2DP4: Microprocessor Systems
Final Project

Instructor: Dr. Doyle

Serial Communication between Esduino Xtreme & MATLAB

Shubham Gupta – L07 – guptasr – 1416773

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by **[Shubham Gupta, guptasr, 1416773]**

Table of Contents

Abstract

In this report will we be implementing a data acquisition system which will allow us to retrieve useful information from an analog AC signal. The main goal of this project is to convert the analog signal through a specific device and into a digital signal. The signal will then be transmitted using a source of communication that will allow the PC to display the acquired data. In order to test the data acquisition system a positive sinusoidal graph must be outputted on the PC display. Overall, our system successfully manages to output a level shifted sinusoidal graph which is amplified and is in the range of 0 to 5V. The system we used implements the previously mentioned data acquisition system by using a specific microcontroller in order to achieve the task.

Introduction & Background

In our modern world, analog signals can we found everywhere; it is in the music we listen to, the light intensity in a room, and are even found in the pressure differences in our weather. These signals carry a lot of information with them; however they are incomprehensible to humans and must be converted into digital signals in order to obtain any valuable information. Today, we can easily quantify any analog signal and convert them into a digital signal by using microcontrollers. Microcontrollers are basically small computers that have a single integrated circuit which contains a processor core, programmable input/output peripherals (IO) and a memory. They are used to convert these signals into a digital domain in a process known as data acquisition. This process takes the analog signal through a transducer (Temperature, pressure or sound) and serially communicating to output the data on a computer in a form of graphs/tables.

In this project, the analog signal was a sinusoidal wave with a center at x-axis and four volts peak to peak. The signal is precondition through an electric circuit which level shifts the signal to a positive sinusoidal wave and passes it to the Esduino Xtreme (ESDX); an analog to digital converter module. This is where the conversions are done per sample and an output is serially communicated through a software and displayed on the PC. Furthermore, a very common example of analog to digital converter (ADC) is a scanner. It works very similarly to our project as it converts an analog signal (light) to a digital signal, which is then processed through data acquisition in order to replicate and display the original image on a digital screen. The main objective of this report is to provide a breakdown for the design that was implemented and the testing that were done during each step. In order to provide a better insight I will thoroughly discuss the minute information, design limitations as well as difficulty faced during the project.

Design Methodology

Esduino Xtreme

As mentioned earlier Esduino Xtreme (ESDX) is an analog to digital converter module, also known as a microcontroller, refer to Fig. 1. It is very similar to an Arduino due to its layout; however it has an 8Mhz crystal after reset and comes from the HSC 12 Family. It supports multiple languages for assembling coding such as C, Assembler and basic.

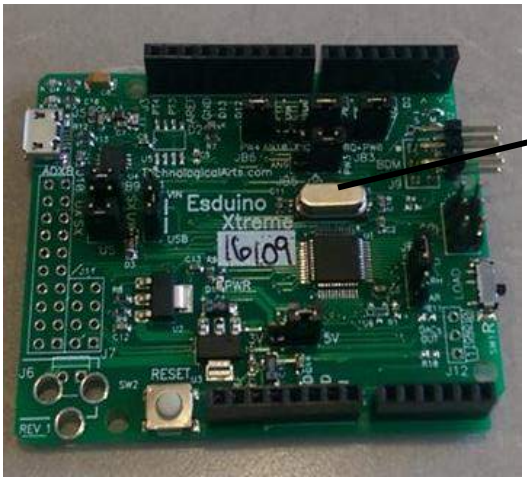

Fig. 1: Esduino Xtreme

Quantification of Input Signal

The Esduino Xtreme (ESDX) has a set possible voltage range of 0 to 5 volts or 0 to 3 volts. For this project, the range of 0 to 5 volts was chosen thus resulting in output voltages in the range of 0 to 5 volts. However, the signal we were told to input into our data acquisition device had a 4 Volts peak to peak (-2V to 2V) and AC voltage signal centered at the x-axis. Since, the voltage is centred at zero we will have a positive and a negative voltage but our ESDX does not accept negative voltage. Thus, we would need to level shift this input with a preconditioning circuit in order to make sure that only a positive AC signal is inputted.

Transducer

For the preconditioning design of this project, a transducer was **not** used because the input analog signal is a voltage signal by itself so it will not require a transducer to convert the signal into an electric voltage. Thus, the only problem we will encounter is a negative voltage. As mentioned before the microcontroller can only accept positive voltages ranging from 0-5 volts thus in order to ensure we get a positive voltage, level shifting would be used. Level shifting is the process of shifting voltages at a given level to another level. Thus, in order to achieve the same 4 volts peak to peak a voltage divider was used in the preconditioning circuit. We will discuss more about level shifting later in the report under preconditioning circuit.

Assigned Specification for the Project

According to my student number 1 4 1 6 7 7 3, I was assigned the following specifications;

Least Significant Digit	Operational Parameter	Assigned Specifications
3	Analog to Digital Converter (ADC) Channel	AN5
7	Analog to Digital Converter (ADC) Resolution	10 bits
7	Bus Speed	16 Mhz
6	Sampling Frequency	480 Hz

Fig. 2: Table for Assigned Parameters

Preconditioning Circuit

The design that was used to precondition and level shift the output signal is shown in the OrCAD Pspice capture in Fig. 3. The topology is known as a non-inverting summing amplifier and has four resistors; 10K ohm, 25k ohm, 14.30K ohm and 10K ohm, respectively. It also has an AC voltage source with an input frequency of 100 Hz with amplitude of 2V and a 5V DC source. The AC voltage source is connected to the 10K ohm resistor and the DC voltage source is connected with the 25K ohm resistor. These two resistors (R1&R2) play a crucial role by working as a voltage divider providing and input of 1.429V on the non-inverting pin of the amplifier. The remaining two resistors (R3&R4) are connected to the non-inverting pin of the amplifier and the output respectively. They perform a crucial role in controlling the gain of the amplifier. By referring to the equations below we can determine that the output voltage would be 2.428V and thus the gain would be 1.699V.

$$Vo = Vop\_in\left(\frac{R3 + R4}{R3}\right)$$

$$Vop_{in} = \left(\frac{Vac(R2) + Vdc(R1)}{R1 + R2}\right)$$

Furthermore, the simulation from OrCAD verifies that the circuit is indeed functioning as promised and is level shifting all the outputs into the positive domain as well as amplifying it to be 5 V peak to peak.

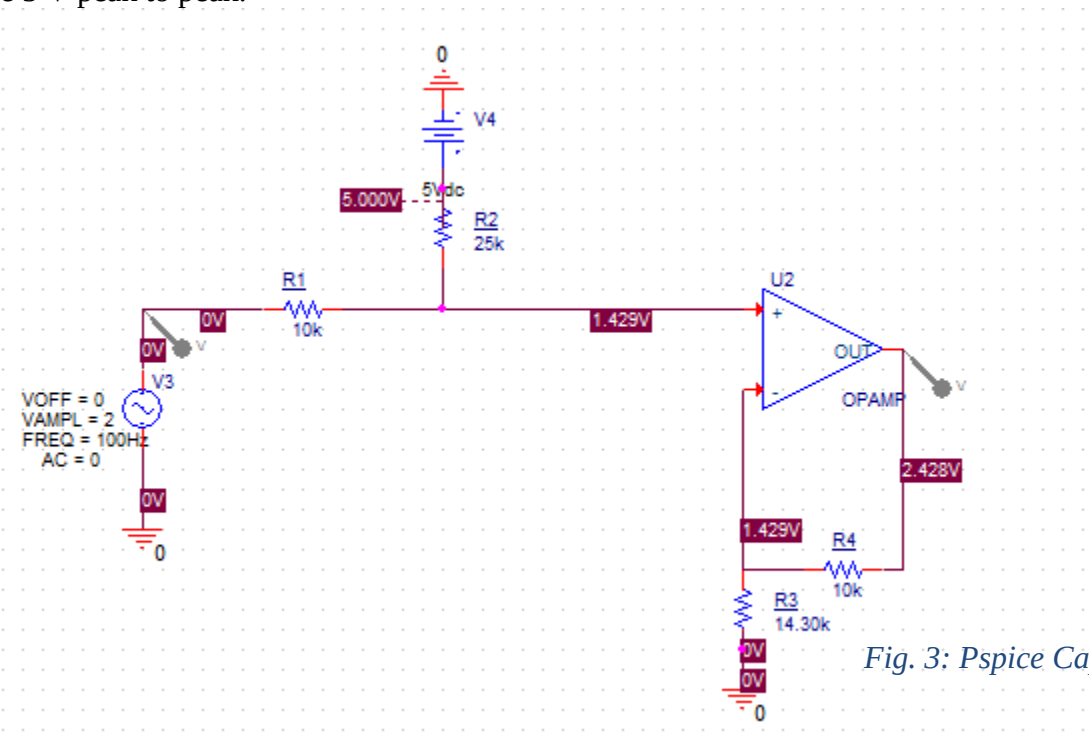

Fig. 3: Pspice Capture

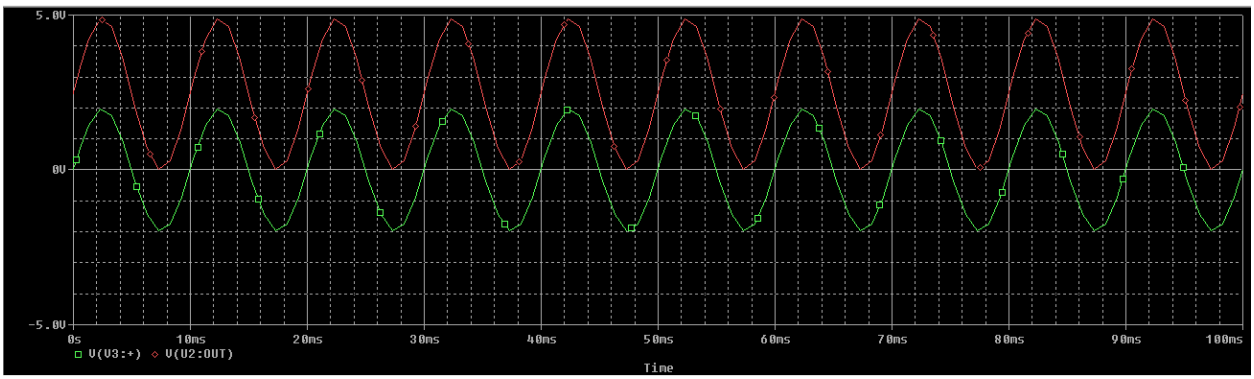

Fig. 4: Pspice Simulation Output

From Fig. 4 we can verify that the theoretical preconditioning circuit was successful in acquiring the analog signal and also level shift it to an appropriate value which is centred at 2.5V instead of 0V.

Physical Circuit

After the successful theoretical simulation from OrCAD the actual circuit was built. Referring to Fig. 5-6, the circuit used a CMOS Dual Operational Amplifier (LMC 662, refer to the schematic for the pin layout in Fig.7), two 10K ohm resistors, two 25K ohm resistors and three 100K ohm resistors. The Analog Discovery Module was also used as a oscilloscope and the function generator for the input, refer to Fig.8. The first 10K ohm resistor was set up as a voltage divider with the 25K ohm resistor which was fed into the positive pin of the op amp. The 10K ohm resistor got the input from AC source and the 25K ohm got the 5V DC input from the Esduino Xtreme. To account for the 14.30K ohm resistor from the OrCAD simulation above; three 100K ohm and a 25K ohm resistors were set in parallel which were fed into the inverting pin of the op amp. The last resistor of 10K ohm was then set to the output pin of the op amp and everything was grounded across the board using the Analog Discovery Module.

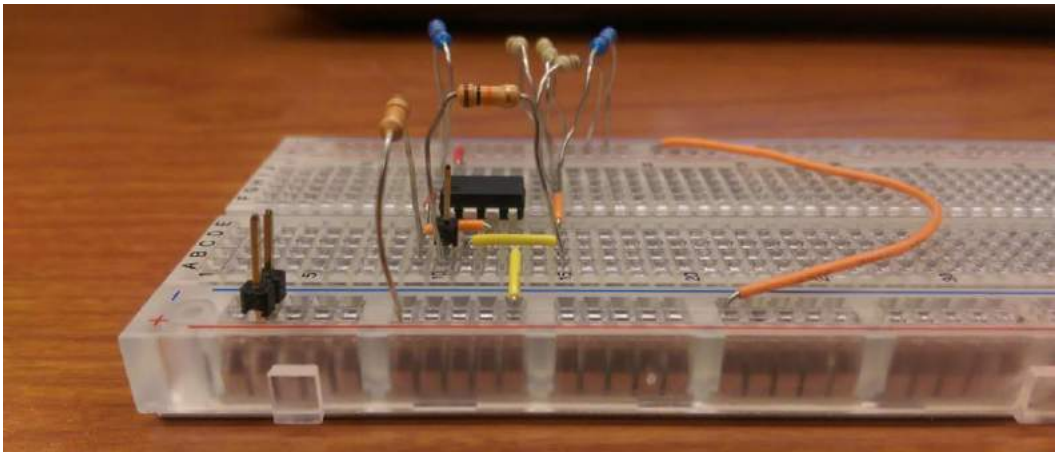

Fig. 5: Side View of the Circuit

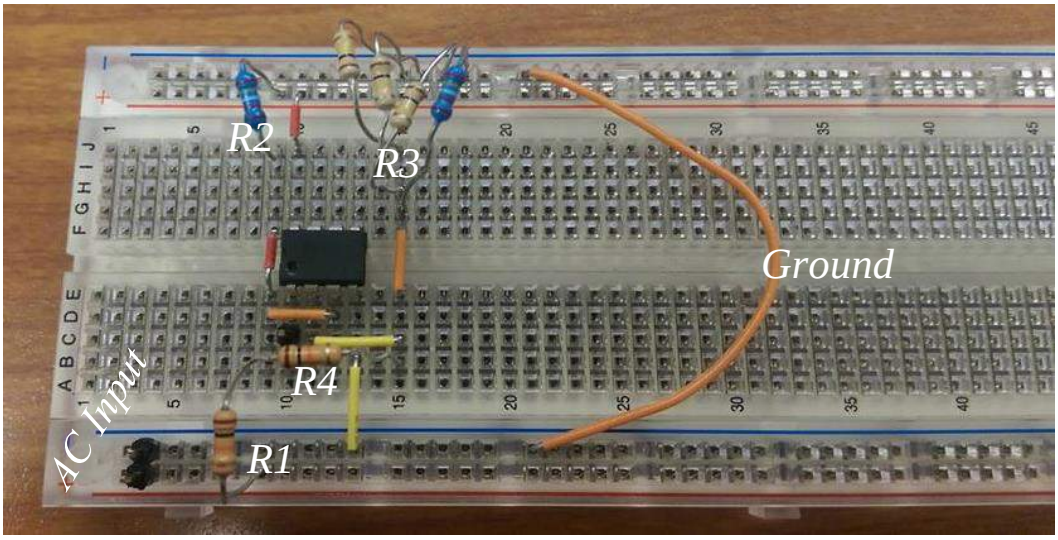

Fig. 6: Top View of the Circuit

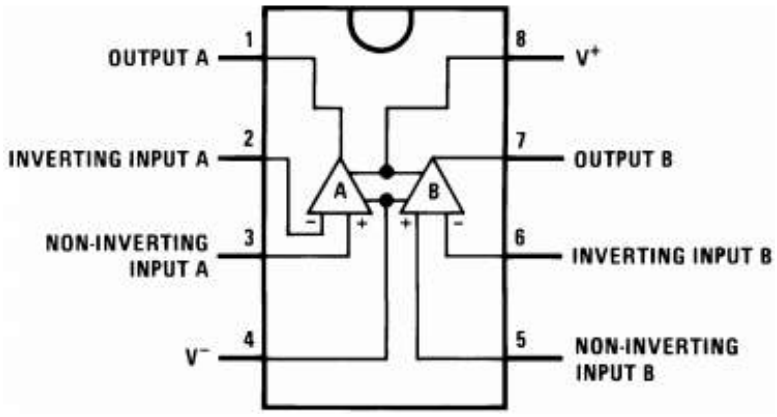

Fig. 7: LMC662 Pin Layout

Fig. 8: Analog Discovery Module Used as a Function Generator & Oscilloscope

Verification of the Physical Circuit

The physical circuit was then verified by the oscilloscope refer to Fig. 9-10. As seen in the picture the output corresponds and matches with the theoretical results determined by OrCAD as it level shifts and displays a 5V peak to peak centred on 2.5V. Thus, these results verify that the preconditioning circuit is working as promised and is ready for further process.

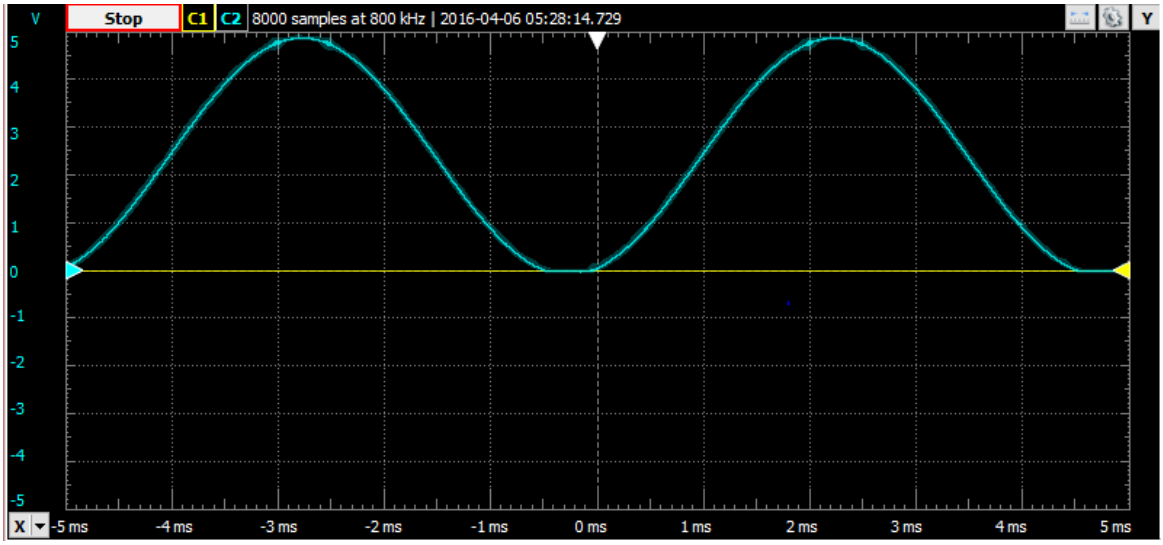

Fig. 9: Output from the Physical Circuit

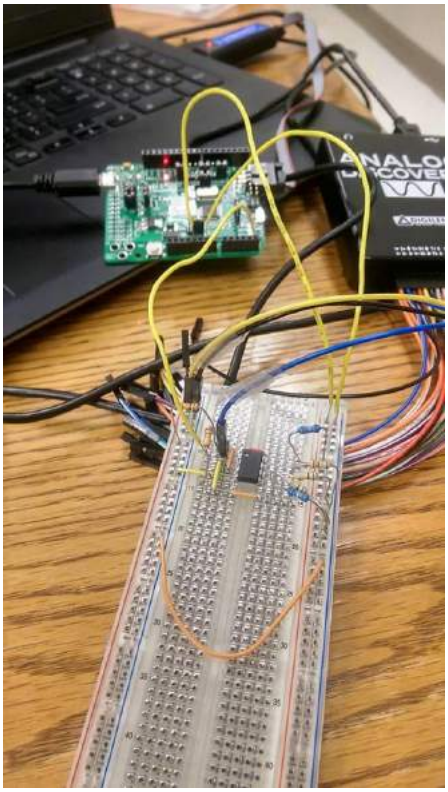

Fig. 10: Connection to the Scope Using Analog Discovery

Design Limitations

There were a few limitations while implementing this design. Firstly, my component kit only had sets of 100K resistors, a couple 25K ohm resistor and a few 10K resistors. It would have been really efficient if I had 1K ohm resistors or even 5K ohm resistors since this would have allowed me to use fewer resistors to come up with the equivalent for the 14.30K ohms. Using four sets of resistors; three 100K and one 25K all parallel with each other, in order to come up with the 14.30K ohm equivalence increases the tolerance in the circuit and can lead to minor errors in the output. This is because each resistor comes with its own tolerance, i.e. the four sets of resistors used to come up with the equivalence of 14.30K have a total tolerance of 17%, thus it can be concluded that my circuit might deviate by 2K ohms from the 14.30K ohms. Therefore, having even the minimalistic tolerances in these components can lead to errors in the outputs.

Analog to Digital Conversion (ADC)

Analog to Digital conversion is a process that converts a continuous physical quantity usually a voltage to a digital number which represents the quantities amplitude. The conversion involves quantization of the input, and thus introduces a small error known as quantization error. Using the full scale voltage of 5 Volts provided by the Esduino and the 10 bit resolution required by the project specification; we can determine all the possible sample ADC values refer to Fig. 11.

Analog Voltage Vh	ADC Values (Decimal)	ADC Values (Hex)
5.00 (100%)	1024	400
3.75 (75%)	768	300
2.50 (50%)	512	200
1.25 (25%)	256	100
0.00 (0%)	0	000

Fig. 11: Sample ADC Values

ADC Channel Configurations

The ADC channels must be configured appropriately in order for the Esduino to accurately receive data from the preconditioning circuit and convert it into digital information. Fig. 12 goes over the configuration of all the ADC registers and sets them according to the channel 5 as required by the project specifications.

ADC Register	Value	What does it do?
ATDCTL0	0x05	Wrap around channel select bits to AN5
ATDCTL1	0x2F	Set ADC to a 10 bit resolution
ATDCTL3	0x09	Right Justified with 2 samples per sequence (2 packets due to 10 bit resolution)
ATDCTL4	0x02	Prescaler set to 2
ATDCTL5	0x25	Continuous single channel conversion on AN5

Fig. 12: Configuration of ADC Registers

ADC Verification

Analog to Digital Conversion (ADC) is a very crucial stage in data acquisitions. This process is the most important aspect of this project as it packets data in terms of bits and uses a universal serial bus (USB) in order to transmit/receive data from the micro-controller and to the serial communication software. The rate at which these packets are transmitted is known as the **Baud Rate**, which is configured through the Esduino Xtreme and using its SCI.h and C codes. Furthermore, in order to verify if the ADC configurations were set appropriately for our project we used RealTerm, a Baud Rate of 9600 and a circuit consisting of a 10K ohm resistor in parallel with a potentiometer; refer to Fig. 13. As mentioned before, we were assigned a resolution of 10 bits and an analog channel of 5. Thus, the ranges of values that directly correlates with the on/off state would be 2^n, where n is the bit resolution. Since, the resolution we were assigned was 10 bits the values would range from a minimum value of 0 to a maximum value of 1024. However, since we used a 10K ohm resistor in parallel with a potentiometer this would guarantee that the voltages are divided equally. Thus when the potentiometer is set at 10K ohms, RealTerm outputs analog voltage of 511 as seen in Fig. 14 and similarly, when the potentiometer is set to 0K ohms, RealTerm outputs an analog voltage of 1023 as seen in Fig. 15. Therefore, this testing verifies that our ADC is set appropriately and is ready to be implemented in a different serial communication software which will be MATLAB.

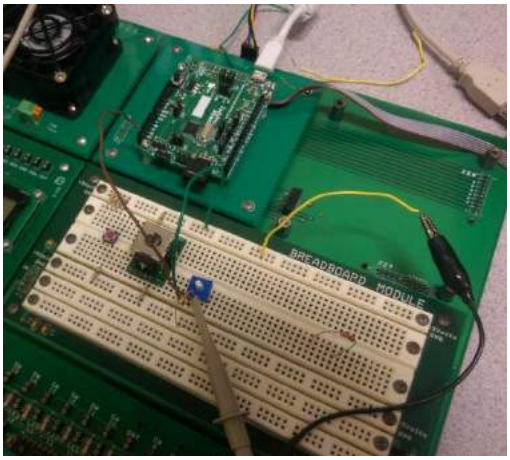

Fig. 13: Potentiometer Circuit

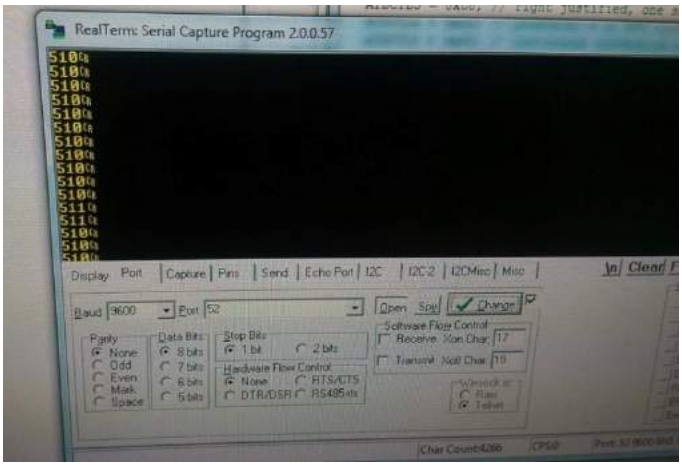

Fig. 14: RealTerm Output when Potentiometer is set to 10K

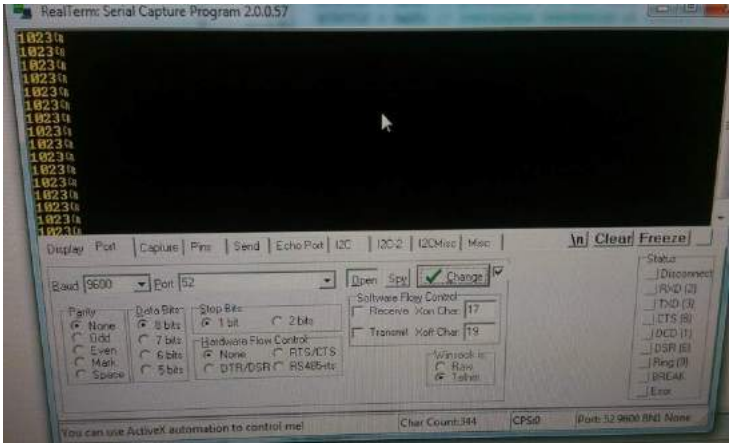

Fig. 15: RealTerm Output when Potentiometer is set to 0K

Data Processing

Data processing is essentially the process where acquired data is transformed into meaningful information. In our case it is the stage where we analyze our results in order to process meaningful voltage information. So far we have taken an input signal from the transducer which was a pure AC source and passed it through a pre conditioning circuit which level shifted and amplified/filtered our output in terms of an electrical voltage. We then fed this electrical voltage into the analog to digital converter which converted the voltage as a digital value (binary, hexadecimal *et al*) using the ADC registers which was configured as assigned. Furthermore, we were able to serially communicate this transformed data into meaningful information which was displayed in terms of a graph on MATLAB which we will be discussed later; Fig. 16 summarizes everything we have done so far in terms of flow chart.

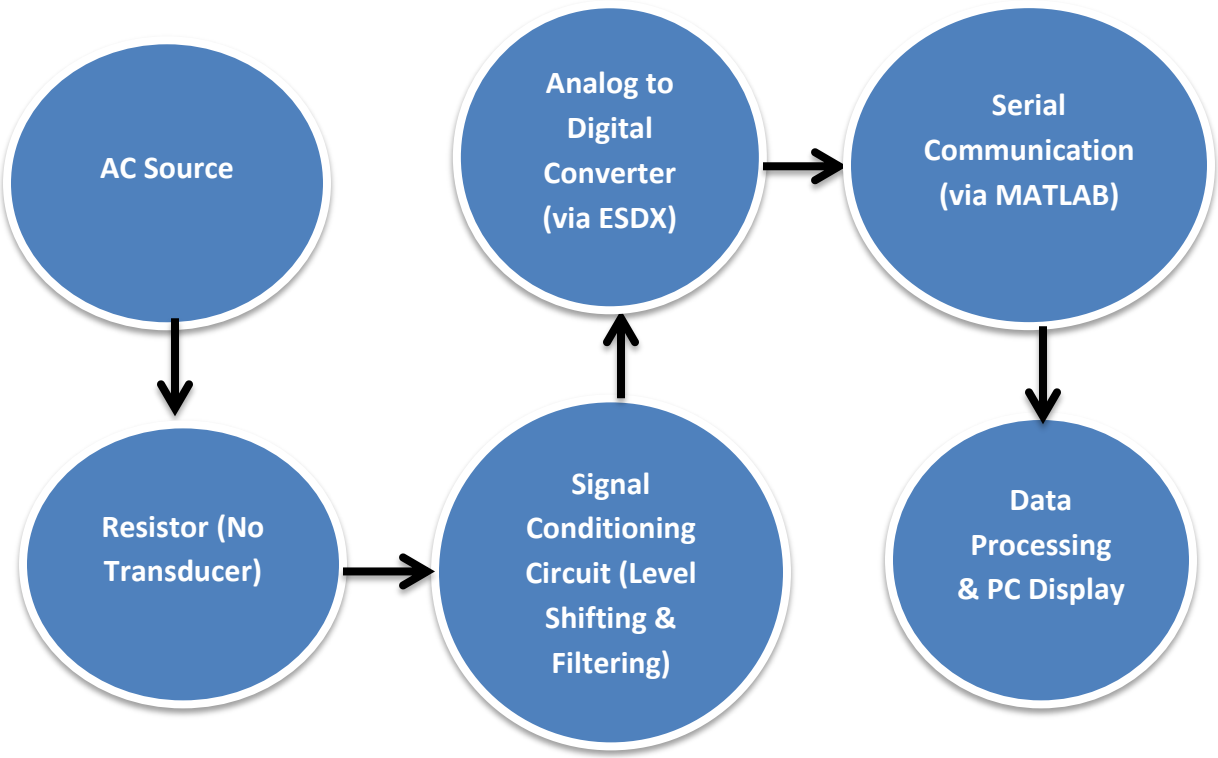

Fig. 16: Summary of Data Acquisition

Setting the clock

As mentioned before we used a microcontroller called the Esduino Xtreme in order to process the data. We also used CodeWarrior and C language for programming this micro controller. However, in order for any data that is acquired to be meaningful we must sample it at an appropriate rate which can only be achieved if the clock speed is configured correctly. The bus clock we were assigned was 16 MHz and recalling that the Esduino Xtreme's default bus speed is 6.25 MHz we must change it to suite it to our specification. In order to achieve 16MHz we did the following calculations;

$$PLLCLK = 2 \times OSCCLK \times \frac{(SYNR + 1)}{(REFDIV + 1)}$$

$$BUSCLK = \frac{PLLCLK}{2}$$

$$BUSCLK = OSCCLK \times \frac{(SYNR + 1)}{(REFDIV + 1)}$$

$$16\ MHz = 8\ MHz \times \frac{(SYNR + 1)}{(REFDIV + 1)}$$

When the SYNR register is chosen to be the value of 1 and REFDIV to be a value of 0, the following is achieved;

$$16\ MHz = 8\ MHz \times \frac{(1 + 1)}{(0 + 1)}$$

$$16\ MHz = 8\ MHz \times 2$$

$$16\ MHz = 16\ MHz$$

Thus in order to achieve the desired 16MHz we set SYNR to 1 and REFDIV to 0 in our C code, refer to appendix for SetClk( ) function.

Verification of Bus Clock

For verifying if our bus clock was set properly, in addition with our SetClk( ) function we implemented a for loop in our C code which configured Port J which is our on board LED with delay function of 1000ms. The for-loop would allow the onboard LED to flash every 1 second, therefore it will stay ON for 1 second and OFF for 1 second. If we have set our SetClk( ) function properly this phenomenon would replicate in an identical manner, thus we would expect it to be ON for 1 second (active high) and OFF for 1 second (active low). In order to physically verify, we connected the onboard LED pin (DIG13) to the probe of the scope and grounded the Esduino. As seen in Fig. 17 the LED was ON for exactly 1s and OFF for exactly 1s. Therefore, we have verified that our SetClk( ) function was properly set.

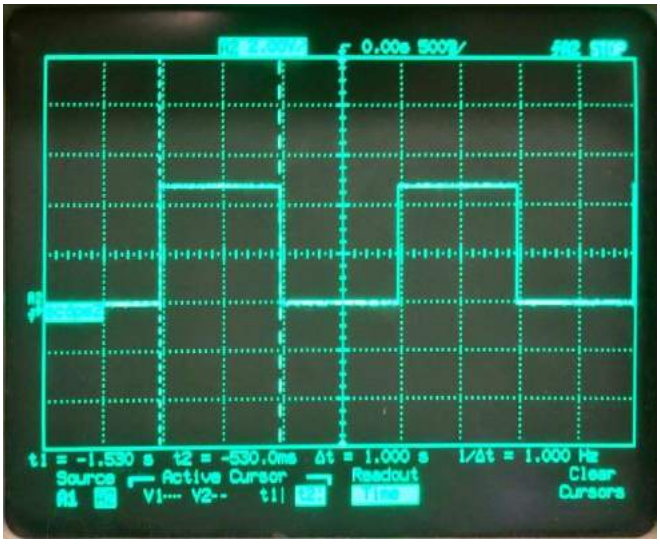

Fig. 17: Testing for Clock

CodeWarrior Algorithm

Below we have provided the data processing algorithm used on the Esduino Xtreme in order to demonstrate the events that occurred while converting an analog signal to digital signal and displaying the results on MATLAB. Refer to Fig. 18

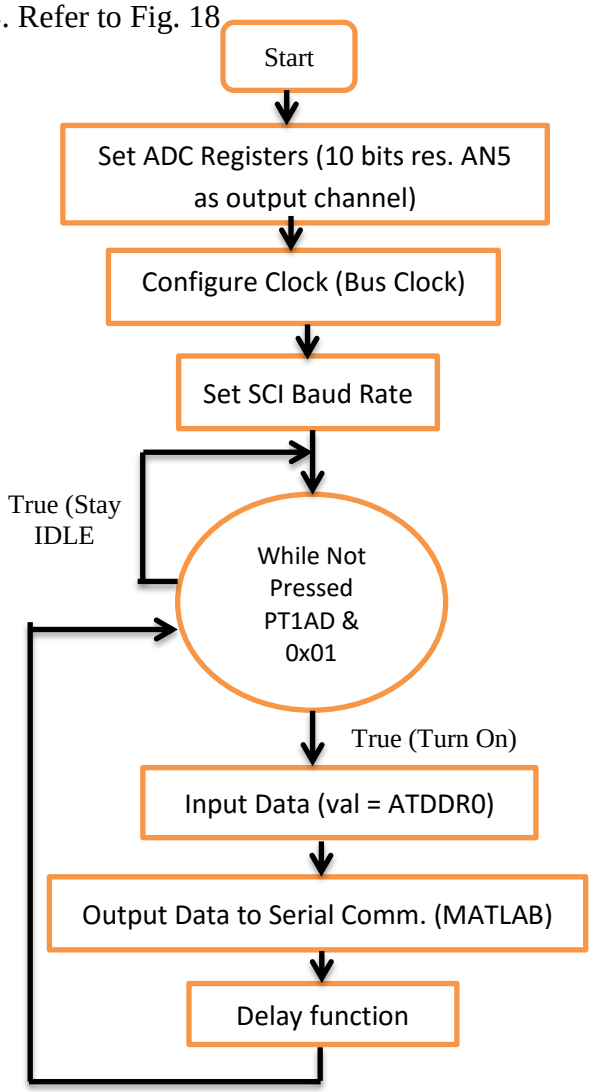

Fig. 18: Flow-Chart for CodeWarrior

Control & Communication

The PC that was used to create and run all of these data acquisition components was a Dell Inspiron 5447 with an Intel Core i7-4510 CPU at 2.60GHz, 64-bit operating system and 8 GB RAM. As, mentioned earlier the serial communication software that was used to receive the digital data from the Esduino was MATLAB r2013a. MATLAB was chosen primarily due to its previous exposure and familiarity, it is very versatile software used by engineers, scientists to explore and analyze different data processing, control systems and communication. Many functions were utilized in order to retrieve and plot the data received from the Esduino. However, the main components that were the driving factor for the serial communication were the Baud Rate and the COM port. The COM port that was used to transfer data was COM4 and a baud rate of 9600 was used. In theory, any baud rate from the table in Fig. 19 can be used these baud rates are included in SCI.C file and can be changed. After successfully altering the SCI.C code from lab 4 we were able to establish a connection between the Esduino and MATLAB due the common communication port and baud rate and allowed the appropriate data processing. The terminator that was used to close the port was the characters "CR" which is outputted by the Esduino after every value in displayed. In Fig. 20 we can see how beautifully the output was displayed.

Baud Rate (bits/second)	Baud Divisor
2400	417
4800	208
9600	104
19200	52
38400	26

Fig. 19: Baud Divisors According to Baud Rate

$$\boldsymbol{Baud\ Divisor = \frac{Bus\ Clock}{(16(Baud\ Rate))}}$$

$$\boldsymbol{Baud\ Divisor = \frac{16 \times 10^6 Hz}{(16(2400))} = 417}$$

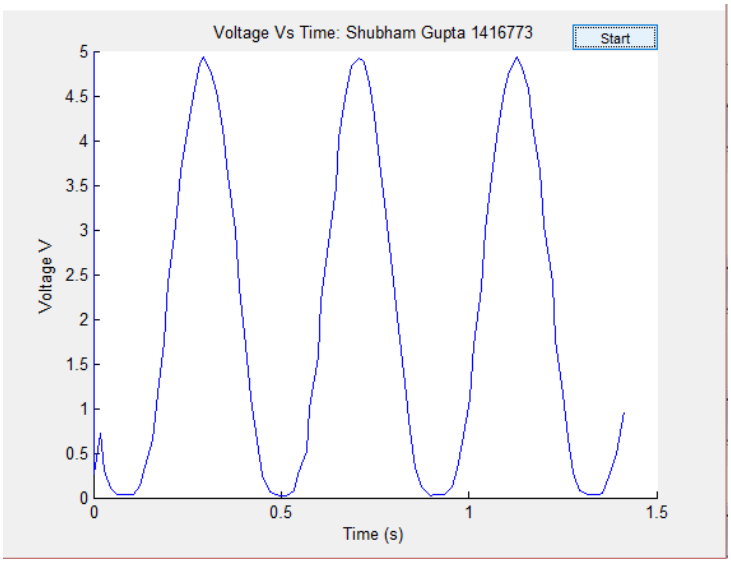

Fig. 20: Output from MATLAB

Implementation of MATLAB

The MATLAB algorithm that was used to graphically display the output of the Esduino Xtreme that is being converted from analog to digital at a required sampling frequency is shown in Fig. 21. It is initialed by opening the appropriate port i.e. for our project COM4, setting a baud rate and initializing the terminator; which acts like as a new line in order to create spaces between the values. When the functions tic( ) and toc( ) are used we are able to keep track of the time that is elapsed which helps in the construction of the graph. The timer is initiated by calling the tic( ) function allowing the program to start the while loop which can iterate a set number of samples chosen by the user. A software button was also implemented on MATLAB using GUI through the command window. When the button is pressed the 10 bit values from the Esduino will be received at the port using fscanf() function, and are converted from characters to double using the str2double( ) function and then converted into the voltage values. These values are then plotted on MATLAB with the desired number of sample that was chosen by the user, when completed the port is closed and the program will end.

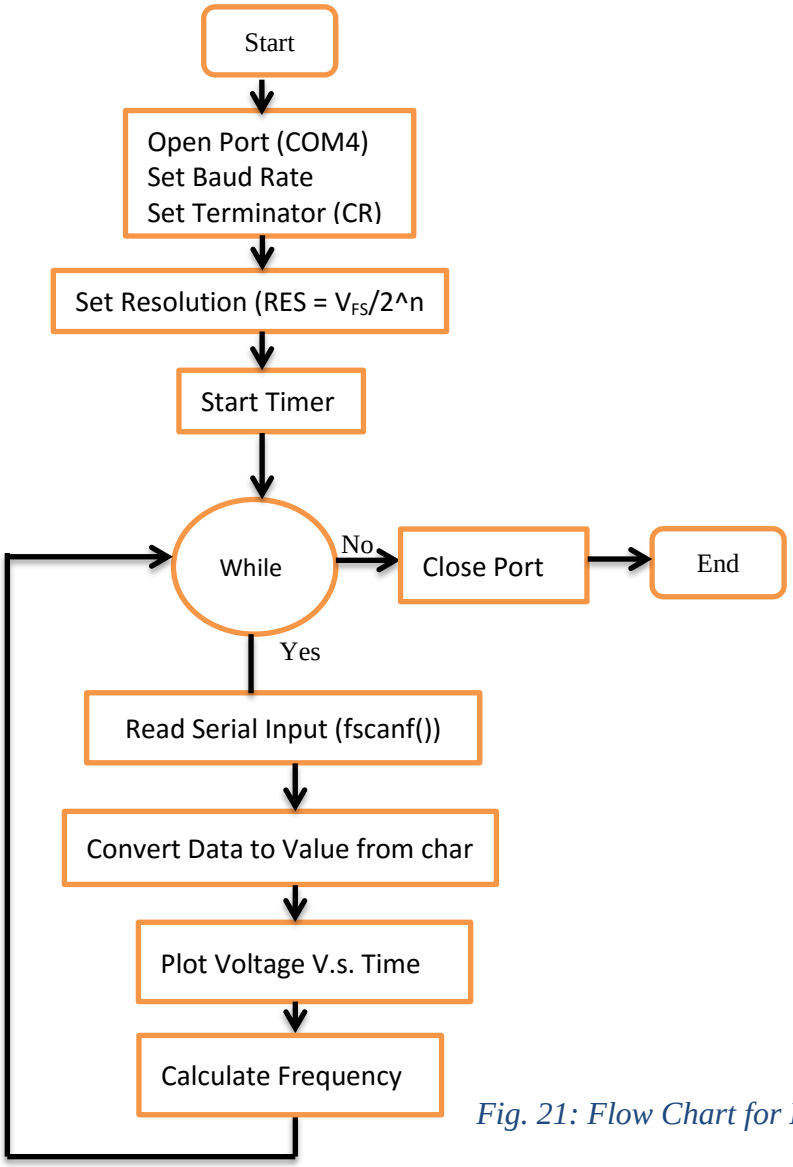

Fig. 21: Flow Chart for MATLAB

Full System Block Diagram

We have provided a flow chart for the full system from start to end in Fig. 22. Also in Fig. 23 we have provided a picture of the whole system in action.

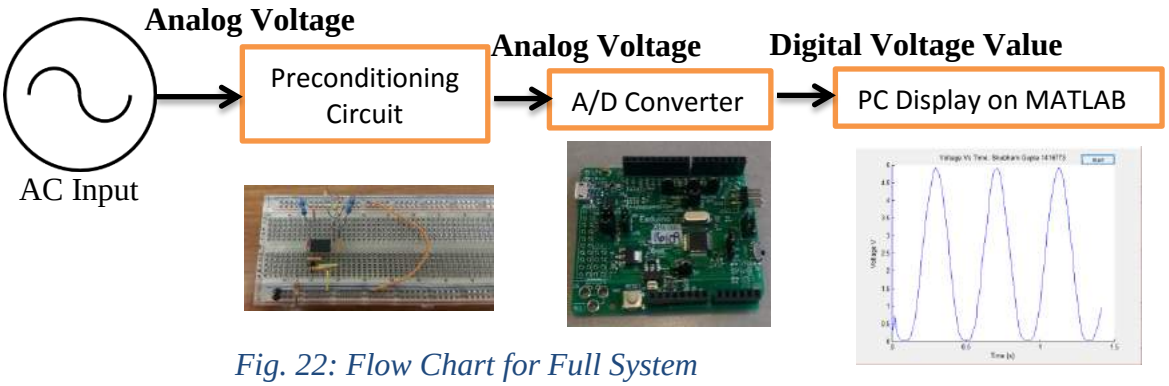

Fig. 22: Flow Chart for Full System

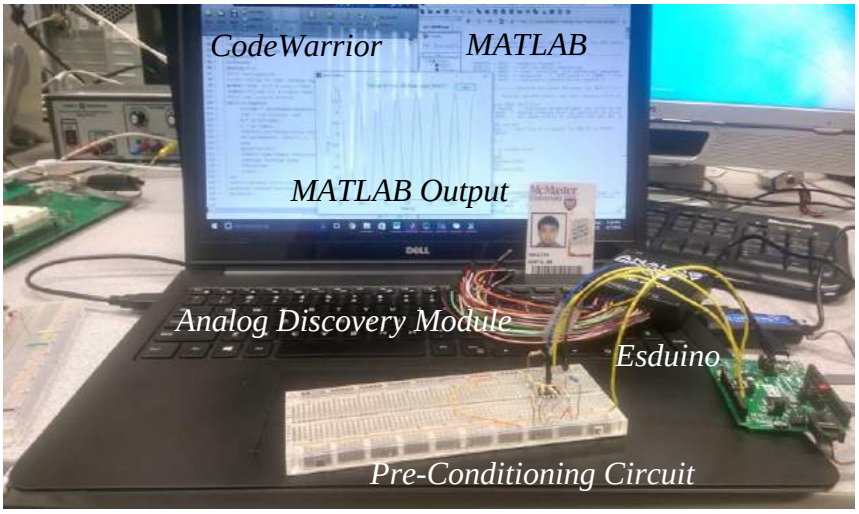

Fig. 23: Complete System Setup

Discussion

1. The maximum quantization error is the difference between the actual analog voltage value and the quantized digital voltage value. In the equation below V_{fs} is the full scale voltage which is 5V and bits is the resolution bits that were assigned to this project.

$$Q_e = \frac{V_{fs}}{2^{bits}} = \frac{5}{2^{10}} = 0.004882813$$

2. Based upon the assigned 16MHz bus speed the maximum standard serial communication rate that can be implemented will be the setting that gives the smallest possible baud divisor would be a baud divisor of 1. Therefore doing some reverse calculations we can determine that the maximum baud rate would be 1000 000. However, since we are only allowed to use the standard the maximum baud rate would be 57600. Refer to Fig. 24 for the calculations of different Baud Rates.

Baud Rate (bits/second)	Baud Divisor	Error
2400	417	0.083%
4800	208	0.1666%
9600	104	0.15625%
19200	52	0.16145%
38400	26	0.1614%
57600	17	2.1241%
1000000	1	0.0000%

Fig. 24: Calculations of Different Baud Rates

$$Max_{BaudRate} = \frac{F_{bus}}{(baud\ divisor(16))} = \frac{16 \times 10^6}{1(16)} = 100{,}000\ bits/sec$$

3. After reviewing the entire system the element that poses a limitation on speed is the serial communication software that is used, MATLAB. Firstly, a qualitative test was done by setting up a serial communication input and plotting output voltage values on MATLAB in real time with different frequencies from the function generator. It was very clear that the MATLAB output was frequently few seconds behind compared to the input. A second test was also done by removing the delay function from code warrior and examining the maximum output frequency. Even though the delay function was removed from code warrior MATLAB was able to sample at a frequency of about 70Hz while the Esduino could sample at 5000Hz.

4. Based on the Nyquist Rate the maximum frequency of the analog signal that we can effectively reproduce is half of what we were assigned. Since we were assigned a sampling frequency of 480 Hz the maximum frequency would be 240 Hz. This limits the range of possible values for analog signal to be anywhere from 0 to 240 Hz. When the input frequency is higher than the specified range it will inaccurately sample and produce an inaccurate output, a phenomena known as aliasing. In simpler terms aliasing means that the sampling rate is not fast enough to correctly sample the input it is receiving this can be seen In Fig. 25 with an input frequency 264 Hz which is 110% of the maximum frequency.

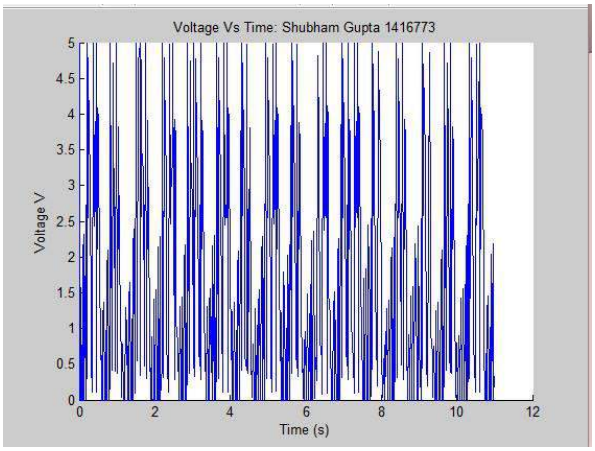

Fig. 25: Aliasing at 264 Hz

5. Input signals with sharp transitions were not accurately reproduced on our serial communication software, MATLAB. At lower frequencies approximately at 10% of the max frequency (~24 Hz), both the square wave and the triangle wave worked fine. However, at frequencies above 24 Hz the sampling was not accurate at all and a lot of aliasing was seen. Due to the aliasing it was really hard to verify if the output was valid. We believe that the sharp transitions do not reproduce accurate results because aliasing itself is in the form of pointy peaks thus a sharp input would only cause confusion at higher frequencies which are still within the range of 0 to 240 Hz and are supposed to output an accurate result.

Conclusion

The main goal of our project was to design and develop a functioning data acquisition system which would interface with Esduino Xtreme, our microcontroller. The process of this project has increased our knowledge and skills by challenging us with various errors on CodeWarrior and our serial communication software, MATLAB. It has allowed us to innovate and design any data acquisition system on a microcontroller which can help improve any aspect of our society.

The Microcontroller received an analog voltage signal with the help of a preconditioning circuit that allowed the signal to be amplified and to level shift from a pure +/- 2V AC source to a 4V peak to peak centred at 2.5V. The analog voltage signal was then converted into useful information with the use of the analog to digital converter. With the use of a serial communication software, MATLAB we were able to physically see an output in terms of a graph in real time, which actively represented the state and the frequency that was coming from the AC source.

Overall, we can implement our system with other types of transducers that intake temperature, pressure, light, weight, airflow, humidity *et al.* and convert them to do useful tasks such as controlling the temperature from your phone, dimming the light in a room during day and controlling airflow into/out of any system. By adding any digital to analog component we can harness and convert its energy into any medium and make any device as efficient as possible. Furthermore, the use of a microcontroller is endless and with the knowledge we have gained from this project implementing a data acquisition system is straightforward. It is indeed a step forward in improving the technology that already exists in our society and make them more efficient.

Appendix

CodeWarrior

```
//*********************************************************************
//*                        McMaster University                        *
//*                       2DP4 Microcontrollers                       *
//*********************************************************************
//* COMP 2DP4 Final Project                                           *
//* By Shubham Gupta 04/01/2016                                       *
//*********************************************************************

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

#define initSYNR 0x01 // Defines the SYNR variable, and assigns it a value of
0x01, as shown in the calculations.
#define VCOFRQ 0x40
#define initREFDIV 0x00
#define REFFRQ 0x80

char string[20];
unsigned short val;

//function prototypes
void setClk(void); //function initiation for setting up bus speed
void delayby01ms(int k);//function initiation for setting up delay and
sampling frequency
void ADC_output(void);////function initiation for outputting data

void OutCRLF(void){
 SCI_OutChar(CR);
 SCI_OutChar(LF);
 PTJ ^= 0x20; // toggle LED D2
}


void main(void) {
// Refer to Chapter 14 in S12G Reference Manual for ADC subsystem details
// Setup and Enable ADC Channel to 5

//----------------------ADC Settings (Channel 5)--------------------------

 ATDCTL0 = 0x05; //selects channel 5
 ATDCTL1 = 0x2F; // set for 10-bit resolution
 ATDCTL3 = 0x90; // right justified, two sample per sequence
 ATDCTL4 = 0x02; // prescaler = 2; ATD clock = 6.25MHz / (2 * (2 + 1)) ==
1.04MHz
 ATDCTL5 = 0x25; // continuous conversion on channel 5

//-----------------Desired Bus Clock Settings (16 Mhz)----------------------

 setClk(); //Enables the PLL clock, and sets the bus clock to 16 Mhz
```

```
 // Configure PAD0-PAD7 Pins
 ATDDIEN = 0x000F; // Configure PT1AD3-PT1AD0 (A3 to A0 on the Esduino) as
digital inputs (as opposed to analog inputs)
 PER1AD = 0x0F; // Enable internal pull up resistors to avoid indeterminate
state when not connected
 DDR1AD = 0x00; // Set upper 4 bits to outputs (D6, D4, A5, A4 on Esduino)
and lower 4 bits to inputs (A3-A0 on Esduino)

// Setup LED and SCI
 DDRJ |= 0x01; // PortJ bit 0 is output to LED D2 on DIG13
 SCI_Init(9600);
 for(;;) {
 ADC_output();
 }
//For loop to verify Clock
  // for(;;){
 //  PTJ=0x01;
 //  delayby01ms(1000);
  // PTJ=0x00;
  // delayby01ms(1000);
//}
}
void setClk(void)
{
    CPMUSYNR=initSYNR+VCOFRQ; //Set PLL multiplier (0x41 = 0100 0001)
    CPMUREFDIV = initREFDIV+REFFRQ; //Set PLL divider (0x80 = 1000 0000)
    CPMUPOSTDIV=0x00; // Set Post Divider
    CPMUOSC = 0xC0; // Enable external oscillator
    while (CPMUFLG_LOCK == 0) {} // wait for it
    CPMUPLL = CPMUCLKS; // Engage PLL to system.
}
//Delay Function
void delayby01ms(int k){//delays by 100 micro seconds, or a tenth of a
milisecond for more accuracy
 int i;
 TSCR1 = 0x90; /* enable timer and fast timer flag clear */
 TSCR2 = 0x00; /* disable timer interrupt, set prescaler to 1*/
 TIOS |= 0x01; /* enable OC0 */
 TC0 = TCNT + 16000; //16000 was used to be more precise due to the 2.08ms
 for(i = 0; i < k; i++) {
 while(!(TFLG1_C0F));
 TC0 += 16000;
 }
 TIOS &= ~0x01; /* disable OC0 */
}

void ADC_output(void){
 PTJ ^= 0x01; // toggle LED
 val=ATDDR0;

// SCI_OutString("ADC Values:");
 SCI_OutUDec(val);//SCI output value
 SCI_OutChar(CR); //new line
 delayby01ms(2); //This was done to achieve a 480 Hz sampling frequency
}
```

MATLAB

```
% ********************************************************************
% *                       McMaster University                        *
% *                      2DP4 Microcontrollers                       *
% ********************************************************************
% * COMP 2DP4 Final Project                                          *
% * By Shubham Gupta 04/01/2016                                      *
% * MATLAB VERSION r2013a was used                                   *
% * A GUI PUSH BUTTON FROM MATLAB WAS IMPLEMENTED                    *
% ********************************************************************

function varargout = push_button(varargin)
% PUSH_BUTTON MATLAB code for push_button.fig
%      PUSH_BUTTON, by itself, creates a new PUSH_BUTTON or raises the
existing
%      singleton*.
%
%      H = PUSH_BUTTON returns the handle to a new PUSH_BUTTON or the handle
to
%      the existing singleton*.
%
%      PUSH_BUTTON('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PUSH_BUTTON.M with the given input
arguments.
%
%      PUSH_BUTTON('Property','Value',...) creates a new PUSH_BUTTON or
raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before push_button_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to push_button_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help push button
% Last Modified by GUIDE v2.5 05-Apr-2016 19:05:22
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @push_button_OpeningFcn, ...
                   'gui_OutputFcn',  @push_button_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before push_button is made visible.
function push_button_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to push_button (see VARARGIN)
% Choose default command line output for push_button
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes push_button wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% --- Outputs from this function are returned to the command line.
function varargout = push_button_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

delete(instrfindall);% erase and close any ports
s = serial('COM4');% COM4 Selected
s.BaudRate = 9600;%Baud Rate Initialized
s.Terminator = 'CR' %CR is used as terminator
fopen(s);%open com port

%Initialization of Variables
VFS = 5; %full scale voltage
bits = 10;
voltage = 0;
res= VFS/(2^bits);
i = 0;
t=0; v=5; tPrev=0;
plotHandle = line(nan, nan);
Samples=200;
totFreq=0;
avgFreq = 0;

%Plotting Configuration
title('Voltage Vs Time: Shubham Gupta 1416773')
xlabel('Time (s)') %x-axis
ylabel('Voltage V') %y-axis
tic; %counter
while(i<Samples)
     voltage = str2double(fscanf(s))*res;
     time = toc; %current time
     v = [v voltage];
     t = [t time];
     if(mod(i,10)==0) %plotting after 10 samples
     set(plotHandle, 'xData', t, 'yData', v);
```

```
    end
    pause(1e-16);
    freq=1/(time-tPrev); % calculation for frequency
    totFreq= totFreq+ freq;
    tPrev=time;
    i=i+1;
end
toc; %measure total time
avgFreq= totFreq/i %average frequency
fclose(s)
```