Analog to Digital Conversion of an Angle

Instructor: Dr. Doyle

Nathen Mathew – L02 – mathen3 – 400074896

Analog to Digital Conversion of an Angle

Nathen, Mathew, mathen3, 400074896, L02

I. Table of Contents

II. Abstract

Abstract—In this paper we study various processes involved in microcontroller systems, such as bus clocks, interrupts, and delays, but mainly the use of serial communication and analog to digital conversion. A device was build using the Esduino Xtreme Microcontroller which would measure an angle and output the angle to a PC. The angle is the analog input that was to be converted to a digital signal, and the output to the PC used serial communication. This system was able to successfully read the analog signal and output it to the PC.

III. Introduction & Background

This report delves into the design of a project that was made to convert an analog signal to a digital one through a process called Analog Digital Conversion (ADC). ADC is used in many places, for example, when recording music. Audio waves are an analog signal, and for these waves to be properly read and manipulated my various computer devices, they must be first converted into Digital signals. In this project, the analog signal came from the accelerometer, whose output correlated to the angle the device was currently in, and we were able to read and quantify this analog signal effectively by first converting it to a digital one.

To quickly breakdown the contents of this report: the design methodology section of the report will go through in detail the design and requirements that were needed for various components of this device. It will also go in depth on the technical aspects of each component. Next, the results section will showcase how each part of this device works in the project. The discussion section will answer a few questions about design choices and address some of the technical aspects of the project. Furthermore, the conclusion will summarize the main points from this report. Finally, the appendix will include all the code used in this project.

In the project, the Esduino Xtreme microcontroller first reads the analog signal from the accelerometer and displays the angle of 0 to 90 degrees using LEDs in 1 of 2 modes. Mode 0 used the LEDs to display the angle as a Binary Coded Decimal, and Mode 1 displayed the angle in a linear bar. This angle would then be serially communicated to the PC and graphically displayed. There were two buttons in this design, the first button toggled the mode of the device and the second turned on and off serial communication with the PC. This will be further discussed in the report.

What we are doing in this project is trying to effectively read an analog signal and convert it into a digital one without losing the integrity of the input. This is important because many real-life devices need accurate digital replications of analog signals. Going back to the music example, what would happen if these audio waves were not

properly replicated? The result would be subpar, and in many cases that is simply not good enough. It is very important to read and convert these signals properly.

IV. Design Methodology

A. Final Pin Assignment

This ADC system was built using the Esduino Xtreme (ESDX) Microcontroller which is described in the ESDX user guide as a "a full-featured Freescale S12 microcontroller having Flash, EEPROM, and SRAM memories, SCI, SPI, and CAN communications subsystems, dual 8-bit DACs, sophisticated 16-bit timer channels, PWM, and multi-channel 12-bit analog-to-digital conversion capability." Seeing that this microcontroller had timer channels, ADC capability, and the ability for Serial communication made this an ideal choice for this project. The pin layout for the ESDX is shown in the figures and tables below.

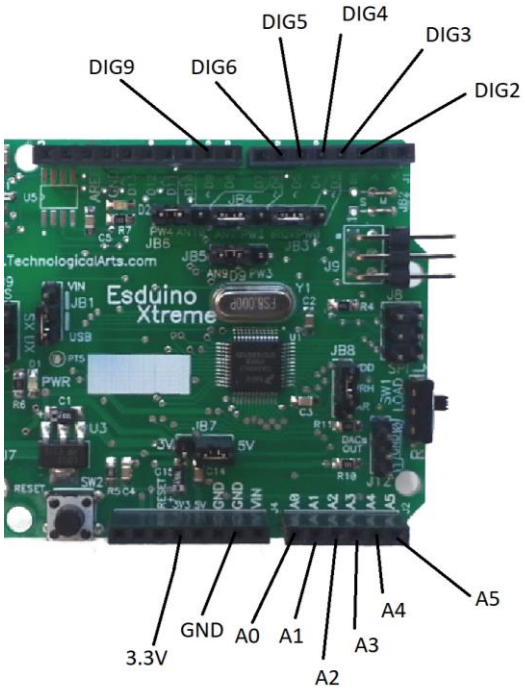

Figure 1: Pin Layout

ON/OFF	DIG2	IOC0
Mode toggle	DIG3	IOC1
ADC channel	DIG4	AN6

Table 1: Pin Layout – Inputs

LED0	DIG5	PP1
LED1	A0	AD0
LED2	A1	AD1
LED3	A2	AD2
LED4	A3	AD3
LED5	A4	AD4
LED6	A5	AD5
LED7	DIG6	PP2
LED8	DIG9	PP3

Table 2: Pin Layout – Outputs

B. Quantify Signal Properties

As stated in the ADXL337 Accelerometer datasheet: "The output signals are analog voltages that are proportional to acceleration. The accelerometer can measure the static acceleration of gravity in tilt-sensing applications as well as dynamic acceleration resulting from motion, shock, or vibration." The accelerometer was connected to a 3.3V power source, as the device was not meant to take anything above 3.6V. To test that the output from the X pin varied correctly, the output was attached to an oscilloscope to verify. The oscilloscope showed that the output voltage would only change to acceleration and tilt in the X direction, and not the Z or Y.

C. Transducer

Figure 2: Accelerometer

A transducer is a device that converts changes of physical quantities to electric signals. In this project, the transducer that was used was an accelerometer. The accelerometer converted acceleration or tilt in the X axis to an electrical signal.

D. Precondition/Amplification/Buffer

This is stage where an input signal is amplified, buffered, level shifted, or filtered. This is called preconditioning, for this project however, it was not required, and no conditioning stages were used.

E. ADC

```
/* setup and enable channel 4 ADC */
ATDCTL1 = 0b00100110; //0x36; 10-bit data transfer
ATDCTL3 = 0b10001000; //right justified, 1 sample/sequence
ATDCTL4 = 0b00000010; //sets prescaler to 2
ATDCTL5 = 0b00100110; //continous conversion on channel 6
```

Figure 3: ADC setup on CodeWarrior

There were specific requirements based on my student number for the ADC conversion. I had to use channel 6 on the ESDX, my ADC resolution was 10 bits, and my bus speed was 8 Mhz. Channel 6 was set up to continuously take in analog input, and this was stored in a variable defined in CodeWarrior was "xout". After using linear approximation, this value was converted into an angle.

F. LED Display MODE 0/1

There are two modes of application, to control which mode the device is in, an interrupt-based button was used. There were 9 LEDs used in the display. Mode 0 displayed the output on the LEDs using BCD. To do this, certain bits of output had to be set without affecting the other bits, so bit-masks were used. This way, the first digit could be set, and then the second BCD digit would be set without affecting the first digit. A series of if-else statements were used to determine which state the LEDs should be in. Mode 1 displayed the angle in a linear bar, the higher the angle the more lights turned on in the bar. Mode 1 similarly used a series of if-else statements, but for mode 1 the entire LED output was changed each time, so no bit-masking was necessary. In mode 1, the first LED would always be on, because it would always be greater than 0 degrees, and in mode 0 the first LED would always be off because only the last 8 LEDs were needed to display 2 BCD numbers, therefore the 1st LED took the output of the mode.

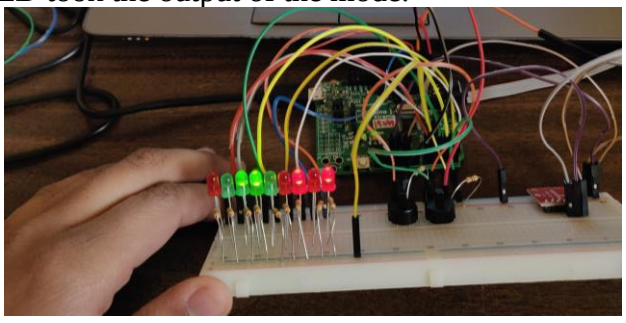

Figure 4: Mode 0 displaying 65 degrees

Figure 5: Mode 1 displaying 65 degree

G. Data Processing

The process used to find the angle from the output of the accelerometer was linear approximation. I measured the output at 0 and 90 degrees, and found an equation connecting these two points in a straight line. This method is good because it avoids any trigonometric functions, something the ESDX cannot use. As seen in figure 6, the output is somewhat linear. The slope does flatten out at the beginning and end which would cause some error, but it would be relatively small.

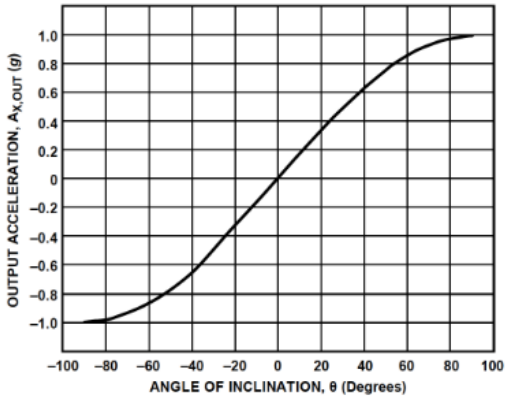

Figure 6: Accelerometer Output

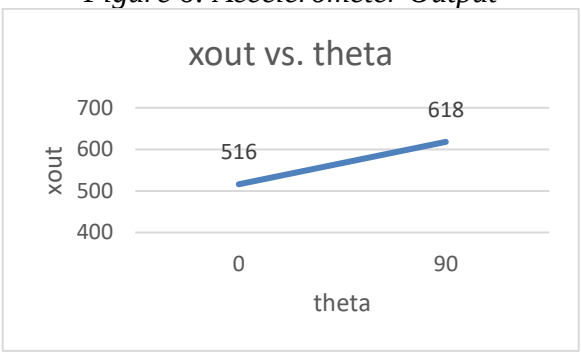

Figure 7: Transfer Function (xout v. theta)

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{618 - 516}{90 - 0} = 1.13$$

$$xout = 1.13x + 516$$

$$theta = \frac{xout - 516}{1.13}$$

$$theta = \frac{(xout - 516)}{113} \times 100 \quad (1)$$

Equation (1) was what was used to find the angle from the accelerometer output. Since the ESDX has no floating-point capability, the fraction 100/113 was used instead of 1/1.13. The programming language used was C.

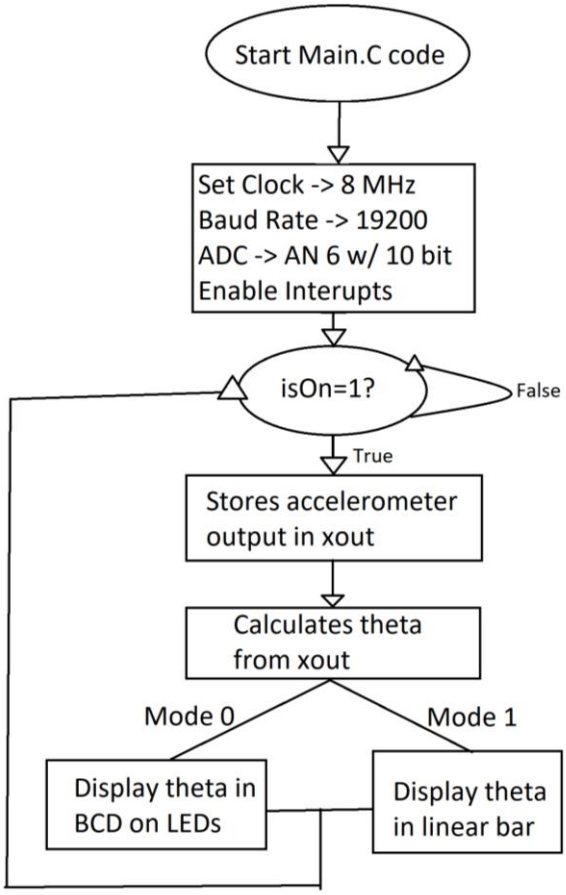

Figure 8: CodeWarrior Flow Chart

H. Control/Communicate

The angle is outputted from the ESDX in two ways, one way is through the LEDs in mode 0 or 1, and the other way is graphically on the PC with MATLAB. The ESDX serially communicates with the PC the calculated value of theta which is outputted as a character and read from MATLAB using “fread(s,1,'uchar')”. Serial Communication can also be stopped by one of the buttons, which also essentially pauses the entire program.

These are the steps to open this project:

1. Connect USBDM and Micro USB to the ESDX
2. Wire the ESDX as shown in the full circuit Diagram
3. Load the application from CodeWarrior
4. Run the Program
5. Open MATLAB and run the MATLAB program (ensuring the correct comport and baudrate is being used)
6. The button connected to DIG2 controls the serial communication and the button connected to DIG3 controls the mode of operation.

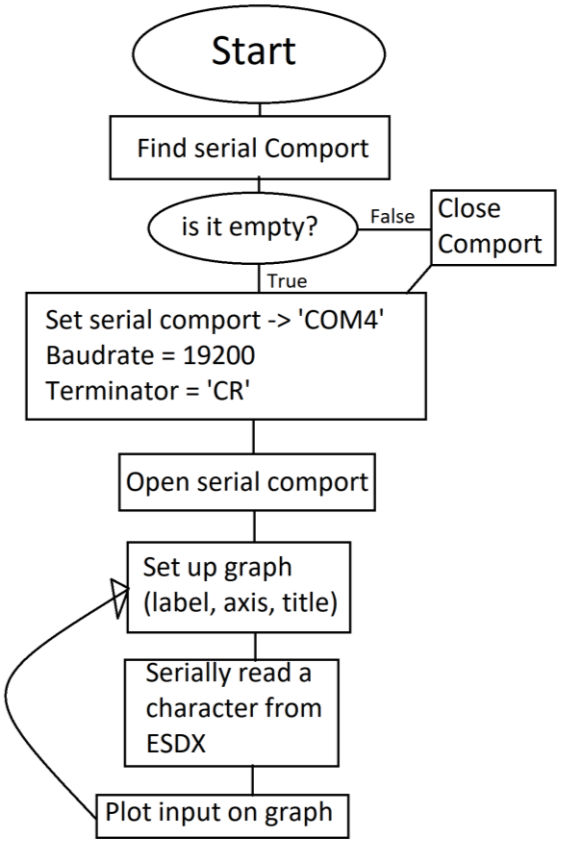

Figure 9: MATLAB Flow Chart

I. Full System Block Diagram

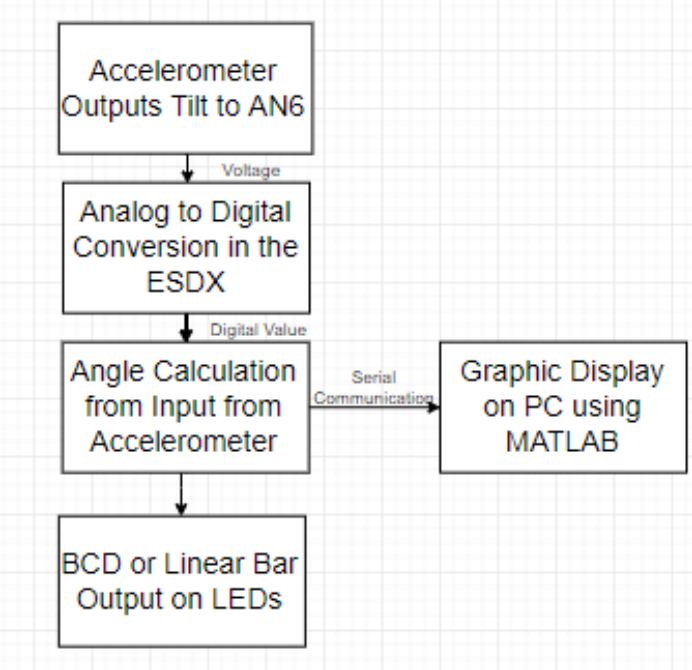

Figure 10: Full System Block Diagram

J. Full System Circuit Schematic

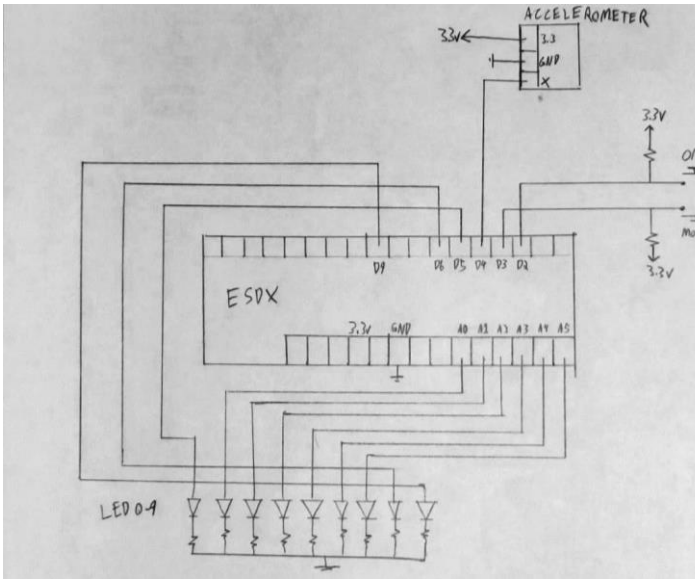

Figure 11: Full System Circuit Schematic

V. Results

A. Accelerometer Testing

Looking at the output from the accelerometer, the output at 0 degrees is approximately 1.5 Volts and the output at 90 degrees is approximately 2 Volts. When changing the inclination in the Z or Y axis, no change occurred in the voltage reading, meaning that it was successfully verified that the accelerometer was outputting a voltage representing the inclination in the X axis.

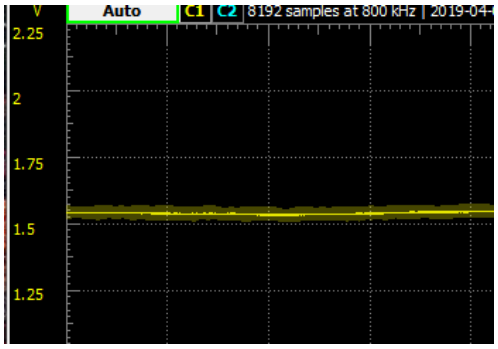

Figure 12: Accelerometer Output at 0 degrees

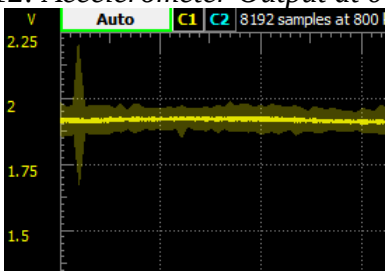

Figure 13: Accelerometer Output at 90 degrees

B. Testing the ADC

Using the wave generator, a triangle wave was created with an amplitude of 250 mV and an offset of 1.75V. This represented a constant signal between 0 and 90 degrees, the result was as expected: a triangle wave ranging from 0 to 90 degrees.

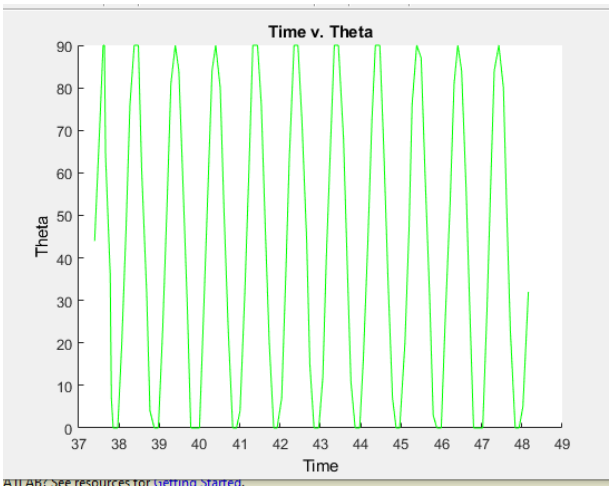

Figure 14:MATLAB Output of Sine Wave Input

C. Testing the program

When moving the accelerometer sensor, the inclination was accurately recorded and displayed in real time on a graph generated by MATLAB. Unfortunately, the graph output is somewhat noisy.

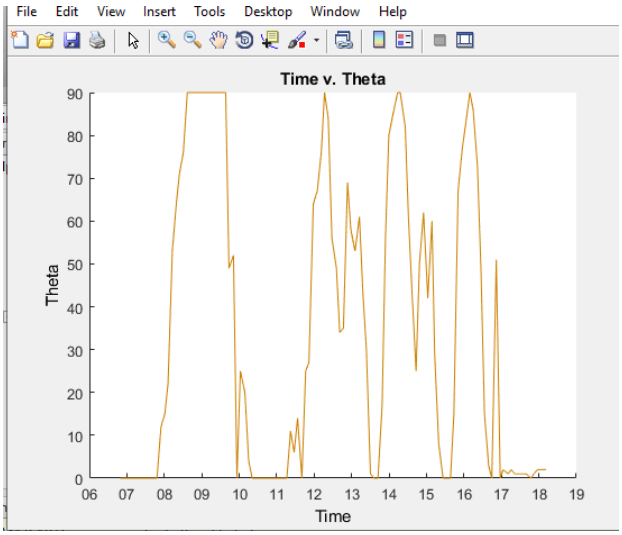

Figure 14: MATLAB Output of Accelerometer Input

D. Overview

This system did work effectively; however, it could be made more accurate. Instead of using just one linear approximation, multiple linear approximations over the entire curve would have resulted in a much more accurate representation of the angle. The ADC did work very well, it was able to convert any signal in the proper range with a high level of effectiveness. Unfortunately, the MATLAB code could not run properly without a delay of around 100ms in the for loop, being the largest limitation of this system.

VI. Discussion

A. Summarize how you were able to overcome the ESDX not having floating point capability and not having trigonometric functions

To overcome the ESDX not having floating point capability, instead of using floats, fractions were used, after calculating and rounding the answer as an integer, no decimal points would be required and therefore not saving the angle as a float was acceptable.

Since the ESDX did not have trigonometric functions, a linear approximation was used instead, since we were measuring only 0 to 90 degrees, this would suffice because in this range the accelerometer output is fairly linear in regard to inclination.

B. Calculate your maximum quantization error

By measuring the voltage output at 0 and 90 degrees to be approximately 1.5 and 2 respectively, we can use these values as the Vmin and Vmax in our maximum quantization error calculation. N is the number of bits.

$$Quantization\ Error = \frac{\Delta V}{2^n} = \frac{2 - 1.5}{2^{10}}$$

$$Quantization\ Error = 4.88 \times 10^{-4} \quad (2)$$

Quantization error results from over rounding values, something which could greatly affect the accuracy of the analog to digital conversion.

C. Based upon your assigned bus speed, what is the maximum standard serial communication rate you can implement. How did you verify?

Another aspect of data processing in this project was sending data over to MATLAB through serial communication. To do this a baud rate would have to be selected with a baud divisor with a percent error of under 6% with the given bus speed of 8MHz.

$$Baud\ Divisor = \frac{8\ MHz}{16 \cdot Baudrate} = 26.04$$

$$\%Error = \frac{|Theoretical - Actual|}{Theoretical} \cdot 100\%$$

$$\%Error = \frac{|26.04 - 26|}{26.04} \cdot 100\% = 0.15\% \quad (3)$$

Since the percent error was less than 6% when using the baud rate of 19200, this is the baud rate that was selected. A similar calculation was done for all the baud rates in the SCI.C file to make the code more robust.

```
switch(baudRate){
  case 2400:   SCI0BDL=208; break;
  case 4800:   SCI0BDL=104;  break;
  case 9600:   SCI0BDL=52;  break;
  case 19200:  SCI0BDL=26;  break;
  case 38400:  SCI0BDL=13;  break;
  default:     SCI0BDL=26 ;  // 19200
}
```

Figure 15: The updated baud divisors in the SCI.C file

D. Reviewing the entire system, which element is the primary limitation on speed? How did you test this

In this system, the greatest limitation on speed was MATLAB. If the delay in the for loop was too

small, MATLAB would unfortunately start lagging and be unable to perform real time output.

E. Based upon the Nyquist Rate, what is the maximum frequency of the analog signal you can effectively reproduce? What happens when your input signal exceeds this frequency

According to the Nyquist theorem, the minimum sampling rate is 2 times the largest frequency from the analog input. The largest frequency from the accelerometer is 500 Hz, as that is the sampling frequency of the accelerometer. This means to avoid any aliasing; the program must sample at a frequency of 1000 Hz and the delay in the for loop would be set to 1ms.

Unfortunately, when the delay was set this small, the MATLAB program would start to lag heavily, therefore a delay of 100ms was used instead. This may have caused some error, but the output still appeared to be accurate.

F. In general, are analog input signals with sharp transitions (e.g., square, sawtooth, etc.) accurately reproduced digitally? Justify your answer.

I would say yes, this is because if the output jumps up in between two consecutive samples, the output seen would be a sharp increase. This is only if the sampling frequency is sufficiently high, when the sampling frequency is higher, then the slope between these points would seem even sharper.

VII. Conclusion

In this paper, we successfully discussed analog to digital conversion in this project. The system built converts an analog signal of an angle to a digital one and displays the result on an LED display and a graphical display using MATLAB. This system can successfully perform analog to digital conversion without losing the integrity of the original signal, and thus the project in large is a success.

VIII. Appendix

A. *Main.C*

```
//*****************************************************************
//*                          FINAL PROJECT
//*                       McMaster University
//*                      2DP4 Microcontrollers
//*                          Lab Section 02
//*                  Nathen Mathew mathen3 400074896
//*****************************************************************
//*****************************************************************
//*                          Description
//*
//*
//* Read an analog input of an angle from the accelerometer through the
//* ESDX and serially output it to the PC
//*
//*****************************************************************


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

/*Prototypes*/
 void setClk8(void);
 void delay1ms(unsigned int multiple);

 unsigned int isOn = 0;
 unsigned int mode = 0;
 unsigned int xout;
 unsigned int theta;
 unsigned int firstDigit;
 unsigned int secndDigit;

void main(void) {

  setClk8(); //set bus clock to 8 MHz

  /* setup and enable channel 4 ADC */
  ATDCTL1 = 0b00100110; //0x36; 10-bit data transfer
  ATDCTL3 = 0b10001000; //right justified, 1 sample/sequence
  ATDCTL4 = 0b00000010; // sets prescaler to 2
  ATDCTL5 = 0b00100110; //continous conversion on channel 6
  //ATDCTL0 not needed, only one register
  //ATDCTL2 = falling edge v rising edge


  DDRJ = 0xFF;      //set all port J as output
  DDRP = 0b00001110;   //p ports 1,2,3 as output  //for LED outputs
  DDR1AD = 0b00111111; //set port AD as output except 6,7
  PTJ = 0x00;    // start LED off
  PTP = 0x00;


  // The next six assignment statements configure the Timer Input Capture
  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 = 0b11111100;     //Timer Input Capture or Output capture
                    //set TIC[0] and input (similar to DDR)
  PERT = 0x03;     //Enable Pull-Up resistor on TIC[0] and TIC[1]  IOC0 + IOC1

  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

  SCI_Init(19200);

  EnableInterrupts;


  for(;;) {


    while(isOn!=0){ //stops serial communication if isOn = 0

      xout = ATDDR0;
      theta = (xout-516)*100/113;
      if (xout < 516) theta = 0;
      if (xout > 618) theta = 90;
      SCI_OutChar(theta);
      //SCI_OutChar(CR);

      if (mode == 0){  // mode 0

        PTJ = 0x00;
        PTP_PTP1 = 0;

        firstDigit = theta/10;
        secndDigit = theta%10;

        if(firstDigit == 0) { PT1AD = 0b00000000; // 0000
        } else if (firstDigit == 1) { PT1AD = 0b00001000; // 0001
        } else if (firstDigit == 2) { PT1AD = 0b00000100; // 0010
        } else if (firstDigit == 3) { PT1AD = 0b00001100; // 0011
        } else if (firstDigit == 4) { PT1AD = 0b00000010; // 0100
        } else if (firstDigit == 5) { PT1AD = 0b00001010; // 0101
        } else if (firstDigit == 6) { PT1AD = 0b00000110; // 0110
        } else if (firstDigit == 7) { PT1AD = 0b00001110; // 0111
        } else if (firstDigit == 8) { PT1AD = 0b00000001; // 1000
        } else if (firstDigit == 9) { PT1AD = 0b00001001; // 1001
        }
```

```
        PT1AD ^= 0b00110000; // PT1AD = 0bXXABXXXX BCD2nd = BAXX
        PTP ^= 0b00001100; // PTP = 0bXXXXABXX BCD2nd = XXBA
        if(secndDigit == 0) { PT1AD &= 0b11001111; PTP &= 0b11110011;
        } else if (secndDigit == 1) { PT1AD &= 0b11001111; PTP &= 0b11111011;
        } else if (secndDigit == 2) { PT1AD &= 0b11001111; PTP &= 0b11110111;
        } else if (secndDigit == 3) { PT1AD &= 0b11001111; PTP &= 0b11111111;
        } else if (secndDigit == 4) { PT1AD &= 0b11101111; PTP &= 0b11110011;
        } else if (secndDigit == 5) { PT1AD &= 0b11101111; PTP &= 0b11111011;
        } else if (secndDigit == 6) { PT1AD &= 0b11101111; PTP &= 0b11110111;
        } else if (secndDigit == 7) { PT1AD &= 0b11101111; PTP &= 0b11111111;
        } else if (secndDigit == 8) { PT1AD &= 0b11011111; PTP &= 0b11110011;
        } else if (secndDigit == 9) { PT1AD &= 0b11011111; PTP &= 0b11111011;
        }

      } else {  // mode 1


        PTJ = 0x01;
        PTP_PTP1 = 1;

        if (theta >80)  {PT1AD=0b00111111; PTP_PTP2=1; PTP_PTP3=1;}
        else if (theta >70)  {PT1AD=0b00111111; PTP_PTP2=1; PTP_PTP3=0;}
        else if (theta >60)  {PT1AD=0b00111111; PTP_PTP2=0; PTP_PTP3=0;}
        else if (theta >50)  {PT1AD=0b00011111; PTP_PTP2=0; PTP_PTP3=0;}
        else if (theta >40)  {PT1AD=0b00001111; PTP_PTP2=0; PTP_PTP3=0;}
        else if (theta >30)  {PT1AD=0b00000111; PTP_PTP2=0; PTP_PTP3=0;}
        else if (theta >20)  {PT1AD=0b00000011; PTP_PTP2=0; PTP_PTP3=0;}
        else if (theta >10)  {PT1AD=0b00000001; PTP_PTP2=0; PTP_PTP3=0;}
        else if (theta <=10) {PT1AD=0b00000000; PTP_PTP2=0; PTP_PTP3=0;}

      }


      delay1ms(100);

    }



  } /* loop forever */
  /* please make sure that you never leave main */


}

interrupt  VectorNumber_Vtimch0 void ISR_Vtimch0(void)// switches mode
{
  unsigned int temp;

  if (isOn == 0) isOn = 1;  // toggles isOn
  else isOn = 0;

  temp = TC0;
}

interrupt VectorNumber_Vtimch1 void ISR_Vtimch1(void)
{
 unsigned int temp;


 if (mode == 0) mode = 1;   // toggles mode
```

```
 else mode = 0;

 temp = TC1;

}

void setClk8(void){
  CPMUPROT = 0x26;      //Protection of clock configuration is disabled, maybe
CPMUPROT=0.

  CPMUCLKS = 0x80;      //PLLSEL=1. Select Bus Clock Source is PLL clock
  CPMUOSC = 0x80;       //OSCE=1. Select Clock Reference for PLLclk as fOSC (8 MHz).

  CPMUREFDIV = 0x41;    //fREF= fOSC/(REFDIV+1) -> fREF= fOSC/(2) -> fREF= 4 MHz.

  CPMUSYNR=0x01;        //VCOCLK = fVCO= 2 * fREF* (SYNDIV+1) -> fVCO= 2 * 4 MHz *
(1+1) fVCO = 16 MHz.

  CPMUPOSTDIV=0x00;     //PLLCLK = VCOCLK/(POSTDIV+1) -> PLLCLK = 16 MHz/(0+1) ->
PLLCLK = 16 MHz.
                        // fBUS=fPLL/2=16 MHz/2 = 8 MHz

  while (CPMUFLG_LOCK == 0) {}  //Wait for PLL to achieve desired tolerance of target
frequency. NOTE: For use when the source clock is PLL. comment out when using external
oscillator as source clock

  CPMUPROT = 1;                 //Protection for clock configuration is reenabled

}

void delay1ms(unsigned int multiple){ //based off of delay from previous labs

  unsigned int i;
  unsigned int j;

  for(j = 0; j<multiple; j++){
    for(i = 0; i<100; i++){
      // Delay
      PTJ = PTJ;
      PTJ = PTJ;
      PTJ = PTJ;
      PTJ = PTJ;
      PTJ = PTJ;
    }
  }

}
```

B. SCI.C

```
// filename  ***************  SCI.C ******************************
// Simple I/O routines to 9S12C32 serial port
// Jonathan W. Valvano 2/14/06

//  This example accompanies the books
//   "Embedded Microcomputer Systems: Real Time Interfacing",
//        Thompson, copyright (c) 2006,
//   "Introduction to Embedded Microcomputer Systems:
//    Motorola 6811 and 6812 Simulation", Brooks-Cole, copyright (c) 2002

// Copyright 2006 by Jonathan W. Valvano, valvano@mail.utexas.edu
//    You may use, edit, run or distribute this file
```

```
//    as long as the above copyright notice remains
// Modified by EE345L students Charlie Gough && Matt Hawk
// Modified by EE345M students Agustinus Darmawan + Mingjie Qiu

//****************************************************************
// Adapted by Carl Barnes @ Technological Arts for S12G - November 2014
//****************************************************************

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

#define RDRF 0x20   // Receive Data Register Full Bit
#define TDRE 0x80   // Transmit Data Register Empty Bit

//------------------------SCI_Init-----------------------
// Initialize Serial port SCI0
// Input: baudRate is tha baud rate in bits/sec
// Output: none
// Important! After reset, MCU clock is in PEI mode (refer to 10.1.2 in S12G Manual),
// so Bus Clock (MCLK) is 6.25 MHz
// baud divisor = MCLK/(16*baudrate)
// baudRate =  2400 bits/sec   SCI0BDL=163
// baudRate =  4800 bits/sec   SCI0BDL=81
// baudRate =  9600 bits/sec   SCI0BDL=41
// baudRate =  19200 bits/sec  SCI0BDL=20
// baudRate =  384000 bits/sec SCI0BDL=10
// sets baudRate to 96000 bits/sec if doesn't match one of the above
void SCI_Init(unsigned short baudRate){
  SCI0BDH = 0;   // br=MCLK/(16*baudRate)

  switch(baudRate){
    case 2400:  SCI0BDL=208; break;
    case 4800:  SCI0BDL=104;  break;
    case 9600:  SCI0BDL=52;  break;
    case 19200: SCI0BDL=26;  break;
    case 38400: SCI0BDL=13;  break;
    default:    SCI0BDL=26 ;  // 19200
  }
  SCI0CR1 = 0;
/* bit value meaning
    7   0    LOOPS, no looping, normal
    6   0    SCISWAI, SCI stop in Wait Mode
    5   0    RSRC, not appliable with LOOPS=0
    4   0    M, 1 start, 8 data, 1 stop
    3   0    WAKE, wake by idle (not applicable)
    2   0    ILT, short idle time (not applicable)
    1   0    PE, no parity
    0   0    PT, parity type (not applicable with PE=0) */
  SCI0CR2 = 0x0C;
/* bit value meaning
    7   0    TIE, no transmit interrupts on TDRE
    6   0    TCIE, no transmit interrupts on TC
    5   0    RIE, no receive interrupts on RDRF
    4   0    ILIE, no interrupts on idle
    3   1    TE, enable transmitter
    2   1    RE, enable receiver
    1   0    RWU, no receiver wakeup
    0   0    SBK, no send break */
}
```

```
//------------------------SCI_InChar------------------------
// Wait for new serial port input, busy-waiting synchronization
// The input is not echoed
// Input: none
// Output: ASCII code for key typed
char SCI_InChar(void){
  while((SCI0SR1 & RDRF) == 0){};
  return(SCI0DRL);
}

//------------------------SCI_OutChar------------------------
// Wait for buffer to be empty, output 8-bit to serial port
// busy-waiting synchronization
// Input: 8-bit data to be transferred
// Output: none
void SCI_OutChar(char data){
  while((SCI0SR1 & TDRE) == 0){};
  SCI0DRL = data;
}


//------------------------SCI_InStatus-------------------------
// Checks if new input is ready, TRUE if new input is ready
// Input: none
// Output: TRUE if a call to InChar will return right away with data
//         FALSE if a call to InChar will wait for input
char SCI_InStatus(void){
  return(SCI0SR1 & RDRF);
}

//-----------------------SCI_OutStatus-----------------------------
// Checks if output data buffer is empty, TRUE if empty
// Input: none
// Output: TRUE if a call to OutChar will output and return right away
//         FALSE if a call to OutChar will wait for output to be ready
char SCI_OutStatus(void){
  return(SCI0SR1 & TDRE);
}


//------------------------SCI_OutString------------------------
// Output String (NULL termination), busy-waiting synchronization
// Input: pointer to a NULL-terminated string to be transferred
// Output: none
void SCI_OutString(char *pt){
  while(*pt){
    SCI_OutChar(*pt);
    pt++;
  }
}

//----------------------SCI_InUDec-------------------------------
// InUDec accepts ASCII input in unsigned decimal format
//     and converts to a 16 bit unsigned number
//     valid range is 0 to 65535
// Input: none
// Output: 16-bit unsigned number
// If you enter a number above 65535, it will truncate without an error
// Backspace will remove last digit typed
unsigned short SCI_InUDec(void){
unsigned short number=0, length=0;
```

```
char character;
  character = SCI_InChar();
  while(character != CR){ // accepts until <enter> is typed
// The next line checks that the input is a digit, 0-9.
// If the character is not 0-9, it is ignored and not echoed
    if((character>='0') && (character<='9')) {
      number = 10*number+(character-'0');   // this line overflows if above 65535
      length++;
      SCI_OutChar(character);
    }
// If the input is a backspace, then the return number is
// changed and a backspace is outputted to the screen
    else if((character==BS) && length){
      number /= 10;
      length--;
      SCI_OutChar(character);
    }
    character = SCI_InChar();
  }
  return number;
}


//-----------------------SCI_OutUDec-----------------------
// Output a 16-bit number in unsigned decimal format
// Input: 16-bit number to be transferred
// Output: none
// Variable format 1-5 digits with no space before or after
void SCI_OutUDec(unsigned short n){
// This function uses recursion to convert decimal number
//   of unspecified length as an ASCII string
  if(n >= 10){
    SCI_OutUDec(n/10);
    n = n%10;
  }
  SCI_OutChar(n+'0'); /* n is between 0 and 9 */
}



//---------------------SCI_InUHex----------------------------------------
// Accepts ASCII input in unsigned hexadecimal (base 16) format
// Input: none
// Output: 16-bit unsigned number
// No '$' or '0x' need be entered, just the 1 to 4 hex digits
// It will convert lower case a-f to uppercase A-F
//     and converts to a 16 bit unsigned number
//     value range is 0 to FFFF
// If you enter a number above FFFF, it will truncate without an error
// Backspace will remove last digit typed
unsigned short SCI_InUHex(void){
unsigned short number=0, digit, length=0;
char character;
  character = SCI_InChar();
  while(character != CR){
    digit = 0x10; // assume bad
    if((character>='0') && (character<='9')){
      digit = character-'0';
    }
    else if((character>='A') && (character<='F')){
      digit = (character-'A')+0xA;
```

```
    }
    else if((character>='a') && (character<='f')){
      digit = (character-'a')+0xA;
    }
// If the character is not 0-9 or A-F, it is ignored and not echoed
    if(digit <= 0xF){
      number = number*0x10+digit;
      length++;
      SCI_OutChar(character);
    }
// Backspace outputted and return value changed if a backspace is inputted
    else if((character==BS) && length){
      number /= 0x10;
      length--;
      SCI_OutChar(character);
    }
    character = SCI_InChar();
  }
  return number;
}

//-------------------------SCI_OutUHex---------------------------
// Output a 16 bit number in unsigned hexadecimal format
// Input: 16-bit number to be transferred
// Output: none
// Variable format 1 to 4 digits with no space before or after
void SCI_OutUHex(unsigned short number){
// This function uses recursion to convert the number of
//   unspecified length as an ASCII string
  if(number >= 0x10){
    SCI_OutUHex(number/0x10);
    SCI_OutUHex(number%0x10);
  }
  else{
    if(number < 0xA){
      SCI_OutChar(number+'0');
     }
    else{
      SCI_OutChar((number-0x0A)+'A');
    }
  }
}


//------------------------SCI_InString------------------------
// Accepts ASCII characters from the serial port
//    and adds them to a string until <enter> is typed
//    or until max length of the string is reached.
// It echoes each character as it is entered.
// If a backspace is entered, the string is modified
//    and the backspace is echoed
// terminates the string with a null character
// uses busy-waiting synchronization on RDRF
// Input: pointer to empty buffer, size of buffer
// Output: Null terminated string
// -- Modified by Agustinus Darmawan + Mingjie Qiu --
void SCI_InString(char *bufPt, unsigned short max) {
int length=0;
char character;
  character = SCI_InChar();
  while(character != CR){
```

```
    if(character == BS){
      if(length){
        bufPt--;
        length--;
        SCI_OutChar(BS);
      }
    }
    else if(length < max){
      *bufPt = character;
      bufPt++;
      length++;
      SCI_OutChar(character);
    }
    character = SCI_InChar();
  }
  *bufPt = 0;
}
```

C. MATLAB

```
%% finds if comport is not been closed and deleted and does so if it has been not
old = instrfind;
if(~isempty(old))
    fclose(old);
    delete(old);
end
clear;

%%set serial port
s = serial('COM4','BaudRate',19200,'Terminator','CR');
fopen(s);

%%axis variables
time = now;
theta = 0;

%figure
display = figure('Name', 'Time v. Theta');

% set axes and ranges
axesHandle = axes('Parent',display,'Color',[1 1 1]);
hold on;
ylim([0 90]);

% titles and labels
title('Time v. Theta','FontWeight','bold','Color','k');
xlabel('Time','Color','k');
ylabel('Theta','Color','k');

% plot data on graph
count = 1;
while true
    time(count) = now;
    theta(count) = fread(s,1,'uchar');
    count = count + 1;
    plot(axesHandle,time,theta,'Color','#D95319');
    datetick('x','SS');
    pause(0.01);
end
```