KINGS COLLEGE OF ENGINEERING

PUNALKULAM

Traffic Management System Using Iot with ESP32

Phase -2

Team Name :

D. Rajkumar

C. Vasikkaran

T. Rakesh

R.Yuvaraj

P. Pragathiswaran

Table of Contents

Traffic Management System with ESP32

Abstract

In the abstract, provide a brief summary of your traffic management system project. Describe its purpose, key features, and why it's important. This section should give readers a clear overview of what to expect in the report.

1. Introduction

In the introduction, give readers a background on the project. Explain the context and importance of traffic management and introduce the problem you aim to solve. State the objectives of your system.

2. Hardware Components

In this section, list and provide detailed descriptions of all the hardware components used in your traffic management system. Explain the role of each component, how it's integrated into the system, and its specifications. Include photos or diagrams if possible.

Introduction to ESP32:

The ESP32 is a low-cost, low-power, and highly integrated microcontroller from Espressif Systems.

It is designed for a wide range of applications, including IoT (Internet of Things) devices, home automation, and more.

Dual-Core Processor:

The ESP32 features a dual-core Tensilica LX6 microprocessor, which can be independently programmed.

Dual-core architecture allows multitasking and better performance for complex applications.

Wireless Connectivity:

ESP32 supports both Wi-Fi (802.11 b/g/n) and Bluetooth (Bluetooth Classic and BLE).

It's a versatile choice for wireless communication in IoT projects.

Rich Peripherals:

The ESP32 comes with a variety of built-in peripherals, including GPIO pins, UART, SPI, I2C, analog-to-digital converters (ADC), and more.

These peripherals make it easy to interface with sensors, displays, and other external devices.

Memory and Storage:

It typically has 520KB of SRAM, which is crucial for running applications and handling data.

ESP32 also supports external flash memory for program and data storage.

Low Power Consumption:

ESP32 is designed with power efficiency in mind. It offers different sleep modes to reduce power consumption.

It's suitable for battery-powered and energy-efficient applications.

Real-Time Operating System (RTOS):

ESP32 has an integrated FreeRTOS operating system, which can be useful for multitasking and real-time applications.

Development Environment:

It can be programmed using the Arduino IDE with the ESP32 board support package.

Espressif also provides its own development framework called ESP-IDF (IoT Development Framework).

Community Support:

The ESP32 has a large and active community of developers and enthusiasts.

This community provides extensive documentation, libraries, and support through forums and online resources.

Security Features:

ESP32 includes security features like secure boot, flash encryption, and support for secure connections (SSL/TLS).

This is crucial for IoT devices that handle sensitive data.

Wireless Protocols:

ESP32 supports a range of wireless protocols, including MQTT, HTTP, CoAP, and more.

It can be used to build IoT devices that communicate over the internet or local networks.

Applications:

ESP32 is used in a wide variety of applications, such as smart home devices, wearable technology, environmental monitoring, and industrial automation.

Cost-Effective:

It's cost-effective and readily available, making it a popular choice for hobbyists and professional projects alike.

Open-Source Hardware:

The ESP32's hardware design is open-source, allowing for flexibility in custom designs and modifications.

Upgradability:

Espressif has introduced variations of the ESP32 with additional features, such as the ESP32-S2 and ESP32-C3, for specific use cases.

The ESP32's versatility, low cost, and strong community support make it a preferred choice for a wide range of embedded and IoT projects. Its combination of wireless capabilities, rich peripherals, and power efficiency makes it a compelling microcontroller for various applications

3. Software Components

Describe the software components used to develop the system. Explain the software stack, including the Arduino IDE, libraries, and any other programming languages or frameworks. Discuss their importance in achieving the project's goals.

4. System Architecture

Present a high-level system architecture diagram that visually represents how different hardware and software components interact in your traffic management system. Explain the data flow between these components, detailing how data moves from sensors to the ESP32 and to the traffic lights.

5. Sensors and Data Collection

This section should delve into the specifics of the IR sensors used for vehicle detection. Describe how the sensors are positioned at intersections and provide insights into how data is collected, processed, and interpreted to detect vehicles.

6. Traffic Light Control

Explain the logic used to control traffic lights based on sensor data. Describe the different phases of the traffic lights, such as red, green, and yellow, and how servo motors are used to change the lights.

7. Web Interface

Detail the web interface that users interact with to monitor and control the traffic management system. Explain how users can start and stop the system, manually control traffic lights, and view real-time traffic status. If possible, include screenshots of the web interface.

8. Data Storage and Analysis (if applicable)

If your system collects and stores data, describe the structure of the database and any data analysis methods you employ. Explain how this data is used for traffic management or future improvements.

9. Challenges and Solutions

Discuss the challenges you encountered during the project, such as sensor accuracy, power management, and connectivity issues. Explain the solutions you implemented to overcome these challenges.

10. Future Enhancements

Share your vision for the future of the traffic management system. Describe potential improvements and enhancements, such as integration with smart city infrastructure, advanced traffic data analysis, and scalability for larger intersections.

Code :

```
#include <WiFi.h>

#include <WebServer.h>

#include <Wire.h>

#include <Adafruit_SSD1306.h>

#include <Adafruit_GFX.h>

// Replace with your network credentials
```

```
Const char* ssid = “Your_SSID”;
Const char* password = “Your_PASSWORD”;
WebServer server(80);
// Initialize OLED display
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
Void setup() {
// Connect to Wi-Fi
WiFi.begin(ssid, password);
While (WiFi.status() != WL_CONNECTED) {
Delay(1000);
Serial.println(“Connecting to WiFi…”);
}
// Initialize OLED display
If(!display.begin(SSD1306_I2C_ADDRESS, OLED_RESET)) {
Serial.println(F(“SSD1306 allocation failed”));
For(;;);
}
Display.display();
Delay(2000);
// Web server routes
Server.on(“/”, HTTP_GET, handleRoot);
Server.begin();
Serial.println(“HTTP server started”);
}
```

```
Void loop() {
Server.handleClient();
}
Void handleRoot() {
Display.clearDisplay();
Display.setTextSize(1);
Display.setTextColor(SSD1306_WHITE);
Display.setCursor(0, 0);
Display.println(“Welcome to Traffic”);
Display.println(“Management System”);
Display.display();
Server.send(200, “text/html”, “Welcome to Traffic Management System”);
}
```

11. Conclusion

Summarize the key achievements of the project. Reflect on the significance of the traffic management system and highlight any lessons learned during its development.