and red lights. The system operates offline, eliminating reliance on internet connectivity, and incorporates features like countdown timers, buffer delays, and dynamic adjustments to ensure safety and efficiency. Additionally, the default system routine includes a 60-second green-light duration, with extensions of up to 30 seconds if no vehicle is detected at the opposing signal.

A mobile application serves as the system’s central interface, providing real-time monitoring and manual control capabilities for traffic wardens. The app enables users to start and stop the system, activate "all-red" signals for emergencies, override traffic signals, and adjust countdown durations dynamically. Designed with a clean and intuitive user interface, the app ensures ease of use and flexibility, allowing wardens to respond to varying traffic and road conditions effectively. By combining automation, real-time adaptability, and user-centric control, this project delivers an innovative solution to enhance traffic safety and efficiency during partial road closures.

1.2 Problem Statement

Managing traffic flow during partial road closures often relies on manual procedures, which expose road workers to significant safety risks and operational challenges. Workers are frequently required to stand near active traffic zones, making them vulnerable to oncoming vehicles and adverse weather conditions. This approach not only endangers the workers' safety but also compromises the overall efficiency of traffic management, as manual signalling methods are prone to miscommunication and delays. These risks are amplified during night operations or in areas with poor visibility, where accidents and misunderstandings become more likely.

In rural areas, these challenges are even more pronounced due to limited resources and infrastructure. Manual traffic control is often the only available option, as rural regions typically lack access to advanced traffic management systems or permanent traffic control infrastructure. Poor internet connectivity further restricts the implementation of modern, IoT-based solutions that depend on online networks. The reliance on basic and outdated methods in these regions often leads to longer delays, increased driver frustration, and heightened risks of accidents.

The Portable Wireless Traffic Control System for Partial Road Closures addresses these problems by offering an offline, portable, and adaptive solution tailored to rural and resource-limited environments. This system integrates synchronized LED traffic signals, infrared vehicle sensors, and a controlled communication protocol to automate traffic management and minimize risks. A mobile application provides real-time monitoring and manual override capabilities, empowering traffic wardens to dynamically adapt to changing traffic conditions. By enhancing safety, reducing worker exposure to hazards, and offering a practical solution for rural areas, this project provides a modern alternative to traditional manual methods.

1.3 Project Scope

This project involves the development of a Portable Wireless Traffic Control System for Partial Road Closures that operates offline to ensure portability and suitability for rural areas. The system features two portable LED traffic signals that communicate wirelessly via a controlled communication protocol, enabling synchronized and conflict-free traffic light operations. Vehicle detection sensors are integrated into the system to dynamically adjust signal timing based on real-time traffic conditions, ensuring smooth and efficient traffic flow even in resource-limited environments. The offline functionality allows the system to be deployed in remote locations where internet connectivity is unavailable.

A mobile application serves as the central interface for traffic wardens, offering real-time system monitoring and control. Key features of the app include the ability to initiate or shut down the system, activate "all-red" signals for emergencies, and manually override traffic signals or countdown timers. The app also provides live updates on the status of each traffic light, ensuring wardens can monitor operations effectively. Designed for user-friendliness, the app empowers traffic wardens to adapt to varying traffic scenarios and address unexpected conditions with ease.

The system is designed to be portable, allowing for quick deployment and reconfiguration at different locations. This makes it particularly suitable for temporary road closures in rural or remote areas, where traditional traffic management resources are often unavailable. By combining automation, dynamic adaptability, and user-driven control, this project provides a versatile and practical solution for improving traffic safety and efficiency during partial road closures.

1.4 Aims and Objectives

This project aims to improve traffic management during partial road closures by developing a Portable Wireless Traffic Control System for partial road closures. The following are the project objectives:

1. To develop a portable traffic control system with two temporary traffic signals that communicate wirelessly to synchronize operations for managing traffic during partial road closures.
2. To design and implement a controlled communication protocol for reliable and consistent data exchange between the traffic lights, ensuring synchronization based on real-time traffic conditions.
3. To create a mobile application that provides manual control capabilities, allowing traffic wardens to override automated operations and adapt to varying traffic situations.

1.5 Project Methodology

Figure 1. 1 Agile Methodology Phases (Ali & Sameh, 2024)

This project adopts an Agile methodology, an iterative and flexible approach well-suited for projects requiring continuous refinement and user feedback (Kumar & Bhatia, 2012). By breaking the project into smaller phases, Agile ensures that each component is developed, tested, and improved incrementally, resulting in a reliable and user-friendly system (Kumar & Bhatia, 2012). The methodology focuses on collaboration, adaptability, and continuous improvement, aligning with the objectives of creating a safe, efficient, and portable traffic control system.

The first phase is requirement gathering, which involves collecting insights from traffic wardens and other stakeholders through surveys and interviews. The objective is to understand the key challenges faced during partial road closures, particularly in rural areas, and identify system requirements such as offline functionality, synchronized signals, and manual override capabilities. The feedback from this phase helps shape the features and functionalities of both the hardware and mobile application.

The subsequent phase is design and prototyping, where the system architecture, circuit diagrams, and mobile app wireframes are created. This phase also includes developing a low-fidelity prototype of the traffic control system to test basic functionalities like synchronization of signals, countdown timers, and buffer delays. The design phase ensures that the system meets safety requirements and is easy to use, laying a solid foundation for implementation.

The third phase focuses on development and testing, where the hardware and software components are integrated. Using ESP32 microcontrollers, the traffic signal synchronization and vehicle detection systems are implemented and connected to the mobile application. Iterative testing ensures that features such as real-time signal updates, countdown adjustments, and manual overrides function reliably under various scenarios.

Finally, the deployment and review phase involve testing the system in a controlled environment, where user feedback is gathered to identify potential improvements. The system is refined based on this feedback to enhance usability and performance. Once all objectives are met, the system is finalized and prepared for real-world application, ensuring it provides a safe, efficient, and adaptable solution for managing traffic during partial road closures.

1.6 Significance of Project

The Portable Wireless Traffic Control System for Partial Road Closures offers a transformative approach to managing traffic in temporary closure scenarios, addressing critical safety and operational challenges. By automating traffic control, the system minimizes the exposure of road workers to hazardous conditions such as oncoming traffic and adverse weather. This ensures a safer working environment while reducing the likelihood of accidents caused by human error or fatigue, especially in areas with limited visibility or during nighttime operations.

The system's offline functionality and portability make it particularly valuable for rural or remote areas where traditional traffic management resources, such as permanent traffic lights or internet-based systems, are unavailable. Its ability to operate independently of internet connectivity ensures reliability and adaptability in these regions, offering a practical solution to improve traffic flow and reduce congestion. The use of synchronized LED traffic lights with real-time vehicle detection enhances efficiency, ensuring smooth and conflict-free transitions between signals.

Additionally, the integration of a user-friendly mobile application empowers traffic wardens with advanced control options, such as manual overrides, countdown adjustments, and real-time status monitoring of the signals. This combination of automation and manual control provides unparalleled flexibility to adapt to dynamic traffic conditions, such as emergencies or construction vehicle movement. By addressing the limitations of traditional methods and introducing innovative features, this project contributes to safer, more efficient, and adaptable traffic management solutions for both urban and rural scenarios.

1.7 Project Schedule

The project schedule is shown in the Table 1 below:

Tasks	Start Date	End Date	Duration (day(s))
Brief Proposal	7/10/2024	20/10/2024	14
Feedback and Comment from Reviewer/Examiner	20/10/2024	28/10/2024	9
Full FYP Proposal	28/10/2024	14/11/2024	18
Chapter 1: Introduction	14/11/2024	21/11/2024	8
Chapter 2: Literature Review	21/11/2024	13/12/2024	23
Chapter 3: Requirement Analysis and Design	13/12/2024	5/1/2025	24
Phase 1.0 (Requirement Gathering): Interview with potential traffic warden	13/12/2024	20/11/2024	8
Phase 1.1 (Requirement Gathering): Distribute online survey via google form	15/12/2024	3/1/2025	20
Phase 2.0 (System Design) – Prototype and wireframes	13/12/2024	5/1/2025	24
FYP 1 Report Submission	5/1/2025	8/1/2025	4
FYP 1 Report Amendment	8/1/2025	10/1/2025	3
FYP 1 Final Report Submission after Amendment	10/1/2025	17/1/2025	8
Iteration 1	7/3/2025	14/4/2025	39
Chapter 4: Implementation	7/3/2025	9/4/2025	34
Phase 3.0 (Development): Initial coding and testing	7/3/2025	9/4/2025	34

Chapter 5: Testing	10/4/2025	14/4/2025	5
Phase 4.0 (Testing): User testing	10/4/2025	14/4/2025	5
Iteration 2	15/4/2025	20/5/2025	36
Chapter 4: Implementation	15/4/2025	16/5/2025	32
Phase 3.1 (Development): Enhanced coding and testing	15/4/2025	16/5/2025	32
Chapter 5: Testing	17/5/2025	20/5/2025	4
Phase 4.1 (Testing): User testing	17/5/2025	20/5/2025	4
Iteration 3	21/5/2025	25/6/2025	36
Chapter 4: Implementation	21/5/2025	21/6/2025	32
Phase 3.2 (Development): Final coding and testing	21/5/2025	21/6/2025	32
Chapter 5: Testing	22/6/2025	25/6/2025	4
Phase 4.2 (Testing): User testing	22/6/2025	25/6/2025	4
Chapter 6: Conclusion and future work	26/6/2025	27/6/2025	2
Phase 5.0 (Deployment): System deployment and demonstration	28/6/2025	30/6/2025	3
Phase 6.0 (Review): Maintenance	1/7/2025	4/7/2025	4
Final FYP report submission	1/7/2025	8/7/2025	8

Table 1. 1 Project Schedule

1.8 Expected Outcome

This project is expected to deliver a fully functional Portable Wireless Traffic Control System that operates offline and is suitable for managing traffic during partial road closures. The system will feature two synchronized LED traffic signals that communicate wirelessly using a controlled communication protocol to ensure seamless operation. Vehicle detection through infrared sensors will enable dynamic adjustments to traffic signal timings, optimizing traffic flow while maintaining safety. The inclusion of a default countdown system with a 2-second buffer delay during transitions ensures smooth and conflict-free signal changes.

A key component of the system is a mobile application designed to provide traffic wardens with real-time monitoring and control capabilities. The app will enable users to start and stop the system, activate "all-red" signals for emergencies, manually override signal operations, and adjust countdown durations. The intuitive interface ensures ease of use, empowering wardens to adapt to varying traffic conditions and address unexpected scenarios efficiently.

By combining automation, adaptability, and user-centric control, this project aims to improve road safety, optimize traffic management, and provide a practical solution for areas with limited resources. The system’s offline functionality and portability make it especially suited for rural or remote regions, where it offers a modern, reliable alternative to traditional manual traffic management methods.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

Effective traffic management plays a crucial role in maintaining road safety and ensuring the smooth flow of vehicles, particularly during partial road closures required for maintenance or emergencies. Traditional traffic control methods, such as manual signaling or fixed-schedule systems, are inefficient and risky, often resulting in delays, miscommunication, and heightened safety risks for road workers and users. These challenges are especially severe in rural areas, where limited infrastructure, poor connectivity, and scarce resources make advanced traffic solutions difficult to implement (Teja et al., 2024).

Recent advancements in traffic management systems, particularly those utilizing IoT technologies, have demonstrated the potential to address these challenges. Kumar and Patel (2023) highlight how real-time adaptive traffic systems powered by IoT can optimize traffic flow and improve safety through dynamic signal adjustments. Similarly, studies have shown the effectiveness of integrating microcontrollers, such as the ESP32, with sensors like IR detectors to automate traffic control and enhance efficiency (Teja et al., 2024). However, many of these solutions rely heavily on internet connectivity, making them unsuitable for offline deployment in rural or remote settings.

By synthesizing these insights, this chapter establishes the foundation for developing a Portable Wireless Traffic Control System for Partial Road Closures. The proposed system integrates synchronized traffic lights, adaptive control mechanisms, and a mobile application to deliver an efficient and user-friendly solution for traffic management. This review evaluates existing traffic management solutions to identify their strengths, limitations, and applicability to this project, providing a robust basis for addressing gaps in current traffic control systems.

2.2 Review of Similar Existing Systems

This section reviews existing traffic management systems, emphasizing their strengths, limitations, and applicability to the proposed solution. The systems analyzed include the IoT-Enabled Smart Traffic System for Public and Emergency Mobility in Smart City (Nagarjuna et al., 2020), the Smart Traffic Signal Management System (Manasa et al., 2023), and the Smart Traffic Management System for Priority Vehicle Clearance using IoT (Raj et al., 2022). These systems leverage advancements in IoT, automation, and dynamic traffic control, providing valuable insights for the development of the proposed Portable Wireless Traffic Control System for Partial Road Closures. While these systems address challenges such as congestion, emergency vehicle prioritization, and real-time monitoring, they also reveal limitations, such as reliance on internet connectivity, limited portability, and reduced adaptability in offline or rural scenarios. The analysis highlights how these gaps can be addressed by the proposed system, which integrates portable LED traffic lights, a controlled communication protocol, and a user-friendly mobile application to enhance efficiency, safety, and adaptability.

2.2.1 IoT-Enabled Smart Traffic System for Public and Emergency Mobility in Smart City (Nagarjuna et al., 2020)

The IoT-Enabled Smart Traffic System for Public and Emergency Mobility focuses on addressing urban traffic congestion and prioritizing emergency vehicles using IoT-based automation. The system integrates ESP32 microcontrollers, infrared (IR) sensors, RFID readers, and the Blynk App to achieve dynamic traffic signal management. This setup facilitates real-time adjustments to traffic lights and provides priority for emergency and stolen vehicles (Nagarjuna et al., 2020). Designed for smart city applications, the system aims to enhance traffic efficiency, reduce delays, and optimize road safety.

The system architecture is represented in a block diagram that outlines the flow of data between its components, as shown in Figure 2.1 below. The ESP32 microcontroller acts as the central unit, processing inputs from IR sensors, RFID readers, and other connected devices. These inputs include vehicle density data from the IR sensors and vehicle identification information from RFID tags. Outputs include the activation of traffic lights, seven-segment displays for timing, and LCDs for system status updates (Nagarjuna et al., 2020).

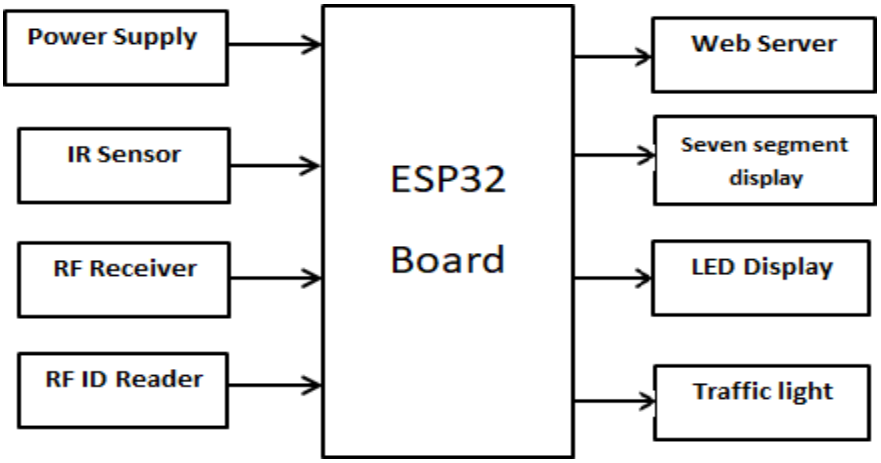

Figure 2. 1 Block diagram of the system

Communication within the system is facilitated through multiple protocols. For instance, the MQTT protocol is employed for real-time data exchange with the Blynk platform, while UART and SPI protocols manage interactions between the microcontroller and other peripherals. This structured data flow ensures seamless and efficient operation, enabling dynamic signal adjustments and real-time system updates (Nagarjuna et al., 2020).

The system's operational flow is depicted in a flowchart, as illustrated in Figure 2.2 below. It begins with system initialization, where the ESP32 establishes connections with sensors and the Blynk App. The system then enters a monitoring state, continuously detecting vehicle density on each road using IR sensors. If one road is detected to have higher density than others, the corresponding signal is turned green while others remain red. For emergency vehicles, RFID tags trigger immediate green signals for the relevant lane, regardless of traffic density (Nagarjuna et al., 2020).

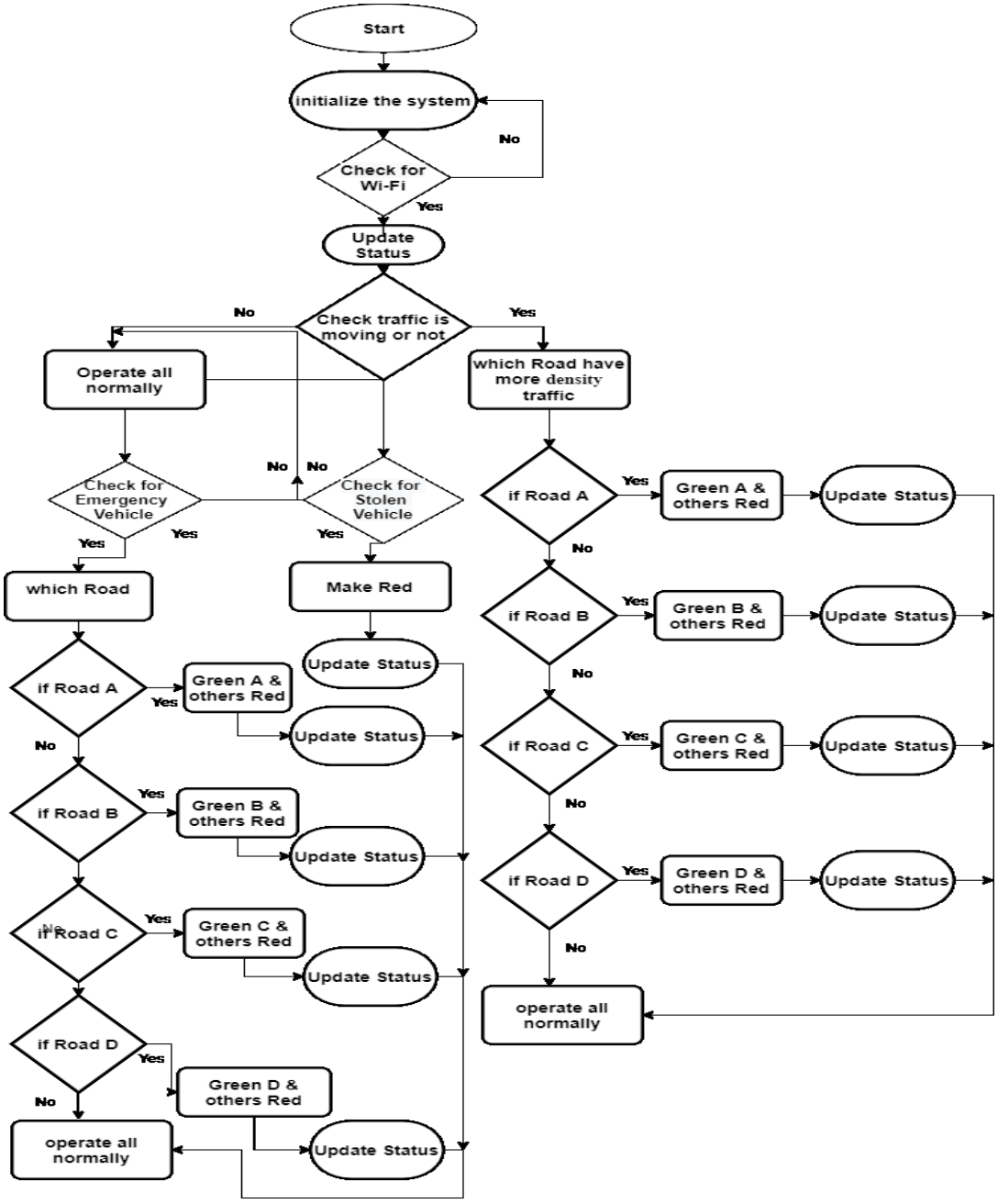

Figure 2. 2 Flowchart of the Model

Additionally, the system includes decision loops to manage connectivity and data updates. For instance, if the Wi-Fi connection is lost, the system retries to establish communication before resuming normal operations. These logical pathways ensure reliability in both routine traffic management and special scenarios, such as stolen vehicle detection or emergency prioritization (Nagarjuna et al., 2020).

The Blynk App provides the system's graphical user interface, enabling traffic managers to monitor and control signals remotely. As shown in Figure 2.3 below, the app allows users to view traffic density on each road, control signals manually, and observe automated adjustments for emergency vehicles. The app also offers virtual representations of signal states, providing real-time feedback for better traffic management (Nagarjuna et al., 2020).

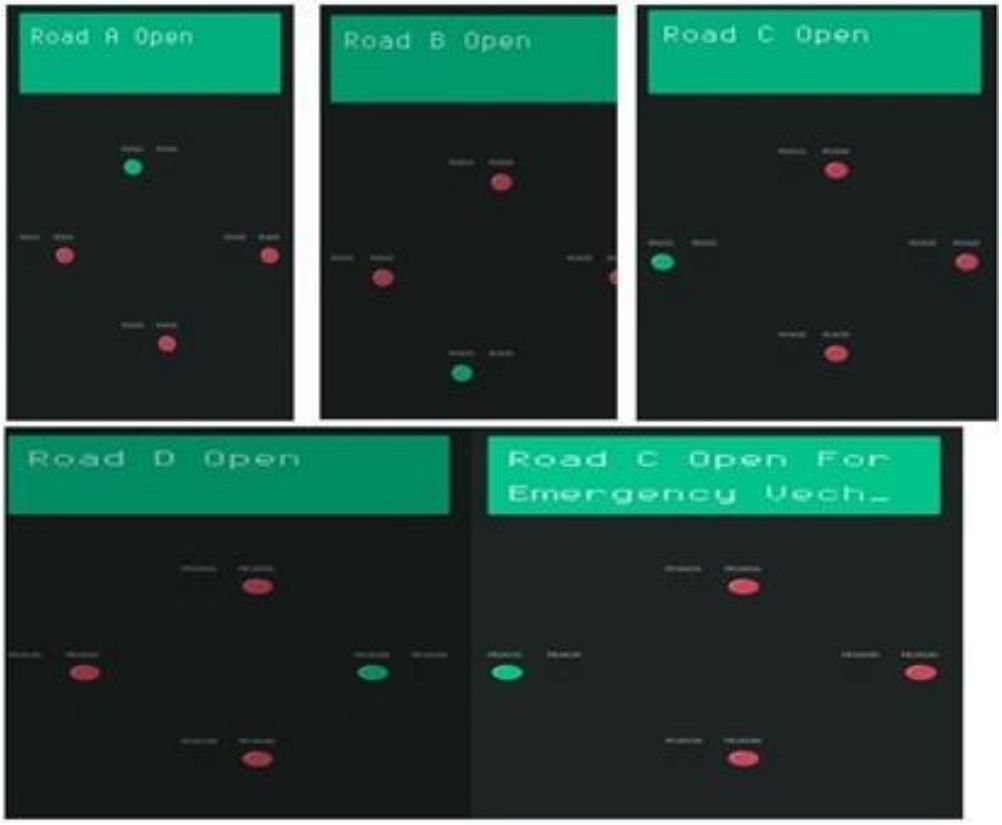

Figure 2. 3 Different output scenarios in the Blynk app

The app operates seamlessly with the ESP32, utilizing the Blynk Server for data storage and retrieval. This ensures scalability and efficient processing of IoT sensor data. Its compatibility with both Android and iOS devices makes it accessible for a wide range of users, further enhancing its practicality for smart city applications (Nagarjuna et al., 2020).

A prototype of the IoT-Enabled Smart Traffic System was developed to simulate a four-way intersection, as shown in Figure 2.4 below. The physical model features IR sensors placed on each road to detect vehicle density, RFID readers to identify vehicles, and LED traffic lights for signal control. This setup effectively demonstrates the system's ability to dynamically adjust traffic signals based on real-time inputs and prioritize emergency vehicles (Nagarjuna et al., 2020).

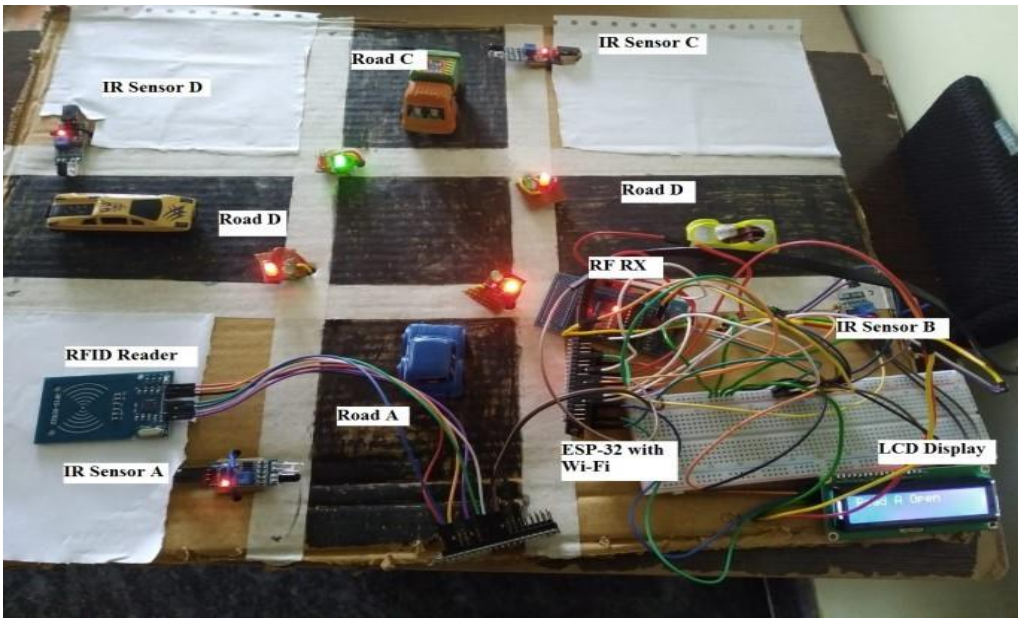

Figure 2. 4 Prototype model of the system proposed system.

The model highlights the integration of hardware components, including ESP32 microcontrollers, sensors, and display units, showcasing a practical implementation of the proposed architecture. By employing a realistic intersection layout, the prototype validates the system's functionality under simulated traffic conditions (Nagarjuna et al., 2020).

The IoT-Enabled Smart Traffic System demonstrates innovative approaches to traffic management through the integration of IoT and automation. The system's use of ESP32 microcontrollers and IR sensors aligns with the proposed Portable Wireless Traffic Control System, particularly in its focus on dynamic signal adjustments and real-time data processing. However, its reliance on internet connectivity and RFID technology for stolen vehicle detection reduces its applicability in offline or rural settings. Nonetheless, the modular design and IoT integration provide valuable insights into building efficient and adaptable traffic control solutions (Nagarjuna et al., 2020).

2.2.2 Smart Traffic Signal Management System (Manasa et al., 2023)

The Smart Traffic Signal Management System addresses urban traffic congestion and prioritizes emergency vehicles through IoT integration and dynamic traffic control. Using IR sensors for density detection and Arduino UNO as the microcontroller, this system ensures efficient traffic signal operation. The integration of the Blynk App enables real-time monitoring and manual control of traffic signals, further improving system adaptability and usability (Manasa et al., 2023). This system is particularly relevant to the proposed project due to its focus on density-based traffic control and IoT-enabled monitoring.

The block diagram of the system, as shown in Figure 2.5, illustrates the integration of key components. IR sensors are placed strategically to detect vehicle density, and the data is transmitted to the Arduino UNO for processing. The NodeMCU (ESP8266) connects the system to the Blynk App, enabling real-time control and monitoring. Outputs include traffic signal LEDs, which are dynamically adjusted based on input from the sensors (Manasa et al., 2023). This structure highlights the modularity of the system, allowing for easy integration of additional features such as GPS or image processing in the future.

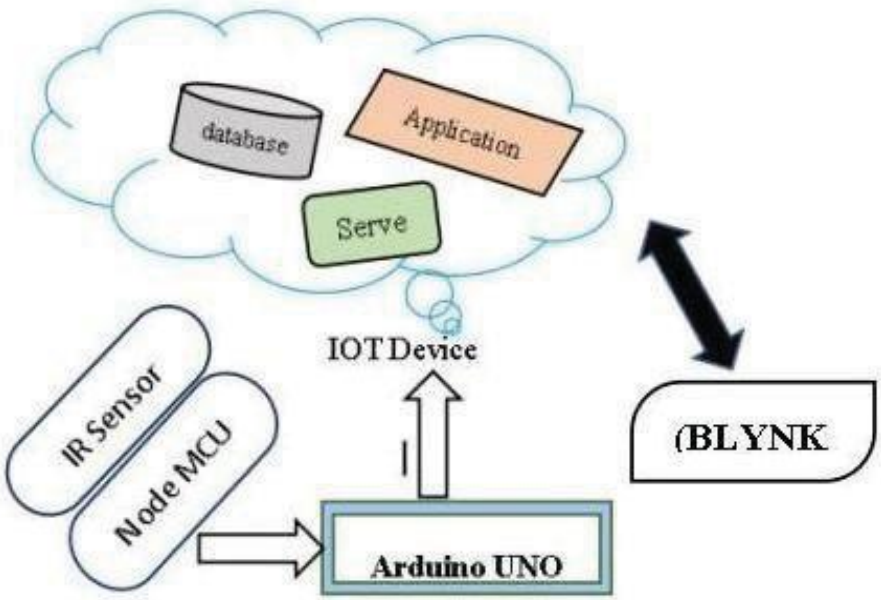

Figure 2. 5 Block diagram of the system.

Setting up the Blynk App involves creating a device template and linking it to the hardware. As shown in Figure 2.6, the setup includes generating a template ID, device name, and authentication token. These parameters are configured in the Arduino IDE, enabling communication between the microcontroller and the app. The app utilizes Wi-Fi provisioning, where the NodeMCU generates a dedicated network for the Blynk platform to connect and exchange data (Manasa et al., 2023).

Figure 2. 6 Preparing of Code.

Virtual pins, as shown in Figure 2.7, are used to establish communication pathways between the Blynk App and the hardware. These pins allow seamless data transfer and enable features such as manual signal control, which are essential for dynamic traffic management.

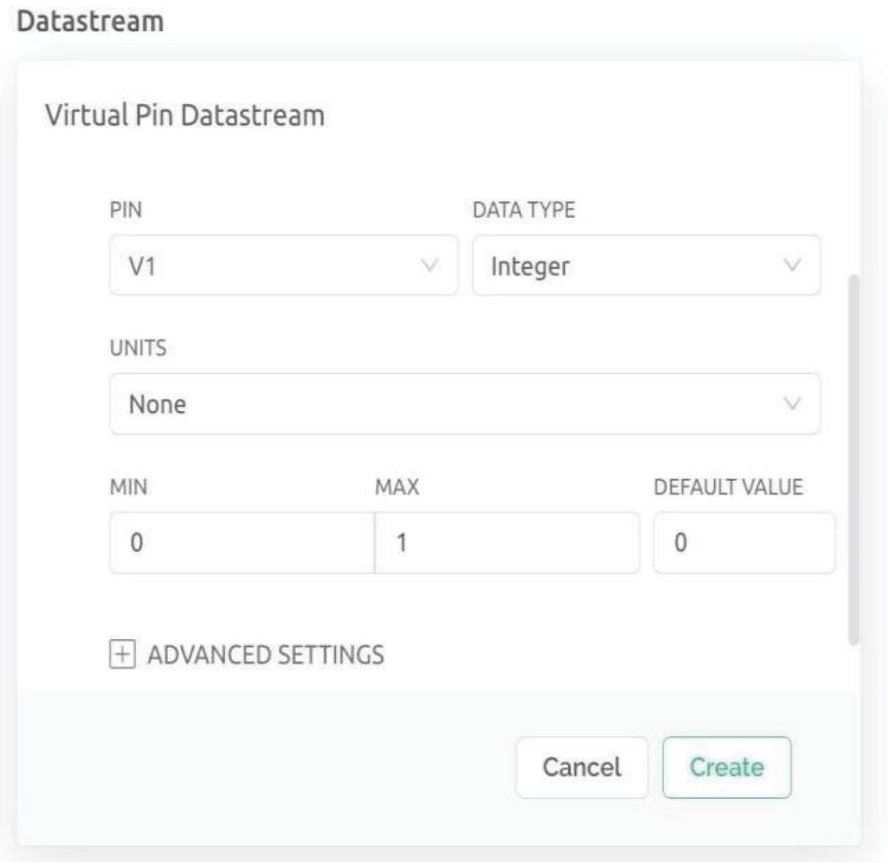

Figure 2. 7 Virtual pins creation for signal control.

Object detection is a core functionality of this system, demonstrated in Figure 2.8. The IR sensors operate by emitting infrared light from a transmitter, which is reflected back to a receiver when an object (vehicle) is present. This interaction triggers the sensor to send data to the Arduino UNO, which processes the input and adjusts the traffic signals accordingly (Manasa et al., 2023). The system prioritizes high-density lanes by assigning green signals, ensuring efficient traffic flow.

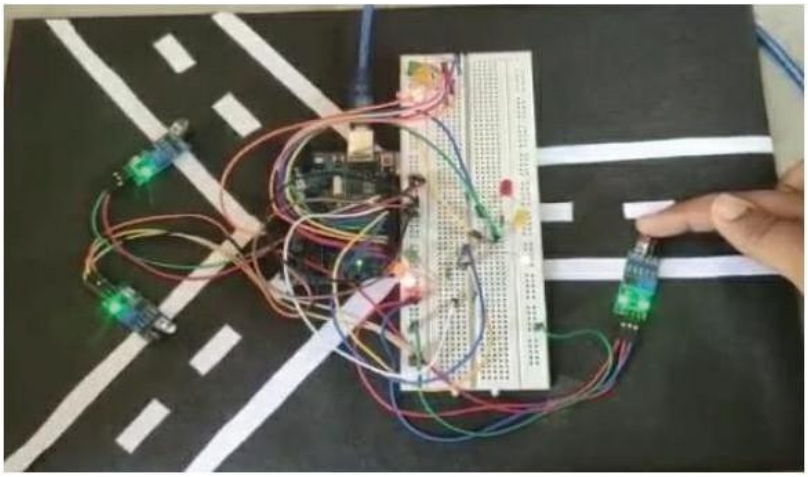

Figure 2. 8 Object Detection by IR Sensor.

The Blynk App's interface, as shown in Figure 2.9, provides users with a simple and intuitive platform to monitor and control traffic signals. Using on-screen buttons, users can toggle signals for specific lanes, ensuring smooth passage for emergency vehicles. The app's real-time feedback and control features enhance the system's adaptability, making it a robust solution for dynamic traffic management (Manasa et al., 2023).

Figure 2. 9Blynk App displaying signals.

The Smart Traffic Signal Management System demonstrates an effective approach to addressing traffic congestion and prioritizing emergency vehicles through IoT and automation. Its use of IR sensors for density detection, Arduino for signal processing, and the Blynk App for real-time control aligns closely with the goals of the proposed system. However, the reliance on internet connectivity limits its applicability in offline scenarios, highlighting a key difference from the proposed Portable Wireless Traffic Control System. Despite this, the modular design and IoT integration offer valuable insights for developing efficient and adaptable traffic management solutions (Manasa et al., 2023).

2.2.3 Smart Traffic Management System for Priority Vehicle Clearance using IoT (Raj et al., 2022)

The Smart Traffic Management System for Priority Vehicle Clearance focuses on addressing urban traffic congestion and ensuring the smooth flow of vehicles by dynamically managing traffic signals based on vehicle density. This system utilizes IR sensors for density detection, an 8051 microcontroller for traffic signal management, and the Blynk App for real-time monitoring and manual control. Additionally, the system integrates RF transmitters and receivers to prioritize vehicles like ambulances and fire trucks, enhancing emergency response times (Raj et al., 2022). The relevance of this system lies in its focus on dynamic signal control and emergency vehicle prioritization, aligning closely with the objectives of the proposed project.

The block diagram, as depicted in Figure 2.10, provides an overview of the system's architecture. IR sensors detect vehicle density and transmit data to the 8051 microcontroller, which processes the information to adjust signal timing. The system also integrates an RF transmitter and receiver for priority vehicle detection. Outputs include traffic light LEDs, which change based on vehicle density and priority vehicle presence (Raj et al., 2022). The modular design ensures scalability and ease of implementation, making it suitable for both urban and rural settings.

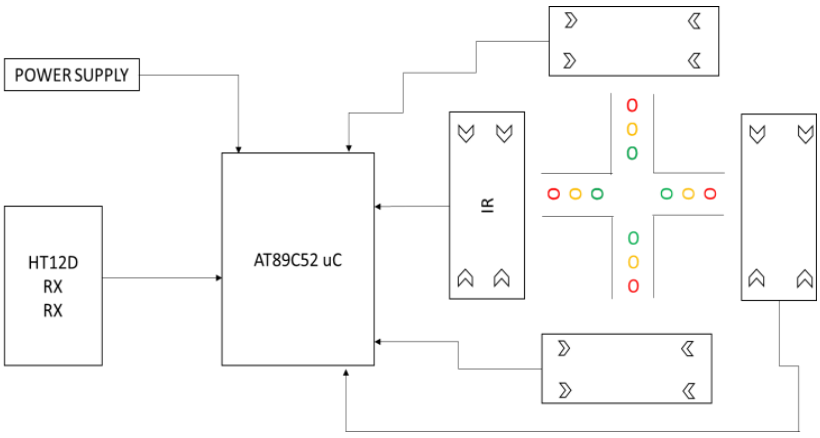

Figure 2. 10 Block diagram of the system.

The flow diagram of the system, shown in Figure 2.11, illustrates the operational logic. The process begins with IR sensors detecting vehicle density in each lane. The data is sent to the microcontroller, which evaluates the density and assigns green signals to lanes with higher traffic. Simultaneously, RF signals from priority vehicles override the default signal pattern, ensuring immediate green light clearance for emergency vehicles. This flow ensures efficient traffic management and minimizes waiting times, particularly for priority vehicles (Raj et al., 2022).

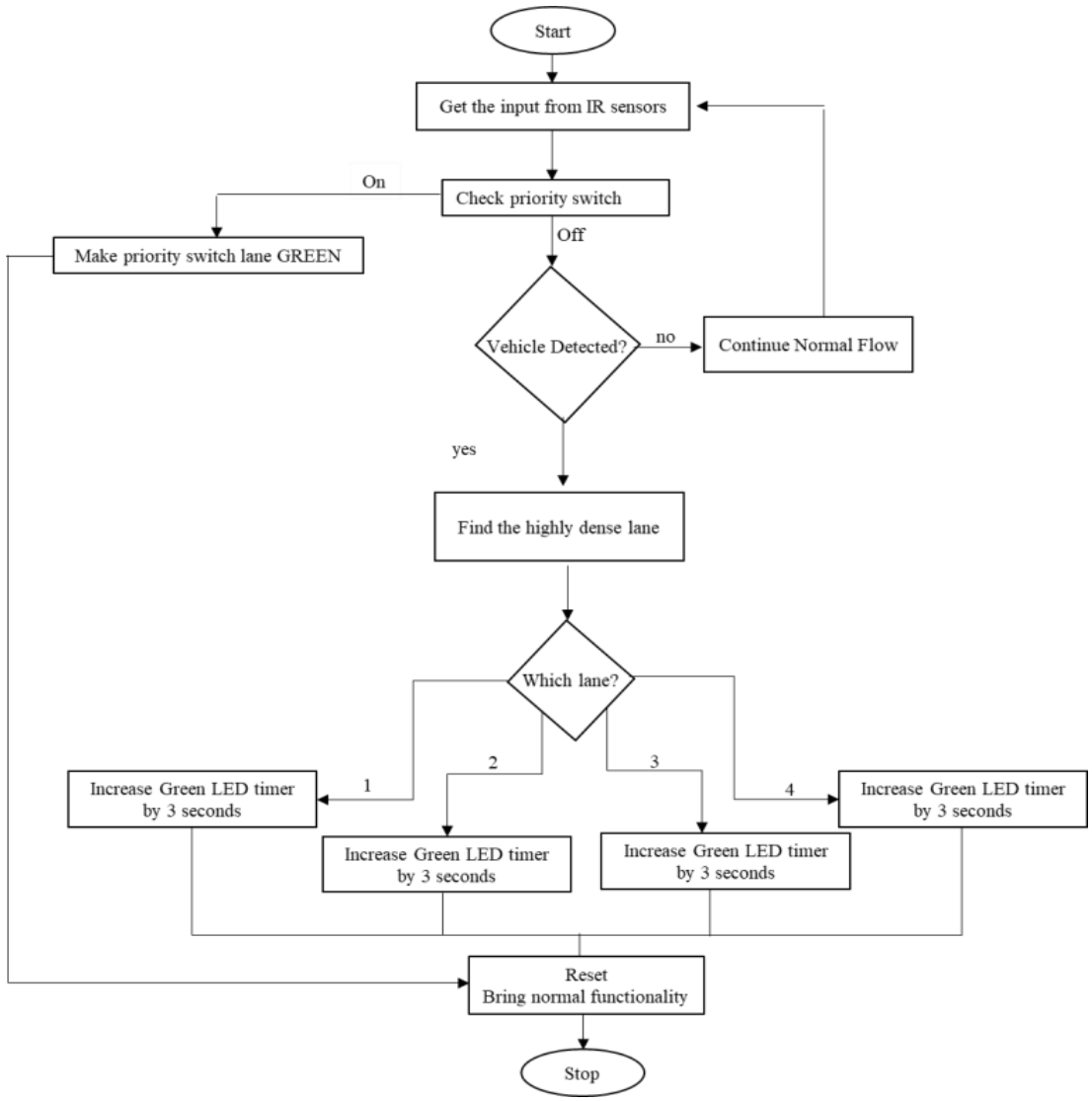

Figure 2. 11Flow Diagram of the system.

The Blynk App serves as the system's user interface, providing remote monitoring and control capabilities. As shown in Figure 2.12, the app enables users to manually control traffic signals, monitor real-time traffic density, and prioritize specific lanes for emergency vehicle clearance. The app interface is user-friendly, allowing traffic wardens and emergency vehicle drivers to efficiently manage traffic operations with minimal training (Raj et al., 2022).

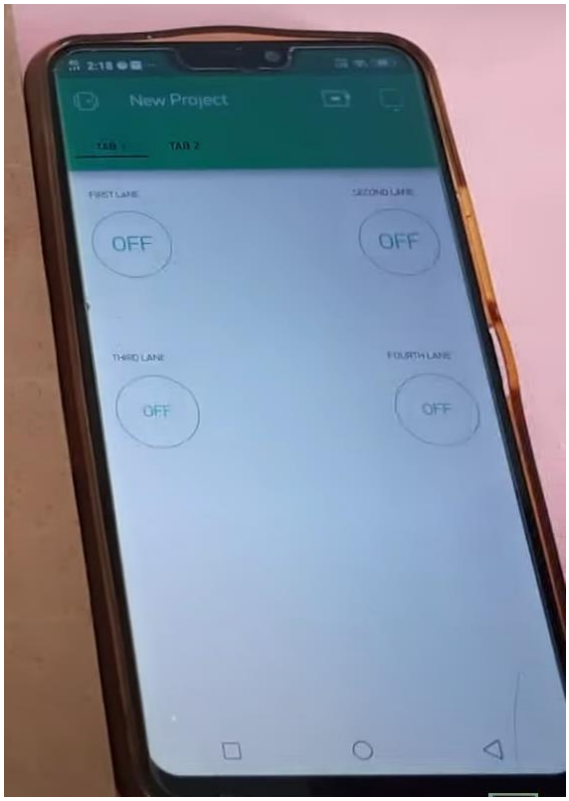

Figure 2. 12 Blynk App interface.

The hardware setup of the system, depicted in Figure 2.13, consolidates all components into a functional prototype. It includes IR sensors for density detection, an 8051 microcontroller for processing, and LEDs for traffic light representation. The RF transmitter and receiver facilitate priority vehicle detection, ensuring timely green light allocation. The entire setup is powered by a regulated power supply, ensuring reliable operation under varying conditions (Raj et al., 2022). This comprehensive setup effectively demonstrates the system’s capability to handle real-world traffic scenarios.

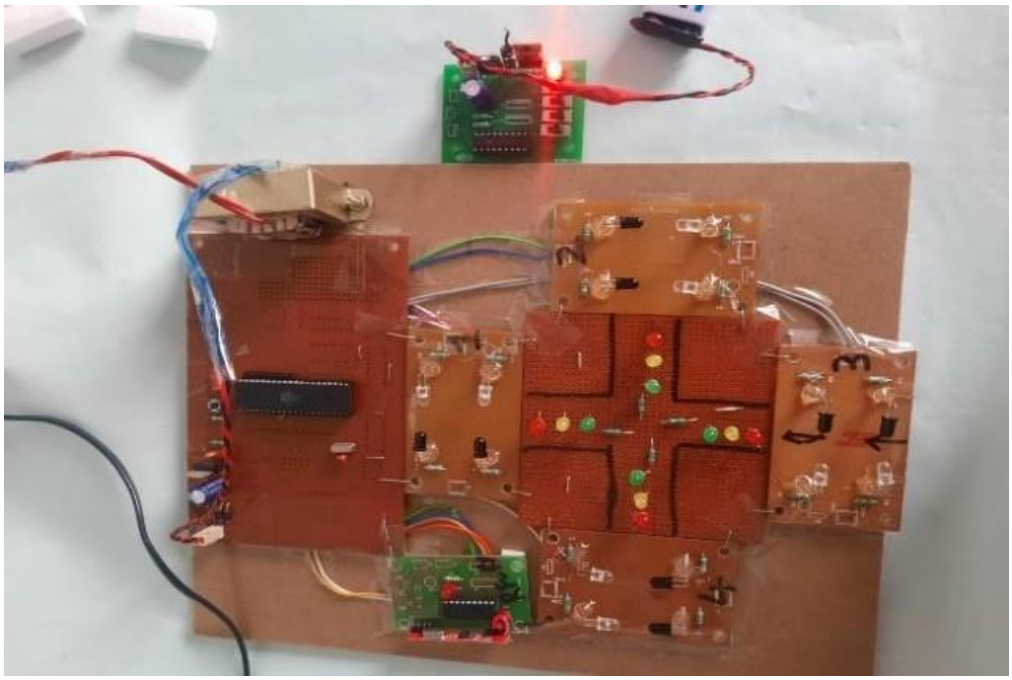

Figure 2. 13Hardware setup of the system.

The Smart Traffic Management System for Priority Vehicle Clearance demonstrates a practical approach to addressing traffic congestion and prioritizing emergency vehicles through IoT and automation. Its use of IR sensors, microcontrollers, and the Blynk App aligns closely with the goals of the proposed Portable Wireless Traffic Control System. However, its reliance on an 8051 microcontroller limits its scalability compared to more modern microcontrollers like the ESP32. Additionally, the system's dependence on internet connectivity reduces its applicability in offline or rural scenarios. Nonetheless, its modular design and focus on real-time adaptability offer valuable insights for developing efficient and scalable traffic management solutions (Raj et al., 2022)

2.3 Comparison of Existing Systems with the Proposed System

Table 2. 1 Comparison between existing systems and proposed systems.

Functionality and Features	IoT-Enabled Smart Traffic System for Public and Emergency Mobility (Nagarjuna et al., 2020)	Smart Traffic Signal Management System (Manasa et al., 2023)	Smart Traffic Management System for Priority Vehicle Clearance using IoT (Raj et al., 2022)	Proposed Portable Wireless Traffic Control System
Microcontroller	ESP32	Arduino UNO	8051 Microcontroller	ESP32 with Wifi/Bluetooth
Sensors Used	IR Sensors for density detection; RFID for stolen vehicle detection	IR Sensors for density detection	IR Sensors and RF for priority vehicle detection	IR Sensors for vehicle presence detection
Communication Protocols	MQTT, UART, and SPI for real-time communication	Wi-Fi for connecting to the Blynk App	RF Communication for priority vehicle alerts	Controlled protocol for offline, reliable synchronization

Mobile App Features	Blynk App with real-time signal control and monitoring	Blynk App for density-based signal adjustments	Blynk App for manual control of priority vehicle signals	Custom app with advanced controls: signal overrides, countdown edits, and status updates
Traffic Signal Control	Real-time adjustments based on traffic density and emergency vehicle detection	Signal priority for high-density lanes	Emergency vehicle prioritization with manual overrides	Dynamic traffic management with synchronized lights and adaptable countdowns
Power Supply	Fixed power source	Modular but dependent on grid power	Fixed power with limited portability	Portable battery packs
Portability	Low portability due to reliance on fixed infrastructure and RFID	Moderate portability with modular IoT integration	Moderate portability; limited by fixed installations	High portability: lightweight hardware, easy deployment
Scalability and Adaptability	Limited scalability due to reliance on RFID and online features	Scalable but lacks adaptability for emergency scenarios	High scalability for urban setups but unsuitable for offline rural areas	Highly scalable and adaptable for rural and urban setups, with offline and portable operation

2.4 Summary

This chapter reviewed existing traffic management systems, highlighting their strengths, limitations, and relevance to the proposed Portable Wireless Traffic Control System for Partial Road Closures. The systems analyzed include the IoT-Enabled Smart Traffic System for Public and Emergency Mobility, the Smart Traffic Signal Management System, and the Smart Traffic Management System for Priority Vehicle Clearance using IoT. These systems demonstrated advancements in automation, IoT-based monitoring, and adaptive traffic signal control.

However, significant limitations were identified, such as reliance on internet connectivity, limited portability, and restricted adaptability in rural or offline scenarios. While these systems provide valuable insights into dynamic traffic management and emergency vehicle prioritization, they fall short in addressing the unique challenges posed by temporary road closures, particularly in resource-limited environments.

The proposed system builds upon the strengths of these solutions by integrating portable LED traffic lights, a controlled communication protocol, and a user-friendly mobile application. By addressing the identified gaps, the proposed solution aims to enhance traffic safety, adaptability, and user experience during partial road closures, particularly in rural and offline settings.

CHAPTER 3: REQUIREMENT ANALYSIS AND DESIGN

3.1 Introduction

The Portable Wireless Traffic Control System for Partial Road Closures is designed to address the unique challenges associated with managing traffic during temporary road closures. These challenges include ensuring safety for road users and workers, minimizing congestion, and providing a reliable, portable system that can adapt to varying scenarios. The system emphasizes offline functionality, making it suitable for deployment in rural or remote areas where internet connectivity is unavailable.

This chapter focuses on the requirement analysis and design phases, which are critical for defining the system's functionality and architecture. During the requirement analysis phase, inputs from traffic wardens and road users were gathered through surveys and interviews to identify the system's key features. These include real-time traffic signal synchronization, manual override capabilities via a mobile application, and adjustable countdown timers to enhance adaptability. The analysis also identified the hardware and software components necessary for implementing these features. Key hardware components include ESP32 microcontrollers for wireless communication, infrared (IR) sensors for vehicle detection, and portable LED traffic lights for efficient signaling. The software components include a custom communication protocol to enable offline synchronization and a mobile application for real-time monitoring and control, developed using Flutter for cross-platform compatibility.

Building on these requirements, the design phase provides a detailed blueprint of the system, incorporating diagrams and designs such as the system architecture, flowchart, block diagram, context diagram, sequence diagrams and user interface design of the mobile application. These elements demonstrate the seamless integration of ESP32 microcontrollers, IR sensors, and the controlled communication protocol, enabling efficient signal synchronization and real-time adaptability. By addressing the limitations of existing systems, this chapter outlines a robust, user-friendly design tailored to manage traffic during partial road closures effectively.

3.2 Requirement Analysis

The requirement analysis phase serves as the foundation for the design and implementation of the Portable Wireless Traffic Control System for Partial Road Closures. This phase focuses on identifying the key system requirements through surveys, interviews, and a review of relevant tools and technologies. The analysis ensures that the proposed system meets the functional, operational, and user-specific needs effectively.

3.2.1 User Requirements

The user requirements were identified through surveys (Google Form) and an interview was conducted with a traffic warden and road users involved in traffic management. The following key features were derived from this analysis:

1. **Real-Time Signal Synchronization**: Traffic lights must be synchronized to prevent conflicts and ensure smooth traffic flow.
2. **Manual Overrides**: Traffic wardens require the ability to override signal operations in emergencies or unforeseen scenarios via a mobile application.
3. **Adjustable Countdown Timers**: The system must allow customization of countdown durations based on traffic conditions and distances between signals.
4. **Portability**: The system must be easy to deploy and remove, making it suitable for temporary road closures in rural or urban settings.
5. **Offline Functionality**: The system must operate without internet connectivity, ensuring reliability in resource-limited environments.

3.2.2 Hardware Specifications

Component	Description	Reason for Selection
ESP32 Microcontrollers	Microcontrollers with built-in Wi-Fi and Bluetooth capabilities for wireless communication.	Reliable for real-time communication and synchronization between traffic signals.
Infrared (IR) Sensors	Sensors used to detect vehicles approaching the traffic signals.	Provides accurate real-time traffic density data.
Portable LED Traffic Lights	Energy-efficient lights for signalling traffic during temporary road closures.	Highly visible, durable, and suitable for various environmental conditions.
Battery Packs	Rechargeable lithium-ion batteries to power the microcontrollers, sensors, and lights.	Ensures system portability and continuous operation in areas without grid power.
Voltage Regulators	Devices to stabilize the power supply for sensitive components like the ESP32 and sensors.	Prevents power fluctuations, ensuring reliable system performance.
Breadboard	Reusable platform for assembling and testing circuits without soldering.	Allows for easy prototyping and modification of circuit designs.
Small LCD Screens	Screens to display countdown timers.	Enhances system feedback for debugging and user monitoring.
Jumper Wires	Wires used to connect components on the breadboard.	Ensures reliable connections between sensors, microcontrollers, and LEDs.

Table 3. 1 List of Hardware Specifications

3.2.3 Software Specifications

Component	Description	Reason for Selection
Controlled Communication Protocol	Protocol designed to synchronize signals offline between ESP32 microcontrollers.	Ensures reliable communication without requiring internet connectivity.
Mobile Application	Developed using Flutter for cross-platform compatibility.	Provides real-time monitoring, manual overrides, and countdown adjustments for traffic wardens.
Arduino IDE	Integrated Development Environment for coding and uploading firmware to ESP32.	Simplifies programming and debugging of ESP32 microcontrollers.
Visual Studio Code	Advanced IDE for coding, debugging, and managing project files.	Enhances coding efficiency and supports custom configurations.
SQLite Database	Lightweight database embedded in the mobile app.	Logs traffic events, user actions, and system performance for future reference.
Programming Languages	Primarily C/C++ for ESP32 microcontroller programming.	Well-suited for low-level hardware control and real-time processing.

Table 3. 2 List of Software Specifications

3.2.4 Functional Requirements

The functional requirements ensure that the system delivers its intended performance:

1. The system must detect vehicles via IR sensors and adjust traffic signals dynamically.
2. Countdown timers must initiate during the last 10 seconds of green-light transitions, with a 2-second buffer delay between red-to-green transitions.
3. The system must synchronize traffic lights using a master-slave configuration, with real-time communication via the controlled protocol.
4. Manual overrides must take precedence over automatic operations when activated.

3.2.5 Non - Functional Requirements

The non-functional requirements ensure reliability, usability, and scalability:

1. **Reliability**: The system must maintain consistent operation, even in harsh weather or resource-limited areas.
2. **Usability**: The mobile app interface must be intuitive, requiring minimal training for traffic wardens.
3. **Scalability**: The system should support future integration with additional features, such as density-based prioritization or solar-powered operation.
4. **Portability**: All components must be lightweight and easy to set up or dismantle.

.

3.3 System Design

3.3.1 System Architecture Diagram

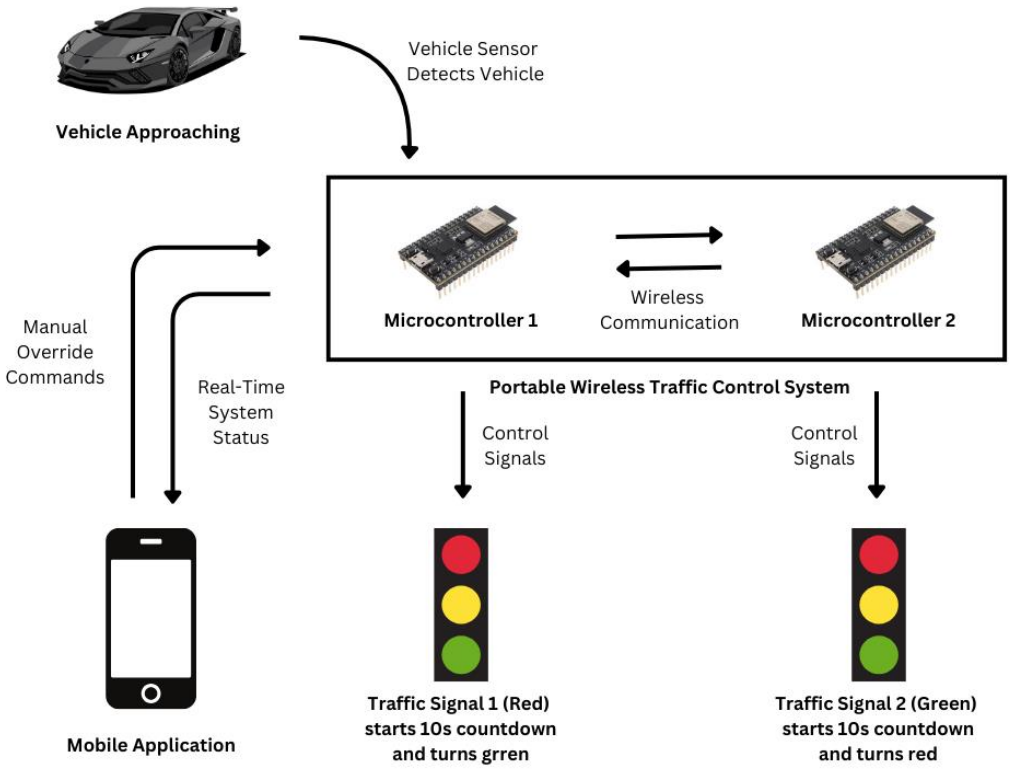

Figure 3. 1 Overall System Architecture of the whole system

The System Architecture Diagram, as shown in Figure 3.1, illustrates the interaction between the hardware and software components of the proposed system. At the core of the system are two ESP32 microcontrollers configured in a master-slave architecture, with MCU 1 (Master) managing system synchronization and processing commands from the mobile application. The mobile app enables traffic wardens to send manual override commands, monitor real-time system status, and adjust countdown timers. Infrared sensors detect approaching vehicles, sending signals to the master microcontroller, which determines signal transitions and synchronizes operations with MCU 2 (Slave) through a controlled communication protocol. The output is relayed to traffic signals, which display the appropriate red or green light states, including countdowns for transitions. This architecture ensures seamless integration of portable hardware components with an offline and user-friendly traffic management system.

3.3.2 Flowchart Diagram

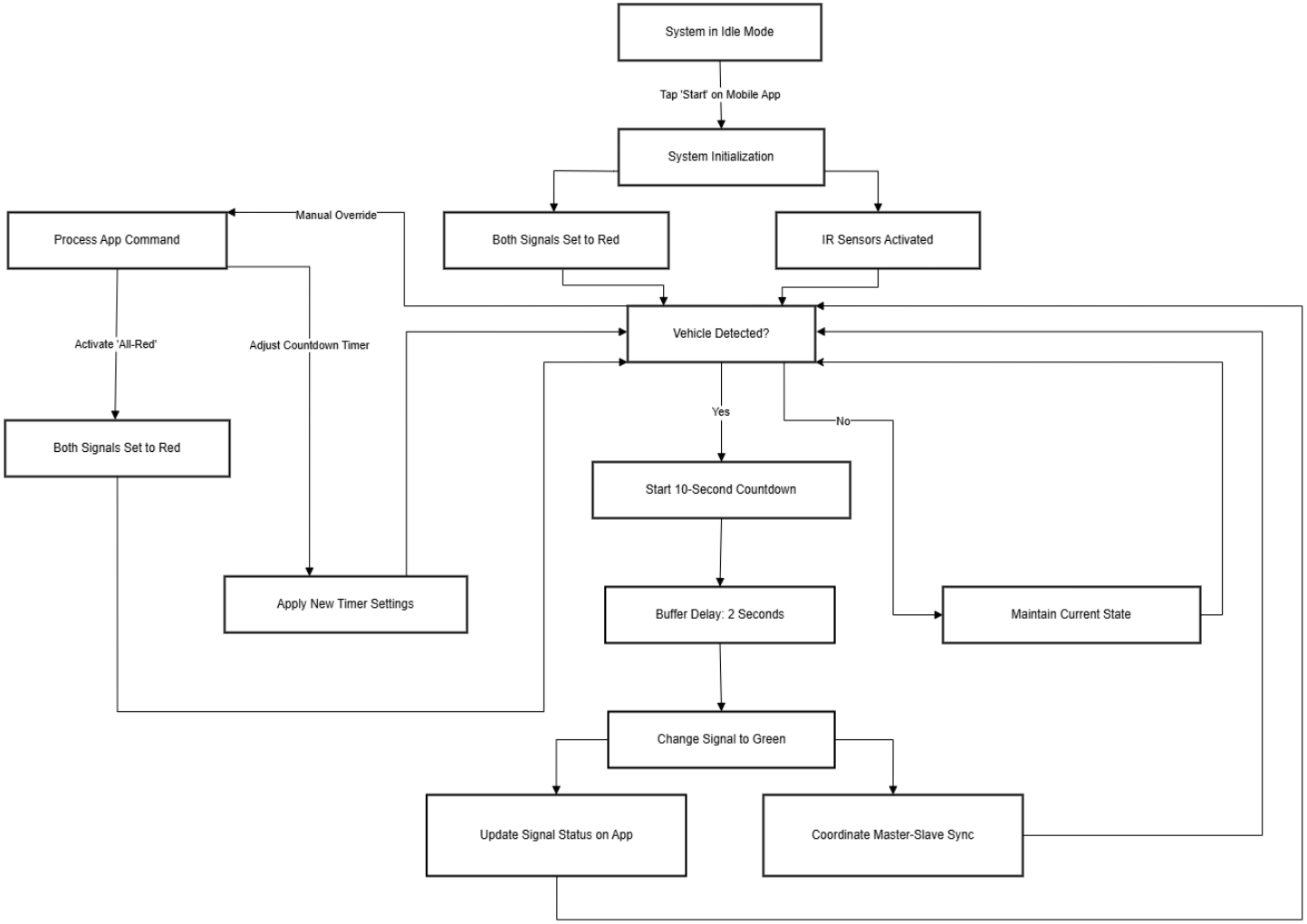

Figure 3. 2 Flowchart of the whole system

The flowchart, as shown in Figure 3.2, demonstrates the step-by-step operational flow of the Portable Wireless Traffic Control System for Partial Road Closures. It highlights how the system manages vehicle detection, signal transitions, and manual overrides using a master-slave architecture.

When a vehicle is detected by the infrared sensors, the system begins a 10-second countdown for the corresponding signal to turn green. A 2-second buffer delay ensures smooth and safe transitions before the signal changes. The master ESP32 coordinates the state of both signals, ensuring one is green while the other remains red. During this time, the system continuously updates the mobile application to reflect the current signal statuses.

If no vehicles are detected, the system remains in its current state, maintaining both traffic signals as red. The sensors continue monitoring for vehicle presence in each lane. This ensures that the system avoids unnecessary signal changes, conserving energy and maintaining traffic flow efficiency. The system seamlessly returns to its normal routine once a vehicle is detected, resuming countdown and transition logic.

Manual overrides initiated via the mobile application allow traffic wardens to intervene when necessary. They can activate an "All-Red" mode to halt traffic for construction vehicles or emergencies. Additionally, the app provides functionality to adjust countdown timers dynamically, applying new settings in real time. These overrides take precedence over the default logic, ensuring the system can adapt to special circumstances efficiently.

3.3.3 Block Diagram

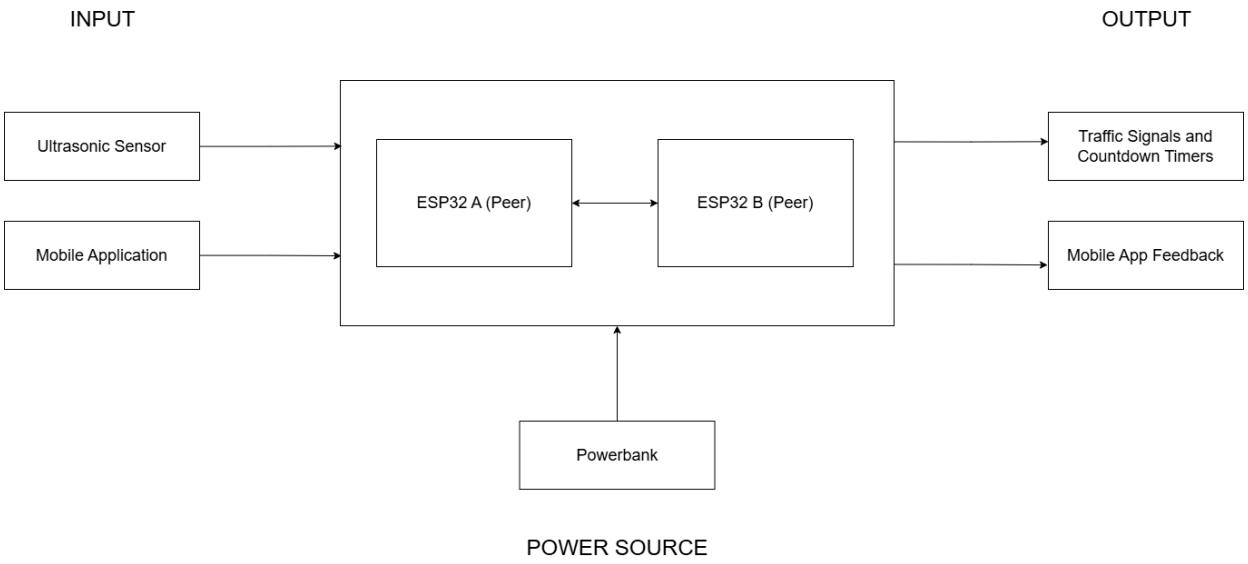

Figure 3. 3 Block Diagram of the whole system

The Block Diagram, as shown in Figure 3.3, illustrates the interaction between the core components of the Portable Wireless Traffic Control System for Partial Road Closures. Input components include Infrared Sensors, which detect vehicle presence and send data to the ESP32 Master, and the Mobile Application, which allows traffic wardens to issue manual overrides, adjust countdown timers, and monitor system statuses. The ESP32 Master processes these inputs and communicates with the ESP32 Slave using a controlled communication protocol to synchronize signal operations.

Output components include LED Traffic Lights and their countdown timers, with the Master controlling Traffic Signal 1 and the Slave managing Traffic Signal 2. Real-time updates are sent to the Mobile Application for user feedback and control. The system is powered by a Battery Pack connected to a Voltage Regulator, ensuring a stable power supply for all components. This modular design supports the system's portability, offline functionality, and ability to manage traffic effectively.

3.3.4 Context Diagram

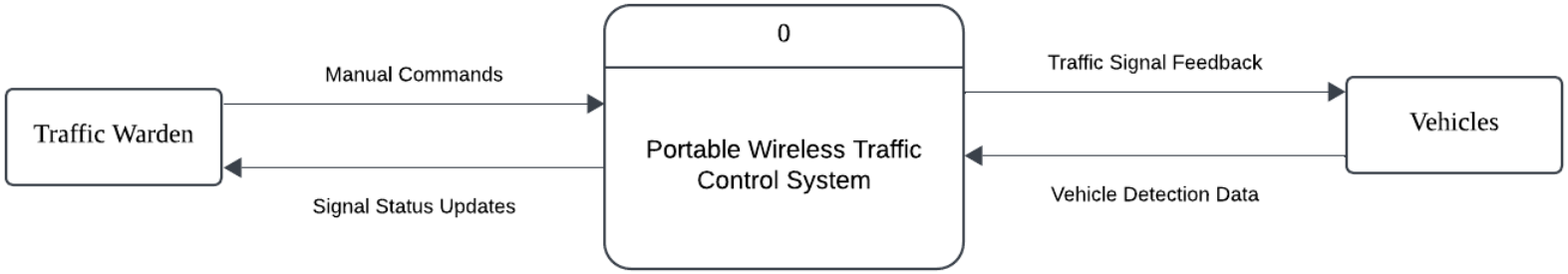

Figure 3. 4 Block Diagram of the whole system

The Context Diagram, shown in Figure 3.4, provides an overview of the interactions between the external entities and the Portable Wireless Traffic Control System. It highlights how the Traffic Warden and Vehicles communicate with the system, ensuring smooth traffic management and real-time control.

The Traffic Warden interacts with the system via the Mobile Application, sending manual commands to adjust traffic signals, as well as receiving real-time traffic status updates. This allows the warden to monitor the state of the traffic signals and adjust as needed. The Vehicles, detected by Infrared Sensors, send vehicle detection data to the system. This data triggers the appropriate traffic signal changes, allowing vehicles to pass while maintaining synchronization between the two signals.

This context diagram highlights the simplicity and effectiveness of the system, with clear inputs from both external entities and outputs to the traffic signals. It demonstrates the system's focus on offline functionality, using the mobile app and sensors to control traffic signals in real-time without the need for cloud integration.

3.3.5 Sequence Diagram

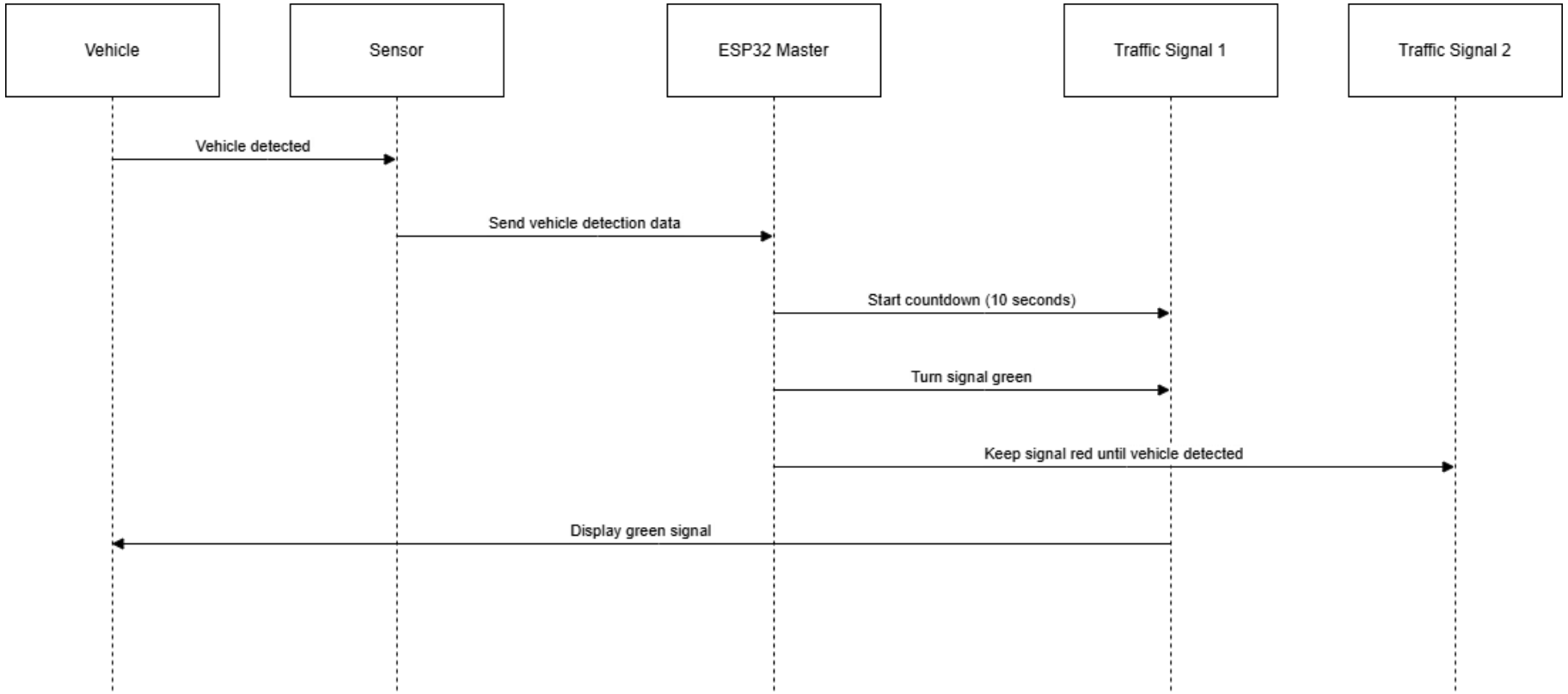

Figure 3. 5 Sequence Diagram during Vehicle Detection and Signal Change

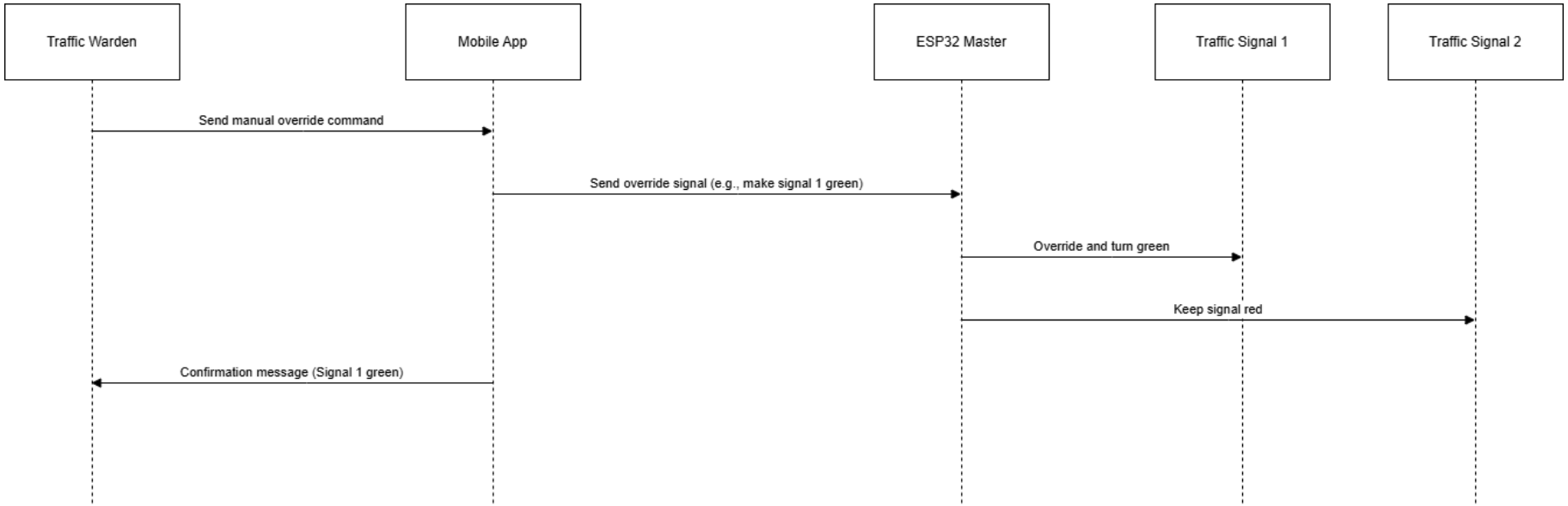

Figure 3. 6 Sequence Diagram during Manual Override via Mobile App

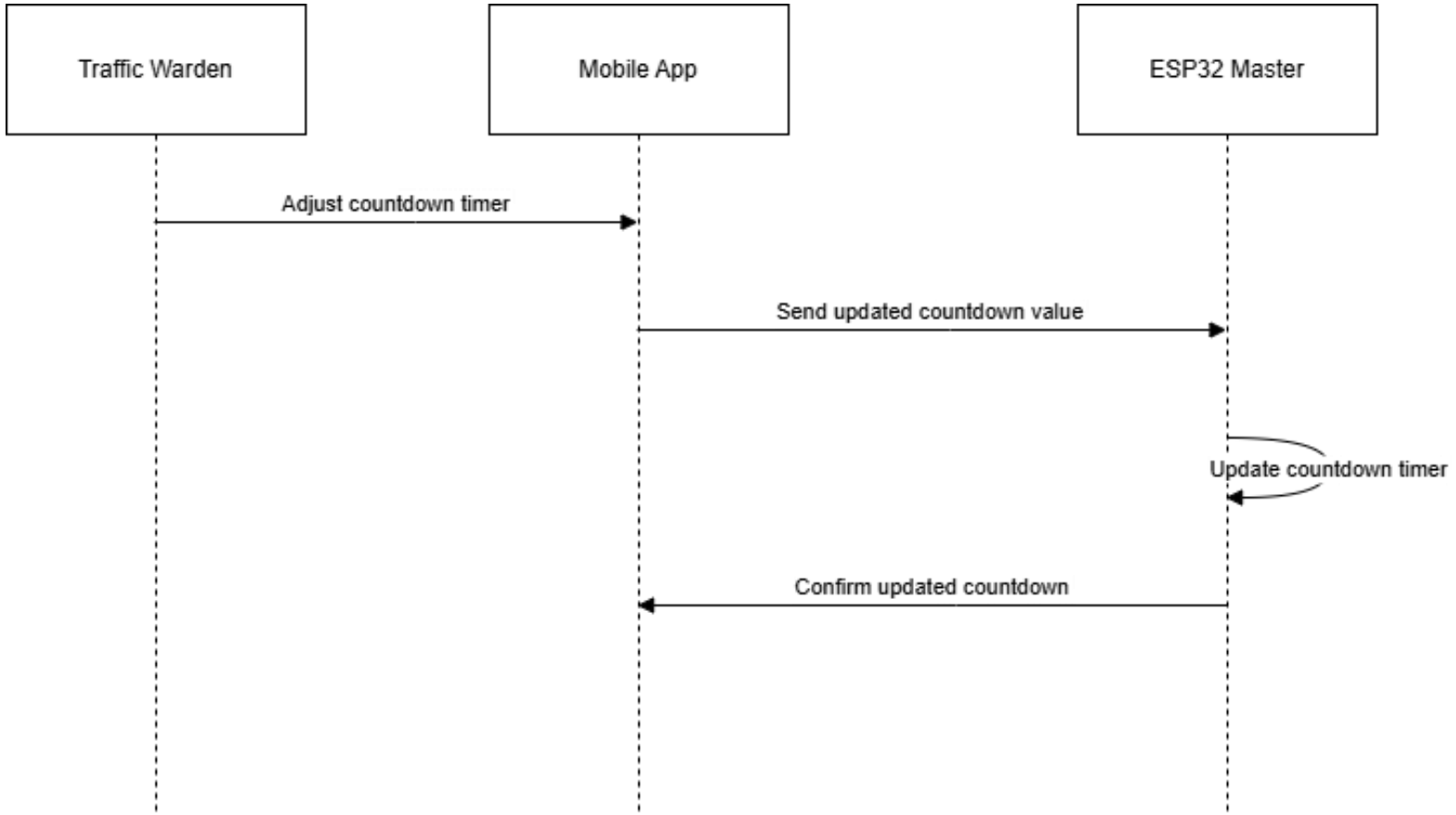

Figure 3. 7 Sequence Diagram during Countdown Adjustment by Traffic Warden

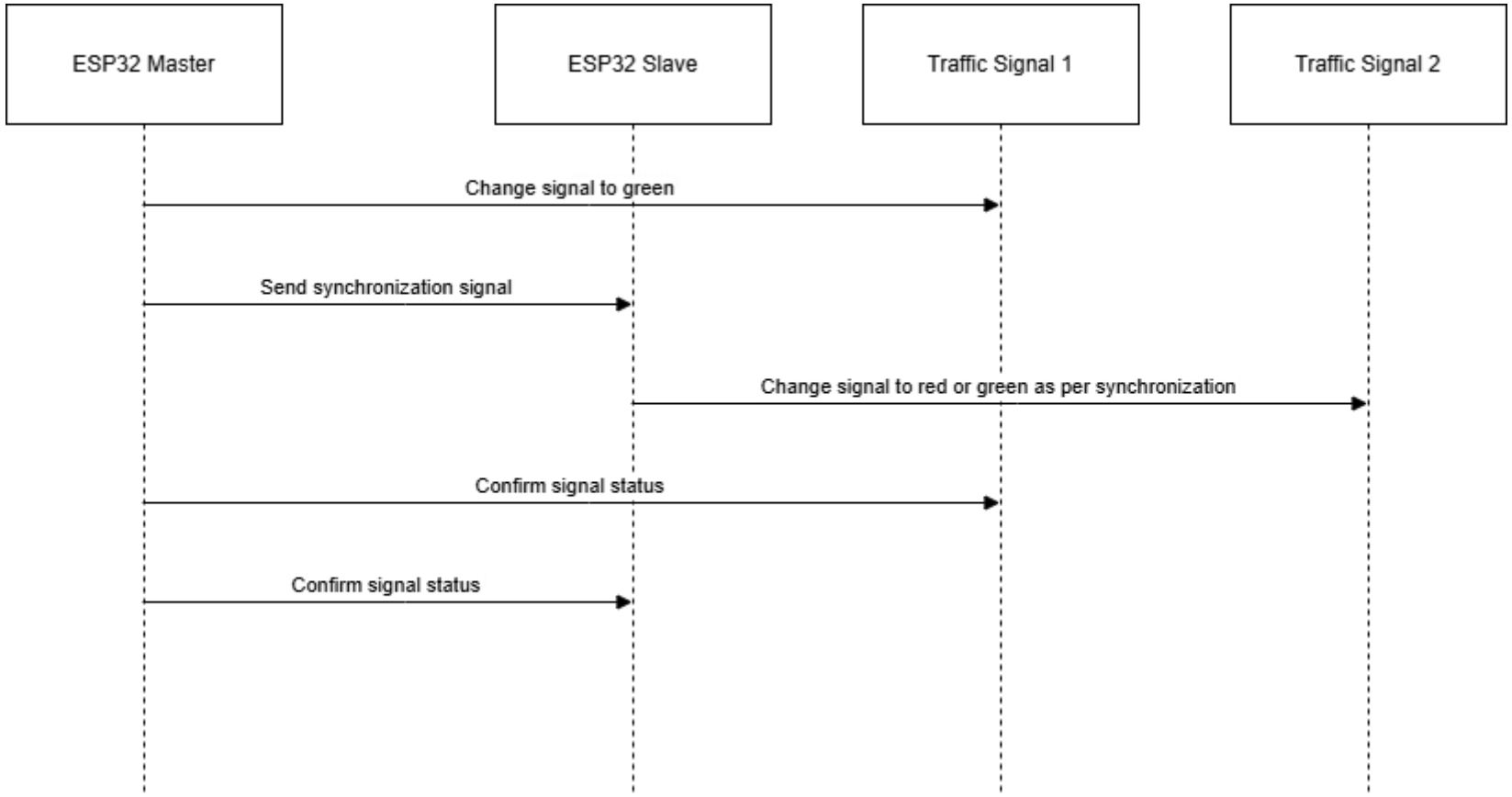

Figure 3. 8 Sequence Diagram for Signal Synchronization (Master-Slave Communication)

The sequence diagrams presented in Figures 3.5 to 3.8 illustrate the key events and responses in the system. They show how components like the Vehicle, Sensor, ESP32 Master, ESP32 Slave, and Traffic Signals communicate and function in scenarios such as vehicle detection and signal change, manual countdown adjustment, and signal synchronization between the master and slave units.

The first sequence diagram (Figure 3.5) focuses on the process when a vehicle is detected by the infrared sensor. Upon detection, the sensor sends vehicle detection data to the ESP32 Master, which triggers a 10-second countdown for Traffic Signal 1 to turn green. Meanwhile, Traffic Signal 2 stays red until another vehicle is detected. This diagram effectively demonstrates how the system responds in real-time to vehicle presence and manages the traffic signal changes accordingly.

The second sequence diagram (Figure 3.6) depicts how the Traffic Warden can manually adjust the countdown timer using the Mobile App. The Mobile App sends the updated countdown value to the ESP32 Master, which then updates the countdown timer. Afterward, the Mobile App receives confirmation that the new countdown has been applied. This sequence diagram illustrates the manual control feature, ensuring flexibility in traffic management, especially in situations that require dynamic signal timing adjustments.

The third sequence diagram (Figure 3.7) shows how the ESP32 Master and ESP32 Slave work together to maintain synchronized traffic signals. When the ESP32 Master changes Traffic Signal 1 to green, it sends a synchronization signal to the ESP32 Slave, which then updates Traffic Signal 2 to match the status of Signal 1, ensuring that both signals are in sync. This diagram highlights the master-slave communication used in the system to keep traffic signals aligned and prevent accidents due to unsynchronized signals.

The fourth sequence diagram (Figure 3.8) addresses the real-time feedback between the ESP32 Master and Traffic Signals. This diagram shows how the ESP32 Master continuously checks and updates the traffic signal statuses, ensuring that both Traffic Signal 1 and Traffic Signal 2 reflect the current state of the system. This process allows the system to maintain accurate, up-to-date information, providing the Traffic Warden with the most current signal data via the Mobile App.

3.3.6 User Interface Design

The User Interface (UI) of the Portable Wireless Traffic Control System was designed with simplicity and usability in mind, providing users with an intuitive way to manage traffic signals and adjust system settings in real time. The system's UI consists of three main screens: the Login Screen, the Main Control Screen, and the User Profile Screen.

Figure 3. 9 Login Screen of the Mobile App

The Login Screen (Figure 3.9) is the first point of interaction for the user. It features a clean and straightforward layout, with the title "Portable Wireless Traffic Control System" at the top, giving users a clear indication of the system's purpose. Below this, users are prompted to enter their username and password in clearly labelled fields, and the Login button is easily visible for submission. This page ensures secure access to the system, with the option to log in with a pre-filled username for returning users. The login interface is designed to be minimalistic to reduce clutter and enhance user focus, making the login process efficient.

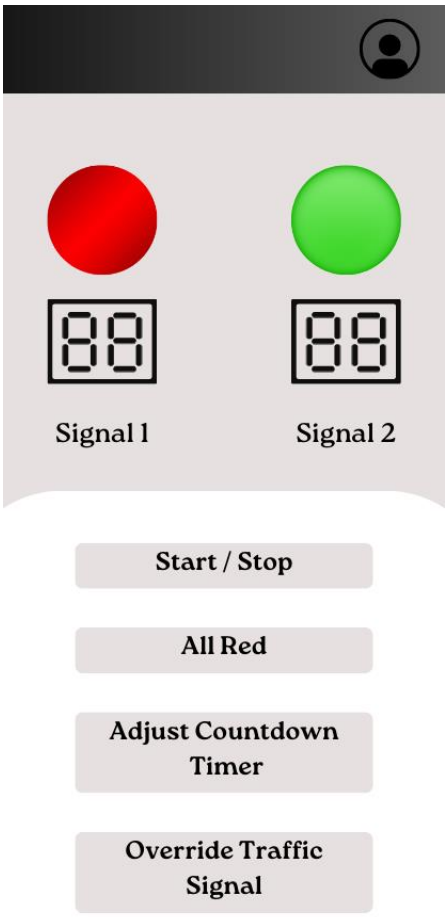

Figure 3. 10 Main Control Screen of the Mobile App

Once logged in, users are taken to the Main Control Screen (Figure 3.10), where they can manage the traffic signals. The screen displays the status of Signal 1 (red) and Signal 2 (green), each accompanied by a countdown timer that shows how many seconds are left before the signals change. Below the traffic signals, four functional buttons are provided: Start/Stop, All Red, Adjust Countdown Timer, and Override Traffic Signal. These buttons are clearly labeled and arranged in a user-friendly layout, making it easy to control the traffic signals. The "Start/Stop" button activates or halts the system, while the "All Red" button allows the user to override the signals and turn both signals red. The "Adjust Countdown Timer" button enables the user to modify the countdown timer, and the "Override Traffic Signal" button allows the user to manually switch either traffic signal. This screen's layout and functionality ensure quick and efficient control of the traffic system.

3.4 System Functionalities

This section describes the core functionalities of the Portable Wireless Traffic Control System that empower the traffic warden to efficiently manage traffic signals and optimize the flow of vehicles. Table 3.3 below provides an overview of the key features of the system, designed to ensure smooth traffic control, manual override options, and synchronization between traffic signals in real-time.

Table 3. 3 System functionalities for students

System functionalities	Description
Manage Traffic Signal Status	The system allows the warden to monitor and manage the status of Traffic Signal 1 and Traffic Signal 2. Each signal's status is displayed clearly on the mobile app, showing whether they are red, green, or yellow.
Adjust Countdown Timer	The Mobile App allows the warden to adjust the countdown timer for each traffic signal. This feature ensures that signal timings can be adjusted based on traffic flow, improving flexibility.
Manual Override Control	The warden can override the automatic sequence of traffic lights, providing manual control over the signals. For example, they can set both signals to red for special purposes, such as allowing emergency vehicles to pass.
Synchronization of Traffic Signals	The system ensures that Traffic Signal 1 (Master) and Traffic Signal 2 (Slave) remain synchronized. When one signal changes, the system automatically adjusts the other to maintain proper traffic flow.
Start/Stop System	The system can be started or stopped via the mobile app. This functionality is useful for maintenance or testing purposes.
Real-Time Traffic Status Feedback	The system continuously updates the Mobile App with real-time feedback about each signal’s current state (e.g., green or red). This allows the warden to stay informed about the operational status at all times.

3.5 Summary

Chapter 3 provided a detailed analysis of the requirement gathering and system design for the Portable Wireless Traffic Control System for Partial Road Closures. In the requirement analysis phase, the system's user, hardware, software, functional, and non-functional requirements were outlined. Insights from surveys and interviews with traffic wardens and users guided the identification of key features, such as real-time traffic signal synchronization, manual override capabilities, and adjustable countdown timers. These features were deemed essential for the system's adaptability and efficiency in managing traffic during partial road closures. The hardware and software specifications were also discussed, focusing on the selection of components like ESP32 microcontrollers, infrared sensors, and a custom communication protocol to enable offline functionality and reliable synchronization.

The design phase of the project included the creation of system architecture diagrams, flowcharts, block diagrams, context diagrams, sequence diagrams, and user interface designs. These visual elements provided a comprehensive blueprint for the system, demonstrating the interaction between hardware and software components. Notably, the system is designed for portability, offline operation, and ease of use through a mobile application that allows traffic wardens to monitor real-time traffic status, override signals, and adjust countdown timers. The design and user interface were aimed at ensuring the system's flexibility and practicality, particularly in rural or resource-limited environments where internet connectivity is unavailable.

In summary, Chapter 3 established the system's functional framework and detailed its design elements, all of which are geared toward enhancing traffic safety and efficiency during temporary road closures. The integration of automation, real-time adaptability, and user-driven control ensures that the system can meet the operational needs of traffic wardens while addressing the challenges faced in rural and offline scenarios.

CHAPTER 4: SYSTEM IMPLEMENTATION

4.1 Introduction

This chapter explains the implementation phase of the Portable Wireless Traffic Control System for Partial Road Closures. It focuses on the development of both the hardware and software components that make up the system. The system consists of two ESP32 microcontrollers, each responsible for controlling a set of traffic lights and detecting vehicle presence using ultrasonic sensors. A mobile application was also developed to monitor the real-time status of the traffic system and provide remote override functionalities.

The implementation process covers the setup of hardware modules, integration of LED traffic lights and ultrasonic sensors, development of the ESP32 software using the Arduino framework, and the design of a mobile application using Flutter. This chapter also details how the ESP32 devices communicate with each other wirelessly using ESP-NOW, while simultaneously hosting a SoftAP-based HTTP server for mobile app connectivity.

4.2 System Overview

The Portable Wireless Traffic Control System for Partial Road Closures is designed to manage temporary one-way traffic during road maintenance, emergencies, or partial closures. The system consists of two identical units—ESP32 A and ESP32 B—each equipped with an ultrasonic sensor, a 3-color LED traffic light, and a two-digit 7-segment countdown display. These two units communicate with each other over a low-power peer-to-peer protocol, ESP-NOW, to coordinate traffic flow and prevent simultaneous green lights.

Each ESP32 unit functions independently but synchronizes decisions based on the other unit's current state and sensor readings. When a vehicle is detected within a short distance (≤ 7 cm), the unit requests control of the green light. Only one ESP32 is allowed to activate the green light at a time. To reinforce this safety mechanism, a default 3-second red buffer is applied before switching control from one ESP32 to the other.

A mobile application built with Flutter allows users to connect wirelessly to either ESP32 via Wi-Fi (SoftAP mode). Through this connection, users can monitor the system status in real time and issue control commands such as:

- **Emergency Shutdown**: Immediately turns off all lights and disables system logic.
- **Force All Red**: Forces both ESP32 units to show red light for safety or override purposes.
- **Duration of Green**: Allows the user to set the minimum green light duration for active units. Available options include 10 seconds (default), 15s, 20s, 25s, and 30s.
- **Buffer Delay Control**: Lets the user adjust the safety delay between one ESP32 turning red and the other turning green. Available options include 3s (default), 5s, 10s, and 15s.

The communication between the mobile app and each ESP32 is handled via HTTP requests to a local web server hosted on the ESP32. Each ESP32 acts as a SoftAP, allowing a nearby mobile device to connect directly without needing an external Wi-Fi network.

Figure 4.1 below illustrates the high-level architecture of the system, while Figure 4.2 shows the photo of the physical prototype.

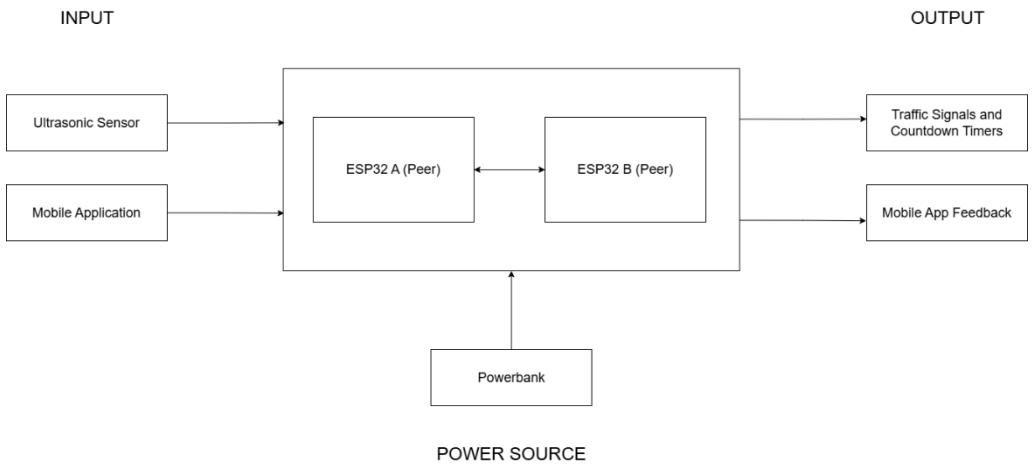

Figure 4. 1 Block Diagram of the whole system

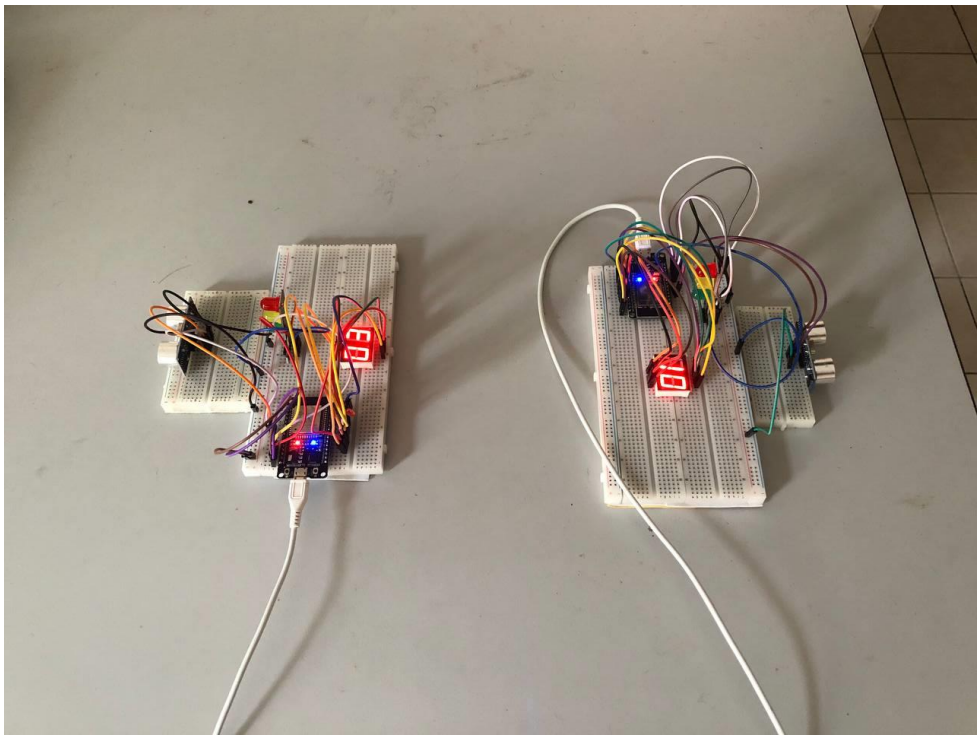

Figure 4. 2 Photo of real prototype

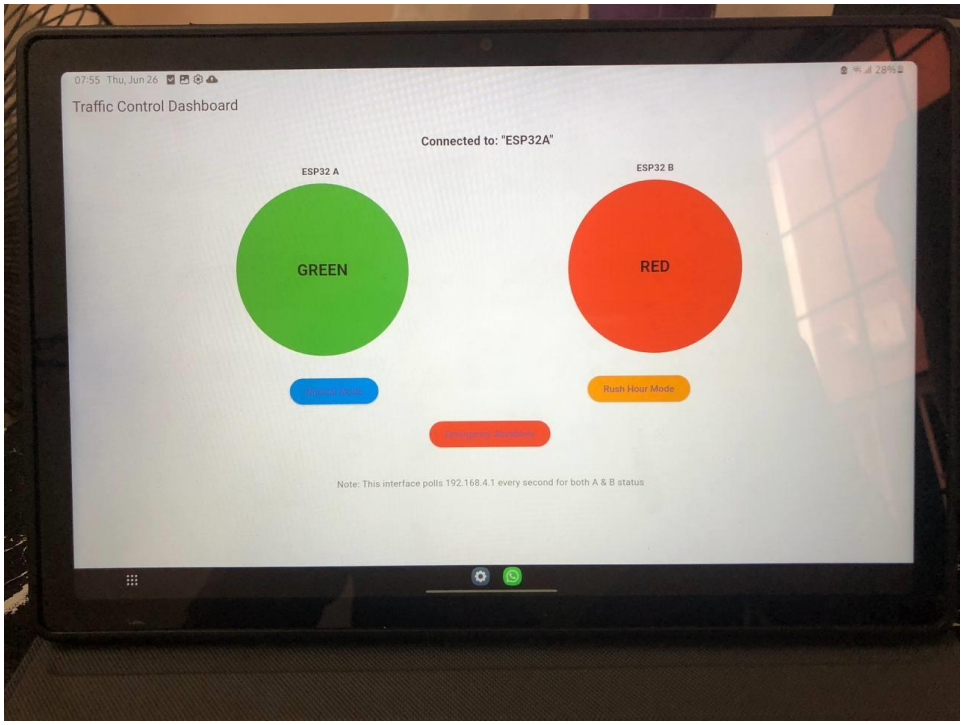

Figure 4. 3 Screenshot of the Home Screen of the Mobile Application

4.3 Hardware Implementation

4.3.1 Hardware Components

The hardware components of the system involves two identical ESP32-based units (Unit A and Unit B), each responsible for monitoring vehicle presence and controlling a set of traffic lights. The components were directly connected without using a breadboard, and the entire system was powered by portable power banks, supporting mobility for roadside deployment.

Each unit consists of the following components:

- **ESP32 Dev Module**: Acts as the main controller, responsible for sensor reading, LED control, communication, and running a local HTTP server.
- **HC-SR04 Ultrasonic Sensor**: Used to detect vehicles approaching the traffic control point. It measures the distance in real-time to determine the presence of a vehicle within a 7 cm range.
- **3-Color LED Traffic Light (Red, Yellow, Green)**: Provides visual signals for traffic control, indicating stop, prepare, or go.
- **Two-digit 7-Segment Display (Common Cathode)**: Displays a countdown before switching to GREEN or YELLOW, offering a visual cue to approaching vehicles or road workers.
- **Traffic Light Module (Red Yellow Green):** A compact module used for visual traffic control. Each LED is individually addressable and controlled by the ESP32 to represent STOP (Red), GET READY (Yellow) and GO (Green) states.

All components were wired directly to the ESP32 pins using male-to-female jumper wires. The simplicity of the connection allows the system to be compact, portable, and quick to set up in a real-world partial road closure scenario.

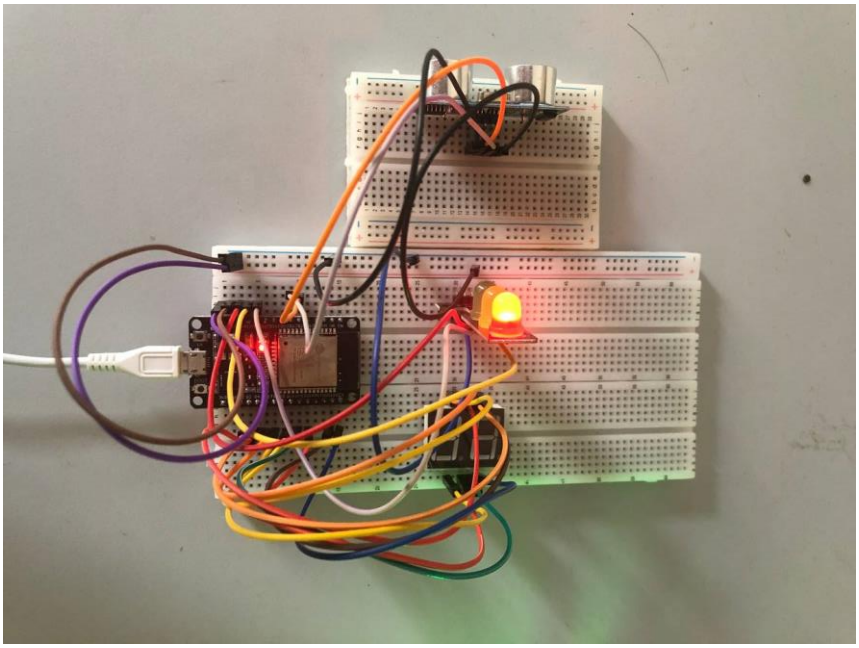

Figure 4. 4 shows the actual prototype of one ESP32 unit with all hardware components assembled.

4.3.2 Pin Configuration

The table below shows the GPIO pin assignments used for connecting the ESP32 to all hardware components. Both ESP32 A and B use the same configuration for consistency and simplicity during deployment.

Component	Signal Name	ESP32 GPIO Pin
Ultrasonic Sensor (HC-SR04)	`trigPin`	GPIO 33
	`echoPin`	GPIO 32
Traffic Light Module (LEDs)	`redPin`	GPIO 13
	`yellowPin`	GPIO 12
	`greenPin`	GPIO 14
7-Segment Display – Segment A	`segmentPins[0]`	GPIO 16

7-Segment Display – Segment B	`segmentPins[1]`	GPIO 17
7-Segment Display – Segment C	`segmentPins[2]`	GPIO 18
7-Segment Display – Segment D	`segmentPins[3]`	GPIO 19
7-Segment Display – Segment E	`segmentPins[4]`	GPIO 21
7-Segment Display – Segment F	`segmentPins[5]`	GPIO 22
7-Segment Display – Segment G	`segmentPins[6]`	GPIO 23
7-Segment Display – Digit 1	`digitPins[0]`	GPIO 2
7-Segment Display – Digit 2	`digitPins[1]`	GPIO 4

Table 4. 1 ESP32 GPIO Pin Configuration

4.4 Software Implementation

This section describes the software setup and implementation process for both the ESP32 firmware and the mobile application. It includes the required development environments, software tools, libraries, and debugging methods.

4.4.1 Arduino IDE Setup and Firmware Upload

The ESP32 microcontrollers were programmed using the **Arduino IDE**. The setup process involved installing the board manager, relevant libraries, and configuring the board and port settings.

Steps to Set Up **Arduino IDE**:

1. **Download and Install Arduino IDE** - To begin the firmware development, the Arduino IDE was downloaded from the official Arduino website at https://www.arduino.cc/en/software.

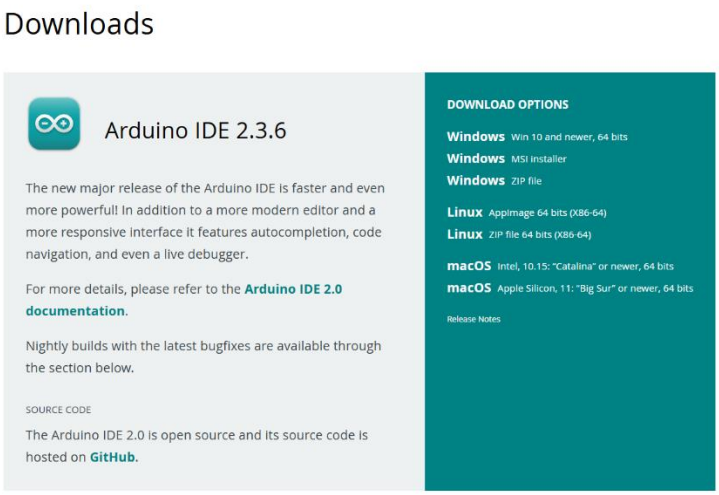

Figure 4. 5 illustrates the Arduino IDE download page

2. **Install ESP32 Board Support -** To install support for the ESP32 board, the user must first open the Arduino IDE and navigate to **File > Preferences**. In the "Additional Board Manager URLs" field, the following URL is added:

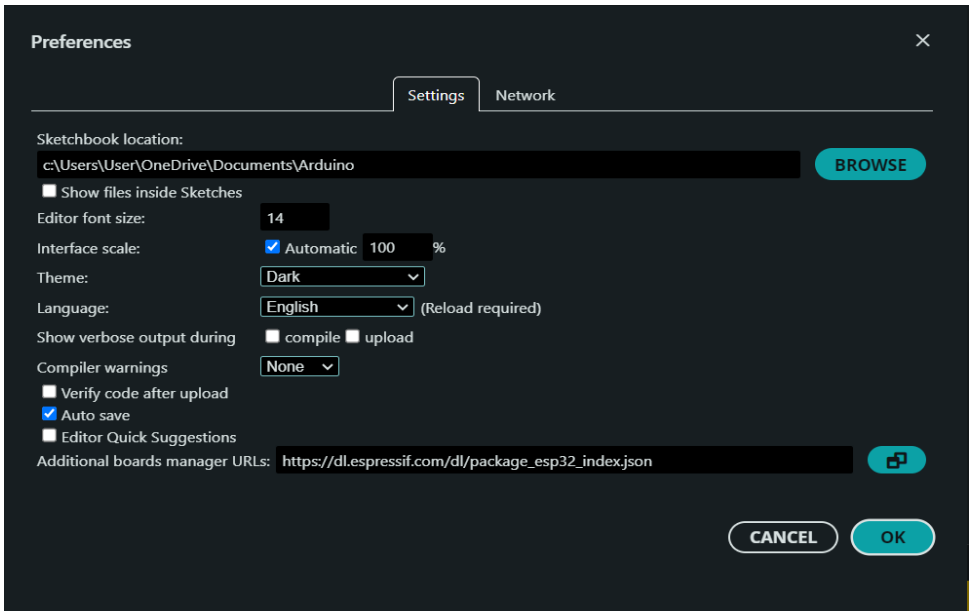

Figure 4. 6 shows the configuration of the ESP32 board URL in the Arduino IDE preferences.

After adding the ESP32 board URL, the user navigates to **Tools > Board > Boards Manager**, searches for "ESP32", and installs the **ESP32 by Espressif Systems** package.

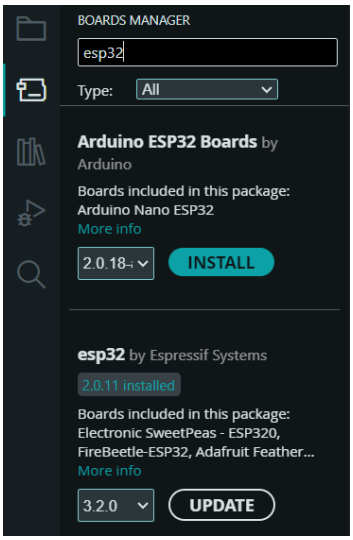

Figure 4. 7 shows the installation of the ESP32 board package via the Boards Manager.

3. **Set the Board and COM Port** - Once the board package is installed, the user selects ESP32 Dev Module from the Tools > Board menu. Then, under Tools > Port, the appropriate COM port corresponding to the connected ESP32 device is selected.

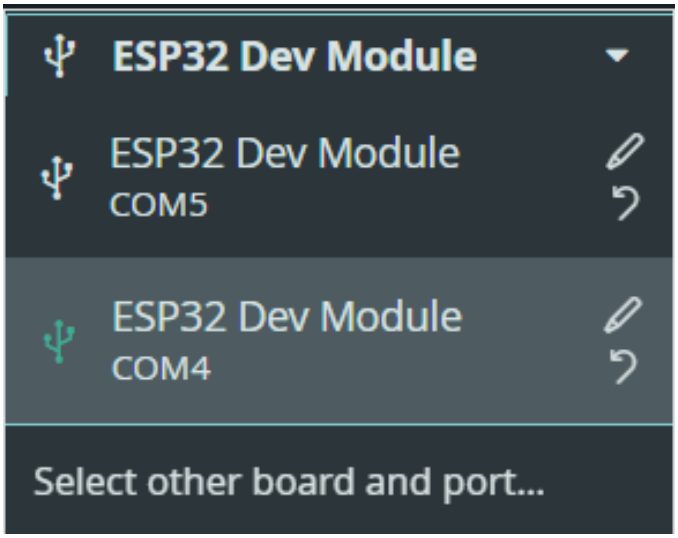

Figure 4. 8 shows the Board and COM Port.

4. **Install Required Libraries -** Next, the necessary libraries are installed using the Library Manager in the Arduino IDE. These include ESPAsyncWebServer, AsyncTCP, WiFi, and esp_now. These libraries enable network connectivity, asynchronous HTTP server handling, and peer-to-peer communication between ESP32 devices.

Name	Date modified	Type	Size
AsyncTCP	23/6/2025 10:41 PM	File folder	
Fixed_ESP_Async_WebServer	23/6/2025 10:41 PM	File folder	

Figure 4. 9 illustrates the installation of the required libraries using the Library Manager.

5. **Upload Code -** After setting up the board and installing the necessary libraries, the ESP32 is connected to the computer via a USB cable. The user then clicks the Upload button in the Arduino IDE to compile and upload the code to the ESP32.

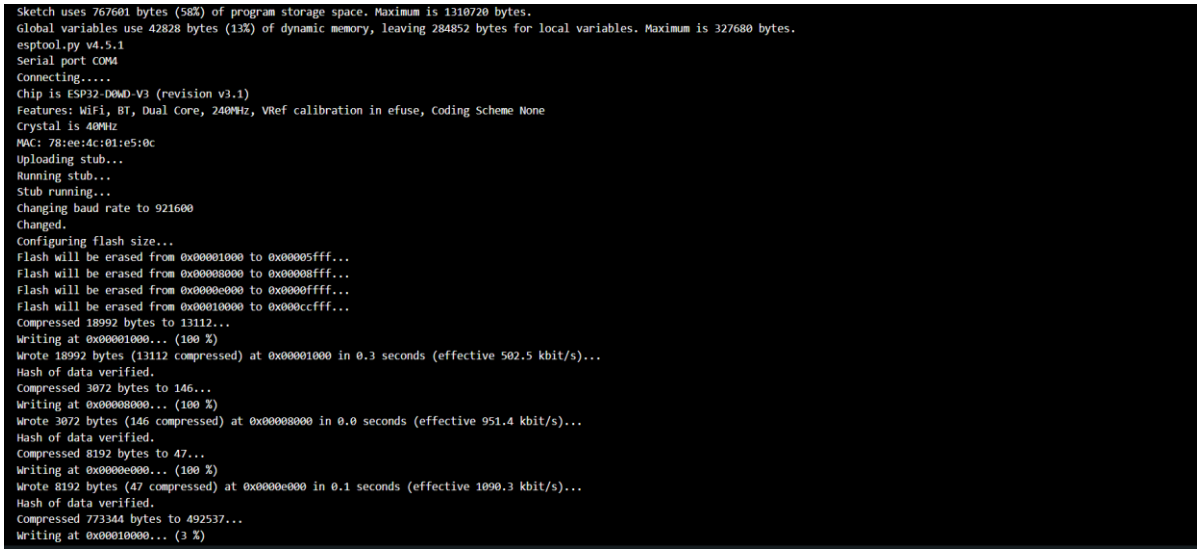

Figure 4. 10 shows the process of uploading the code to the ESP32 via the Arduino IDE.

4.4.2 Flutter and Android Studio Setup

To develop the mobile application, the **Flutter SDK** was used along with **Android Studio** as the integrated development environment (IDE). The following steps were followed to prepare the development environment:

1. **Install Flutter SDK -** The Flutter SDK was downloaded and installed by following the official documentation at https://docs.flutter.dev/get-started/install. After installation, the command flutter doctor was executed in the terminal to verify that all dependencies were correctly configured.

```
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.3.10, on macOS 12.6.2 21G320 darwin-arm (Rosetta), locale en-TH)
[!] Android toolchain - develop for Android devices (Android SDK version 33.0.1)
    ! Some Android licenses not accepted.  To resolve this, run: flutter doctor --android-licenses
[✗] Xcode - develop for iOS and macOS
    ✗ Xcode installation is incomplete; a full installation is necessary for iOS development.
      Download at: https://developer.apple.com/xcode/download/
      Or install Xcode via the App Store.
      Once installed, run:
        sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer
        sudo xcodebuild -runFirstLaunch
    ✗ CocoaPods not installed.
        CocoaPods is used to retrieve the iOS and macOS platform side's plugin code that responds
        to your plugin usage on the Dart side.
        Without CocoaPods, plugins will not work on iOS or macOS.
        For more info, see https://flutter.dev/platform-plugins
      To install see https://guides.cocoapods.org/using/getting-started.html#installation for
      instructions.
[✓] Chrome - develop for the web
[✓] Android Studio (version 2022.1)
[✓] VS Code (version 1.74.3)
[✓] Connected device (2 available)
[✓] HTTP Host Availability

! Doctor found issues in 2 categories.
```

Figure 4. 11 shows the Flutter installation verification using the flutter doctor command.

2. **Install Android Studio -** Android Studio was installed from the official Android developer website at https://developer.android.com/studio. Once installed, the Flutter and Dart plugins were added by navigating to Settings > Plugins, searching for each plugin, and clicking Install.

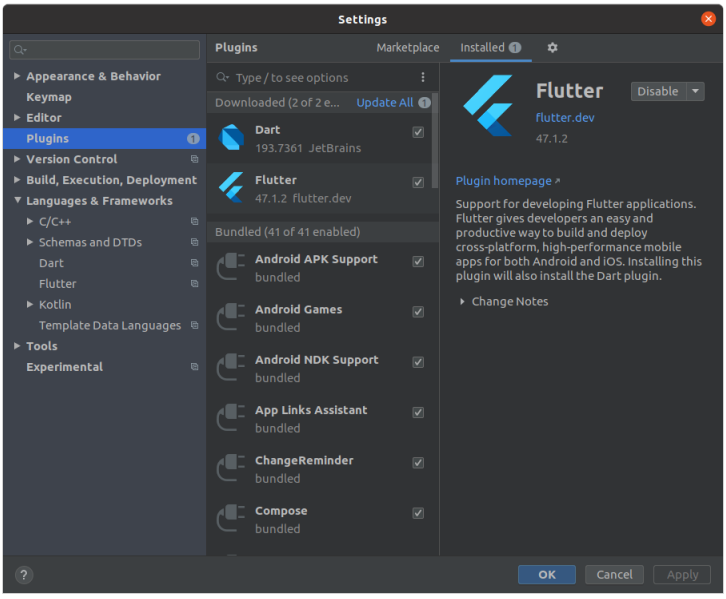

Figure 4. 12 shows the Flutter and Dart plugins installed within Android Studio.

3. **Open the Flutter Project -** After setting up the IDE, the user opened the existing Flutter project by selecting Open Project from the Android Studio welcome screen and navigating to the project folder. The project was then run using either an emulator or a physical Android tablet connected via USB.

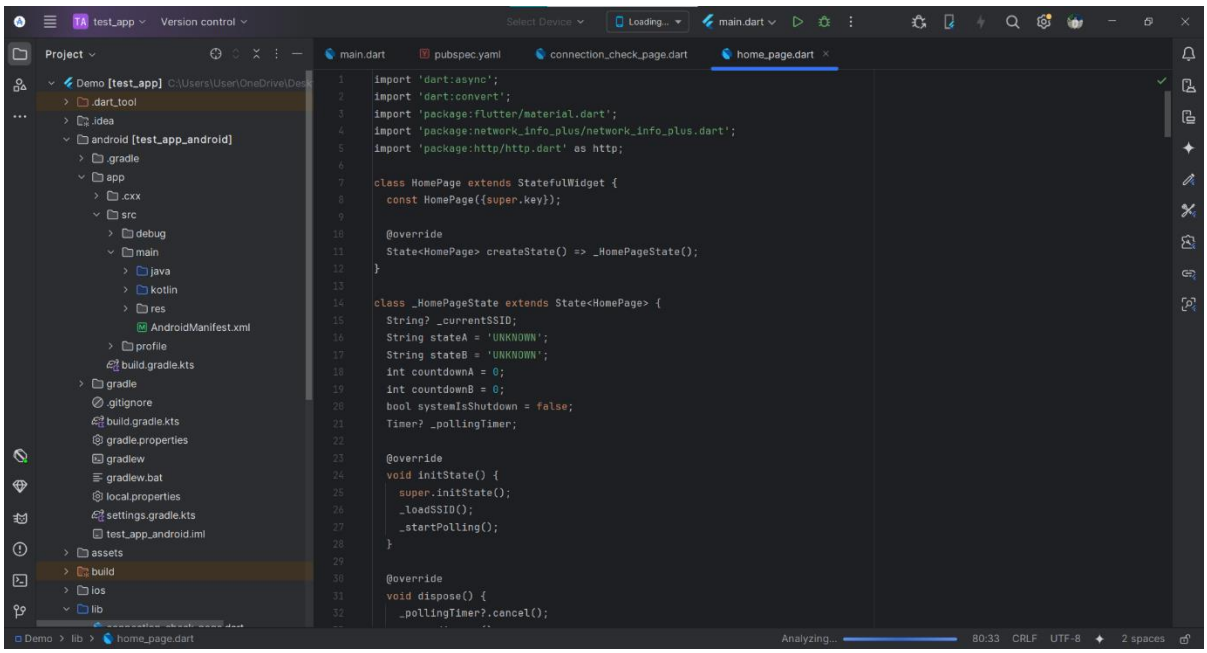

Figure 4. 13 shows the Flutter project loaded inside Android Studio.

4. **Install Flutter Dependencies -** In order to run the app, several Flutter packages were included in the pubspec.yaml file, such as http, connectivity_plus, network_info_plus, and permission_handler. These packages were installed using the flutter pub get command in the terminal.

```
PS C:\Users\User\OneDrive\Desktop\Flut_Proj\Demo> flutter pub get
  petitparser 6.1.0 (7.0.0 available)
  test_api 0.7.4 (0.7.6 available)
  vector_math 2.1.4 (2.2.0 available)
  vm_service 14.3.1 (15.0.2 available)
  win32 5.13.0 (5.14.0 available)
  xml 6.5.0 (6.6.0 available)
Got dependencies!
21 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
```

Figure 4. 14 displays the terminal output when fetching the Flutter dependencies.

4.4.3 Serial Monitor for Debugging

During development and testing, the **Serial Monitor** in the Arduino IDE played a critical role in debugging the ESP32 firmware. It allowed real-time observation of system behavior, sensor readings, state transitions, and communication between devices.

To access the Serial Monitor, the user navigated to **Tools > Serial Monitor** within the Arduino IDE. The baud rate was set to **115200**, which matches the value defined in the code using Serial.begin(115200);.

Figure 4. 15 displays the terminal output when fetching the Flutter dependencies.

The Serial Monitor was used to print out various debug messages. These included the current traffic light state (e.g., RED, GREEN, YELLOW), distance readings from the ultrasonic sensor, remaining countdown values, and messages received via ESP-NOW or HTTP.

These logs were especially helpful in:

- Verifying sensor detection accuracy
- Confirming whether the correct countdowns were triggered
- Identifying if any ESP-NOW message was dropped or duplicated
- Ensuring only one ESP32 was in the GREEN state at any given time

An example output is shown below:

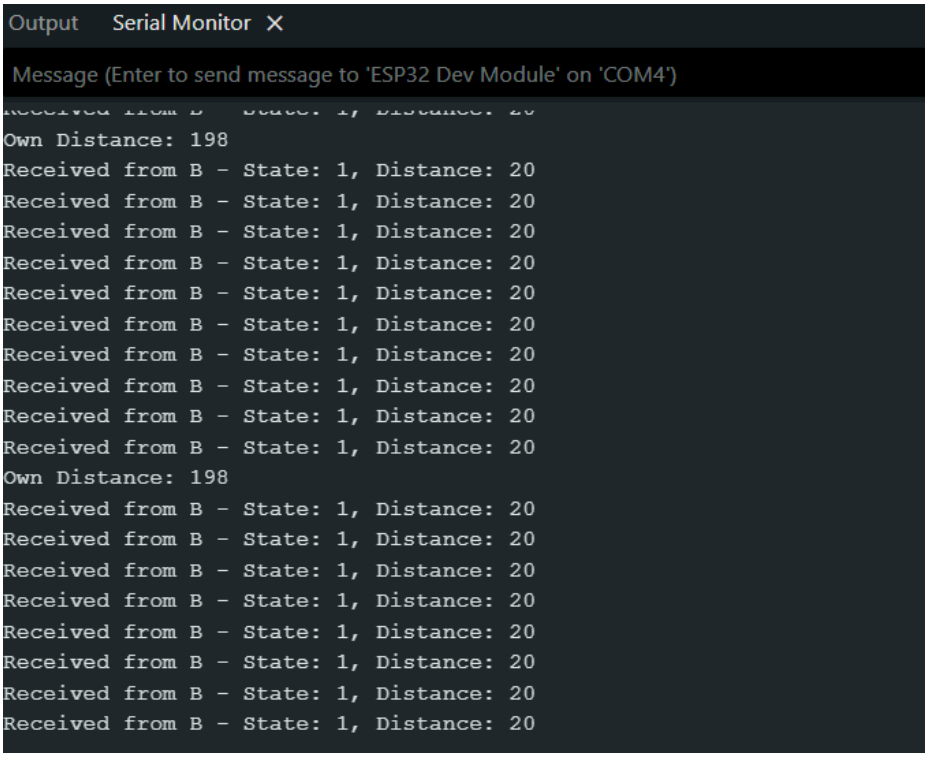

Figure 4. 16 shows a sample debug output in the Serial Monitor during system testing

By monitoring these outputs, issues could be identified and fixed more quickly during both the firmware development and system integration phases.

4.5 Communication Mechanism

The communication mechanism in this system integrates **two wireless protocols** to ensure seamless coordination and control:

- **ESP-NOW** is used for peer-to-peer communication between ESP32 A and ESP32 B.
- **HTTP (**over **Wi-Fi SoftAP)** is used for communication between the mobile application and either ESP32.

This dual-protocol approach ensures that both autonomous coordination and user-based manual override can function reliably and independently.

4.5.1 ESP-NOW Communication (ESP32 to ESP32)

ESP-NOW is a low-latency, connectionless, peer-to-peer protocol provided by Espressif. It allows ESP32 devices to exchange data directly without relying on Wi-Fi routers or external infrastructure.

For this FYP Project, ESP32 A and ESP32 B use ESP-NOW to:

- Send their current traffic light state
- Share countdown values
- Exchange distance sensor readings
- Notify each other to trigger countdowns
- Prevent simultaneous GREEN states

Data is sent in a structured format every **500 milliseconds**, and each ESP32 updates its peer status in real time.

Figure 4. 17 illustrates the ESP-NOW communication between ESP32 A and ESP32 B.

To implement this communication, each ESP32 uses the **esp_now_send()** and **OnDataRecv()** functions. The data is structured using a **struct_message** object and sent at regular intervals.

Below is a code snippet showing how data is transmitted via ESP-NOW:

```
// === STRUCT ===
typedef struct struct_message {
  int state;
  int distance;
  bool triggerCountdown;
  bool inCountdown;  // ✅ NEW
  int countdown;
  unsigned long messageID;
  char mode[10];  // e.g., "NORMAL" or "RUSH"
  // ✅ fields for override control
  bool emergencyShutdown;   // true = turn off system logic and lights
  bool forceAllRed;         // true = begin forcing everything to red
} struct_message;

struct_message outgoingData;
struct_message incomingData;
```

Figure 4. 18 ESP-NOW Struct Definition

```
void sendData() {
  outgoingData.state = currentState;
  outgoingData.distance = ownDistance;
  outgoingData.triggerCountdown = false;
  outgoingData.inCountdown = inCountdown;  // ✅ NEW
  outgoingData.countdown = countdown;  // ✅ Include countdown value in outgoing data
  outgoingData.messageID = millis();
  outgoingData.emergencyShutdown = emergencyShutdown;

  strcpy(outgoingData.mode, currentMode.c_str());

  if (currentState == GREEN && inCountdown) {
    int elapsed = (millis() - countdownStart) / 1000;
    if (elapsed == 6) {
      outgoingData.triggerCountdown = true;
      Serial.println("A: Triggering B to start RED-to-GREEN countdown");
    }
  }

  esp_now_send(peerAddress, (uint8_t*)&outgoingData, sizeof(outgoingData));
}
```

Figure 4. 19 Function to Send ESP-NOW Data

```
void OnDataRecv(const uint8_t *mac, const uint8_t *data, int len) {
  struct_message incoming;
  memcpy(&incoming, data, sizeof(incoming));

  if (incoming.messageID == lastMessageID) return;
  lastMessageID = incoming.messageID;

  peerPreviousState = peerState;
  peerState = incoming.state;
  peerDistance = incoming.distance;
  peerInCountdown = incoming.inCountdown;
  peerCountdown = incoming.countdown;  // ✅ FIXED
  peerEmergencyShutdown = incoming.emergencyShutdown;
  peerForceAllRed = incoming.forceAllRed;
  peerEmergencyShutdown = incoming.emergencyShutdown;

  // Optional debug prints
  if (peerEmergencyShutdown) {
    Serial.println("⚠️  Peer is in EMERGENCY SHUTDOWN");
  }
  if (peerForceAllRed) {
    Serial.println("🔴 Peer is forcing all RED");
  }
```

Figure 4. 20 Function to Receive ESP-NOW Data (Part of it)

```
249   // === ESP-NOW Init ===
250   if (esp_now_init() != ESP_OK) {
251     Serial.println("ESP-NOW init failed");
252     return;
253   }
254
255   esp_now_register_recv_cb(OnDataRecv);
256   memcpy(peerInfo.peer_addr, peerAddress, 6);
257   peerInfo.channel = 0;
258   peerInfo.encrypt = false;
259   if (esp_now_add_peer(&peerInfo) != ESP_OK) {
260     Serial.println("Peer add failed");
261     return;
262   }
```

Figure 4. 21 ESP-NOW Initialization and Peer Registration

4.5.2 HTTP Server (ESP32 to Mobile App)

Each ESP32 is configured as a **Wi-Fi Soft Access Point (SoftAP)**. This means the ESP32 creates its own Wi-Fi network, which the mobile app can connect to directly without needing a router or internet.

Each ESP32 runs an **asynchronous HTTP server** using the ESPAsyncWebServer library. This server exposes several endpoints that allow the mobile app to:

- **Read current state and countdown** (/status)
- **Trigger emergency shutdown** (/emergency)
- **Force both units to RED** (/forceAllRed)
- **Set minimum GREEN duration** (/setMode)
- **Change buffer delay duration** (/setBuffer)

All data is transferred via standard HTTP GET and POST requests, and responses are formatted in **JSON**, making it easy for the mobile app to parse and display.

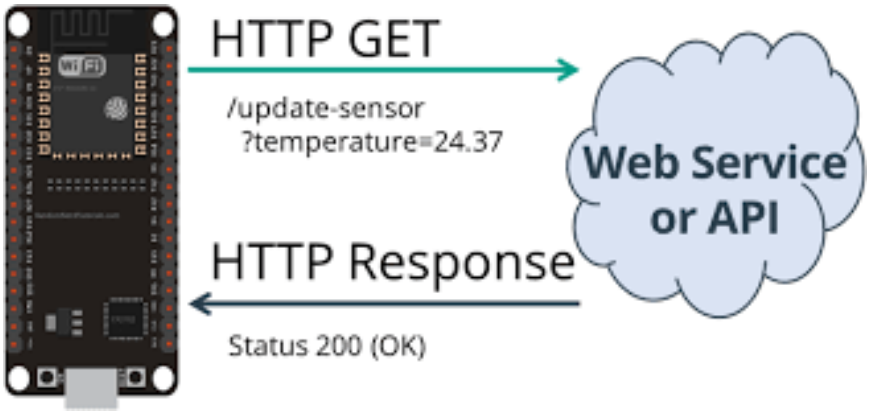

Figure 4. 22 Example of HTTP Communication Flow between Mobile App and ESP32

```
242   // === SoftAP Setup ===
243   WiFi.mode(WIFI_AP_STA);  // Enable both AP and STA
244   WiFi.softAP("ESP32A", "traffic123");  // AP SSID and password
245   IPAddress myIP = WiFi.softAPIP();
246   Serial.print("SoftAP IP: ");
247   Serial.println(myIP);  // Typically 192.168.4.1
```

Figure 4. 23 SoftAP Setup for HTTP Server

```
269 server.on("/status", HTTP_GET, [](AsyncWebServerRequest *request){
270   String json = "{";
271   json += "\"a\":{";
272   json += "\"state\":\"" + String(currentState == 0 ? "RED" : currentState == 1 ? "GREEN" : "YELLOW") + "\",";
273   json += "\"countdown\":" + String(inCountdown ? countdown : 0);
274   json += "},";
275
276   json += "\"b\":{";
277   json += "\"state\":\"" + String(peerState == 0 ? "RED" : peerState == 1 ? "GREEN" : "YELLOW") + "\",";
278   json += "\"countdown\":" + String(peerInCountdown ? peerCountdown : 0);
279   json += "},";
280
281   json += "\"mode\":\"" + currentMode + "\",";
282   json += "\"emergency\":" + String(emergencyShutdown ? "true" : "false");
283   json += "}";
284
285   request->send(200, "application/json", json);
286 });
```

Figure 4. 24. /**status** Endpoint for State and Countdown for ESP32A

```
289   server.on("/setMode", HTTP_POST, [](AsyncWebServerRequest *request){
290     if (request->hasParam("mode", true)) {
291       String mode = request->getParam("mode", true)->value();
292       mode.toUpperCase();
293       if (mode == "NORMAL") {
294         minGreenTime = 10000;
295         currentMode = "NORMAL";
296       } else if (mode == "RUSH") {
297         minGreenTime = 30000;
298         currentMode = "RUSH";
299       }
300       request->send(200, "text/plain", "Mode set to " + currentMode);
301       Serial.println("Mode changed to: " + currentMode);
302     } else {
303       request->send(400, "text/plain", "Missing 'mode' parameter");
304     }
305   });
```

Figure 4. 25/**setMode** Endpoint to Change Green Duration

```
307   // ✅ Send emergency shutdown to ESP32 B via ESP-NOW
308   server.on("/emergency", HTTP_POST, [](AsyncWebServerRequest *request){
309     if (request->hasParam("passcode", true)) {
310       String code = request->getParam("passcode", true)->value();
311
312       if (code == "1234") {
313         emergencyShutdown = true;
314         Serial.println("🚨 EMERGENCY MODE ACTIVATED (ESP32 A)");
315
316         // Try notifying peer (ESP32 B) via ESP-NOW
317         outgoingData.emergencyShutdown = true;
318         esp_err_t result = esp_now_send(peerAddress, (uint8_t*)&outgoingData, sizeof(outgoingData));
319
320         if (result == ESP_OK) {
321           request->send(200, "text/plain", "Emergency shutdown successful");
322         } else {
323           request->send(500, "text/plain", "Failed to notify peer");
324         }
325
326       } else {
327         request->send(403, "text/plain", "Invalid passcode");
328       }
329     } else {
330       request->send(400, "text/plain", "Missing parameter: passcode");
331     }
332   });
```

Figure 4. 26/**emergency** Endpoint for Shutdown Trigger

```
341   // === FORCE ALL RED ===
342   server.on("/forceAllRed", HTTP_POST, [](AsyncWebServerRequest *request){
343     if (request->hasParam("enable", true)) {
344       String enable = request->getParam("enable", true)->value();
345       forceAllRed = (enable == "true");
346       request->send(200, "text/plain", forceAllRed ? "All RED mode ENABLED" : "All RED mode DISABLED");
347       Serial.println(forceAllRed ? "🔴 Forcing all RED" : "🟢 All RED override cleared");
348     } else {
349       request->send(400, "text/plain", "Missing parameter: enable");
350     }
351   });
```

Figure 4. 27/**forceAllRed** Endpoint to Stop All Traffic

```
353    server.begin();
```

Figure 4. 28 Starting the Server

4.6 Traffic Light Logic and Control Flow

4.6.1 Core Logic Highlights

The traffic control system is based on a finite state machine shared between two ESP32 devices, each handling its own set of components (ultrasonic sensor, traffic light LEDs, and 7-segment display) while coordinating actions using ESP-NOW. The entire system is designed to ensure that **only one ESP32 is GREEN at any time**, with a buffer and synchronization logic for smooth transitions.

1. **Startup in RED State -** At boot, both ESP32 units begin in the RED state. The red LED is turned on, and they remain in this state while continuously measuring vehicle distance using the ultrasonic sensor.

```
227  void setup() {
228    Serial.begin(115200);
229
230    pinMode(redPin, OUTPUT);
231    pinMode(yellowPin, OUTPUT);
232    pinMode(greenPin, OUTPUT);
233    pinMode(trigPin, OUTPUT);
234    pinMode(echoPin, INPUT);
235
236    for (int i = 0; i < 7; i++) pinMode(segmentPins[i], OUTPUT);
237    for (int i = 0; i < 2; i++) pinMode(digitPins[i], OUTPUT);
238
239    digitalWrite(redPin, HIGH); // Start RED
240    redStart = millis();
```

Figure 4. 29 Startup Code for RED State Initialization

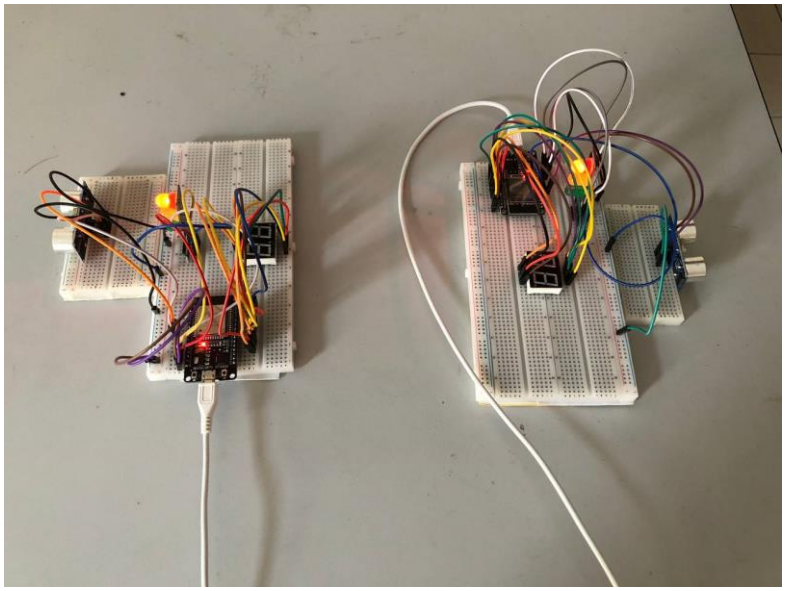

Figure 4. 30 Both ESP32 units showing RED at startup (real prototype photo)

2. **Vehicle Detection and Countdown Initiation -** If a vehicle is detected (≤7 cm) while both units are RED, the one with priority (typically A) starts a 10-second countdown. The red LED begins blinking, and the 7-segment display shows the countdown.

```
case RED:
  digitalWrite(yellowPin, LOW);
  digitalWrite(greenPin, LOW);

  if (inCountdown) {
    int elapsed = (now - countdownStart) / 1000;
    countdown = max(0, 10 - elapsed);
    displayTwoDigitNumber(countdown);

    if (now - lastBlinkTime >= 500) {
      lastBlinkTime = now;
      ledOn = !ledOn;
      digitalWrite(redPin, ledOn ? HIGH : LOW);
    }

    if (countdown == 0 && readyToGoGreen) {
      resetAllLEDs();
      currentState = GREEN;
      greenStart = now;
      inCountdown = false;
      readyToGoGreen = false;
      displayActive = false;
      Serial.println("A: Switching to GREEN");
      peerJustFinishedGreen = false;
    }
  } else {
    digitalWrite(redPin, HIGH);
    if (now - redStart >= 10000) {
      bool bothRed = (peerState == RED);
      bool selfDetect = (ownDistance <= 7);
      bool peerWaitComplete = peerJustFinishedGreen && (now - peerGreenEndTime >= bufferAfterPeerGreen);

      if (!inCountdown && bothRed && selfDetect && !peerInCountdown) {
        if (!peerJustFinishedGreen || peerWaitComplete) {
          inCountdown = true;
          readyToGoGreen = true;
          countdownStart = now;
          lastTriggerTime = now;
          Serial.println("A: RED done, starting countdown to GREEN");
        }
      }
    }
  }
  break;
```

Figure 4. 31 Code Snippet for RED-to-GREEN Countdown Trigger on Vehicle Detection

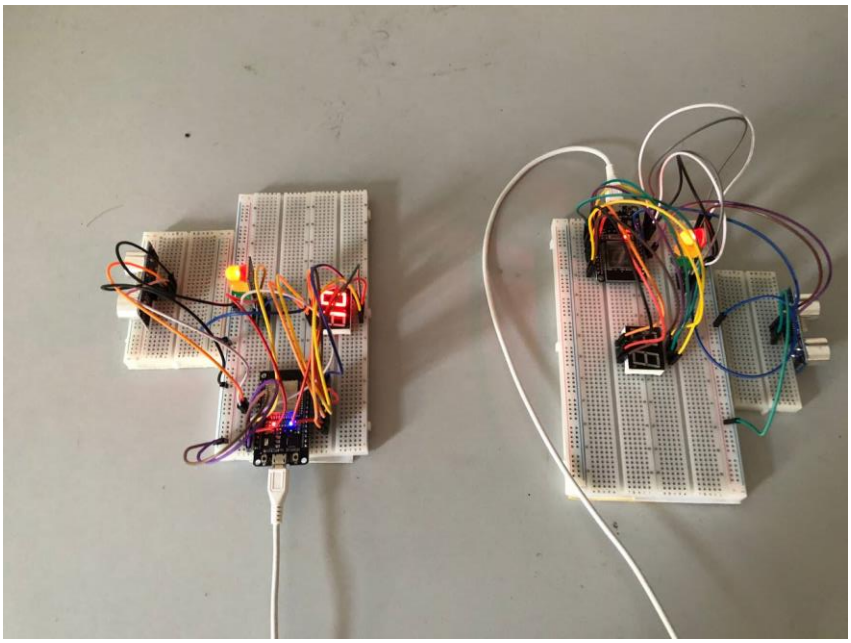

Figure 4. 32 Countdown from RED to GREEN initiated (display shows 10s, blinking red LED)

3. **GREEN State and Minimum Duration -** When countdown ends, the RED light turns off and the GREEN light activates. It stays GREEN for a minimum time (10s–30s) before transitioning.

```
413         if (countdown == 0 && readyToGoGreen) {
414           resetAllLEDs();
415           currentState = GREEN;
416           greenStart = now;
417           inCountdown = false;
418           readyToGoGreen = false;
419           displayActive = false;
420           Serial.println("A: Switching to GREEN");
421           peerJustFinishedGreen = false;
422         }
```

Figure 4. 33 Code for Transitioning from RED to GREEN After Countdown

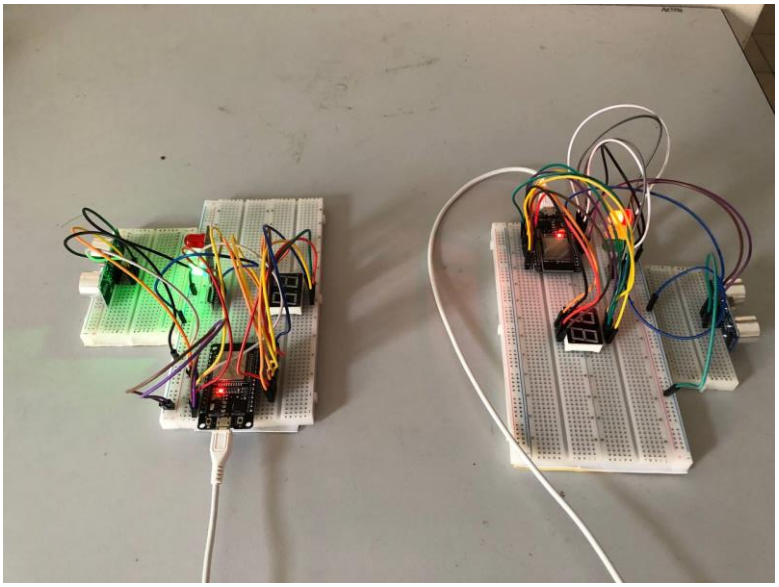

Figure 4. 34 Traffic light turns GREEN after countdown (prototype photo)

4. **Triggering the Peer ESP32 -** When GREEN countdown reaches 4s left, ESP32 A sends a triggerCountdown flag to ESP32 B to prepare its own RED-to-GREEN transition.

```
180   if (currentState == GREEN && inCountdown) {
181     int elapsed = (millis() - countdownStart) / 1000;
182     if (elapsed == 6) { // 10 - 6 = 4 seconds remaining
183       outgoingData.triggerCountdown = true;
184       Serial.println("A: Triggering B to start RED-to-GREEN countdown");
185     }
186   }
```

Figure 4. 35 Code to Trigger Peer ESP32 to Begin Countdown (4 Seconds Remaining)

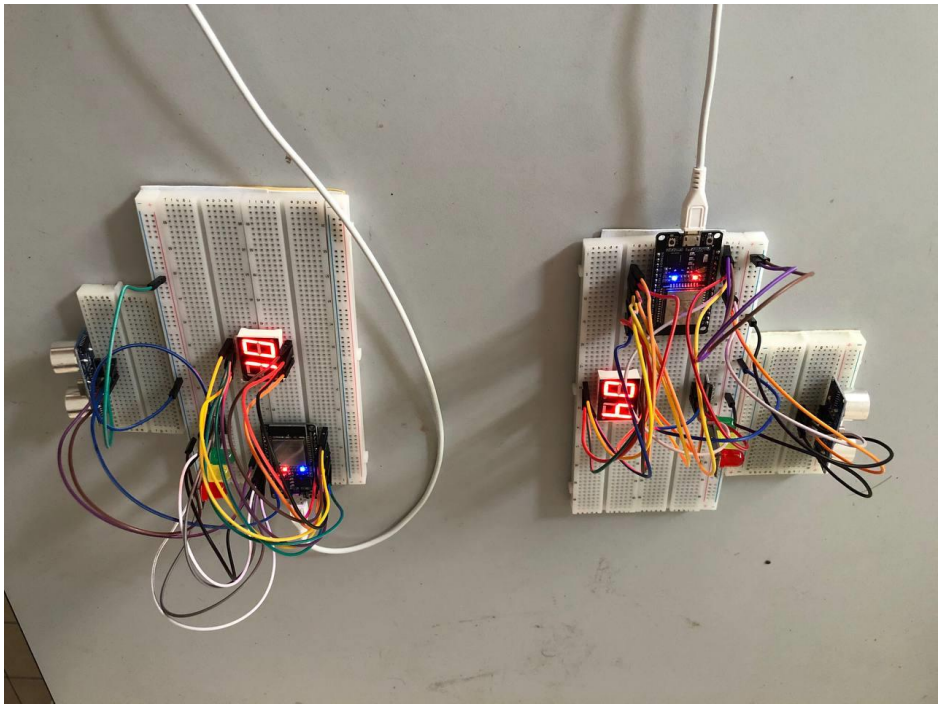

Figure 4. 36 Trigger sent to peer ESP32 (highlight on display showing 4)

5. **YELLOW Phase and Buffer Handling -** After the GREEN period, the light enters YELLOW state for 3 seconds. After that, RED resumes — but the peer must wait a buffer period (e.g., 3s) before turning GREEN.

```
458         if (countdown == 0) {
459           resetAllLEDs();
460           currentState = YELLOW;
461           yellowStart = now;
462           inCountdown = false;
463           displayActive = false;
464           Serial.println("A: Switching to YELLOW");
465         }
```

Figure 4. 37 Code for Entering YELLOW State After GREEN Ends

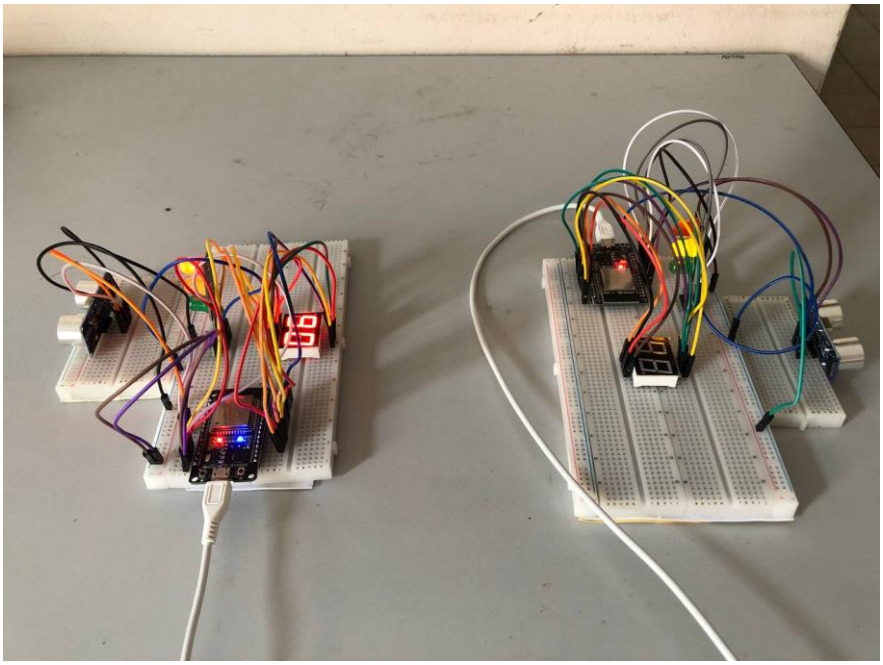

Figure 4. 38 YELLOW state active (prototype photo)

6. **LED Blinking and Display Behavior -** During countdowns, the active LED (RED or GREEN) blinks every 500ms, and the 7-segment display shows the timer.

```
402     if (inCountdown) {
403       int elapsed = (now - countdownStart) / 1000;
404       countdown = max(0, 10 - elapsed);
405       displayTwoDigitNumber(countdown);
406
407       if (now - lastBlinkTime >= 500) {
408         lastBlinkTime = now;
409         ledOn = !ledOn;
410         digitalWrite(redPin, ledOn ? HIGH : LOW);
411       }
```

Figure 4. 39 Code Handling LED Blinking and Countdown Display Logic

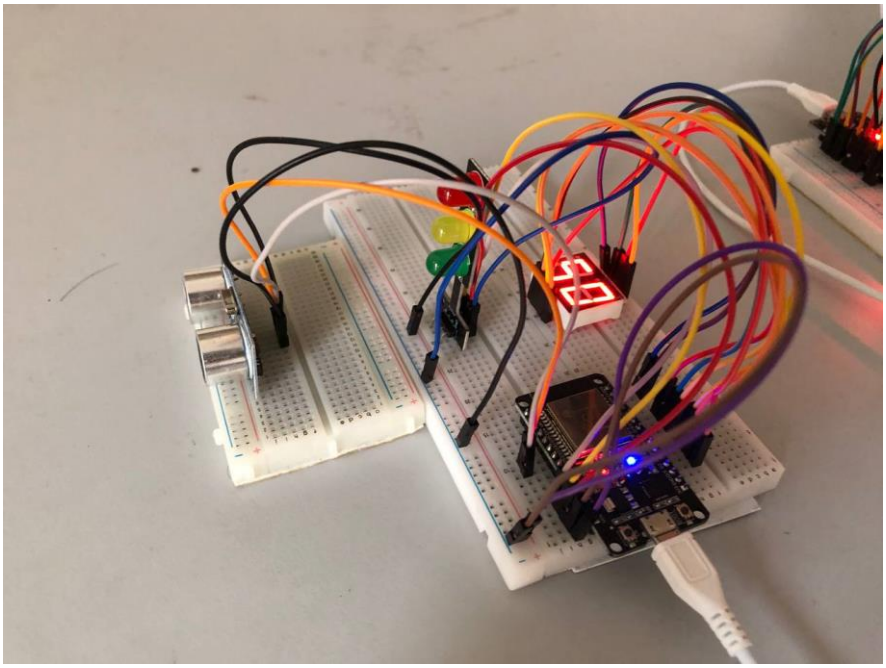

Figure 4. 40 Blinking LED + countdown display during transition 1

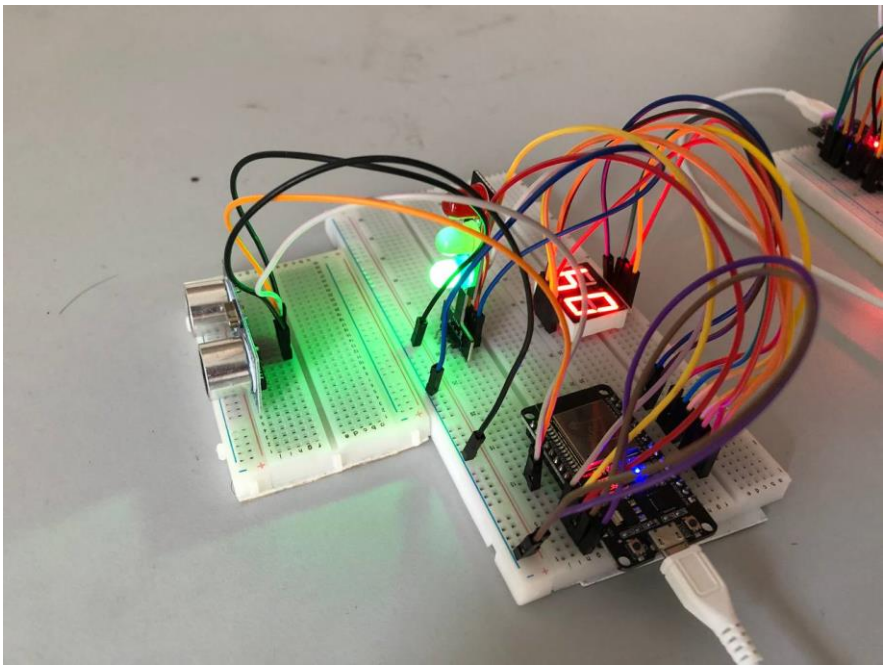

Figure 4. 41 Blinking LED + countdown display during transition 2

4.6.2 Main Loop Execution

The **loop()** function acts as the central controller for the traffic light system, continuously managing tasks such as distance sensing, ESP-NOW data transmission, LED transitions, and countdown display updates. This loop ensures that the traffic light logic runs smoothly and in sync with the peer ESP32 unit.

1. **Periodic Distance Sensing and Data Transmission -** Every 500 milliseconds, the system reads the distance from the HC-SR04 ultrasonic sensor and transmits the current state and other variables to the peer via ESP-NOW.

```
361   if (now - lastSentTime >= sendInterval) {
362     lastSentTime = now;
363     ownDistance = measureDistance();
364     sendData();
365
366     if (now - lastPrintTime >= printInterval) {
367       lastPrintTime = now;
368       Serial.print("Own Distance: ");
369       Serial.println(ownDistance);
370     }
371   }
```

Figure 4. 42 Code snippet of distance measurement and data transmission every 500 ms.

To help with debugging and system validation, the current distance is also printed every 5 seconds:

```
366     if (now - lastPrintTime >= printInterval) {
367       lastPrintTime = now;
368       Serial.print("Own Distance: ");
369       Serial.println(ownDistance);
370     }
371   }
```

Figure 4. 43 Code snippet for printing the measured distance to the serial monitor every 5 seconds for debugging purposes.

2. **Emergency Shutdown Handling -** If either ESP32 detects an emergency shutdown (via the app or peer signal), the logic is halted immediately. All LEDs are turned off and display is disabled to indicate system halt.

```
373   if (emergencyShutdown || peerEmergencyShutdown) {
374   resetAllLEDs();
375   displayActive = false;
376   return; // 🔴 Fully halted — nothing runs
377   }
```

Figure 4. 44 Code snippet demonstrating how the system enters emergency shutdown mode and halts all traffic light operations.

3. **Force All RED Override Handling -** This logic ensures that, when the user requests all traffic lights to be red (via the app), the system transitions safely, even if a GREEN state is in progress.

```
379   if (forceAllRed || peerForceAllRed) {
380     if (currentState == GREEN && !inCountdown) {
381       // Begin GREEN to YELLOW countdown
382       inCountdown = true;
383       countdownStart = millis();
384       Serial.println("🔁 GREEN interrupted — forcing RED transition");
385     }
386
387     // If already in RED, stay there — nothing to do
388     if (currentState == RED) {
389       digitalWrite(redPin, HIGH);
390       displayActive = false;
391     }
392
393     // If in YELLOW or countdown, allow normal logic to complete
394     // to avoid abrupt cut
395   }
```

Figure 4. 45 Code snippet handling override logic to force all traffic lights to RED for safety, including safe transition from GREEN.

4. **RED State Logic: Transition to GREEN -** If a vehicle is detected and both ESP32 units are red (and conditions are safe), the system initiates a RED-to-GREEN countdown.

```
430         if (!inCountdown && bothRed && selfDetect && !peerInCountdown) {
431           if (!peerJustFinishedGreen || peerWaitComplete) {
432             inCountdown = true;
433             readyToGoGreen = true;
434             countdownStart = now;
435             lastTriggerTime = now;
436             Serial.println("A: RED done, starting countdown to GREEN");
437           }
438         }
```

Figure 4. 46 Code snippet showing logic for transitioning from RED to GREEN when vehicle is detected and conditions are met.

5. **GREEN State Logic: Transition to YELLOW -** Once the minimum GREEN duration has passed and the peer detects a vehicle, the ESP32 transitions from GREEN to YELLOW.

```
468       if ((now - greenStart >= minGreenTime && peerDistance <= 7) || forceAllRed || peerForceAllRed) {
469         inCountdown = true;
470         countdownStart = now;
471         Serial.println("A: GREEN done, starting countdown to YELLOW");
472       }
```

Figure 4. 47 Code snippet showing transition logic from GREEN to YELLOW based on minimum duration and peer sensor input.

6. **YELLOW State Logic: Transition to RED -** The YELLOW state lasts for a fixed duration (e.g., 3 seconds), after which the system re-enters the RED state.

```
481         if (now - yellowStart >= yellowDuration) {
482           currentState = RED;
483           redStart = now;
484           Serial.println("A: Switching to RED");
485         }
```

Figure 4. 48 Code snippet showing YELLOW-to-RED transition after a fixed delay, completing the full state cycle.

4.7 Mobile Application Logic

The mobile application, built using Flutter, serves two primary functions: real-time monitoring and manual control of the ESP32-based traffic light system. It communicates with the ESP32's built-in HTTP server using GET and POST requests.

4.7.1 Mobile App Interface and Control Logic

1. **Fetching Real-Time Traffic Status -** The app periodically polls the /status endpoint to retrieve the current state and countdown of both ESP32 A and B.

```
server.on("/status", HTTP_GET, [](AsyncWebServerRequest *request){
  String json = "{";
  json += "\"a\":{";
  json += "\"state\":\"" + String(currentState == 0 ? "RED" : currentState == 1 ? "GREEN" : "YELLOW") + "\",";
  json += "\"countdown\":" + String(inCountdown ? countdown : 0);
  json += "},";

  json += "\"b\":{";
  json += "\"state\":\"" + String(peerState == 0 ? "RED" : peerState == 1 ? "GREEN" : "YELLOW") + "\",";
  json += "\"countdown\":" + String(peerInCountdown ? peerCountdown : 0);
  json += "},";

  json += "\"mode\":\"" + currentMode + "\",";
  json += "\"emergency\":" + String(emergencyShutdown ? "true" : "false");
  json += "}";

  request->send(200, "application/json", json);
});
```

Figure 4. 49 Code snippet showing how the app fetches traffic light status using an HTTP GET request to /status.

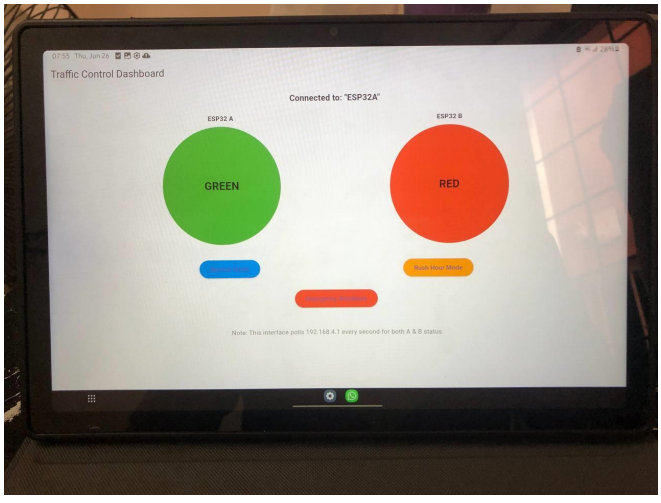

Figure 4. 50 Screenshot of the home screen showing live status, countdown timers and override buttons for ESP32 A and B.

2. **Control Features via HTTP POST -** Users can send control commands to the ESP32 using buttons in the app. Each button corresponds to a different endpoint.

    i. **Duration of GREEN -** The dropdown sends a POST request to adjust the minimum GREEN light duration.

```
server.on("/setMode", HTTP_POST, [](AsyncWebServerRequest *request){
  if (request->hasParam("mode", true)) {
    String mode = request->getParam("mode", true)->value();
    mode.toUpperCase();
    if (mode == "NORMAL") {
      minGreenTime = 10000;
      currentMode = "NORMAL";
    } else if (mode == "RUSH") {
      minGreenTime = 30000;
      currentMode = "RUSH";
    }
    request->send(200, "text/plain", "Mode set to " + currentMode);
    Serial.println("Mode changed to: " + currentMode);
  } else {
    request->send(400, "text/plain", "Missing 'mode' parameter");
  }
});
```

Figure 4. 51 Code snippet for sending the selected GREEN duration to the ESP32 via a POST request to /setMode.

    ii. **Buffer Delay Control -** (Similar logic to be implemented on the ESP32 to support this feature.)

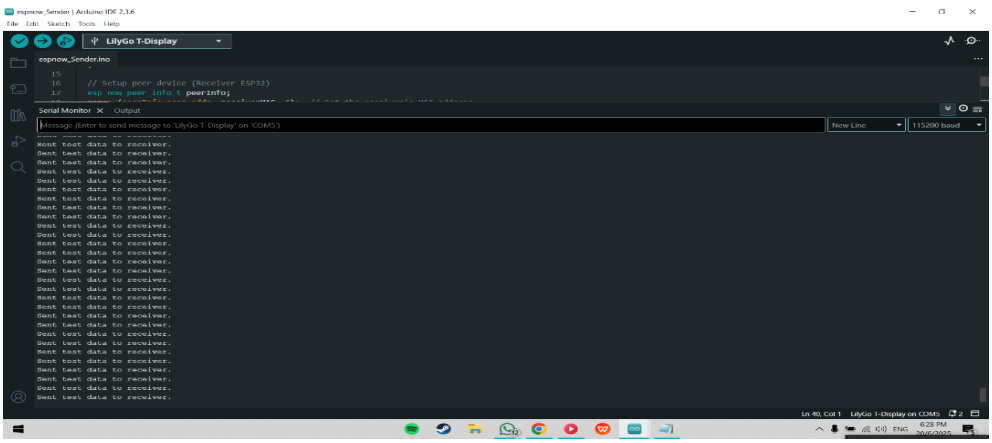

Figure 4. 52 Code snippet for configuring the buffer delay between RED and GREEN transitions.

iii. **Force All RED -** This button forces both ESP32 units to go into RED state immediately for emergency or manual override purposes. It sends a POST request to /forceAllRed with "enable": "true".

```
// === FORCE ALL RED ===
server.on("/forceAllRed", HTTP_POST, [](AsyncWebServerRequest *request){
  if (request->hasParam("enable", true)) {
    String enable = request->getParam("enable", true)->value();
    forceAllRed = (enable == "true");
    request->send(200, "text/plain", forceAllRed ? "All RED mode ENABLED" : "All RED mode DISABLED");
    Serial.println(forceAllRed ? "🔴 Forcing all RED" : "🟢 All RED override cleared");
  } else {
    request->send(400, "text/plain", "Missing parameter: enable");
  }
});
```

Figure 4. 53 Code snippet for forcing all ESP32 lights to RED via the /forceAllRed endpoint.

iv. **Emergency Shutdown -** Used in critical situations, this feature disables all LEDs and pauses the traffic system. It sends a POST request to /emergency with a passcode for security.

```
// ✅ Send emergency shutdown to ESP32 B via ESP-NOW
server.on("/emergency", HTTP_POST, [](AsyncWebServerRequest *request){
  if (request->hasParam("passcode", true)) {
    String code = request->getParam("passcode", true)->value();

    if (code == "1234") {
      emergencyShutdown = true;
      Serial.println("🚨 EMERGENCY MODE ACTIVATED (ESP32 A)");

      // Try notifying peer (ESP32 B) via ESP-NOW
      outgoingData.emergencyShutdown = true;
      esp_err_t result = esp_now_send(peerAddress, (uint8_t*)&outgoingData, sizeof(outgoingData));

      if (result == ESP_OK) {
        request->send(200, "text/plain", "Emergency shutdown successful");
      } else {
        request->send(500, "text/plain", "Failed to notify peer");
      }

    } else {
      request->send(403, "text/plain", "Invalid passcode");
    }
  } else {
    request->send(400, "text/plain", "Missing parameter: passcode");
  }
});
```

Figure 4. 54 Code snippet for triggering emergency shutdown using a secure POST request to /emergency.

v. **Restore System Power -** After an emergency shutdown, this button reactivates the ESP32 system by sending a POST request to /powerOn.

```
// === POWER ON ===
server.on("/powerOn", HTTP_POST, [](AsyncWebServerRequest *request){
  emergencyShutdown = false;
  request->send(200, "text/plain", "Power restored");
  Serial.println("✅ SYSTEM POWER RESTORED");
});
```

Figure 4. 55 Code snippet for restoring system power after an emergency shutdown.

4.7.2 Connection Page

Before accessing the traffic light system, users must first connect their mobile device to the ESP32's Soft Access Point (SoftAP) network. This is handled in the Connection Page of the Flutter app, which detects active Wi-Fi connections and validates the SSID prefix (e.g., "ESP32A").

The app uses the **connectivity_plus**, **network_info_plus**, and **permission_handler** packages to monitor Wi-Fi state and request necessary location permissions for Android devices.

Below is a simplified code snippet used to retrieve the current Wi-Fi name:

```
if (connectivityResult.contains(ConnectivityResult.wifi)) {
  final info = NetworkInfo();
  final ssid = await info.getWifiName();
  print("📶 Connected SSID: $ssid");

  if (ssid != null && ssid.toLowerCase().contains("esp32")) {
    print("✅ Connected to ESP32. Navigating...");
    await Future.delayed(const Duration(seconds: 1));
    if (!mounted) return;
    Navigator.pushReplacement(
      context,
      MaterialPageRoute(builder: (_) => const HomePage()),
    );
  } else {
    print("❌ SSID doesn't match ESP32.");
    setState(() {
      _connectionFailed = true;
      _statusMessage = "❌ Please connect to ESP32 Wi-Fi.";
      _isChecking = false;
    });
  }
```

Figure 4. 56 Code snippet used to retrieve the current Wi-Fi name

The app checks if the device is connected to the correct SoftAP (e.g., "ESP32A" or “ESP32B”) and then allows access to the home screen:

```
  final connectivityResult = await Connectivity().checkConnectivity();
  print("🟡 Connectivity result: $connectivityResult");

  if (connectivityResult.contains(ConnectivityResult.wifi)) {
    final info = NetworkInfo();
    final ssid = await info.getWifiName();
    print("📶 Connected SSID: $ssid");

    if (ssid != null && ssid.toLowerCase().contains("esp32")) {
      print("✅ Connected to ESP32. Navigating...");
      await Future.delayed(const Duration(seconds: 1));
      if (!mounted) return;
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (_) => const HomePage()),
      );
    } else {
      print("❌ SSID doesn't match ESP32.");
      setState(() {
        _connectionFailed = true;
        _statusMessage = "❌ Please connect to ESP32 Wi-Fi.";
        _isChecking = false;
      });
    }
  } else {
    print("❌ Not connected to Wi-Fi.");
    setState(() {
      _connectionFailed = true;
      _statusMessage = "❌ Not connected to any Wi-Fi.";
      _isChecking = false;
    });
```

Figure 4. 57 Code snippet checking for SoftAP connection before allowing access to the control interface.

```dart
Future<void> togglePowerOnAll() async {
  final espIPs = ["http://192.168.4.1", "http://192.168.4.2"];
  bool allRestored = true;

  for (final ip in espIPs) {
    try {
      final response = await http.post(Uri.parse('$ip/powerOn'));
      if (response.statusCode != 200) {
        allRestored = false;
      }
    } catch (e) {
      allRestored = false;
    }
  }

  setState(() {
    systemIsShutdown = !allRestored ? true : false;
  });

  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
      content: Text(allRestored ? "✅ Power restored" : "❌ Failed to power on all systems"),
    ),  // SnackBar
  );
}
```

Figure 4. 58 Code snippet for restoring system power after an emergency shutdown.

This validation ensures that control features are only available when the device is connected to the ESP32 network, preventing accidental usage on unrelated Wi-Fi connections.

CHAPTER 5: TESTING

5.1 Introduction

This chapter presents the testing and evaluation process of the developed **Portable Wireless Traffic Control System for Partial Road Closures**. The primary goal of the testing phase is to ensure that the system performs as intended in real-world scenarios, including communication between ESP32 units, accurate detection of vehicles, correct traffic light behavior, and responsiveness of the mobile application.

Testing is divided into two categories: **functional testing**, which focuses on the system's core features and behavior, and **non-functional testing**, which evaluates aspects such as performance, reliability, and user interaction. These tests help identify any weaknesses or limitations while validating the system's stability and effectiveness in controlling vehicle flow at a controlled access point.

5.2 System Testing

To verify the reliability and functionality of the traffic control system, a series of tests were conducted on both the hardware and software components. These tests were carried out in a real-world prototype environment, where two ESP32 units, equipped with ultrasonic sensors and traffic light modules, communicated using ESP-NOW while simultaneously being monitored and controlled through a Flutter-based mobile application. Testing was categorized into two main areas:

- **Functional Testing**: Focuses on validating that each feature of the system works as expected, including vehicle detection, traffic light transitions, inter-device coordination, and mobile app controls (such as emergency shutdown and mode switching).
- **Non-Functional Testing**: Focuses on evaluating performance metrics like communication delay, system responsiveness, reliability over extended periods, and user interaction experience through the mobile interface.

Each test scenario was designed to simulate realistic usage, including simultaneous vehicle detection, emergency overrides, and power resets. The results from these tests serve to demonstrate the system's readiness for practical deployment in low-traffic or partially closed road environments.

5.2.1 Functional Testing

Functional testing was carried out to validate the core behavior of the system. The following key functions were tested using real-time observation, serial monitoring, and the mobile application interface:

Table 5. 1 Functional Testing Results

Test Case	Description	Steps	Expected Result	Success	Fail
TC1	Vehicle Detection Triggers Countdown	1. Place object ≤ 7 cm 2. Ensure both units are RED 3. Watch countdown	ESP32 starts 10s countdown, then goes GREEN	✓	
TC2	No Simultaneous GREEN States	1. Make ESP32 A GREEN 2. Place object near B	ESP32 B stays RED until A fully completes	✓	

TC3	Coordinated Peer Triggering	1. Let A run countdown 2. Observe B	B starts countdown when A reaches 4s remaining	✓	
TC4	Emergency Shutdown from App	1. Tap “Emergency” in app	Both units turn off and logic halts	✓	
TC5	Mode Switching Changes Green Duration	1. Set mode to 30s in app	GREEN lasts at least 30s before switching	✓	
TC6	Force All RED from App	1. Tap “Force All RED” in app	Both systems move to RED after current cycle	✓	
TC7	Distance Sensor Accuracy	1. Move object to 7 cm 2. Read monitor	Distance triggers countdown around ≤ 7 cm	✓	
TC8	App Status Accuracy	1. Trigger countdown 2. Watch app	App shows correct state/countdown in real-time	✓	

5.2.2 Non-Functional Testing

Non-functional testing focuses on evaluating the performance, usability, and reliability of the system under various conditions. While the system's core logic was tested functionally, the following non-functional aspects were also assessed:

Table 5. 2 Non-Functional Testing Results

Test Type	Objective	Result Summary	Tested
A. Performance	Ensure both ESP32 units operate in sync and communicate efficiently.	ESP-NOW messages exchanged reliably every 500ms; countdowns and buffer delays executed with no noticeable lag.	✓
B. Usability	Determine if users can easily interact with the mobile app.	Brief tests with peers confirmed the interface is intuitive. Users easily controlled traffic lights and understood app functions.	✓
C. Reliability	Validate system stability during extended operation.	System ran for over an hour without failure. Repeated emergency triggers and mode switches handled without issues.	✓
D. Compatibility	Check app functionality across different Android devices.	Tested on multiple Android phones (Android 10+); all worked without display or connectivity problems.	✓

E. Maintainability	Assess ease of debugging and future maintenance.	Serial Monitor and debug messages allowed easy tracking of traffic states, countdowns, and logic flow, simplifying bug-fixing.	✓

5.3 Summary

This chapter detailed the testing procedures carried out to evaluate the overall performance and reliability of the system. Functional testing verified that all major features — including traffic light coordination, countdown display, and mobile control — operated as intended. Non-functional testing further confirmed that the system met key attributes such as usability, performance, and stability across different devices and conditions. All tests yielded successful results, indicating that the portable wireless traffic control system performs effectively and is ready for deployment in real-world scenarios.

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 Introduction

This chapter presents a summary of the overall project, highlighting the major accomplishments, encountered limitations, and suggestions for future enhancements. It provides a reflection on the development journey of the portable wireless traffic control system and evaluates how the system meets its intended objectives in real-world applications.

6.2 Achievements

The project successfully achieved its primary objective of designing and developing a **portable wireless traffic control system for partial road closures** using two ESP32 modules. The system demonstrated synchronized traffic light behaviour, real-time sensor-based decision-making, and remote manual control through a mobile application. The table below summarizes how each key objective was successfully met:

Table 6. 1 Summary of Project Objectives and Corresponding Achievements

Objective	Achievement
1. To develop a portable traffic control system with two temporary traffic signals that communicate wirelessly to synchronize operations for managing traffic during partial road closures.	Successfully developed two ESP32-based traffic light units that communicate wirelessly using ESP-NOW. The system operates fully without physical cabling and maintains strict coordination to avoid simultaneous green signals.
2. To design and implement a controlled communication protocol for reliable and consistent data exchange between the traffic lights, ensuring synchronization based on real-time traffic conditions.	Implemented a custom communication protocol using ESP-NOW, with structured data packets, peer state tracking, message IDs, and sensor-based triggers. System accurately responds to real-time vehicle detection and synchronizes countdowns between both units.

3. To create a mobile application that provides manual control capabilities, allowing traffic wardens to override automated operations and adapt to varying traffic situations.	A functional Flutter-based mobile app was developed. It allows users to view live system status and send override commands such as emergency shutdown, force all-red, and mode switching for different traffic patterns.

6.3 Problems Encountered

Throughout the development of the system, several technical and practical challenges were encountered. Although they were mostly resolved, they highlighted the limitations of the current prototype:

1. **ESP-NOW Message Timing Conflicts -** At shorter communication intervals (e.g., 500ms), the ESP32 units occasionally experienced delayed or missed messages, causing synchronization issues between the two traffic lights. This required fine-tuning of message intervals and the introduction of message IDs and debounce logic.

2. **Simultaneous Detection Conflicts** - During startup or when both sensors detected vehicles at the same time, both ESP32s initially attempted to claim the GREEN state. This was resolved by prioritizing ESP32 A in the logic, but the system still depends on proper timing for consistent coordination.

3. **Unstable Countdown Skipping -** There were cases where one ESP32 skipped its RED-to-GREEN countdown and went straight to GREEN. Fixes included adding trigger validation, hasReceivedTrigger flags, and debounce delays to ensure safe state transitions.

4. **Ultrasonic Sensor Interference -** In certain placements or environmental conditions, ultrasonic sensor readings fluctuated due to reflective surfaces or poor alignment. This was minimized by setting a valid distance range and using default fallback values.

5. **No Real-Time Cloud Connectivity -** While the system supports real-time monitoring via a local mobile app, it does not currently support remote cloud-based control or logging due to the limitations of SoftAP mode without internet routing.

6. **No Hardware Debouncing or Protection -** The system does not include hardware-level resistors, debounce circuits, or electrical protections. Long-term durability may be affected if deployed in harsh environments without proper shielding.

6.4 Future Work

To further enhance the system's usability, reliability, and deployment potential, the following improvements are suggested:

1. **Cloud-Based Remote Control and Monitoring -** Future versions can integrate internet connectivity to enable real-time traffic control and monitoring through a cloud platform, making it accessible beyond local Wi-Fi range.

2. **Solar Power Integration -** Adding solar panels with battery management would allow the system to operate independently outdoors, especially useful in remote or rural road closure scenarios.

3. **Support for Multi-Junction Traffic Control -** Expand the system to support more than two ESP32 units for managing larger or more complex traffic environments, ensuring synchronized control across multiple intersections.

6.5 Summary

This chapter presented a comprehensive overview of the project's results, including its main accomplishments, the challenges encountered, and possible areas for future improvement. The system successfully demonstrated a portable, wireless traffic control solution using ESP32 microcontrollers, ultrasonic sensors, and a mobile application for real-time monitoring and manual override. While the system met its primary objectives, several limitations were identified, such as synchronization conflicts, sensor reliability, and lack of internet connectivity. Nonetheless, the proposed enhancements outlined in the future work section show promising potential for expanding the system into a more robust, scalable, and field-ready solution.

References

1. Kumar, A., & Patel, S. (2023). A Deep Q-Learning Based Adaptive Traffic Light Control System for Urban Safety. Proceedings of the 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N–22), IEEE. https://doi.org/10.1109/ICAC3N56670.2022.10074123

2. Teja, P. P., Manasa, S., Saravanan, S., Vijaykumar, A., Deepika, S., & Prameela, M. (2024). Dynamic Traffic Management and Prioritization for Emergency Vehicles using IoT-enabled Density Control. 2024 International Conference on Signal Processing, Computation, Electronics, Power, and Telecommunication (IConSCEPT), IEEE. https://doi.org/10.1109/ICONSCept61884.2024.10627923

3. Nagarjuna, R. S., Madhu, J., & Sufiyanuddin, M. (2020). IoT-Enabled Smart Traffic System for Public and Emergency Mobility in Smart City. International Journal of Advanced Science and Technology, 29(4), 7463–7468. https://ieeexplore.ieee.org/abstract/document/9243489

4. Manasa, R. S., Mounica, P., & Sufiyanuddin, M. (2023). Smart Traffic Signal Management System. 2023 IEEE 8th International Conference for Convergence in Technology (I2CT), 112–117. https://ieeexplore.ieee.org/document/10126180

5. Raj, A. E., Bhargavi, R., Meghana, S., Anjali, S., & Teja, A. (2022). Smart Traffic Management System for Priority Vehicle Clearance using IoT. 2022 International Conference on Automation, Computing, and Renewable Systems (ICACRS), IEEE. https://ieeexplore.ieee.org/document/10029272

6. Kumar, G., & Bhatia, P. K. (2012). Impact of Agile Methodology on Software Development Process. International Journal of Computer Technology and Electronics Engineering (IJCTEE), 2(4), 46–50. https://www.researchgate.net/publication/255707851_Impact_of_Agile_Methodology_on_Software_Development_Process

7. Ali, A., Sameh, H. Developing a theoretical framework for enhancing green project approaches via Agile methodology. Sci Rep 14, 27786 (2024). https://doi.org/10.1038/s41598-024-78613-x