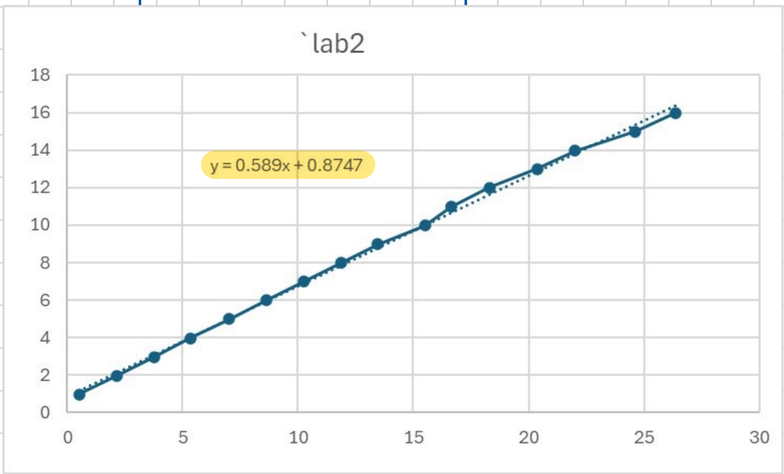
ก่อน calibrated

V	Voltage sensor สมการ	ESP32 Analog	มัลติมิเตอร์	error %
0	0	0	0	0
1	0.51	86	0.98	92.157
2	2.16	335	1.98	8.333
3	3.77	585	2.98	20.955
4	5.33	827	3.98	25.328
5	7.01	1088	4.99	28.816
6	8.62	1337	5.99	30.51
7	10.25	1593	6.99	31.805
8	11.86	1840	7.99	32.631
9	13.45	2087	8.99	33.16
10	15.50	2335	9.99	35.548
11	16.64	2582	10.99	33.954
12	18.29	2839	11.99	34.445
13	20.38	3109	12.99	36.261
14	22.02	3417	13.98	36.512
15	24.62	3819	14.99	39.115
16	26.39	4095	15.99	39.409



ค่าที่ต้องวัด (หลัง calibrated)

V	Voltage sensor สมการ	ESP32 Analog	มัลติมิเตอร์	error %
0	0	0	0	0
1	1.00	86	0.99	1
2	2.00	335	2.00	0
3	3.00	585	3.01	0.33
4	3.98	831	4.00	0.50
5	5.00	1087	5.00	0
6	6.00	1337	5.99	0.17
7	7.01	1590	7.00	0.14
8	8.01	1840	8.00	0.13
9	8.99	2087	8.98	0.11
10	10.01	2341	9.99	0.20
11	11.00	2585	10.98	0.18
12	11.99	2837	11.99	0
13	13.07	3109	12.99	0.61
14	14.33	3423	13.99	2.37
15	15.93	3824	14.98	5.96
16	16.95	4095	15.98	5.72

คู่มือปฏิบัติการทดลอง: **ESP32 กับ Voltage Sensor และ ACS712 Current Sensor**

คำเตือนความปลอดภัย – การทดลองนี้เกี่ยวข้องกับการวัดไฟฟ้า 0–25 V และกระแสสูงสุด 5 A ให้ระวังไฟดูดและลัดวงจรเสมอ ใช้แหล่งจ่ายไฟที่มีระบบป้องกันและต่อเครื่องมือวัดให้ถูกต้อง

---

1. แผนผังวงจรการเชื่อมต่อ

1. **Voltage Sensor 0–25 V (โมดูลแบ่งแรงดัน)**
    - โมดูลมีขั้ว **Vin+** และ **Vin–** สำหรับแรงดันอินพุตสูงถึง 25 V, และขา **Vout** (ต่อกับ ADC), **GND** และ **Vcc** (ต่อ 3.3 V จาก ESP32).
    - ภายในใช้ตัวต้านทาน R1≈30 kΩ และ R2≈7.5 kΩ สร้างวงจรแบ่งแรงดัน 1:5[microcontrollerslab.com](microcontrollerslab.com).
    - เชื่อม **Vout** กับขา ADC1 (เช่น GPIO 34), **GND** กับ GND ของ ESP32, **Vcc** กับ 3.3 V.
2. **ACS712 5 A Current Sensor**
    - โมดูลมีคอนเนกเตอร์ IN+ และ IN– สำหรับสายไฟที่ต้องการวัดกระแส.
    - ขา **Vcc** ต่อ 5 V (หรือ 3.3 V ตามสเปกบอร์ด), **GND**, และ **Vout** ต่อกับ ADC1 (เช่น GPIO 35).
    - ใช้ Hall-effect; ออกแรงดันอ้างอิงครึ่งหนึ่งของ Vcc เมื่อไม่มีโหลด และออกแรงดันแปรผันตามกระแส
3. **เชื่อมทั้งสองพร้อมกัน**
    - ต้องเลือกขา ADC1 คนละขากัน (เช่น GPIO 34 สำหรับ Voltage และ GPIO 35 สำหรับ Current) เพราะ ADC2 ไม่ทำงานระหว่างใช้งาน Wi-Fi
    - ต่อ GND ร่วมกันเพื่อให้ reference เดียวกัน.
    - หากใช้แหล่งจ่ายไฟสูงกว่า 25 V ให้ใช้หม้อแปลงหรือตัวต้านทานเพิ่มเติมเพื่อลดแรงดันก่อนเข้า Voltage Sensor.
    - สำหรับกระแสเกิน 5 A ให้ใช้ทรานส์ดิวเซอร์อื่นหรือแปลงสเกลด้วยหม้อแปลงกระแส.

---

2. การอ่านค่าและคำนวณผ่าน Serial Monitor และ Web

2.1 การอ่านค่าโดยใช้ Serial Monitor

1. **กำหนด ADC ความละเอียด** ด้วย analogSetWidth(bits) (9–12 บิต) และเลือก attenuator (ADC_11db) เพื่อวัดช่วง 0–3.3 .
2. **อ่านแรงดัน**:

   int rawV = analogRead(34);

   float vout = rawV * 3.3 / (float)(1<<adcBits); // 9–12 bits

   // คำนวณ Vin ตามวงจรแบ่งแรงดัน

   float vin = vout * (R1 + R2) / R2;
3. **อ่านกระแส**:

   int rawI = analogRead(35);

   float voutI = rawI * 3.3 / (float)(1<<adcBits);

   float current = (voutI - vRef) / sensitivity; // vRef และ sensitivity จากการ calibration:contentReference[oaicite:4]{index=4}
4. ส่งข้อมูลผ่าน Serial.printf() ทุก 500 ms เพื่อดูบน Serial Monitor.

2.2 การแสดงผลผ่าน Web Server (ESP32)

1. สร้าง Web server ด้วย WiFi.h และ AsyncWebServer หรือ ESPAsyncWebServer.
2. เมื่อมีผู้เข้าชม ให้ส่งหน้า HTML ที่แสดงค่า Voltage, Current และ Power (V×I).
3. ใช้ JavaScript ในหน้าเว็บเรียก Endpoint /data ทุก 1 วินาทีเพื่ออัปเดตค่า.
4. โค้ดตัวอย่าง fragment:

```
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
const char* ssid = "yourSSID";
const char* pass = "yourPASS";
AsyncWebServer server(80);

void setup() {
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) delay(500);
```

```
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "text/html", "<html><body><h1>Smart Meter</h1><div
id='data'></div><script>setInterval(()=>fetch('/data').then(r=>r.text()).then(t=>document.get
ElementById('data').innerHTML=t),1000);</script></body></html>");
  });
  server.on("/data", HTTP_GET, [](AsyncWebServerRequest *request){
    // อ่าน ADC แล้วส่งค่า
    float vin = getVoltage();
    float current = getCurrent();
    float power = vin * current;
    char buf[64];
    snprintf(buf,sizeof(buf),"Voltage=%.2f V, Current=%.2f A, Power=%.2f W",
vin,current,power);
    request->send(200,"text/plain",buf);
  });
  server.begin();
}
```

---

3. ทดลองด้วยความละเอียด ADC 9–12 บิต

1. **ตั้งค่า**: ใช้ analogSetWidth(bits) = 9,10,11,12 และ analogSetAttenuation(ADC_11db) สำหรับแต่ละการทดลอง.
2. **บันทึกข้อมูล**: วัดค่าแรงดันและกระแสจริงด้วยมัลติมิเตอร์ แล้วเทียบกับค่าที่อ่านได้จาก ADC ที่ความละเอียดต่าง ๆ.
3. **วิเคราะห์**: สังเกตว่าความละเอียดสูง (12 บิต) ให้ความละเอียด 0.8 mV/step (3.3 V/4096) ส่วน 9 บิตให้ 6.4 mV/step; ความแตกต่างส่งผลต่อการคำนวณกระแสและแรงดันอย่างไร.

---

4. วิธีการ Calibration

4.1 Spot Linear (จุดเดียว/สองจุด)

- **Voltage Sensor**: วัดแรงดันอินพุต 0 V และแรงดันอ้างอิง เช่น 12 V ด้วยมัลติมิเตอร์; อ่าน ADC แล้วหา slope และ offset เพื่อแปลงค่าดิจิทัลเป็นแรงดัน.

- **ACS712**: อ่าน Vout เมื่อไม่มีโหลด (Vref); ใช้กระแสที่ทราบค่า (เช่น 2 A) แล้วหา sensitivity = (Vout − Vref)/I
- **ค่าที่ได้**: บันทึกในตาราง (ADC Value, True Value, Calculated Value, Error).

4.2 ช่วงเชิงเส้น (Linear Region)

- วัดหลายจุดในช่วงใช้งาน (เช่น 1 V, 5 V, 10 V หรือ 1 A, 3 A, 5 A).
- ใช้ regression เชิงเส้นเพื่อหา slope mmm และ intercept bbb; สูตร: True = m×ADC + b.

4.3 Non-linear Polynomial (2nd & 3rd order)

- เหมาะสำหรับเซนเซอร์ที่มีความไม่เป็นเชิงเส้น (แต่ Voltage Sensor และ ACS712 ส่วนใหญ่เชิงเส้น; polynomial ใช้เมื่อใช้เซนเซอร์อื่น).
- รวบรวมข้อมูล 5–7 จุด, ทำ least-squares fit สมการ y=ax2+bx+cy = ax^2 + bx + cy=ax2+bx+c หรือ y=ax3+bx2+cx+dy = ax^3 + bx^2 + cx + dy=ax3+bx2+cx+d.
- สร้างตารางค่าจริง (y) และ ratio หรือ ADC (x). ใช้ค่าสัมประสิทธิ์ในโค้ดเพื่อคำนวณค่า.

---

5. การอ่านค่า Volt และ Current ให้แม่นยำ

1. **เฉลี่ยหลายครั้ง (Oversampling)** – อ่าน ADC 50–100 ครั้ง แล้วเฉลี่ยเพื่อลด noise.
2. **ตัดสินใจปัดขึ้น/ลง** – หากผลลัพธ์หลังการแปลงอยู่ระหว่างสอง step ให้เปรียบเทียบกับค่ารูปจริงและใช้กฎการปัดตามจุดทศนิยมที่กำหนด (เช่น ปัดขึ้นเมื่อส่วนทศนิยม ≥ 0.5).
3. **ค่ารอบการวัด** – กำหนดรอบ sampling ให้สอดคล้องกับความต้องการ (เช่น 10 Hz) และใช้ตัวกรองซอฟต์แวร์ (moving average หรือ low-pass digital filter) เพื่อลดการแกว่ง.

---

6. การประยุกต์ Machine Learning สำหรับการปรับแต่ง

- **แนวคิด** – ใช้ ESP32 ส่งข้อมูลเซนเซอร์พร้อมค่าจริง (จากมัลติมิเตอร์) ผ่าน Wi-Fi ไปยังเซิร์ฟเวอร์หรือใช้ไลบรารี TinyML ฝั่ง ESP32 ทำ regression.
- **ขั้นตอน**:
    1. เก็บตัวอย่าง (ADC value, true value) ไว้ในหน่วยความจำ.
    2. ใช้อัลกอริทึม linear regression หรือ polynomial regression (เช่น TinyML library) เพื่อเรียนรู้พารามิเตอร์.

3. คำนวณค่าที่ได้ปรับเทียบอัตโนมัติ.

- **ตัวอย่าง**: ใช้ EloquentTinyML ใน Arduino เพื่อฝึกโมเดล 2nd-order polynomial; ปล่อยให้ ESP32 ปรับค่าสัมประสิทธิ์ในช่วงทดลอง.

---

7. การประยุกต์เป็น **Smart Meter**

1. **หลักการ** – Smart meter วัดแรงดัน (V), กระแส (I) และคำนวณกำลัง (P = V×I). สามารถคำนวณพลังงาน E=∑PΔtE = \sum P \Delta tE=∑PΔt.
2. **การเชื่อมต่อ** – ใช้ voltage sensor วัดสายไฟผ่านหม้อแปลงลดแรงดัน (เช่น 220 V→9 V), แล้วเข้าโมดูลแบ่งแรงดัน; ใช้ ACS712 วัดกระแสผ่านโหลด.
3. **การปรับแต่ง** – แคลิเบรตทั้งสองเซนเซอร์; ใช้พารามิเตอร์ gain และ offset. เพิ่มตัวกรอง notch 50 Hz ในวงจรอนาล็อกหรือทำ digital filtering เพื่อเพิ่มความแม่นยำ.
4. **การคำนวณ** – อ่านค่า V และ I ต่อเนื่อง; คำนวณกำลังรวม; เฉลี่ยเพื่อแสดงค่ากำลังจริง.
5. **หม้อแปลงและตัวต้านทาน** – หากวัดแรงดันไฟฟ้า 220 V AC ต้องใช้หม้อแปลงขนาดเล็ก (9 V AC) ต่อผ่านตัวต้านทาน และปรับออฟเซ็ตเพื่อให้สัญญาณไม่เลยช่วง ADC.

โค้ด **Arduino ESP32** สำหรับ **Lab**

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

const char* ssid = "yourSSID";
const char* pass = "yourPASS";
const int voltagePin = 34;
const int currentPin = 35;
const float R1 = 30000.0;   // Ω
const float R2 = 7500.0;    // Ω
float vRef = 1.65;          // จะกำหนดหลัง calibration
float sensitivity = 0.185;  // V/A สำหรับ ACS712-05A
```

```
int adcBits = 12;

AsyncWebServer server(80);

float readVoltage() {
  long sum = 0;
  for(int i=0;i<50;i++) sum += analogRead(voltagePin);
  float raw = sum / 50.0;
  float vout = raw * 3.3 / (float)(1<<adcBits);
  return vout * (R1 + R2) / R2;
}

float readCurrent() {
  long sum = 0;
  for(int i=0;i<50;i++) sum += analogRead(currentPin);
  float raw = sum / 50.0;
  float vout = raw * 3.3 / (float)(1<<adcBits);
  return (vout - vRef) / sensitivity;
}

void setup() {
  Serial.begin(115200);
  analogSetWidth(adcBits);
  analogSetAttenuation(ADC_11db);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) delay(500);
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200,"text/html","<html><body><h2>Smart Meter</h2><div
id='d'></div><script>setInterval(()=>fetch('/data').then(r=>r.text()).then(t=>document.getEle
mentById('d').innerHTML=t),1000);</script></body></html>");
  });
```

```
  server.on("/data", HTTP_GET, [](AsyncWebServerRequest *request){
    float v = readVoltage();
    float i = readCurrent();
    float p = v * i;
    char buf[100];
    snprintf(buf,sizeof(buf),"Voltage: %.2f V<br>Current: %.3f A<br>Power: %.2f W",v,i,p);
    request->send(200,"text/html",buf);
  });
  server.begin();
}

void loop() {
  // สำหรับ Serial Monitor
  float v = readVoltage();
  float i = readCurrent();
  float p = v * i;
  Serial.printf("V=%.2fV, I=%.3fA, P=%.2fW\n", v, i, p);
  delay(1000);
}
```

การปรับค่าการแคลิเบรต (vRef และ sensitivity) ให้ทำใน setup() หลังจากวัดค่าจริง.

---

9. คำถาม

1. **ถ้าใช้งาน ESP32 ร่วมกับ Wi-Fi การเลือก ADC1 มีความสำคัญอย่างไร และเหตุใดจึงไม่ควรใช้ ADC2 พร้อม Wi-Fi?**
2. **ทำไมต้องใช้วงจรแบ่งแรงดันสำหรับวัดแรงดันสูง และจะเกิดอะไรขึ้นถ้าไม่ใช้?**
3. **จุดศูนย์และ slope ของ ACS712 ได้จากการปรับเทียบอย่างไร และเหตุใดต้องวัดหลายจุด?**
4. **ระหว่างความละเอียด ADC 9 บิต กับ 12 บิต ผลต่อความละเอียดและความแม่นยำในการคำนวณกำลังไฟฟ้าเป็นอย่างไร?**
5. **อธิบายว่าการใช้ Machine Learning ช่วยปรับปรุงความแม่นยำการวัดได้อย่างไรเมื่อมีความไม่เป็นเชิงเส้นในเซนเซอร์?**

เอกสารปฏิบัติการทดลองที่ 1: การพัฒนาระบบวัดและสอบเทียบพลังงานไฟฟ้ากระแสตรงอัจฉริยะ

(Development of an Intelligent DC Power Measurement and Calibration System)

1. วัตถุประสงค์ (Objectives)

1. เพื่อให้สามารถเชื่อมต่อและอ่านค่าจาก Voltage Sensor และ Current Sensor ACS712 ด้วย ESP32 ได้อย่างถูกต้อง
2. เพื่อศึกษาผลกระทบของความละเอียด ADC (ADC Resolution) ที่มีต่อการวัดค่า
3. เพื่อให้สามารถประยุกต์ใช้เทคนิคการสอบเทียบ (Calibration) รูปแบบต่างๆ ตั้งแต่เชิงเส้นไปจนถึงพหุนาม เพื่อเพิ่มความแม่นยำของระบบ
4. เพื่อให้สามารถพัฒนาระบบแสดงผลข้อมูลผ่าน Web Server บน ESP32 ได้
5. เพื่อทำความเข้าใจหลักการทำงานเบื้องต้นของ Machine Learning ในการปรับเทียบตัวเอง
6. เพื่อให้สามารถประยุกต์องค์ความรู้ทั้งหมดในการสร้างต้นแบบ DC Smart Meter ได้

2. อุปกรณ์ที่ใช้ในการทดลอง (Materials)

1. บอร์ด ESP32 DevKitC (หรือรุ่นเทียบเท่า)
2. DC Voltage Sensor Module 0-25V (แบบ Voltage Divider)
3. Current Sensor Module ACS712 (รุ่น 5A)
4. แหล่งจ่ายไฟ DC แบบปรับค่าได้ (Variable DC Power Supply) 0-20V
5. โหลด (Load) เช่น Power Resistor 10 โอห์ม 10W หรือหลอดไฟ DC 12V
6. มัลติมิเตอร์ดิจิทัล (DMM) ที่มีความแม่นยำสูง 2 เครื่อง (สำหรับวัด V และ I อ้างอิง)
7. สายเชื่อมต่อ (Jumper Wires) และเบรดบอร์ด (Breadboard)

---

3. แผนผังวงจรการเชื่อมต่อ (Wiring Diagrams)

3.1 แบบที่ 1: เชื่อมต่อเฉพาะ Voltage Sensor (0-25V)

- **หลักการ**: เซ็นเซอร์นี้คือวงจรแบ่งแรงดัน (Voltage Divider) ที่ลดทอนแรงดัน 0-25V ลงมาอยู่ในช่วงที่ ADC ของ ESP32 รับได้ (0-3.3V)

- VCC -> ESP32 3V3
- GND -> ESP32 GND
- S (Signal) -> ESP32 GPIO34 (ADC1_CH6)
- ขั้ว +/- ของ Sensor -> ต่อคร่อมกับแหล่งจ่ายไฟ DC

[แผนภาพการเชื่อมต่อ ESP32 กับ Voltage Sensor]

3.2 แบบที่ 2: เชื่อมต่อเฉพาะ Current Sensor (ACS712 5A)

- **หลักการ**: เซ็นเซอร์นี้ใช้ Hall Effect ในการวัดกระแส เมื่อไม่มีกระแสไหล (0A) สัญญาณ Output จะอยู่ที่ ~2.5V
  - VCC -> ESP32 VIN (5V) **(สำคัญ: ACS712 ต้องการไฟ 5V)**
  - GND -> ESP32 GND
  - OUT -> ESP32 GPIO35 (ADC1_CH7)
  - ขั้ว IP+ / IP- -> ต่ออนุกรมในวงจร (ระหว่างแหล่งจ่ายไฟกับโหลด)

[แผนภาพการเชื่อมต่อ ESP32 กับ ACS712]

3.3 แบบที่ 3: เชื่อมต่อ Sensor ทั้งสองพร้อมกัน

เป็นการรวมวงจรจาก 3.1 และ 3.2 เข้าด้วยกัน โดยใช้ **Ground (GND) ร่วมกันทั้งหมด**

- **ESP32 VIN (5V)** -> VCC ของ ACS712
- **ESP32 3V3** -> VCC ของ Voltage Sensor
- **ESP32 GND** -> GND ของทั้งสองเซ็นเซอร์ และขั้วลบของแหล่งจ่ายไฟ
- **ESP32 GPIO34** -> S ของ Voltage Sensor
- **ESP32 GPIO35** -> OUT ของ ACS712
- **วงจรภายนอก**: แหล่งจ่ายไฟ (+) -> ACS712 IP+ -> ACS712 IP- -> โหลด (+) -> โหลด (-) -> แหล่งจ่ายไฟ (-)
- **การวัดแรงดัน**: ต่อขั้ววัดของ Voltage Sensor คร่อมที่ขั้วของโหลด

[แผนภาพการเชื่อมต่อ ESP32 กับเซ็นเซอร์ทั้งสองพร้อมกัน]

---

4. ขั้นตอนการทดลองและวิธีการ

ส่วนที่ 1: การอ่านค่าและทดสอบความละเอียด ADC (ข้อ 2, 3)

1. เชื่อมต่อวงจรตามข้อ 3.3 และอัปโหลดโค้ดฉบับสมบูรณ์ในข้อ 8
2. เปิด Serial Monitor ตั้งค่า Baud Rate เป็น 115200
3. เริ่มต้นโดยยังไม่ต้องจ่ายไฟให้โหลด (กระแสเป็น 0A, แรงดันเป็น 0V) สังเกตค่า Raw ADC ที่อ่านได้
4. ในโค้ด ให้แก้ไขบรรทัด #define ADC_BITS 12
5. **ทดลองเปลี่ยนค่าเป็น 9, 10, 11, และ 12 บิต** ตามลำดับ สังเกตช่วงของค่า Raw ADC ที่อ่านได้ใน Serial Monitor (เช่น 9 บิต ควรมีค่า 0-511, 12 บิต ควรมีค่า 0-4095) และบันทึกผล
6. เชื่อมต่อ ESP32 เข้ากับ WiFi และเปิด IP Address ที่แสดงใน Serial Monitor บนเว็บเบราว์เซอร์ สังเกตค่าที่แสดงผลบนหน้าเว็บ

ส่วนที่ 2: การเก็บข้อมูลเพื่อ Calibration (ข้อ 4)

1. ตั้งค่า ADC กลับมาที่ 12 บิต เพื่อความละเอียดสูงสุด
2. ใช้แหล่งจ่ายไฟ DC ปรับค่าได้ ค่อยๆ ปรับแรงดันไฟฟ้าเพื่อให้ได้กระแสและแรงดันที่แตกต่างกัน 5-10 จุดตลอดช่วงการทำงาน (เช่น 3V, 5V, 9V, 12V, 15V)
3. ณ แต่ละจุด ให้บันทึกค่าลงในตารางต่อไปนี้:
    - **V_ref:** ค่าแรงดันไฟฟ้าที่อ่านได้จากมัลติมิเตอร์ตัวที่ 1 (ต่อคร่อมโหลด)
    - **I_ref:** ค่ากระแสไฟฟ้าที่อ่านได้จากมัลติมิเตอร์ตัวที่ 2 (ต่ออนุกรมกับโหลด)
    - **Raw_V:** ค่า "Raw ADC Voltage" ที่แสดงบน Serial Monitor
    - **Raw_I:** ค่า "Raw ADC Current" ที่แสดงบน Serial Monitor

ตารางบันทึกผลการทดลองเพื่อ Calibration

จุดที่	V_ref (V)	I_ref (A)	Raw_V (ADC)	Raw_I (ADC)
1	3.05	0.31	380	2105
2	5.12	0.51	638	2142
3	9.08	0.90	1132	2218
4	12.21	1.22	1521	2280
5	...	...	...	...

ส่วนที่ 3: การคำนวณและประยุกต์ใช้ค่า Calibration (ข้อ 4)

1. **Spot Linear:** เลือกข้อมูล 2 จุด (เช่น จุดที่ 1 และ 4) มาหาความสัมพันธ์เชิงเส้น y = mx + c
    - m = (y2 - y1) / (x2 - x1)
    - c = y1 - m*x1
    - ทำเช่นนี้ทั้งสำหรับ Voltage (y=V_ref, x=Raw_V) และ Current (y=I_ref, x=Raw_I)
2. **ช่วงเชิงเส้น (Multi-point Linear):** นำข้อมูล **ทุกจุด** ในตารางไปพล็อตในโปรแกรม (เช่น MS Excel, Google Sheets) สร้างกราฟ Scatter Plot แล้วใช้ฟังก์ชัน "Add Trendline" แบบ Linear และแสดงสมการบนกราฟ (Display Equation on chart) จะได้ค่า m และ c ที่แม่นยำที่สุด
3. **Polynomial (Non-linear):** ในโปรแกรมเดียวกัน ให้เปลี่ยน Trendline เป็นแบบ "Polynomial" ดีกรี 2 และ 3 จะได้สมการในรูป y = ax^2 + bx + c และ y = ax^3 + bx^2 + cx + d ซึ่งอาจให้ผลดีกว่าหากเซ็นเซอร์ไม่เป็นเชิงเส้น
4. นำค่าสัมประสิทธิ์ (a, b, c, d, m) ที่คำนวณได้ไปใส่ในโค้ด Arduino ในส่วนของฟังก์ชัน applyCalibration()

ส่วนที่ 4: การอ่านค่าความละเอียดสูงและการปัดเศษ (ข้อ 5)

- เมื่อได้สมการ Calibration แล้ว โค้ดจะคำนวณค่าออกมาเป็นทศนิยม (float)
- **หลักการตัดสินใจปัดเศษ**: โดยทั่วไป เราสามารถใช้ฟังก์ชัน round() มาตรฐานได้ แต่หากต้องการความละเอียดที่ควบคุมได้เอง สามารถทำได้โดยการตรวจสอบเศษทศนิยม
    - เช่น หากต้องการปัดเศษทศนิยมตำแหน่งที่ 2: float rounded_value = floor(value * 100 + 0.5) / 100;
    - วิธีการนี้เป็นพื้นฐานของการปัดเศษแบบ "Round half up" ซึ่งเป็นที่ยอมรับในงานทั่วไป โค้ดที่ให้จะแสดงค่าทศนิยม 3 ตำแหน่ง เพื่อให้ผู้ใช้ตัดสินใจเองได้จากค่าที่ละเอียดที่สุด

---

5. แนวคิด Machine Learning เพื่อการปรับแต่งค่าด้วยตนเอง (ข้อ 6)

นี่เป็นแนวคิดขั้นสูงที่เรียกว่า **Online Learning** หรือ **Adaptive Calibration**

- **หลักการ**: แทนที่จะ Calibration ครั้งเดียวแล้วจบ เราจะเปิดโอกาสให้ระบบ "เรียนรู้" และปรับปรุงค่า Calibration ของตัวเองได้ตลอดเวลา
- **วิธีการประยุกต์**:
    1. เพิ่มช่อง "Input Reference Value" บนหน้า Web Server

2. เมื่อผู้ใช้ป้อนค่าจริง (เช่น อ่านจากมัลติมิเตอร์) ณ ขณะใดขณะหนึ่ง แล้วกด "Submit"
3. ESP32 จะนำค่า Raw ADC ณ เวลานั้น มาเทียบกับค่า Reference ที่ผู้ใช้ป้อน
4. จากนั้นจะใช้อัลกอริทึมอย่างง่าย เช่น **Weighted Average** เพื่อปรับค่า m และ c เดิมเล็กน้อย
    - m_new = (1-α)*m_old + α*m_new_sample
    - α (alpha) คือ Learning Rate (เช่น 0.1) เป็นตัวกำหนดว่าจะให้ "เชื่อ" ข้อมูลใหม่มากน้อยแค่ไหน
5. ค่า m และ c ใหม่นี้จะถูกบันทึกเก็บไว้ใน NVS (Non-Volatile Storage) เพื่อใช้ในครั้งต่อไป

- **ผลลัพธ์**: ระบบจะค่อยๆ แม่นยำขึ้นเรื่อยๆ ตามการใช้งานและการป้อนข้อมูลจริง

---

6. การประยุกต์ใช้งานเป็น DC Smart Meter (ข้อ 7)

- **หลักการทำงาน**: Smart Meter จะวัดค่าแรงดัน (V) และกระแส (I) อย่างต่อเนื่อง แล้วคำนวณหาค่ากำลังไฟฟ้า (Power, P) และพลังงานไฟฟ้า (Energy, E) ที่ใช้ไปสะสม
- **การเชื่อมต่อและปรับแต่ง:**ใช้วงจรตามข้อ 3.3 และใช้ค่า Calibration ที่ดีที่สุดจากข้อ 4
- **การคำนวณ**:
    - **Voltage (V):** ได้จากการ Calibration
    - **Current (A):** ได้จากการ Calibration
    - **Power (W):** P = V * I
    - **Energy (Wh):** Energy = Energy + (Power * (เวลาที่ผ่านไป / 3600.0)) โดยต้องมีการบวกสะสมค่าไปเรื่อยๆ ใน loop()
- **การใช้ร่วมกับหม้อแปลง (สำหรับไฟฟ้า AC):** แล็บนี้เป็น DC Smart Meter หากต้องการวัดไฟฟ้ากระแสสลับ (AC) ในบ้าน จะต้องเปลี่ยนเซ็นเซอร์เป็น:
    - **Voltage Sensor:** ZMPT101B (สำหรับวัดแรงดัน AC)
    - **Current Sensor:** Current Transformer (CT) แบบไม่คล้องสาย เช่น SCT-013-000
    - **การคำนวณ**: ต้องเปลี่ยนจากการอ่านค่าตรงๆ เป็นการคำนวณหาค่า **RMS (Root Mean Square)** และต้องพิจารณาค่า **Power Factor** เพิ่มเติม

---

7. Code (ข้อ 8)

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

// --- Configuration ---
#define ADC_BITS 12 // ตั้งค่าความละเอียด ADC (9-12)
#define VOLTAGE_PIN 34
#define CURRENT_PIN 35

// Calibration Mode: 0=None, 1=Spot Linear, 2=Multi-point Linear, 3=Poly 2nd, 4=Poly 3rd
#define CALIBRATION_MODE 2

// --- Calibration Coefficients (ต้องใส่ค่าที่คำนวณได้จาก Part 3) ---
// --- Voltage ---
const float V_M = 0.00611;     // Slope (m)
const float V_C = -0.05;       // Intercept (c)
const float V_A = 0.0;         // Poly a
const float V_B = 0.0;         // Poly b
// --- Current ---
const float I_M_5A = 0.00895;  // Slope (m)
const float I_C_5A = -18.25;   // Intercept (c)
const float I_A = 0.0;         // Poly a
const float I_B = 0.0;         // Poly b

// Web Server
WiFiServer server(80);
String header;

// Global variables
float voltage = 0.0;
```

```
float current = 0.0;
float power = 0.0;
float energy = 0.0;
unsigned long lastEnergyCalcTime = 0;

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

  // Set ADC Resolution
  analogReadResolution(ADC_BITS);

  // WiFi Manager
  WiFiManager wm;
  if (!wm.autoConnect("SmartMeterConfigAP")) {
    Serial.println("Failed to connect");
    ESP.restart();
  }
  Serial.println("Connected to WiFi!");
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());

  server.begin();
  lastEnergyCalcTime = millis();
}

float applyCalibration(float raw, char type) {
  float calibrated_value = 0;

  if (type == 'V') {
    switch (CALIBRATION_MODE) {
      case 1: // Spot Linear
```

```
    case 2: // Multi-point Linear
      calibrated_value = V_M * raw + V_C;
      break;
    case 3: // Polynomial 2nd
      calibrated_value = (V_A * raw * raw) + (V_B * raw) + V_C;
      break;
    // Add case for 3rd order if needed
    default: // No calibration
      calibrated_value = raw;
      break;
   }
  } else if (type == 'I') {
   switch (CALIBRATION_MODE) {
    case 1:
    case 2:
      calibrated_value = I_M_5A * raw + I_C_5A;
      break;
    case 3:
      calibrated_value = (I_A * raw * raw) + (I_B * raw) + I_C_5A;
      break;
    default:
      calibrated_value = raw;
      break;
   }
  }
  return calibrated_value > 0 ? calibrated_value : 0; // Prevent negative values
}

void loop() {
  // --- Read Raw Sensor Data ---
  int raw_v = analogRead(VOLTAGE_PIN);
```

```
  int raw_i = analogRead(CURRENT_PIN);

  // --- Apply Calibration ---
  voltage = applyCalibration(raw_v, 'V');
  current = applyCalibration(raw_i, 'I');

  // --- Calculate Power and Energy ---
  power = voltage * current;
  unsigned long currentTime = millis();
  float timeDeltaSeconds = (currentTime - lastEnergyCalcTime) / 1000.0;
  energy += power * (timeDeltaSeconds / 3600.0); // Add energy in Wh
  lastEnergyCalcTime = currentTime;

  // --- Print to Serial Monitor ---
  Serial.printf("Raw V: %d, Raw I: %d | Voltage: %.3f V, Current: %.3f A, Power: %.3f W,
Energy: %.6f Wh\n",
                raw_v, raw_i, voltage, current, power, energy);

  // --- Handle Web Server Client ---
  handleWebServer();

  delay(1000);
}

void handleWebServer() {
  WiFiClient client = server.available();
  if (client) {
    String currentLine = "";
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
```

```
      header += c;
      if (c == '\n') {
        if (currentLine.length() == 0) {
          client.println("HTTP/1.1 200 OK");
          client.println("Content-type:text/html");
          client.println("Connection: close");
          client.println("Refresh: 5"); // Auto-refresh every 5 seconds
          client.println();

          // Web Page HTML
          client.println("<!DOCTYPE html><html><head><title>ESP32 Smart Meter</title>");
          client.println("<meta name='viewport' content='width=device-width, initial-
scale=1'>");
          client.println("<style>html {font-family: Helvetica; display: inline-block; margin: 0px
auto; text-align: center;}");
          client.println(".value{font-size: 3rem; font-weight: bold;}</style></head>");
          client.println("<body><h1>ESP32 DC Smart Meter</h1>");
          client.println("<p>Voltage: <span class='value'>" + String(voltage, 3) + "
V</span></p>");
          client.println("<p>Current: <span class='value'>" + String(current, 3) + "
A</span></p>");
          client.println("<p>Power: <span class='value'>" + String(power, 3) + "
W</span></p>");
          client.println("<p>Energy Consumed: <span class='value'>" + String(energy, 6) + "
Wh</span></p>");
          client.println("</body></html>");
          client.println();
          break;
        } else {
          currentLine = "";
        }
      } else if (c != '\r') {
        currentLine += c;
```

```
      }
     }
    }
    header = "";
    client.stop();
  }
}
```

---

8. คำถามท้ายการทดลอง (ข้อ 9)

1. การเพิ่มความละเอียดของ ADC จาก 10 บิต เป็น 12 บิต ส่งผลดีต่อ "ความละเอียด (Resolution)" ในการวัดอย่างไร และอาจส่งผลเสียในด้านใดบ้าง (เช่น ความเร็ว, สัญญาณรบกวน)?
2. จงเปรียบเทียบข้อดีและข้อเสียระหว่างการทำ Calibration แบบ Multi-point Linear และ Polynomial ดีกรี 2 ท่านจะเลือกใช้วิธีใดในการวัดพลังงานที่ต้องการความน่าเชื่อถือสูง เพราะเหตุใด?
3. นอกเหนือจากความไม่สมบูรณ์ของเซ็นเซอร์แล้ว ท่านคิดว่ามีปัจจัยแวดล้อมใดอีกบ้างที่สามารถเป็น "แหล่งที่มาของความคลาดเคลื่อน (Source of Error)" ในการทดลองนี้?
4. จากแนวคิด Machine Learning ในข้อ 6 หากเราต้องการให้ระบบปรับเทียบตัวเองโดยอัตโนมัติโดยสมบูรณ์ (ไม่มีคนป้อนค่า) ท่านพอจะจินตนาการถึงวิธีการหรือเซ็นเซอร์อ้างอิงประเภทใด ที่จะทำให้ระบบสามารถหา "ค่าจริง" มาใช้ในการเรียนรู้ได้?
5. หากท่านได้รับมอบหมายให้นำระบบนี้ไปพัฒนาต่อเพื่อวัดการใช้พลังงานของ "เครื่องปรับอากาศ" ในบ้าน ซึ่งใช้ไฟฟ้า AC ท่านจะต้องปรับเปลี่ยนฮาร์ดแวร์และซอฟต์แวร์ในส่วนใดบ้างที่สำคัญที่สุด?

การวัดค่าลบ (Bi-directional measurement) เป็นความสามารถที่จำเป็นในหลายสถานการณ์ เช่น การวัดการชาร์จ/ดิสชาร์จของแบตเตอรี่ หรือการวัดแรงดันในวงจรที่มีไฟเลี้ยงทั้งบวกและลบ

ในฐานะศาสตราจารย์ ผมจะอธิบายหลักการ, การออกแบบวงจรเพิ่มเติม, และโค้ดที่จำเป็นในการยกระดับแล็บของเราให้เป็น Digital Voltmeter และ Ammeter ที่สามารถวัดค่าบวกลบได้จริง

หลักการสำคัญ: ข้อจำกัดของ ADC และแนวทางแก้ไข

หัวใจของปัญหาคือ **ADC ของ ESP32 สามารถรับแรงดันไฟฟ้าได้ในช่วง 0V ถึง ~3.3V เท่านั้น** มันไม่สามารถเข้าใจ "แรงดันไฟฟ้าลบ" ได้โดยตรง ดังนั้นภารกิจของเราคือ:

1. **สำหรับ Current Sensor (ACS712):** เซ็นเซอร์นี้ **ถูกออกแบบมาให้วัดค่าบวกลบได้อยู่แล้ว!** สัญญาณ Output ของมันจะอยู่ที่กึ่งกลางของไฟเลี้ยง (ประมาณ 2.5V) เมื่อไม่มีกระแสไหล หากกระแสไหลในทิศทางหนึ่ง แรงดันจะสูงกว่า 2.5V และหากไหลในทิศทางตรงกันข้าม แรงดันจะต่ำกว่า 2.5V เราจึงไม่ต้องแก้อะไรที่ฮาร์ดแวร์ แต่ต้องแก้ที่ **ซอฟต์แวร์** เพื่อตีความค่านี้ให้ถูกต้อง
2. **สำหรับ Voltage Sensor:** โมดูล Voltage Divider ทั่วไป **ไม่สามารถวัดค่าลบได้** หากเราป้อนแรงดันลบเข้าไป สัญญาณ Output ที่ออกมาก็จะเป็นลบ ซึ่งอาจสร้างความเสียหายให้กับขา ADC ของ ESP32 ได้ ดังนั้น เราจึง **จำเป็นต้องมีอุปกรณ์อิเล็กทรอนิกส์ต่อเพิ่ม** เพื่อ "ปรับ" สัญญาณให้อยู่ในช่วงที่ ADC รับได้

อุปกรณ์อิเล็กทรอนิกส์ที่ต้องต่อเพิ่ม (สำหรับ Voltmeter)

เราจะใช้วงจร **Operational Amplifier (Op-Amp)** ในการทำหน้าที่ **"Level Shifting and Scaling"** คือทั้งปรับขนาดและยกระดับแรงดันไฟฟ้า วงจรที่เราจะสร้างจะแปลงแรงดัน Input ในช่วง **-25V ถึง +25V** ให้มาอยู่ในช่วง **0V ถึง ~3.3V**

วงจรที่แนะนำ: **Inverting Summing Amplifier**

[แผนภาพวงจร Op-Amp สำหรับ Level Shifting]

คำอธิบายวงจร:

1. **Input Voltage Divider (R1, R2):** ขั้นแรก เราต้องลดทอนแรงดัน Input -25V ถึง +25V ลงมาอยู่ในช่วงที่ปลอดภัยสำหรับ Op-Amp ก่อน เราจะใช้ตัวต้านทาน R1=100kΩ และ R2=10kΩ ซึ่งจะลดทอนแรงดันลง 11 เท่า ทำให้ช่วงแรงดันใหม่คือ **-2.27V ถึง +2.27V**
2. **Op-Amp (LM358 หรือเทียบเท่า)**: เราจะใช้ Op-Amp ยอดนิยมอย่าง LM358 ซึ่งสามารถทำงานได้จากแหล่งจ่ายไฟเดี่ยว (Single Supply) 5V
3. **Level Shifting (R4, R5):** เราสร้างแรงดันอ้างอิง **1.65V** (ครึ่งหนึ่งของ 3.3V) ด้วย Voltage Divider (R4, R5) เพื่อใช้เป็น "จุดศูนย์ใหม่" ของเรา
4. **Inverting Amplifier (R3, Rf):** วงจรนี้จะทำการรวมสัญญาณจาก Input ที่ลดทอนแล้ว กับแรงดันอ้างอิง 1.65V เข้าด้วยกัน และขยาย/ลดทอนตามอัตราส่วน Rf/R3
    - เมื่อ Input เป็น -25V (หลังลดทอนเป็น -2.27V) -> Output จะเข้าใกล้ 3.3V
    - เมื่อ Input เป็น 0V -> Output จะอยู่ที่ 1.65V พอดี
    - เมื่อ Input เป็น +25V (หลังลดทอนเป็น +2.27V) -> Output จะเข้าใกล้ 0V *(หมายเหตุ: วงจรนี้จะให้ผลกลับเฟส ซึ่งเราสามารถแก้ได้ในซอฟต์แวร์)*

การเชื่อมต่อ Sensor กับ ESP32 (ฉบับสมบูรณ์)

แผนผังวงจรการเชื่อมต่อ ESP32 กับวงจร Op-Amp และ ACS712

[แผนภาพการเชื่อมต่อ ESP32 กับวงจร Op-Amp และ ACS712]

สรุปการเชื่อมต่อ:

- **ESP32 VIN (5V)** -> VCC ของ ACS712, VCC ของ Op-Amp (LM358 Pin 8)
- **ESP32 3V3** -> ใช้เป็นแหล่งจ่ายสำหรับสร้างแรงดันอ้างอิง 1.65V ในวงจร Op-Amp
- **ESP32 GND** -> GND ของ ACS712, GND ของ Op-Amp (LM358 Pin 4), GND ของวงจรทั้งหมด
- **ESP32 GPIO34 (ADC สำหรับ Voltage)** -> Vout จากวงจร Op-Amp (LM358 Pin 1)
- **ESP32 GPIO35 (ADC สำหรับ Current)** -> OUT จาก ACS712

Code ESP32 สำหรับ Digital Volt/Amp Meter (วัดค่าลบได้)

โค้ดนี้จะถูกปรับปรุงจากแล็บเดิม โดยเพิ่มตรรกะในการจัดการกับค่าศูนย์ (Zero Offset) และสมการ Calibration ใหม่

C++

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

// --- Configuration ---
#define ADC_BITS 12
#define VOLTAGE_PIN 34
#define CURRENT_PIN 35

// --- Zero Offset Calibration (สำคัญมาก!) ---
// ค่า ADC ที่อ่านได้เมื่อ Input เป็นศูนย์ (ต้องทำการวัดจริง)
const int ZERO_VOLTAGE_ADC = 2048;  // ค่าทางทฤษฎีคือ 4096/2 = 2048 (เมื่อ V_in = 0V)
const int ZERO_CURRENT_ADC = 2035;  // ค่าจริงที่วัดได้เมื่อ I_in = 0A (อาจไม่ตรง 2048 เป๊ะ)
```

```
// --- Calibration Coefficients (ต้องหาค่าใหม่จากการทดลองกับวงจรนี้) ---
// y = m * (x - x_offset)
// สังเกตว่าเราจะไม่มี c อีกต่อไป เพราะ offset ถูกจัดการโดย ZERO_ADC แล้ว
// ค่า Slope สำหรับ Voltage จะเป็นลบ เพราะวงจร Op-Amp ของเรากลับเฟส
const float VOLTAGE_SLOPE_M = -0.0245; // ตัวอย่าง
const float CURRENT_SLOPE_M = 0.0244;  // ตัวอย่าง (สำหรับ ACS712 5A, VCC=5V, จะประมาณ 5A / 1024 ADC steps = 0.00488 V/step -> 0.0244 A/step)


// Web Server
WiFiServer server(80);

// Global variables
float voltage = 0.0;
float current = 0.0;
float power = 0.0;

void setup() {
  Serial.begin(115200);
  analogReadResolution(ADC_BITS);

  WiFiManager wm;
  if (!wm.autoConnect("Bi-DirectionalMeter")) {
    Serial.println("Failed to connect");
    ESP.restart();
  }
  Serial.println("Connected!");
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
```

```
  server.begin();
}

void loop() {
  // --- อ่านค่า Raw ADC ---
  int raw_v = analogRead(VOLTAGE_PIN);
  int raw_i = analogRead(CURRENT_PIN);

  // --- คำนวณค่าจริง (มีการจัดการ Offset) ---
  // 1. คำนวณ Current
  // ลบค่า Offset ก่อน แล้วค่อยคูณด้วย Slope
  current = (raw_i - ZERO_CURRENT_ADC) * CURRENT_SLOPE_M;

  // 2. คำนวณ Voltage
  voltage = (raw_v - ZERO_VOLTAGE_ADC) * VOLTAGE_SLOPE_M;

  // 3. คำนวณ Power
  power = voltage * current;

  // --- แสดงผลทาง Serial Monitor ---
  Serial.printf("Raw V: %d, Raw I: %d | Voltage: %+.3f V, Current: %+.3f A, Power: %+.3f
W\n",
          raw_v, raw_i, voltage, current, power);

  // --- จัดการ Web Server ---
  handleWebServer();

  delay(1000);
```

```
}

void handleWebServer() {
  WiFiClient client = server.available();
  if (client) {
    // (โค้ดส่วนของ Web Server เหมือนเดิม แต่ปรับการแสดงผล)
    client.println("<!DOCTYPE html><html><head><title>Bi-directional Meter</title>");
    client.println("<meta name='viewport' content='width=device-width, initial-scale=1'>");
    client.println("<style>html {font-family: Helvetica; text-align: center;}");
    client.println(".value{font-size: 3rem; font-weight: bold; color: #4CAF50;}");
    client.println(".neg_value{color: #f44336;}</style></head>");
    client.println("<body><h1>ESP32 Bi-directional Meter</h1>");

    // Logic for positive/negative color
    String v_class = (voltage < 0) ? "neg_value" : "value";
    String i_class = (current < 0) ? "neg_value" : "value";
    String p_class = (power < 0) ? "neg_value" : "value";

    client.println("<p>Voltage: <span class='" + v_class + "'>" + String(voltage, 3) + "
V</span></p>");
    client.println("<p>Current: <span class='" + i_class + "'>" + String(current, 3) + "
A</span></p>");
    client.println("<p>Power: <span class='" + p_class + "'>" + String(power, 3) + "
W</span></p>");

    client.println("</body></html>");

    // Close connection
    delay(1);
    client.stop();
  }
```

}

สรุปและขั้นตอนการปฏิบัติ

1. **สร้างวงจร Op-Amp:** ประกอบวงจร Level Shifter สำหรับ Voltmeter ตามแผนภาพอย่างระมัดระวัง
2. **เชื่อมต่อทุกอย่าง**: ต่อวงจร Op-Amp และ ACS712 เข้ากับ ESP32 ตามแผนภาพฉบับสมบูรณ์
3. **หาค่า Zero Offset:**
    - อัปโหลดโค้ดเบื้องต้นที่อ่านค่า analogRead() แล้วแสดงผลทาง Serial Monitor เท่านั้น
    - **สำคัญ**: **ยังไม่ต้องต่อแหล่งจ่ายไฟหรือโหลดใด ๆ เข้ากับเซ็นเซอร์** (Input V และ I ต้องเป็นศูนย์)
    - จดค่า Raw ADC ที่นิ่งที่สุดจาก GPIO34 และ GPIO35 ค่าเหล่านี้คือ ZERO_VOLTAGE_ADC และ ZERO_CURRENT_ADC ของคุณ นำไปใส่ในโค้ด
4. **หาค่า Slope (m):**
    - ต่อแหล่งจ่ายไฟและมัลติมิเตอร์
    - จ่ายแรงดัน/กระแส ค่าบวก 1 ค่า (เช่น +12V) และค่าลบ 1 ค่า (เช่น -12V)
    - บันทึกค่า Raw ADC และค่าจริงจากมัลติมิเตอร์
    - คำนวณหาค่า Slope m = (y2 - y1) / ((x2 - x_offset) - (x1 - x_offset))
5. **อัปโหลดโค้ดฉบับสมบูรณ์**: ใส่ค่า Offset และ Slope ที่หาได้ลงในโค้ด แล้วอัปโหลดเพื่อใช้งานจริง