CPE 325: Intro to Embedded Computer System

Lab10

ADC/DAC with MSP430

Submitted by: Dan Otieno

Date of Experiment: 03/31/22

Report Deadline: 04/01/22

Demonstration Deadline: 04/05/22

Introduction

This lab involves using the MSP430 board to communicate with an ADXL335 accelerometer. We use UART communication with the UAH serial app and an Oscilloscope for outputting our waves for the third part of the lab.

Theory

Topic 1: **ADXL335 ACCELEROMETER:**

This device is a 3-axis with outputs controlled by signals. We connect the accelerometer to the MSP430 board via Header 8 pins, the ground and VCC pins. When programmed, we can read the axis output signals, in this case using the UAH Serial App and observe the changes as we change the accelerometer's orientation. Parts 1 and 2 of this lab required interfacing the accelerometer via ADC signals and configuring the LEDs on the MSP430 to logics high or low depending on the angular deviation relative to the x-axis and comparing that deviation to the values 15 and -15.

Topic 2: **ADC and DAC:**

Analog-to-Digital Conversion is achieved by converters that enable us to interact with analog signals, convert them to digital values, store, interpret and observe them for further studying. For this lab, we utilized the ADC12 converter of the MSP430 to achieve our outputs. The first part of this lab involves utilizing ADC12 to convert signals using the accelerometer into our microcontroller. DAC achieves Digital to Analog conversion, and we use this method in the third part of the lab to output waveform signals at different amplitudes using an Oscilloscope.

Results & Observation

Program 1:

Program Description:

Program 1, covering parts one and two of this lab, as explained, involves using our accelerometer and MSP430 to observe 3-axis output signals on the UAH Serial App. The output screenshots are attached below but will be demonstrated in the laboratory.

Program Output:

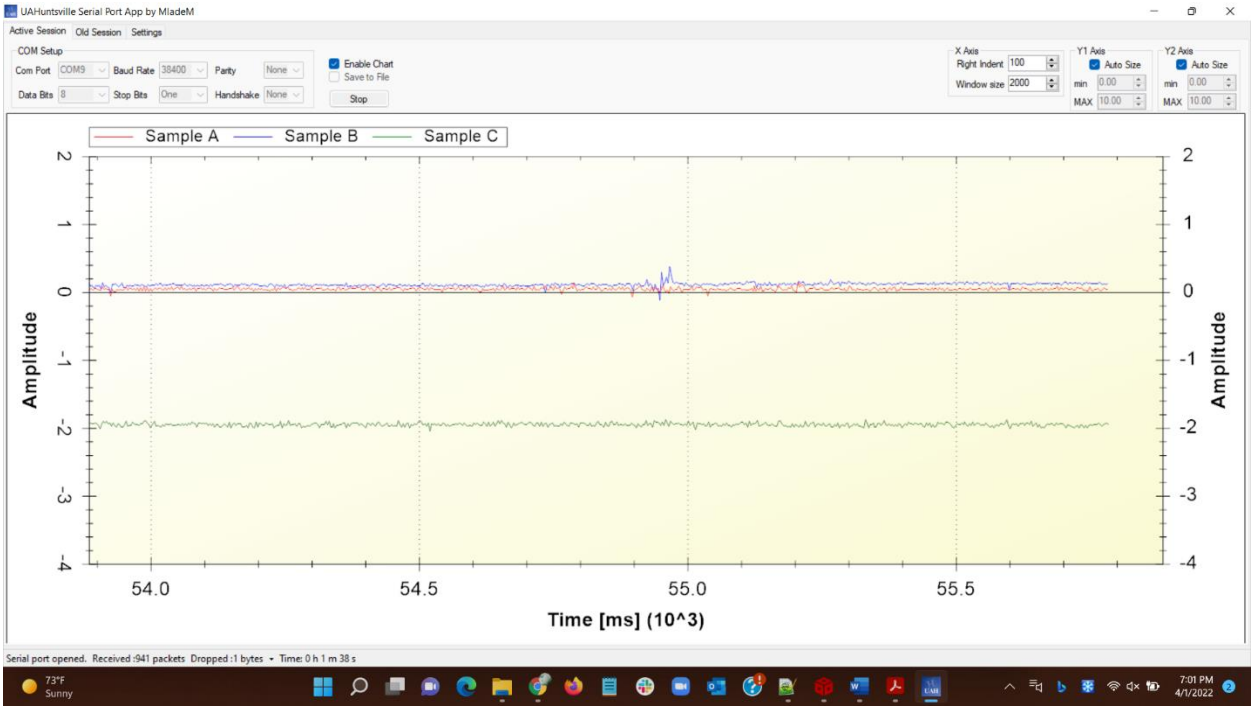

Figure 01: Axes outputs on UAH Serial App.

Program 2:

Program Description:

Program two was using DAC to output waveforms in an Oscilloscope. The first waveform is a sine wave, followed by a triangle wave when Switch 1 is pressed, and finally doubling the amplitude of the sine wave when switch 2 is pressed. We also used MATLAB to generate a lookup table for the waveforms and added that as a local header file in our project folder.

Program Output:

This will be demonstrated using an Oscilloscope in the laboratory.

Program LUT MATLAB code:

```
x=(0:2*pi/512:2*pi);
y=(2*(1+sin(x)));
dac12=y*4095/4;
dac12r=round(dac12);
dlmwrite('waves_lut_512.h',dac12r,',');
```

Conclusion

This was a good experience to learn ADC and DAC using actual instruments and experimentation.

Table 01: Part 1&Part 2 source code

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

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

void TimerA_setup();
void UART_setup();
void sendData();
void UART_Send_Character(char);

int main(void)
{
      WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer.
      TimerA_setup();              // Setup timer to send ADC data.
      ADC_setup();                 // Setup ADC.
      UART_setup();                // Setup UART for RS-232.

      P2DIR |= BIT1 + BIT2;          // P2.1 and P2.2 set up as output
      P2OUT &= ~(BIT1 + BIT2);       // ensure LED1 and LED2 are off
      _EINT();

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

#pragma vector = ADC12_VECTOR
__interrupt void ADC12ISR(void)
{
    ADCXval = ADC12MEM0;                   // Move results, IFG is cleared
    ADCYval = ADC12MEM1;
    ADCZval = ADC12MEM2;
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}
#pragma vector = TIMERA0_VECTOR
__interrupt void timerA_ISR()
{
```

```
    sendData();                             // Send data to serial app
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}

// Watchdog Timer interrupt service routine
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void)
{
    P2OUT ^= BIT2;                      // Toggle P2.2 using exclusive-OR
}

void TimerA_setup(void)
{
    TACCR0 = 3277;                      // 3277/ 32768 Hz = 0.1s
    TACTL = TASSEL_1 + MC_1;            // ACLK, up mode
    TACCTL0 = CCIE;                     // Enabled interrupt
}

void UART_setup(void)
{
    P2SEL |= BIT4 + BIT5;               // 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;
    Xper = ((ADCXval*3.0/4095-1.5)/0.3);    // Calculate percentage outputs
    Yper = ((ADCYval*3.0/4095-1.5)/0.3);
    Zper = ((ADCZval*3.0/4095-1.5)/0.3);
    aDev = atan(Xper/Zper)*180/3.141592;

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

    UART_Send_Character(0x55);          // Send header
    for(i = 0; i < 4; i++)
    {           // Send x percentage - one byte at a time
        UART_Send_Character(xpointer[i]);
```

```
    }
    for(i = 0; i < 4; i++)
    {            // Send y percentage - one byte at a time
        UART_Send_Character(ypointer[i]);
    }
    for(i = 0; i < 4; i++)
    {            // Send y percentage - one byte at a time
        UART_Send_Character(zpointer[i]);
    }

    if(aDev > 15)
    {
        WDTCTL = WDT_MDLY_32;           // 1 s interval timer
        //LED1 ON; LED2 OFF.
        P2OUT |= BIT2;
        P2OUT &= ~BIT1;
    }
    else if(aDev < (-1*15))
    {
        WDTCTL = WDT_MDLY_32;
        //LED2 ON; LED1 OFF.
        P2OUT |= BIT1;
        P2OUT &= ~BIT2;
    }
    else if(aDev > (-1*15) && aDev < 15)
    {
        //LED1 OFF; LED2 OFF.
        P2OUT &= ~(BIT1 + BIT2);
    }
}

void UART_Send_Character(char my_char)
{
    while(!(IFG2&UCA0TXIFG));   // Wait until can transmit.
    UCA0TXBUF = my_char;        // Tx Buffer gets my_char variable.
}

void ADC_setup(void)
{
    int i =0;

    P6DIR &= ~BIT3 + ~BIT5 + ~BIT7;     // Configure P6.4, P6.6 and P6.8 as input pins.
    P6SEL |= BIT3 + BIT5 + BIT7;        // Configure P6.4, P6.6 and P6.8 as analog pins.

    ADC12CTL0 = ADC12ON + SHT0_6 + MSC; // configure ADC converter
    ADC12CTL1 = SHP + CONSEQ_1;         // Use sample timer, single sequence
```

```
    ADC12MCTL0 = INCH_3;                    // ADC A3 pin - Stick X-axis.
    ADC12MCTL1 = INCH_7;                    // ADC A4 pin - Stick Y-axis.
    ADC12MCTL2 = INCH_5 + EOS;              // ADC A5 pin - Stick Z-axis.
                                            // EOS - End of Sequence for Conversions.
    ADC12IE |= 0x02;                        // Enable ADC12IFG.1
    for (i = 0; i < 0x3600; i++);           // Delay for reference start-up
    ADC12CTL0 |= ENC;                       // Enable conversions
}
```

Your next code goes here, if any.

Table 02: Part 3 source code

```
#include <msp430.h>
#include <waves_lut_512.h>  /*512 samples are stored in this table */

#define   SW1_PRESSED ((BIT0&P1IN)==0)
#define   SW2_PRESSED ((BIT1&P1IN)==0)

void TimerA_setup(void)
{
    TACTL = TASSEL_2 + MC_1;                // SMCLK, up mode
    TACCR0 = 68;                           // Sets Timer Freq, 30Hz (1048576*0.0333sec/512)
    TACCTL0 = CCIE;                         // CCR0 interrupt enabled
}

void DAC_setup(void)
{
    ADC12CTL0 = REF2_5V + REFON;           // Turn on 2.5V internal ref voltage
    unsigned int i = 0;
    for (i = 50000; i > 0; i--);             // Delay to allow Ref to settle
    DAC12_0CTL = DAC12IR + DAC12AMP_5 + DAC12ENC;   //Sets DAC12
}

int main(void)
{
     WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer.
     TimerA_setup();              // Set timer to uniformly distribute the samples
     DAC_setup();                // Setup DAC
     unsigned int i = 0;

     while (1)
     {
          __bis_SR_register(LPM0_bits + GIE); // Enter LPM0, interrupts enabled
```

```
        if(SW1_PRESSED)
        {
            if(i>255)
            {
                DAC12_0DAT = (512-i);
            }
            else
            {
                DAC12_0DAT = i;
            }
        }
        else
        {
            DAC12_0DAT = WAVELUT512[i];
        }

        if(SW2_PRESSED)
        {
            TACCR0 = 34;
        }
        else
        {
            TACCR0 = 68;
        }

        i=(i+1)%512;
    }
}

#pragma vector = TIMERA0_VECTOR
__interrupt void TA0_ISR(void)
{
    __bic_SR_register_on_exit(LPM0_bits);  // Exit LPMx, interrupts enabled
}
```

```
int WAVELUT512[]=
{2048,2073,2098,2123,2148,2173,2198,2223,2248,2273,2298,2323,2348,2373,2398,242
2,2447,2472,2496,2521,2545,2569,2594,2618,2642,2666,2690,2714,2737,2761,2784,28
08,2831,2854,2877,2900,2923,2946,2968,2990,3013,3035,3057,3078,3100,3122,3143,3
164,3185,3206,3226,3247,3267,3287,3307,3327,3346,3366,3385,3404,3423,3441,3459,
3477,3495,3513,3530,3548,3565,3581,3598,3614,3630,3646,3662,3677,3692,3707,3722
,3736,3750,3764,3777,3791,3804,3816,3829,3841,3853,3865,3876,3888,3898,3909,391
9,3929,3939,3949,3958,3967,3975,3984,3992,3999,4007,4014,4021,4027,4034,4040,40
```

45,4051,4056,4060,4065,4069,4073,4076,4080,4083,4085,4087,4089,4091,4093,4094,4
094,4095,4095,4095,4094,4094,4093,4091,4089,4087,4085,4083,4080,4076,4073,4069,
4065,4060,4056,4051,4045,4040,4034,4027,4021,4014,4007,3999,3992,3984,3975,3967
,3958,3949,3939,3929,3919,3909,3898,3888,3876,3865,3853,3841,3829,3816,3804,379
1,3777,3764,3750,3736,3722,3707,3692,3677,3662,3646,3630,3614,3598,3581,3565,35
48,3530,3513,3495,3477,3459,3441,3423,3404,3385,3366,3346,3327,3307,3287,3267,3
247,3226,3206,3185,3164,3143,3122,3100,3078,3057,3035,3013,2990,2968,2946,2923,
2900,2877,2854,2831,2808,2784,2761,2737,2714,2690,2666,2642,2618,2594,2569,2545
,2521,2496,2472,2447,2422,2398,2373,2348,2323,2298,2273,2248,2223,2198,2173,214
8,2123,2098,2073,2048,2022,1997,1972,1947,1922,1897,1872,1847,1822,1797,1772,17
47,1722,1697,1673,1648,1623,1599,1574,1550,1526,1501,1477,1453,1429,1405,1381,1
358,1334,1311,1287,1264,1241,1218,1195,1172,1149,1127,1105,1082,1060,1038,1017,
995,973,952,931,910,889,869,848,828,808,788,768,749,729,710,691,672,654,636,618
,600,582,565,547,530,514,497,481,465,449,433,418,403,388,373,359,345,331,318,30
4,291,279,266,254,242,230,219,207,197,186,176,166,156,146,137,128,120,111,103,9
6,88,81,74,68,61,55,50,44,39,35,30,26,22,19,15,12,10,8,6,4,2,1,1,0,0,0,1,1,2,4,
6,8,10,12,15,19,22,26,30,35,39,44,50,55,61,68,74,81,88,96,103,111,120,128,137,1
46,156,166,176,186,197,207,219,230,242,254,266,279,291,304,318,331,345,359,373,
388,403,418,433,449,465,481,497,514,530,547,565,582,600,618,636,654,672,691,710
,729,749,768,788,808,828,848,869,889,910,931,952,973,995,1017,1038,1060,1082,11
05,1127,1149,1172,1195,1218,1241,1264,1287,1311,1334,1358,1381,1405,1429,1453,1
477,1501,1526,1550,1574,1599,1623,1648,1673,1697,1722,1747,1772,1797,1822,1847,
1872,1897,1922,1947,1972,1997,2022,2047};