Embedded Systems Lab

CPE 325-02

Introduction to Device Interfacing and

Analog to Digital Conversion on MSP430

By: David Thornton

Lab Date: October 27, 2020

Lab Due: November 11, 2020

Demonstration Due: November 11, 2020

Introduction

This lab builds on serial communication and UART with the MSP430. This lab utilizes the UAH serial app. This lab introduces analog to digital conversion.

Theory

Topic 1: Analog To Digital Converter

- “An ADC converts a continuous-time and continuous-amplitude analog signal to a discrete-time and discrete-amplitude digital signal. The conversion involves quantization of the input, so it necessarily introduces a small amount of error or noise. Furthermore, instead of continuously performing the conversion, an ADC does the conversion periodically, sampling the input, limiting the allowable bandwidth of the input signal.” - Source

Topic 2: 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 millivolts). 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.” - Source

Topic 3: Accelerometer Interfacing

1. ADCXval/ADCYval
2. Divide by 4095 steps
3. Multiply by 3.3 V (input voltage)

4. For X/Y axises, subtract 1.65 V (0g max voltage for X/Y). For the Z axis, subtract 1.8 V (0g max voltage for X/Y).
5. Divide by 0.330 V/g (max sensitivity for X, Y, Z)

Lab Assignment

1. Write a C program to interface the 3-dimensional accelerometer ADXL335. Your program should achieve the following tasks:
    a. Sample the x, y, and z axises 10 times per second.
    b. Calculate the accelerations in terms of ‘g’ (gravity of Earth) and send the sample to the workstation. i.e. the samples (gx, gy and gz) should be sent such that it is displayed in the UAH serial app.
    c. Calculate a value, netG, which is a vector sum of g values in each axis.
    d. If the value of netG is greater than or equal to 2g of earth, blink two LEDs alternately at 1Hz using Watchdog Timer ISR. This blinking should only stop when any one of the switches is pressed.
    e. Similar to the calculations presented in tutorial at Section 1.2 for temperature sensor, using the transfer characteristics and voltage sensitivity of the accelerometer, and the knowledge about the reference voltages, show how you compute gx, gy and gz from the ADC values.

Observations

All code satisfies the lab requirements. There are two potential improvements. Firstly, for some reason upon starting the program for the first time, the z component spikes, causing the LEDs to blink. This is rectified by pressing a button. Secondly, the program does not check if the LEDs are already blinking. This causes the LEDs to occasionally not work as intended if the condition has already been met.

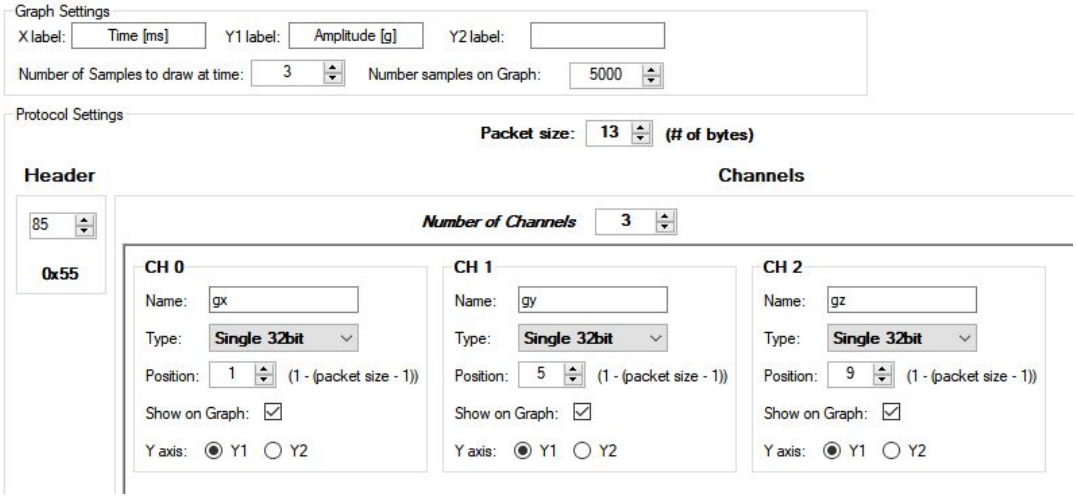

Figure 1: Settings in the UAH Serial App

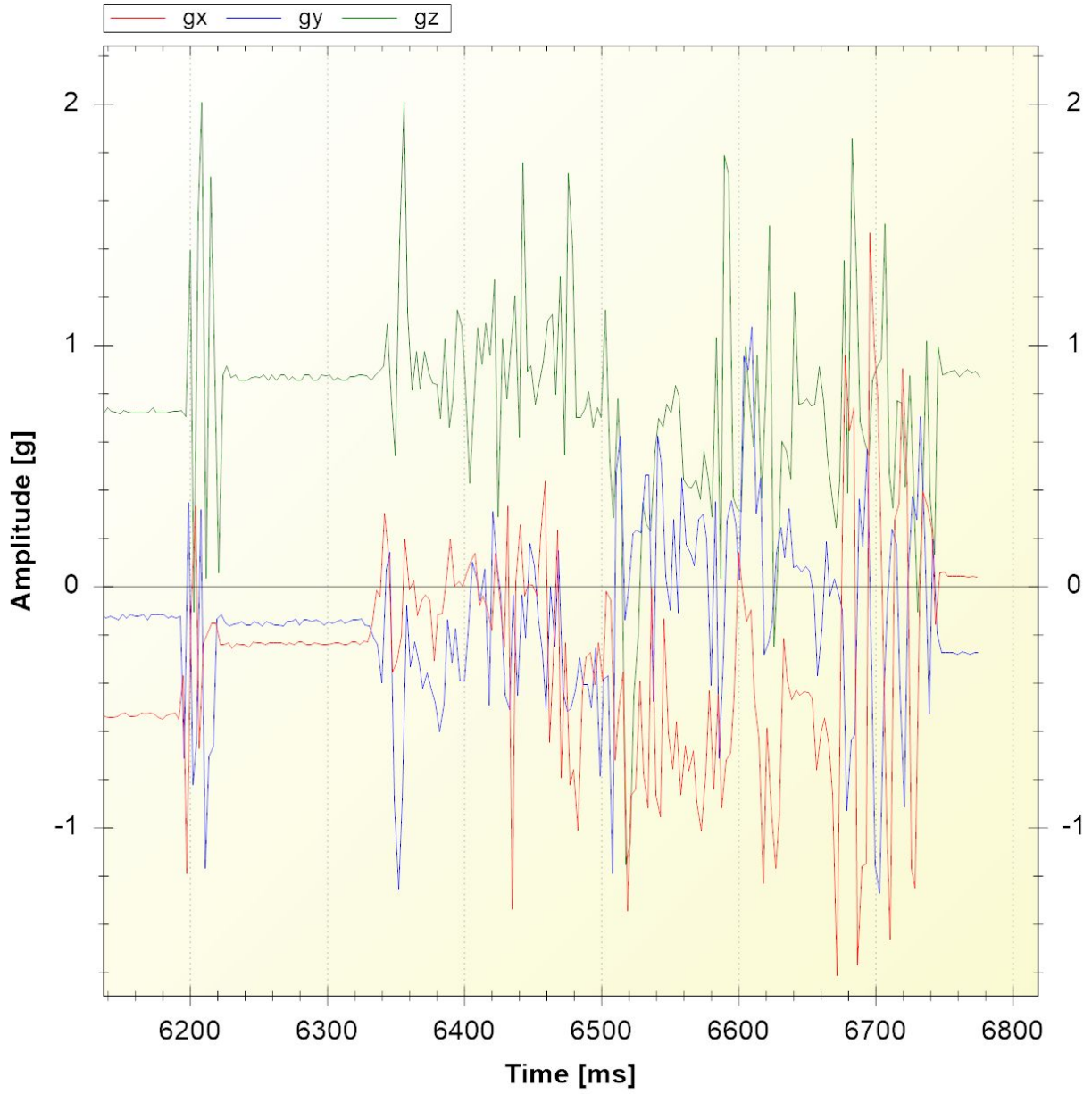

Figure 2: Normal operation of the program.

Conclusion

[Demo link]

Appendix

Appendix 1: Lab_9.c

```
/*-------------------------------------------------------------
* File:           Lab_9.c
* Description:    This program measures acceleration
* Input:          ADXL335 accelerometer
* Output:         Blinking LEDs and data in UAH serial app
* Author:         David Thornton
* Lab Section:    2
* Date:           November 11, 2020
* *-------------------------------------------------------------*/
#include <msp430.h>
#include <math.h>
#define RED 0x01                                // Red LED
#define GREEN 0x80                                  // Green LED
#define S1 ((P2IN&BIT1) == 0)             // Switch 1 push defined
#define S2 ((P1IN&BIT1) == 0)             // Switch 2 push defined

// Global variables for later use
volatile long int ADCXval, ADCYval, ADCZval = -1;
volatile float XDir, YDir, ZDir, netG = -1;

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

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)   // Infinite loop
	{
		__bis_SR_register(LPM0_bits + GIE); // Enter LPM0
		sendData();       // Send out the data
	}
}

void sendData(void)
{
	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;

	unsigned int i;
	UART_putCharacter(0x55);      // Send header

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

```
      }
      for(i = 0; i < 4; i++)
      {
            UART_putCharacter(ypointer[i]);
      }
      for(i = 0; i < 4; i++)
      {
            UART_putCharacter(zpointer[i]);
      }

      // Use vector sum for calculation of netG
      netG = sqrtf((XDir*XDir) + (YDir * YDir) + (ZDir * ZDir));
      if(netG >= 2)
      {
            WDTCTL = WDT_MDLY_0_5;  // 0.5ms interval timer
            P1OUT |= RED;                 // Turn on REDLED
            P4OUT &= ~GREEN;        // Make sure GREENLED is off
      }
}


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
}


#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 = 1;
      if(i == 1000)
      {
            P4OUT ^= GREEN;   // Toggle green LED
            P1OUT ^= RED;     // Toggle the red LED
            i = 1;
      }
      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
}
```