CPE 325: Intro to Embedded Computer System

Lab 10

Analog to Digital Converter, Digital to Analog Converter

Submitted by: Anshika Sinha

Date of Experiment: 03/19/2025

Report Deadline: 03/25/2025

Demonstration Deadline: 03/31/2025

Theory

Topic 1: Accelerometers

a) An accelerometer is a sensor used to measure acceleration forces in one or more directions.

b) The ADXL335 is a small, low-power, 3-axis accelerometer with analog voltage outputs for direct interfacing with ADCs. It has a minimum full-scale range of ±3 g, making it suitable for both static and dynamic acceleration measurements.

c) The ADXL335 offers customizable bandwidth through external capacitors connected to the XOUT, YOUT, and ZOUT pins, with bandwidth ranges of 0.5 Hz to 1600 Hz for the X and Y axes and 0.5 Hz to 550 Hz for the Z axis.

d) Internally, it uses a polysilicon surface-micromachined structure suspended by springs over a silicon wafer. When subjected to acceleration, the moving mass deflects, altering the capacitance between fixed and movable plates. This change is detected using phase-sensitive demodulation, generating an analog voltage proportional to the acceleration.

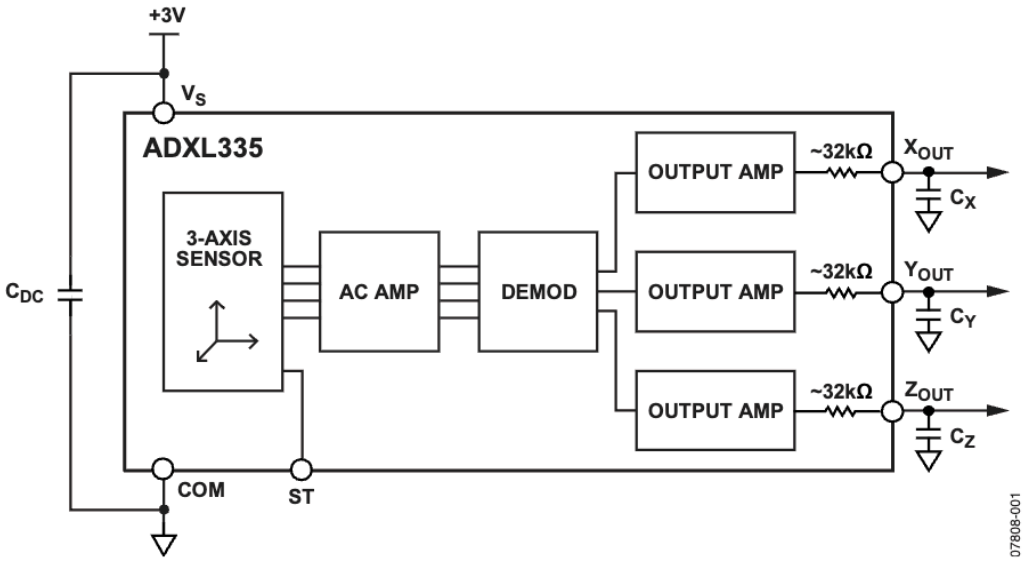

Figure 01: Block Diagram of the ADXL335 Accelerometer

Topic 2: ADC and DAC

a) Analog to Digital Converters(ADCs) convert analog signals (e.g., voltage) into digital values for processing by microcontrollers or computers.
   a. The MSP430F5529 includes 16 configurable input channels, with 12 mapped to external analog pins and the rest to internal voltages and an on-chip temperature sensor.
   b. The ADC12 has a 12-bit resolution, distinguishing between 4096 voltage levels (0 to 4095).

c. The ADC uses a timer trigger for periodic sampling. The sampling frequency should be at least twice the highest harmonic of the input signal to avoid aliasing.

b) Digital to Analog Converters(DACs) convert digital values into corresponding analog signals, used for tasks like generating audio or controlling motors.
   a. The resolution, measured in bits, determines the number of discrete voltage steps the DAC can produce.
   b. The DAC12 supports both 8-bit and 12-bit resolution, with 12-bit mode offering finer voltage steps for smoother output signals.
   c. The DAC12 does not have an internal reference voltage, so it is usually configured to use the ADC12 reference voltage.
   d. The DAC12 does not have an internal timer, and requires Timer_A to periodically trigger outputs.
   e. A lookup table (LUT) is often used to store waveform values, which are sequentially passed to the DAC12 for continuous signal output. Increasing the number of samples per period improves the waveform resolution but requires a faster timer refresh rate to maintain the desired frequency.
   f. DACs are used for waveform generation, analog signal synthesis, and motor control systems.

Results & Observation

Program 1:

Program Description:

This C program interfaces the ADXL335, a 3-dimensional accelerometer, with the MSP430F5529. It reads acceleration values from the X, Y, and Z axes, converts them into digital values using the ADC12 module, and sends the data over UART to the UAH Serial App. The ADC12 is configured to read analog signals from the accelerometer's three axes, connected to pins P6.3 (X), P6.4 (Y), and P6.5 (Z). The ADC values are converted to acceleration in g units using the formula:

$$Acceleration\ (g) = (\frac{3.0 \times ADC\ value}{4095} - 1.5) \div 0.3$$

The TimerA module generates periodic interrupts 10 times per second using ACLK in up mode, triggering the ADC to capture new samples. The ADC12ISR captures the sampled data and stores it in ADC12MEM0, ADC12MEM1, and ADC12MEM2, while the TimerA ISR calculates the acceleration magnitude, controls the LEDs, and transmits the data over UART.

Program Output:

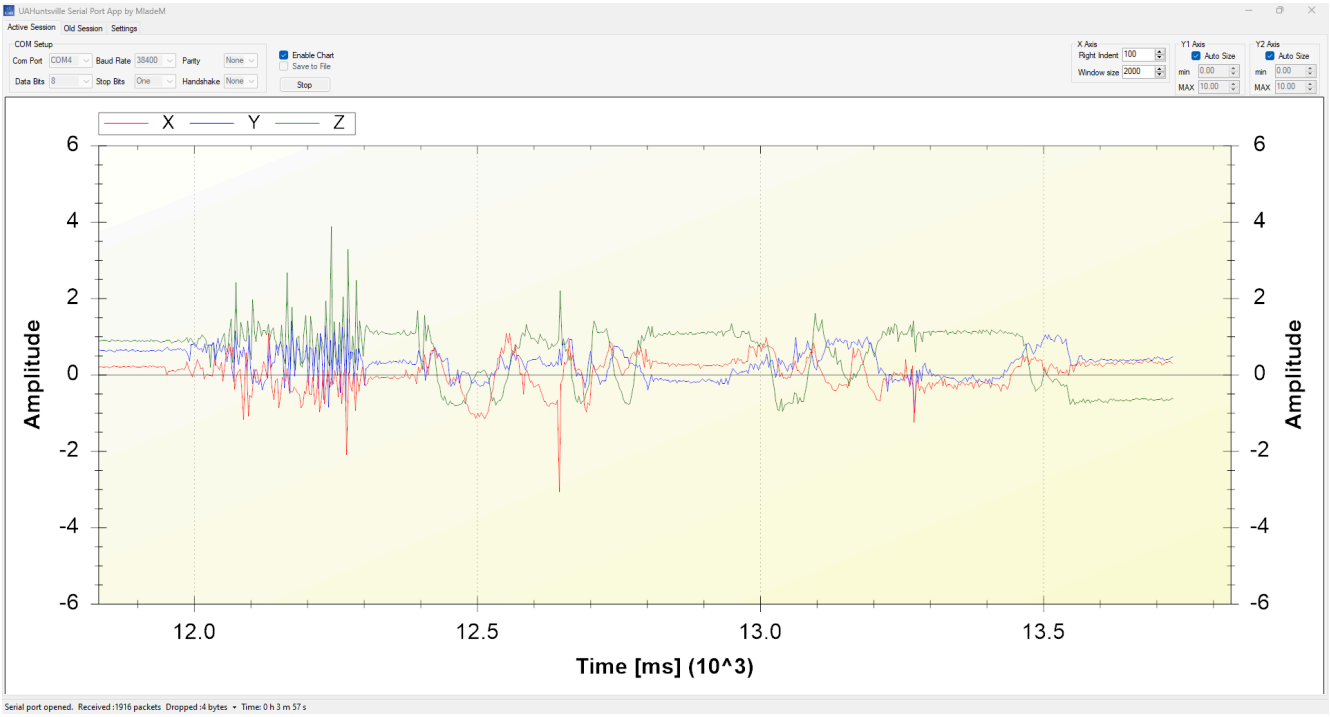

Figure 02: Program 1 Output on UAH Serial App

Program 2:

Program Description:

This C program modifies Program 1 to control the LEDs to visually represent whether the magnitude of acceleration reaches or exceeds 3g. The program calculates the magnitude of acceleration (M) with data from all three axes using the formula:

$$M = \sqrt{X^2 + Y^2 + Z^2}$$

If the magnitude is 3g or higher, the RED LED (P1.0) turns ON. When the magnitude is below 3g, the RED LED turns OFF, and the GREEN LED (P4.7) is ON, providing a visual indication of the acceleration threshold.

Program 3:

Program Description:

This C program generates waveforms using a Digital-to-Analog circuit (DAC) connected to the MSP430 microcontroller. It produces different output waveforms depending on the state of two switches. The program uses Timer B to periodically update the waveform values and outputs them on

Port 3. The waveform data is stored in a lookup table (lut256), which contains 256 samples for a sine wave representation. The program configures Timer B to trigger interrupts 30 times per second by using SMCLK as the clock source

In the Timer B interrupt service routine, the program increments the index variable i to iterate through the waveform samples. It then checks the state of the switches:

- When no switches are held: The program outputs a sine waveform using the values from the lookup table.
- When SW1 is held: The program generates a triangular waveform by directly assigning the value of i to Port 3
- When SW2 is held: The program doubles the frequency of the sine wave by incrementing the index by 2 on each interrupt cycle, resulting in the waveform changing twice as fast.

Program Output:

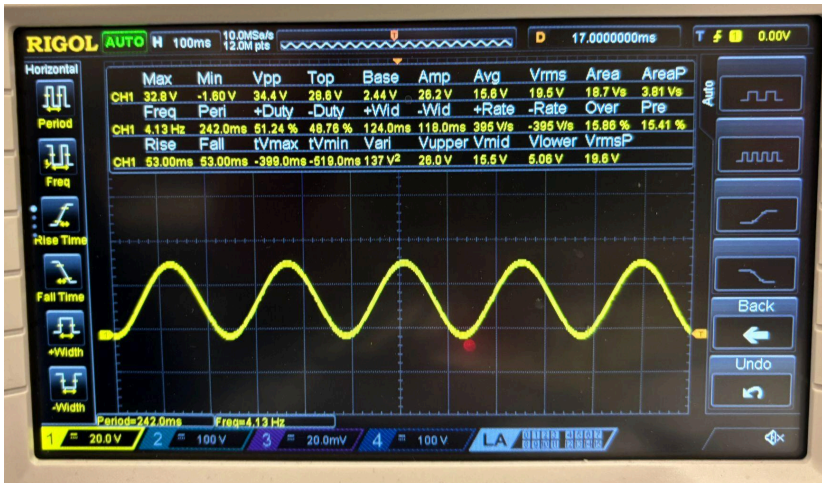

Figure 03: When no switches are held

Figure 04: When Switch 1 is held

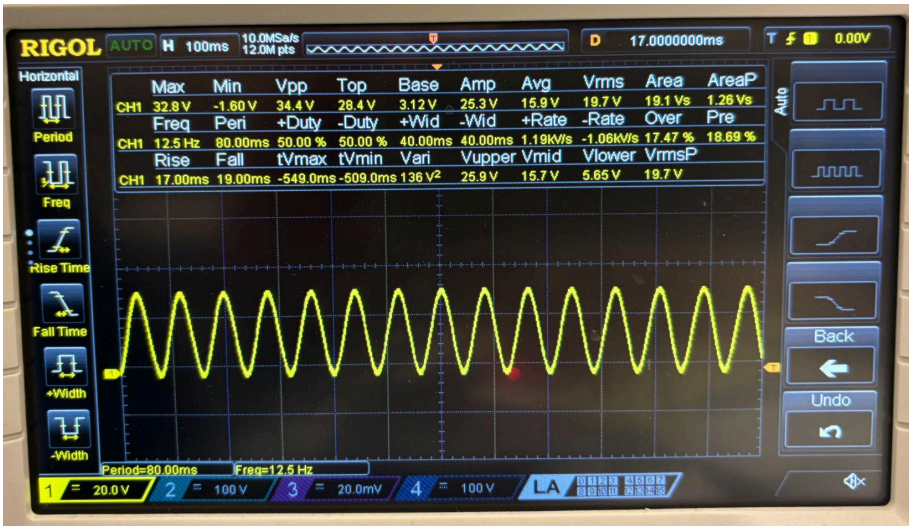

Figure 05: When Switch 2 is held

Program Flowchart:

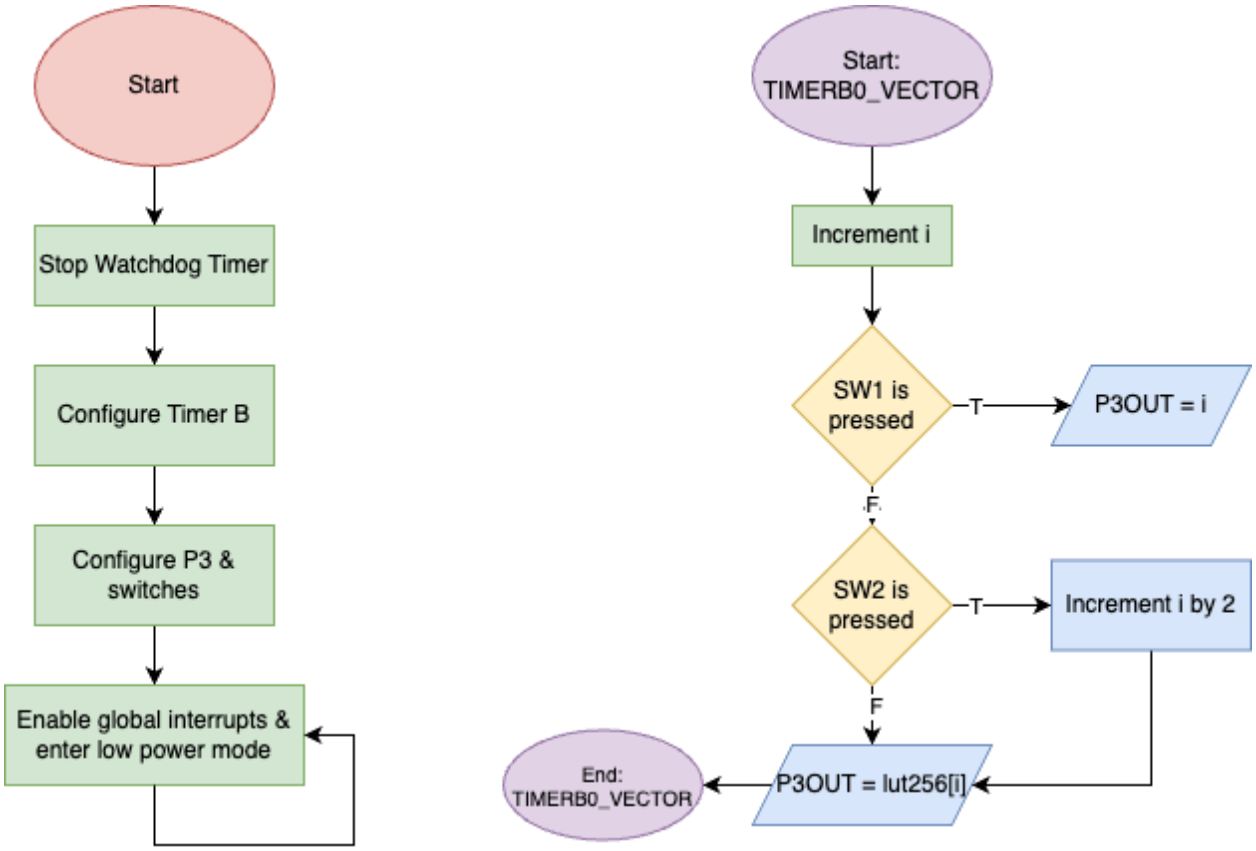

Figure 06: Program 3 Flowchart

Appendix

Table 01: Program 1 & 2 source code

```
/*-------------------------------------------------------------------------
* File:         Lab10_P1_P2.c
* Function:     Sample 3D Accelerometer
* Description: This C program interfaces an accelerometer, with the
MSP430F5529.
                It reads acceleration values from the X, Y, and Z axes,
converts them
                into digital values using the ADC12 module, and sends the data
over
                UART to the UAH Serial App.
* Input:        Accelerometer signal
* Output:       P1: X, Y, Z sample lines on UAH Serial App
*               P2: RED LED is on when |acceleration| reaches 3g, otherwise
                    GREEN LED is on.
* Author(s):    Anshika Sinha
* Date:         03/25/25
*-------------------------------------------------------------------------*/
#include <msp430.h>
#include <math.h>

#define ADC_TO_G(n) ((3.0 * n / 4095 - 1.5) / 0.3) //convert to g

volatile long int ADCXval, ADCYval, ADCZval;
volatile float Xper, Yper, Zper;
volatile float M;

void TimerA_setup(void)
{
   TA0CCTL0 = CCIE;                         // Enabled interrupt
   TA0CCR0 = 3276;                          // 3277 / 32768 = .1s for ACLK
   TA0CTL = TASSEL_1 + MC_1;                // ACLK, up mode
   P1DIR |= BIT0;                           // Setup LED1 at P2.1
   P4DIR |= BIT7;                           // Setup LED2 at P4.7
}
```

```
void ADC_setup(void) {
   int i =0;
   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
   // configure ADC converter
   ADC12CTL0 = ADC12ON + ADC12SHT0_6 + ADC12MSC;
   ADC12CTL1 = ADC12SHP + ADC12CONSEQ_3;   // 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
                                    // EOS - End of Sequence for Conversions
   ADC12IE = ADC12IE0;                 // Enable ADC12IFG.1
   for (i = 0; i < 0x3600; i++);       // Delay for reference start-up
   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 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;
```

```
   // Part 1 - get samples from ADC
   Xper = (ADC_TO_G(ADCXval));    // Calculate percentage outputs
   Yper = (ADC_TO_G(ADCYval));    // Calculate percentage outputs
   Zper = (ADC_TO_G(ADCZval));    // Calculate percentage outputs

   // Use character pointers to send one byte at a time
   char *xpointer=(char *)&Xper;
   char *ypointer=(char *)&Yper;
   char *zpointer=(char *)&Zper;

   UART_putCharacter(0x55);            // Send header
   for(i = 0; i < 4; i++)
   {
       UART_putCharacter(xpointer[i]); // Send x percentage - one byte at a
time
   }
   for(i = 0; i < 4; i++)
   {
       UART_putCharacter(ypointer[i]);// Send y percentage - one byte at a
time
   }
   for(i = 0; i < 4; i++)
   {
       UART_putCharacter(zpointer[i]);// Send z percentage - one byte at a
time
   }
}

//*********************************************************************//
MAIN
//*********************************************************************
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
    _EINT();

    while (1){
        ADC12CTL0 |= ADC12SC;               // Start conversions
        __bis_SR_register(LPM0_bits + GIE); // Enter LPM0
    }
}

#pragma vector = ADC12_VECTOR
__interrupt void ADC12ISR(void) {
    ADC12IFG = 0x00;
    ADCXval = ADC12MEM0;          // Move results, IFG is cleared
    ADCYval = ADC12MEM1;
    ADCZval = ADC12MEM2;
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void timerA_isr() {
    // Calculate magnitude of acceleration
    M = sqrt((Xper * Xper) + (Yper * Yper) + (Zper * Zper));

    if (M >= 3)
    {
        P1OUT |= BIT0;          // Turn on RED LED
        P4OUT &= ~BIT7;         // Turn off GREEN lED
    }
    else
    {
        P1OUT &= ~BIT0;         // Turn off RED LED
        P4OUT |= BIT7;          // Turn on GREEN LED
    }
    sendData();                             // Send data to serial app
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}
```

Table 02: Program 3 source code

```
/*-------------------------------------------------------------------------
* File:         Lab10_P3.c
* Function:     Generate waveforms using a Digital to Analog circuit
* Description:  This C program generates waveforms using a Digital-to-Analog
                circuit (DAC) connected to the MSP430 and alter them using
                the switches.
* Input:        DAC circuit connected to the MSP430
* Output:       When no switches are held: sine waveform using lookup table
                When Switch 1 is held: triangular waveform
                When Switch 2 is held: the frequency doubles
* Author(s):    Anshika Sinha
* Date:         03/25/25
*-------------------------------------------------------------------------*/
#include <msp430.h>
#include "Assignment_p3_sine_lut_256.h"

#define SW1 (P2IN & BIT1)
#define SW2 (P1IN & BIT1)

unsigned char i = 0;

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

   // Timer B Configuration
   TB0CCTL0 = CCIE;             // Timer Count Triggers Interrupt
   TB0CTL = TBSSEL_2 + MC_1;    // SMCLK is source, and set to up mode
   TB0CCR0 = 29;        // Cause the interrupt to trigger 30 times per second
   P3DIR = 0xFF;                // Enable all pins on Port 3 as outputs.
   P3OUT = 0x00;

   // Configure Switch 1
   P2DIR &= ~BIT1;              // Set P2.1 as input for SW1 input
   P2REN |= BIT1;               // Enable pull-up register at P2.1
   P2OUT |= BIT1;               // Set register
```

```
    // Configure Switch 2
    P1DIR &= ~BIT1;                 // Set P1.1 as input for SW1 input
    P1REN |= BIT1;                  // Enable pull-up register at P1.1
    P1OUT |= BIT1;                  // Set register

    __bis_SR_register(LPM0_bits + GIE);
    return 0;
}

#pragma vector=TIMERB0_VECTOR
__interrupt void timerISR2(void){
    i += 1;                 // Increment sine wave index

    if ((SW1) == 0) {       // If Switch1 is pressed
        P3OUT = i;          // Generate triangular waveform
    }
    else if ((SW2) == 0) {  // If Switch2 is pressed
        i += 2;             // Double frequency
        P3OUT = lut256[i];  // Generate sine wave using lookup table
    }
    else {
        P3OUT = lut256[i];  // Generate sine wave using lookup table
                            // when no switches are held
    }
}
```

Table 03: Script used to generate Lookup Table

```
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, ',');
```