runlinc Intermediate Project 14: Temperature Monitoring with DS18B20 (E32W Version)

Contents

Introduction

This project focuses on building a real-time Temperature Monitoring System using the DS18B20 digital sensor and an ESP32-based E32W development board running runlinc. Data is displayed locally on a web page hosted inside the Wi-Fi chip. The project is structured in two incremental steps: (1) simple real-time numerical display of temperature, and (2) plotting a scrolling graph of the last readings.

Problem

Accurate and continuous temperature monitoring is a common requirement in many applications such as food storage, environmental control, and laboratory experiments. Traditional methods often rely on manual readings or expensive data-logging equipment. A lightweight IoT approach that can be quickly prototyped and deployed using readily available components helps lower the barrier to adoption.

Background

The DS18B20 is a 1-Wire digital temperature sensor capable of measuring temperatures from –55 °C to +125 °C with 12-bit resolution. When paired with runlinc on an ESP32, the sensor's readings can be accessed directly from the browser, allowing fast development of interactive dashboards without flashing new firmware.

Ideas

Leverage runlinc's in-browser programming model to:

- Read temperature from the DS18B20 sensor (named tempSensor on port D4).
- Update the displayed value every second.
- After verifying correct readings, extend the page with an SVG graph that scrolls horizontally, showing the last ~100 s of data.

Plan

We have a DS18B20 digital temperature sensor that can measure the ambient temperature around the device, and we plan to deploy it for temperature monitoring in crop-growing areas.

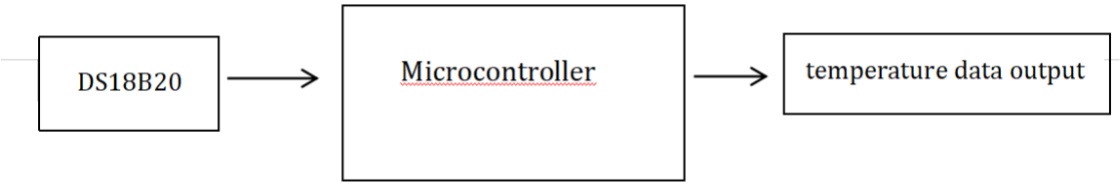

Figure 1: Microchip Input/Output Block Diagram

The DS18B20 digital temperature sensor is used as a real-time temperature acquisition device.
In the main loop, the DS18X20 function is called every second to read the current temperature value, and the result is appended to an array to store the temperature history. The latest temperature is immediately displayed in the "Current Temperature" text box on the webpage. At the same time, an SVG line chart is used to plot the temperature data in real time, enabling dynamic visualization of temperature changes over time.

In summary, the DS18B20 is used in this project to continuously monitor environmental temperature (e.g., in a crop growing area) at a per-second level and visualize the collected data in real time, allowing users to conveniently track temperature trends online.

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**

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

For port D4 name it tempSensor and set it as DS18X20_IN.

In our circuit design, we will be using the DS18B20 temperature sensor. We happen to have this in our kits, so these can be used on our circuit design, as per the plan.

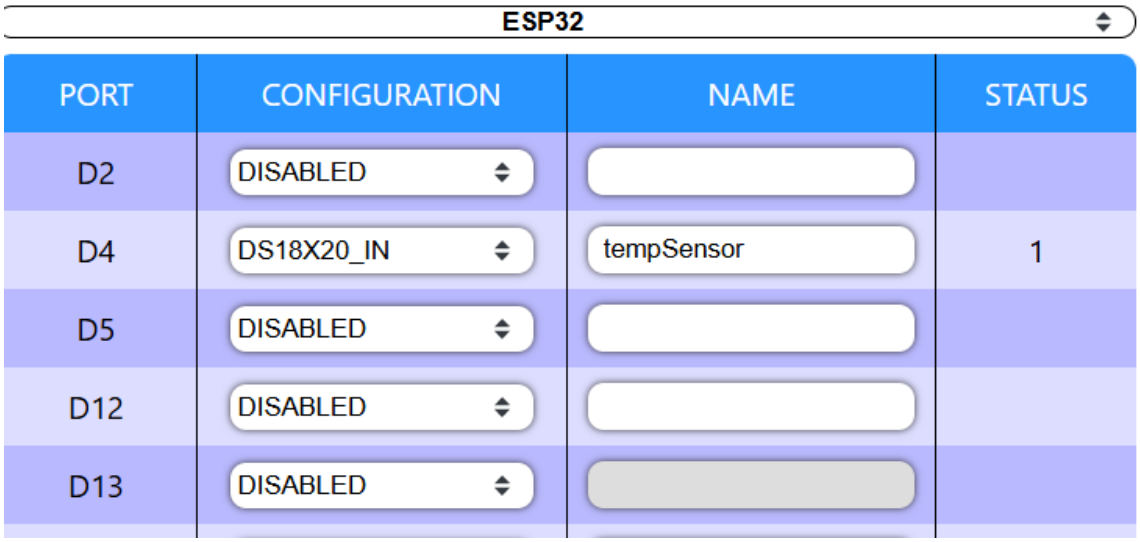

Figure 2: I/O configurations connections

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).

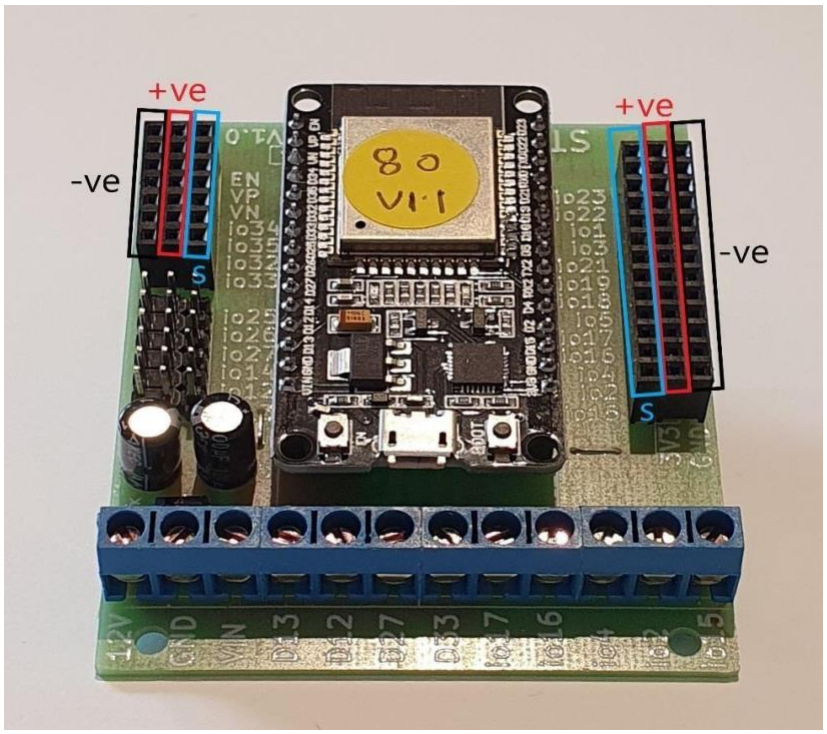

Figure 3: Negative, Positive and Signal port on the E32W board

There is one I/O part we are using for this project, a DS18B20 temperature sensor. The part has two variants which are built differently for different scenarios, you need to choose one type to connect with. their respective pins are shown in the figures below.

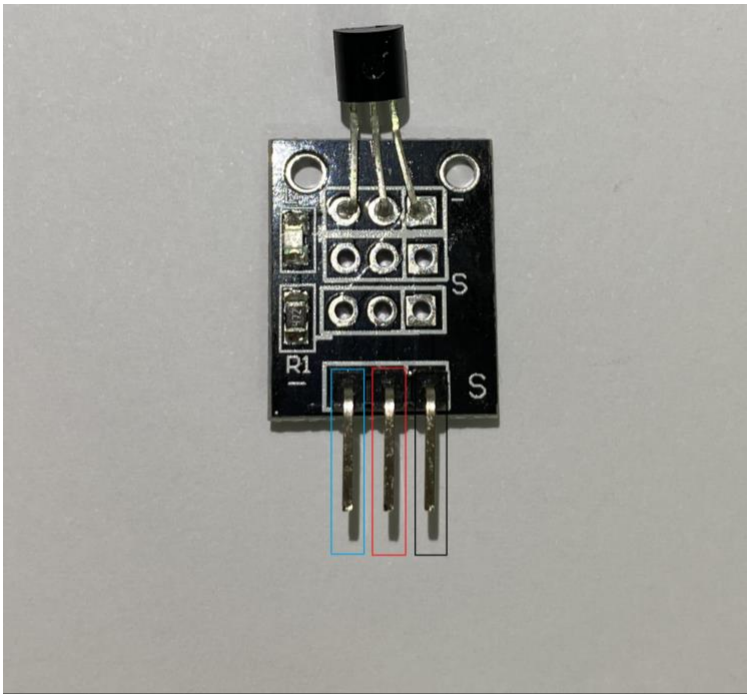

Figure 4: I/O part with negative, positive and signal pins indicated

Figure 5: I/O part with negative, positive and signal pins indicated

Wiring instructions

a) Plug in the DS18B20 temperature sensor to io4 on the E32W board.

b) Make sure the (-ve) pin are on the GND (outer) side of the I/O ports. If you are using the sensor with wires, make sure the connections are as the one shown in the figure below.

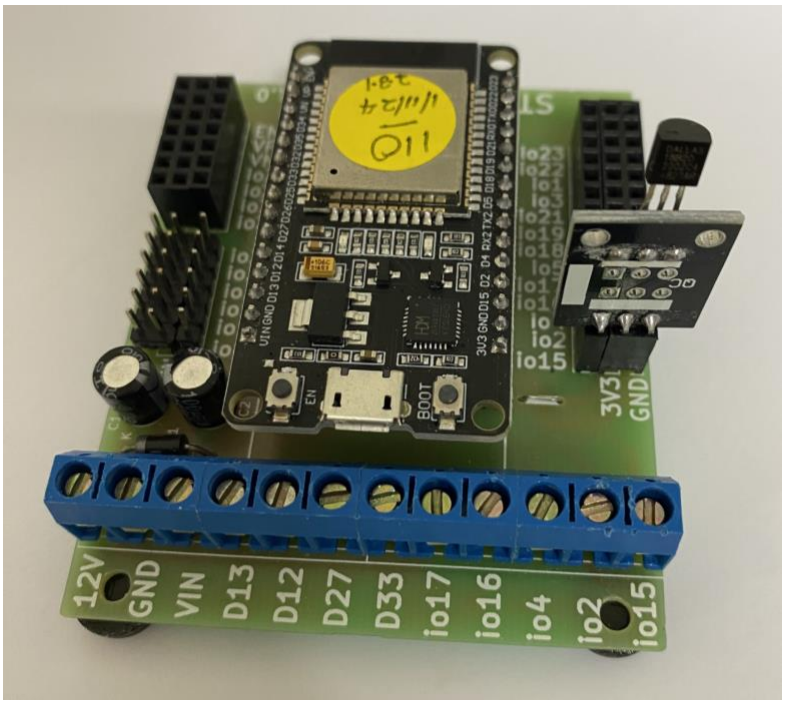

Figure 6: Circuit board connection with I/O part (side view)

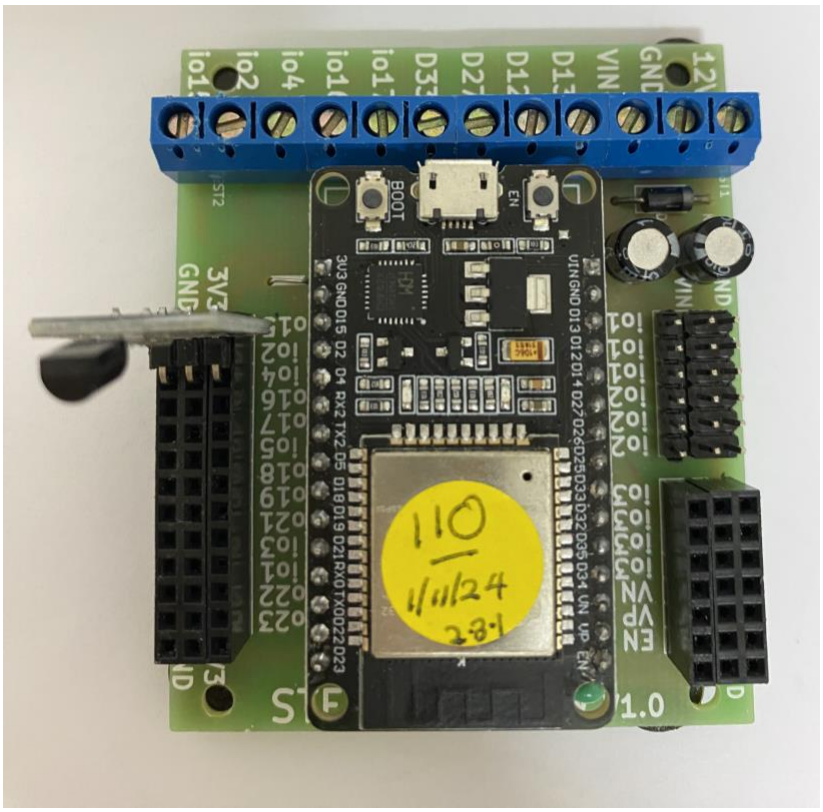

Figure 7: Circuit board connection with I/O part (top view)

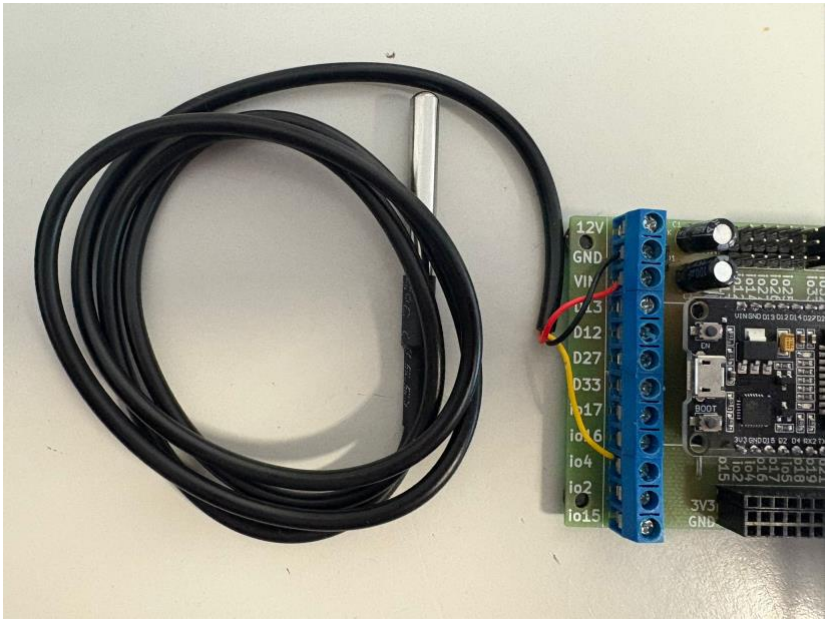

Figure 8: Circuit board connection with I/O part (top view)

Temperature Data Monitoring and Storage

And then, we want to display the real-time temperature around the DS18B20 sensor.

Program the Circuit

HTML

The main purpose of this code is to build a web interface for displaying real-time temperature data collected by the sensor. Specifically, it consists of the following parts:

1.Title and Current Temperature Display
A prominent heading at the top of the page, plus a text area that updates in real time to show the current temperature value.

2.Temperature Line Chart
An SVG element renders the coordinate axes—horizontal and vertical lines with tick marks and labels—and dynamically plots a polyline on it to visually reflect how temperature changes over time.

3.Style and Script Inclusion

An external stylesheet is linked to style the page layout and typography, and an external JavaScript file is loaded to periodically fetch temperature data and update both the numeric display and the line chart.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DS18B20 Temperature Monitor</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div style="text-align:center">
        <h1><font id="Title" color="darkblue">Temperature Sensor Graph</font></h1>
<div class="a">
            <td><font id="temperature">Current Temperature: </font></td>
        </div>
        <br>
        <svg height="350" width="600">
            <polyline id="temperatureGraph" style="fill: none; stroke: darkblue; stroke-width: 3" />
```

```
<text x="95" y="315" fill="black">0</text>
        <text x="70" y="50" fill="black">50</text>
        <text x="70" y="100" fill="black">40</text>
        <text x="70" y="150" fill="black">30</text>
        <text x="70" y="200" fill="black">20</text>
        <text x="70" y="250" fill="black">10</text>
        <text x="320" y="340" fill="darkblue">Time</text>
        <text x="0" y="150" fill="darkblue" id="yTitleOne">Temperature</text>
        <text x="0" y="170" fill="darkblue" id="yTitleTwo">°  C</text>
        <line x1="100" y1="300" x2="600" y2="300" stroke="black" />
        <line x1="100" y1="0" x2="100" y2="300" stroke="black" />
      </svg>
   </div>
   <script src="temperature.js"></script>
</body>
</html>
```

CSS

```
div.a {
    font-size: 25px;
}
```

JavaScript:

Next, we will write the JavaScript code as follows:

1. Initialize the Drawing and Data Storage Environment

The code first defines the canvas dimensions, the horizontal spacing for each sample point, and the maximum number of points that can fit on the canvas.

It also prepares two empty arrays: one to store the temperature values and another for the corresponding polyline coordinates. This provides a unified structure and container for later data collection and visualization.

2. Periodically Collect Temperature and Update the Current Reading

Within an infinite loop, the system triggers a temperature reading every 1 second.

The latest temperature is displayed in the "Current Temperature" section of the webpage. If the reading is valid, it is stored in the temperature array for future use in drawing the historical curve.

3. Maintain a Fixed-Length History Queue

To keep the polyline graph within the fixed canvas width, the oldest temperature value is removed once the number of collected points exceeds the canvas limit.

This creates a “sliding window” effect: the graph moves forward in time, older data gets pushed out, and the graph always maintains a fixed, scrollable width.

4. Convert Temperature Sequence into Polyline Coordinates

For each temperature value in the queue, the code calculates its vertical position on the canvas (the direction depends on the coordinate system settings).

These coordinates are joined into a string and assigned to the `points` attribute of the SVG polyline element, allowing the browser to redraw the graph instantly.

5. Create a Continuous Loop of “Collect → Store → Draw”

The main loop keeps running, constantly repeating the three steps: reading the temperature, updating the value, and updating the graph.

As a result, the webpage simultaneously shows the real-time temperature value and a scrolling polyline chart that reflects the surrounding temperature changes over time.

```
var Height = 215;
var Width = 500;
var StartingWidth = 100;
var widthPerSample = 5;
var maxWidthSample = Width / widthPerSample;
var i = 0;
var temperature;
var temperatureValues = [];
var temperatureElement;
var updatedTemperaturePoints;
var temperaturePoints = [];


async function fetchTemperature() {
    temperature = DS18X20_ln(tempSensor);
    document.getElementById('temperature').innerHTML = "Current Temperature: " + temperature + " °C";
    if (isNaN(parseFloat(temperature)))
        return;
    temperature = parseFloat(temperature);
    temperatureValues.push(temperature);
    console.log(temperature)
    updateGraph();
}
```

```
function updateGraph() {
    if (temperatureValues.length > maxWidthSample) {
        temperaturePoints = [];
        temperatureValues.shift();
        var A = 0;
        var B = 100;
        while (A < maxWidthSample) { // This condition avoids overflow
            temperaturePoints[A] = (B + "," + (Height - temperatureValues[A]));
            A++;
            B += 5;
        }
    } else {
        temperaturePoints[i] = (StartingWidth + "," + (Height - temperatureValues[i]));
        StartingWidth += 5;
        i++;
    }
    temperatureElement = document.getElementById('temperatureGraph');
    updatedTemperaturePoints = temperaturePoints.join(" ");
    temperatureElement.setAttribute('points', updatedTemperaturePoints);
}
```

```
// Update temperature every second
async function loop() {
    while (true) {
        await new Promise(resolve => setTimeout(resolve, 1000));
        fetchTemperature();
    }
}

loop();
```

Expected Result

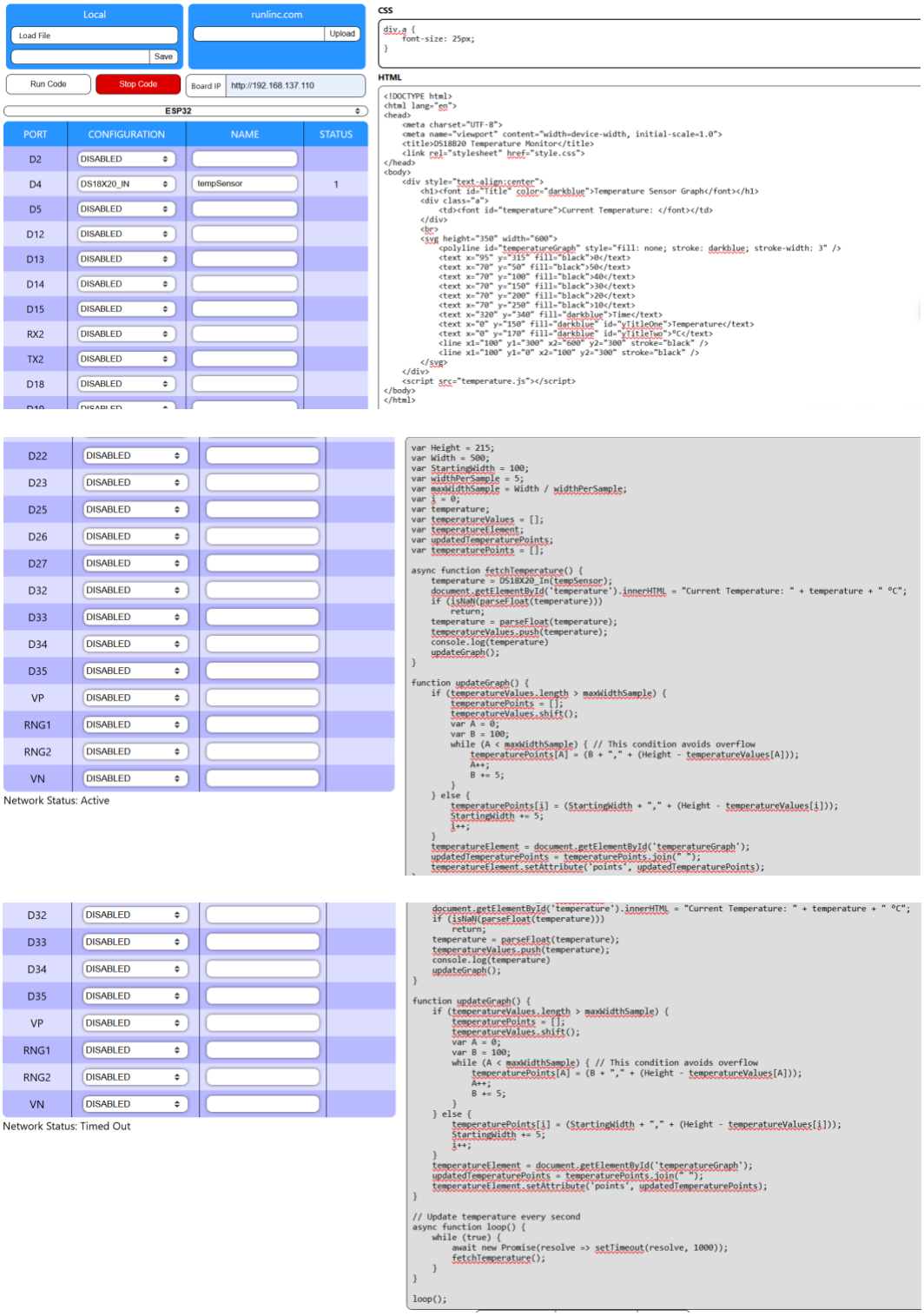

Figure 9: Expect runlinc result screenshot

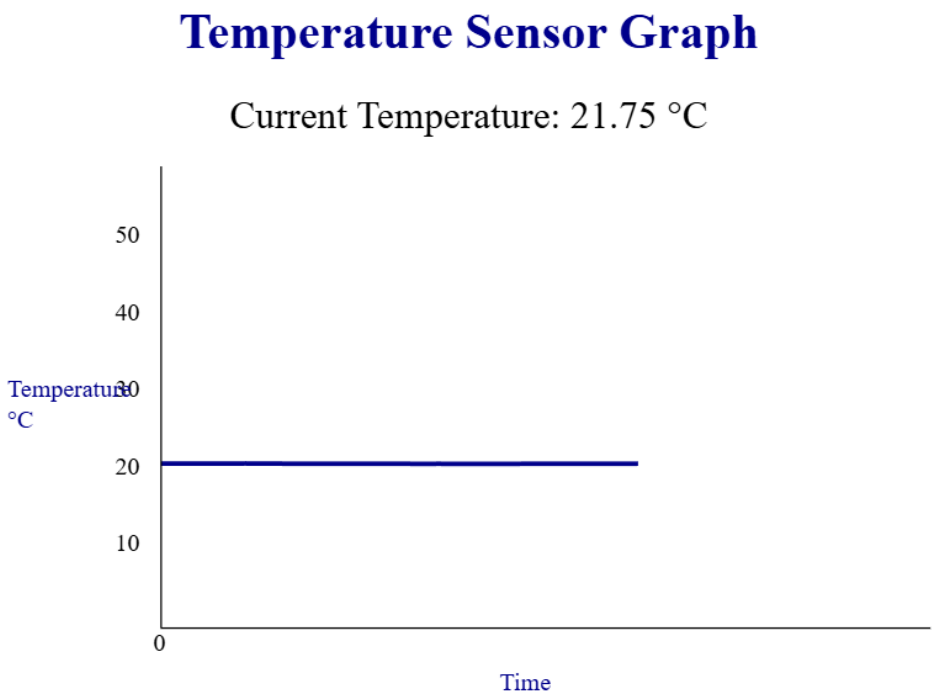

Figure 10 Expect webpage screenshot

Summary

Project I14 demonstrates how a single DS18B20 sensor can be used with runlinc to display and graph environmental data in real time—all within the browser on an ESP32 chip. The two-step approach gradually introduces newcomers to sensor data acquisition before adding visualisation techniques.