Lab 3: Arduino Temperature Sensor

Argenis Aquino, Rachel DuBois, Diego Lopez, Jonathan Sumner Department of Engineering Technology, Rochester Institute of Technology
1 Lomb Memorial Drive, Rochester NY, 14623, USA

***Abstract*—This document demonstrates how an Arduino Uno was used to record analog temperature data by connecting it to an LM61 temperature sensor. Data collection was optimized by leveraging the microcontroller's 10-bit ADC along with the MsTimer2 library and hardware interrupts.**

I. Introduction

ARDUINOS are open-source microcontrollers that provide a beginner-friendly introduction to circuit and firmware design. In this lab, the Arduino was used with an LM61 precision IC temperature sensor to collect temperature samples and plot them using MATLAB. The LM61 has a temperature range from -30°C to +100°C. The datasheet provides the transfer function used to determine the temperature:

$$V_0 = (10 \text{ mV}/°\text{C} \times T°\text{C}) + 600 \text{ mV} \tag{1}$$

II. Methodology

For P1-1 a delay was hard-coded in order to collect samples at a rate of 500 millisecond.

```
int nSmpl = 1, sample; // global variables
void setup()
{
  Serial.begin(9600); // Set baudrate to 9600
  Serial.print("\nsmpl\tADC\n"); // column headers
}
void loop()
{
  sample = analogRead(A0);// Read ADC value on pin A0
  Serial.print(nSmpl);    // Print sample number
  Serial.print('\t');     // Print tab
  Serial.println(sample); // Print sample value
  ++nSmpl;                // Increment sample number
  delay(500);             // Delay of 500 milliseconds
}
```

P1-2 data was collected with a few changes. Here the sampling rate was changed to be a rate 1 sample per 100 milliseconds. Next a timer interrupt was integrated with the MsTimer2 Arduino library.

```
#include <MsTimer2.h>

const int TSAMP_MSEC = 100;
volatile boolean sampleFlag = false;
int nSmpl = 1, sample;

void setup() {
  Serial.begin(9600);
  MsTimer2::set(TSAMP_MSEC, ISR_Sample); // Set sample msec, ISR name
  MsTimer2::start(); // start running the Timer
}

void loop() {
  while (sampleFlag == false); // spin until ISR trigger
  sampleFlag = false; // disarm flag: enable next dwell

  sample = analogRead(A0);
  // Display results to console
  if (nSmpl == 1) Serial.print("\nsmpl\tADC\n");
  Serial.println(String(nSmpl) + '\t' + String(sample));
  ++nSmpl;
} // loop()

void ISR_Sample() {
 sampleFlag = true;
}
```

The P1-3 code setup() function was edited so that instead of the sample collecting beginning immediately after the microcontroller being flashed it waited for an input in the terminal. This allowed for the user to control the beginning of the data collection. Another change added was that a max amount of samples collected of 256 was set. By taking the $++nSmpl$; and putting it in an if statement that compares its current value with the value of max samples it can be determined whether the max sample count hasa been reached or not. If the max count has been reached the ISR is not called and the program is infinitely stuck in the while (sampleFlag == false); loop.

```
#include <MsTimer2.h>

const int TSAMP_MSEC = 100;
volatile boolean sampleFlag = false;
int nSmpl = 1, sample;
const int NUM_SAMPLES = 256;

void setup()
{
 Serial.begin(9600);
 Serial.println("Enter 'g' to go .....");
 while (Serial.read() != 'g'); // spin until 'g' entry
 MsTimer2::set(TSAMP_MSEC, ISR_Sample); // Set sample msec, ISR name
 MsTimer2::start(); // start running the Timer
 }

void loop()
{
  while (sampleFlag == false); // spin until ISR trigger
  sampleFlag = false; // disarm flag: enable next dwell

  sample = analogRead(A0);
  // Display results to console
  if (nSmpl == 1) Serial.print("\nsmpl\tADC\n");
```

```
    Serial.println(String(nSmpl) + '\t' + String(sample));

    if (++nSmpl > NUM_SAMPLES) MsTimer2::stop();
  } // loop()

  void ISR_Sample()
  {
   sampleFlag = true;
  }
```

For P1-4 Matlab was used in unison with the Arduino IDE to collect and plot the IC data. The Arduino code sends a string “%Arduino Ready” on the terminal that Matlab will recognize and then send a ’g’ in response. The Matlab scripts let us automate the process of recording the data, plotting it, and saving the data collected into a .mat file.

```
#include <MsTimer2.h>

const int TSAMP_MSEC = 100;
volatile boolean sampleFlag = false;
int nSmpl = 1, sample;
const int NUM_SAMPLES = 256;

void setup()
{
 Serial.begin(9600);
 Serial.println("%Arduino Ready"); // Send the ready string
 Serial.println("Enter 'g' to go .....");
 while (Serial.read() != 'g'); // spin until 'g' entry
 MsTimer2::set(TSAMP_MSEC, ISR_Sample); // Set sample msec, ISR name
 MsTimer2::start(); // start running the Timer
 }

void loop()
{
  while (sampleFlag == false); // spin until ISR trigger
  sampleFlag = false; // disarm flag: enable next dwell

  sample = analogRead(A0);
  // Display results to console
  if (nSmpl == 1) Serial.print("\nsmpl\tADC\n");
  Serial.println(String(nSmpl) + '\t' + String(sample));

  if (++nSmpl > NUM_SAMPLES) MsTimer2::stop();
} // loop()

void ISR_Sample()
{
 sampleFlag = true;
}
```

P1-5 added the calculation of the mean and standard deviation of the samples collected. The Arduino code would calulcate the mean and standard deviation. The standard deviation is calulcated using this equation.

$$s^2 = \frac{1}{n(n-1)}\left(n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2\right) \tag{2}$$

In the C code was written as

$$\text{runningMean} = \text{oldMean} + \frac{x_i - \text{oldMean}}{\text{numSamples}}$$

$$\text{runningVar} = \text{oldRunningSumVar} + (x_i - \text{oldMean})\cdot(x_i - \text{runningMean})$$

$$\text{variance} = \frac{\text{runningVar}}{\text{tick} - 1}$$

```
// OPEN NEW ARDUINO SKETCH.
// CLICK IN THIS TEXT BOX. CTRL-A, CTRL-C.
// CLICK IN SKETCH. CTRL-A, CTRL-V.

// Lab 3 starter: Cook stats

const int TOTAL_SAMPLES = 600; // simulated samples
int numSamples = 1;

// Declare the variables that are computed in the calculateStats function
float sample, runningMean = 0.0, runningVar = 0.0, stdev = 0.0, variance=0.0;

void setup()
{
  Serial.begin(9600);

// This line tells MATLAB that the Arduino is ready to accept data
  Serial.println("%Arduino Ready");

// Wait until MATLAB sends a 'g' to start sending data
while (Serial.read() != 'g'); // spin until 'g' entry

} // setup

void loop()
{
  sample = simSample();

  // Call the statistics calculation function
  calculateStats(sample);

  // Display the statistics
  displayStatsData();

  // Run TOTAL_SAMPLES iterations then halt
  if (++numSamples > TOTAL_SAMPLES) while (true);

} // loop()

float simSample(void)
{
  // Simulate sensor for stats calculation development
  float simSmpl, simAmp = 1.0, simT = 60;

  simAmp = ((numSamples > 180) && (numSamples < 300)) ? 0.125 : 1.0; // burst amplitude
  // simT = ((numSamples > 180) && (numSamples < 300)) ? 30.0 : 60.0; // burst frequency
  simSmpl = 180.0 + simAmp*sin((numSamples/simT)*TWO_PI); // fixed amplitude, frequency

  return simSmpl;
} // simSample

void calculateStats(float xi)
{
  // Calculate running statistics per Cook pseudo code.
  static int tick = 1;
  static float oldMean, oldRunningSumVar;
```

```
    if (numSamples == 1) {
      runningMean = xi;
      runningVar = 0;
      variance = 0;

      //set up for next iteration
      oldMean = runningMean;
      oldRunningSumVar = runningVar;
    } // if
    else {
      runningMean = oldMean + (xi - oldMean)/numSamples; // Calculate Mean
      runningVar = oldRunningSumVar + (xi - oldMean) * (xi - runningMean); // Calculate Variance
      variance = runningVar/(tick - 1);

      oldMean = runningMean;
      oldRunningSumVar = runningVar;
    }

    tick++;

  } // calculateStats

  void displayStatsData(void)
  {
    // Display results to console
    if (numSamples == 1)
    {
      Serial.print("\nn\tsmpl\trunningMean\tstdev\r\n");
    }

    Serial.print(numSamples);
    Serial.print('\t');
    Serial.print(sample);
    Serial.print('\t');
    Serial.print(runningMean);
    Serial.print("\t\t");
    Serial.println(variance);
  } // displayStatsData
```

III. Results

Multiple temperature data sets were captured at different sampling rates by modifying the Arduino code. The first data set was recorded with a 500 ms sampling rate by having a hard-coded delay.

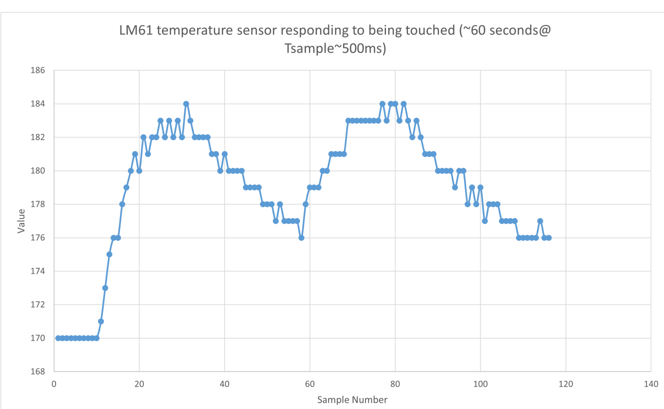

Fig. 1: 500 ms Sampling Rate Results

In this initial setup, the Arduino recorded temperature data at a fixed interval of 500 ms using a hard-coded delay. The results show a smooth progression of temperature values, with minimal fluctuations due to the slow sampling rate.

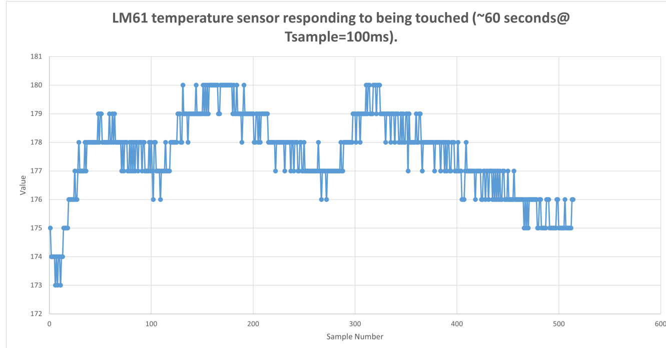

Fig. 2: 100 ms Sampling Rate Results

By replacing the hard-coded delay with a timer interrupt, the sampling rate was increased to 100 ms. The collected data exhibited finer temporal resolution, capturing more details about short-term temperature variations.

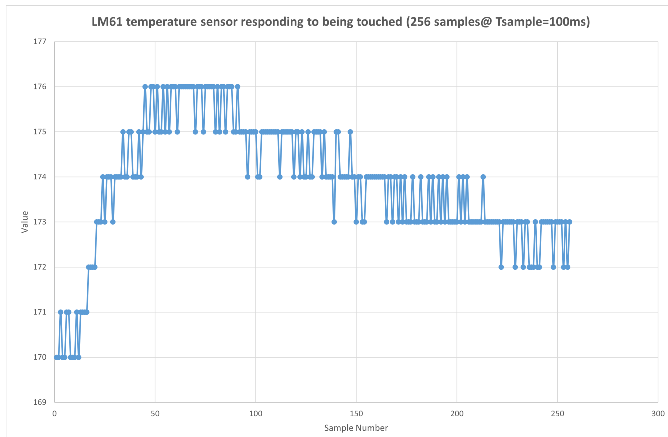

Fig. 3: 256 Samples Collected at 100 ms Intervals

In this iteration, the program was modified to start recording only upon user input and to terminate after collecting 256 samples. This approach ensured consistent data lengths for analysis.

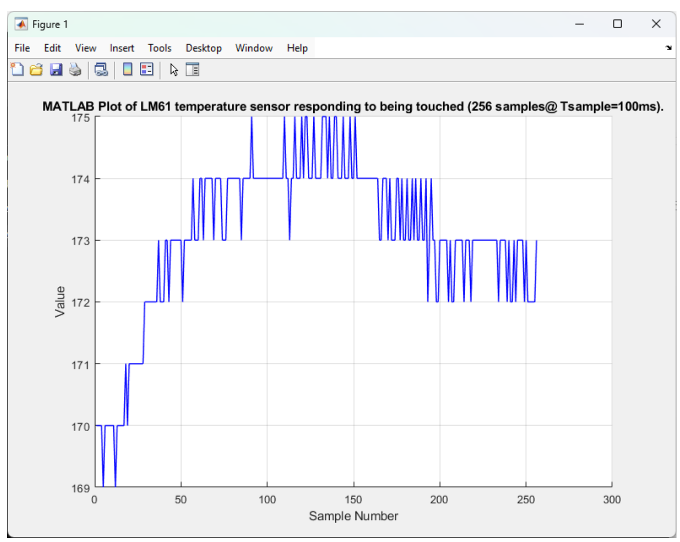

Fig. 4: LM61 Sensor: 256 Samples at 100 ms in MATLAB

The data collection was integrated with MATLAB, allowing for automated processing and visualization. This setup enabled real-time plotting and streamlined data handling.

Statistical analysis was performed to examine fluctuations in the recorded temperature values. Key metrics such as the mean and standard deviation were computed for different scenarios.

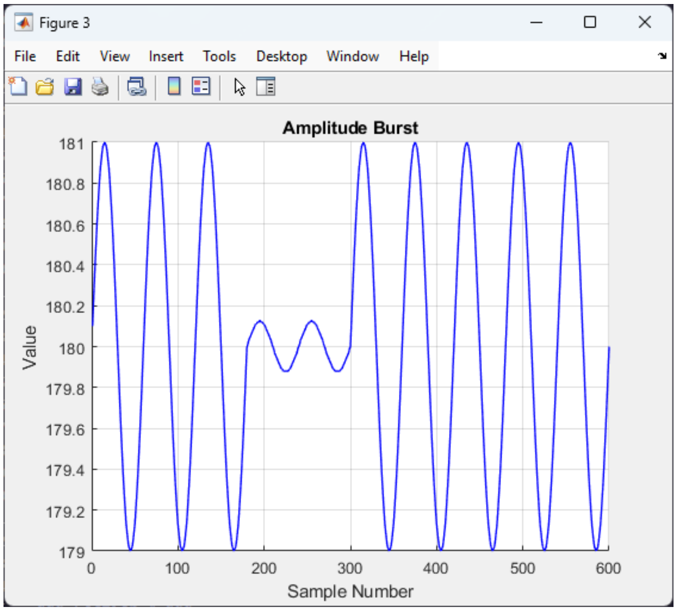

Fig. 5: Amplitude Burst Plot

This test introduced an amplitude burst in the data, simulating abrupt temperature changes. The results show clear deviations in the readings, as illustrated below.

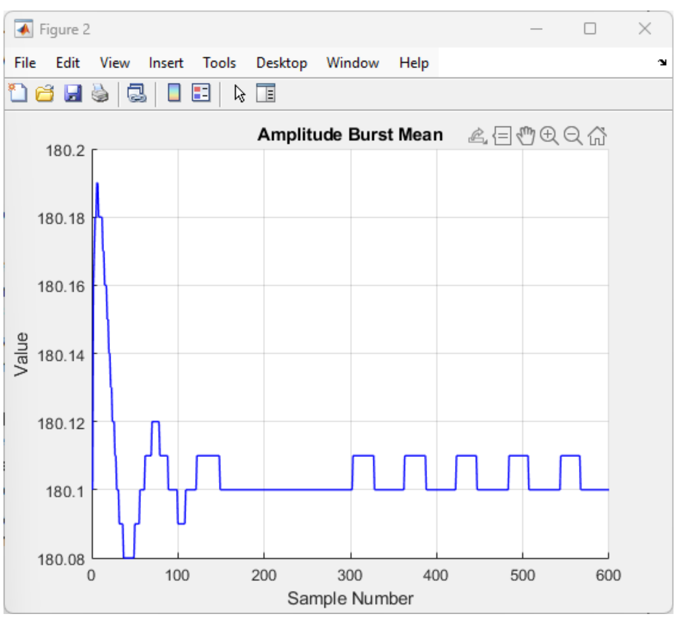

Fig. 6: Amplitude Burst Mean

The calculated mean of the amplitude burst dataset provides insight into the overall temperature shift

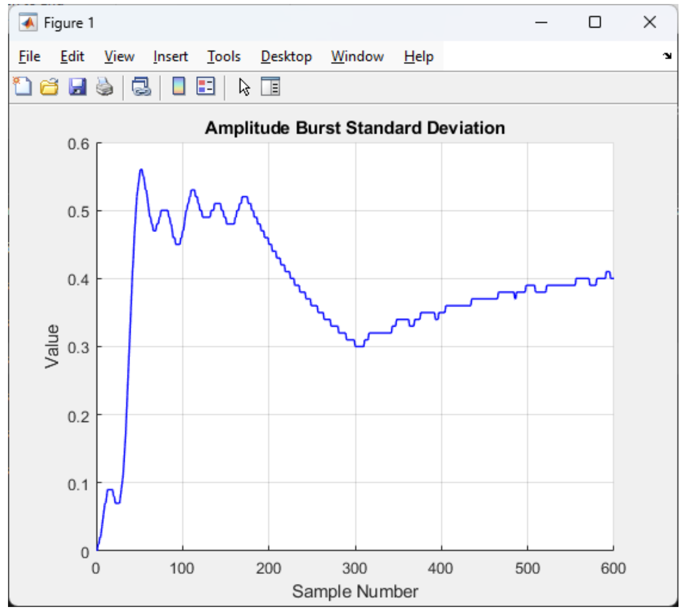

Fig. 7: Amplitude Burst Standard Deviation

The standard deviation plot highlights increased variability during the burst phase.

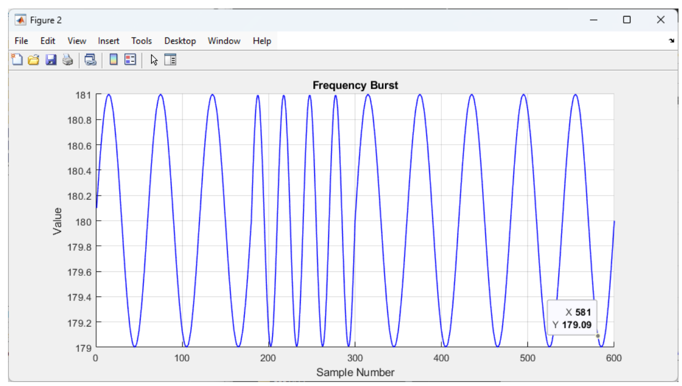

Fig. 8: Frequency Burst Plot

In this case, a frequency burst was introduced, altering the rate of temperature fluctuations. The resulting data captures rapid changes in sensor readings.

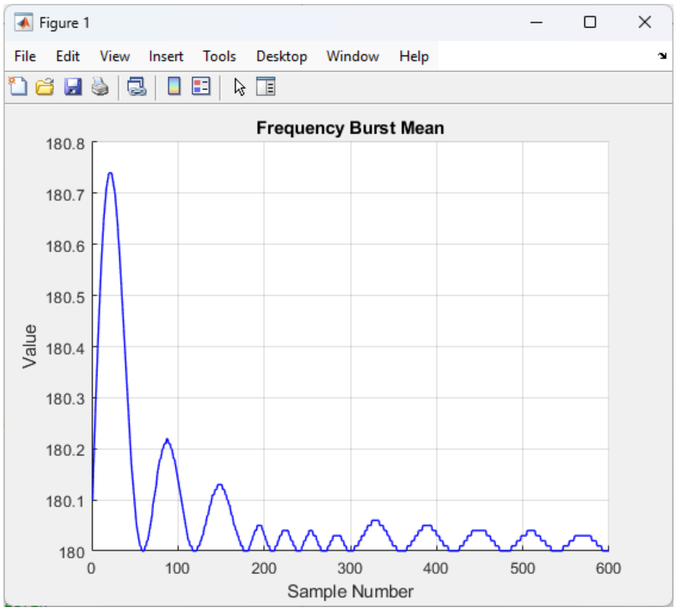

Fig. 9: Frequency Burst Mean

The mean temperature over the frequency burst interval is shown below.

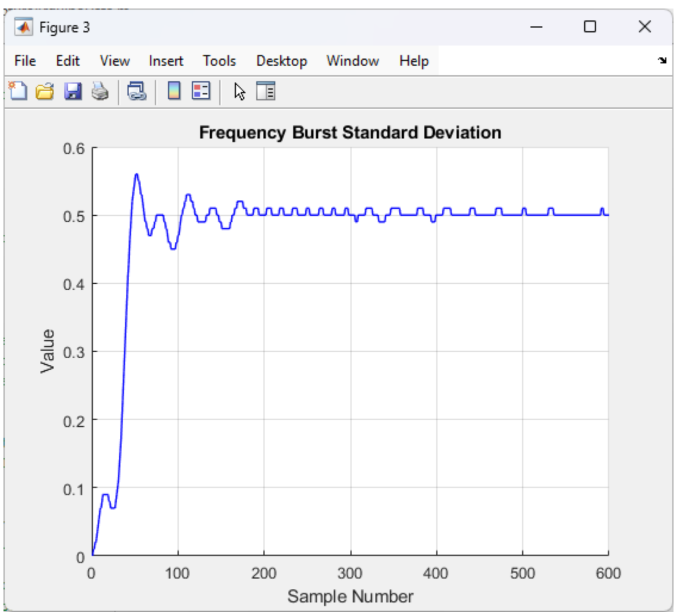

Fig. 10: Frequency Burst Standard Deviation

The standard deviation plot indicates increased variation due to the burst.

IV. Analysis

The analysis of mean and standard deviation in statistics is crucial, as these variables are indicators of the procedure's success. For P1-5 the Arduino code was edited to calculate the mean and standard deviation of the samples.

The standard deviation is more responsive to temperature variations than the mean because it captures the extent of changes in the data, where the mean only provides a central value. The formula for standard deviation accounts for the squared differences between each data point and the mean, making it more sensitive to larger deviations. In contrast, the mean smooths out variations by averaging all values, reducing the impact of extreme temperature changes. For instance, if temperature readings fluctuate significantly over time, the standard deviation will increase, reflecting the variation, while the mean may remain relatively stable. This makes standard deviation a better indicator of dynamic temperature changes.

V. Conclusion

The experimental setup successfully demonstrated an optimized method for temperature data collection using an Arduino and IC system. By progressively improving data recording techniques, the lab highlighted the advantages of interrupt-based sampling, user-controlled initiation, and MATLAB integration for enhanced data analysis. The statistical evaluation reinforced the reliability of the collected data and provided deeper insights into temperature changes.

Future work could explore additional enhancements such as adaptive sampling rates, machine learning-based anomaly detection, and integration with wireless communication for remote data monitoring. These improvements would further increase the system's versatility and applicability in real-world temperature monitoring scenarios.