2025 TEEP Progress Report Week-7

Used Wokwi to complete the tasks for Labs 1 through 6

(Before creating the physical project, please use **Wokwi** for online simulation to complete the initial design of the circuit and code. Then, use **Thonny** together with the **ESP32** development board to build the actual project. Please **record a video of the project** in operation, upload it to YouTube, and provide the video link(LAB7) so we can better understand your learning progress.)

Lab1- Short Answer :

1. When using an I²C LCD to communicate with the ESP32, which two main signal lines does the I²C protocol require?

Ans: SDA (Serial Data) and SCL (Serial Clock).

---

2. In MicroPython, which external libraries can be used to drive an I²C LCD?

Ans: Common libraries include `i2c_lcd` (custom driver), `lcd_api.py`, or third-party libraries such as `esp32_i2c_lcd`.

---

3. What are the common display sizes of I²C LCD modules? (e.g., 16x2, 20x4)

Ans: 16x2 (16 characters × 2 lines) and 20x4 (20 characters × 4 lines).

---

4. What is the recommended input voltage range for the TP4056 charging IC, and what is the typical value used in applications?

Ans: Recommended range is 4.0V to 8.0V; the typical input voltage used is **5V (USB power supply)**.

---

5. What is the maximum charging current supported by the TP4056, and how can it be adjusted?

Ans: Up to **1A maximum**; the charging current is set by the **value of the PROG resistor** connected to the IC.

---

6. Why can the TP4056 only be used to charge a single-cell Li-Ion/Li-Polymer battery and not multiple cells in series?

Ans: Because it regulates charging based on a single-cell voltage (4.2V). Multiple cells in series require a **battery management system (BMS)** with cell balancing.

---

7. What safety features are integrated into the TP4056 to protect the battery during charging?

Ans:

- Over-temperature protection
- Over-current protection
- Trickle charge for deeply discharged cells
- Automatic constant-current/constant-voltage charging
- Automatic termination when fully charged

---

8. When the battery is fully charged, what happens to the charging process in TP4056, and how is this indicated on the module (LED status)?

Ans: The TP4056 automatically terminates charging and enters standby mode. The **CHRG LED (red)** turns **off**, and the **STDBY LED (blue/green)** turns **on** to indicate a full charge.

Lab2 – On ESP32 , complete the following tasks:

1. Create a temperature and humidity sensor system using DHT11, along with two LEDs (a yellow LED named LED_Hum and a red LED named LED_Temp).

Connect the data pin of the DHT11 to GPIO XX, LED_Hum to GPIO XX, and LED_Temp to GPIO XX.

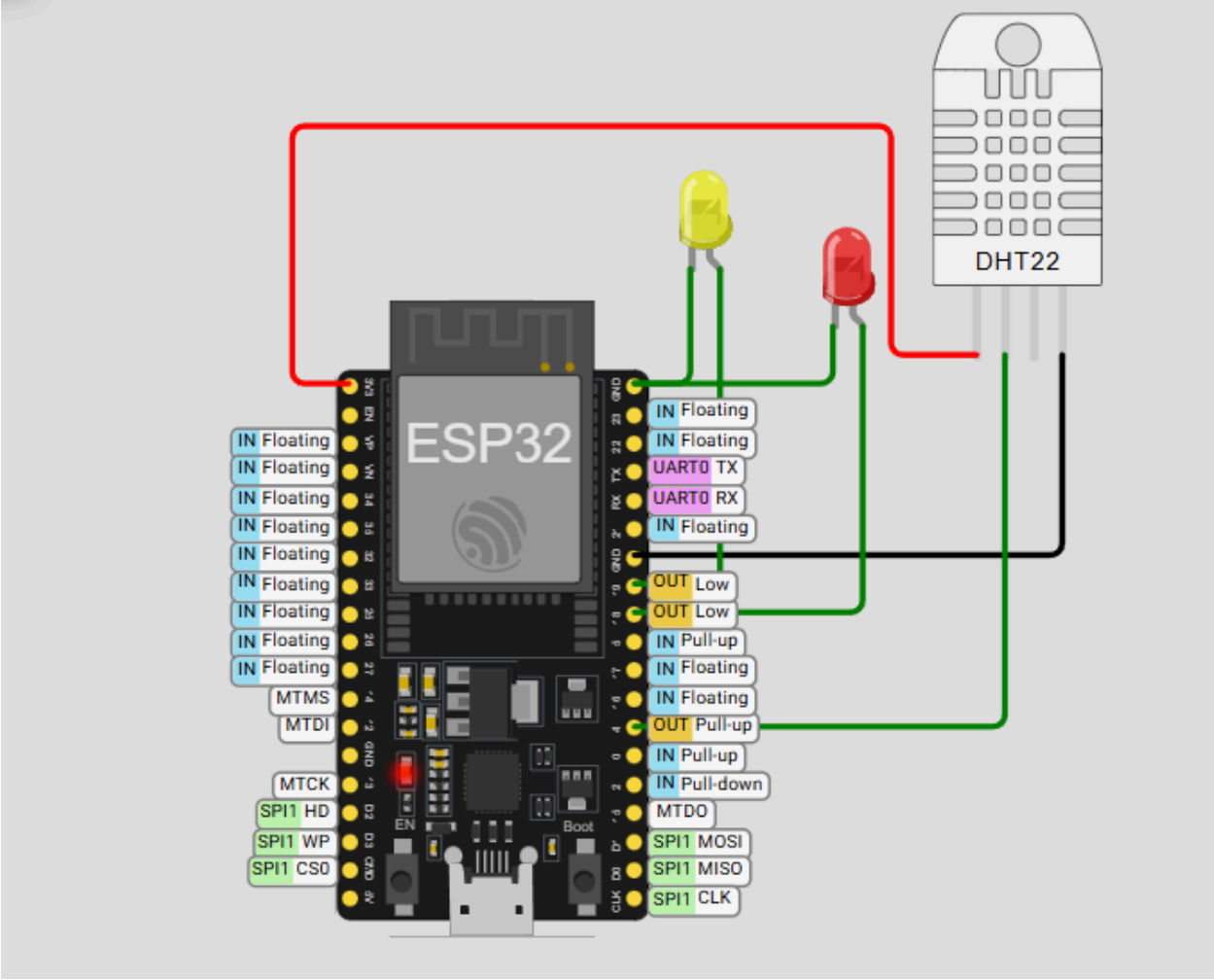
- **Function 1**: When the program starts, perform voltage detection on the ADC and read the temperature and humidity from the DHT11. Every 3 seconds, collect one set of temperature and humidity data from the DHT11. After obtaining 10 sets of data, calculate the average values and display them in the Shell, then **publish** the results to the ThingSpeak platform using MQTT.
- **Function 2**:When the temperature is above 30 degrees, LED_Temp lights up.
- **Function 3:**When the humidity is higher than 60%, LED_Hum lights up.

Hint: The name needs to be displayed on each Field

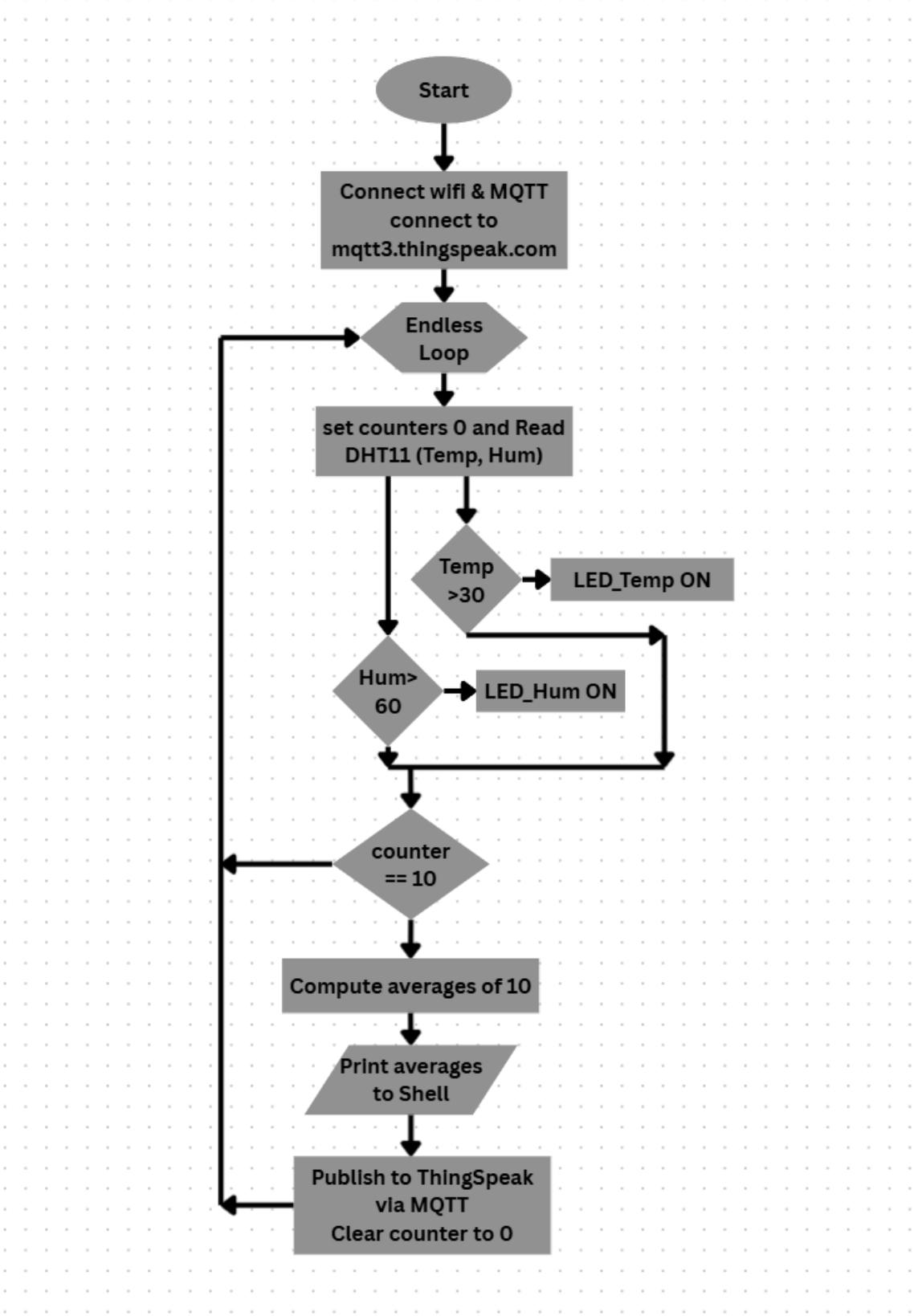
- When the temperature exceeds 35°C, LED_Temp lights up.
- When the humidity exceeds 60%, LED_Hum lights up.
- On the ThingSpeak platform, collect 10 sets of temperature and humidity data, and take a screenshot as a record.

Connections :

- DHT11 data → GPIO 4
- LED_Temp (red) → GPIO 18
- LED_Hum (yellow) → GPIO 19



Flowchart :


Start
Connect wifi & MQTT connect to mqtt3.thingspeak.com
Endless Loop
set counters 0 and Read DHT11 (Temp, Hum)
Temp >30
LED_Temp ON
Hum> 60
LED_Hum ON
counter == 10
Compute averages of 10
Print averages to Shell
Publish to ThingSpeak via MQTT
Clear counter to 0

Source code :

```
import time
import network
from machine import Pin, ADC
import dht
from umqtt.simple import MQTTClient

# ---------- USER SETTINGS ---------- Enetr your wifi details
WIFI_SSID = "ISILAB CR"
WIFI_PASS = "isilab.ncut.CR"

# ThingSpeak MQTT (create Devices->MQTT and download credentials)
TS_CLIENT_ID = "FQgvNw4IAT01NiMJPSoVJx0"
TS_USERNAME  = "FQgvNw4IAT01NiMJPSoVJx0"
TS_PASSWORD  = "p+2q/lPu8fCOMAclznbe1eQc"
TS_CHANNEL_ID = "3064985"  # numeric string, e.g., "2345678"
MQTT_BROKER = "mqtt3.thingspeak.com"
MQTT_PORT   = 1883
MQTT_TOPIC  = "channels/{}/publish".format(TS_CHANNEL_ID)

# GPIOs
PIN_DHT = 4
PIN_LED_TEMP = 18
PIN_LED_HUM  = 19
PIN_ADC_VSENSE = 34  # voltage check

TEMP_THRESHOLD = 30.0  # LED Temp ON
HUM_THRESHOLD  = 60.0  # LED Hum ON
SAMPLE_INTERVAL_S = 3
BATCH_SIZE = 10
```

```
# ---------- Setup ----------
sensor = dht.DHT11(Pin(PIN_DHT))
led_temp = Pin(PIN_LED_TEMP, Pin.OUT, value=0)
led_hum  = Pin(PIN_LED_HUM,  Pin.OUT, value=0)
adc = ADC(Pin(PIN_ADC_VSENSE))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_12BIT)

def wifi_connect():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print("Connecting Wi-Fi...")
        wlan.connect(WIFI_SSID, WIFI_PASS)
        while not wlan.isconnected():
            time.sleep(0.5)
    print("Wi-Fi OK:", wlan.ifconfig())
    return wlan

def mqtt_connect():
    client = MQTTClient(TS_CLIENT_ID, MQTT_BROKER, port=MQTT_PORT,
                                    user=TS_USERNAME, password=TS_PASSWORD,
keepalive=60)
    client.connect()
    print("MQTT connected")
    return client

def read_adc_voltage():
    raw = adc.read()
    v = (raw / 4095.0) * 3.3
```

```
    return raw, v
def safe_read_dht():
    try:
        sensor.measure()
        return sensor.temperature(), sensor.humidity()
    except:
        return None, None
def update_leds(t, h):
    led_temp.value(1 if (t and t > TEMP_THRESHOLD) else 0)
    led_hum.value(1  if (h and h > HUM_THRESHOLD) else 0)


# ---------- Main ----------
def main():
    wifi_connect()
    client = mqtt_connect()
    raw, v = read_adc_voltage()
    print("ADC startup voltage: raw={}, approx={:.2f} V".format(raw, v))
    temps, hums = [], []
    sample_count = 0
    while True:
        t, h = safe_read_dht()
        if t is not None and h is not None:
            sample_count += 1
            temps.append(t)
            hums.append(h)
            update_leds(t, h)
                print("[{:02d}/{}] T={:.1f}°C   H={:.1f}%   | LEDs: Temp={}
Hum={}".format(
                        sample_count, BATCH_SIZE, t, h, led_temp.value(),
led_hum.value()))
```

```
            if sample_count >= BATCH_SIZE:
                avg_t = sum(temps) / len(temps)
                avg_h = sum(hums)  / len(hums)
                                        print("Averages  →  Temp={:.2f}°C,
Hum={:.2f}%".format(avg_t, avg_h))
                                                                payload    =
"field1={:.2f}&field2={:.2f}&status=OK".format(avg_t, avg_h)
                client.publish(MQTT_TOPIC, payload)
                print("Published to ThingSpeak:", payload)
                temps.clear()
                hums.clear()
                sample_count = 0
        else:
            print("DHT11 read error")
        time.sleep(SAMPLE_INTERVAL_S)


main()
```

Test Records :

```
Connecting Wi-Fi...
Wi-Fi OK: ('10.10.0.2', '255.255.0.0', '10.0.0.1', '10.0.0.1')
MQTT connected
ADC startup voltage: raw=1135, approx=0.91 V
[01/10] T=-1.9°C  H=30.0%  | LEDs: Temp=0 Hum=0
[02/10] T=-1.9°C  H=30.0%  | LEDs: Temp=0 Hum=0
[03/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[04/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[05/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[06/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[07/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[08/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[09/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[10/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
Averages → Temp=42.90°C, Hum=71.60%
Published to ThingSpeak: field1=42.90&field2=71.60&status=OK
[01/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[02/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[03/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[04/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[05/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[06/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[07/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[08/10] T=54.1°C  H=82.0%  | LEDs: Temp=1 Hum=1
[09/10] T=-16.4°C  H=82.0%  | LEDs: Temp=0 Hum=1
[10/10] T=-16.4°C  H=9.5%  | LEDs: Temp=0 Hum=0
Averages → Temp=40.00°C, Hum=74.75%
Published to ThingSpeak: field1=40.00&field2=74.75&status=OK
[01/10] T=-16.4°C  H=9.5%  | LEDs: Temp=0 Hum=0
[02/10] T=-16.4°C  H=9.5%  | LEDs: Temp=0 Hum=0
[03/10] T=-16.4°C  H=9.5%  | LEDs: Temp=0 Hum=0
[04/10] T=6.8°C  H=25.0%  | LEDs: Temp=0 Hum=0
[05/10] T=6.8°C  H=25.0%  | LEDs: Temp=0 Hum=0
[06/10] T=6.8°C  H=25.0%  | LEDs: Temp=0 Hum=0
[07/10] T=6.8°C  H=25.0%  | LEDs: Temp=0 Hum=0
```

```
MPY: soft reboot
Connecting Wi-Fi...
Wi-Fi OK: ('192.168.50.209', '255.255.255.0', '192.168.50.1', '192.168.50.1')
MQTT connected
ADC startup voltage: raw=0, approx=0.00 V
[01/10] T=0.0°C  H=0.0%  | LEDs: Temp=0 Hum=0
[02/10] T=23.0°C  H=37.0%  | LEDs: Temp=0 Hum=0
[03/10] T=23.0°C  H=37.0%  | LEDs: Temp=0 Hum=0
[04/10] T=23.0°C  H=36.0%  | LEDs: Temp=0 Hum=0
[05/10] T=23.0°C  H=36.0%  | LEDs: Temp=0 Hum=0
[06/10] T=23.0°C  H=68.0%  | LEDs: Temp=0 Hum=1
[07/10] T=24.0°C  H=74.0%  | LEDs: Temp=0 Hum=1
[08/10] T=24.0°C  H=75.0%  | LEDs: Temp=0 Hum=1
[09/10] T=24.0°C  H=81.0%  | LEDs: Temp=0 Hum=1
[10/10] T=24.0°C  H=80.0%  | LEDs: Temp=0 Hum=1
Averages → Temp=21.10°C, Hum=52.40%
Published to ThingSpeak: field1=21.10&field2=52.40&status=OK
[01/10] T=24.0°C  H=80.0%  | LEDs: Temp=0 Hum=1
[02/10] T=24.0°C  H=77.0%  | LEDs: Temp=0 Hum=1
[03/10] T=24.0°C  H=72.0%  | LEDs: Temp=0 Hum=1
[04/10] T=24.0°C  H=67.0%  | LEDs: Temp=0 Hum=1
[05/10] T=24.0°C  H=61.0%  | LEDs: Temp=0 Hum=1
[06/10] T=24.0°C  H=55.0%  | LEDs: Temp=0 Hum=0
[07/10] T=24.0°C  H=50.0%  | LEDs: Temp=0 Hum=0
[08/10] T=24.0°C  H=46.0%  | LEDs: Temp=0 Hum=0
[09/10] T=24.0°C  H=43.0%  | LEDs: Temp=0 Hum=0
[10/10] T=24.0°C  H=41.0%  | LEDs: Temp=0 Hum=0
Averages → Temp=24.00°C, Hum=59.20%
Published to ThingSpeak: field1=24.00&field2=59.20&status=OK
[01/10] T=24.0°C  H=40.0%  | LEDs: Temp=0 Hum=0
[02/10] T=24.0°C  H=39.0%  | LEDs: Temp=0 Hum=0
[03/10] T=24.0°C  H=39.0%  | LEDs: Temp=0 Hum=0
[04/10] T=24.0°C  H=38.0%  | LEDs: Temp=0 Hum=0
[05/10] T=24.0°C  H=37.0%  | LEDs: Temp=0 Hum=0
[06/10] T=24.0°C  H=37.0%  | LEDs: Temp=0 Hum=0
[07/10] T=24.0°C  H=36.0%  | LEDs: Temp=0 Hum=0
[08/10] T=24.0°C  H=36.0%  | LEDs: Temp=0 Hum=0
[09/10] T=24.0°C  H=36.0%  | LEDs: Temp=0 Hum=0
[10/10] T=24.0°C  H=36.0%  | LEDs: Temp=0 Hum=0
Averages → Temp=24.00°C, Hum=37.40%
Published to ThingSpeak: field1=24.00&field2=37.40&status=OK
[01/10] T=24.0°C  H=35.0%  | LEDs: Temp=0 Hum=0
[02/10] T=24.0°C  H=35.0%  | LEDs: Temp=0 Hum=0
[03/10] T=24.0°C  H=35.0%  | LEDs: Temp=0 Hum=0
[04/10] T=24.0°C  H=35.0%  | LEDs: Temp=0 Hum=0
[05/10] T=24.0°C  H=35.0%  | LEDs: Temp=0 Hum=0
[06/10] T=24.0°C  H=35.0%  | LEDs: Temp=0 Hum=0
[07/10] T=24.0°C  H=34.0%  | LEDs: Temp=0 Hum=0
[08/10] T=24.0°C  H=34.0%  | LEDs: Temp=0 Hum=0
```

7-1

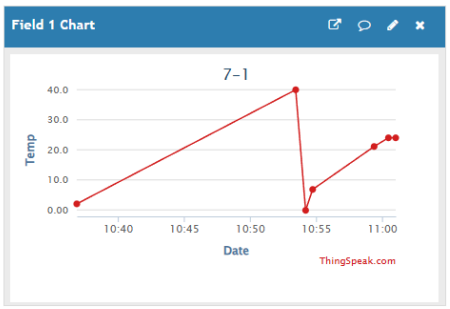
Channel ID: **3064985**
Author: mwa0000038533551
Access: Public

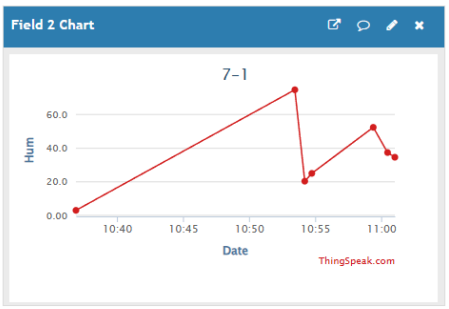
Private View | Public View | Channel Settings | Sharing | API Keys | Data Import / Export

Add Visualizations | Add Widgets | Export recent data | MATLAB Analysis | MATLAB Visualization

Channel Stats

Created: about an hour ago
Last entry: 34 minutes ago
Entries: 9





Lab3 – OnESP32, complete the following tasks:

Two LEDs (a yellow LED named LED_Hum and a red LED named LED_Temp).

Connect LED_Hum to GPIO XX and LED_Temp to GPIO XX.

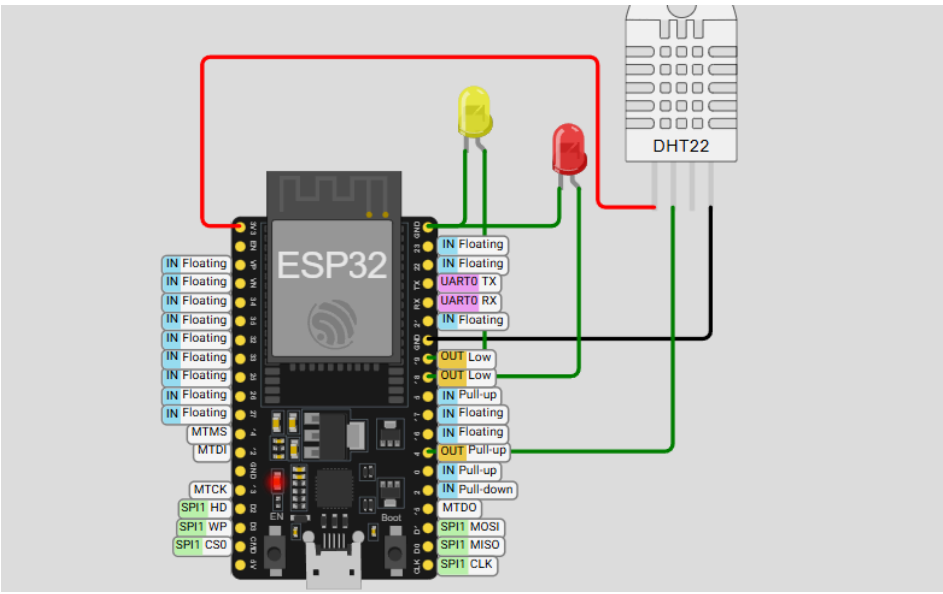
- Function 1: When the program starts, use MQTT to subscribe to the DHT11 temperature data (Field2) and DHT11 humidity data (Field3) from ThingSpeak, retrieve the data, and display it in the Shell.
- Function 2:When the temperature is above 30 degrees, LED_Temp lights .up
- Function 3:When the humidity is higher than 60%, LED_Hum lights up.

Hint: The name needs to be displayed on each Field

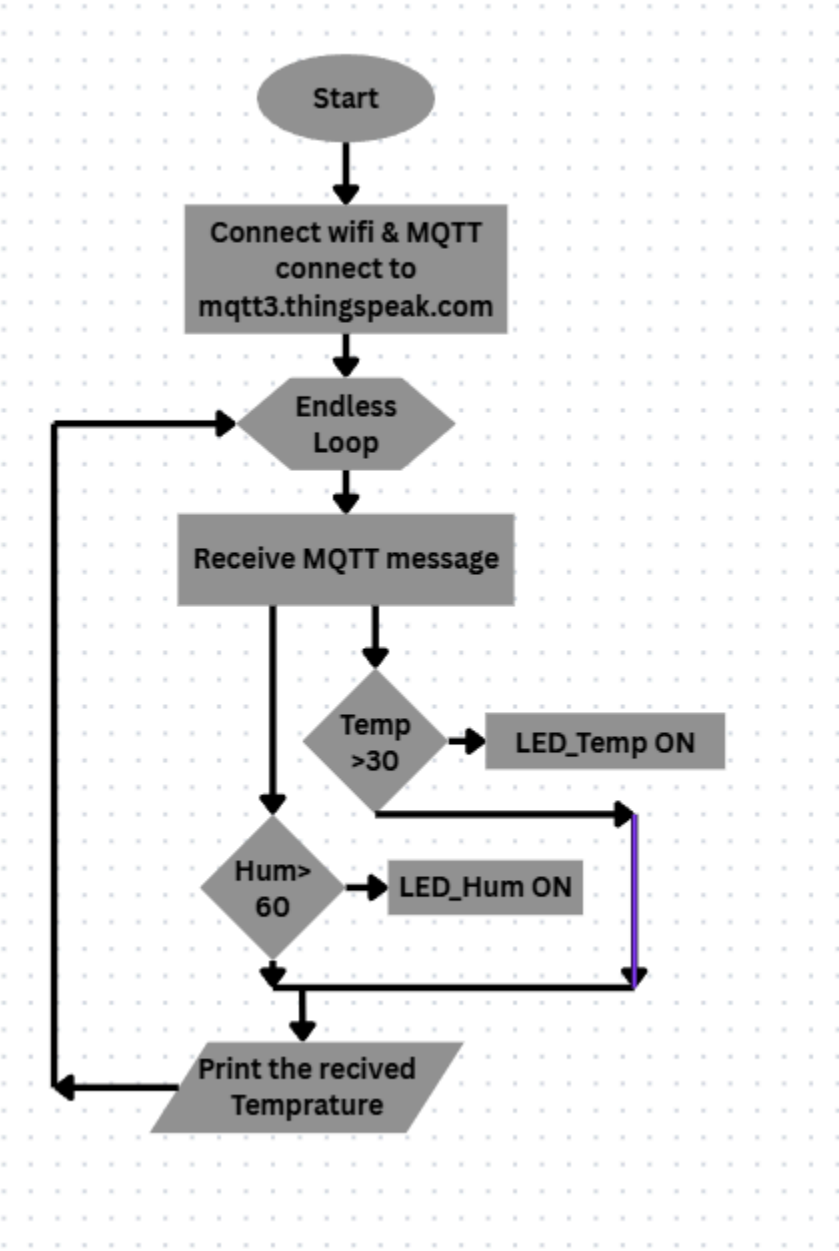
- When the temperature exceeds 35°C, LED_Temp lights up.
- When the humidity exceeds 60%, LED_Hum lights up.
- On the ThingSpeak platform, collect 10 sets of temperature and humidity data, and take a screenshot as a record.

Connections :

- DHT11 data → GPIO 4
- LED_Temp (red) → GPIO 18
- LED_Hum (yellow) → GPIO 19



Flowchart :



Source Code :

```
# Integrated Lab2 (Publisher) + Lab3 (Subscriber) Example
# ESP32 MicroPython

import time
import network
from machine import Pin
from umqtt.simple import MQTTClient
```

```
from dht import DHT11

# ---------- USER SETTINGS ----------
WIFI_SSID = "ISILAB CR"
WIFI_PASS = "isilab.ncut.CR"

# ThingSpeak MQTT (create Devices->MQTT and download credentials)
TS_CLIENT_ID = "FQgvNw4IAT01NiMJPSoVJx0"
TS_USERNAME  = "FQgvNw4IAT01NiMJPSoVJx0"
TS_PASSWORD  = "p+2q/lPu8fCOMAclznbe1eQc"
TS_CHANNEL_ID = "3064985"  # numeric string, e.g., "2345678"
MQTT_BROKER = "mqtt3.thingspeak.com"
MQTT_PORT   = 1883

# GPIO Pins
DHT_PIN = 22        # DHT11 data pin
PIN_LED_TEMP = 18
PIN_LED_HUM  = 19

TEMP_THRESHOLD = 30.0
HUM_THRESHOLD  = 60.0

# ---------------- SETUP ----------------
dht_sensor = DHT11(Pin(DHT_PIN))
led_temp = Pin(PIN_LED_TEMP, Pin.OUT, value=0)
led_hum  = Pin(PIN_LED_HUM, Pin.OUT, value=0)

latest_temp = None
latest_hum = None

# ---------------- WIFI ----------------
def wifi_connect():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print("Connecting Wi-Fi...")
        wlan.connect(WIFI_SSID, WIFI_PASS)
        while not wlan.isconnected():
            time.sleep(0.5)
    print("Wi-Fi OK:", wlan.ifconfig())
```

```
# ---------------- LED Update ----------------
def update_leds():
    global latest_temp, latest_hum
    if latest_temp is not None:
        led_temp.value(1 if latest_temp > TEMP_THRESHOLD else 0)
    if latest_hum is not None:
        led_hum.value(1 if latest_hum > HUM_THRESHOLD else 0)

# ---------------- MQTT Callback ----------------
def sub_cb(topic, msg):
    global latest_temp, latest_hum
    try:
        value = float(msg.decode())
    except:
        print("Invalid payload:", msg)
        return

    if b"field1" in topic:  # Temperature
        latest_temp = value
        print("Received Field1 (Temperature): {:.2f}
°C".format(latest_temp))
    elif b"field2" in topic:  # Humidity
        latest_hum = value
        print("Received Field2 (Humidity): {:.2f} %".format(latest_hum))

    update_leds()

# ---------------- MQTT Connect ----------------
def mqtt_connect():
    client = MQTTClient(client_id=TS_CLIENT_ID,
                        server=MQTT_BROKER,
                        port=MQTT_PORT,
                        user=TS_USERNAME,
                        password=TS_PASSWORD,
                        keepalive=60)
    client.set_callback(sub_cb)
    client.connect()
    # Subscribe to Field1 and Field2 for Lab3 reading
```

```
client.subscribe("channels/{}/subscribe/fields/field1".format(TS_CHANNEL_I
D))

client.subscribe("channels/{}/subscribe/fields/field2".format(TS_CHANNEL_I
D))
    print("MQTT connected and subscribed to topics.")
    return client

# ---------------- Publish Function ----------------
def publish(client, temp, hum):
    try:
        topic = "channels/{}/publish/fields/field1".format(TS_CHANNEL_ID)
        client.publish(topic, str(temp))
        topic2 = "channels/{}/publish/fields/field2".format(TS_CHANNEL_ID)
        client.publish(topic2, str(hum))
        print("Published -> Temp: {:.2f}, Hum: {:.2f}".format(temp, hum))
    except OSError as e:
        print("Publish failed:", e)

# ---------------- MAIN LOOP ----------------
def main():
    wifi_connect()
    client = mqtt_connect()

    try:
        while True:
            # --- Read DHT sensor ---
            try:
                dht_sensor.measure()
                temp = dht_sensor.temperature()
                hum  = dht_sensor.humidity()
                latest_temp = temp
                latest_hum  = hum
                update_leds()
            except Exception as e:
                print("DHT read error:", e)
                temp = 0
                hum  = 0
```

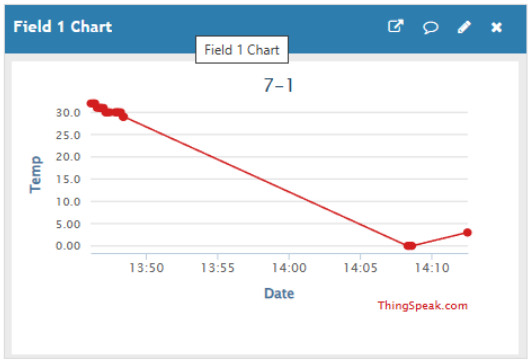
```
            # --- Publish to ThingSpeak ---
            publish(client, temp, hum)

            # --- Check for incoming MQTT messages ---
            try:
                client.check_msg()
            except OSError as e:
                if e.args[0] != -1:
                    print("MQTT error:", e, "→ reconnecting in 5s")
                    try:
                        client.disconnect()
                    except:
                        pass
                    time.sleep(5)
                    client = mqtt_connect()

            time.sleep(3)  # wait 3 seconds before next reading

    except KeyboardInterrupt:
        print("Stopped.")
        try:
            client.disconnect()
        except:
            pass
        led_temp.value(0)
        led_hum.value(0)
main()
```

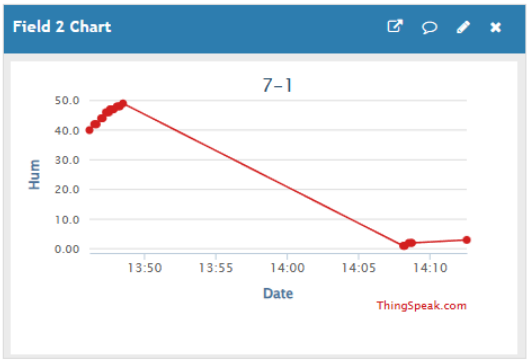
Test Records :

```
Connecting Wi-Fi...
Wi-Fi OK: ('10.10.0.2', '255.255.0.0', '10.0.0.1', '10.0.0.1')
MQTT connected and subscribed to topics.
Published -> Temp: 0.00, Hum: 1.00
Published -> Temp: 0.00, Hum: 1.00
Received Field2 (Humidity): 1.00 %
Published -> Temp: 0.00, Hum: 1.00
Received Field2 (Humidity): 1.00 %
Published -> Temp: 0.00, Hum: 1.00
Received Field2 (Humidity): 1.00 %
Published -> Temp: 0.00, Hum: 2.00
Received Field1 (Temperature): 0.00 °C
Published -> Temp: 0.00, Hum: 2.00
Received Field1 (Temperature): 0.00 °C
Published -> Temp: 0.00, Hum: 2.00
Received Field1 (Temperature): 0.00 °C
Published -> Temp: 0.00, Hum: 2.00
Received Field1 (Temperature): 0.00 °C
Published -> Temp: 0.00, Hum: 2.00
Received Field2 (Humidity): 2.00 %
Published -> Temp: 0.00, Hum: 2.00
Received Field1 (Temperature): 0.00 °C
Published -> Temp: 0.00, Hum: 2.00
Received Field1 (Temperature): 0.00 °C
Published -> Temp: 0.00, Hum: 2.00
Received Field2 (Humidity): 2.00 %
```

Channel Stats

Created: about 4 hours ago
Last entry: 13 minutes ago
Entries: 156





```
Wi-Fi OK: ('192.168.50.209', '255.255.255.0', '192.168.50.1',
MQTT connected and subscribed to topics.
Published -> Temp: 33.00, Hum: 38.00
Published -> Temp: 33.00, Hum: 38.00
Received Field1 (Temperature): 33.00 °C
Published -> Temp: 33.00, Hum: 39.00
Received Field2 (Humidity): 38.00 %
Published -> Temp: 32.00, Hum: 40.00
Received Field1 (Temperature): 33.00 °C
Published -> Temp: 32.00, Hum: 40.00
Received Field1 (Temperature): 32.00 °C
Published -> Temp: 32.00, Hum: 40.00
Received Field1 (Temperature): 32.00 °C
Published -> Temp: 32.00, Hum: 40.00
Received Field2 (Humidity): 40.00 %
Published -> Temp: 32.00, Hum: 41.00
Received Field1 (Temperature): 32.00 °C
Published -> Temp: 32.00, Hum: 41.00
Received Field1 (Temperature): 32.00 °C
Published -> Temp: 32.00, Hum: 41.00
Received Field1 (Temperature): 32.00 °C
Published -> Temp: 32.00, Hum: 41.00
Received Field1 (Temperature): 32.00 °C
Published -> Temp: 31.00, Hum: 42.00
Received Field1 (Temperature): 32.00 °C
Published -> Temp: 31.00, Hum: 42.00
Received Field2 (Humidity): 42.00 %
Published -> Temp: 31.00, Hum: 42.00
Received Field1 (Temperature): 31.00 °C
Published -> Temp: 31.00, Hum: 42.00
Received Field2 (Humidity): 42.00 %
Published -> Temp: 31.00, Hum: 43.00
Received Field2 (Humidity): 42.00 %
Published -> Temp: 31.00, Hum: 43.00
Received Field1 (Temperature): 31.00 °C
Published -> Temp: 31.00, Hum: 43.00
Received Field1 (Temperature): 31.00 °C
Published -> Temp: 31.00, Hum: 43.00
Received Field1 (Temperature): 31.00 °C
Published -> Temp: 31.00, Hum: 44.00
Received Field1 (Temperature): 31.00 °C
Published -> Temp: 31.00, Hum: 44.00
Received Field1 (Temperature): 31.00 °C
Published -> Temp: 31.00, Hum: 44.00
Received Field2 (Humidity): 44.00 %
Published -> Temp: 31.00, Hum: 44.00
Received Field1 (Temperature): 31.00 °C
Published -> Temp: 30.00, Hum: 45.00
```

Lab4– On ESP32 and Wokwi, complete the following tasks:

1. Build a display system using ESP32 with an I²C LCD to read temperature and humidity data from the DHT11 sensor, display it on the LCD, and simultaneously output the data to the Shell.

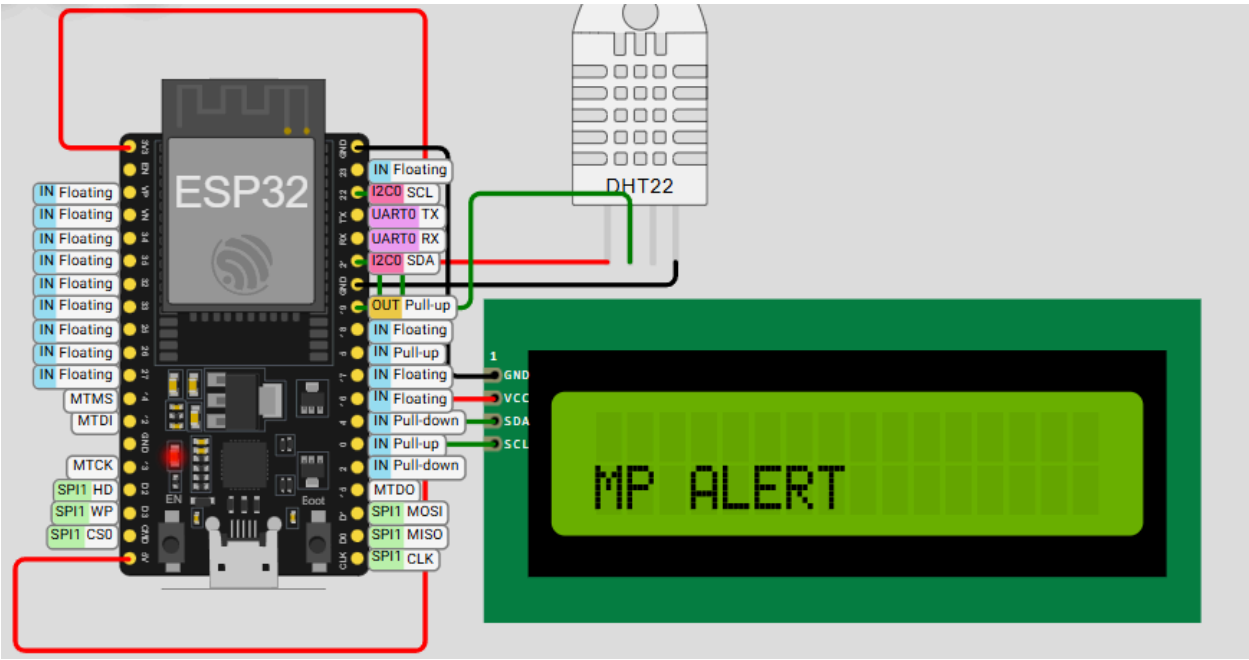
Connect I²C connections: SDA to GPIO 21, SCL to GPIO 22,DHT11 to GPIOxxx

- Function 1: Initialize I²C and the LCD. On the first line, display the fixed string. Every 3 seconds, read the temperature and humidity from the DHT11 and display them dynamically on the second line, while also printing the values to the Shell.
- Function 2: When the temperature > 30°C, the LCD first line switches to display: HIGH TEMP ALERT
- Function 3: When the humidity > 60%, add the note H>60% on the second line.

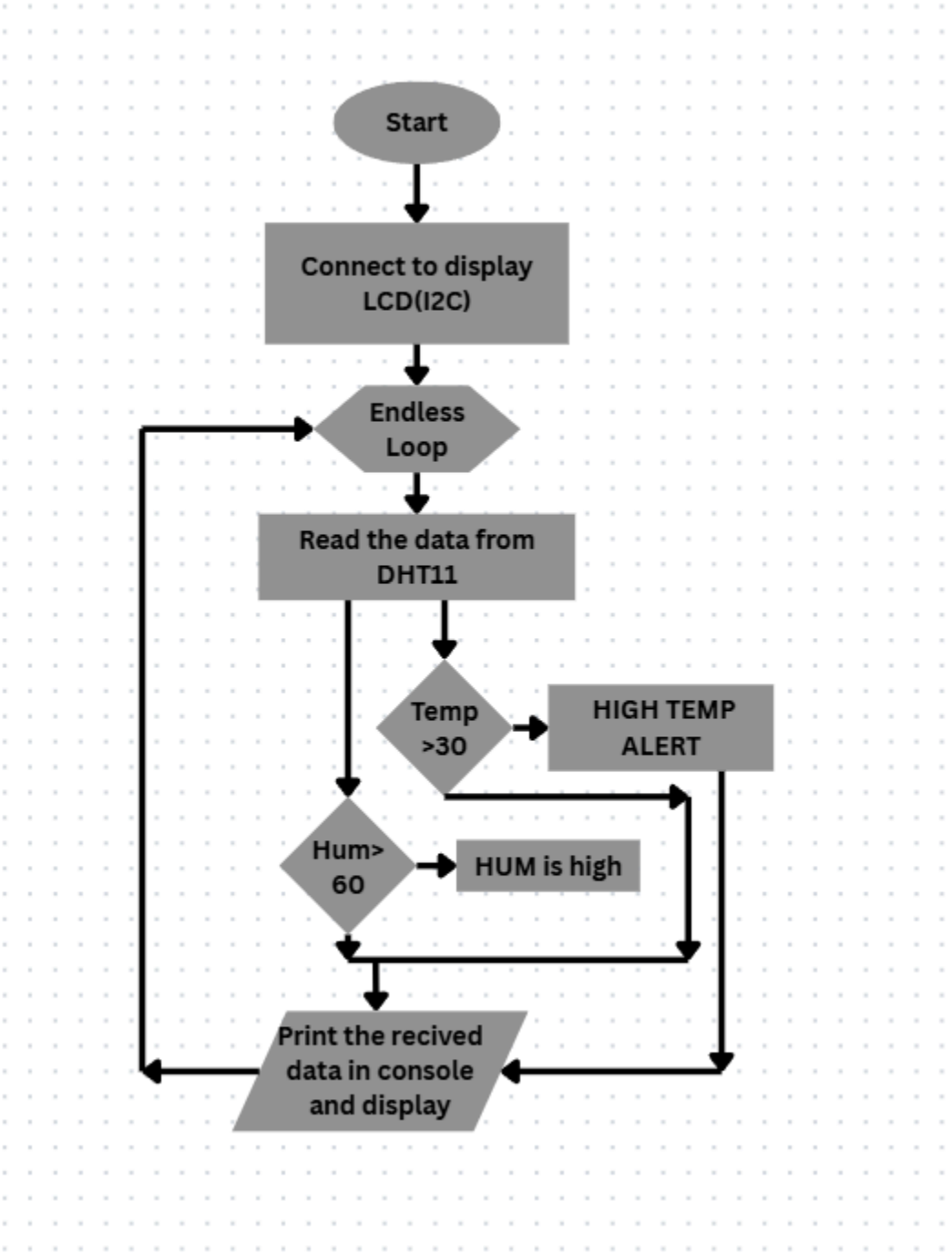
Hint: Use MicroPython's machine.I2C and an LCD API (e.g., i2c_lcd.py / lcd_api.py).

Connections :

- DHT22 DATA → GPIO19
- SDA → GPIO21
- SCL → GPIO22
- LCD VCC → 5V, GND → GND
- DHT VCC → 3.3V, GND → GND



Flowchart :


Start
Connect to display LCD(I2C)
Endless Loop
Read the data from DHT11
Temp >30
HIGH TEMP ALERT
Hum> 60
HUM is high
Print the recived data in console and display

Source code :

```
from machine import Pin, I2C
from dht import DHT11
import time

# ---------- SETTINGS ----------
I2C_SDA = 21
I2C_SCL = 22
I2C_ADDR = 0x27     # Change if your LCD has different address
DHT_PIN = 19        # DHT11 data pin

TEMP_THRESHOLD = 30
HUM_THRESHOLD  = 60

# ---------- LCD CLASSES ----------
# Copy from Wokwi single-file code
class LcdApi:
    def __init__(self, num_lines, num_columns):
        self.num_lines = num_lines
        self.num_columns = num_columns
        self.cursor_x = 0
        self.cursor_y = 0

    def clear(self):
        self.move_to(0,0)
        self.putstr(" " * (self.num_lines * self.num_columns))
        self.move_to(0,0)

    def putstr(self, string):
        for char in string:
            self.putchar(char)

    def putchar(self, char):
        if char == '\n':
            self.cursor_x = 0
            self.cursor_y += 1
            if self.cursor_y >= self.num_lines:
                self.cursor_y = 0
        else:
            self.write_char(char)
```

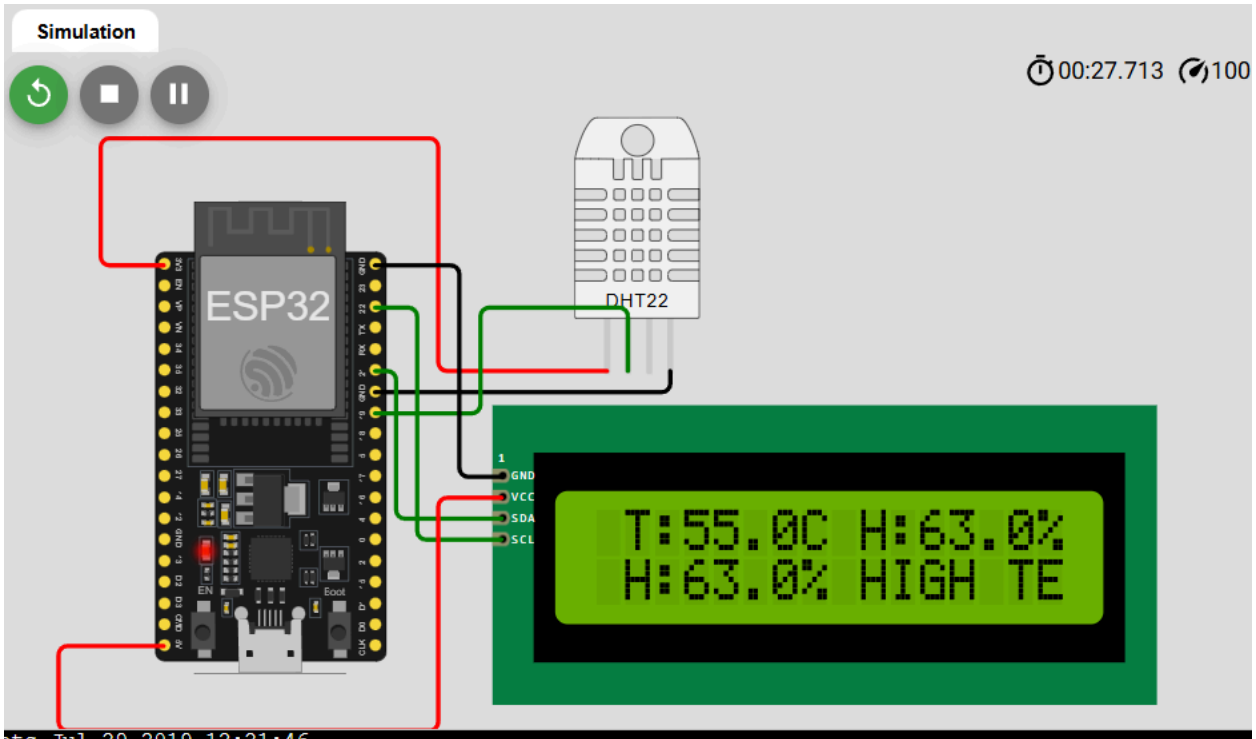
```
            self.cursor_x += 1
            if self.cursor_x >= self.num_columns:
                self.cursor_x = 0
                self.cursor_y += 1
                if self.cursor_y >= self.num_lines:
                    self.cursor_y = 0

    def move_to(self, col, row):
        self.cursor_x = col
        self.cursor_y = row

    def write_char(self, char):
        raise NotImplementedError

MASK_RS = 0x01
MASK_RW = 0x02
MASK_E  = 0x04
MASK_BACKLIGHT = 0x08

class I2cLcd(LcdApi):
    def __init__(self, i2c, i2c_addr, num_lines, num_columns):
        self.i2c = i2c
        self.i2c_addr = i2c_addr
        self.backlight = MASK_BACKLIGHT
        super().__init__(num_lines, num_columns)
        self.init_lcd()

    def init_lcd(self):
        time.sleep_ms(20)
        self.hal_write_init_nibble(0x03)
        time.sleep_ms(5)
        self.hal_write_init_nibble(0x03)
        time.sleep_us(150)
        self.hal_write_init_nibble(0x03)
        self.hal_write_init_nibble(0x02)
        self.hal_write_command(0x28)
        self.hal_write_command(0x0C)
        self.hal_write_command(0x06)
        self.clear()
```

```
    def write_char(self, char):
        self.hal_write_command(ord(char), char_mode=True)

    def hal_write_init_nibble(self, nibble):
        self.i2c.writeto(self.i2c_addr, bytearray([nibble << 4 |
self.backlight | MASK_E]))
        self.i2c.writeto(self.i2c_addr, bytearray([nibble << 4 |
self.backlight]))

    def hal_write_command(self, cmd, char_mode=False):
        high = cmd >> 4
        low = cmd & 0x0F
        rs = MASK_RS if char_mode else 0
        self.i2c.writeto(self.i2c_addr, bytearray([high << 4 |
self.backlight | rs | MASK_E]))
        self.i2c.writeto(self.i2c_addr, bytearray([high << 4 |
self.backlight | rs]))
        self.i2c.writeto(self.i2c_addr, bytearray([low << 4 |
self.backlight | rs | MASK_E]))
        self.i2c.writeto(self.i2c_addr, bytearray([low << 4 |
self.backlight | rs]))
        time.sleep_us(50)

# ---------- SETUP ----------
i2c = I2C(0, sda=Pin(I2C_SDA), scl=Pin(I2C_SCL))
lcd_detected = False

if I2C_ADDR in i2c.scan():
    try:
        lcd = I2cLcd(i2c, I2C_ADDR, 2, 16)
        lcd.putstr("Temp/Humidity Sys")
        lcd_detected = True
        print("LCD detected and initialized.")
    except Exception as e:
        print("LCD detected but failed:", e)
else:
    print("No LCD detected, using terminal only.")

dht_sensor = DHT11(Pin(DHT_PIN))
```

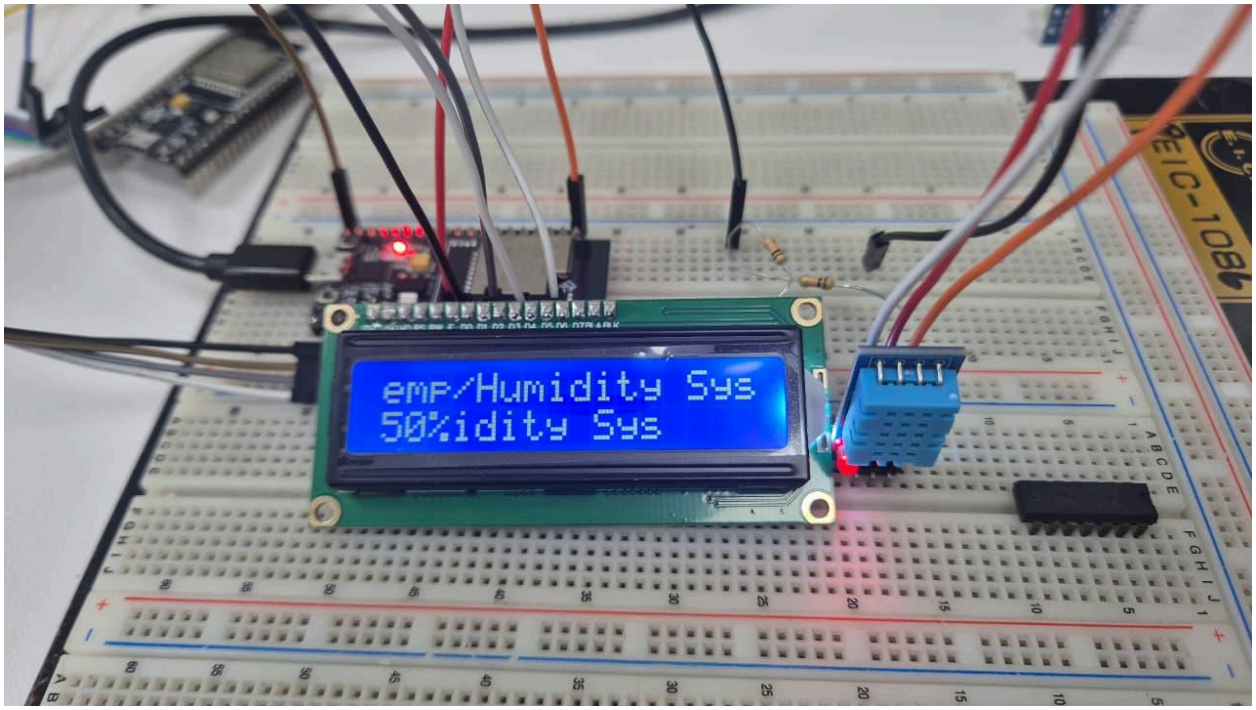
```
# ---------- MAIN LOOP ----------
while True:
    try:
        # Retry logic for DHT11 (sometimes fails)
        for attempt in range(5):
            try:
                dht_sensor.measure()
                temp = dht_sensor.temperature()
                hum  = dht_sensor.humidity()
                break
            except Exception as e:
                time.sleep(1)
                if attempt == 4:
                    raise e

        # Terminal output
        alert_msg = ""
        if temp > TEMP_THRESHOLD:
            alert_msg += " HIGH TEMP ALERT"
        if hum > HUM_THRESHOLD:
            alert_msg += " H>60%"
        print("Temperature: {}C, Humidity: {}%{}".format(temp, hum,
alert_msg))

        # LCD update
        if lcd_detected:
            lcd.move_to(0,0)
            if temp > TEMP_THRESHOLD:
                lcd.putstr("HIGH TEMP ALERT ")
            else:
                lcd.putstr("Temp/Humidity Sys")

            lcd.move_to(0,1)
            lcd.putstr(" " * 16)
            lcd.move_to(0,1)
            line2 = "T:{}C H:{}%".format(temp, hum)
            if hum > HUM_THRESHOLD:
                line2 += " H>60%"
            lcd.putstr(line2[:16])
```

```
    except Exception as e:
        print("DHT read error:", e)

    time.sleep(3)
```

Test records :




```
ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:4728
load:0x40078000,len:14888
load:0x40080400,len:3368
entry 0x400805cc
I2C devices found: [39]
LCD detected and initialized.
Temperature: 24.0C, Humidity: 40.0%
Temperature: 24.0C, Humidity: 40.0%
Temperature: 24.0C, Humidity: 40.0%
Temperature: 55.0C, Humidity: 63.0% HIGH TEMP ALERT H>60%
Temperature: 55.0C, Humidity: 63.0% HIGH TEMP ALERT H>60%
Temperature: 55.0C, Humidity: 63.0% HIGH TEMP ALERT H>60%
Temperature: 55.0C, Humidity: 63.0% HIGH TEMP ALERT H>60%
```


emp/Humidity Sys
50%idity Sys


dity Sys
Humidity Sys

Shell

```
>>> %Run -c $EDITOR_CONTENT

  MPY: soft reboot
  LCD detected and initialized.
  Temperature: 0C, Humidity: 0%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  DHT read error: [Errno 19] ENODEV
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 52%
  Temperature: 26C, Humidity: 52%
  Temperature: 26C, Humidity: 51%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 50%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 49%
  Temperature: 26C, Humidity: 49%
```

Lab5 – On ESP32, complete the following tasks:

Design a monitoring system using an ESP32 together with a TP4056 charging module and a Li-Ion battery protection board. The system should be able to display the battery charging status in real time and provide monitoring and protection for battery capacity and charging current.Battery monitoring and charging circuit design to ensure safety and stability.

1. Hardware Requirements

- ESP32 development board
- TP4056 Li-Ion battery charging module (with protection board)
- Li-Ion battery (single cell, 3.7V)
- LCD display module (I^2C interface)
- LED indicators (4 LEDs, showing battery level status)

2. Functional Requirements

Single-cell Battery Charging Monitoring

- During charging, measure the voltage and current across the battery.

Real-time LCD Display

- The LCD should display:
  - Battery level (percentage)
  - Voltage
  - Current
  - Charging status (indicate Full charge when fully charged)

LED Battery Level Indication

- Use the number of lit LEDs to represent the battery level, for example:
  - 1 LED on = 25%
  - 2 LEDs on = 50%
  - 3 LEDs on = 75%
  - 4 LEDs on = 100%

Current Monitoring and Protection

· Continuously monitor the charging current to prevent it from becoming too high.

· If the current exceeds a safe range, a warning must be displayed.

Connections :

ESP32:

- GND -> TP4056 GND (common)
- GPIO21 -> SDA (LCD, INA219)
- GPIO22 -> SCL (LCD, INA219)

LCD:

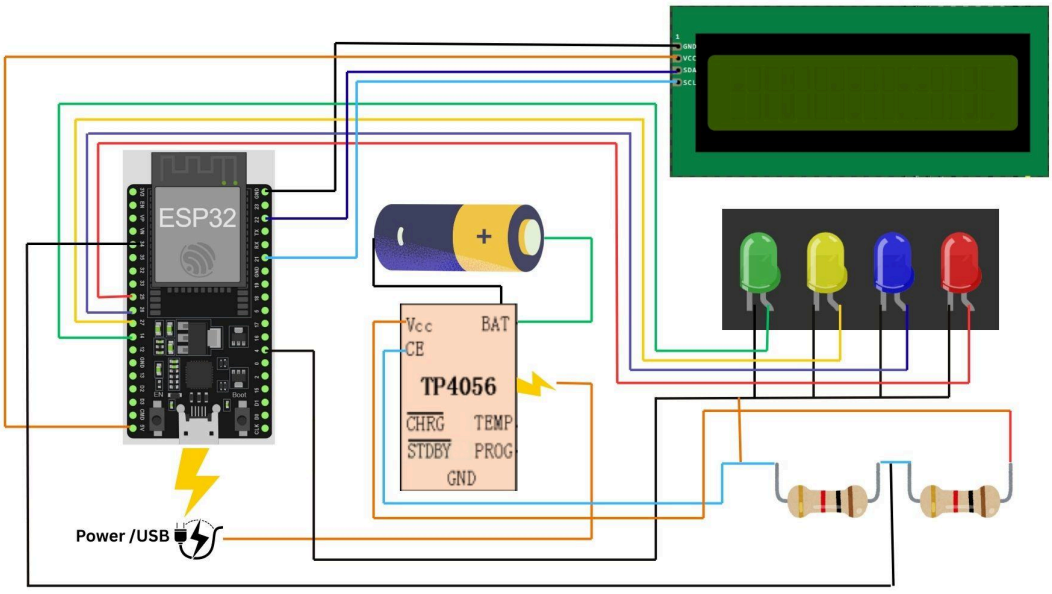
- SDA <- GPIO21
- SCL <- GPIO22
- VCC <- 5V (or 3.3V if module requires)
- GND <- common GND

LEDs:

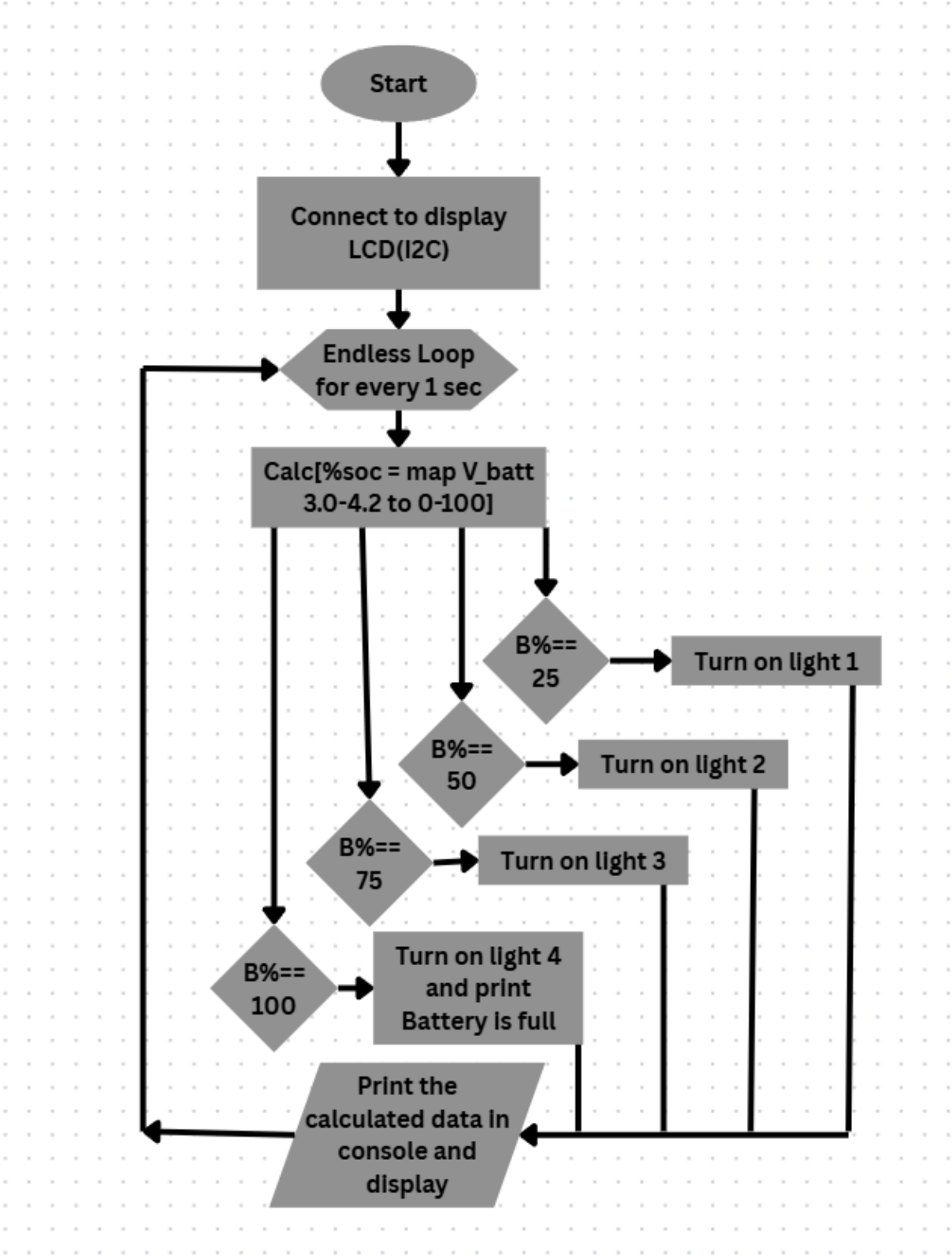
- GPIO14 -> LED1
- GPIO27 -> LED2
- GPIO26 -> LED3
- GPIO25 -> LED4

LCD (I²C) and LEDs

- LCD SDA → GPIO21, SCL → GPIO22
- LCD VCC → 5V (or 3.3V depending on module), LCD GND → GND



Flowchart :


Start
Connect to display LCD(I2C)
Endless Loop for every 1 sec
Calc[%soc = map V_batt 3.0-4.2 to 0-100]
B%== 25
Turn on light 1
B%== 50
Turn on light 2
B%== 75
Turn on light 3
B%== 100
Turn on light 4 and print Battery is full
Print the calculated data in console and display

Source Code :

Source Code :

```
from machine import Pin, I2C, ADC
from i2c_lcd import I2cLcd
import time

# -------------------- Hardware Setup --------------------
# I2C LCD
i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=400000)
lcd = I2cLcd(i2c, 0x27, 2, 16)

# ADC pin for battery voltage
adc = ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)  # full-scale ~3.3V

# LED pins for battery levels
led25 = Pin(14, Pin.OUT)
led50 = Pin(27, Pin.OUT)
led75 = Pin(26, Pin.OUT)
led100 = Pin(25, Pin.OUT)

# -------------------- Battery Calibration --------------------
VREF = 3.3       # ESP32 reference voltage
MAX_ADC = 4095   # 12-bit ADC
# Voltage divider resistors (adjust based on your hardware)
R1 = 100000      # top resistor to battery +
R2 = 10000       # bottom resistor to GND

# Number of ADC samples to average
NUM_SAMPLES = 10

# -------------------- Functions --------------------
def read_battery_voltage():
    """Read battery voltage via ADC and scale through voltage divider."""
    raw = sum(adc.read() for _ in range(NUM_SAMPLES)) / NUM_SAMPLES
    v_adc = raw / MAX_ADC * VREF
    # Calculate battery voltage using voltage divider
    v_bat = v_adc * ((R1 + R2) / R2)
    # Clamp voltage to max 4.2V
    v_bat = max(0, min(v_bat, 4.2))
```

```
    return round(v_bat, 2)

def battery_percent(voltage):
    """Convert voltage to approximate Li-Ion battery percentage."""
    if voltage >= 4.2:
        return 100
    elif voltage <= 3.0:
        return 0
    else:
        return int(((voltage - 3.0) / (4.2 - 3.0)) * 100)

def update_leds(percent):
    """Turn on LEDs based on battery percentage."""
    led25.value(1 if percent >= 25 else 0)
    led50.value(1 if percent >= 50 else 0)
    led75.value(1 if percent >= 75 else 0)
    led100.value(1 if percent >= 100 else 0)

# -------------------- Main Loop --------------------
lcd.clear()
lcd.putstr("Battery Monitor")
time.sleep(2)

while True:
    # Read voltage and calculate percentage
    vbat = read_battery_voltage()
    percent = battery_percent(vbat)

    # Charging / full status (simulate current monitoring)
    if percent >= 99:
        status = "Full"
        current = 0.0
    else:
        status = "Charging"
        current = 0.45  # Simulated current in Amps

    # Update LCD display
    lcd.clear()
    lcd.move_to(0, 0)
    lcd.putstr("V:{:.2f} P:{}%".format(vbat, percent))
```

```
    lcd.move_to(0, 1)
    lcd.putstr("I:{:.2f}A {}".format(current, status))

    # Update LED bar
    update_leds(percent)

    # Print debug info to console
    print("Voltage: {:.2f} V | Percent: {}% | Current: {:.2f} A | Status:
{}".format(
        vbat, percent, current, status))

    time.sleep(2)
```

Save this file as `i2c_lcd.py` in ESP32

```
# i2c_lcd.py
from lcd_api import LcdApi
from machine import I2C
from time import sleep_ms

# PCF8574 I2C backpack bit mapping
MASK_RS = 0x01
MASK_RW = 0x02
MASK_E = 0x04
SHIFT_BACKLIGHT = 3
SHIFT_DATA = 4

class I2cLcd(LcdApi):
    def __init__(self, i2c, i2c_addr, num_lines, num_columns):
        self.i2c = i2c
        self.i2c_addr = i2c_addr
        self.backlight = 1
        self.mcp_output(0)
        sleep_ms(20)

        # Initialize LCD
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
```

```
        sleep_ms(5)
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        sleep_ms(1)
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        sleep_ms(1)
        self.hal_write_init_nibble(self.LCD_FUNCTION)
        sleep_ms(1)

        LcdApi.__init__(self, num_lines, num_columns)

        cmd = self.LCD_FUNCTION
        if num_lines > 1:
            cmd |= self.LCD_FUNCTION_2LINES
        self.write_cmd(cmd)
        self.display_on()
        self.clear()
        self.write_cmd(self.LCD_ENTRY_MODE | self.LCD_ENTRY_INC)

    def mcp_output(self, data):
        self.i2c.writeto(self.i2c_addr, bytes([data]))

    def hal_write_init_nibble(self, nibble):
        data = ((nibble >> 4) & 0x0F) << SHIFT_DATA
        self.pulse_enable(data)

    def pulse_enable(self, data):
        self.mcp_output(data | MASK_E | (self.backlight <<
SHIFT_BACKLIGHT))
        sleep_ms(1)
        self.mcp_output(data & ~MASK_E | (self.backlight <<
SHIFT_BACKLIGHT))
        sleep_ms(1)

    def hal_backlight_on(self):
        self.backlight = 1
        self.mcp_output(self.backlight << SHIFT_BACKLIGHT)

    def hal_backlight_off(self):
        self.backlight = 0
        self.mcp_output(0)
```

```
    def write_cmd(self, cmd):
        self.hal_write(cmd, 0)

    def write_data(self, data):
        self.hal_write(data, MASK_RS)

    def hal_write(self, data, mode):
        high = (data & 0xF0) | mode
        low = ((data << 4) & 0xF0) | mode
        self.pulse_enable(high)
        self.pulse_enable(low)
```

Save this as `lcd_api.py` in ESP32

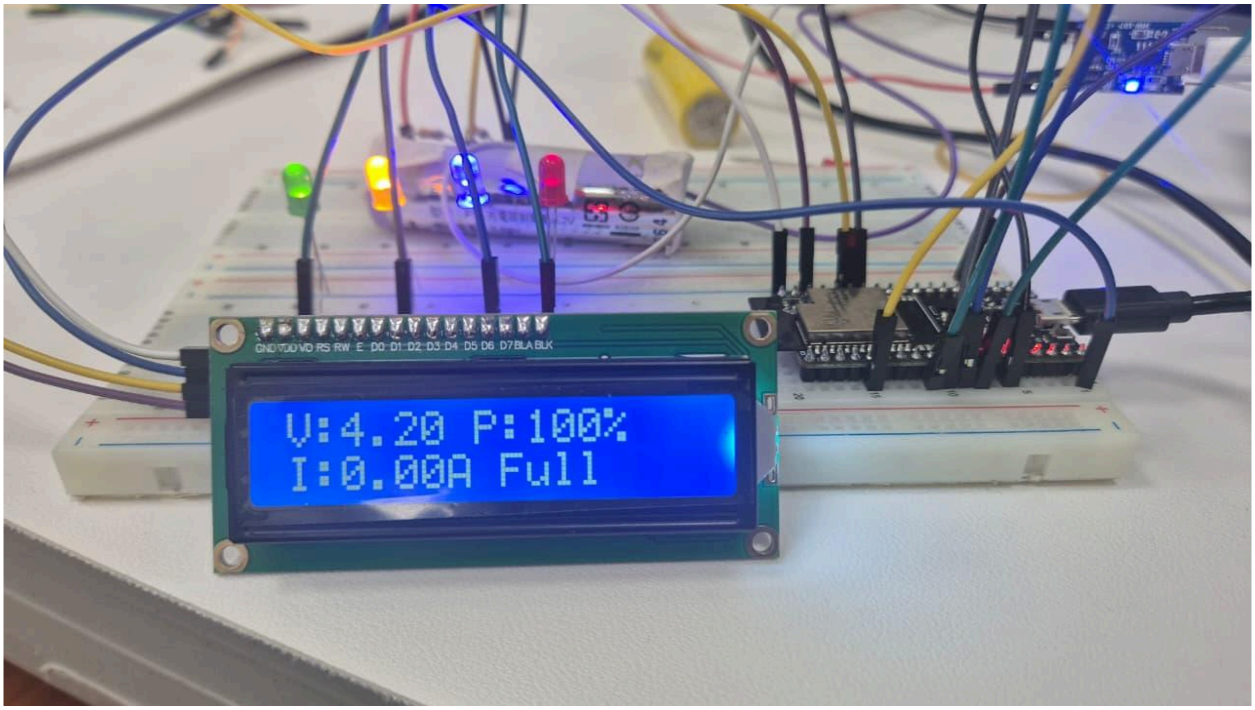
```
# lcd_api.py

class LcdApi:
    # LCD commands
    LCD_CLR = 0x01
    LCD_HOME = 0x02

    LCD_ENTRY_MODE = 0x04
    LCD_ENTRY_INC = 0x02
    LCD_ENTRY_SHIFT = 0x01

    LCD_ON_CTRL = 0x08
    LCD_ON_DISPLAY = 0x04
    LCD_ON_CURSOR = 0x02
    LCD_ON_BLINK = 0x01

    LCD_MOVE = 0x10
    LCD_MOVE_DISP = 0x08
    LCD_MOVE_RIGHT = 0x04

    LCD_FUNCTION = 0x20
    LCD_FUNCTION_8BIT = 0x10
```

```
    LCD_FUNCTION_2LINES = 0x08
    LCD_FUNCTION_10DOTS = 0x04
    LCD_FUNCTION_RESET = 0x30

    LCD_CGRAM = 0x40
    LCD_DDRAM = 0x80

    LCD_RS_CMD = 0
    LCD_RS_DATA = 1
    LCD_RW_WRITE = 0
    LCD_RW_READ = 1

    def __init__(self, num_lines, num_columns):
        self.num_lines = min(num_lines, 4)
        self.num_columns = min(num_columns, 40)
        self.cursor_x = 0
        self.cursor_y = 0

    def clear(self):
        self.write_cmd(self.LCD_CLR)
        self.cursor_x = 0
        self.cursor_y = 0

    def home(self):
        self.write_cmd(self.LCD_HOME)
        self.cursor_x = 0
        self.cursor_y = 0

    def show_cursor(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
self.LCD_ON_CURSOR)

    def hide_cursor(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

    def blink_cursor_on(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                       self.LCD_ON_CURSOR | self.LCD_ON_BLINK)

    def blink_cursor_off(self):
```

```
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
self.LCD_ON_CURSOR)

    def display_on(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

    def display_off(self):
        self.write_cmd(self.LCD_ON_CTRL)

    def move_to(self, cursor_x, cursor_y):
        self.cursor_x = cursor_x
        self.cursor_y = cursor_y
        addr = cursor_x & 0x3F
        if cursor_y & 1:
            addr += 0x40
        if cursor_y & 2:
            addr += 0x14
        self.write_cmd(self.LCD_DDRAM | addr)

    def putchar(self, char):
        if char != '\n':
            self.write_data(ord(char))
            self.cursor_x += 1
            if self.cursor_x >= self.num_columns:
                self.cursor_x = 0
                self.cursor_y += 1
                if self.cursor_y >= self.num_lines:
                    self.cursor_y = 0
                self.move_to(self.cursor_x, self.cursor_y)

    def putstr(self, string):
        for char in string:
            self.putchar(char)

    def custom_char(self, location, charmap):
        location &= 0x7
        self.write_cmd(self.LCD_CGRAM | (location << 3))
        for i in range(8):
            self.write_data(charmap[i])
```

Test Records :

The four lights blinks because the battery get fully charged




```
MPY: soft reboot
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
Voltage: 4.20 V | Percent: 100% | Current: 0.00 A | Status: Full
```

Lab6 – On ESP32, complete the following tasks:

Design a monitoring system using ESP32 together with a TP4056 charging module and a Li-Ion battery protection board, with three 3.7V Li-Ion cells connected in series to form a 12V battery pack.

The system should be able to display the battery charging status in real time and provide monitoring and protection for battery capacity and charging current.

The battery monitoring and charging circuit should be designed to ensure safety and stability.

1. Hardware Requirements

- ESP32 development board
- TP4056 Li-Ion battery charging module
- Li-Ion batteries (3 single cells, 3.7V each, connected in series to form a 12V pack)
- LCD display module (I^2C interface)
- LED indicators (4 LEDs, showing battery level status)

2. Functional Requirements

Multi-cell Battery Charging Monitoring

- During charging, measure the total pack voltage (about 12.6V when fully charged) and the charging current.
- The system must be able to determine the individual voltage and charging status of each of the three cells, to avoid imbalance.

Real-time LCD Display

- The LCD should display:
- Battery pack level (percentage)
- Individual voltages of the three cells
- Total battery pack voltage
- Current
- Charging status (indicate Full charge when fully charged)

LED Battery Level Indication

· Use the number of lit LEDs to represent the battery level, for example:

- 1 LED on = 25%
- 2 LEDs on = 50%
- 3 LEDs on = 75%
- 4 LEDs on = 100%

Current Monitoring and Protection

· Continuously monitor the charging current to prevent it from becoming too high.

· If the current exceeds a safe range, a warning must be displayed.

Connections :

ESP32:

- GND -> TP4056 GND (common)
- GPIO21 -> SDA (LCD, INA219)
- GPIO22 -> SCL (LCD, INA219)

LCD:

- SDA <- GPIO21
- SCL <- GPIO22
- VCC <- 5V (or 3.3V if module requires)
- GND <- common GND

LEDs:

- GPIO14 -> LED1
- GPIO27 -> LED2
- GPIO26 -> LED3
- GPIO25 -> LED4

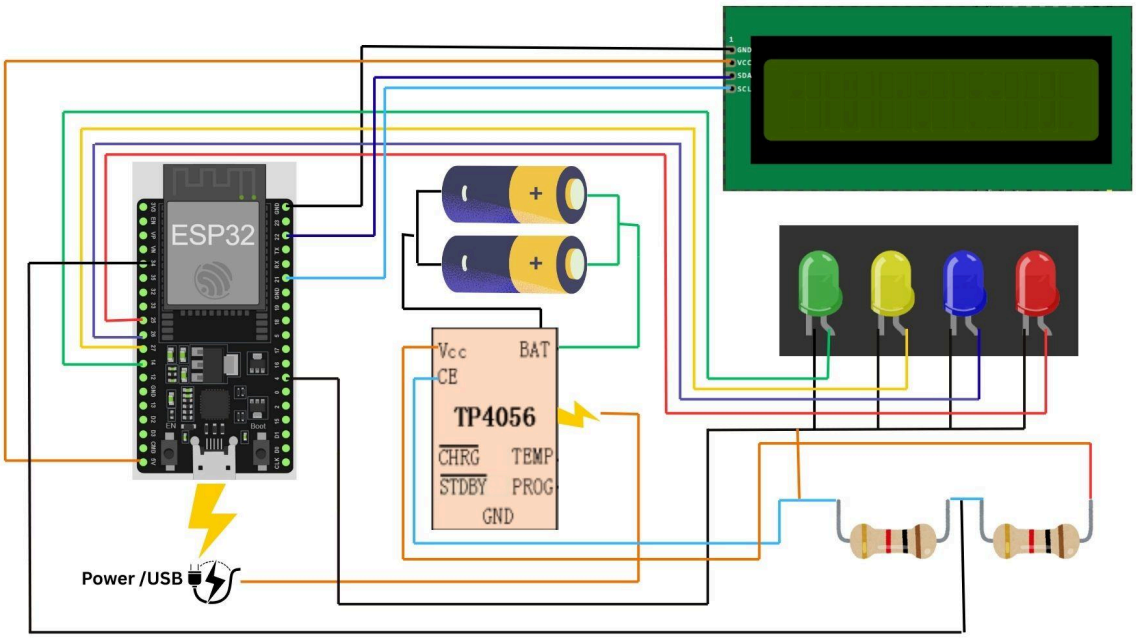
LCD (I²C) and LEDs

- LCD SDA → GPIO21, SCL → GPIO22
- LCD VCC → 5V (or 3.3V depending on module), LCD GND → GND

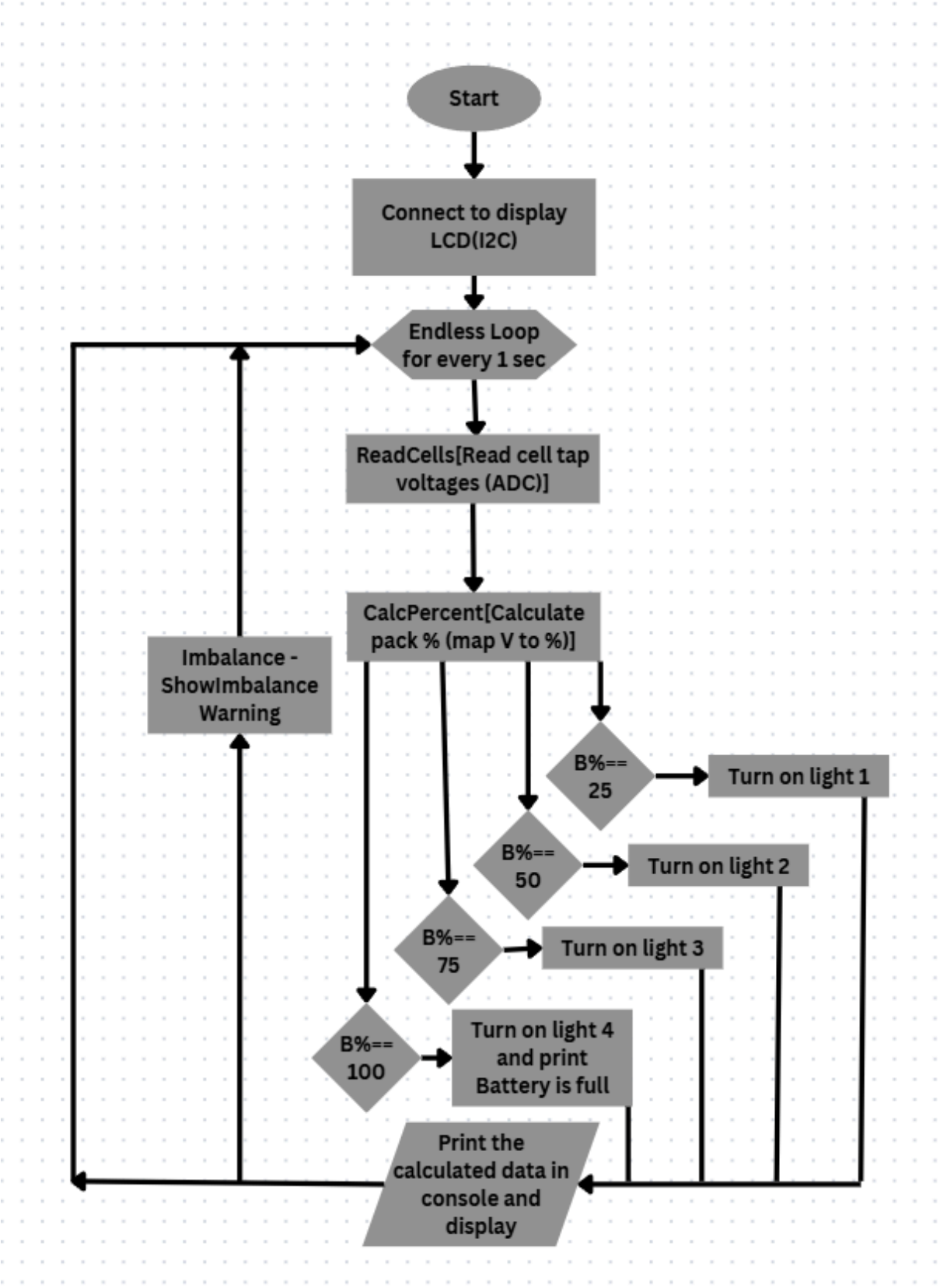
Note : The TP4056 is designed for one single 3.7 V Li-ion cell only.

- If we connect many cells in parallel (all positive terminals tied together, all negative terminals tied together), then the TP4056 *can* charge them as if they were one big cell — but only if all the cells are at the same voltage level before connecting. Otherwise, a higher-charged cell will dump current into a lower-charged cell, which can cause overheating or damage. This method is risky unless the pack is properly matched.

- If you connect cells in series (e.g., 2S = 7.4 V, 3S = 11.1 V, 4S = 14.8 V, etc.), then one TP4056 cannot be used.
    - The TP4056 expects to charge only up to 4.2 V.
    - If you give it a 7.4 V or 12 V pack, it cannot regulate charging correctly and will either fail or be destroyed.



Flowchart :


Start
Connect to display LCD(I2C)
Endless Loop for every 1 sec
ReadCells[Read cell tap voltages (ADC)]
CalcPercent[Calculate pack % (map V to %)]
Imbalance - ShowImbalance Warning
B%== 25
Turn on light 1
B%== 50
Turn on light 2
B%== 75
Turn on light 3
B%== 100
Turn on light 4 and print Battery is full
Print the calculated data in console and display

Source Code :

```
from machine import Pin, ADC, I2C
from time import sleep
from lcd_api import LcdApi
from i2c_lcd import I2cLcd

# -------------------- Hardware Setup --------------------
# I2C LCD 16x2
i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=400000)
lcd = I2cLcd(i2c, 0x27, 2, 16)

# Battery sensing pins (use voltage dividers!)
adc_batt1 = ADC(Pin(34))  # Battery 1
adc_batt2 = ADC(Pin(35))  # Battery 2
adc_batt1.atten(ADC.ATTN_11DB)  # Full range: 0-3.3V
adc_batt2.atten(ADC.ATTN_11DB)

# Charging status pin (TP4056 "STDBY/CHRG")
charge_pin = Pin(32, Pin.IN)

# LEDs
led_charge = Pin(25, Pin.OUT)
led_full = Pin(26, Pin.OUT)

# -------------------- Helper Functions --------------------
def read_voltage(adc, divider=2):
    """Read voltage from ADC and scale with voltage divider."""
    raw = adc.read()
    voltage = (raw / 4095) * 3.3 * divider
    return round(voltage, 2)

def battery_percentage(v):
    """Convert Li-ion voltage to % (3.0V = 0%, 4.2V = 100%)."""
    if v <= 3.0:
        return 0
    elif v >= 4.2:
        return 100
    else:
```

```
        return int(((v - 3.0) / (4.2 - 3.0)) * 100)

# -------------------- Main Loop --------------------
while True:
    # Read voltages
    v1 = read_voltage(adc_batt1)
    v2 = read_voltage(adc_batt2)
    total_v = round(v1 + v2, 2)

    # Calculate percentages
    percent1 = battery_percentage(v1)
    percent2 = battery_percentage(v2)
    total_percent = int((percent1 + percent2) / 2)

    # Charging status (TP4056: LOW = charging)
    if not charge_pin.value():
        status = "Charging"
        led_charge.value(1)
        led_full.value(0)
    elif total_percent >= 99:
        status = "Full"
        led_full.value(1)
        led_charge.value(0)
    else:
        status = "Discharging"
        led_charge.value(0)
        led_full.value(0)

    # -------------------- Update LCD --------------------
    # Page 1: individual batteries
    lcd.clear()
    lcd.putstr("B1:{:.2f}V {}%".format(v1, percent1))
    lcd.move_to(0, 1)
    lcd.putstr("B2:{:.2f}V {}%".format(v2, percent2))
    sleep(2)

    # Page 2: total voltage + status
    lcd.clear()
    lcd.putstr("Tot:{:.2f}V".format(total_v))
    lcd.move_to(0, 1)
```

```
    lcd.putstr("{} {}%".format(status, total_percent))
    sleep(2)

    # -------------------- Debug Console --------------------
    print("B1: {:.2f}V {}% | B2: {:.2f}V {}% | Total: {:.2f}V | Status:
{}"
          .format(v1, percent1, v2, percent2, total_v, status))
```

Save this file as `i2c_lcd.py` in ESP32

```
# i2c_lcd.py
from lcd_api import LcdApi
from machine import I2C
from time import sleep_ms

# PCF8574 I2C backpack bit mapping
MASK_RS = 0x01
MASK_RW = 0x02
MASK_E = 0x04
SHIFT_BACKLIGHT = 3
SHIFT_DATA = 4

class I2cLcd(LcdApi):
    def __init__(self, i2c, i2c_addr, num_lines, num_columns):
        self.i2c = i2c
        self.i2c_addr = i2c_addr
        self.backlight = 1
        self.mcp_output(0)
        sleep_ms(20)

        # Initialize LCD
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        sleep_ms(5)
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        sleep_ms(1)
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        sleep_ms(1)
```

```
        self.hal_write_init_nibble(self.LCD_FUNCTION)
        sleep_ms(1)

        LcdApi.__init__(self, num_lines, num_columns)

        cmd = self.LCD_FUNCTION
        if num_lines > 1:
            cmd |= self.LCD_FUNCTION_2LINES
        self.write_cmd(cmd)
        self.display_on()
        self.clear()
        self.write_cmd(self.LCD_ENTRY_MODE | self.LCD_ENTRY_INC)

    def mcp_output(self, data):
        self.i2c.writeto(self.i2c_addr, bytes([data]))

    def hal_write_init_nibble(self, nibble):
        data = ((nibble >> 4) & 0x0F) << SHIFT_DATA
        self.pulse_enable(data)

    def pulse_enable(self, data):
        self.mcp_output(data | MASK_E | (self.backlight <<
SHIFT_BACKLIGHT))
        sleep_ms(1)
        self.mcp_output(data & ~MASK_E | (self.backlight <<
SHIFT_BACKLIGHT))
        sleep_ms(1)

    def hal_backlight_on(self):
        self.backlight = 1
        self.mcp_output(self.backlight << SHIFT_BACKLIGHT)

    def hal_backlight_off(self):
        self.backlight = 0
        self.mcp_output(0)

    def write_cmd(self, cmd):
        self.hal_write(cmd, 0)

    def write_data(self, data):
```

```
        self.hal_write(data, MASK_RS)

    def hal_write(self, data, mode):
        high = (data & 0xF0) | mode
        low = ((data << 4) & 0xF0) | mode
        self.pulse_enable(high)
        self.pulse_enable(low)
```

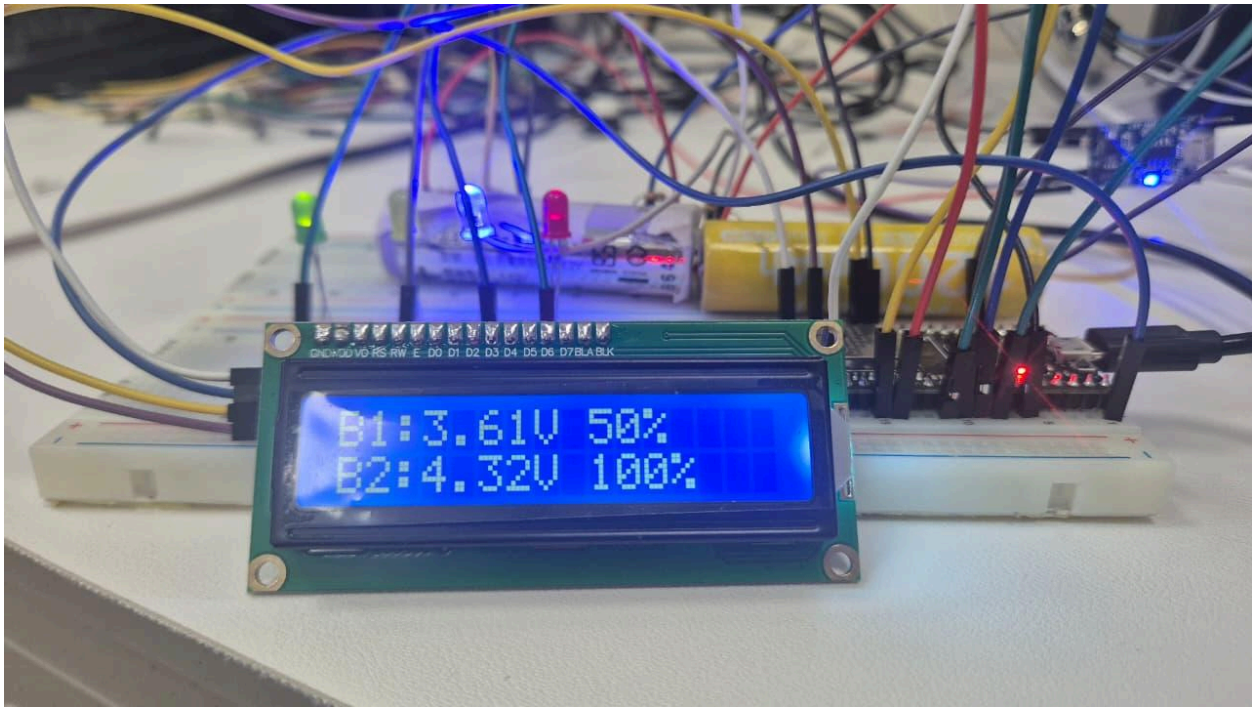
Save this as `lcd_api.py` in ESP32

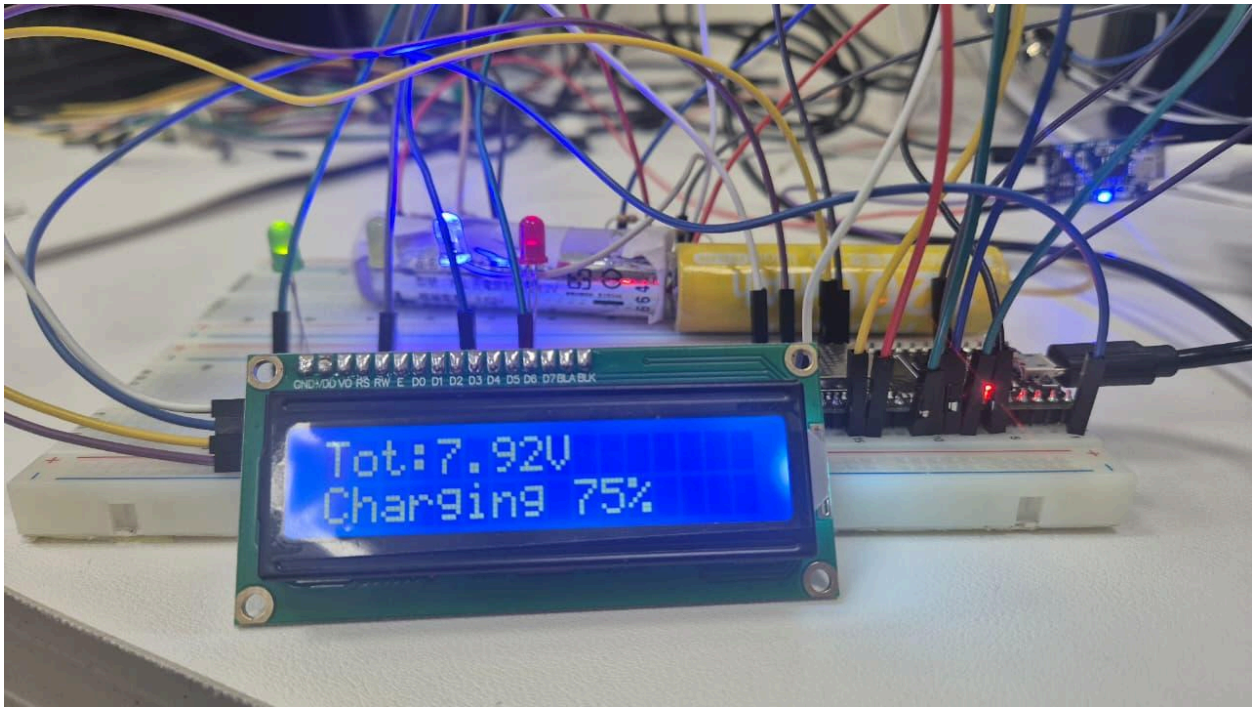
```
# lcd_api.py

class LcdApi:
    # LCD commands
    LCD_CLR = 0x01
    LCD_HOME = 0x02

    LCD_ENTRY_MODE = 0x04
    LCD_ENTRY_INC = 0x02
    LCD_ENTRY_SHIFT = 0x01

    LCD_ON_CTRL = 0x08
    LCD_ON_DISPLAY = 0x04
    LCD_ON_CURSOR = 0x02
    LCD_ON_BLINK = 0x01

    LCD_MOVE = 0x10
    LCD_MOVE_DISP = 0x08
    LCD_MOVE_RIGHT = 0x04

    LCD_FUNCTION = 0x20
    LCD_FUNCTION_8BIT = 0x10
    LCD_FUNCTION_2LINES = 0x08
    LCD_FUNCTION_10DOTS = 0x04
    LCD_FUNCTION_RESET = 0x30
```

```
    LCD_CGRAM = 0x40
    LCD_DDRAM = 0x80

    LCD_RS_CMD = 0
    LCD_RS_DATA = 1
    LCD_RW_WRITE = 0
    LCD_RW_READ = 1

    def __init__(self, num_lines, num_columns):
        self.num_lines = min(num_lines, 4)
        self.num_columns = min(num_columns, 40)
        self.cursor_x = 0
        self.cursor_y = 0

    def clear(self):
        self.write_cmd(self.LCD_CLR)
        self.cursor_x = 0
        self.cursor_y = 0

    def home(self):
        self.write_cmd(self.LCD_HOME)
        self.cursor_x = 0
        self.cursor_y = 0

    def show_cursor(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
self.LCD_ON_CURSOR)

    def hide_cursor(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

    def blink_cursor_on(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                       self.LCD_ON_CURSOR | self.LCD_ON_BLINK)

    def blink_cursor_off(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
self.LCD_ON_CURSOR)
```

```
    def display_on(self):
        self.write_cmd(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

    def display_off(self):
        self.write_cmd(self.LCD_ON_CTRL)

    def move_to(self, cursor_x, cursor_y):
        self.cursor_x = cursor_x
        self.cursor_y = cursor_y
        addr = cursor_x & 0x3F
        if cursor_y & 1:
            addr += 0x40
        if cursor_y & 2:
            addr += 0x14
        self.write_cmd(self.LCD_DDRAM | addr)

    def putchar(self, char):
        if char != '\n':
            self.write_data(ord(char))
            self.cursor_x += 1
            if self.cursor_x >= self.num_columns:
                self.cursor_x = 0
                self.cursor_y += 1
                if self.cursor_y >= self.num_lines:
                    self.cursor_y = 0
                self.move_to(self.cursor_x, self.cursor_y)

    def putstr(self, string):
        for char in string:
            self.putchar(char)

    def custom_char(self, location, charmap):
        location &= 0x7
        self.write_cmd(self.LCD_CGRAM | (location << 3))
        for i in range(8):
            self.write_data(charmap[i])
```

Test Records :





Shell

```
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
B1: 3.60V 50% | B2: 4.32V 100% | Total: 7.92V | Status: Charging
B1: 3.60V 50% | B2: 4.33V 100% | Total: 7.93V | Status: Charging
B1: 3.64V 53% | B2: 4.34V 100% | Total: 7.98V | Status: Charging
B1: 3.61V 50% | B2: 4.31V 100% | Total: 7.92V | Status: Charging
B1: 3.61V 50% | B2: 4.31V 100% | Total: 7.92V | Status: Charging
B1: 3.61V 50% | B2: 4.30V 100% | Total: 7.91V | Status: Charging
B1: 3.60V 50% | B2: 4.33V 100% | Total: 7.93V | Status: Charging
B1: 3.61V 50% | B2: 4.33V 100% | Total: 7.94V | Status: Charging
B1: 3.62V 51% | B2: 4.33V 100% | Total: 7.95V | Status: Charging
B1: 3.61V 50% | B2: 4.32V 100% | Total: 7.93V | Status: Charging
B1: 3.61V 50% | B2: 4.31V 100% | Total: 7.92V | Status: Charging
B1: 3.61V 50% | B2: 4.32V 100% | Total: 7.93V | Status: Charging
B1: 3.61V 50% | B2: 4.33V 100% | Total: 7.94V | Status: Charging
B1: 3.61V 50% | B2: 4.32V 100% | Total: 7.93V | Status: Charging
B1: 3.62V 51% | B2: 4.32V 100% | Total: 7.94V | Status: Charging
B1: 3.59V 49% | B2: 4.32V 100% | Total: 7.91V | Status: Charging
B1: 3.62V 51% | B2: 4.31V 100% | Total: 7.93V | Status: Charging
B1: 3.61V 50% | B2: 4.32V 100% | Total: 7.93V | Status: Charging
B1: 3.64V 53% | B2: 4.32V 100% | Total: 7.96V | Status: Charging
```

Lab7 learning reflection or comments

Initially, working with the ESP32, various sensors like DHT11/DHT22, TP4056 charging modules, and I²C LCDs seemed overwhelming because of the multiple hardware and software interfaces involved. Configuring the devices, ensuring proper wiring, and writing MicroPython code to communicate with the peripherals required a lot of attention to detail. At times, the problems, such as the LCD not displaying readings or MQTT errors when connecting to ThingSpeak, tested my patience and problem-solving skills.

However, overcoming these challenges was extremely satisfying. I learned how crucial correct wiring, voltage scaling, and library usage are in embedded systems. For example, understanding voltage dividers and ADC readings allowed me to safely monitor battery voltages without damaging the ESP32. Writing code for reading sensors, calculating averages, and controlling LEDs based on thresholds strengthened my programming skills and reinforced the importance of careful debugging.

One memorable moment was when I successfully implemented Lab 5's battery monitoring system. Seeing the voltage, current, and battery percentage accurately displayed on the LCD for the first time gave me a real sense of accomplishment. It also made me appreciate the integration of hardware and software in a real-time system.

Overall, the course was well-balanced. It was challenging enough to push me out of my comfort zone, but not so difficult that it became discouraging. I feel that my understanding of IoT devices, MicroPython programming, and embedded system debugging has improved significantly. I also learned to systematically test hardware and software step by step, which is a critical skill in real-world projects.

Lab8 Video Link

Record a video of approximately 3 minutes to explain your research process. The video should include the following:

1. A description of the tools or methods used during your research process.
2. A presentation of the results and outcomes from LAB1 to LAB6**.**

Link : https://youtu.be/XBz4RWiRRdY

Thank you