MICROCONTROLLER PROJECT REPORT

Ayana Tleulenova 160412059

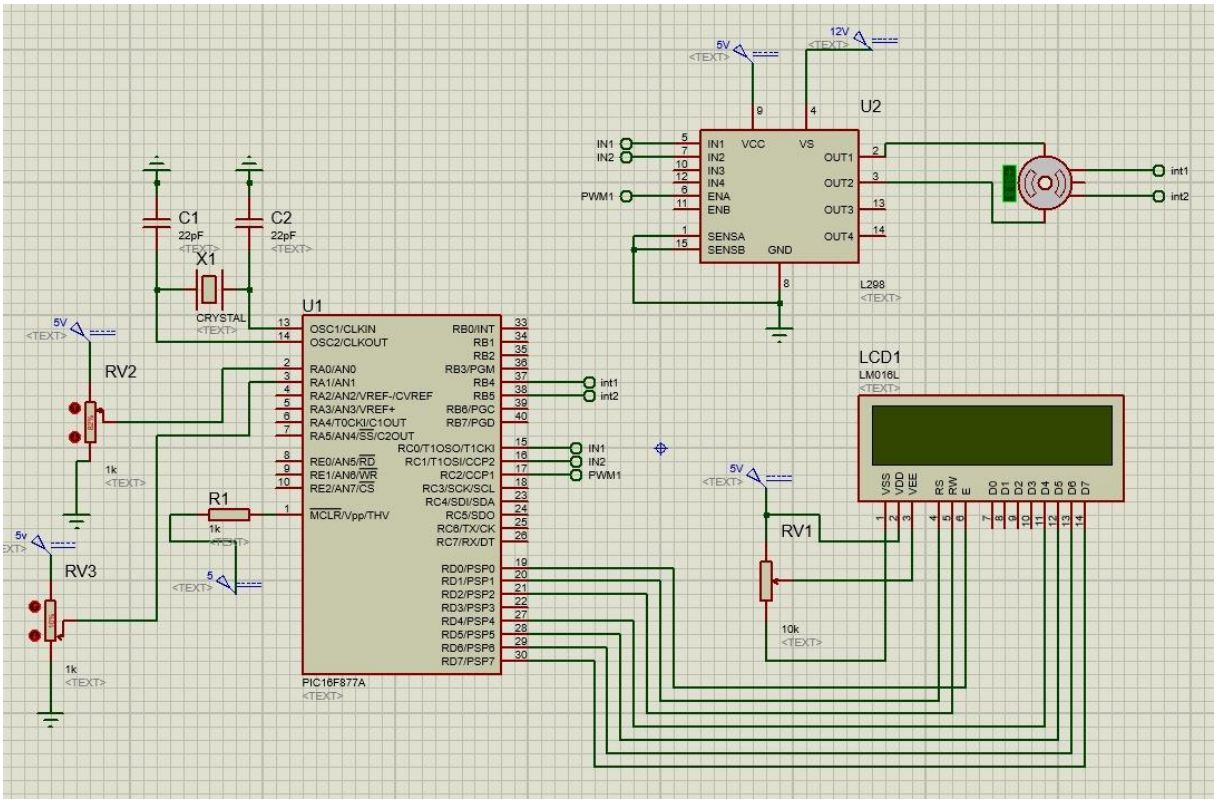
OVERVIEW OF THE PROJECT

The task of the project is to control the position of a dc motor with given inputs from the user by using PIC microcontroller. The inputs are proportional gain parameter Kp and a reference angle. First input is Kp parameter and it is received by serial communication. We achieve this serial communication with another PIC using the Rx and Tx pins, also controller does not execute any other code and action until Kp input value is received. Second input, reference angle, is obtained from an analog voltage divider reading and converted by using ADC converter module of the microcontroller. Then based on the reference angle ,and another input ,motor encoder input which is used to calculate momentary position, the position error is calculated (starting position of the motor is always 0 degrees). Lastly , the shorter direction to rotate is calculated.

After all the reading and calculations, I get output a pwm signal to the motor enable pin to set its speed with a certain duty cycle based on a function of Kp and error. I set the input pins of motor driver low or high according to the calculated shorter direction of rotation. When the desired position is reached and there is no more error, motor is stopped.

PROCEDURE OF THE PROJECT

1. Electrical Curcuit:



2. Code

PIC Code :

```
#include <16F877A.h>
#device adc=10 // set adc module conversion range as 10bit
#FUSES XT, NOWDT, NOPROTECT, NOBROWNOUT, NOLVP, NOPUT, NODEBUG, NOCPD
#use delay(crystal=4000000)
#include <lcd.c>

int8 last_read; //variable to check changing encoder value
signed int8 quad = 0; //variable to change position variable
#INT_RB // RB port interrupt on change
void RB_IOC_ISR(void) //interrupt function for every encoder read
{
 int8 encoderRead; // variable to be used in motor encoder reading
 clear_interrupt(INT_RB);
 encoderRead = input_b() & 0x30;  //reading the encoder input
 if(encoderRead == last_read)     // check if the position is changed
  return;  // if the position is not changed ,do nothing
 //checking the direciton of the change
 if(bit_test(encoderRead, 4) == bit_test(last_read, 5))  //to determine the rotation
  quad -= 1;       //if the rotation is ccw add one to quad variable
 else
  quad += 1;       //if the rotation is cw subtract one from quad variable
 last_read = encoderRead;    //store the latest read
}
int16 EncoderGet(void) {
//function for converting readed values to position change
  signed int16 value = 0;          //reset the change to zero
```

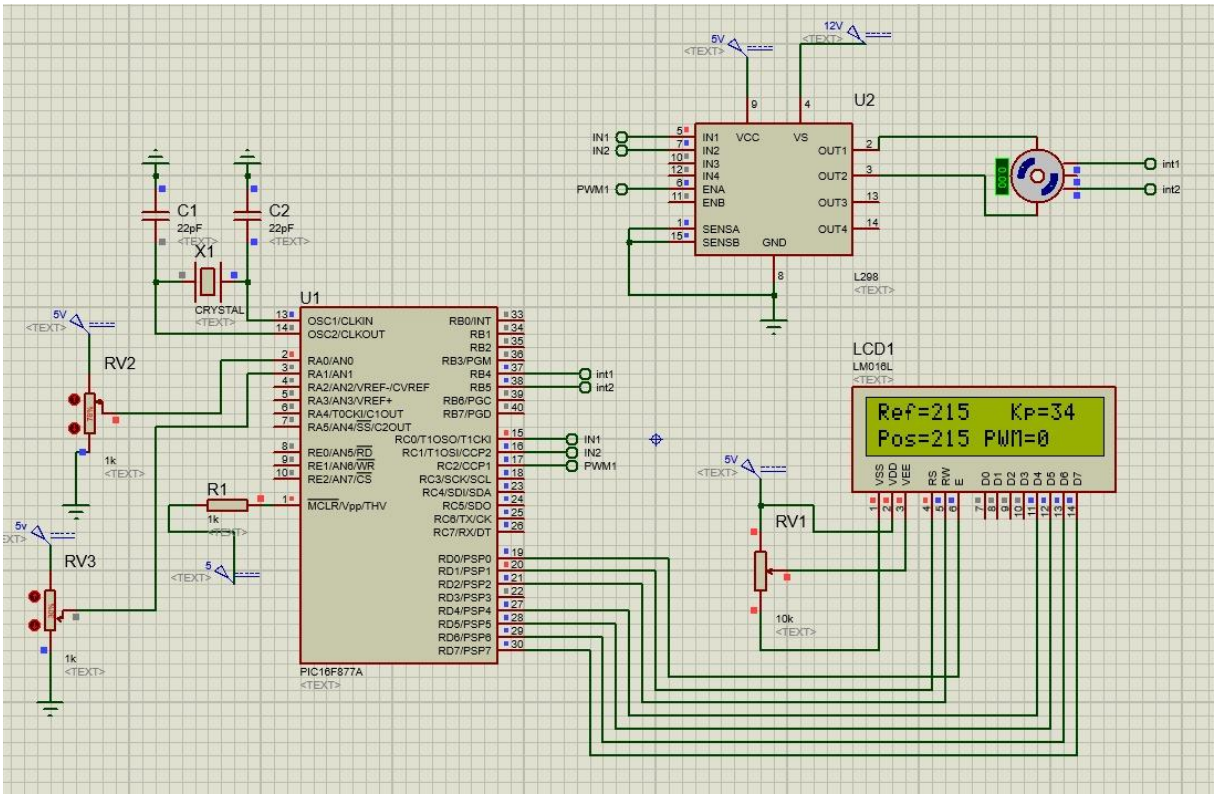
```
 //returning directional value
 while(quad >= 4){    //if quad is 4, 1 pulse period is done in ccw direction

   value += 1;  //add 1 to value to store change in the number of pulses or angles
   quad -= 4;   //reset quad to zero

 }
 while(quad <= -4){    //if quad is -4, 1 pulse period is done in cw direction
   value -= 1;         //subtract 1 from value to store change in the number of pulses of angles
   quad += 4;      // reset quad to zero
 }
 return value;    // function returns the value of change in the angle
}
int16 PWM(int16 a,int16 b) //function for calculating PWM value
{
 signed int16 value;
 value=a*b; //will get error and Kp values as inputs
 //bounding the PWM value
 if (value>1023)
  return 1023;
 else if (value<0)
  return 0;
 else
  return value;
}

void main()
{
 //variables
 int16 refangle;
 int16 analog_controller;
 int default_kp = 10;
```

```
int V = 80;
int16 realPosition = 0;
int16 error = 0;

lcd_init();
delay_ms(10);
set_tris_c(0x00000000); //setting the outputs
output_c(0x00);

setup_ccp1(CCP_PWM); //setuping the PWM
setup_timer_2(T2_DIV_BY_16, 255, 1);

setup_adc_ports(AN0_AN1_AN3); //setuping the analog reading
setup_adc(ADC_CLOCK_DIV_32);

enable_interrupts(INT_RB);
enable_interrupts(GLOBAL);
clear_interrupt(INT_RB);
while(True)
{
 //analog reading
 set_adc_channel(0);
 delay_us(10);
 refangle = read_adc();
 //converting readed value to wanted variable
 int16 reference_angle =((float)refangle*250/1023)+20;
 set_adc_channel(1);
 delay_us(10);
 analog_controller = read_adc();
 int16 control_gain = default_kp +
 ((float)analog_controller/1023)*V;
```

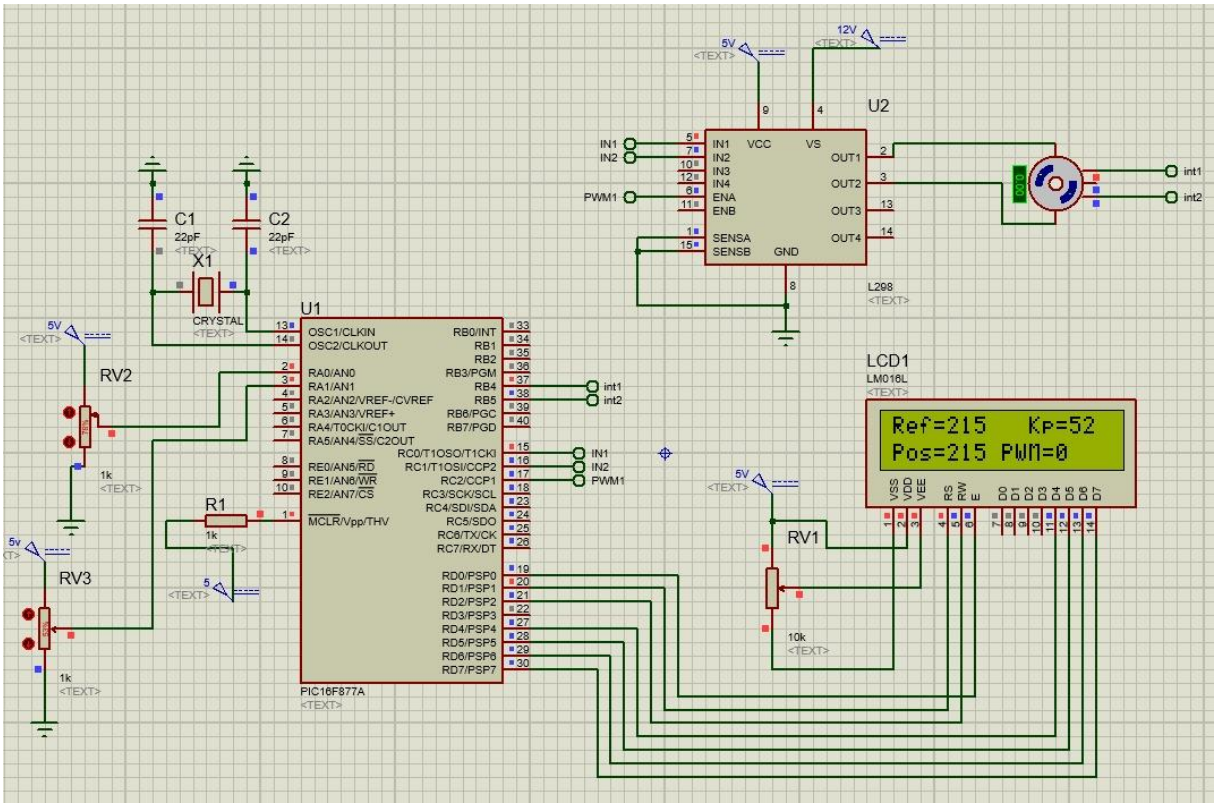
```
 long change = EncoderGet(); //getting position change from encoder
 if(change)
 {
  realPosition += change; //updating the position
 }
 float rev = realPosition/360.0f;
 int16 angle = ((int16)(rev*360))%360;
 //getting error direction and starting the motor and its velocity accordingly
 if(angle<=reference_angle)
  {
   error = reference_angle-angle;
   set_pwm1_duty(PWM(control_gain,error));
   output_high(PIN_C0);
   output_low(PIN_C1);
  }
 if(angle>reference_angle)
  {
   error = angle-reference_angle;
   set_pwm1_duty(PWM(control_gain,error));
   output_low(PIN_C0);
   output_high(PIN_C1);
  }
 //printing desired values on LCD
 printf(lcd_putc, "\fRef=%lu   ", reference_angle);
 printf(lcd_putc, "Kp=%lu", control_gain);
 printf(lcd_putc, "\nPos=%lu ", realPosition);
 printf(lcd_putc, "PWM=%lu",PWM(control_gain,error));
 delay_ms(10);
 }
}
```

3. Simulation Snapshots:

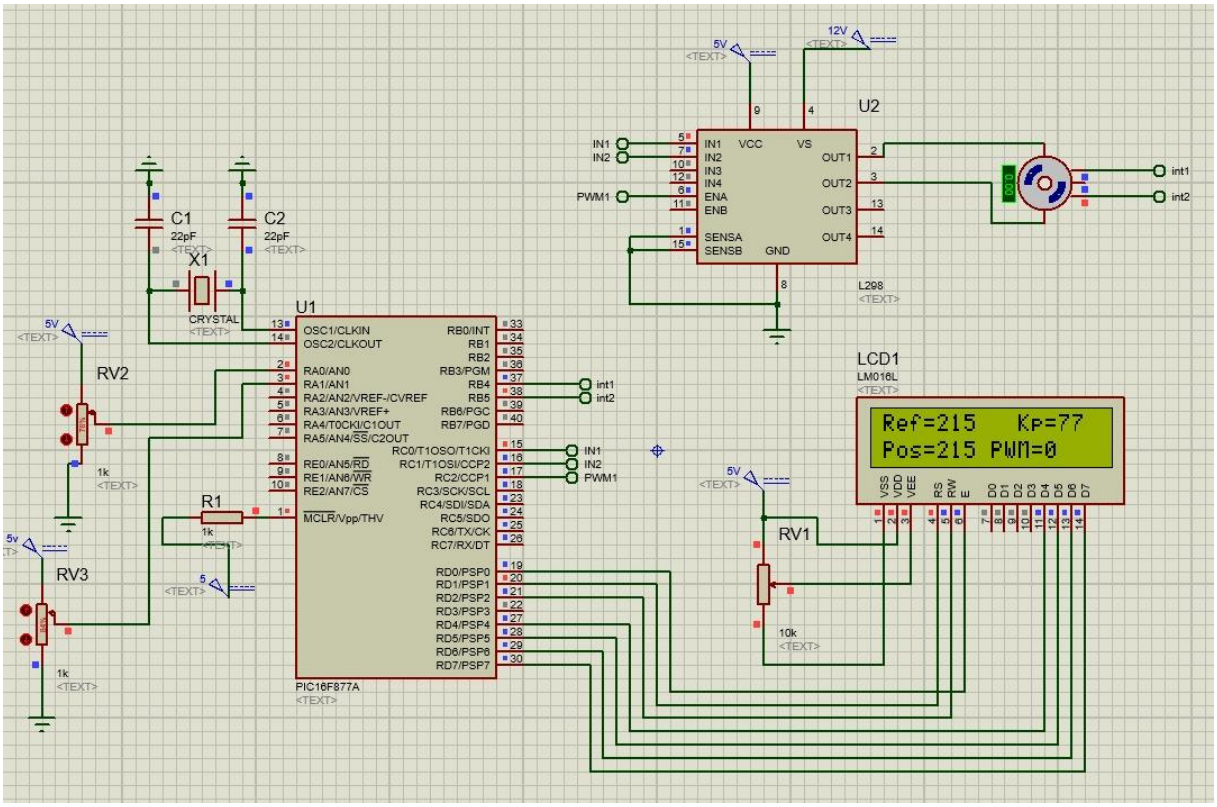
While Kp=34:



While Kp=52:



While Kp=77:


5V
12V
U2
IN1
IN2
PWM1
VCC
VS
OUT1
OUT2
OUT3
OUT4
SENSA
SENSB
GND
L298
int1
int2
C1
22pF
C2
22pF
X1
CRYSTAL
U1
OSC1/CLKIN
OSC2/CLKOUT
RA0/AN0
RA1/AN1
RA2/AN2/VREF-/CVREF
RA3/AN3/VREF+
RA4/T0CKI/C1OUT
RA5/AN4/SS/C2OUT
RE0/AN5/RD
RE1/AN6/WR
RE2/AN7/CS
MCLR/Vpp/THV
RB0/INT
RB1
RB2
RB3/PGM
RB4
RB5
RB6/PGC
RB7/PGD
RC0/T1OSO/T1CKI
RC1/T1OSI/CCP2
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RD0/PSP0
RD1/PSP1
RD2/PSP2
RD3/PSP3
RD4/PSP4
RD5/PSP5
RD6/PSP6
RD7/PSP7
PIC16F877A
RV2
RV3
RV1
R1
1k
10k
LCD1
LM016L
Ref=215 Kp=77
Pos=215 PWM=0
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7

CONCLUSION

From this project I understood that I can change position of a motor in real time with basic voltage input using PIC. I could do that due to the varied modules in the microcontroller. I used ADC module, PWM module, RS 232 module that I learned to use throughout the semester. I also laearned how to implement encoder information in applications.

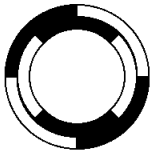
After carrying out the serial communication with other PIC only once, the main codes starts to execute. Firstly ADC module reads the analog input which is converted into reference angle value between 20-270 degrees with this function:

$$Q_{ref} = analogValue * \frac{250}{1023} + 20$$

Then I get the encoder readings and with appropriate codes convert it into position information :

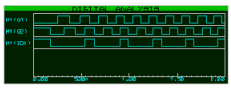
I can use the help of the proteus. In the picture ,there are 2 pulses for each sensor. We have 360. That means each pulse change is 1 degree change. Also, it is clear that at every one fourth of a pulse , encoder information changes. I store these changes in the quad variable. Every four quad means one pulse and one degree angle. For the rotation detection, I used characterisitc changes in encoder value for each rotation with an elegant function.

It is quite common in electro-mechanical applications to attach a position encoder to the shaft of a DC motor. Such devices typically consist of a wheel marked with two concentric patterns as shown below, and two optical sensors.

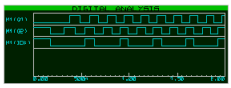


In some cases, there are more alternating black and white segments in each ring giving a higher angular precision, and also there can be a third ring containing just one black segment to indicate the zero position.

When the motor spins, the black and white marks pass the two photo-sensors and this results in two square wave patterns at the encoder outputs, as shown below on the top two waveforms. In this case, the motor is rotating anti-clockwise, a black mark generates a logic 1 and a white mark generates a logic 0.



The clever part is shown in what happens when the motor rotates in the opposite direction (clockwise), as shown below:



If you study these two diagrams carefully, you will see that in the first case, rising edges of Q1 occur when Q2 is low, but in the second case, rising edges of Q1 occur when Q2 is high. This means that a device (typically a microcontroller) attached to the Q1 and Q2 outputs can determine both the rotational speed and the absolute angular position of the motor shaft, even if it rotates both clockwise and anti-clockwise.

According to this encoder value the error is calculated and based on the error we set the duty cycle of the PWM value with this function :

$$pwm\_duty\_cycle = Kp * error$$

Effect of Different Kp values:

$$pwm\_duty\_cycle = Kp * error$$

This means that as the error gets smaller, motor rotates slower, results in a more stable system. In addition , as Kp get gets bigger,motor willl rotate faster (as I showed on video

simulation when Kp=77). Here, I came to the comclusion that, by the time error is zero and speed is set to zero, it would have rotated pass the reference value.

Moreover the second potentionmeter replies for the alter the value of the P-controller gain. A default value of Kp that defined and according to potentiometer value it must be changed as below:

$$K_p = K_p^{Default} + analogueValue \times \frac{V}{1023}$$

where V is the maximum defined variance.

The max duty cycle value is 1023. It is calculated like :

$$T_{pwm} = T_{osc} * 4 * (PR2 + 1) * (TMR2_{prescaler})$$

Finally , while doing this project, I learned which variables types are appropriate for which task in coding. Also I learned about signed and unsigned varaibles.