LED Blinking

Project Name: Interfacing LED Blinking with PIC Microcontroller

Objectives:

- To understand the basic configuration of a PIC Microcontroller.
- To understand how a PIC Microcontroller outputs power
- To understand how to program and connect code with the Microcontroller to control the LED

Apparatus List:

- PIC16F877A Microcontroller
- Crystal Oscillator
- 2 pcs of 22 pF capacitor
- 10K Resistor
- 100 Ohm Resistor
- LED
- Power supply

Theory:

The PIC16F877A is a popular microcontroller from Microchip Technology, and it is commonly used for embedded systems and microcontroller-based projects. When you connect an LED to a PIC microcontroller and program it to blink at a certain time interval, you are essentially utilizing the microcontroller's capabilities to control the LED's state based on a predetermined timing sequence. Here's a brief overview of the theory behind it:

1. **Microcontroller Basics:**
    - A microcontroller is a compact integrated circuit that contains a processor core, memory, and various peripherals. The PIC16F877A, in particular, is an 8-bit microcontroller with Flash program memory.
2. **I/O Ports:**

- The PIC16F877A has multiple General Purpose Input/Output (GPIO) ports, which can be configured as digital inputs or outputs. You connect the LED to one of these ports configured as an output.

3. **Programming the Microcontroller:**

    - You write a program in a programming language like Assembly or C, which is then compiled into machine code that the microcontroller can execute. In the program, you specify the behavior of the microcontroller, including how it interacts with external components like LEDs.

4. **Blinking LED Algorithm:**

    - To make the LED blink, you typically use a loop in your program. Inside the loop, you turn the LED on, wait for a certain amount of time, turn it off, and then wait again. This process repeats indefinitely.

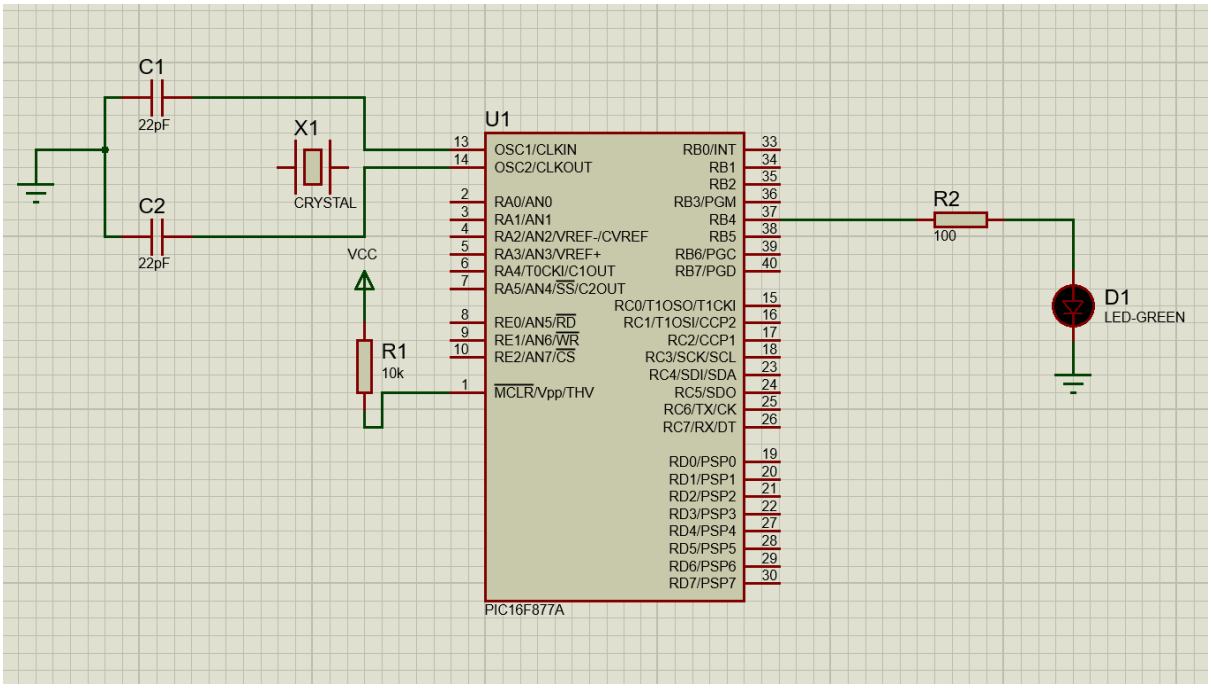
5. **Timing Control:**

    - Microcontrollers have built-in timers/counters that can be used to generate precise time intervals. You can configure these timers to control the timing of your LED blinking. For example, you might use the Timer/Counter to create delays between turning the LED on and off.

Code:

```
void main() {
    TRISB = 0x00;
    portb = 0x00;

    while (1) {
        portb.f4 = 1;
        delay_ms(2000);

        portb.f4 = 0;
        delay_ms(500);
    }
}
```

Circuit:



Result: The result of the programmed PIC16F877A microcontroller is a blinking LED. The LED turns on for a specific duration, determined by the programmed delay, and then turns off for the same duration. This on-off cycle repeats indefinitely, creating a blinking effect.

Single 7 Segment Display

Project Name: Interfacing 7 Segment Display with PIC Microcontroller

Objectives:

- To understand the basic configuration of a PIC Microcontroller.
- To understand the basic configuration of a 7 Segment Display
- To design a counter controlled by time or delay and display the corresponding numeric increment

Apparatus List:

- PIC16F877A Microcontroller
- Crystal Oscillator
- 2 pcs of 22 pF capacitor
- 10K Resistor
- 8 pcs of 150 Ohm Resistor
- 7 segment CC/CA LED display
- Power supply

Theory:

Connecting a single 7-segment display to the PIC16F877A involves using the microcontroller's GPIO (General Purpose Input/Output) pins to control the segments of the display. The PIC16F877A has a total of 8 pins (RB0 to RB7) that I am using to interface with the 7-segment display.

Here's a brief overview of the theory behind it:

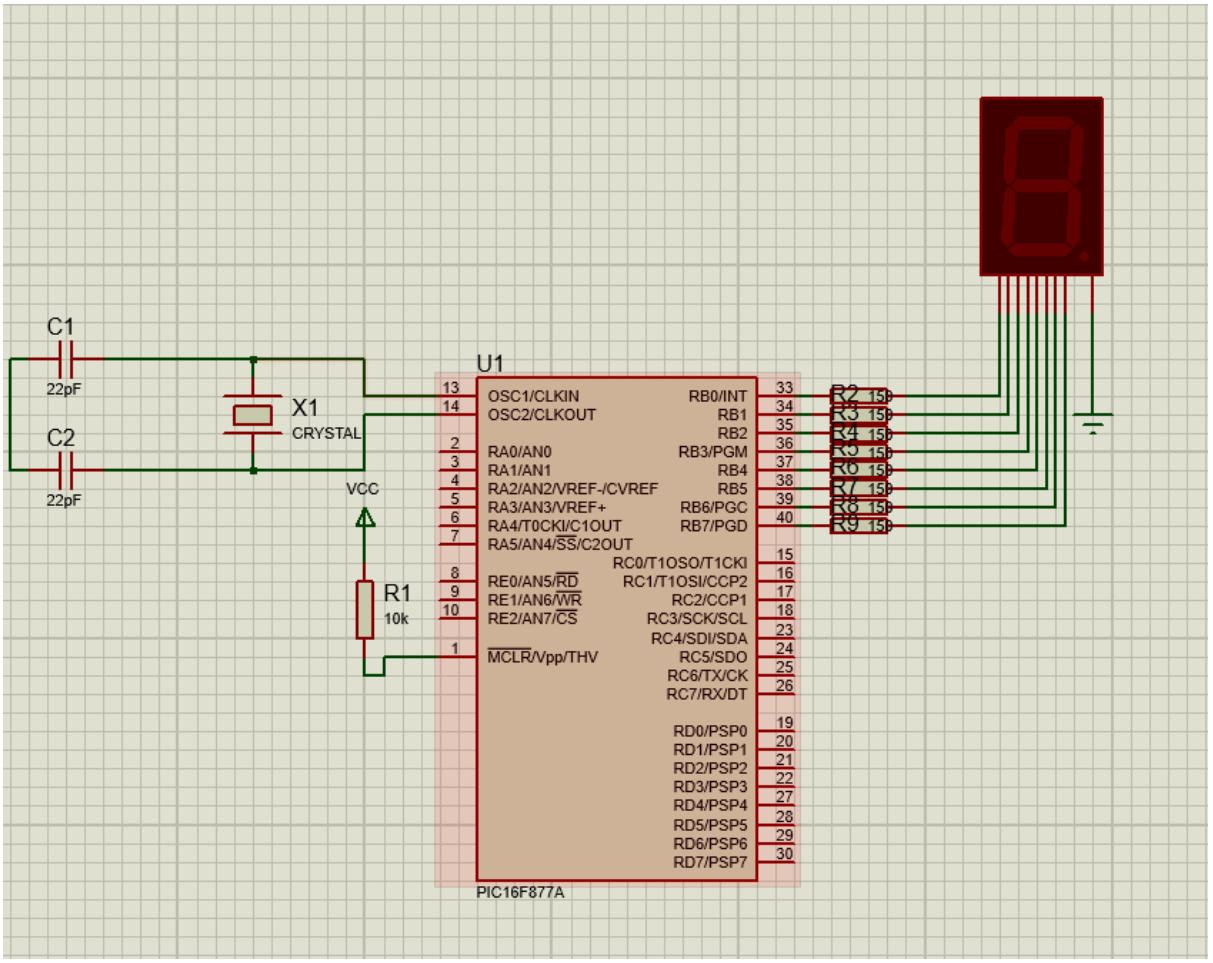
1. **7-Segment Display:**
    - A 7-segment display is a type of electronic display device that can represent numerals and some letters of the alphabet using seven segments. Each segment is a light-emitting diode (LED) or other lighting source.

    - The segments are labeled as a, b, c, d, e, f, and g. These segments are arranged in a specific pattern to form numbers from 0 to 9 and some letters.
2. **Common Cathode vs. Common Anode:**
    - 7-segment displays can be either common cathode or common anode. In a common cathode display, all cathodes of the LEDs are tied together, while in a common anode display, all anodes are tied together.
    - In this case, you need to know whether your 7-segment display is common cathode or common anode, as the wiring and code will differ accordingly.
3. **Interfacing with PIC16F877A:**
    - Each segment of the 7-segment display is connected to one of the RB0 to RB7 pins on the PIC16F877A.
    - You control the display by turning on and off the individual segments to display the desired number or character.
    - The RB0 to RB7 pins will be set as outputs, and you will send specific patterns to these pins to turn on the corresponding segments.
4. **Multiplexing (if needed):**
    - If you want to display multiple digits, you may use multiplexing. This involves rapidly switching between different digits to give the illusion that all digits are continuously displayed.
    - For multiplexing, you would typically use additional GPIO pins to select the active digit, and then update the segments for that digit before moving on to the next one.
5. **Programming:**
    - In your code, you will define patterns for each digit (0-9) and possibly some letters, depending on your application.
    - By sending the appropriate pattern to the RB0-RB7 pins, you can control which segments of the 7-segment display are lit, forming the desired character.

Code:

```
char array_CC[]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07,
0x7F, 0x6F};
void main() {
    TRISB = 0x00;
    portb = 0x00;


    while (1) {
        int i;
         for (i = 0; i < 10; i++) {
            portb = array_CC[i];
            delay_ms(1000);

         }

    }
}
```

Circuit:



7 Segment Multiplexing

Project Name: Interfacing multiple 7 Segment Display with PIC Microcontroller

Objectives:

- To understand the basic configuration of a PIC Microcontroller.
- To understand connecting multiple 7 segment display and their working principle.
- To understand the basic configuration of a 7 Segment Display
- To design a counter controlled by time or delay and display the corresponding numeric increment

Apparatus List:

- PIC16F877A Microcontroller
- Crystal Oscillator
- 2 pcs of 22 pF capacitor
- 10K Resistor
- 8 pcs of 150 Ohm Resistor
- Multiple digit 7 segment CC/CA LED display
- 2N2222 bipolar NPN transistors
- Power supply

Theory:

Multiplexing is a technique used to control multiple displays (or digits of a single display) using fewer pins on a microcontroller. In the context of 7-segment displays, multiplexing involves cycling through each digit one at a time, turning on the segments for that digit, and then quickly moving to the next digit. This process is repeated fast enough that the human eye perceives all digits as continuously illuminated.

1. **Multiplexing Basics:**
    - In a simple static display setup, each digit of a 7-segment display would require its own set of pins on the microcontroller. This can quickly consume a significant number of pins.
    - Multiplexing reduces the number of required pins by time-sharing them among multiple digits. Instead of using separate pins for each digit, the microcontroller cycles through the digits sequentially.
2. **Common Cathode or Common Anode:**
    - In a common cathode display, all cathodes of the 7-segment digits are tied together, and each anode is connected to an individual pin. In a common anode display, all anodes are tied together, and each cathode is connected to an individual pin.

    - For the purpose of multiplexing, common cathode displays are often used. Each digit's cathode is sequentially connected to the common cathode pin.
3. **Microcontroller Connections:**

    - The segments of all digits are connected to the respective segment pins (a, b, c, d, e, f, g) on the microcontroller.
    - The common cathode (or anode) pins of each digit are connected to individual pins on the microcontroller.
4. **Multiplexing Process:**

    - The microcontroller cycles through each digit, turning on its common cathode pin while simultaneously setting the appropriate segments to display the desired number or character.
    - For example, to display the number "123" on a 3-digit common cathode display, the microcontroller would:
        - Enable the common cathode pin for the first digit, set the segment values for "1," and then quickly move to the next digit.
        - Repeat the process for the second and third digits.
5. **Timing and Persistence of Vision:**

    - Multiplexing requires precise timing to ensure that each digit is displayed for a short enough duration that the human eye perceives a continuous display.
    - The persistence of vision phenomenon allows the eye to blend the rapidly changing images, making it seem like all digits are continuously lit, even though they are actually illuminated sequentially.
6. **Code Implementation:**

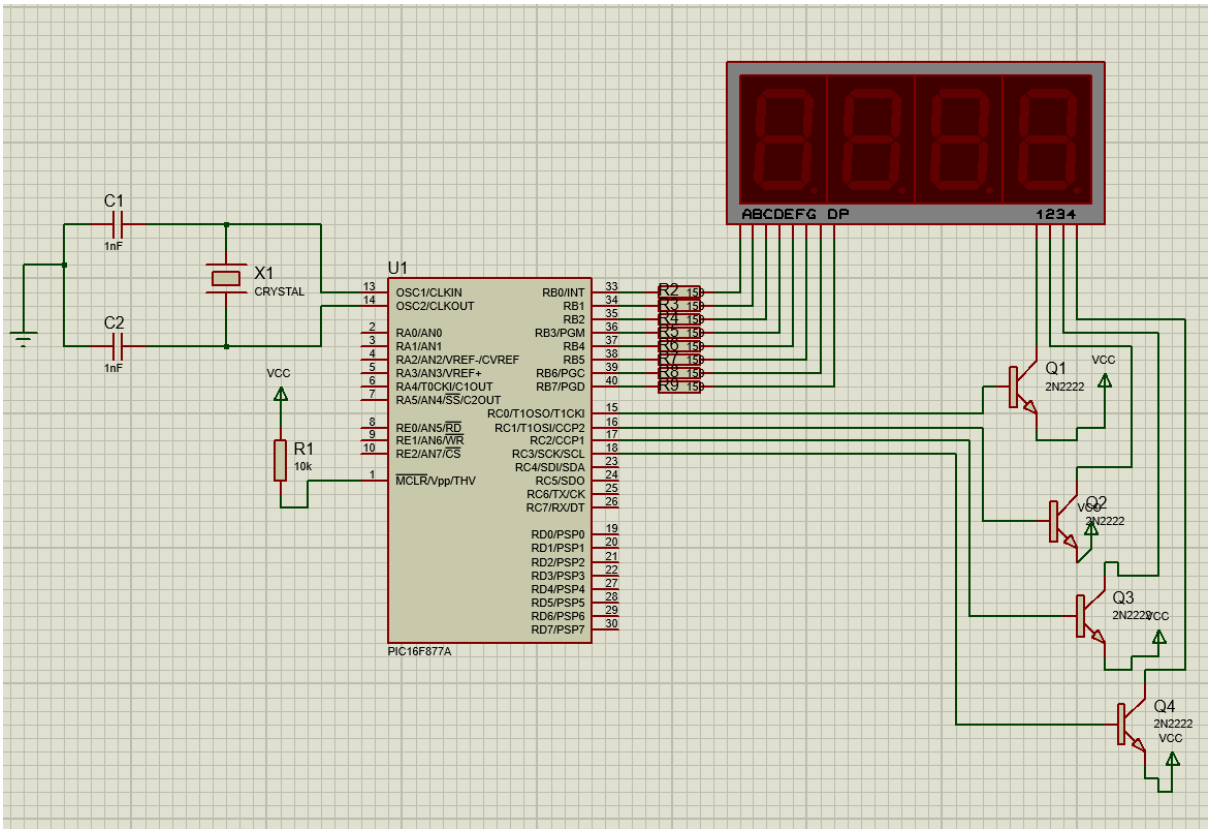
    - In the software code, a loop is typically used to cycle through each digit and update the segment values accordingly.
    - Delays may be introduced to control the display refresh rate. The delay should be short enough to provide seamless

visual continuity but long enough to ensure that all digits are updated within the display cycle.

Code:

```
char ca[] = {0xC0, 0xF9, 0xA4, 0xB0,0x99, 0x92};
int i = 0;
int number = 5423;
void main() {
    TRISB = 0x00;
    TRISC = 0x00;

    while (1) {
       for (i = 0; i < 25; i++) {
          portc.f0 = 1;
          portb = ca[5];
          delay_ms(10);
          portc.f0 = 0;

          portc.f1 = 1;
          portb = ca[4];
          delay_ms(10);
          portc.f1 = 0;

          portc.f2 = 1;
          portb = ca[2];
          delay_ms(10);
          portc.f2 = 0;

          portc.f3 = 1;
          portb = ca[3];
          delay_ms(10);
          portc.f3 = 0;
       }
    }
}
```

Circuit:



LCD Display

Project Name: Interfacing LCD Display with PIC Microcontroller

Objectives:

- Interface PIC16F877A with LM016L LCD for displaying information.
- Utilize microcontroller commands to control LCD functions.
- Achieve successful communication and data display on the LCD.
- Create a simple user interface for information presentation.

Apparatus List:

- PIC16F877A Microcontroller
- Crystal Oscillator
- 2 pcs of 22 pF capacitor
- 10K Resistor
- LM016L LCD Display
- POT-HG Potentiometer
- Power supply

Theory:

The LM016L is a popular 16x2 character LCD (Liquid Crystal Display) module that can be interfaced with microcontrollers like the PIC16F877A. Let's break down the theory behind connecting and interfacing the LM016L with the PIC16F877A.

1. LCD Basics:

- LCDs (Liquid Crystal Displays) are devices that use liquid crystals to modulate light and display information. The LM016L is a 16x2 character LCD, meaning it can display 16 characters in each of its two rows.
- Each character position is made up of a 5x8 dot matrix.

2. Pin Configuration:

- The LM016L typically has 14 pins. The important pins for interfacing are:
    - VSS and VDD: Power supply (Ground and +5V).
    - RS (Register Select): Selects between command (0) and data (1) registers.
    - RW (Read/Write): Selects between read (1) and write (0) operations.
    - E (Enable): Used to latch information into the LCD.
    - D0-D7: Data lines for 8-bit mode. In 4-bit mode, typically only D4-D7 are used.

3. Interfacing with PIC16F877A:

- The PIC16F877A is a microcontroller from Microchip's PIC family. To interface the LM016L with the PIC, you will typically use some of the digital I/O pins of the PIC for data and control signals.
- Connect the data lines (D4-D7) of the LM016L to the digital pins of the PIC. The RS, RW, and E pins of the LCD will also be connected to the PIC's digital pins.
- A potentiometer can be used to adjust the contrast by varying the voltage at the VO pin.

4. Initialization Sequence:

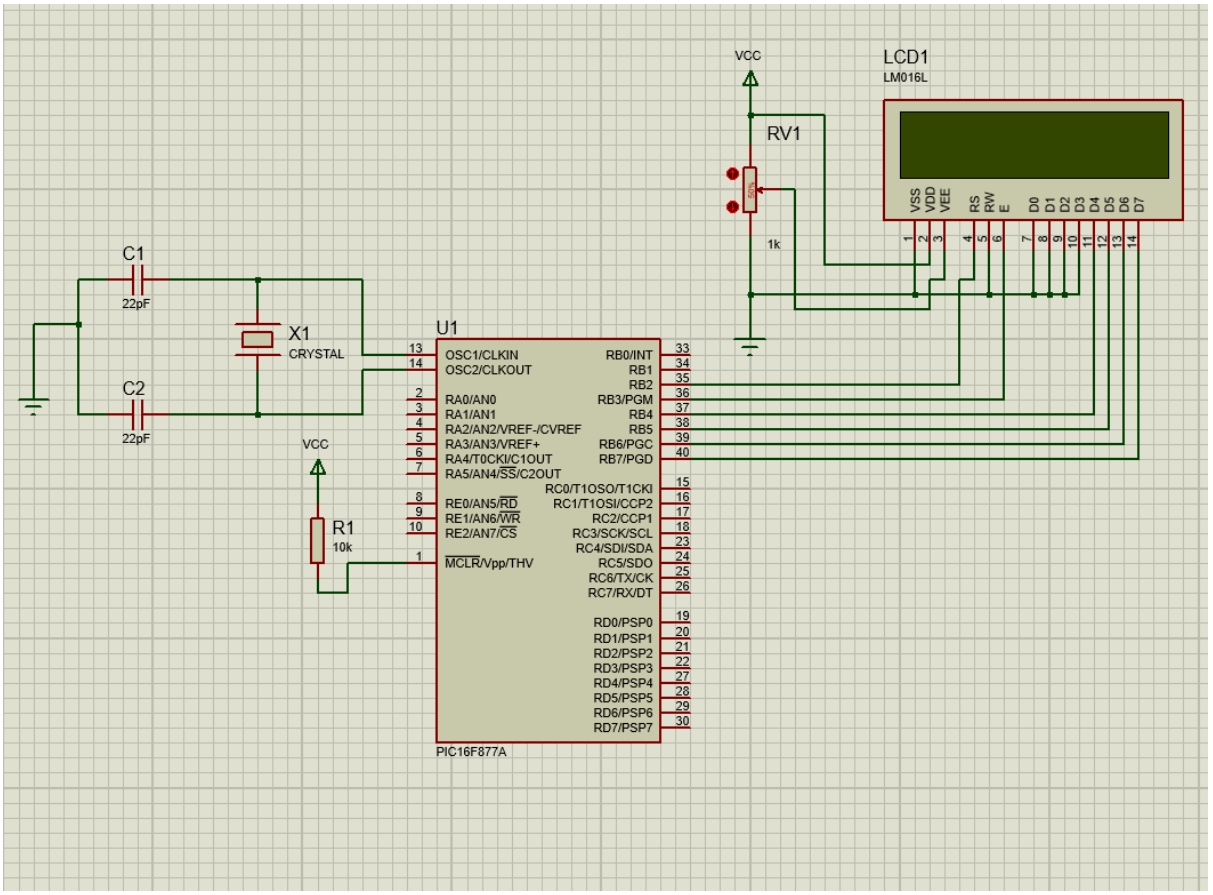
- Before using the LCD, it needs to be initialized. This involves sending specific commands to set up the LCD, such as setting the display to 2 lines, configuring the cursor, etc.
- Typically, a delay is introduced after each command to allow the LCD time to process the command.

Code:

```
sbit LCD_RS at RB2_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;

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;

char a[] = "Nayem";
char b[] = "CSE, Pabna University of Science & Technology.";
```

```
int i;

void main() {
    LCD_Init();
    LCD_Cmd(_LCD_CURSOR_OFF);
    LCD_Cmd(_LCD_CLEAR);
    LCD_Out(1,1,a);
    LCD_Out(2,1,b);
    delay_ms(80);

    while (1) {
        for (i = 0; i < 20; i++) {
           LCD_Cmd(_LCD_SHIFT_LEFT);
           delay_ms(100);
        }
        //Lcd_Cmd(_LCD_CLEAR);

        for (i = 0; i < 20; i++) {
           LCD_Cmd(_LCD_SHIFT_RIGHT);
           delay_ms(100);
        }
        //LCD_Cmd(_LCD_CLEAR);
    }
}
```

Circuit:



Motor Speed Control

Project Name: Interfacing a DC motor and control it's speed with PIC Microcontroller

Objectives:

- Interface PIC16F877A with L293D motor driver for DC motor control.

- Implement Pulse Width Modulation (PWM) to regulate motor speed.
- Develop PIC16F877A firmware for bidirectional motor control.
- Optimize code for efficient speed control and ensure hardware compatibility.

Apparatus List:

- PIC16F877A Microcontroller
- Crystal Oscillator
- 2 pcs of 22 pF capacitor
- 10K Resistor
- L293D motor driver IC
- DC motor
- Push buttons
- Power supply

Theory:

Controlling the speed of a DC motor using a microcontroller like the PIC16F877A and an H-bridge motor driver like the L293D involves pulse width modulation (PWM). Here's a brief explanation of the theory behind it:

1. **DC Motor Basics:**
    - The speed of a DC motor is directly proportional to the voltage applied to it. However, providing a constant voltage for speed control is not efficient and can lead to overheating. PWM can control it's speed.
2. **Pulse Width Modulation (PWM):**
    - PWM is a technique where you rapidly switch a signal (usually a square wave) between high and low states. The average voltage over time is controlled by varying the duty cycle (percentage of time the signal is high).

- In the context of motor control, PWM is used to simulate a variable voltage by controlling the average voltage applied to the motor.

3. **Speed Control Using PWM:**

   - Connect the PWM output from your PIC16F877A to the input of the L293D that controls the motor speed (often labeled as EN or PWM on the driver).
   - As you change the duty cycle of the PWM signal, you effectively change the average voltage applied to the motor. A higher duty cycle results in a higher average voltage and, consequently, a higher motor speed.

4. **PIC16F877A Configuration:**

   - Set up the PIC16F877A to generate PWM signals using one of its PWM modules by connecting with CCP1
   - Configure the PWM duty cycle according to the desired motor speed. This is usually done through software by changing the value in the PWM duty cycle register.

5. **Code Implementation:**

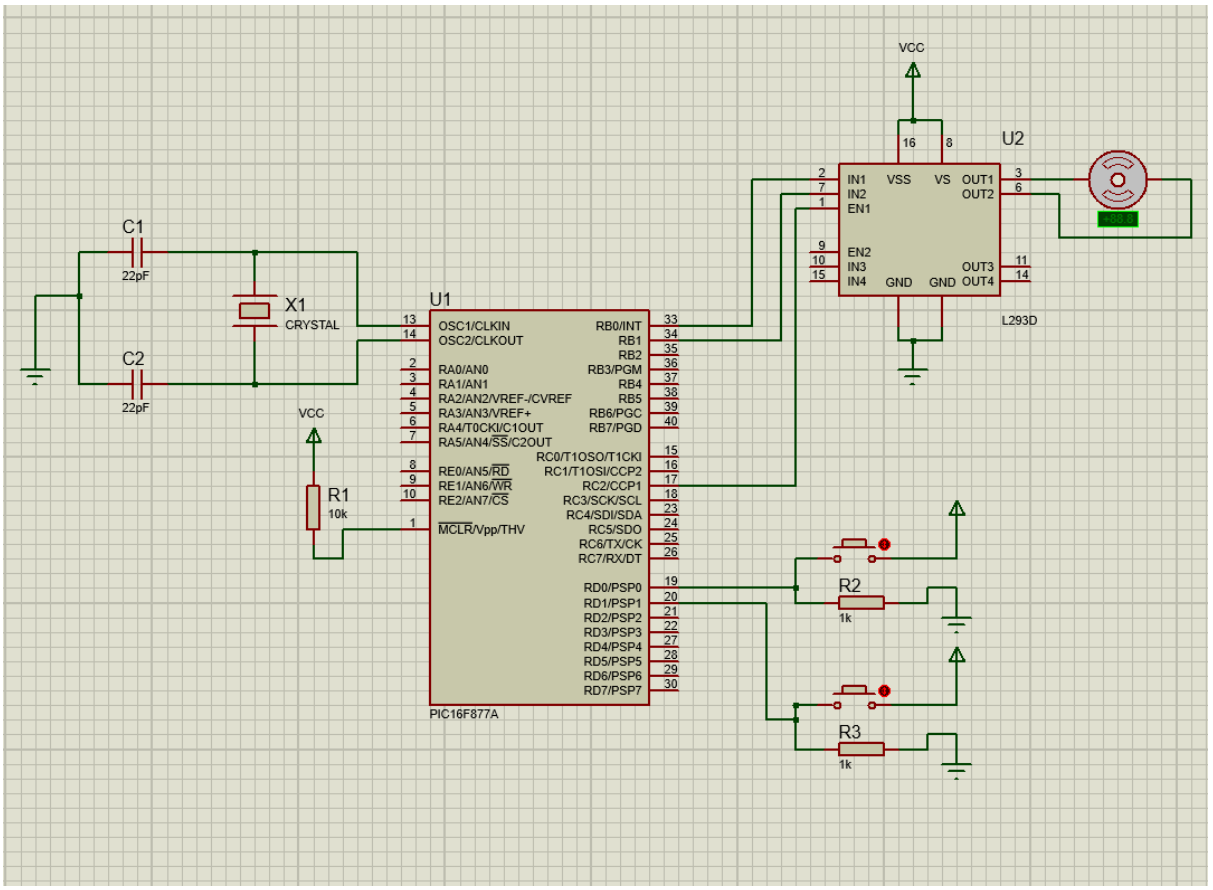
   - program that initializes the PWM module on the PIC16F877A and adjusts the duty cycle based on user input or some other control mechanism.
   - The code should also include logic to control the direction of the motor by appropriately configuring the input pins of the L293D.

Code:

```
void main() {
    short duty = 0;
    TRISB = 0x00;
```

```
    TRISD = 0xFF;

    portb.f0 = 1;
    portb.f1 = 0;



    PWM1_Init(1000);
    PWM1_Start();
    PWM1_Set_Duty(duty);

    while (1) {
        if (portd.f0 == 1 && duty < 250) {
          delay_ms(100);
          if (portd.f0 == 1 && duty < 250) {
            duty += 10;
            PWM1_Set_Duty(duty);
          }
        }
        if (portd.f1 == 1 && duty > 0) {
          delay_ms(100);
          if (portd.f1 == 1 && duty > 0) {
            duty -= 10;
            PWM1_Set_Duty(duty);
          }
        }

        delay_ms(10);

    }

}
```

Circuit:



Servo Motor Rotate

Project Name: Interfacing a Servo motor and rotate it to 0, 90 and 180 degrees.

Objectives:

- To understand the basic configuration of a PIC Microcontroller.
- To understand how a PIC Microcontroller outputs power
- To understand how to program and connect code with the Microcontroller to control the LED

Apparatus List:

- PIC16F877A Microcontroller
- Crystal Oscillator
- 2 pcs of 22 pF capacitor

- 10K Resistor
- DC motor
- Power supply

Theory:

The PIC16F877A is a popular microcontroller, and connecting a servo motor to it to control its position at 0, 90, and 180 degrees involves understanding the theory behind servo motors and the PWM (Pulse Width Modulation) signal.

Here's a brief overview:

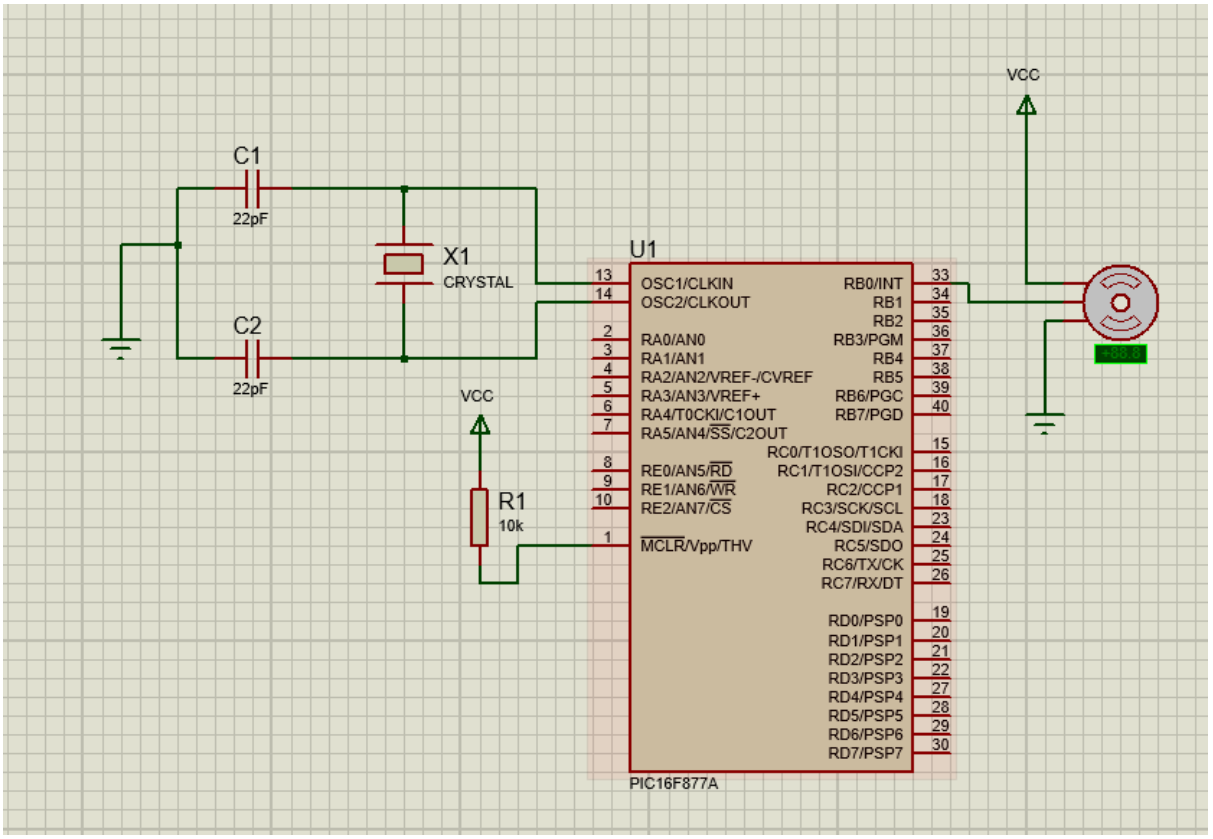
1. **Servo Motor:**
    - A servo motor is a rotary actuator that allows for precise control of angular position. It consists of a small DC motor, a gearbox, and a control circuit.
    - Servo motors are commonly used in applications where accurate positioning is required, such as in robotics and control systems.
2. **PWM (Pulse Width Modulation):**
    - PWM is a technique used to encode information in the width of the pulses in a pulsing signal. In the context of servo motors, PWM is used to control the position of the motor.
    - The PWM signal typically has a fixed frequency, and the duty cycle (percentage of time the signal is high) determines the position of the servo motor.
    - The servo motor interprets the PWM signal and moves to the corresponding position based on the duty cycle
3. **Rotate using delay_us:**
    - delay_us(800) for 0 degrees, delay_us(1500) for 90 degrees and delay_us(2200) for 180 degrees

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;
    while (1) {
```

```
        servoRotate0();
        delay_ms(2000);
        servoRotate90();
        delay_ms(2000);
        servoRotate180();
    }
}
```

Circuit:



DC Motor Relay

Project Name: Interfacing a DC Motor with relay with PIC Microcontroller.

Objectives:

- **Microcontroller Control** - Utilize the PIC16F877A to generate control signals, enabling the activation and deactivation of the relay.

- **Relay Operation** - Interface the relay with the microcontroller, ensuring its proper operation in switching the power supply to the DC motor.

- **DC Motor Control** - Enable the microcontroller to regulate the DC motor's power by controlling the relay's state through programmed logic.

- **Protection Mechanisms** - Implement protective measures, such as diodes, to safeguard the microcontroller and associated components from voltage spikes during motor operation.

Apparatus List:

- PIC16F877A Microcontroller
- Crystal Oscillator
- 2 pcs of 22 pF capacitor
- 10K Resistor
- Relay
- 1N4007 diode
- 2N2222 NPN bipolar transistor
- Power supply

Theory:

Connecting a relay with a DC motor using a PIC16F877A microcontroller involves using the microcontroller to control the operation of the relay, which in turn controls the DC motor. Here's a basic explanation of the theory behind this setup:

1. **Microcontroller (PIC16F877A):**

- The PIC16F877A is a microcontroller that can be programmed to perform various tasks. It has digital output pins that can be configured to generate logic levels (high or low voltage) to control external devices.

2. **Relay:**

   - A relay is an electromagnetic switch that is used to control high-power devices with a low-power signal. It typically consists of an electromagnetic coil and one or more switch contacts. When the coil is energized, the switch contacts are either opened or closed, depending on the relay type.

3. **DC Motor:**

   - A DC motor converts electrical energy into mechanical energy. It consists of a rotor and a stator. When a voltage is applied to the motor, a magnetic field is created, causing the rotor to rotate.

4. **Connection:**

   - The PIC16F877A's digital output pin is connected to the relay's control input. When the microcontroller sends a high signal to the relay, it energizes the coil, causing the relay switch contacts to change state. This change in state can either connect or disconnect the DC motor from the power source.

   - The DC motor is connected to the relay switch contacts. When the relay is activated, the motor receives power and starts rotating. When the relay is deactivated, the motor loses power and stops.

5. **Programming:**

   - The microcontroller needs to be programmed to control the relay. This involves writing a program that sets the appropriate digital output pin to a high or low state to control the relay based on the desired operation of the motor.

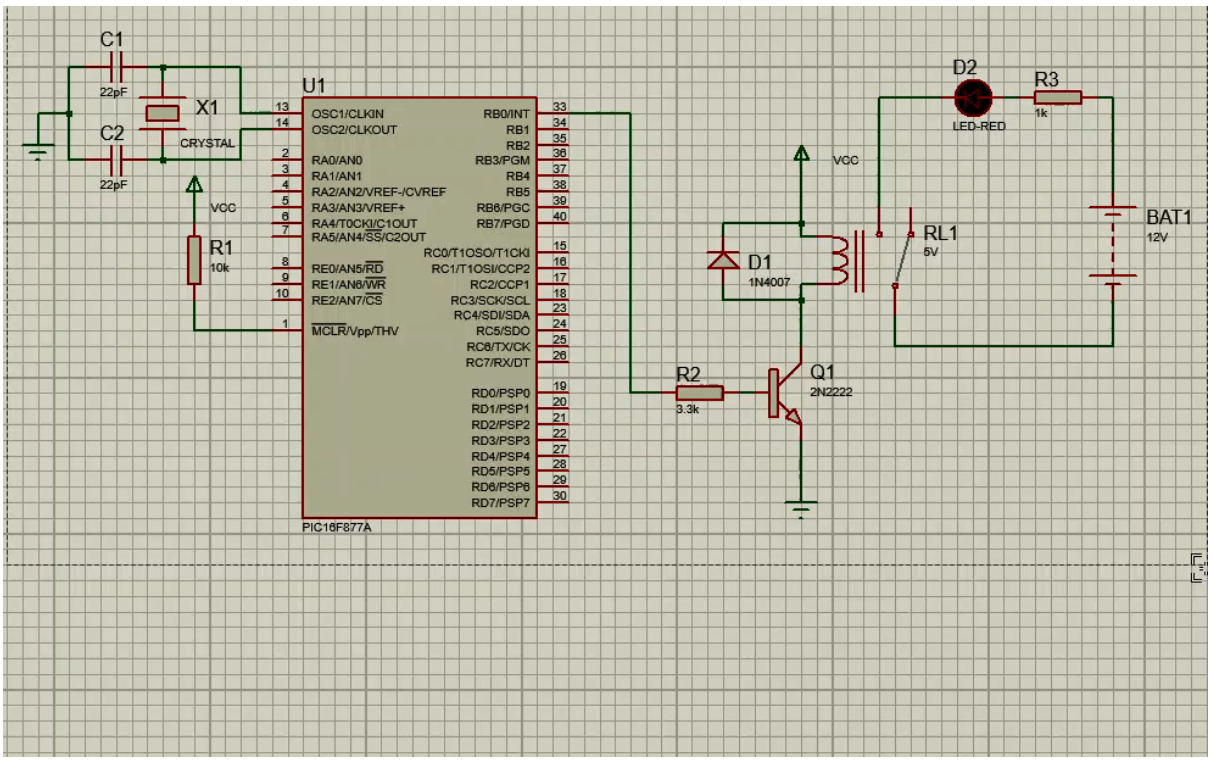
6. **Protection:**

    - It's important to include protection mechanisms, such as diodes, to suppress voltage spikes that may occur when the motor is turned off. These spikes can potentially damage the microcontroller or other components.

Code:

```
void main() {
    TRISB = 0x00;
    portb = 0x00;

    while (1) {
        portb.f4 = 1;
        delay_ms(2000);

        portb.f4 = 0;
        delay_ms(500);
    }
}
```

Circuit:


C1
22pF
X1
CRYSTAL
C2
22pF
VCC
R1
10k
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
R2
3.3k
Q1
2N2222
D1
1N4007
VCC
RL1
5V
D2
LED-RED
R3
1k
BAT1
12V