Lab No. : 01

Experiment Name: Blinking LED

Objective: The primary objective of this experiment is to understand the basic principles behind LED (Light Emitting Diode) blinking, explore the theory behind it, and demonstrate the practical application through hands-on experimentation.

Theory: Light Emitting Diodes (LEDs) are semiconductor devices that emit light when an electric current passes through them. The basic theory behind LED blinking involves the manipulation of the electric current to create a periodic on-and-off cycle, resulting in the characteristic blinking effect. LEDs are polarized components, meaning they have a positive (anode) and negative (cathode) lead. When a voltage is applied across the anode and cathode, electrons move through the semiconductor material, releasing energy in the form of light.

The blinking of an LED can be achieved by controlling the flow of electric current using a microcontroller or other electronic components. The simplest method involves turning the LED on and off at regular intervals, creating a flashing or blinking pattern.

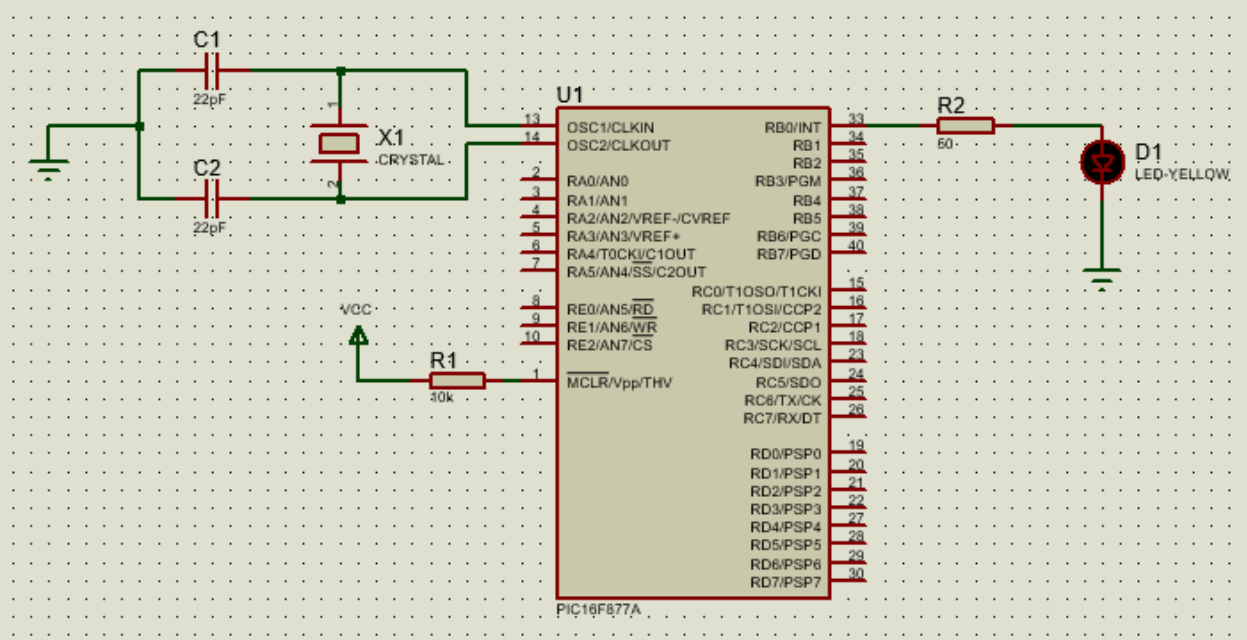


Fig: Blinking LED

Code:

```
void main() {
      TRISB=0x00;  //b as output
      PORTB=0x00; //bo off
      while(1){
          portb=0xff;
          delay_ms(200);
          portb=0x00;
          delay_ms(200);
      }
}
```

Lab No. : 02

Experiment Name: Single digit 7 segment LED design

Objectives: To design and implement a single-digit 7-segment LED display to represent digits 0 through 9. The purpose of this experiment is to understand how a 7-segment display works and how to control it using digital logic or a microcontroller.

Theory: 7-segment LED (Light Emitting Diode) or LCD (Liquid Crystal Display) type displays provide a very convenient way of displaying information or digital data in the form of numbers, letters, or even alpha-numerical characters. Typically, 7-segment displays consist of seven individual colored LEDs (called the segments), with one single display package. In order to produce the required numbers or HEX characters from 0 to 9 and A to F respectively, on the display, the correct combination of LED segments need to be illuminated, and BCD to 7-segment Display Decoders such as the 74LS47 just do that.

A standard 7-segment LED display generally has 8 input connections, one for each LED segment and one that acts as a common terminal or connection for all the internal display segments. Some single displays have also an additional input pin to display a decimal point in their lower right or left-hand corner.

In electronics, there are two important types of 7-segment LED digital display:

- *The Common Cathode Display (CCD):* In the common cathode display, all the cathode connections of the LEDs are joined together to logic “0” or ground. The individual segments are illuminated by the application of a “HIGH”, logic “1” signal to the individual Anode terminals.
- *The Common Anode Display (CAD):* In the common anode display, all the anode connections of the LEDs are joined together to logic “1” and the individual segments are illuminated by connecting the individual Cathode terminals to a “LOW”, logic “0” signal.

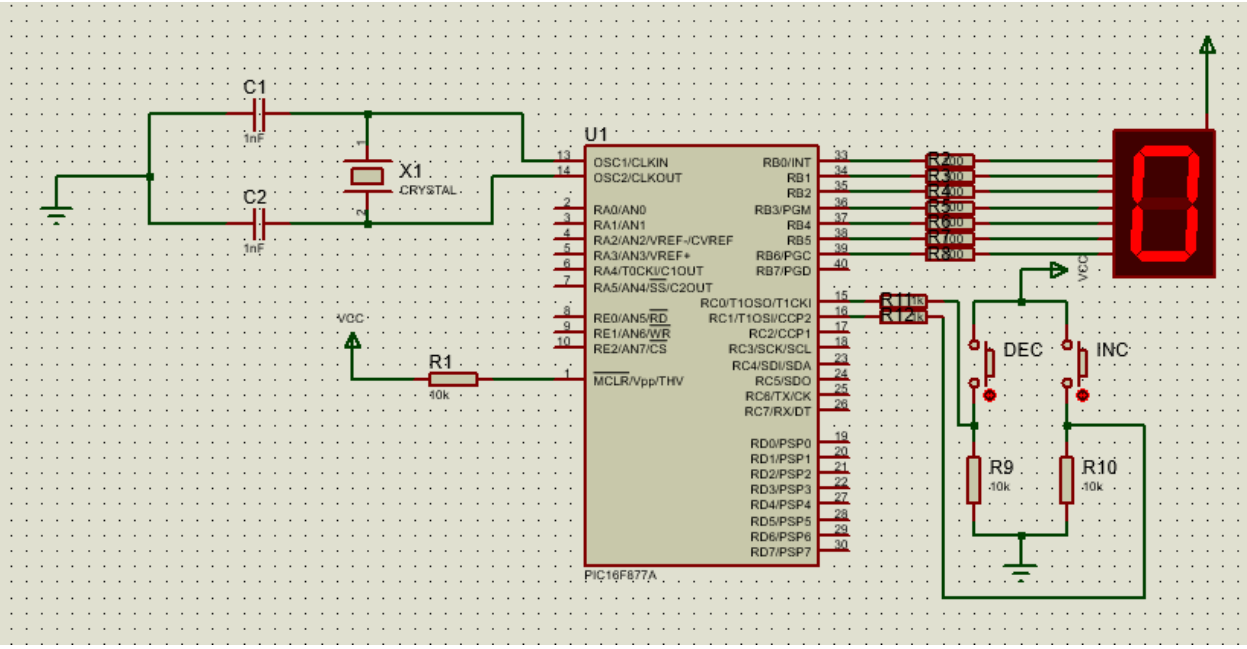


Fig: Single digit 7 segment LED design

Code:

```
char digits[]={0xC0, 0xF9, 0xA4,
0xB0, 0x99, 0x92, 0x82, 0xF8,
0x80, 0x90};

void main() {

    int i=0;

    const int delay=150;

    TRISB=0x00; //output

    TRISC=0xff; //input

    portb=0xC0;

    while(1){

              //dec

           if(portc.f0==0xff){
           delay_ms(delay);

            if(portc.f0==1){
           if(i>0) i--;
           portb=digits[i];
           }

              }
```

```
              //inc

              else
if(portc.f1==0xff){


delay_ms(delay);


if(portc.f1==1){


if(i<9) i++;


portb=digits[i];

                        }

              }

     }

}
```

Experiment Name: 4 digit 7 segment LED design

Objectives: The objective of this lab is to understand the operation and design of a 4-digit 7-segment LED display. The goal is to create a functional circuit using a microcontroller or a decoder IC to control the display, enabling it to show numbers from 0 to 9999. Multiplexing techniques will be applied to manage all four digits using minimal pins, ensuring efficient control. Additionally, the experiment will explore timing and refresh rate requirements to ensure smooth, flicker-free operation. The practical applications of 4-digit displays, such as digital clocks and counters.

Theory: The seven segments were driven individually through separate I/O pins of the microcontroller. If we do just like that then for 4 seven segment LED displays, 28 I/O pins will be required, which is quite a bit of resources and is not affordable by mid-range PIC microcontrollers. That's why a multiplexing technique is used for driving multiple seven segment displays. This tutorial shows how to multiplex 4 common anode type seven segment LED displays with a PIC16F877A microcontroller.

The theory behind the multiplexing technique is simple. All the similar segments of multiple LED displays are connected together and driven through a single I/O pin. In the circuit below, the seven segments are connected to PORTB through current limiting resistors Rs. A particular segment is active when the corresponding PORTB pin is low. However, it will not glow until it's anode is connected to Vcc. You can see the anodes of the four LED displays are not directly connected to Vcc. Instead, 4 PNP transistors are used as switches to connect or disconnect the anode terminals from Vcc. When the base of the PNP transistor is low, the transistor conducts and corresponding digit's common anode is connected to Vcc. Therefore, the transistor selects which displays is active. The conduction of the transistors are controlled by RA0 through RA3 pins of PORTA. Suppose, if we want to display 7 in the units digit place, then segments a, b, and c should be turned on first (which means RB0, RB1, RB2 are 0 and RB3-RB6 are 1) and then RA0 should be pulled low (while keeping RA1-RA3 high) so that only units digit display will be active. In order to display all 4 digits, each seven-segment display is activated sequentially using an appropriate refresh frequency so that it will appear that all the them are turned on at the same time.

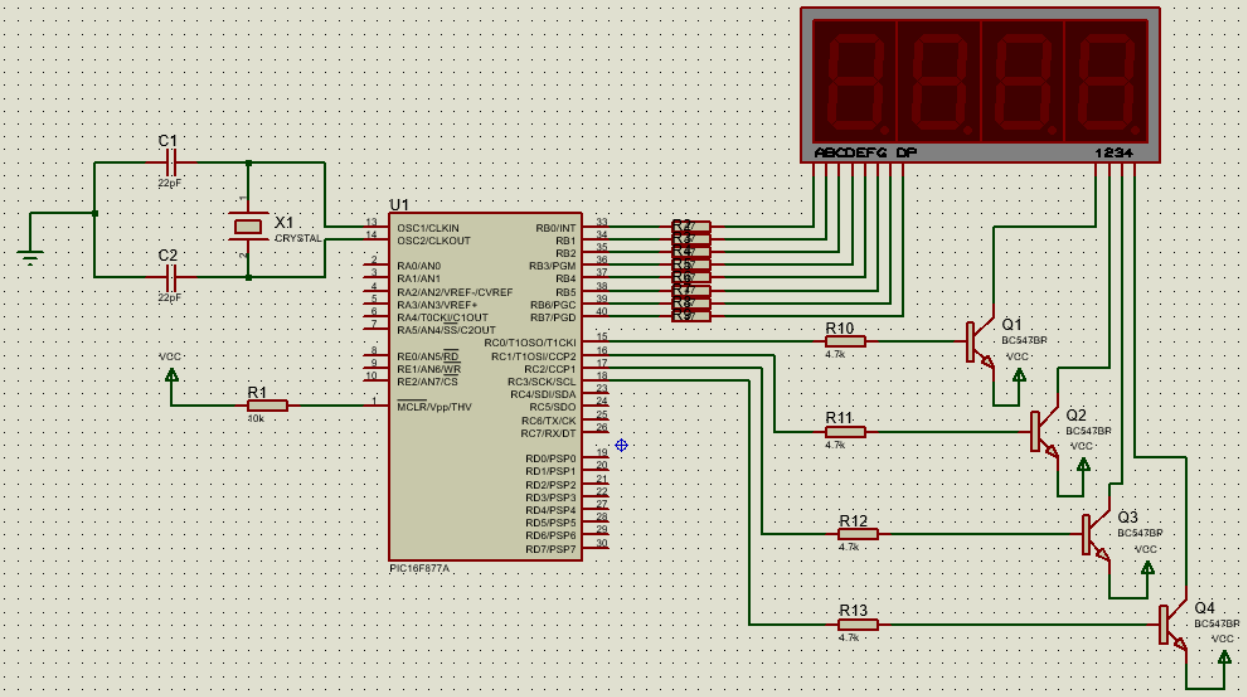


Fig: 4 digit 7 segment LED design

Code:

```
char digits[]={0xC0, 0xF9, 0xA4,
0xB0, 0x99, 0x92, 0x82, 0xF8,
0x80, 0x90};

void main(){

    const int delay=5;

    int i, j, digit[4];

    TRISB=0x00;//out

    TRISC=0x00;

    portb=0xff;

    portc=0x00;

    for(i=0;i<=9999;i++){

        digit[0]=i/1000;

        digit[1]=(i/100)%10;

        digit[2]=(i/10)%10;

        digit[3]=i%10;
```

```
        for(j=0;j<50;j++){

            portc.f0=0xff;
portb=digits[digit[0]];

            delay_ms(delay);

            portc.f0=0x00;

            portc.f1=0xff;
portb=digits[digit[1]];

            delay_ms(delay);

            portc.f1=0x00;

            portc.f2=0xff;
portb=digits[digit[2]];

            delay_ms(delay);

            portc.f2=0x00;

            portc.f3=0xff;
portb=digits[digit[3]];

            delay_ms(delay);
```

```
            portc.f3=0x00;
        }
    }
}
```

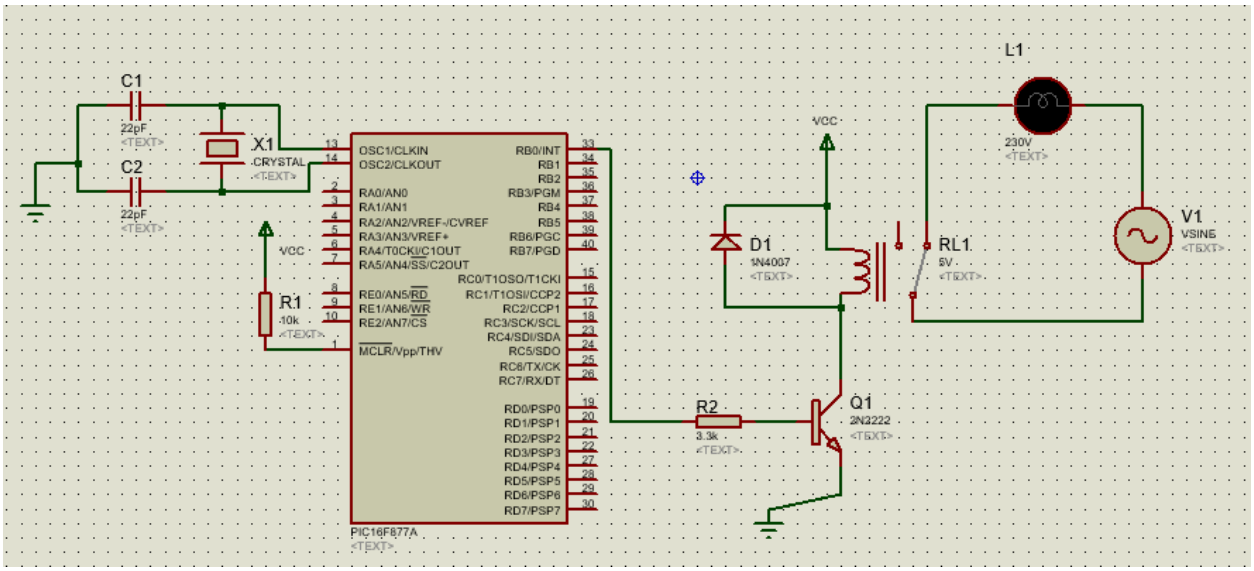
Lab No. : 03

Experiment Name: Controlling AC Current Using DC Current

Objectives: To design and implement a circuit to control the flow of AC current using a DC control signal. The primary aim is to understand the interaction between AC and DC currents and to explore components like relays.

Theory: The seven segments were driven individually through separate I/O pins of the microcontroller. If we do just like that then for 4 seven segment LED displays, 28 I/O pins will be required, which is quite a bit of resources and is not affordable by mid-range PIC microcontrollers. That's why a multiplexing technique is used for driving multiple seven segment displays. This tutorial shows how to multiplex 4 common anode type seven segment LED displays with a PIC16F877A microcontroller.

The theory behind the multiplexing technique is simple. All the similar segments of multiple LED displays are connected together and driven through a single I/O pin. In the circuit below, the seven segments are connected to PORTB through current limiting resistors Rs. A particular segment is active when the corresponding PORTB pin is low. However, it will not glow until it's anode is connected to Vcc. You can see the anodes of the four LED displays are not directly connected to Vcc. Instead, 4 PNP transistors are used as switches to connect or disconnect the anode terminals from Vcc. When the base of the PNP transistor is low, the transistor conducts and corresponding digit's common anode is connected to Vcc. Therefore, the transistor selects which displays is active. The conduction of the transistors are controlled by RA0 through RA3 pins of PORTA. Suppose, if we want to display 7 in the units digit place, then segments a, b, and c should be turned on first (which means RB0, RB1, RB2 are 0 and RB3-RB6 are 1) and then RA0 should be pulled low (while keeping RA1-RA3 high) so that only units digit display will be active. In order to display all 4 digits, each seven-segment display is activated sequentially using an appropriate refresh frequency so that it will appear that all the them are turned on at the same time.



Code:

```
void main() {
    TRISB=0x00;
    portb=0x00;
    while(1)
    {
        PORTB.F0=1;
        delay_ms(1000);
        PORTB.F0=0;
        delay_ms(1000);
    }
}
```

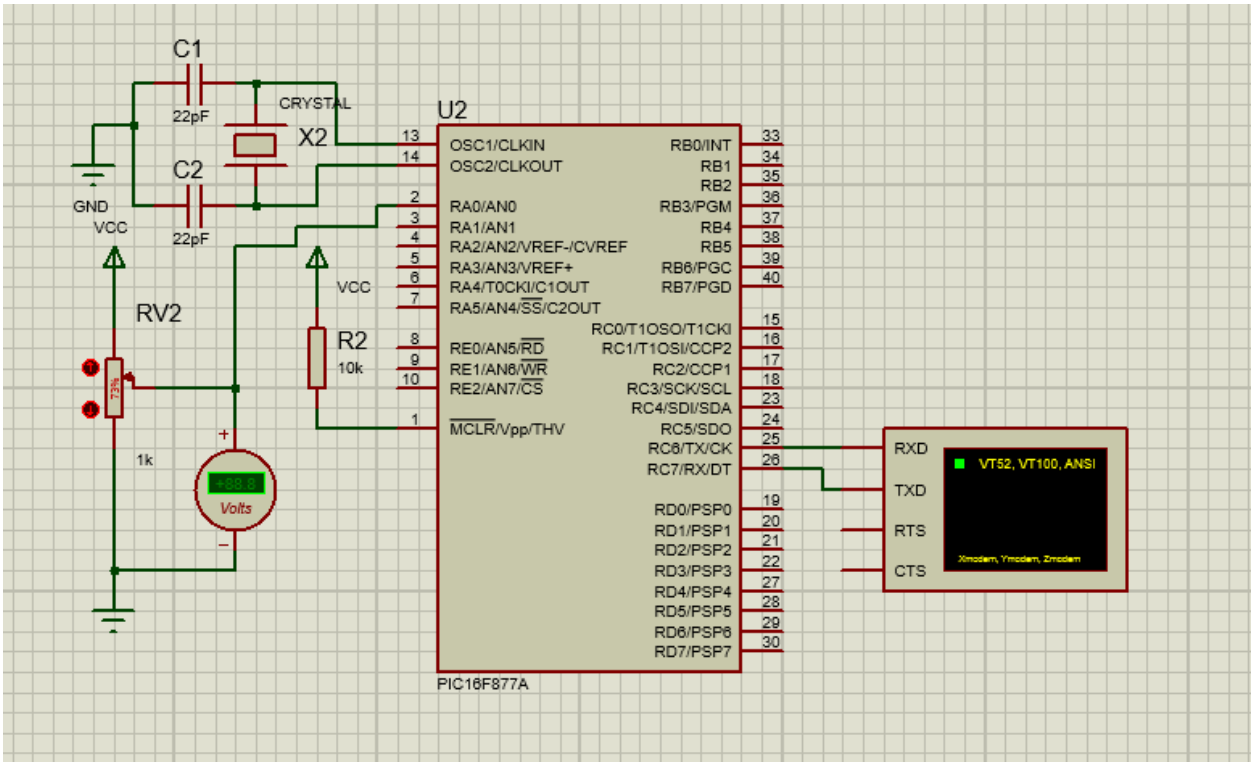
Lab No. : 04

Experiment Name: Analog-to-Digital Converter (ADC) Interfacing with Microcontroller

Objectives: To understand the concept of analog-to-digital conversion (ADC) and its importance in microcontroller applications. To interface an ADC with a microcontroller and read analog sensor data. And, to analyze the relationship between the analog sensor input and the corresponding digital output.

Theory: Analog-to-digital conversion (ADC) is the process of converting analog signals into digital representations. This is essential for microcontrollers, which can only process digital signals, to interact with the physical world, where analog signals are prevalent. ADCs utilize various techniques to quantize analog signals into a discrete set of digital values.

The resolution of an ADC determines the number of distinct digital values it can produce, directly influencing the precision of the conversion. A higher resolution ADC provides finer granularity, resulting in more accurate digital representations of the analog input.



Code:

```
int valADC;
char x[4];
void main() {
    UART1_Init(9600);
    ADC_Init();
    while(1){
        valADC=ADC_Read(3);
        IntToStr(valADC, x);
        UART1_Write_Text("Analog Value = ");
        UART1_Write_Text(x);
        strcpy(x,"");
        UART1_Write(13);
        delay_ms(1000);
    }}
```

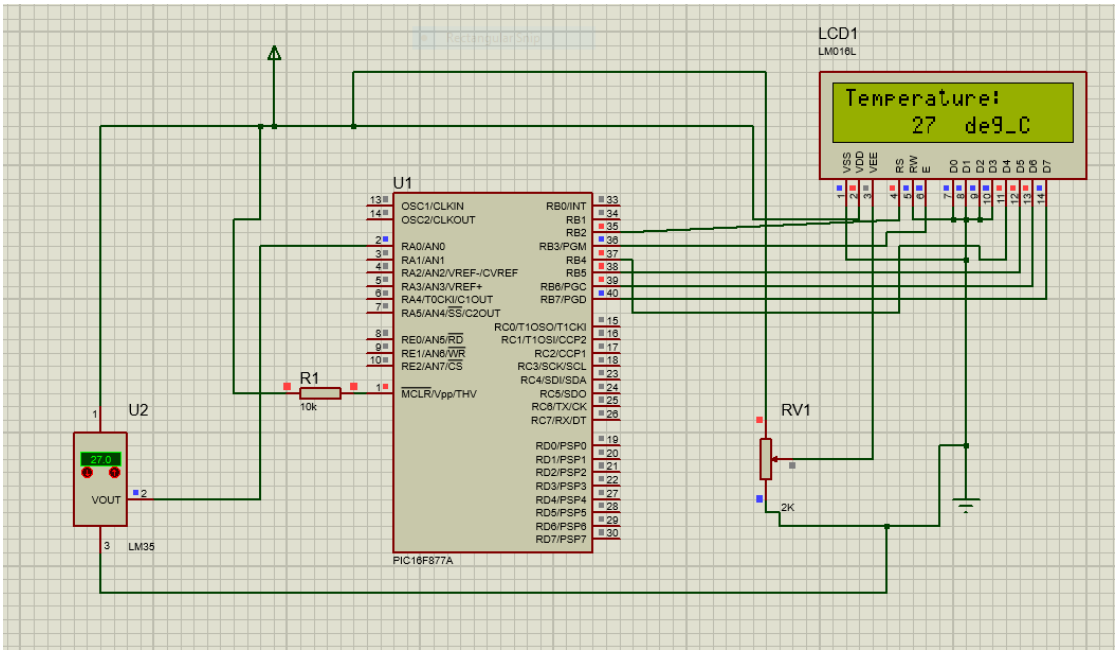
Lab No. : 05

Experiment Name: Temperature Sensor Interfacing with Microcontroller

Objectives: To understand the concept of temperature sensing and its applications in microcontroller-based systems. To interface a temperature sensor with a microcontroller and read temperature data. And, to analyze the relationship between the temperature and the corresponding digital output.

Theory: Temperature sensors are devices that convert temperature measurements into electrical signals. These sensors are widely used in various applications, including environmental monitoring, industrial control, and consumer electronics.

Thermistors and LM35 sensors are two common types of temperature sensors. Thermistors are semiconductor devices whose resistance changes with temperature. LM35 sensors are integrated circuits that output a voltage proportional to the temperature.



Code:

```
  sbit LCD_RS at RB2_bit; //LCD
reset

  sbit LCD_EN at RB3_bit; //LCD
enable

  sbit LCD_D4 at RB4_bit; //Data

  sbit LCD_D5 at RB5_bit; //Data

  sbit LCD_D6 at RB6_bit; //Data

  sbit LCD_D7 at RB7_bit; //Data

  sbit LCD_RS_Direction at
TRISB2_bit;

  sbit LCD_EN_Direction at
TRISB3_bit;

  sbit LCD_D4_Direction at
TRISB4_bit;

  sbit LCD_D5_Direction at
TRISB5_bit;

  sbit LCD_D6_Direction at
TRISB6_bit;

  sbit LCD_D7_Direction at
TRISB7_bit;

  int i;
  float temp,display;

  char t[16];

void main() {

  TRISB=0x00;

  TRISA=0xff;

  adcon1=0x80;

  Lcd_Init();

  ADC_init();

  Lcd_cmd(_LCD_CLEAR);

  Lcd_cmd(_LCD_CURSOR_OFF);

  Lcd_out(1,1,"Temp:  ");

  while(1) {

      unsigned int
temp1=ADC_read(3);

      temp=temp1*4.88;

      temp=temp/10;

      floattostr(temp,t);

      lcd_out(1,6,t);
```

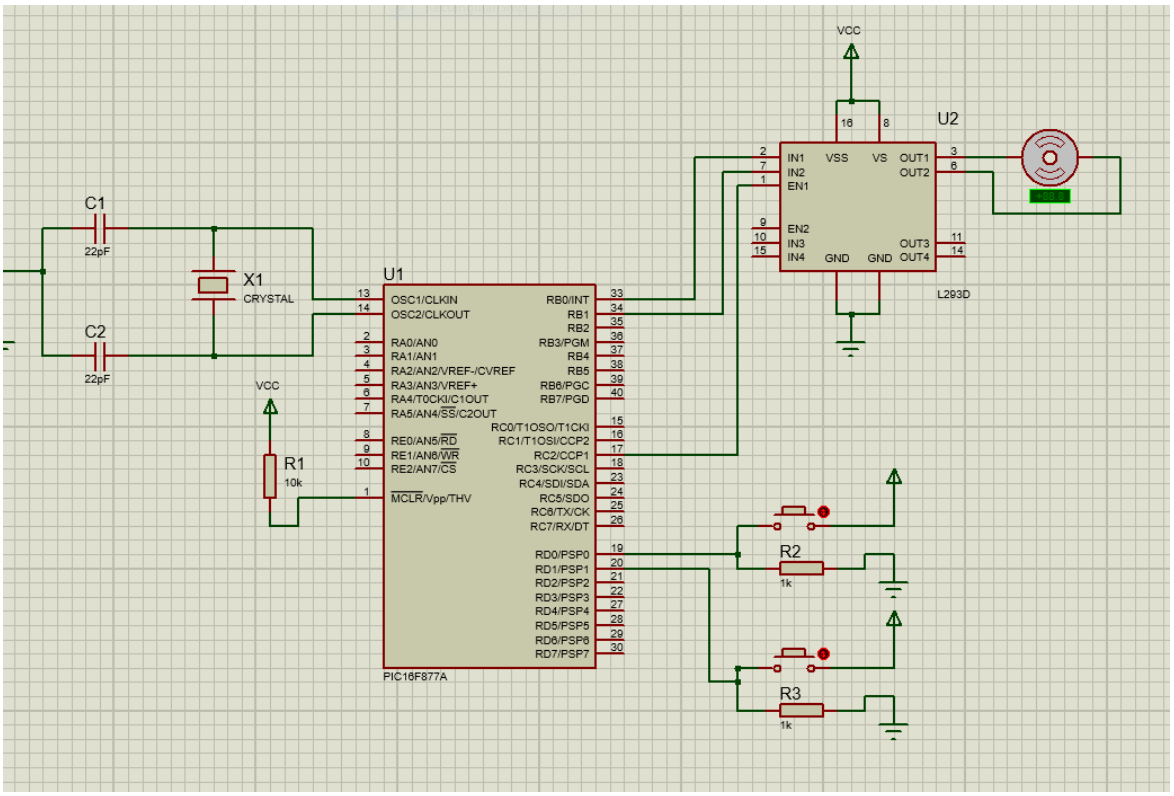
```
        lcd_chr(1,15,223);
        lcd_out_cp("C");
        }
    }
```

Lab No. : 06

Experiment Name: DC Motor Speed Control Using PWM and Microcontroller

Objectives: The objective of this lab is to understand and implement DC motor speed control using Pulse Width Modulation (PWM) generated by a microcontroller. This involves learning how to use PWM signals to regulate motor speed by varying the duty cycle of the signal. Explore the relationship between PWM duty cycle and motor speed. Implement PWM signal generation using a microcontroller (such as Arduino, PIC, or 8051). Control the speed of a DC motor by adjusting the duty cycle of the PWM signal. Understand how to interface a DC motor with a microcontroller using a driver

Theory: Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between the full Vcc of the board (e.g., 5 V on UNO, 3.3 V on a MKR board) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and Vcc controlling the brightness of the LED.



Code:

```
void main() {
    short duty=0;
    TRISB=0x00;
    TRISD=0xff;
    //anti clock
    portb.f0=0xff;
    portb.f1=0x00;
    PWM2_Init(1000);
    PWM2_Start();
    PWM2_Set_Duty(duty);
    while(1){
        if(portd.f0==0xff &&
duty<246){
            delay_ms(100);
            if(portd.f0==0xff &&
duty<246){
                duty=duty+10;
PWM2_Set_Duty(duty);
            }
        }
        if(portd.f1==0xff &&
duty>9){
            delay_ms(100);
            if(portd.f1==0xff &&
duty>9){
                duty=duty-10;
PWM2_Set_Duty(duty);
            }
        }
        delay_ms(10);
    }
}
```

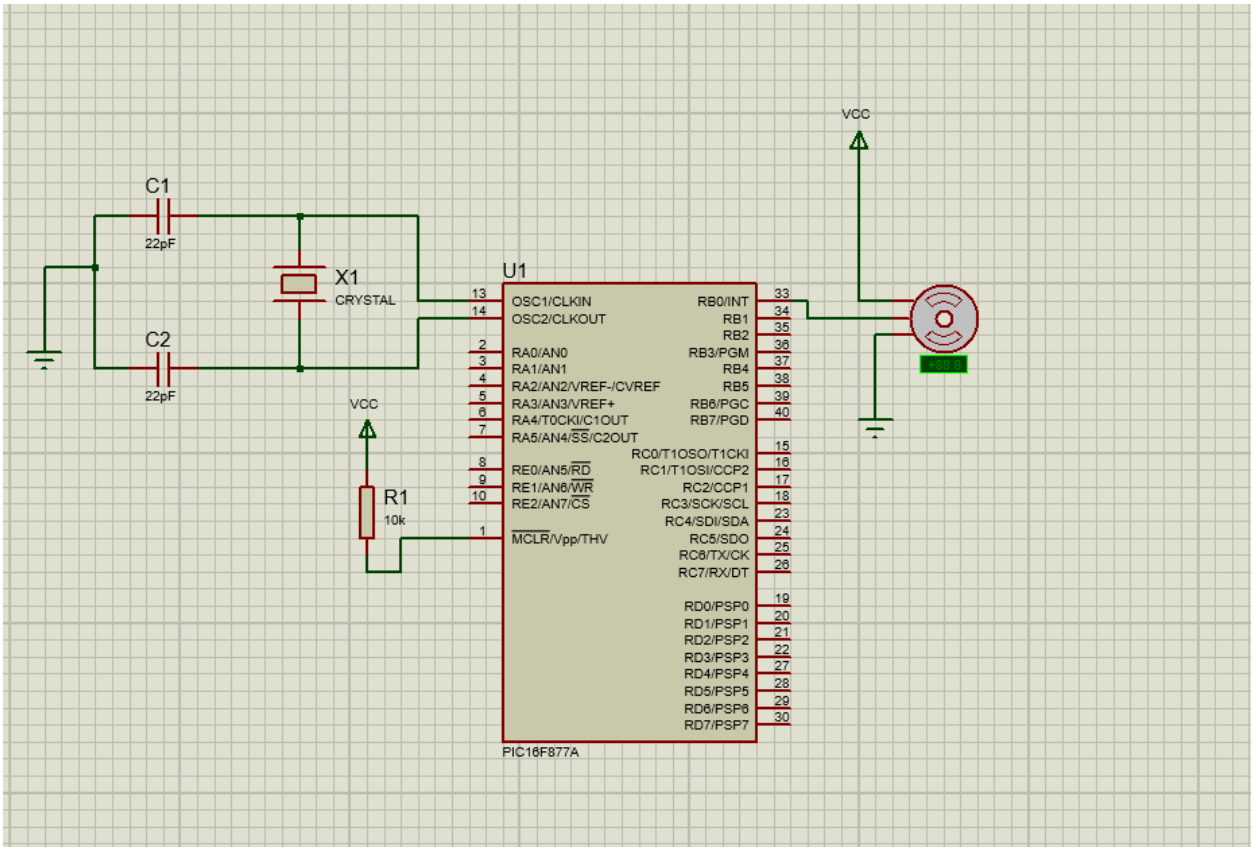
Lab No. : 07

Experiment Name: Servo Motor Control Using Microcontroller

Objectives: The objective of this lab is to explore the control of a servo motor using a microcontroller. The primary goal is to understand how pulse width modulation (PWM) signals can be used to control the angular position of the servo motor's shaft. The lab involves programming a microcontroller (such as Arduino, PIC, or 8051) to generate precise PWM signals that correspond to desired angles. Additionally, the lab will focus on understanding the interface between the microcontroller and the servo motor, and how the motor's feedback mechanism maintains position accuracy. Through this experiment, practical applications of servo motor control, such as in robotics, automation, and precision positioning systems, will be examined.

Theory: A servo motor (or servo) is a little box that contains a DC motor, an output shaft (servo arm) which is connected to the motor through a series of gears, and an electronic circuit to control the position of the shaft. The objective of using a servo is to achieve precise angular positioning of an object.

In order to accomplish a servo function, an instantaneous positioning information of the output shaft is fed back to the control circuit using a transducer. A simplest way of doing this is by attaching a potentiometer to the output shaft or somewhere in the gear train. The control electronics compares the feedback signal (which contains the current position of the shaft) from the potentiometer to the control input signal (which contains information of the desired position of the shaft), and any difference between the actual and desired values (known as an error signal) is amplified and used to drive the DC motor in a direction necessary to reduce or eliminate the error. The error is zero when the output shaft gets to the desired position.



Code:

```
void servoRotate0(){
    int i;
    for(i=0;i<50;i++){
        portb.f0=1;
        delay_us(800);
        portb.f0=0;
        delay_us(19200);
    }
}
void servoRotate90(){
    int i;
    for(i=0;i<50;i++){
        portb.f0=1;
        delay_us(1500);
        portb.f0=0;
        delay_us(18500);
    }
}
void servoRotate180(){
    int i;
    for(i=0;i<50;i++){
        portb.f0=1;
        delay_us(2200);
        portb.f0=0;
        delay_us(17800);
    }
```

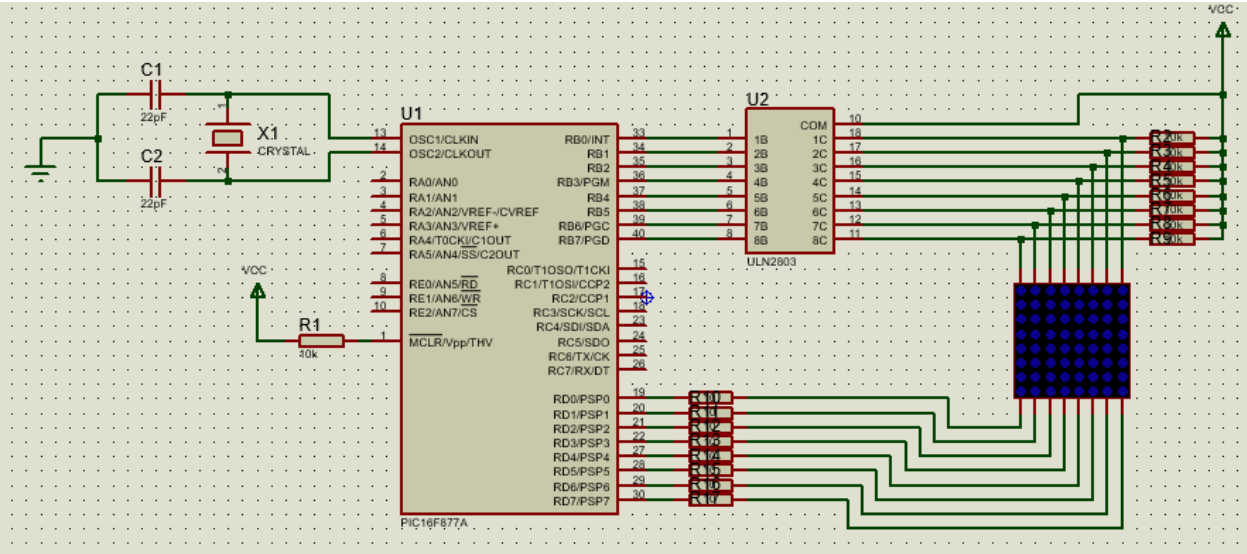
```
void main() {
    TRISB=0x00;
    PORTB=0x00;
    while(1){
        servoRotate0();
        delay_ms(5000);
        servoRotate90();
        delay_ms(5000);
        servoRotate180();
        delay_ms(5000);
    }
}
```

Lab No. : 08

Experiment Name: Dot Matrix Display Interfacing with Microcontroller

Objectives: The objective of this lab is to interface a dot matrix display with a microcontroller to display custom patterns or characters. The focus is on understanding how to control individual LEDs in the matrix using rows and columns driven by the microcontroller. The lab will explore how to program the microcontroller to scan and refresh the display dynamically while minimizing the number of pins required through techniques such as multiplexing. Additionally, the goal is to learn how to generate alphanumeric characters and graphical designs on the dot matrix display using efficient code and data manipulation techniques.

Theory: A dot matrix display consists of a grid of LEDs arranged in rows and columns, where each LED represents one pixel. To control a specific pixel, both its row and column must be activated. By rapidly scanning through each row or column, the microcontroller can refresh the display to make it appear that all LEDs are lit simultaneously, a process known as multiplexing. This technique reduces the number of required microcontroller pins, as only one row or column is activated at any given time. Various driver ICs that can further simplify the interfacing, allowing complex patterns and animations to be displayed efficiently.



Code:

```
void main() {
    const int delay=5;
    TRISB=0x00;
    TRISD=0x00;
    while(1){
        portb=0x00;
        portd=0x80;
        delay_ms(delay);
        portb=0xff;
        portd=0x40;
        delay_ms(delay);
        portb=0xff;
        portd=0x20;
        delay_ms(delay);
        portb=0x18;
        portd=0x10;
        delay_ms(delay);
        portb=0x18;
        portd=0x08;
        delay_ms(delay);
        portb=0xff;
        portd=0x04;
        delay_ms(delay);
        portb=0xff;
        portd=0x02;
        delay_ms(delay);
        portb=0x00;
        portd=0x01;
        delay_ms(delay);
    }
}
```

Lab No. : 09

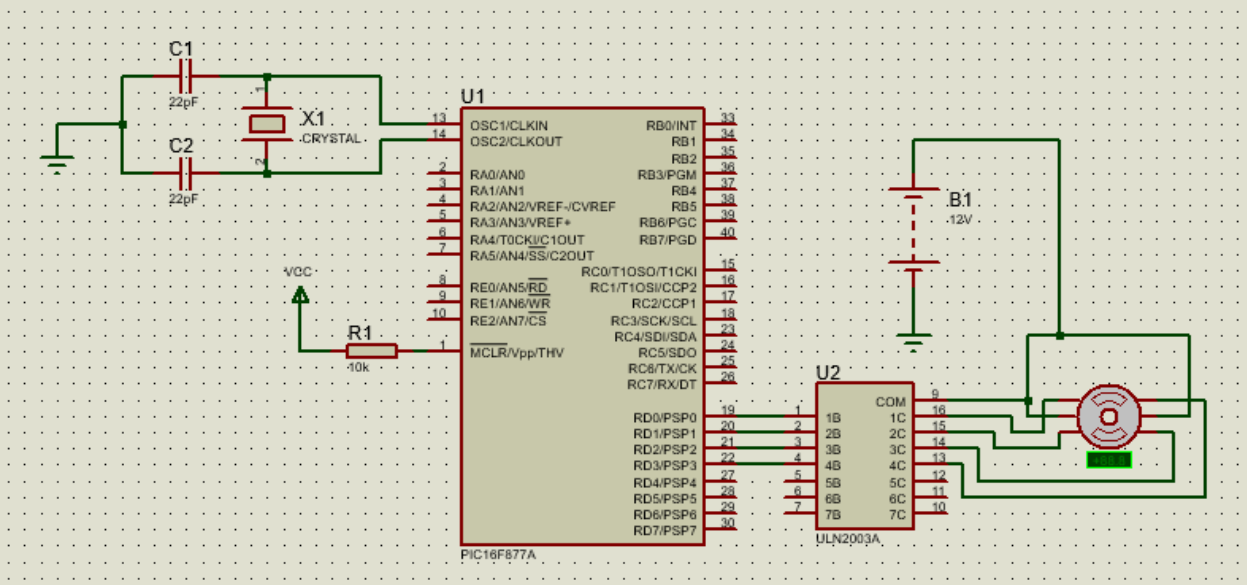
Experiment Name: Stepper Motor Interfacing with Microcontroller

Objectives: The aim of this lab is to interface a stepper motor with a microcontroller to achieve precise control over its rotation. The focus is on understanding how stepper motors operate and how to control their speed and direction using a microcontroller. This involves generating the necessary step pulses to drive the motor in both full-step and half-step modes. Additionally, the lab will cover the design of an appropriate driver circuit and explore practical applications of stepper motors, such as in CNC machines, 3D printers, CNC machines and robotics.

Theory: A stepper motor is a type of synchronous motor that moves in discrete steps, allowing for precise control of angular position. It operates by energizing coils in a sequential manner, causing the motor's rotor to rotate in fixed increments. Unlike other motors, stepper motors do not rotate continuously but move in "steps" as they respond to each electrical pulse.

Stepper motors can be driven in various modes, including full-step, half-step, and microstepping. In full-step mode, one coil is energized at a time, producing maximum torque. In half-step mode, two coils are energized simultaneously, allowing for finer control and smoother motion. The stepper motor's position is determined by the sequence and timing of these pulses, which are generated by the microcontroller.

The microcontroller interfaces with the stepper motor via a driver circuit, such as the ULN2003a or L298N, which amplifies the control signals to drive the motor's coils. The microcontroller sends pulse signals in a specific sequence, controlling the speed by adjusting the delay between pulses and the direction by altering the pulse sequence. Stepper motors are ideal for applications requiring precise control over movement, such as in printers, robotics, and automated manufacturing systems.



Code:

```
void main() {
    const int delay=1000;
    TRISD=0x00;
    PORTD=0xff;

    while(1){
        portd=0b00001001;
        delay_ms(delay);
        portd=0b00001100;
        delay_ms(delay);
        portd=0b00000110;
        delay_ms(delay);
        portd=0b00000011;
        delay_ms(delay);
    }
}
```