General Purpose Signal Acquisition and Visualization System

Aditya Thakkar thakkaap Lab Section 6

***Abstract*— This paper outlines a system which will allow the user to implement a an analog signal from nature into a digital signal which can be processed by a computer. Major components include a transducer, conditioning circuit, AD conversion and computer to graph. This can be used in many industries to analyze and process all sorts of natural phenomena.**

I. Introduction

To analyze and process data from the physical world, engineers and scientists often need to send numerical data to a computer. Once the discretized data has been sent to the computer, it can be processed. As computers are digital machines and physical phenomena are analog; a system that converts analog signals to digital signals is crucial.

Accurately modeling analog signals in a digital setting is a problem faced in many industries. As always, cost and quality must be balanced. The cost to implement an signal acquisition system increases with the fidelity of the system. The idea is to get the required conversion accuracy within the financial constraints set forward by the problem.

One example of a signal acquisition system is the CAMAC (Computer Automated Measurement and Control), which is a standard bus and modular crate electronics standard used in the nuclear physics industry. Another example is the PowerLab, which is a numerical data acquisition system used in life science research and development.

This paper outlines a design for a basic data acquisition system which can be manipulated to meet a wide range of technical specifications.

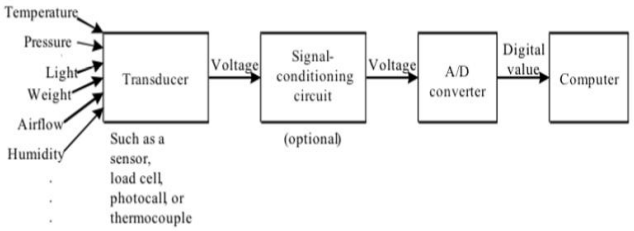
The system overview section will give a description of the system, the results section will outline the data obtained from the system, and the discussion section will compare these results with the expected values.

II. System Overview

The first main component of the system is a transducer. The transducer is a sensor that converts physical phenomena such as temperature, speed, weight or pressure into a measureable electrical signal, in the form of voltage or current. The next component is a conditioning circuit. Its purpose is to ensure that the voltage or current coming into the Analog/Digital Converter (ADC) is adequate enough for processing. The signal is then sent into the ADC, where the analog continuous signal is converted into a discrete digital signal, and sent to the computer for processing and analysis. For this paper,

A. Transducer

The transducer circuit is the component of this system that acquires the signal that will be processed later. Some examples of a transducer include a thermometer for measuring temperature or an accelerometer for measuring acceleration. The key feature about this system is that it can be adapted to fit any existing transducer. Depending on the transducer, the following component, the conditioning circuit may have to be



B. Esduino Xtreme

The analog to digital conversion is done by the analog input pins on the EsduinoXtreme, which is a 9S12GA240 microcontroller based on an Arduino design. It can be programmed to perform a number of different embedded systems tasks.

It can be purchased for $59.00 USD and can be programmed to have up to a 25 MHz clock speed. It also has 12 bit analog to digital conversion pins, but for this application, we will be using only 10 bits. One of the main reasons for choosing this board was its robustness. It can still operate in -40 to 85 degrees Celsius. It can be programmed in C/C++ or Assembly.

The Esduino will handle most of the computations required

Aditya Thakkar is a student in his 3rd year of electrical and biomedical program at McMaster University in Hamilton, Ontario. He has internship experience in software development and surgical robotics research.

for this application. The preconditioning circuit will limit the voltage coming into the analog pins of the circuit in between 0 and 5 V. For this application, the frequency of the AC signal can be up to 32 Hz to get both the shape and frequency of the graph reproduced in MATLAB, as the sampling rate for this application is 320 Hz. The Nyquist Theorem states that we must use a sampling frequency of 10 times the maximum input frequency to get an accurate representation of the signal's shape and frequency. If the parameter of interest is frequency, then a sampling frequency of 2 times the maximum input frequency can be used.

The signal from the conditioning circuit will come into analog pin 4, and then it will go through a successive approximation algorithm to get made into a discrete digital value., using a 10 bit resolution.

The maximum quantization error using this method is 5/2^10, which is equal to 0.0048828125 volts. The higher range of voltage, 5 V is converted to 1023 in the Esduino, while the lowest acceptable voltage, 0 V will be converted to 0 in the Esduino.

Analog Voltage	***10 Bit ADC (Hex)***
0	0
1.25	0x0FF
2.5	0x1FF
3.75	0x2FF
5 V	0x3FF

Table 1: Analog Voltages to the 10 bit values produced by the analog to digital conversion pins on the Esduino – in Hexadecimal

The sampling rate of the A/D converter is the amount of times the Esduino samples the input voltage to turn it into an integer from 0 – 1023. For this case, we have chosen a sampling rate of 320 Hz, as it will allow accurate frequency and shape reproduction of a signal up to 30 Hz in frequency.

A button is fed into pin D13 and used to start and stop serial transmission. This functionality is implemented using an interrupt, which goes off when the button is pressed. The button is an active high button.

The control for the button is based on a variable called onOff, which is a counter of how many times the button has been pressed. If it has been pressed a multiple of 2 times, the serial communication will remain or turn on. If not, then serial communication is terminated. This is checked using the modulus operator.

This discretized data is then sent to a computer for further processing with serial communication. The baud rate is set at 19200 bits/second, as this allow a very quick transmission of the data, so that the data can be processed as quickly as possible.

Setting the clock is the first step of the algorithm. The clock is set using the following expression:

$$Bus\ Clock = \frac{2 * f_{REF} * (SynDiv + 1)}{8}$$

To set the clock at a certain frequency, we just have to set the value for SynDiv. For this application, we need a 4 MHz bus clock frequency, so we set SynDiv to be 15.

The algorithm on the Esduino is constantly polling analog pin 4, always checking what the voltage is at the specified sampling frequency. There is also a button based interrupt. The button is an on or off switch that tells the Esduino whether to use the A/D converter or not. The default position of the switch is on, and then can be changed as the user presses the switch.

Due to the nature of the SCI_OutUDec function used to transmit the 10 bit number from the ADC, we also must pad the transmission with zeros if the number is less than 1000. For example, it will take 10/19200 seconds to transmit one ASCII digit using the method above. But if the number is less than 1000, then we are only transmitting 3 digits. If it is greater, than we are transmitting 4. This 10/19200 second difference is enough to cause a substantial error as higher frequencies are passed into the Esduino. To avoid this, instead of transmitting say 913, we transmit 0913. This makes the transmission time an even 40/19200 for every number we send out, thus avoiding the error described earlier.

On the PC side, this 0913 is read as 913 and does not need any special processing.

To verify the sampling frequency, we are also swapping the state of pin D13 on the Esduino. Every time the pin changes state, a sample has occurred. This pin can then be wired up to an oscilloscope to be analyzed to ensure that the correct sampling frequency of 320 Hz has been met. See Results section for more details.

Serial communication is used to transmit the value read into the Esduino. We are transmitting an 8 bit ASCII value for each digit, each of which have a stop and start bit, totaling 10 bits.

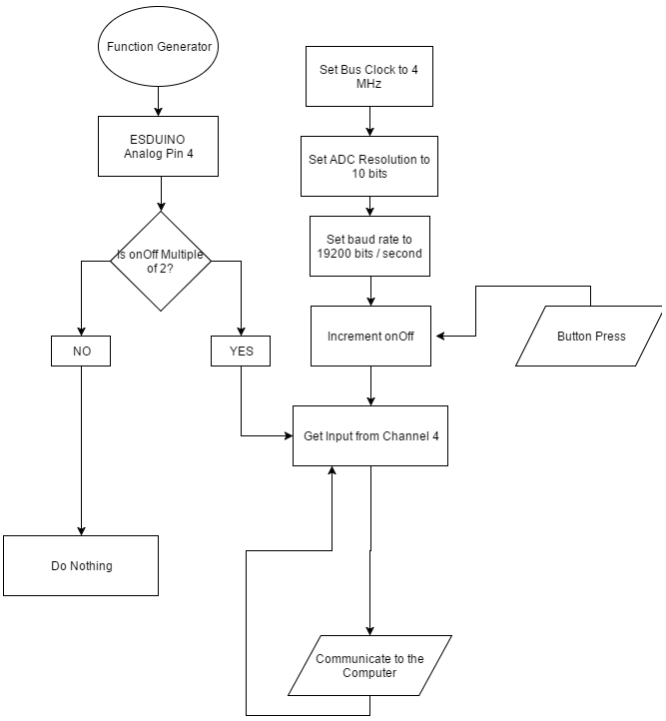


Figure 1: Algorithm Flow Chart for Esduino Code to Read and Transmit Values from Analog Pin 4

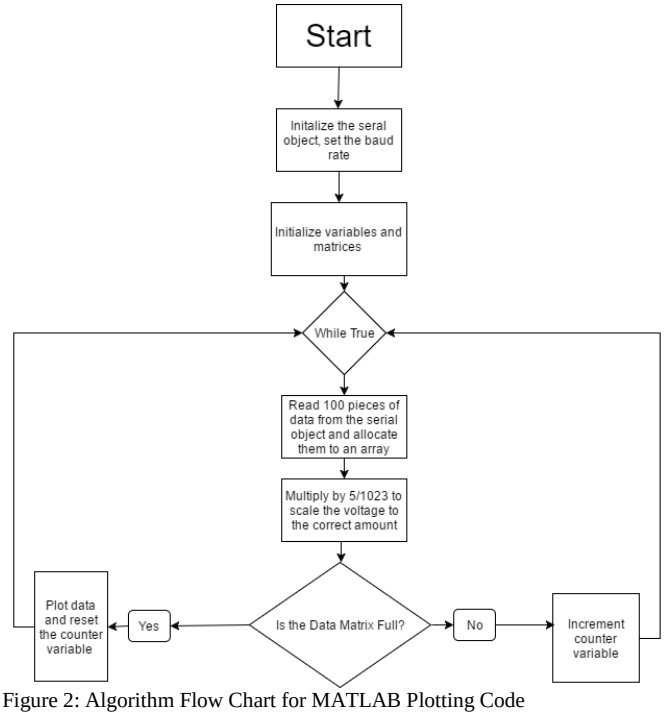


Figure 2: Algorithm Flow Chart for MATLAB Plotting Code

The baud rate is 19200 bits/second, ensuring a fast transmission rate. SCI_OutUDec sends out the four digits, and each are 10 bits long, along with a termination bit. This gives us a delay of 50/19200 seconds, along with a 6uS conversion time. To find the delay we must add to the code, we use the equation:

$$\frac{1}{320} = \frac{50}{19200} + 6us + Delay$$

Solving for the delay gives a 0.5 ms delay that we must add in to the code to get a sampling frequency of 320 Hz.

C. Computer and MATLAB Algorithm

The computer is a 64 bit Acer machine running Windows 10. MATLAB R2014 is being used to plot the data as it comes in though the serial terminal at a rate of 19200 bits/second.

The MATLAB algorithm updates every 0.103 seconds, and pulls in a new set of data to graph. The code to plot the data every 0.103 seconds can be found in the appendix.

We read the serial object, divide the number coming in from the Esduino by 5/1023, which gives the analog value from 0 to 5 V.

We read the serial input 320 times a second, and then fill up a data matrix. Once the data matrix is full, we then plot the values.

A flow chart for the MATLAB algorithm can be found next. Results from this plotting algorithm can be found in the results section of this paper.

III. Results and Experimentation

To ensure that the system works with the highest fidelity possible, we must perform unit testing on each part of this system.

First, we were to set the sampling rate and bus speed. To test this, an extra line of code was added, where the state of an LED was alerted every time a sample was taken from analog channel 4. This output was then sent to an oscilloscope to be verified. The oscilloscope HUD can be seen in figure 3.

The scope is displaying a frequency of around 160 Hz, which confirms that our sampling frequency is 320 Hz. This is due to the fact that we are alternating the state of the LED every time we take a sample, whereas the frequency measurement tool on the oscilloscope expects every 0 to 1 and back time as the period, and then converts that to a frequency.

The next test is to ensure that serial communication is working correctly. To do this, RealTerm was used to read the ASCII characters that the Esduino is transmitting. If they are from 0 to 1023, then the serial communication has been implemented correctly. A screenshot of the output can be seen in figure 4.

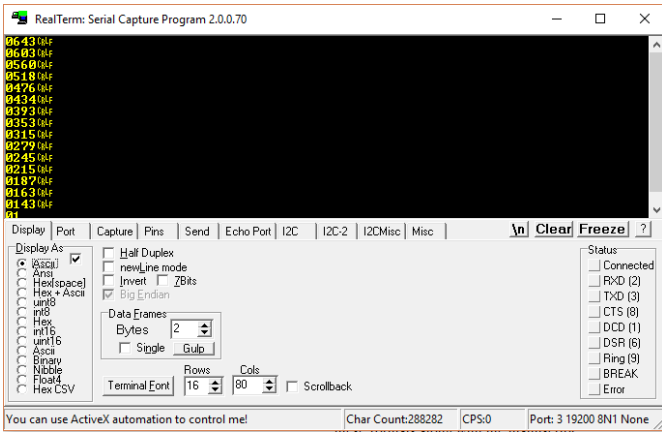


Figure 4: Confirming Serial Communication – Note the values from 0 to 1023 sent out by the Esduino and received by the computer

The next step is to test the MATLAB algorithm. Figure 5 is the output from the plotting algorithm when a 5 Hz input has been applied to analog pin 4 on the Esduino.

We can see that the shape of the sine wave has been accurately reproduced, with the correct frequency.

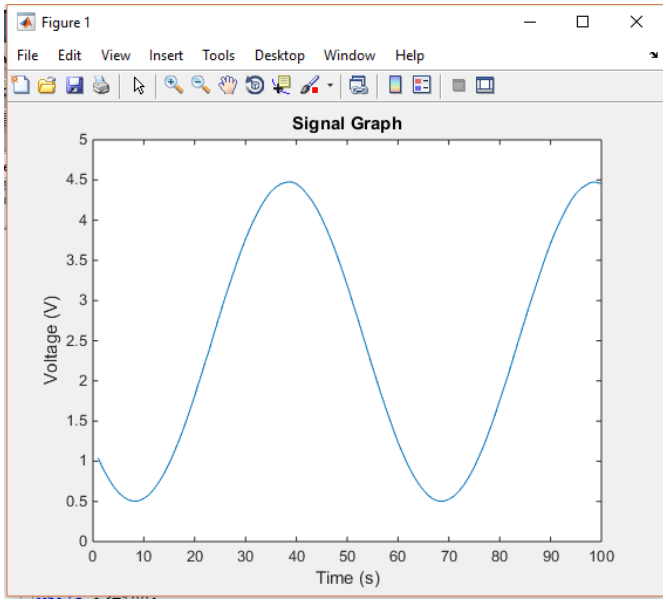


Figure 5: MATLAB Output for a 5 Hz Sine Wave

The next step is to test the entire system as a whole. A picture of the whole system working can be seen in figure 5.

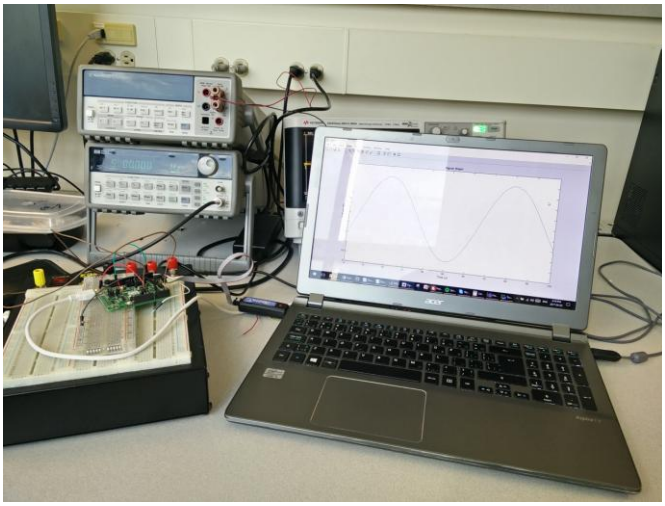


Figure 6: Entire System

We can see the signal generator, the computer and the Esduino working in tandem to reproduce the signal sent in by the function generator.

The next step is to try different kinds of inputs to ensure proper function. Figure 6 shows the output when a sawtooth wave is applied.

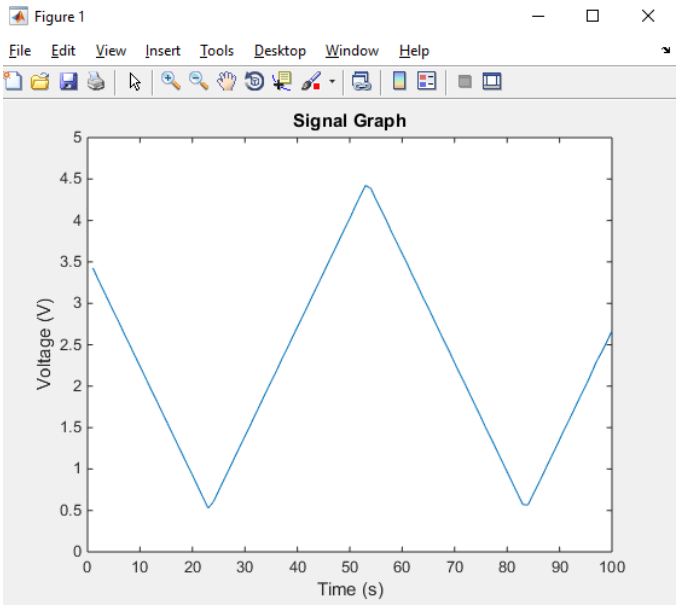


Figure 6: Output for a Sawtooth Wave

We must also test Nyquist's Theorem. We have confirmed that at values below 32 Hz, the signal appears clear and smooth. For values in between 32 Hz and 160 Hz, we can get an accurate representation of the frequency, but not the amplitude. Figure 7 shows this.

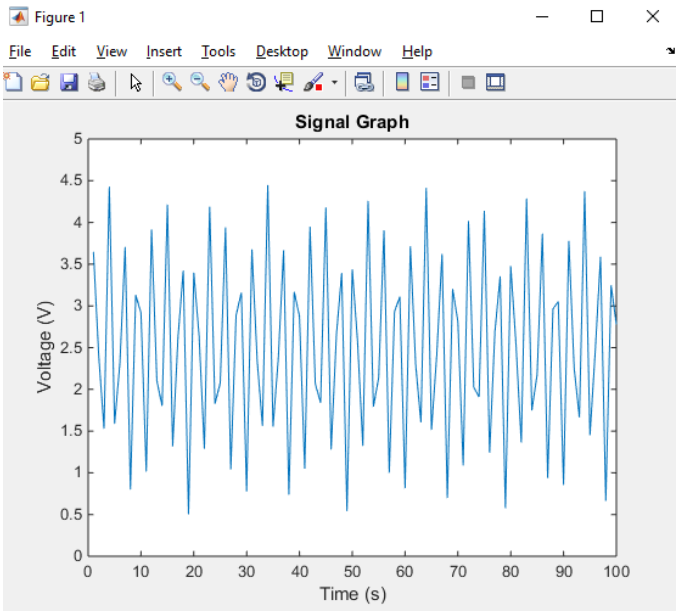


Figure 7: Output at a 120 Hz sine wave

According to Nyquist's Theorem, values in between 160 and 320 Hz will be a mess. This is shown in figure 8.

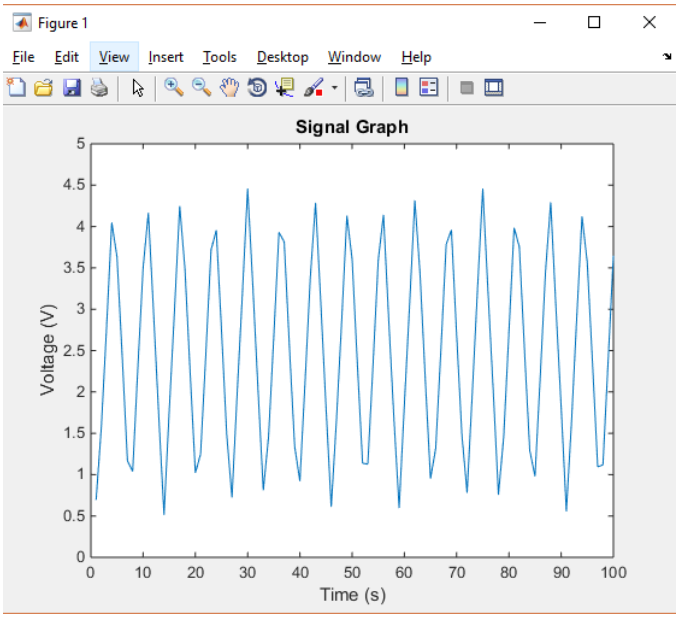


Figure 8: Output for a 270 Hz Sine Wave

We also know that for a signal at the frequency of the sampling frequency, we should get a single line. This is confirmed with figure 9.

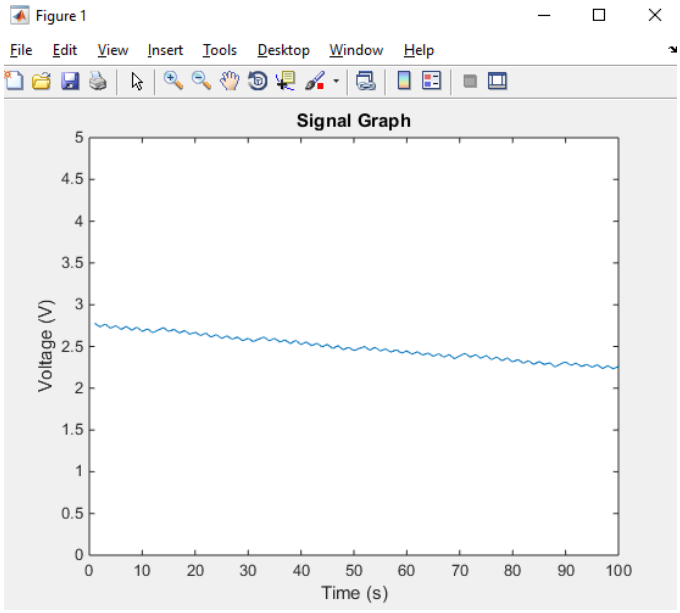


Figure 9: Output When Input Frequency is the same as the sampling frequency, ie 320 Hz

IV. Discussion

The maximum quantized error with a 10 bit ADC is is 5/2^10, which is equal to 0.0048828125 volts.

Based on the bus speed, the maximum baud rate that can be implemented is 19200 bits/sec. Anything else, and the error is too large to produce an accurate signal.

The primary element that limits the speed is the baud rate, as can be seen from the transmission time equation in the system overview.

Based on the Nyquist rate, the maximum frequency that we can faithfully produce is 32 Hz, or 1/10 of the sampling frequency. When the signal is between 32 and 160 Hz, we can get the frequency, but not the shape or magnitude. After this point, the signal is just random, and no meaningful info can be extracted from it.

A sawtooth wave is reasonably accurately reproduced, but the transitions are flatter than expected. Please refer to figure 6.

V. Conclusion

In this paper, we outlined a general purpose signal acquisition system, which takes a value in from a transducer, conditions it, passes it into an analog to digital converter and displays it in a graph.

Such a system has many applications, such as being able to visualize the forces on an accelerometer, or the forces applied by a surgeon during a surgery, ensuring better outcomes.

The results shown in section IV show that this system can accurately reproduce signals up to 32 Hz in nature. This can be improved by using a higher sampling frequency, which is a future consideration.

VI. Appendix

C Code for Arduino:

```
//;*************************************
*****************************
//;*                          McMaster
University                              *
//;*                        2DP4
Microcontrollers                              *
//;*                            Lab Section
06                             *
//;*                     Aditya Thakkar
thakkaap 001429465                    *
//;*************************************
*****************************
//;*************************************
*****************************
//;*                          Final Project
*
//;*  This code implements an A/C
conversion channel at a sampling     *
//;*  rate of 320 Hz, with a clock speed
of 4 MHz, and a 10 bit ADC    *
//:*          resolution. The input
channel is channel 4.                 *
//;*************************************
*****************************


#include <hidef.h>      /* common defines
and macros */
#include "derivative.h"      /*
derivative-specific definitions */
#include "SCI.h"

void setClk(void);
void delayneg4s(int delayBy);
void OutCRLF(void);

int onOff = 0;       // Counter to check
if button is on
int adcInput = 0;    // ADC input value
int temp;


void main(void) {

    ATDCTL1 = 0x2F;     // set for 10-bit
resolution
    ATDCTL3 = 0x88;     // right
justified, one sample per sequence
    ATDCTL4 = 0x02;     // prescaler = 2;
ATD clock = 4MHz / (2 * (2 + 1)) ==
0.667MHz
    ATDCTL5 = 0x24;     // continuous
conversion on channel 4

  EnableInterrupts;

  //Set Ports
  DDRJ = 0xFF;      //set all port J as
output


  TSCR1 = 0x90;    //Timer System Control
Register 1
                   // TSCR1[7] = TEN:
Timer Enable (0-disable, 1-enable)
                   // TSCR1[6] = TSWAI:
Timer runs during WAI (0-enable, 1-
disable)
                   // TSCR1[5] = TSFRZ:
Timer runs during WAI (0-enable, 1-
disable)
                   // TSCR1[4] = TFFCA:
Timer Fast Flag Clear All (0-normal 1-
read/write clears interrupt flags)
                   // TSCR1[3] = PRT:
Precision Timer (0-legacy, 1-precision)
                   // TSCR1[2:0] not used

  TSCR2 = 0x00;    //Timer System Control
Register 2
                    // TSCR2[7] = TOI:
Timer Overflow Interrupt Enable (0-
inhibited, 1-hardware irq when TOF=1)
                   // TSCR2[6:3] not used
                     // TSCR2[2:0] =
Timer Prescaler Select: See Table22-12 of
MC9S12G Family Reference Manual r1.25
(set for bus/1)


  TIOS = 0xFE;     //Timer Input Capture
or Output capture
                     //set TIC[0] and
input (similar to DDR)
```

```
  PERT = 0x01;      //Enable Pull-Up
resistor on TIC[0]

  TCTL3 = 0x00;     //TCTL3 & TCTL4
configure which edge(s) to capture
  TCTL4 = 0x02;     //Configured for
falling edge on TIC[0]



  TIE = 0x01;       //Timer Interrupt
Enable


  setClk();         // Set Clock
  SCI_Init(19200);
  DDRJ |= 0x01;


  for(;;) {

    if (onOff%2 == 0) {  // Check if we
should be transmitting based on the
number of times the button has been
pressed

      adcInput = ATDDR0;       // Take in
analog value

      if (adcInput < 1000) {    // If
adcInput is < 1000 (ie adcInput has less
than 4 digits) pad the value with an
extra 0 at the start
                            // This is to
ensure that the transmission time is even
        SCI_OutUDec(0);    // Send out 0
pad
        SCI_OutUDec(adcInput); // Send
out value
        OutCRLF();         // New line
        delayneg4s(10);    // Delay to get
320 Hz sampling time
        PTJ ^= 1;        // Alternate the
state of the on board LED
      }

      else {

      SCI_OutUDec(adcInput); // Send out
the value read into the ADC
      OutCRLF();         // New line
      delayneg4s(10);  // Delay to get
320 Hz sampling time
      PTJ ^= 1;         // Change state of
the on board LED

      }

   }

   else {       // If the button is off
        delayneg4s(10);    // Delay

    }

  } // Loop forever


}


// Function to set the clock speed to 4
MHz
void setClk(void) {

     // Project spec tells to use a 4 MHz
Bus Speed

     TIE = 0x01;

     CPMUCLKS = 0x80; // Set PLLSEL = 1
     CPMUOSC = 0x00;  // Use the 1 MHz
Internal Reference Clock


     CPMUSYNR = 0x0F; // Set VCOFRQ = 0
and SYNDIV = 15 for the project spec

     // fREF = 1 MHz and fVCO =
2*fREF*(15+1) = 32 MHz for the project
spec

     CPMUFLG = 0x00;
     CPMUPOSTDIV = 0x03;

     // PLLCLK = fVCO/4 = 8 MHz for this
project spec
     // Bus Clock = 4 MHz for this
project spec

     while(!(CPMUFLG & 0x08));

}

// Function that delays by a multiple of
1e-4 s
void delayneg4s(int delayBy) {

    int a;
    TSCR1 = 0x90; // Enable time and
clear fast timer flag
    TSCR2 = 0x00; // Disable timer
interrupt, set prescaler to 1

    TIOS |= 0x02;
    TC1 = TCNT + 400; // Delay by 1e-4

    for (a = 0; a < delayBy; a++) {

      while(!(TFLG1_C1F));
```

```
      TC1 += 400;

    }

}


// Taken from the Lab 4
// Output a CR,LF to SCI to go to a new
line
void OutCRLF(void){

  SCI_OutChar(CR);
  SCI_OutChar(LF);

}

// Taken from Lab 4
interrupt  VectorNumber_Vtimch0 void
ISR_Vtimch0(void)
{

  unsigned int temp;
  onOff = onOff + 1;    // Increment the
On/Off counter
  temp = TC0;       //Refer back to
TFFCA, we enabled FastFlagClear, thus by
reading the Timer Capture input we
automatically clear the flag, allowing
another TIC interrupt

}
```

MATLAB Code:

```
% Serial Data Plot
% Aditya Thakkar thakkaap 001429465

clear;
delete(instrfindall);
serialObj = serial('COM3'); % Define
Serial Object
serialObj.BaudRate = 19200; %Set baud
rate
serialObj.Terminator = 'CR';
fopen(serialObj); % Open the serial
object

numBits = 10; % ADC Resolution
peakVoltage = 5; % Maximum voltage
mag = 1/(2^numBits/peakVoltage); %
Division factor
dataMatrix = zeros(1,100); % Define data
matrix

while true % Loop forever
    i = 1;
while i<=100;
    dataMatrix(1,i) =
str2double(fgetl(serialObj)); % Allocate
value into matrix
    i = i+1;
 end

dataMatrix = mag*dataMatrix; % Voltage
value
plot(dataMatrix); % Plot the data
title('Signal Graph') % Set title
ylabel('Voltage (V)')  %Y label
xlabel('Time (s)') % X label
axis([0, 100, 0, 5]); % Set axis

pause(0.1); % Pause, this is also the
resolution time to up date for the graph
end
```