runlinc Project : Strain Gauge (E32W Version)

Contents

Introduction

Problem

Strain Gauges are good engineering tools which can detect strain on objects even in um. When the strain on the structure changes, the gauge's resistance will change accordingly. Can we make an electronic strain gauge with the gauge and the STEMSEL E32W chip?

Background

Strain gauge (or strain gage) is a device which being used to measure the strain applied on an object. It was invented in 1938. Strain gauge we use is like a piece of chip with patterned metal and flexible backing. You can see several metal "lines" on the gauge surface when you take a closer look. When you attach the piece of chip on a surface, and the object is bended towards one side which makes the chip stretch (a stretch in um will work) on the direction parallel to the metal line's direction, the chip's resistance will become higher. If the object is bended the other direction, which put stress onto the chip, the chip's resistance will become lower. Strain gauges are being used for many applications, from scales to structural health monitoring and so on.

Ideas

The most common way to use a strain gauge with a micro chip is connecting it through an HX711 amplifier. The HX711 is designed for measuring the output from strain gauges and converting it into a digital signal that can be processed by a microcontroller. We can make it work by connecting it to the STEMSEL E32W board and then take the signal from the HX711 and output it onto a browser page.

Plan

We will prepare a strain gauge, some resistors for connection, an HX711 amplifier, a STEMSEL E32W chip and an electronic device to connect with the chip. We will connect the strain gauge to the amplifier, then connect the amplifier to the microcontroller as input. After coding we will output data from the microcontroller to the browser screen.

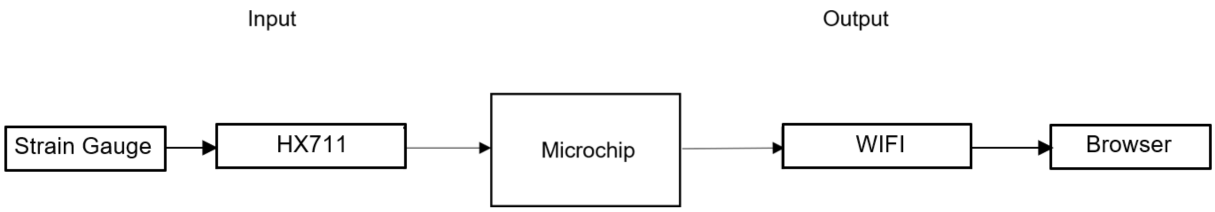


Figure 1: Block diagram of Microchip input and output

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

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 D18, name it HX711Data and set it as HX711_IN.

In our circuit design, we will be using an HX711 amplifier. We happen to have these in our kits, so these can be used on our circuit design, as per the plan.

TX2	DISABLED		
D18	HX711_IN	HX711Data	1
D19	DISABLED		

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).

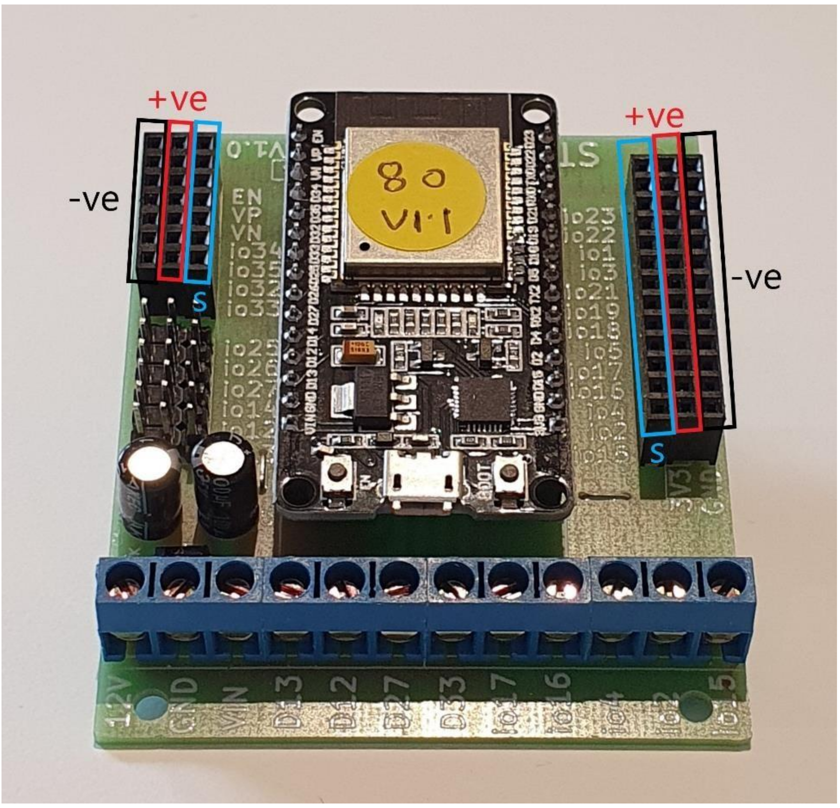


Figure 3: Negative, Positive and Signal port on the E32W board

Wiring instructions

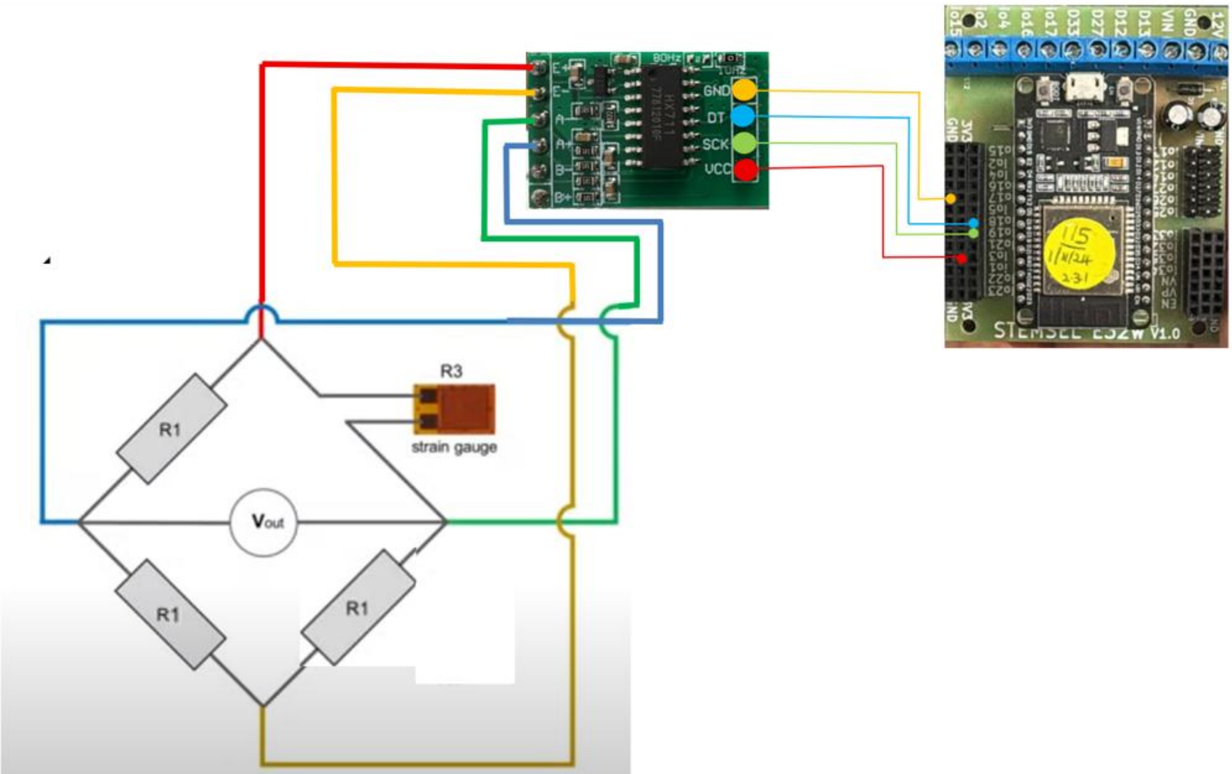


Figure 4: Circuit Diagram

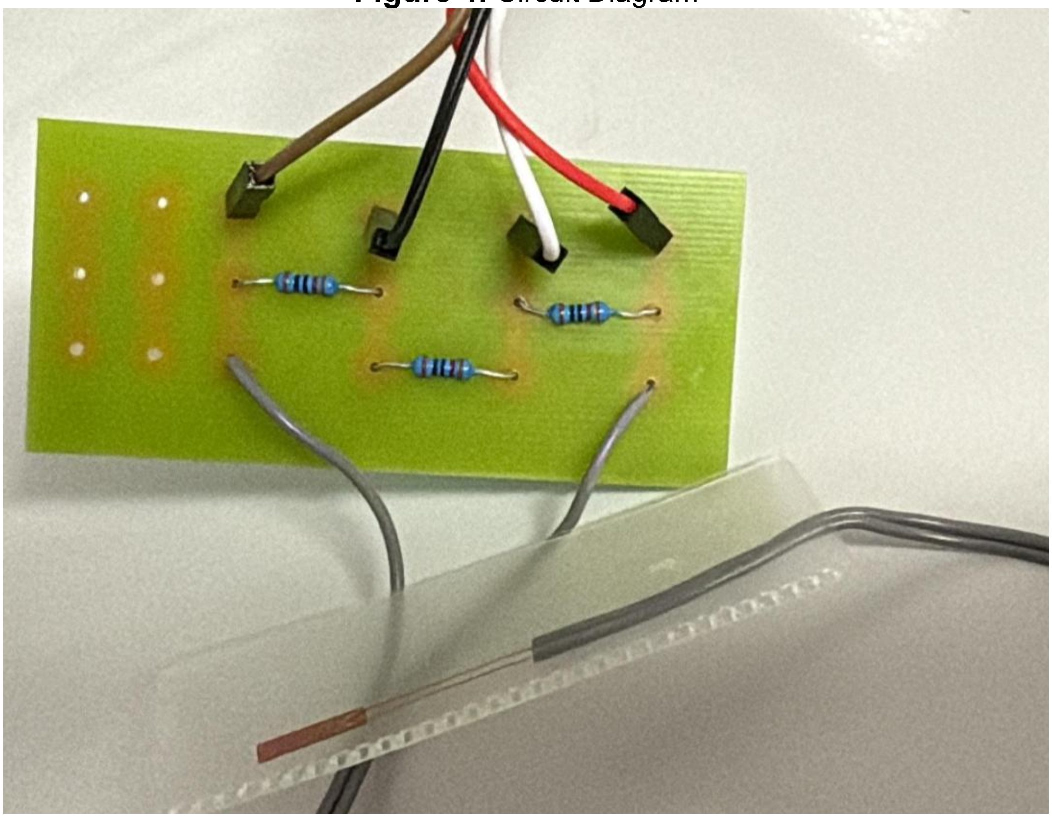


Figure 5: corresponding connection of the resistor, gauge and wires to the HX711 using a PCB [[(grey wires are connected to the gauge)

Refer to the circuit diagram to connect the components. The 'V_{out}' is a symbol of the output voltage by the circuit containing 3 'R1's and the strain gauge (R3). The chosen 'R1's' resistance should be as close to the resistance of the strain gauge as possible, buy 20 or more resistors with resistance close to the gauge and test them to find the closest one. The lines in the graph are wires, so you can just connect them and make a square correspondingly, then connect the rest of the wires, which are the colored lines in the graph. Please note that the connection between hx711 and the E32W chip need us to connect the wires to not just different ports, but also different rows. The 'GND' pin on hx711 need to be connected to -ve row, 'DT' and 'SCK' need to be connected to signal row (the one on the other side of GND next to 3V3) and the 'VCC' pin need to be connected to +ve row.

Part C: Program the Circuit

Now we can start to program the functions to use the sensor and display the data on our screen.

For **CSS** type in the following code:

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

p2 {
  color: blue;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 30px;
}

p {
  font-family: Arial, Helvetica, sans-serif;
}

h1 {
  font-family: Arial, Helvetica, sans-serif;
}
```

For **HTML** type in the following code, including a simple data display and a data graph canvas:

```
<p>Raw Strain Sensor: <font id="Message"></font>
<br>
<button onclick="zero()">ZERO</button>
<font id="ZeroValue"></font>
<br>
Scale Factor: <input type="text" id="InputText" value="0.000851">
<br>
</p>
```

```
<p2><span id="StrainD"></span></p2>
<div style="text-align:center">
   <h1 style="color: darkblue;">Pipe Strain Graph</h1>
   <div>
      <span id="StrainD"></span><br>
   </div>
   <br>
   <svg height="350" width="600">
      <polyline id="tempGraph" style="fill: none; stroke: darkred; stroke-width:3" />
      <text x="95" y="315" fill="black">0</text>
      <text x="65" y="15" fill="black">1000</text>
      <text x="70" y="72" fill="black">800</text>
      <text x="70" y="130" fill="black">600</text>
      <text x="70" y="189" fill="black">400</text>
      <text x="70" y="247" fill="black">200</text>
      <text x="320" y="340" fill="darkblue">Time</text>
      <text x="10" y="150" fill="darkblue" transform="rotate(-90 20,150)">Pipe Strain
(umm/mm)</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>
```

Put the following code into the **JavaScript** window:

```
ZValue = 0;
value = 0;

function zero() {
ZValue = parseFloat(rawSensor);
}

function updateStrain() {
  rawSensor = HX711_In( HX711Data );
  value = parseFloat(rawSensor) - ZValue;

  var ScaleVtxt = document.getElementById("InputText").value;
  ScaleV =  parseFloat( ScaleVtxt );
  Avalue = parseInt( value * ScaleV );

  document.getElementById("Message").innerHTML = rawSensor;
  document.getElementById("ZeroValue").innerHTML = ZValue;
  document.getElementById("StrainD").innerHTML = Avalue;

  let strainVal = Avalue;
  document.getElementById("StrainD").innerText = strainVal ;
  // Getting sensor value and pushing it to the corresponding array
  var currentTemp = parseInt(strainVal / 10 );  // Scaled to graph
  console.log(`${currentTemp}`);

  // Push current sensor value if it exists
  if (!isNaN(currentTemp)) {
     tempValues.push(currentTemp);

     // Update the graph and display values
     updatedGraph();
     displayValues();
  }

}
```

```
// Graph display parameters (You'll still need to alter the position of the y axis numbers
manually)
var Height = 300; // Height of the graph
var Width = 500; // Width of the graph
var MaxValue = 103; // The maximum value that you want the graph to have within its
bounds
var StartingWidth = 100; // The starting width of the graph
var widthPerSample = 5; // The horizontal distance between 2 graph points
var maxWidthSample = Width / widthPerSample;
var i = 0;

// Sensor values + points
var currentTemp;

var tempValues = [];
var tempPoints = [];

// Function to update the graph
function updatedGraph(){
   // If the amount of points is greater than the width of the graph
   if (tempValues.length > maxWidthSample) {
      // Clear the temperature points
      tempPoints = [];
      tempValues.shift();

      // Set the new points
      for (var j = 0; j < maxWidthSample; j++) {
         tempPoints[j] = ((StartingWidth - 5 * i) + widthPerSample * j) + "," + (Height -
tempValues[j] * (Height / MaxValue));
      }
   }
   else {
      // Append a point to the array
      tempPoints[i] = StartingWidth + "," + (Height - tempValues[i] * (Height /
MaxValue));

      StartingWidth += 5;
      i++;
   }

   // Set the points on the polyline
   var tempGraph = document.getElementById('tempGraph');
   var updatedTempPoints = tempPoints.join(" ");
   tempGraph.setAttribute('points', updatedTempPoints);
}

// Simple function to display the value of the strain
function displayValues(){
   document.getElementById('StrainD').innerHTML = "Current Pipe Strain: " + Avalue +
" umm/mm";
}
```

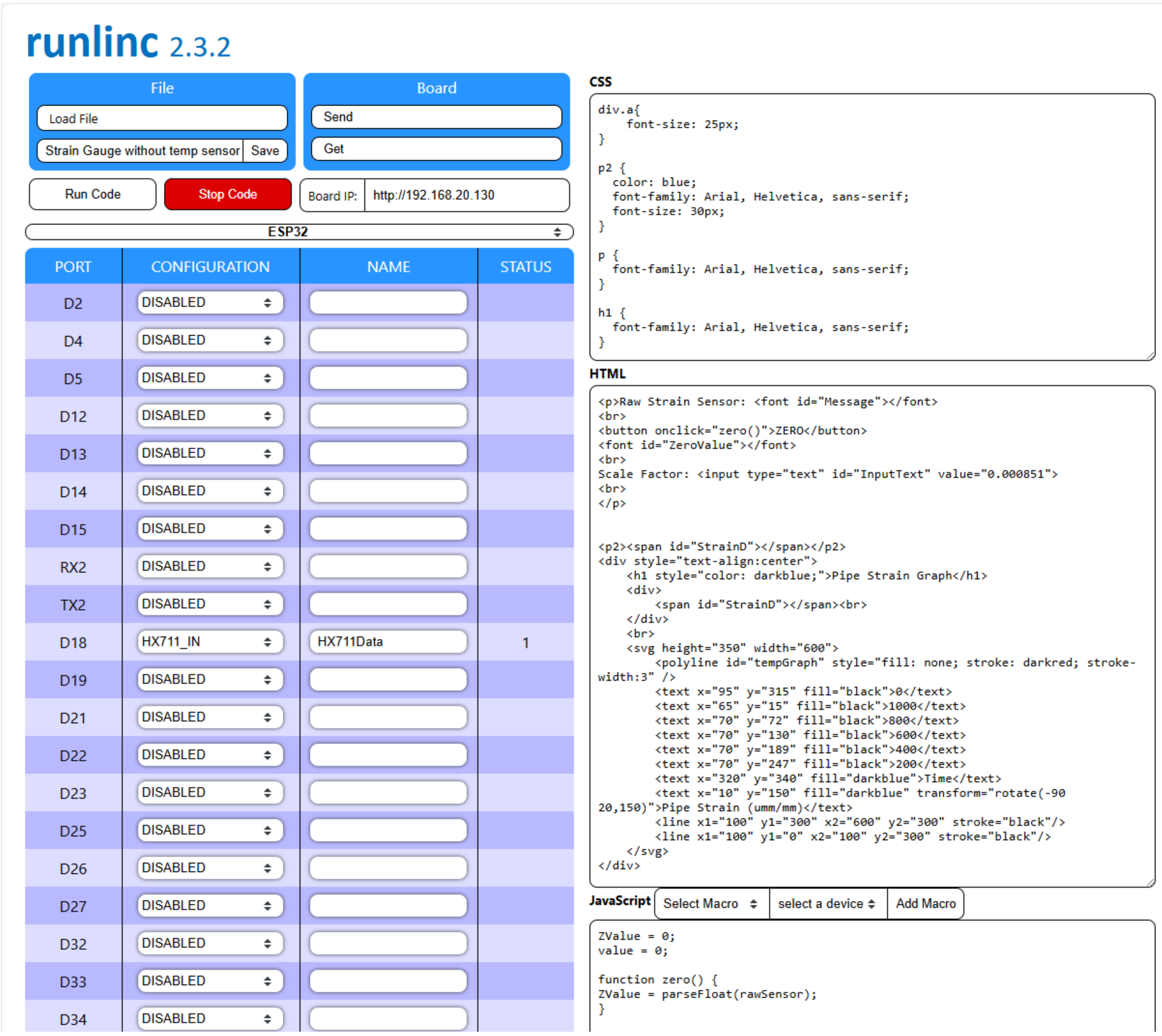
The code here includes 4 parts, the 1st part is the function for the ‘ZERO’ button, which will add or minus a value to the current reading, to zero the readings on the screen. The 2nd part is the ‘update strain’ function. It includes taking the reading from the sensor, and some calculations to distribute the current reading onto the screen, and the calling of the 3rd and 4th parts. The 3rd part gives parameters for the drawing of the line on the graph, and automatically updates the graph. The 4th part simply display the strain value on the screen.

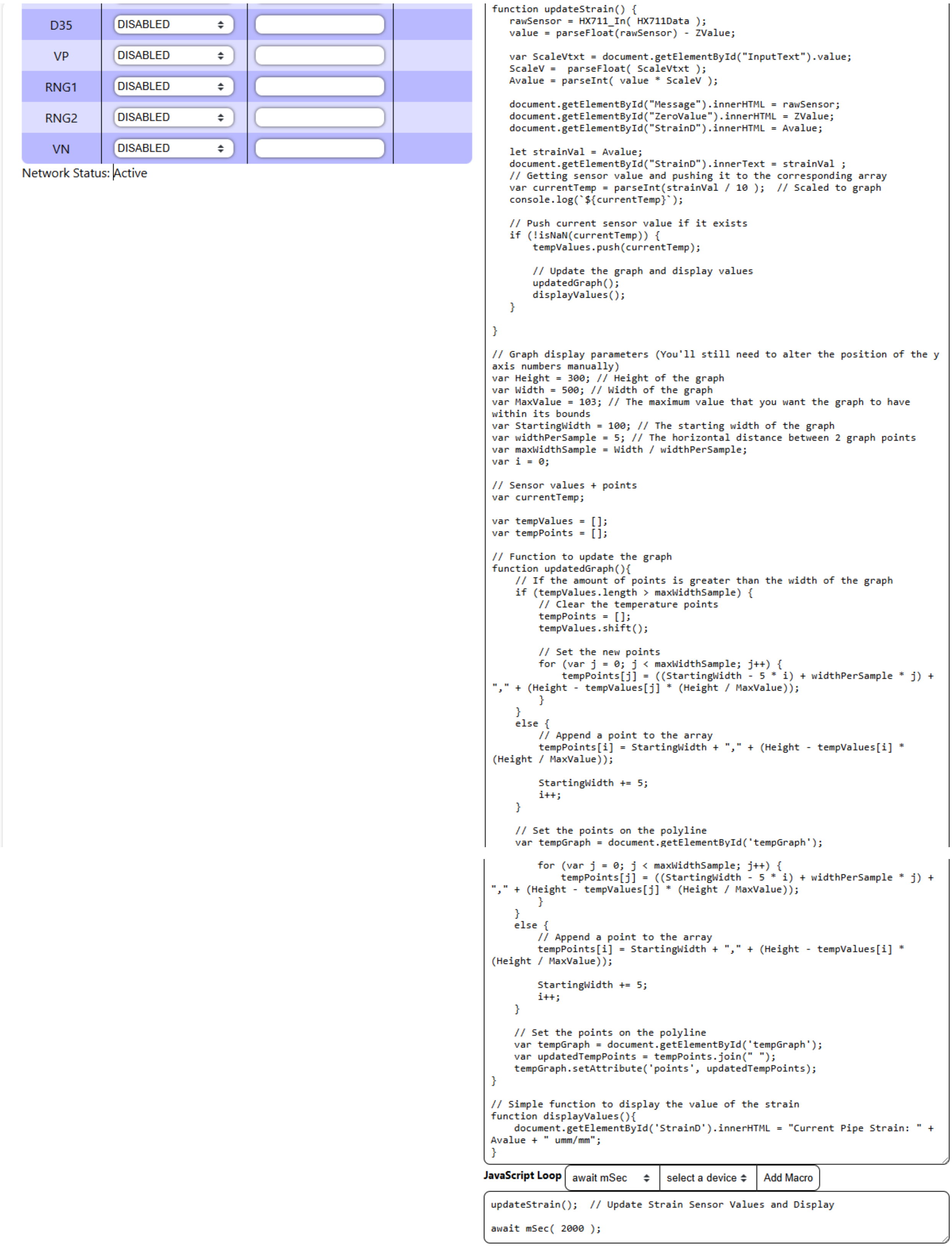
Put the following code into the **JavaScript Loop** window to make the program run and update the values on the screen every 2 seconds:

```
updateStrain();  // Update Strain Sensor Values and Display

await mSec( 2000 );
```

The final runlinc IDE page should look like this:





D35	DISABLED	
VP	DISABLED	
RNG1	DISABLED	
RNG2	DISABLED	
VN	DISABLED	

Network Status: Active

```
function updateStrain() {
   rawSensor = HX711_In( HX711Data );
   value = parseFloat(rawSensor) - ZValue;

   var ScaleVtxt = document.getElementById("InputText").value;
   ScaleV =  parseFloat( ScaleVtxt );
   Avalue = parseInt( value * ScaleV );

   document.getElementById("Message").innerHTML = rawSensor;
   document.getElementById("ZeroValue").innerHTML = ZValue;
   document.getElementById("StrainD").innerHTML = Avalue;

   let strainVal = Avalue;
   document.getElementById("StrainD").innerText = strainVal ;
   // Getting sensor value and pushing it to the corresponding array
   var currentTemp = parseInt(strainVal / 10 );  // Scaled to graph
   console.log(`${currentTemp}`);

   // Push current sensor value if it exists
   if (!isNaN(currentTemp)) {
       tempValues.push(currentTemp);

       // Update the graph and display values
       updatedGraph();
       displayValues();
   }

}

// Graph display parameters (You'll still need to alter the position of the y
axis numbers manually)
var Height = 300; // Height of the graph
var Width = 500; // Width of the graph
var MaxValue = 103; // The maximum value that you want the graph to have
within its bounds
var StartingWidth = 100; // The starting width of the graph
var widthPerSample = 5; // The horizontal distance between 2 graph points
var maxWidthSample = Width / widthPerSample;
var i = 0;

// Sensor values + points
var currentTemp;

var tempValues = [];
var tempPoints = [];

// Function to update the graph
function updatedGraph(){
    // If the amount of points is greater than the width of the graph
    if (tempValues.length > maxWidthSample) {
        // Clear the temperature points
        tempPoints = [];
        tempValues.shift();

        // Set the new points
        for (var j = 0; j < maxWidthSample; j++) {
            tempPoints[j] = ((StartingWidth - 5 * i) + widthPerSample * j) +
"," + (Height - tempValues[j] * (Height / MaxValue));
        }
    }
    else {
        // Append a point to the array
        tempPoints[i] = StartingWidth + "," + (Height - tempValues[i] *
(Height / MaxValue));

        StartingWidth += 5;
        i++;
    }

    // Set the points on the polyline
    var tempGraph = document.getElementById('tempGraph');
```

```
        for (var j = 0; j < maxWidthSample; j++) {
            tempPoints[j] = ((StartingWidth - 5 * i) + widthPerSample * j) +
"," + (Height - tempValues[j] * (Height / MaxValue));
        }
    }
    else {
        // Append a point to the array
        tempPoints[i] = StartingWidth + "," + (Height - tempValues[i] *
(Height / MaxValue));

        StartingWidth += 5;
        i++;
    }

    // Set the points on the polyline
    var tempGraph = document.getElementById('tempGraph');
    var updatedTempPoints = tempPoints.join(" ");
    tempGraph.setAttribute('points', updatedTempPoints);
}

// Simple function to display the value of the strain
function displayValues(){
    document.getElementById('StrainD').innerHTML = "Current Pipe Strain: " +
Avalue + " umm/mm";
}
```

JavaScript Loop | await mSec | select a device | Add Macro

```
updateStrain();  // Update Strain Sensor Values and Display

await mSec( 2000 );
```

Figure 5: runlinc webpage screenshot

Part D: The Working Project

If you successfully built the circuit and the program correctly, it should act as what this part describe:

After pressing 'run code', the following screen will show up. If you click on the 'ZERO' button, the reading on the graph will return to zero.

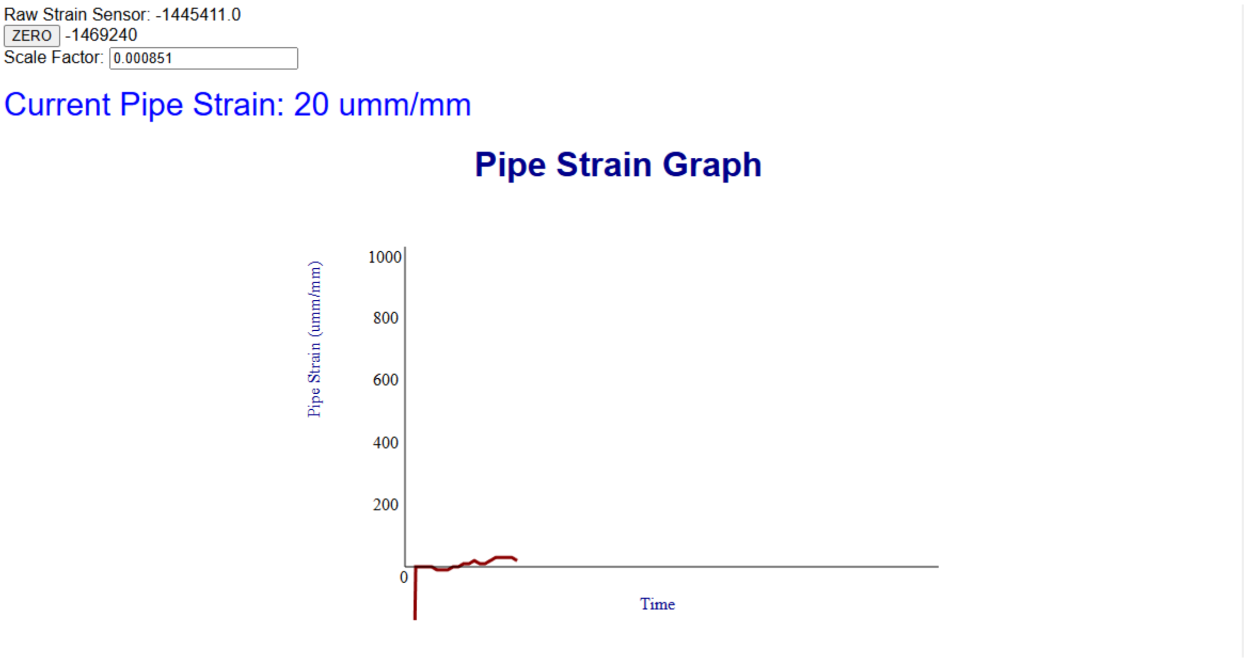


Figure 6: working reading screen

When the project is running, a small strain change to the strain gauge will make the reading change accordingly.

Summary

Strain gauges are being widely used in our lives, and we can use them with the E32W board to make a strain detector as well. By simply connecting the amplifier and the gauge to the board and using the runlinc IDE's built-in function, we can read the strain from the gauge easily. We can also make some other functions, such as a strain graph that shows the current strain, which updates over time if we want, and even make a connected system which reacts based on the strain gauge's reading…