Temperature and Humidity Data Logging to Local CSV (runlinc + Python)

Contents

Introduction

This project aims to build a real-time temperature and humidity monitoring and data storage system. This version utilizes a local Python server to write sensor data into CSV files, enabling users to directly analyze and visualize the data using Excel.

Problem

In many application scenarios (such as greenhouse environmental monitoring and server room environmental control), real-time collection of temperature and humidity data is only the first step. Reliably storing and analyzing this data is equally important. Cloud-based solutions rely on networks and third-party services, while local storage solutions are simpler, avoiding authentication and complex API calls. However, traditional databases are complex to set up and costly to maintain, necessitating a lightweight, straightforward storage method.

Background

CSV files represent a simple and widely adopted structured data storage format that can be directly opened and processed by Excel, Numbers, and Google Sheets. By integrating the runlinc (E32W) board with a Python Flask

server, real-time data acquisition, storage, and local persistence can be easily achieved.

Ideas

The core concept involves using the runlinc board to collect real-time temperature and humidity data from a DHT11 sensor. This data is then uploaded via HTTP POST requests to a local Python server, which appends the data to a CSV file. This approach builds a lightweight yet reliable data storage system.

Plan

We have a DHT11–temperature and humidity sensor in our kits which we can use to
check the temperature and humidity around the sensor. We want to use the DHT11 sensor
to monitor the temperature and humidity around the crops.

Figure 1: Block diagram of Microchip outputs

Use the DHT11 sensor to collect temperature and humidity data.

The runlinc board uploads data to the local server via HTTP POST every second.

The Python Flask server receives the data and writes it to a CSV file.

Users can open the CSV file in Excel/Google Sheets for subsequent analysis.

runlinc Background

runlinc is a web page inside a Wi-Fi chip. The programming is done inside the browsers compare to programming inside a chip. The runlinc web page inside the Wi-Fi chip will command the microchips to do sensing, control, data logging Internet of Things (IoT). It can predict and command.

Part A: Design the Circuit on runlinc

Note: Refer to runlinc Wi-Fi Setup Guide document to connect to runlinc

Upgrade your runlinc V1.2 to V2.0 with reference to the file 'runlinc V2.0 Upgrade'.

Use the left side of the runlinc web page to construct an input/output (I/O).

For port D23 name it **tempHumidSensor** and set it as **DHT11_IN**.

In our circuit design, we will be using the DHT11–temperature and humidity sensor. We happen to have this in our kits, so these can be used on our circuit design, as per the plan.

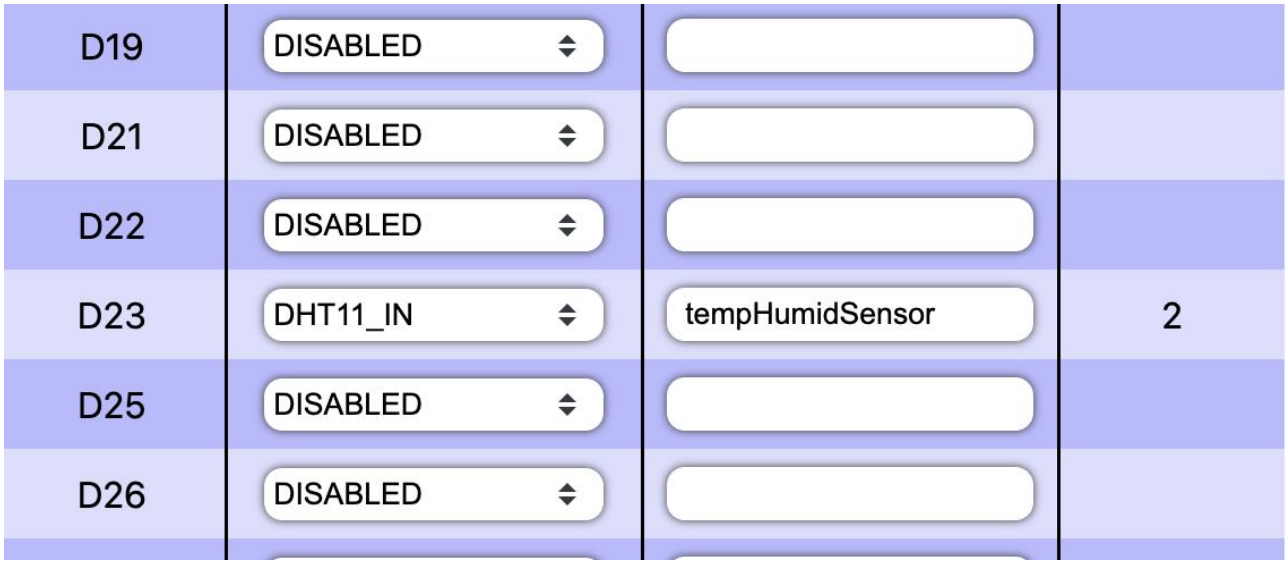

Figure 2: I/O configurations connections

Part B: Build the Circuit

Use the STEMSEL E32W board to connect the hardware. For this project we are using both the left and right I/O ports, with **negative port (-ve)** on the outer side, **positive port (+ve)** on the middle and **signal port (s)** on the inner side (as shown below).

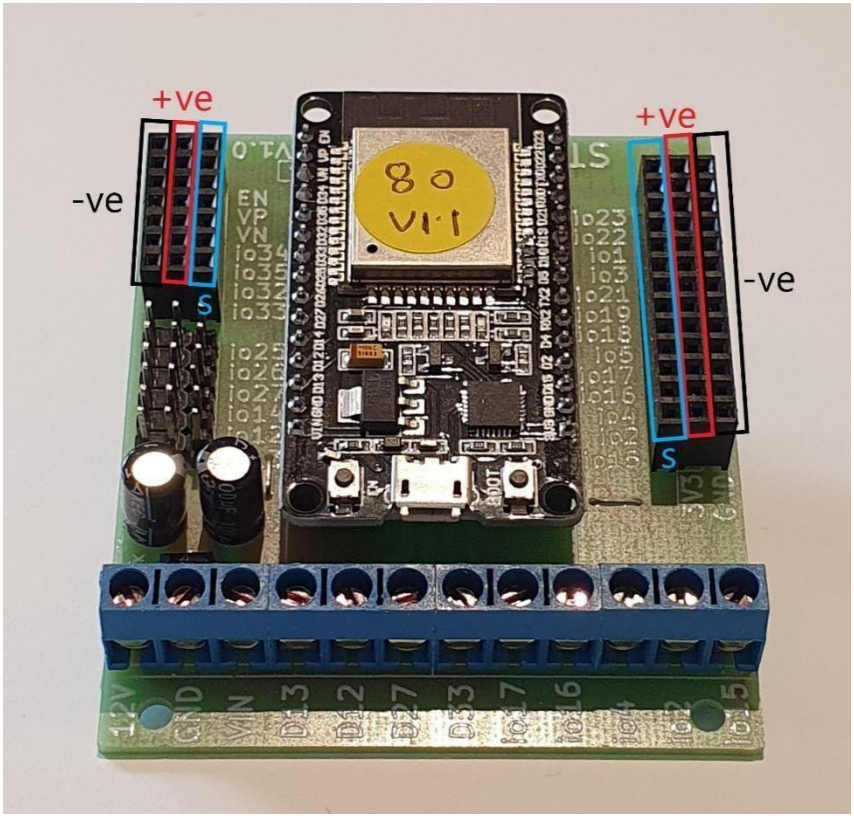

Figure 3: Negative, Positive and Signal port on the E32W board

There is one I/O part we are using for this project, a DHT11–temperature and humidity sensor, their respective pins are shown in the figure below.

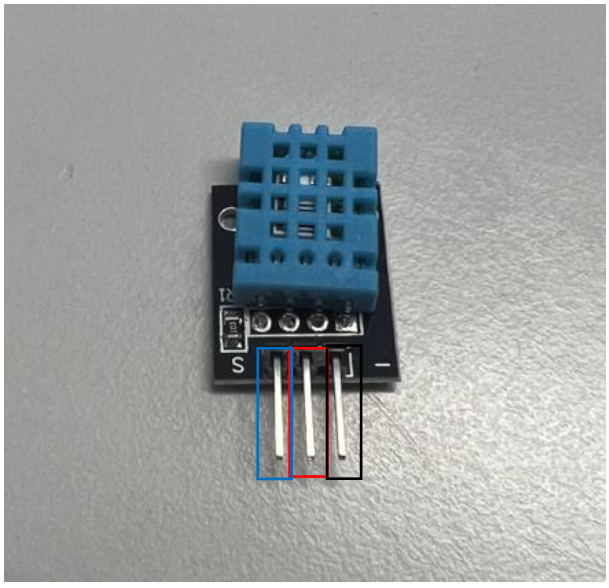

Figure 4: I/O part with negative, positive and signal pins indicated

Wiring instructions

a) Plug in the DHT11–temperature and humidity sensor to io23 on the E32W board.

b) Make sure the (-ve) pin are on the GND (outer) side of the I/O ports.

Figure 5: Circuit board connection with I/O part (side view)

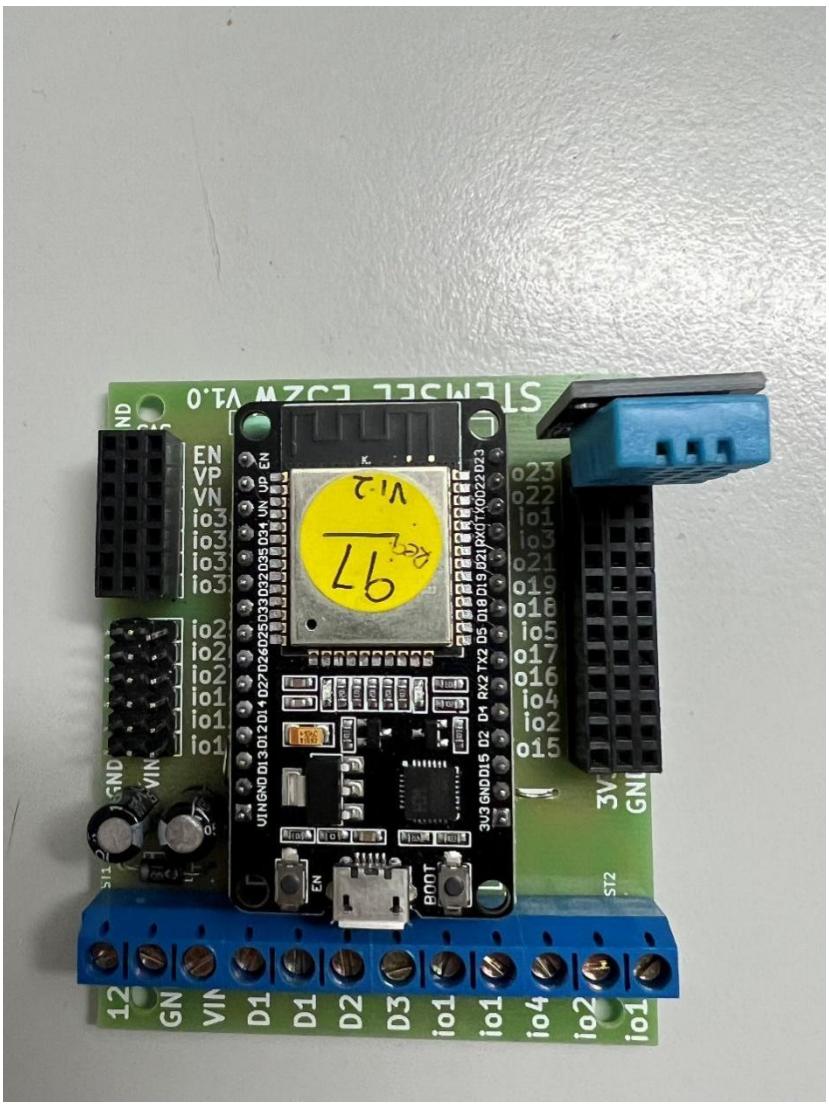

Figure 6: Circuit board connection with I/O part (top view)

Temperature and Humidity Data Monitoring and Storage

A. Run the Python Server

Python (Flask Server):

On the computer side, a Python Flask server receives the uploaded data. When the runlinc board sends temperature and humidity readings, the server writes each record into data.csv. If the file does not exist, it is created with headers. This ensures that all sensor readings are logged locally and can be opened directly in spreadsheet applications.

Save the following code as server.py on your computer (e.g., Desktop):

```
# server.py
from flask import Flask, request, make_response, jsonify
import csv, os

app = Flask(__name__)
CSV_FILE = "data.csv"

# Write header on first startup
if not os.path.exists(CSV_FILE):
    with open(CSV_FILE, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(["timestamp", "temperature", "humidity"])

# Add CORS response headers globally
@app.after_request
def add_cors_headers(resp):
    resp.headers["Access-Control-Allow-Origin"] = "*"
    resp.headers["Access-Control-Allow-Headers"] = "Content-Type"
    resp.headers["Access-Control-Allow-Methods"] = "POST, OPTIONS"
    return resp

@app.route("/upload", methods=["POST", "OPTIONS"])
def upload():
    if request.method == "OPTIONS":
        return ("", 204)   # Pre-screening request

    data = request.get_json(force=True, silent=True) or {}
```

```
    ts = data.get("timestamp")
    t  = data.get("temperature")
    h  = data.get("humidity")

    # basic verification
    if ts is None or t is None or h is None:
        return jsonify({"ok": False, "error": "missing fields"}), 400

    with open(CSV_FILE, "a", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow([ts, t, h])

    return jsonify({"ok": True})

if __name__ == "__main__":
    # 0.0.0.0 Make it accessible to other devices on the local area network; port 5000
    app.run(host="0.0.0.0", port=5000)
```

Open a terminal in the same folder and run:

pip install flask
python server.py

The server will start, showing:

Running on http://127.0.0.1:5000
Running on http://192.168.1.24:5000

Use the **192.168.x.x** address for runlinc.

B. Configure runlinc IDE

HTML:

First, we created a simple web page that allows the user to configure the Python server address via an input box. Then, it displays the current temperature and humidity values on the page, using dark red for temperature and blue for humidity. A status field is also included to indicate whether data has been successfully uploaded.

```
<div style="max-width:900px;margin:0 auto;font-family:Arial,Helvetica,sans-serif">
  <h2 style="color:#0b5bd3;margin-bottom:8px">Temperature & Humidity Logger →
CSV</h2>
```

```
<!-- Server address configuration: Replace <your computer IP> with your computer's IP
address on the same WiFi network -->
  <label>Python Server Upload URL:</label>
  <input id="serverUrl" type="text" style="width:100%"
          value="http://<your computer IP>:5000/upload">
  <small>Example：http://192.168.1.24:5000/upload</small>
  <hr>
  <div>
    <div style="font-size:18px;color:#a40000" id="Temperature">Current Temperature: N/A
(°C)</div>
    <div style="font-size:18px;color:#0044cc" id="Humidity">Current Humidity: N/A (%)</div>
    <div style="font-size:14px;color:#666" id="Status">Status: idle</div>
  </div>

  <p style="margin-top:12px">
   Explanation: Please set<b>D23</b>to<b>DHT11 IN</b>in the left I/O area and name it
<b>TempHumidSensor</b>(consistent with code).
  </p>
</div>
```

JavaScript:

Next, we used JavaScript to update the displayed values in real time and to prepare the sensor data for transmission. The script packs the timestamp, temperature, and humidity into JSON format, and then sends it to the Python server using an HTTP POST request. The status field is updated according to the result of the upload.

```
var currentTempHumidity;
// Display current measurement values
function displayValues(temp, humi) {
  document.getElementById('Temperature').innerHTML =
    "Current Temperature: " + temp + " °C";
  document.getElementById('Humidity').innerHTML =
    "Current Humidity: " + humi + " %";
}
// POST data to the local Python server, and the server writes to CSV.
async function postToServer(payload) {
  const url = document.getElementById('serverUrl').value.trim();
  if (!url) return;
  // Send request
  const res = await fetch(url, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(payload)
```

```
  });
  return res.ok;
}
// Update status bar
function setStatus(text) {
  document.getElementById('Status').innerText = "Status: " + text;
}
```

JavaScript Loop:

Finally, we set a one-second loop to continuously read the temperature and humidity data from the DHT11 sensor. Each reading is displayed on the web page and simultaneously sent to the Python server, where it is appended into the CSV file for later analysis.

```
// Read from DHT11
currentTempHumidity = DHT11_In(tempHumidSensor).split(",");
var currentTemp = parseFloat(currentTempHumidity[0]);
var currentHumidity = parseFloat(currentTempHumidity[1]);
if (!isNaN(currentTemp) && !isNaN(currentHumidity)) {
  // Display on page
  displayValues(currentTemp.toFixed(2), currentHumidity.toFixed(2));
  // Assemble uploaded data (with timestamp)
  const nowISO = new Date().toLocaleString();
  const data = {
    timestamp: nowISO,
    temperature: Number(currentTemp.toFixed(2)),
    humidity: Number(currentHumidity.toFixed(2))
  };
  // Send to Python server
  try {
    setStatus("uploading...");
    const ok = await postToServer(data);
    setStatus(ok ? ("uploaded " + JSON.stringify(data)) : "upload failed");
  } catch (e) {
    setStatus("error: " + e);
  }
}
// 1-second interval
await mSec(1000);
```

How to find your ip address

On Windows, we can check the computer's IP address directly from the Wi-Fi connection information:

1. Click the **Wi-Fi icon** on the bottom right of the taskbar.
2. Right-click your connected Wi-Fi network → select **Properties** (or **Status**).
3. Scroll down to the **Properties** section.
4. Find the entry **IPv4 address** (usually in the format 192.168.x.x or 10.x.x.x).

For example: IPv4 address: **192.168.1.24**

Figure 7: How to find your ip address

Then enter the following in runlinc:

http://192.168.1.24:5000/upload

Expected Results

runlinc webpage displays current temperature and humidity values.
Every second, data is sent to the Python server and stored in data.csv.

```
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.1.24:5000
Press CTRL+C to quit
192.168.1.24 - - [25/Aug/2025 14:11:51] "OPTIONS /upload HTTP/1.1" 204 -
192.168.1.24 - - [25/Aug/2025 14:11:51] "POST /upload HTTP/1.1" 200 -
192.168.1.24 - - [25/Aug/2025 14:11:52] "POST /upload HTTP/1.1" 200 -
192.168.1.24 - - [25/Aug/2025 14:11:53] "POST /upload HTTP/1.1" 200 -
192.168.1.24 - - [25/Aug/2025 14:11:54] "POST /upload HTTP/1.1" 200 -
192.168.1.24 - - [25/Aug/2025 14:11:55] "POST /upload HTTP/1.1" 200 -
PS C:\Users\61048\Desktop>
```

Figure 8: Python server

data.csv grows line by line,which shows on figure 10:

2025/8/25 14:23	22	61
2025/8/25 14:23	22	64
2025/8/25 14:23	22	64
2025/8/25 14:23	22	64
2025/8/25 14:23	22	64
2025/8/25 14:23	22	64
2025/8/25 14:23	22	64
2025/8/25 14:23	22	63

Figure 9: data.csv

The CSV file can be opened directly in Excel or Google Sheets for analysis.

Notes

Ensure runlinc and the computer are connected to the **same Wi-Fi network**.
Always use the **192.168.x.x** IP (not 127.0.0.1) in runlinc.
To stop the server, press **CTRL+C** in the terminal.

Summary

This project provides a concise local temperature and humidity monitoring and data storage solution. Compared to the cloud based Google Sheets version,

this solution eliminates API authentication and external dependencies, making it more suitable for rapid deployment and local analysis in laboratories and small environments.