Experiment #6 (HW#1)
Introduction to PIC Microcontroller

1. Prerequisites

ENCS 538, C programming language.

2. Objectives

- To be familiar with simple microcontroller (PIC)
- How to use MPLAB software to write C code and then generate the hex file.
- How to deploy the hex file on PIC using USB programmer.

3. Background

A PIC microcontroller is a processor with built in memory and RAM and you can use it to control your projects (or build projects around it). The microcontrollers are used due to many reasons, which are:

a. *Cost:* Microcontrollers with the supplementary circuit components are much cheaper than a computer with an analog and digital I/O.
b. *Size and Weight:* Microcontrollers are compact and light compared to computers.
c. *Simple applications:* If the application requires very few number of I/O and the code is relatively small, which do not require extended amount of memory and a simple LCD display is sufficient as a user interface, a microcontroller would be suitable for this application.
d. *Reliability:* Since the architecture is much simpler than a computer it is less likely to fail.
e. *Speed:* All the components on the microcontroller are located on a single piece of silicon. Hence, the applications run much faster than it does on a computer.

Moreover, it saves building a circuit that has separate external RAM, ROM and peripheral chips. This really means that the PIC is a very powerful device that has many useful built in modules e.g.

- EEPROM.
- Timers.
- Analogue comparators.
- UART.

Even with just these four modules (note these are just example modules - there are more) you can make up many projects e.g.:

- Frequency counter - using the internal timers and reporting through UART (RS232) or output to LCD.
- Capacitance meter - analogue comparator oscillator.
- Event timer - using internal timers.

- Event data logger -capturing analogue data using an internal ADC and using the internal EEPROM for storing data (using an external I2C for high data storage capacity.

The PIC 16f877A will be used in this experiment because it is considered one of the most famous PIC microcontrollers and it's easy to see why - it comes in a 40 pin DIP pinout and it has many internal peripherals. The 40 pins (see Figure 1) make it easier to use the peripherals as the functions are spread out over the pins. This makes it easier to decide what external devices to attach without worrying too much if there are enough pins to do the job.

One of the main advantages is that each pin is only shared between two or three functions, so it is easier to decide what are the pin functions (other devices have up to 5 functions for a particular pin). However, the main disadvantages of the device is that it has no internal oscillator so you will need to add an external crystal of other clock source.

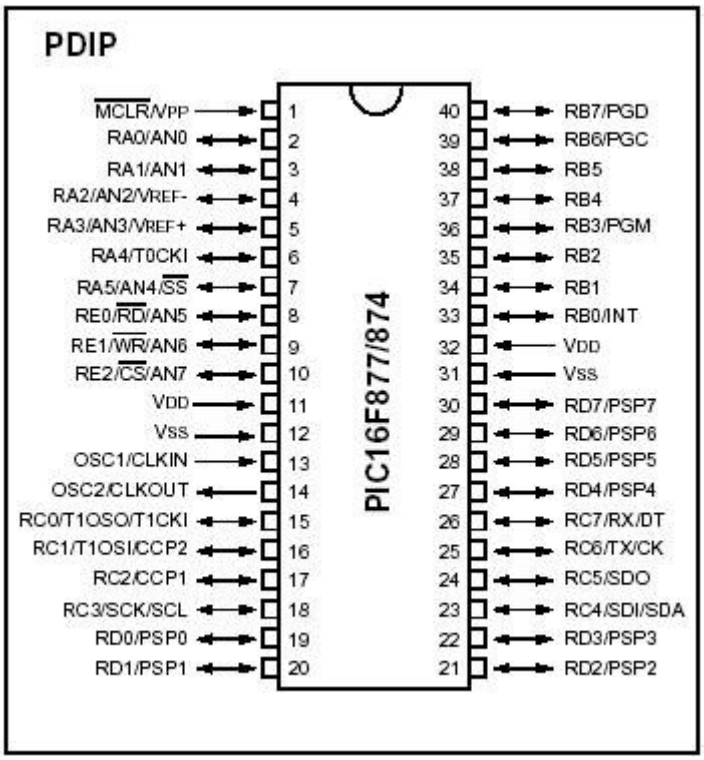

Figure 1: PIC 16F877 Microcontroller Pinout

4. Procedure

4.1 Programming Using MPLAB

1. Run the **MPLAB IDE** software.
2. Create 3 files: **delay.h, delay.c** and **ledf.c** by copying the attached code at the experiment codes section.
3. Place these files into a new folder and name it **ledsDemo.**
4. Click on **Project** then **Project Wizard** then press **Next**.
5. In "**Step One: Select a device"**, choose the **PIC16F877A** then click **Next**. In case you didn't choose the right device, you can always be able to change it by clicking on **Configure** then **Select Device…**

6. In "**Step Two: Select a language toolsuite"**, choose **HI-TECH Universal Toolsuite** then click **Next**. In case you didn't set the right toolsuite for the compiler, you can always change it by clicking on **Project**, then **Select Language Toolsuite…**
7. In **"Step Three: Create new project, or reconfigure the active project"**, tick **Create New Project File** and choose the **ledsDemo** folder and name your project **ledsDemo**, click **Save**.
8. In **"Step Four: Add any existing files to your project"**, tick the three files you already created in step 2 of the procedure and click **Add >>** then **Next**.
9. Compile the project and you should get an **Output** that ends with **"Build Successful"** message.
10. Once step 9 is accomplished successfully, a **HEX** file will be generated within the project folder. You can use that **HEX** file to simulate your program and download it on a PIC device.

4.2 Simulation Using Real PIC Simulator

Sometimes before downloading or deploying the generated hex on PIC microcontroller, you should verify and ensure that the functionality of the program is working well as the requirements need. As a result of that, in this section, a simulator called **Real PIC Simulator** will be introduced to see how the program is simulated using the generated HEX file as an input. The Real PIC Simulator is usually used for simple simulations. Its weaknesses are that it doesn't provide any actual representation of the connections applied within the real-time project and that it lacks so many IC modules. Real PIC Simulator is used to provide beginners with so much basic approach to start from.

1. Open the **Real PIC Simulator**, then **load** the **HEX** file from **File** menu.
2. Go to **Visual** tab, and the drag and drop **LED Bar** inside the empty space.
3. Left-Click on any **LED** and then select the appropriate pin that you need.
4. Press **Play Icon** and observe the results.

4.3 Simulation Using Proteus ISIS

The Proteus ISIS is a program that is also used to run real-time simulation with providing ICs and connections details. The Proteus ISIS also provides a large library of modules such as sensors, timers, basic circuitry components… etc. In other words, it provides the developer with the capability of representing an actual real-time application at the simulation time with every single detail about the circuit implementation. The Proteus ISIS is used by professionals and it is highly recommended.

1. Open **Proteus.**
2. Click on the **ISIS** icon viewed within the provided toolbars.
3. Click on the **P** symbol on the left panel "the components panel". This panel is used to list all the components you intend to use within your project.
4. Pick the components CAP, CRYSTAL, LED-BLUE, PIC16F877A and RES as listed in Figure 2.

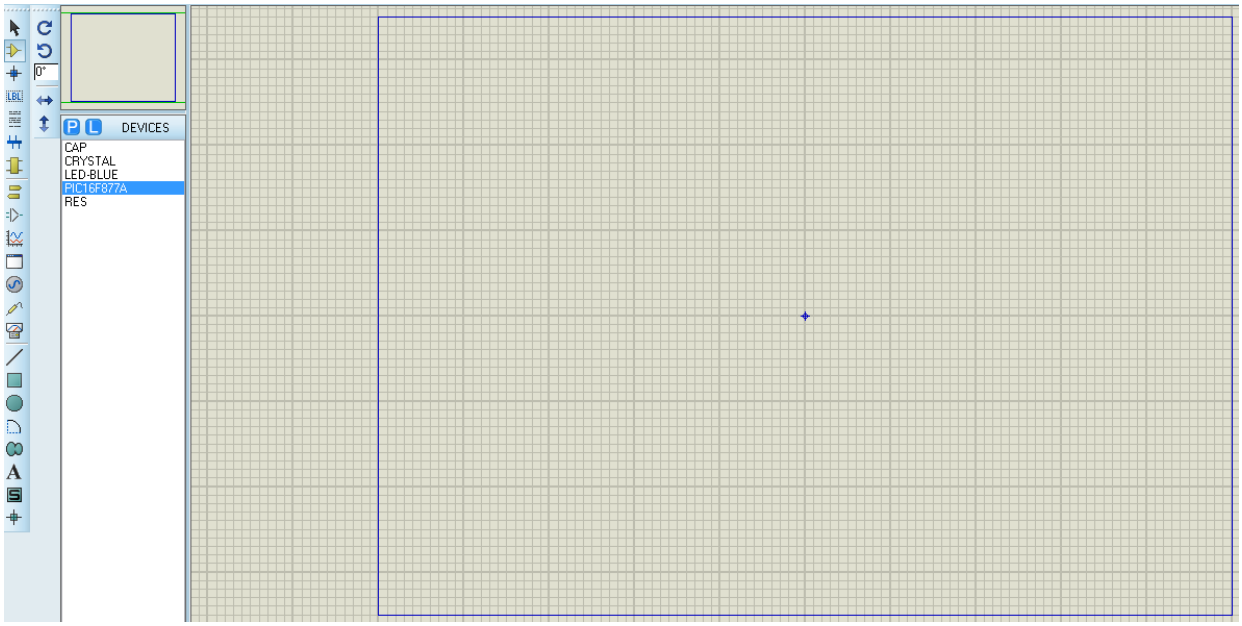

Figure 2: Proteus ISIS Components List

5. You can get POWER and GND from the "terminal mode" components as in Figure 3.

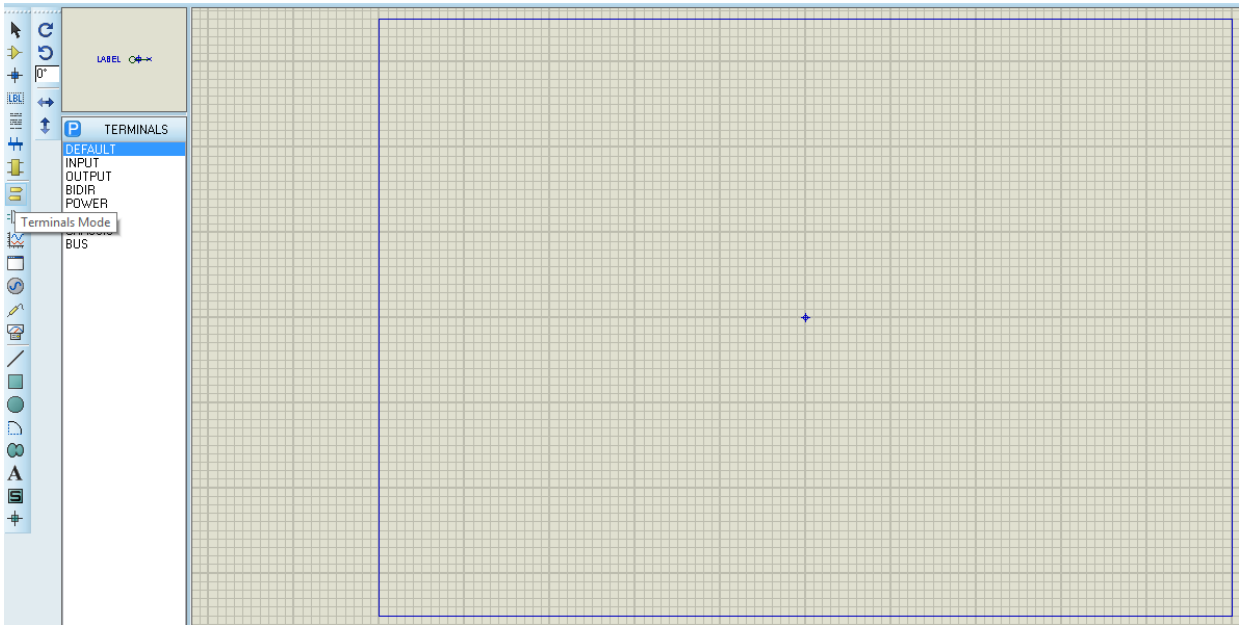

Figure 3: Proteus ISIS Terminal Mode

6. Apply the connections and components values as in Figure 4.

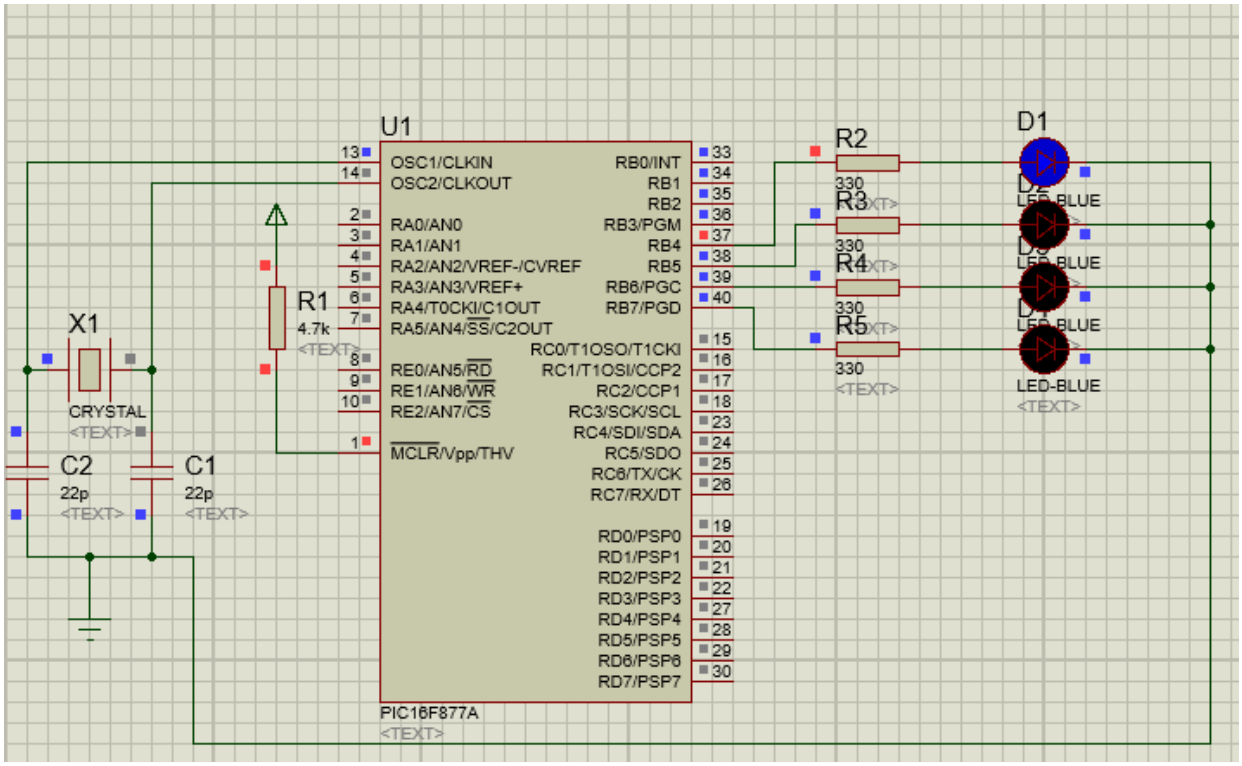

Figure 4: LEDs Interfacing

7. Double click on the PIC16F877A module, set the Program to ledf.hex found in the ledf project folder (automatically generated at compilation time). Set the clock frequency to 4MHz.
8. On the bottom-left corner, click on “play” button and observe the output you get.

4.4 Deploying Hex file on PIC Microcontroller

Once the program is simulated well, the last step is downloading the **HEX** file on PIC. In this section, a program called "**PICKit 3 Programmer**" will be introduced to show the deployment process on the PIC.

1. Connect the **PICkit3 USB Programmer** to PC first and then with **16F877A** PIC.
2. Run the **PICkit3** program as administrator.
3. Import the needed **HEX** file (File > Import). A message is shown to notify you that the HEX successfully imported.
4. Under **VDD PICkit 3 pane**, choose the voltage to be 5.0 and tick the **On** option.
5. Press **Write** button to begin downloading **HEX** file on PIC.

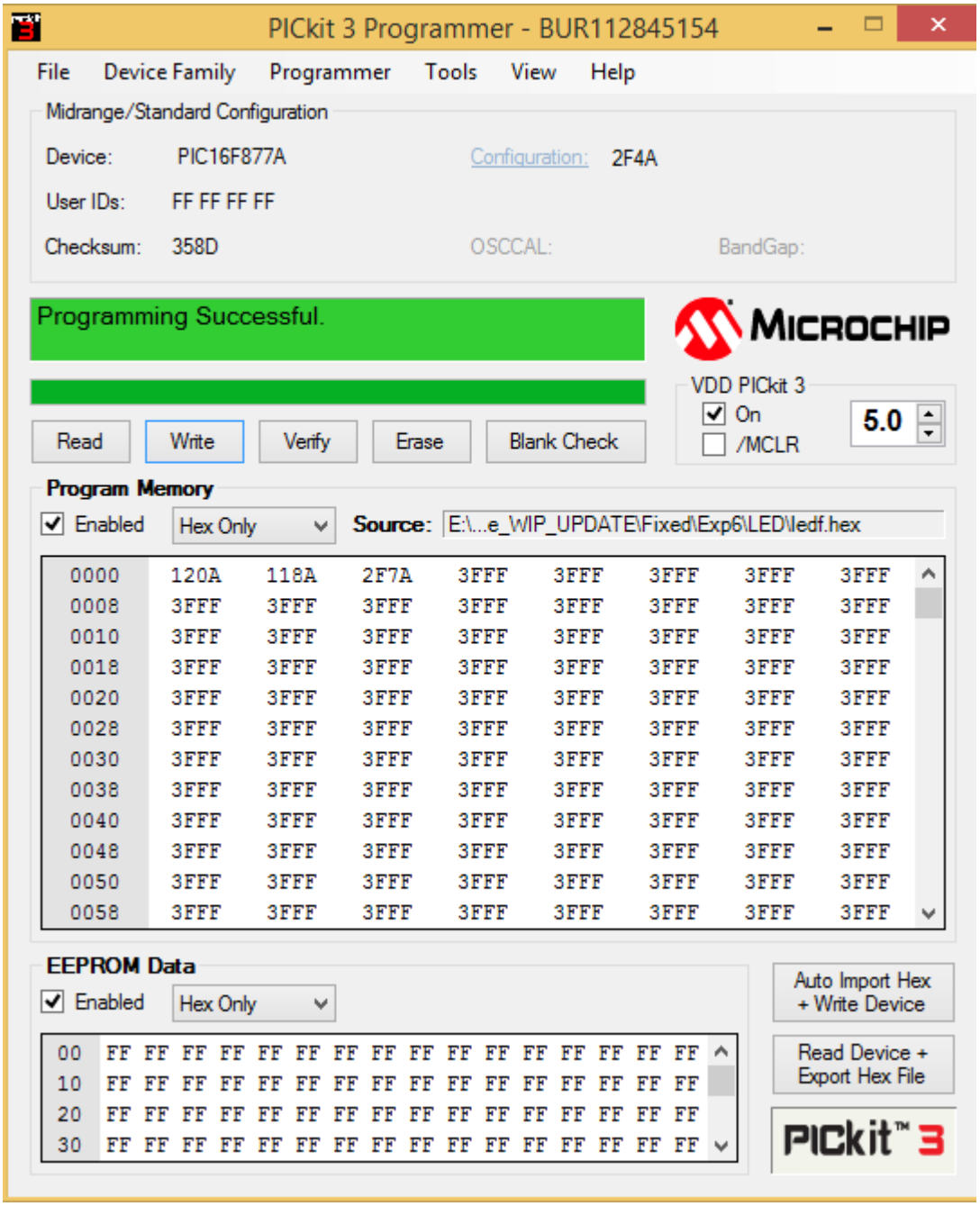

Figure 5: PICKit3 UI

4.5 Interfacing PIC Microcontroller Circuit

After finishing from deploying the HEX file on PIC, the last step is constructing the hardware circuit to run the program and see how it works on real time. Figure 7 shows the circuit to interface the PIC. It is clear that you need two **22pf capacitors, one 4MHz external crystal oscillator**, **one resistor**, and power supply to get **5 volts.** Figure 6 shows PIC **activation connections**.

1. Connect the circuit as shown in the **Figure 4**.
2. Connect LEDs with **330 Ohms** protecting resistors on the specified port.
3. Observe the output and compare it with the simulated one.

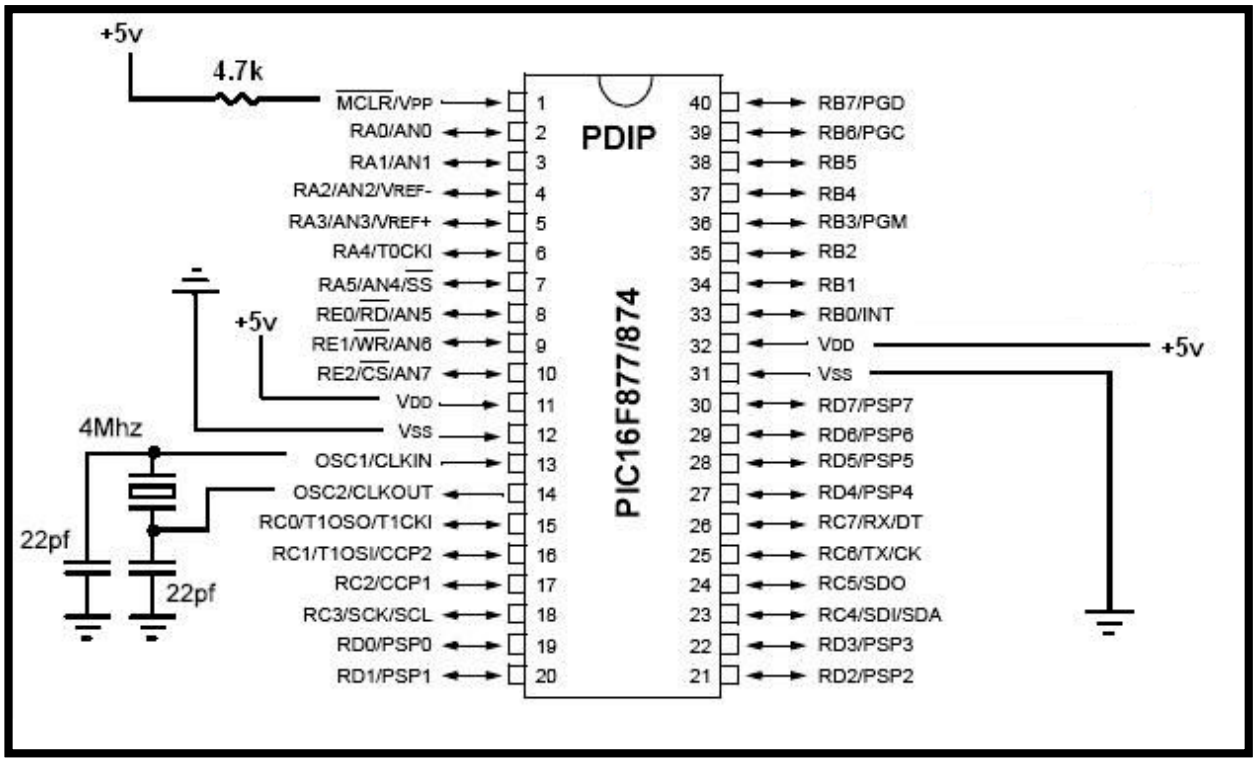

Figure 6: PIC16F877A Microcontroller Interfacing Circuit

4.6 Interfacing PIC Microcontroller Circuit with LCD

In this subsection, interfacing PIC microcontroller with LCD is introduced with code and schematic which shows the connections between PIC and LCD. But first, we need to explain the three control lines are referred to as EN, RS, and RW.

The EN line is called "Enable." This control line is used to tell the LCD that you are sending it data. To send data to the LCD, your program should make sure this line is low (0) and then set the other two control lines and/or put data on the data bus. When the other lines are completely ready, bring EN high (1) and wait for the minimum amount of time required by the LCD datasheet (this varies from LCD to LCD), and end by bringing it low (0) again.

The RS line is the "Register Select" line. When RS is low (0), the data is to be treated as a command or special instruction (such as clear screen, position cursor on a particular cell, etc.). When RS is high (1), the data being sent is text data which should be displayed on the screen. For example, to display the letter "T" on the screen you would set RS high.

The RW line is the "Read/Write" control line. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading) the LCD. Only one instruction ("Get LCD status") is a read command. All others are write commands. So RW will almost always be low.

Finally, the data bus consists of 4 or 8 lines (depending on the mode of operation selected by the user). In case of an 8-bit data bus, the lines are referred to as DB0, DB1, DB2, DB3, DB4, DB5, DB6, and DB7. In case of a 4-bit data bus, the lines are referred to as DB4, DB5, DB6, and DB7.

1. Create a new project using **MPLAB.**
2. Add the codes: **delay.h, delay.c, lcd.h, lcd.c, lcddemo.c** to it.
3. Compile the project and simulate it using **Proteus ISIS** (Add LM016L and POT-LN to the list of the LEDs program) and **Real Pic Simulator,** use connections as in **Figure7.**

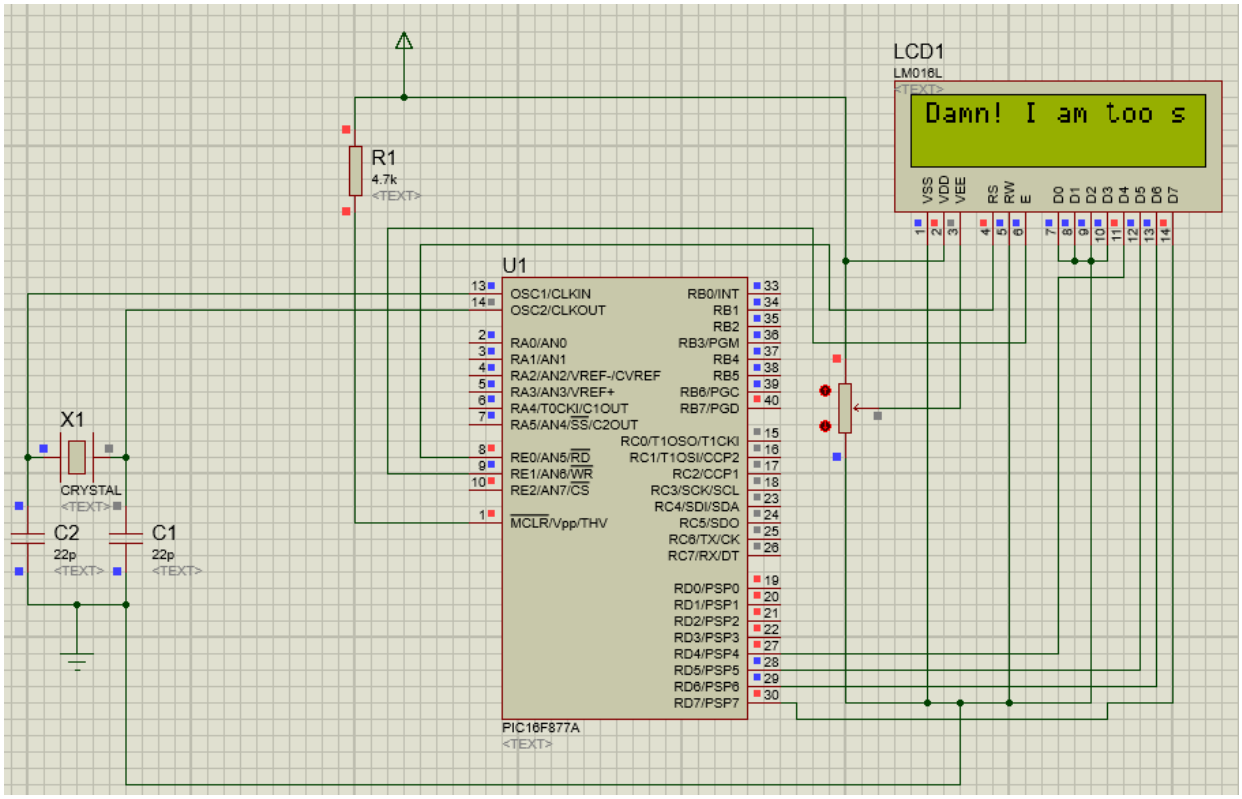

Figure 7: LCD Interfacing

1. The **LCD** must display the text **"Damn! I am smart since I use LCD".**
2. Download the program on the **PIC16f877A**
3. Apply the connections as in **Figure 7**.

5. Experiment Codes

Delay.h

```
/*
 *  Delay functions for HI-TECH C on the PIC
 *
 *  Functions available:
 *      DelayUs(x)  Delay specified number of microseconds
 *      DelayMs(x)  Delay specified number of milliseconds
 *
 *  Note that there are range limits: x must not exceed 255 - for xtal
 *  frequencies > 12MHz the range for DelayUs is even smaller.
 *  To use DelayUs it is only necessary to include this file; to use
 *  DelayMs you must include delay.c in your project.
 *
 */

/*  Set the crystal frequency in the CPP predefined symbols list in
    HPDPIC, or on the PICC commmand line, e.g.
    picc -DXTAL_FREQ=4MHZ

    or
    picc -DXTAL FREQ=100KHZ

    Note that this is the crystal frequency, the CPU clock is
    divided by 4.

 */

#ifndef XTAL_FREQ
#define XTAL_FREQ   4MHZ        /* Crystal frequency in MHz */
#endif

#define MHZ *1000L          /* number of kHz in a MHz */
#define KHZ *1          /* number of kHz in a kHz */

#if XTAL_FREQ >= 12MHZ

#define DelayUs(x)  { unsigned char _dcnt; \
              _dcnt = (x)*((XTAL_FREQ)/(12MHZ)); \
              while(--_dcnt != 0) \
                  continue; }
#else

#define DelayUs(x)  { unsigned char _dcnt; \
              _dcnt = (x)/((12MHZ)/(XTAL_FREQ))|1; \
              while(--_dcnt != 0) \
                  continue; }
#endif

extern void DelayMs(unsigned char cnt);
```

Delay.c

```
/*
 *  Delay functions
 *  See delay.h for details
 *
```

```
 *  Make sure this code is compiled with full optimization!!!
 */

#include "delay.h"

void DelayMs(unsigned char cnt)
{
#if XTAL_FREQ <= 2MHZ
    do {
        DelayUs(996);
    } while(--cnt);
#endif

#if    XTAL_FREQ > 2MHZ
    unsigned char   i;
    do {
        i = 4;
        do {
            DelayUs(250);
        } while(--i);
    } while(--cnt);
#endif
}
```

Ledf.c

```
#include <pic.h>
#include "delay.h"

__CONFIG(DEBUG_OFF & WDTE_OFF & LVP_OFF & FOSC_HS & BOREN_ON);

int i,j;
void pause(int d);

void main(void)
{
 TRISB = 0; // set port B to output
 PORTB=0;//set zero to output

 while (1) {

 PORTB=0b10000000; //set port B7 to 1
 pause(500);       //delay

 PORTB=0b01000000; //set port B6 to 1
 pause(500);       //delay

 PORTB=0b00100000; //set port B5 to 1
 pause(500);       //delay

 PORTB=0b00010000; //set port B4 to 1
 pause(500);

 }

}

void pause(int d)
{
   for(i=0;i<20;i++)
```

```
    for(j=0;j<d;j++)
        ;
}
```

Lcd.h

```
extern void lcd_write(unsigned char);
extern void lcd_clear(void);
extern void lcd_puts(const char * s);
extern void lcd_goto(unsigned char pos);
extern void lcd_init(void);
extern void lcd_putch(char);
#define lcd_cursor(x) lcd_write(((x)&0x7F)|0x80)
```

Lcd.c

```
/*
*   the list of included files contains:
*   pic.h since we're going to use it with our pic microcontroller, duah
*   ldc.h which contains all the prototypes of the functions used for the
lcd
*   delay.h which contains the prototype of the melli-second delay and the
implementation
*   of the micro-second delay (which is used in the melli-second
implementation, so it has to be included)
*   delay.c contains the implementation of the delay_ms function, so it has
to be here
*/

#include <pic.h>
#include "lcd.h"
#include "delay.h"
#define LCD_STROBE ((RE1 = 1),(RE1=0))  //* The E bit on the lcd, where it
tells the lcd that we're writing data to it when it's set to 1*/
/*
*   the following write functions take a character inpute or 8 bits where
it takes the higher 4 bits and passes their values to the D port of the pic
*   then it shifts the character by 4 bits to the left in order to take the
values of the lower 4 bits and put them on the used D port bits. The strobe
*   is used indicate that the LCD is receving data so that the values are
guaranteed to be passed in order and without interference
*
*   As for the RS bit which is connected to RE0. It is used to tell the LCD
to accept the character as a command when it's set to 0, or as a character
to
*   be displayed on the screen when it's set to 1
*/
void lcd_write(unsigned char c)
{
PORTD = (PORTD & 0x0F) | (c);
LCD_STROBE;
PORTD = (PORTD & 0x0F) | (c << 4);
LCD_STROBE;
DelayUs(40);
}
void lcd_clear(void)
{
RE0 = 0;
lcd_write(0x1);
DelayMs(2);
```

```
}
void lcd_puts(const char * s)
{
RE0 = 1;
while(*s)
lcd_write(*s++);
}
void lcd_putch(char c)
{
RE0 = 1;
PORTD = (PORTD & 0x0F) | (c);
LCD_STROBE;
PORTD = (PORTD & 0x0F) | (c << 4);
LCD_STROBE;
DelayUs(40);
}
void lcd_goto(unsigned char pos)
{
RE0 = 0;
lcd_write(0x80+pos);
}
void lcd_init(void)
{
RE0 = 0;
DelayMs(15); // power on delay
PORTD = (0x3 << 4);
LCD_STROBE;
DelayMs(5);
LCD_STROBE;
DelayUs(100);
LCD_STROBE;
DelayMs(5);
PORTD = (0x2 << 4);
LCD_STROBE;
DelayUs(40);
lcd_write(0x28); // 4 bit mode, 1/16 duty, 5x8 font
lcd_write(0x08); // display off
lcd_write(0x0F); // display on, blink curson on
lcd_write(0x06); // entry mode
}
```

Lcddemo.c

```
#include <pic.h>
#include "lcd.h"
#include "delay.h"
/*
*   since we're using 4MHz oscillator, we need to set the oscillator bit to
HS
*   the following are needed to make the LCD get to work:
*   BODEN, WDTDIS and WRTEN
*/

__CONFIG(DEBUG_OFF & WDTE_OFF & LVP_OFF & FOSC_HS & BOREN_ON);
int i,j;
void pause(int d);
void main(void)
{
```

```
TRISE = 0; //set PORT E to output mode
TRISD = 0; //set PORT D to output mode
TRISB = 0; //set PORT B to output mode
TRISA = 0; //set PORT A to output mode
ADCON1 = 7;//set PORT A to Digital mode
RE0 = 0;// set PORT E0 to zero
pause(1); //sleep for 1 sec
lcd_init(); //intialiaze LCD

while (1)
{
    lcd_clear(); //Clear LCD
    lcd_goto(0);  // move LCD's cursor to location zero
    lcd_puts("Damn! I am too smart"); //put string on LCD
    lcd_goto(40);//move the cursor
    lcd_puts("Since I use LCDs :-)"); //put String

    PORTB=0b10000000;//set PORT B7 to 1 , Just for testing issue
    pause(1);// //wait for 1 sec, Testing Issue
    PORTB=0b00000000;// sec PORT B7 to zero , Testing issue
    pause(1);//wait for 1 sec
}
}

//Function to delay or sleep to number of seconds, based on the value of
integer d
void pause(int d)
{
   for(i=0;i<4;i++)
    for(j=0;j<d;j++)
        DelayMs(255);
}
```