

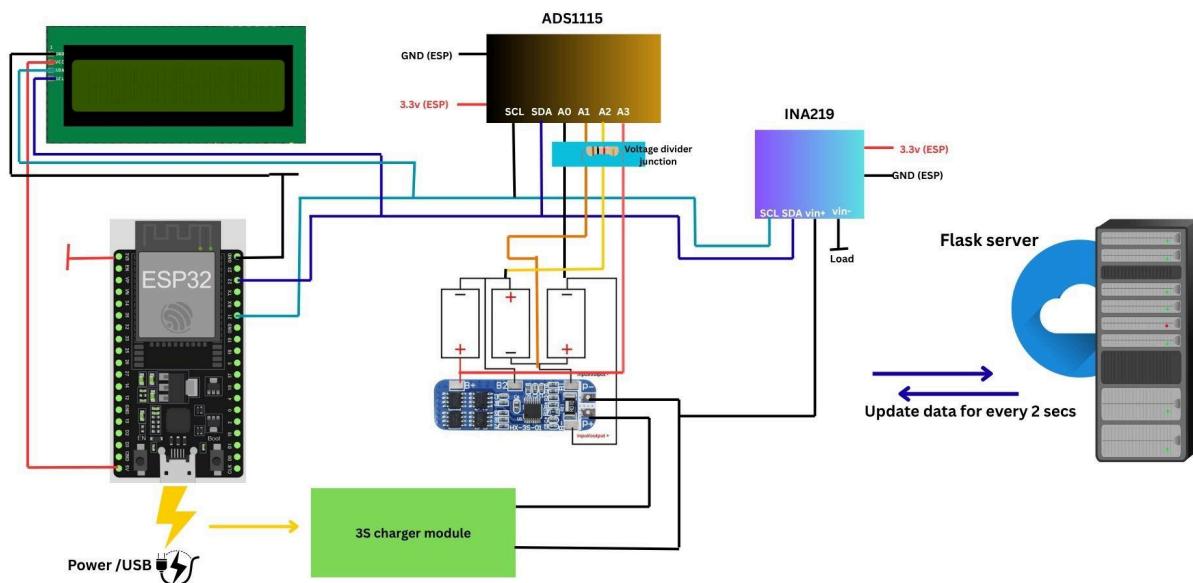
2025 TEEP Progress Report Week-12

Used Wokwi to complete the tasks for Labs 1 to 4

(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 – On ESP32, complete the following tasks:

This project focuses on optimizing the single-cell battery charging monitoring system completed last week, with the goal of improving the overall stability of measurement and display. Each step of the experimental process must be thoroughly documented, compiled into the project report, and also presented in video format to ensure completeness and traceability of the research process.



Experimental Record

Date: 17 / OCT / 2025

Course/Project Title: Development of IoT Situation Room for Elevator Operation Monitoring and Health Diagnosis

1. Experiment Title

Real-Time Multi-Cell (3S) Li-Ion Battery Monitoring System using ESP32, INA219, ADS1115, LCD, and Flask Web Dashboard

2. Objective

The objective of this experiment is to build a real-time battery monitoring system that:

- Reads battery voltages and current from a 3S Li-Ion pack using ESP32 with MicroPython.
- Calculates approximate battery percentage and smooths readings using an Exponential Moving Average (EMA) for stable visualization.
- Sends live data to a Flask API over Wi-Fi.
- Displays the pack voltage and battery state-of-charge (SoC) on a 16x2 LCD.
- Visualizes live data and trends on a web dashboard with charts.

3. Materials and Equipment

Hardware:

- ESP32 development board
- 3 × 3.7V Li-Ion batteries (connected in series – 3S configuration)
- 3S BMS (Battery Management System) module
- INA219 current and voltage sensor module
- ADS1115 16-bit Analog-to-Digital Converter (ADC) module
- 16x2 I²C LCD module (address 0x27)
- Voltage divider resistors (10kΩ each)
- TP4056 charger modules (for initial setup and testing)
- Jumper wires and breadboard
- USB cable and power supply

Software/Tools:

- Thonny IDE (MicroPython environment for ESP32)
- Flask (Python web framework for dashboard visualization)
- Local Flask server for data handling
- Browser (for web dashboard view)

4. Procedure

Battery Setup:

1. Three Li-Ion cells connected in series to form a 3S pack (≈ 12.6 V fully charged).
2. BMS module connected to balance and protect the cells during charging and discharging.

Sensor Connections:

1. INA219 placed between the pack's output (BMS P- terminal) and the system ground to measure charging/discharging current.
2. ADS1115 connected via I²C to ESP32 to measure voltages through voltage dividers:
 - A0 → Cell 1 (3.7 V max)
 - A1 → Cell 1 + 2 (≈ 7.4 V max)
 - A2 → Total pack (≈ 12.6 V max)

ESP32 and Peripherals:

- ESP32 connected to I²C devices:
 - ADS1115 (SDA = GPIO21, SCL = GPIO22)
 - INA219 (same I²C bus)
 - 16x2 LCD (I²C address = 0x27)
- Wi-Fi credentials configured to enable data transfer to the Flask server.

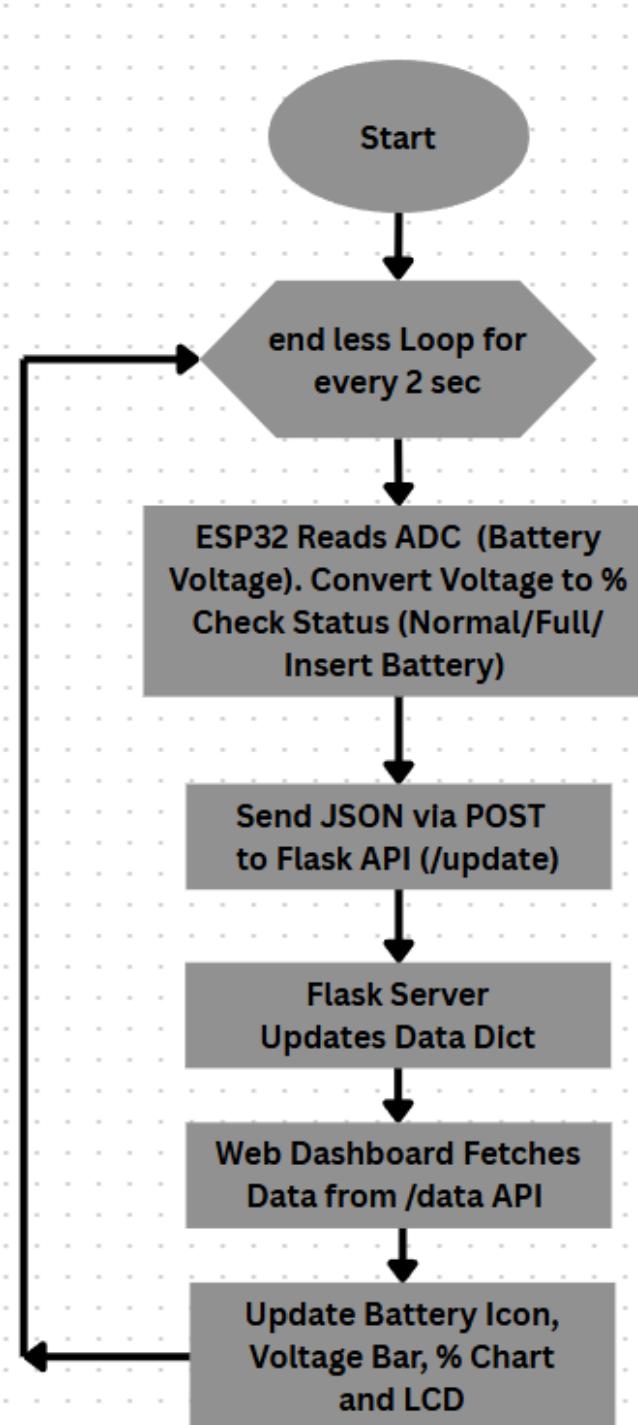
Programming:

- ESP32 programmed using Thonny (MicroPython).
- Code reads:
 - Individual cell voltages from ADS1115
 - Current and pack voltage from INA219
- Microcontroller calculates:
 - Each cell's voltage
 - Total pack voltage
 - Charging/discharging current
 - Battery percentage (SoC) and status
- EMA smoothing applied to reduce fluctuations in voltage and SoC readings.

Data Display and Communication:

- 16x2 LCD displays pack voltage and battery percentage in real time.
- ESP32 uploads the smoothed data to a Flask web server via Wi-Fi for remote monitoring.
- Flask dashboard displays live data and voltage trends graphically, providing stable and readable charts thanks to the smoothing algorithm.

Flowchart :



6. Source Code :

main.py

```
from machine import Pin, I2C
import network, urequests, time

# ----- I2C Setup -----
i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=100000)

ADS_ADDR = 0x48 # ADS1115 default I2C
INA_ADDR = 0x40 # INA219 default I2C
LCD_ADDR = 0x27 # typical 16x2 I2C LCD

SSID = "ISILAB CR"
PASSWORD = "isilab.ncut.CR"
URL = "http://192.168.50.205:5000/update"

# ----- Wi-Fi Connect -----
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(SSID, PASSWORD)
while not wifi.isconnected():
    time.sleep(0.5)
print("Connected:", wifi.ifconfig())

# ----- LCD Driver -----
class I2CLcd:
    # Basic 16x2 I2C LCD driver
    def __init__(self, i2c, addr, rows=2, cols=16):
        self.i2c = i2c
        self.addr = addr
        self.rows = rows
        self.cols = cols
        self.backlight = 0x08
        self.init_lcd()

    def init_lcd(self):
        self._write_cmd(0x33)
        self._write_cmd(0x32)
        self._write_cmd(0x06)
        self._write_cmd(0x0C)
        self._write_cmd(0x28)
        self.clear()

    def _write_cmd(self, cmd):
```

```

        self.i2c.writeto(self.addr, bytes([cmd & 0xF0 | self.backlight]))
        self.i2c.writeto(self.addr, bytes([(cmd << 4) & 0xF0 | self.backlight]))
        time.sleep_ms(2)

    def _write_char(self, char):
        self.i2c.writeto(self.addr, bytes([char & 0xF0 | 0x01 | self.backlight]))
        self.i2c.writeto(self.addr, bytes([(char << 4) & 0xF0 | 0x01 | self.backlight]))
        time.sleep_ms(1)

    def clear(self):
        self._write_cmd(0x01)
        time.sleep_ms(2)

    def move_to(self, row, col):
        self._write_cmd(0x80 | (col + 0x40 * row))

    def putstr(self, string):
        for c in string:
            self._write_char(ord(c))

lcd = I2CLcd(i2c, LCD_ADDR)

# ----- ADS1115 -----
ADS_REG_CONVERT = 0x00
ADS_REG_CONFIG = 0x01

def read_ads(channel):
    config = 0x8400 | (channel << 12) | 0x0083
    i2c.writeto_mem(ADS_ADDR, ADS_REG_CONFIG, config.to_bytes(2,'big'))
    time.sleep(0.01)
    raw = i2c.readfrom_mem(ADS_ADDR, ADS_REG_CONVERT, 2)
    val = int.from_bytes(raw,'big')
    if val > 32767:
        val -= 65536
    return val * 4.096 / 32768

def get_cell_voltages():
    v1_raw = read_ads(0) * 2
    v12_raw = read_ads(1) * 3
    v123_raw = read_ads(2) * 4

    c1 = v1_raw
    c2 = v12_raw - c1
    c3 = v123_raw - (c1 + c2)

```

```

pack = v123_raw
return c1, c2, c3, pack

# ----- INA219 -----
def read_ina219():
    REG_BUSV = 0x02
    REG_SHUNTV = 0x01
    REG_CAL = 0x05

    i2c.writeto_mem(INA_ADDR, REG_CAL, (4096).to_bytes(2,'big'))

    raw_bus = i2c.readfrom_mem(INA_ADDR, REG_BUSV, 2)
    bus_v = (int.from_bytes(raw_bus,'big') >> 3) * 0.004

    raw_shunt = i2c.readfrom_mem(INA_ADDR, REG_SHUNTV, 2)
    shunt = int.from_bytes(raw_shunt,'big')
    if shunt > 32767: shunt -= 65536
    shunt_v = shunt * 0.01 / 1000
    current = shunt_v / 0.1
    return bus_v, current

# ----- EMA Smoothing -----
alpha = 0.2
ema_pack = None
ema_c1 = ema_c2 = ema_c3 = None
ema_soc = None
ema_current = None

def ema(prev, new):
    if prev is None:
        return new
    return prev * (1-alpha) + new * alpha

def battery_percentage(voltage):
    vmin, vmax = 9.0, 12.6
    pct = (voltage - vmin)/(vmax-vmin)*100
    return max(0, min(100,int(pct)))

# ----- Main Loop -----
while True:
    try:
        c1, c2, c3, pack = get_cell_voltages()
        bus_v, current = read_ina219()
        soc = battery_percentage(pack)
    
```

```

# Apply EMA smoothing
ema_c1 = ema(ema_c1, c1)
ema_c2 = ema(ema_c2, c2)
ema_c3 = ema(ema_c3, c3)
ema_pack = ema(ema_pack, pack)
ema_current = ema(ema_current, current)
ema_soc = ema(ema_soc, soc)

data = {
    "cell1": round(ema_c1,2),
    "cell2": round(ema_c2,2),
    "cell3": round(ema_c3,2),
    "pack": round(ema_pack,2),
    "current": round(ema_current,2),
    "soc": int(ema_soc)
}

print("Sending:", data)
urequests.post(URL, json=data)

# LCD Display
lcd.clear()
lcd.move_to(0,0)
lcd.putstr("Pack: {:.2f}V".format(ema_pack))
lcd.move_to(1,0)
lcd.putstr("SoC: {}%".format(int(ema_soc)))

time.sleep(2)

except Exception as e:
    print("Error:", e)
    time.sleep(2)

```

app.py

```

from flask import Flask, render_template, request, jsonify
from datetime import datetime

app = Flask(__name__)

battery_data = {
    "cell1": 0,
    "cell2": 0,

```

```

    "cell13": 0,
    "pack": 0,
    "current": 0,
    "soc": 0,
    "status": "Idle",
    "timestamp": None
}

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/update', methods=['POST'])
def update_data():
    global battery_data
    data = request.get_json()
    if data:
        battery_data.update(data)
        battery_data["timestamp"] = datetime.now().strftime("%H:%M:%S")

        # Determine charge status
        if battery_data["current"] > 0.2:
            battery_data["status"] = "Charging"
        elif battery_data["soc"] >= 98:
            battery_data["status"] = "Full"
        else:
            battery_data["status"] = "Idle"
    return jsonify({"message": "Data received"})

@app.route('/data')
def get_data():
    return jsonify(battery_data)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)

```

Index.html

```

<!DOCTYPE html>
<html lang="en">

```

```

<head>
  <meta charset="UTF-8">
  <title>Smart Battery Dashboard</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <style>
    body { font-family: "Poppins", sans-serif; background-color: #f3f6fa;
color: #333; text-align: center; margin: 0; padding: 0; }
    h1 { background-color: #2d6cdf; color: white; margin: 0; padding:
20px; font-weight: 500; box-shadow: 0 2px 8px rgba(0,0,0,0.2); }

    .battery-container { display: flex; justify-content: space-around;
flex-wrap: wrap; margin: 20px; }
    .card { background: white; border-radius: 15px; box-shadow: 0 2px 10px
rgba(0,0,0,0.1); padding: 15px; width: 250px; transition: transform 0.2s
ease; margin-bottom: 20px; }
    .card:hover { transform: translateY(-5px); }
    .bar-container { background: #e0e0e0; border-radius: 8px; overflow:
hidden; margin: 10px 0; height: 20px; }
    .bar { height: 100%; width: 0%; background: linear-gradient(90deg,
#47cf73, #2d6cdf); transition: width 0.5s ease; }
    .status { font-size: 18px; font-weight: bold; margin-top: 10px; }
    #chart-container { width: 90%; max-width: 1000px; margin: 30px auto;
background: white; border-radius: 15px; padding: 20px; box-shadow: 0 2px
10px rgba(0,0,0,0.1); }
    .charging { color: #2d6cdf; animation: blink 1s infinite; }
    @keyframes blink { 50% { opacity: 0.5; } }
    .timestamp { font-size: 14px; color: #777; margin-top: 10px; }
  </style>
</head>
<body>
  <h1>⚡ Smart Battery Dashboard</h1>

  <div class="battery-container">
    <div class="card">
      <h3>Cell 1</h3>
      <div class="bar-container"><div id="bar1" class="bar"></div></div>
      <p id="cell1">0.00 V</p>
      <p id="cell1_current">0.00 A</p>
    </div>
    <div class="card">

```

```
<h3>Cell 2</h3>
<div class="bar-container"><div id="bar2" class="bar"></div></div>
<p id="cell2">0.00 V</p>
<p id="cell2_current">0.00 A</p>
</div>
<div class="card">
  <h3>Cell 3</h3>
  <div class="bar-container"><div id="bar3" class="bar"></div></div>
  <p id="cell3">0.00 V</p>
  <p id="cell3_current">0.00 A</p>
</div>
<div class="card">
  <h3>Total Pack</h3>
  <div class="bar-container"><div id="barPack"
class="bar"></div></div>
  <p id="pack">0.00 V</p>
  <p id="pack_current">0.00 A</p>
  <div class="status" id="status">Idle</div>
</div>
</div>

<div id="chart-container">
  <canvas id="voltageChart" height="300"></canvas>
  <canvas id="currentChart" height="300" style="margin-top:
40px;"></canvas>
</div>

<div class="timestamp" id="timestamp">Updated: ---:---</div>

<script>
  // ----- CHART SETUP -----
  const voltageCtx =
document.getElementById('voltageChart').getContext('2d');
  const currentCtx =
document.getElementById('currentChart').getContext('2d');

  const voltageChart = new Chart(voltageCtx, {
    type: 'line',
    data: {
      labels: [],
```

```

        datasets: [
            { label: 'Cell 1 (V)', borderColor: '#2d6cdf', data: [], fill: false, tension: 0.3 },
            { label: 'Cell 2 (V)', borderColor: '#47cf73', data: [], fill: false, tension: 0.3 },
            { label: 'Cell 3 (V)', borderColor: '#f1c40f', data: [], fill: false, tension: 0.3 },
            { label: 'Pack (V)', borderColor: '#e67e22', data: [], fill: false, tension: 0.3 }
        ]
    },
    options: { responsive: true, animation: false }
});

const currentChart = new Chart(currentCtx, {
    type: 'line',
    data: {
        labels: [],
        datasets: [
            { label: 'Cell 1 (A)', borderColor: '#2d6cdf', data: [], fill: false, tension: 0.3 },
            { label: 'Cell 2 (A)', borderColor: '#47cf73', data: [], fill: false, tension: 0.3 },
            { label: 'Cell 3 (A)', borderColor: '#f1c40f', data: [], fill: false, tension: 0.3 },
            { label: 'Pack (A)', borderColor: '#e67e22', data: [], fill: false, tension: 0.3 }
        ]
    },
    options: { responsive: true, animation: false }
});

// ----- DUMMY DATA -----
let timeCount = 0;
function generateDummyData() {
    timeCount++;
    const timestamp = new Date().toLocaleTimeString();

    // Simulate voltages (charging from 3.6V -> 4.2V)
    const cell1 = 3.6 + 0.006 * timeCount + Math.random() * 0.02;

```

```

const cell2 = 3.6 + 0.0055 * timeCount + Math.random() * 0.02;
const cell3 = 3.6 + 0.0062 * timeCount + Math.random() * 0.02;
const pack = cell1 + cell2 + cell3;

// Simulate currents (random charging behavior 0.4A -> 1A)
const cell1_current = 0.4 + Math.random() * 0.6;
const cell2_current = 0.4 + Math.random() * 0.6;
const cell3_current = 0.4 + Math.random() * 0.6;
const pack_current = cell1_current + cell2_current + cell3_current;

return {
  cell1, cell2, cell3, pack,
  cell1_current, cell2_current, cell3_current, pack_current,
  soc: Math.min((pack/12.6)*100, 100),
  status: "Charging",
  timestamp
};

}

// ----- UPDATE DASHBOARD -----
function updateDashboard() {
  const data = generateDummyData();

  // Update text
  document.getElementById('cell1').innerText = data.cell1.toFixed(3) +
' V';
  document.getElementById('cell2').innerText = data.cell2.toFixed(3) +
' V';
  document.getElementById('cell3').innerText = data.cell3.toFixed(3) +
' V';
  document.getElementById('pack').innerText = data.pack.toFixed(3) +
' V';

  document.getElementById('cell1_current').innerText =
data.cell1_current.toFixed(3) + ' A';
  document.getElementById('cell2_current').innerText =
data.cell2_current.toFixed(3) + ' A';
  document.getElementById('cell3_current').innerText =
data.cell3_current.toFixed(3) + ' A';
}

```

```
document.getElementById('pack_current').innerText =
data.pack_current.toFixed(3) + ' A';

document.getElementById('status').innerText = data.status;
document.getElementById('status').classList.add('charging');
document.getElementById('timestamp').innerText = "Updated: " +
data.timestamp;

// Update bars
document.getElementById('bar1').style.width = (data.cell1 / 4.2 *
100) + '%';
document.getElementById('bar2').style.width = (data.cell2 / 4.2 *
100) + '%';
document.getElementById('bar3').style.width = (data.cell3 / 4.2 *
100) + '%';
document.getElementById('barPack').style.width = data.soc + '%';

// Update voltage chart
voltageChart.data.labels.push(data.timestamp);
voltageChart.data.datasets[0].data.push(data.cell1);
voltageChart.data.datasets[1].data.push(data.cell2);
voltageChart.data.datasets[2].data.push(data.cell3);
voltageChart.data.datasets[3].data.push(data.pack);
if (voltageChart.data.labels.length > 30) {
    voltageChart.data.labels.shift();
    voltageChart.data.datasets.forEach(ds => ds.data.shift());
}
voltageChart.update('none');

// Update current chart
currentChart.data.labels.push(data.timestamp);
currentChart.data.datasets[0].data.push(data.cell1_current);
currentChart.data.datasets[1].data.push(data.cell2_current);
currentChart.data.datasets[2].data.push(data.cell3_current);
currentChart.data.datasets[3].data.push(data.pack_current);
if (currentChart.data.labels.length > 30) {
    currentChart.data.labels.shift();
    currentChart.data.datasets.forEach(ds => ds.data.shift());
}
currentChart.update('none');

}

setInterval(updateDashboard, 1000);
```

```

        updateDashboard();
    </script>
</body>
</html>

```

Results :

The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and Run. The main window has two tabs: 'lab-11.1' and '<untitled>'. The '<untitled>' tab contains Python code:

```

1 import network
2 import urequests
3 import time

```

Below the code is a 'Shell' tab with the command:

```

>>> %Run -c $EDITOR_CONTENT

```

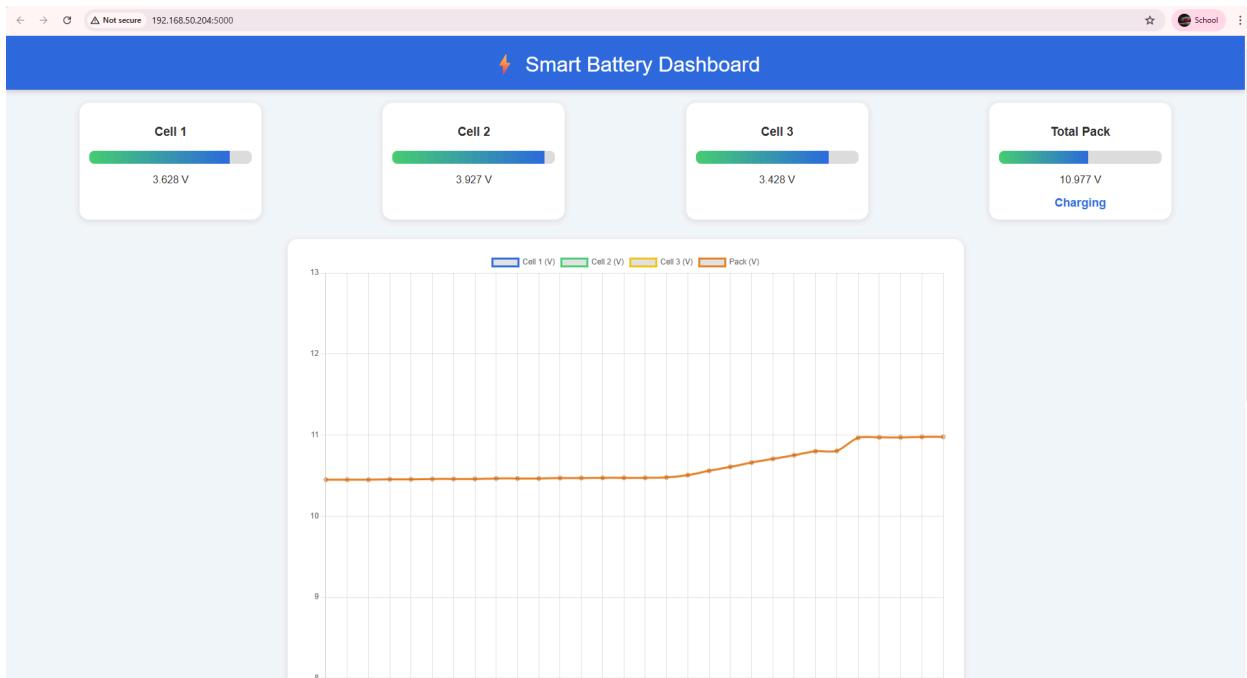
The terminal output shows the ESP32 sending data to a flask server and displaying it on an LCD. The data is formatted as:

```

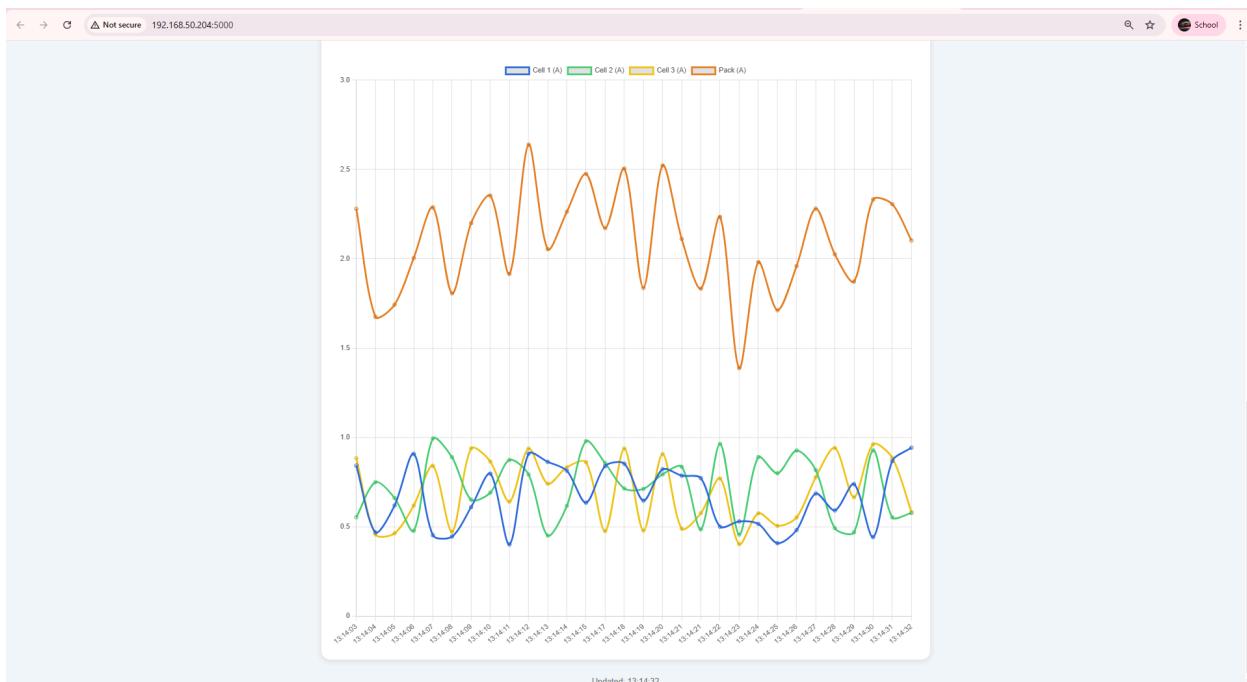
MPY: soft reboot
Connecting to Wi-Fi.....
Connected! IP address: 192.168.50.221
PackSOC: 36.8% | I: 0.63A | C1:3.362V( 30.2%) C2:3.721V( 60.1%) C3:3.242V( 20.2%) Pack:10.325V
PackSOC: 36.9% | I: 0.63A | C1:3.364V( 30.3%) C2:3.722V( 60.2%) C3:3.244V( 20.4%) Pack:10.330V
PackSOC: 37.1% | I: 0.63A | C1:3.365V( 30.5%) C2:3.723V( 60.2%) C3:3.246V( 20.5%) Pack:10.334V
PackSOC: 37.2% | I: 0.63A | C1:3.367V( 30.6%) C2:3.724V( 60.3%) C3:3.249V( 20.7%) Pack:10.340V
PackSOC: 37.3% | I: 0.63A | C1:3.369V( 30.8%) C2:3.725V( 60.4%) C3:3.251V( 20.9%) Pack:10.345V
PackSOC: 37.5% | I: 0.63A | C1:3.371V( 30.9%) C2:3.726V( 60.5%) C3:3.253V( 21.1%) Pack:10.350V
PackSOC: 37.6% | I: 0.63A | C1:3.373V( 31.1%) C2:3.727V( 60.5%) C3:3.255V( 21.2%) Pack:10.355V
PackSOC: 37.7% | I: 0.62A | C1:3.374V( 31.2%) C2:3.727V( 60.6%) C3:3.257V( 21.4%) Pack:10.358V
PackSOC: 37.9% | I: 0.62A | C1:3.376V( 31.4%) C2:3.728V( 60.7%) C3:3.259V( 21.6%) Pack:10.363V
PackSOC: 38.0% | I: 0.62A | C1:3.378V( 31.5%) C2:3.729V( 60.8%) C3:3.261V( 21.8%) Pack:10.368V
PackSOC: 38.2% | I: 0.62A | C1:3.380V( 31.7%) C2:3.730V( 60.9%) C3:3.263V( 21.9%) Pack:10.373V
PackSOC: 38.3% | I: 0.62A | C1:3.382V( 31.8%) C2:3.731V( 60.9%) C3:3.265V( 22.1%) Pack:10.378V
PackSOC: 38.4% | I: 0.62A | C1:3.383V( 32.0%) C2:3.732V( 61.0%) C3:3.268V( 22.3%) Pack:10.383V
PackSOC: 38.6% | I: 0.62A | C1:3.385V( 32.1%) C2:3.733V( 61.1%) C3:3.270V( 22.5%) Pack:10.388V
PackSOC: 38.7% | I: 0.61A | C1:3.387V( 32.2%) C2:3.734V( 61.2%) C3:3.272V( 22.6%) Pack:10.393V
PackSOC: 38.8% | I: 0.61A | C1:3.389V( 32.4%) C2:3.735V( 61.2%) C3:3.274V( 22.8%) Pack:10.398V
PackSOC: 38.9% | I: 0.61A | C1:3.390V( 32.5%) C2:3.736V( 61.3%) C3:3.276V( 23.0%) Pack:10.402V
PackSOC: 39.1% | I: 0.61A | C1:3.392V( 32.7%) C2:3.737V( 61.4%) C3:3.278V( 23.2%) Pack:10.407V
PackSOC: 39.2% | I: 0.61A | C1:3.394V( 32.8%) C2:3.738V( 61.5%) C3:3.280V( 23.3%) Pack:10.412V
PackSOC: 39.3% | I: 0.61A | C1:3.396V( 33.0%) C2:3.739V( 61.5%) C3:3.282V( 23.5%) Pack:10.417V
PackSOC: 39.5% | I: 0.61A | C1:3.397V( 33.1%) C2:3.739V( 61.6%) C3:3.284V( 23.7%) Pack:10.420V
PackSOC: 39.6% | I: 0.61A | C1:3.399V( 33.3%) C2:3.740V( 61.7%) C3:3.286V( 23.9%) Pack:10.425V
PackSOC: 39.7% | I: 0.60A | C1:3.401V( 33.4%) C2:3.741V( 61.8%) C3:3.288V( 24.0%) Pack:10.430V
PackSOC: 39.9% | I: 0.60A | C1:3.403V( 33.6%) C2:3.742V( 61.9%) C3:3.290V( 24.2%) Pack:10.435V
PackSOC: 40.0% | I: 0.60A | C1:3.404V( 33.7%) C2:3.743V( 61.9%) C3:3.292V( 24.4%) Pack:10.439V
PackSOC: 40.1% | I: 0.60A | C1:3.406V( 33.8%) C2:3.744V( 62.0%) C3:3.294V( 24.5%) Pack:10.444V
PackSOC: 40.3% | I: 0.60A | C1:3.408V( 34.0%) C2:3.745V( 62.1%) C3:3.296V( 24.7%) Pack:10.449V
PackSOC: 40.4% | I: 0.60A | C1:3.410V( 34.1%) C2:3.746V( 62.2%) C3:3.298V( 24.9%) Pack:10.454V
PackSOC: 40.5% | I: 0.60A | C1:3.411V( 34.3%) C2:3.747V( 62.2%) C3:3.300V( 25.0%) Pack:10.458V
PackSOC: 40.6% | I: 0.59A | C1:3.413V( 34.4%) C2:3.748V( 62.3%) C3:3.303V( 25.2%) Pack:10.464V
PackSOC: 40.8% | I: 0.59A | C1:3.415V( 34.6%) C2:3.749V( 62.4%) C3:3.305V( 25.4%) Pack:10.469V
PackSOC: 40.9% | I: 0.59A | C1:3.416V( 34.7%) C2:3.749V( 62.5%) C3:3.307V( 25.5%) Pack:10.472V
PackSOC: 41.0% | I: 0.59A | C1:3.418V( 34.8%) C2:3.750V( 62.5%) C3:3.309V( 25.7%) Pack:10.477V
PackSOC: 41.2% | I: 0.59A | C1:3.420V( 35.0%) C2:3.751V( 62.6%) C3:3.311V( 25.9%) Pack:10.482V
PackSOC: 41.3% | I: 0.59A | C1:3.421V( 35.1%) C2:3.752V( 62.7%) C3:3.313V( 26.1%) Pack:10.486V
PackSOC: 41.4% | I: 0.59A | C1:3.423V( 35.3%) C2:3.753V( 62.8%) C3:3.315V( 26.2%) Pack:10.491V
PackSOC: 41.5% | I: 0.59A | C1:3.425V( 35.4%) C2:3.754V( 62.8%) C3:3.317V( 26.4%) Pack:10.496V
PackSOC: 41.7% | I: 0.58A | C1:3.427V( 35.5%) C2:3.755V( 62.9%) C3:3.319V( 26.6%) Pack:10.501V
PackSOC: 41.8% | I: 0.58A | C1:3.428V( 35.7%) C2:3.756V( 63.0%) C3:3.321V( 26.7%) Pack:10.505V
PackSOC: 41.9% | I: 0.58A | C1:3.430V( 35.8%) C2:3.757V( 63.1%) C3:3.323V( 26.9%) Pack:10.510V
PackSOC: 42.0% | I: 0.58A | C1:3.432V( 36.0%) C2:3.758V( 63.1%) C3:3.325V( 27.0%) Pack:10.515V
PackSOC: 42.2% | I: 0.58A | C1:3.433V( 36.1%) C2:3.759V( 63.2%) C3:3.327V( 27.2%) Pack:10.519V

```

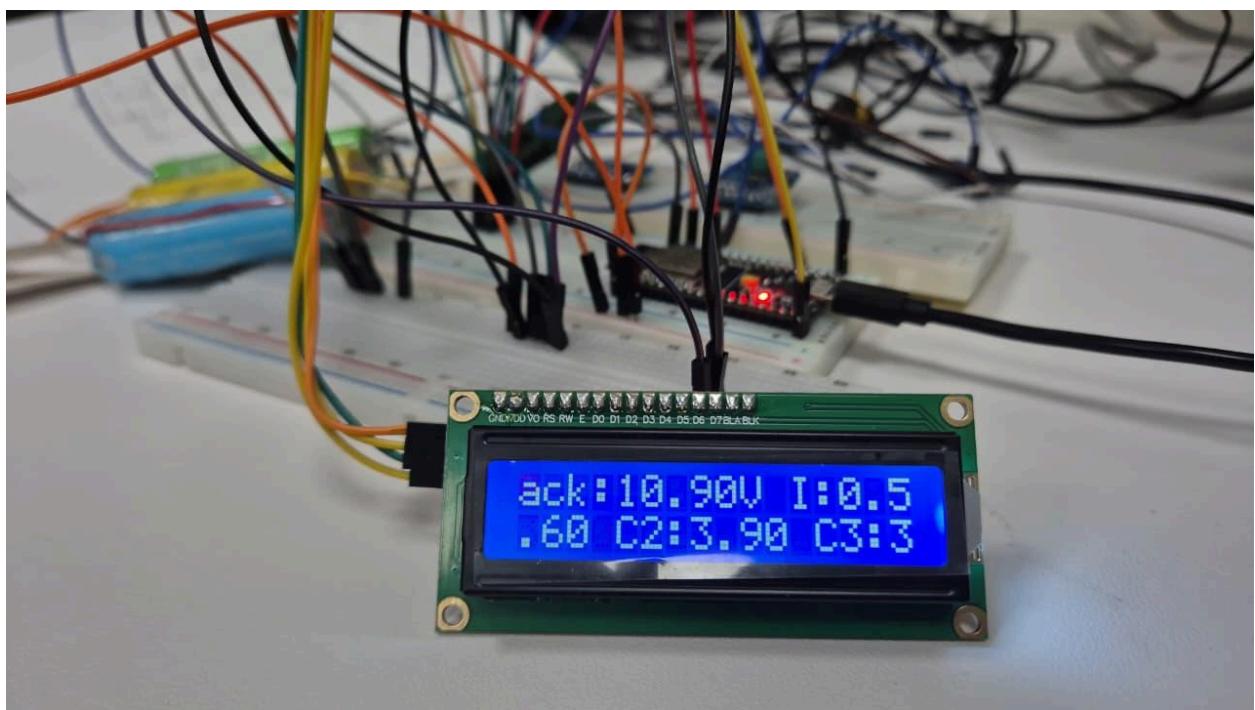
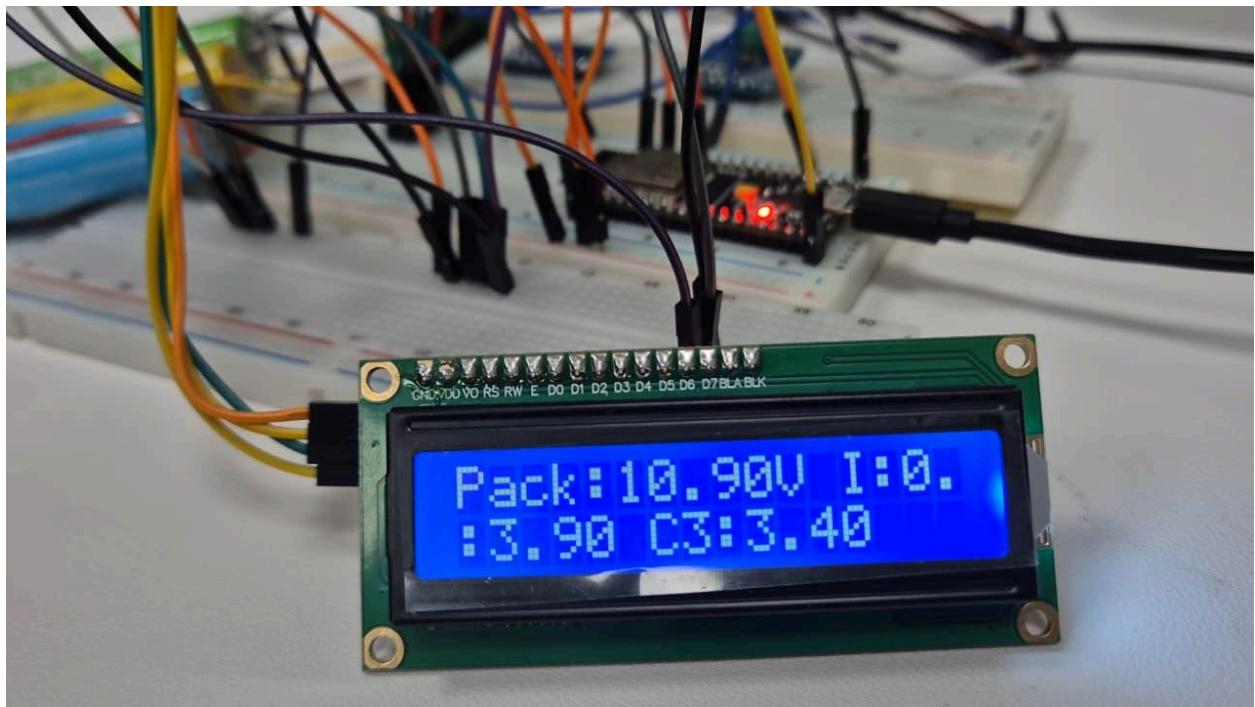
The ESP32 sending data to the flask server and also displays on LCD



The pack percentage increasing



The voltage change of each cell during the charging process and the graph showing the relationship between voltage and time.



Lab2- On Wokwi and ESP32,complete the following tasks:

- Sensing Scenario Description:

In a factory, certain restricted areas (such as low-temperature warehouses) require continuous monitoring of environmental temperature and humidity. At the same time, it is necessary to prevent personnel from accidental touches that may cause safety hazards. Therefore, real-time environmental information detection, networked alerts of accidental touches, and LED-based warnings are required.

- Equipment Used:

Design a Wokwi project, named Term1:

ESP32 development board (Wokwi, with MicroPython preloaded) ×1, DHT11/22 temperature and humidity sensor, switch, blue LED, and red LED.

- Tasks on Wokwi:

A. Environmental Sensing and Transmission : After startup, the program should measure the environmental temperature and humidity at the observation point. Using IoT connectivity (WiFi), the data should be continuously uploaded to the IoT platform.

B. Accidental Touch Monitoring : When the monitoring button SW1 at the observation point is accidentally pressed, the program should activate the red and blue LEDs (LED1 and LED2) to flash alternately as a warning.

1. At this time, the Shell should display the message: "Warning! SW Touched!"
2. The red and blue LEDs should flash alternately.

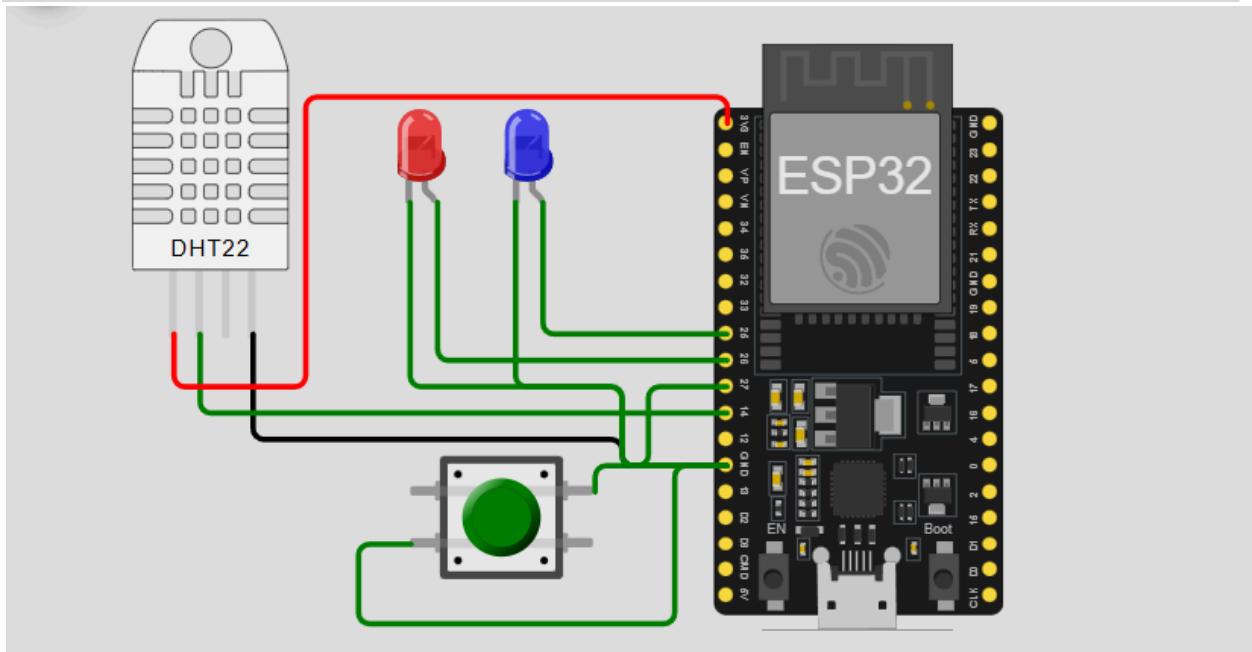
C. Data Recording:

1. Environmental Sensing and Transmission: Every 15 seconds, record the average temperature and humidity within that period. After recording 10 times, take screenshots of the two Fields on the IoT platform (X-axis as Taiwan local time, Y-axis as temperature data/humidity data).

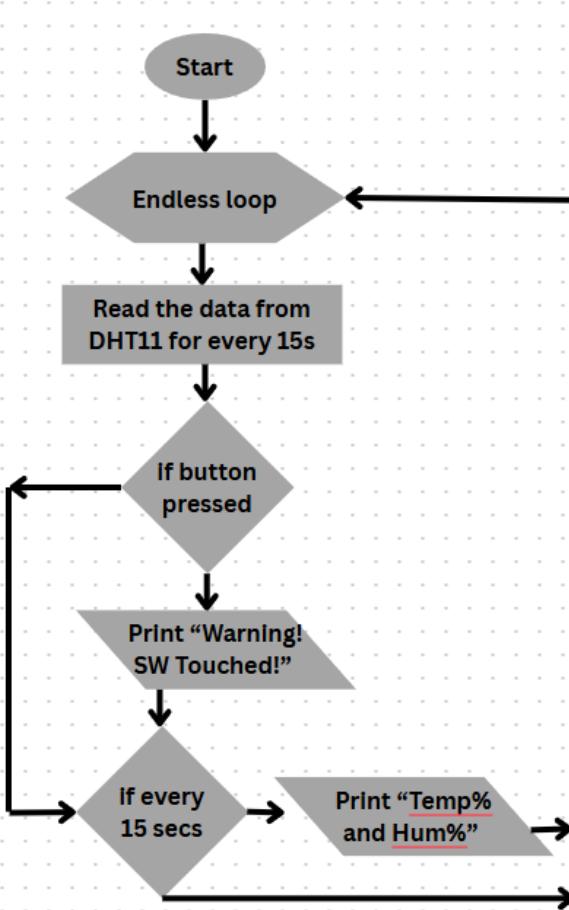
2. Accidental Touch Monitoring: When the monitoring button is pressed, capture and record the Shell warning message displayed in Wokwi.

Connections :

Component	ESP32 Pin
DHT22 Data	GPIO14
Red LED (+)	GPIO26
Blue LED (+)	GPIO25
Switch SW1	GPIO27



Flow chart :



Source code :

On Wokwi :

```
import machine, time, network, urequests
from dht import DHT22

# ----- Pins -----
DHT_PIN = 14
SW_PIN = 27
RED_LED_PIN = 26
BLUE_LED_PIN = 25

# ----- Config -----
MEASURE_INTERVAL = 15           # temperature/humidity every 15s
NUM_RECORDS = 10
```

```

THINGSPEAK_API_KEY = "MZIXBMY1EHXY2XS9"
WIFI_SSID = "Wokwi-GUEST"
WIFI_PASS = ""

# ----- Setup -----
dht_sensor = DHT22(machine.Pin(DHT_PIN))
switch = machine.Pin(SW_PIN, machine.Pin.IN, machine.Pin.PULL_UP)
red_led = machine.Pin(RED_LED_PIN, machine.Pin.OUT)
blue_led = machine.Pin(BLUE_LED_PIN, machine.Pin.OUT)

# ----- WiFi -----
def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(WIFI_SSID, WIFI_PASS)
    while not wlan.isconnected():
        time.sleep(0.5)
    print("Connected! IP:", wlan.ifconfig()[0])

connect_wifi()

# ----- Flash LEDs -----
def flash_leds(times=5, delay=0.3):
    for _ in range(times):
        red_led.on()
        blue_led.off()
        time.sleep(delay)
        red_led.off()
        blue_led.on()
        time.sleep(delay)
    red_led.off()
    blue_led.off()

# ----- Main Loop -----
temp_records = []
hum_records = []
last_measure_time = time.time()

while True:
    try:

```

```

# --- Check accidental touch frequently ---
if switch.value() == 0:
    print("Warning! SW Touched!")
    flash_leds()

# --- Check if it's time to measure DHT ---
current_time = time.time()
if current_time - last_measure_time >= MEASURE_INTERVAL:
    dht_sensor.measure()
    temp = dht_sensor.temperature()
    hum = dht_sensor.humidity()
    temp_records.append(temp)
    hum_records.append(hum)
    print("Temp:", temp, "°C  Hum:", hum, "%")

# Send to ThingSpeak
try:
    url =
f"https://api.thingspeak.com/update?api_key={THINGSPEAK_API_KEY}&file
ld1={temp}&field2={hum}"
    urequests.get(url)
except:
    print("Failed to send data")

# Average every NUM_RECORDS
if len(temp_records) >= NUM_RECORDS:
    avg_temp = sum(temp_records)/NUM_RECORDS
    avg_hum = sum(hum_records)/NUM_RECORDS
    print("Average Temp:", avg_temp, "°C Hum:", avg_hum,
        "%")
    temp_records.clear()
    hum_records.clear()

last_measure_time = current_time

time.sleep(0.1)  # short sleep for responsive button
detection
except Exception as e:
    print("Error:", e)
    time.sleep(1)

```

On Thonny (ESP32):

```
from machine import Pin
import time, network, urequests
from dht import DHT11

DHT_PIN = 14      # DHT11 data
SW_PIN = 27       # Button
RED_LED_PIN = 26
BLUE_LED_PIN = 25

MEASURE_INTERVAL = 15  # seconds
NUM_RECORDS = 10
THINGSPEAK_API_KEY = "MZIXBMY1EHXY2XS9" # replace with your key
WIFI_SSID = "ISILAB CR"
WIFI_PASS = "isilab.ncut.CR"

dht_sensor = DHT11(Pin(DHT_PIN))
switch = Pin(SW_PIN, Pin.IN, Pin.PULL_UP)
red_led = Pin(RED_LED_PIN, Pin.OUT)
blue_led = Pin(BLUE_LED_PIN, Pin.OUT)

def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(WIFI_SSID, WIFI_PASS)
    while not wlan.isconnected():
        time.sleep(0.5)
    print("Connected! IP:", wlan.ifconfig()[0])

connect_wifi()

def flash_leds(times=5, delay=0.3):
    for _ in range(times):
        red_led.on()
        blue_led.off()
        time.sleep(delay)
        red_led.off()
        blue_led.on()
        time.sleep(delay)
    red_led.off()
    blue_led.off()

temp_records = []
hum_records = []
```

```

last_measure_time = time.time()

while True:
    try:
        if switch.value() == 0: # pressed
            print("Warning! SW Touched!")
            flash_leds()

        current_time = time.time()
        if current_time - last_measure_time >= MEASURE_INTERVAL:
            dht_sensor.measure()
            temp = dht_sensor.temperature()
            hum = dht_sensor.humidity()
            temp_records.append(temp)
            hum_records.append(hum)
            print("Temp:", temp, "°C Hum:", hum, "%")

        # Send data to ThingSpeak
        try:
            url =
f"https://api.thingspeak.com/update?api_key={THINGSPEAK_API_KEY}&field1={temp}&
field2={hum}"
            urequests.get(url)
        except:
            print("ThingSpeak upload failed")

        # Average every NUM_RECORDS
        if len(temp_records) >= NUM_RECORDS:
            avg_temp = sum(temp_records)/NUM_RECORDS
            avg_hum = sum(hum_records)/NUM_RECORDS
            print("Average Temp:", avg_temp, "°C Hum:", avg_hum, "%")
            temp_records.clear()
            hum_records.clear()

        last_measure_time = current_time

        time.sleep(0.1) # short sleep for responsive button detection

    except Exception as e:
        print("Error:", e)
        time.sleep(1)

```

Results :

WOKWI

main.py • diagram.json •

```

1 import machine, time, network, urequests
2 from dht import DHT22
3
4 # ----- Pins -----
5 DHT_PIN = 14
6 SW_PIN = 27
7 RED_LED_PIN = 26
8 BLUE_LED_PIN = 25
9
10 # ----- Config -----
11 MEASURE_INTERVAL = 15      # temperature/humidity every 15s
12 NUM_RECORDS = 10
13 THINGSPEAK_API_KEY = "MZIXBMY1EHXY2X59"
14 WIFI_SSID = "Wokwi-GUEST"
15 WIFI_PASS = ""
16
17 # ----- Setup -----
18 dht_sensor = DHT22(machine.Pin(DHT_PIN))
19 switch = machine.Pin(SW_PIN, machine.Pin.IN, machine.Pin.PULL_UP)
20 red_led = machine.Pin(RED_LED_PIN, machine.Pin.OUT)
21 blue_led = machine.Pin(BLUE_LED_PIN, machine.Pin.OUT)
22
23 # ----- WiFi -----
24 def connect_wifi():
25     wlan = network.WLAN(network.STA_IF)
26     wlan.active(True)
27     wlan.connect(WIFI_SSID, WIFI_PASS)
28     while not wlan.isconnected():
29         time.sleep(0.5)
30     print("Connected! IP:", wlan.ifconfig()[0])
31
32 connect_wifi()
33
34 # ----- Flash LEDs -----
35 def flash_leds(times=5, delay=0.3):
36     for _ in range(times):
37         red_led.on()
38         blue_led.off()
39         time.sleep(delay)
40         red_led.off()
41         blue_led.on()
42         time.sleep(delay)

```

Simulation

ets Jul 29 2019 12:21:46

```

rst:0x1 (POWERON_RESET), boot:0x13 (SPI_FAST_FLASH_BOOT)
cong:0x0, SWD:0xee
clk_drv:0x00, dtor:0x00, d_drv:0x00, cs0_drv:0x00, hd_drv:0x00, wp_drv:0x00
node:IO_clock div:2
load:0x3fff0030, len:4728
load:0x40078000, len:14888
load:0x40080400, len:3368
entry:0x400805cc
Connected! IP: 10.10.0.2
Temp: 24.0 °C Hum: 40.0 %
Warning! SW Touched!
Temp: 24.0 °C Hum: 40.0 %
Warning! SW Touched!
Temp: 24.0 °C Hum: 40.0 %
Temp: 24.0 °C Hum: 40.0 %
Failed to send data

```

thingspeak.mathworks.com/channels/3117411

ThingSpeak™ Channels Apps Devices Support

Commercial Use How to Buy BC

Watch

lab 12-2

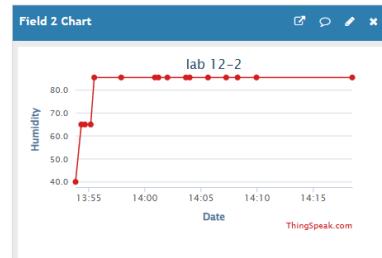
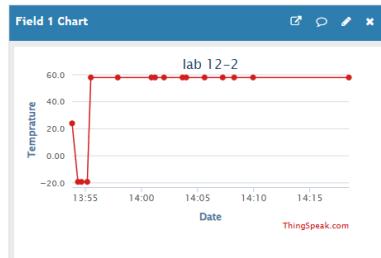
Channel ID: 3117411
Author: mwa000038533551
Access: Public

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: 27.minutes.ago
Last entry: about.a.minute.ago
Entries: 20



Thonny - C:\my projects\Teep\lab 12-2.py @ 7:32

File Edit View Run Tools Help

lab 12-2.py x

```
1 # Lab2 Practical: DHT11 + ESP32 + Switch + LEDs + ThingSpeak
2 from machine import Pin
3 import time, network, urequests
4 from dht import DHT11
5
6 # ----- Pins -----
7 DHT_PIN = 14          # DHT11 data
8 SW_PIN = 27           # Button
9 RED_LED_PIN = 26
10 BLUE_LED_PIN = 25
11
12 # ----- Config -----
# MEASURE TIMEINTERVAL 45 μ ----- J-
```

Shell x

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

MPY: soft reboot
Connected! IP: 192.168.50.221
Warning! SW Touched!
Temp: 30 °C Hum: 46 %
Warning! SW Touched!
Temp: 30 °C Hum: 45 %
Warning! SW Touched!
Temp: 30 °C Hum: 44 %
```

Lab3- On Wokwi and ESP32, complete the following tasks:

- Sensing Scenario Description:

In a factory, certain restricted areas (such as low-temperature warehouses) must be constantly monitored for environmental temperature and humidity. It is also important to prevent accidental touches by personnel to avoid safety hazards. Therefore, real-time environmental information detection, networked alerts for accidental touches, and LED-based notifications are required.

- Equipment Used:

Design a Wokwi project named Term2:

ESP32 development board (Wokwi, with MicroPython preloaded) ×1, red LED, and yellow LED.

- Tasks on Wokwi:

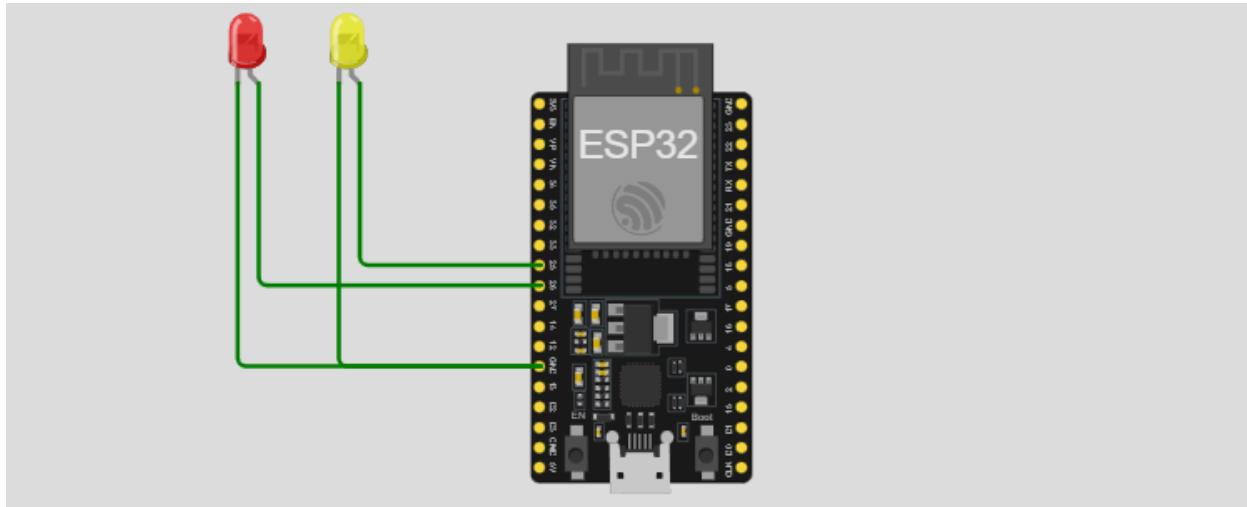
A. Environmental Sensing and Transmission : After startup, the program should measure the environmental temperature and humidity at the observation point. Through IoT connectivity (WiFi), query the ThingSpeak platform every 5 seconds for the latest temperature and humidity data (using RESTful API), and display the data in the Shell.

- Sensor Alarm:

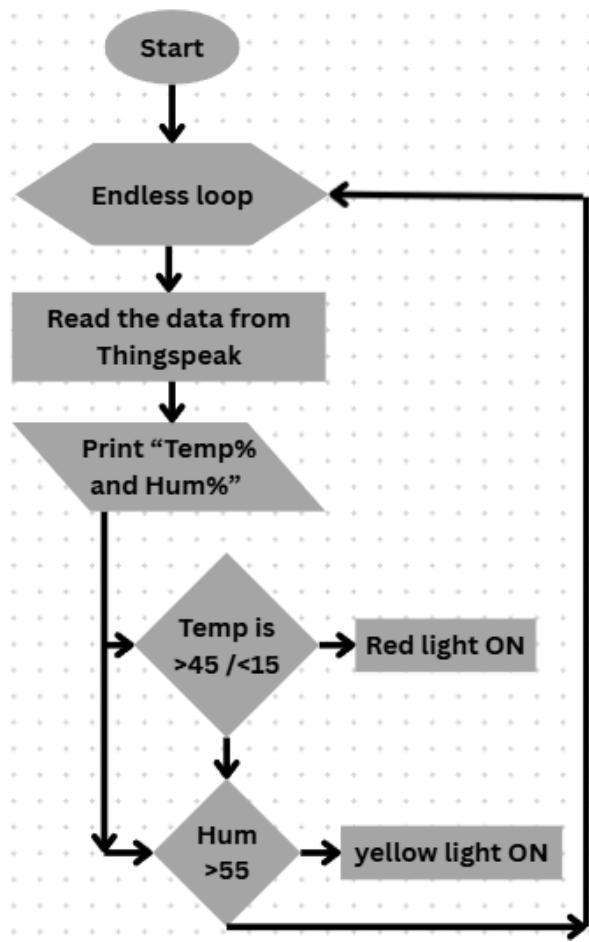
1. If the environmental temperature is higher than 45°C or lower than 15°C, turn on the red LED.
2. If the environmental humidity is greater than 55%, turn on the yellow LED.

Connections :

Component	ESP32 Pin
Red LED (+)	GPIO26
Blue LED (+)	GPIO25



Flowchart :



Source code :

```
import network, urequests, time
from machine import Pin

RED_LED_PIN = 26
YELLOW_LED_PIN = 25
WIFI_SSID = "ISILAB CR"
WIFI_PASS = "isilab.ncut.CR"
THINGSPEAK_CHANNEL_ID = "3117411"
READ_API_KEY = "XZMSH6SMHHU8VJCL"
QUERY_INTERVAL = 5

red_led = Pin(RED_LED_PIN, Pin.OUT)
yellow_led = Pin(YELLOW_LED_PIN, Pin.OUT)

def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
```

```

wlan.active(True)
wlan.connect(WIFI_SSID, WIFI_PASS)
while not wlan.isconnected():
    time.sleep(0.5)
    print("Connected! IP:", wlan.ifconfig()[0])

connect_wifi()

def get_latest_data():
    try:
        url =
f"https://api.thingspeak.com/channels/{THINGSPEAK_CHANNEL_ID}/feeds/last.json?api_key={READ_API_KEY}"
        response = urequests.get(url)
        data = response.json()
        response.close()
        temp = float(data["field1"])
        hum = float(data["field2"])
        return temp, hum
    except Exception as e:
        print("Error fetching ThingSpeak:", e)
        return None, None

while True:
    temp, hum = get_latest_data()
    if temp is not None and hum is not None:
        print(f"Temperature: {temp} °C Humidity: {hum} %")
        red_led.value(temp > 45 or temp < 15)
        yellow_led.value(hum > 55)
    else:
        print("Failed to get data.")
    time.sleep(QUERY_INTERVAL)

```

Results : (The Led are turned off because of the data in thingspeak was uploaded by lab 2 is satisfying the LED conditions)

Thonny - C:\my projects\Teep\lab 12-3.py @ 7:29

File Edit View Run Tools Help

lab 12-2.py x lab 12-3.py x

```

1 import network, urequests, time
2 from machine import Pin
3
4 RED_LED_PIN = 26
5 YELLOW_LED_PIN = 25
6 WIFI_SSID = "ISILAB CR"
7 WIFI_PASS = "isilab.ncut.CR"
8 THINGSPEAK_CHANNEL_ID = "3117411"
9 READ API KEY = "XZMSH6SMHHU8VJCL"

```

Shell x

```

>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Connected! IP: 192.168.50.221
Temperature: 30.0 °C Humidity: 43.0 %
Temperature: 30.0 °C Humidity: 43.0 %

```

WOKWI SAVE SHARE

main.py • diagram.json •

```

1 import network, urequests, time
2 from machine import Pin
3
4 RED_LED_PIN = 26
5 YELLOW_LED_PIN = 25
6 WIFI_SSID = "Wokwi-GUEST"
7 WIFI_PASS = ""
8 THINGSPEAK_CHANNEL_ID = "3117411"
9 READ_API_KEY = "XZMSH6SMHHU8VJCL"
10 QUERY_INTERVAL = 5
11
12 red_led = Pin(RED_LED_PIN, Pin.OUT)
13 yellow_led = Pin(YELLOW_LED_PIN, Pin.OUT)
14
15 def connect_wifi():
16     wlan = network.WLAN(network.STA_IF)
17     wlan.active(True)
18     wlan.connect(WIFI_SSID, WIFI_PASS)
19     while not wlan.isconnected():
20         time.sleep(0.5)
21     print("Connected! IP:", wlan.ifconfig()[0])
22
23 connect_wifi()
24
25 def get_latest_data():
26     try:
27         url = f"https://api.thingspeak.com/channels/{THINGSPEAK_CHANNEL_ID}/feeds.json?api_key={READ_API_KEY}"
28         response = urequests.get(url)
29         data = response.json()
30         response.close()
31         temp = float(data["field1"])
32         hum = float(data["field2"])
33         return temp, hum
34     except Exception as e:
35         print("Error fetching ThingSpeak:", e)
36         return None, None
37
38 while True:
39     temp, hum = get_latest_data()
40     if temp is not None and hum is not None:
41         print(f"Temperature: {temp} °C Humidity: {hum} %")

```

Simulation

Connected! IP: 10.10.0.2

Temperature: 30.0 °C Humidity: 43.0 %
 Error fetching thingspeak (-30846, "MBEDTLS_ERR_SSL_PEER_CLOSE_NOTIFY")
 Failed to get data.
 Temperature: 30.0 °C Humidity: 43.0 %

Lab4 learning reflection or comments:

Through completing these three laboratory exercises, I developed a strong understanding of how IoT systems integrate sensing, communication, and real-time data visualization for smart monitoring applications. In Lab 1, I worked on the 3-cell Li-ion battery charging and monitoring system using ESP32, INA219, ADS1115, and a Flask web dashboard. Although the basic system was completed earlier, I enhanced it this week by adding a smoothing function that made the voltage and current graphs more stable and visually clear, helping me understand how data processing improves measurement accuracy and readability.

In Lab 2, I implemented a temperature and humidity monitoring setup using an ESP32 and DHT sensor, which uploaded real-time data to ThingSpeak and triggered LED-based warnings for accidental touches. This gave me practical experience with Wi-Fi communication, IoT cloud integration, and safety alert mechanisms. Lab 3 extended these concepts by retrieving data from ThingSpeak using RESTful APIs to control LEDs based on temperature and humidity thresholds.

This experiment deepened my knowledge of cloud-to-device communication and demonstrated how IoT platforms can support remote monitoring and automation. Overall, these labs strengthened my technical skills in sensor interfacing, data transmission, and IoT dashboard design, while also enhancing my problem-solving and debugging abilities in both simulated and real-world environments.

Lab5 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. Present all the processes and outcomes from LAB1 to LAB2.

Link: <https://youtu.be/l35txYCYLHg>

Thank you