United International University

Department of CSE

Course Code: CSE 4326

Course Name: Microprocessors and Microcontrollers Lab.

Experiment no.: 02

Experiment Name: Wifi Communication and building IoT-based systems Using Arduino IoT Cloud

Submitted by:

Name: **Md Musfiqur Rahman & Md Mahbubur Rahman**

Student ID: **011221334** **&** **011223441**

Section: **D**

Date of Performance: 07-10-2024

Date of Submission: **09-10-2024**

Objective:

The main goal of this report is to show how to set up an IoT system using ESP32 microcontrollers, the Arduino IoT Cloud, and different sensors and actuators. The report will cover:

1. Describing how the ESP32 microcontroller communicates with the Arduino IoT Cloud for sending data and receiving control commands.
2. Demonstrating how to build a practical IoT system for monitoring a farm, particularly focusing on soil moisture sensors and controlling a water pump.
3. Providing a detailed block diagram that outlines the system's architecture and components.
4. Creating a program that sends and receives data via Wi-Fi to and from the cloud, along with integrating LED indicators for visual feedback.
5. Setting up a local ESP32 server with button switches and LED widgets for easy remote control and monitoring.
6. Implementing a gas sensor monitoring system in a kitchen that sends data to the cloud and stores it in Google Sheets or Excel files.
7. Writing programs to pull data from the cloud and trigger an alarm if gas levels go above a safe limit (50 ppm).

By addressing these points, the report aims to give a clear and practical guide on building IoT systems with ESP32 microcontrollers and Arduino IoT Cloud, covering everything from collecting sensor data to controlling devices and monitoring remotely.

Components:

Hardware:

1) ESP-32 Development Board

2) LED

3) MQ-5 Gas Sensor

Software:

Arduino Cloud IoT

Theory:

Hardware Overview:

1. **The ESP32 Microcontroller**

The ESP32 is a versatile and powerful microcontroller, perfect for embedded projects that need Wi-Fi and Bluetooth connectivity. Created by Espressif Systems, it builds on the success of the earlier ESP8266, offering more processing power, better connectivity, and greater flexibility. Here’s an overview of what makes the ESP32 stand out:

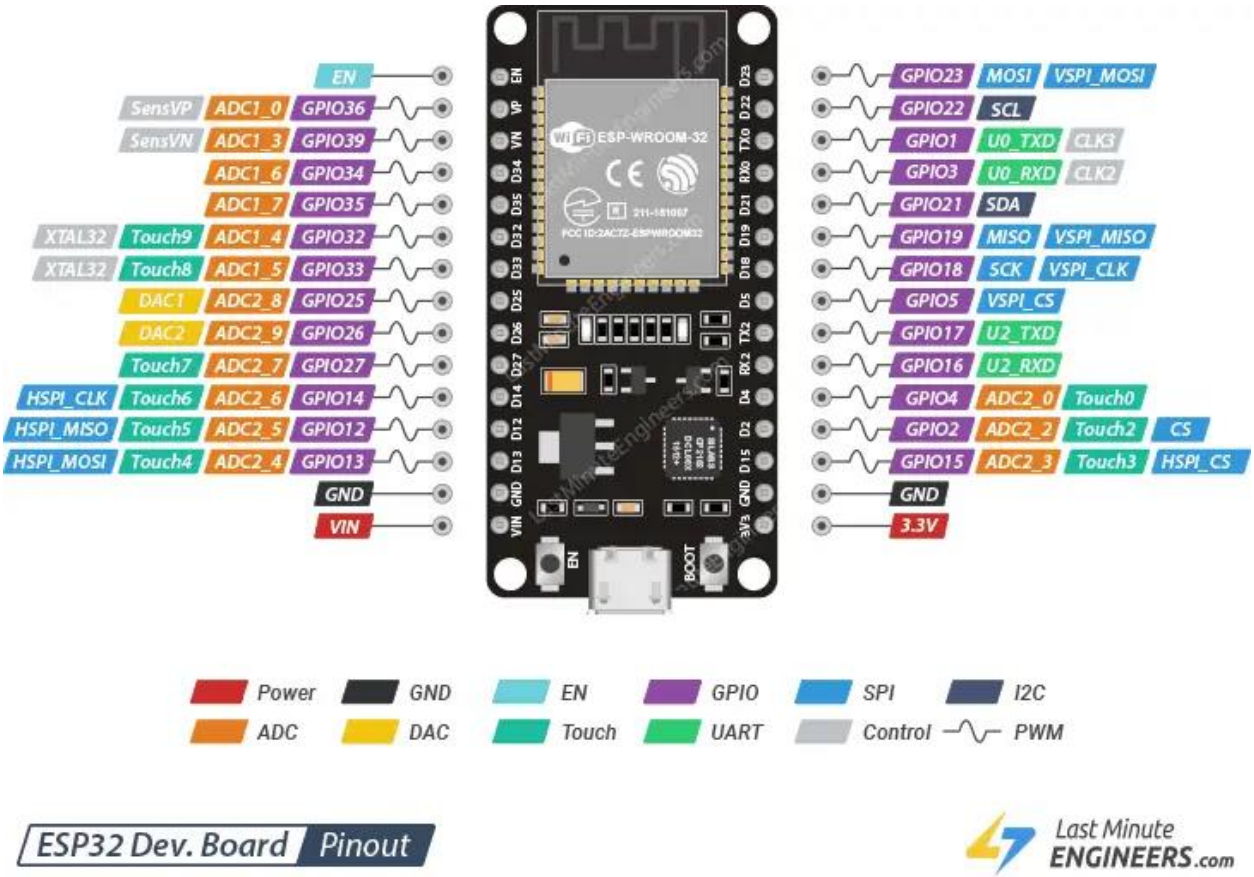


Fig 1: ESP32 Pinout

a) Architecture:

- The ESP32 is powered by a dual-core Xtensa LX6 microprocessor, which can run at speeds up to 240 MHz

- It's loaded with a variety of peripherals like digital and analog interfaces, GPIO pins, SPI, I2C, UART, ADC, DAC, and more, allowing it to easily connect with sensors, actuators, and other devices.

b) Wireless Connectivity:

- Wi-Fi: The ESP32 supports 802.11 b/g/n Wi-Fi, making it easy to connect to local networks for internet access and communication.

- Bluetooth: It also includes Bluetooth Low Energy (BLE), enabling it to communicate with smartphones, tablets, and other Bluetooth devices.

c) Memory:

- The ESP32 usually comes with built-in flash memory for storing programs and SRAM for data storage and execution.

- It has plenty of memory capacity to store firmware, data, and support dynamic program execution.

d) Peripheral Interfaces:

- GPIO: The ESP32 offers many General Purpose Input/Output (GPIO) pins that can be used for various functions like digital input/output, handling interrupts, PWM, and more.

- Serial Interfaces: It supports several serial communication protocols like SPI, I2C, and UART, allowing easy integration with a wide range of external devices.

- Analog Inputs: The ESP32 includes ADC pins for reading analog sensor inputs and DAC pins for generating analog signals.

e) Development Environment:

- SDKs: Espressif provides comprehensive SDKs and tools for programming the ESP32 in languages like C/C++ and Python.

- Arduino IDE: The ESP32 is well-supported by the Arduino community, making it easy to create and deploy projects using the familiar Arduino IDE.

f) Power Management:

- The ESP32 has advanced power management features, including various sleep modes and low-power operation, making it ideal for battery-powered projects that need long battery life.

g) Security Features:

- It comes with strong security measures like secure boot, flash encryption, and cryptographic accelerators to ensure data integrity and device authentication in connected applications.

h) Applications:

- IoT Devices: The ESP32 is widely used in Internet-of-Things (IoT) projects, powering devices like smart home gadgets, environmental sensors, and wearable tech.

- Industrial Automation: It's also used in industrial automation, offering connectivity and control for monitoring and managing equipment remotely.

- Consumer Electronics: The ESP32 finds its way into various consumer electronics, including smartwatches, fitness trackers, and home automation systems.

Software Overview:

Arduino IDE:

The Arduino Integrated Development Environment (IDE) is essential software for programming Arduino boards, which are popular in many electronics projects. The Arduino IDE makes it easy to write, compile,

and upload code to an Arduino board. It's designed to be simple enough for beginners to use, yet powerful enough for more experienced users.

Two Key functions of Arduino IDE are explained below:

```
void setup() {
  // runs once when the program starts
}

void loop() {
  // repeats over and over
}
```

These two functions form the backbone of any Arduino program. The `setup()` function runs just once at the beginning, when the program starts. It's used for tasks that need to be done upfront, like setting pin modes or starting serial communication.

Once `setup()` has done its job, the `loop()` function takes over. This function runs continuously, repeating if the Arduino is on or until the program stops. The `loop()` function is where the main action happens, such as reading inputs from sensors, controlling outputs like LEDs or motors, and handling calculations.

Problem 1

Explain briefly how ESP32 is communicating via Arduino IoT Cloud. Suppose you are observing the moisture in an agricultural farm using a sensor. You have to send the sensor data to a cloud. Based on the sensor data, the cloud would send a signal to the microcontroller used in the farm to turn on the motor pump. Explain how you can build such a system using the concept we have seen in this experiment. Use a proper block diagram to draw your system.

Problem 1 Explanation:

The ESP32 communicates with the **Arduino IoT Cloud** to monitor soil moisture levels in an agricultural farm and control a motor pump based on the sensor readings. The system uses Wi-Fi to connect the ESP32 to the cloud, where moisture data is sent periodically. If the soil moisture level drops below a defined threshold, the cloud sends a signal back to the ESP32, instructing it to turn on the motor pump via a relay module. The current system status, including moisture level and motor pump activity, is displayed on an **LCD screen**.

System Components:

1. **ESP32 Microcontroller**:

    - Handles all communication with the Arduino IoT Cloud, reads the soil moisture data from the sensor, and controls the motor pump.
    - Built-in Wi-Fi enables the ESP32 to connect to the cloud and send/receive data over the internet.

2. **Soil Moisture Sensor**:

    - Measures the moisture level in the soil by providing an analog signal. The ESP32 reads this signal to determine if the soil is too dry, triggering the pump control logic.

3. **Relay Module**:

    - Acts as a switch to control the motor pump. The relay is controlled by a signal from the ESP32, turning the pump on or off based on commands from the cloud.

4. **Motor Pump**:
    - This water pump is activated when the soil moisture is below the set threshold, providing irrigation to the farm.
5. **Liquid Crystal Display (LCD)**:

    - Displays real-time data such as the moisture level and whether the pump is running (ON) or not (OFF).
    - The **LiquidCrystal_I2C** library is used for communication with the LCD, which connects via the I2C protocol.

6. **Power Supply**:

    - Powers all components, ensuring the ESP32, relay, and sensor function properly.

Software Explanation:

1. **Wi-Fi Connection**:

    - The ESP32 connects to a Wi-Fi network using the WiFi.h library. This allows it to communicate with the Arduino IoT Cloud and update moisture levels in real-time.

2. **Arduino IoT Cloud**:

    - The ArduinoIoTCloud.h library is used to send sensor data to the cloud and receive commands for motor control. Cloud properties like moistureLevel (for the sensor) and motorPump (for controlling the pump) are defined and synchronized.

3. **Sensor Data Collection**:

    - The ESP32 reads the moisture sensor data using analogRead() from the defined pin. The reading is processed and sent to the cloud every few seconds.

4. **Motor Pump Control**:

    - The motorPumpControl() function is triggered whenever the motorPump variable changes in the cloud. This function activates or deactivates the relay, which controls the motor pump.

5. **LCD Display**:
    - The **LCD** shows real-time sensor readings and pump status using the **LiquidCrystal_I2C** library. It updates continuously, providing local feedback on system status.

How the System Works:

1. **Moisture Sensing**:
    - The soil moisture sensor is inserted into the soil and connected to the ESP32's **analog input pin**. The sensor provides a reading of the moisture level in the form of a voltage signal.
2. **Cloud Communication**:
    - The ESP32 connects to the internet using its **Wi-Fi module** and communicates with the **Arduino IoT Cloud**. It continuously sends moisture data from the sensor to the cloud at regular intervals (every 5 seconds in this example).
3. **Decision-Making in the Cloud**:
    - The Arduino IoT Cloud analyzes the sensor data. If the moisture level is below a predefined threshold, the cloud sends a command to the ESP32 to turn on the motor pump. This is achieved by updating the cloud property motorPump, which is synchronized with the ESP32.
4. **Motor Control**:
    - The ESP32 uses the **relay module** to control the motor pump. When the cloud sets the motorPump variable to **true**, the ESP32 activates the relay, which switches the motor pump on. If the cloud sets motorPump to **false**, the relay is deactivated, turning off the pump.
5. **LCD Feedback**:
    - The **LCD display** shows real-time information on the current soil moisture level and the status of the motor pump (whether it's ON or OFF). This allows a user in the field to see the system's state without needing to check the cloud dashboard.

Code:

```
#include "thingProperties.h"

#include <WiFi.h

#include <ArduinoIoTCloud.h>

#include <Wire.h>

#include <Adafruit_Sensor.h

#include <LiquidCrystal_I2C.h>

// Define Wi-Fi credentials

const char SSID[] = "Atiqur";

const char PASS[] = "09870987";
```

```
// Pin definitions
#define MOISTURE_SENSOR_PIN 34          // Pin for soil moisture sensor
#define RELAY_PIN 4                     // Pin connected to the relay module



// Variables for controlling moisture and motor pump
int moistureLevel;                      // Variable for moisture sensor value
bool motorPump;                         // Cloud-controlled variable for motor status
unsigned long lastSync = 0;             // Used to track time for cloud update
const unsigned long PUBLISH_INTERVAL = 5000; // Time interval for publishing data


LiquidCrystal_I2C lcd(0x27, 16, 2);     // LCD object for a 16x2 display


void setup() {
  Serial.begin(9600);                   // Initialize serial communication
  lcd.begin();                          // Initialize LCD display
  lcd.backlight();                      // Turn on the backlight of the LCD


  // Set up pins for sensor and relay
  pinMode(RELAY_PIN, OUTPUT);
  pinMode(MOISTURE_SENSOR_PIN, INPUT);


  // Connect to Wi-Fi
  WiFi.begin(SSID, PASS);               // Begin Wi-Fi connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");                  // Print dots while waiting for Wi-Fi connection
  }
   Serial.println("Connected to Wi-Fi!");
  // Initialize the Arduino IoT Cloud connection
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);              // Set debug level
  ArduinoCloud.printDebugInfo();
// Define properties for the cloud
  ArduinoCloud.addProperty(moistureLevel, READ, ON_CHANGE, NULL);  // Moisture level
sent to the cloud
```

```
ArduinoCloud.addProperty(motorPump, READWRITE, ON_CHANGE, motorPumpControl);  // Motor pump status

  // Ensure motor pump is off initially

  digitalWrite(RELAY_PIN, LOW);

}

void loop() {

  // Update cloud data and connections

  ArduinoCloud.update();

  // Read moisture sensor data

  int moisture = analogRead(MOISTURE_SENSOR_PIN);

  // Publish moisture level to Arduino IoT Cloud periodically

  if (millis() - lastSync > PUBLISH_INTERVAL) {

    moistureLevel = moisture;           // Update cloud with the new moisture level

    ArduinoCloud.update();

    lastSync = millis();

  }

  // Display data on the LCD

  lcd.setCursor(0, 0);                  // Set cursor to the first row

  lcd.print("Moisture: ");

  lcd.print(moisture);                  // Print moisture level

  lcd.setCursor(0, 1);                  // Set cursor to the second row

  lcd.print(motorPump ? "Pump: ON " : "Pump: OFF"); // Show motor pump status

}

int readMoistureSensor() {

  // Function to read moisture sensor data

  return analogRead(MOISTURE_SENSOR_PIN); // Reading from the sensor pin

}

void motorPumpControl() {

  // Callback function to control the motor pump based on cloud instructions

  if (motorPump) {

    digitalWrite(RELAY_PIN, HIGH);      // Turn on the motor pump

    Serial.println("Motor Pump ON");

  } else {

    digitalWrite(RELAY_PIN, LOW);       // Turn off the motor pump

    Serial.println("Motor Pump OFF");

  }
```

```
  // Display the pump status on the LCD as well
  lcd.setCursor(0, 1);                // Set cursor to second row
  lcd.print(motorPump ? "Pump: ON " : "Pump: OFF");
}
```

1. Setup the Development Environment:

- **Install Arduino IDE:** Begin by installing the Arduino IDE on your computer. This software is used for writing, compiling, and uploading code to the ESP32.
- **Add ESP32 Board Support:** Configure the IDE to support ESP32 by adding the necessary board URLs and installing the ESP32 package from the Board Manager.
- **Install Required Libraries:** Ensure the necessary libraries for Wi-Fi, Arduino IoT Cloud, and LCD control are installed through the Library Manager in the Arduino IDE.

2. Prepare the Hardware:

- **Assemble Components:** Gather all required components, including the ESP32 microcontroller, soil moisture sensor, relay module, LCD display, and power supply.
- **Connect Sensor and Relay:** Wire the soil moisture sensor to an analog pin on the ESP32 and connect the relay module's control pin to a digital GPIO pin on the ESP32.
- **Setup LCD Display:** Connect the LCD to the ESP32 using the I2C interface (SDA and SCL pins).
- **Power Supply:** Ensure all components are properly powered, typically with a 5V source.

3. Writing the Code:

- **Library Inclusions:** The code includes libraries to handle Wi-Fi connectivity, cloud communication, I2C communication with the LCD, and control functions.
- **Variable Declarations:** Variables are defined for sensor readings, relay control, and LCD management.
- **Setup Function:**

  Initializes the serial communication, LCD display, and hardware pins.

  - Connects to the Wi-Fi network and the Arduino IoT Cloud.
  - Configures cloud properties to sync sensor data and motor control.
- **Loop Function:**
  - Continuously updates cloud properties with the latest sensor data.
  - Reads moisture levels from the sensor and publishes the data to the cloud at regular intervals.

    - Updates the LCD display with the current moisture level and pump status.
- **Callback Function:**
    - Handles changes in cloud properties (e.g., motor control commands) and updates the relay state accordingly to turn the motor pump ON or OFF.

4. Upload the Code:

- **Prepare for Upload:** Connect the ESP32 to your computer via USB. Ensure the correct board and port settings are selected in the Arduino IDE.
- **Upload Process:** Compile and upload the code to the ESP32, transferring the instructions that control the system's operations.

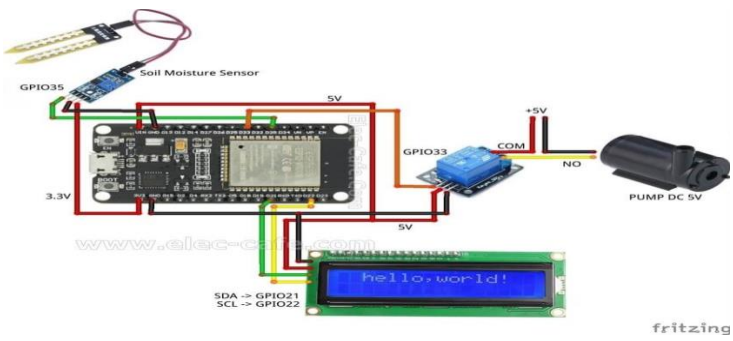
5. Test the System:

- **Verify Connections:** Ensure that the ESP32 successfully connects to the Wi-Fi and communicates with the Arduino IoT Cloud.
- **Check Sensor Data:** Confirm that the moisture sensor data is correctly read and displayed on the LCD.
- **Test Motor Control:** Verify that the relay properly controls the motor pump based on commands received from the cloud.
- **Validate LCD Display:** Ensure the LCD accurately shows real-time moisture levels and pump status.

6. Monitor and Adjust:

- **Monitor Performance:** Regularly check the system’s functionality by observing LCD outputs and relay operations.
- **Make Adjustments:** If any discrepancies are found (e.g., incorrect data readings or motor control issues), review the code and hardware connections, and make necessary adjustments to improve performance.

Block Diagram:



Discussion:

When solving these problems, I gained a deeper understanding of integrating various hardware components with the ESP32 to create a functional IoT system. I learned how to manage cloud communication, handle sensor data, and control peripherals like relays and LCD displays using libraries and coding practices. One significant challenge was ensuring reliable Wi-Fi connectivity and seamless communication between the ESP32 and the Arduino IoT Cloud, which required troubleshooting connection issues and optimizing data synchronization. Additionally, configuring the LCD display for real-time feedback and ensuring correct relay operation posed initial difficulties, but these were overcome with careful testing and adjustments. Overall, the experience highlighted the importance of meticulous hardware setup and robust code structure for successful IoT project implementation.

Problem 2

Write a program to send “Hello!” and “CSE 4326!” using WIFI to the cloud and then read the data from the cloud. A green and red led light is connected to another ESP32. If “Hello!” is sent then a green led would turn on and if “CSE 4326!” is sent then the red led would turn on

Problem 2 Explanation:

In this problem, two **ESP32** microcontrollers communicate via the Arduino IoT Cloud. The first ESP32 sends specific messages ("Hello!" or "CSE 4326!") over Wi-Fi to the cloud. The second ESP32 reads these messages from the cloud and activates the corresponding LEDs: **Green LED** for "Hello!" and **Red LED** for "CSE 4326!". This system allows real-time communication between two ESP32 devices using the cloud, with LED indicators providing visual feedback on the received messages.

System Components:

1. **ESP32 Microcontroller (Sender)**:
    - This ESP32 sends messages, either "Hello!" or "CSE 4326!", to the Arduino IoT Cloud.
    - It connects to the Wi-Fi network and synchronizes the message property with the cloud.
2. **ESP32 Microcontroller (Receiver)**:
    - This ESP32 reads the messages from the cloud and controls the LEDs based on the received message.
    - Like the sender, it connects to the Wi-Fi network and the Arduino IoT Cloud to receive updates.
3. **Green and Red LEDs**:
    - The **Green LED** lights up when the message "Hello!" is received.
    - The **Red LED** lights up when the message "CSE 4326!" is received.
4. **Resistors (220Ω)**:

- These resistors are connected in series with the LEDs to limit the current, protecting them from damage.

5. **Wi-Fi Network**:
    - This provides the communication link between the ESP32 microcontrollers and the Arduino IoT Cloud, enabling them to send and receive data.
6. **Power Supply**:
    - The ESP32 devices and the LEDs are powered through the USB connection to a computer or external power supply.

Software Implementation:

1. **Wi-Fi Connection**:
    - Both ESP32 microcontrollers connect to the Wi-Fi network using the WiFi.h library, allowing communication with the Arduino IoT Cloud.
2. **Arduino IoT Cloud**:
    - The ArduinoIoTCloud.h library is used to manage cloud properties. The **Sender** ESP32 sends a message to the cloud, and the **Receiver** ESP32 reads the cloud property to determine which LED to activate.
3. **Message Sending (Sender ESP32)**:
    - The sender ESP32 sends either "Hello!" or "CSE 4326!" to the cloud based on predefined conditions or intervals, updating the cloud property with the new message.
4. **Message Receiving and LED Control (Receiver ESP32)**:
    - The receiver ESP32 reads the message from the cloud. If the message is "Hello!", it activates the Green LED. If the message is "CSE 4326!", it activates the Red LED.
5. **Cloud Synchronization**:
    - Both ESP32 devices synchronize in real-time with the cloud. The sender ESP32 updates the message property, and the receiver ESP32 responds by activating the appropriate LED based on the message.

How the System Works:

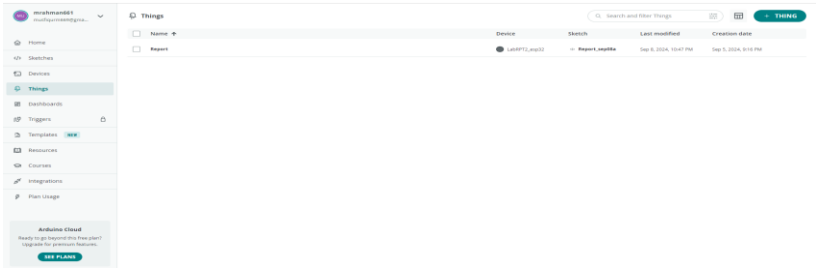
1. **Message Sending (Sender ESP32)**:
    - The sender ESP32 connects to Wi-Fi and the Arduino IoT Cloud. It periodically or conditionally sends a message, "Hello!" or "CSE 4326!", to a cloud property that is accessible to both devices.
2. **Message Receiving and LED Control (Receiver ESP32)**:
    - The receiver ESP32 constantly monitors the cloud property. When a new message is detected, it checks whether the message is "Hello!" or "CSE 4326!" and activates the corresponding LED (Green for "Hello!" and Red for "CSE 4326!").

3. **Wi-Fi and Cloud Communication**:
    - The two ESP32 devices communicate through the cloud, with the cloud acting as an intermediary. The sender updates the cloud property with a message, and the receiver reads this property to control the LEDs in real-time.

Step-by-Step Procedure:

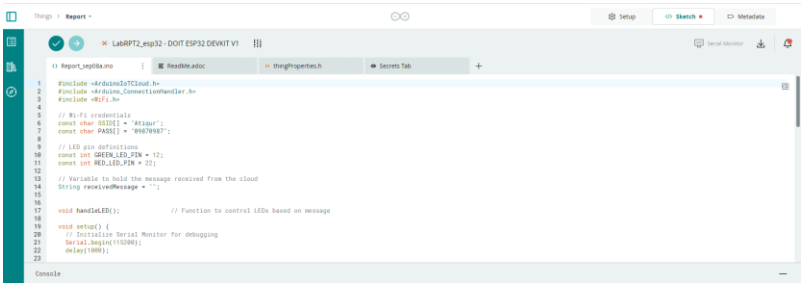
1. **Setup the Development Environment**:
    - Install the Arduino IDE on your computer and add support for ESP32 boards by following the official instructions.
    - Install the necessary libraries (WiFi.h, ArduinoIoTCloud.h, etc.) using the Library Manager in the Arduino IDE.

    Create Things-



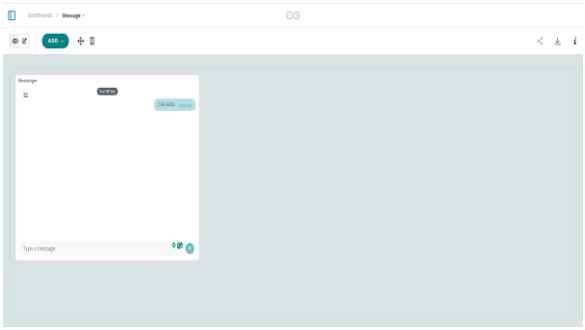
2. **Prepare the Hardware**:
    - **ESP32 Sender**: Connect the ESP32 to your computer via USB for programming.
    - **ESP32 Receiver**: Connect two LEDs (Green and Red) to GPIO pins on the ESP32 with 220Ω resistors to limit current.
    - Ensure both ESP32 devices are powered via USB and have access to the Wi-Fi network.
3. **Writing the Code**:
    - **Sender ESP32**: Program this ESP32 to send messages ("Hello!" or "CSE 4326!") to the cloud using the Arduino IoT Cloud dashboard.
    - **Receiver ESP32**: Program this ESP32 to read the messages from the cloud and control the LEDs based on the received message.
    - Both ESP32 devices use a **loop function** to continuously update the cloud (sending and receiving messages).
    - The receiver has a **callback function** to handle cloud property changes and activate the appropriate LED.
4. **Upload the Code**:
    - Connect both ESP32 devices to the computer via USB.
    - In the Arduino IDE, select the correct board and port for each ESP32, and upload the respective code to the sender and receiver devices.

Upload Code-



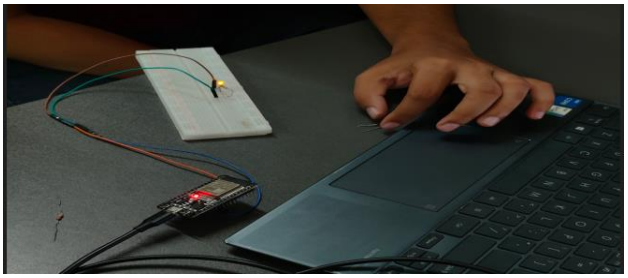
5. **Test the System**:
    - For the **Sender ESP32**, verify that it sends the correct messages ("Hello!" or "CSE 4326!") to the cloud.
    - For the **Receiver ESP32**, check that the correct LED (Green or Red) lights up based on the message received from the cloud.
    - Monitor the **serial output** to ensure the messages are being sent and received as expected.

    Test-



6. **Monitor and Adjust**:
    - Regularly check the behavior of the LEDs and the messages from the cloud to ensure the system is working in real-time.
    - If any issues arise (e.g., incorrect message handling, LED behavior), adjust the Wi-Fi connection, LED wiring, or code logic accordingly to resolve the problem.

    Output-



Code:

```
#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>
#include <WiFi.h>

// Wi-Fi credentials
const char SSID[] = "Atiqur";
const char PASS[] = "09870987";
// LED pin definitions
const int GREEN_LED_PIN = 12;
const int RED_LED_PIN = 22;

// Variable to hold the message received from the cloud
String receivedMessage = "";

void handleLED();              // Function to control LEDs based on message
void setup() {
  // Initialize Serial Monitor for debugging
  Serial.begin(115200);
  delay(1000);
  // Initialize the LEDs and set them to off initially
  pinMode(GREEN_LED_PIN, OUTPUT);
  pinMode(RED_LED_PIN, OUTPUT);
  digitalWrite(GREEN_LED_PIN, LOW);
  digitalWrite(RED_LED_PIN, LOW);

  // Connect to Wi-Fi
  WiFi.begin(SSID, PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");         // Print dots while waiting for Wi-Fi connection
  }
  Serial.println("\nConnected to Wi-Fi!");

  // Initialize the Arduino IoT Cloud connection and properties
```

```
ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();

  // Set the cloud-synced property for message handling
  ArduinoCloud.addProperty(receivedMessage, READWRITE, ON_CHANGE,
onMessageChange);
}

void loop() {
  // Keep the IoT Cloud connection alive and updated
  ArduinoCloud.update();
}

// Callback function for when the cloud-synced message changes
void onMessageChange() {
  Serial.println("Received from cloud: " + receivedMessage);

  // Handle the LED control based on the message received
  handleLED();
}

// Function to handle LED control based on the received message
void handleLED() {
  // If message is "Hello!", turn on green LED, off red
  if (receivedMessage.equals("Hello!")) {
    digitalWrite(GREEN_LED_PIN, HIGH);  // Green LED ON
    digitalWrite(RED_LED_PIN, LOW);     // Red LED OFF
    Serial.println("Green LED ON");
  }
  // If message is "CSE 4326!", turn on red LED, off green
  else if (receivedMessage.equals("CSE 4326!")) {
    digitalWrite(GREEN_LED_PIN, LOW);   // Green LED OFF
    digitalWrite(RED_LED_PIN, HIGH);    // Red LED ON
    Serial.println("Red LED ON");
  }
```

```
// For any other message, turn off both LEDs
  else {
    digitalWrite(GREEN_LED_PIN, LOW);
    digitalWrite(RED_LED_PIN, LOW);
    Serial.println("Both LEDs OFF");
  }
}
```

Discussion:

While working on this project, I learned how to integrate **ESP32 microcontrollers** with the **Arduino IoT Cloud** for real-time communication. I gained experience in handling cloud properties, sending and receiving data, and managing hardware components like LEDs based on cloud-synced messages. One challenge I faced was ensuring **stable Wi-Fi connectivity**, as connection drops could disrupt message synchronization between the devices. Additionally, configuring the correct **GPIO pins** for the LEDs was critical to avoid conflicts with the ESP32's default functionalities. Debugging through the **Serial Monitor** helped in identifying issues with connectivity and message handling, ultimately improving the system's reliability.

Problem 3

A gas sensor is connected in your kitchen to measure the gas in ppm. The sensor is connected to ESP32 to send the data in the cloud. Now, write the necessary programs to save the gas data from the cloud in a google sheet or excel file. Then, read from the Excel file and turn on a buzzer if the gas value crosses 50 ppm

Problem 3 Explanation:

In this problem, we need to create a local server on an ESP32 with a web interface that includes the:

1. Button Switch Widget: A button to turn a light ON or OFF.
2. LED Status Widget: An LED image that shows the current status of the light (ON or OFF).

Challenges include setting up the web interface, handling HTTP requests, real-time status updates for the LED, and managing communication between the ESP32 and the web page for smooth control and feedback.

ESP32 Web Server with Button and LED Status:

Components and Tools Required

- ESP32 Development Board
- LED (to simulate the room light)
- Resistor (220 ohms for the LED)
- Jumper wires
- Breadboard

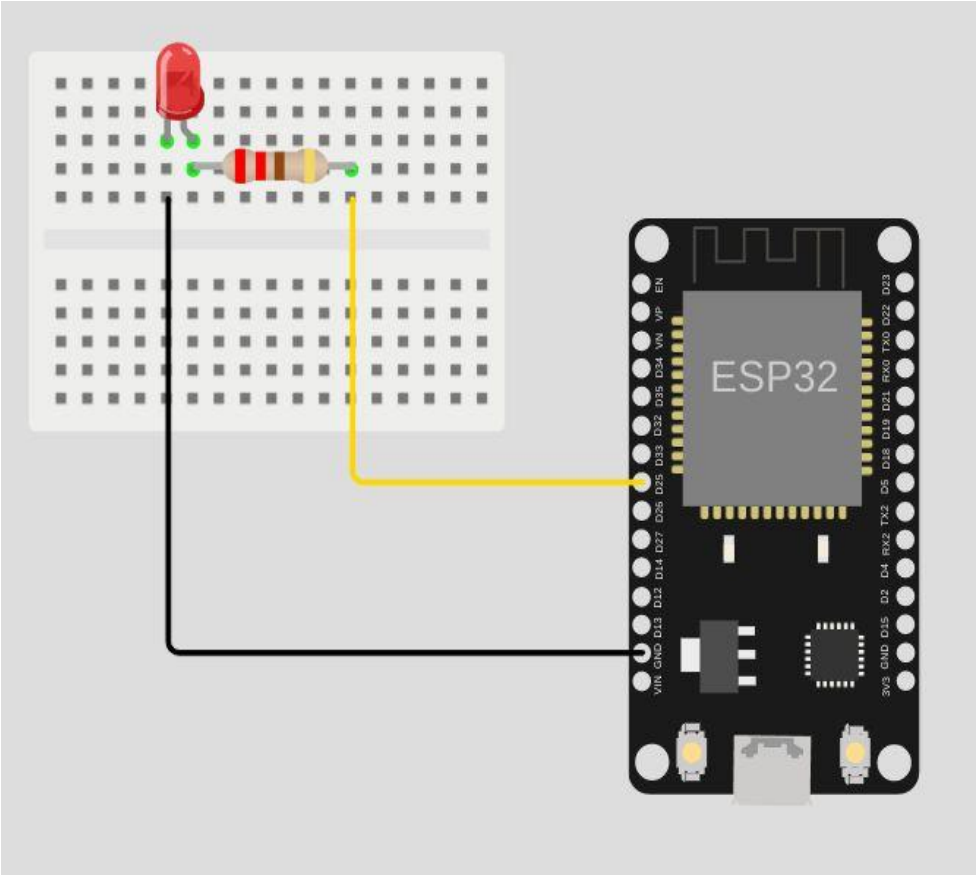
- Arduino IDE (with ESP32 board package installed)

Overview

- Button Switch: A button that allows users to turn the LED (representing a room light) ON and OFF.
- LED Status Indicator: An image or symbol that changes according to the LED state. If the LED is ON, the symbol will be highlighted (green), and if OFF, it will be dim (red).

Wiring the Circuit:

- Connect the positive leg (anode) of the LED to GPIO pin 2 of the ESP32.
- Connect the negative leg (cathode) to the ground (GND) via a 220-ohm resistor.



ESP32 Code

```
#include <WiFi.h>

#include <WebServer.h>


// Replace with your network credentials

const char* ssid = "mahbub61";

const char* password = "67677676";


// Define the LED GPIO

const int ledPin = 2;

bool ledState = false;


// Create an instance of the server

WebServer server(80);


String generateHTML() {

  String html = "<!DOCTYPE html><html>";

  html += "<head><meta name='viewport' content='width=device-width, initial-scale=1.0'>";

  html += "<title>ESP32 LED Control</title>";

  html += "<style>body { text-align: center; font-family: Arial; }";

  html += ".led-status { font-size: 24px; margin: 10px; }";

  html += ".button { font-size: 20px; padding: 10px 20px; background-color: #008CBA; color: white; border: none; cursor: pointer; }";

  html += ".button:hover { background-color: #005f6b; }";

  html += "</style></head><body>";
```

```
  html += "<p class='led-status'>LED is currently: ";
  html += (ledState ? "<strong>ON</strong>" : "<strong>OFF</strong>");
  html += "</p>";

  html += "<form action='/toggle' method='POST'>";
  html += "<button class='button' type='submit'>";
  html += (ledState ? "Turn OFF" : "Turn ON");
  html += "</button></form>";

  html += "</body></html>";

  return html;
}

// Handle root URL ("/") requests
void handleRoot() {
  server.send(200, "text/html", generateHTML());
}

// Handle LED toggle requests
void handleToggle() {
  ledState = !ledState;
  digitalWrite(ledPin, ledState ? HIGH : LOW);
  server.send(200, "text/html", generateHTML());
}

void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  html += "<h1>ESP32 LED Control</h1>“; }
```

```
 Serial.println("Connected to WiFi");
  Serial.println(WiFi.localIP());


  // Set up server routes
  server.on("/", handleRoot);
  server.on("/toggle", HTTP_POST, handleToggle);


  // Start the server
  server.begin();
}


void loop() {
  // Handle incoming client requests
  server.handleClient();
}
```

Code Explanation:

1. Wi – fi connection setup:

The WiFi.begin(ssid, password); function connects the ESP32 to the specified Wi-Fi network. The while (WiFi.status() != WL_CONNECTED) loop waits until the connection is established, printing the IP address to the serial monitor.

2. Web Server Setup:

We create an instance of Web Server on port 80 (default HTTP port) and define routes for the webpage ("/") and a button action ("/toggle").

3. HTML Page Generation:

The generateHTML() function returns the HTML content that is displayed in the browser. It contains:
A title.
A paragraph showing the current LED state.
A form with a button to toggle the LED.

4. Handling Client Requests:

Root Route (/): Displays the current state of the LED with a toggle button.

Toggle Route (/toggle): Toggles the LED state between ON and OFF and updates the LED symbol.

5. **LED Control**

The GPIO pin 2 is used to control the LED. If the LED is ON, the symbol and the button change accordingly.

How it works

User Interface:

When you access the ESP32's IP address in a web browser, the ESP32 serves the HTML page containing the button and LED status. The button changes dynamically depending on whether the LED is ON or OFF.

Interactivity:

When the button is pressed, a POST request is sent to /toggle, which flips the LED's state and refreshes the page with the updated LED status.

Testing the Server:

Open the Serial Monitor in the Arduino IDE to view the ESP32's IP address after it connects to the Wi-Fi network.

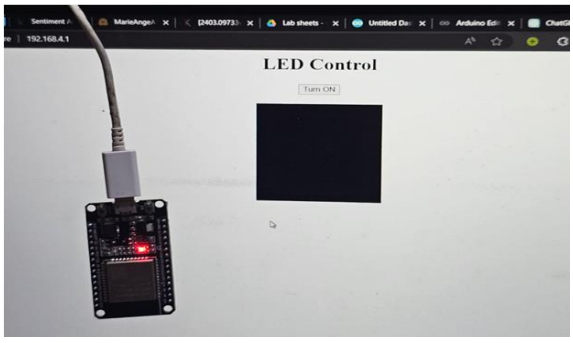
In a browser, type the IP address, and the webpage will load.

Use the button on the webpage to control the LED, and observe the LED symbol update accordingly.

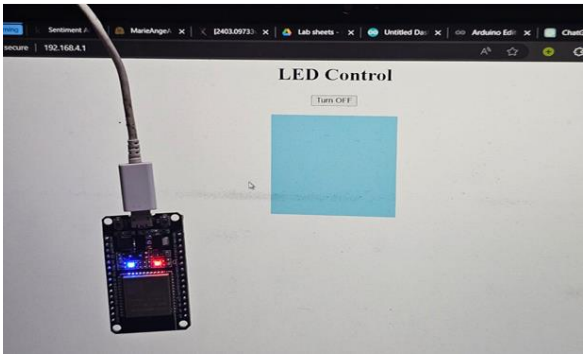
Discussion:

By following the steps outlined in this report, an ESP32 can be used to create a local web server that allows real-time control and monitoring of an LED (simulating room lights). The server handles both the LED state and its representation through an intuitive web interface. This project demonstrates how easily the ESP32 can be integrated into home automation systems using its built-in web server capabilities.

Before Light ON:



After Light ON:



Problem 4

A gas sensor is connected in your kitchen to measure the gas in ppm. The sensor is connected to ESP32 to send the data in the cloud. Now, write the necessary programs to save the gas data from the cloud in a google sheet or excel file. Then, read from the Excel file and turn on a buzzer if the gas value crosses 50 ppm.

Problem 4 Explanation:

This project involves connecting a gas sensor to an ESP32 to monitor gas concentration in the kitchen, sending the gas data to the cloud (Google Sheets), and controlling a buzzer if the gas value crosses a threshold (50 ppm). We'll walk through the entire process of sending data to Google Sheets and reading from it using an external application (e.g., a Python script) to control the buzzer based on the gas concentration.

Components and Tools Required

- ESP32 Development Board
- Gas Sensor (e.g., MQ-2 or MQ-135)
- Buzzer
- Resistor (for the buzzer if required)
- Jumper wires
- Breadboard
- Arduino IDE
- Google Account (for Google Sheets API)
- Python Environment (for reading data and controlling the buzzer)

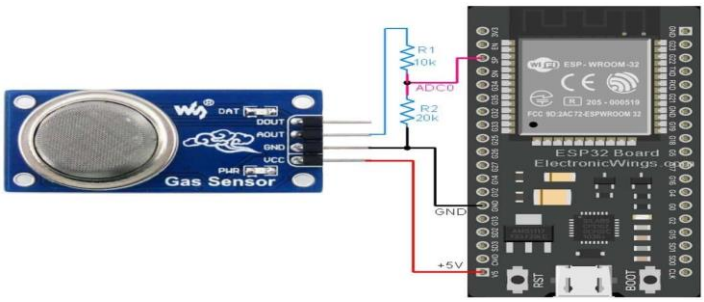
Project Overview

- The gas sensor is connected to the ESP32, which reads gas levels and sends the data to Google Sheets.
- A Python script is used to fetch the data from Google Sheets and analyze it.
- If the gas value exceeds 50 ppm, the script activates the buzzer.

Step 1: Setting Up the Gas Sensor with ESP32

First, connect the gas sensor to the ESP32 to read the gas concentration. The analog output from the sensor will provide gas readings, which can be mapped to ppm.

Circuit Diagram



Step 2: Code for Sending Data to Google Sheet

```
#include <WiFi.h>

#include <HTTPClient.h>

const char* ssid = "mahbub61";

const char* password = "67677676";

const char* ifttt_event = "gas_alert";

const char* ifttt_key = "mahbubpart_4";

const int gasSensorPin = 34;  // Gas sensor connected to GPIO 34

int gasValue = 0;           // Variable to store gas sensor reading

void setup() {

 Serial.begin(115200);

 pinMode(gasSensorPin, INPUT);

 // Connect to Wi-Fi

 WiFi.begin(ssid, password);
```

```
while (WiFi.status() != WL_CONNECTED) {
   delay(1000);
   Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
}


void sendToIFTTT(int gasPPM) {
  if (WiFi.status() == WL_CONNECTED) {
   HTTPClient http;
   String url = "http://maker.ifttt.com/trigger/" + String(ifttt_event) +
"/with/key/" + String(ifttt_key);
   url += "?value1=" + String(gasPPM);
   http.begin(url);
   int httpResponseCode = http.GET();
   if (httpResponseCode > 0) {
    Serial.println("Data sent to IFTTT: " + String(gasPPM) + " ppm");
   } else {
    Serial.println("Error sending data to IFTTT");
   }
   http.end();
  } else {
   Serial.println("WiFi Disconnected");
  }
}
  delay(6000data every 60 seconds
}
```

```
void loop() {
  gasValue = analogRead(gasSensorPin);
  int gasPPM = map(gasValue, 0, 4095, 0, 1000); // Convert to ppm (adjust based on sensor)

Serial.println("Gas PPM: " + String(gasPPM));

  // Send the gas data to Google Sheets via IFTTT
  sendToIFTTT(gasPPM);

  delay(60000);  // Send data every 60 seconds
}
```

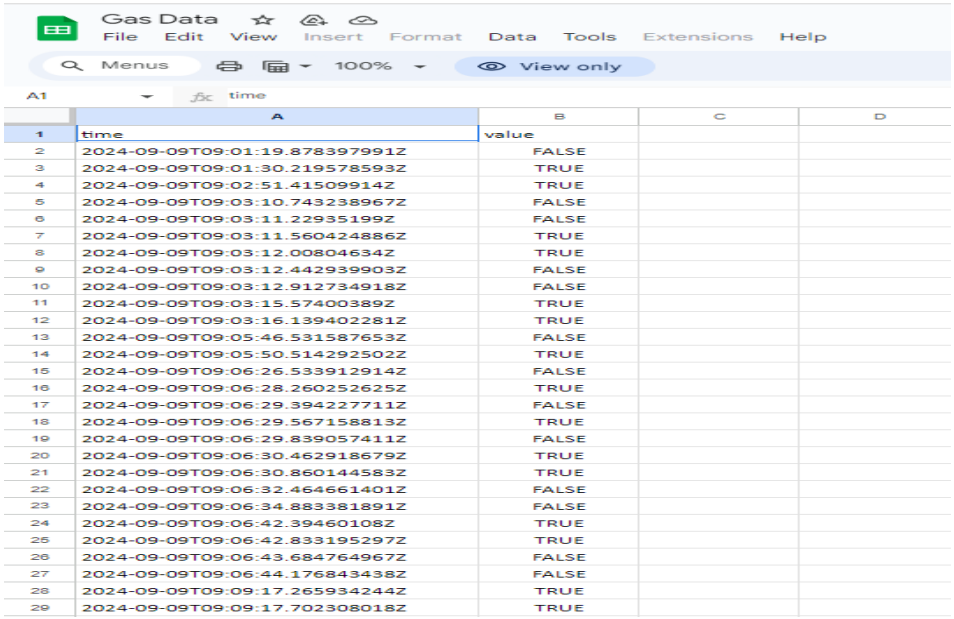
Explanation:

- WiFi Connection: The ESP32 connects to Wi-Fi using the WiFi.begin() function.
- Gas Sensor Reading: The gas sensor is read from the analog pin, and the value is mapped to ppm using the map() function.
- Sending Data to IFTTT: The gas concentration is sent to Google Sheets via the IFTTT Webhooks service, where the HTTPClient library is used to make an HTTP GET request.

Step 3: Google Sheets Setup

The gas data is now being saved to Google Sheets. Each row contains the gas concentration in ppm along with the timestamp of when the data was sent.

- Go to your Google Drive and open the Google Sheets document created by IFTTT.
- You'll see columns for timestamps and the gas sensor readings in ppm.



	A	B	C	D
1	time	value		
2	2024-09-09T09:01:19.878397991Z	FALSE		
3	2024-09-09T09:01:30.219578593Z	TRUE		
4	2024-09-09T09:02:51.41509914Z	TRUE		
5	2024-09-09T09:03:10.743238967Z	FALSE		
6	2024-09-09T09:03:11.22935199Z	FALSE		
7	2024-09-09T09:03:11.5604248867	TRUE		
8	2024-09-09T09:03:12.008046347	TRUE		
9	2024-09-09T09:03:12.442939903Z	FALSE		
10	2024-09-09T09:03:12.912734918Z	FALSE		
11	2024-09-09T09:03:15.57400389Z	TRUE		
12	2024-09-09T09:03:16.139402281Z	TRUE		
13	2024-09-09T09:05:46.531587653Z	FALSE		
14	2024-09-09T09:05:50.514292502Z	TRUE		
15	2024-09-09T09:06:26.533912914Z	FALSE		
16	2024-09-09T09:06:28.260252625Z	TRUE		
17	2024-09-09T09:06:29.394227711Z	FALSE		
18	2024-09-09T09:06:29.567158813Z	TRUE		
19	2024-09-09T09:06:29.839057411Z	FALSE		
20	2024-09-09T09:06:30.462918679Z	TRUE		
21	2024-09-09T09:06:30.860144583Z	TRUE		
22	2024-09-09T09:06:32.464661401Z	FALSE		
23	2024-09-09T09:06:34.883381891Z	FALSE		
24	2024-09-09T09:06:42.39460108Z	TRUE		
25	2024-09-09T09:06:42.833195297Z	TRUE		
26	2024-09-09T09:06:43.684764967Z	FALSE		
27	2024-09-09T09:06:44.176843438Z	FALSE		
28	2024-09-09T09:09:17.265934244Z	TRUE		
29	2024-09-09T09:09:17.702308018Z	TRUE		

Step 4: Reading Data from Google Sheets and Controlling the Buzzer

Now, we'll write a Python script that reads the gas sensor data from Google Sheets and activates the buzzer if the gas level exceeds 50 ppm.

Python code:

```
import gspread

import serial

import time

from oauth2client.service_account import ServiceAccountCredentials


# Google Sheets API setup

scope = ["https://spreadsheets.google.com/feeds", "https://www.googleapis.com/auth/drive"]

creds = ServiceAccountCredentials.from_json_keyfile_name('credentials.json', scope)  # Use your credentials file

client = gspread.authorize(creds)


# Access the Google Sheet

sheet = client.open('Gas Data').sheet1  # Replace with your sheet name


# Serial communication with ESP32

esp32 = serial.Serial('COM3', 115200)  # Replace COM3 with the correct port
```

```
def check_gas_level():
 records = sheet.get_all_records()
latest_record = records[-1]  # Get the latest gas sensor reading
    gas_ppm = int(latest_record['Value1'])  # Assuming Value1 column stores gas data


   print(f"Latest Gas PPM: {gas_ppm}")
    if gas_ppm > 50:
    # Send signal to turn on the buzzer
esp32.write(b'1')  # Send '1' to ESP32 to activate buzzer
    else:
            # Turn off the buzzer
esp32.write(b'0')  # Send '0' to ESP32 to deactivate buzzer


while True:
check_gas_level()
 time.sleep(60)  # Check every 60 seconds
```

Explanation:

- Google Sheets API: The gspread library is used to access the Google Sheets API. It reads the latest gas reading from the sheet.
- Serial Communication: The pySerial library is used to communicate with the ESP32. Based on the gas concentration, the Python script sends commands to the ESP32 to turn the buzzer ON or OFF.

Step 5: ESP32 Code for Buzzer Control

```
const int buzzerPin = 5;


void setup() {
  Serial.begin(115200);
  pinMode(buzzerPin, OUTPUT);
```

```
  digitalWrite(buzzerPin, LOW);  // Turn off the buzzer initially
}

void loop() {
  if (Serial.available() > 0) {
    char command = Serial.read();
    if (command == '1') {
      digitalWrite(buzzerPin, HIGH);  // Turn on the buzzer
    } else if (command == '0') {
      digitalWrite(buzzerPin, LOW);   // Turn off the buzzer
    }
  }
}
```

Explanation:

The ESP32 reads data from the serial port. If it receives '1', it turns on the buzzer, and if it receives '0', it turns it off.

Discussion:

This project demonstrates how to monitor gas levels using a gas sensor connected to an ESP32, send the data to Google Sheets, and control a buzzer based on the data. The system is ideal for a kitchen gas monitoring setup, where a buzzer alerts you when gas concentration exceeds a dangerous level.

Data- here

Reference

1.https://youtu.be/RpQxJkEZ-fA?si=QxEr8Ii5WaoZK7kN

2.https://www.google.com/sheets/about/

3.https://docs.arduino.cc/arduino-cloud/