Nov 24th, 2020
Amr Usef
ECE 216

Laboratory 11

I affirm that I have not given or received any unauthorized help on this laboratory assignment, and that all work is my own- Amr Usef.

Section 1: Overview

This laboratory exercise is for students to demonstrate their understanding and ability to write program which is able to read an input from a potentiometer and communicate that input to a servomotor over UART communication. To be able to complete this lab successfully students must demonstrate a strong understanding of the UART communication protocol and the Pololu Protocol. Overall, in this laboratory exercise students are to take in the value of the potentiometer, communicate that value over via UART, and signal to the Micro Maestro 6-Channel USB servo controller where to move and by how much.

Section 2: Design

The design I choose for this program is a simple design which relays on UART 2, timer2, and the ADC conversion of the potentiometer to communicate to the server motor at what degree it should be. This program also uses the LCD screen to show the 7 bit value of the server motor control, to display on to the LCD screen I used the same codes from pervious labs.

I started my codes by configuring TRISE's last 8 bits to outputs to control the LCD screen, following that with DDPCON set to zero for debugging. Also, I call to initialize my LCD screen. This follows with initiating my UART2 for communication in which we were asked to set 8 bits, no parity, and 1 stop bit as well as setting our U2BRG using the baud rate of 9600. Also, I set up my timer2 to run at 10 Hz therefore calculating PR2 which came out to be 31249 using a scaler of 256.

Before my main function where much of the heavy lifting is happening, I set up three static volatile floats pervious_value, current_vaule, and first_time. The pervious value and first time variables are to be used to check a condition in the timer2 ISR to not allow overprinting to the LCD screen. Also, using my code for ADC_CONVERTER from lab 5 I set up my conversion from analog to digital for the value of the potentiometer which will be used to control the direction and speed of the servomotor.

Following that with creating a function that will write the UART message to U2TXREG which will write to the servomotor controller. The function for writing the UART message to the servomotor controller follows that of the function used to write UART messages in lab 10 with the exception that we will not be dealing with an end line character.

Finally, following the write_message function is the Timer2 ISR which will write on the LCD screen and write the commands to the Maestro 6-channel USB Servo Controller to control

speed and direction of the motor using the value coming in from the ADC_CONVERTER. I start by creating a char variable of 100 which will hold all the commands sent to the servomotor controller. A variable named val which will hold the value of direction we are trying to move our motor in. Following that with to unsigned ints of 8bits long lower_bits and upper_bits to hold the commands for the lower 7 bits for lower_bits and bits 7 to 13 for upperbits of val. Then I set my current_val to the reading coming in from ADC_CONVERTOR for potentiometer, and I divide that value by 1023 and save it into div (0<= div <=1 at all times). I follow that with the calculation for val, I multiply 1000 microseconds by 4 since we operate in a quarto of microseconds and multiply that by 1 + div therefore val will always be between 4000(-45)-8000(45). The calculation for val will decide whether to move the motor to -45 degrees or positive +45 degrees. I follow that with setting the lower bits to val & 0x7F (127), and I shift val to the upper 7 bits and apply the same method to the upper bits. Then I set a condition statement that will check if the potentiometer has changed or if it is our first time running the code, so we don't over print to LCD screen. Using the sprintf function I save the 6 values into hex_val and send the 5 values over to the servomotor controller using the UART write function. The 6 commands that are sent are the start byte (0xAA), the device ID (0x0C), command byte (0x04), command for using channel 0 (0x00), and the lower and upper bits for direction control.

Section 3: Expected behavior

The expected behavior of this program is as follows; the servomotor will be at 0 degrees if the potentiometer is at the middle point (510), will rotate to +45 degrees when the potentiometer is moved towards the right (1023), and will rotate to -45 degrees when the potentiometer is moved to the left (0). Also, accordingly the 7 hex values should reflect the command given to the servomotor and be displayed on the LCD screen. However only the most significant bits should be changing with the change of the position of the potentiometer while the other 5 values should stay the same.

Section 4: code

Main.c-

```
/*
 * File:   main.c
 * Author: amr usef
 * lab:11
 *date: 11/23/20
 * Discription: in the main file what we are dong is using UART communcation
to
 * write to a servomotor controller. Also, we are using the potentometer to
adjust the
 * dirction of the servomotor by either +45 or -45 degrees.
 */


#include <stdio.h>
#include <stdlib.h>
#include <sys/attribs.h>
#include <xc.h>
#include <string.h>
#include "lcd_display_driver.h"
//Furn CPU at 80MHz
#pragma config POSCMOD = HS
#pragma config FNOSC = PRIPLL
#pragma config FPLLMUL = MUL_20
#pragma config FPLLIDIV = DIV_2
#pragma config FPLLODIV = DIV_1
#pragma config FPBDIV = DIV_1

#define SAMPLE_TIME 10 //250 ns
#define BAUD 9600
#define FREQ_SYSTEM 80000000

static volatile float pervious_vaule=0.0, current_vaule=0.0, first_time=1.0;
//to allow the lcd to display with out flackring

unsigned int ADC_CONVERTER(void){
    unsigned int elapsed =0, finish_time =0;//to be used in my sampling
    AD1CHSbits.CH0SA = 2;//MUXA pin
    AD1CON1bits.SAMP = 1;// start sampling
    elapsed =_CP0_GET_COUNT();//to get current time
    finish_time = elapsed + SAMPLE_TIME;//to set to get finish time
    while(_CP0_GET_COUNT() < finish_time){
        ;
```

```
    }
    AD1CON1bits.SAMP = 0;// stop sampling
    while(!AD1CON1bits.DONE){
        ;
    }
    return ADC1BUF0;
}
void write_message(char message){
        while(U2STAbits.UTXBF){
            ;//wait until TX buffer isn't full
        }
        U2TXREG=message;
}
void __ISR(_TIMER_2_VECTOR, IPL4SOFT) Timer2ISR(void) {
    char hex_val[100];//store the target angle to be used later on
    int val =0;
    uint8_t lower_bits=0;
    uint8_t upper_bits=0;
    current_vaule=ADC_CONVERTER();
    float div= (current_vaule/1023.0);
    val = (int)(4*1000)*(1+div);//operating in a quator of micro sec
    lower_bits = val & 0x7F;
    upper_bits =(val >> 7) & 0x7F;

    //display the target degree and angle
    if(first_time=1.0 ||pervious_vaule!= current_vaule){
    sprintf(hex_val,"%x %x %x %x %x
%x",0xAA,0xC,0x04,0x0,lower_bits,upper_bits);
    lcd_display_driver_clear();//clear before writing
    lcd_display_driver_write(hex_val, strlen(hex_val));//writing into the
first line
    write_message(0xAA);//start byte
    write_message(0xC);//device ID
    write_message(0x04);//Command byte
    write_message(0x0);//using channel 0
    write_message(lower_bits);
    write_message(upper_bits);
    pervious_vaule=current_vaule;//for conditon to be checked
    first_time=first_time+1.0;//to be used to print on to the LCD screen on
first run in timer
    }
    IFS0bits.T2IF = 0;//clear timer interrupt flag
    return;
}
int main() {
    TRISE = 0xFF00;//tri-state E for output config for LCD
    DDPCON =0x0;   //debugging tool used when ever calling Tri-state
registers or using LATx
    lcd_display_driver_initialize();//calling to initialize the lcd display

    INTCONbits.MVEC=1;
      __builtin_disable_interrupts();
    U2MODEbits.BRGH=0;
    U2BRG =((FREQ_SYSTEM /BAUD)/16)-1;//((80*10^6)/(9600)/(16))-1=520
    //8 bits, no parity, and 1 stop bit
    U2MODEbits.PDSEL=0;
    U2MODEbits.STSEL=0;
```

```
    U2STAbits.UTXEN=1;//enable TX
    U2STAbits.URXEN=1;//enable RX
    U2MODEbits.UEN=0;//only U2TX and U2RX pins to be used
    U2MODEbits.ON=1; //turn on UART

    PR2=31249; //(80MHz*(1/10)*(1/256))-1=PR2 -> 31249
    TMR2 = 0;//initializes count to zero
    T2CONbits.TCKPS = 7;//prescaler 256
    T2CONbits.TGATE = 0;
    T2CONbits.TCS = 0;
    T2CONbits.ON = 1;//time on
    IPC2bits.T2IP = 4;//priority 4
    IPC2bits.T2IS = 0;//sub-priority
    IFS0bits.T2IF = 0;//clears flag
    IEC0bits.T2IE = 1;//enables interrupt
    __builtin_enable_interrupts();

    while(1){
        ;
    }

    return (0);
}
```

lcd_display_driver.h-

```
/*
 * File:   lcd_display_driver.h
 * Author: amr usef
 * Date: Created on Nov 08, 2020, 8:46 PM
 * Discripition: This file contains the function naming to be used in main.c
 */
#ifndef LCD_DISPLAY_DRIVER_H
#define LCD_DISPLAY_DRIVER_H

void lcd_display_driver_enable();

void lcd_display_driver_initialize();

void lcd_display_driver_clear();

void lcd_display_driver_write(char * data, int length);

void display_driver_use_first_line();

void display_drive_use_second_line();

#endif  /* LCD_DISPLAY_DRIVER_H */
```

lcd_display_driver.c-

```
/*
 * File:   lcd_display_driver.c
 * Author: amr usef
 * Date: Created on Nov 01, 2020, 8:46 PM
 * Discripition: This file contains the functions implemntation to be able to
read and write to the LCD screen
 */
```

```
#include <stdio.h>
#include <xc.h>
#include "lcd_display_driver.h"

//implementing lcd display driver enable
void lcd_display_driver_enable(){
    TRISDbits.TRISD4 =0;//setting D4 as output
    LATDbits.LATD4 =1; //D4 enable bit to on
    int j;//naming a var
    for(j=0; j<=15000; j++); //delay of 15000 for enable to stay on
    LATDbits.LATD4 =0; //D4 enable to off
}

void lcd_display_driver_clear(){
    TRISBbits.TRISB15 =0;//set RS to be output
    TRISDbits.TRISD5 =0; //reasure that R/W is output
    LATBbits.LATB15 = 0; //setting RS to 0V
    LATDbits.LATD5 = 0;  //setting RS to 0V
    LATE = 0b00000001; //set display driver clear by setting DB0 to 1
    lcd_display_driver_enable();
}
void lcd_display_driver_write(char*data, int length){
     TRISBbits.TRISB15 =0;//set RS to be output
     LATBbits.LATB15 = 1; //setting RS to 3.3V
     int i;
     LATE = 0b00000000;// clear LATE just incase it is preset
     for(i=0; i < length; i++){
         LATE = data[i]; // translate between asc and letters to lcd
         lcd_display_driver_enable();
     }
}

void lcd_display_driver_initialize(){
    //function set
     TRISBbits.TRISB15 =0;//set RS to be output
     TRISDbits.TRISD5 =0; //reasure that R/W is output
     LATBbits.LATB15 = 0; //setting RS to 0V
     LATDbits.LATD5 = 0;  //setting RS to 0V
    LATE = 0b00111000;//initiat set function to have dual line
     int i;
     for(i =0; i< 1000; i++);//delay
     lcd_display_driver_enable(); //to be able to write and read
     //display on/off function
     LATE =0b00001100; //cursor off blink off display on
     for(i =0; i< 1000; i++);//delay
     lcd_display_driver_enable(); //to be able to write and read

     lcd_display_driver_clear();//calling clear function to clear any
displays
     for(i =0; i< 16000; i++);//delay

     LATE = 0b00000100; //entry mod set
     lcd_display_driver_enable(); //to be able to write and read
}
```