# How to Succeed With This Lab

Before executing the procedure, skim-read through the whole lab handout.

Skim-read through the Write Up section so you know what is expected for the lab report. Work together as a team with your lab partners to solve problems and overcome any snags (e.g. syntax issues with C, fabrication of the sensor leads, writing up an explanation of results).

# The number one problem students face doing this lab:

Incorrectly wiring the sensor/ not correctly reading the datasheet pinout. Not sure? Ask. On the datasheet, the sensor pins are shown come out of the page towards you.

# Objectives

1. Become familiar with the Arduino integrated development environment (IDE)
2. Become familiar with the Arduino hardware’s input and output capabilities: analogRead(), analogWrite(), Serial.println(), etc.
3. Use the Arduino to read a temperature signal from the LM61 temperature sensor and display the sensor reading to serial terminal window on the host PC.
4. Use the serial monitor to collect data and plot the data with Excel.
5. Setup an interrupt routine to provide a regular sampling interval of 100 milliseconds, using MsTimer2.h.
6. Practice modifying the source code, uploading and debugging.

# Reading

http://Arduino.cc

# Intended Learning Outcomes:

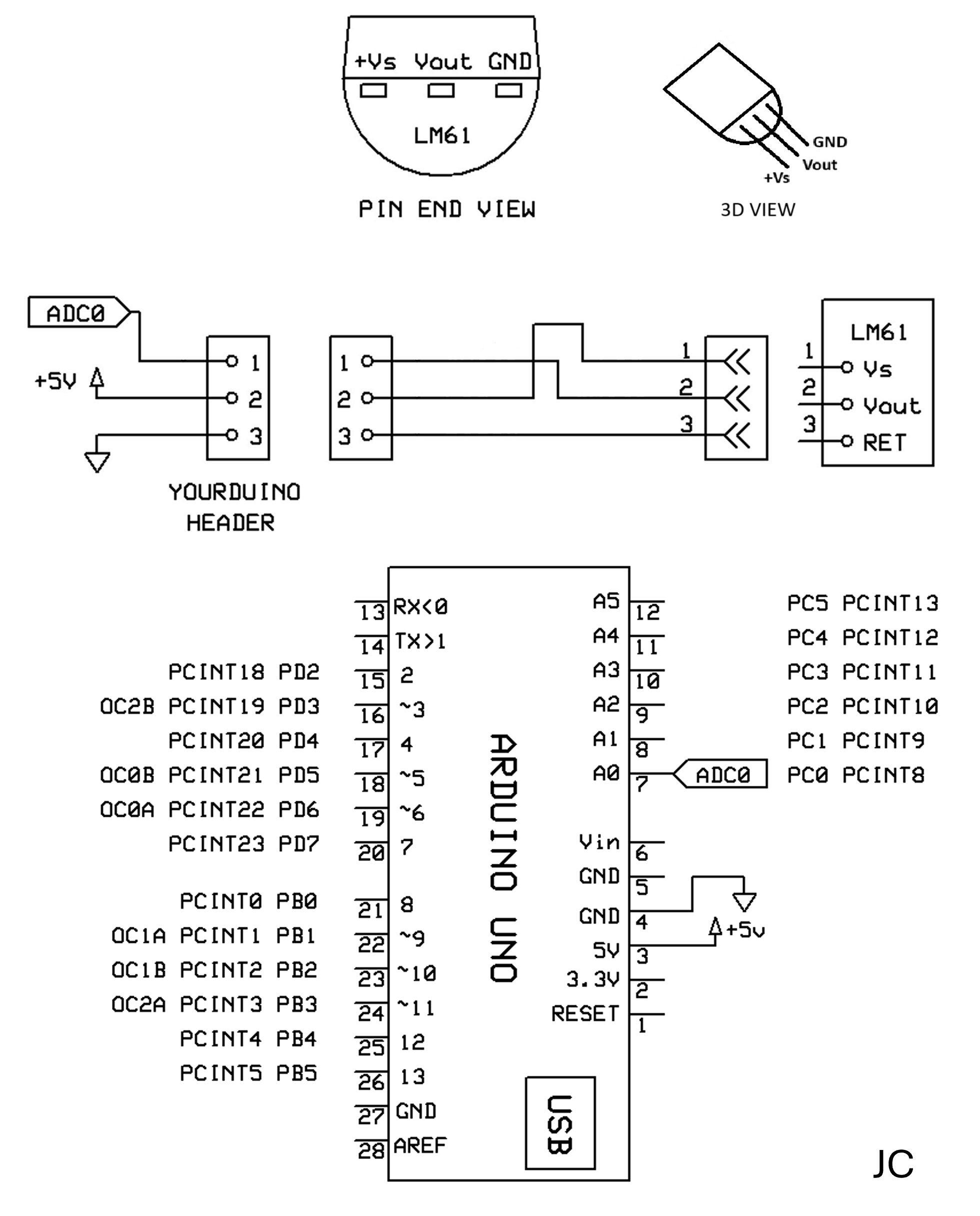
|  |  |
| --- | --- |
| Utilize Arduino environment to edit, compile, load and run a program. |  |
| Demonstrate understanding of how to use Arduino serial terminal monitor window to debug a program, display data and copy that data to Excel for post processing. |  |
| Demonstrate use of software interrupts to establish a regular sampling interval. |  |

**Overview of Arduino Uno Microprocessor**

|  |  |
| --- | --- |
| Microcontroller | ATmega328 |
| Operating Voltage | 5V |
| Input Voltage (recommended) | 7-12V |
| Input Voltage (limits) | 6-20V |
| Digital I/O Pins | 14 (of which 6 provide PWM output) |
| Analog Input Pins | 6 |
| DC Current per I/O Pin | 40 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 32 KB (ATmega328) of which 0.5 KB used by bootloader |
| SRAM | 2 KB (ATmega328) |
| EEPROM | 1 KB (ATmega328) |
| Clock Speed | 16 MHz |

# Procedure

1. Setup the following items in the lab.
   1. Arduino Uno microprocessor board. **IF YOU HAVE THE RED YOURDUINO VERSION, SET THE OPERATING VOLTAGE SWITCH TO 5 VOLTS.**
   2. USB cable from Uno to host PC.
   3. Insert the TO-92 LM61 temperature sensor pins into the 3 pin header of the supplied cable.
   4. Carefully observe the schematic below. Your cable may have one end with 3 connections molded together and one end with three free wires. The pictures below are shown with an older cable so your colors will differ! Make sure that you follow the schematic and not the color code in the picture. Getting this right will save you many hours in later labs!! Note that the drawing of the LM61 shows the pins as if they are pointing directly at you. I prefer to have the molded end of the cable with the 3 fixed connections used for holding the sensor. This will keep the leads from breaking off. If you prefer you can solder a 3-pin header to the temperature sensor and plug that into the cable. The most important thing is that the +Vs of the sensor goes to +5V of the Arduino, Vout of the sensor goes to A0 and GND or RET pin on the sensor goes to ground on the Arduino. If you are unsure ask!

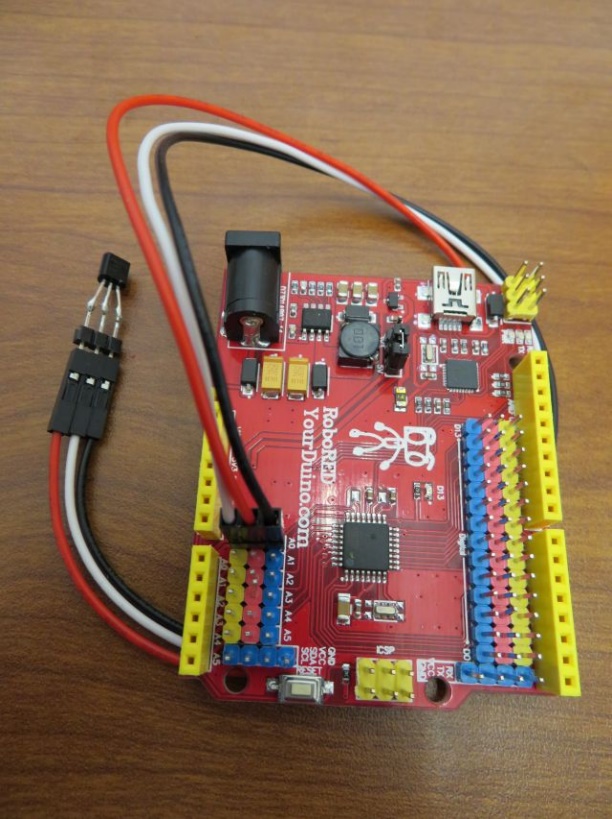


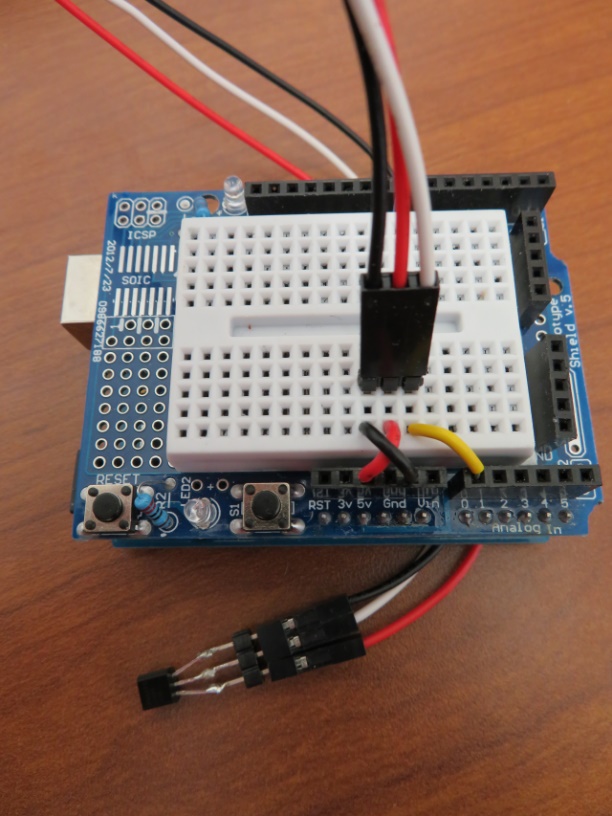
* 1. Using jumpers from your kit and the protoboard, jumper wires from the LM61 cable to +5V, ground and analog input zero (A0) on the microprocessor board.

If you are using a Yourduino Uno (red board) the free end of the sensor cable may be plugged directly into the header adjacent to A0.

If you are using an Arduino Uno, map the free end of the sensor cable through a three-element male/male header into a proto board and then with jumpers to the Uno headers.

Double check that you have each pin of the LM61 temperature sensor connected to the correct signal on the Arduino.





**NOTE! Your wire colors may differ! Follow the schematic!**

1. Launch the Arduino Integrated Development Environment (ID Arduino.exe program located in the C:software folder.
2. Load the AnalogReadSerial program.
   1. Start a new sketch using File/New
   2. Click in the text box below. Type CTRL-A and copy the entire text from the box using CTRL-C. There may be more code than you can see in the text box so be sure to use CTRL-A to copy the entire code.
   3. Paste the copied text into the IDE replacing all of the text that is currently in the IDE editor.

// AnalogReadSerial: public domain code

// Reads analog input on pin 0, prints to console.

void setup() // runs once when you press reset

{

Serial.begin(9600);

}

void loop() // runs over and over forever

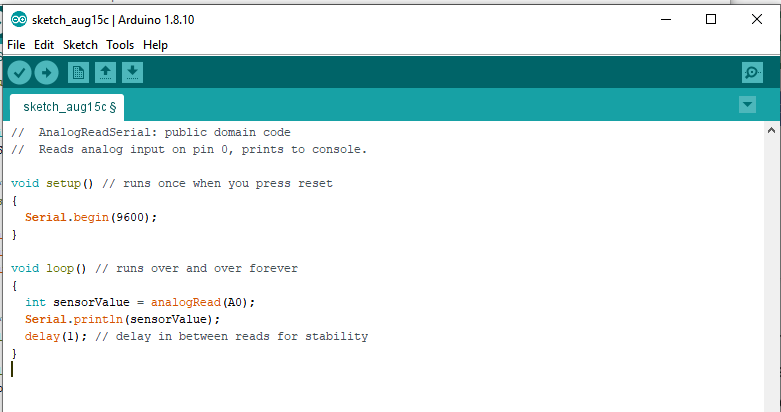
{

int sensorValue = analogRead(A0);

Serial.println(sensorValue);

delay(1); // delay in between reads for stability

}



* 1. This code sets up serial communication at 9600 baud to the PC, then reads an analog value from pin zero (A0), and prints the decimal value of the ADC converter reading to the serial port.
  2. Upload the program to the microprocessor File->Upload, or Ctrl+U. This will compile the code, check for errors, and upload the file.
  3. If you see an error message that says file did not upload, go to Tools-> Serial Port and change the serial port e.g. select com6 instead of com1. Try uploading again
  4. Open a serial terminal window and view the data coming back from the microprocessor. Tool->Serial Monitor or Ctrl+Shift+M. Typical values will be around 170.
  5. Put your finger on the temperature sensor to warm it up and watch for the A/D reading values to increase slightly.

This code reads the ADC on analog input A0 and prints the output to the serial port as rapidly as it can with only a slight delay between reads (1 mSec). It will continue to read the port until the program is terminated by the user.

1. Now modify the code so that there is a delay between display updates:
   1. Copy the code in the text box below by clicking in the box, typing CTRL-A and CTRL-C to select and copy all of the text. Paste the code to the IDE being sure to replace all of the existing code.
   2. The code in the text box below increases the loop delay to 500 msec by modifying the delay command to “delay(500);” after each read of the ADC.

This slows the loop down to two samples per second, making it easier to see changes.

It also adds a header text line to the setup() routine to show the sample number “smpl” and the ADC value “ADC”.

* 1. The code also adds a loop counter variable nSmpl, and adds a statement to print the sample number.

int nSmpl = 1, sample; // global variables

void setup()

{

Serial.begin(9600);

Serial.print("\nsmpl\tADC\n"); // column headers

}

void loop()

{

sample = analogRead(A0);

Serial.print(nSmpl); Serial.print('\t'); // tabbed output

Serial.println(sample);

++nSmpl;

delay(500);

}

* 1. Upload the code. Close the old serial monitor. Open a new monitor window. Observe data values as you warm the sensor with your finger.
  2. Copy and paste data from the Serial Monitor window into Excel. Plot temperature data for 60 seconds or so. Capture temperature rise and fall.

**NOTE: I have had problems using the CTRL-A then CTRL-C combination to copy data from the serial monitor. I found that this doesn’t work reliably. You may have to select the values you want manually by clicking and dragging the mouse then using CTRL-C to copy.**

* 1. Plot the data using an Excel scatter plot of the temperature response in your report. Put the plot in a new Tab (P1-1) of the spreadsheet. Label axes and title the graph with the text below:

**LM61 temperature sensor responding to being touched (~60 seconds@ Tsample~500ms).**

# SECTION 2

# Setting up an interrupt to provide a uniform sampling interval

## Overview:

In the last section, you made the microprocessor take two samples per second, but the actual sampling period was not exactly 0.5 seconds. Rather it is the sum of the explicit delay from the delay(500) function and the execution time of the rest of the loop() code.

In this section loop timing is established with a hardware timer driven interrupt routine. Using an interrupt is more accurate than a delay.

The microprocessor will be idle while waiting for the interrupt to occur then at a precise time the interrupt “fires” and the code to read the analog port executes.

The interrupt used is defined in the header file MsTimer2.h, which stands for millisecond timer2. It is a library which contains an interrupt timer routine.

#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.print(nSmpl); Serial.print('\t');

Serial.print(sample); Serial.print('\n');

++nSmpl;

} // loop()

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void ISR\_Sample()

{

sampleFlag = true;

}

## Program Description:

The first line includes the library MsTimer2.h. Global variable sampleFlag is used to signal that it is time to take a new sample. It is declared as volatile boolean to ensure that when an interrupt routine is called, it is not over-written or misplaced when execution jumps from the main loop to the interrupt routine and back.

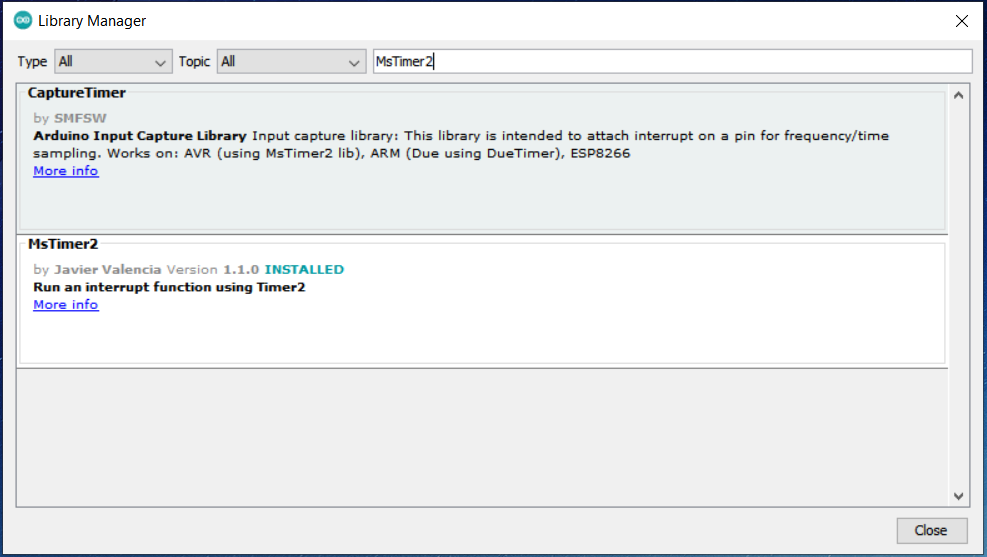
In Setup, the serial port is started at 9600 baud. Then the MsTimer2 is configured to jump every 100 milliseconds to the interrupt routine called ISR\_Sample. In ISR\_Sample, the variable sampleFlag is set true. This is a message, or semaphore, to the main loop that it is time to read a new voltage. The voltage is not read by this interrupt routine, it just says that it is time to do so. After the MsTimer2 is configured, the next line starts the timer that will jump to the interrupt routine.

In the main loop, a while() loop blocks progress until sampleFlag is set true by ISR\_sample. Once the flag is detected true, execution escapes the while() loop. A new analog voltage is read and sent out over the serial port.

By setting up the code this way, a sample is taken every 100 milliseconds regardless of the execution time of the sampling and console output code. This regular sampling interval is critical for low noise, high performance sampled data system operation.

## Procedure

1. Install the MsTimer2.h library going to Tools\Manage Libraries from within the Arduino IDE. Search for MsTimer2 in the search bar. When it comes up press the Install button if not already installed. The figure below shows the module already installed.



1. After you do this, the Arduino IDE must be quit and restarted so that it recognizes the new library files.
2. Copy and paste the file code in the text block above, upload, and run the program. Open the serial port and watch the data come in at a faster rate.
3. Collect approximately 60 seconds of data. Starting from ambient temperature, wait one second then pinch the sensor between your fingers for 10 seconds and then let go of the sensor. Plot the data in Excel.
4. Include an Excel plot (in a separate tab labeled P1-2) of the temperature response. Label axes and title the graph as:

**LM61 temperature sensor responding to being touched (~60 seconds@ Tsample=100ms).**

# SECTION 3a – Controlling start of data collection

Now that the temperature samples are taken at regular time intervals, we would like to control the time when data collection begins. Enable user control of the start of data collection by modifying just the setup() with the following code.

This code sends a prompt to the Serial Monitor then waits in a while loop reading the serial port until it reads a ‘g’ character from the serial port. The ‘g’ character comes from the console of the Serial Monitor. If you upload this new code, the microprocessor will wait until you enter a ‘g’ character on the console.

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

}

# SECTION 3b – Collecting exactly 256 samples of data during touch test

A counter can be used to track the number of samples collected. A comparison can then be made of the number of samples collected to the desired number of samples and the program can be stopped.

For ease of maintenance, define the sample count as a global constant variable. Add the statement before the setup() function. At the end of the loop, replace the simple increment of nSmpl with an increment and comparison to the target count. When the target is achieved, disable ISR\_Sample() calls. The loop then freezes in the while(sampleFlag) loop.

See the code in the text box below for the modifications to the code that you need to make.

...

const int NUM\_SAMPLES = 256;

...

if (++nSmpl > NUM\_SAMPLES) MsTimer2::stop();

...

Include in your report an Excel plot (on a separate tab labeled P1-3) of the temperature response for exactly 256 samples. Label axes and title the graph as:

**LM61 temperature sensor responding to being touched (256 samples@ Tsample=100ms)**

# SECTION 4 – Reading Data from the Arduino into MATLAB

We’d like to be able to capture the data from the Arduino directly into MATLAB for later plotting and processing. MATLAB can read data from the Arduino serial port and plot the data as it is being read. This is useful to see what is happening to the temperature sensor in near real time.

Three MATLAB scripts are included in myCourses that are used to read and display data from the Arduino. These files are:

ArduinoSerial.m – This is a MATLAB class that supports operation of the serial port to interface to the Arduino

CaptureArduinoData.m – This is a MATLAB function that is called to perform the initial handshake with the Arduino and read data from the serial port. Once data has stopped being read, that data can be saved into a MATLAB data file (.mat).

ActivePlot.m – This is a MATLAB class that supports plotting data received from the serial port and updating the plot each time that a sample is received.

Save these three files in a MATLAB folder on your user drive within a main folder for the lab. MATLAB must have access to these files in order to execute properly. Using this folder as the starting location for running MATLAB allows MATLAB to access the files. Putting this folder on your MATLAB path will also make them accessible.

Just as we waited for the prompt on the terminal “Enter 'g' to go .....” and responded with the typing the letter 'g' to tell the Arduino to begin executing code a similar “ready and response” handshake needs to take place between the Arduino and MATLAB.

In the code in the text box below, the Arduino sends out a string on the serial port “%Arduino Ready” that MATLAB recognizes. When MATLAB sees this prompt it will send the letter ‘g’ back to the Arduino indicating that it can start executing its code. MATLAB will capture the data and echo the values to the command window. After the data has stopped being sent, MATLAB will display a prompt asking if you want to save the data in a file. If you respond ‘y’ then you will be prompted to enter a file location and name. Do not enter the file at the initial prompt. Respond with either ‘y’ or ‘n’.

## Procedure

1. Copy and paste the Arduino code in the text box below into the Arduino IDE

#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);

// This line tells MATLAB that the Arduino is ready to accept data

Serial.println("%Arduino Ready");

//Serial.println("Enter 'g' to go .....");

// Wait until MATLAB sends a 'g' to start sending data

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.print(nSmpl); Serial.print('\t');

Serial.print(sample); Serial.print('\n');

if (++nSmpl > NUM\_SAMPLES) MsTimer2::stop();

} // loop()

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

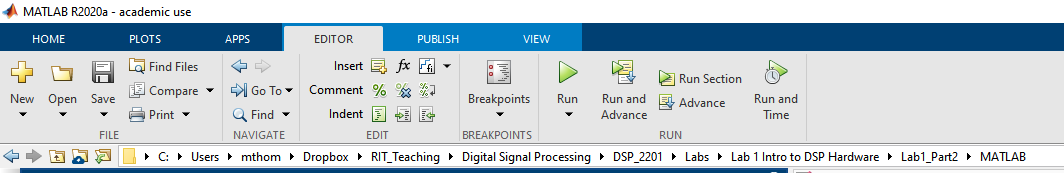
void ISR\_Sample()

{

sampleFlag = true;

}

1. Open MATLAB and set the working directory to the location where you saved the three MATLAB files described above. I have shown an example of where they are located on my machine.



1. Upload the Arduino code to the hardware.
2. In the MATLAB command window (we won’t be using the Live Editor for this) type the command “data = CaptureArduinoData” as shown below using the Name-Value property pairs for the Com Port and the Baudrate. The default value for the Com Port is 3 and Baudrate is 9600. If these are the values that you need, then you don’t need to include the arguments.

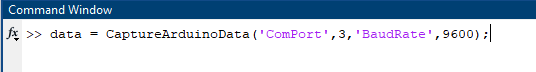


Figure Command Line to Capture Data from the Arduino on the Serial Port

Here is an easily copyable line of text for the command shown in the figure

data = CaptureArduinoData('ComPort',3,'BaudRate',9600);

The MATLAB script will echo what is sent to the serial port just as was done using the serial monitor.

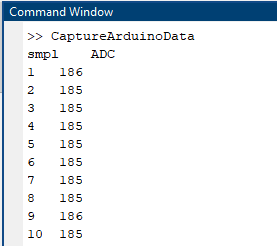


Figure MATLAB Echo of Arduino Serial Port

After the Arduino stops sending data (the default timeout is 1 second with no data), MATLAB will issue a prompt asking whether you want to save the data. The default is NO. Type ‘y’ to save the data. A file selection interface will appear. Save the data using a descriptive file name so that you know what the data was later.

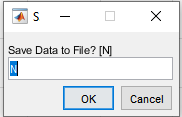


Figure MATLAB Prompt to Save Data after Arduino Stops Sending

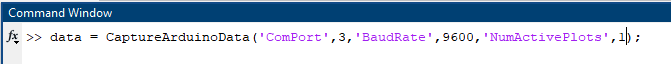
**NOTE: There may be times when MATLAB and the Arduino are trying to use the serial port at the same time and you won’t be able to upload the Arduino sketch. If this happens type the following command into the MATLAB command window.**

**A screen shot of a computer

Description automatically generated**

**This will attempt to release the serial port from MATLAB. If this doesn’t work try to re-load the Arduino sketch.**

1. It is useful to see what is happening with the values captured by the temperature sensor as they occur. An active plot of the data is a convenient tool and this can be done using the CaptureArduinoData script. Adding an option to the calling arguments of the script will enable the plot. Use the command “CaptureArduinoData;” with an additional Name-Value property called NumActivePlots followed by the number of plots to be created.

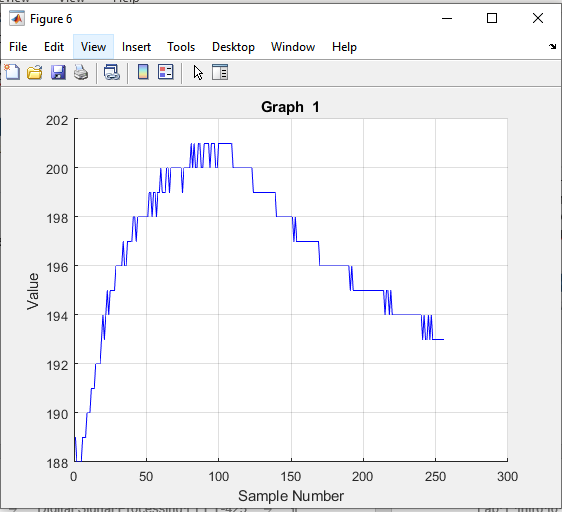


The command line as text for easy copy is shown below

data = CaptureArduinoData('ComPort',3,'BaudRate',9600,'NumActivePlots',1);

After executing the command in MATLAB a figure will open. As data comes in on the serial port, you will see the data actively being drawn and updated on the plot. The data is always plotted versus the sample number in the first column. To create one active graph you must have at least NumActivePlots + 1 columns of data, the first column is always used as the x-axis value and the remaining columns are the y-axis values for each plot.

1. Upload the Arduino script again. You must wait for the script to load before starting MATLAB as the serial port is used to upload the script and transfer data. Capture data in MATLAB using an Active plot. After about 1 second, hold the temperature sensor between your fingers for about five seconds and then release. Watch what happens to the data on the graph. Save the data to a file with a descriptive name that you will remember.



1. Once the plot is complete you can use “Tools/Edit Plot” in the figure window to add a descriptive graph title and x and y axis labels. Save the figure into a document for your report. Use the following title for the graph

**MATLAB Plot of LM61 temperature sensor responding to being touched (256 samples@ Tsample=100ms).**

# Section 5 – Computing the Running Variance

Write code that runs in the Arduino to calculate the running mean and the running standard deviation. In class we talked about one approach that computes a running sum and a running sum of squares. This approach can result in some problems when the mean is large relative to the standard deviation. An approach that improves on this is discussed in the article “Accurately computing running variance” which you will find in myCourses. This is summarized below.

Accurately computing running variance

Adapted from http://www.johndcook.com/standard\_deviation.html

The most direct way of computing sample variance or standard deviation can have severe numerical problems. Mathematically, sample variance can be computed as follows.

\sigma^2 = \frac{1}{ n(n-1)}\left(n \sum_{i=1}^n x_i^2 -\left(\sum_{i=1}^n x_k\right)^2\right)

The most obvious way to compute variance then would be to have two sums: one to accumulate the sum of the x's and another to accumulate the sums of the squares of the x's. If the x's are large and the differences between them small, direct evaluation of the equation above would require computing a small number as the difference of two large numbers, a red flag for numerical computing. The loss of precision can be so bad that the expression above evaluates to a *negative* number even though variance is always positive. See [Comparing three methods of computing standard deviation](http://www.johndcook.com/blog/2008/09/26/comparing-three-methods-of-computing-standard-deviation/) for examples of just how bad the above formula can be.

There is a way to compute variance that is more accurate and is guaranteed to always give positive results. Furthermore, the method computes a running variance. That is, the method computes the variance as the x's arrive one at a time. The data do not need to be saved for a second pass. Most people are probably unaware that computing sample variance can be difficult until the first time they compute a standard deviation and get an exception for taking the square root of a negative number. The method has superior numerical properties. The algorithm is as follows.

Initialize M1 = x1 and S1 = 0.

For subsequent x's, use the recurrence formulas

Mi = Mi-1+ (xi – Mi-1)/i   
Si = Si-1 + (xi – Mi-1)\*(xi - Mi).

For 2 ≤ i ≤ n, the ith estimate of the variance is s2 = Si/(i - 1).

The numerically advantaged method is written out in pseudo code below to more easily see how to implement it.

Pseudo code that uses less memory ( assumes ‘i’ starts at 1 )

If i==1 then

Mean=x[i];

RunningSumVar=0;

Variance =0;

//set up for next iteration

Mean\_old=Mean;

RunningSumVar\_old=RunningSumVar;

Else

Mean= Mean\_old+ (x[i]-Mean\_old)/i

RunningSumVar = RunningSumVar\_old + (x[i]-Mean\_old)\* (x[i]-Mean)

Variance = RunningSumVar/(i-1)

//set up for next iteration

Mean\_old=Mean;

RunningSumVar\_old=RunningSumVar;

Adapt the pseudo code here into C code for the Arduino. Start with the base code in the text box below.

// 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, runningStdev = 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

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void calculateStats(float xi )

{

// Calculate running statistics per Cook pseudo code.

static int tick = 1;

static float oldMean, oldRunningSumVar;

float mean, runningSumVar, variance;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void displayStatsData(void)

{

// Display results to console

if (numSamples == 1)

{

Serial.print("\nn\tsmpl\trunningMean\tstdev\n");

}

Serial.print(numSamples); Serial.print('\t');

Serial.print(sample); Serial.print('\t');

Serial.print(runningMean); Serial.print('\t');

Serial.print(runningStdev); Serial.print('\n');

}

Note that there are some differences in the variable names used in the pseudo code text and the code below. In particular “xi” represents the variable x[i] in the pseudo code and “tick” represents the number of iterations “i”. Don’t forget to increment the number of iterations “tick” each time! This is not shown in the pseudo code.

Note that the pseudo code computes the variance. We’d like to return the standard deviation “runningStdev” so you need to add a line of code for that. The result of the computation should be an update of the two global variables “runningMean” and “runningStdev”.

Include a text box in your report that contains just your statistics code. Investigate the running calculation of statistics on simulated breathing data starting with the code in the text box below. Put your code in the function “calculateStats”. In the function there is one line that creates a simulated sample variable (simSmpl).

simSmpl = 180.0 + simAmp\*sin((numSamples/simT)\*TWO\_PI);

This equation uses two variables, simAmp and simT which are created on the previous lines which are initially commented out. These variables are the sinusoid amplitude and the sinusoid period in samples.

Uncommenting the line defining simAmp will create a set of samples with a varying amplitude.

simAmp = ((numSamples > 180) && (numSamples < 300)) ? 0.125 : 1.0; // burst amplitude

Uncommenting the line defining simT will create a set of samples with a varying frequency.

simT = ((numSamples > 180) && (numSamples < 300)) ? 30.0 : 60.0; // burst frequency

Create two different sets of simulated samples on which to compute statistics. First uncomment the line defining simAmp. Then comment out the line defnining simAmp and uncommenting the line defining simT.

Include separate plots of running statistics of the mean and standard deviation. Include in your report a discussion that explains why the graphs look as they do.

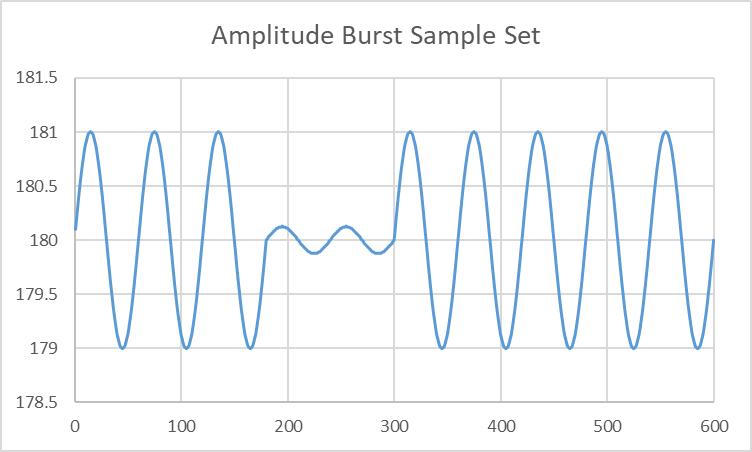


Figure Simulated signal using modified simAmp variable

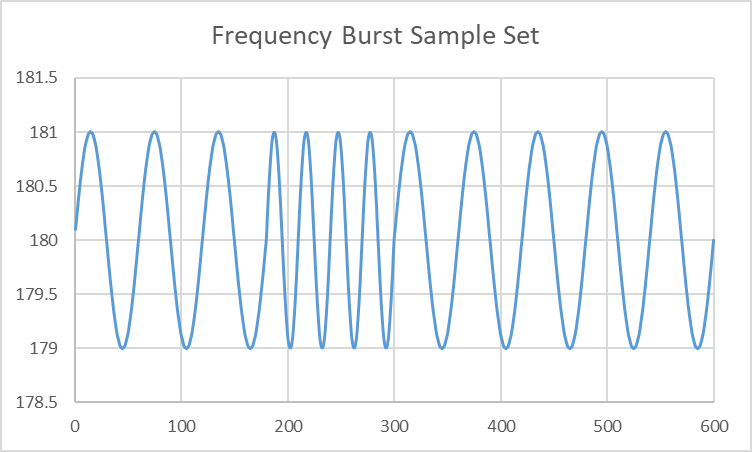


Figure Simulated signal using modified simT variable

# Report Write up

1. Your lab report must be submitted in PDF format. It must use the IEEE Journal Transactions Format. A template Word document is posted on myCourses for you to download. Write your report directly into the template document. Use the two-column format as given. Do not change the margins, font type or sizes. The format doesn’t work well in Google Docs so keep it in Word format, then when you are completed save it as a PDF document. In order to collaborate using Google Docs some groups have written the text of the report in Google Docs, then when completed copy and paste the text and figures into the Word File.
2. Include a simple title. Include the names of all your lab partners as co-authors.

3) Include the eight requested plots.

a) Fully label the vertical and horizontal axes of graphs. The vertical axis is the A/D converter reading as an integer from 0-1023 which represents temperature. The horizontal axis is the sample number. You could convert both axis labels to engineering units (e.g. temperature in degrees C, or time in seconds), however that is not required for this report.

b) Place a descriptive caption below each figure. The description that you write in the caption should sufficiently explain what is going on in the figure that a person who could not see the graph (e.g. a blind person using a text-to-speech program) would still be able to understand what the graph is showing.

c) For Tables, the title of the table goes above the Table. See the example in the IEEE journal paper template.

Here is a list of the eight required plots and their titles

**LM61 temperature sensor responding to being touched (~60 seconds@ Tsample~500ms).**

**LM61 temperature sensor responding to being touched (~60 seconds@ Tsample=100ms).**

**LM61 temperature sensor responding to being touched (256 samples@ Tsample=100ms).**

**MATLAB Plot of LM61 temperature sensor responding to being touched (256 samples@ Tsample=100ms).**

**Mean (running statistics) of Arduino Generated Amplitude Burst**

**Standard deviation (running statistics) of Arduino Generated Amplitude Burst**

**Mean (running statistics) of Arduino Generated Frequency Burst**

**Standard deviation (running statistics) of Arduino Generated Frequency Burst**

4) Include a very short discussion of your results. In addition to a general discussion of your results, address the following questions:

a) When, and how often is sampleFlag set true?

b) Why is sampleFlag set false every time the main loop executes? (hint: What would happen if it was not set false?)

1. Your lab partner believes when a person breathes on the temperature sensor (causing the temperature signal to go up and down) that the mean statistic will show this better than the standard deviation statistic. Write a one paragraph that explains why you think the standard deviation statistic will be more responsive to temperature variations than the mean statistic. Support your argument with either data from your experiments or a rationale based on the formulas for mean and standard deviation.

**Grading Rubric – LAB #3 Introduction to Arduino Microprocessor Hardware**

|  |  |  |  |
| --- | --- | --- | --- |
| Lab Section | Result Description | Points Available | Points Obtained |
| Report  Formatting | Report document Title  Names of Lab Partners on report  Two column Format used properly in report | 10 |  |
| Abstract | A brief introduction to the lab, its purpose and approach | 10 |  |
| Temperature Plots | 4 plots for the temperature vs sample number as described in the handout | 40 |  |
| Statistics  Plots | 4 plots of the running statistics, mean and variance.  Discussion of the responsiveness of the standard deviation | 20 |  |
| Discussion | Include discussion of the operation of the interrupt and why you believe that the standard deviation is more responsive to temperature changes than the mean. | 20 |  |