Sukkur IBA University
COMPUTER SYSTEMS ENGINEERING

WEATHER REPORTING SYSTEM

Internet Of Things

Presented To

DR. SURESH KUMAR

Presented By

KASHIF MUJEEB (133-20-0009)

ABDUL RAQEEB (133-20-0001)

MUHAMMAD USMAN (133-20-0004)

SHAHROZ MAJID (133-20-0006)

Table of Contents

ABSTRACT

This project demonstrates the integration of the ESP-NOW wireless communication protocol and the Adafruit IO cloud service to establish an efficient sensor network. The system is comprised of two ESP32 devices: a sender and a receiver. The sender collects environmental data using DHT11 and BMP180 sensors, which measure temperature, humidity, pressure, and altitude. This data is then transmitted to the receiver via ESP-NOW. The receiver displays the incoming data on an OLED screen for immediate local monitoring and also publishes the data to Adafruit IO for remote access. Additionally, the receiver collects its own sensor data, providing redundancy and enhancing data reliability. The data is published to Adafruit IO at five-minute intervals, enabling continuous and remote monitoring through the Adafruit IO platform. This project highlights the practical application of ESP32 devices in wireless sensor networks and their integration with cloud services for remote monitoring and data analysis.

INTRODUCTION

In today's interconnected world, the need for efficient, wireless communication systems is more prominent than ever. The integration of wireless sensor networks with cloud services provides a powerful solution for real-time data collection, monitoring, and analysis. This project leverages the capabilities of ESP32 microcontrollers, ESP-NOW wireless communication protocol, and the Adafruit IO cloud service to create a robust and efficient sensor network.

The system comprises two ESP32 devices: a sender and a receiver. The sender is equipped with DHT11 and BMP180 sensors to measure environmental parameters such as temperature, humidity, pressure, and altitude. Utilizing the low-power, high-speed ESP-NOW protocol, the sender transmits the collected data to the receiver without the need for WiFi infrastructure, making the system both versatile and efficient.

The receiver, upon receiving the data, displays it on an OLED screen for immediate local monitoring. Furthermore, the receiver is also equipped with its own set of sensors to gather additional data, which ensures redundancy and increases the reliability of the data. This data is then published to Adafruit IO, a cloud-based platform, allowing for remote monitoring and analysis. The publication of data to Adafruit IO occurs at regular intervals of five minutes, providing continuous and up-to-date information accessible from anywhere with an internet connection.

This project not only demonstrates the practical application of ESP32 devices and ESP-NOW in building wireless sensor networks but also showcases the seamless integration with cloud services like Adafruit IO. It underscores the potential of such systems in various fields, including environmental monitoring, smart agriculture, and industrial automation. Through this project, we aim to provide insights into the design and implementation of efficient, real-time monitoring systems using modern microcontroller technology and cloud services.

METHODOLOGY

System Design and Architecture

The project consists of two primary nodes: a sender and a receiver. The sender node is equipped with sensors to measure temperature, humidity, pressure, and altitude. It collects data from these sensors and transmits it to the receiver node using ESP-NOW. The receiver node, upon receiving the data, displays it on an OLED screen and publishes it to Adafruit IO via MQTT for remote monitoring.

Hardware Components

ESP32 Microcontroller (x2):

Function: Acts as the core of the system, with one ESP32 designated as the sender and the other as the receiver.

Features: Built-in WiFi and Bluetooth capabilities, low-power consumption, and multiple GPIO pins for sensor interfacing.

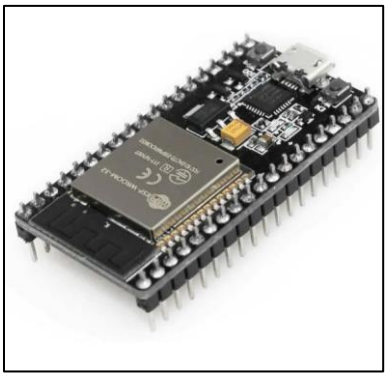

Figure 1 ESP 32 Development Board

DHT11 Sensor:

Function: Measures temperature and humidity.

Features: Digital output, low-cost, and easy-to-use with microcontrollers.

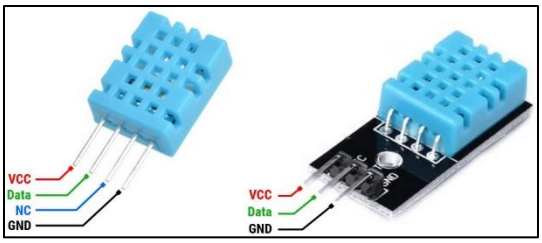

Figure 2 DHT 11 Sesnsor

BMP180 Sensor:

Function: Measures atmospheric pressure and calculates altitude.

Features: High precision, low power consumption, and I2C interface.

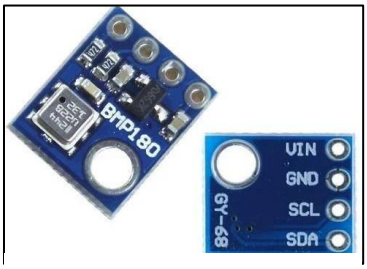

Figure 3 BMP180 Sensor

Adafruit SSD1306 OLED Display:

Function: Displays the sensor data received by the ESP32 receiver.

Features: 128x64 pixels resolution, I2C interface, and low power consumption.

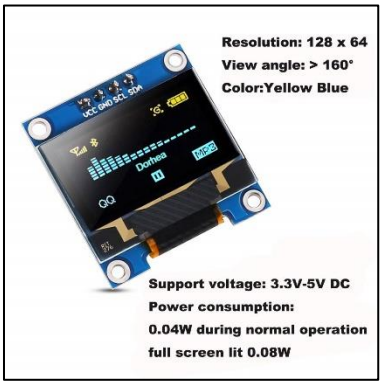

Figure 4 SSD1306 OLED DISPLAY

Software Components

Arduino IDE:

Function: Development environment for writing, compiling, and uploading code to the ESP32 boards.

Features: Extensive library support, user-friendly interface, and wide community support.

ESP-NOW Protocol:

Function: Enables low-power, high-speed wireless communication between the ESP32 devices.

Features: Peer-to-peer communication without WiFi infrastructure, low latency, and low power consumption.

Adafruit IO Cloud Service:

Function: Cloud platform for storing, visualizing, and analyzing the sensor data.

Features: Real-time data updates, secure data storage, and easy integration with IoT devices.

Adafruit MQTT Library:

Function: Facilitates communication between the ESP32 receiver and Adafruit IO.

Features: Lightweight, efficient, and supports secure connections.

Adafruit Sensor Libraries (for DHT11 and BMP180):

Function: Provides functions to interface with and read data from the sensors.

Features: Simplifies sensor data acquisition, ensures compatibility, and provides example codes.

By combining these hardware and software components, the project creates a comprehensive sensor network capable of local and remote monitoring, demonstrating the power of integrating microcontroller technology with cloud services for real-time data analysis and visualization.

IMPLEMENTATION

The project is divided into two main components: the sender node and the receiver node, both built using ESP32 microcontrollers. The sender node is responsible for collecting sensor data and transmitting it wirelessly using the ESP-NOW protocol. The receiver node receives this data, displays it on an OLED screen, and publishes it to Adafruit IO for remote monitoring. Additionally, a web interface was created to visualize the data in real-time.

Sender Node:

ESP32 Microcontroller: Acts as the central processing unit.

DHT11 Sensor: Connected to a digital pin on the ESP32 for measuring temperature and humidity.

BMP180 Sensor: Connected via I2C for measuring pressure and altitude.

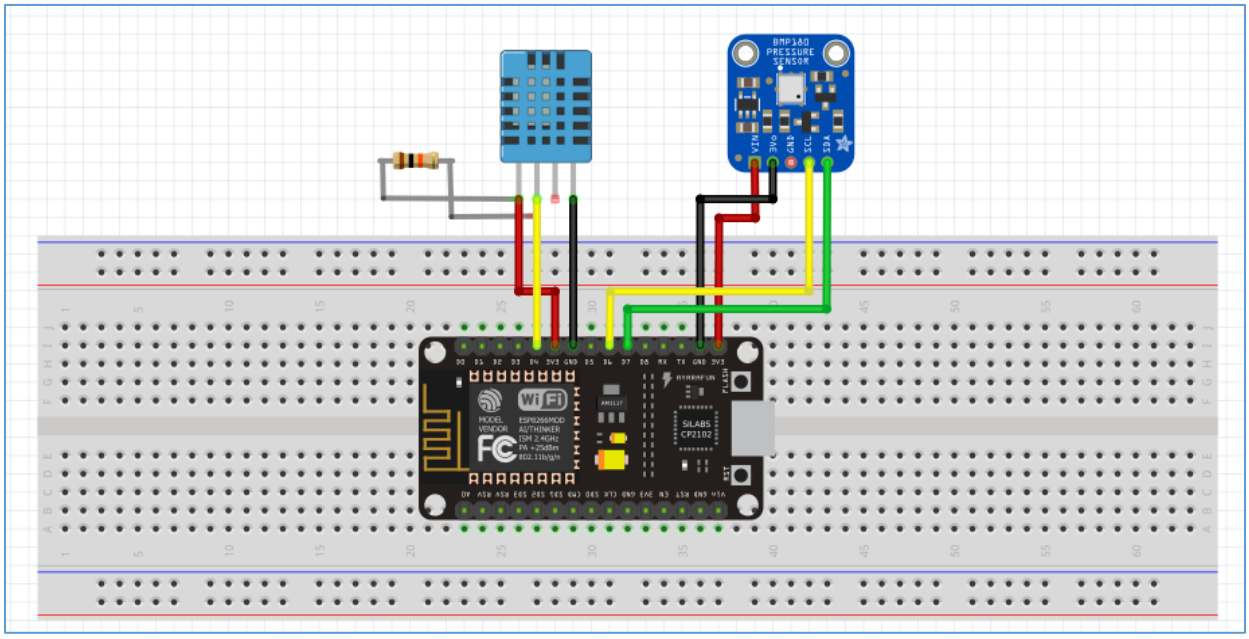

Figure 5 Sender Node Connected with DHT11 and BMP 180

Sender Code:

```
#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <Adafruit_BMP085.h>
#include <DHT.h>

#define DHTPIN 4       // DHT11 data pin
#define DHTTYPE DHT11

uint8_t broadcastAddress[] = {0x08, 0x3A, 0xF2, 0x51, 0x71, 0x84}; // Replace with your receiver's MAC

// Combined data structure for both sensors
typedef struct sensor_data {
   float bmpTemperature; // BMP180 temperature
   float bmpPressure;    // BMP180 pressure in hPa
   float bmpAltitude;    // BMP180 altitude (optional)
   float dhtTemperature; // DHT11 temperature
   float dhtHumidity;    // DHT11 humidity
} sensor_data;

sensor_data mySensorData;
esp_now_peer_info_t peerInfo;

Adafruit_BMP085 bmp;
DHT dht(DHTPIN, DHTTYPE);

// Callback function for ESP-NOW send status
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
   Serial.print("\r\nLast Packet Send Status:\t");
```

```
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);

  // Initialize ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  esp_now_register_send_cb(OnDataSent); // Register callback for send status

  // Register peer
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }

  // Initialize BMP180 sensor
  if (!bmp.begin()) {
    Serial.println("Could not find a valid BMP180 sensor, check wiring!");
    while (1); // Wait indefinitely (or add error handling)
  }

  dht.begin(); // Initialize DHT11 sensor
}
```

```
void loop() {
  // Read BMP180 sensor data
  mySensorData.bmpTemperature = bmp.readTemperature();
  mySensorData.bmpPressure = bmp.readPressure() / 100.0; // Convert to hPa (optional)
  mySensorData.bmpAltitude = bmp.readAltitude();

  // Read DHT11 sensor data
  mySensorData.dhtHumidity = dht.readHumidity();
  mySensorData.dhtTemperature = dht.readTemperature();

  // Send sensor data via ESP-NOW
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)&mySensorData, sizeof(mySensorData));

  if (result == ESP_OK) {
    Serial.println("Sent with success");
  } else {
    Serial.println("Error sending the data");
  }

  // Display sensor data (optional, for debugging)
  Serial.println("----- BMP180 -----");
  Serial.print("Temperature: ");
  Serial.print(mySensorData.bmpTemperature);
  Serial.print(" C\tPressure: ");
  Serial.print(mySensorData.bmpPressure);
  Serial.print(" hPa\tAltitude: ");
  Serial.print(mySensorData.bmpAltitude);
  Serial.println(" meters");
  Serial.println("----- DHT11 -----");
  Serial.print("Temperature: ");
```

```
    Serial.print(mySensorData.dhtTemperature);
    Serial.print(" C\tHumidity: ");
    Serial.print(mySensorData.dhtHumidity);
    Serial.println(" %");
    Serial.println("-----------------"); // Separator for readability

    delay(2000); // Adjust the delay between readings as needed
}
```

Receiver Node:

ESP32 Microcontroller: Acts as the central processing unit.

OLED Display (128x64): Connected via I2C for displaying sensor data.

DHT11 Sensor: Connected to a digital pin for measuring local temperature and humidity.

BMP180 Sensor: Connected via I2C for measuring local pressure and altitude.

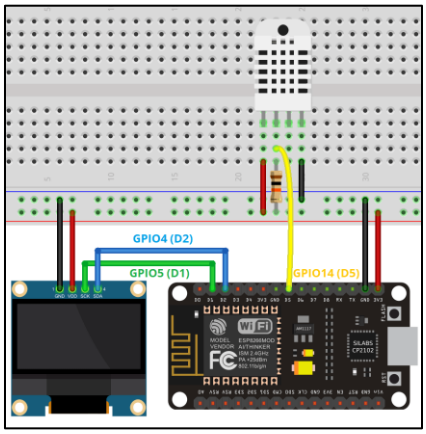

Figure 6 Reciever Node Connected with DHT11 and OLED

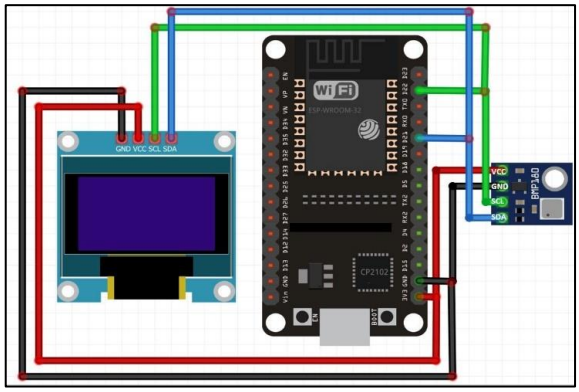

Figure 7 Reciever Node Connected with OLED and BMP180

Reciever Node:

```
#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_BMP085.h>
#include <DHT.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"

// Define OLED display size and reset pin
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// Define DHT11 sensor on receiver node
#define DHTPIN 4       // DHT11 data pin on receiver
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// Define BMP180 sensor on receiver node
Adafruit_BMP085 bmp;

// WiFi credentials and Adafruit IO
#define WLAN_SSID       "kashif"
#define WLAN_PASS       "12345678"
#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883
#define AIO_USERNAME    "kashif_mujeeb"
#define AIO_KEY         "aio_jcBR67Wtl163KZOIkTG0uGeWqBLn"
```

```
WiFiClient client;
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);
Adafruit_MQTT_Publish feedIntemp = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/intemp");
Adafruit_MQTT_Publish feedInhum = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/inhum");
Adafruit_MQTT_Publish feedInalt = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/inalt");
Adafruit_MQTT_Publish feedInpress = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/inpress");
Adafruit_MQTT_Publish feedInbmptemp = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/inbmptemp");

// New feeds for receiver's own sensors
Adafruit_MQTT_Publish feedTemp = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/temp");
Adafruit_MQTT_Publish feedHum = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/hum");
Adafruit_MQTT_Publish feedAlt = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/alt");
Adafruit_MQTT_Publish feedPressure = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/pressure");

bool data_received = false;

// Structure to hold received sensor data
typedef struct sensor_data {
    float bmpTemperature; // BMP180 temperature from sender
    float bmpPressure;    // BMP180 pressure in hPa from sender
    float bmpAltitude;    // BMP180 altitude from sender
    float dhtTemperature; // DHT11 temperature from sender
    float dhtHumidity;    // DHT11 humidity from sender
} sensor_data;
```

```
sensor_data receivedData;

// Function to initialize OLED display
void initOLED() {
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;);
  }
  display.display();
  display.clearDisplay();
}

// Function to display data on OLED
void displayDataOnOLED(float bmpTemp, float bmpPress, float bmpAlt, float dhtTemp, float dhtHum, const char* source) {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);

  display.setCursor(0, 0);
  display.print(F("Source: "));
  display.println(source);

  display.setCursor(0, 10);
  display.print(F("BMP Temp: "));
  display.print(bmpTemp);
  display.println(F(" C"));

  display.setCursor(0, 20);
  display.print(F("BMP Press: "));
  display.print(bmpPress);
```

```
    display.println(F(" hPa"));

    display.setCursor(0, 30);
    display.print(F("BMP Alt: "));
    display.print(bmpAlt);
    display.println(F(" m"));

    display.setCursor(0, 40);
    display.print(F("DHT Temp: "));
    display.print(dhtTemp);
    display.println(F(" C"));

    display.setCursor(0, 50);
    display.print(F("DHT Hum: "));
    display.print(dhtHum);
    display.println(F(" %"));

    display.display();
}

// Callback function when data is received
void OnDataRecv(const uint8_t *mac, const uint8_t *incomingData, int len) {
    data_received = true;
    memcpy(&receivedData, incomingData, sizeof(receivedData));  // Copy received data into the struct
    Serial.println("Received Data:");
    Serial.print("BMP180 Temperature: ");
    Serial.print(receivedData.bmpTemperature);
    Serial.println(" C");
    Serial.print("BMP180 Pressure: ");
    Serial.print(receivedData.bmpPressure);
    Serial.println(" hPa");
```

```
    Serial.print("BMP180 Altitude: ");
    Serial.print(receivedData.bmpAltitude);
    Serial.println(" m");
    Serial.print("DHT11 Temperature: ");
    Serial.print(receivedData.dhtTemperature);
    Serial.println(" C");
    Serial.print("DHT11 Humidity: ");
    Serial.print(receivedData.dhtHumidity);
    Serial.println(" %");
}

void setup() {
    Serial.begin(115200);
    initOLED();

    WiFi.mode(WIFI_STA);

    // Initialize ESP-NOW
    if (esp_now_init() != ESP_OK) {
        Serial.println("Error initializing ESP-NOW");
        return;
    }

    // Register receive callback
    esp_now_register_recv_cb(OnDataRecv);

    // Initialize BMP180 sensor on receiver node
    if (!bmp.begin()) {
        Serial.println("Could not find a valid BMP180 sensor, check wiring!");
        while (1);
    }
```

```
    dht.begin(); // Initialize DHT11 sensor on receiver node
}

void loop() {
    start_now();
    if (data_received) {
        esp_now_deinit();
        connect_wifi();
        connect();
        send_data_msg();
        delay(1000);
        data_received = false;
    }

    // Read BMP180 sensor data from receiver node
    float bmpTempReceiver = bmp.readTemperature();
    float bmpPressReceiver = bmp.readPressure() / 100.0;
    float bmpAltReceiver = bmp.readAltitude();

    // Read DHT11 sensor data from receiver node
    float dhtHumReceiver = dht.readHumidity();
    float dhtTempReceiver = dht.readTemperature();

    // Display received data from sender node
    displayDataOnOLED(receivedData.bmpTemperature, receivedData.bmpPressure, receivedData.bmpAltitude, receivedData.dhtTemperature, receivedData.dhtHumidity, "Sender");
    delay(2000); // Adjust delay to create animation effect

    // Display data from receiver node
    displayDataOnOLED(bmpTempReceiver, bmpPressReceiver, bmpAltReceiver, dhtTempReceiver, dhtHumReceiver, "Receiver");
    delay(2000); // Adjust delay to create animation effect
```

```
    // Publish receiver node data to Adafruit IO
    feedTemp.publish(dhtTempReceiver);
    feedHum.publish(dhtHumReceiver);
    feedAlt.publish(bmpAltReceiver);
    feedPressure.publish(bmpPressReceiver);

    delay(2000); // Adjust delay between readings
}

void connect_wifi() {
    int wifi_connection = 0;
    Serial.println("Connecting WiFi..");
    WiFi.mode(WIFI_STA);
    WiFi.begin(WLAN_SSID, WLAN_PASS);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        if (wifi_connection > 99) { ESP.restart(); }
        wifi_connection++;
        Serial.println(F("."));
    }

    Serial.println();
    Serial.println(F("WiFi connected to: "));
    Serial.println(WiFi.SSID());
    Serial.println(F("IP address: "));
    Serial.println(WiFi.localIP());
    delay(500);
}

void disconnect_wifi() {
    WiFi.disconnect();
```

```
  WiFi.mode(WIFI_OFF);
  int wifi_connection = 0;
  while (WiFi.status() == WL_CONNECTED) {
    delay(100);
    if (wifi_connection > 99) { ESP.restart(); }
    wifi_connection++;
  }
  Serial.println("WiFi going to Sleep\n");
}

// Connect to Adafruit IO via MQTT
void connect() {
  int io_connection = 0;
  Serial.println(F("Connecting to Adafruit IO... "));
  int8_t ret;

  while ((ret = mqtt.connect()) != 0) {
    io_connection++;
    switch (ret) {
      case 1: Serial.println(F("Wrong protocol")); break;
      case 2: Serial.println(F("ID rejected")); break;
      case 3: Serial.println(F("Server unavailable")); break;
      case 4: Serial.println(F("Bad user/pass")); break;
      case 5: Serial.println(F("Not authorized")); break;
      case 6: Serial.println(F("Failed to subscribe")); break;
      default: Serial.println(F("Connection failed")); break;
    }
    if (io_connection == 10) { ESP.restart(); }

    if (ret >= 0)
      mqtt.disconnect();
```

```
    Serial.println(F("Retrying connection..."));
    delay(5000);
  }
  Serial.println(F("Adafruit IO Connected!"));
  delay(500);
}

void send_data_msg() {
  Serial.println("Sending data");

  // Publish data to Adafruit IO
  feedIntemp.publish(receivedData.dhtTemperature);
  feedInhum.publish(receivedData.dhtHumidity);
  feedInalt.publish(receivedData.bmpAltitude);
  feedInpress.publish(receivedData.bmpPressure);
  feedInbmptemp.publish(receivedData.bmpTemperature);

  delay(500);
}

void start_now() {
  WiFi.mode(WIFI_STA);
  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  Serial.println("Starting ESP Now");
  // Once ESPNow is successfully initialized, we will register for recv callback to get received packet info
  esp_now_register_recv_cb(OnDataRecv);
}
```

RESULTS

The implementation of the ESP-NOW based sensor network, integrated with Adafruit IO for remote monitoring, yielded successful results. Here are the key outcomes:

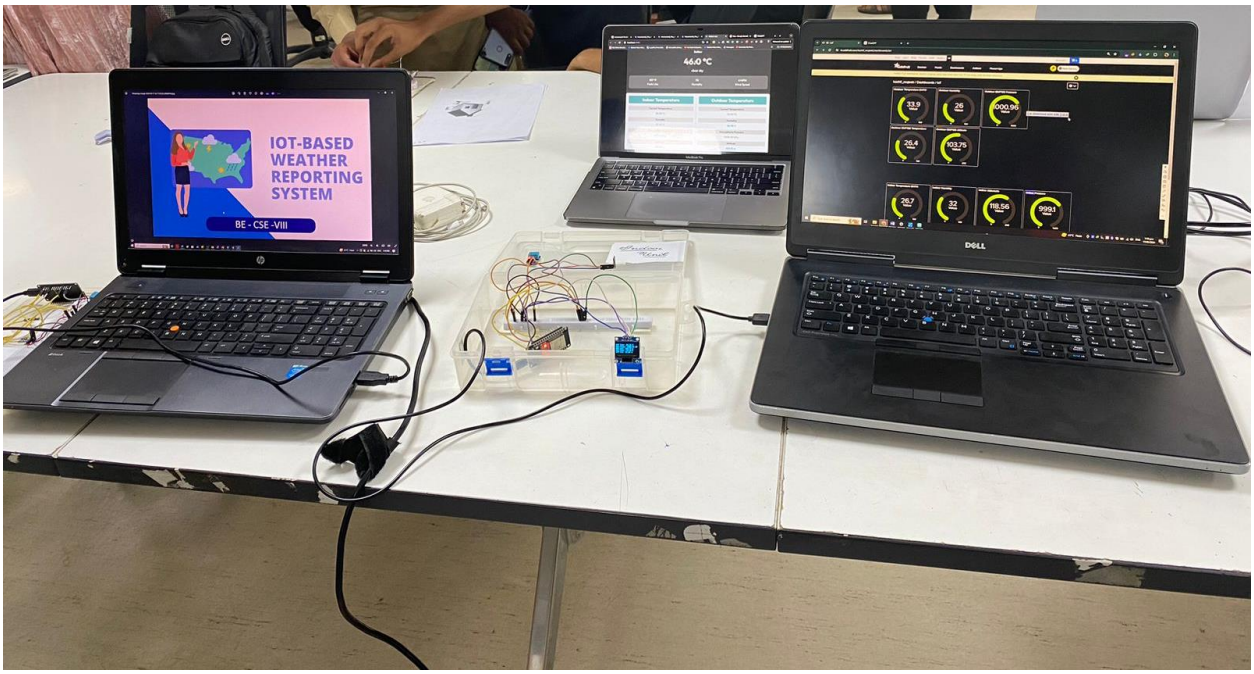

Figure 8 Sender Node sends data to Reciever Node and upload it to cloud

Local Display and Visualization

Local Data Display: The OLED display on the receiver node effectively showcased the received sensor data, providing real-time insights into environmental conditions.

Figure 9 Sender Node Sensors Data Displays on OLED

Figure 10 Reciever Node Sensors* Data *Displays on OLED

Remote Monitoring: Data from both the sender and receiver nodes were seamlessly published to Adafruit IO, enabling remote access and monitoring through the web interface.

Figure 11: Published Data of Sender Node on Adafruit IO

Figure 12 Publish Data of Reciever Node on Adafruit IO

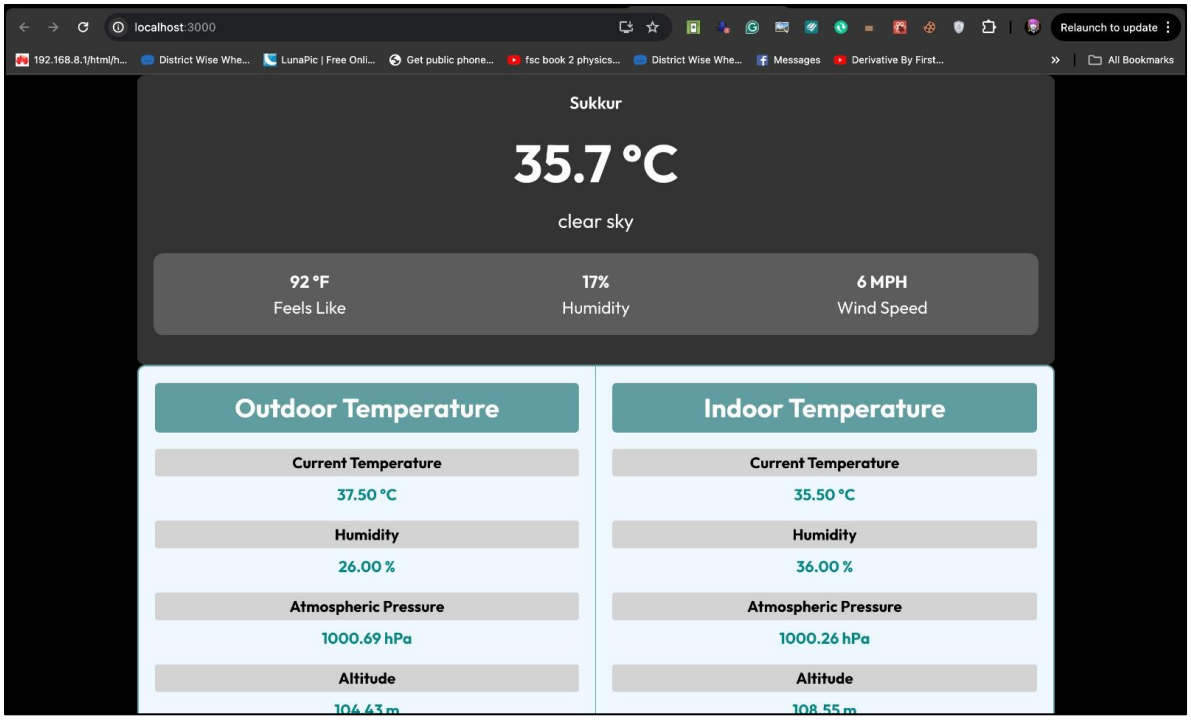

Figure 12 Web Interface with Weather API

CONCLUSION

The project successfully achieved its objectives of creating an IoT-based sensor network using ESP-NOW and Adafruit IO. The seamless integration of wireless communication, local display, and remote monitoring provides a robust foundation for diverse IoT applications. Further refinements and enhancements can unlock additional functionalities and broaden the system's scope for various real-world scenarios.

REFERENCE

1. Tutorials Provided by Dr. Suresh Kumar

2. ESP32/ESP8266: DHT Temperature and Humidity Readings in OLED Display:

   https://randomnerdtutorials.com/esp32-esp8266-dht-temperature-and-humidity-oled-display/

3. ESP32 I2C Communication Using OLED Showing BMP 180 Measurement:

   https://medium.com/@dwicakradanielle/esp32-i2c-communication-using-oled-showing-bmp-180-measurement-a84a8de74116

4. Interface OLED Graphic Display Module with ESP32:

   https://lastminuteengineers.com/oled-display-esp32-tutorial/

5. ESP-NOW Two-Way Communication Between ESP32 Boards:

   https://randomnerdtutorials.com/esp-now-two-way-communication-esp32/