Dec 7th, 2020
Amr Usef
ECE 216

Laboratory 12

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 use the SPI communication interface to take a message from an external device and output that message to a motor. Using SPI4 and the SPI interrupt student are to process a coordinate message from a joystick on a controller and using that coordinate decide whether the dule motors should move either forwards, backwards, or not move at all. Also, using the position of the joystick students are to come up with an equation to calculate the duty cycle of OC4RS.

Section 2: Design

The design I choose for this program is a simple design which follows the in class exercise and relays on SPI4 to take in a message display it on to the LCD screen such that it shows as “X:x.xx Y:x.xx”, also using the message to control the speed and direction of the motor. I stated the program by setting my system to run at 80 MHz, set the necessary tri-state registers to be able to control the motor and the LCD screen. To be able to communicate with the H-bridge I used F0 and F1 as system outputs. Following that with setting my OC4 with the PWM mode with no fault pin, an 8:1 pre-scaler and used timer 3 to control the PWM. I set my PR3=1023 and my OC4R and OC4RS to zero, which means the motor will not rotate when we start our program.

I followed that with setting up my SPI4 and the SPI ISR. I set my SPI4 with 9-bit data packets, slave mode, clock polarity selects to zero (clock is idle when low, active when high), clock edge select bit to zero, clearing my RX buffer before reading it in my ISR, and disabling my SDO4 pin. We are setting SPI4 to be the slave device since it will only be receiving messages from the Logitech warless gamepad, and the SPI interface will not be used to send messages. Following that with setting the priority and sub-priority of my ISR for SPI4, clearing the interrupt flag, and enabling my ISR.

In my while(1) loop I print the X and Y coordinates on to the LCD screen, the message only prints whenever the value of X or Y changes or if it’s our first time running the code. In my SPI4 ISR if a message is received, I extract that message out of the SPI’s buffer whenever it is full. To extract the message, I created a variable “done” that starts as zero and becomes one whenever we reach the end of the message ‘\n’. I place the message received into a char of length 20, and I follow that with the extraction of the X and Y coordinates. To extract the X and Y coordinates separately I used a token to indicate ‘:’, following that with using a string compare for x_check and y_check. The x_check is equale to zero whenever token is equal to “GX:” and the same for the y_check. Using the NULL pointer, I store the value into char named value and extract the coordinate using atoi and place it into an int. I follow that with an if statement that checks if x_check is equal to zero, and if so, places the normalized value of the coordinate into precent_gamepad_x, and the same condational for y_check.

Out precent_gamepad_x and precnt_gamepad_y are values between -1.0 and 1.0 both variables are initiated as static volatile floats. To control the speed and direction of the motors I used 3 conditionals which check the range of precent_gamepad_y. The first condition checks whether precent_gamepad_y is less then -.1, if so than I set F0=1 and F1=0 which means the motors rotations should decrease. Also, to calculate the speed of the motor I subtract the value of precent_gamepad_y from zero (making it positive) and multiplying it by 1023 (OC4RS will always be <= 1023) and setting that value to OC4RS. The conditional statement for precent_gamepad_y is greater than 0.1 follows the same logic as the above conditional for setting speed however, to decrease the rotations of the motor I set F1=1 and F0=0 here. My third conditional check if percent_gamepad_y is between -0.1 and 0.1 if so, we set F1=0 and F0=0 telling the H-bridge not to send current to the motors. And finally, I clear my RX flag for the ISR.

Section 3: Expected behavior

The expected behavior of this program is as follows; when the program is run for the first time the LCD screen should display “X:0.00 and Y:0.00”. Whenever the user moves the joystick on the controller downwards the motor should rotate forwards at a speed proportional to the position of the joystick. Also, the new values of X and Y should be displayed on to the LCD screen. And whenever the joystick is moved forwards the motor should rotate backwards, and the new X and Y values should display on to the LCD screen.

Section 4: Code

Main.c:

```
/*
 * File:   main.c
 * Author: amr usef
 * lab:12
 *date: 12/3/20
 * Discription: This is the main file of this lab, in our main file we want
to be able to display on to the LCD screen,
 * get messages from controller via raspary *pie, and to write to the motor
to move forwards or backword.
 */


#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
#pragma FWDTEN = OFF

static volatile int first_time_in=1;//for the condation to display
static volatile float precent_gamepad_x =0.0, past_x=0.0;
static volatile float precent_gamepad_y =0.0, past_y=0.0;

void __ISR(_SPI_4_VECTOR, IPL1SOFT) SPI4Handler(void){
    //check to see if there was a receive
    if(IFS1bits.SPI4RXIF){
        int done =0;//incidates message is recived
        int i=0;
        char strread[20];// string to read the message sent by ras pi

        while(!done){
            while(!SPI4STATbits.SPIRBF){
             ;
             }
        strread[i] = SPI4BUF;// reading the buffer and saving it to strread
        if(strread[i]=='\n'){
            done=1;//have reached end of message
        }
        i++;
        }
        char *token = strtok(strread,":");//breaking up data by ":"
        int x_check=strcmp("GX", token);//extacting GX into cmp
        int y_check=strcmp("GY", token);//extacting GX into cmp
        char*value = strtok(NULL, ":");
        int sh = atoi(value);
        if(x_check==0){
            precent_gamepad_x =(float) (sh)/32768.0;
        }else if(y_check==0){
            precent_gamepad_y =(float) (sh)/32768.0;
        }

        //motion control
        if(precent_gamepad_y<-.1){//moving back
             LATFbits.LATF0=1;//F0=1
             LATFbits.LATF1=0;//F1=0
             int speed =(int) (0-precent_gamepad_y)*1023;
             OC4RS = speed;
        }else if(precent_gamepad_y>.1){//moving forwards
            LATFbits.LATF0=0;//F0=0
            LATFbits.LATF1=1;//F1=1
            int speed =(int) (0+precent_gamepad_y)*1023;
            OC4RS = speed;
        }else{//no movement
            LATFbits.LATF0=0;//F0=0
            LATFbits.LATF1=0;//F1=0
        }
         IFS1bits.SPI4RXIF=0;//clear RX flag
    }else if(IFS1bits.SPI4TXIF){
        IFS1bits.SPI4TXIF =0; //clear TX flag
    }else{
        //do nothing
    }
    }

int main(void) {
    TRISE = 0xFF00;//tri-state E for output config for LCD
    DDPCON =0x0;   //debugging tool used when ever calling Tri-state
registers or using LATx
    LATA =0x0; //set the value of lATA to zero
    char disp[20];//to display the X and Y corrd on LCD screen
    lcd_display_driver_initialize();//calling to initialize the lcd display

    TRISFbits.TRISF0=0;//F0 as an ouput
    TRISFbits.TRISF1=0;//F0 as an ouput

    INTCONbits.MVEC = 1;//allow many interrputs


    __builtin_disable_interrupts();
    PR3 = 1022;//sets period
    TMR3 = 0;//start at 0
    T3CONbits.TCKPS = 0b011;//prescaler-8
    OC4CONbits.OCM = 0b110;//no fault pins, PWM
    OC4CONbits.OCTSEL = 1;//to use timer3 for timing
    OC4R= 0;//set to be used with potenitmeter
    OC4RS = 0;//set to be used with potenitmeter
    T3CONbits.ON = 1;//turn on timer
    OC4CONbits.ON = 1;//turn on timer

    //Slave - SPI4
    SPI4BUF;    //clear the rx buffer be reading from it
    SPI4STATbits.SPIROV=0; //clear the overflow bit
    SPI4CONbits.CKP=0;//clock is idle when low, active when high
    SPI4CONbits.DISSDO=1;//SDO4 is disabled
    SPI4CONbits.CKE=0;//clock edge select bit
    SPI4CONbits.SRXISEL=1;
    //operating in 8 bits
    SPI4CONbits.MODE16 =0;
    SPI4CONbits.MODE32 =0;
    SPI4CONbits.MSTEN=0; //operating in slave mode
    SPI4CONbits.ON=1;

    //SPI 4 interrupt
    IPC8bits.SPI4IP = 1; //for now
    IPC8bits.SPI4IS=1;
    IFS1bits.SPI4RXIF=0; //clear flag
    IEC1bits.SPI4RXIE=1; //enable
    __builtin_enable_interrupts();

  while(1){
    sprintf(disp,"X:%5.2f Y:%5.2f",precent_gamepad_x, precent_gamepad_y);
    if(first_time_in=1 ||precent_gamepad_x!=
past_x||precent_gamepad_y!=past_y){
            lcd_display_driver_clear();//clear before writing
            lcd_display_driver_write(disp, strlen(disp));//writing into the
first line
            past_x=precent_gamepad_x;//for conditon to be checked
            past_y=precent_gamepad_y;//for condation to be checked
            first_time_in++;//to be used to print on to the LCD screen on
first run in timer 2 ;
     }
  }
    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);

#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<=1000; 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
}
```