# CPE 325: Intro to Embedded Computer System

**Lab09**

**Analog to Digital Convertors, Accelerometers, UART, Serial Communication.**

**Submitted by**: Nolan Anderson

**Date of Experiment**: 11/09/2020

**Report Deadline**: 11/09/2020

## Introduction

This lab covers interfacing with an ADC12 accelerometer to gather its data and output it through UART to the UAH Serial Communication app, or Analog to Digital conversion. It also covers how to configure the ADC12 accelerometer, baud rates, and data manipulation. Continually, learning the interfacing formulas from the voltage ratings of the accelerometer is very important to understand. If your functions are incorrect, then your code will not output the data correctly. Getting these formulas correct is imperative to good data.

## Theory

### Analog To Digital Converter

**Analog-to-Digital Converters**, (ADCs) allow micro-processor controlled circuits, Arduinos, Raspberry Pi, and other such digital logic circuits to communicate with the real world. In the real world, analogue signals have continuously changing values which come from various sources and sensors which can measure sound, light, temperature or movement, and many digital systems interact with their environment by measuring the analogue signals from such transducers.

### Temperature Sensor Interfacing

**The MSP430's ADC12 has an internal temperature sensor that creates an analog voltage proportional to its temperature. A sample transfer characteristic of the temperature sensor in a different MSP430 (namely MSP430FG4618) is shown in Figure 1. The output of the temperature sensor is connected to the input multiplexor channel 10 (INCHx=1010 which is true for MSP430F5529 as well). When using the temperature sensor, the sample time (the time ADC12 is looking at the analog signal) must be greater than 30 �s. From the transfer characteristic, we get that the temperature in degrees Celsius can be expressed as TEMPC = VTEMP−986 mV / 3.55 mV, whereVTEMP is the voltage from the temperature sensor (in milivolts). The transfer characteristic mentioned in Figure 1 is expressed in Volts. The ADC12 transfer characteristic gives the following equation: ADCResult = 4095 ·VTEMP/VREF, or VTEMP = VREF ·ADCResult/4095. This can easily be deduced using the relation ADCResult = (2^n − 1) ·VTEMP−VREF−.VREF+ − VREF−. By using the internal voltage generator VREF+=1,500 mV (1.5 V) and VREF- = 0V, we can derive temperature as follows: TEMPC = (ADCResult−2692)·423 / 4095.**

## Accelerometer Interfacing

**ADCXval** / 4095 steps X 3.3 V (input voltage) - 1.65 V (0g max voltage for X)/ 0.330 V/g (max sensitivity)

**ADCXval** / 4095 steps X 3.3 V (input voltage) - 1.65 V (0g max voltage for Y)/ 0.330 V/g (max Sensitivity)

**ADCZval**/ 4095 steps* 3.3 V (input voltage) - 1.8 V (0g max voltage for Z)/ 0.330 V/g (max sensitivity)

```
XDir = ((((ADCXval * 3.3)/4095)-1.65)/.330);
YDir = ((((ADCYval * 3.3)/4095)-1.65)/.330);
ZDir = ((((ADCZval * 3.3)/4095)-1.80)/.330);
```

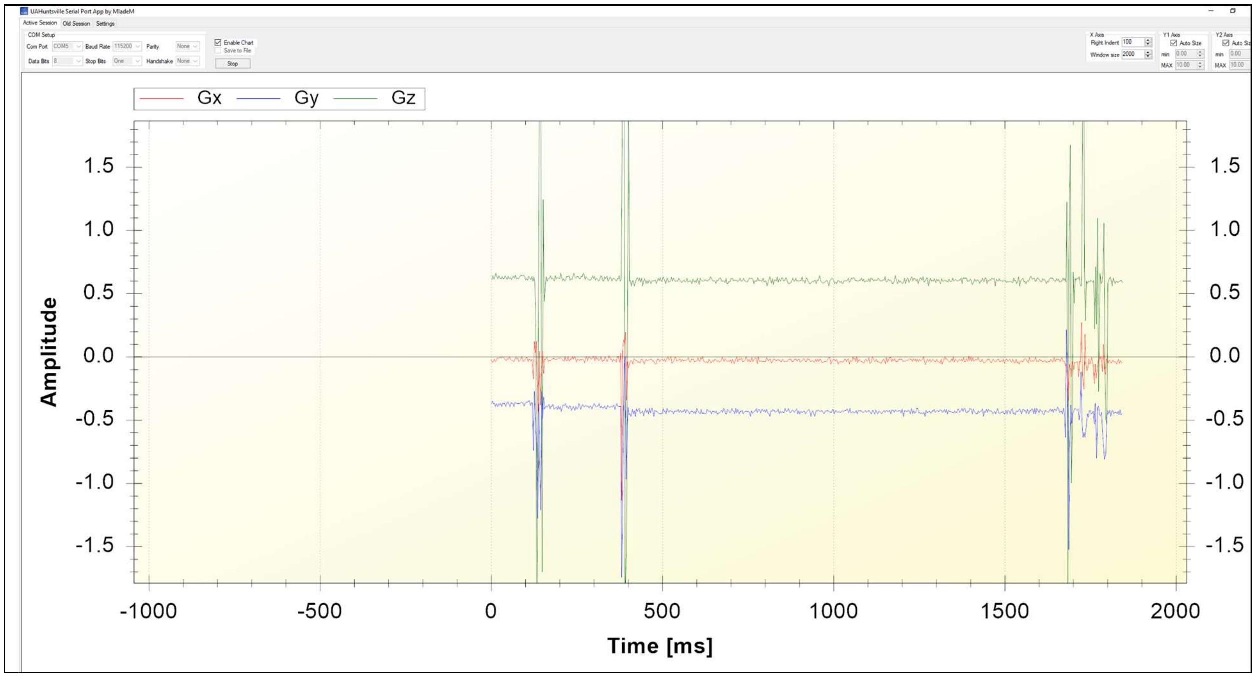
## Results & Observation

1 and 2 are covered in the demonstration.

### 3. Can you position your accelerometer to display:

The x and y components can get to 1 and the others to 0 easily, but the z component was more difficult to get a sustained value. I covered this in depth more on the demonstration but overall, yes for x and y, no for z.

### Results Screenshots/Pictures:



UAHuntsville Serial Port App by MladeM

COM Setup: Com Port COM5, Baud Rate 115200, Parity None, Data Bits 8, Stop Bits One, Handshake None

☑ Enable Chart ☐ Save to File

X Axis: Right Indent 100, Window size 2000

Y1 Axis: ☑ Auto Size, min 0.00, MAX 10.00

Y2 Axis: ☑ Auto Size, min 0.00, MAX 10.00

## Observations:

Being able to transmit data using UART is very useful for data analysis. The UAH Serial App is very good and useful for data analysis as well.

## Conclusion

UART communication is very sensitive especially when trying to use timers and interrupts. I had a lot of issues getting data and the LED's to blink correctly throughout the entire process. Specifically getting the LED's working was very difficult. When I would mess with the LEDs and hence messing with the watchdog interrupt, sometimes I would not get any data to be inputted. I later realized that I did not stop the watchdog timer at the start of the code. Overall, this was a very difficult and confusing lab but I see its importance.

https://drive.google.com/file/d/1FtC64A1lvE4JD2OfmH2-ZxBZBUgM2rzR/view?usp=sharinAppendix

## Appendix

```c
/*------------------------------------------------------------------------------
* Student:      Nolan Anderson
* Program:      Lab_9_1.c
* Date:         Nov 08, 2020
* Input:        ADC12 Acclerometer
* Output:       Data on UAH serial app and LED's when condition met
* Description:  This code takes in values from the ADC12 accelerometer and output
*               the data through UART to the UAH Serial app.
*-----------------------------------------------------------------------------*/
#include <msp430.h>
#include <math.h>
#define RED 0x01                                // Red LED Pin
#define GREEN 0x80                              // Green LED Pin
#define S1 ((P2IN&BIT1) == 0)                   // Switch 1 push defined.
#define S2 ((P1IN&BIT1) == 0)                   // Switch 2 push defined.

volatile long int ADCXval, ADCYval, ADCZval = -1;   // These are the values coming from the ADC12.
volatile float XDir, YDir, ZDir, netG = -1;         // Using XDir instead of gx.
volatile float g = 9.81;                            // Variable for gravity.

void ADC_setup(void);
void UART_putCharacter(char c);
void UART_setup(void);
void sendData(void);


void TimerA_setup(void)
{
    TA0CCTL0 = CCIE;                // Enabled interrupt
    TA0CCR0 = 3277;                 // 3277 / 32768 Hz = 0.1s, 10 samples/sec
    TA0CTL = TASSEL_1 + MC_1;       // ACLK, up mode
}


void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer
    __enable_interrupt();       // Enable interrupts globally
    SFRIE1 |= WDTIE;            // Enable WDT interrupt

    P1DIR |= RED;           // P1.0 is output direction for REDLED
    P1REN |= BIT1;          // Enable the pull-up resistor at P1.1
    P1OUT &= ~RED;          // LED is off at start

    P4DIR |= GREEN;         // P4.7 is output direction for GREENLED
    P2REN |= BIT1;          // Enable the pull-up resistor at P2.1
    P4OUT &= ~GREEN;        // LED is off at start

    // Configuring switch 1 interrupts
    P2IE  |= BIT1;
    P2IES |= BIT1;
    P2IFG &= ~BIT1;

    // Configuring switch 2 interrupts
    P1IE  |= BIT1;
    P1IES |= BIT1;
    P1IFG &= ~BIT1;

    ADC_setup();        // Setup ADC
    UART_setup();       // Setup UART Comms.
    TimerA_setup();     // Setup Timer A for 10 times a second


    while (1)           // Always run.
    {
        __bis_SR_register(LPM0_bits + GIE); // Enter LPM0
        sendData();             // Send out the data.
    }
}

void sendData(void)
{
    double sensitivity = pow(0.36, -1); // 1/0.360 (g/V)
    int i;
    XDir = ((((ADCXval * 3.3)/4095)-1.65)/.330);
    YDir = ((((ADCYval * 3.3)/4095)-1.65)/.330);
    ZDir = ((((ADCZval * 3.3)/4095)-1.80)/.330);

    // Use character pointers to send one byte at a time
    char *xpointer=(char *)&XDir;
    char *ypointer=(char *)&YDir;
    char *zpointer=(char *)&ZDir;

    UART_putCharacter(0x55);            // Send header

    // Send x one byte at a time
    for(i = 0; i < 4; i++)
    {
        UART_putCharacter(xpointer[i]);
    }

    // Send y one byte at a time
    for(i = 0; i < 4; i++)
    {
        UART_putCharacter(ypointer[i]);
    }

    // Send z one byte at a time
    for(i = 0; i < 4; i++)
    {
        UART_putCharacter(zpointer[i]);
    }

    netG = sqrtf((XDir*XDir) + (YDir * YDir) + (ZDir * ZDir));       // Use vector sum for calculation of netG.
    if(netG >= 2)
    {
        WDTCTL = WDT_MDLY_8;      // 5ms interval timer
        P1OUT |= RED;
        P4OUT &= ~GREEN;
    }
}

#pragma vector = ADC12_VECTOR
__interrupt void ADC12ISR(void)
{
    ADCXval = ADC12MEM0;                    // Move ADC12MEM0 (x) to ADCXval
    ADCYval = ADC12MEM1;                    // Move ADC12MEM1 (y) to ADCYval
    ADCZval = ADC12MEM2;                    // Move ADC12MEM2 (z) to ADCZval
    __bic_SR_register_on_exit(LPM0_bits);   // Exit LPM0
}


#pragma vector = TIMER0_A0_VECTOR
__interrupt void timerA_isr()
{
    ADC12CTL0 |= ADC12SC;
}
```

```
// Switch 2 Press
#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void)
{
    WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer
    P1OUT &= ~RED;              // Turn Red LED off
    P4OUT &= ~GREEN;            // Turn Green LED off
    P1IFG &= ~BIT1;
}

// Switch 1 Press
#pragma vector = PORT2_VECTOR
__interrupt void PORT2_ISR(void)
{
    WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer
    P1OUT &= ~RED;              // Turn Red LED off
    P4OUT &= ~GREEN;            // Turn Green LED off
    P2IFG &= ~BIT1;
}

// Watchdog Timer ISR.
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void)
{
    static int i = 0;
    if(i == 125)
    {
        P4OUT ^= GREEN;         // Toggle green LED
        P1OUT ^= RED;           // Toggle the red LED
        i = 0;
    }
    i++;
}


void ADC_setup(void)
{
    P6SEL = 0x07;                                       // Enable A/D channel inputs for x, y, and z (0000 0111)
    ADC12CTL0 = ADC12ON + ADC12MSC + ADC12SHT0_8;   // Turn on ADC12, extend sampling time to avoid overflow of
results
    ADC12CTL1 = ADC12SHP + ADC12CONSEQ_1;           // Use sampling timer, repeated sequence
    ADC12MCTL0 = ADC12INCH_0;                       // ref += AVcc, channel = A0
    ADC12MCTL1 = ADC12INCH_1;                       // ref += AVcc, channel = A1
    ADC12MCTL2 = ADC12INCH_2 + ADC12EOS;            // ref += AVcc, channel = A2, end sequence
    ADC12IE = 0x02;                                 // Enable ADC12IFG.1
    ADC12CTL0 |= ADC12ENC;                          // Enable conversions
}


void UART_putCharacter(char c)
{
    while (!(UCA0IFG&UCTXIFG)); // Wait for previous character to transmit
    UCA0TXBUF = c;              // Put character into tx buffer
}


void UART_setup(void)
{
    P3SEL |= BIT3 + BIT4;       // Set USCI_A0 RXD/TXD to receive/transmit data
    UCA0CTL1 |= UCSWRST;        // Set software reset during initialization
    UCA0CTL0 = 0;               // USCI_A0 control register
    UCA0CTL1 |= UCSSEL_2;       // Clock source SMCLK
    UCA0BR0 = 0x09;             // 1048576 Hz  / 115200 lower byte
    UCA0BR1 = 0x00;             // upper byte
    UCA0MCTL |= UCBRS0;         // Modulation (UCBRS0=0x01, UCOS16=0)
    UCA0CTL1 &= ~UCSWRST;       // Clear software reset to initialize USCI state machine
}
```