2DP4: Microprocessor Systems Project

Data Acquisition and Display: Digital Inclination Angle Measurement System

INSTRUCTOR: MS. FAZLIANI

LAB TAs: SALAH HESSIEN, MOHAMED ABUELALA

RAJDEEP MUDHAR

MUDHARR

400088877

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 Rajdeep Mudhar, mudharr, 400088877

TABLE OF CONTENTS

Introduction and Background

The purpose of this project is to use the properties of ADC (analog to digital) conversion and serial communication, as well as microprocessor fundamentals to acquire and calculate an angle generated from the ADXL337 accelerometer. It will then be displayed via a BCD (binary converted decimal) representation through 8 LEDS and a live graph generated on the computer through MATLAB. It will feature a single interrupt mode button for measuring x and y directions, as well as start stop functionality to momentarily halt serial communication.

This project will also give hands on experience in embedded systems and insights on how real-world data acquisition systems operate, such as accelerometers used in drones to measure their tilt relative to the horizon. The important aspect however is how the data is acquired and computed, and the aspects of ADC are used to acquire such data.

A real-life example of an ADC is a microphone. A microphone takes the waveforms generated by sound and converts them to digital values to be processed and converts them back to analog signal for other things like amplification or modulation. In this project, an accelerometer will be used to convert the analog voltage generated from it to a digital value that can be processed into an angle.

For this project, the following pre-assignments were given according to my student number 400088877:

5.3.1: ADC Channels AN5 (Mode 0: x), AN4 (Mode 1: y)

5.3.2: ADC Resolution – 12 bits

5.3.3: Bus Speed – 8MHz

Design Methodology

Final Pin Assignment Map

The pins were chosen based on the given channels for the ADC. A pin map was made so that simple conversions could be made for the BCD and mode button. The pin assignment map is as follows:

Mode Switch Button: DIG2 (interrupt service port)

Mode 0 (x): A5 (AN5 analog read port)

Mode 1 (y): A4 (AN4 analog read port)

BCD Display LEDS: A0-A4 (lower 4 bits – 1s place) and A6-A9 (upper 4 bits – 10s place)

Quantifying Signal Properties

To successfully convert the given signal to an angle, an equation needed to be made such that the full range of angles could be displayed given the voltage reading. First the datasheet of the ADXL337 was consulted. It was realized that the arcsine of the acceleration over the ideal value of gravity would give the correct angle, however it cannot be computed in CodeWarrior. The go-to response was to use linear approximation, which although would cause a significant error on lower angles, would be the easier route to make calculations on the ESDX simpler.

To quantity the analog signal, random SparkFun tutorials online were consulted and it was found that when the accelerometer outputs 3.3V, the acceleration is 3g, and when 1.65V is output, then the acceleration is 1g. Knowing the ADC resolution is 12, and that the output of the accelerometer outputs +-3g's, then there are 0.55g's per volt (6 g's represented by 3.3V), then an equation can be made to convert the voltage to acceleration in g. To derive this equation, the resolution of the ADC (12 in this case) can be used to quantify the signal. If $2^{12} = 4096$, then the max decimal output by the ADC is 4095, and so a reading of 4095/2 = 2047.5 will give a reading of 0 g's. If 4095 gives a reading of 3g's, then the generic conversion from the decimal reading on the serial port to g is given by the following equation:

g = out*6/4095 – 3, where 'out' is the decimal reading at the serial port.

Next, linear approximation can be used to calculate the angle using the Esduino.

Furthermore, the max frequency at which the ADXL337 sends data is 500Hz, so the sampling frequency cannot be any lower than this. The source of the signal comes the 3-axis sensor which is then fed into an amplifier and then demodulated to get each signal from x, y, z out.

Transducer

The transducer is what converts the real-world analog acceleration into quantifiable voltage. On the ADXL337, there is a 3-axis sensor which is filtered and amplified to generate voltages depending on the angle of the PCB. These voltages are then fed into the ADCs on the Esduino to be quantified into a digital value.

Precondition/Amplification/Buffer

There is an optional stage between the transducer and ADC that can be implemented to amplify or level shift depending on the ADC used. Because the board is being supplied a voltage of 3.3V the reference voltage is at 3.3V and does not need to be changed. Also, the preconditioner can also act as a power supply decoupling circuit to reduce any noise by adding capacitors and resistors. For the purpose of this project, and because the preconditioner is purely optional, it was not used.

ADC (Analog to Digital Conversion)

To setup the ADC on the Esduino, the ATD control registers were modified. The voltage supplied to the accelerometer is via the 3.3V supply on the Esduino. This was changed by setting JB7 left. Also, the V_{ref} for the ADC is determined by modifying JB8. JB8 was set to V_{DD}, which

is 3.3V, thus satisfying the voltage requirement and reference voltage for the ADC. For the sampling frequency, the prescaler, ATDCTL4 can be used to lower the rate at which the analog data is sampled. Because the rate at which the accelerometer is much lower (500Hz), due to the 0.01uF capacitors at the ADXL337 output pins, there was no need to change the sampling frequency for the ADC, as it followed the bus clock of 8MHz. The channels that were chosen according to my student number are AN5 and AN4 for x and y (mode 0 and mode 1), respectively. To set these channels, control register 5 was set to 0x34, effectively sampling channels 4 and 5 continuously. Also, the resolution is determined by control register 1. To set a 12-bit resolution, control register 1 was set to 4F, by viewing the bits in the mc9s12ga240 manual. To allow for multi-conversion sequencing, control register 3 was set to 0x90 to right justify and generate two samples per sequence.

LED Display in Mode 0 and Mode 1

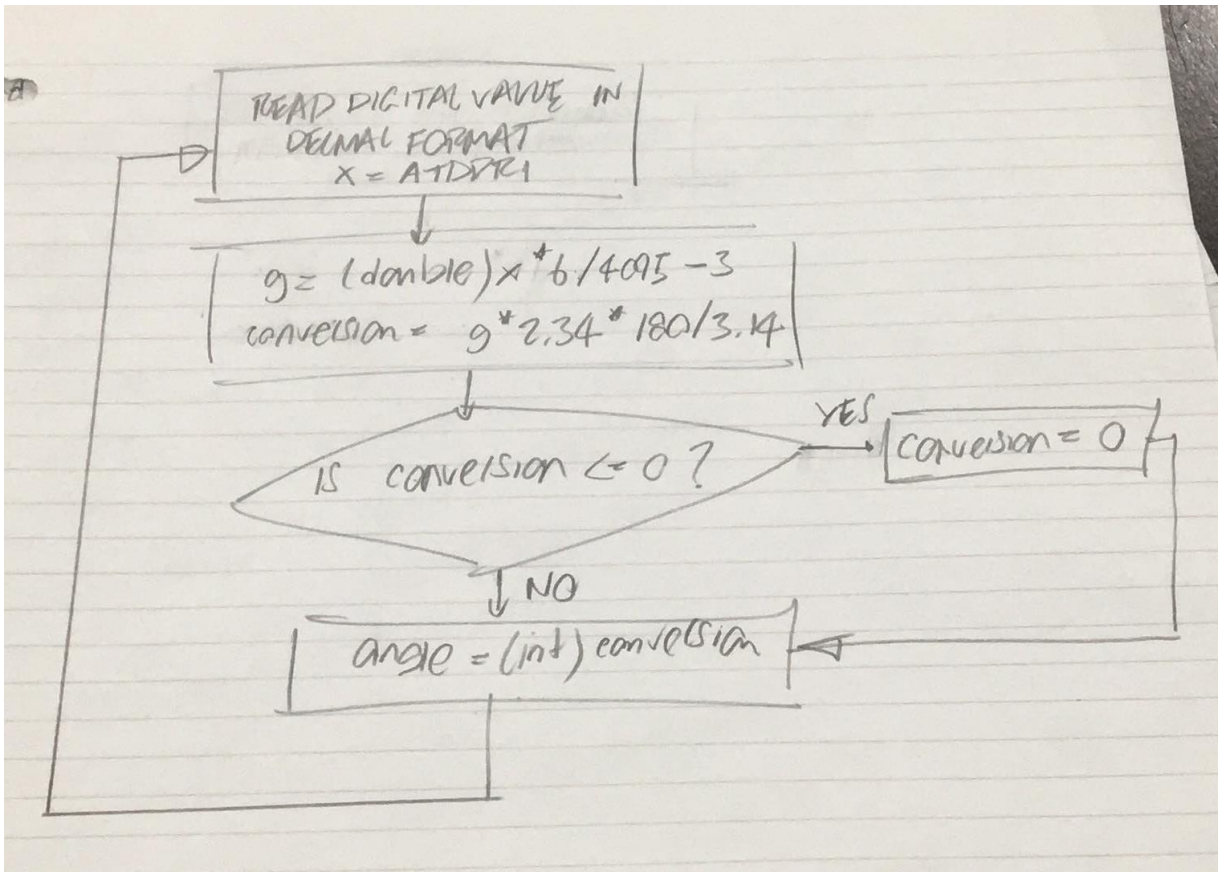
The design behind modes 0 and 1 remain the same, as the BCD uses whatever value it is given from mode 0 or mode 1. The conversion from angle to BCD uses two lines. Also, since port AD bits 0-3 and 6-9 were used, the jumpers that select bits 9-12 to be used were set, and instead of using DDR1AD, DDR01AD was used, which modifies bits 9 and above, unlike DDR1AD. Thus, the bits that will be used to display the angle values are in the form 0b'1111'00'1111', which signifies that all LEDs, are on, for example.

To convert the angle reading to BCD, bitwise and modulus operators were used. To set the upper 4 bits (Port AD 6 to 9), the 10s column from the calculated angle is needed. To get the 10s column, the integer value was divided by 10 and the modulus 10 was used on the result. The final result was stored in PT01AD bits 6 to 9 by bit shifting 6 times to the right (<<6). After these, the lower four bits were needed to be set. To get the lower 4 bits, the 1s column was needed. The integer angle was simply divided by 10 to get the remainder by using modulus and then added to PT01AD.

Data Processing

This explanation can be continued from "Quantifying Signal Properties' on page 4. Knowing that the reading on the serial port can vary between 0 and 4095 given a 12-bit resolution and 3.3v reference voltage, this was tested by connecting the ADXL337 to the ADC channel and reading the digital value from the resulting register. When the board was around 0 degrees, a value of 2050 was showing, which should be correct since a value of 4095/2 or 2047.5 should give a value of 0g or 0 degrees. As the board was tilted up to 90 degrees, the maximum value that was obtained was around 2500. This was an issue as when using the equation of g = 'out'*6/4095 – 3 as discussed in page 4, the resulting g value was around 0.65-0.7. According to the resolution and ADX337, a g value of 1 should equate to 90 degrees. There was no solution to this problem other than using a scaling factor k to equate this value to 90 degrees. The scaling factor that was chosen using linear approximation was 2.34. This value was specifically chosen so that the range of 90 could be fulfilled when the board was tilted at its max value of 90. It was chosen as 2.34 since to get the angle from a g range of 0.65-0.7, the resultant g value was multiplied by 180 and divided by 3.14 to get a max value of 90.

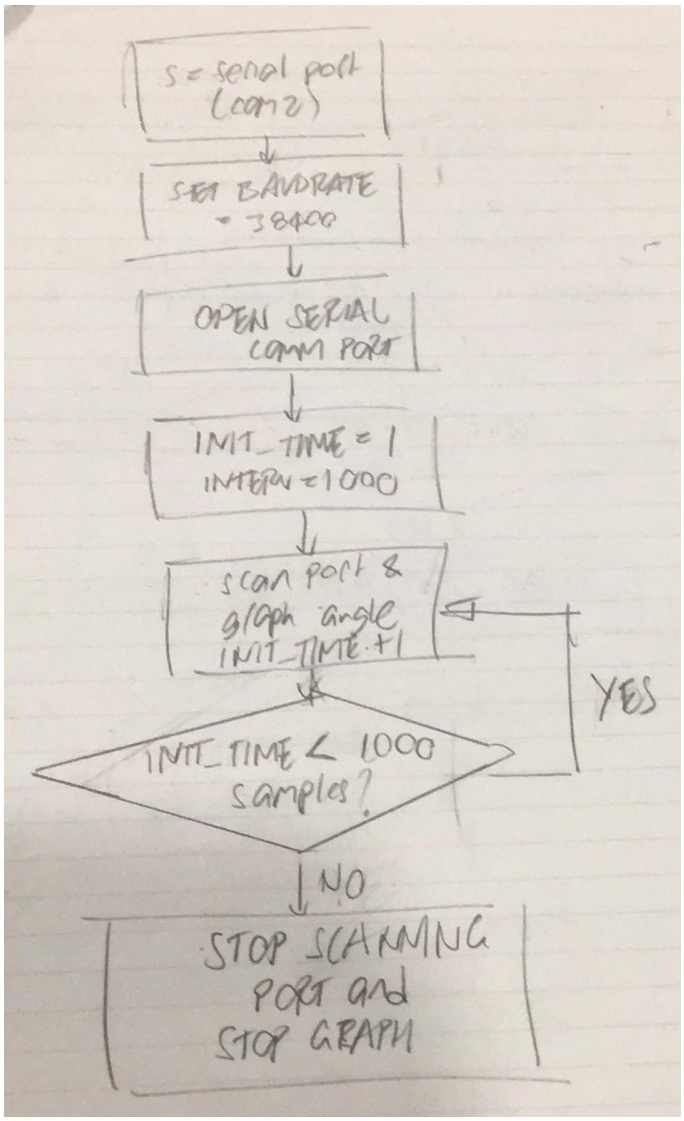
The algorithm below shows how the reading from the ADC register for x is converted to an angle using linear approximation. The same flowchart would be for y but at ATDDR0. It first reads the decimal value, converts it to a representable acceleration value in (g), and then converts the respective g value to an angle, while also ensuring negative angles are set to 0.



Control/Communication

The computed angle, which is typecasted to an int is sent through the serial communication port (COM2). Next, using MATLAB, the port is read at a baud rate of 38400, although this depends on the delay that is set between each interval while reading the ADC resultant register in the Esduino. In MATLAB, an initial time variable is set with a max interval length of x values. These represent the number of data points the graph will plot for the sample rate at which the data is sent through the Esduino. An array is made to continuously load the calculate angles in the data and then graphed in real time until the interval is satisfied.

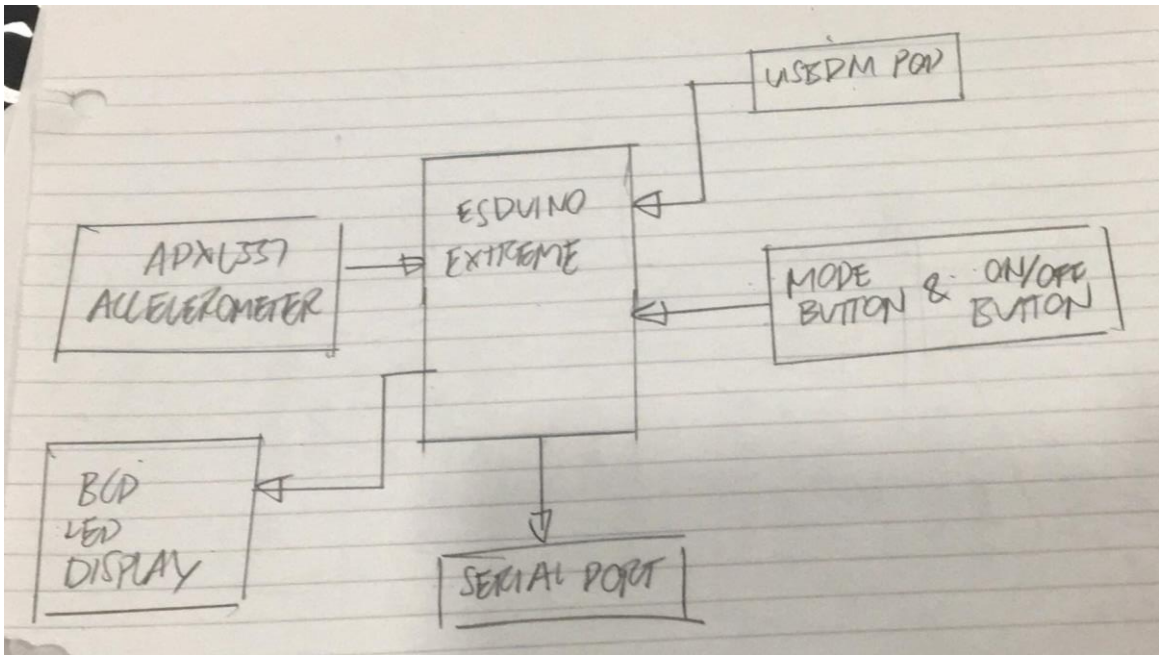
The algorithm below shows how this is done in MATLAB:



Full System Block Diagram

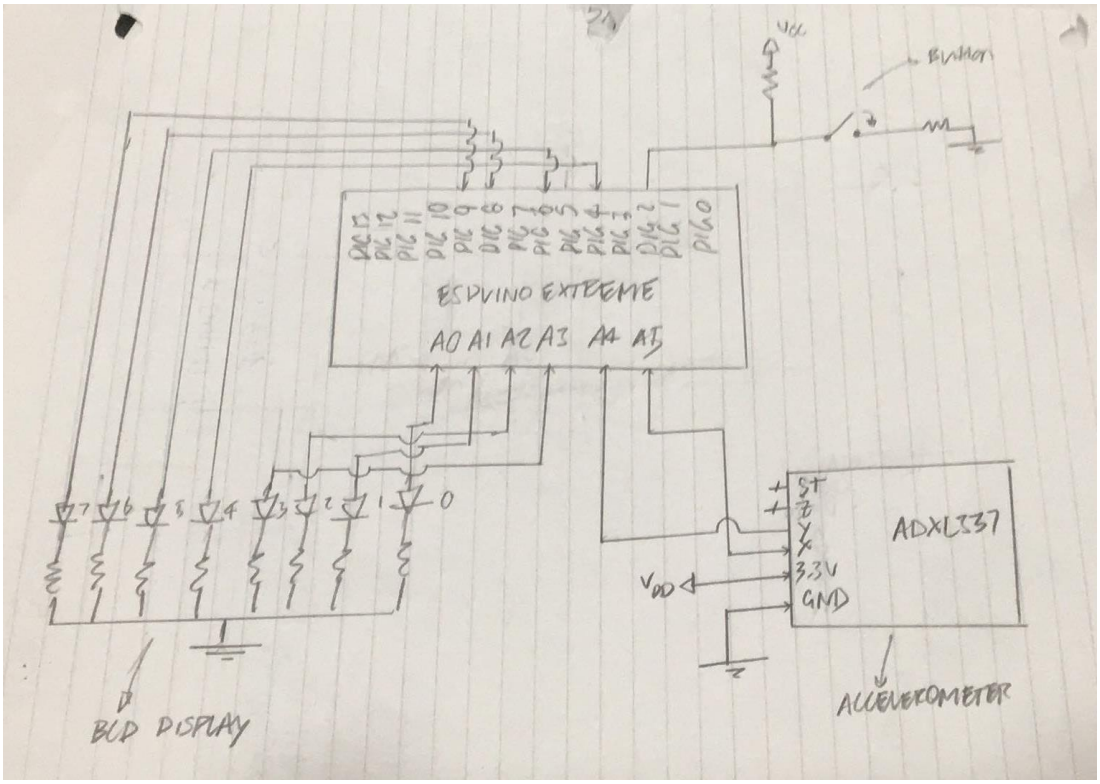
The full system block diagram displays the I/O peripherals for the whole system, and shows what the Esduino is connected to, whether is receiving or sending data. The arrows going towards the Esduino signify that data is being sent and the Esduino is processing it, while the arrows going out of the Esduino signify that the Esduino is sending data out towards that peripheral, like the BCD LEDs and serial port.

The full system block diagram is show below:



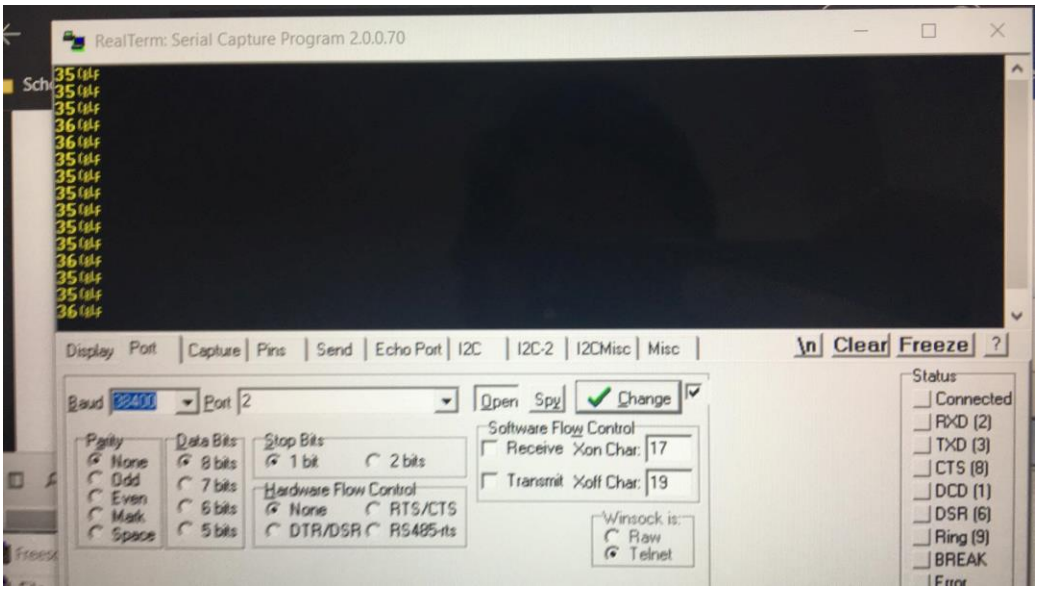
Full System Circuit Schematic

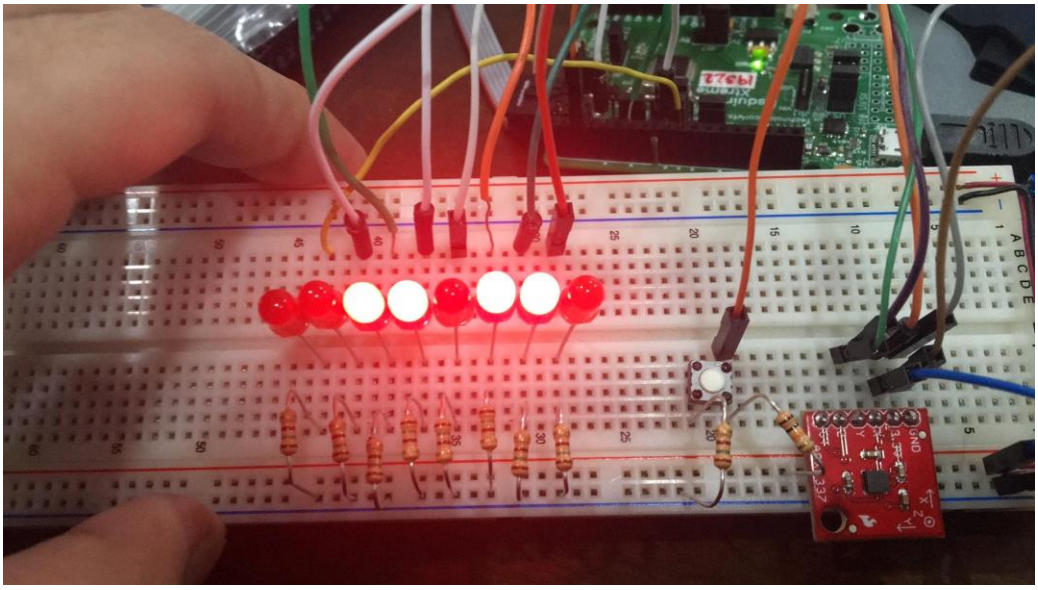
The full system schematic represents all the peripherals and components the system is connected to. These include things such as LEDs, resistors, power, ground and transducers like the accelerometer. The schematic is shown below:



Results and Testing

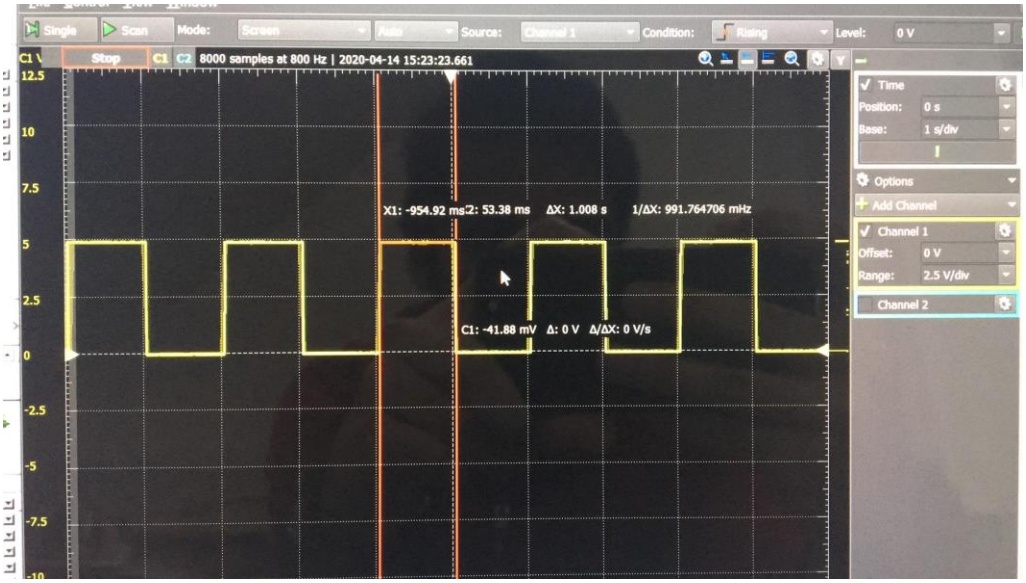
(b) Accelerometer





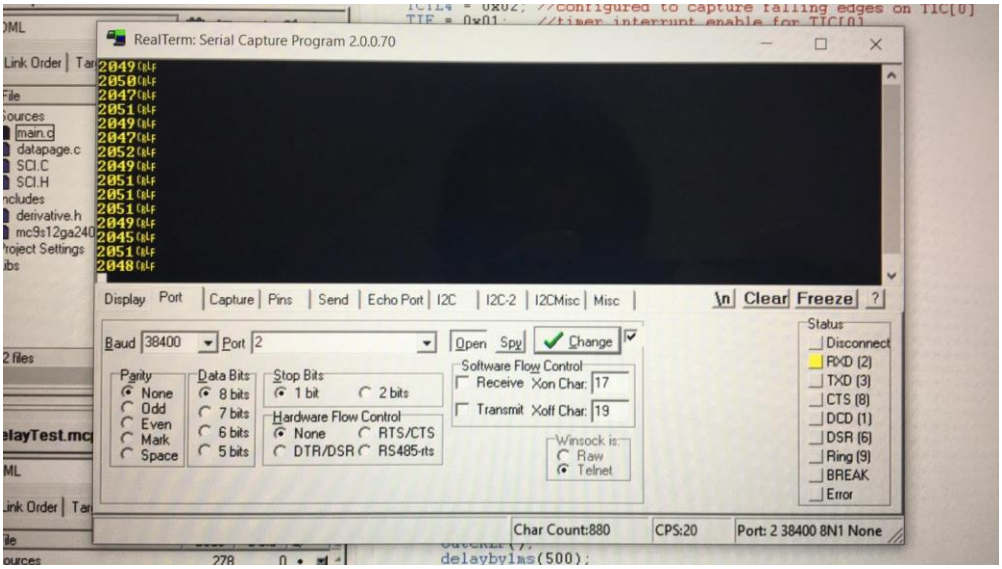
To verify the accelerometer the BCD and real term serial capture program was used. In the above pictures, the angle reading shows as 36, while also displaying the correct representation on the BCD display, 0011 0110.

(c) Bus Clock and Delay



To test the bus clock, the example code for week 8 was used by using the timer interrupt system. Knowing that one increment of the TCNT counter is 1 bus cycle, the delay that was needed to reach an interrupt flag was adding 8000 to TC0 every loop. This is because 1 bus cycle is 1/8MHz for this project, and by multiplying that by 8000 would give 1ms. The delay function was then used by turning the LED on and off for 1 second and using the AD2 to verify if the 1ms delay was correct. Because the use of interrupts was enabled in the main code, a different delay function was needed. The 'lazy delay' function from lab 4 was used to generate a 1ms delay, and trial and error was used using the AD2 to choose a value in the four loop that would give a 1 millisecond delay.

(d) ADC System

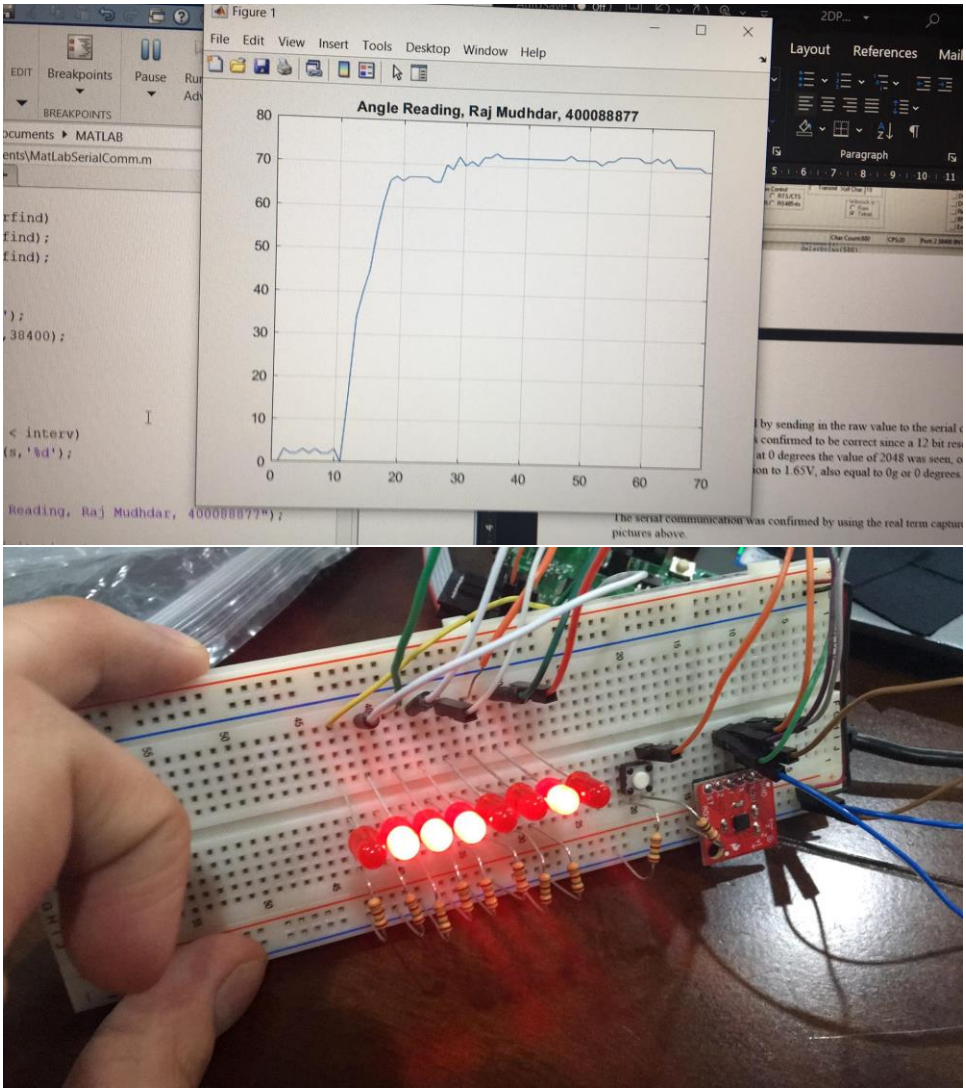


The ADC was simply tested by sending in the raw value to the serial capture by estimating the board at 0 degrees. This was confirmed to be correct since a 12 bit resolution generates 4095 data values between 0 and 3.3V, at 0 degrees the value of 2048 was seen, or otherwise 4095/2 = 2047.5, which should equation to 1.65V, also equal to 0g or 0 degrees.

(e) Serial Communication

The serial communication was confirmed by using the real term capture program, as seen in the pictures above.

(f) Whole System



To verify the whole system in working order, the MATLAB serial accessor and real time graphing code was used to verify that the accelerometer, BCD, and serial communication worked together as intended. The board was tilted upwards to display a high angle reading, and the resultant values were shown on the MATLAB live graph, around 70, as well as the BCD, which shows a value of 72.

Discussion

(1) When creating a new project, there is an option to provide floating point and double capabilities for the Esduino. At first, it was thought that basic calculations with decimal points could be performed due to errors being shown when doing so, but allowing the Esduino to support 32 bit numbers was more than enough to calculate accurate numbers that were computed from the raw data values read from the accelerometer.

Also, instead of using trigonometric functions on the Esduino, linear approximation was used to mitigate any unwanted errors that would pop up when calculating the angle via serial comm. The idea of sending the raw data value to MATLAB, then computing the arc sin of the acceleration, and sending the angle value back to the Esduino was theorized, but was not successful due to MATLAB not properly sending back the angle value to the serial port. It was then decided that all calculations would be done in 32-bit double format on the Esduino itself via linear approximation.

(2) The max quantization error is defined as $Q = delta(x)/2^{N+1}$, where delta(x) is the range that the analog signal is distributed on, and N is the number of quantization bits. Therefore, the error is $3.3\text{-}0/2^{12+1} = 0.04\%$.

(3) The baud rate is the maximum standard serial communication rate. Given a bus speed of 8MHz, the equation BD = BC/16*BR, where BD is the baud divisor for the SCIOBDL register, BC is the bus clock, and BR is the baud rate. Given that the baud rate needs to be a value such that there is low error in the divisor, a table was made to test all rates with the lowest possible error:

Baud Rate	**Baud Divisor**	**Error (%)**
9600	52.083 -> 52	0.159
19200	26.04 -> 26	0.154
38400	13.02 -> 13	0.154
57600	8.68 -> 9	3.56
115200	4.34 -> 4	8.5%

The max baud rate with the least error according to this table is 38400. Therefore, a serial communication rate of 38400 was chosen.

(4) The primary limitation speed is based on the how fast the system can sample values and send them to the serial port. This is relatively high at 8MHz, however the ADXL337 can only update its voltage at 500Hz, or 500 times per second (0.002s), due to the 0.01uF capacitors attached at the x, y, z pins. Therefore, the max speed at which the data can be update is 500Hz.

(5) Because the primary limitation speed is the 500Hz that the ADXL337 reproduces, the Nyquist rate is 1kHz. As such, the maximum frequency of the analog signal that can be effectively be reproduced is 500Hz.

(6) When input signals are reproduced digitally, an accurate representation for sharp waveforms depends on how fast data is sampled. In general they are not reproduced accurately, since a

square or sawtooth wave represents infinite frequency at its edges, the digital representation would be a slope and not a rising or falling edge.

Conclusion

Overall, the data acquisition system works as intended. It properly displays calculated angle values with some error due to linear approximation and displays live data on MATLAB. It was found that the Esduino cannot properly compute trigonometric functions, as well as MATLAB not sending serial data back, which is why 32-bit double values for on board calculations used in the end. The impact of using this method greatly increases the error in the final angle calculation, and because the ADXL337 did not accurately give a reading for 1g, the scaling factor of k = 2.34 was needed to represent the full range of angle values from 0 to 90, no matter how much error was generated by this.

Users Guide

This guide will demonstrate how to properly use and display the angle values via MATLAB. It will assume that the LEDs and peripherals are connected in proper fashion to the proper ports

1. Open the CodeWarrior IDE and open the project file named Code_mudharr
2. Make sure that the power is connected via the BDM pod and a USB cable is connected to the computer via the micro USB port on the Esduino
3. Run the code in the CodeWarrior IDE and test the system by tilting the ADXL337 in the positive x direction (up). If this does not work, reopen and rerun the code.
4. Next, open the MATLAB code named livePlotter_mudharr and start running it.
5. There are three different modes: Mode 0 measures the angle in the x direction, Mode 1 measures the angle in the y direction, and Mode 2 stops serial communication and data acquisition. To change between modes, press the button once and test the system by tilting the accelerometer. If the same mode is still in place, press the button again.
6. The BCD display and MATLAB live plot should now be showing the same angle values for the accelerometer.
7. To turn off the system, simply stop the MATLAB code and halt the CodeWarrior code to stop running on the Esduino.

Appendix

CodeWarrior IDE Code

```
/*2DP4 Project Code*/
/*Author: Raj Mudhar, mudharr, 400088877*/

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

/*funciton declerations*/
void delayby1ms(int);
void setClk(void);
void OutCRLF(void);
unsigned int mode;

void main(void) {

  /*setup variables, ports and bus speed*/
  unsigned short x;       //x analog reading for mode 0
  unsigned short y;       //y analog reading for mode 1
  double g = 0;           //acceleration to be calculated
  double conversion = 0;  //converting g to angle
  int angle = 0;          //the angle to be displayed
  setClk();               //set bus clock to 8MHz
  DDRJ = 0x01;            //set LED as output
  DDR01AD = 0b1111001111; //set LED outputs

  /*setup and enable ADC channel 5 (mode 0 - x) and channel 4 (mode 1 - y)*/
  ATDCTL1 = 0x4F; //12-bit resolution
  ATDCTL3 = 0x90; //right justified, two samples per sequence
  ATDCTL5 = 0x34; //continuous conversion on channel 4 and 5

  /*setup and configure timer input capture*/
  TSCR1 = 0x90; //timer system control register 1
                //timer enable and runs during WAI
  TSCR2 = 0x03; //timer system control register 2
                //timer prescaler select set to bus clock/8 = 1MHz
  TIOS = 0xFE;  //timer input capture/output compare
                //set TIC[0] and input
  PERT = 0x01;  //enable pull up resistor on TIC[0]
  TCTL3 = 0x00; //TCTLX configures which edge(s) to capture
  TCTL4 = 0x02; //configured to capture falling edges on TIC[0]
  TIE = 0x01;   //timer interrupt enable for TIC[0]
  EnableInterrupts;

  SCI_Init(38400); //initialize baudrate

  //infinte loop
  for(;;){
    //read channel 5 (x)
    if(mode == 0){
      x = ATDDR1;
      g = (double)x*6/4095-3;
```

```
      conversion = g*2.34*180/3.14;
      if(conversion <= 0)
        conversion = 0;
      angle = (int)conversion;
      SCI_OutUDec(angle);
      OutCRLF();
      delayby1ms(500); //MATLAB Samplig rate ~ 2 samples per second
    }
    //read channel 4 (y)
    else if(mode == 1){
      y = ATDDR0;
      g = (double)y*6/4095-3;
      conversion = g*2.34*180/3.14;
      if(conversion <= 0)
        conversion = 0;
      angle = (int)conversion;
      SCI_OutUDec(angle);
      OutCRLF();
      delayby1ms(500); //MATLAB sampling rate ~ 2 samples per second
    }
    //momentarily halt serial communications
    else if(mode == 2){
      delayby1ms(500); //MATLAB samping rate ~ 2 samples per second
    }
    PT01AD = (angle/10%10)<<6;  //output 10s column on BCD display
    PT01AD = PT01AD + angle%10; //output 1s column on BCD display
  }
}

/*interrupt service routine for TC0*/
static interrupt  VectorNumber_Vtimch0 void ISR_Vtimch0(void){
  unsigned int temp;
  //resets and increments modes
  //mode 0 reads x
  //mode 1 reads y
  //mode 2 halts serial communication
  mode++;
  if(mode == 3){
    mode = 0;
  }
  temp = TC0;
}

/*sets bus-clk to 8MHz*/
void setClk(void){
  CPMUCLKS = 0x80; //PLLSEL = 1
  CPMUOSC = 0x00; //OSCE = 0
  CPMUSYNR = 0x07; //fref = 2*1*(7+1)=16
  CPMUFLG = 0x08; //lock = 1, PLLCLK = VCOCLK/(POSTDIV+1)
  CPMUPOSTDIV = 0x00; //POSTDIV = 0, so BUSCLK = PLL/2 = 8MHz
}

/*delay function for 1ms*/
/*this was verified to be correct for 1s using the AD2*/
void delayby1ms(int time)
{
  unsigned int j,k;
```

```
  for(j=0;j<time;j++){
    for(k=0;k<1333;k++){
    }
  }
}

/*Output a CR,LF to SCI to move cursor to a new line*/
void OutCRLF(void){
  SCI_OutChar(CR);
  SCI_OutChar(LF);
}
```

For the SCI.C file, the code was modified to fit a baud rate of 38400.

BUS CLOCK TEST CODE

```
/*Bus Clock test code*/
/*Raj Mudhar, mudharr, 400088877*/

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

/*funciton declerations*/
void delayby1ms(int);
void setClk(void);

void main(void) {

  setClk();             //set bus clock to 8MHz
  DDRJ = 0x01;          //set LED as output

  //Flash LED for 1s
  while(1){
    PTJ = 0xFF;
    delayby1ms(1000);
    PTJ= 0x00;
    delayby1ms(1000);
  }

}

/*sets bus-clk to 8MHz*/
void setClk(void){
  CPMUCLKS = 0x80; //PLLSEL = 1
  CPMUOSC = 0x00; //OSCE = 0
  CPMUSYNR = 0x07; //fref = 2*1*(7+1)=16
  CPMUFLG = 0x08; //lock = 1, PLLCLK = VCOCLK/(POSTDIV+1)
  CPMUPOSTDIV = 0x00; //POSTDIV = 0, so BUSCLK = PLL/2 = 8MHz
}

/*delay function for 1ms*/
void delayby1ms(int time){
  int ix;
  /*enable timer and fast flag clear*/
  TSCR1 = 0x90;
  /*disable timer interrupt, set prescaler to 1*/
  TSCR2 = 0x00;

  /*enable OC0*/
  TIOS |= 0x01;

  /*1 bus cycle is 1/8E6, thus 1 bus cycle*8000 is 1ms*/
  for(ix = 0; ix < time; ix++){
    while(!(TFLG1_C0F));
      TC0 += 8000;
  }

  /*disable OC0*/
  TIOS &= ~0x01;
}
```

MATLAB LIVE PLOTTER CODE

```
%MATALB Live Plotter for 2DP4 Project
%By Raj Mudhar, mudharr, 400088877

clear;
%Delete empty serial ports to stop connection issues
if ~isempty(instrfind)
    fclose(instrfind);
    delete(instrfind);
end

%Create new serial object and open serial port
s = serial('COM2');
set(s,'BaudRate',38400); % set baud rate to 38400
fopen(s);

%time variables to sample for interv length
init_time = 1;
interv = 1000; %1000 samples at 2Hz
x = 0; %graph data

%scan port and make live graph
while (init_time < interv)
    out = fscanf(s,'%d');
    x = [x,out];
    plot(x);
    title("Angle Reading, Raj Mudhdar, 400088877");
    grid ON
    init_time = init_time+1;
    drawnow
end
```