Nathan Roe
EN.606.715.81.SP23
2023-02-20

Project 2 - Temperature Logging

The first project for this class involves using an Arduino log temperature data on a certain time interval. A full demonstration video can be found on Youtube HERE and the source code for my project is included in the document, or available on Github HERE.

Requirements

- Calibrate a temperature sensor connected to an Arduino
    - Derived Requirement - Calibrate Thermistor
        - Met by measuring the resistance of my thermistor at three different temperatures to derive Steinhart-Hart coefficients
- Round Robin with Interrupts design
    - Derived Requirement - Logging should be triggered by interrupt in Round Robin w/ Interrupts design
        - Met by adding a timer interrupt that triggers the logger, while the thermistor read operation is done in the Loop function
- Get the Arduino to capture the temperature and convert to Fahrenheit
    - Derived Requirement - Log temperature in Fahrenheit
        - Met by converting Kelvin temperature from Steinhart-Hart coefficient to Farenheit and logging that value
- Record the temperature at a periodic rate of around 10s at room temperature, then in the fridge for 5 minutes, then remove from the fridge and keep recording
    - Derived Requirement - Log temperature at periodic rate
        - Met by logging temperature at 2x the longest timer interrupt for Arduino, which corresponds to a time of around 8.4 seconds
    - Derived Requirement - Log temperature at several locations of varying temperatures to demonstrate calibration
        - Met by logging at room temperature, then fridge, freezer, and heater vent temperatures
- Transmit the time and temp across a Serial bus like USB to your Host
    - Derived Requirement - Log values across Serial Interface
        - Met by transmitting across Serial bus to both PC in the Arduino serial monitor as well as an OpenLog logger to allow disconnected operation from PC
- Export as a comma separated value file, read into a spreadsheet program and plot the temperature vs time
    - Derived Requirement - Save file with values in CSV format containing time and temperature
    - Met by writing output as two comma separated values and plotting results in the Results section

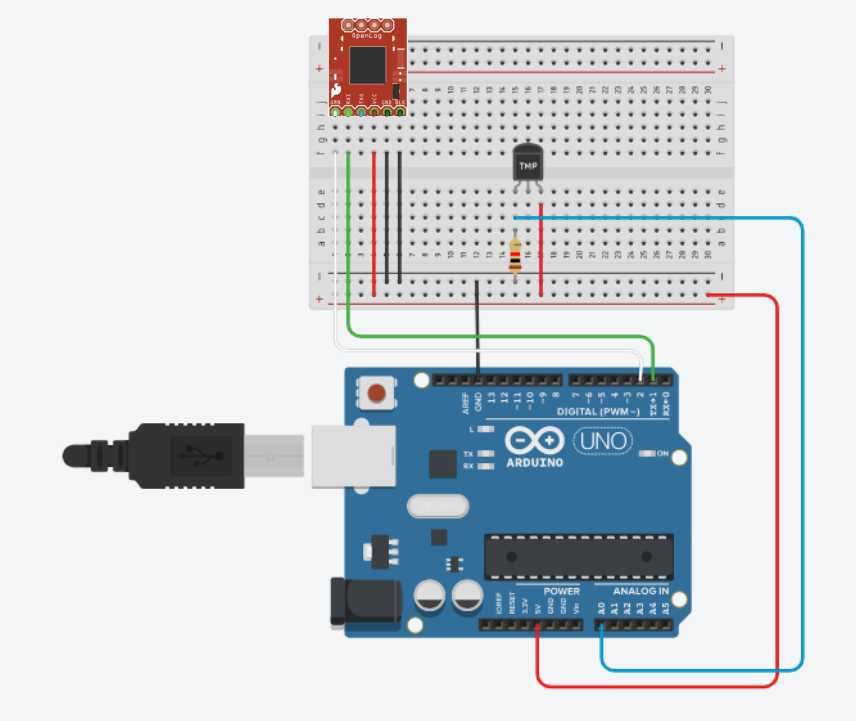
Nathan Roe
EN.606.715.81.SP23
2023-02-20

Hardware Design

The hardware design for my project consists of two main component sets, the thermistor for measuring temperature, and the SD card logger for logging data across the Serial bus.

The first consists of a thermistor and resistor connected in series to create a voltage divider. The thermistor measured 13 kΩ at room temperature, so I used a 10 kΩ resistor so the voltage was distributed roughly in half. The 5 V pin was connected to the thermistor, Gnd was connected to the resistor, and the Analog 0 pin was connected at the junction to measure the voltage drop across the thermistor.

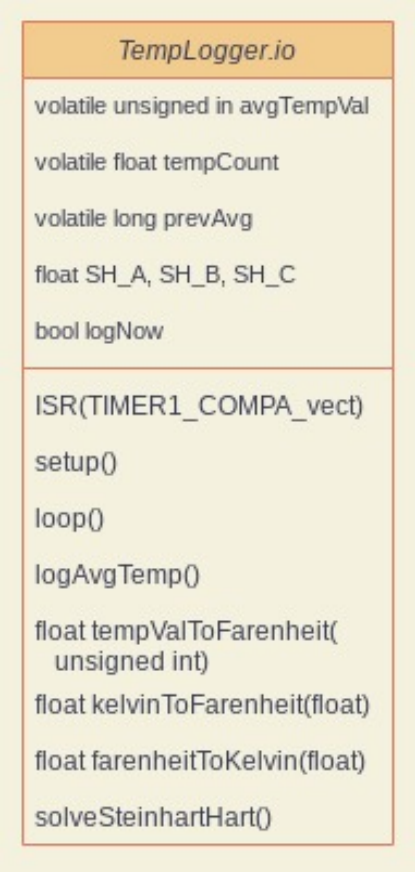
The second consists of an micro-SD card reader OpenLog component that records the Serial bus. The 5 V pin was connected to the logger's VCC pin, and Gnd was connected to the logger's GND and BLK pins. The Arduino Tx pin was then connected to the logger's Rx pin to allow the logger to receive and record Serial bus communications. I also connected the logger's GRN pin to a Digital pin 2 of the Arduino; having the pin set to high and then pulling it low allowed me to reset the logger, creating a new logging file, which was useful for testing.



Nathan Roe
EN.606.715.81.SP23
2023-02-20

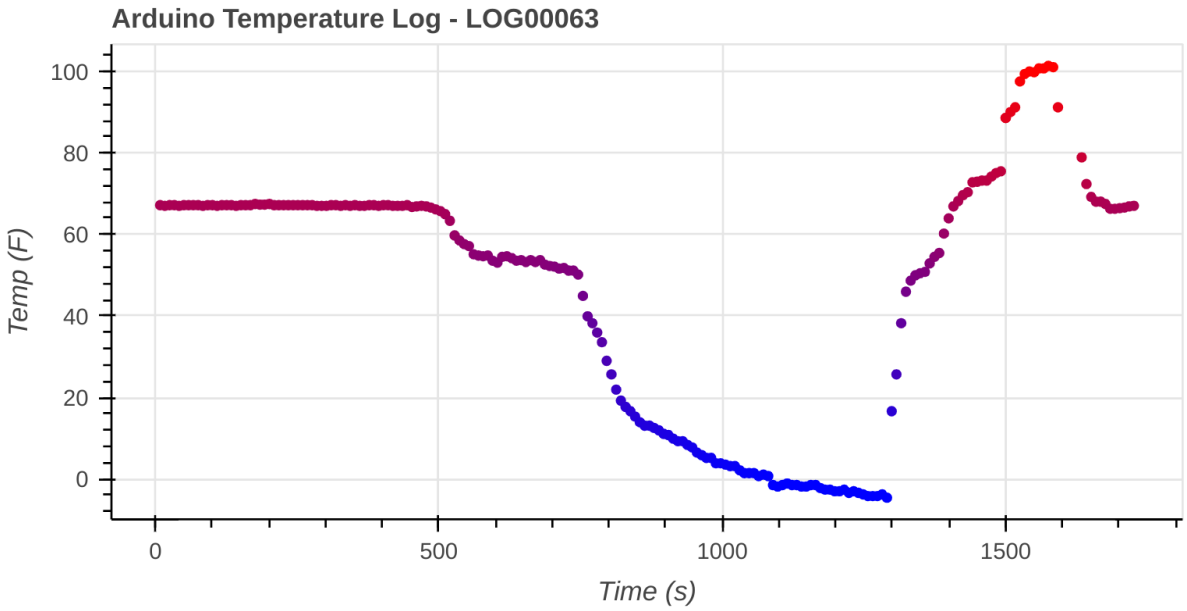
Software Design

The software design for my program is fairly straightforward. I used a Round Robin with Interrupts design as specified by the project requirements. In the *loop()* function, the thermistor is repeatedly checked and this value is used to update current variables for calculating a running average temperature since the last log. I also set up an interrupt service routine (ISR) that calls the *logAvgTemp* function, which writes the time and average temperature to the Serial bus before resetting logging. The ISR is set to trigger based on a value comparison between the timer time and a 4-byte value, in this case 0xFFFF for the maximum allowed time. This corresponds to about 4.2 seconds, so I added a boolean that flips each time the ISR is called, and the logging is only triggered when this boolean is True, doubling the time between logs to roughly 8.4 seconds. The rest of the functions in the Sketch are temperature utility functions used for converting between Analog-Read value, temperature in Kelvin, and temperature in Farenheit, or for setting thermistor calibration (*solveSteinhartHart*). I added the calibration function to the Sketch to allow me to quickly iterate on resistance/temperature pairs for three measurements to properly calibrate the setup.



Nathan Roe
EN.606.715.81.SP23
2023-02-20

Results



The plot above shows the results of logging from my project demo. The logger correctly measures the room temperature of my house, 67 °F. It then begins to drop when placed in the fridge making it to ~ 48 °F before being moved to the freezer. In the freezer, it settles at about -4 °F before being removed and placed in front of the heat vent. The temperature jumps as the air turns on, and then again as the heat kicks in, before shutting off and returning to room temperature.

```
/*
 * Timer Resources - for future reference
 * https://microcontrollerslab.com/arduino-timer-interrupts-tutorial/
 * https://www.best-microcontroller-projects.com/arduino-timer-interrupt.html
 */
#include <SoftwareSerial.h>

// I/O Data
const int thermistorPin = A0;
//const int rxPin = 0; // Currently unused
const int txPin = 1;
const int logResetPin = 2;
//const int interruptPin = 3;

// Temperature Data
volatile unsigned int avgTempVal = 0;
volatile float tempCount = 0;
volatile long prevAvg = 0;
const int KtoC = -273.15;
float SH_A = 0;
float SH_B = 0;
float SH_C = 0;

// Voltage Data
const float inputVoltage = 5.0;
const float resistanceUsed_ohm = 10000.0;

// Interrupt Data
bool logNow = false;
ISR(TIMER1_COMPA_vect) {
    logAvgTemp();
}

// Set up interrupt info and calculate coffecients for temp
// sensor callibration allowing for easy updates in solveSteinhartHart()
void setup() {
    // put your setup code here, to run once:
    Serial.begin(9600);
    pinMode(logResetPin, OUTPUT);
    // Default state high, low to reset logging
    digitalWrite(logResetPin, HIGH);

    Serial.print("Time (s), Temp (F)\n");

    solveSteinhartHart();

    /*
     * Uses Compare-Match interrupt with max compare value FFFF
     * This is equivalent to doing overflow - overflow would have
     * TIMER1_COMPA_vect replaced with TIMER1_OVF_vect
     * and OCIE1A replaced with TOIE1
     */
    // Reset Timer 1 Registers
    TCCR1A = 0;
    TCCR1B = 0;
    // Set Timer 1 compare value to max
```

```
    OCR1A = 0xFFFF;
    // Set Timer 1 to prescaler 1024
    TCCR1B = (1<<CS12) | (1<<CS10);
    // Set for comparison to OCR1A
    TCCR1B = TCCR1B | (1<<WGM12);
    // Set Timer 1 to use compare match
    TIMSK1 = (1<<OCIE1A);
}

// Continuously update rolling pin read average
void loop() {
    // Keep rolling average of input pin read value
    prevAvg = avgTempVal * tempCount;
    unsigned int newTempVal = analogRead(thermistorPin);
    // Protect against interrupt causing div-by-0 error
    noInterrupts();
    avgTempVal = (prevAvg + newTempVal) / ++tempCount;
    interrupts();
}

// Log current average temp to Serial bus
void logAvgTemp()
{
    if (logNow){
        Serial.print(millis()/1000.);
        Serial.print(", ");
        Serial.print(tempValToFarenheit(avgTempVal));
        Serial.print("\n");
        avgTempVal = 0;
        tempCount = 0;
        prevAvg = 0;
    }
    logNow = !logNow;
}

// Convert an Analog Pin read value to a temperature
float tempValToFarenheit(unsigned int tempVal)
{
    float measVoltage = ((1023.0 - tempVal)/ 1023.0) * inputVoltage;
    float measReistance = (measVoltage / inputVoltage) * resistanceUsed_ohm / (1 -
(measVoltage / inputVoltage));
    // Steinhart-Hart Equation
    float lnTemp = log(measReistance);
    float inverseTempK = SH_A + (SH_B * lnTemp) + (SH_C * lnTemp * lnTemp * lnTemp);
    return kelvinToFarenheit(1 / inverseTempK);
}

// Convert Temperature from Kelvin to Farenheit
float kelvinToFarenheit(float tKelvin) {
    return ((tKelvin + KtoC) * 9 / 5.0) + 32;
}

// Convert Temperature from Farenheit to Kelvin
float farenheitToKelvin(float tFarenheit) {
    return ((tFarenheit - 32.0) * 5 / 9.0) - KtoC;
}
```

```
// Use three Temp/Resistance Pairs to calibrate thermistor
void solveSteinhartHart()
{
    // Set Calibration Pairs - Measure Resistance (Ohms) of
    // thermistor at three different Temperatures (Kelvin)
    float R1 = 5500;
    float T1 = farenheitToKelvin(98);
    float R2 = 13000;
    float T2 = farenheitToKelvin(67);
    float R3 = 27000;
    float T3 = farenheitToKelvin(40);

    // Using Cal Temps, solve for SH Coefficients
    float L1 = log(R1);
    float L2 = log(R2);
    float L3 = log(R3);
    float Y1 = 1 / T1;
    float Y2 = 1 / T2;
    float Y3 = 1 / T3;
    float gamma2 = (Y2 - Y1) / (L2 - L1);
    float gamma3 = (Y3 - Y1) / (L3 - L1);
    SH_C = ((gamma3 - gamma2) / (L3 - L2)) / (L1 + L2 + L3);
    SH_B = gamma2 - (SH_C * ((L1*L1) + (L1*L2) + (L2*L2)));
    SH_A = Y1 - ((SH_B + (L1*L1*SH_C))*L1);
}

// Reset Logger to create new log file
void resetLogger() {
    // Pull logger interrupt low
    digitalWrite(logResetPin, LOW);
    delay(10);
    digitalWrite(logResetPin, HIGH);
}
```

```
import glob
import os

import numpy as np
import pandas as pd

from bokeh.plotting import ColumnDataSource, figure, show

# Find all log files -  end in .TXT extension
file_list = glob.glob('*.TXT')

# Create plot for each file
for file in file_list:
    file_name = os.path.splitext(os.path.basename(file))[0]

    x_axis_label = 'Time (s)'
    y_axis_label = 'Temp (F)'

    # Read in CSV data
    vals = pd.read_csv(file, header=None, names=[x_axis_label, y_axis_label])

    temp_copy = vals[y_axis_label].copy(deep=True)
    temp_copy[temp_copy > 100] = 100.0
    temp_copy[temp_copy < 0] = 0.0

    # Set up color array - 0F->#0000FF, 100F->#FF0000
    reds = np.array(['0x{:02x}'.format(int(255*temp/100))
                     for temp in temp_copy])
    reds[vals[y_axis_label] >= 100] = "0xff"
    reds[vals[y_axis_label] <= 0] = "0x00"

    blues = np.array(['0x{:02x}'.format(int(255 - (255*temp/100)))
                      for temp in temp_copy])
    blues[vals[x_axis_label] <= 0] = "0xff"
    blues[vals[y_axis_label] >= 100] = "0x00"

    colors = [
        f'#{reds[idx].split("x")[1]}00{blues[idx].split("x")[1]}' for idx in
range(len(reds))]

    # Begin Bokeh plotting
    source = ColumnDataSource({'Time': vals[x_axis_label],
                               'Temp': vals[y_axis_label],
                               'colors': colors})

    TOOLTIPS = [(x_axis_label, '@Time'),
                (y_axis_label, '@Temp')]

    tools = 'hover,wheel_zoom,pan,box_zoom,reset,undo,save'

    fig = figure(title=f'Arduino Temperature Log - {file_name}',
                 toolbar_location="right",
                 aspect_ratio=2,
                 tools=tools,
                 tooltips=TOOLTIPS)
```

```
    fig.xaxis.axis_label = x_axis_label
    fig.yaxis.axis_label = y_axis_label

    fig.scatter('Time',
                'Temp',
                source=source,
                fill_color='colors',
                line_color='colors')
    show(fig)
```

```
8.38,67.18
16.76,67.03
25.14,67.18
33.52,67.18
41.9,67.03
50.28,67.18
58.66,67.18
67.04,67.18
75.42,67.18
83.8,67.03
92.18,67.18
100.56,67.18
108.94,67.03
117.32,67.18
125.7,67.18
134.08,67.18
142.46,67.03
150.84,67.18
159.22,67.18
167.6,67.18
175.98,67.47
184.36,67.32
192.74,67.32
201.12,67.47
209.5,67.18
217.88,67.18
226.26,67.18
234.64,67.18
243.02,67.18
251.4,67.18
259.78,67.18
268.16,67.18
276.54,67.18
284.92,67.03
293.3,67.03
301.68,67.03
310.06,67.18
318.44,67.18
326.82,67.03
335.2,67.18
343.58,67.03
351.96,67.18
360.34,67.03
368.72,67.03
377.1,67.18
385.48,67.18
393.86,67.03
402.24,67.18
410.62,67.18
419.0,67.03
427.38,67.03
435.76,67.03
444.14,67.18
452.52,66.74
460.9,66.89
469.28,67.03
```

```
477.66,66.89
486.04,66.59
494.42,66.16
502.8,65.72
511.18,64.98
519.56,63.36
527.94,59.76
536.32,58.54
544.7,57.62
553.08,57.16
561.46,55.13
569.84,54.82
578.22,54.66
586.6,54.82
594.98,53.55
603.36,53.07
611.74,54.5
620.12,54.66
628.5,54.19
636.88,53.55
645.26,53.71
653.64,53.23
662.02,53.71
670.4,53.23
678.78,53.71
687.16,52.59
695.54,52.27
703.92,52.11
712.3,51.62
720.68,51.78
729.06,51.14
737.44,51.14
745.82,50.15
754.2,44.92
762.58,39.91
770.96,38.23
779.34,35.93
787.72,33.56
796.1,28.99
804.48,25.67
812.86,21.93
821.24,19.22
829.62,17.68
838.0,16.63
846.38,15.3
854.76,13.92
863.14,13.08
871.52,13.08
879.9,12.52
888.28,11.94
896.66,11.07
905.04,10.78
913.42,9.89
921.8,9.28
930.18,9.28
938.56,8.37
```

```
946.94,7.75
955.32,6.48
963.7,5.84
972.08,5.18
980.46,5.18
988.84,3.85
997.22,3.85
1005.6,3.52
1013.98,3.17
1022.36,3.17
1030.74,2.14
1039.12,1.44
1047.5,1.44
1055.88,1.44
1064.26,0.72
1072.64,1.08
1081.02,0.72
1089.4,-1.48
1097.78,-1.86
1106.16,-1.48
1114.54,-1.11
1122.92,-1.48
1131.3,-1.48
1139.68,-1.86
1148.06,-1.86
1156.44,-1.48
1164.82,-1.48
1173.2,-2.24
1181.58,-2.62
1189.96,-2.62
1198.34,-3.01
1206.72,-3.01
1215.1,-2.62
1223.48,-3.4
1231.86,-3.01
1240.24,-3.4
1248.62,-3.79
1257.0,-4.19
1265.38,-4.19
1273.76,-4.19
1282.14,-3.79
1290.52,-4.59
1298.9,16.63
1307.28,25.67
1315.66,38.23
1324.04,45.96
1332.42,48.66
1340.8,49.99
1349.18,50.48
1357.56,50.81
1365.94,52.91
1374.32,54.5
1382.7,55.45
1391.08,60.21
1399.46,63.95
1407.84,66.89
```

```
1416.22,68.19
1424.6,69.64
1432.98,70.36
1441.36,72.8
1449.74,72.94
1458.12,73.22
1466.5,73.22
1474.88,74.22
1483.26,75.07
1491.64,75.5
1500.02,88.59
1508.4,90.03
1516.78,91.19
1525.16,97.55
1533.54,99.37
1541.92,99.98
1550.3,99.8
1558.68,100.74
1567.06,100.74
1575.44,101.36
1583.82,101.05
1592.2,91.19
1500.02,88.59
1634.1,78.91
1642.48,72.37
1650.86,69.21
1659.24,68.05
1667.62,68.05
1676.0,67.47
1684.38,66.3
1692.76,66.3
1701.14,66.45
1709.52,66.59
1717.9,66.89
1726.28,67.03
```