Nov 18th, 2020
Amr Usef
ECE 216

Laboratory 10

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 DC motor, write back to the DC motor, display the current angle and desired angle on to the LCD screen, as well as communicate with MATLAB over UART communication protocol. To be able to write a successful program, students must have a solid understanding of timers, output compares, interrupts, H-bridge, the DC motors, as well as using the UART communication protocol to transmit and receive messages. Also, in this lab students were to use the UART communication protocol to receive the target angle (reference angle), and transmit back to the MATLAB code the time, reference angle, current angle error, and integral of the error during a window time.

Section 2: Design

The design that I choose for my program is a simple design which relies on outputs of functions and global variables to execute successfully. I started my program with setting the needed Tri-state registers as outputs, the Tri-state registers initialization is mainly used for the LCD screen set up. Following that with the setting DDPCON and LATA both to zero and calling the lcd_display_driver_initalize(). Following the call to lcd_display_driver_initalize() with making switch S4 and S5 to inputs, and F0 and F1 as outputs to control direction of the motor. Also, in my main I initialize my timers, interrupts, and output compare, following that with a while(1) loop that will wait on timers and interrupts.

To read the angle of my motor I used a change notice ISR to read the relative encoders A and B which are ports G6 and G7. I started by creating a few global variables count, bit_A_old, and bit_B_old. Count will be storing the count ever time the state machine of the relative encoders changes either count +1 or count -1. I started by using the state machine provided by the book which looks for the current state of the relative encoder and compares that to the old value of the relative encoder. In my case I used an if statement to check in what state are the relative encoders are in and following that with an embedded if statement to see what the pervious state of the relative encoders are and updating the count accordingly.

In my ISR timer4 I convert the count to degrees to be used to control the direction of the motor, the equation used I used is (360*counts)/4741. Using a global variable target_count which is updated inside the external interrupts by +90 when S5 is pressed and –90 when S4 is pressed and can be updated by the transmitted message from the UART 2 interrupt. I calculate the difference between the target angle and the angle at which the motor is currently in and storing it in diff_error, diff_error is to be used in calculating the Euler integral and to be transmitted to the MATLAB code to plot. The equation I used to calculate the Euler integral is eint = eint +e*dt, in which e is the difference between current and target angle and dt is the speed at which the timer is running at $\frac{1}{150\,Hz}$ =6.67x10^-3 (s). And, to calculate my proportional_controle I used the equation u = Kp*e + Ki*eint, Kp and Ki where given as 10 and 3.0 respectively.

Using the value of the proportional integral controller (PIC) found above I calculated my PWM by setting the value of OC4RS to the absolute value of PIC with the limit of 0-1023 (0-100% duty cycle). The PIC value can be both negative and positive depending on the value of the error. To control the direction of the motor I created conditional statements which will set F0 = 0 (M2IN2) and F1=1 (M2IN1) (moves motor counter clock wise) if the PIC is positive, set F0 = 1 and F1=0 (moves the motor clockwise) if the PIC is negative, and not move the motor at all if the degrees is 0.

In addition, I used timer2 to print on to the LCD screen at a speed of 15 Hz. Timer2 was to print out the Target angle on the first line of the LCD screen and Current angle on the second line of the LCD screen. To prevent timer 2 from over printing I implemented a conditional statement that only allows printing if this is our first time running the program, or if the values of the current or target angle have change. Also, in my external interrupts 1 and 2 which are used for switch 4 and 5 whenever the ISR is called I rest my value for error integral (eint) back to zero. The reasoning behind this is as follow whenever a switch is pressed the user is now interested in a new degree and not resting eint would make it a very large number which would lead to the motor running at the incorrect speed and go to incorrect angles.

Using the UART communication to receive and transmit messages with MATLAB I start by initializing my UART 2 portal. Setting my U2BRG to 21, setting my protocol to 8 bits, no parity, and 1 stop bit. Following that with enabling my TX and RX lines since we will be receiving and transmitting data, clearing my RX flag, setting my priority and sub priority, enabling my RX interrupt for interrupt UART2, configuring my U2TX and U2RX, and turning on my interrupt.

I also created a function to read messages and write messages. The read function takes in a char and its max length reads the data from U2RXREG until ‘\n’ or ‘\r’ is received and places that message into a char. The write function takes in an augment of a sting and places the string into U2TXREG until a ‘\0’ is reached indicating end of message to be transmitted. In my UART2Handler which is my UART ISR, I check if there are any message generated by checking the RX flag, if so I start a timer (to calculate time since message is received for until step to be transmitted back to MATLAB) , I read the message using my read_message function and using atof convert that message into a float to set it to my target angle. I also set my error_integral back to 0 here.

To transmit message, I used my timer5 to do so. Following the same ideas from timer4 and timer2 I initiate my timer5 at a frequency of 50Hz. Inside the timer I convert counts into degrees calculate my error, error_integral, and proportional controller. Using a conditional statement that states if sample_count<=100 to do the following; get the current time using CP0 timer, calculate the time difference from message received until now and multiple that by 25ns.

Using the sprintf function a place my time into timer_send (a char of 100), append it with '\r\n' to indicate end of message to the MATLAB code, and using my write function I send this message to MATLAB. I follow the same process to tansmite target_float (ref angle), angle (current angle), error_diff (ref angle – current angle), and the error_integral (Euler integral). At the end of the condition statement I increment the value of sample_count by one. And in my else statement I sent the sample_count to zero in case we want to transmit more than once. And finally, I clear my timer5 flag.

Section 3: Expected behavior

The expected behavior of this program if I did not test it in advance is that it will start by displaying the angles (current and target) on to the LCD screen as 0.00. The motor is not expected to rotate until the message from MATLAB containing the target angle is transmitted. After the message is transmitted, I expect that the motor will set its target angle to 45 degrees. It will start rotating overshoot the reference angle, then undershoot it and correct until its very close to target angle or is target angle. In the mean while timer5 will be transmitting the timer for until step, reference angle, current angle, error, and integral of error during that window. I also expect that MATLAB will draw the graph of refence angle and angle vs time with angle overshooting and undershooting. Also, I expect MATLAB to draw error and integral of error vs time with error decreasing, increasing by a fraction of the decrease, and approximating a horizontal line. On the other hand, I expect a steady increase of the integral.

Section 4: code

Main.c-

```
/*
 * File:   main.c
 * Author: amr usef
 * lab:9
 *date: 11/18/20
 * Discription: in the main file what we are dong is reciving a refernce
angle
 * (target angle), and we pass that to the motor to achive as current angle.
 *Using PID to controle speed and dirction of the motor, we try to get as
close as
 * possible to the target angle. Also, we are transmitting to the MATLAB code
 * time, reference angle, current angle, error, and integral of error. The
receving and transmition
 * is using the UART communcation.
 */


#include <stdio.h>
#include <stdlib.h>
#include <sys/attribs.h>
#include <xc.h>
#include <string.h>
#include "lcd_display_driver.h"
#define SAMPLE_TIME 10 //10 corse timer ticks = 250 ns
#define BAUD 230400
#define FREQ_SYSTEM 80000000
//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

static volatile int counts=0,count_pervous=0,bit_A_old =0,bit_B_old =0,
target_count=0, target_pervous=0,first_time_in=1, sample_count=1;//for the
counts
static volatile float timer=0,time_start=0,time_finish=0,error_integral=0,
proportional_controller=0,error_diff =0, dt = 0.00667,
core_tick=0.000000025;//dt=1/150 which is the speed that ISR timer4 running

void __ISR(_CHANGE_NOTICE_VECTOR, IPL6SOFT) CNISR(void) {
    //lcd_display_driver_clear();//clearing if anything changes
    if(PORTGbits.RG6 == 0 && PORTGbits.RG7 == 0){//is the new state AB=00
        if(bit_A_old == 0 && bit_B_old == 1){//is pervious state AB=01
            counts= counts-1;//clockwise therefore subtract one from count
        }
        else if(bit_A_old == 1 && bit_B_old == 0){//is old state AB=10
            counts= counts+1; //counter clockwsise therefore add one to count
        }
         bit_A_old = PORTGbits.RG6;//set the new states of A to old for
comparing
```

```
         bit_B_old = PORTGbits.RG7;//set the new state of B to old for 
comparing
    }

    else if( PORTGbits.RG6== 0 && PORTGbits.RG7 == 1){//is the new state 
AB=10
        if(bit_A_old == 1 && bit_B_old  == 1){//is the old state 11=AB
            counts= counts-1;//we moved clockwise subtract one
        }
        else if(bit_A_old == 0 && bit_B_old== 0){//is the old state 00
            counts= counts+1; //counter clockwise add one
        }
         bit_A_old = PORTGbits.RG6;//set the new states of A to old for 
comparing
         bit_B_old = PORTGbits.RG7;//set the new state of B to old for 
comparing
    }

    else if(PORTGbits.RG6 == 1 && PORTGbits.RG7== 1){//is the new state AB=11
        if(bit_A_old == 1 && bit_B_old == 0){//is the old state 10=AB
           counts= counts-1;//we moved clockwise subtract one
        }
        else if(bit_A_old == 0 && bit_B_old == 1){//is the old state 01
             counts= counts+1; //counter clockwise add one
        }
         bit_A_old = PORTGbits.RG6;//set the new states of A to old for 
comparing
         bit_B_old = PORTGbits.RG7;//set the new states of B to old for 
comparing
    }
    else if(PORTGbits.RG6 == 1 && PORTGbits.RG7 == 0){//is the new state 
AB=10
        if(bit_A_old == 0 && bit_B_old  == 0) {//is the old state 00=AB
            counts= counts-1; //we moved clockwise subtract one
        }
        else if(bit_A_old == 1 && bit_B_old  == 1){//is the old state 11=AB
             counts= counts+1; //counter clockwise add one
        }
         bit_A_old = PORTGbits.RG6;//set the new states of A to old for 
comparing
         bit_B_old = PORTGbits.RG7;//set the new states of B to old for 
comparing
    }
    IFS1bits.CNIF = 0;//Set interrupt flag to 0
    return;
}

void __ISR(_EXTERNAL_1_VECTOR, IPL5SOFT) s4ISR(void){
 target_count=target_count-90;//decrease by 90
 error_integral=0;//setting back our eurler integral back to zero
 IFS0bits.INT1IF=0;//set flag back to zero

}

void __ISR(_EXTERNAL_2_VECTOR, IPL5SOFT) s5ISR(void){
 target_count=target_count+90;//increase by 90
 error_integral=0;//setting back our eurler integral back to zero
 IFS0bits.INT2IF=0;//set flag back to zero
}
void __ISR(_TIMER_2_VECTOR, IPL4SOFT) Timer2ISR(void) {
    float angle; //not recommended but necessary
    float target_float=0;//to be used in the sprintf function instead of int
target count
    char degrees_store[100];//store degrees to be displayed
    char target_degree[100];//store the target angle to be used later on
    angle =(float)(360*counts)/4741.0;//Converts count-> degrees
    target_float=(float)target_count;//place target count into target float
to be displayed as a float var

    //display the target degree and angle
    if(first_time_in=1 ||counts!=
count_pervous||target_count!=target_pervous){
    sprintf(target_degree, "Target: %.2f%c", target_float,0xDF);//line one
will show this
    sprintf(degrees_store, "Current: %.2f%c", angle,0xDF);//line 2 will show
the angle stored in degrees_stored
    lcd_display_driver_clear();//clear before writing
    display_driver_use_first_line();//calling first line to write into it
    lcd_display_driver_write(target_degree, strlen(target_degree));//writing
into the first line
    display_drive_use_second_line();//calling second line
    lcd_display_driver_write(degrees_store, strlen(degrees_store));//writing
into the second line
    target_pervous=target_count;//for conditon to be checked
    count_pervous=counts;//for condation to be checked
    first_time_in++;//to be used to print on to the LCD screen on first run
in timer 2 ;
    }
    IFS0bits.T2IF = 0;//clear timer interrupt flag
    return;
}
void reading_message(char * message, int maxLength){
    char data =0;
    int complete=0, num_bytes=0;
    while(!complete){
        if(U2STAbits.URXDA){//if data is avilable
            data = U2RXREG; //read data
            if((data=='\n')||data=='\r'){
                complete=1;
            }
            else{
                message[num_bytes]=data;
                ++num_bytes;
                //roll over if array is too small
                if(num_bytes>=maxLength){
                    num_bytes=0;
                }
            }
        }
    }
    //end string
    message[num_bytes]='\0';
}
void write_message(const char * string){
    while(*string!= '\0'){
        while(U2STAbits.UTXBF){
            ;//wait until TX buffer isn't full
        }
        U2TXREG=*string;
        ++string;
    }
}

void __ISR(_TIMER_4_VECTOR, IPL4SOFT) Timer4ISR(void) {
    float proportional_gain=10.0;        //given
    float integral_gain =1.0;           //given
    float angle=0;                      //not recommended but necessary
    float target_float=0;               //will be used to calculate
error_diff using a float
    float u=0;                          //proportional_controller to be used
to control dirction

    angle =(float)(360.0*counts)/4741.0;//Converts count-> degrees
    target_float=(float)target_count;//place target count into target float
to be displayed as a float var
    error_diff =target_float-angle;//for correction
    error_integral= error_integral + (error_diff*dt); //error sum over time=
eint=eint+e*dt
    proportional_controller=(proportional_gain*error_diff)
+(integral_gain*error_integral);//u=kp*e+ki*eint
    u=proportional_controller;//used in dirction control

    //speed
    if(proportional_controller >1023){//if proportional_controller is greater
than 1023 set it back to 1023
        proportional_controller =1023;//setting back here
    }

    OC4RS = abs(proportional_controller);//set the PWM duty cycle(aka speed)

    //dirction
    if(u<0){//increase the number of counts
        LATFbits.LATF0=0;//F0=0
        LATFbits.LATF1=1;//F1=1
    }
    else if(u>0){//decrease number of counts
        LATFbits.LATF0=1;//F0=1
        LATFbits.LATF1=0;//F1=0
    }
    else if (abs(u)==0){//do nothing
        LATFbits.LATF0=0;//F0=0
        LATFbits.LATF1=0;//F1=0
    }

    IFS0bits.T4IF = 0;//clear timer interrupt flag
    return;
}
void __ISR(_UART_2_VECTOR, IPL3SOFT) UART2Handler(void){
    if(IFS1bits.U2RXIF){//check if interrupt generated by RX event
       time_start = (float)_CP0_GET_COUNT();//start recording time when
message is recived
```

```
        char temp[100];//to place the message
        reading_message(temp,100);
        target_count= atof(temp);//convert from string to float the reference 
angle
        error_integral=0;//setting back our eurler integral back to zero
        LATAbits.LATA0=1;
    }else if(IFS1bits.U2ATXIF){//if it is a TX interrupt

    }else if(IFS1bits.U2EIF){//if it is an error interrupt

    }
}
void __ISR(_TIMER_5_VECTOR, IPL4SOFT) Timer5ISR(void) {
    float angle=0;                      //not recommended but necessary
    float proportional_gain=10.0;       //given
    float integral_gain =1.0;           //given
    float target_float=0;               //will be used to calculate 
error_diff using a float
    char timer_send[100];               //store data to be transmitted to 
matlab
    char ref_angle_send[100];           //store reference angle to send to 
matlab
    char angle_send[100];               //sore current angle to send to 
matlab
    char error_diff_send[100];          //store error to send to matlab
    char error_integral_send[100];      //store the integral to send to 
matlab

    angle =(float)(360.0*counts)/4741.0;//Converts count-> degrees
    target_float=(float)target_count;//place target count into target float 
to be displayed as a float var
    error_diff =target_float-angle;//for correction
    error_integral= error_integral + (error_diff*dt); //error sum over time= 
eint=eint+e*dt
    proportional_controller=(proportional_gain*error_diff)
+(integral_gain*error_integral);//u=kp*e+ki*eint

    if(sample_count<=100){
        time_finish=(float)_CP0_GET_COUNT();//current time after message is 
recived
        timer=(float)(time_finish-time_start)*core_tick;//from time message 
is recived until now
        sprintf(timer_send,"%0.4f\r\n",timer);
        write_message(timer_send);
        sprintf(ref_angle_send,"%0.4f\r\n",target_float);
        write_message(ref_angle_send);
        sprintf(angle_send,"%0.4f\r\n",angle);
        write_message(angle_send);
        sprintf(error_diff_send,"%0.4f\r\n",error_diff);
        write_message(error_diff_send);
        sprintf(error_integral_send,"%0.4f\r\n",error_integral);
        write_message(error_integral_send);
        LATAbits.LATA1=1;
        sample_count++;
    }else{
        sample_count=0;
    }
    IFS0bits.T5IF = 0;//clear timer interrupt flag
    return;
}

int main(void) {
    TRISA = 0xFF00; //output the last 8 LEDS
    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
    lcd_display_driver_initialize();//calling to initialize the lcd display

    TRISDbits.TRISD13=1;//S4 as input
    TRISAbits.TRISA7=1;//S5 as input

    TRISGbits.TRISG6=1;//CN 8 as input
    TRISGbits.TRISG7=1;//CN 9 as input
    TRISFbits.TRISF0=0;//F0 as an ouput
    TRISFbits.TRISF1=0;//F0 as an ouput

    INTCONbits.MVEC = 1;//allow many interrputs


    __builtin_disable_interrupts();
    //S5 interrupt
    //change prority for one interrput
    IPC1bits.INT1IP = 5; //Set Priority 5
    IPC1bits.INT1IS = 1; //Set Sub Priority 1
    IFS0bits.INT1IF = 0; //interrupt flag 0
    IEC0bits.INT1IE = 1; //Enable the interrupt
   //S4 interrupt set up
    IPC2bits.INT2IP = 5; //Set Priority 5
    IPC2bits.INT2IS = 1; //Set Sub Priority control for to 1 for external
interrupt 0 which is connected to S5
    IFS0bits.INT2IF = 0; //Set the interrupt flag back to 0
    IEC0bits.INT2IE = 1; //Enable the interrupt to control s5.


    CNCONbits.ON = 1; //turns on CN.
    CNENbits.CNEN8 = 1; //use as change notification CN8
    CNENbits.CNEN9 = 1; //use as change notification CN9

    IPC6bits.CNIP = 6;//Set Priority 6
    IPC6bits.CNIS = 1;//Set Sub Priority 1
    IFS1bits.CNIF = 0;//interrupt flag 0
    IEC1bits.CNIE = 1; //enables the CN interrupt.

    PR2 = 20832;//1/15=(PR2+1)*256*1/(80*10^6)=20832
    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 < CN priority
    IPC2bits.T2IS = 0;//sub-priority
    IFS0bits.T2IF = 0;//clears flag
```

```
  IEC0bits.T2IE = 1;//enables interrupt

  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= 1023;//set to be used with potenitmeter
  OC4RS = 1023;//set to be used with potenitmeter
  T3CONbits.ON = 1;//turn on timer
  OC4CONbits.ON = 1;//turn on timer


  //533333=(PR4+1)*256=2082
  PR4 = 2082;//1/150=(PR4+1)(256)(1/80*10^6) => PR4 = 2082.33
  TMR4 = 0;//initializes TMR4 to zero
  T4CONbits.TCKPS = 7;//prescaler- 256
  T4CONbits.TGATE = 0;//Gatted timer off
  T4CONbits.TCS = 0;//default
  T4CONbits.ON = 1;//turn on timer
  IPC4bits.T4IP = 4;//priority 4 < CN priority
  IPC4bits.T4IS = 0;//sub-priority 0
  IFS0bits.T4IF = 0;//clear flag -0
  IEC0bits.T4IE = 1;//enables interrupt


  PR5 = 6249;//1/50=(PR5+1)(256)(1/80*10^6) => PR5 = 6249
  TMR5 = 0;//initializes TMR5 to zero
  T5CONbits.TCKPS = 7;//prescaler- 256
  T5CONbits.TGATE = 0;//Gatted timer off
  T5CONbits.TCS = 0;//default
  T5CONbits.ON = 1;//turn on timer
  IPC5bits.T5IP = 4;//priority 4 < CN priority
  IPC5bits.T5IS = 0;//sub-priority 0
  IFS0bits.T5IF = 0;//clear flag -0
  IEC0bits.T5IE = 1;//enables interrupt

  //UART initialization
  U2MODEbits.BRGH=0;
  U2BRG =((FREQ_SYSTEM /BAUD)/16)-1;
  //8 bits, no parity, and 1 stop bit
  U2MODEbits.PDSEL=0;
  U2MODEbits.STSEL=0;
  U2STAbits.UTXEN=1;//enable TX
  U2STAbits.URXEN=1;//enable RX
  IFS1bits.U2RXIF=0;//clearing the interrupt RX flag
  IPC8bits.U2IP=3;//prority
  IPC8bits.U2IS=0;//sub-prority
  IEC1bits.U2RXIE=1;//enable  the RX interrupt
  U2MODEbits.UEN=0;//only U2TX and U2RX pins to be used
  U2MODEbits.ON=1; //turn on UART
  __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<=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
}

  void display_driver_use_first_line(){
     TRISBbits.TRISB15 =0;//set RS to be output
     TRISDbits.TRISD5 =0; //reasure that R/W is output
     LATBbits.LATB15 = 0;//RS=0
     LATDbits.LATD5 = 0;//RW =0
     LATE = 0x80;
     lcd_display_driver_enable();

  }
  void display_drive_use_second_line(){
     TRISBbits.TRISB15 =0;//set RS to be output
     TRISDbits.TRISD5 =0; //reasure that R/W is output
     LATBbits.LATB15 = 0;//RS =0
     LATDbits.LATD5 = 0;//RW =0
     LATE = 0xC0;
     lcd_display_driver_enable();
 }
```