2DP4: Microprocessor Systems
Final Project

Instructor: Dr. Doyle and Ms. Fazliani

Amr ABD EL SHAKOUR–L03–abdelsha

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 **Amr ABD EL SHAKOUR, abdelsha, 400008212**

A. Amr, ABD EL SHAKOUR, ABDELSHA, 400008212, L03

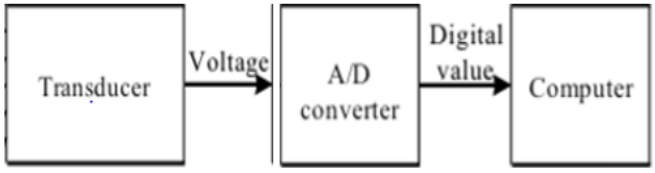
Detection of Abnormal Posture and Balance

I. Introduction

With an increasing elderly population, more and more individuals are prone to falling and shattering their bones. This could be corrected if they were to receive haptic feedback alerting them to when they are out of balance. The needed components for this system would be a microprocessor and an accelerometer. The accelerometer would provide the microprocessor with a digital representation of the current g force experienced. This would then be analyzed to determine whether the individual is prone to falling with their current posture. An example of an accelerometer that can be used would be the ADXL337. It has an operating frequency between 0.5HZ and 1600Hz and sampling rate of 10Hz is enough to sample small changes in the gravitational forces. When a large change in gravitational force is detected by the accelerometer, the sampling rate of 10Hz is below the Nyquist rate and will result in bizarre measurement. This could then alert the patients with haptic feedback or some sort of feedback to re-correct their posture and restore their balance. The purpose of this project was to design and implement a working prototype of a system that is tasked with acquiring inclination angle measurement from 0-90 degrees. This was to be completed using an Esdunio Extreme microcontroller and an ADXL337 accelerometer. The format of this report will go as follows: 1.System Overview, 2.Experiment And Results, 3.Discussion, 4.Conclusion, and 5.Appendix.

II. System Overview

This section will be broken down to the following categories: 1.Transducer, 2. Pin Assignment And ADC Table, 3. Flowchart, 4.Baud Rate, and 5.PC. A visual representation of the break could be expressed in the figure below:



A. Transducer

The transducer that was used for this project is ADXL337. It has a Vs of -0.3V to + 3.6V and an optimal operating

temperature of −55°C to +125°C. The input to the transducer is the gravitational force accomplishes this by using a polysilicon springs that suspend the structure over the surface of a wafer and provides a resistance against acceleration. Bandwidth is within a range of 0.5 Hz to 1600 Hz for X and Y axes and a range of 0.5 Hz to 550 Hz for the Z axis. For this project, only the X axes was used. The Esduino Extreme was to accomplish the A/D conversion. The reason to this choice was due to the flexibility that the Esdunio offers when it comes to multiple analog to digital channels. This would be needed if we choose to incorporate both the Y and Z axes in the future. By shifting the JP7 jumper to 3v, we were able to match that with the Vs of the transducer. This Esduino has 12 A/D conversion channels and comes at a cost of 120 dollars. It can be coded in either C or assembly and for the purpose of this project, C was used. The bus clock could be set up to 24MHz, but for this project, it was scaled down to 8Mhz.

B. Pin Assignment And ADC Table

For this project, the pins that were used were pin 0 to 4 and 6 from port AD and 2 to 5 from port P. pins 0 to 4 on port AD were set as output pins as well as those from port P. The only input pin that was used for this project was pin 6 from port AD as well as IOC0 and IOC1. Port AD is 16 Bits but only 12 were available to be configured to pins. Port P is 8 Bits but only 6 were available to be configured as pins. A visual representation of the ports is present under Datasheet. The resolution that was used for the analog to digital conversion was 10 bits. The table below addresses ADC values based on a 10-bit resolution with the maximum value of 1023:

Table 1: Percentage VS as ADC Values

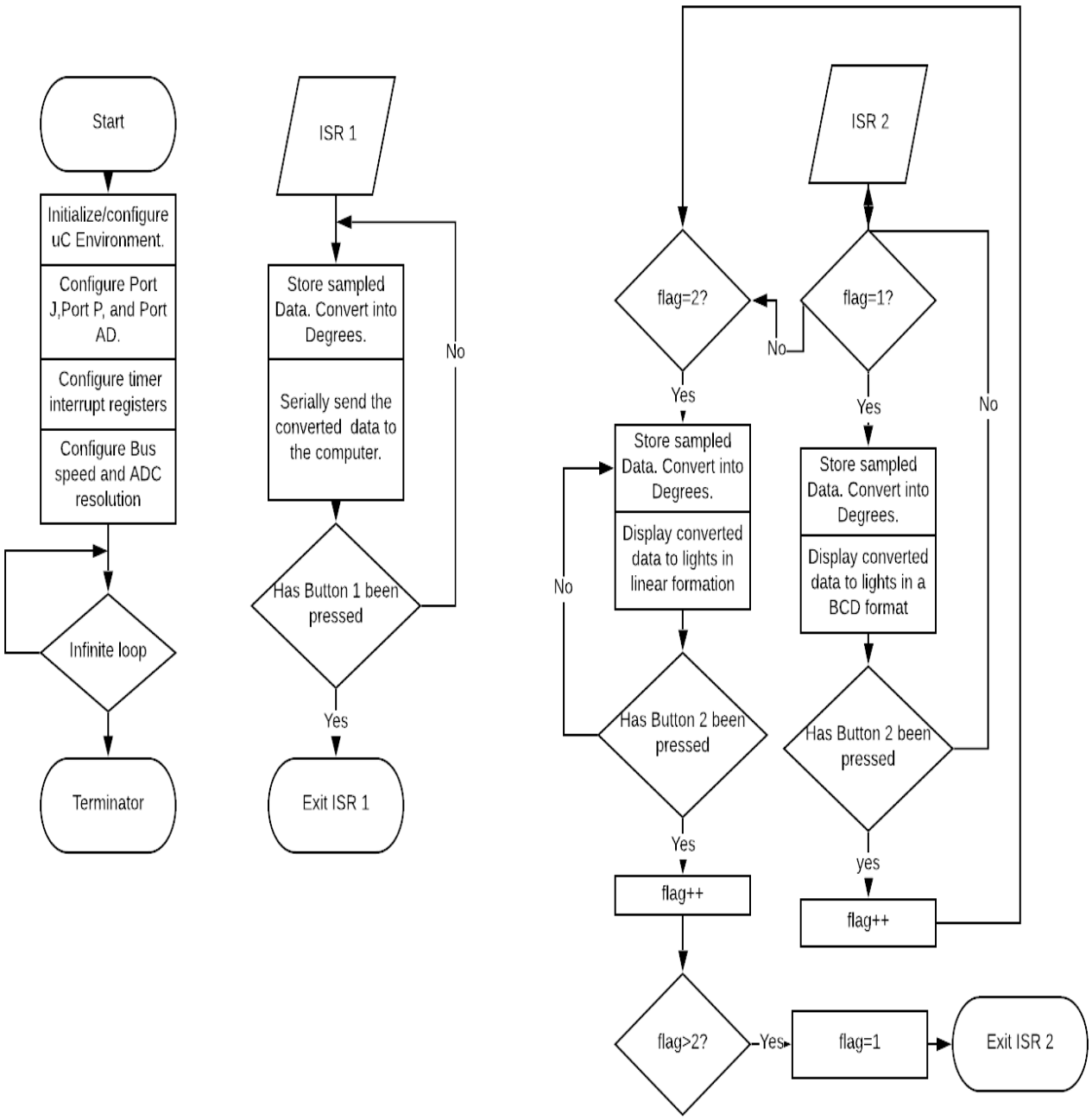
Percentage of VS	ADC Value
0%	0000000000
25%	0011111111
50%	0111111111
75%	1011111111
100%	1111111111

The sampling rate that was chosen for this project was 10Hz. Since this system will be used in a slow environment where a rapid change in angle is not present, it is not necessary to implement a high sampling rate. Assuming the frequency of the change in angles would range from 1 to 5Hz, this system would hold valid. When considering the bottleneck of serial communication, the maximum possible sampling rate is 2880Hz. This would allow for an accurate reproduction of the analog signal ranging from 1 to 1440Hz. Please keep in mind that this is not the maximum value the system is capable of. Also, if the sampling frequency is increased passed 60Hz, visualization of the LED change would be difficult.

C. Flowchart

The Flowchart of the program is shown below:

Flowchart 1: CodeWarrior Algorithm flow chart



ISR 1 is initiated when button 1 is pressed. This button is in an active high state and when pressed it becomes low. Pressing the button again exits ISR 1. Similarly, ISR 2 is initiated when button 2 is pressed. Button 2 acts as a toggle between mode 0 and mode 1. If button 2 is pressed for the first time, the ISR starts in mode 0. By pressing the button a second time, the ISR switches to mode 1. Pressing the button a third time exits from ISR 2. Mode 0 is the BCD representation of the degree tilt detected by the ADXL337. Mode 1 is linear progression bar where it outputs the most significant value of the angle. An example of mode 1 would look as follows: An angle of 26 degrees would be represented by turning on LED 2. An angle of 38 would be represented by turning on LED 3.

D. Baud Rate

The baud rate that was used in this project is 57600 Bits/s. This was achieved with a baud divisor of 9. The table below shows the error associated with choosing 9 as a baud rate divisor.

Table 2: Error Rate Associated With Rounded Baud Devisor

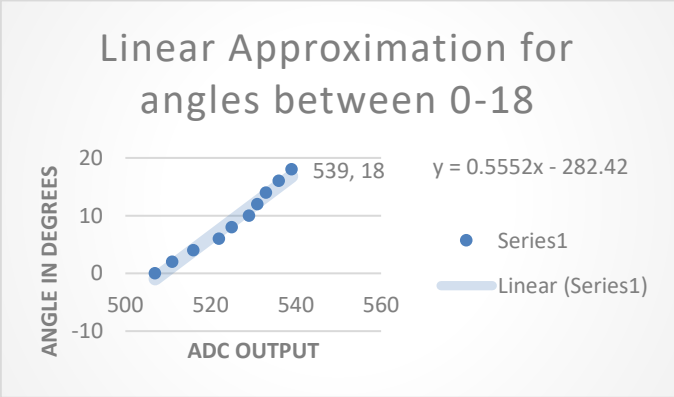
Baud Devisor Rounding	Baud Devisor	Baud Rate	BUSCLK (MHz)	ESDX Rate	Error %
208	208.33	2400	8	2403.8	0.16
104	104.167	4800	8	4807.7	0.16
52	52.0833	9600	8	9615.4	0.16
26	26.0417	19200	8	19230.8	0.16
13	13.0208	38400	8	38461.5	0.16
9	8.68056	57600	8	55555.6	3.55
4	4.34028	115200	8	125000	8.51

The mode used to interface between the microcontroller and the computer is Serial Communications Interface (SCI). The terminator that is used is "CR", and the data being transmitted are the angles correlating to the degree of tilt.

E. PC

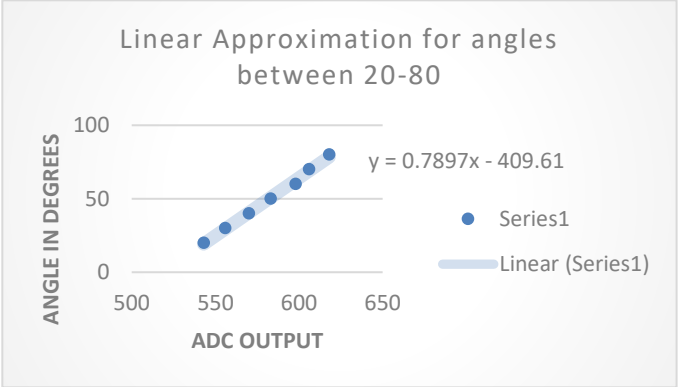
The computer that was used to receive the data was running on an Intel Core i5-8300H CPU running at 2.3 GHz. It contain 8GB of ram and is running windows 10 on an M.2 NVME. It has 3 USB 3.1 ports where only 2 of them were used for this project. The model of the computer is a DELL G3 3379 17 inch laptop. The software that was used to capture the data was MATLAB. The software that was used to program the Esduino was CodeWarrior. The data was converted from raw analog voltage to degrees using linear approximations. This was done by correlating the degree of the tilt with the outputted value of the ADC. The tilt of a degree was measured using a protractor while the outputted value of the ADC was serially communicated to MATLAB using SCI_OutUDec. Three linear approximations were used to gain the best accuracy. The first approximation was from 0 to 18 degrees. This is shown in the graph below.

Graph 1: Linear Approximation for 0 to 18 degrees:



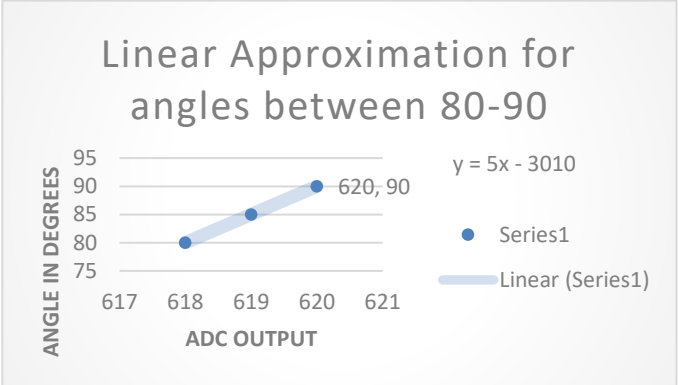
The second linear approximation was for angles between 20 to 80 degrees. This is shown in the graph below.

Graph 2: Linear Approximation for 20 to 80 degrees:



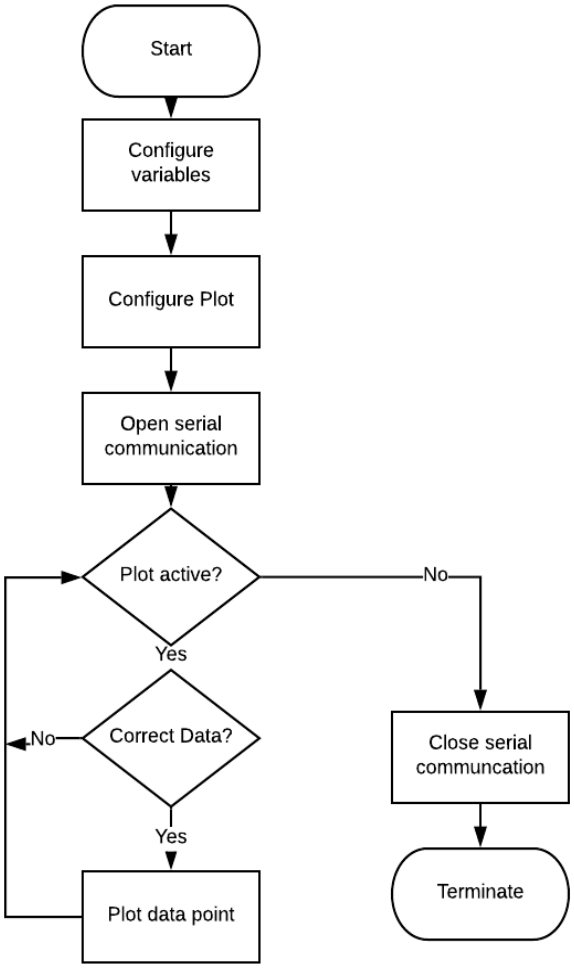
The third linear approximation was for angles between 80 to 90 degrees. This is also shown in the graph below.

Graph 3: Linear Approximation for 80 to 90 degrees:



After computing these linear approximations, displaying the angle in BCD format was accomplished by equating the value of the upper four bits to the angle/10. Since Integers were used to complete this calculation, only the most significant value got stored. For example, and angle of 26 would lead to the upper four bits equalling 2. The lower four were used the mod method to extract the lease significant value. For the same example of 26 mentioned above, the lower four bits would be a representation of 26%10=6. The linear progression accomplished by taking the most significant value of the angle and dividing it by 10. This lead to the values ranging from 0 to 9. These values were then sent to Matlab to live plot. The flowshart for MATLAB is shown below.

Flowchart 2: MATLAB Algorithm flow chart:



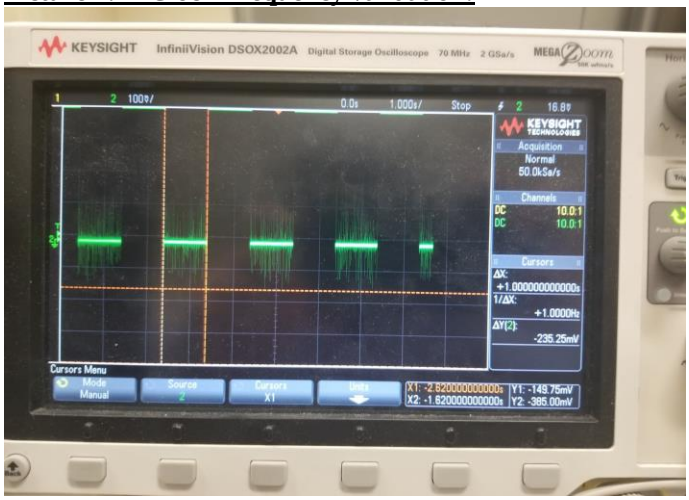
III. EXPERIMENT AND RESULTS

This section will be broken down into two components: 1.Validation and 2.Results.

A. Validation

There were 4 components that were necessary for the overall system to work properly. Those 4 components will be discussed below as along with the methods used to validate that they are working. The first component that was validated was the clock frequency. The function that was used to configure the clock frequency is called "`setClk();`". To validate that the frequency was working properly, a 1 second delay was used to turn on and off an LED. The LED was then attached to an oscilloscope to verify the delay. Attached below is the picture of the oscilloscope reading for a 1 second delay.

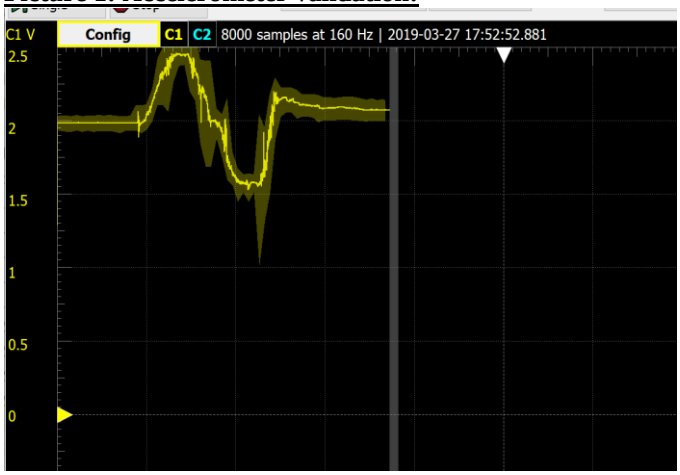
Picture 1: E-Clock Frequency validation:



The above picture validates that the clock is set correctly and is running at 8MHz. This is done by incorporating a 1s delay in CodeWarrior and then verifying it using the oscilloscope.

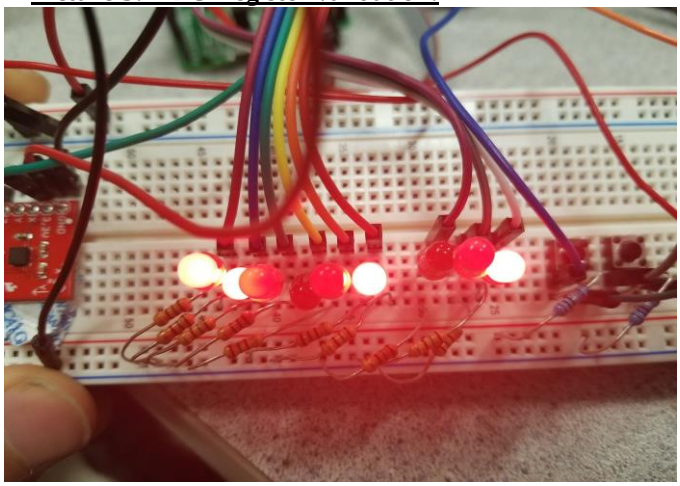
The second component in this project that was validated was the ADXL337 accelerometer. The process used for validation was connecting the X port in the ADXL337 to the oscilloscope and feeding the ADXL337 3.3V. By tilting the ADXL337, the voltage should increase for positive angles and decrease for negative angles. This was verified in the picture below.

Picture 1: Accelerometer validation:



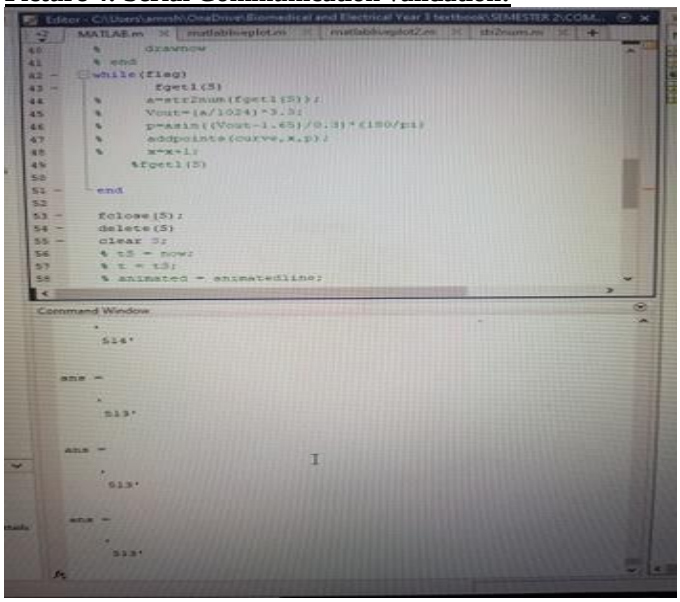
The third component validated component was the ADC registers. The method used for validation involved setting an unsigned short variable equal to the ADDR0 register. By setting the PTDAD register equal to the variable, the LED's would change values depending on the sampled data. The picture below is and example of what a random output from the ADDR0 register looks like.

Picture 3: ADC Register validation:



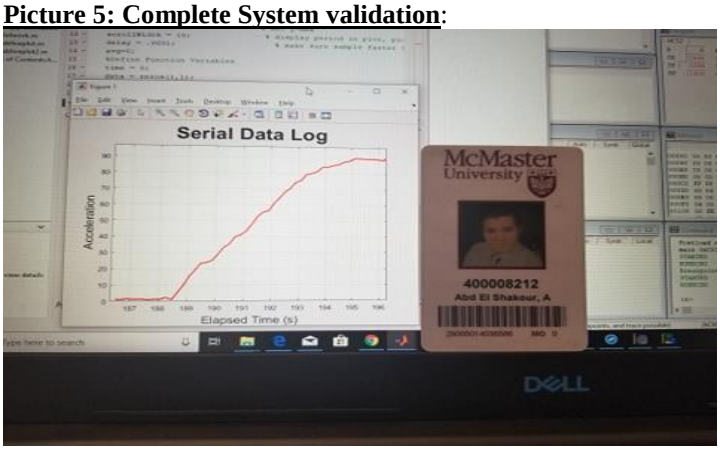
The fourth validated component was serial communication. In order to validate this component, the fgetl() function in MATLAB was used. This determined whether the Baud divisor calculation was correct, and the data was transmitted at the set Baud rate of 57600 Bits/s. The picture below shows that serial communication was working as intended.

Picture 4: Serial Communication validation:



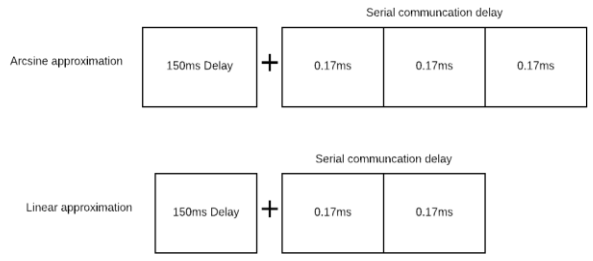
After validating each component separately, the entire system was validated by plotting live inclination angles in MATLAB. This is shown in the picture below:

Picture 5: Complete System validation:



B. Results

With the use of 3 linear approximations, the accuracy was far better than that of one. By using a protractor to position the tilt of the accelerometer, the value displayed on MATLAB was compared to the tilt according to the protractor. It was found that the displayed values were inaccurate from 0 to 4 and from 80-90. The arcsine equation shown below was used to test out whether the inaccuracy was due to the linear approximations or the accelerometer itself. $\text{Arcsine}\left(\frac{\text{Vout}-1.65}{0.3}\right) * \left(\frac{180}{\text{pi}}\right)$. This equation showed that the accelerometer was very inaccurate after 80 degrees and instead of increasing by increments of one, it increases by increments of 10 from 80. This was also due to the outputted ADC values 618,619 and 620 corresponding to angles of 80 to 90 degrees. At approximately 81 degrees, the angle shown from the Arcsine equation is 90 degrees. This leads to an error of 11 percent. By using linear approximation, possible angles above 80 degrees were 80, 85 and 90 degrees. This led to a better approximation of the degree of tilt compared to the arcsine function. Another reason why the linear approximation was used rather than the arcsine function would be due to the extra delay imposed by serial communication. Below is a block diagram showing the delay it would take using linear approximation vs arcsine.



By using the arcsine approximation, we would need to serially send a 3-digit number which would impose a delay of 0.521ms. This is larger than the 0.347ms delay that would be imposed using the linear approximation method. Another benefit of the linear approximation is with an increase in resolution, the delay imposed from serial communication stays constant, but the accuracy increases. Likewise, using the arcsine with a higher resolution would result in more

accuracy but the delay imposed from serial communication will increase. The last benefit seen from the linear approximation is that the system is capable of higher sampling rate. Serial communication becomes the bottleneck to how fast you can sample and not lose data. With that in mind, the fastest sampling rate using linear approximation is 2880Hz. For the arcsine approximation, the maximum is 1919Hz. This frequency will be further reduced for the Arcsine approximation if a higher resolution is used.

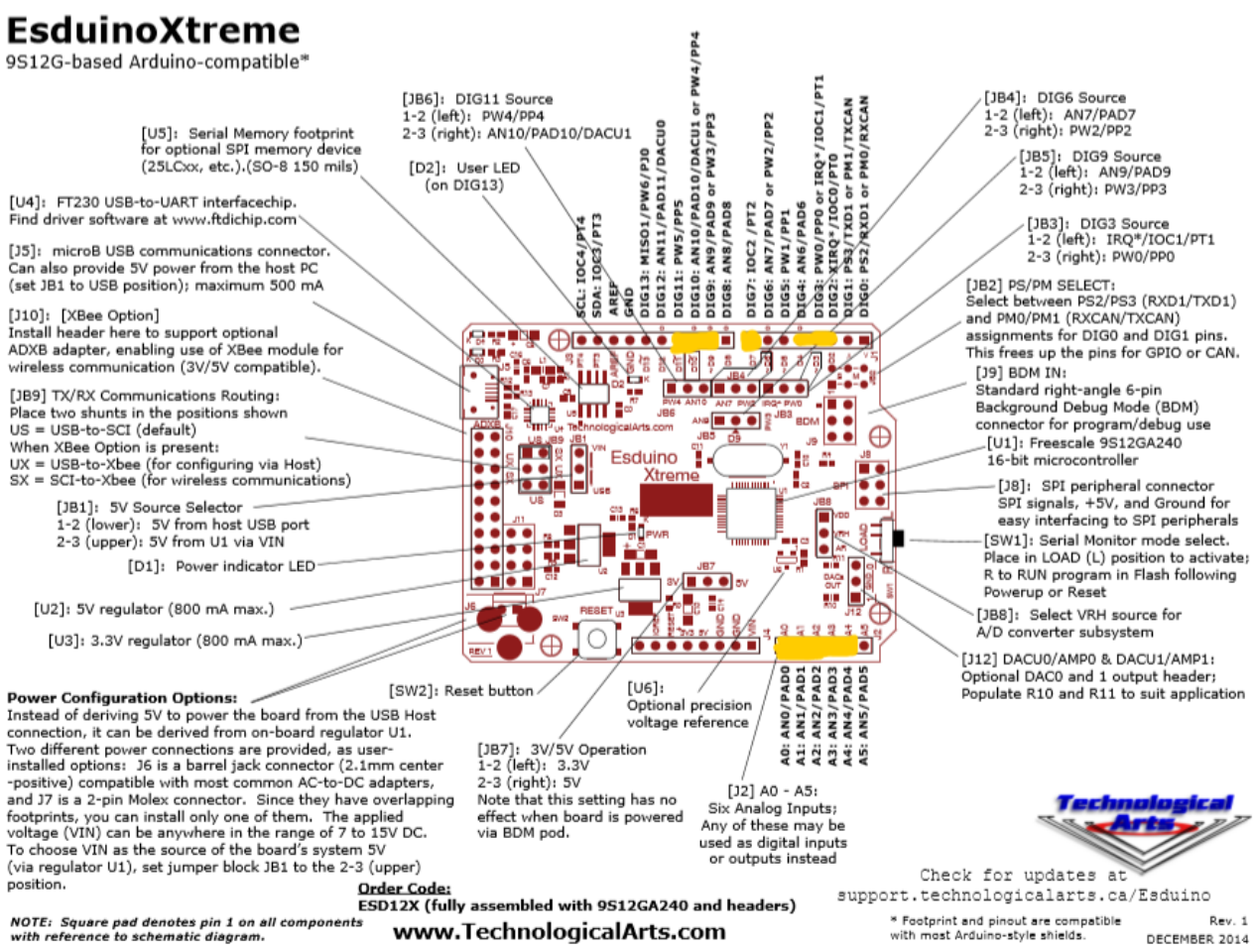
IV. Discussion

Some degree of accuracy had to be sacrificed to overcome the ESDX not having trigonometric functions and floating-point capability. To overcome the absence of trigonometric function, a linear approximation method was used. To do this, the accelerometer was set on a flat surface and the value outputted to MATLAB through serial communication was recorded. This value correlated to 0 degrees. The accelerometer was then tilted to 2 degrees and the correlated MATLAB values was recorded. By increasing the tilt by 2 degrees while using the protractor as the reference, the serially communicated values were recorded in a table. This was done from 0 to 18 degrees and the values were plotted. A line of best fit was then used to derive an equation for linear approximation. Similarly, for 20 to 80 degrees, the increments of degree tilt were increased from 2 to 10. This data was then graphed, and a line of best fit was used to derive the equation for linear approximation. Due to the limitation of the resolution, the difference between 80 and 90 degrees were 618 to 620. This didn't provide enough data points to create an accurate approximation. Nevertheless, those values were used to plot and derive the equation for the line of best fit. For the equation derived for angles between 0 to 18 and 20 to 80, the slope was multiplied by a fraction. In order to overcome this problem, the slope was multiplied by 100. The slope multiplied with the ADC value was completed as two integer multiplication and then it was divided by 100. This allowed for multiplication but sacrificed accuracy. An example of this calculation is as follows: 0.552x-282.42 would be written in CodeWarrior as (79x/100) -409. If x was 556, the error in the calculation is 30.24-30/30= 0.8%. This is the best-case scenario. Similarly, there are certain points where the accuracy is not as good and have an error of upwards of 23 percent. This is seen at 16 degrees. With a 10-bit resolution, the quantization error 0.000977. This represents the minimum amount of change required in voltage to change the digital bits. As shown previously in System overview, the maximum implementable serial communication rate is 57600 Bits/s. This is accomplished with having a Baud Divisor of 9 and an error of 3.55%. A Baud rate of 115200 was not chosen due to the error of 8.51% associated with it. This is greater than the acceptable 6% error and would result in improper serial communication. The element with the most limitation on speed was the human factor. MATLAB was able to handle data faster than 50Hz but pressing the button and removing your finger at that speed was difficult. For that reason, 10Hz was chosen as the sampling speed. Since the frequency of the x axis is between 0.5 to 1600Hz, we can effectively reproduce analog signals up to 1Hz. From 1Hz to 5Hz the quality of the reproduced signal is worse. Greater than 5Hz is below the Nyquist rate and won't reproduce an accurate representation of the analog signal. Analog signals with sharp transitions won't be accurately reproduced to the self-imposed sampling rate of 10Hz. If allowed to sample at the maximum rate of 500KHz, then sharp transitions with a frequency of 250KHz and less could be accurately reproduced. This will require investment in a debounce button and a faster communication standard.

V. Conclusion

In conclusion, the system works as intended to. It effectively live graphs the tilt angle using Matlab and shows the correct LED configuration based on the modes. A couple of improvements could be made to enhance the overall performance and accuracy of the system. First by increasing the resolution from 10 bits to 12 bits, a better representation of angles between 80 to 90 could be derived. This would be far better than the current limitation faced with a 10-bit resolution. Another improvement would be investing in debouncing buttons. This would allow for a smaller delay increasing the communication rate. The application for this system can vary to detecting improper changes in balance and orientation to determining the orientation of electronic devices. For detection of improper changes in balance and orientation of an individual, the sampling frequency has to be increased to avoid picking up excessive amount of noise. The sample rate would be determined by the X and Y frequencies since they have a larger range than that of the Z plane. A sampling rate could be chosen by finding the average frequency of the accelerometer associated with a walking individual. This rate would be able to successfully reproduce the analog signal, but when an individual is in an unbalanced state or is falling, their associated frequencies could be out of the range of the sampling rate and cause spikes in the reproduced signals. Those spikes could then be used to send haptic feedback to the individual to current their posture and restore their balance.

VI. DATASHEET



VII. APPENDIX

a) CodeWarrior C

```
#include <hidef.h>      /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */
#include "SCI.h"//modified baud rate
#include <math.h>

void setClk(void);
void delaby1ms(int);
void ledflash(int);
void delaby1msv2(int);
void ledflash2(int);
void timerinterrupt (void);
void OutCRLF(void);
void serialtrans (void);
void ledprogress(void);
void calc(unsigned short s);
void binarytobcd(void);
//void binarytobcd(void);
int ix=0;
int i=0;
int t=0;
```

```
int k=0;
int m=0;
int MSD;
int LSD;
int MSDD;
int fat=0;
char string[20];
unsigned short n;
int count=1;
unsigned short Val1;
unsigned int Temp;
unsigned short val2;
unsigned int output = 0;
unsigned short input;
signed char a;
int flag=1;


void main(void) {
  //*****************************************************     Port     Configuration
************************************************//

  DDRJ = 0x01;          // Set pin D13 (on board LED) to an output
  PTJ = 0x00;
  DDRP=0xFF;
  PTP=0x00;            // Make sure the LED isn't on to begin with, since that would
be confusing

  // Configure PAD0-PAD7 Pins
    DDR1AD = 0b1111111110111111;        // Configure all of the AD ports except port
6 as outputs
    ATDDIEN = 0xFFBF;     // Configure all of the AD ports except port 6 as digital
outputs
    //PER1AD = 0x40;        // Enable pull-up registers for input pints (A0-A3)

  SCI_Init(57600);

//*****************************************************  IRQ  Intterrupts  setting
************************************************//

  //IRQCR=0b11000000;  // PT1 CONFIGURED FOR FALLING EDGE. THIS TAKES PROIORITY.

//*****************************************************       timer       interrupts
************************************************//
   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 = 0x04;    //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 = 0xFC;     //Timer Input Capture or Output capture
                    //set TIC[0] and input (similar to DDR)
  PERT = 0x03;     //Enable Pull-Up resistor on TIC[0]

  TCTL3 = 0x00;    //TCTL3 & TCTL4 configure which edge(s) to capture
  TCTL4 = 0x0A;    //Configured for falling edge on TIC[0]

/*
 * The next one assignment statement configures the Timer Interrupt Enable
 */

  TIE = 0x03;      //Timer Interrupt Enable
//*****************************************************      Intterrupts      enable
***********************************************//


 //*****************************************************         Main         portion
***********************************************//

 //*****************************************************          clock          set
***********************************************//
    setClk(); //clock set to 8MGHz
    //*****************************************************      ADC      RESOLUTION
***********************************************//

    ATDCTL1 = 0x3F;//0b00101111     // set for 10-bit resolution AND CHANNEL AN6
    //ATDCTL2=0x00
    ATDCTL3 = 0x88;     // right justified, only one channel per sequence
    ATDCTL4 = 0x67; // prescaler = 7; ATD clock = 8.0MHz / (2 * (1 + 1)) == 2MHz
(calulation). sample time select= ATDCLK/4= 500kHZ(sampling time)
    ATDCTL5 = 0x26;      // continuous conversion on channel 6( if we want  multiple
channels then we do     ATDCTL5 = 0x36;)
  //ATDDRn stores the conversion results
  //PTT=0b11111111;
  EnableInterrupts;
  for(;;) {
    //ledflash(10);
    //ledflash2(10);
    //ledflash(10);
  } /* loop forever */
  /* please make sure that you never leave main */
}

 //*****************************************************          Set          Clock
***********************************************//

void setClk(void){
  //VCLOCk=Fvco-2*vref*(syndiv+1)
  CPMUPROT = 0x26;                //Protection of clock configuration is disabled
  CPMUCLKS = 0x80;//vref=1 PLLSEL=1

  CPMUSYNR=0x07;     //VCOFRQ=0,SYNDIV=7
  CPMUFLG=0x01;
  //CPMUPLL=0x01;
  CPMUPOSTDIV=0x00;
  CPMUPLL=0x01;

  //CPMUREFDIV = initREFDV+REFFRQ;//set PLL divider (0x80 = 10 000000)
  //CPMUPOSTDIV=0x00;           // set Post Divider
  CPMUOSC = 0x00;               // Enable external oscillator
 // while (CPMUFLG_LOCK == 0) {}  // wait for it
```

```
  //CPMUPLL = CPMUCLKS;            // Engage PLL to system
}

 //*****************************************************          LED          FLASH
************************************************//

void ledflash(int n){
  for(i=0;i<n;i++){
   PTJ=0x01;
   delayby1ms(1000);
   PTJ=0x00;
   delayby1ms(1000);

  }
}
//*****************************************************          LED          FLASH
************************************************//

void ledflash2(int n){
  for(i=0;i<n;i++){
   PTJ=0x01;
   delayby1msv2(1000);
   PTJ=0x00;
   delayby1msv2(1000);

  }
}

 //*****************************************************                DelayBy1ms
************************************************//

void delayby1ms(int k){
  TSCR1= 0x90; //enable timer and fast timer flag clear
  TSCR2=0x00;//Disable timer interrupt
  TIOS|= 0x01; //enable OC0
  TC0=TCNT +8000;
  for(ix=0;ix<k;ix++){
    while(!(TFLG1_C0F));
    TC0+=8000;
  }
  TIOS &=-0x01;//DISABLE OC0
  timerinterrupt();// restoring the timeinterrut values
}


 //*****************************************************                DelayBy1ms
************************************************//

void delayby1msv2(int k){
  TSCR1= 0x90; //enable timer and fast timer flag clear
  TSCR2=0x07;//Disable timer interrupt
  TIOS|= 0x01; //enable OC0
  TC0=TCNT;
  for(ix=0;ix<k;ix++){
    while(!(TFLG1_C0F));
  }
  TIOS &=-0x01;//DISABLE OC0
  timerinterrupt();// restoring the timeinterrut values
}
```

```
 //*****************************************************  Interrupt  first  Button
setup 1 ************************************************//
interrupt VectorNumber_Vtimch0 void Vtimch(void){
   //unsigned int temp; //DON'T EDIT THIS
  /* DECLARE ALL YOUR LOCAL VARIABLES ABOVE*/

    /* YOUR CODE GOES BELOW*/
  ////////////////////////////////////////////////
     serialtrans();
     Temp=TC0;


}
//*****************************************************  Interrupt  second  Button
setup 2 ************************************************//
interrupt VectorNumber_Vtimch1 void Vtimch01(void){


    if (flag==1){
     binarytobcd();
     flag++;
    }
    if (flag==2){
      fat=1;
      ledprogress();

   }
   PT1AD=0x0000;
   PTP=0x00;
   flag++;
   if (flag>2){

    flag=1;
   }
   delayby1ms(200);
   Temp=TC1;
}


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

//***************************************************** Interrupt One Button setup 1
************************************************//
 void timerinterrupt (void){
    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 = 0x04;    //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 = 0xFC;     //Timer Input Capture or Output capture
                    //set TIC[0] and input (similar to DDR)
  PERT = 0x03;     //Enable Pull-Up resistor on TIC[0]

  TCTL3 = 0x00;    //TCTL3 & TCTL4 configure which edge(s) to capture
  TCTL4 = 0x0A;    //Configured for falling edge on TIC[0]

/*
 * The next one assignment statement configures the Timer Interrupt Enable
 */

  TIE = 0x03;      //Timer Interrupt Enable
 }
//******************************************************* Serial transmission function
**********************//
 void serialtrans (void){
   delayby1ms(150);
   PTJ^= 0x01;
   //delayby1ms(250);
   m=1;
   while (m){
    count++;
    Val1 = ATDDR0 ;
    //SCI_OutUDec(Val1);
    //OutCRLF();
    calc (Val1);
    if (val2>6000){
      val2=0;
    }
    if (val2>90 & val2<6000){
      val2=90;
    }
    SCI_OutUDec(val2);
    OutCRLF();

    //delayby1ms(205);
    m=PTT&0b0000001;  //mask, this will determine if the button is pressed again
    delayby1ms(50);
   }
   count=0;
   delayby1ms(50);
 }
 void calc (unsigned short s){
     if (s<=539){
      val2=((56*s)/100)-282;
     }

     else if (s<=617 & s>539){

      val2=((79*s)/100)-410;
     }
     else if(s>617){
       val2=(5*s)-3010;
     }
```

```
 }
//*****************************************************     linear     light     mode
**********************//
 void ledprogress(void){
      delayby1ms(150);
    t=1;
    //PT1AD=0x0010;
    //PTP=0x1111;
    while (t){
    count++;
    //PT1AD=0x0010;
    Val1 = ATDDR0 ;
    calc (Val1);
    if (val2>100){
      val2=0;
    }
    if (val2>90 & val2<100){
      val2=90;
    }
    MSDD=val2/10;
    if (MSDD==0){
       PTP= 0x00;
       PT1AD=0b0000000000000000;
    }
    else if (MSDD==1){
        PTP= 0x00;
       PT1AD=0b0000000000000001;
    }
    else if (MSDD==2){
       PTP= 0x00;
       PT1AD=0b0000000000000010;
    }
    else if (MSDD==3){
       PTP= 0x00;
       PT1AD=0b0000000000000100;
    }
   else if (MSDD==4){
       PTP= 0x00;
       PT1AD=0b0000000000001000;
    }
    else if (MSDD==5){
       PTP= 0x00;
       PT1AD=0x0010;
    }
    else if (MSDD==6){
       PT1AD= 0x00;
       PTP=0b00000100;
    }
    else if (MSDD==7){
       PT1AD= 0x00;
       PTP=0b00001000;
    }
    else if (MSDD==8){
       PT1AD= 0x00;
       PTP=0b00010000;
    }
    else if (MSDD==9){
      PT1AD= 0x00;
       PTP=0b00100000;
    }
```

```
    //delayby1ms(205);
    t=PTT&0b0000010;  //mask, this will determine if the button is pressed again
    delayby1ms(50);
    }
 }
//*****************************************************   binary   to   bcd   mode
**********************//
 void binarytobcd(void){
    //enter any number from 0 to 9999 here:
    delayby1ms(50);

    k=1;
    while(k){
      delayby1ms(100);
      Val1 = ATDDR0 ;
      calc(Val1);
      if (val2>100){
      val2=0;
    }
    if (val2>90 & val2<100){
      val2=90;
    }
      //input = val2;
      MSD = val2/10;
      LSD =val2%10;
      LSD = LSD<<2;
        PT1AD=MSD;
        PTP = LSD;
        //delayby1ms(100);
        k=PTT&0b0000010;  //mask, this will determine if the button is pressed again
        delayby1ms(50);
      }
 }
```

b) MATLAB

```
clear
clc
delete(instrfindall);
%User Defined Properties
serialPort = 'COM3';
plotTitle = 'Serial Data Log';
xLabel = 'Elapsed Time (s)';
yLabel = 'Acceleration';
plotGrid = 'on';
min = 0;
max = 97;
scrollWidth = 10;
delay = .0001;
avg=0;
%%%%%%%%%%%%%%%%%%%%%%%%Define Function Variables%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
time = 0;
data = zeros(1,1);
count = 0;
%plot setup
plotGraph = plot(time,data(1,:),'-
r','LineWidth',2,'MarkerFaceColor','w','MarkerSize',2);
hold on
```

```
title(plotTitle,'FontSize',25);
xlabel(xLabel,'FontSize',15);
ylabel(yLabel,'FontSize',15);
axis([0 10 min max]);
grid(plotGrid);
%Open Serial COM Port
s = serial(serialPort, 'BaudRate', 57600)
disp('Close Plot to End Session');
fopen(s);

tic

while ishandle(plotGraph) %&& ishandle(plotGraph2) %Loop when Plot is Active
    for i=0:3
        a=str2num(fgetl(s));
        %Vout=(a/1024)*3.3;
        %p=asin((Vout-1.65)/0.3)*(180/pi);
        avg=avg+a;% collects the average of 3 points.
    end
    avg=avg/3;
    if (avg>90)
        avg=90;
    end
    dat = real(avg); %Read Data from Serial
    avg=0;%reset average
if(~isempty(dat) && isfloat(dat)) %Make sure Data Type is Correct
    count = count + 1;
    time(count) = toc;    %Extract Elapsed Time in seconds
    data(:,count) = dat(:,1); %Extract Data Element

    if(scrollWidth > 0)
    set(plotGraph,'XData',time(time > time(count)-scrollWidth),'YData',
data(1,time > time(count)-scrollWidth));

    axis([time(count)-scrollWidth time(count) min max]);
    else
    set(plotGraph,'XData',time,'YData',data(3,:));

    axis([0 time(count) min max]);
    end

    %Allow MATLAB to Update Plot
    pause(delay);
end
end

%Close Serial COM Port and Delete Variables
fclose(s);

clear count dat delay max min plotGraph plotGraph1 plotGraph2 plotGrid...
    plotTitle s scrollWidth serialPort xLabel yLabel;

disp('Session Terminated');


clear str prompt;
```