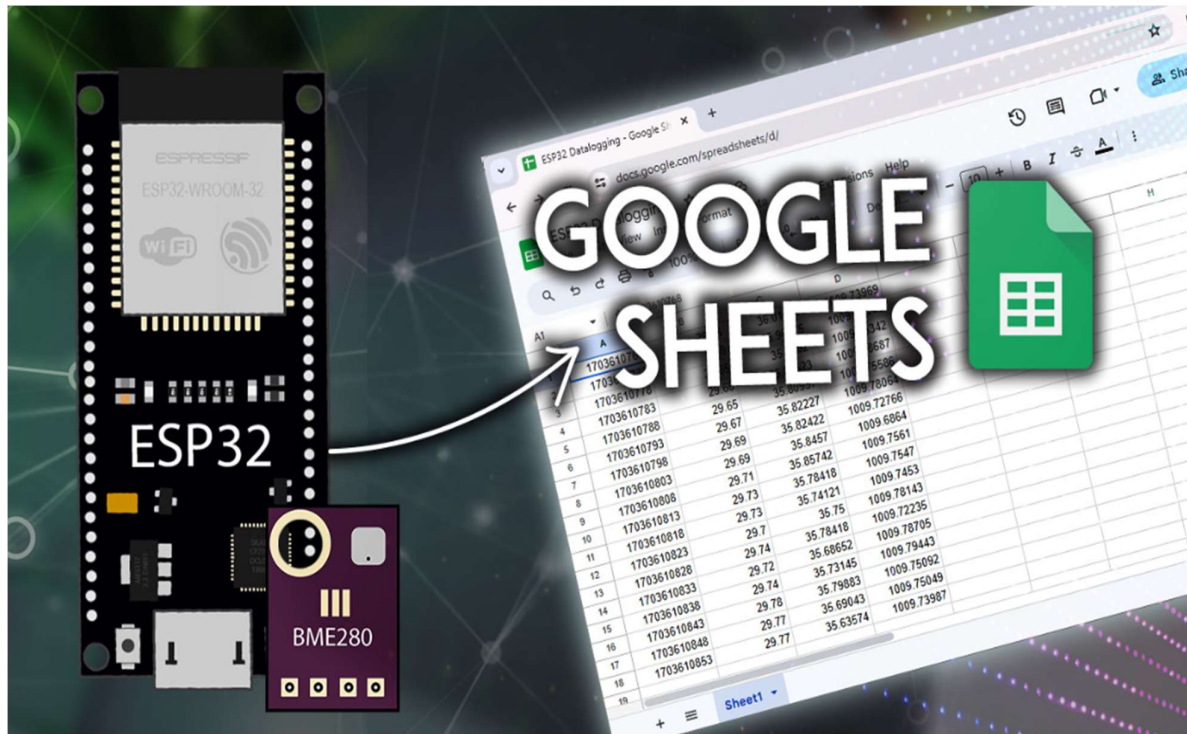


Design and Implementation of a Data Logging System Using ESP32



PROJECT DESCRIPTION:

The project involves the design and implementation of a data logging system using an ESP32 microcontroller, which captures data from analog and digital sensors and logs this data to Google Sheets at regular intervals. The system is intended to provide an efficient and scalable solution for real-time data monitoring and storage in cloud-based environments. The core of the project revolves around the ESP32 microcontroller, chosen for its integrated Wi-Fi capabilities and processing power, making it ideal for Internet of Things (IoT) applications. The microcontroller reads data from four analog inputs, which capture varying voltage levels, and four digital inputs, which detect binary states (HIGH/LOW). These inputs enable the system to monitor various environmental or operational parameters, depending on the types of sensors connected.

PROJECT OBJECTIVES:

Implement Data Logging Functionality: Develop firmware for the ESP32 to read data from four analog and four digital inputs and log this data every 30 seconds. 2. Establish Reliable Internet Connectivity: Use the ESP32's Wi-Fi capabilities to connect to the internet and communicate with Google Sheets API, ensuring stable and secure data transmission. 3. Develop Google Sheets Integration: Set up Google Sheets with appropriate columns and utilize the API to write data programmatically, handling authentication and authorization securely.

PROJECT GOALS:

1. Real-Time Data Monitoring: Develop a system that captures data from multiple input channels and logs it in real-time, providing an accurate and timely record of sensor readings.
2. Cloud-Based Data Storage: Utilize Google Sheets for storing data, ensuring easy access and integration with other cloud services and tools for further analysis and visualization.
3. Scalability and Flexibility: Design the system to be easily scalable, allowing additional sensors to be integrated with minimal modifications to the existing setup.
4. User-Friendly Interface: Ensure the system is straightforward to set up and operate, providing clear documentation and support for users with varying technical expertise.

Problem Statement

Your task is to design and implement a data logging system that records data on Google Sheets. The system will capture data from four analog inputs and four digital inputs, logging each input's data at a frequency of every 30 seconds. Each data entry row in Google Sheets will include the timestamp of the log, specifying the date and time of data collection.

Introduction to ESP32

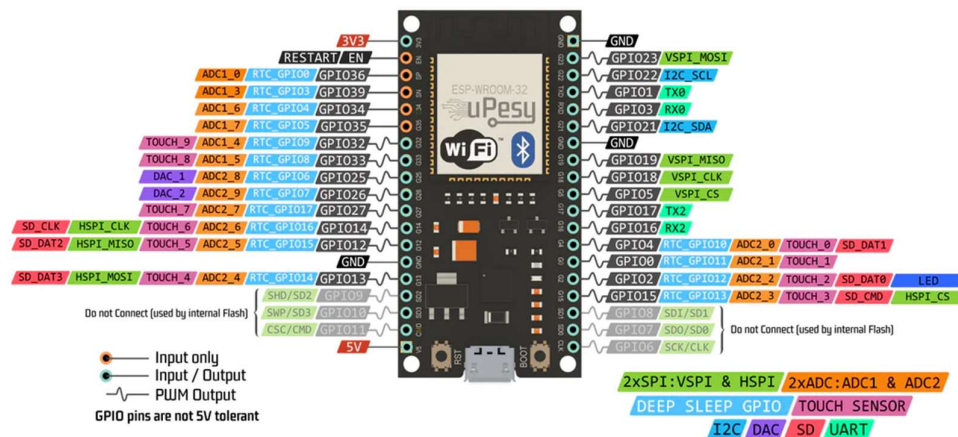
What is ESP32?

ESP32 is a powerful microcontroller with built-in Wi-Fi and Bluetooth capabilities, developed by Espressif Systems. It's widely used in IoT projects because of its versatility and low cost. The ESP32 can handle various peripherals like analog sensors, digital inputs, and communication protocols, making it ideal for data logging tasks.

Key Features of ESP32:

- **Dual-Core Processor:** Two Xtensa LX6 cores, providing strong computational power.
- **Built-in Wi-Fi and Bluetooth:** Enables easy communication with cloud services, mobile devices, and other IoT devices.
- **GPIO Pins:** Provides a range of General-Purpose Input/Output pins that can be configured for analog or digital input/output, PWM, I2C, SPI, etc.
- **ADC/DAC:** 12-bit Analog-to-Digital Converter (ADC) and 8-bit Digital-to-Analog Converter (DAC) for analog signal processing.
- **Large Memory:** Sufficient flash memory and RAM for complex tasks and data storage.

ESP32 Wroom DevKit Full Pinout



Pinout of an ESP32 board

Introduction to Google Sheets API

What is Google Sheets?

Google Sheets is an online spreadsheet application offered by Google. It allows users to create, edit, and share spreadsheets online. Google Sheets supports collaborative editing and can be integrated with other Google services via APIs.

Google Sheets API:

The Google Sheets API allows you to interact programmatically with Google Sheets. You can read, write, and update data in a Google Sheet from your applications. This is particularly useful for data logging projects where real-time data needs to be logged remotely.

Key Features of Google Sheets API:

- **Read Data:** Retrieve data from any cell or range in the sheet.
- **Write Data:** Programmatically enter data into specific cells or ranges.
- **Update Data:** Modify existing data in the sheet.
- **Authentication:** Secure API access via OAuth2.0, ensuring that only authorized applications can access or modify the sheet.

Setting up the Google Sheet with ESP32:

In a data logging system, ESP32 collects data from its analog and digital inputs. After collecting the data, it uses its Wi-Fi capability to send this data to Google Sheets via the Google Sheets API.

Step 1

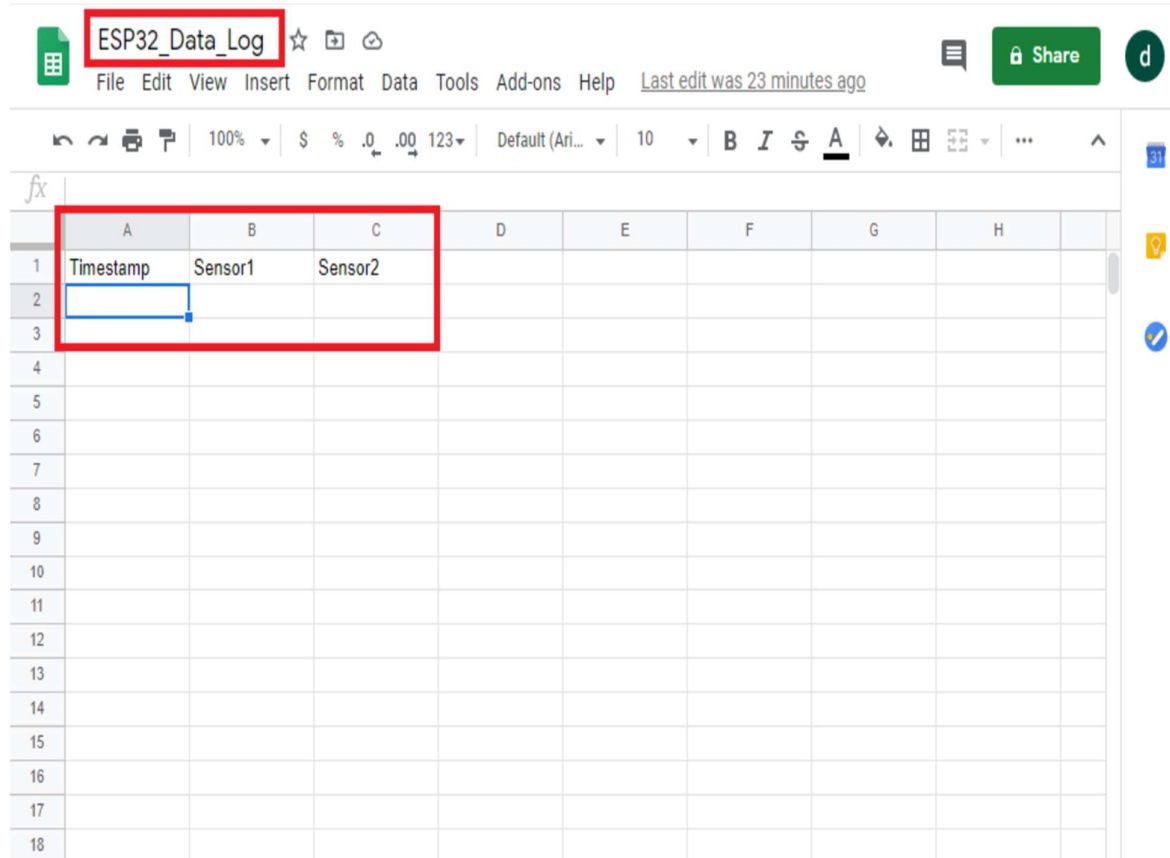
Log in to your Google account and open a blank Google Spread Sheet.

Step 2

Change the document name and the sheet name as preferred and create the required columns. In this example:

- Document name: ESP32_Data_log
- Sheet name: Sensor_Data
- Columns: Timestamp, Analog input 1, Analog input 2, Analog input 3, Analog input 4, Digital input 1, Digital input 2, Digital input 3, Digital input 4

CEA REPORT



Step 3

Now the Google sheets needs to be published as a web application which will be listening to web requests. For that we use Google Script. Now go to Extensions -> App Script. This will open up a new window to write the script. Copy and paste the code in the [g_script_code.txt](#) file to the Google Script editor. Save the script with a preferred name. (Example: ESP32_Data_Log_Script).

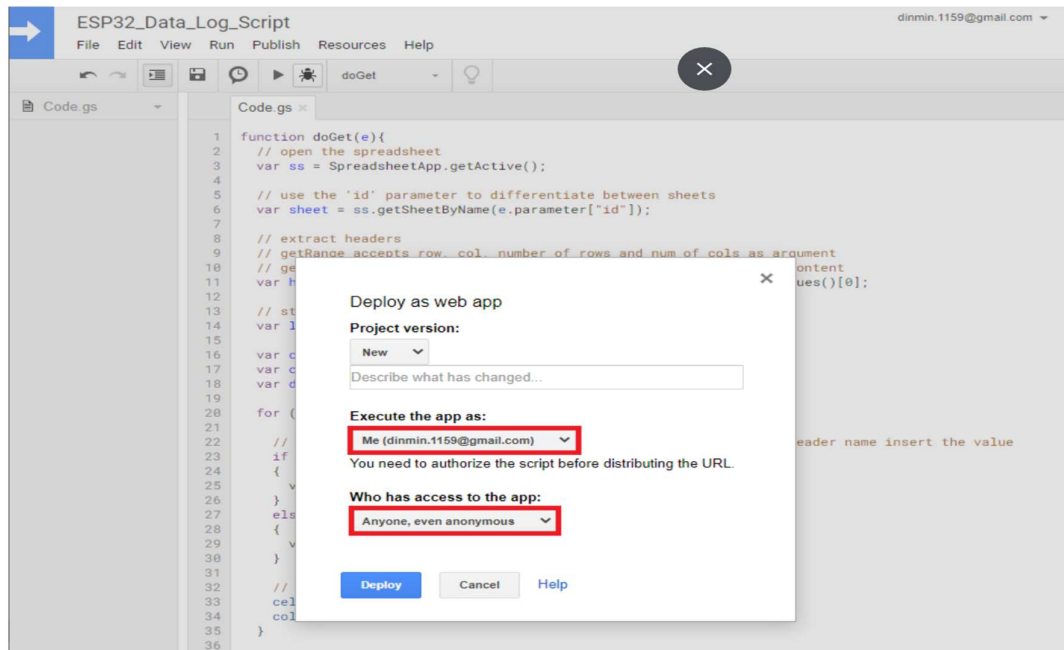
Step 4

Go to **Deploy -> New deployment**

Select deployment type as Web app.

In the pop-up window make the configurations as shown in the picture.

CEA REPORT



Click **Deploy**.

From the pop-up window click **Review Permissions**. This will open up a new windows After Signing in the page in the new window will re-direct to a warning page. There click **Advanced** and click **Go to yourscripname (unsafe)**.



This app isn't verified

This app hasn't been verified by Google yet. Only proceed if you know and trust the developer.

[Hide Advanced](#)

[BACK TO SAFETY](#)

Google hasn't reviewed this app yet and can't confirm it's authentic. Unverified apps may pose a threat to your personal data. [Learn more](#)

[Go to ESP32_Data_Log_Script \(unsafe\)](#)

Then the page in the new window will direct to another page where you will be asked for permission to run the app. Click **Allow**.

ESP32_Data_Log_Script wants to access your Google Account

 dinmin.1159@gmail.com

This will allow ESP32_Data_Log_Script to:

- See, edit, create, and delete your spreadsheets in Google Drive



Make sure you trust ESP32_Data_Log_Script

You may be sharing sensitive info with this site or app. Learn about how ESP32_Data_Log_Script will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your [Google Account](#).

[Learn about the risks](#)

[Cancel](#)

[Allow](#)

Step 5 Copy and save the URL and click OK .

Deploy as web app

This project is now deployed as a web app.

Current web app URL:

https://script.google.com/macros/s/AKfycbxx4bP8KZLaj_3Bs

Test web app for your [latest code](#).

OK

Your URL will look something like this:

<https://script.google.com/macros/s/{yourkey}/exec>

The yourkey is a unique key for that specific Google Sheet you created. Therefore do not share this key.

CEA REPORT

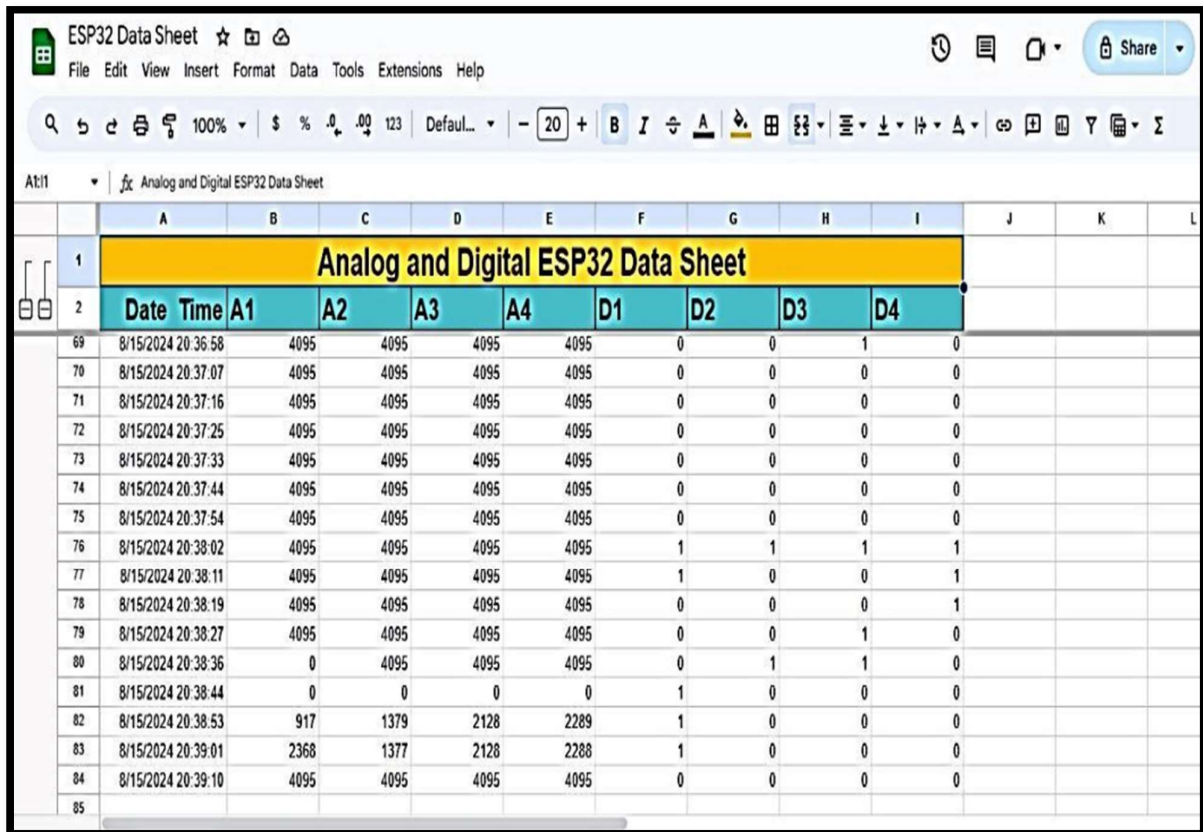
Step 6

To test the server open a browser and enter the URL below.

https://script.google.com/macros/s/{yourkey}/exec?id=Sensor_Data&Sensor1=100&Sensor2=100

If everything is correct, the word 'success' will be shown on your web page.

Now if you go back to your google sheet you will be able to see your data (analog input 1 to 4 & digital input 1 to 4) been logged.



Analog and Digital ESP32 Data Sheet									
Date Time	A1	A2	A3	A4	D1	D2	D3	D4	
8/15/2024 20:36:58	4095	4095	4095	4095	0	0	1	0	
8/15/2024 20:37:07	4095	4095	4095	4095	0	0	0	0	
8/15/2024 20:37:16	4095	4095	4095	4095	0	0	0	0	
8/15/2024 20:37:25	4095	4095	4095	4095	0	0	0	0	
8/15/2024 20:37:33	4095	4095	4095	4095	0	0	0	0	
8/15/2024 20:37:44	4095	4095	4095	4095	0	0	0	0	
8/15/2024 20:37:54	4095	4095	4095	4095	0	0	0	0	
8/15/2024 20:38:02	4095	4095	4095	4095	1	1	1	1	
8/15/2024 20:38:11	4095	4095	4095	4095	1	0	0	1	
8/15/2024 20:38:19	4095	4095	4095	4095	0	0	0	1	
8/15/2024 20:38:27	4095	4095	4095	4095	0	0	1	0	
8/15/2024 20:38:36	0	4095	4095	4095	0	1	1	0	
8/15/2024 20:38:44	0	0	0	0	1	0	0	0	
8/15/2024 20:38:53	917	1379	2128	2289	1	0	0	0	
8/15/2024 20:39:01	2368	1377	2128	2288	1	0	0	0	
8/15/2024 20:39:10	4095	4095	4095	4095	0	0	0	0	

System Design

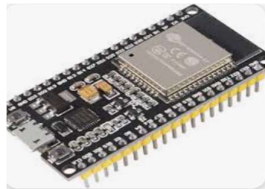
Hardware Components

ESP32 Microcontroller:

- **Pin Configurations:**
 - **Analog Inputs (ADC Pins):**
 - Analog Input 1: GPIO 34 (ADC1 Channel 6)
 - Analog Input 2: GPIO 35 (ADC1 Channel 7)
 - Analog Input 3: GPIO 32 (ADC1 Channel 4)
 - Analog Input 4: GPIO 33 (ADC1 Channel 5)
 - **Digital Inputs:**
 - Digital Input 1: GPIO 12
 - Digital Input 2: GPIO 13
 - Digital Input 3: GPIO 14
 - Digital Input 4: GPIO 27
- **Power Supply:** 5V via USB or external power supply.
- **Wi-Fi Capability:** Used to connect to Google Sheets via the internet.
- **Analog Sensors:** Sensors connected to the analog pins to capture varying voltage levels.
- **Digital Sensors/Switches:** Sensors or switches connected to the digital pins to record HIGH/LOW states.

Components Required:

1. **ESP32 Microcontroller**
 - Provides the analog and digital input capabilities.



2. **4 Potentiometers (for Analog Inputs)**
 - Used to vary the voltage to the analog input pins.



3. 4 Buttons (for Digital Inputs)

- Used to provide HIGH or LOW signals to the digital input pins.



4. Jumper Wires

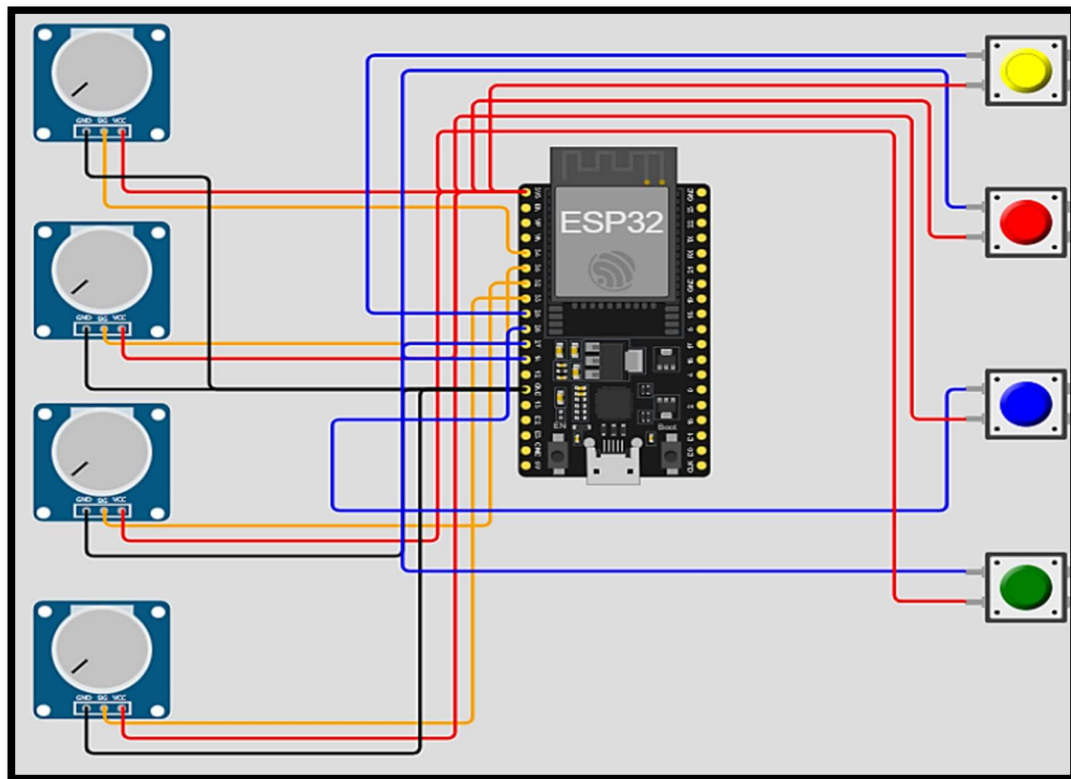
- For making connections between components and the ESP32.

5. Vero Board

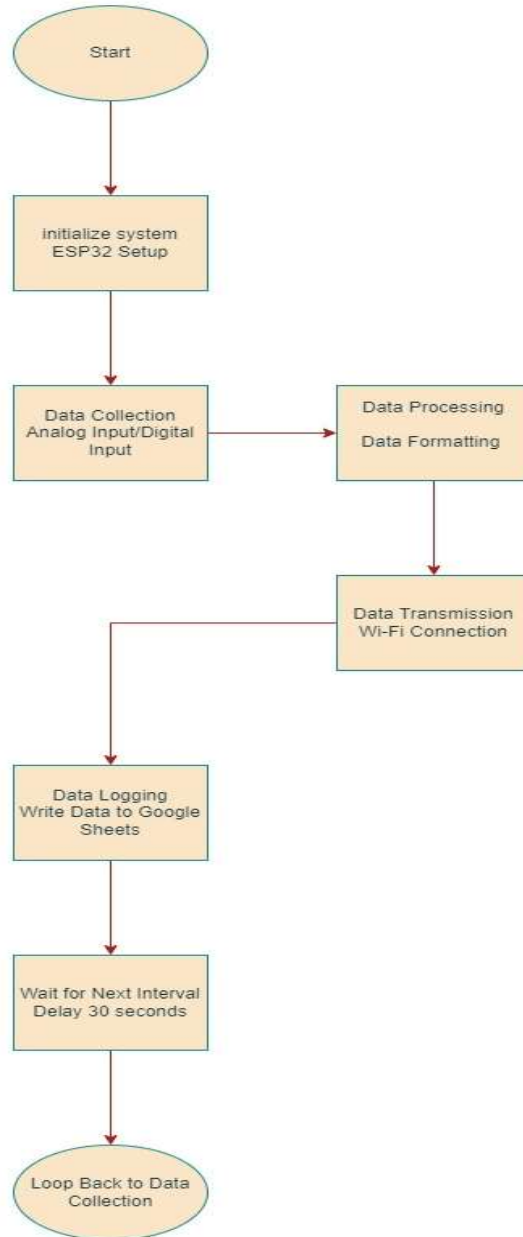
- To organize the components and wiring

CIRCUIT DIAGRAM:

The circuit consists of 4 Analog Sensors (Potentiometer) connected to Pins 32,33,34,35 and 4 Digital Sensors/ Push Buttons connected to Pins 14, 25, 26, 27. Here's the circuit diagram shown below:



Flow Chart:



Arduino IDE Code:

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

```
#include <HTTPClient.h>
```

```
const char* ssid = "YOUR_SSID";
```

```
const char* password = "YOUR_PASSWORD"; // No password required for Wokwi-GUEST
```

CEA REPORT

// Google Apps Script URL

```
const char* serverName =  
"https://script.google.com/macros/s/AKfycbzV89U3buphk_rYhlglNANt77_Gail6lZ2bNF-  
MQWsCFvK5ZkZLOF1Y-yLpRILxueUxg/exec";
```

// Define the ADC (Analog to Digital Converter) pins

```
const int potPin1 = 32; // GPIO 32
```

```
const int potPin2 = 33; // GPIO 33
```

```
const int potPin3 = 34; // GPIO 34
```

```
const int potPin4 = 35; // GPIO 35
```

// Define the digital input pins

```
const int digitalPin1 = 25; // GPIO 25
```

```
const int digitalPin2 = 26; // GPIO 26
```

```
const int digitalPin3 = 27; // GPIO 27
```

```
const int digitalPin4 = 14; // GPIO 14
```

```
void setup() {
```

```
  Serial.begin(9600);
```

```
  Serial.print("Connecting to WiFi");
```

```
  WiFi.begin(ssid, password, 6);
```

```
  while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(100);
```

```
    Serial.print(".");
```

```
  }
```

```
  Serial.println(" Connected!");
```

// Set the potentiometer pins as input

```
pinMode(potPin1, INPUT);
```

```
pinMode(potPin2, INPUT);
```

CEA REPORT

```
pinMode(potPin3, INPUT);

pinMode(potPin4, INPUT);


// Set the digital input pins as input

pinMode(digitalPin1, INPUT);

pinMode(digitalPin2, INPUT);

pinMode(digitalPin3, INPUT);

pinMode(digitalPin4, INPUT);

}

void loop() {

  if (WiFi.status() == WL_CONNECTED) {

    // Read the analog values from each potentiometer

    int potValue1 = analogRead(potPin1);

    int potValue2 = analogRead(potPin2);

    int potValue3 = analogRead(potPin3);

    int potValue4 = analogRead(potPin4);

    // Read the digital values from each digital input

    int digitalInput1 = digitalRead(digitalPin1);

    int digitalInput2 = digitalRead(digitalPin2);

    int digitalInput3 = digitalRead(digitalPin3);

    int digitalInput4 = digitalRead(digitalPin4);

    // Print the values to the Serial Monitor

    Serial.print("Potentiometer 1: ");

    Serial.print(potValue1);

    Serial.print("\tPotentiometer 2: ");

    Serial.print(potValue2);

    Serial.print("\tPotentiometer 3: ");
```

CEA REPORT

```
Serial.print(potValue3);

Serial.print("\tPotentiometer 4: ");

Serial.println(potValue4);


Serial.print("Digital Input 1: ");

Serial.print(digitalInput1);

Serial.print("\tDigital Input 2: ");

Serial.print(digitalInput2);

Serial.print("\tDigital Input 3: ");

Serial.print(digitalInput3);

Serial.print("\tDigital Input 4: ");

Serial.println(digitalInput4);

// Prepare the data as a JSON object

String postData = "{\"potValue1\": " + String(potValue1) +

    ", \"potValue2\": " + String(potValue2) +

    ", \"potValue3\": " + String(potValue3) +

    ", \"potValue4\": " + String(potValue4) +

    ", \"digitalInput1\": " + String(digitalInput1) +

    ", \"digitalInput2\": " + String(digitalInput2) +

    ", \"digitalInput3\": " + String(digitalInput3) +

    ", \"digitalInput4\": " + String(digitalInput4) + "}";


// Print the JSON data to the Serial Monitor

Serial.println("Sending data: " + postData);


// Make the HTTP POST request

HTTPClient http;

http.begin(serverName);
```

CEA REPORT

```
http.addHeader("Content-Type", "application/json");

http.setTimeout(5000); // Set timeout to 15 seconds

int httpResponseCode = http.POST(postData);

// Print response for debugging

if (httpResponseCode > 0) {

    String response = http.getString();

    Serial.print("HTTP Response Code: ");

    Serial.println(httpResponseCode);

    Serial.print("Response: ");

    Serial.println(response);

} else {

    Serial.print("Error on sending POST: ");

    Serial.println(httpResponseCode); }

// Free resources

http.end();

} else {

    Serial.println("WiFi Disconnected");

}

// Add a small delay to avoid overwhelming the server

delay(3000); // Adjust delay as needed

}
```

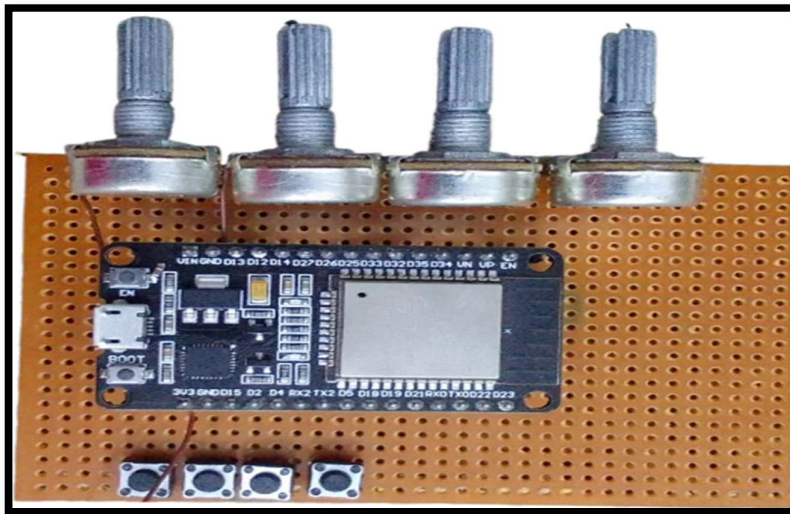

Algorithm for ESP32 Data Logging to Google Sheets:

1. **Initialize Variables:**
 - Define Wi-Fi credentials (`ssid` and `password`).
 - Specify the Google Sheets API URL (`serverName`).
 - Define the GPIO pins for analog and digital inputs (for potentiometers and buttons).
2. **Setup Function:**
 - **Start Serial Communication:** Begin serial communication for debugging.
 - **Connect to Wi-Fi:**
 - Attempt to connect to the specified Wi-Fi network.
 - Continuously check the connection status and print progress to the serial monitor.
 - Once connected, print a confirmation message.
 - **Configure GPIO Pins:**
 - Set the potentiometer pins (analog inputs) as input.
 - Set the button pins (digital inputs) as input.
3. **Main Loop:**
 - **Check Wi-Fi Connection:**
 - If the ESP32 is connected to Wi-Fi, proceed to the next steps.
 - If disconnected, print a message and skip to the next iteration.
 - **Read Input Values:**
 - **Analog Inputs:** Read the values from the potentiometers using `analogRead()`.
 - **Digital Inputs:** Read the states (HIGH/LOW) from the buttons using `digitalRead()`.
 - **Display Input Values:**
 - Print the analog and digital values to the Serial Monitor for debugging.
 - **Prepare Data:**
 - Create a JSON object containing the analog and digital input values.
 - **Send Data to Google Sheets:**
 - Initialize an HTTP client to send the data.
 - Set the request header to indicate that the content is JSON.
 - Make a POST request to the Google Sheets API with the JSON data.
 - **Handle the Response:**
 - If the POST request is successful, print the HTTP response code and server response to the Serial Monitor.
 - If the POST request fails, print an error message.
 - **Free Resources:**
 - End the HTTP connection to free up resources.
 - **Delay:**
 - Add a delay (3 seconds) to avoid overwhelming the server with too many requests.
4. **Repeat Loop:**
 - The loop will continue indefinitely, repeating the data collection and logging process every 3 seconds.

Code For Google Sheet:

```
function doPost(e) {  
  
  try {  
    // Extract the data from the POST request  
    var data = JSON.parse(e.postData.contents);  
  
    // Open the active Google Sheet  
    var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();  
  
    // Append a new row with the received data  
    sheet.appendRow([  
      new Date(),  
      data.potValue1, data.potValue2, data.potValue3, data.potValue4,  
      data.digitalInput1, data.digitalInput2, data.digitalInput3, data.digitalInput4  
    ]);  
  
    // Return a success message  
    return ContentService.createTextOutput("Data received successfully!");  
  } catch (error) {  
    // Return an error message if something goes wrong  
    return ContentService.createTextOutput("Error: " + error.message);  
  }  
}
```

HARDWARE:



CONCLUSION:

The Smart Data Logging System project stands as a testament to our comprehensive understanding and practical application of embedded systems design. From the initial concept to the final implementation, we meticulously designed and integrated both hardware and software components to meet the project's objectives. The system reliably captures data from four analog and four digital inputs, logging each input's state and value with precision every 30 seconds. This data is automatically uploaded to Google Sheets, enabling seamless and continuous data monitoring and analysis in real-time.

Our approach involved a careful selection of components, including the ESP32 microcontroller, which provided the necessary computational power and Wi-Fi connectivity for real-time data transmission. The system's software architecture was developed with a focus on efficiency and reliability, incorporating a timer function to ensure consistent data logging intervals and leveraging the Google Sheets API for secure and accurate data uploads. The project required not only technical expertise in coding and circuit design but also a deep understanding of cloud integration and data management.

Throughout the project, we conducted extensive testing to ensure the system's performance met the required standards. We tackled challenges related to data synchronization, network stability, and API integration, resolving each issue to achieve a fully functional and reliable data logging system. The comprehensive documentation, including the system design, firmware code, and testing results, further underscores our commitment to delivering a robust and scalable solution.

In conclusion, the successful completion of this project showcases our ability to design, implement, and document a complex embedded system from start to finish. The Smart Data Logging System not only fulfills its intended purpose but also serves as a valuable learning experience, enhancing our skills in embedded systems, cloud integration, and real-time data management. This project positions us to tackle even more challenging and sophisticated systems in the future, with a strong foundation in both theoretical knowledge and practical application.

