**Lab #2 Sensor Noise and Quantization Noise Measurements (Two Weeks)**

**Objectives:**

1. Learn how to create uniformly distributed random numbers.
2. Learn how to plot the histogram of a signal using Excel and characterize the signal’s histogram.
3. Measure the noise from a sensor using analog to digital converter (ADC) readings.
4. Learn how to create a digital to analog converter to create a source of dithering voltage.
5. Apply dithering to reduce quantization noise and increase signal to noise ratio.
6. Characterize a measurement system in terms of signal to noise ratio, precision and accuracy.
7. Compute the running variance of a signal.

**Reading:**

Chp 2 & 3 in Smith.

http://Arduino.cc

AN-804 Improving A/D Converter Performance Using Dither, Leon Melkonian, National Semiconductor, February 1992.

**Intended Learning Outcomes:**

|  |  |
| --- | --- |
| Demonstrate ability to analyze a dataset to determine its histogram. |  |
| Calculate statistical parameters of mean, standard deviation from a data set |  |
| Demonstrate the use of dithering noise and averaging to increase the signal to noise ratio of a measurement system. |  |
| Utilize typical error formula to compute equivalent number of bits of ADC resolution after dithering and averaging. |  |
| Demonstrate the ability to scale ADC converter count measurements to engineering units of voltage and Celsius. |  |

**Section 1 – Random Signal Generation in Arduino**

**Overview**

The Arduino microprocessor can produce a random variable using its random() function. To understand this random value, you will plot its histogram using Excel.

// Lab2\_random\_variables.ino

// Outputs 256 random variables in range 0..255

// created by: Clark Hochgraf 2015

// modified by: David Orlicki, August 17, 2017

#include <MsTimer2.h>

const int TSAMP\_MSEC = 10;

const int NUM\_SAMPLES = 256;

volatile boolean sampleFlag = false;

int nSmpl = 1;

float sample;

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

void setup()

{

Serial.begin(9600);

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

while (Serial.read() != 'g'); // spin until 'g' entry

analogRead(A0); // read and discard to init ADC registers

MsTimer2::set(TSAMP\_MSEC, ISR\_Sample);

MsTimer2::start();

} // setup()

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

void loop()

{

syncSample();

// sample = analogRead(A0);

sample = random(0,256); // uniform random value (0..255)

displayData();

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

} // loop()

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

void displayData(void)

{

if (nSmpl == 1) Serial.print("\nn\tsample\n");

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

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

}

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

void syncSample(void)

{

while (sampleFlag == false); // spin until ISR trigger

sampleFlag = false; // disarm flag: enforce dwell

}

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

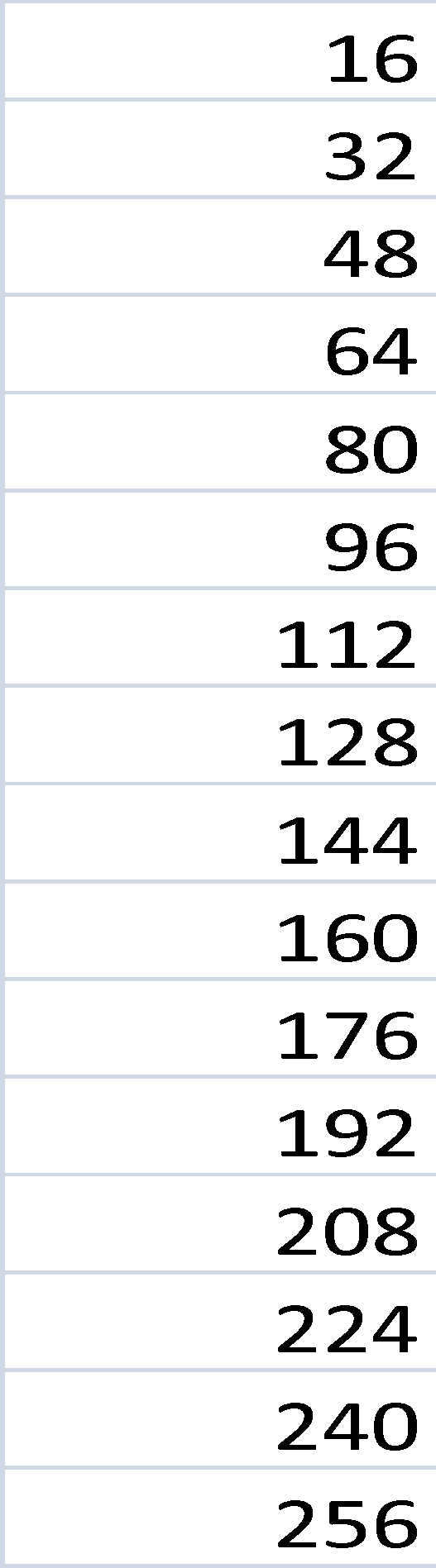
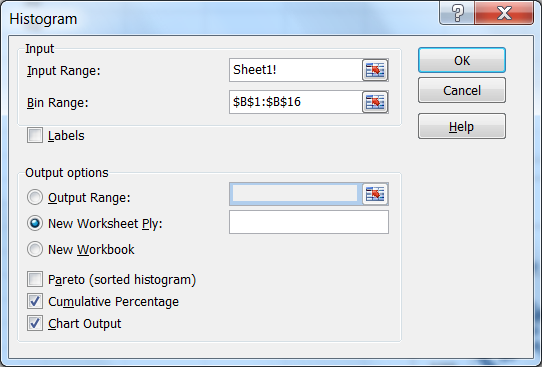
void ISR\_Sample() { sampleFlag = true; }

**Description of the code:**

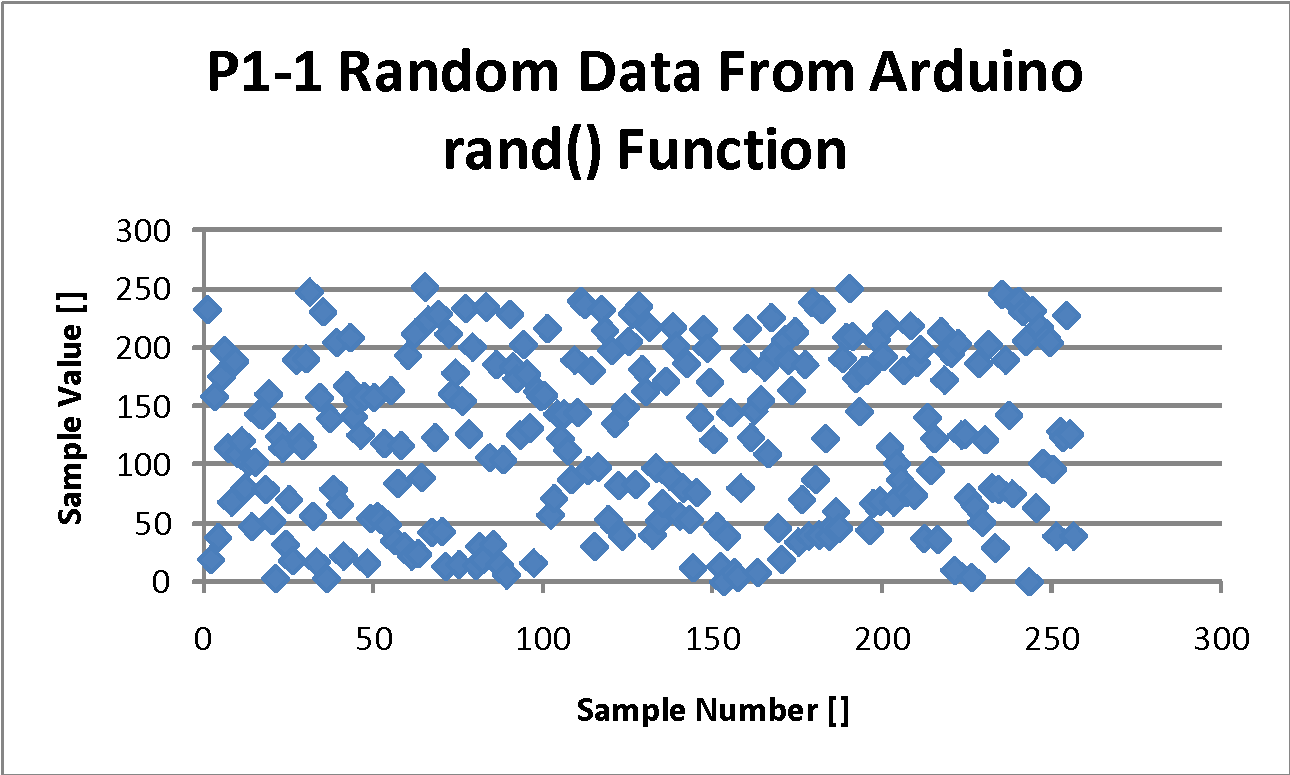
The code is essentially that generated in Lab 1 with replacement of the LM61 analog read by a call to random() for a variable uniformly distributed over 0…255. Sample synchronization and display tasks have been recast as functions to streamline loop() code.

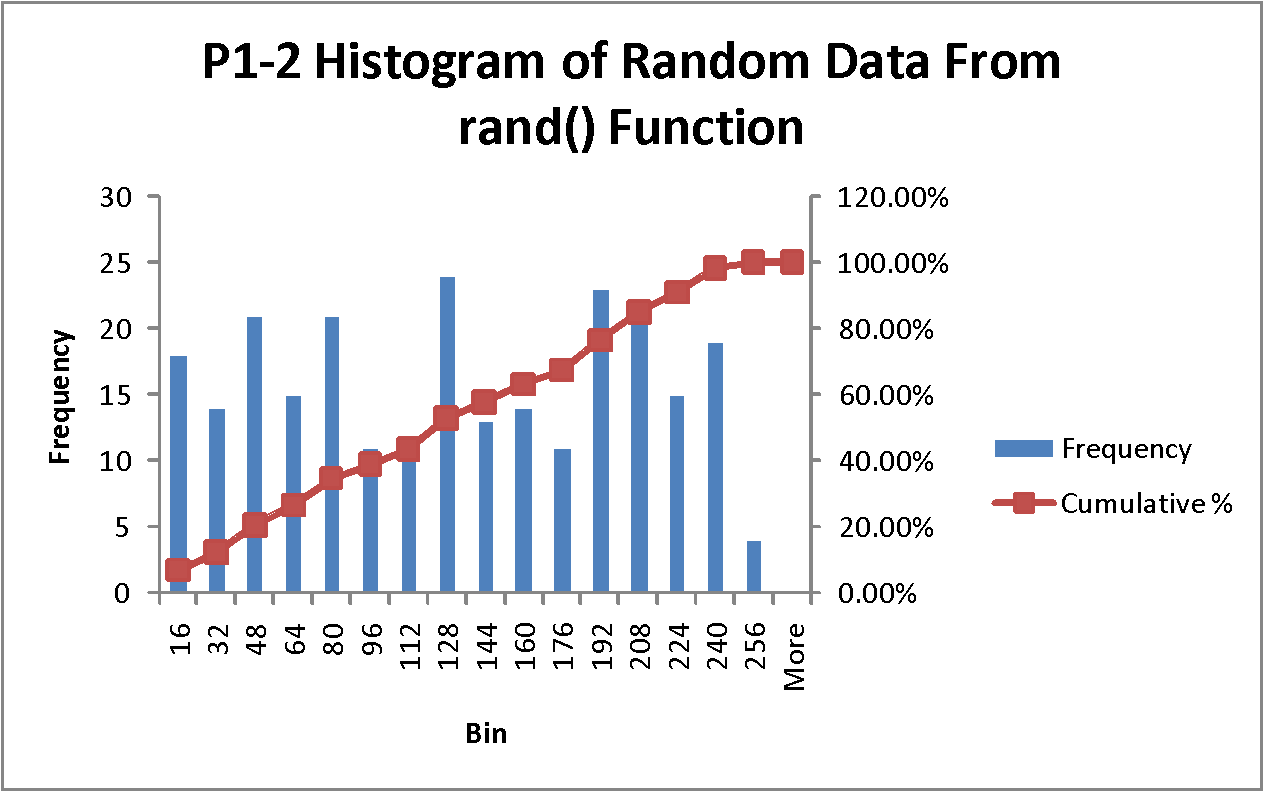
**Procedure:**

1. Setup the Arduino board, connect USB cable
2. Paste the code from the text box into the Arduino editor IDE.
3. Upload the program, fix any errors, bring up the serial monitor and trigger the start of the Arduino code.
4. Copy and paste the 256 random values into Excel and plot them as histogram and as a scatter plot X-Y using dots as data point markers, not lines.
5. To make a histogram, use Data -> Data Analysis -> histogram.
6. When making the histogram, check boxes for Cumulative Percentage and for Chart Output.
7. Get nicely spaced bins for the histogram by creating a list of bins starting at 16 and stepping by 16 to 256.
8. If you do not see a tab for Data Analysis under Data in Excel, go to Excel options (under File or under the Office button) and go to Options, Add-ins, Manage Add-ins (click Go) and check the box to Add Analysis Toolkit.



1. **P1-1 Include an Excel plot of the random data versus sample number in your report.**
2. **P1-2 Include an Excel plot of the histogram (both cumulative and bin data) in your report.**
3. Sample plots: Note: include axes labels in your plots





**SECTION 2a – Assessing sensor noise, accuracy, and signal to noise ratio (SNR)**

**Overview:**

In this section, you will measure the temperature sensor signal to characterize the signal to noise ratio (SNR) of the sensor and ADC combination. The temperature of the sensor is held at a constant temperature by covering the sensor or placing it in a box to protect it from air currents (e.g. from the room HVAC system or the PC’s cooling fan). Verify that sensor signal is at steady state (not increasing or decreasing during the data collection) by looking at the values you collect. In this case, you want to see a stationary signal.

**Program Description:**

Comment deselect the call to random(). Comment select the call to analogRead(A0). Note that A0 is read and the result not recorded in setup(). This first call initializes the internal registers of the ADC. Subsequent calls need no further initialization.

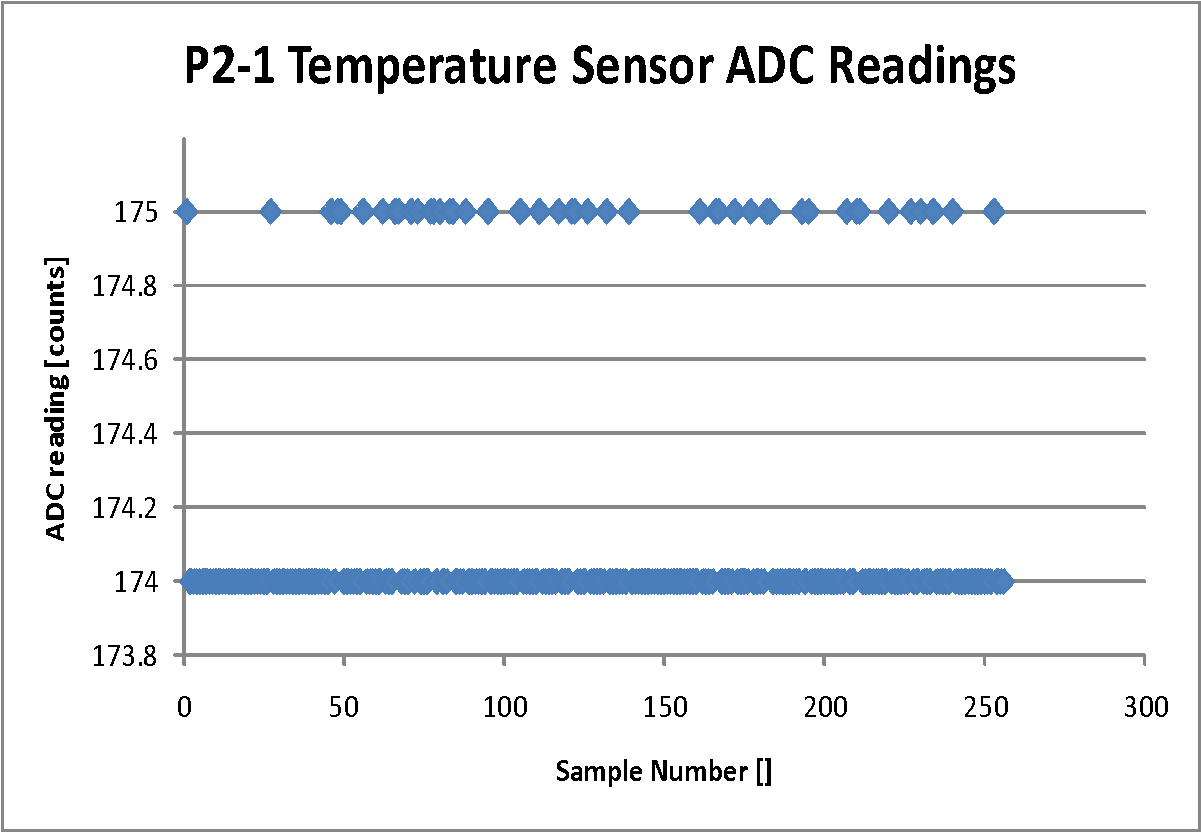
**Procedure:**

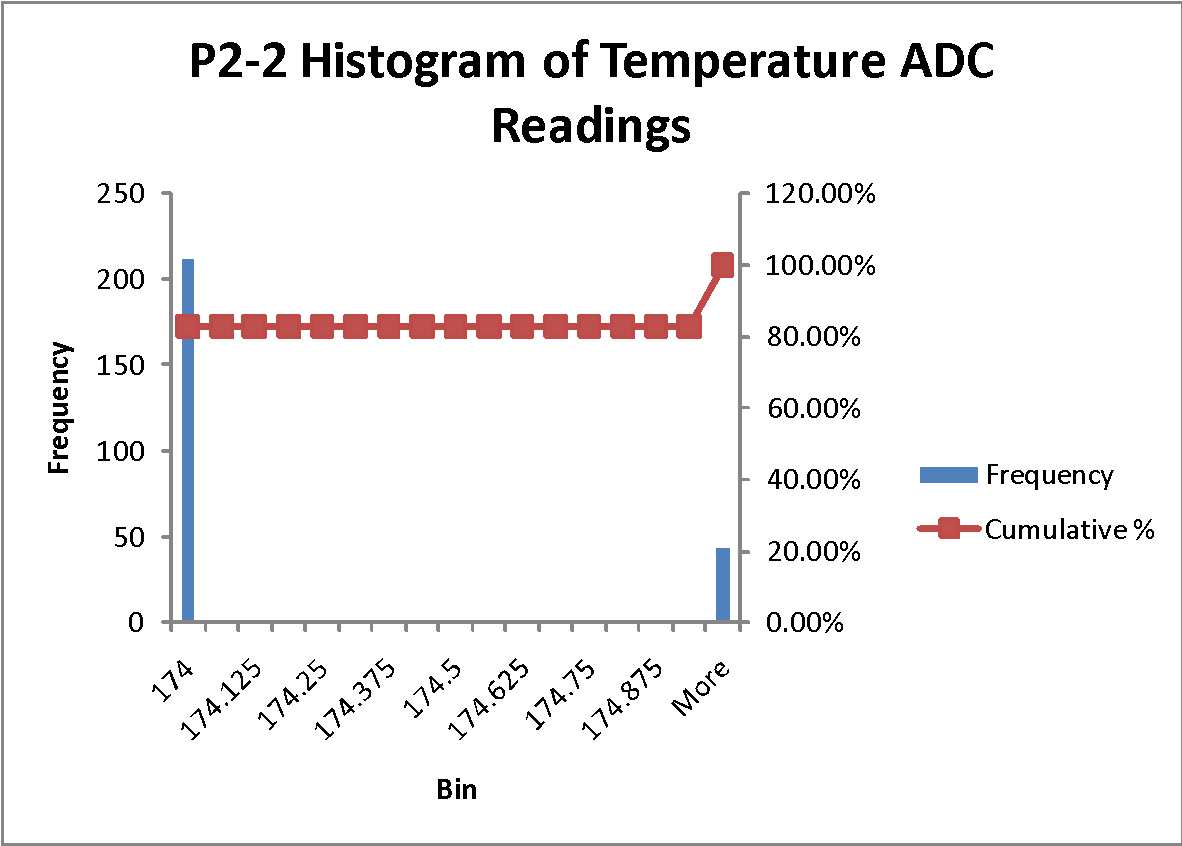
1. Connect the temperature sensor in the same manner as in Lab 1. POLARITY MATTERS !!!!
2. Make the code modifications to the file.
3. Upload the program, fix any errors, bring up the serial monitor and trigger the start of the Arduino code.
4. Copy and paste the 256 sensor reading values into Excel and plot them as histogram and as a scatter plot X-Y using dots as data point markers, not lines.
5. **P2-1 Include an Excel plot of the sensor data versus sample number in your report. Use a descriptive title for your plot such as: “Raw data from 256 samples of temperature sensor ADC reading without dithering (Tsample = 100msec, ADCref=5.0 volts)”**
6. **P2-2 Include an Excel plot of the histogram (both cumulative and bin data) of your temperatures samples in your report. Use a descriptive title for your plot such as: “Histogram of 256 samples of temperature sensor reading without dithering   
   (Tsample = 100msec, ADCref=5.0 volts)”**
7. Assume that the signal in the data is the mean value. In Excel, compute the mean, standard deviation, signal to noise ratio from the measured data. Variation in this case is due to signal and quantization noises. Compute the mean, standard deviation from theoretical quantization alone, and the resulting signal to noise ratio.
8. **Provide a table T2-1 of your calculations as shown below: (your numbers will be different).**

**Sample Plots and Tables**

**Table 2-1 Calculations based on raw temperature sensor ADC readings**

|  |  |  |
| --- | --- | --- |
|  | Value | formula |
| mean | 174.18 | AVERAGE(A1:A256) |
| stdev | 0.378 | STDEVA(A1:A256) |
| SNR (dB) raw | 53.3 | 20\*LOG10(C2/C3) |
| Quantization Noise (in LSB) | 0.29 | 0.29 |
| SNR (dB) Qn only | 55.6 | 20\*LOG(C2/C5) |
| Typical Error (in LSB) | 0.024 | C3/SQRT(256) |





**Observations**

1. If all is working well, you will see that the temperature sensor ADC reading is very stable (plus or minus one or two ADC counts) and that the computed standard deviation (e.g. 0.378) is roughly about the same size as the theoretical quantization.
2. The histogram of the ADC readings does not look like random noise, e.g. a normal distribution bell curve or a uniform distribution as in the previous section, instead, it looks like two or three distinct spikes.

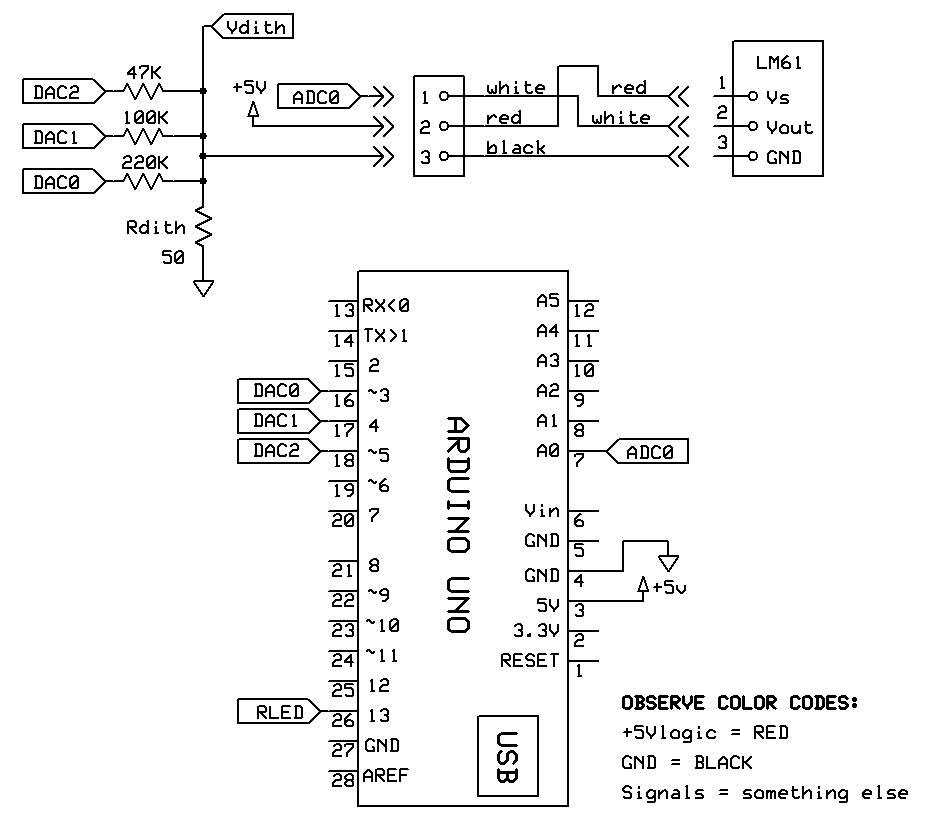
**SECTION 2b – Adding dithering noise to allow averaging and higher SNR.**

**Overview**

In the data you just collected, the histogram of data points most likely did not have a distribution that looked like random noise with a wide variety of different values. Because of this, it is not possible to directly average the sample points to reduce quantization noise. Dithering noise must be added to the signal before the ADC conversion for meaningful averaging to be done. For example, averaging 256 values that are all 174 will not increase the resolution of the measurement. It will simply give you a value of 174.

**Dither Voltage Generator Circuit Using Resistor Digital to Analog Converter**

A digital to analog converter (DAC) can be created using a few resistors and a few digital outputs from the Arduino. The digital outputs are connected through different value resistors creating a voltage divider fed by three different signals. See the figure below.



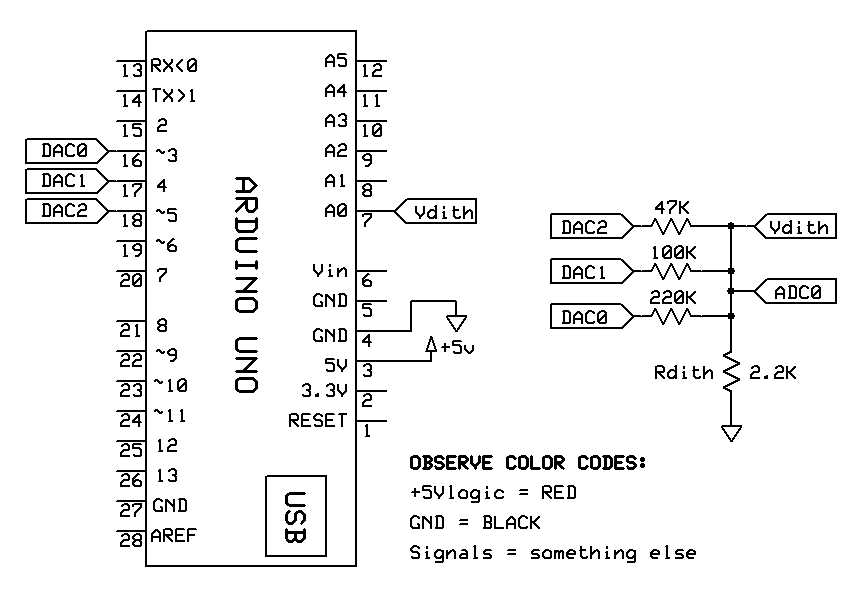
Arduino pins 3, 4, and 5 (DAC0, DAC1, DAC2, respectively) are configured as digital outputs. When DAC1and DAC2 are set to LOW (0 volts) and DAC0 is set to HIGH (+5 volts), a voltage divider circuit is formed where the +5 volts from DAC0 is divided according to the ratio   
50 ohms / (50 ohms+220kohms). This results in a voltage of 1.25 mV at the Vdith node, which is the return reference (GND) of the LM61. Because the LM61 output voltage is enforced between its local GND and Vout, the temperature sensor voltage above ground by Vdith.

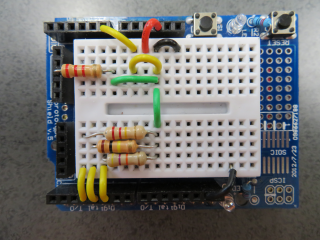
DAC1 is attached to half the divider resistance of DAC0 and DAC2 is attached to one fourth the resistance of DAC0. The pins then make binary weighted voltage contributions to Vdith. Eight states are possible resulting in Vdith from 0 (all LOW) to 7 x 1.25 = 9.75 mV (all HIGH).

By controlling the DAC(0..2) pins, a ramp waveform can be impressed on the LM61 output that is about 1 LSB in amplitude when using the DEFAULT 5 volt reference for the A/D converter. While not a random voltage, it has the same histogram as a uniformly distributed random signal.

**Dither Circuit Testing**

The dithering voltage is too small to measure with the ADC. Verify dithering circuitry by building a modified version with Rdith = 2.2K. This makes a dithering voltage large enough (0 to 55 mV) to be read by the Arduino ADC. Test by augmenting your existing code with the additional declarations, modified setup() and loop() functions and new testDitherRamp() function shown below. Retain the previous utility functions shown only as prototypes.



****

**Code for Dither Circuit Testing**

// file: Lab2\_dither\_average.ino

// created by: Clark Hochgraf 150901

// modified by: David Orlicki 170817

// purpose: creates a dither signal and reads it back into the ADC

#include <MsTimer2.h>

const int DAC0 = 3; // PD3, 220K resistor ladder weight

const int DAC1 = 4; // PD4, 100K resistor ladder weight

const int DAC2 = 5; // PD5, 47K resistor ladder weight

const int LM61 = A0; // LM61 analog input to Arduino

const int TSAMP\_MSEC = 10;

const int NUM\_SAMPLES = 256;

volatile boolean sampleFlag = false;

int ramp, nSmpl = 1;

float sample;

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

void setup()

{

pinMode(DAC0,OUTPUT); digitalWrite(DAC0,LOW);

pinMode(DAC1,OUTPUT); digitalWrite(DAC1,LOW);

pinMode(DAC2,OUTPUT); digitalWrite(DAC2,LOW);

analogRead(LM61); // read and discard to prime ADC registers

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

} // setup()

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

void loop()

{

syncSample();

// sample = analogRead(A0);

// sample = random(0,8); // uniform random value (0..7)

sample = testDitherRamp(); // Rdith = 2.2K

displayData();

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

} // loop()

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

float testDitherRamp(void)

{

int ramp = (nSmpl % 8); // ramp 0..7 using Modulo function

digitalWrite(DAC0, (ramp & B00000001)); // LSB bit mask

digitalWrite(DAC1, (ramp & B00000010));

digitalWrite(DAC2, (ramp & B00000100)); //MSB bit mask

return analogRead(LM61);

}

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

void displayData(void) {}

void syncSample(void) {}

void ISR\_Sample() {}

**System Description**

Output pins for the dither circuit are defined and initialized in setup(), based on pin number aliases defined as global constants. For dither circuit testing the sample interval is set to 10 msec (This experiment is decoupled from the LM61 thermal time constant.)

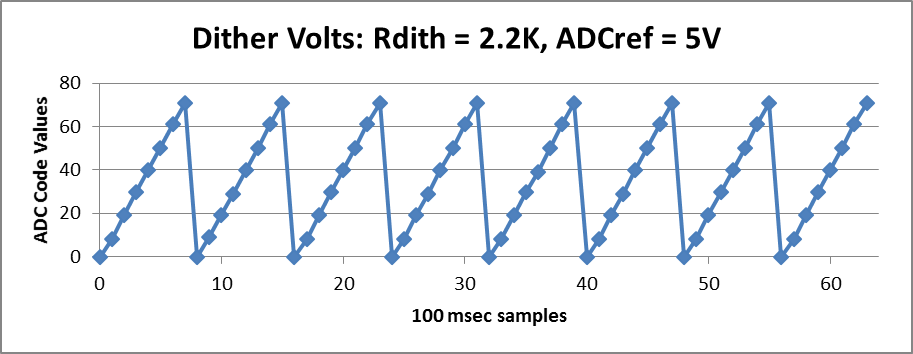
The dither ramp count 0..7 is created as the base 8 modulo of the sample count. The most significant bit of the DAC output is obtained by bit masking ramp with binary ‘100’. If the third bit of ramp is high (cases 4, 5, 6, and 7), DAC3 is set HIGH. Otherwise, DAC2 is low. DAC1 and DAC0 are similarly set based on bits 1 and 0, respectively.

The analog dither voltage Vdith is created by summing DAC2, DAC1 and DAC0 with binary weights of 4, 2, and 1, respectively. The weights are enforced by the values of the resistor divider taps. Vdith is the superposition of three resistor divider outputs. Rdith is small enough relative to the weighting resistors that the individual paths are approximately independent.

**Procedure:**

1. Build the voltage divider circuit on your proto board with Rdith = 2.2kohms and the ADC channel 0 connected to Vdith.
2. Paste the code from the text box above into the Arduino IDE. Replace the prototype functions near the end with their full functions from your earlier work. Upload, compile, and debug the code. Launch the serial monitor.
3. Collect 256 samples of dither voltage data and plot them in Excel.
4. **P2-3 In your report, provide an Excel plot of dither data versus sample number. Use a descriptive title for your plot such as: “Dither Volts: Rdith = 2.2K, ADCref = 5V” (Tsample = 100msec, ADCref=5.0 volts)”**
5. **P2-4 Include an Excel plot of the histogram (both cumulative and bin data) of your dither voltage samples in your report. Use a descriptive title for your plot such as: “Histogram of 256 samples of dithering voltage signal using 2.2kohm resistor as read by ADC (Tsample = 100msec, ADCref=5.0 volts)”**
6. In Excel, compute the mean, standard deviation, and variance of the large dither signal. Explain these results in your report.

The dithering voltage should look something like the following. Your results may differ.

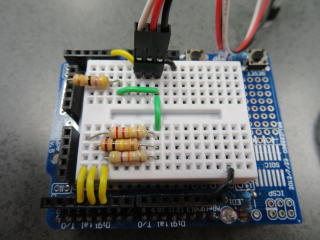


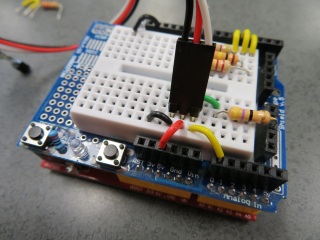
**Reading Temperature Signal Using Dithering Voltage Applied to ADC Input**

Now that you have verified the dithering system operation, it is possible to use the dither voltage to reduce system noise.

To reduce quantization noise, many samples must be taken and averaged together to create a single data point. The code will still collect and print 256 total data points, but each data point will consist of the average of 160 sub-readings. Each sub-reading will include a scaled version of the dither voltage you measured in the previous section.

Start with a dither resistor value of about 50 ohms for an ADC voltage reference of 5V. The peak dither voltage is then about 9 mV, or 2 LSB. See the following proto board implementation example. The unconnected black wire near the sensor connector is a visual orientation clue.





**Program Description:**

When the average of many ADC readings is done, the resulting value should have more resolution (i.e. be precise to more decimal places) than a single reading of the ADC. Since returned samples are averages of multiple readings, the sample variable is declared float.

In the main loop, instead of populating a sample with a single call to the analogRead() Arduino function, the sample is taken as the return of a custom function analogReadDitherAve() that collects and averages NUM\_SUB\_SMPL dithered sensor readings. Eight step dithering is implemented as during the circuit test. For each dither level an individual ADC reading is made. The readings are summed and divided by NUM\_SUB\_SMPL to return an average value.

**Code for Sensor Quantization Noise Reduction Using Dither-Averaged Readings**

// file: Lab2\_dither\_average.ino

// created by: Clark Hochgraf 150901

// modified by: David Orlicki 170817

// purpose: drive and read dither-averaged LM61 readings

#include <MsTimer2.h>

const int DAC0 = 3, DAC1 = 4, DAC2 = 5, LM61 = A0;

const int TSAMP\_MSEC = 100;

const int NUM\_SAMPLES = 256;

volatile boolean sampleFlag = false;

int nSmpl = 1;

float sample;

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

void setup()

{

configureArduino();

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

} // setup()

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

void loop()

{

syncSample();

sample = analogReadDitherAve(); // Rdith = 50

displayData();

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

} // loop()

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

float analogReadDitherAve(void)

{

const int NUM\_SUB\_SMPL = 160;

int ramp;

float sum = 0.0;

for (int i = 0; i < NUM\_SUB\_SMPL; i++)

{

ramp = i % 8; // ramp 0..7 using Modulo function

digitalWrite(DAC0, (ramp & B00000001)); //Bit mask

digitalWrite(DAC1, (ramp & B00000010)); //Bit mask

digitalWrite(DAC2, (ramp & B00000100)); //Bit mask

sum += analogRead(LM61);

}

return sum/NUM\_SUB\_SMPL; // averaged subsamples

}

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

void configureArduino(void)

{

pinMode(DAC0,OUTPUT); digitalWrite(DAC0,LOW);

pinMode(DAC1,OUTPUT); digitalWrite(DAC1,LOW);

pinMode(DAC2,OUTPUT); digitalWrite(DAC2,LOW);

analogRead(LM61); // read and discard to prime ADC registers

Serial.begin(9600);

}

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

void displayData(void){}

void syncSample(void) {}

void ISR\_Sample() {}

**Procedure**

1. Change the dither resistor to 50 ohms and re-wire the circuit so that the ADC channel A0 reads the temperature sensor.
2. Paste the code from the text box above into the Arduino IDE. Upload the code, populate the prototypes, and launch the serial monitor.
3. Set up the temperature sensor so that it covered up and has a constant temperature. You will need to protect the temperature sensor from even weak air currents, such as your hand passing over it or the fan from the PC. Try putting the sensor inside a plastic bag with some air space in the bag.
4. Collect 256 samples of data and plot them in Excel in a manner similar to plots P2-1 and  
   P2-2. The values should print as ADC counts, but with decimal places of resolution to the right of the decimal point.
5. **P2-5 In your report, provide an Excel plot of the sensor data versus sample number. Use a descriptive title for your plot such as: “256 samples of temperature sensor ADC reading with dithering and 160 points averaged per sample (Tsample = 100msec, ADCref=5.0 volts)”**
6. **P2-6 Include an Excel plot of the histogram (both cumulative and bin data) of your temperatures samples in your report. Use a descriptive title for your plot such as: “Histogram of 256 samples of temperature sensor reading with dithering and 160 points averaged per sample (Tsample = 100msec, ADCref=5.0 volts)”**
7. Collect 256 samples each for sub-sample rates of 1 (single ADC read), 16, 160, and 256. Assume that the signal in the data is the mean value. Copy the data to Excel and compute the entries for each element of Table 2-2 below. Partial example entries are shown for a 256x oversampling case. Your data will vary.

**Table 2-2 Calculations based on temperature sensor ADC readings using dithering and 256 points averaged per sample.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | value  1x sampling | value  16x sampling | value  160x sampling | value  256x sampling |
| mean (ADC CV) | 187.4609375 | 188.8936078 | 188.865 | 188.8331641 |
| stdev raw (ADC CV) | 1.451989001 | 0.3221609952 | 0.04922138862 | 0.03360305246 |
| SNR (dB) raw | 42.21894916 | 55.36288607 | 71.67995242 | 74.9939908 |

1. **Q2-1 What is the value of the SNR of the temperature signal now that dithering has been added and 160 points averaged per sample? Is this consistent with the 1x and 16x oversampled cases? Why did the SNR change? Is the value the same as theory would predict or is it different?**

Recall the typical error formula:

**Note**

Typical error is, in essence, the noise level in the estimate of the true mean, after averaging N subsamples

**SECTION 2c – Assessing if Dithering Is Required**

**Overview**

It is a reasonable question to ask whether dithering is necessary or whether simply averaging subsamples is sufficient. To assess this, you will now conduct a set of experiments with and without dithering, and with and without averaging.

It is easy to see the effect of ADC quantization when the sensor is produces a slowly changing signal that is passing through a quantization level. In this section, you will warm up the sensor for a few seconds by pinching it with your fingers, and then let the sensor slowly cool down in free-air (non-moving air). This will give a slowly falling temperature reading that passes through quantization levels.

**Code for Comparing Simple Sampling, Averaging, and Dither-Averaging**

float analogReadDitherAve(void)

{

// SINGLE SAMPLE: set NUM\_SUB\_SMPL = 1, ramp = 0 in for loop

// AVERAGE ONLY: set NUM\_SUB\_SMPL as desired, ramp = 0 in for loop

const int NUM\_SUB\_SMPL = 160;

int ramp;

float sum = 0.0;

for (int i = 0; i < NUM\_SUB\_SMPL; i++)

{

ramp = (i % 8); // ramp 0..7 using Modulo function

digitalWrite(DAC0, (ramp & B00000001)); // LSB

digitalWrite(DAC1, (ramp & B00000010));

digitalWrite(DAC2, (ramp & B00000100)); // MSB

sum += analogRead(LM61);

}

return sum/NUM\_SUB\_SMPL; // averaged subsamples

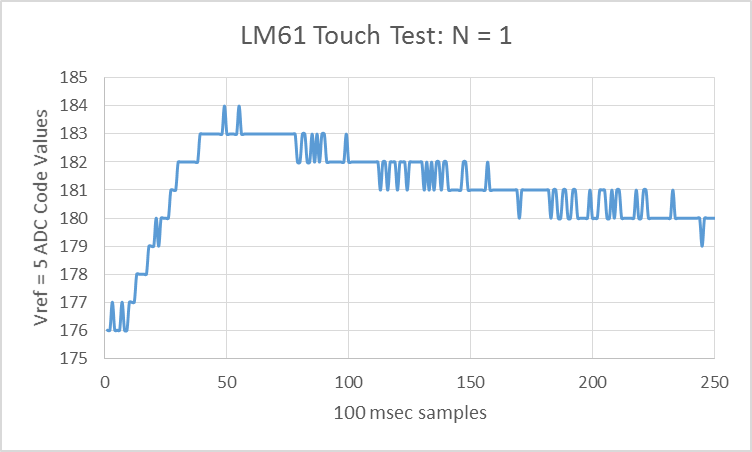
}

**Program Description:**

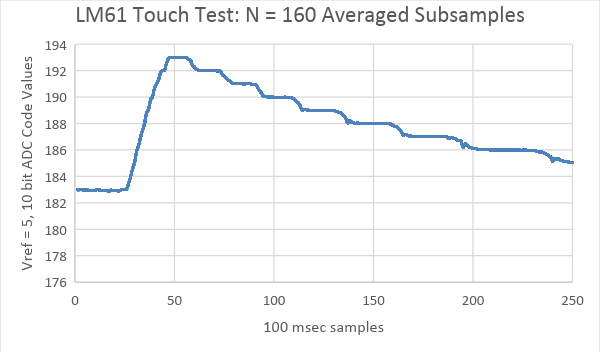
Use the dithered and averaged ADC code from the previous section. Setting ramp = 0 inside the loop holds all three DAC pins low, thereby disabling dithering. Setting NUM\_SUB\_SMPL = 1 results in the return of a single sample with minimum dither level.

**Procedure:**

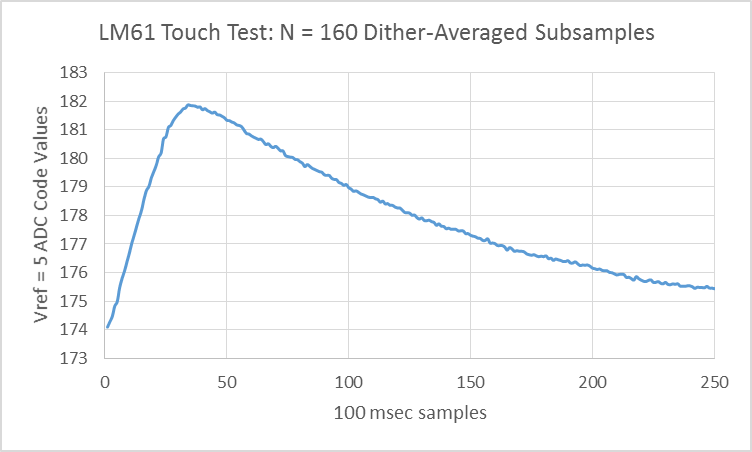
1. Modify the function analogReadDitherAve() for single sample return.
2. Upload and run the code. To provide a temperature signal, start the program running by entering ‘g’. Immediately pinch the temperature sensor between two figures for about 3 seconds until you see the ADC reading go up by at least three integer counts. Release the sensor and let it cool in free air. This procedure is called a ‘pinch test’ in what follows.
3. Paste the resulting data into Excel and plot the results. Use a descriptive title and label both axes. **Include this plot in your report.** Your plot should resemble the following.



1. Modify the code to collect data with 160 averaged, undithered subsamples. Conduct a pinch test. Paste the resulting data into Excel and plot the results. **Include this plot in your report.** Your plot should resemble the following.



1. Modify the code to collect data with 160 averaged, dithered subsamples. Conduct a pinch test. Paste the resulting data into Excel and plot the results. **Include this plot in your report.** Your plot should resemble the following.



1. **In your report, explain what features of your graph of the cool-down response without dithering indicate that dithering is needed.**

**SECTION 2d –Increased effective ADC bit resolution from dithering noise and averaging**

By dithering and averaging, the noise level in the previous section was reduced from a quantization noise level of 0.29 LSB (least significant bits) to a significantly lower typical error. One way of viewing this is that the ADC converter has increased resolution i.e. it has more bits (e.g. 8 bit vs 10 bits vs 12 bits).

If we treat all of the noise as coming from quantization, how many effective bits does the ADC have now that we have added dithering noise and averaged subsamples? Easy. The standard deviation of the baseline sampling system is 0.29LSB. If the stdev of the processed (averaged or dither-averaged) samples is 8x smaller, the LSB is effectively 8x smaller as well. For a fixed Vref, this corresponds to an apparent added 3 bits of ADC resolution. The improvement ratio may not be an even power of 2. If the improvement ratio is 9.67x, for example, then  
. If the original system was 10 bits, the new system has performance better than one of 13 bits, but less than 14 bits.

**Q2-4 Based on your experimental data, what is the equivalent number of bits in the ADC conversion when dithering noise is used and 256 samples are averaged?**

**SECTION 2e – Engineering Units of Celsius**

**Overview**

The data acquisition system you are developing is ultimately designed to work with temperature in degrees Celsius. Using the LM61 datasheet and the sensor characteristic, modify the code to read out directly in Celsius. Capture 256 sample sets in ADC code values, millivolts, and degrees Celsius with the sensor at equilibrium temperature and shielded from air currents.

...

syncSample();

sample = analogReadDitherAve(); // ADC code values

sample = sample; // convert from CV to millivolts

sample = sample; // convert from millivolts to degC

displayData();

...

**Procedure**

1. Make the appropriate modifications to the code.
2. Collect ADC code value, millivolt and degree Celsius records by comment selecting appropriate sample definitions.
3. Calculate values for a 1 degC signal excursion in each unit system.
4. **Provide a table T2-3 of your calculations of mean, standard deviation, typical error and SNR, expressed in millivolts. Assume a peak to peak temperature signal of 1 degC. Typical format (your numbers will vary).**

**Table 2-3 Temperature sensor readings by units with dither and averaging**

|  |  |  |  |
| --- | --- | --- | --- |
|  | ADC CV  (5 Vref) | Millivolts | Celsius |
| mean | 178.5 |  |  |
| stdev | 0.32 |  |  |
| 1 degC signal | 2.0 |  |  |
| SNR(dB) raw | 9.9 |  |  |
| Typical error (256 dthr-avg subsamples) | 0.053 |  |  |
| SNR(dB) (256 dthr-avg subsamples) | 31.5 |  |  |

1. **Q2-5 What is the effective temperature resolution of your final measurement system? Justify your answer.**

**SECTION 2f – Computing the variance of signal using the running sum**

**Overview**

So far, you have computed mean and standard deviation using Excel. It is possible to continuously compute the mean, standard deviation and variance directly in your program.

**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.

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 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.

If i==1 then

Mean=x[i];

RunningSumVar=0;

Variance =0;

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;

The direct, but numerically disadvantaged method is coded in C in the box below. Note that the variables mean and stdev are declared globally and updated by the accumulateStats() function. Results are passed by reference through the function parameter list. Although the function could operate directly on the global variables, the pass by reference method will provide modularity benefits in future labs.

Note that the sample sequence is generated by a mathematical expression rather than from ADC readings. This provides a repeatable test vector with known statistics that simplifies algorithm testing. (One can hand calculate the statistics.)

**Procedure**

1. Add display functionality for mean and stdev and integrate code from your earlier work for the prototype functions shown. Upload, run and debug the code. Import and chart the data in Excel. Verify correct calculation of steady state mean and standard deviation.
2. Implement the numerically advantaged method shown in the pseudo code by completing the stub function recurseStats(), which uses the same pass by reference parameter list as accumulateStats(). Confirm performance in Excel.
3. **Q2-6 What was the value of the variance that you calculated using the recursive running variance method in your code? How did it compare to Excel’s variance calculate on the same data? Was there any difference?**

**Code for Running Statistics Development**

**m**

// file: Lab2\_statistics.ino

// created by: David Orlicki 170817

// purpose: calculate running mean, stdev with parms pass by reference

#include <MsTimer2.h>

const int DAC0 = 3, DAC1 = 4, DAC2 = 5, LM61 = A0;

const int TSAMP\_MSEC = 10, NUM\_SAMPLES = 256;

volatile boolean sampleFlag = false;

int nSmpl = 1;

float sample, mean, stdev;

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

void setup()

{

configureArduino();

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

} // setup()

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

void loop()

{

syncSample();

sample = 25+5\*sin((nSmpl/50.0)\*TWO\_PI); // simple model degC

accumulateStats(sample, mean, stdev); // poor numerical properties

// recurseStats(sample, mean, stdev); // good numerical properties

displayData();

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

} // loop()

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

void accumulateStats(float smpl, float &mean, float &stdev)

{

static float sum = 0.0, sumSquares = 0.0;

sum += sample;

sumSquares += sample\*sample;

mean = sum/nSmpl;

if (nSmpl > 1) stdev = sqrt((sumSquares - sum\*sum/nSmpl)/(nSmpl-1));

}

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

void recurseStats(float smpl, float &mean, float &stdev)

{

static float oldMean, runSumVar, oldrunSumVar;

mean = 0.0;

stdev = 0.0;

}

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

void displayData(void){} // add mean, stdev

void syncSample(void){}

void ISR\_Sample(){}

**References:**

Here are a couple of references on computing sample variance.

Chan, Tony F.; Golub, Gene H.; LeVeque, Randall J. (1983). Algorithms for Computing

the Sample Variance: Analysis and Recommendations. The American Statistician 37, 242-247.

Ling, Robert F. (1974). Comparison of Several Algorithms for Computing Sample Means and Variances. Journal of the American Statistical Association, Vol. 69, No. 348, 859-866.

**Write Up:**

Include the cover sheet (following page) on top of your lab report.

**For a lab report grade of B, include the following.**

1. Your lab report must be submitted in MS Word using the IEEE Journal Transactions Format. A template Word document is posted on MyCourses for you to download.
2. Your lab report must use the two column format as given in the template.
3. Include a simple title. Include the names of all your lab partners as co-authors.
4. Include the requested plots and tables
5. Fully label the vertical and horizontal axes of graphs. For example, in some plots the vertical axis is the A/D converter reading as an integer from 0-1023 which represents temperature and the horizontal axis is the sample number. If the vertical axis is in engineering units, label the axis with the units.
6. 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.
7. Provide responses to each of the questions.
8. When responding to each question, restate the question in your own words and answer the question. Do not simply copy and paste the question text. For example, Q-1 was “What happened to the signal to noise ratio when the dither signal was added?”

**Do This:**

*As can be seen in figure X and in table Y, the addition of dithering noise caused the signal to noise ratio to initially increase. However, later when….. was done, the SNR went….*

**Not This:**

*“Q-1 What happened to the signal to noise ratio when the dither signal was added?”*

*It went down.*

1. Briefly discuss of your results. In addition to a general discussion of your results, address the following:

**For a lab report grade of A, include the following.**

Your report will have section headings (see this handout for examples).

Your report need not include the procedure items from this handout.

Your report will not have more than 4 pages. Note that in two column format this is the same as approximately 8 pages of single column data.

Provide correct and well explained answers to the following questions. In your answers, restate the question.

1. For a nasal airflow temperature measurement system, the signal may be a little as 1 degree Celsius (i.e. the observed change in temperature from breathing in to breathing out).
   1. Assuming that no dithering or averaging is done, and a 10 bit A/D conversion is used, what is the signal to noise ratio in dB?
   2. If dithering and averaging is used, what is the SNR?

Note: For both cases, the signal is the variation in temperature (i.e. 1 Celsius) not the nominal temperature (i.e. 26 Celsius).

1. What is the approximate accuracy of the temperature measurement system assuming that the ADC is perfectly accurate and that the temperature sensor (LM61C) has accuracy as specified in its datasheet? Express this in degrees C and in millivolts.
2. What is the approximate precision of the temperature measurement system including the effect of the ADC, dithering, and averaging? Express this in degrees C and in millivolts. Be sure to state your assumptions (e.g. How many samples you averaged).
3. Review the objectives and the learning outcomes at the beginning of the handout. Have you addressed each of these topics in your report?
4. Your lab partner believes that the averaging and dithering process increased the accuracy of the measurement system. You believe that the measurement system has been made more precise but not more accurate. Write one to two paragraphs to explain your position to your lab partner.
   1. Provide supporting data (test results) and/or theory/equations to support any statements you make. Specifically include a section of results where you calculate the accuracy of the measurement system in degrees C and the precision of the system in degrees C.
   2. State any assumptions you made to come up with these results.

**Write Up: Lab #2 Noise, SNR measurements**

Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| LABORATORY GRADE | |
| Plots and tables – complete set, correct labels  Answers to questions– addresses specific question  Narrative– explains key results and the implications  Lab participation | \_\_\_\_\_ / 40  \_\_\_\_\_ / 20  \_\_\_\_\_ / 20  \_\_\_\_\_ / 20 |
| Final Grade | \_\_\_\_\_/ 100 |

1. P1-1, P1-2, P2-1, P2-2, P2-3, P2-4, P2-5, P2-6
2. Table T2-1, Table T2-2, Table T2-3
3. Q2-1 What is the value of the SNR of the temperature signal now that dithering has been added and 160 points averaged per sample? How does the SNR compare to when no dithering was used? Why did this change? Is the value the same as theory would predict or different?
4. Q2-2 Compare the SNR when there was no dithering and when dithering plus averaging was used. Which has better SNR, the signal without dithering noise or the signal with dithered noise added and 256 samples were averaged? How many dB better? Why? How does this compare the theoretical improvement?
5. Q2-3 If only 16 samples are averaged, what is the SNR with dithering noise and 16 samples averaged? How does this SNR compare to when no dithering noise is added and no averaging is used?
6. Q2-4 Based on your experimental data, what is the equivalent number of bits in the ADC conversion when dithering noise is used and 256 samples are averaged?
7. Q2-5 What is the effective temperature resolution of your final measurement system? Justify your answer.
8. Q2-6 What was the value of the variance that you calculated using the running variance method in your code? How did it compare to Excel’s variance calculate on the same data?Was there any difference?