# CPE 325: Intro to Embedded Computer Systems

## Lab10

## Analog-to-Digital Converter

**Submitted by:** Michael Agnew

Date of Experiment: 4/1/2024

Report Deadline: 4/2/2024

Demonstration Deadline: 4/9/2024

## Introduction:

The aim of this lab was to learn about ADC and DAC in the realm of the MSP430 microcontroller. Analog-to-digital converters, or ADC bridge the gap that exists between the digital and analog worlds. This also holds true for Digital-to-Analog converters, or DAC. This lab includes the use of an accelerometer that transmits telemetry signals to the MSP430 which performs a gravity calculation on it before using it to control the LEDs located on the board based on angular deviation. The other part of this lab includes outputting signals to an oscilloscope in various ways at a set frequency.

## Theory:

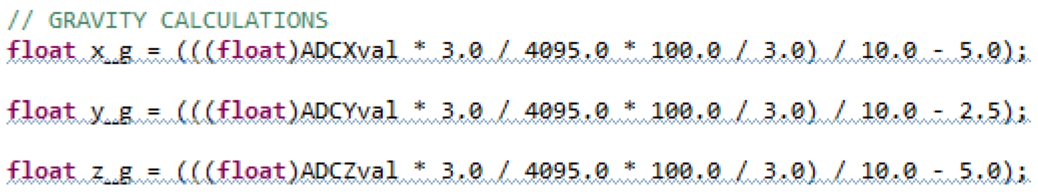
**ADC:** ADC components convert analog signals such as sound, light temperature or other real-world data into the digital landscape where it can be graphed, charted, used and more. These digital signals that this data is converted into are typically represented digitally as voltage. This voltage can then be interpreted in the context of whatever data is being collected. This component allows the trivial collection of real-world data where it can easily be stored and interpreted in the necessary manner.

**DAC:** The same, but inversely can be said for DAC systems, which convert digital signals and information into the real world proportions that are desired. These can include things such as heaters or audio playback systems. This works in the same way as ADC, where data will be coveted into a voltage signal which will then be emulated by the system outputting it. Overall, the use of these components are instrumental in daily life as we know it and are extremely important to learn about and use.

**Accelerometers:** Accelerometers, like the one that was used for this lab, are used for retrieving telemetry data such as position, orientation, velocity, and sending it back to whatever program is taking it in. This telemetry data is then taken and used to figure out the position or speed of the accelerometer for whatever purposes it might serve. In the case of this lab, the accelerometer was used in order to figure out the angular deviation of it at different positions made by the user, which were then calculated by the program which activated the different LEDs based on the position of the accelerometer.

## Problem 01:

The first problem of this lab was implementing a C program that interfaces with a 3-dimensional accelerometer. The telemetry data that the accelerometer passed back to the program needed to be calculated in terms of gravity before being properly utilized. Demo code was provided for this assignment, but only worked for two dimensions, so in order to solve this problem, a third axis was implemented in order to have a three dimensional setup with a z-axis. The gravity calculation was also done in the following way:



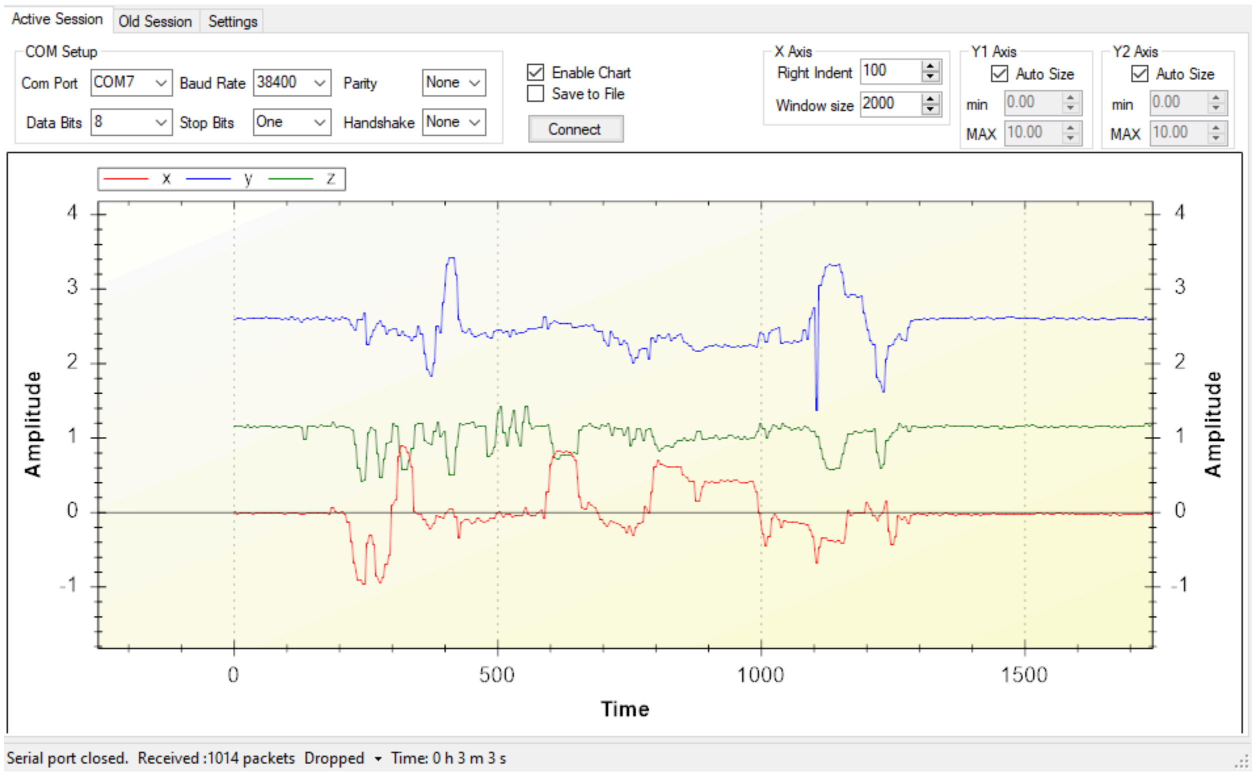
```
// GRAVITY CALCULATIONS
float x_g = (((float)ADCXval * 3.0 / 4095.0 * 100.0 / 3.0) / 10.0 - 5.0);

float y_g = (((float)ADCYval * 3.0 / 4095.0 * 100.0 / 3.0) / 10.0 - 2.5);

float z_g = (((float)ADCZval * 3.0 / 4095.0 * 100.0 / 3.0) / 10.0 - 5.0);
```

Figure 1. Gravity Calculations in CCS.

These values are multiplied by 3 because the voltage value is 3, and are then divided by 4095 because the ADC component is done in 12 bits. The multiplication by 100 then scales up the calculation in the proper way. The subtraction at the end is to negate any sort of bias that might occur in the sensor readings. The output for this program in the serial app is shown here:



COM Setup
Com Port COM7
Baud Rate 38400
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

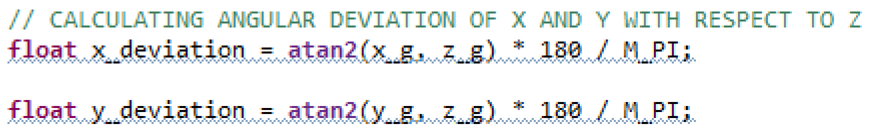
Y2 Axis
☑ Auto Size
min 0.00
MAX 10.00

Serial port closed. Received :1014 packets Dropped Time: 0 h 3 m 3 s

Figure 2. Serial App Output for Problem 01. Red is X, Blue is Y, and Green is Z.

## Problem 02:

The second problem in this lab was to make a bubble level out of the accelerometer program by calculating the angular deviation and turning on the LEDs accordingly to illustrate the current angle of the accelerometer. The way that this would be done, is when the angular deviation of the accelerometer was above 15 degrees, the red LED would come on, and when the angular deviation was lower than 15 degrees, the green LED would be turned on. When the angular deviation was between -15 and 15 degrees, both of the LEDs would remain off. The angular deviation was calculated by taking the arctangent of x and y with respect to the z axis, multiplied by 180 for degrees and then divided by pi. This calculation is shown here:



```
// CALCULATING ANGULAR DEVIATION OF X AND Y WITH RESPECT TO Z
float x_deviation = atan2(x_g, z_g) * 180 / M_PI;

float y_deviation = atan2(y_g, z_g) * 180 / M_PI;
```

Figure 3. Angular Deviation Formula Used in Program.

The deliverables for this project also required a flowchart be made for problem 2, this can be found below:

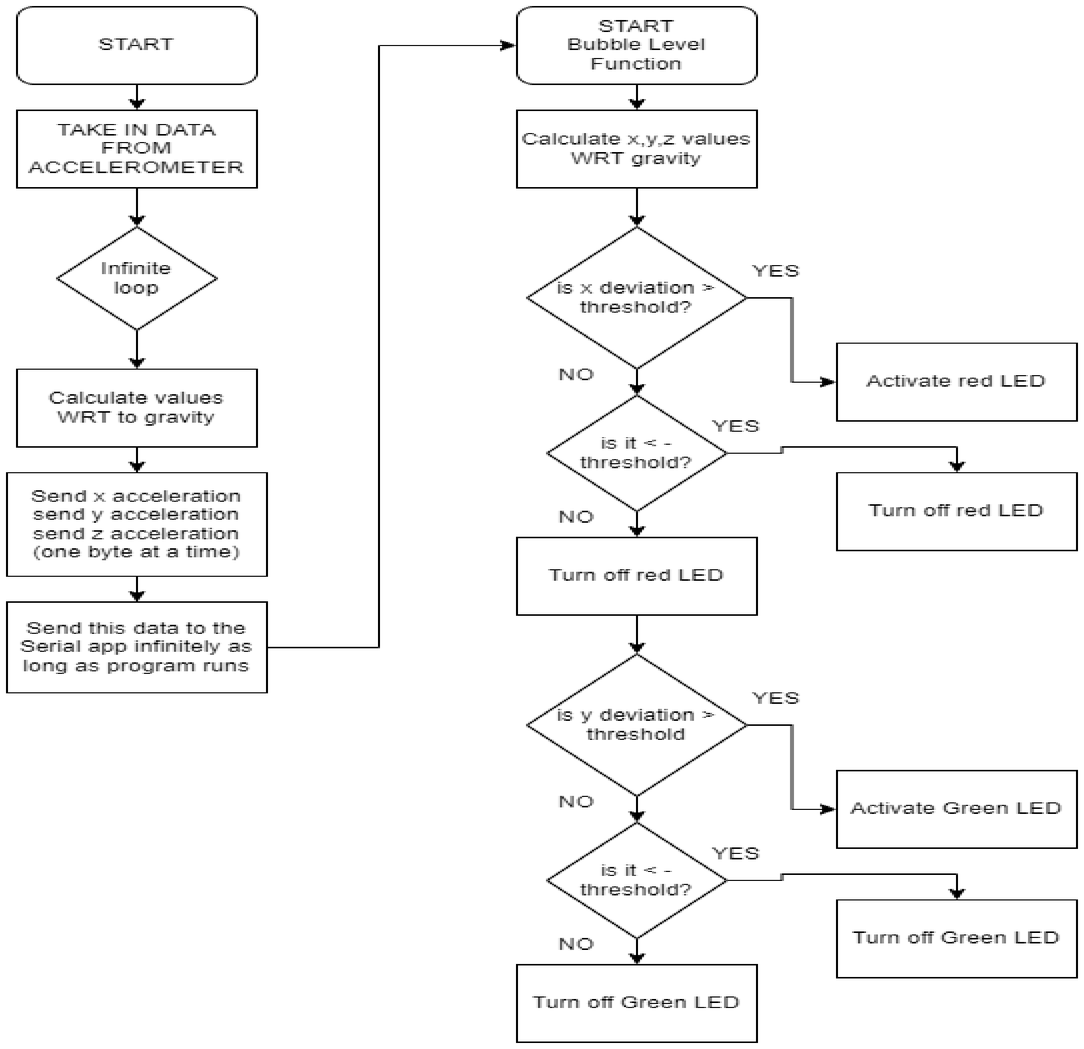

START
TAKE IN DATA FROM ACCELEROMETER
Infinite loop
Calculate values WRT to gravity
Send x acceleration send y acceleration send z acceleration (one byte at a time)
Send this data to the Serial app infinitely as long as program runs
START Bubble Level Function
Calculate x,y,z values WRT gravity
is x deviation > threshold?
YES
Activate red LED
NO
is it < - threshold?
YES
Turn off red LED
NO
Turn off red LED
is y deviation > threshold
YES
Activate Green LED
NO
is it < - threshold?
YES
Turn off Green LED
NO
Turn off Green LED

Figure 4. Flowchart for Problem 2.

## Questions to Answer:

### 1. Explain why you chose the sampling rate you used for the accelerometer.

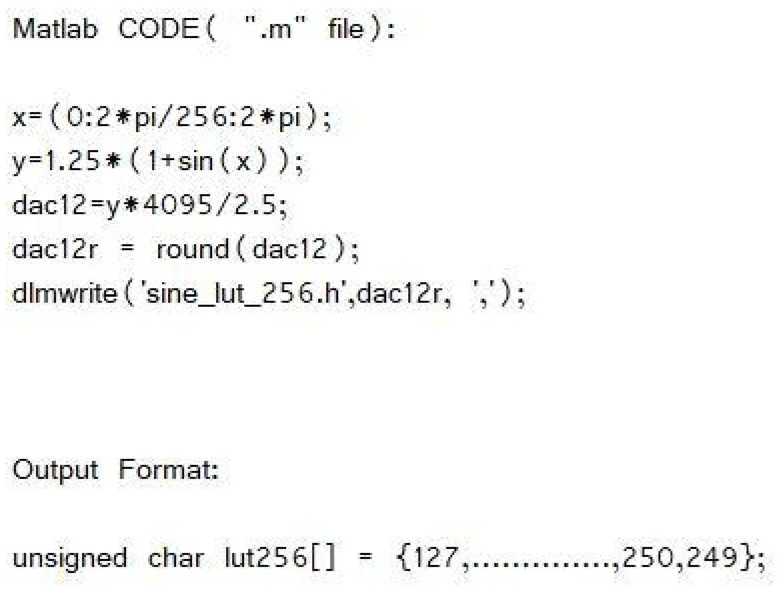
ANS: The sampling rate that was chosen for the accelerometer was 1. This was chosen because it was the rate that allowed the serial app to output the accelerometer data at a good rate that properly illustrated the data being processed.

### 2. Explain how you chose a threshold value for the bubble level.

ANS: A threshold value of 15 was chosen for the bubble level because the assignment required that the level sit between the values of -15 and 15. So, the logical choice for this program was to set the threshold value to 15 in order to properly activate the LEDs when the angular deviation was at the proper value.

## Problem 03:

The third part of this lab had to do with outputting different waveform graphs to an oscilloscope at a set frequency. The MSP430 is not entirely capable of doing complex trigonometry, so a lookup table was generated first by using a MATLAB script. This is shown here:



Matlab CODE( ".m" file):

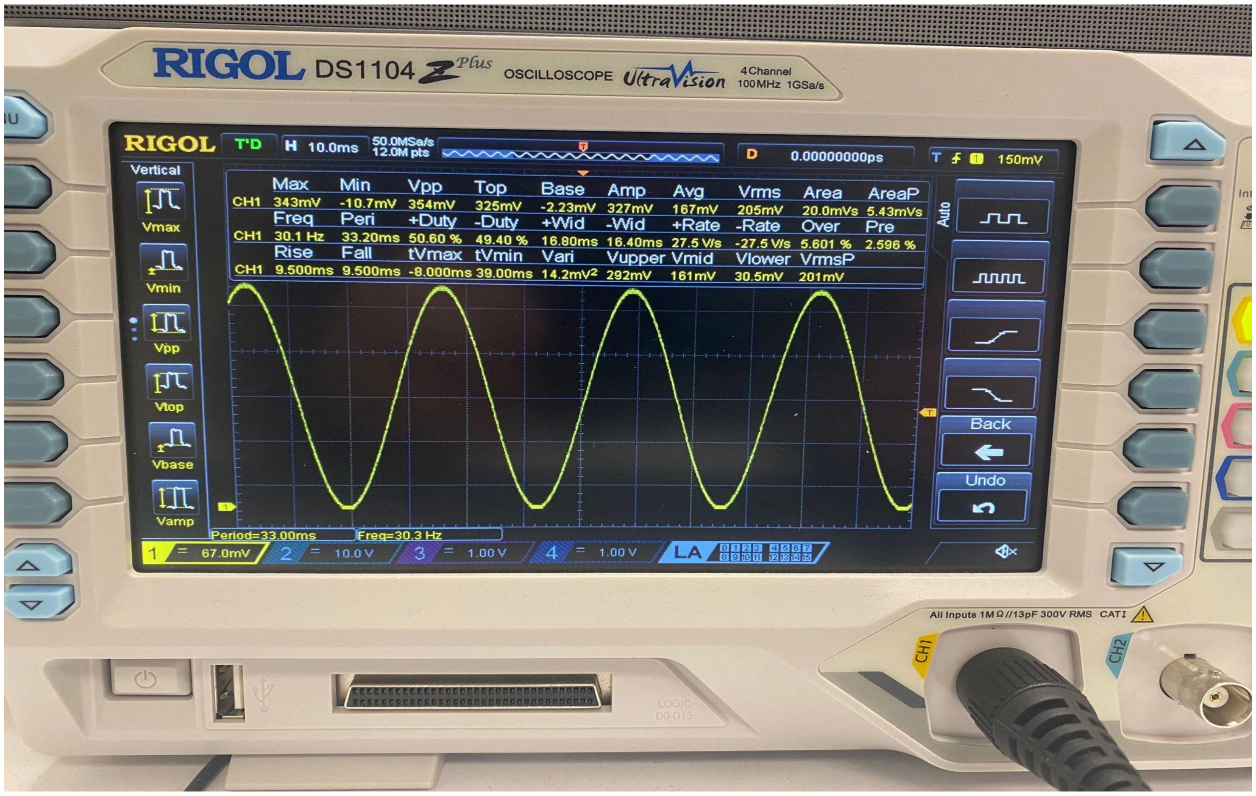
```
x=(0:2*pi/256:2*pi);
y=1.25*(1+sin(x));
dac12=y*4095/2.5;
dac12r = round(dac12);
dlmwrite('sine_lut_256.h',dac12r, ',');
```

Output Format:

```
unsigned char lut256[] = {127,.............,250,249};
```

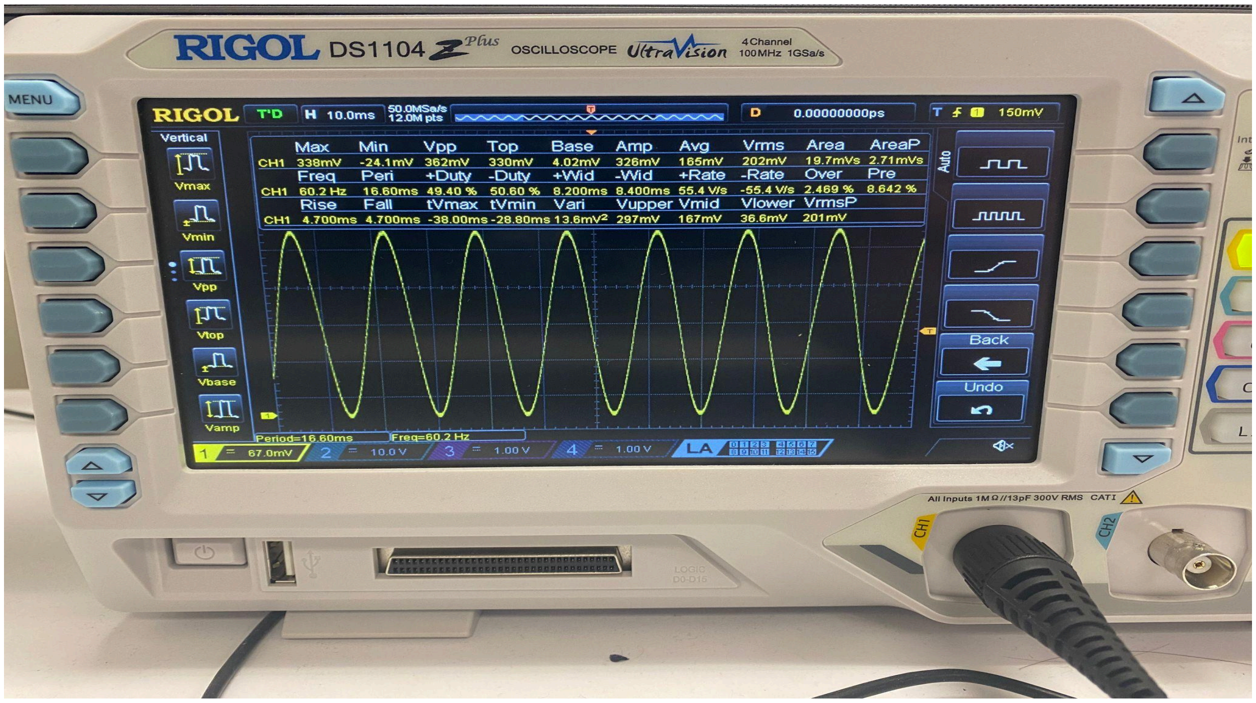
Figure 5. Matlab Script for Generating Lookup Table.

The first of these waveforms was meant to be a sine wave at 30hz. When switch 1 is held, a triangular waveform is generated on the fly at the same 30hz. When switch 2 is held, the frequency of the sine wave is doubled to 60 hz. Below is the output of these waveforms on the oscilloscope:



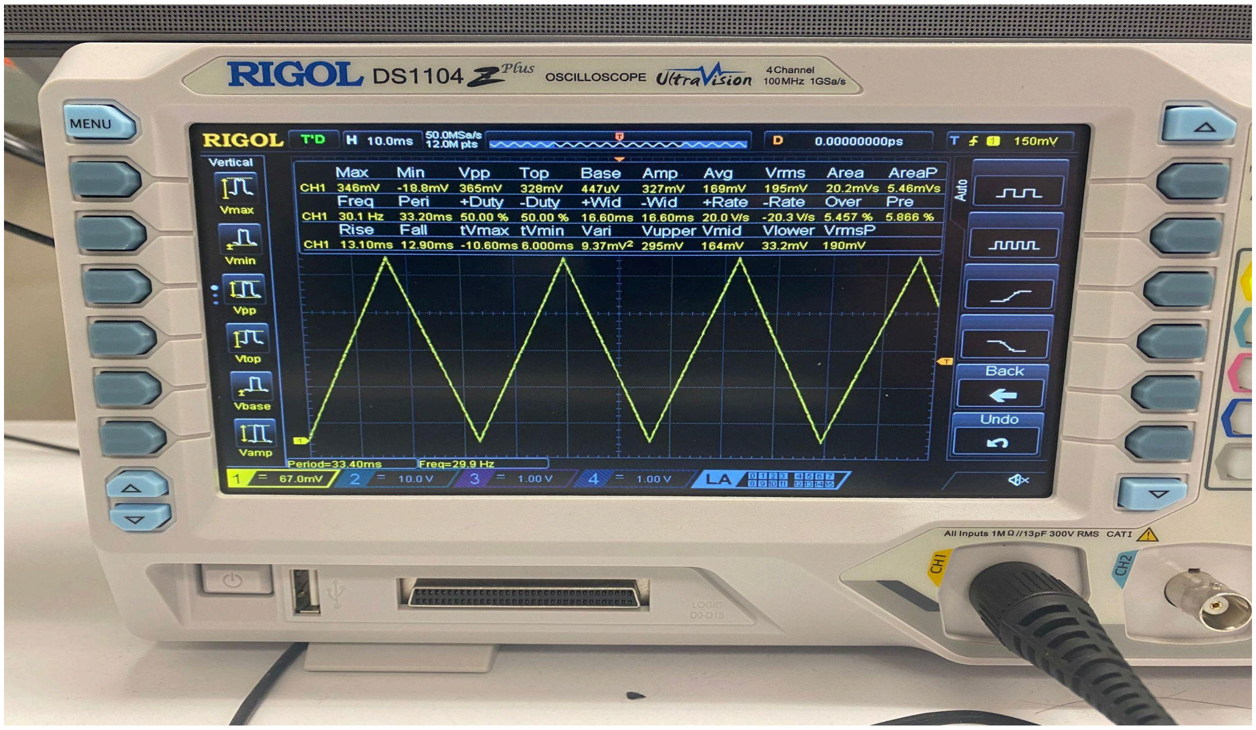
```
     Max      Min      Vpp      Top      Base     Amp      Avg      Vrms     Area     AreaP
CH1  343mV    -10.7mV  354mV    325mV    -2.23mV  327mV    167mV    205mV    20.0mVs  5.43mVs
     Freq     Peri     +Duty    -Duty    +Wid     -Wid     +Rate    -Rate    Over     Pre
CH1  30.1 Hz  33.20ms  50.60 %  49.40 %  16.80ms  16.40ms  27.5 V/s -27.5 V/s 5.601 % 2.596 %
     Rise     Fall     tVmax    tVmin    Vari     Vupper   Vmid     Vlower   VrmsP
CH1  9.500ms  9.500ms  -8.000ms 39.00ms  14.2mV²  292mV    161mV    30.5mV   201mV
```

Figure 6. Sine Waveform at 30Hz.



```
     Max     Min      Vpp      Top      Base     Amp      Avg      Vrms      Area      AreaP
CH1  338mV   -24.1mV  362mV    330mV    4.02mV   326mV    165mV    202mV     19.7mVs   2.71mVs
     Freq    Peri     +Duty    -Duty    +Wid     -Wid     +Rate    -Rate     Over      Pre
CH1  60.2 Hz 16.60ms  49.40 %  50.60 %  8.200ms  8.400ms  55.4 V/s -55.4 V/s 2.469 %   8.642 %
     Rise    Fall     tVmax    tVmin    Vari     Vupper   Vmid     Vlower    VrmsP
CH1  4.700ms 4.700ms  -38.00ms -28.80ms 13.6mV²  297mV    167mV    36.6mV    201mV
```

Figure 7. Sine Waveform with Doubled Frequency (60Hz).



```
     Max     Min      Vpp      Top      Base     Amp      Avg      Vrms      Area      AreaP
CH1  346mV   -18.8mV  365mV    328mV    447uV    327mV    169mV    195mV     20.2mVs   5.46mVs
     Freq    Peri     +Duty    -Duty    +Wid     -Wid     +Rate    -Rate     Over      Pre
CH1  30.1 Hz 33.20ms  50.00 %  50.00 %  16.60ms  16.60ms  20.0 V/s -20.3 V/s 5.457 %   5.866 %
     Rise    Fall     tVmax    tVmin    Vari     Vupper   Vmid     Vlower    VrmsP
CH1  13.10ms 12.90ms  -10.60ms 6.000ms  9.37mV²  295mV    164mV    33.2mV    190mV
```

Figure 8. Triangular Waveform at 30Hz.

## Conclusion:

Thankfully, no issues were encountered over the course of this lab. There was a lot to go over and a lot to learn and implement with this lab, especially since there were three parts instead of the normal two. Overall however, everything that was meant to be completed was done so and in the proper amount of time. The accelerometer was able to be instantiated with a C program, and the 3-dimensional setup in the serial app output the proper results. The bubble level for part 2 was also properly implemented with the LEDs turning on when the angular deviation is at the proper values. Finally, for part 3, the correct waveforms at the correct frequency were output with the switches being able to make the required changes as described in the assignment. Overall, ADC and DAC, along with accelerometers are very important components and instruments that one must learn to use as the real-world applications are extremely important for any aspiring engineer.

## Appendix:

### Problem 01-02:

```
/*---------------------------------------------------------------------------
 * File:          Lab10P1P2.c
 * Function:      Accelerometer Telemetry Processor
 * Description: This code performs calculations on data passed from accelerometer and turns
on LEDs based on angular deviation.
 * Input:         None
 * Output:        LEDs on the circuit board and waveform in serial app.
 * Author(s):   Michael Agnew, ma0133@uah.edu
 * Date:          April 1, 2024
 *---------------------------------------------------------------------------*/
```

```
#include <msp430.h>
#include <math.h>
#include <stdint.h>


#define samplerate 10 // sample rate
#define threshhold 15 // threshold for angular deviation
#define grav 10 // gravity


volatile uint16_t ADCXval, ADCYval, ADCZval;

void TimerA_setup(void)
{
        TA0CCR0 = 3277 / samplerate; // Calculate CCR0 for desired sampling rate
        TA0CTL = TASSEL_1 + MC_1; // ACLK, up mode
        TA0CCTL0 = CCIE; // Enabled interrupt
}

void ADC_setup(void)
{
        P6DIR &= ~(BIT3 + BIT4 + BIT5); // Configure P6.3, P6.4, P6.5 as input pins
        P6SEL |= BIT3 + BIT4 + BIT5; // Configure P6.3, P6.4, P6.5 as analog pins

        ADC12CTL0 = ADC12ON + ADC12SHT0_8 + ADC12MSC; // Configure ADC
converter
        ADC12CTL1 = ADC12SHP + ADC12CONSEQ_1; // Use sample timer, single
sequence
        ADC12MCTL0 = ADC12INCH_3; // ADC A3 pin - X-axis
        ADC12MCTL1 = ADC12INCH_4; // ADC A4 pin - Y-axis
        ADC12MCTL2 = ADC12INCH_5 + ADC12EOS; // ADC A5 pin - Z-axis, End of
Sequence
        ADC12IE |= ADC12IE2; // Enable ADC12IFG.2
        ADC12CTL0 |= ADC12ENC; // Enable conversions
}

void UART_putCharacter(uint8_t c)
{
        while (!(UCA0IFG & UCTXIFG)); // Wait for previous character to be sent
        UCA0TXBUF = c; // Send byte to the buffer for transmitting
}

void UART_setup(void)
{
        P3SEL |= BIT3 + BIT4;                   // Set up Rx and Tx bits
```

```
	UCA0CTL0 = 0;			// Set up default RS-232 protocol
	UCA0CTL1 |= BIT0 + UCSSEL_2;		// Disable device, set clock
	UCA0BR0 = 27;			// 1048576 Hz / 38400
	UCA0BR1 = 0;
	UCA0MCTL = 0x94;
	UCA0CTL1 &= ~BIT0;			// Start UART device
}

void sendData(void)
{
	int i;

	// grav CALCULATIONS

	float yGrav = (((float)ADCYval*3.0/4095.0*100.0/3.0)/10.0-2.5);

	float zGrav = (((float)ADCZval*3.0/4095.0*100.0/3.0)/10.0-5.0);

	float xGrav = (((float)ADCXval*3.0/4095.0*100.0/3.0)/10.0-5.0);



	UART_putCharacter(0x55);

	uint8_t *y_pointer = (uint8_t *)&yGrav;
	uint8_t *x_pointer = (uint8_t *)&xGrav;
	uint8_t *z_pointer = (uint8_t *)&zGrav;

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

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

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

void sendDataDebug(float data)
{
        int i;
        uint8_t *data_pointer = (uint8_t *)&data;
        for (i = 0; i < 4; i++)
        {
        UART_putCharacter(data_pointer[i]);
        }
}

void bubbleLevel(void) {
        float xGrav = (((float)ADCXval*3.0/4095.0*100.0/3.0)/10.0-5.0);
        float yGrav = (((float)ADCYval*3.0/4095.0*100.0/3.0)/10.0-5.0);
        float zGrav = (((float)ADCZval*3.0/4095.0*100.0/3.0)/10.0-5.0);


        // CALCULATING ANGULAR DEVIATION OF X AND Y WITH RESPECT TO Z
        float x_deviation = atan2(xGrav, zGrav) * 180 / M_PI;

        float y_deviation = atan2(yGrav, zGrav) * 180 / M_PI;

        // X AXIS LED
        if (x_deviation > threshhold)
        {
        P1OUT |= BIT0; // activate red LED
        } else if (x_deviation < -threshhold)
        {
        P1OUT &= ~BIT0;
        } else {
        P1OUT &= ~BIT0;
        }

        // Y AXIS LED
        if (y_deviation > threshhold)
        {
        P4OUT |= BIT7; // turn on green LED
        }
        else if (y_deviation < -threshhold)
        {
        P4OUT &= ~BIT7;
        }
        else
        {
        P4OUT &= ~BIT7;
        }
}
```

```
void main(void)
{
        WDTCTL = WDTPW + WDTHOLD; // Stop WDT
        TimerA_setup(); // Setup timer to send ADC data
        ADC_setup(); // Setup ADC
        UART_setup(); // Setup UART for RS-232
        P1DIR |= BIT0 + BIT1; // Set P1.0 and P1.1 as output for LEDs
        P4DIR |= BIT7;
        __enable_interrupt(); // Enable global interrupt

        while (1)
        {
        ADC12CTL0 |= ADC12SC; // Start conversions
        __bis_SR_register(LPM0_bits + GIE); // Enter LPM0
        sendData(); // Send data to serial app
        bubbleLevel(); // Check bubble level and control LEDs
        }
}

#pragma vector = ADC12_VECTOR
__interrupt void ADC12ISR(void)
{
        ADCXval = ADC12MEM0; // X-axis
        ADCYval = ADC12MEM1; // Y-axis
        ADCZval = ADC12MEM2; // Z-axis
        __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void timerA_isr()
{
        __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}
```

## Problem 03:

```
/*------------------------------------------------------------------------------
 * File:          Lab10Part3.c
 * Function:    Generates Waveforms on Oscilloscope
 * Description: Generates Sine wave and triangle wave at 30-60hz based on switch presses.
 * Input:       Switches
 * Output:      Oscilloscope waveforms
 * Author(s):   Michael Agnew, ma0133@uah.edu
 * Date:        April 1, 2024
 *-------------------------------------------------------------------------*/

#include <msp430.h>
#include "sine_lut_256.h"


#define S1 P2IN&BIT1 // Switch 1
#define S2 P1IN&BIT1 // Switch 2


unsigned char index = 0;

unsigned int value= 127;

unsigned int i = 0;

volatile int up = 1;
int main(void)
{
        WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer

        // SWITCH 1 SETUP
        P2DIR &= ~BIT1; // Set P2.1 as input for S1 input
        P2REN |= BIT1; // Enable the pull-up resistor at P2.1
        P2OUT |= BIT1; // Required for proper IO

        // SWITCH 2 SETUP
        P1DIR &= ~BIT1; // Set P1.1 as input for S2 input
        P1REN |= BIT1; // Enable the pull-up resistor at P1.1
        P1OUT |= BIT1; // Required for proper IO



        // Timer B Configuration
        TB0CCTL0 = CCIE; // Timer Count Triggers Interrupt
        TB0CTL = TBSSEL_2 + MC_1; // SMCLK is source, and set to up mode
        TB0CCR0 = 135; // 30hz

        P3DIR = 0xFF; // Enable All pins on Port 3 as outputs.
        P3OUT = 0;
        __bis_SR_register(LPM0_bits + GIE);


        return 0;
}



#pragma vector=TIMERB0_VECTOR
__interrupt void timerISR2(void)
{
        // Check if S2 is pressed
        if ((S2) == 0)
        {
        TB0CCR0 = 67; // 60hz

        }
        else
        {
        TB0CCR0 = 135; // 30hz
        }

        if ((S1) == 0)
        {
        TB0CCR0 = 205; // for triangle
        // When S1 is held
        if (up)
        {
        value += 3;

        if (value == 255) // once value reaches max
        {
                up = 0; // go down
        }
        }
        else
        {
        value -= 3;

        if (value == 0) // once value reaches min
        {
                up = 1; // go up again
        }

        }
        P3OUT = value; // output value to p3
        }
        else
        {
        // If S1 is not pressed, continue using the sine lookup table
        index += 1;
        P3OUT = lut256[index % 256]; // Ensure index wraps around
        }
}
```

## LUT Lookup Table Script:

```
Matlab CODE(".m" file):

x=(0:2*pi/256:2*pi);
y=1.25*(1+sin(x));
dac12=y*4095/2.5;
dac12r = round(dac12);
dlmwrite('sine_lut_256.h',dac12r, ',');
```