DEVELOPMENT KIT of PIC16f877a MCU DOCUMENTATION

List of Contents

1. Description of

- Introduction
- Pin Diagram
- Specifications
- Application

2. Installing Software

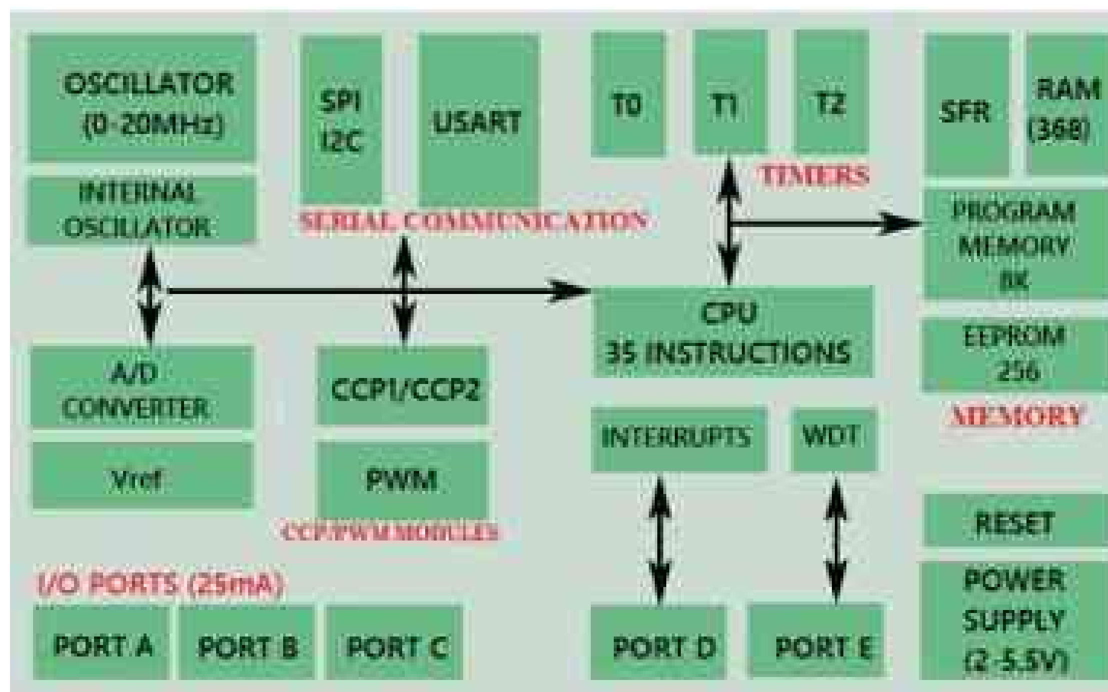
- MPLAB IDE
- XC8 Compiler
- PICkit2
- Programmer

3. Examples

- LED
- Seven Segment Display
- Buzzer
- LCD
- Relay
- Stepper Motor
- Servo Motor
- DC Motor
- RTC
- Multiplexed Display
- Keypad
- Rs232

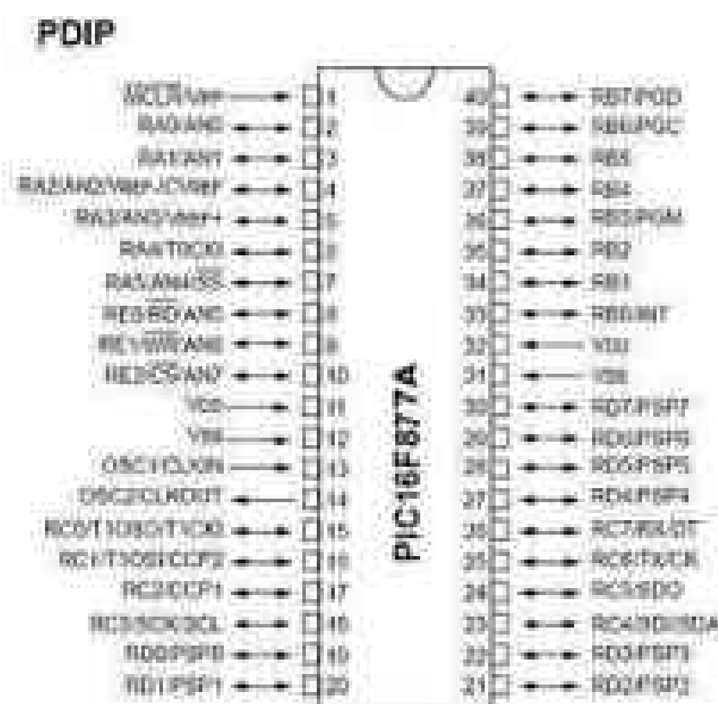
Introduction

The PIC microcontroller from Microchip is one the famous and most used microcontrollers. Because of its reliability it is commonly preferred by embedded engineers for industrial applications. The PIC microcontroller PIC16f877a is one of the most renowned microcontrollers in the industry. This microcontroller is very convenient to use, the coding or programming of this controller is also easier. The PIC16F877A Microcontroller consists of an inbuilt CPU, I/O ports, memory organization, A/D converter, timers/counters, interrupts, serial communication, oscillator and CCP module which to gather makes the IC a powerful microcontroller for beginners to start with. The general block diagram of the PIC Architecture is shown below.



One of the main advantages is that it can be write-erase as many times as possible because it uses FLASH memory technology. It has a total number of 40 pins and there are 33 pins for input and output. PIC16F877A is used in many pic microcontroller projects. PIC16F877A also have much application in digital electronics circuits.

Pin Diagram



Specifications

- 8K of Code space
- 256 Bytes of EEPROM
- 384 bytes SRAM
- 8-level deep hardware stack
- Up to 20 MHz clock
- 1 16-bit, 2 8-bit timers

- Synchronous Serial Port – SPI and I2C
- USART
- 8 channel, 10-bit ADC
- Brown-Out Reset
- 2 Analog Comparators
- Capture, Compare, PWM module

Application

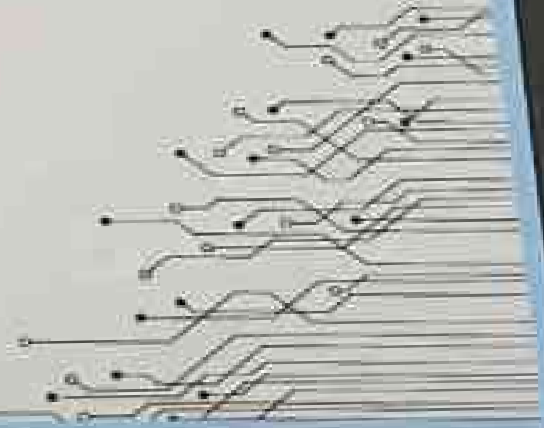
PIC16F877A also have much application in digital electronics circuits. PIC16f877a finds its applications in a huge number of devices. It is used in remote sensors, security and safety devices, home automation and many industrial instruments.

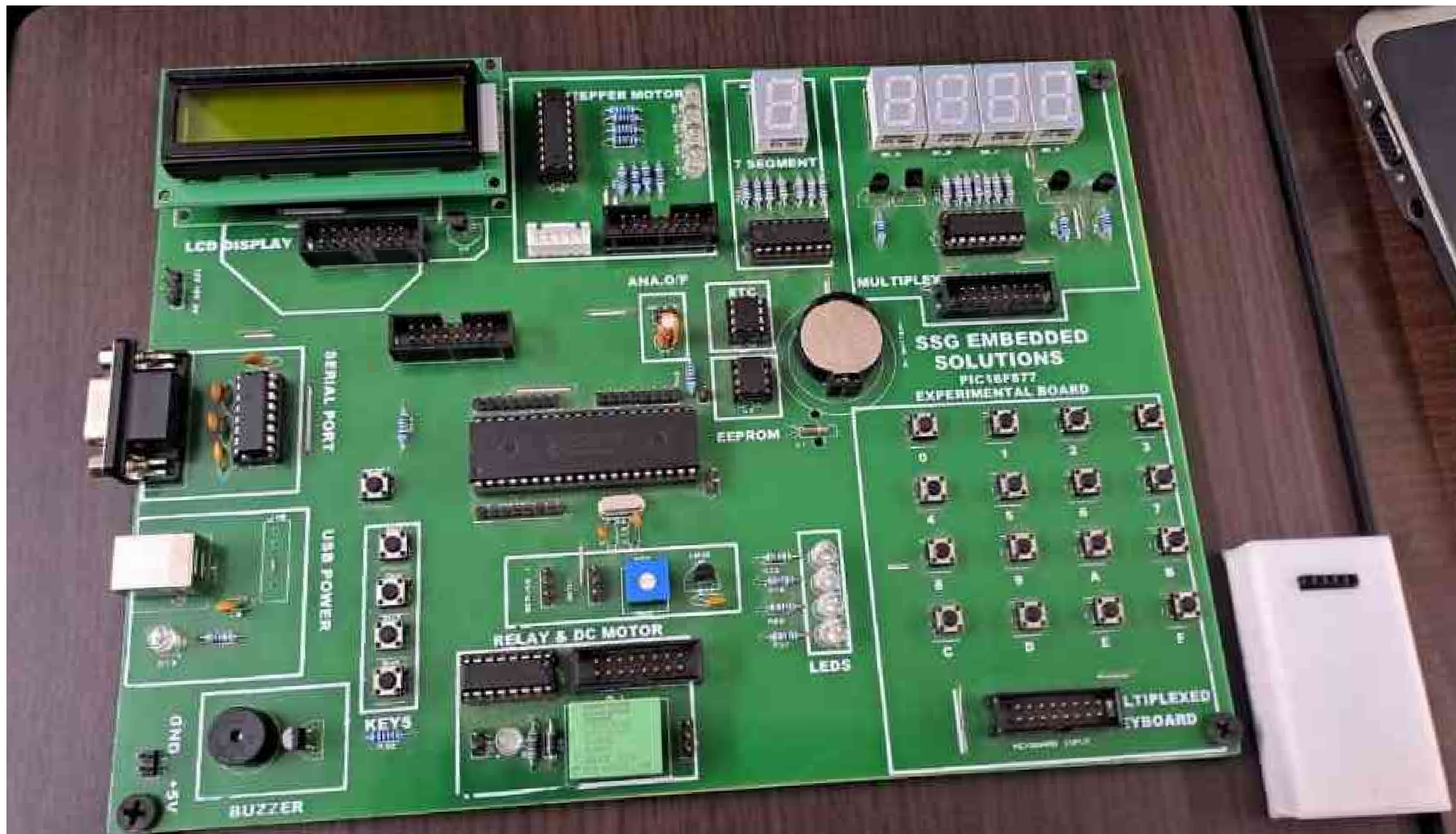
SSG Embedded Solutions, Nagpur

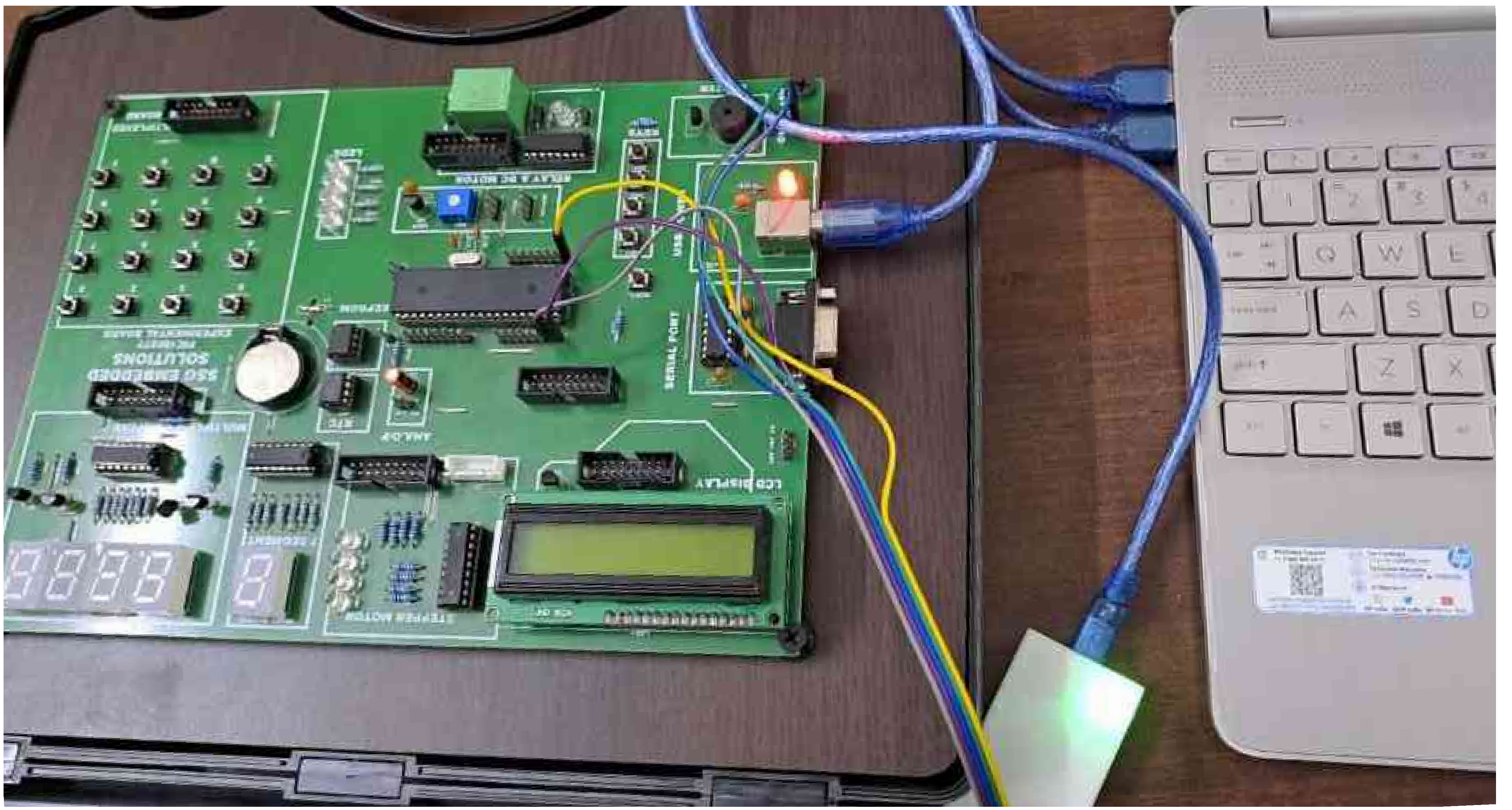
Email: info@ssges.co.in
Office address: Plot No. 25, Vashistha, NIT Layout Swavalambi Nagar, Nagpur-25

PIC MICROCONTROLLER Development Board

(Model No: SSG-PIC)



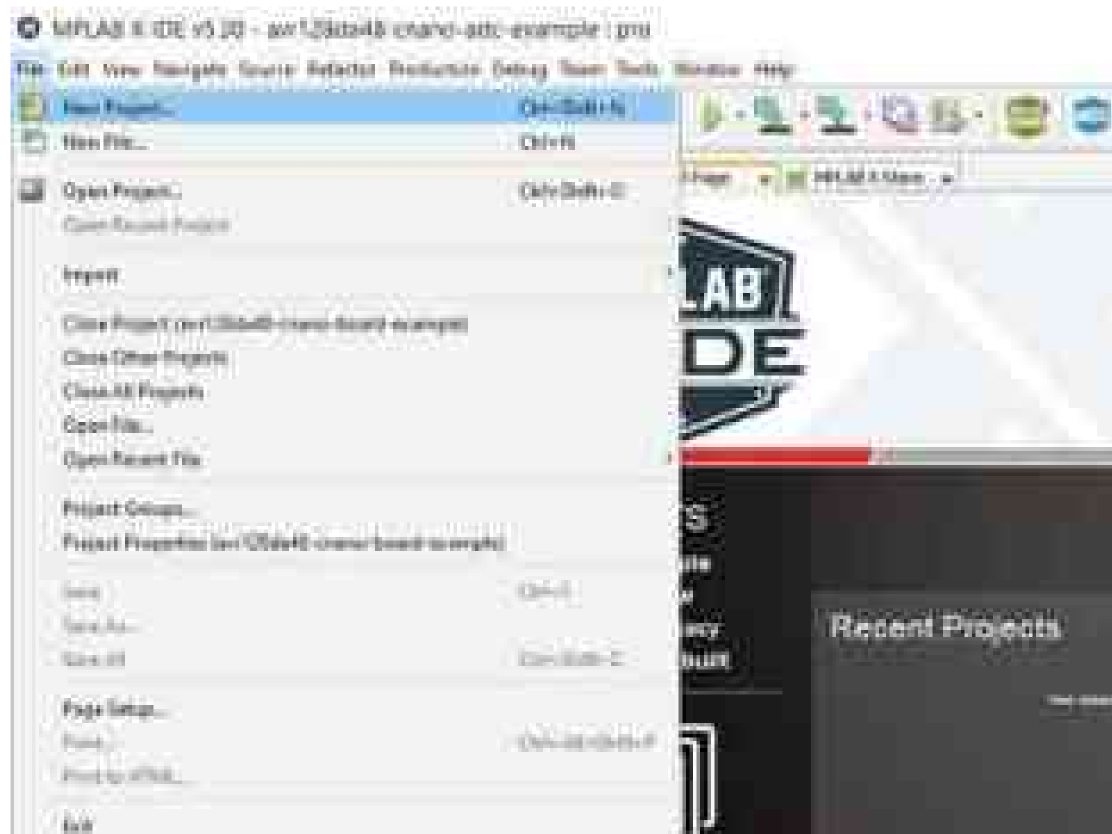




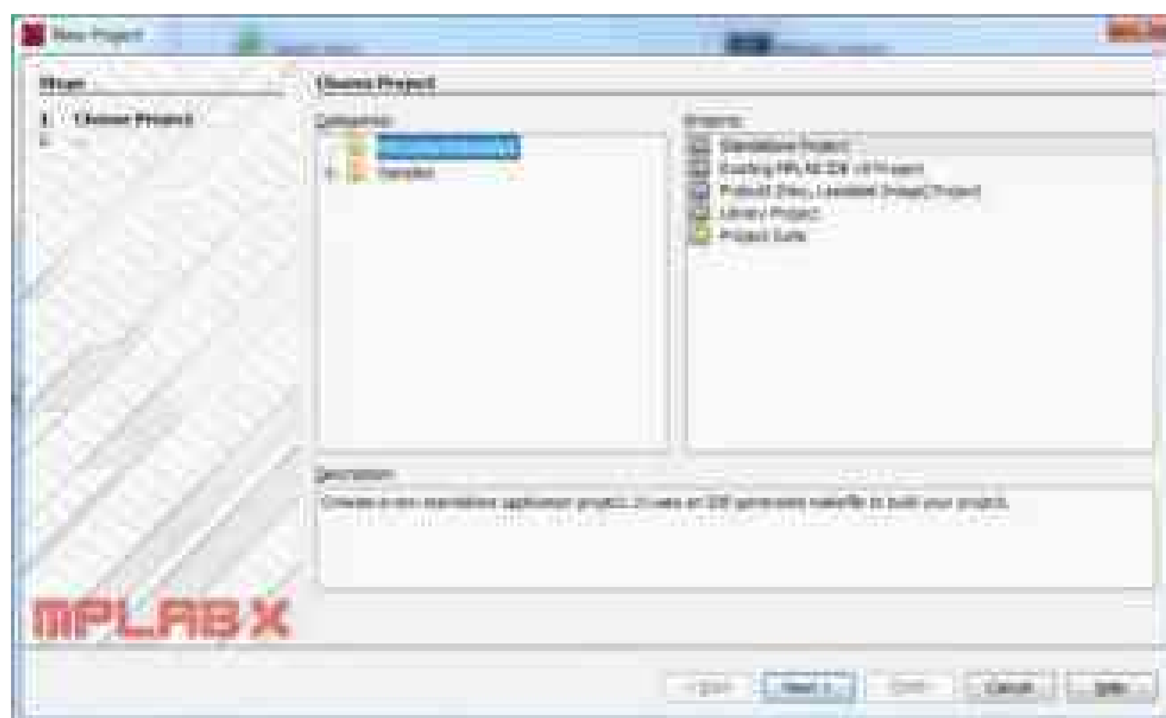
MPLAB IDE

Step 1: The first step is to run the MPLAB IDE.exe file on your desktop or whatever the installation path is.

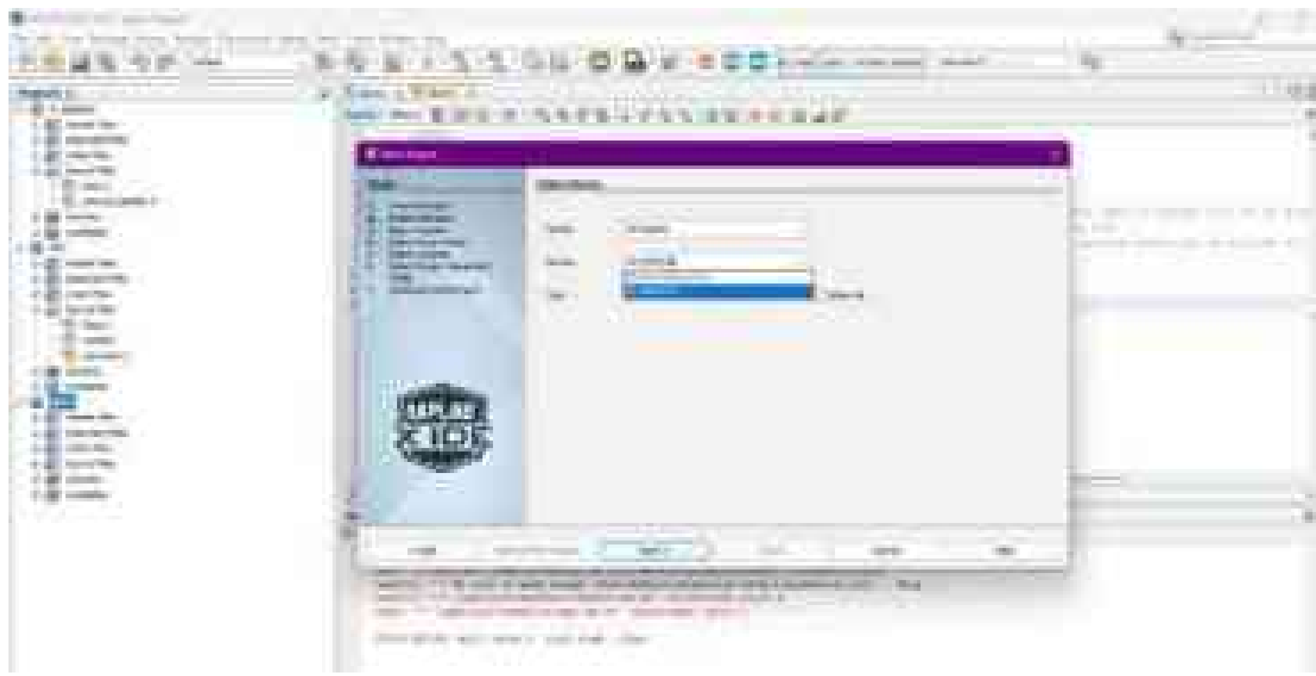
Step 2: From the “Project” choose “New Project”, Choose Embedded Standalone Project. Then Next.



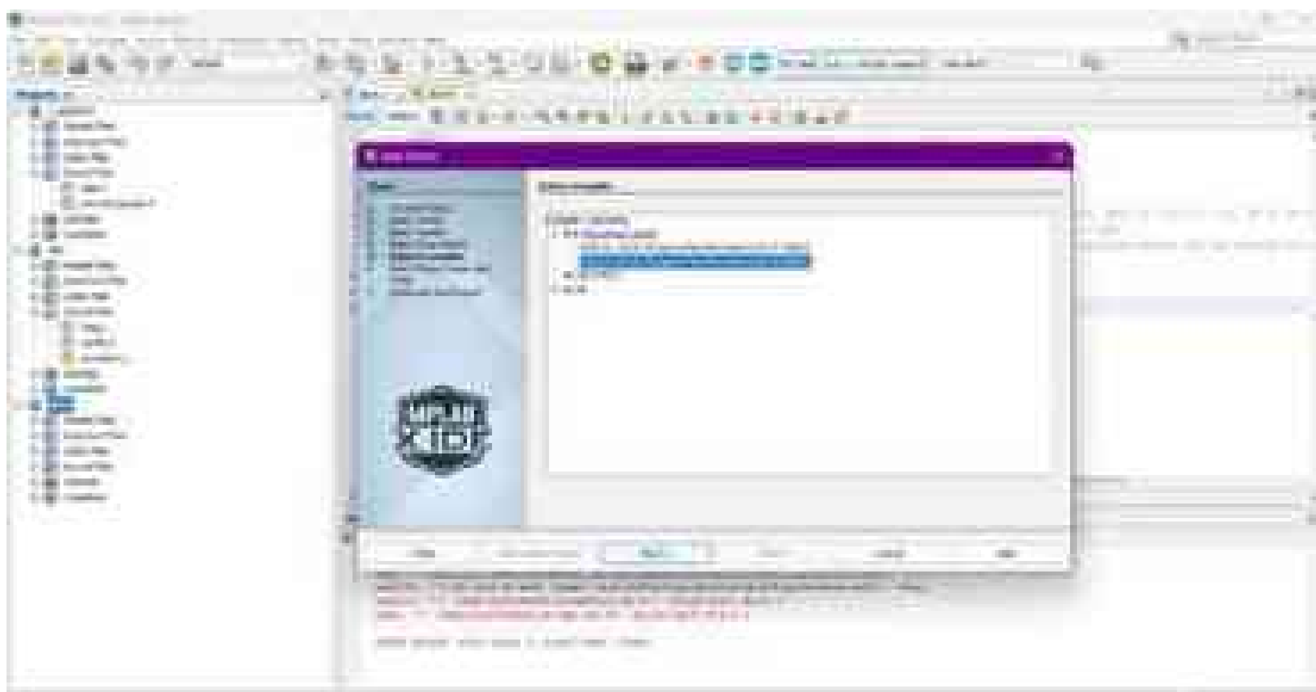
Step 3: Choose the family of the MCU which is 8-bit Mid-range. Then write the name of the chip in the box below which will be PIC16F877A. Then Next.



Step 4: Here you’ll choose the debugging hardware tool for your project Pick any one then click Next.

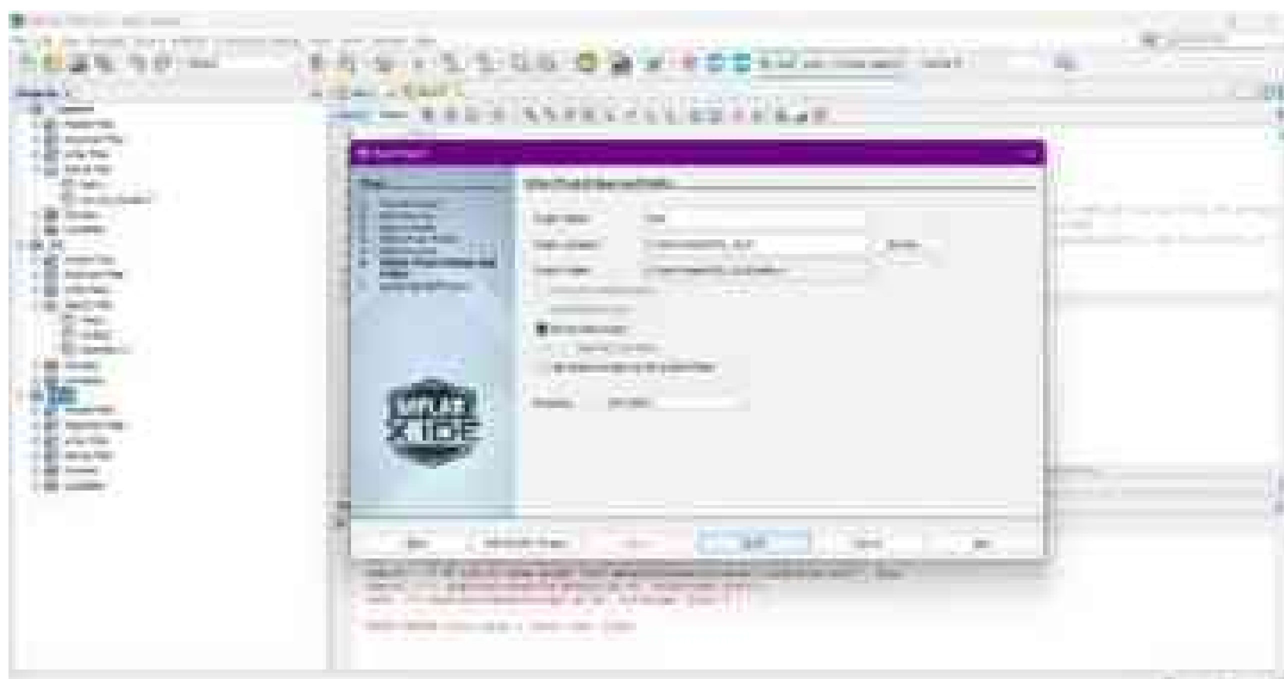


Step 5: Choose the compiler for your project. We’ll be using XC8 compiler for our projects. Then Next.



Step 6: Choose the path to save your project into. And give your project a relevant name

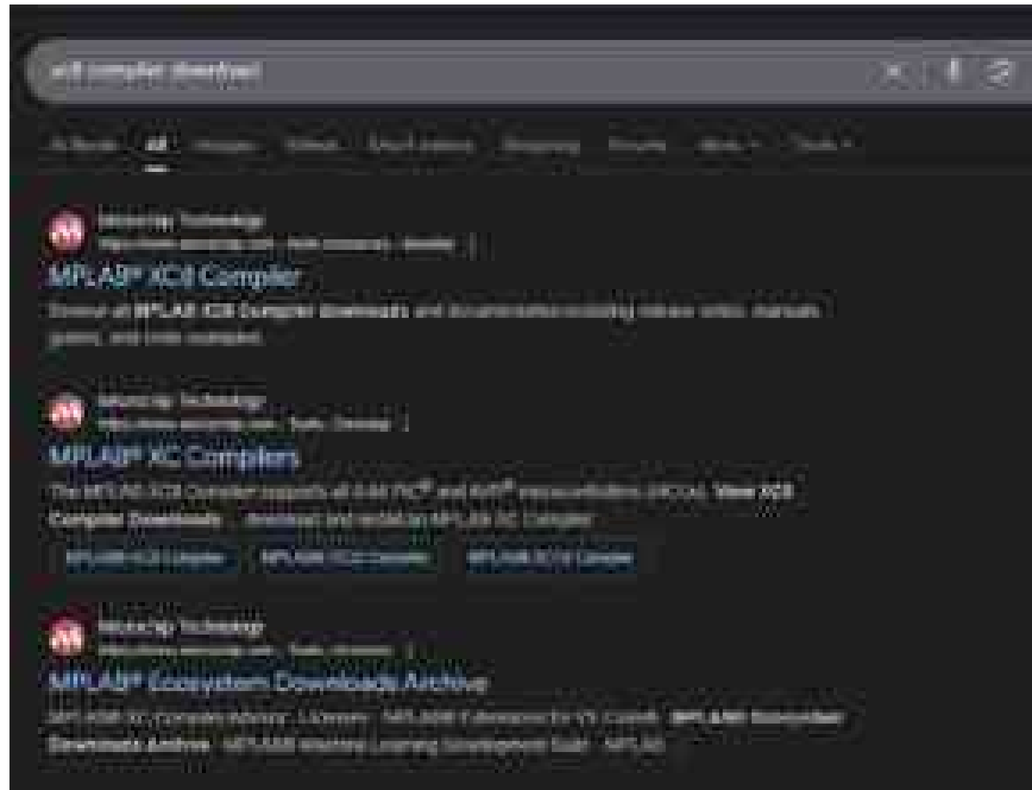
Click Finish and you should see something like this screen down below



Step 7: Now, let’s create the file in which we’ll write our source C-Code. Right-Click on the source files and choose to create a new main.c file. And give it a relevant name. It’s usually named as main.c.

XC8 Compiler

Step 1: Open browser and search XC8 compiler



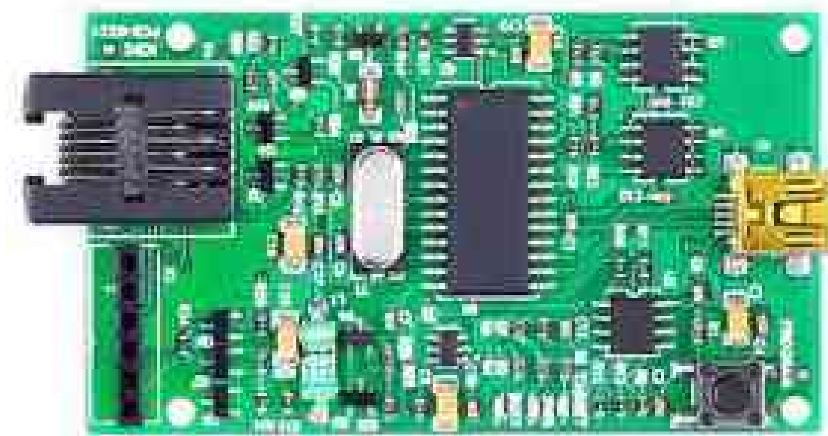
Step 2: Click on MPLAB XC8 compiler

Step 3: After downloading open xc8 compiler

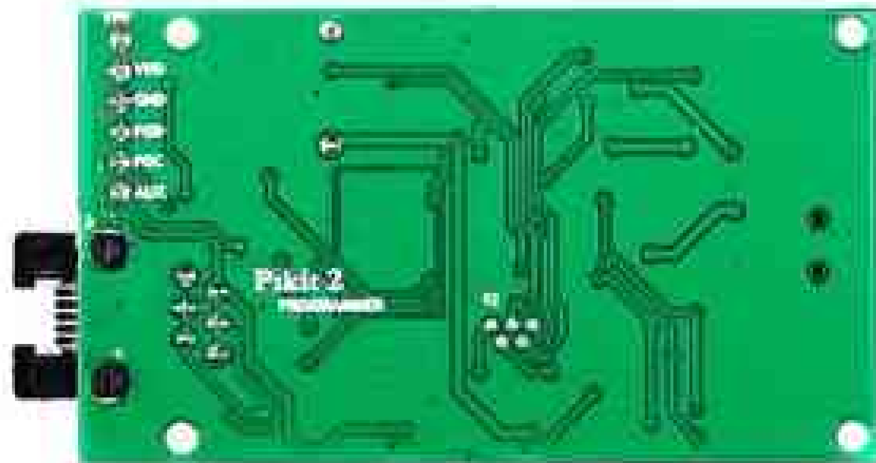
Step 4: ready to install

PICkit2 Programmer

PICkit2 Programmer is a low cost MPLAB Compatible PIC programmer. When connected, the MPLAB IDE detects it automatically. It can program PIC controllers operating at 3V3 and 5V.



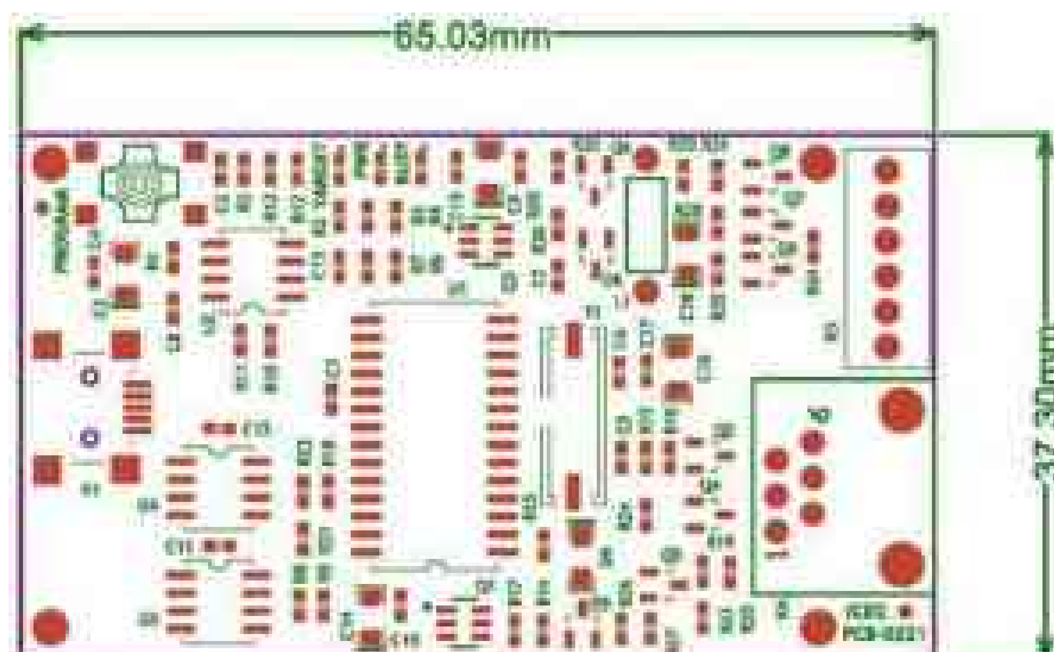
The PICkit2 Development Programmer/Debugger, a full featured Windows programming interface, supports baseline (PIC10F, PIC12F5xx, PIC16F5xx), mid rang (PIC12F6xx, PIC16F) ,PIC18F, PIC24, dsPIC30, dsPIC33,and PIC32 families of 8-bit, 16-bit, and 32-bit microcontrollers, and many Microchip Serial EEPROM products. With Microchip's powerful MPLAB IDE (Integrated Development Environment), the PICkit 2 enables in-circuit debugging on most PIC microcontrollers. In-Circuit-Debugging runs and halts the program on break points while the PIC microcontroller is embedded in the application. When halted at a breakpoint, the file registers can be examined and modified. Using pickit2, we can use about 95% of the controller memory. If we use boot-loader to program, then we will have to reserve separate memory just to fuse boot-loader.



Features

- USB Connection (cable Included)
- Automatic voltage switching (3V3 and 5V)
- Does not require external power supply
- 100% compatible with Microchip's MPLAB IDE (Pickit2)
- Compact and handy design
- Both RJ11 connector and ICSP 6PIN Connector for programming.
- Supports PIC controllers operating at both 5V and 3V

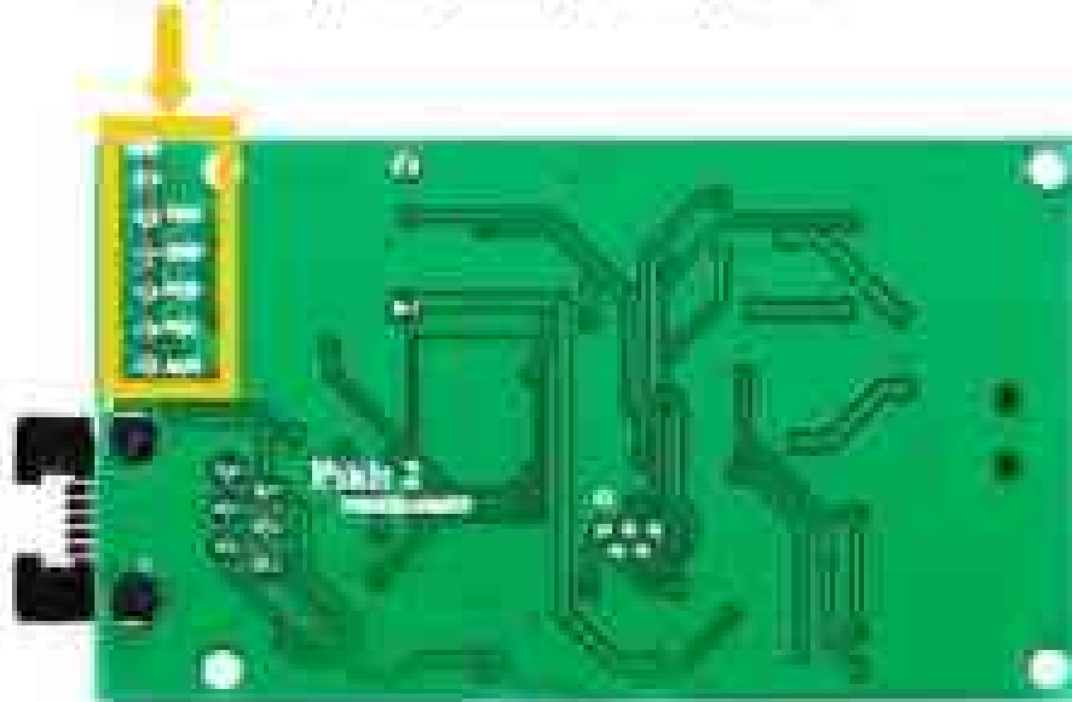
Layout



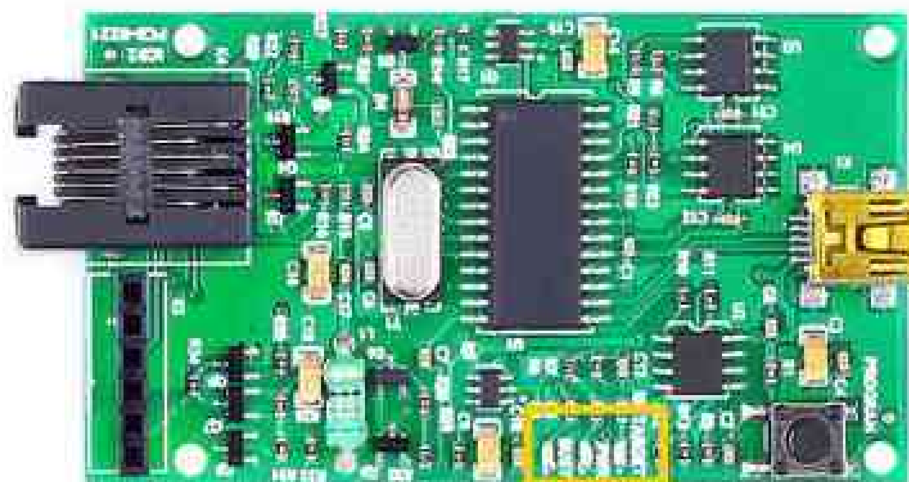
The PIC Programmer uses a USB port, for both powering and connecting to the computer, as opposed to the traditional serial port, eliminating the need for a separate power supply. **It will automatically power the circuit, if the target controller is not powered**. The programmer can be used to program 3.3V and 5V micro controllers. Automatic selection of voltage level is one of the unique features of this programmer.

PIN Description

PICkIT 2 Programmer Pin	Target UART
(1) VPP	MCLR/Reset
(2) Vdd	Vdd (Vcc)
(3) GND	GND
(4) PGD	TX - inverted, logic level
(5) PGC	RX - inverted, logic level
(6)AUX	



- **TARGET** : The PIC Programmer Lite is powering the target device
- **PWR** : Power is applied to the PIC Programmer Lite via the USB port
- **BUSY** : The PIC Programmer Lite is busy with a function in progress, such as programming

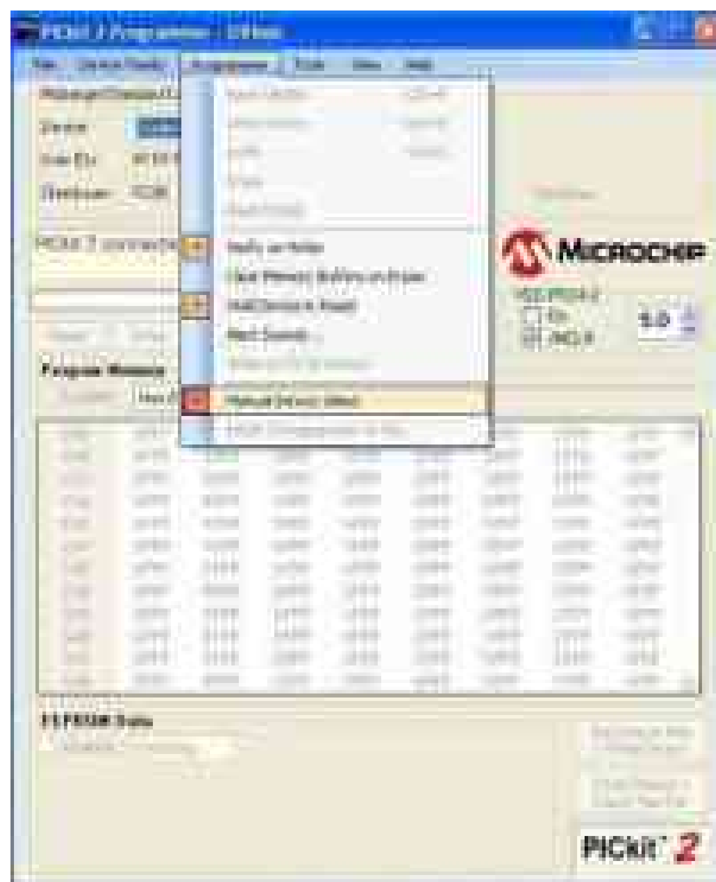


Power up the **PICKIT2 Programmer** , **PWR LED** glows.

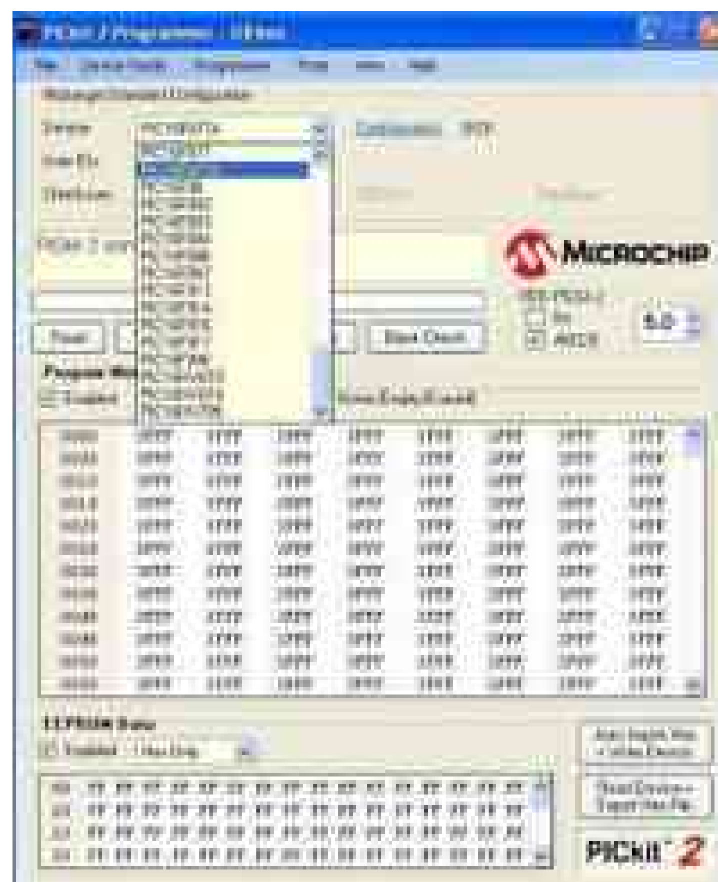
Selecting PICKIT2 Programmer as a Programmer tool

Step 1: PICkit 2 Programmer opens as follows,(click here to download)

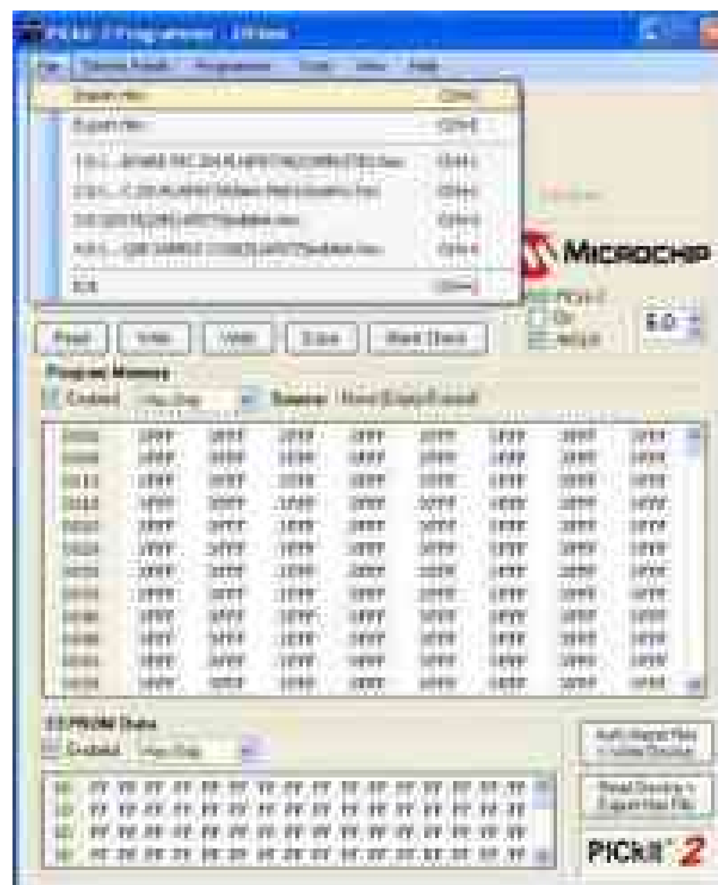
- **Step 2:** Select the Programmer and set the options given below
  Programmer >Verify on Write
- Programmer >Hold Device in Reset
- Programmer >Manual Device Select



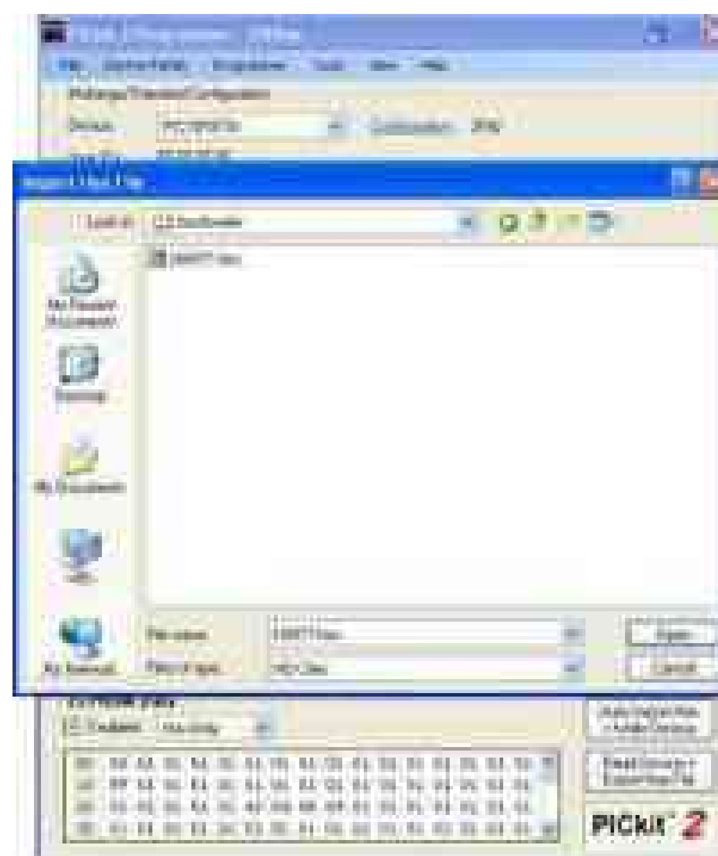
Step 3: Select the device from the drop down list



Step 4: Import the hex file from **File > Import Hex**



Step 5: A small window opens up, select the hex code of the bootloader program(or hex code of required program) from the window and click ‘**open**‘

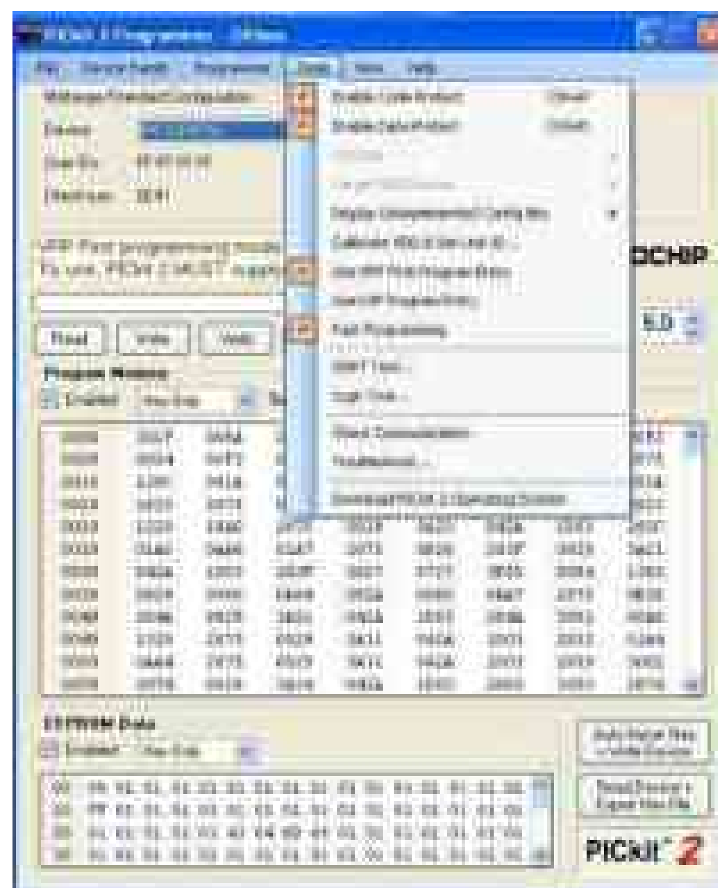


Step 6: The path can be viewed in **Source** .Once loaded, window displays hex file successfully imported as shown below

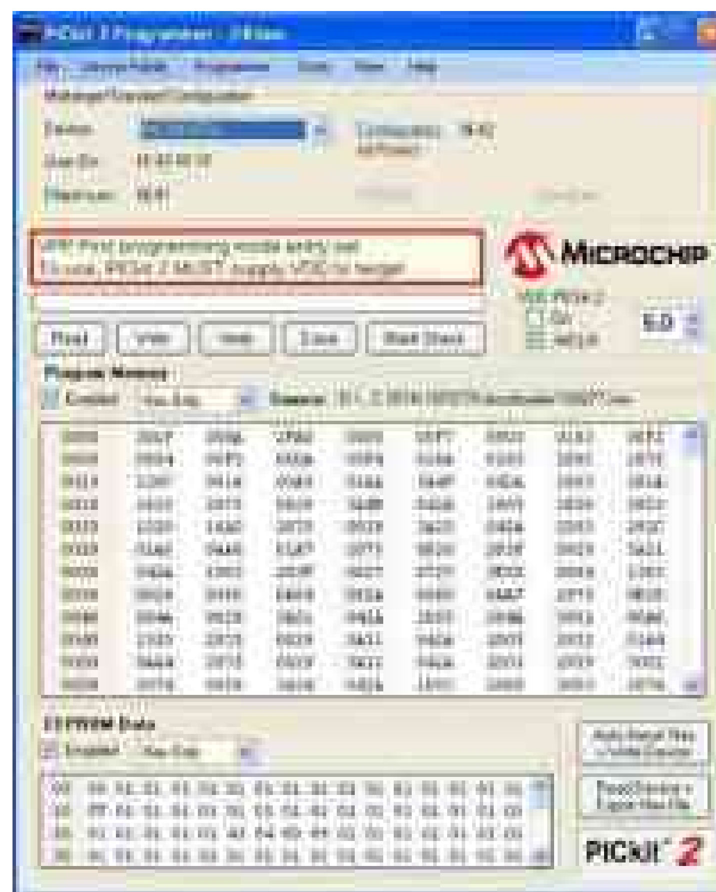
Step 7: Select the Tool and set the following options.

- Tools >Enable Code Protect
- Tools >Enable Data Protect
- Tools >Use VPP First Program Entry
- Tools >Fast Programming

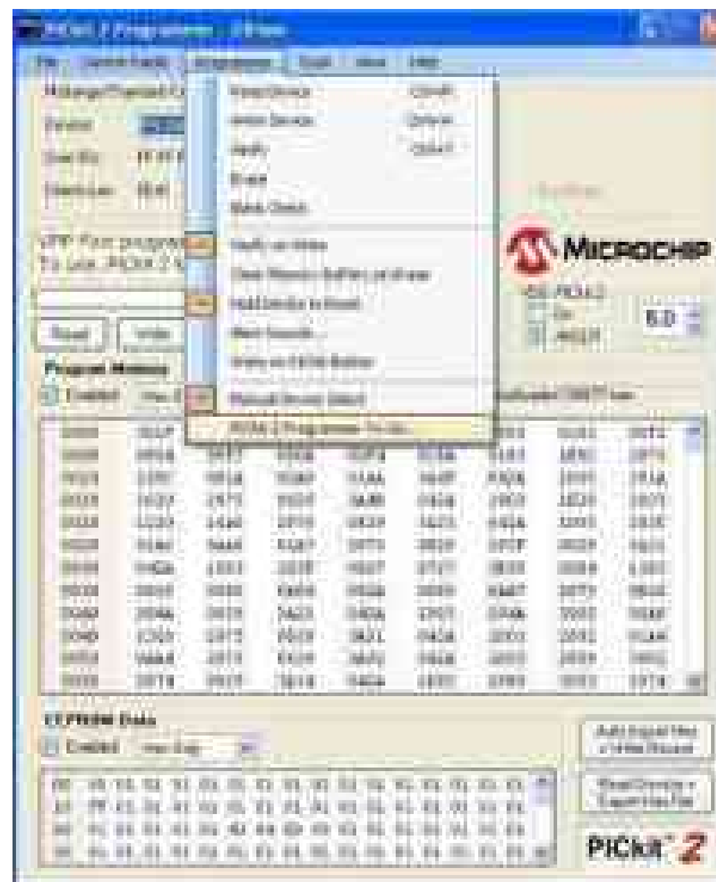
The “VDD PICkit 2” device VDD box determines the provided VDD voltage (when the device is powered from PICkit2).



Step 8: After setting the Tools options it will be displayed shown in the image below



Step 9: Once the memory image and programming options for intended target device are set up and tested, start the Programmer-To-Go Setup Wizard via **Programmer > PICkit 2 Programmer-To-Go...**

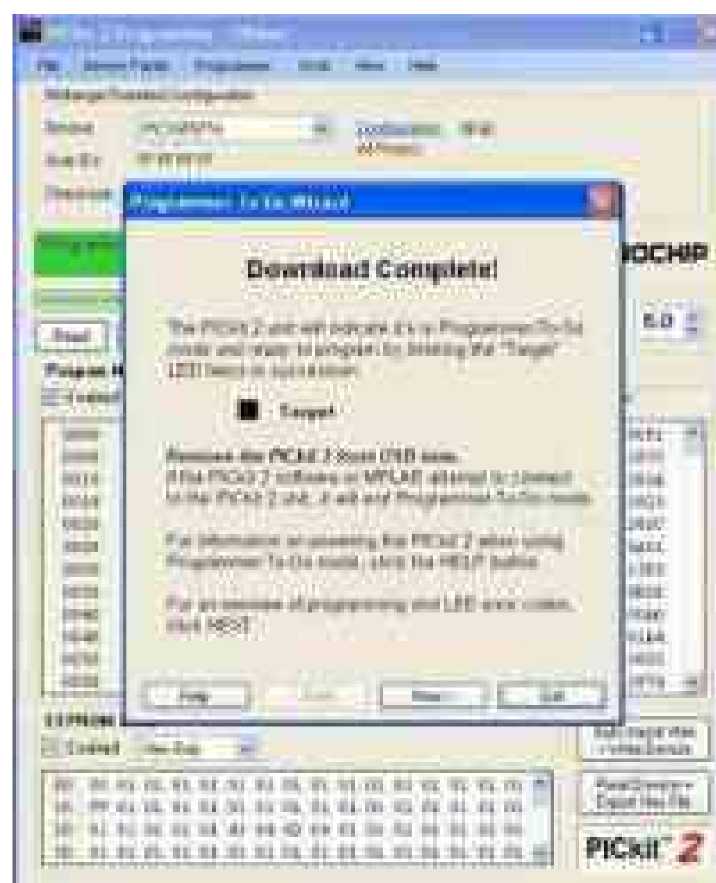


Step 10: The wizard dialog box opens up with a welcome screen. Click **Next** to go to the "Programmer Settings" screen.

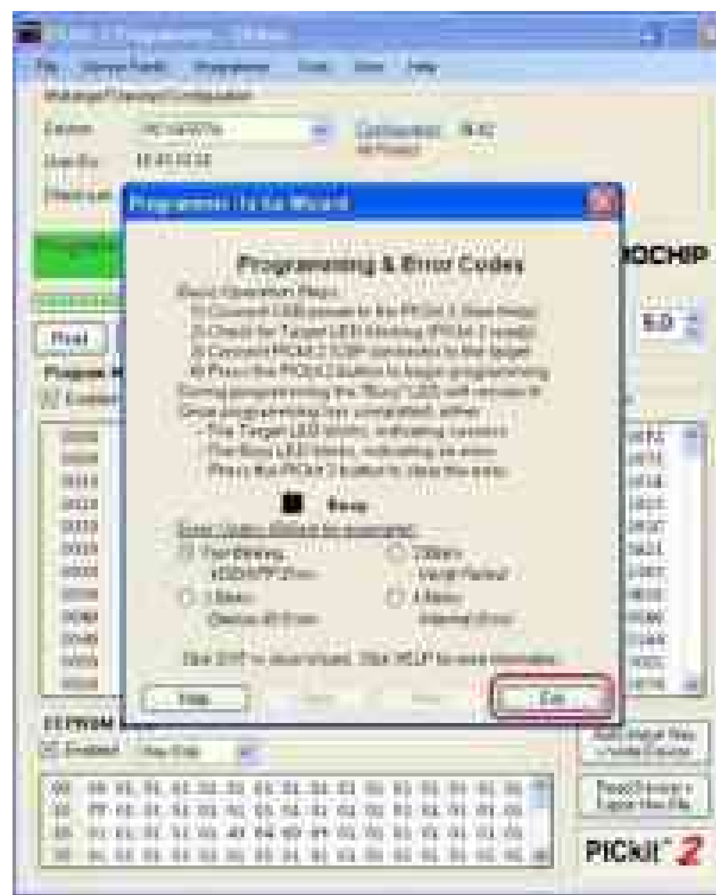
Step 11: The Programmer Settings screen allows the user to verify the memory image buffer settings, and also to select the target VDD power options to be used. Check it and click '**Next**'.

Step 12: Now the "Programmer to go-Wizard" window will provide a summary of the settings from the previous window, indicating whether the "**Tools >Fast Programming and Programmer >Hold Device in Reset"** are enabled or not. For most situations, both these options should be enabled. Click the **Download** button to store the memory image and settings in the PICkit 2 unit and place it in Programmer-To-Go mode.

Step 13: When we go for Download operation the PICkit 2 “Busy” LED will remain lit continuously. The “Target” LED should now be blinking twice in succession to indicate that the PICkit2 is in Programmer-To-Go mode and it is ready to program. Disconnect the PICkit 2 from the PC USB port. When any USB power source is applied, the PICkit 2 unit will power up in Programmer-To-Go mode and “Target” LED will blink. Click **Next** to view the wizard screen with examples of Programmer-To-Go error codes, or click Exit to close the Wizard dialog box.



Step 14: Here we have opted for the **‘Next’** button, just to show you what will happen next, and a window appears with programming & error code examples as shown below.



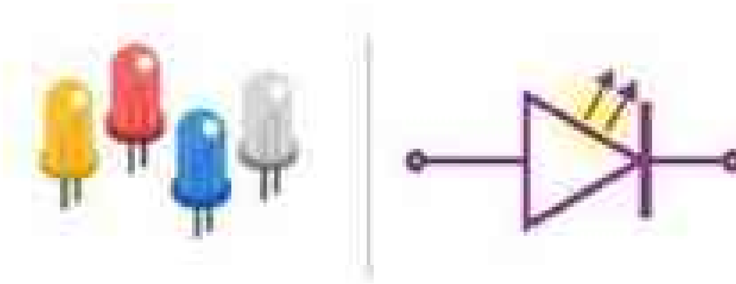
Step 15: To use **PICkit 2 Programmer-To-Go** to program a target device once it has been set up, follow the steps below.

1. **Connect a USB power source to the PICkit 2 unit,and also power up the target unit seperately.**
   Ensure the PICkit 2 "Power" LED is lit,and the "Target" LED is blinking twice in succession to indicate the unit is in Programmer-To-Go mode and ready to program.
2. **Connect the PICkit 2 unit ICSP connector to the target. Ensure that the target is powered properly.**
3. **Press the PICkit 2 pushbutton to begin programming.**

During the programming operation the PICkit 2 "Busy" LED will remain lit continuously. The "Target" LED will be lit if the target is powered from PICkit 2, and otherwise not. When the programming operation is complete, the PICkit 2 unit will provide feedback on the operation via the unit LEDs.

LED

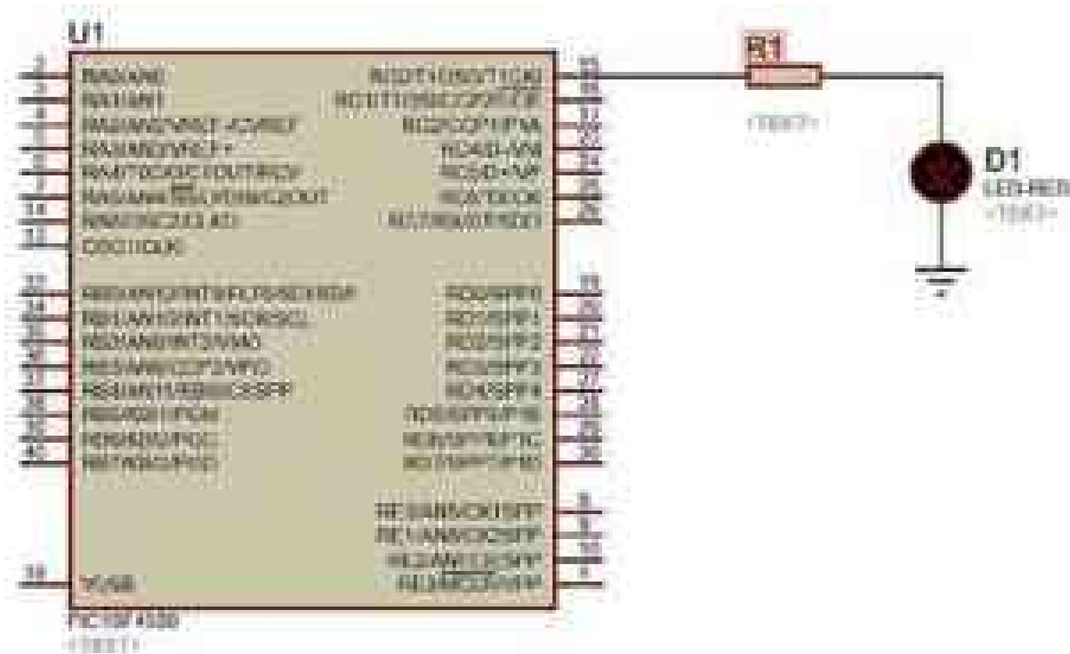
It is most widely used semiconductor which emit either visible light or invisible infrared light when forward biased. Remote controls generate invisible light. A Light emitting diodes (LED) is an optical electrical energy into light energy when voltage is applied.



These are the applications of LEDs:

- · Digital computers and calculators.
- · Traffic signals and Burglar alarms systems.
- · Camera flashes and automotive heat lamps
- · Picture phones and digital watches.

LED Schematic



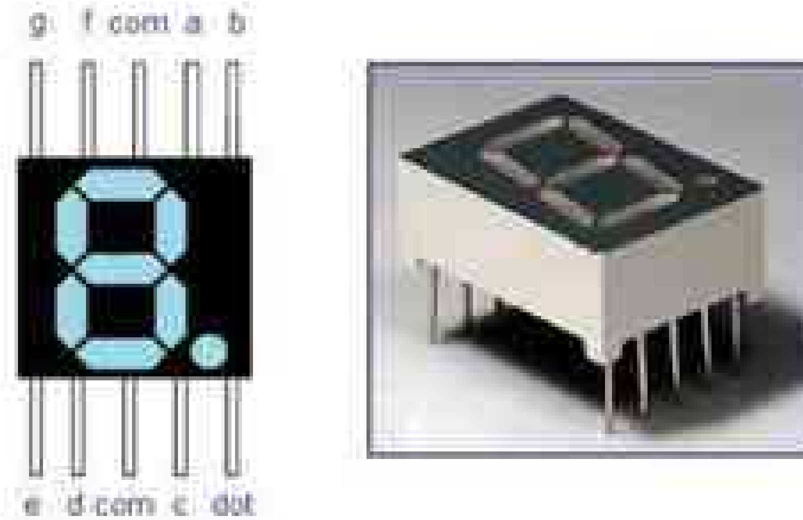
LED Code

```
#define _XTAL_FREQ 4000000

#include <xc.h>

void init_config(void);
// CONFIG
#pragma config FOSC = XT // Oscillator Selection bits (XT oscillator)
#pragma config WDTE = ON // Watchdog Timer Enable bit (WDT enabled)
#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = ON // Brown-out Reset Enable bit (BOR enabled)
#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)
#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)
#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)
#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)
```

```c

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.
#define LED_ARRAY_DDR TRISC
#define LED_ARRAY PORTC

int main()
{
 init_config();
 unsigned int delay;
 unsigned char led_mask = 0b00000001; // Start with RC0 (bit 0)

 while(1)
 {LED_ARRAY = led_mask;
 led_mask <<= 1; // Shift the mask to the left
 if (led_mask == 0b00010000) // If all LEDs have been lit, reset to RC0
 led_mask = 0b00000001;

 for(delay = 10000; delay > 0; delay--);
 }
}
void init_config(void)
{
 LED_ARRAY_DDR = 0X00;
 LED_ARRAY = 0x00;}
```

Seven Segment Display

Seven segment displays are important display units in Electronics and widely used to display numbers from 0 to 9. It can also display some character alphabets like A,B,C,H,F,E etc. It's the simplest unit to display numbers and characters. It just consists 8 LEDs, each LED used to illuminate one segment of unit and the 8th LED used to illuminate DOT in 7 segment display. We can refer each segment as a LINE, as we can see there are 7 lines in the unit, which are used to display a number/character. We can refer each line/segment "a,b,c,d,e,f,g" and for dot character we will use "h". There are 10 pins, in which 8 pins are used to refer a,b,c,d,e,f,g and h/dp, the two middle pins are common anode/cathode of all he LEDs. These common anode/cathode are internally shorted so we need to connect only one COM pin.

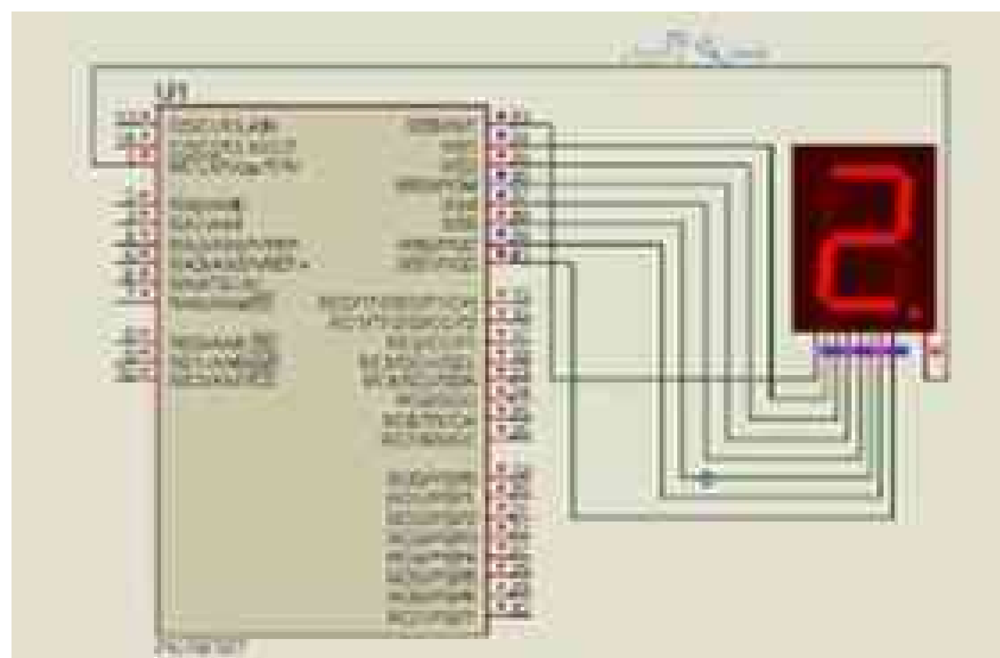


There are two types of 7 segment displays: Common Anode and Common Cathode:

Common Anode: In this all the Negative terminals (cathode) of all the 8 LEDs are connected together (see diagram below), named as COM. And all the positive terminals are left alone.

Common Cathode: In this all the positive terminals (Anodes) of all the 8 LEDs are connected together, named as COM. And all the negative thermals are left alone.

Seven Segment Display Schematic



Seven Segment Display Code

```
// CONFIG

#pragma config FOSC = XT // Oscillator Selection bits (HS oscillator)

#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)

#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)

#pragma config BOREN = OFF // Brown-out Reset Enable bit (BOR disabled)

#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)

#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)

#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)
```

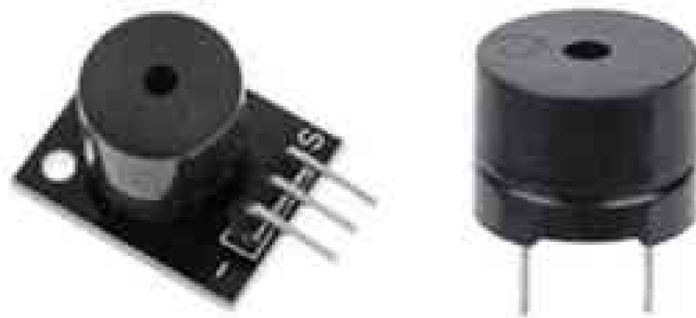
```
#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>

#define _XTAL_FREQ 4000000 // 4MHz Crystal oscillator frequency (Change this to match your oscillator)

#include <pic16f877a.h>
#define bcd1 RD4
#define bcd2 RD5
#define bcd3 RD6
#define bcd4 RD7

void main()
{
// Set PORTD as input for BCD inputs
TRISD = 0x0F;

while(1)
{
// Display 0
bcd1 = 0;
bcd2 = 0;
bcd3 = 0;
bcd4 = 0;
__delay_ms(1000);

// Display 1
bcd1 = 1;
bcd2 = 0;
bcd3 = 0;
bcd4 = 0;
__delay_ms(1000);

// Display 2
bcd1 = 0;
bcd2 = 1;
```

```
bcd3 = 0;
bcd4 = 0;
_delay_ms(1000);

// Display 3
bcd1 = 1;
bcd2 = 1;
bcd3 = 0;
bcd4 = 0;
_delay_ms(1000);

// Display 4
bcd1 = 0;
bcd2 = 0;
bcd3 = 1;
bcd4 = 0;
_delay_ms(1000);

// Display 5
bcd1 = 1;
bcd2 = 0;
bcd3 = 1;
bcd4 = 0;
_delay_ms(1000);

// Display 6
bcd1 = 0;
bcd2 = 1;
bcd3 = 1;
bcd4 = 0;
_delay_ms(1000);

// Display 7
bcd1 = 1;
bcd2 = 1;
bcd3 = 1;
bcd4 = 0;
_delay_ms(1000);

```

```
// Display 8
bcd1 = 0;
bcd2 = 0;
bcd3 = 0;
bcd4 = 1;
__delay_ms(1000);

// Display 9
bcd1 = 1;
bcd2 = 0;
bcd3 = 0;
bcd4 = 1;
__delay_ms(1000);
}
```

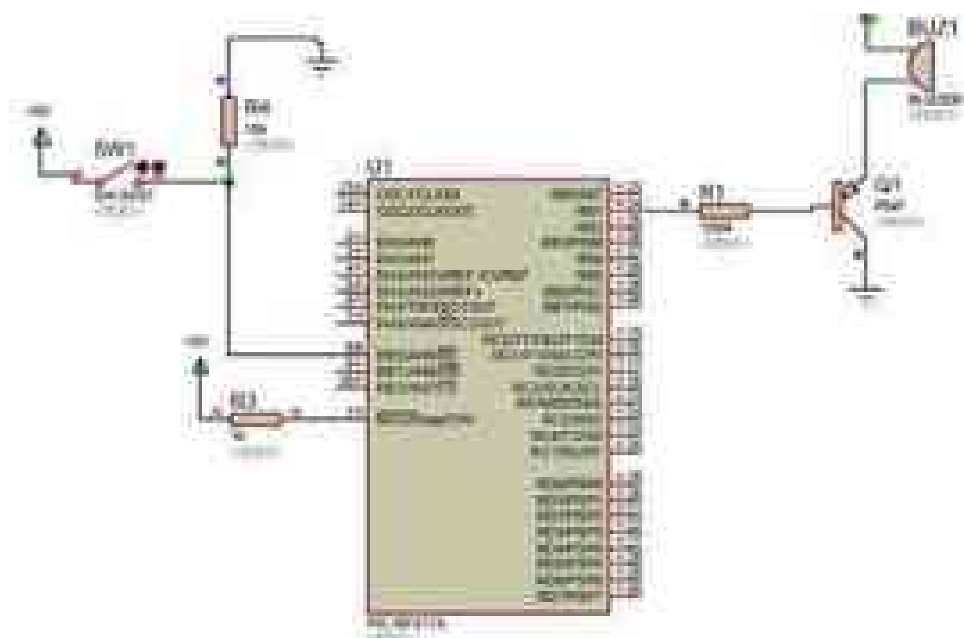
Buzzer

A buzzer is an electronic device that generates sound by converting electrical energy into sound energy. It typically consists of a piezoelectric crystal, which expands and contracts when an alternating current is applied to it, creating sound waves.



Buzzers are commonly used in a wide range of applications such as alarms, timers, and warning systems. They can also be used in electronic devices such as mobile phones, computers, and other electronic devices to generate different sounds and tones.

Buzzer Schematic



Buzzer Code

```
// CONFIG

#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)

#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)

#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)

#pragma config BOREN = OFF // Brown-out Reset Enable bit (BOR disabled)

#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)

#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)

#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)

#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)


// #pragma config statements should precede project file includes.

// Use project enums instead of #define for ON and OFF.

```

```
#include <xc.h>

#define _XTAL_FREQ 4000000 // 20MHz crystal oscillator frequency

void main() {
TRISD0 = 0; // Set RB0 as an output
RD0 = 0; // Initially, turn off the buzzer

while (1) {
RD0 = 1; // Turn on the buzzer (active high)
__delay_ms(5000); // Delay for 500 ms (adjust as needed)
RD0 = 0; // Turn off the buzzer
__delay_ms(5000); // Delay for 500 ms (adjust as needed)
}
}
```

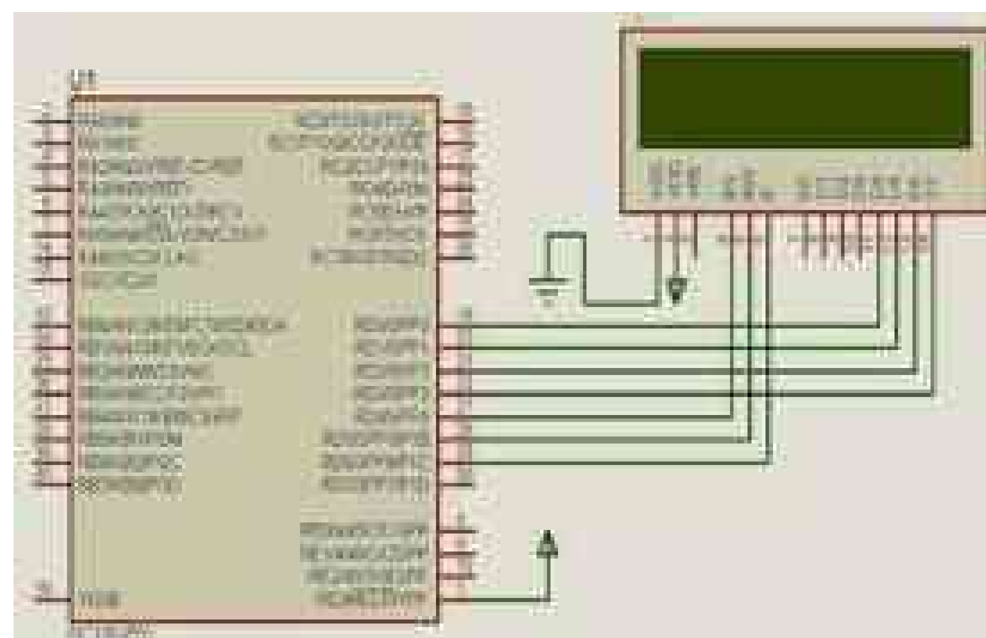
LCD

The term LCD stands for liquid crystal display. It is one kind of electronic display module used in an extensive range of applications like various circuits & devices like mobile phones, calculators, computers, TV sets, etc. These displays are mainly preferred for multi-segment light-emitting diodes and seven segments. The main benefits of using this module are inexpensive; simply programmable, animations, and there are no limitations for displaying custom characters, special and even animations, etc.

The features of this LCD mainly include the following.

- · **The operating voltage of this LCD is 4.7V-5.3V**
- · **It includes two rows where each row can produce 16-characters.**
- · **The utilization of current is 1mA with no backlight**
- · **Every character can be built with a 5×8 pixel box**
- · **The alphanumeric LCDs alphabets & numbers**
- · **Is display can work on two modes like 4-bit & 8-bit**

LCD Schematic



LCD Code

```
#include <stdio.h>

#include <stdlib.h>

#include <xc.h>

#include "config.h"


#define RS RB0
```

```
#define RW RB1
#define EN RB2
#define D4 RB4
#define D5 RB5
#define D6 RB6
#define D7 RB7

#include "lcd.h"

void main()
{
TRISB = 0x00;
Lcd_Init();
while(1)
{
Lcd_Clear();
Lcd_Set_Cursor(1,1);
Lcd_Write_String("Welcome");

__delay_ms(2000);
Lcd_Set_Cursor(1,1);
Lcd_Write_String("All");

Lcd_Clear();
Lcd_Set_Cursor(2,1);
Lcd_Write_String("..Hello World..");

for(int i=0; i<14; i++)
{
__delay_ms(350);
Lcd_Shift_Right();
}

for(int i=0; i<14; i++)
{
__delay_ms(350);
Lcd_Shift_Left();
}
}
```

Relay

Relays are the switches that aim at closing and opening the circuits electronically as well as electromechanically. It controls the opening and closing of the circuit contacts of an electronic circuit. When the relay contact is open (NO), the relay isn't energized with the open contact. However, if it is closed (NC), the relay isn't energized given the closed contact. However, when energy (electricity or charge) is supplied, the states are prone to change.

Relays are normally used in the control panels, manufacturing, and building automation to control the power along with switching the smaller current values in a control circuit. However, the supply of amplifying effect can help control the large amperes and voltages because if low voltage is applied to the relay coil, a large voltage can be switched by the contacts.

Relay Schematic

Relay Code

```
// CONFIG

#pragma config FOSC = XT // Oscillator Selection bits (HS oscillator)

#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)

#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)

#pragma config BOREN = ON // Brown-out Reset Enable bit (BOR enabled)

#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3/PGM pin has PGM function; low-voltage programming enabled)

#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)

#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)
```

```
#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)



#include <xc.h>

// Hardware related definition
#define _XTAL_FREQ 4000000 // Crystal Frequency, used in delay
#define RELAY RB0

void main(void)
{
RELAY = 0;

TRISB0 = 0x00;


while(1) {
// Toggle the relay state based on the LED state

RELAY = 1;
__delay_ms(500);
RELAY = 0;
__delay_ms(500);
}
return;
}
```

Stepper Motor

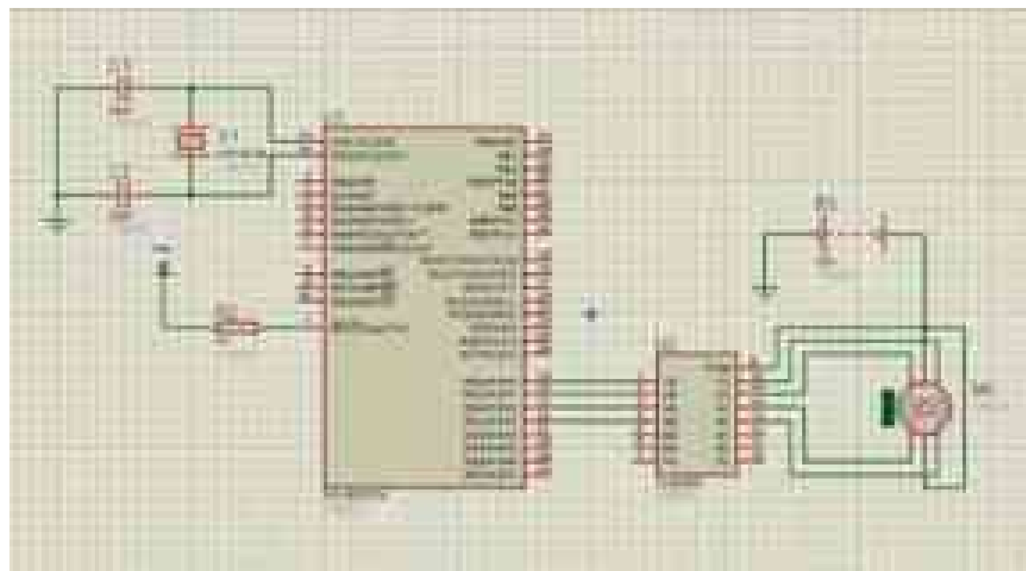
Stepper Motor is a brushless DC Motor. Control signals are applied to stepper motor to rotate it in steps.

Its speed of rotation depends upon rate at which control signals are applied. There are various stepper motors available with minimum required step angle.

Stepper motor is made up of mainly two parts, a stator and rotor. Stator is of coil winding and rotor is mostly permanent magnet or ferromagnetic material.

Step angle is the minimum angle that stepper motor will cover within one move/step. Number of steps required to complete one rotation depends upon step angle. Depending upon stepper motor configuration, step angle varies e.g. 0.72°, 3.8°, 3.75°, 7.5°, 35° etc.

Stepper Motor Schematic



Stepper Motor Code

```
#include <xc.h>

#include <stdio.h>

#include "config.h"


#define _XTAL_FREQ 4000000


#define speed 1 // Speed Range 10 to 1 10 = lowest , 1 = highest


#define steps 250 // how much step it will take


#define clockwise 0 // clockwise direction macro


#define anti_clockwise 1 // anti clockwise direction macro
```

```
//FUNCTION TO OPERATE MOTOR

void system_init (void); // This function will initialise the ports.

void full_drive (char direction); // This function will drive the motor in full drive mode

void half_drive (char direction); // This function will drive the motor in full drive mode

void wave_drive (char direction); // This function will drive the motor in full drive mode

void ms_delay(unsigned int val);


void main(void)

{

system_init();

while(1){

/* Drive the motor in full drive mode clockwise */

for(int i=0;i<steps;i++)

{

 full_drive(clockwise);

}

ms_delay(1000);

/* Drive the motor in wave drive mode anti-clockwise */

for(int i=0;i<steps;i++)

{
```

```
wave_drive(anti_clockwise);

//full_drive(anti_clockwise);

}

ms_delay(1000);

}

}

/*System Initialising function to set the pin direction Input or Output*/
void system_init (void){

TRISB = 0x00; // PORT B as output port

PORTB = 0x0F;

}

/*This will drive the motor in full drive mode depending on the direction*/
void full_drive (char direction){

if (direction == anti_clockwise){

PORTB = 0b00000011;

ms_delay(speed);

PORTB = 0b00000110;

ms_delay(speed);

PORTB = 0b00001100;

ms_delay(speed);
```

```
PORTB = 0b00001001;

ms_delay(speed);

PORTB = 0b00000011;

ms_delay(speed);

}

if (direction == clockwise){

PORTB = 0b00001001;

ms_delay(speed);

PORTB = 0b00001100;

ms_delay(speed);

PORTB = 0b00000110;

ms_delay(speed);

PORTB = 0b00000011;

ms_delay(speed);

PORTB = 0b00001001;

ms_delay(speed);

}
}

/* This method will drive the motor in half-drive mode using direction input */
void half_drive (char direction){

if (direction == anti_clockwise){
```

```
PORTB = 0b00000001;

ms_delay(speed);

PORTB = 0b00000011;

ms_delay(speed);

PORTB = 0b00000010;

ms_delay(speed);

PORTB = 0b00000110;

ms_delay(speed);

PORTB = 0b00000100;

ms_delay(speed);

PORTB = 0b00001100;

ms_delay(speed);

PORTB = 0b00001000;

ms_delay(speed);

PORTB = 0b00001001;

ms_delay(speed);

}

if (direction == clockwise){

PORTB = 0b00001001;
```

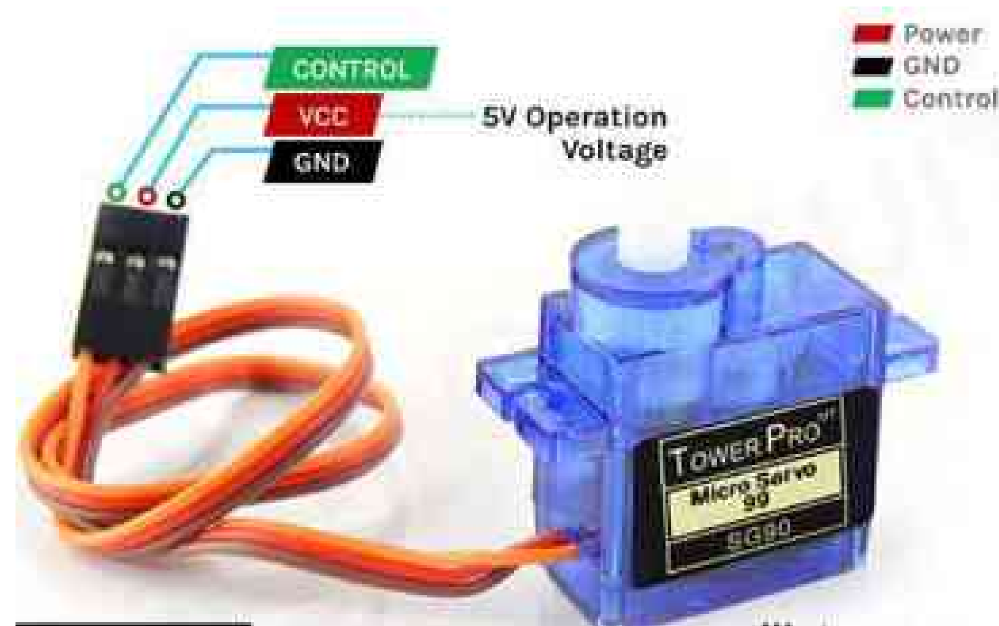
```
ms_delay(speed);

PORTB = 0b00001000;

ms_delay(speed);

PORTB = 0b00001100;

ms_delay(speed);

PORTB = 0b00000100;

ms_delay(speed);

PORTB = 0b00000110;

ms_delay(speed);

PORTB = 0b00000010;

ms_delay(speed);

PORTB = 0b00000011;

ms_delay(speed);

PORTB = 0b00000001;

ms_delay(speed);
}

}

/* This function will drive the the motor in wave drive mode with direction input*/
void wave_drive (char direction){

if (direction == anti_clockwise)
{
PORTB = 0b00000001;
```

```
ms_delay(speed);

PORTB = 0b00000010;

ms_delay(speed);

PORTB = 0b00000100;

ms_delay(speed);

PORTB = 0b00001000;

ms_delay(speed);

}
if (direction == clockwise){

PORTB = 0b00001000;

ms_delay(speed);

PORTB = 0b00000100;

ms_delay(speed);

PORTB = 0b00000010;

ms_delay(speed);

PORTB = 0b00000001;

ms_delay(speed);

}



}
```

```
/*This method will create required delay*/
void ms_delay(unsigned int val)
{

 unsigned int i,j;

 for(i=0;i<val;i++)

 for(j=0;j<1650;j++);

}
```

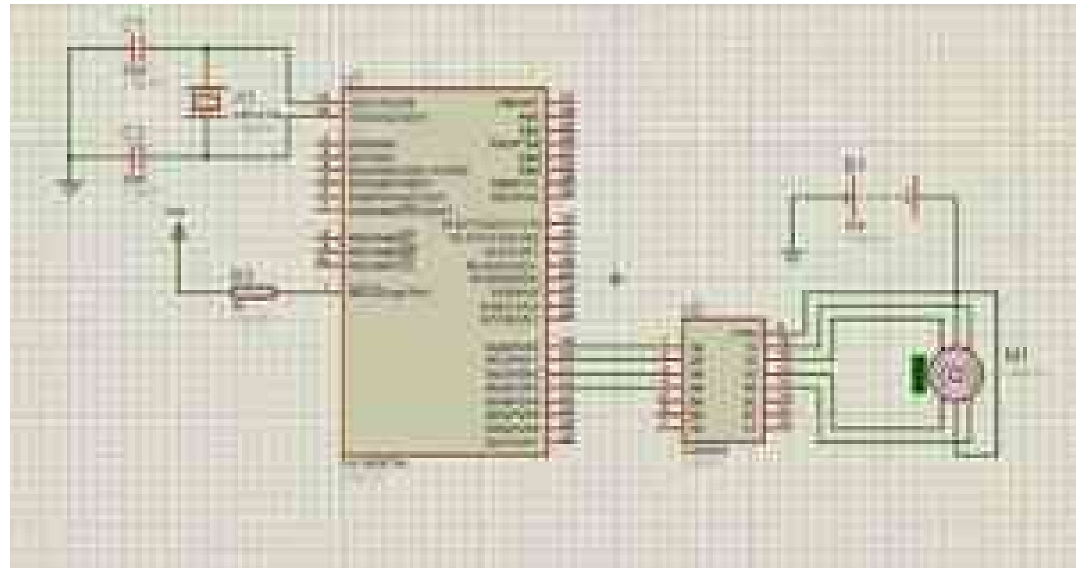
Servo Motor

Servo motor is an electrical device which can be used to rotate objects (like robotic arm) precisely.Servo motor consists of DC motor with error sensing negative feedback mechanism. This allows precise control over angular velocity and position of motor. In some cases, AC motors are used.



It is a closed loop system where it uses negative feedback to control motion and final position of the shaft.It is not used for continuous rotation like conventional AC/DC motors.It has rotation angle that varies from 0° to 380°.

Servo Motor Schematic



Servo Motor Code

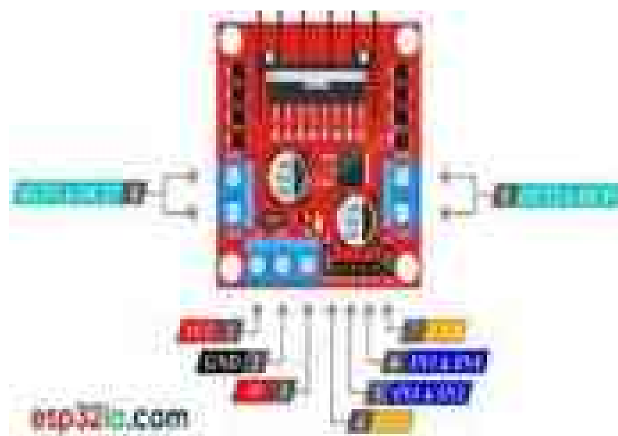
```
// CONFIG
#pragma config FOSC = XT // Oscillator Selection bits (XT oscillator)
#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = OFF // Brown-out Reset Enable bit (BOR disabled)
#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)
#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)
#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)
#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>
```

```

#define _XTAL_FREQ 4000000

void servo0(){
 unsigned int i;
 for(i=0;i<50;i++)
{
PORTAbits.RA4 = 1;
__delay_us(800);
 PORTAbits.RA4 = 0;
 __delay_us(19200);}}

void servo90(){
unsigned int i;
 for(i=0;i<50;i++)
{
PORTAbits.RA4 = 1;
__delay_us(1500);
PORTAbits.RA4 = 0;
__delay_us(18500);
}}

void servo180(){
unsigned int i;
for(i=0;i<50;i++)
{
PORTAbits.RA4 = 1;
__delay_us(2200);
PORTAbits.RA4 = 0;
__delay_us(17800);
}}

void main() {
 TRISA4=0x00;
 PORTAbits.RA4 = 0; // Ensure RA4 is initially low


 while(1){
 __delay_ms(1000);
```

```
servo0();

__delay_ms(1000);

servo90();

__delay_ms(1000);

servo180();

__delay_ms(1000);

}}
```

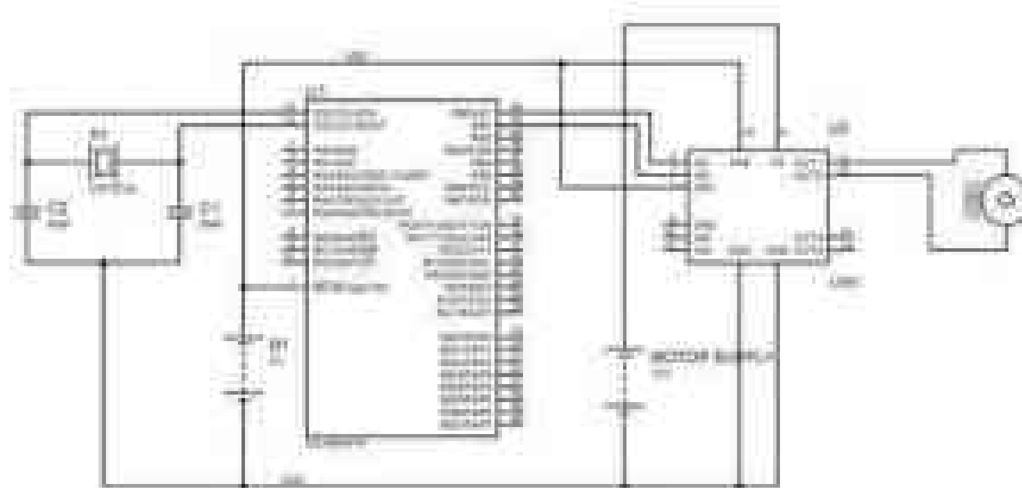
DC Motor

DC motor uses Direct Current (electrical energy) to produce mechanical movement i.e. rotational movement. When it converts electrical energy into mechanical energy then it is called as DC motor and when it converts mechanical energy into electrical energy then it is called as DC generator.



The working principle of DC motor is based on the fact that when a current carrying conductor is placed in a magnetic field, it experiences a mechanical force and starts rotating. Its direction of rotation depends upon Fleming's Left Hand Rule.DC motors are used in many applications like robot for movement control, toys, quadcopters, CD/DVD disk drive in PCs/Laptops etc.

DC Motor Schematic



DC Motor Code

```
// CONFIG

#pragma config FOSC = XT // Oscillator Selection bits (XT oscillator)

#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)

#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)

#pragma config BOREN = ON // Brown-out Reset Enable bit (BOR enabled)

#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)

#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)

#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)

#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)
```

```c
// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

#include <xc.h>
#include <pic.h>

#define MOTOR1 RB3
#define MOTOR2 RB4

void main()
{
 unsigned int i;
 TRISB = 0x00;

 while(1)
 {
 MOTOR1 = 1;
 MOTOR2 = 0;

 for (i=0; i<60; i++);

 MOTOR1 = 0;
 MOTOR2 = 1;
 }
}
```

Multiplex Display

A multiplexed display allows multiple 7-segment displays to be driven using fewer I/O pins. The microcontroller switches between the displays very quickly (persistence of vision), making it appear as though all digits are lit simultaneously.

Multiplex Display Code

```
#include <xc.h>
#define _XTAL_FREQ 4000000

// Control pins for 2-digit multiplex display
#define DIGIT1 RA0
#define DIGIT2 RA1

// Segment pins connected to PORTB
#define SEG PORTB

void display_digit(unsigned char digit, unsigned char value) {
const unsigned char seg_code[10] = {
0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F
}; // 0-9

SEG = seg_code[value];
if(digit==1){ DIGIT1=1; DIGIT2=0; }
else { DIGIT1=0; DIGIT2=1; }

__delay_ms(5);
}

void main() {
TRISB=0x00; TRISA=0x00;
while(1){
display_digit(1,2);
display_digit(2,5); // Displays "25"
}

}
```

Multiplex Keyboard

A multiplexed keyboard (matrix keypad) reduces the number of microcontroller pins required. It works by arranging keys in a row-column structure, where scanning is performed to detect which key is pressed.

Multiplex Keyboard Code

```
#include <xc.h>
#define _XTAL_FREQ 4000000

#define ROW1 RA0
#define ROW2 RA1
#define ROW3 RA2
#define ROW4 RA3

#define COL1 RB0
#define COL2 RB1
#define COL3 RB2

unsigned char read_keypad() {
// Example 4x3 matrix keypad scanning
for(int row=0; row<4; row++){
PORTA = ~(1<<row);
__delay_ms(1);
if(COL1==0) return (row*3)+1;
if(COL2==0) return (row*3)+2;
if(COL3==0) return (row*3)+3;
}
return 0;
}

void main(){
TRISA=0xF0; TRISB=0xFF;
while(1){
unsigned char key = read_keypad();
if(key!=0){
// Do something with key
}
}
}
```

RTC (Real Time Clock)

An RTC (Real Time Clock) such as DS1307 or DS3231 is used to keep track of time. It communicates with PIC using the I2C protocol. This allows applications such as digital clocks and data loggers.

RTC Code (I2C with DS1307)

```
#include <xc.h>
#include "i2c.h"

void main(){
I2C_Master_Init(100000); // Initialize I2C at 100kHz
while(1){
I2C_Master_Start();
I2C_Master_Write(0xD0); // RTC address + Write
I2C_Master_Write(0x00); // Point to seconds register
I2C_Master_Stop();

I2C_Master_Start();
I2C_Master_Write(0xD1); // RTC address + Read
unsigned char sec = I2C_Master_Read(0);
I2C_Master_Stop();
}
}
```

RS232 Serial Communication

RS232 Serial Communication

RS232 is a standard protocol for serial communication. PIC microcontrollers include USART which can be configured to send/receive data over RS232 with the help of a MAX232 driver IC.

RS232 Code

```
#include <xc.h>
#define _XTAL_FREQ 4000000

void UART_Init(long baud_rate){
SPBRG = ((_XTAL_FREQ/16)/baud_rate) - 1;
TXSTAbits.TXEN=1; RCSTAbits.CREN=1; RCSTAbits.SPEN=1;
}

void UART_Write(char data){
while(!TXSTAbits.TRMT);
TXREG = data;
}

char UART_Read(){
while(!PIR1bits.RCIF);
return RCREG;
}

void main(){
UART_Init(9600);
while(1){
UART_Write('H');
__delay_ms(1000);
}
}
```