AIR QUALITY MONITORING SYSTEM

IIOT Lab ARA-255
Project Report

Submitted To:
Dr. Khyati Chopra

Submitted By:
Dhruv Kaushik (067)
Aryan Shukla (078)
Kumar Utkarsh (080)
Shivam Punia (082)
Vaibhav Bansal (094)
AR – B2 (SEM – III)

EXECUTIVE SUMMARY

ABSTRACT

This project focuses on the design and development of an IoT-based air quality monitoring system that leverages modern sensor technologies and cloud connectivity to provide real-time environmental data. The system is powered by the ESP32 microcontroller, known for its low power consumption, Wi-Fi capabilities, and computational power, making it an ideal choice for IoT applications. The project integrates a set of five environmental sensors to measure critical air quality parameters including PM2.5 (Particulate Matter), CO (Carbon Monoxide), NO2 (Nitrogen Dioxide), TVOC (Total Volatile Organic Compounds), and Temperature & Humidity.

SOFTWARE AND CLOUD INTEGRATION

The system uses the Arduino IDE for development, utilizing various libraries for sensor communication and the Thing Speak IoT platform for remote data monitoring. The system sends sensor data to Thing Speak every 15 seconds, where users can track air quality trends over time. The display is updated every 6 seconds, providing real-time feedback to users. This system not only provides valuable data on air quality in real-time but also offers insights for improving indoor and outdoor air conditions, making it a powerful tool for environmental monitoring.

AQI IMPLEMENTATION

This IoT-based air quality monitoring system calculates the Air Quality Index (AQI) using sensor readings for PM2.5, CO and NO2 levels. The system uses a set of predefined breakpoints for each pollutant to convert raw sensor data into AQI values, which range from 0 to 500. For PM2.5, MQ-7 CO sensor, and NO2 sensor, the AQI is computed using a formula that scales the sensor concentration to the corresponding AQI range. The highest individual AQI value from the monitored pollutants determines the overall AQI of the environment.

KEY COMPONENTS

- Microcontroller: The heart of the system is ESP32, chosen for its dual-core processing power and built-in Wi-Fi functionality, which allows seamless connectivity and real-time data transmission.

- Particulate Matter Sensor: This sensor measures particulate matter, specifically PM2.5, which is a critical air quality indicator linked to respiratory diseases.

- Carbon Monoxide Sensor: This sensor detects carbon monoxide, a colourless and odourless gas that is harmful at high concentrations, particularly in urban and industrial environments.

- Nitrogen Dioxide Sensor: NO2 is a harmful gas primarily generated from combustion processes, making this sensor essential for detecting air pollution from vehicle emissions and industrial activities.

- TVOC Sensor: Volatile Organic Compounds (VOCs) are pollutants emitted by household products, paints, and solvents, and this sensor helps monitor their concentration in indoor environments.

- Temperature and Humidity Sensor: This sensor provides temperature and humidity data, both important factors in determining the overall environmental conditions.

- OLED Display: Provides real-time feedback to the user, displaying AQI values, sensor data, and system status.

- Battery & Boost: The system is powered by a Li-Po battery, providing portability for outdoor and off-grid applications. A boost converter steps up the battery voltage from 3.7V to 5V, ensuring a stable power supply to both the ESP32 and the sensors.

- Custom PCB: A custom-designed PCB that integrates all the components into a compact, efficient, and reliable setup.

TABLE OF CONTENTS

INTRODUCTION

PROJECT BACKGROUND

Air quality has a direct impact on public health, with poor air quality contributing to respiratory diseases, cardiovascular diseases, and other serious health issues. According to the World Health Organization (WHO), air pollution is responsible for millions of premature deaths globally, particularly due to fine particulate matter (PM2.5), nitrogen dioxide (NO2), and carbon monoxide (CO).

Traditional air quality monitoring stations, though effective, are typically expensive, stationary, and often limited to urban areas. They also require substantial infrastructure and maintenance. However, advancements in Internet of Things (IoT) technologies now allow for the development of compact, cost-effective, and real-time air quality monitoring systems that can be deployed in various locations, including rural or remote areas, to offer real-time data on air pollution.

This project is focused on creating a portable, IoT-based air quality monitoring system using the ESP32 microcontroller. The ESP32, combined with environmental sensors, enables efficient measurement and transmission of air quality parameters to a cloud platform. This solution provides real-time monitoring, enabling immediate action to be taken to mitigate pollution.

OBJECTIVES

1. Real-time Monitoring: To create a real-time environmental monitoring system that tracks and displays air quality parameters such as PM2.5, CO, NO2, TVOC, temperature, and humidity.

2. Portable Design: To design a compact and portable system powered by a rechargeable 3.7V Li-Po battery, ensuring the system can be used in outdoor or off-grid environments.

3. Cloud Integration: To integrate the system with the Thing Speak cloud platform for storing and visualizing data, allowing users to track air quality remotely.
4. AQI Calculation: To compute the Air Quality Index (AQI) based on sensor data and display it on an OLED screen, offering immediate feedback to users about the air quality.

5. Cost-effective Solution: To develop a low-cost, scalable, and easy-to-assemble system using readily available components.

SCOPE

The scope of this project is to develop a portable IoT-based air quality monitoring system that measures key environmental parameters including PM2.5, CO, NO2, TVOCs and Temperature & Humidity. The system will calculate the Air Quality Index (AQI) based on sensor data, providing real-time insights into air quality. The key components of this project will include an ESP32 microcontroller for wireless communication, a range of environmental sensors, an OLED display for showing live data, and integration with the Thing Speak cloud platform for remote data monitoring.

Functionality:

The system will calculate AQI values based on real-time readings from the integrated sensors and display this data on an OLED screen. Every 6 seconds, the system will update the display, ensuring that users receive the most current air quality information. The data will also be transmitted via Wi-Fi to the Thing Speak cloud platform, allowing users to monitor air quality remotely through web or mobile applications. The system will be portable, powered by a Li-Po battery, and equipped with a Boost Converter to ensure stable voltage for the sensors and microcontroller

Technologies:

This project will utilize the ESP32 microcontroller, which provides the Wi-Fi capability needed to transmit data to Thing Speak.
The following sensors will be used for environmental monitoring:

- WINSEN ZPH02 for PM2.5 measurement.
- MQ-7 for CO detection.
- DFRobot MEMES NO2 for NO2 monitoring.
- WINSEN ZP07-MP503 for TVOC detection.
- DHT22 for temperature and humidity measurements.

The system will also incorporate an OLED Display to show real-time data and a 3.7V 1500mAh Li-Po Battery to ensure portability. Data will be uploaded to the Thing Speak cloud platform for remote monitoring and analysis.

Limitations:

The project will not include advanced weather forecasting or air quality prediction capabilities. It will focus solely on real-time monitoring of the aforementioned pollutants and their respective AQI values. The system is also designed for indoor or portable outdoor use and may not be optimized for industrial-scale environmental monitoring. The project will not support multi-device synchronization or additional complex sensor integration beyond the five sensors specified.

Deliverables:

The project will result in a fully functional prototype of the IoT-based air quality monitoring system, including:

- A working hardware setup with all necessary components integrated into a compact, custom-designed PCB.
- AQI calculation, real-time display updates, cloud data transmission and an interface for remote monitoring via Thing Speak.
- Detailed documentation, including the circuit design, system architecture, and software code, along with a user manual for the system.

LITERATURE REVIEW

The development of the IoT-based Air Quality Monitoring System was deeply inspired by a combination of existing literature, practical projects, and sensor datasheets. These resources guided the design, sensor selection, and implementation of both hardware and software components of the system. By reviewing relevant projects and documentation, we were able to adapt best practices and incorporate modern IoT technologies to create an effective and accurate air quality monitoring solution.

The journey began with a reference to a detailed DIY Air Quality Monitor Project shared on HowToMechatronics. This project showcased the use of various environmental sensors to measure gases such as CO2, NH3, and alcohols using the MQ135 sensor, along with particulate matter sensors like the Sharp GP2Y1010AU0F for PM2.5. The integration of a DHT11 sensor for temperature and humidity monitoring and the use of the Arduino platform for easy prototyping were key takeaways. This resource introduced the concept of combining multiple sensor types into a single system for comprehensive air quality monitoring. It provided foundational ideas for sensor integration and measurement techniques, setting the stage for the project's sensor selection. (Dejan, n.d.)

In parallel with exploring hardware options, it became clear that understanding the regulatory framework for air quality was essential. The Central Pollution Control Board (CPCB) in India provided an important document titled National Air Quality Index (AQI), which established standardized methods for calculating and interpreting air quality based on the concentrations of pollutants like PM2.5, PM10, CO, NO2, and Ozone. The AQI guidelines from this document helped shape our approach to pollutant measurement and classification. The CPCB's AQI scale, which categorizes air quality from “Good” to “Hazardous,” was instrumental in defining the thresholds used to categorize sensor data into an actionable format. This document also provided insight into health impacts at various AQI levels, which helped refine our focus on the most relevant pollutants for public health monitoring. (CPCB, 2014-15)

CPCB AQI classification

AQI Range	AQ Category	Health Implications
0 - 50	Good	Air quality is considered satisfactory, and air pollution poses little or no risk.
51 - 100	Satisfactory	Air quality is acceptable; however, there may be some health concerns for sensitive individuals (e.g., people with respiratory conditions).
101 – 200	Moderate	Health effects may be moderate for some individuals, especially those with respiratory conditions, the elderly, and children.
201 - 300	Poor	Health effects may be significant for individuals with respiratory problems, children, and sensitive groups (e.g., people with asthma).
301 – 400	Very Poor	Health effects are more serious for the general population, with significant risk to people with respiratory conditions and the elderly.
401 - 500	Severe	Health alert: everyone may experience serious health effects, with major health risks for the general population, especially those with pre-existing health conditions.

Source: Central Pollution Control Board (CPCB) AQI Guidelines

Following these foundational studies, our focus shifted to IoT-based systems, specifically those using the ESP32microcontroller for cloud-based monitoring. One such resource was the IoT-Based Air Quality Monitoring System project shared by CircuitDigest. This project demonstrated the use of the ESP32 to interface with environmental sensors such as PM2.5, PM10, and CO sensors, transmitting the collected data to the ThingSpeak cloud platform for real-time visualization. The decision to use the ESP32 was influenced by its ability to connect wirelessly to the internet, making it ideal for remote monitoring applications. The cloud integration aspect allowed us to envision how sensor data could be uploaded and accessed by users for analysis and visualization. The ThingSpeak platform also inspired our own approach to cloud connectivity, leading to the use of Wi-Fi for communication between the ESP32 and the cloud, enabling us to visualize real-time data on a web interface. (Chouddhary, 2022)

Another relevant project was shared on Hackster.io, which featured a similar ESP32-based system for air quality monitoring. This project provided more specific details about integrating the MQ-7 sensor for CO detection and a PM2.5sensor for particulate matter. The project also highlighted the use of an OLED display to visualize sensor data locally, which was crucial for providing immediate feedback to users without needing to access the cloud. This served as a valuable reference for incorporating both local and remote data visualization into our own system. The dual-display approach—local via OLED and remote via ThingSpeak—was adopted in our design to balance ease of access and comprehensive data presentation. (Mark, 2022)

The datasheets for each sensor played a pivotal role in selecting the right components and ensuring their proper integration within the air quality monitoring system. They provided critical details on sensor functionality, operating ranges, calibration methods, and communication protocols, allowing us to optimize performance and accuracy.

1. DHT22 (AM2302) – Temperature and Humidity Sensor: The DHT22 was selected for its precision in measuring temperature and humidity across a wide range of environmental conditions. The DHT22 datasheet provided key information on its pinout, operating voltage (3.3V to 6V), and the digital data protocol, ensuring its seamless integration with the ESP32. It also helped us understand the response time and accuracy of the sensor for reliable environmental monitoring (DHT22 Datasheet). (Aosong Electronics Co.)

2. MQ7 – Carbon Monoxide (CO) Sensor: The MQ7 sensor, chosen for detecting carbon monoxide (CO) levels, provided valuable insights into its operating characteristics, such as calibration instructions and heating cycle requirements. The MQ7 datasheet outlined the sensor's response times, sensitivity, and the necessary setup for accurate CO detection. These details were crucial during the calibration phase to ensure that the MQ7 provided reliable CO measurements (MQ7 Datasheet). (HANWEI ELECTRONICS CO .)

3. ZPH02 – PM2.5 Sensor: The sensor was selected for particulate matter detection, specifically PM2.5. This sensor employs laser scattering technology for highly accurate measurements of fine particulate concentrations. The ZPH02 datasheet provided detailed instructions on interpreting the analog output and converting it to PM2.5 concentration in micrograms per cubic meter ($\mu g/m^3$). This was crucial for integrating the sensor with the ESP32 for real-

time air quality data (ZPH02 Datasheet). (Zhengzhou Winsen Electronics Technology Co., 2018)

4. MEMS NO2 Sensor: For detecting nitrogen dioxide (NO2) levels, the MEMS NO2 sensor was chosen for its compact size and high sensitivity. The MEMS NO2 datasheet provided essential information on sensor calibration, response time, and sensitivity to NO2 gas, ensuring accurate readings of air pollution levels. The datasheet's details on operating conditions, power consumption, and data output were critical for proper sensor integration (MEMS NO2 Datasheet). (DFRobot, n.d.)

5. TVOC Sensor (ZP07-MP503): The TVOC sensor, based on the ZP07-MP503 model, was used for detecting volatile organic compounds in the air. The ZP07-MP503 datasheet outlined the sensor's operating characteristics, including sensitivity and response time to a wide range of TVOCs. This information allowed us to properly configure the sensor and incorporate it into the overall system for effective air quality monitoring (ZP07-MP503 Datasheet). (Zhengzhou Winsen Electronics Technology Co., robu.in, 2014)

6. ESP32 Microcontroller: The ESP32 was selected for its powerful processing capabilities and built-in Wi-Fi/Bluetooth functionality, which allowed the system to upload sensor data to the cloud. The ESP32 datasheet provided comprehensive details on pin configurations, processing power, available communication protocols (I2C, UART), and power requirements, ensuring that it could handle the sensors' data and manage cloud communication effectively (ESP32 Datasheet). (Systems, 2023)

By referencing these datasheets, we were able to ensure proper sensor calibration, optimize integration with the ESP32, and validate that all components would operate efficiently within the desired measurement ranges. These technical documents were instrumental in ensuring that the system could deliver accurate and reliable air quality readings.

METHODOLOGY ADOPTED

SYSTEM DESIGN

The system design is a crucial step in ensuring the functionality, portability, and efficiency of the entire air quality monitoring setup. The system consists of a central processing unit (the ESP32 microcontroller) interfacing with multiple environmental sensors to collect real-time data on air quality parameters.

The key design features of the system include:

- Power Supply and Portability: To ensure portability, the system is powered by a Li-Po battery and utilizes Boost Converter to step up the voltage to 5V for the ESP32 and sensors. This setup ensures that the system is independent of a direct power source and can be used in various settings, such as outdoor environments or remote locations.

- Real-Time Data Collection and Transmission: The sensors collect data at a defined interval (every 2 seconds). This real-time data is sent to a cloud platform (Thing Speak) for remote monitoring and storage. This ensures that users can track the air quality parameters over time, even if they are not physically present near the monitoring system.

- User Interface: The OLED display provides a clear and concise readout of the current air quality conditions. This includes parameters such as PM2.5, CO, NO2, TVOC, and Temperature & Humidity readings, along with the calculated AQI (Air Quality Index).

HARDWARE SELECTION

Selecting the right hardware components is critical to the performance and cost-effectiveness of the system. The major components selected for the system include:

- ESP32 Microcontroller: The ESP32 was chosen due to its powerful dual-core processor, built-in Wi-Fi capabilities, and low power consumption. The microcontroller handles the sensor data processing, communicates with the cloud server (Thing Speak), and drives the OLED display for real-time output. The onboard Wi-Fi Functionality allows seamless cloud communication, a crucial feature for IoT-based systems. It is also highly flexible in terms of interfacing with various sensors, making it an ideal choice for this project.

- WINSEN ZPH02 (PM2.5 Sensor): This sensor uses laser scattering to measure particulate matter in the air, particularly PM2.5 particles. PM2.5 is a significant contributor to air pollution and is linked to respiratory diseases. The WINSEN ZPH02 is widely recognized for its accuracy and efficiency in measuring fine particulate matter, which is why it was selected for this project.

- MQ-7 Carbon Monoxide Sensor: The MQ-7 gas sensor is highly sensitive to carbon monoxide (CO), a colourless and odourless gas that is toxic in high concentrations. It is especially useful in monitoring indoor air quality, where CO emissions are common due to combustion processes such as gas stoves, heaters, and vehicle exhaust. The MQ-7 sensor was chosen due to its ability to accurately detect CO at low concentrations.

- DFRobot MEMES NO2 Sensor: NO2 is a significant pollutant, especially in urban and industrial settings, where it is a byproduct of combustion engines and power plants. The MEMES NO2 sensor was selected for its high sensitivity and low power consumption, making it ideal for real-time environmental monitoring.

- WINSEN ZP07-MP503 (TVOC Sensor): TVOCs are a group of chemicals emitted by paints, solvents, and household cleaners. Long-term exposure to high levels of VOCs can cause respiratory problems and other health issues. The ZP07-MP503 sensor provides reliable and real-time measurements of TVOCs in indoor and outdoor environments, making it an essential component for this system.

- DHT22 Temperature and Humidity Sensor: The DHT22 sensor is widely used in environmental monitoring systems due to its high accuracy and stability in measuring temperature and humidity. These factors directly influence the quality of the air, as higher temperatures can increase the concentration of pollutants in the air, and humidity can affect sensor readings. Therefore, having accurate temperature and humidity data is essential for comprehensive air quality monitoring.

- MT3608 Boost Converter: This Boost Converter is used to step up the 3.7V battery voltage to 5V, providing a stable power supply for the ESP32 and all the sensors. This ensures that the system operates reliably, even when battery levels fluctuate.

SOFTWARE DEVELOPMENT

The software for the air quality monitoring system was developed using the Arduino IDE, a platform well-suited for prototyping and developing software for microcontrollers like ESP32. The software can be broken down into the following key components:

1. Libraries Used:
    - Wire: Used for I2C communication with the OLED display (Adafruit SSD1306).
    - Adafruit_GFX: Provides graphics functions like basic text and shape rendering for the OLED display.
    - Adafruit_SSD1306: This library is specifically for the SSD1306 OLED display, handling text and image rendering.
    - DHT: Used to interface with the DHT22 sensor for Temperature and Humidity measurements.
    - MQ7: This library is used to interface with the MQ7 Carbon Monoxide (CO) sensor, which is used for detecting CO levels.
    - HardwareSerial: The library is used to set up an additional serial port to communicate with the PM2.5 sensor.

2. Sensor Initialization: The sensors are initialized to ensure they are ready to send data. Specific libraries were used to interface with each sensor, such as the Adafruit DHT library for the DHT22 sensor and the MQ-7 library for the carbon monoxide sensor.

3. Data Collection: Every 2 seconds, the sensor data is collected. The system reads the environmental data such as PM2.5, CO, NO2, TVOC, and temperature and humidity. The PM2.5 sensor output is converted from analog to digital form, while the gas sensors use specific formulae to measure gas concentrations.

4. AQI Calculation: The Air Quality Index (AQI) is calculated based on the collected data from the PM2.5, CO, and NO2 sensors. The calculation follows established standards, which converts pollutant concentrations into a standardized index ranging from 0 (good air quality) to 500 (hazardous air quality).

5. Cloud Integration: The system uses the ThingSpeak cloud platform to store and visualize sensor data. The ESP32 communicates with ThingSpeak over Wi-Fi, uploading the real-time data every 15 seconds. The ThingSpeak API allows the system to easily interact with the cloud and visualize the data in graphical formats such as line charts or gauges.

6. Display Update: The OLED display is updated every 6 seconds with the latest AQI values, as well as the real-time sensor readings for PM2.5, CO, NO2, and temperature/humidity.

SYSTEM INTEGRATION

System integration for this IoT-based air quality monitoring system involves combining all hardware and software components to function as a unified system. The ESP32 microcontroller serves as the central unit, integrating data from the five environmental sensors: WINSEN ZPH02 for PM2.5, MQ-7 for Carbon Monoxide, DFRobot MEMS NO2 for Nitrogen Dioxide, WINSEN ZP07-MP503 for TVOCs, and the DHT-22 (AM2302) for temperature and humidity. These sensors interface with the ESP32 using a combination of analog and digital outputs, which are processed to compute the Air Quality Index (AQI). The AQI values are displayed on a 0.96-inch OLED screen every 6 seconds, providing real-time feedback to users. Additionally, the data is transmitted via Wi-Fi to the ThingSpeak cloud platform, allowing for remote monitoring.

The power system is integrated using a 3.7V Li-Po battery, with a Boost Converter stepping up the voltage to 5V to power the ESP32 and sensors. The Arduino IDE is used for programming the microcontroller, managing sensor readings, AQI calculations, and cloud integration. The system was thoroughly tested to ensure that the sensors communicate correctly with the ESP32, the AQI is calculated accurately, and the data is reliably displayed and uploaded to ThingSpeak. This integration ensures the system operates efficiently, offering portable, real-time air quality monitoring with both local and remote access capabilities.

PROTOTYPING AND PCB DESIGNING

The development of the custom PCB for the IoT-based Air Quality Monitoring System was a critical step in ensuring the system's compactness, modularity, and efficient operation. Before starting the PCB design, we used a breadboard prototype to validate the system's basic functionality. This allowed us to test the interactions between components like the ESP32, sensors (PM2.5, CO, NO2, TVOCs, DHT22), and other peripherals to ensure proper sensor communication and accurate data readings. Once the breadboard prototype confirmed the design's viability, we proceeded with the custom PCB development. The PCB design process was carried out using KiCAD, involving footprint and symbol creation, design validation, PCB layout, ordering, and assembly.

1. Creating Footprints and Symbols:
   The first step in designing the PCB was creating the footprints and symbols for each component used in the system. Footprints are the physical representations of components on the PCB, including their pin configurations, dimensions, and pads, while symbols represent these components in the schematic diagram. Carefully referenced the datasheets for each component, including the ESP32, and sensors to create accurate footprints and symbols that would be used in the schematic design. Special attention was given to the pinout configuration, component orientation, and clearance requirements to ensure that the components would fit properly on the board and that the electrical connections were correctly represented.

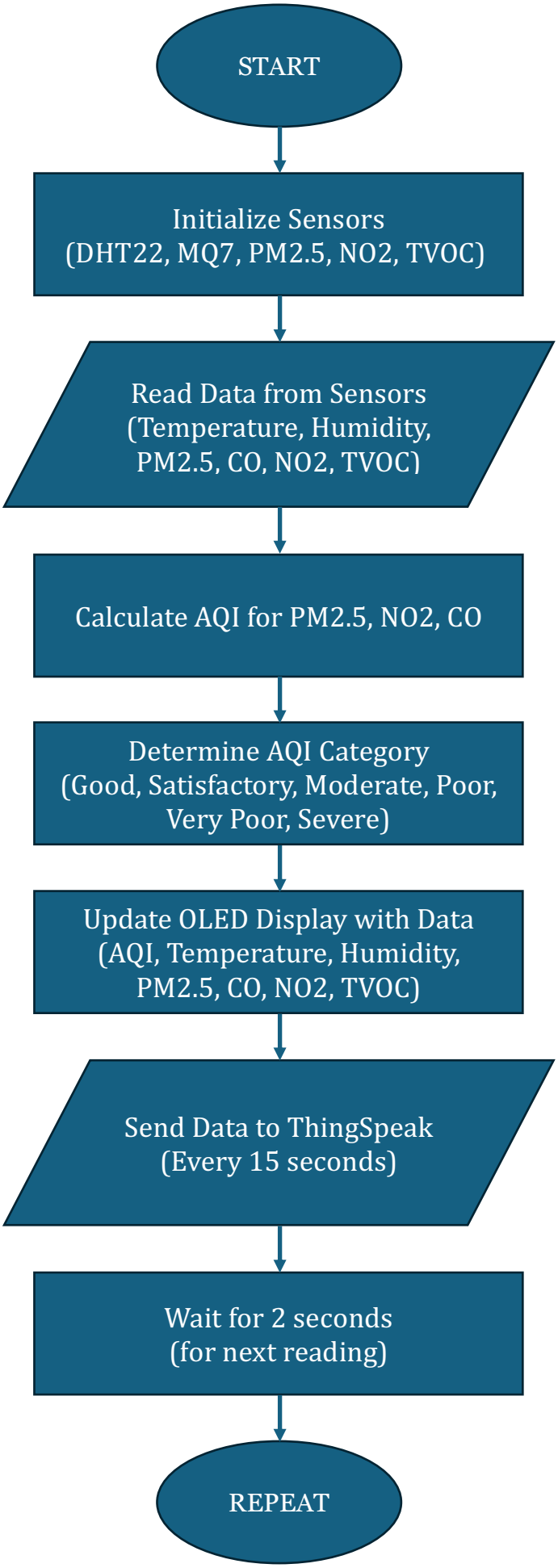
2. Design Rechecks and Validation:
   Once the footprints and symbols were created, thorough design rechecks were conducted to ensure that everything was accurate and aligned with the component specifications. This involved reviewing the footprints against the datasheets to verify that the pad sizes, pin configurations, and dimensions were correct. Additionally, running a Design Rule Check (DRC) to ensure that the layout adhered to manufacturing constraints, such as trace width, pad size, and spacing. An Electrical Rule Check (ERC) was also performed to verify that all connections in the schematic were correct and that there were no unconnected pins or floating nets. These checks ensured that the PCB design was free of errors before moving to the next stage.

3. PCB Layout and Routing:
With the schematic validated, moved on to the PCB layout phase, where we placed the components and routed the traces. The placement of components was done with careful consideration of factors like signal integrity, power distribution, and thermal management. The ESP32 microcontroller was placed in such a way to make it accessible via Micro USB, also minimize the trace lengths, and the sensors were positioned in areas that minimized interference from high-power components. The routing of the traces was optimized to reduce noise and improve signal quality, with power traces being wider to handle the current from the 5V supply. The power distribution network was designed using a ground plane to ensure a stable voltage supply, and used a boost converter (MT3608) to step up the 3.7V battery to 5V for the sensors and the microcontroller.

4. Ordering the PCB:
Once the PCB layout was completed and validated, we generated the Gerber files, which are used by PCB manufacturers to fabricate the physical board. The Gerber files were carefully checked to ensure that all layers, including copper traces, solder mask, and drill holes, were correctly represented. These files were then sent to a professional PCB manufacturer for fabrication. Unlike home fabrication, the PCB was ordered from a manufacturing service, ensuring higher precision and reliability.

5. Assembly and Testing:
After receiving the fabricated PCB, the next step was assembly, where I soldered the components onto the board. Berg Strips were used to mount the components, allowing for easy removal and reusability for future iterations or prototyping. This modular approach not only facilitated component replacement but also made the system adaptable for future upgrades. After assembly, I conducted thorough functional testing to ensure that all components were correctly connected and that the system operated as expected. This involved verifying the sensor readings, power supply, and cloud connectivity, as well as ensuring that the OLED display updated in real time with accurate AQI values.

FLOW CHART


START
Initialize Sensors
(DHT22, MQ7, PM2.5, NO2, TVOC)
Read Data from Sensors
(Temperature, Humidity,
PM2.5, CO, NO2, TVOC)
Calculate AQI for PM2.5, NO2, CO
Determine AQI Category
(Good, Satisfactory, Moderate, Poor,
Very Poor, Severe)
Update OLED Display with Data
(AQI, Temperature, Humidity,
PM2.5, CO, NO2, TVOC)
Send Data to ThingSpeak
(Every 15 seconds)
Wait for 2 seconds
(for next reading)
REPEAT

SOURCE CODE

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include "DHT.h"
#include "MQ7.h"
#include <HardwareSerial.h>
#include <WiFi.h>            // Include WiFi library for ESP32
#include <ThingSpeak.h>      // Include ThingSpeak library

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET -1
#define DHTPIN 4 // DHT sensor pin
#define DHTTYPE DHT22 // DHT 22 (AM2302)
#define A_PIN 2 // MQ7 pin
#define VOLTAGE 3.3 // MQ7 voltage
#define SENSOR_PIN_NO2 33 // NO2 sensor pin
#define SENSOR_PIN_PM25 16 // PM2.5 sensor pin (Serial RX)
#define MAX_VOC_PPM 100.0 // Max VOC PPM
#define MOLECULAR_WEIGHT 110 // Molecular weight for VOC

// ThingSpeak credentials
const char* ssid = "Your WiFi ID";     // WiFi SSID
const char* password = "Your WiFi Password";  // WiFi password
const long channelID = 2697383;         // Replace with your actual ThingSpeak
channel ID
const char* writeAPIKey = "RAYZJW1K4FBNIVP6"; // ThingSpeak Write API Key

DHT dht(DHTPIN, DHTTYPE);
MQ7 mq7(A_PIN, VOLTAGE);
HardwareSerial mySerial(1);

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

float temperature, humidity, vocPPM, no2PPM, pm25Concentration, mq7PPM;

// AQI Calculation Constants
const float NO2_CONVERSION = 1885.0; // PPM to µg/m³ for NO2
const float CO_CONVERSION = 1.131;   // PPM to mg/m³ for CO

// AQI Breakpoints
const int PM25_BREAKPOINTS[6][4] = {{0, 30, 0, 50}, {31, 60, 51, 100}, {61, 90,
101, 200}, {91, 120, 201, 300}, {121, 250, 301, 400}, {250, 500, 401, 500}};
const int NO2_BREAKPOINTS[6][4] = {{0, 40, 0, 50}, {41, 80, 51, 100}, {81, 180,
101, 200}, {181, 280, 201, 300}, {281, 400, 301, 400}, {400, 500, 401, 500}};
```

```
const int CO_BREAKPOINTS[6][4] = {{0, 1.0, 0, 50}, {1.1, 2.0, 51, 100}, {2.1, 10.0,
101, 200}, {10.1, 17.0, 201, 300}, {17.1, 34.0, 301, 400}, {34.1, 500.0, 401,
500}};

unsigned long lastTime = 0; // Store last time data was sent to ThingSpeak
unsigned long lastSensorRead = 0; // Store last time the sensors were read
unsigned long sensorInterval = 2000; // 2 seconds between sensor reads
unsigned long sendInterval = 30000; // 30 seconds to send data to ThingSpeak

WiFiClient client;  // WiFiClient to handle communication with ThingSpeak

void setup() {
    Serial.begin(115200);
    mySerial.begin(9600, SERIAL_8N1, SENSOR_PIN_PM25);
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    dht.begin();
    pinMode(A_PIN, INPUT);
    analogReadResolution(12);
    mq7.calibrate(); // Calibrates MQ7
    Serial.println("Calibration done!");
    display.display();
    delay(2000);

    // Connect to WiFi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi...");
    }
    Serial.println("Connected to WiFi");

    // Initialize ThingSpeak
    ThingSpeak.begin(client);  // Pass the WiFi client object to ThingSpeak
}

void readSensors() {
    // Read DHT sensor
    humidity = dht.readHumidity();
    temperature = dht.readTemperature();

    // Read VOC sensor
    int sensorValue = analogRead(A_PIN);
    float voltage = (sensorValue / 4095.0) * 3.3;
    vocPPM = (voltage / 3.3) * MAX_VOC_PPM;

    // Read NO2 sensor
    int no2SensorValue = analogRead(SENSOR_PIN_NO2);
    no2PPM = (no2SensorValue / 4095.0) * 10.0; // Convert to ppm

    // Read PM2.5 sensor
    if (mySerial.available() >= 9) {
        byte data[9];
        for (int i = 0; i < 9; i++) {
```

```
            data[i] = mySerial.read();
        }
        if (data[0] == 0xFF && data[1] == 0x18) {
            byte lowPulseRateInt = data[3];
            byte lowPulseRateDec = data[4];
            float dutyRatio = lowPulseRateInt + (lowPulseRateDec / 100.0);
            pm25Concentration = 1000 * (dutyRatio / 100.0); // PM2.5 in µg/m³
        }
    }

    // Read MQ7 data
    mq7PPM = mq7.readPpm();

    // Print sensor data to Serial Monitor
    Serial.print("Temperature: ");
    Serial.print(temperature);
    Serial.print(" °C, Humidity: ");
    Serial.print(humidity);
    Serial.print(" %, TVOC: ");
    Serial.print(vocPPM);
    Serial.print(" ppm, NO2: ");
    Serial.print(no2PPM);
    Serial.print(" ppm, PM2.5: ");
    Serial.print(pm25Concentration);
    Serial.print(" µg/m³, CO: ");
    Serial.print(mq7PPM);
    Serial.println(" ppm");
}

int calculateAQI(float concentration, const int breakpoints[][4]) {
    for (int i = 0; i < 6; i++) {
        if (concentration >= breakpoints[i][0] && concentration <=
breakpoints[i][1]) {
            float IHI = breakpoints[i][3]; // Upper AQI value corresponding to BHI
            float ILO = breakpoints[i][2]; // Lower AQI value corresponding to BLO
            if (ILO > 50) ILO--; // Adjust ILO if greater than 50
            float BHI = breakpoints[i][1]; // Breakpoint concentration greater
            float BLO = breakpoints[i][0]; // Breakpoint concentration smaller
            return static_cast<int>((((IHI - ILO) / (BHI - BLO)) * (concentration -
BLO)) + ILO);
        }
    }
    return 0; // Default case if concentration is out of range
}

void calculateAndDisplayAQI(int &overallAQI, const char* &aqiCategory) {
    // Calculate AQI values
    int aqiPM25 = calculateAQI(pm25Concentration, PM25_BREAKPOINTS);
    int aqiNO2 = calculateAQI(no2PPM, NO2_BREAKPOINTS);
    int aqiCO = calculateAQI(mq7PPM * CO_CONVERSION, CO_BREAKPOINTS); // Convert CO
to mg/m³

    // Find maximum AQI
```

```
    overallAQI = max(aqiPM25, max(aqiNO2, aqiCO));

    // Determine AQI category
    if (overallAQI >= 0 && overallAQI <= 50) {
        aqiCategory = "Good";
    } else if (overallAQI <= 100) {
        aqiCategory = "Satisfactory";
    } else if (overallAQI <= 200) {
        aqiCategory = "Moderate";
    } else if (overallAQI <= 300) {
        aqiCategory = "Poor";
    } else if (overallAQI <= 400) {
        aqiCategory = "Very Poor";
    } else {
        aqiCategory = "Severe";
    }

    // Print AQI values to Serial Monitor
    Serial.print("AQI PM2.5: ");
    Serial.print(aqiPM25);
    Serial.print(", AQI NO2: ");
    Serial.print(aqiNO2);
    Serial.print(", AQI CO: ");
    Serial.print(aqiCO);
    Serial.print(", Overall AQI: ");
    Serial.print(overallAQI);
    Serial.print(", Category: ");
    Serial.println(aqiCategory);
}

void displayData(int overallAQI, const char* aqiCategory) {
    // Clear display for new data
    display.clearDisplay();
    display.fillRect(0, 0, 128, 16, WHITE);
    display.setTextSize(1);
    display.setTextColor(BLACK, WHITE);
    display.setCursor(1, 0);
    display.print("AIR QUALITY");
    display.setCursor(1, 8);
    display.print("MONITORING SYSTEM");
    display.display();

    // Display Temp, Humidity and PM2.5
    display.fillRect(0, 16, 128, 48, BLACK);
    display.setTextSize(2);
    display.setTextColor(WHITE);
    display.setCursor(1, 16);
    display.print("TEMP :");
    display.print(static_cast<int>(temperature)); // Display temperature as an
integer
    display.print("C"); // Unit for temperature
    display.setCursor(1, 32);
    display.print("HUMD :");
```

```
    display.print(static_cast<int>(humidity)); // Display humidity as an integer
    display.print("%"); // Unit for humidity
    display.setCursor(1, 48);
    display.print("PM2.5:");
    display.print(static_cast<int>(pm25Concentration)); // Display PM2.5 as an
integer
    display.print("ug/m3"); // Unit for PM2.5
    display.display();
    delay(2000);

    // Display TVOC, NO2 and CO
    display.fillRect(0, 16, 128, 48, BLACK);
    display.setCursor(1, 16);
    display.print("TVOC :");
    display.print(static_cast<int>(vocPPM)); // Display TVOC as an integer
    display.print("ppm"); // Unit for TVOC
    display.setCursor(1, 32);
    display.print("NO2  :");
    display.print(static_cast<int>(no2PPM)); // Display NO2 as an integer
    display.print("ppm"); // Unit for NO2
    display.setCursor(1, 48);
    display.print("CO   :");
    display.print(static_cast<int>(mq7PPM)); // Display MQ7 CO concentration as an
integer
    display.print("ppm"); // Unit for CO
    display.display();
    delay(2000);

    // Display overall AQI
    display.fillRect(0, 16, 128, 48, BLACK);
    display.setTextSize(3);
    display.setTextColor(WHITE);
    display.setCursor(1, 20);
    display.print("AQI:");
    display.print(overallAQI);
    display.setTextSize(2);
    display.setTextColor(WHITE);
    display.setCursor(1, 48);
    display.print(aqiCategory);
    display.display();
    delay(2000);
}

void sendToThingSpeak(int overallAQI, float temperature, float humidity, float
vocPPM, float no2PPM, float pm25Concentration, float mq7PPM) {
    ThingSpeak.setField(1, temperature); // Field 1 for temperature
    ThingSpeak.setField(2, humidity);    // Field 2 for humidity
    ThingSpeak.setField(3, pm25Concentration); // Field 3 for PM2.5
    ThingSpeak.setField(4, vocPPM);      // Field 4 for TVOC
    ThingSpeak.setField(5, no2PPM);      // Field 5 for NO2
    ThingSpeak.setField(6, mq7PPM);      // Field 6 for CO
    ThingSpeak.setField(7, overallAQI);  // Field 7 with overall AQI
```

```
    // Send data to ThingSpeak
    int result = ThingSpeak.writeFields(channelID, writeAPIKey);

    if (result == 200) {
        Serial.println("Data sent to ThingSpeak successfully.");
    } else {
        Serial.print("Failed to send data to ThingSpeak. Error code: ");
        Serial.println(result);
    }
}

void loop() {
    unsigned long currentMillis = millis();

    // Read sensors every 2 seconds
    if (currentMillis - lastSensorRead >= sensorInterval) {
        lastSensorRead = currentMillis;
        readSensors(); // Read sensor values

        // After reading sensors, calculate AQI and update display
        int overallAQI;  // Declare overallAQI as a local variable
        const char* aqiCategory;  // Declare AQI category as a local variable
        calculateAndDisplayAQI(overallAQI, aqiCategory); // Calculate AQI based on
readings
        displayData(overallAQI, aqiCategory);  // Display the AQI and other data on
OLED screen
    }

    // Send data to ThingSpeak every 30 seconds
    if (currentMillis - lastTime >= sendInterval) {
        lastTime = currentMillis;

        // Declare these variables before calling sendToThingSpeak
        int overallAQI;
        const char* aqiCategory;

        // Calculate AQI based on the most recent readings
        calculateAndDisplayAQI(overallAQI, aqiCategory);

        // Now, send data to ThingSpeak
        sendToThingSpeak(overallAQI, temperature, humidity, vocPPM, no2PPM,
pm25Concentration, mq7PPM);
    }
}
```

PROJECT GALLERY

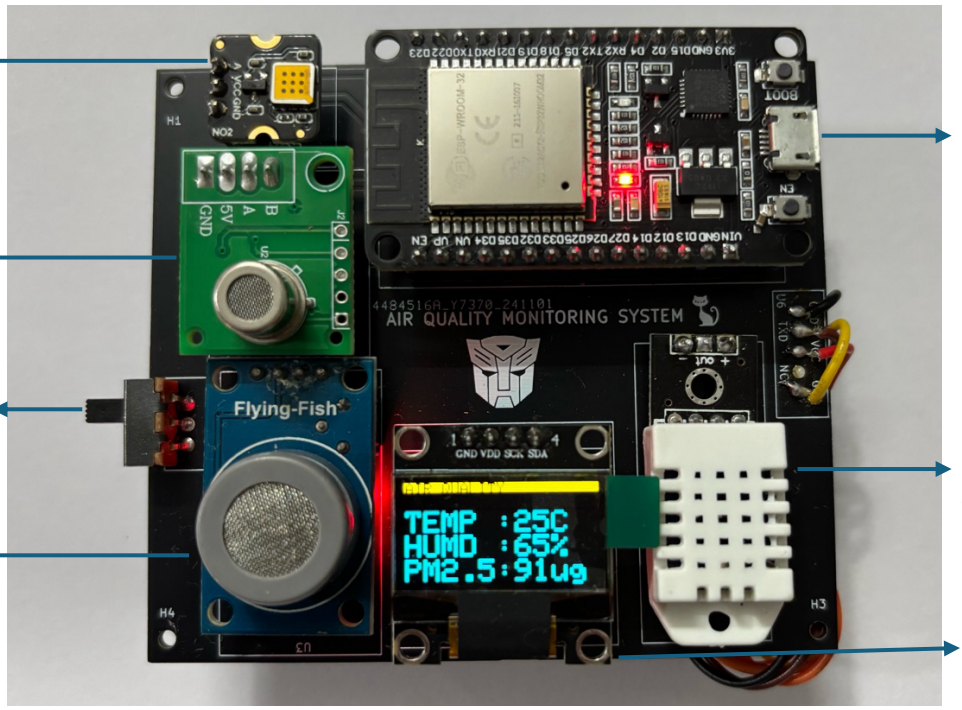


Fig. 1: All the components assembled on the PCB (Front).

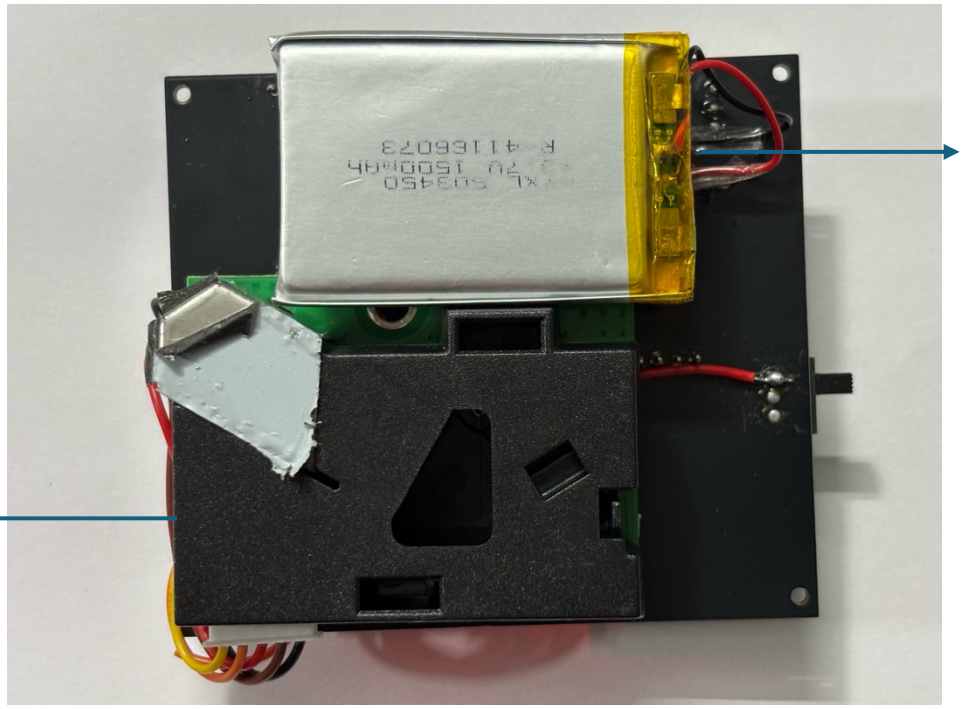


Fig. 2: All the components assembled on the PCB (Back).

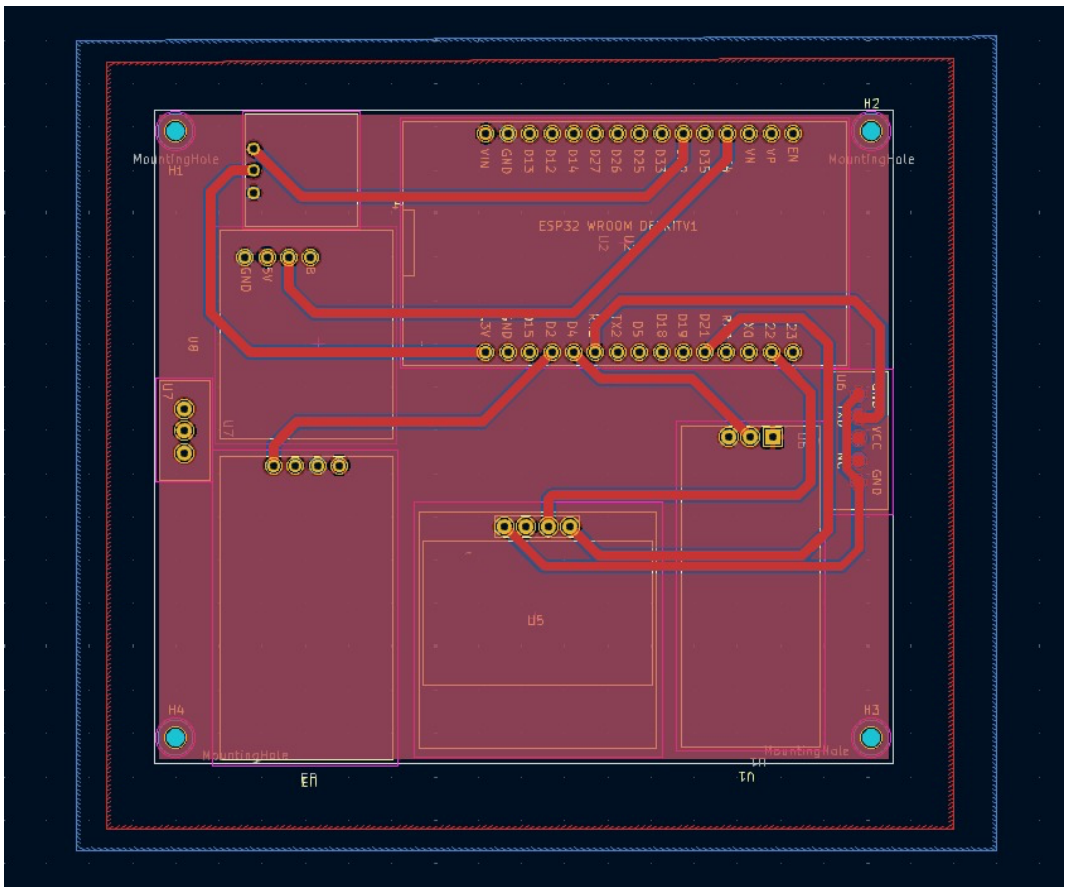


Fig 3: PCB path routing and components layout.

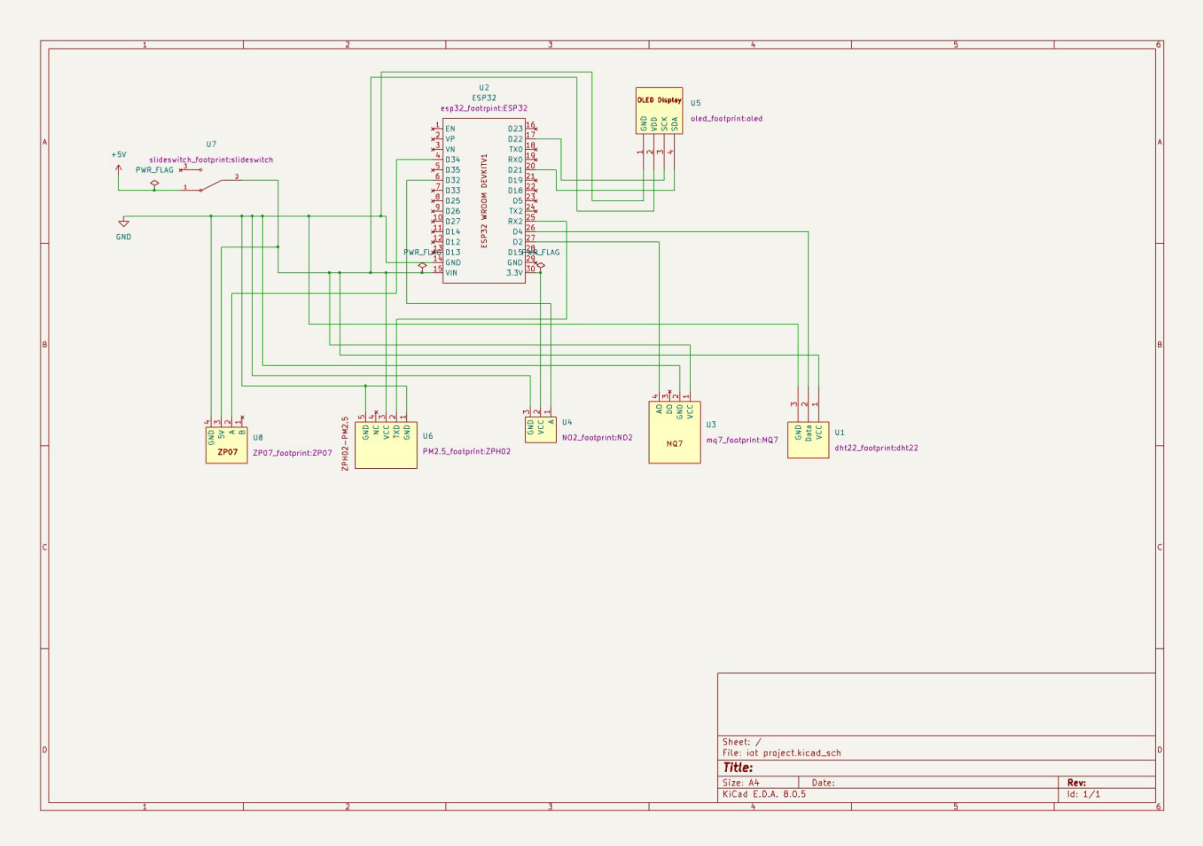


Fig. 4: PCB Schematic.

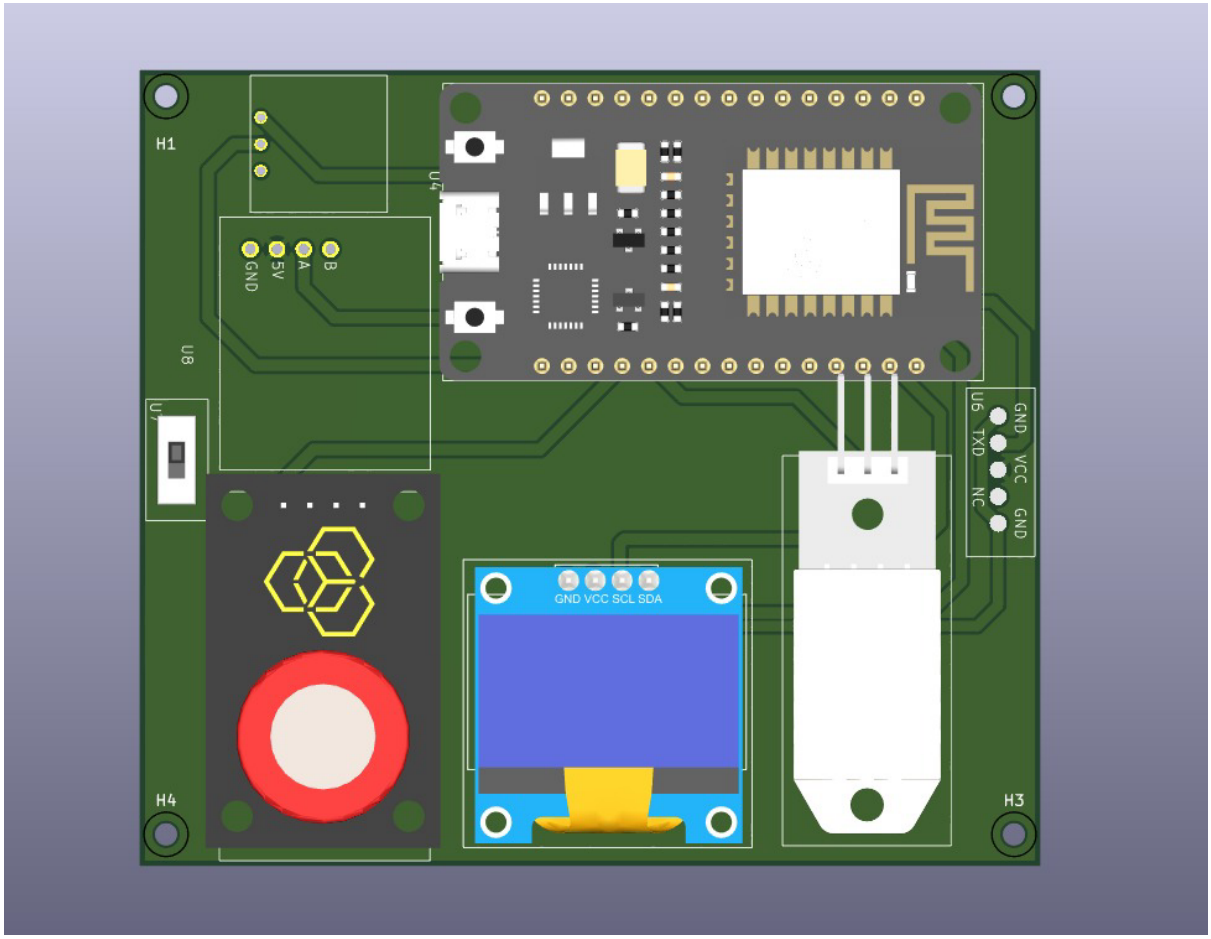


Fig. 5: 3D View of PCB with components in place.

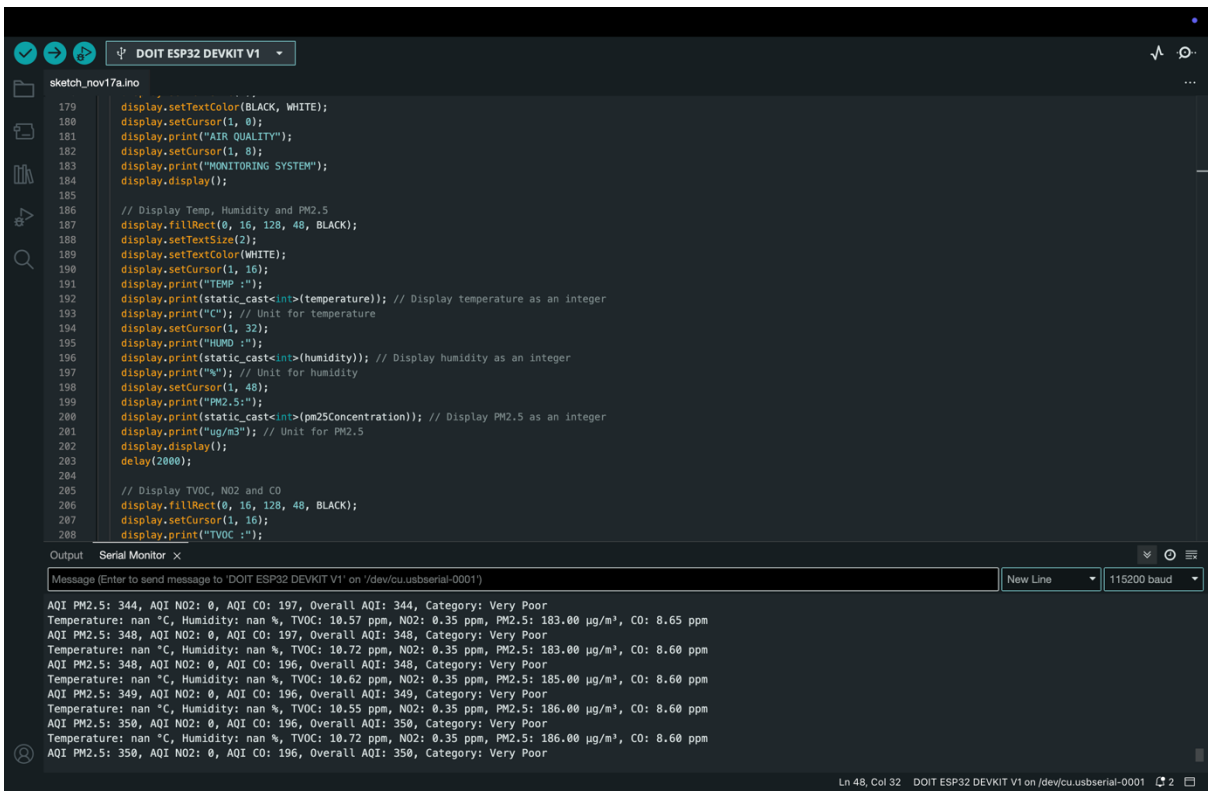


Fig. 6: Arduino IDE with Serial Monitor Window.

ThingSpeak

Fig. 7: ThingSpeak Dashboard.

CONCLUSION

The IoT-based Air Quality Monitoring System developed in this project successfully integrates various sensors, the ESP32 microcontroller, and cloud connectivity to provide a portable, real-time air quality monitoring solution. By utilizing a combination of five different environmental sensors for Particulate Matter 2.5, Carbon Monoxide, Nitrogen Dioxide, Total Volatile Organic Compounds (TVOCs) and Temperature & Humidity, the system captures critical environmental data and processes it to calculate the Air Quality Index (AQI). This AQI with their respective readings is displayed on a compact OLED screen, providing immediate feedback to users about air quality. The system's portability is a key feature, as it is powered by a battery, this ensures mobility and convenience for outdoor use or in areas where a power outlet is not readily available. The use of a custom-designed PCB further contributes to the system's compact design, minimizing the overall footprint while integrating all the components in a streamlined, efficient layout. Additionally, the integration of the ThingSpeak cloud platform enables remote monitoring and data visualization, ensuring that the system is not only useful locally but also accessible from anywhere in real time.

The project demonstrates the feasibility and effectiveness of using the ESP32 as a central microcontroller for portable, real-time environmental monitoring. By leveraging cost-effective sensors, a custom PCB and cloud integration, the system offers an affordable solution for tracking air quality in diverse settings, from urban areas to rural and industrial environments. The ability to remotely monitor and analyse data makes the system highly versatile and scalable, with potential for future enhancements, such as adding more sensors or incorporating advanced data analytics. Overall, this project not only showcases the potential of IoT technologies in environmental monitoring but also underscores how such systems can be easily adapted to meet diverse monitoring needs and improve air quality awareness globally.

REFERENCES

CPCB. (2014-15). *National Air Quality Index*. CPCB.

Dejan. (n.d.). *DIY Air Quality Monitor – PM2.5, CO2, VOC, Ozone, Temp & Hum Arduino Meter*. Retrieved from HowToMechatronics.com: https://howtomechatronics.com/projects/diy-air-quality-monitor-pm2-5-co2-voc-ozone-temp-hum-arduino-meter/

Mark, J. (2022, May 27). *Air Quality Monitoring ESP32-Based System*. Retrieved from hackster.io: https://www.hackster.io/joey-mark/air-quality-monitoring-esp32-based-system-4cbc2d

Chouddhary, A. (2022, May 23). *IoT Based Air Quality Index Monitoring System – Monitor PM2.5, PM10, and CO using ESP32*. Retrieved from circuitdigest.com: https://circuitdigest.com/microcontroller-proejcts/iot-based-air-quality-index-monitoring-system-measure-pm25-pm10-co-using-esp32

Zhengzhou Winsen Electronics Technology Co., L. (2018, May 16). Retrieved from robu.in: https://robu.in/wp-content/uploads/2021/03/ZPH02-Particles-Sensor.pdf

DFRobot. (n.d.). Retrieved from dfrobot.com: https://www.dfrobot.com/product-2711.html

Zhengzhou Winsen Electronics Technology Co., L. (2014, November 10). Retrieved from robu.in: https://robu.in/wp-content/uploads/2021/03/Winsen-Air-Quality-Detection-Module.pdf

HANWEI ELECTRONICS CO ., L. (n.d.). Retrieved from robu.in: https://robu.in/wp-content/uploads/2017/09/MQ-7.pdf

Aosong Electronics Co., L. (n.d.). Retrieved from robu.in: https://robu.in/wp-content/uploads/2017/06/DHT22-Datasheet-Digital-Temperature-and-Humidity-Sensor-Module-AM2302-ROBU.IN_.pdf

Systems, E. (2023). Retrieved from espressif.com: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf