PIC18F/16F Development Board

An alternative to Arduino Uno, for Studying Embedded Systems hands-on

Reference Manual

AMOTECH LABS

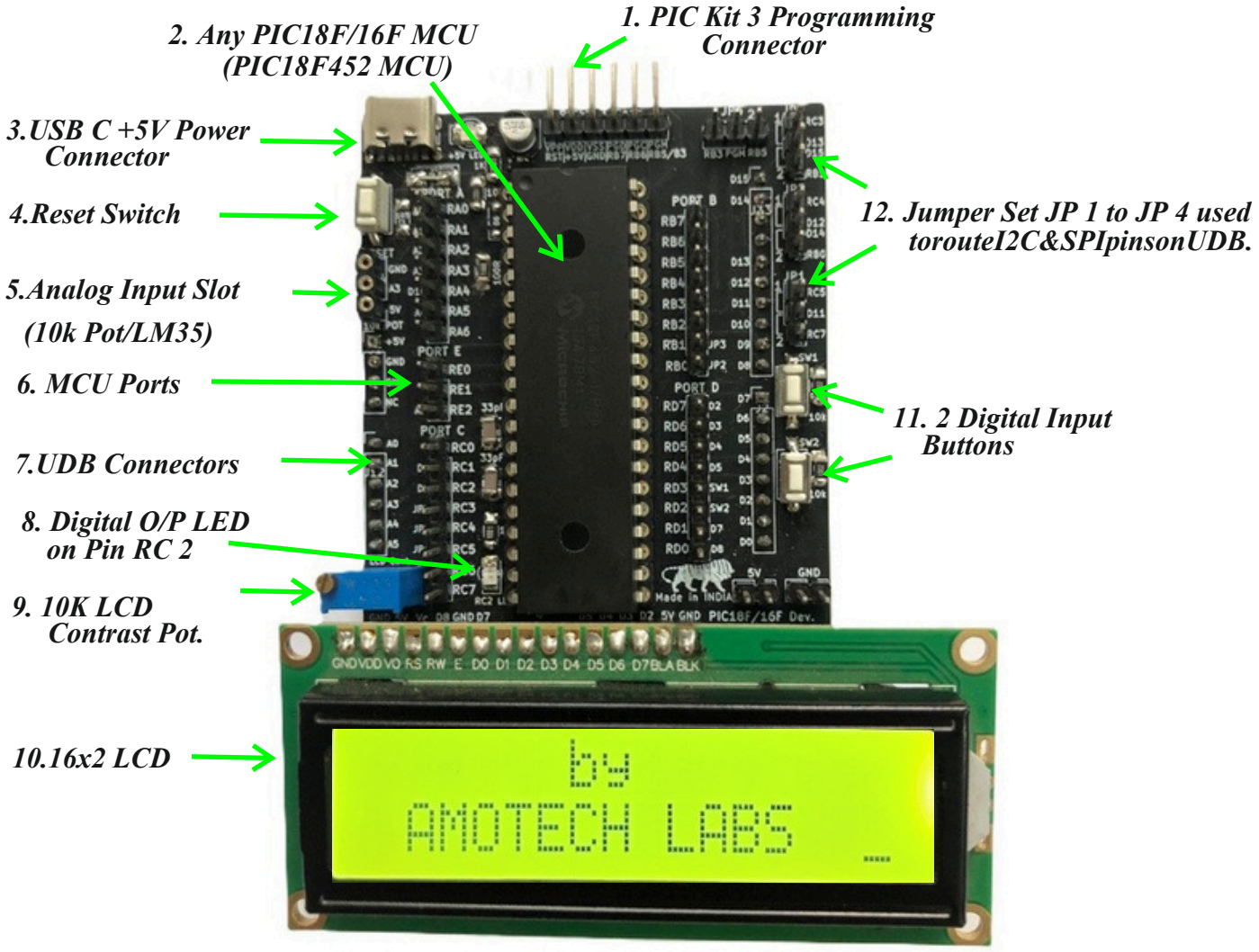
Embedded systems | Industrial Automation | Robotics

Scan to download Soft Copy of Manual & Online Purchase

Scan for Video Tutorials

Table of Content

Overview of 'PIC18F/16F Mini Development Board'



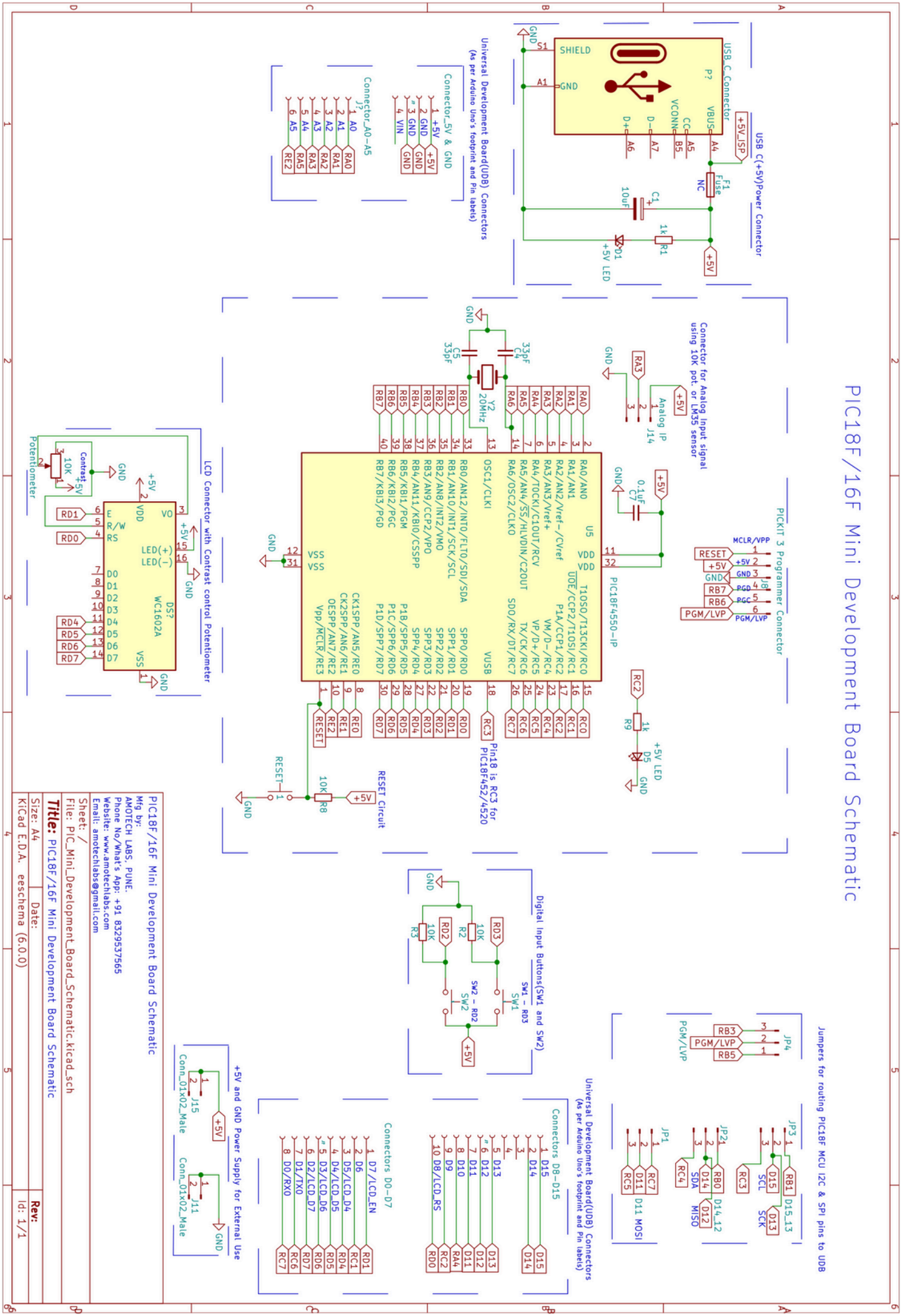
PIC18F/16F mini Development Board:

The Mini Development Kit is designed for learning and experimenting with PIC16F and PIC18F microcontrollers. It provides on-board programming support, power supply options, input/output interfaces, and expansion connectors, enabling users to easily develop, program, and test embedded applications. Most labels on the kit are self-explanatory; descriptions of selected labels are given below. For detailed understanding, refer to the brief introduction to PIC18F452 and the kit schematic provided in the following pages..

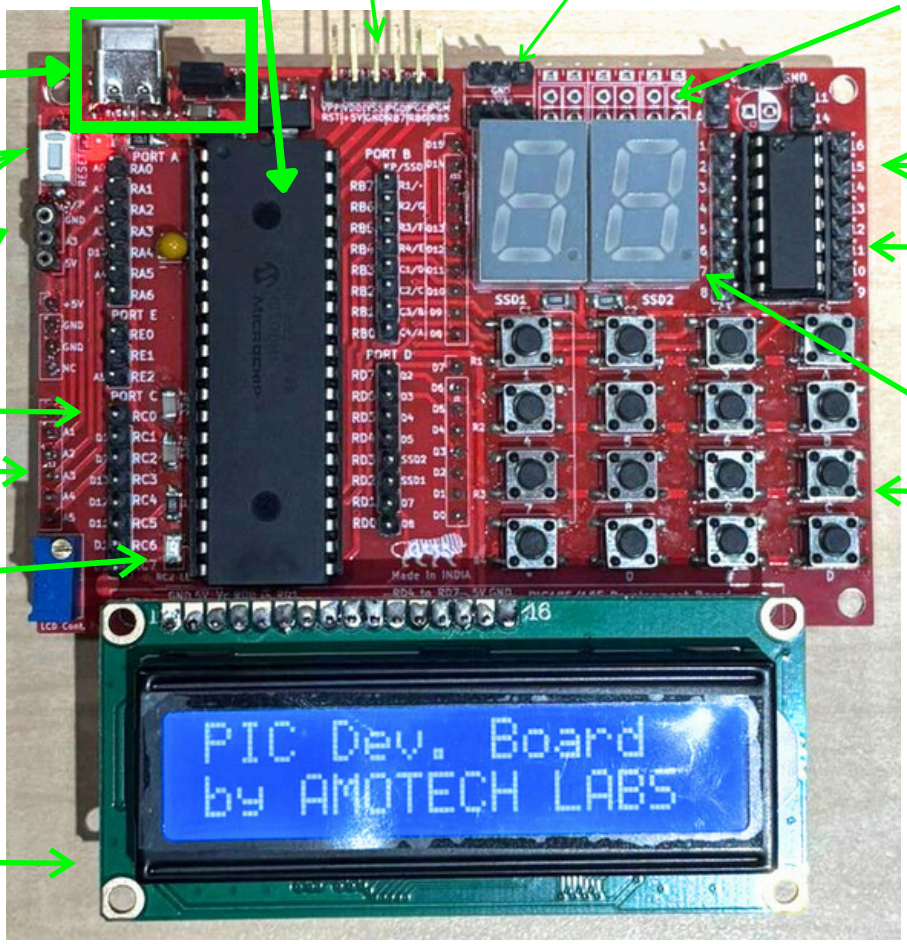
On-Board Features

1. ***PIC Kit 3 Programming Connector:*** This connector is used to interface the kit with a PICkit3/3.5 programmer for programming the MCU. The RST (Reset) pin of the kit is connected to the programmer's reset pin and is indicated by a notch symbol.
2. ***PIC16F/18F MCU:*** The kit supports multiple 40-pin DIP PIC microcontrollers, such as PIC16F887/887A and PIC18F4550/4580/4520, as well as other pin-compatible PIC16F and PIC18F devices.
3. ***USB-C +5 V Power Connector:*** The kit can be powered using a USB-C cable connected to any +5 V USB power adapter or USB port.
4. ***Reset Switch:*** This switch resets the MCU and restarts program execution. The reset button should be pressed after downloading a new program to the microcontroller.

5. ***Analog Input Slot:*** This slot can be used to connect devices such as an LM35 temperature sensor or a 10 kΩ potentiometer to provide a variable analog voltage input to analog channel A3.
6. ***MCU Ports:*** The port pins of Ports A, B, C, D, and E of the PIC MCU are accessible through these connectors. Each pin is provided with right-side pin access and left-side labeling corresponding to UDB connectors.
7. ***UDB Connector:*** These connectors are used to attach the kit to the Universal Development Board (UDB) of Amotech Labs. Details of the UDB are available on the official website.
8. ***Digital Output LED on Pin RC2:*** Pin RC2, which functions as a PWM-capable pin on most PIC16F/18F MCUs, is connected to an on-board LED to allow testing and observation of digital or PWM output signals.
11. T***wo Digital Input Buttons:*** Two digital input push buttons are provided and connected to RD3 and RD2 pins of the MCU for user input.
12. ***Jumper Sets JP1 to JP4:*** These jumpers are used to route SPI and I²C signals of different PIC16F/18F MCUs to the corresponding SPI and I²C pins of the UDB, allocated from D10 to D15.


PIC18F/16F Mini Development Board Schematic
USB C(+5V)Power Connector
USB_C_Connector
SHIELD
GND
+5V_ISP
Universal Development Board(UDB) Connectors
(As per Arduino Uno's footprint and Pin labels)
Connector_A0-A5
Connector_5V & GND
PIC18F4550-IP
PICKIT 3 Programmer Connector
MCLR/VPP
RESET
PGM/LVP
Connector for Analog input signal
using 10K pot. or LM35 sensor
Analog IP
LCD Connector with Contrast control Potentiometer
Potentiometer
WC1602A
RESET Circuit
Pin18 is RC3 for
PIC18F452/4520
Digital Input Buttons(SW1 and SW2)
SW1 - RD3
SW2 - RD2
Jumpers for routing PIC18F MCU I2C & SPI pins to UDB
PGM/LVP
Connectors D8-D15
Connectors D0-D7
+5V and GND Power Supply for External Use
Conn_01x02_Male
PIC18F/16F Mini Development Board Schematic
Mfg by:
AMOTECH LABS, PUNE.
Phone No/What's App: +91 8329537565
Website: www.amotechlabs.com
Email: amotechlabs@gmail.com
Title: PIC18F/16F Mini Development Board Schematic
Size: A4
Date:
KiCad E.D.A. eeschema (6.0.0)
Rev:
Id: 1/1
Sheet: /
File: PIC_Mini_Development_Board_Schematic.kicad_sch

Overview of 'PIC18F/16F Development Board' Kit:

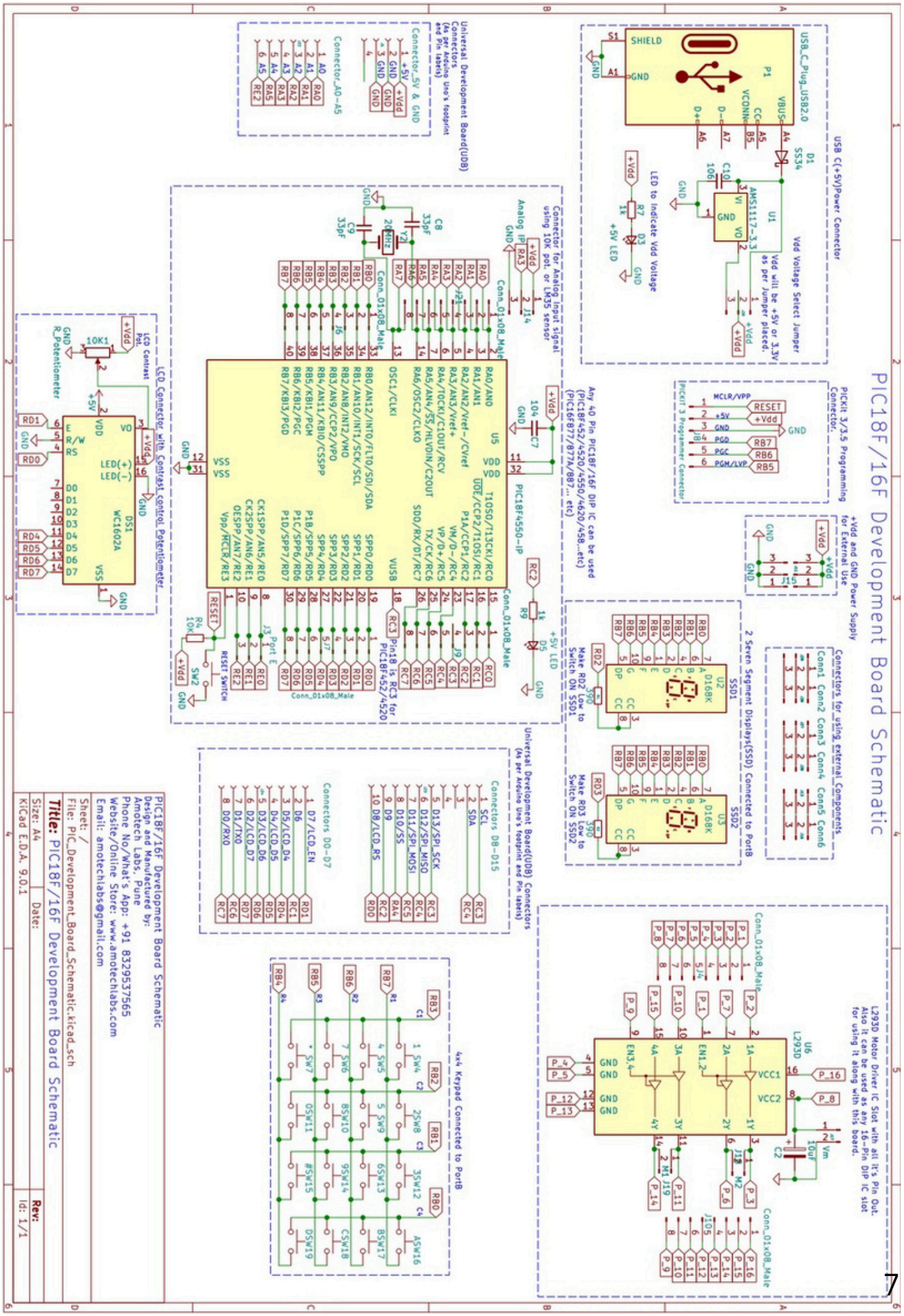


PIC18F/16F Development Board (Enhanced):

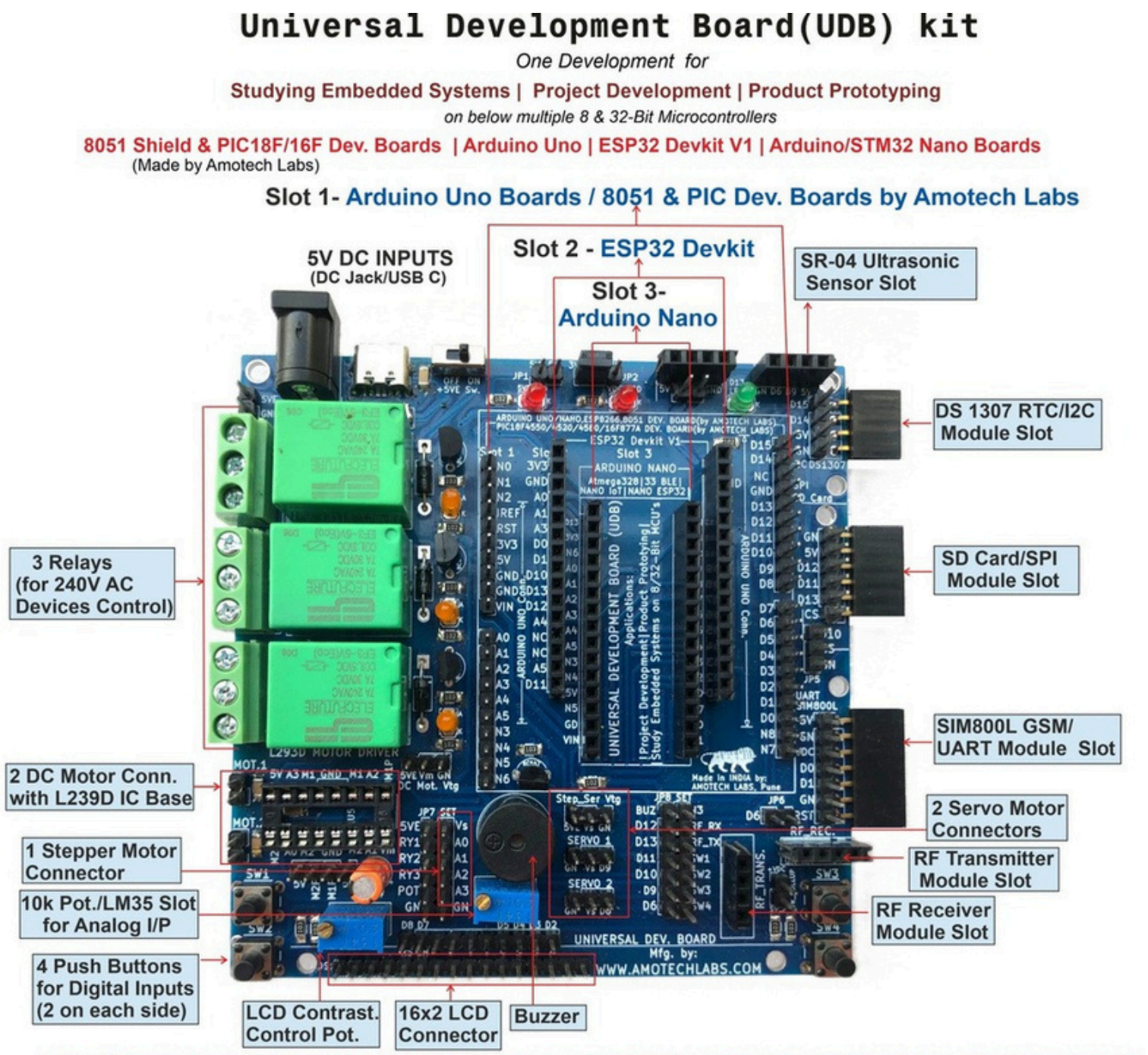
The kit used in this laboratory is similar to the PIC18F/16F Mini Development Board and includes additional on-board features such as a 4×4 keypad, two seven-segment displays, a 16-pin DIP IC socket, and breadboard-style expansion connectors for interfacing external components.

On-Board Features

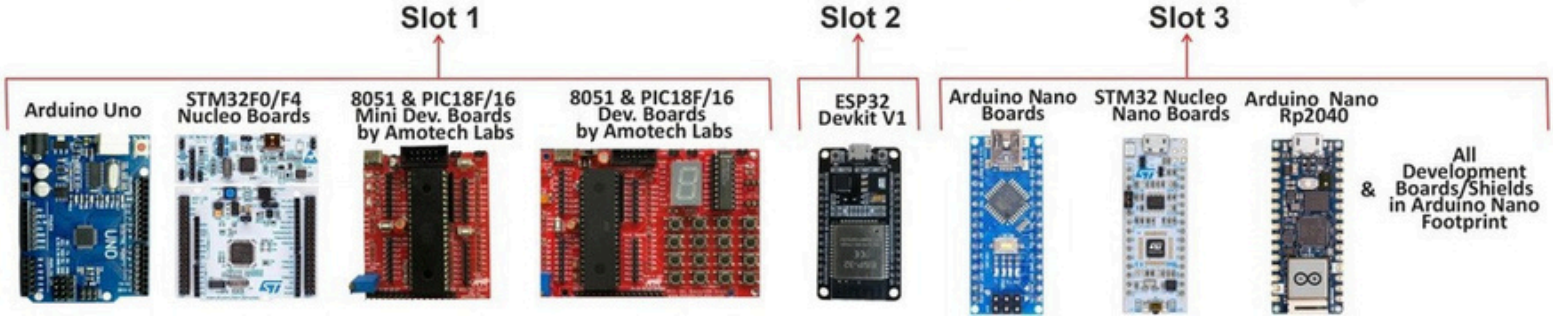
3. ***USB-C +5 V Power Connector with Voltage Selector Jumper:*** The board can be powered using a USB-C +5 V supply from any standard USB adapter. A voltage selector jumper allows the MCU operational voltage Vdd to be selected as 5 V or 3.3 V as per the jumper placement.
11. ***4×4 Keypad:*** An on-board 4×4 matrix keypad is provided for user input. The keypad is directly interfaced to Port B of the PIC microcontroller as per the schematic.
12. ***Seven Segment Displays:*** Two common cathode seven-segment displays (SSD1 and SSD2) are connected to Port B for segment data. The common cathodes are controlled using RD2 and RD3, enabling multiplexed two-digit display.
13. ***16-Pin DIP IC Socket:*** A 16-pin DIP IC socket is available to interface 8-pin or 16-pin ICs such as L293D, ULN2003/2004, or op-amps. All IC pins are accessible for connection to the PIC MCU.


PIC18F/16F Development Board Schematic
USB C(+5V)Power Connector
Vdd Voltage Select Jumper
LED to indicate Vdd Voltage
+Vdd and GND Power Supply for External Use
PICKit 3/3.5 Programming Connector.
Any 40 Pin PIC18F/16F DIP IC can be used (PIC18F452/4520/4550/4620/458...etc) (PIC16F877/8774/887... etc)
PIC18F4550-IP
LCD Connector with Contrast control Potentiometer.
Universal Development Board(UDB) Connectors (As per Arduino Uno's footprint and Pin labels)
Connectors for using external Components
2 Seven Segment Displays(SSD) Connected to PortB
Universal Development Board(UDB) Connectors (As per Arduino Uno's footprint and Pin labels)
4x4 Keypad Connected to PortB
L293D Motor Driver IC Slot with all it's Pin Out. Also it can be used as any 16-Pin DIP IC slot for using it along with this board.
PIC18F/16F Development Board Schematic
Design and Manufactured by:
Amotech Labs, Pune
Phone No/What's App: +91 8329537565
Website/Online Store: www.amotechlabs.com
Email: amotechlabs@gmail.com
Title: PIC18F/16F Development Board Schematic
Size: A4
Date:
KiCad E.D.A. 9.0.1
Rev:
Id: 1/1

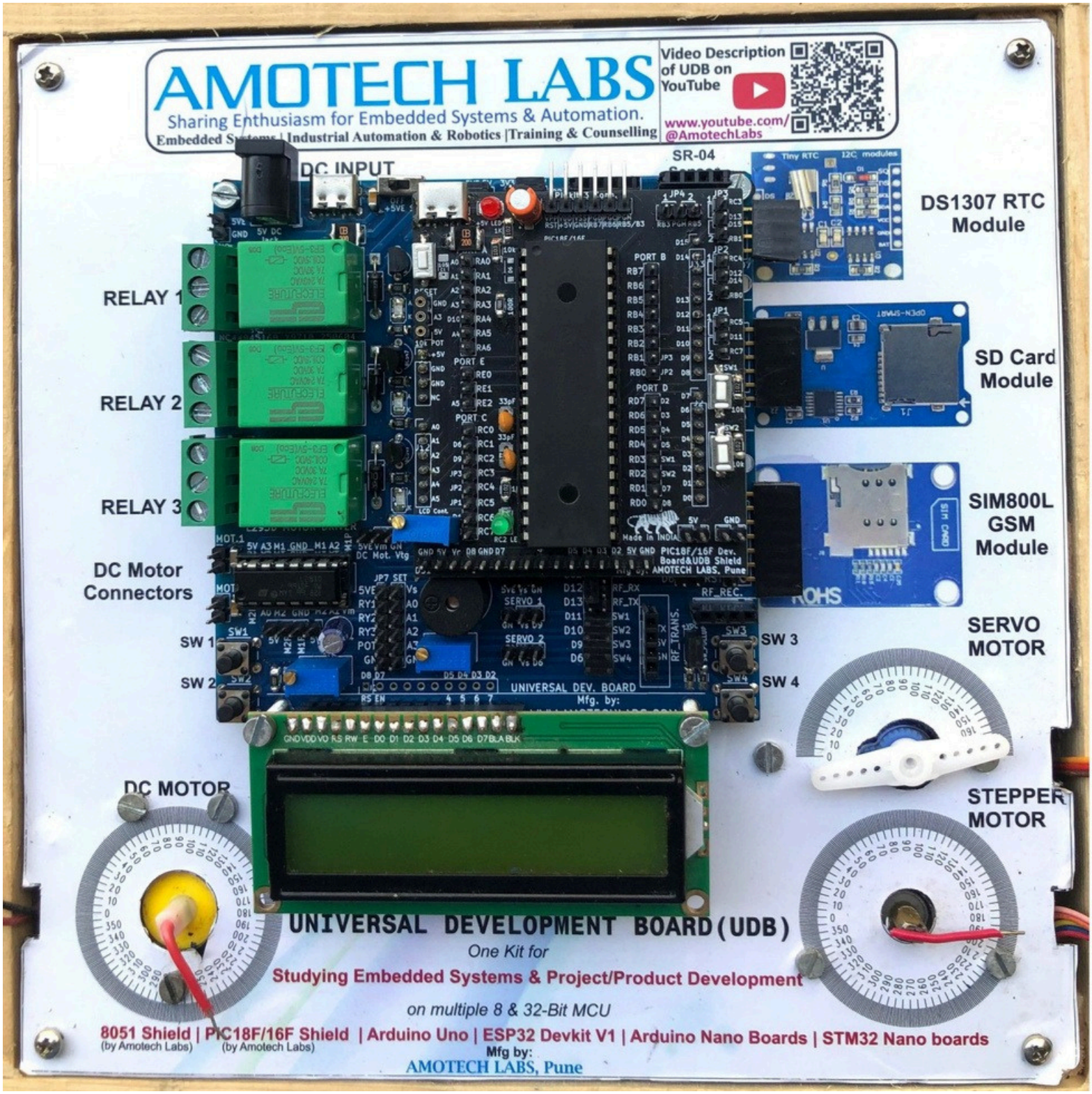
'PIC18F/16F Mini Development Board' can be mounted on below UDB Kit:



Below Micro-controller Shields/Boards can be inserted into UDB and interfaced with all above Peripherals on it

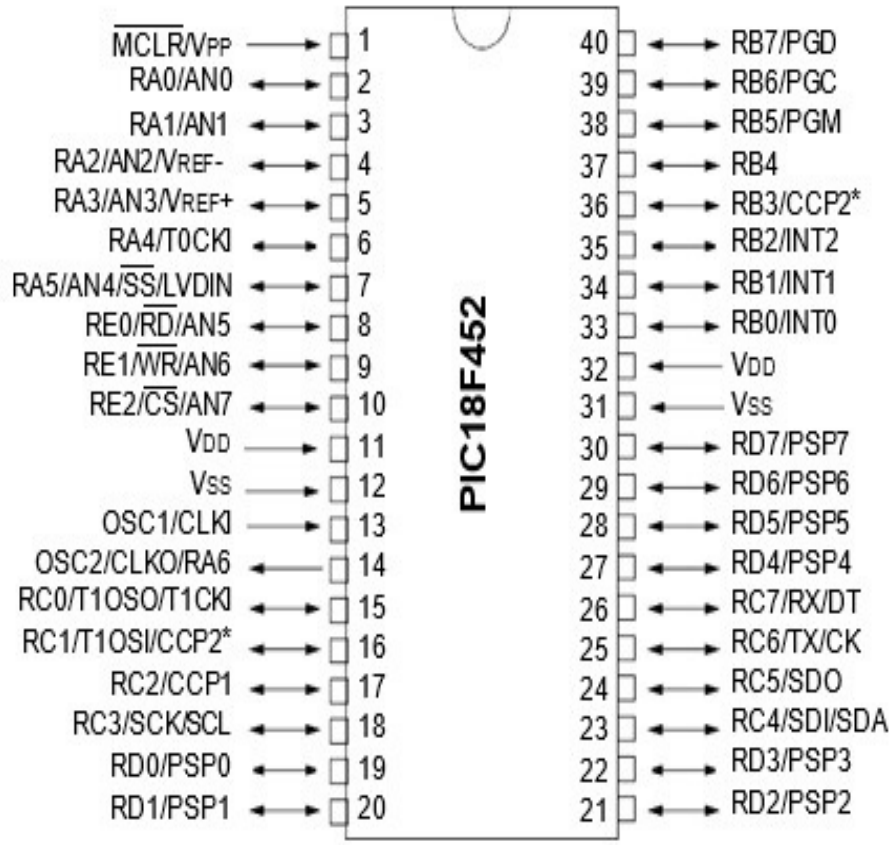


'PIC18F/16F Mini Development Board' mounted on UDB Kit's Slot 1 can be seen in the below UDB Kit which has DC, Servo and Stepper Motors mounted on-board.



Note:This Kit was assembled for College Lab use in Wooden Box Enclosure. That's why it is fitted on a Acrylic Sheet with Motors mounted on it for studying their interfacing using PIC18F MCU & UDB. For Students, only UDB is also available to buy with 'UDB 2 Wheel Robot Acrylic Sheet'. Students can configurethe UDB by adding the peripherals as per their application needs.

Introduction to PIC18F452 Microcontroller



Features of PIC18f452:

- √ Operating Frequency : DC - 40 Mhz
- √ Program Memory (Bytes): 32K
- √ Program Memory (Instructions) : 16384
- √ Data Memory (Bytes): 1536
- √ Data EEPROM Memory (Bytes) : 256
- ✓ Interrupt Sources 18 I/O Ports : Ports A, B, C, D, E
- √ Timers : 4
- √ Capture/Compare/PWM Modules : 2
- √ Enhanced Capture/ Compare/PWM Modules : 2
- √ Serial Communications : MSSP, Addressable USART
- √ Universal Serial Bus (USB) Module : No
- √ Streaming Parallel Port (SPP) : Yes
- √ 10-Bit Analog-to-Digital Module : 8 Input Channels
- √ Comparators : 2
- √ Programmable Low-Voltage Detect : Yes
- √ Instruction Set : 75 Instructions; 75 Instructions
- √ Packages : 40-pin DIP, 44-pin PLCC , 44-pin TQFP

MPLAB X IDE For Downloading Program PIC18F/16F MCU's

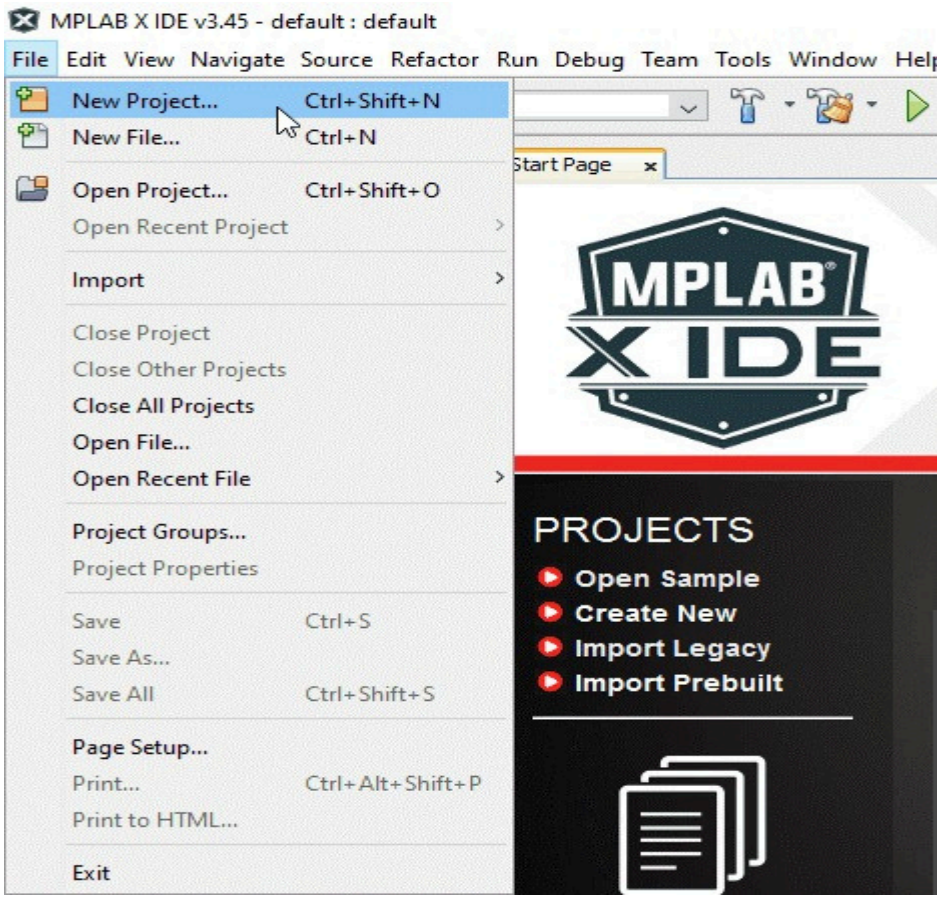
Downloading and installing MPLAB X IDE, XC8 Compiler & MPLAB IPE

Refer: https://youtube.com/@amotechlabs?si=KrWOtqwmzJxJFm6P

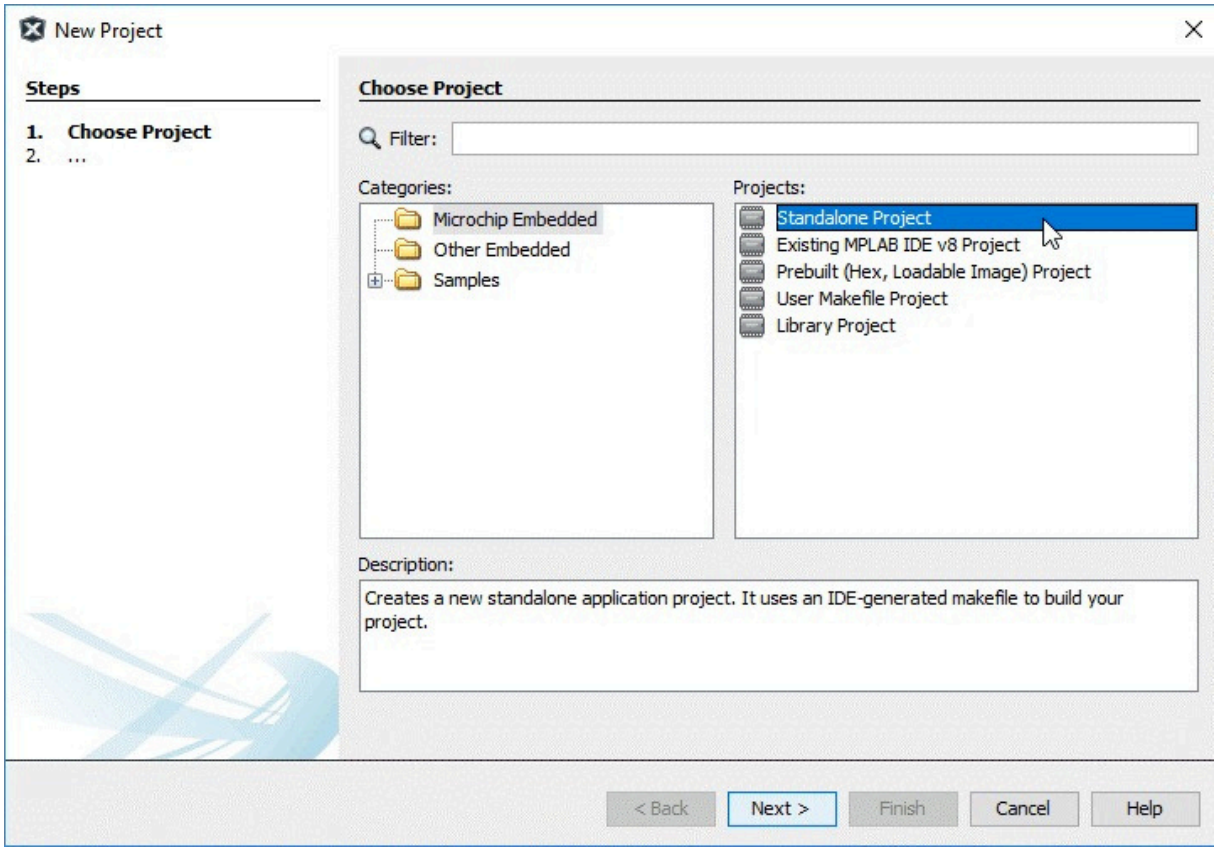
Below are the steps to program any PIC18F/16F MCU in MPLAB X IDE.

1. Double click on MPLAB X IDE icon on the desktop & Open MPLAB X IDE.

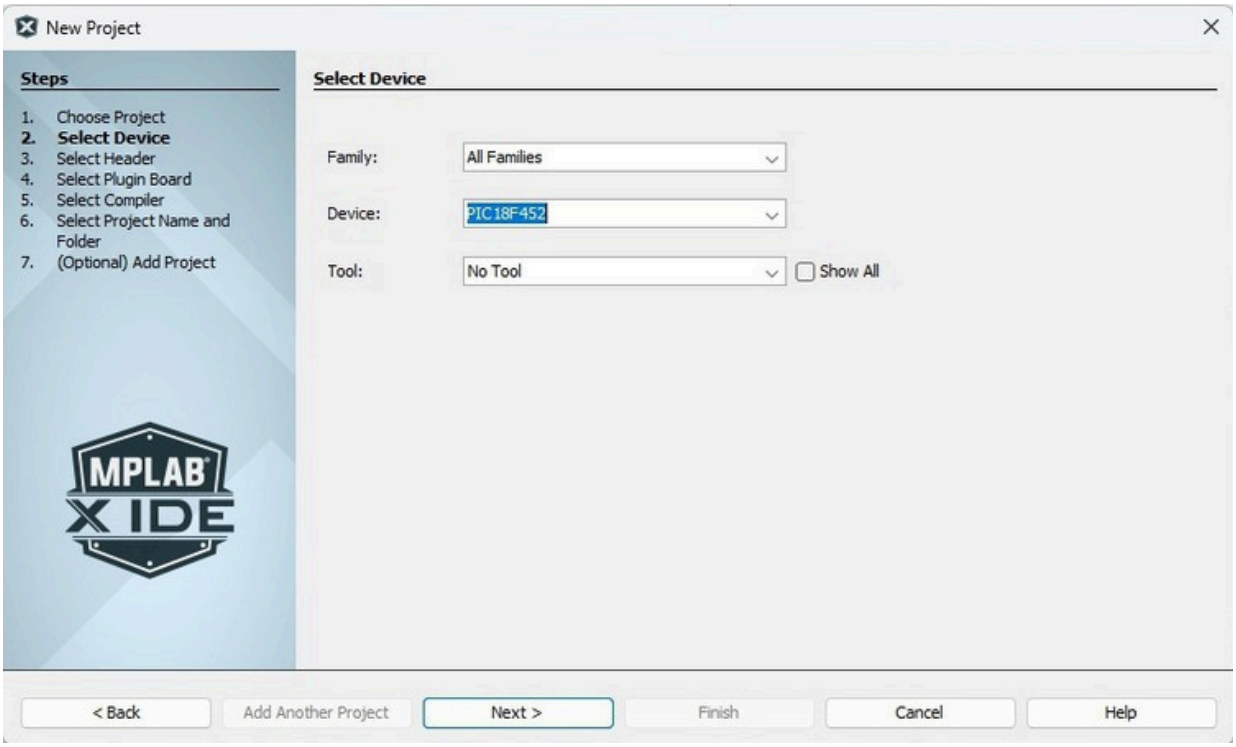
2. To create new project. Select File -> New Project.



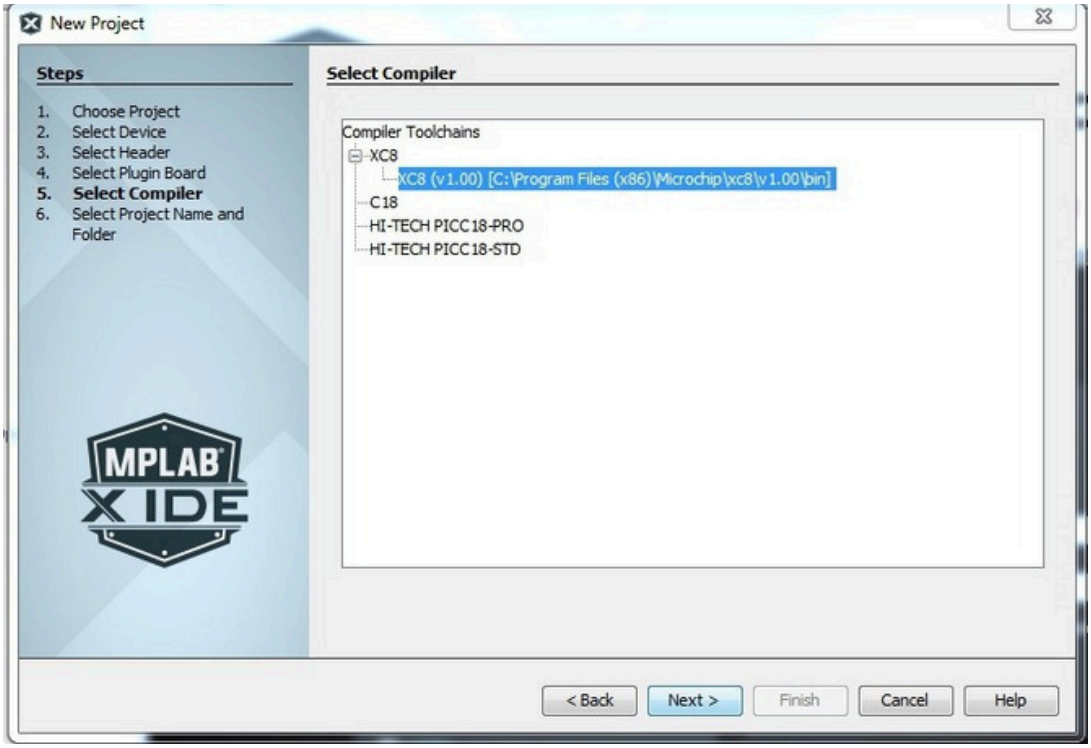
3. After that below window will appear. Select Microchip Embedded ->Standalone Project & Click next.



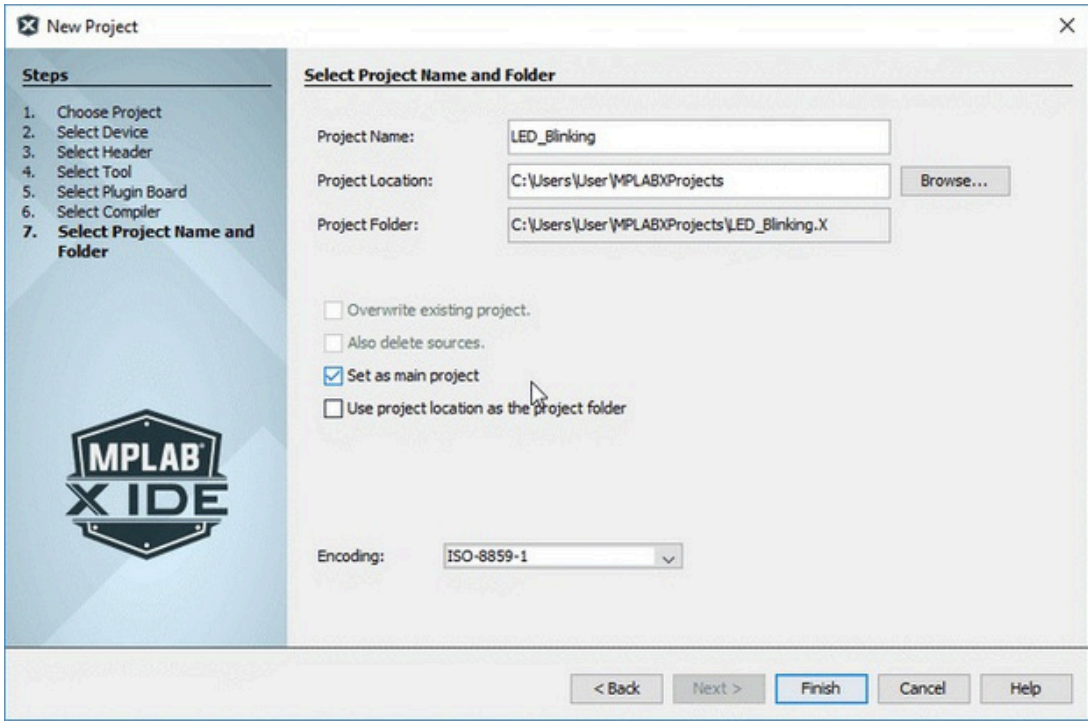
4. Select Family >Advanced 8-bit MCUs(PIC18) then select Device>PIC18F452 or any other MCU. In Tool you can select No Tool.



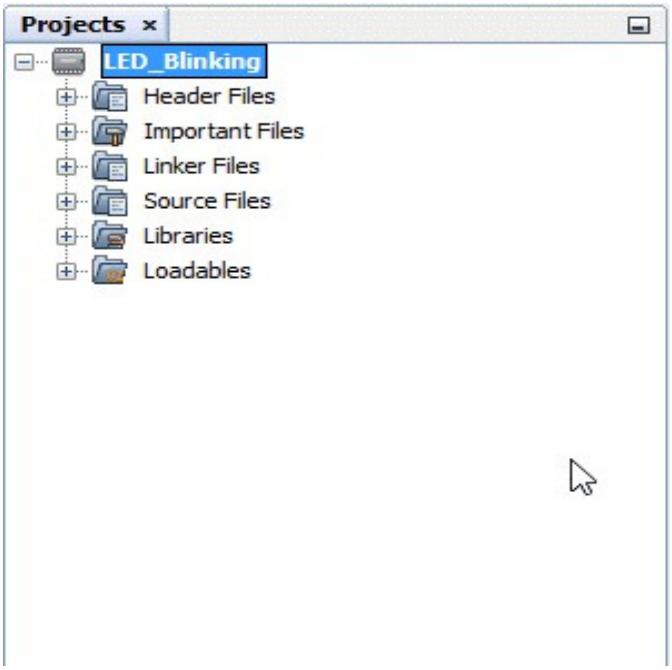
5. Now select the previously installed latest version of XC8 compiler and then click Next.



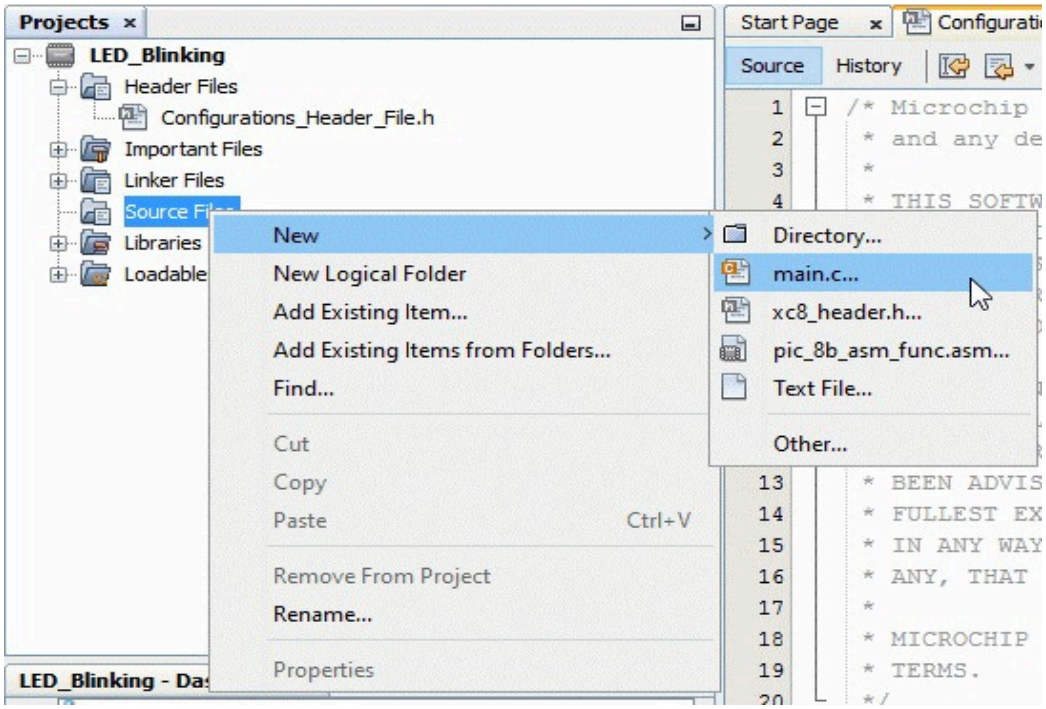
6. E nter your Project name > Click on Browse and Choose Project Folder location > Click on Finish.



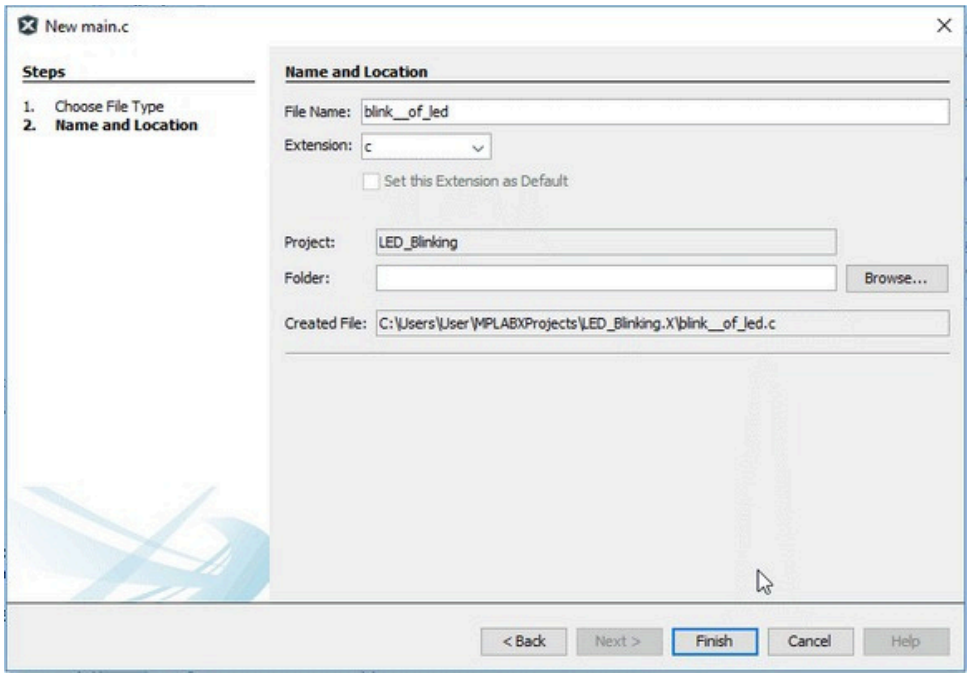
7. Now our project is displayed in Project window.



8. Create the Main file for developing code. Right Click on Source file in the project -> Select new -> Select main.c File.

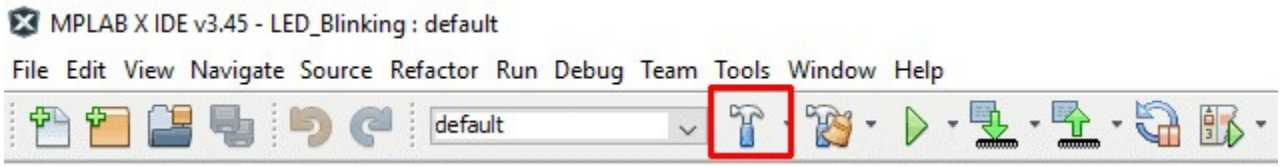


9. Give name to C file and click on finish.

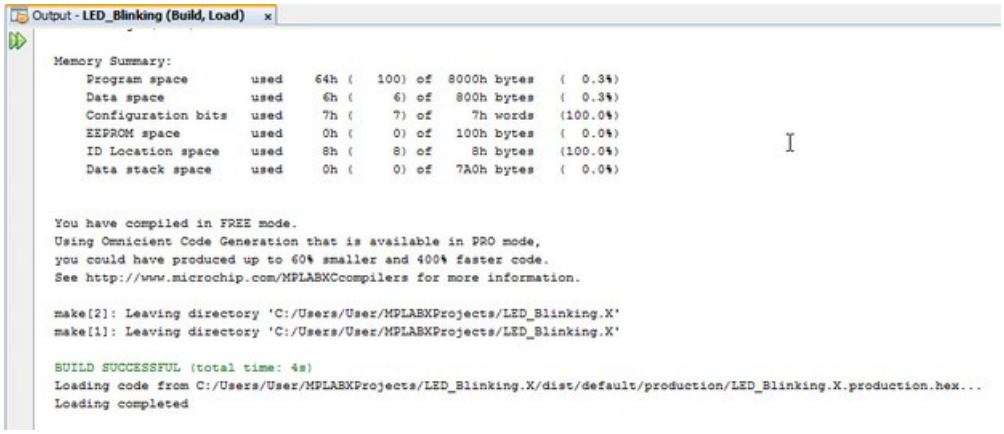


10. Write your program.

1 1. Afer you complete your program then Build it by clicking on a Hammer as seen below.



12. Afer building a Project, output Window appears which gives errors and warnings if any. otherwise Build Successful message appears as seen below.



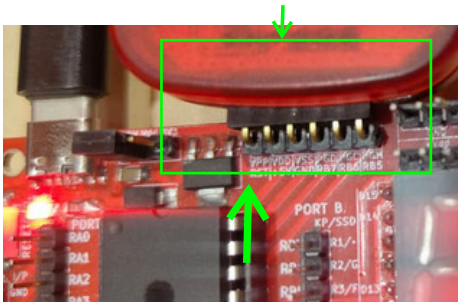
12. After BUILD SUCCESSFUL, Hex file is created in the project folder. In project folder Hex file is created in dist/default/production. You can also see a path to the folder under BUILD SUCCESSFUL.

MPLAB IPE for Downloading Program in PIC18F/16F MCU's

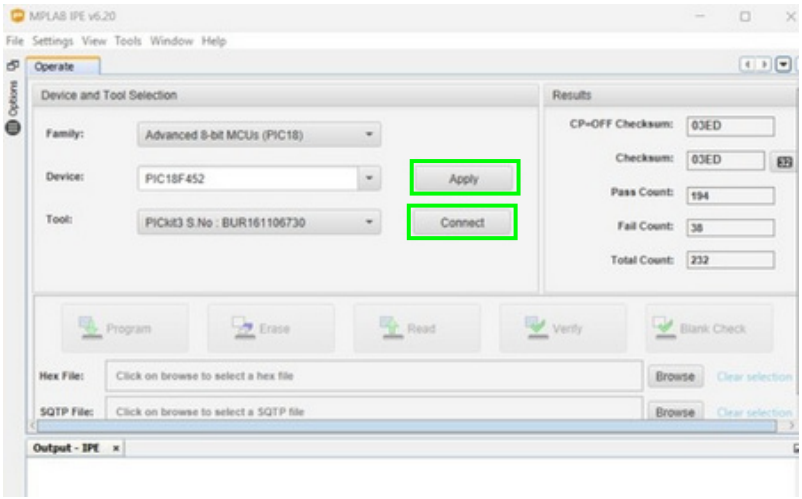
Below are the steps to program anyPIC18F/16F MCU in MPLABX IDE.

1.Open MPLAB IPE software. Select the Family and Device name of the MCU you want to program. Connect the USB of PICKit 3/3.5 Programmer to your PC, after connecting it the programmer name will appear in the tool. Now connect 12V power supply adapter to PIC18F/16F Kit and PICKit 3's 6 Pin connector to your PIC18F/16 kit directly as seen in the Kit Overview Pic on Page 3. Also, you can connect it using 6-pin F-F connector wires. PICKit 3's first pin indicated with arrow should be connected to RST pin of your PIC18F/16F kits 6-pin connector. Now, click on 'Apply' and then on 'Connect'.

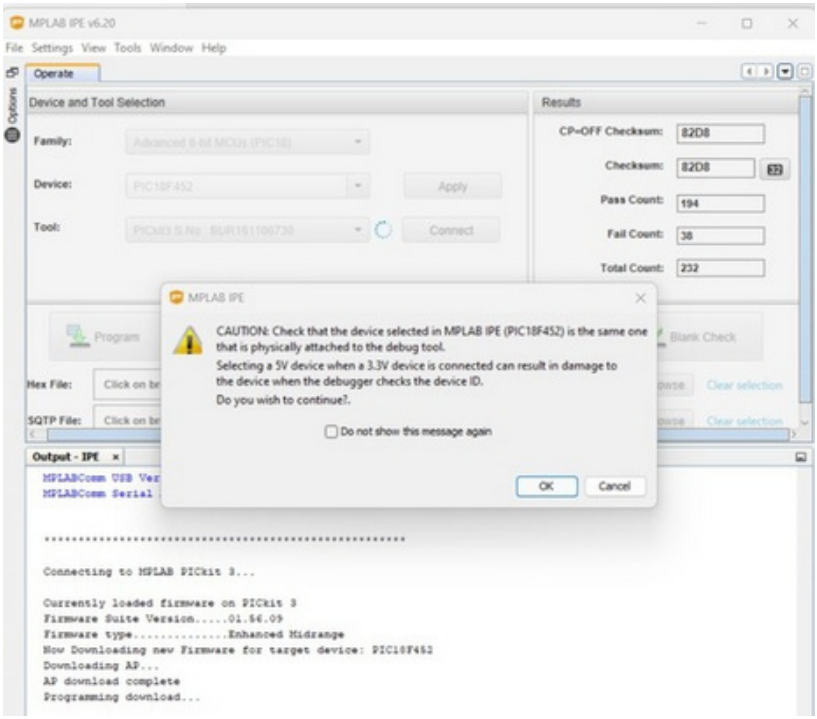
ICSP Conn & PICkit 3 Programmer.



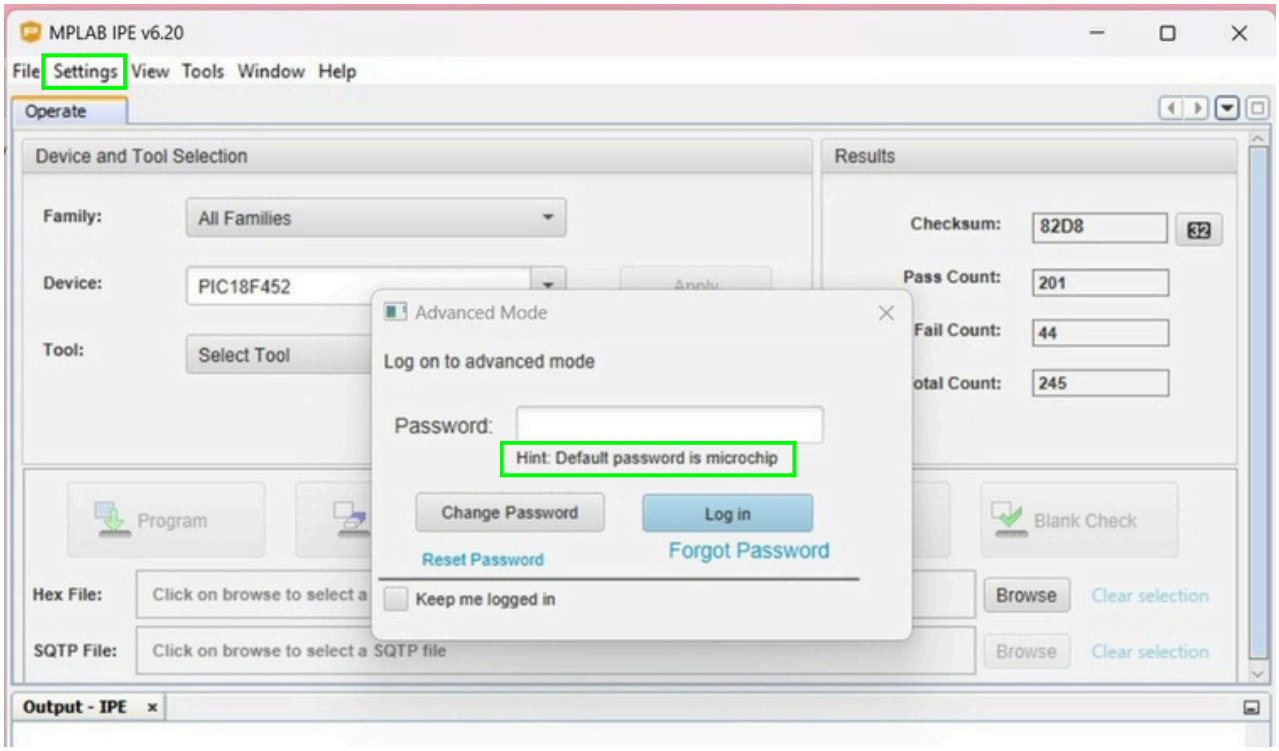
Reset Pin of ICSP Conn. on PIC18F/16F Kit



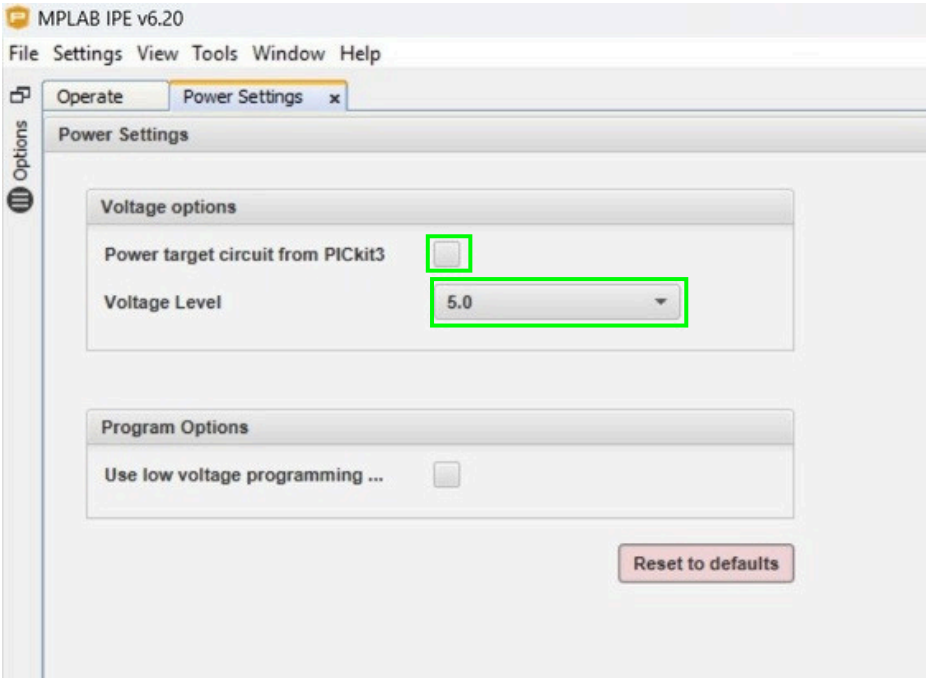
2. Once you click on Connect, below pop-up window may appear if you have not done Programming Power Source and Voltage level settings. Programming voltage and Power settings should be done before downloading the Program.



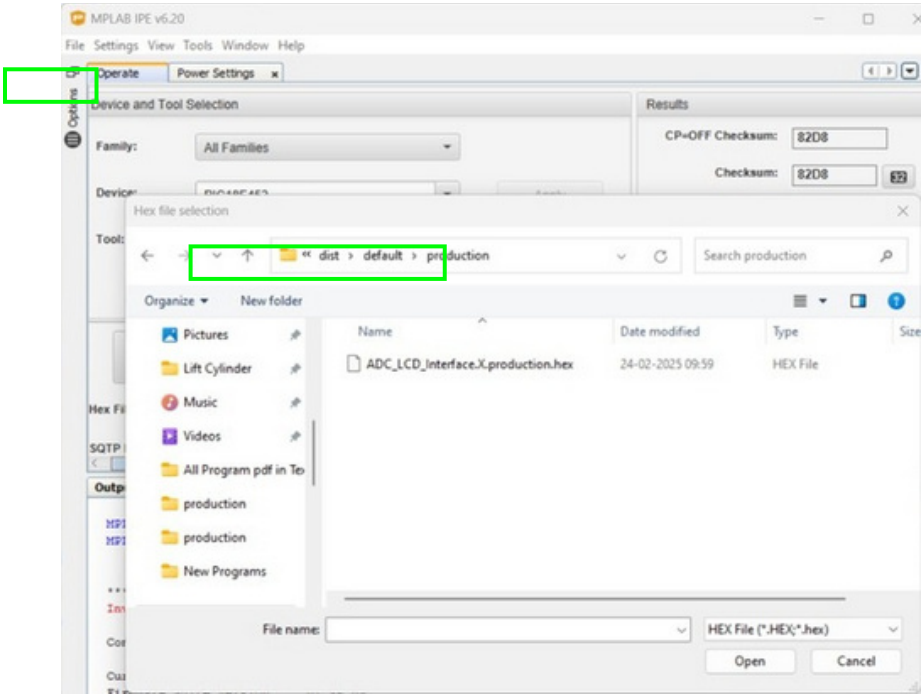
3. To confirm if correct Programming Voltage and Power Source is selected, we need to click on Setting and select Advance Mode. Then below Pop-Up window will appear. Enter the default password 'microchip' as given in the Hint and press Log in. Now you go in Advanced mode.



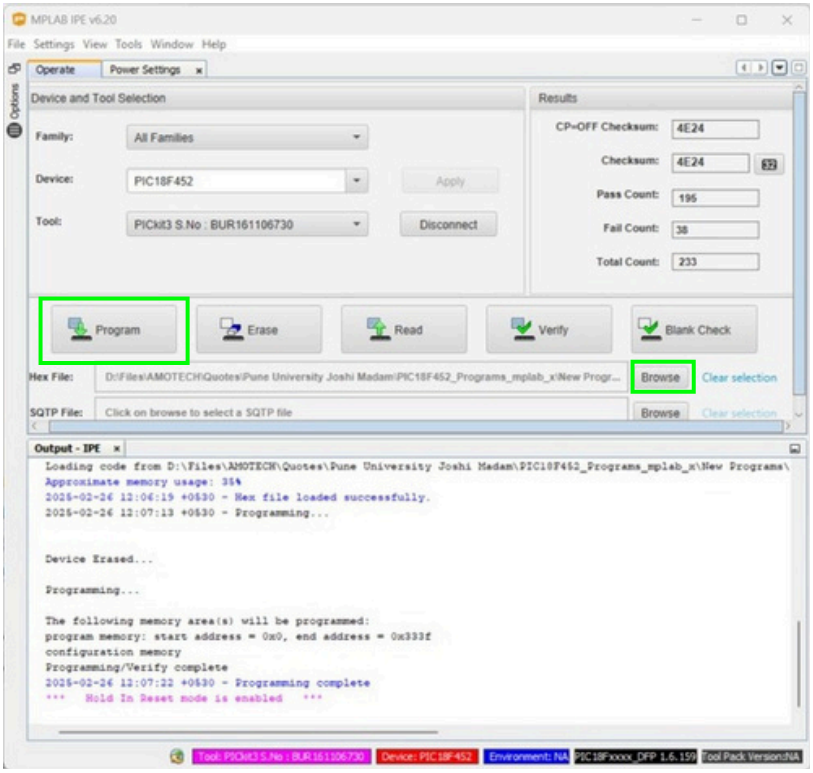
4. Once you are in Advance Mode, you will see the 'Options' tab on the left. Click on it and select 'Power' tab, then below window will appear. Un tick the below check box and select the Voltage level as 5.0 Volt. Now, must be powered by external Power Supply before downloading the program.



5 . Once Voltage and Power Source is selected, click on ‘Operate’ tab. Now, select you Hex file by by clicking o n ‘Browse’ and navigate to your Hex file location. In your Projet Folder you need to navigate to dist>default>production and select your Hex File.

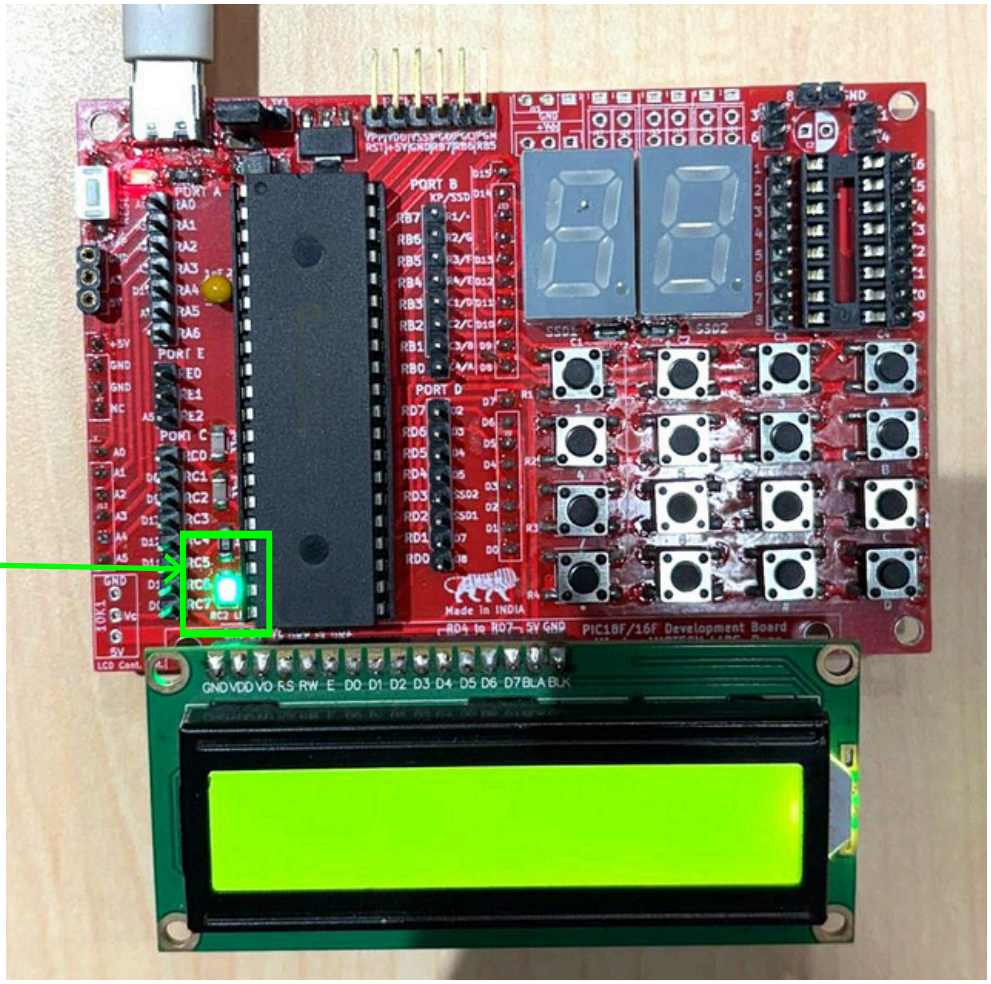


6. Once you select the Hex file you will see the notification Hex File Loaded Successfully in ‘Output IPE’. Click on ‘Program’ button.You will see PICKit 3 LED’s flashing and ‘Programming...’ in Output-IPE. And, finally ‘Programming complete’ once downloaded. Now remove PICKit 3 and see your program running on the Kit.



Lab 1 - LED Blinking

Aim: To study the LED's Interfacing with PIC18F452 mini development.



Note: The same code can be performed on the 'PIC18F/16F Development Board', an extended version of this board that features an on-board 2 seven-segment displays, a 4x4 Keypad, and an additional 16-pin DIP IC slot.

Procedure:

1.Read the LED Blink Program on next page with all the comments which explains the program.

2.Open the Project folder for the program in MPLAB X IDE or write it by making new project as per MPLAB X IDE procedure explained in the manual. Build and generate the Hex file for the same.

3.Open MPLAB IPE and follow the procedure explained to download the Hex file for the above program.

4.Use PICkit3 programmmer to program the PIC microcontroller with the HEX file. 5.Make sure that the arrow side of the programmer must be connected to the RST pin of the ICSP connector of the board.

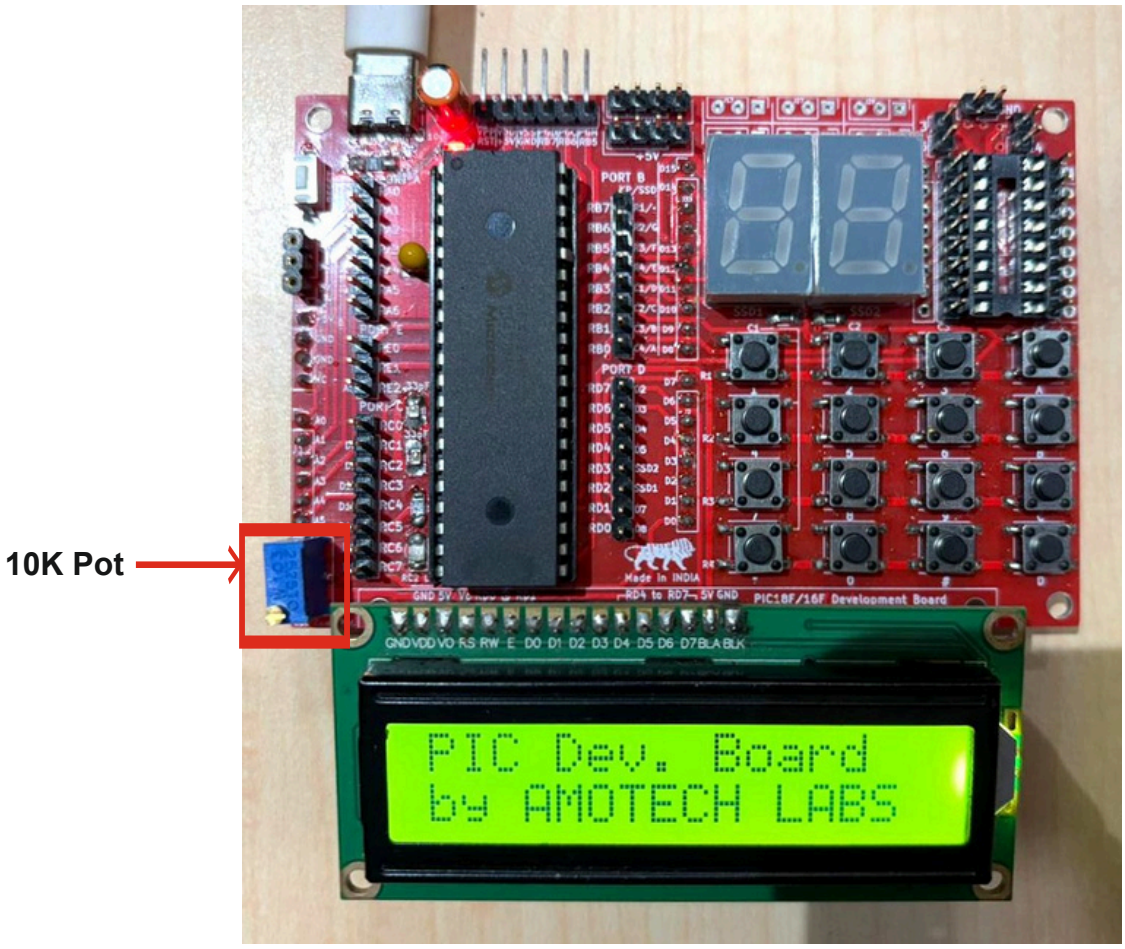
Program:

```
#include <xc.h>                         // XC8 core definitions
#define _XTAL_FREQ 20000000             // 20 MHz crystal frequency (for delays)
/*
  NOTE ABOUT _XTAL_FREQ AND OSCILLATOR SELECTION
  _XTAL_FREQ does NOT select the oscillator source.It only tells the XC8 compiler the CPU clock frequency
  so that __delay_ms() and __delay_us() work correctly.
The ACTUAL oscillator source is selected using the configuration bits (pragma config).

  Examples for PIC18 family MCUs:
  1) PIC18F4620 (Internal Oscillator)
     #pragma config OSC = INTIO67      // Internal RC oscillator
     OSCCON = 0x72;                   // 8 MHz internal clock
     #define _XTAL_FREQ 8000000        // Must match internal frequency
 *
  2) PIC18F4620 (External Crystal)
     ------------------------------
     #pragma config OSC = HS           // External high-speed crystal
     #define _XTAL_FREQ 20000000       // Typical dev-board crystal

  3) PIC18F452 / PIC18F4550
     ----------------------
     Commonly used with 8 MHz, 10 MHz, or 20 MHz external crystals:
     #pragma config OSC = HS
     #define _XTAL_FREQ 8000000   or
     #define _XTAL_FREQ 10000000  or
     #define _XTAL_FREQ 20000000

  IMPORTANT:
  - Internal oscillator mode frees RA6/RA7 as GPIO pins.
  - External crystal mode uses RA6/RA7 as oscillator pins.
  - _XTAL_FREQ MUST always match the actual CPU clock,
    otherwise delay functions will be incorrect.
*/

// Configuration bits
#pragma config OSC = HS               // High-speed external oscillator
#pragma config WDT = OFF              // Disable watchdog timer
#pragma config LVP = OFF              // Disable low-voltage programming

#define LED LATCbits.LATC2            // LED connected to RC2 pin
```

```
void main()
{
    TRISCbits.TRISC2 = 0;           // Set RC2 as output

    while(1)                        // Infinite loop
    {
        LED = 0;                    // Turn LED ON (active low)
        __delay_ms(1000);           // Delay 1 second

        LED = 1;                    // Turn LED OFF
        __delay_ms(1000);           // Delay 1 second
    }

    return;                         // Program never reaches here
}
```

Lab 2 - LCD Interfacing

Aim: To study the16x2 LCD Interfacing with PIC18F452 mini development kit.



Note: The same code can be performed on the 'PIC18F/16F Development Board', an extended version of this board that features an on-board 2 seven-segment displays, a 4x4 Keypad, and an additional 16-pin DIP IC slot.

Procedure:

1. Read the LCD Interfacing Program on next page with all the comments which explains the program.
2.Open the Project folder for the program in MPLAB X IDE or write it by making new project as per MPLAB X IDE procedure explained in the manual. Build and generate the Hex file for the same.
3.Open MPLAB IPE and follow the procedure explained to download the Hex file for the above program.
4.LCD contrast can be adjusted using 10k pot beside LCD. Once you download the program and remove the programmer, you will see above contents on the LCD.
5.Make sure that the arrow side of the programmer must be connected to the RST pin of the ICSP connector of the board.

```
#include <xc.h>
#include <pic18f452.h>

#define _XTAL_FREQ 20000000      // 20 MHz system clock
/*
  NOTE ABOUT _XTAL_FREQ AND OSCILLATOR SELECTION _XTAL_FREQ does NOT select the oscillator source.
  It only tells the XC8 compiler the CPU clock frequency so that __delay_ms() and __delay_us() work correctly.
 The ACTUAL oscillator source is selected using the configuration bits (pragma config).

  Examples for PIC18 family MCUs:
  1) PIC18F4620 (Internal Oscillator)
     --------------------------------
     #pragma config OSC = INTIO67     // Internal RC oscillator
     OSCCON = 0x72;                  // 8 MHz internal clock
     #define _XTAL_FREQ 8000000       // Must match internal frequency

  2) PIC18F4620 (External Crystal)
     ------------------------------
     #pragma config OSC = HS          // External high-speed crystal
     #define _XTAL_FREQ 20000000      // Typical dev-board crystal

  3) PIC18F452 / PIC18F4550
     ----------------------
     Commonly used with 8 MHz, 10 MHz, or 20 MHz external crystals:
     #pragma config OSC = HS
     #define _XTAL_FREQ 8000000   or
     #define _XTAL_FREQ 10000000  or
     #define _XTAL_FREQ 20000000

  IMPORTANT:
  - Internal oscillator mode frees RA6/RA7 as GPIO pins.
  - External crystal mode uses RA6/RA7 as oscillator pins.
  - _XTAL_FREQ MUST always match the actual CPU clock,
    otherwise delay functions will be incorrect.
*/

// Configuration bits
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config BOR = OFF

 // LCD pin connections (4-bit mode)
 #define RS LATDbits.LATD0        // Register Select
 #define EN LATDbits.LATD1        // Enable
 #define D4 LATDbits.LATD4        // Data bit 4
 #define D5 LATDbits.LATD5        // Data bit 5
 #define D6 LATDbits.LATD6        // Data bit 6
 #define D7 LATDbits.LATD7        // Data bit 7

 // LCD function prototypes
 void LCD_Init();
 void LCD_Command(unsigned char);
 void LCD_char(unsigned char);
 void LCD_string(const char*);
 void LCD_string_xy(char, char , const char*);
 void LCD_clear();

 int main(void)
{
    ADCON1 = 0x0F;               // Disable all analog inputs
    TRISD = 0x00;               // PORTD as output (LCD)
    LATD  = 0x00;               // Clear PORTD

    LCD_Init();                 // Initialize LCD
```

```
    LCD_Init();                 // Initialize LCD

    LCD_string_xy(1,0,"PIC Dev. Board");
    __delay_ms(1000);

    LCD_Command(0xC0);          // Move cursor to second line
    LCD_string_xy(2,0,"by AMOTECH LABS");

    while(1);                   // Stay here
}

void LCD_Init()
{
    __delay_ms(40);             // LCD power-up delay

    RS = 0;

    // Force LCD into 4-bit mode (initial sequence)
    D4=1; D5=1; D6=0; D7=0; EN=1; __delay_us(5); EN=0; __delay_ms(5);
    D4=1; D5=1; D6=0; D7=0; EN=1; __delay_us(5); EN=0; __delay_us(150);
    D4=1; D5=1; D6=0; D7=0; EN=1; __delay_us(5); EN=0;
    D4=0; D5=1; D6=0; D7=0; EN=1; __delay_us(5); EN=0;

    LCD_Command(0x28);          // 4-bit mode, 2 lines
    LCD_Command(0x0C);          // Display ON, cursor OFF
    LCD_Command(0x06);          // Auto increment cursor
    LCD_Command(0x01);          // Clear display
    __delay_ms(2);
}

void LCD_Command(unsigned char cmd)
{
    RS = 0;                     // Command mode

    // Send upper nibble
    D4 = (cmd >> 4) & 1;
    D5 = (cmd >> 5) & 1;
    D6 = (cmd >> 6) & 1;
    D7 = (cmd >> 7) & 1;
    EN = 1; __delay_us(5); EN = 0;

    // Send lower nibble
    D4 = cmd & 1;
    D5 = (cmd >> 1) & 1;
    D6 = (cmd >> 2) & 1;
    D7 = (cmd >> 3) & 1;
    EN = 1; __delay_us(5); EN = 0;

    __delay_us(50);             // Command execution delay
}

void LCD_char(unsigned char data)
{
    RS = 1;                     // Data mode

    // Send upper nibble
    D4 = (data >> 4) & 1;
    D5 = (data >> 5) & 1;
    D6 = (data >> 6) & 1;
    D7 = (data >> 7) & 1;
    EN = 1; __delay_us(5); EN = 0;
```

```
    // Send lower nibble
    D4 = data & 1;
    D5 = (data >> 1) & 1;
    D6 = (data >> 2) & 1;
    D7 = (data >> 3) & 1;
    EN = 1; __delay_us(5); EN = 0;

    __delay_us(50);             // Character write delay
}

void LCD_string(const char *msg)
{
    while(*msg)                 // Print until null character
    {
        LCD_char(*msg++);
    }
}

void LCD_string_xy(char row,char pos,const char *msg)
{
    if(row == 1)
        LCD_Command(0x80 + pos);    // First line
    else
        LCD_Command(0xC0 + pos);    // Second line

    LCD_string(msg);
}

void LCD_clear()
{
    LCD_Command(0x01);          // Clear LCD
    __delay_ms(3);
}
```

Lab 3 - ADC Interfacing

Aim: To study the ADC interfacing with PIC18F452 mini development kit.

Note: The same code can be performed on the 'PIC18F/16F Development Board', an extended version of this board that features an on-board 2 seven-segment displays, a 4x4 Keypad, and an additional 16-pin DIP IC slot.

Procedure:

1.Read the ADC Interfacing Program on next page with all the comments which explains the program. Open the Project folder for the program in MPLAB X IDE or write it by making new project as per the MPLAB X IDE procedure explained in the manual. Build and generate the Hex File for the same.
2.OpenMPLABIPEand follow the procedure explained to download the Hex file for the above Program.
3.UsePICkit3programmmer to program the PIC microcontroller with the HEX file.
4.Makesurethatthearrow side of the programmer must be connected to the RST pin of the ICSP connector of the board.

Program:

```
#include <xc.h>
#include <pic18f452.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define _XTAL_FREQ 20000000      // 20 MHz crystal frequency

// Configuration bits
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config BOR = OFF

// LCD pins (4-bit mode)
#define RS LATDbits.LATD0
#define EN LATDbits.LATD1
#define D4 LATDbits.LATD4
#define D5 LATDbits.LATD5
#define D6 LATDbits.LATD6
#define D7 LATDbits.LATD7

// Function declarations
void LCD_Initialize();
void LCD_COMMAND(unsigned char);
void LCD_char(unsigned char);
void LCD_string(const char *);
void LCD_string_xy(char,char,const char *);
void LCD_clear();
void ADC_initialize();
unsigned int ADC_read(unsigned char);

#define vref 5.00               // ADC reference voltage

int main(void){
    char data[10];
    unsigned int digital;
    float voltage;
    int k;
    char ADC_ARRAY[5];
    TRISD = 0X00;               // LCD port as output

    LCD_Initialize();           // Initialize LCD
    ADC_initialize();           // Initialize ADC
    LCD_string_xy(1,0,"voltage is...");
    LCD_COMMAND(0xC0);          // Move to second line

    while(1){
        digital = ADC_read(3);  // Read ADC channel AN3
        voltage = digital * (vref / 1023.0); // Convert ADC count to voltage
        for(k = 0; k <= 3; k++)// Convert ADC value to characters
        {
```

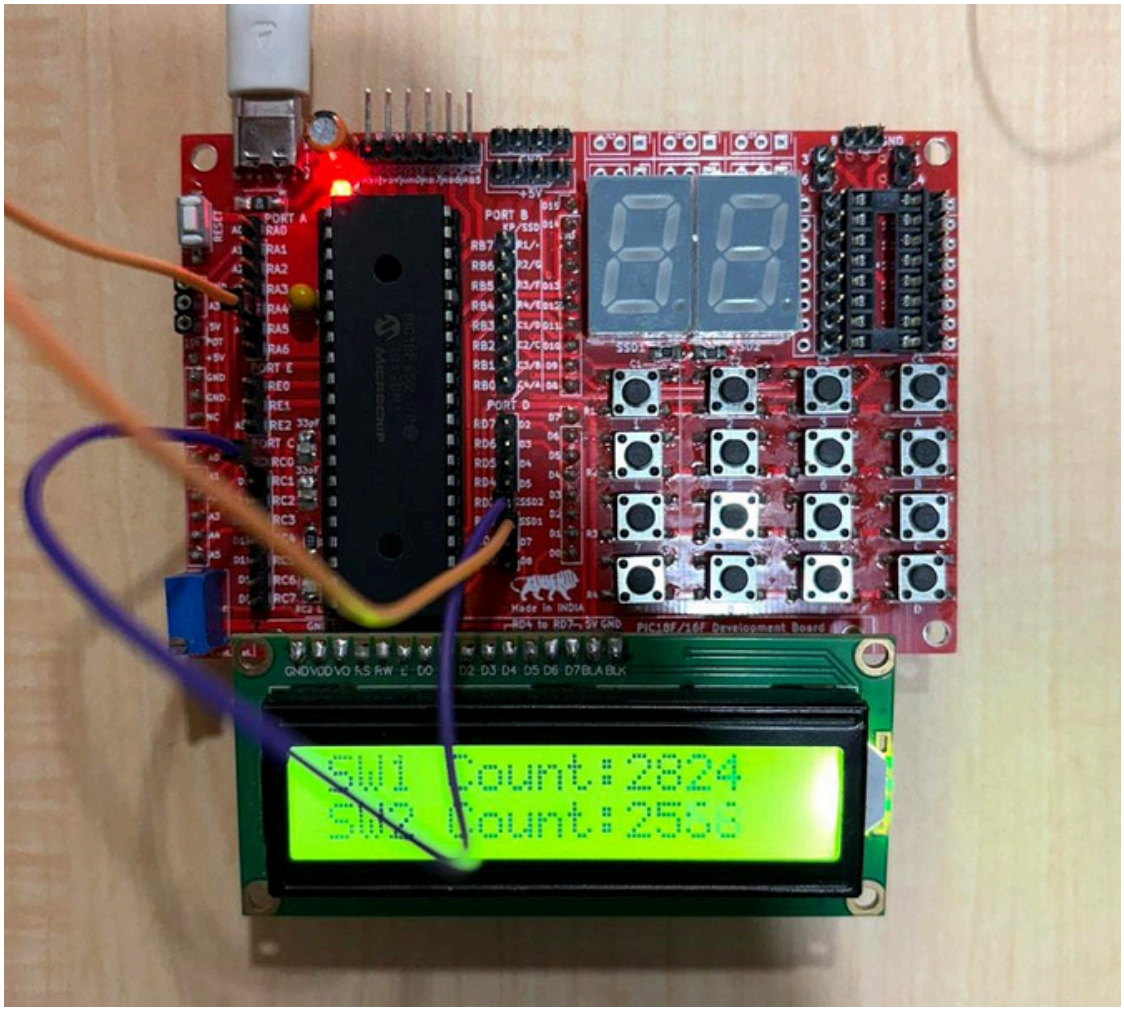
```
            ADC_ARRAY[k] = (digital % 10) + '0';
            digital /= 10;
        }

        LCD_COMMAND(0xC0);      // Second line start
        for(k = 3; k >= 0; k--)
        {
            LCD_char(ADC_ARRAY[k]);
        }

        sprintf(data,"%2f",voltage);
        strcat(data,"V");
        LCD_string_xy(2,6,data);

        __delay_ms(250);
    }
}
// ADC setup
void ADC_initialize()
{
    TRISA = 0XFF;               // PORTA as input
    ADRESH = 0;
    ADRESL = 0;
}

// Read ADC value
unsigned int ADC_read(unsigned char channel)
{
    unsigned int digital;

    ADCON0 = (ADCON0 & 0b11000111) | ((channel << 3) & 0b00111000);
    ADCON0bits.ADON = 1;
    ADCON0bits.GO_nDONE = 1;

    while(ADCON0bits.GO_nDONE); // Wait for conversion

    digital = ADRESH;
    digital = (digital << 8) | ADRESL;
    digital >>= 6;              // Align 10-bit result

    return digital;
}

// LCD initialization
void LCD_Initialize()
{
    LCD_COMMAND(0x02);
    LCD_COMMAND(0x28);
    LCD_COMMAND(0x0C);
    LCD_COMMAND(0x06);
    LCD_COMMAND(0x01);
    __delay_ms(2);
}
```

```
// Send command to LCD
void LCD_COMMAND(unsigned char cmd)
{
    RS = 0;

    D4 = (cmd >> 4) & 1;
    D5 = (cmd >> 5) & 1;
    D6 = (cmd >> 6) & 1;
    D7 = (cmd >> 7) & 1;
    EN = 1; __delay_ms(5); EN = 0;

    D4 = cmd & 1;
    D5 = (cmd >> 1) & 1;
    D6 = (cmd >> 2) & 1;
    D7 = (cmd >> 3) & 1;
    EN = 1; __delay_ms(5); EN = 0;
}

// Send character to LCD
void LCD_char(unsigned char data)
{
    RS = 1;

    D4 = (data >> 4) & 1;
    D5 = (data >> 5) & 1;
    D6 = (data >> 6) & 1;
    D7 = (data >> 7) & 1;
    EN = 1; __delay_us(5); EN = 0;

    D4 = data & 1;
    D5 = (data >> 1) & 1;
    D6 = (data >> 2) & 1;
    D7 = (data >> 3) & 1;
    EN = 1; __delay_us(5); EN = 0;
}

// Display string
void LCD_string(const char *msg)
{
    while(*msg)
        LCD_char(*msg++);
}

// Set cursor and display string
void LCD_string_xy(char row, char pos, const char *msg)
{
    unsigned char location;
```

```
    location = (row == 1) ? (0x80 + pos) : (0xC0 + pos);
    LCD_COMMAND(location);
    LCD_string(msg);
}

// Clear LCD
void LCD_clear()
{
    LCD_COMMAND(0x01);
    __delay_ms(3);
}
```

Lab 4 - Pulse Counter

Aim: To study the Pulse counter program with PIC18F451 mini board.



Note: The same code can be performed on the 'PIC18F/16F Development Board', an extended version of this board that features an on-board 2 seven-segment displays, a 4x4 Keypad, and an additional 16-pin DIP IC slot.

Procedure:

1.Read the pulse counter Program on next page with all the comments which explains the program.
2.Open the Project folder for the program in MPLAB X IDE or write it by making new project as per MPLAB X IDE procedure explained in the manual. Build and generate the Hex file for the same.
3.Open MPLAB IPE and follow the procedure explained to download the Hex file for the above program.
4.Make the connections as above shown in figure.

Program:

```
#include <xc.h>                     // Core XC8 compiler definitions
#define _XTAL_FREQ 20000000         // System clock frequency for delay macros
#include <pic18f452.h>              // Device-specific definitions for PIC18F452
#include <string.h>                 // Required for string operations
#include <stdio.h>                  // Required for sprintf()

#pragma config OSC = HS             // High-speed external crystal oscillator
#pragma config WDT = OFF            // Disable Watchdog Timer
#pragma config LVP = OFF            // Disable Low-Voltage Programming
#pragma config BOR = OFF            // Disable Brown-Out Reset

// LCD pin connections (4-bit mode)
#define RS LATDbits.LATD0            // Register Select pin
#define EN LATDbits.LATD1            // Enable pin
#define D4 LATDbits.LATD4            // LCD data bit D4
#define D5 LATDbits.LATD5            // LCD data bit D5
#define D6 LATDbits.LATD6            // LCD data bit D6
#define D7 LATDbits.LATD7            // LCD data bit D7

// Function declarations
void lcd_init(void);                // Initialize LCD
void lcd_command(unsigned char cmd);// Send command to LCD
void lcd_char(unsigned char data);  // Send character to LCD
void lcd_string(const char *str);   // Send string to LCD
void lcd_clear(void);               // Clear LCD display
void lcd_setcursor(unsigned char row, unsigned char column); // Set LCD cursor
void setup();                       // Initialize timers, ports, LCD
void displaycount(unsigned int count); // Display a count value

volatile unsigned int pulseCount = 0; // Volatile variable (not used here)
```

```
void setup()
{
    ADCON1 = 0x0F;          // All pins digital

    TRISD = 0x00;           // LCD port output

    // Timer1 (SW1 on RC0)
    TRISCbits.TRISC0 = 1;   // RC0 = T1CKI input
    T1CON = 0x87;           // External clock, Timer1 ON
    TMR1H = 0;
    TMR1L = 0;

    // Timer0 (SW2 on RA4)
    TRISAbits.TRISA4 = 1;   // RA4 = T0CKI input
    T0CON = 0xA9;           // Counter mode, external clock, Timer0 ON
    TMR0H = 0;
    TMR0L = 0;

    lcd_init();
}
```

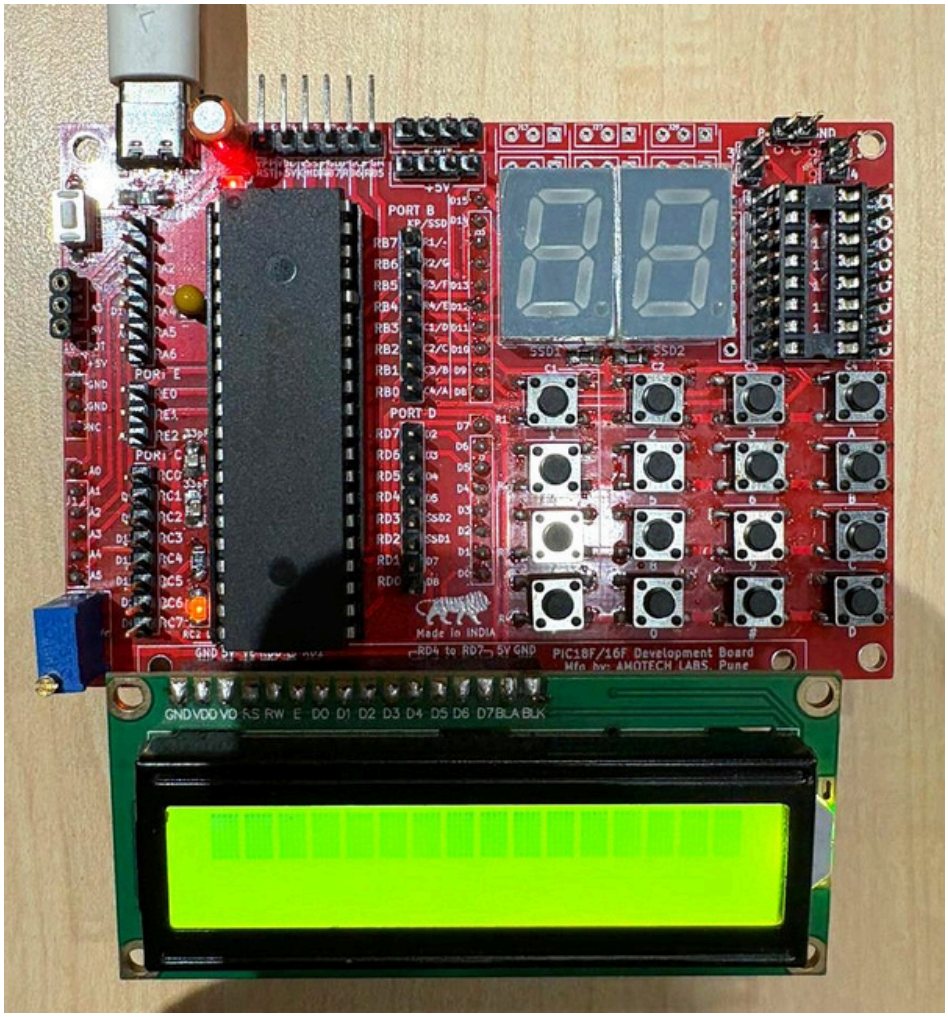
```
void main()
{
    setup();                        // Call hardware and LCD setup
    lcd_clear();                    // Clear LCD display
```

```
    while(1)                        // Infinite loop
    {
        // Read 16-bit Timer1 value
        unsigned int t1PulseCount = (TMR1H << 8) | TMR1L;

        // Read 16-bit Timer0 value
        unsigned int t0PulseCount = (TMR0H << 8) | TMR1L;

        char t0buffer[16], t1buffer[16]; // Buffers for LCD text

        // Format Timer1 count for LCD
        sprintf(t1buffer,"SW1 Count:%u",t1PulseCount);

        // Format Timer0 count for LCD
        sprintf(t0buffer,"SW2 Count:%u",t0PulseCount);

        lcd_setcursor(1,1);         // Move cursor to row 1, column 1
        lcd_string(t1buffer);       // Display Timer1 count

        lcd_setcursor(2,1);         // Move cursor to row 2, column 1
        lcd_string(t0buffer);       // Display Timer0 count

        __delay_ms(50);             // Refresh delay
    }
}

// Display a single count value (not used in main)
void displaycount(unsigned int count)
{
    char buffer[16];                // Buffer for LCD text
    lcd_clear();                    // Clear LCD
    sprintf(buffer,"Count: %u",count); // Format count
    lcd_string(buffer);             // Display count
}

 // Initialize LCD in 4-bit mode
 void lcd_init(void)
 {
     lcd_command(0x02);             // Set 4-bit mode
     lcd_command(0x28);             // 2-line display, 5x7 font
     lcd_command(0x0C);             // Display ON, cursor OFF
     lcd_command(0x06);             // Auto-increment cursor
     lcd_command(0x01);             // Clear display
     __delay_ms(2);                 // Command execution delay
 }

 // Send command to LCD
 void lcd_command(unsigned char cmd)
 {
     RS = 0;                        // Select command register
```

```
    // Send upper nibble
    D4 = (cmd>>4)&0x01;
    D5 = (cmd>>5)&0x01;
    D6 = (cmd>>6)&0x01;
    D7 = (cmd>>7)&0x01;
    EN = 1; __delay_ms(5); EN = 0; __delay_ms(50);

    // Send lower nibble
    D4 = cmd & 0x01;
    D5 = (cmd>>1)&0x01;
    D6 = (cmd>>2)&0x01;
    D7 = (cmd>>3)&0x01;
    EN = 1; __delay_ms(5); EN = 0; __delay_ms(50);
}

// Send one character to LCD
void lcd_char(unsigned char data)
{
    RS = 1;                          // Select data register

    // Upper nibble
    D4 = (data>>4)&0x01;
    D5 = (data>>5)&0x01;
    D6 = (data>>6)&0x01;
    D7 = (data>>7)&0x01;
    EN = 1; __delay_us(5); EN = 0; __delay_us(50);

    // Lower nibble
    D4 = data & 0x01;
    D5 = (data>>1)&0x01;
    D6 = (data>>2)&0x01;
    D7 = (data>>3)&0x01;
    EN = 1; __delay_us(5); EN = 0; __delay_us(50);
}

// Set LCD cursor position
void lcd_setcursor(unsigned char row, unsigned char column)
{
    unsigned char position;

    if(row == 1)
        position = 0x80 + column - 1; // First row address
    else if (row == 2)
        position = 0xC0 + column - 1; // Second row address

    lcd_command(position);           // Send cursor position command
}

// Display string on LCD
void lcd_string(const char *str)
{
    while (*str)                     // Loop until null terminator
    {
        lcd_char(*str++);            // Display each character
    }
}

// Clear LCD display
void lcd_clear(void)
{
    lcd_command(0x01);               // Clear display command
    __delay_ms(2);                   // Execution delay
}
```

Lab 5 - PWM

Aim: To study the PWM with PIC18F452 mini development kit



Note: The same code can be performed on the 'PIC18F/16F Development Board', an extended version of this board that features an on-board 2 seven-segment displays, a 4x4 Keypad, and an additional 16-pin DIP IC slot.

Procedure:

1.Read the PWM Program on next page with all the comments which explains the program.
2.Open the Project folder for the program in MPLAB X IDE or write it by making new project as per MPLAB X IDE procedure explained in the manual. Build and generate the Hex file for the same.
3.Open MPLAB IPE and follow the procedure explained to download the Hex file for the above program.
4.As you vary the PWM signals fed to pin RC2, you will see brightness of LED connected to RC2 changes.
5.Make sure that the arrow side of the programmer must be connected to the RST pin of the ICSP connector of the board.

Program:

```
#include <xc.h>                 // Core XC8 compiler definitions
#include <pic18f452.h>          // Device-specific definitions for PIC18F452
#include <string.h>             // Included for string functions (not used here)

#pragma OSC = HS                // Use High-Speed external crystal oscillator
#pragma WDT = OFF               // Disable Watchdog Timer
#pragma LVP = OFF               // Disable Low-Voltage Programming
#pragma BOR = OFF               // Disable Brown-Out Reset

#define _XTAL_FREQ 20000000     // Define system clock frequency for delay macro

void main()
{
    unsigned int duty_cycle;    // Variable to vary PWM duty cycle

    TRISCbits.TRISC2 = 0;       // Set RC2 pin as OUTPUT (CCP1 PWM output pin)

    PR2 = 124;                  // Set PWM period using Timer2
                                // Determines PWM frequency

    CCPR1L = 1;                 // Initial PWM duty cycle (very small ON time)

    T2CON = 0X04;               // Timer2 ON, prescaler = 1:1

    CCP1CON = 0X0C;             // Configure CCP1 module in PWM mode

    TMR2 = 0;                   // Clear Timer2 register

    T2CONbits.TMR2ON = 1;       // Turn ON Timer2 (starts PWM generation)

    while(1)                    // Infinite loop
    {
        // Gradually increase duty cycle (ramp up)
        for(duty_cycle = 1; duty_cycle < 124; duty_cycle++)
        {
            CCPR1L = duty_cycle; // Update PWM duty cycle
            __delay_ms(20);      // Small delay to make change visible
        }

        __delay_ms(500);         // Hold at maximum speed for a while

        // Gradually decrease duty cycle (ramp down)
        for(duty_cycle = 124; duty_cycle > 1; duty_cycle--)
        {
        CCPR1L = duty_cycle; // Update PWM duty cycle
        __delay_ms(20);      // Small delay to make change visible
    }

    __delay_ms(500);         // Hold at minimum speed for a while
}
}
```

Lab 6-Interfacing of Keypad and Seven Segment Display

Aim: To study the Interfacing of Keypad and 2 Seven Segment Displays on a 'PIC18F/16F Development Board' which has on-board Keypad, 2 Seven Segment Displays and 1 L293D IC/16 pin DIP IC slot.

Procedure:

1.Inthis program, we enter the 4 digit code '4201' after which 'Access Granted' appears on LCD and a 2 digit number from 0 to 99 are displayed on 2 Seven Segment Displays after 1 Second delay.
2.Read the Program on next page with all the comments which explains the program.
3.Open the Project folder for the program in MPLAB X IDE or write it by making new project as per MPLAB X IDE procedure explained in the manual. Build and generate the Hex file for the same.
4.Open MPLAB IPE and follow the procedure explained to download the Hex file for the above program.
5.As you vary the PWM signals fed to pin RC2, you will see brightness of LED connected to RC2 changes.
6.Make sure that the arrow side of the programmer must be connected to the RST pin of the ICSP connector of the board.

Program:

```
#include <xc.h>                 // Core XC8 compiler header
#include <pic18f452.h>          // Device-specific definitions for PIC18F452
#include <string.h>             // Required for string comparison functions
#include <stdio.h>              // Standard I/O (used for formatting if needed)
#include <stdlib.h>             // Standard library functions

#define _XTAL_FREQ 20000000     // Defines crystal frequency for delay macros

#pragma OSC = HS                // High-speed external oscillator
#pragma LVP = OFF               // Disable low-voltage programming
#pragma WDT = OFF               // Disable watchdog timer
#pragma BOR = OFF               // Disable brown-out reset

// LCD pin definitions (4-bit mode)
#define RS LATDbits.LATD0        // LCD Register Select pin
#define EN LATDbits.LATD1        // LCD Enable pin
#define D4 LATDbits.LATD4        // LCD data bit D4
#define D5 LATDbits.LATD5        // LCD data bit D5
#define D6 LATDbits.LATD6        // LCD data bit D6
#define D7 LATDbits.LATD7        // LCD data bit D7

// Keypad column inputs
#define col1 PORTBbits.RB3
#define col2 PORTBbits.RB2
#define col3 PORTBbits.RB1
#define col4 PORTBbits.RB0

// Keypad row inputs
#define row1 PORTBbits.RB7
#define row2 PORTBbits.RB6
#define row3 PORTBbits.RB5
#define row4 PORTBbits.RB4

// Seven-segment digit enable pins
#define SSD1 PORTDbits.RD2       // Enable first 7-segment digit
#define SSD2 PORTDbits.RD3       // Enable second 7-segment digit

// Lookup table for common-cathode 7-segment display digits 0-9
unsigned char value[10] =
{
    0X3F,   // 0
    0X06,   // 1
    0X5B,   // 2
    0X4F,   // 3
    0X66,   // 4
    0X6D,   // 5
    0X7D,   // 6
    0X07,   // 7
```

```
    0X7F,   // 8
    0X6F    // 9
},
j, k;                          // Loop variables for 7-segment counting

// Function declarations
void LCD_Init();
void LCD_Command(unsigned char);
void LCD_Char(unsigned char x);
void LCD_string(const char *);
void lcd_clear();

// Password buffer and index
char pass[4], i = 0;

// Keypad initialization
void keypad_init()
{
    TRISB = 0XF0;                  // Upper nibble input (rows), lower nibble outp
    PORTB = 0X00;                  // Clear PORTB
    INTCON2bits.RBPU = 0;          // Enable PORTB internal pull-ups
}

// LCD initialization sequence
void LCD_Init()
{
    LCD_Command(0x02);             // Initialize LCD in 4-bit mode
    LCD_Command(0x28);             // 2-line display, 5x7 font
    LCD_Command(0x0C);             // Display ON, cursor OFF
    LCD_Command(0x06);             // Auto-increment cursor
    LCD_Command(0x01);             // Clear display
    __delay_ms(2);                 // Command execution delay
}

// Send command to LCD
void LCD_Command(unsigned char cmd)
{
    RS = 0;                         // Select command register

    // Send upper nibble
    D4 = (cmd>>4)& 0x01;
    D5 = (cmd>>5)& 0x01;
    D6 = (cmd>>6)& 0x01;
```

```
    D7 = (cmd>>7)& 0x01;
    EN = 1; __delay_ms(5); EN = 0; __delay_ms(50);

    // Send lower nibble
    D4 = cmd & 0x01;
    D5 = (cmd>>1)& 0x01;
    D6 = (cmd>>2)& 0x01;
    D7 = (cmd>>3)& 0x01;
    EN = 1; __delay_ms(5); EN = 0; __delay_ms(50);
}

// Send a character to LCD
void LCD_Char(unsigned char data)
{
    RS = 1;                     // Select data register

    // Upper nibble
    D4 = (data>>4)& 0x01;
    D5 = (data>>5)& 0x01;
    D6 = (data>>6)& 0x01;
    D7 = (data>>7)& 0x01;
    EN = 1; __delay_ms(5); EN = 0; __delay_ms(50);

    // Lower nibble
    D4 = data & 0x01;
    D5 = (data>>1) & 0x01;
    D6 = (data>>2) & 0x01;
    D7 = (data>>3) & 0x01;
    EN = 1; __delay_ms(5); EN = 0; __delay_ms(50);
}

// Display string on LCD
void LCD_string(const char *msg)
{
    while((*msg) != 0)          // Loop until null terminator
    {
        LCD_Char(*msg);         // Display character
        msg++;
    }
}

// Clear LCD display
void lcd_clear()
```

```
] {
    LCD_Command(0x01);          // Clear display command
    __delay_ms(3);
}

// Read 4x4 keypad and store password
void keypad()
] {
    int cursor = 192, flag = 0; // Cursor position for LCD
    lcd_clear();
    LCD_string("ENTER UR PASSKEY");
    LCD_Command(0xC0);          // Move cursor to second line
    i = 0;

    while(i < 4)                // Read 4 key presses
    {
        flag = cursor;

        // Column 1 active
        col1=0; col2=col3=col4=1;
        if(!row1){ LCD_Char('1'); pass[i++]='1'; cursor++; while(!row1); }
        else if(!row2){ LCD_Char('4'); pass[i++]='4'; cursor++; while(!row2); }
        else if(!row3){ LCD_Char('7'); pass[i++]='7'; cursor++; while(!row3); }
        else if(!row4){ LCD_Char('*'); pass[i++]='*'; cursor++; while(!row4); }

        // Column 2 active
        col2=0; col1=col3=col4=1;
        if(!row1){ LCD_Char('2'); pass[i++]='2'; cursor++; while(!row1); }
        else if(!row2){ LCD_Char('5'); pass[i++]='5'; cursor++; while(!row2); }
        else if(!row3){ LCD_Char('8'); pass[i++]='8'; cursor++; while(!row3); }
        else if(!row4){ LCD_Char('0'); pass[i++]='0'; cursor++; while(!row4); }
```

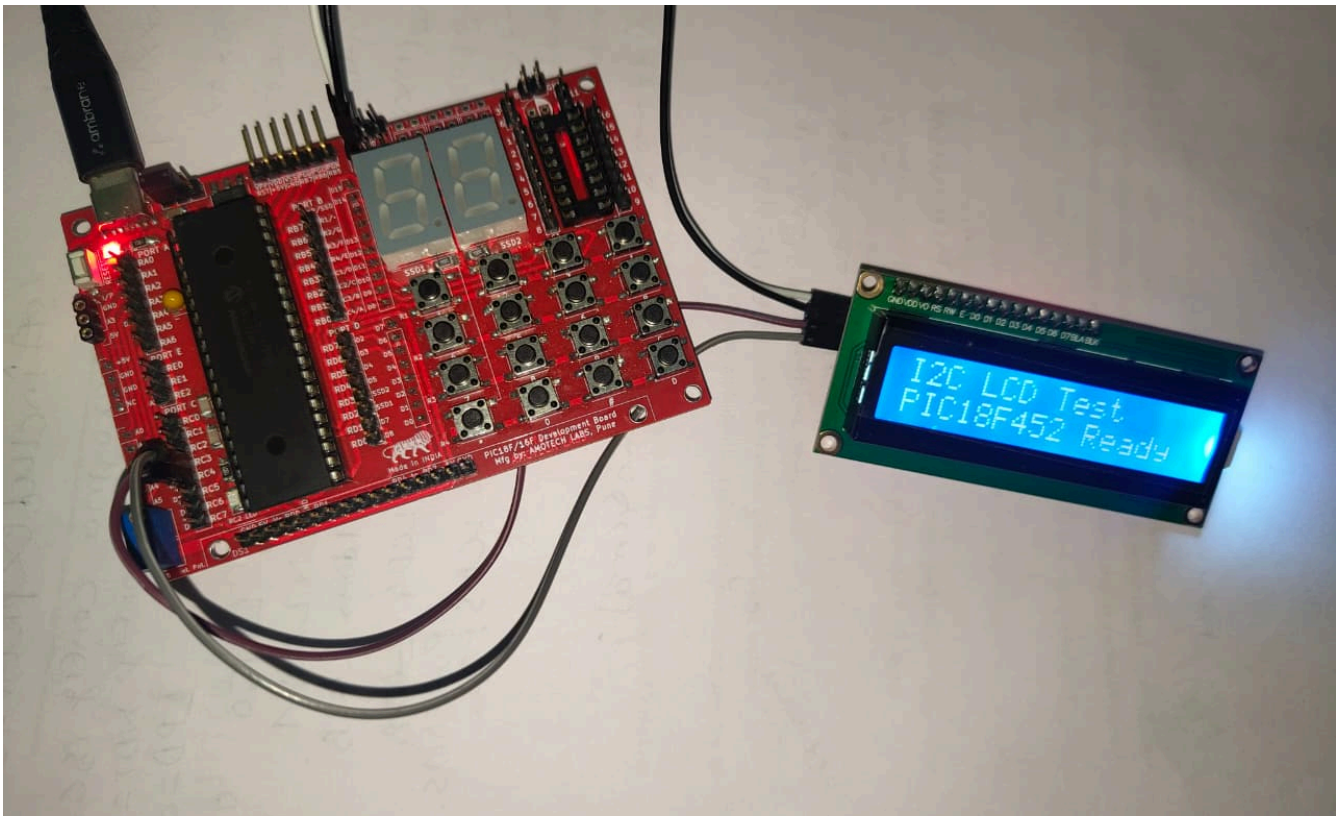
```
        // Column 2 active
        col2=0; col1=col3=col4=1;
        if(!row1){ LCD_Char('2'); pass[i++]='2'; cursor++; while(!row1); }
        else if(!row2){ LCD_Char('5'); pass[i++]='5'; cursor++; while(!row2); }
        else if(!row3){ LCD_Char('8'); pass[i++]='8'; cursor++; while(!row3); }
        else if(!row4){ LCD_Char('0'); pass[i++]='0'; cursor++; while(!row4); }

        // Column 3 active
        col3=0; col1=col2=col4=1;
        if(!row1){ LCD_Char('3'); pass[i++]='3'; cursor++; while(!row1); }
        else if(!row2){ LCD_Char('6'); pass[i++]='6'; cursor++; while(!row2); }
        else if(!row3){ LCD_Char('9'); pass[i++]='9'; cursor++; while(!row3); }
        else if(!row4){ LCD_Char('#'); pass[i++]='#'; cursor++; while(!row4); }

        // Column 4 active
        col4=0; col1=col2=col3=1;
        if(!row1){ LCD_Char('A'); pass[i++]='A'; cursor++; while(!row1); }
        else if(!row2){ LCD_Char('B'); pass[i++]='B'; cursor++; while(!row2); }
        else if(!row3){ LCD_Char('C'); pass[i++]='C'; cursor++; while(!row3); }
        else if(!row4){ LCD_Char('D'); pass[i++]='D'; cursor++; while(!row4); }
```

```
            // Mask typed character with '*'
            if(i > 0)
            {
                if(flag != cursor)
                    __delay_ms(100);
                LCD_Command(cursor - 1);
                LCD_Char('*');
            }
        }
    }

    // Password accepted message
    void accept()
    {
        lcd_clear();
        LCD_string("WELCOME");
        LCD_Command(192);
        LCD_string("Password accept");
        __delay_ms(200);
    }
    // Wrong password message
    void wrong()
    {
        lcd_clear();
        LCD_string("Wrong passkey");
        LCD_Command(192);
        LCD_string("Plz try again!");
        __delay_ms(200);
    }

    // Main program
    void main()
    {
        INTCON = 0X00;              // Disable all interrupts
        TRISD = 0X00;               // PORTD as output (LCD + SSD)
        LCD_Init();                 // Initialize LCD

        LCD_string("ELECTRONIC CODE");
        LCD_Command(0xC0);
        LCD_string("LOCK SYSTEM");
        __delay_ms(2000);

        lcd_clear();
        LCD_string("BY ANII!");
        __delay_ms(2000);

        while(1)
        {
            i = 0;
```

```
        keypad_init();          // Initialize keypad
        keypad();               // Read password

        // Check entered password
        if(strncmp(pass,"4201",4) == 0)
        {
            accept();
            lcd_clear();
            LCD_string("ACCESS GRANTED");
            __delay_ms(300);

            TRISB = 0X00;       // PORTB as output (7-segment)
            INTCON2bits.RBPU = 1;

            // Infinite loop: two-digit up counter on 7-segment
            while(1)
            {
                SSD1 = 1; SSD2 = 1;
                for(i=0;i<10;i++)
                {
                    for(j=0;j<10;j++)
                    {
                        for(k=0;k<100;k++)
                        {
                            SSD1 = 0; SSD2 = 1;
                            PORTB = value[i];
                            __delay_ms(5);

                            SSD1 = 1; SSD2 = 0;
                            PORTB = value[j];
                            __delay_ms(5);
                        }
                    }
                }
            }
        }
        else
        {
            lcd_clear();
            LCD_string("ACCESS DENIED");
            wrong();
            __delay_ms(300);
        }
    }
}
```

Lab 7-Interfacing of I2C LCD Display

Aim: To study and implement the interfacing of a 16×2 LCD module using the I2C communication protocol with a PIC18F/PIC16F micrnocontroller developmet board.



Procedure:

- Connect the I2C LCD (SDA, SCL, VCC, GND) to the PIC development board.
- Read the program with comments to understand I2C initialization and LCD functions.
- Open the project in MPLAB X IDE (or create new), build and generate the Hex file.
- Load the Hex file in MPLAB IPE and program the microcontroller.
- Power ON the board and verify that the text appears on the I2C LCD.
- Ensure the programmer's arrow side is connected towards the RST pin of the ICSP header.

Program:

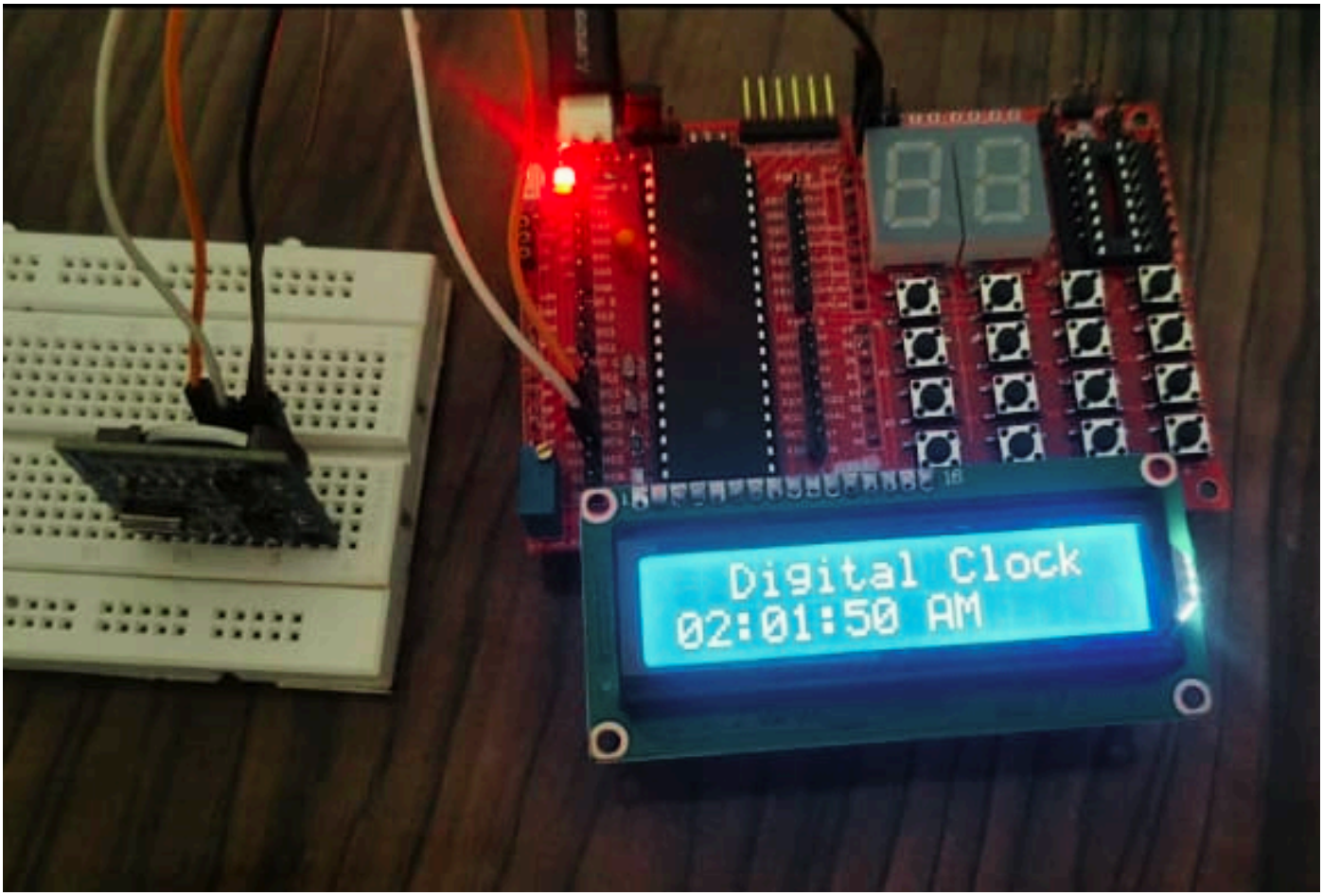
```
#include <xc.h>

#pragma config OSC = HS      // Oscillator Selection (High Speed Crystal)
#pragma config WDT = OFF     // Watchdog Timer (Disabled)
#pragma config PWRT = ON     // Power-up Timer (Enabled)
#pragma config BOR = ON      // Brown-out Reset (Enabled)
#pragma config LVP = OFF     // Low Voltage ICSP (Disabled)

#define _XTAL_FREQ 20000000 // 20 MHz
// --- I2C LCD Settings ---
#define I2C_LCD_ADDR 0x4E   // Common address (Use 0x7E if 0x4E fails)
#define BACKLIGHT    0x08   // Bit 3 controls backlight
#define ENABLE_BIT   0x04   // Bit 2 is 'E' pin
#define RS_BIT       0x01   // Bit 0 is 'RS' pin

// --- I2C Basic Functions ---
void I2C_Init(void) {
    TRISC3 = 1;             // SCL as input
    TRISC4 = 1;             // SDA as input
    SSPSTAT = 0x80;         // Slew rate disabled for 100kHz
    SSPCON1 = 0x28;         // Master Mode, Serial Port Enabled
    SSPADD = 49;            // 100kHz Baud Rate @ 20MHz
}

void I2C_Wait(void) {
    while ((SSPSTAT & 0x04) || (SSPCON2 & 0x1F));
}

void I2C_Start(void) {
    I2C_Wait();
    SEN = 1;
}

void I2C_Stop(void) {
    I2C_Wait();
    PEN = 1;
```

```
}

void I2C_Write(unsigned char data) {
    I2C_Wait();
    SSPBUF = data;
}

// --- LCD Over I2C Functions ---
void LCD_Send_Byte(unsigned char data, unsigned char rs) {
    unsigned char high_nibble = (data & 0xF0) | rs | BACKLIGHT;
    unsigned char low_nibble = ((data << 4) & 0xF0) | rs | BACKLIGHT;

    I2C_Start();
    I2C_Write(I2C_LCD_ADDR);
    I2C_Write(high_nibble | ENABLE_BIT); // Pulse High
    I2C_Write(high_nibble);              // Pulse Low

    // Send Low Nibble
    I2C_Write(low_nibble | ENABLE_BIT);  // Pulse High
    I2C_Write(low_nibble);               // Pulse Low
    I2C_Stop();
}

void LCD_Cmd(unsigned char cmd) {
    LCD_Send_Byte(cmd, 0);               // RS = 0 for commands
    if (cmd == 0x01) __delay_ms(2);      // Clear display needs extra time
}

void LCD_Char(unsigned char data) {
    LCD_Send_Byte(data, RS_BIT);         // RS = 1 for data
}

void LCD_Init(void) {
    __delay_ms(50);                      // Wait for LCD to power up
    LCD_Cmd(0x02);                       // 4-bit mode initialization
    LCD_Cmd(0x28);                       // 2 lines, 5x7 matrix
    LCD_Cmd(0x0C);                       // Display ON, Cursor OFF
    LCD_Cmd(0x06);                       // Entry mode
    LCD_Cmd(0x01);                       // Clear Display
}
```

```
void LCD_String(const char* s) {
    while(*s) LCD_Char(*s++);
}

// --- Main Application ---
void main(void) {
    ADCON1 = 0x07;          // Configure all PORTA/PORTE pins as digital
    I2C_Init();
    LCD_Init();

    LCD_Cmd(0x80);          // Line 1, Position 1
    LCD_String("I2C LCD Test");

    LCD_Cmd(0xC0);          // Line 2, Position 1
    LCD_String("PIC18F452 Ready");

    while(1) {
        // Your application logic here
    }
}
```

Lab 8 – Interfacing of RTC (Real Time Clock) Module

Aim: To study and implement the interfacing of a Real Time Clock (RTC) module using the I2C communication protocol with a PIC18F/PIC16F microcontroller to read and display real-time date and time.



Procedure:

1. Connect the RTC module (SDA, SCL, VCC, GND) to the PIC development board.
2. Read the program to understand I2C communication and time read/write functions.
3. Open or create the project in MPLAB X IDE, build and generate the Hex file.
4. Load the Hex file in MPLAB IPE and program the PIC microcontroller.
5. Power ON the board and verify that the RTC time is displayed on LCD/Serial as programmed.
6. Ensure the programmer's arrow is aligned towards the RST pin on the ICSP connector.

Program:

```
#include <xc.h>
#include <stdio.h>
//================ CONFIGURATION ================
#pragma config OSC  = HS
#pragma config WDT  = OFF
#pragma config LVP  = OFF
#pragma config PWRT = ON
#define _XTAL_FREQ 20000000

//================ PIN DEFINITIONS ================
#define RS PORTDbits.RD0
#define EN PORTDbits.RD1
#define LCD_DATA_PORT PORTD
#define RTC_ADDR 0x68
//================ I2C FUNCTIONS ================
void I2C_Init(void) {
    TRISC3 = 1; TRISC4 = 1;
    SSPCON1 = 0x28;
    SSPADD  = 49;
    SSPSTAT = 0x00;
}
void I2C_Wait(void) {
    while ((SSPCON2 & 0x1F) || (SSPSTAT & 0x04));
}
void I2C_Start(void) { I2C_Wait(); SEN = 1; }
void I2C_Stop(void) { I2C_Wait(); PEN = 1; }
void I2C_Write(unsigned char data) { I2C_Wait(); SSPBUF = data; }

unsigned char I2C_Read(unsigned char ack) {
    unsigned char data;
    I2C_Wait(); RCEN = 1;
    I2C_Wait(); data = SSPBUF;
    I2C_Wait(); ACKDT = (unsigned char)(ack ? 0 : 1);
    ACKEN = 1;
    return data;
}
//================ IMPROVED LCD FUNCTIONS ================
void LCD_Strobe(void) {
    EN = 1;
    __delay_us(5);  // Increased for stability
    EN = 0;
    __delay_us(100); // Increased for stability
}
```

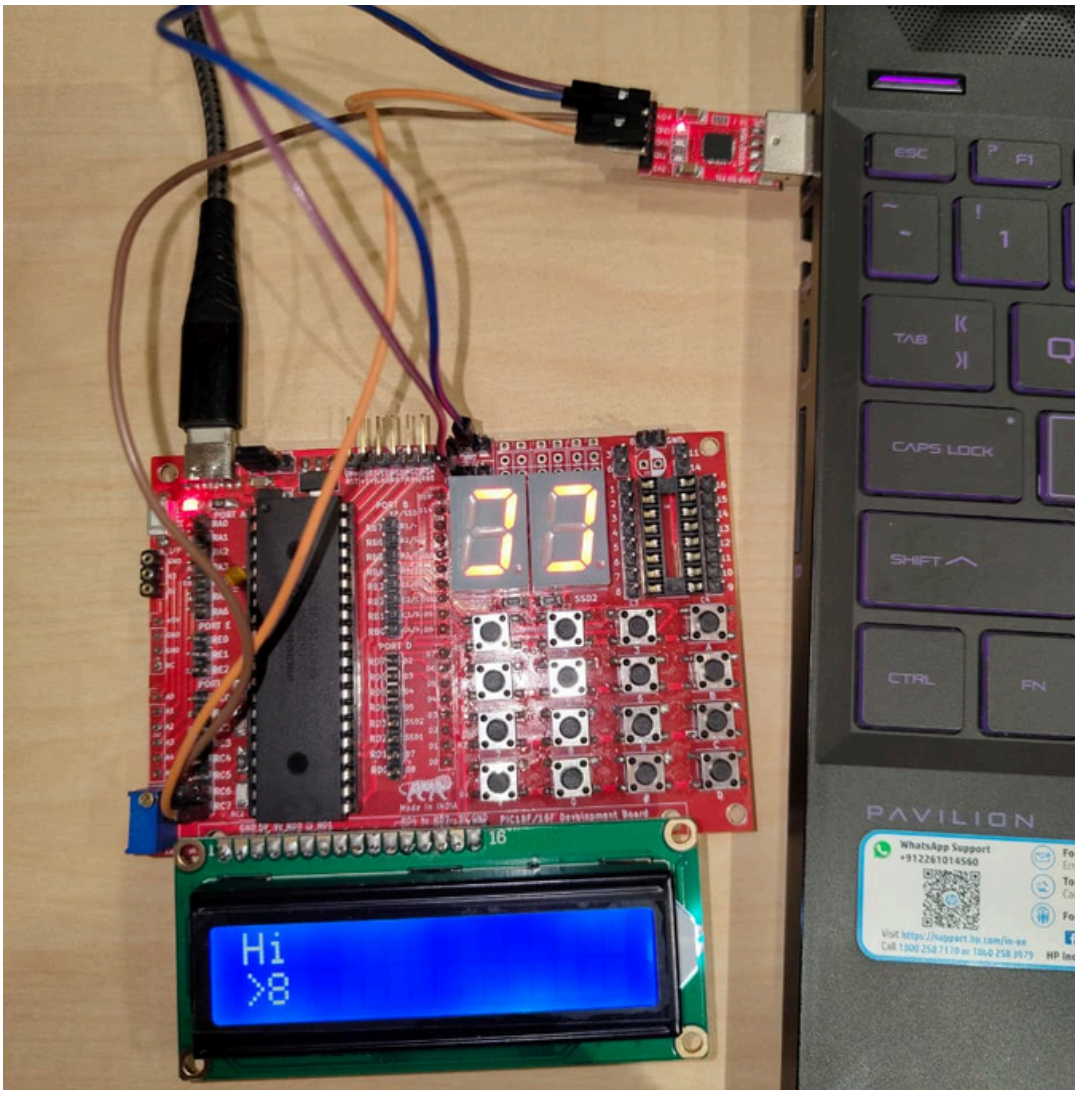
```
void LCD_Write(unsigned char value, int is_data) {
    RS = (unsigned char)(is_data ? 1 : 0);
    LCD_DATA_PORT = (unsigned char)((LCD_DATA_PORT & 0x0F) | (value & 0xF0));
    LCD_Strobe();
    LCD_DATA_PORT = (unsigned char)((LCD_DATA_PORT & 0x0F) | ((value << 4) & 0xF0));
    LCD_Strobe();

    __delay_ms(2);
}
void LCD_Init(void) {
    TRISD = 0x00;
    PORTD = 0x00;
    __delay_ms(100); // Wait for LCD power to stabilize
    RS = 0;
    // Force Reset Sequence for 4-bit mode
    LCD_DATA_PORT = 0x30; LCD_Strobe(); __delay_ms(10);
    LCD_DATA_PORT = 0x30; LCD_Strobe(); __delay_ms(1);
    LCD_DATA_PORT = 0x30; LCD_Strobe(); __delay_ms(1);
    LCD_DATA_PORT = 0x20; LCD_Strobe(); // Switch to 4-bit mode now
    __delay_ms(1);

    LCD_Write(0x28, 0); // 4-bit, 2 lines, 5x8
    LCD_Write(0x0C, 0); // Display ON
    LCD_Write(0x06, 0); // Entry mode
    LCD_Write(0x01, 0); // Clear screen
    __delay_ms(5);
}
void LCD_Print(char *str) {
    while(*str) LCD_Write((unsigned char)(*str++), 1);
}
//================ RTC FUNCTIONS ================
unsigned char BCD_to_DEC(unsigned char val) {
    return (unsigned char)((val >> 4) * 10 + (val & 0x0F));
}
unsigned char RTC_Read(unsigned char reg) {
    unsigned char data;
    I2C_Start();
    I2C_Write((unsigned char)(RTC_ADDR << 1));
    I2C_Write(reg);
    I2C_Wait(); RSEN = 1; // Repeated Start
    I2C_Write((unsigned char)((RTC_ADDR << 1) | 1));
    data = I2C_Read(0);
    I2C_Stop();
    return data;
}
//================ MAIN ============================
void main(void) {
    char buf[20];
    unsigned char h, m, s, hl2;
    char *period;
    ADCON1 = 0x07;
    I2C_Init();
    LCD_Init();
```

```
    while (1) {
        s = BCD_to_DEC(RTC_Read(0x00));
        m = BCD_to_DEC(RTC_Read(0x01));
        h = BCD_to_DEC(RTC_Read(0x02) & 0x3F);

        if (h >= 12) {
            period = "PM";
            h12 = (unsigned char)((h > 12) ? (h - 12) : 12);
        } else {
            period = "AM";
            h12 = (unsigned char)((h == 0) ? 12 : h);
        }
        sprintf(buf, "%02u:%02u:%02u %s", h12, m, s, period);
        LCD_Write(0x80, 0); // Row 1
        LCD_Print("  Digital Clock ");
        LCD_Write(0xC0, 0); // Row 2
        LCD_Print(buf);
        _delay_ms(500);
    }
}
```

Lab 9 – UART Interfacing with Serial Monitor (Putty)

Aim:

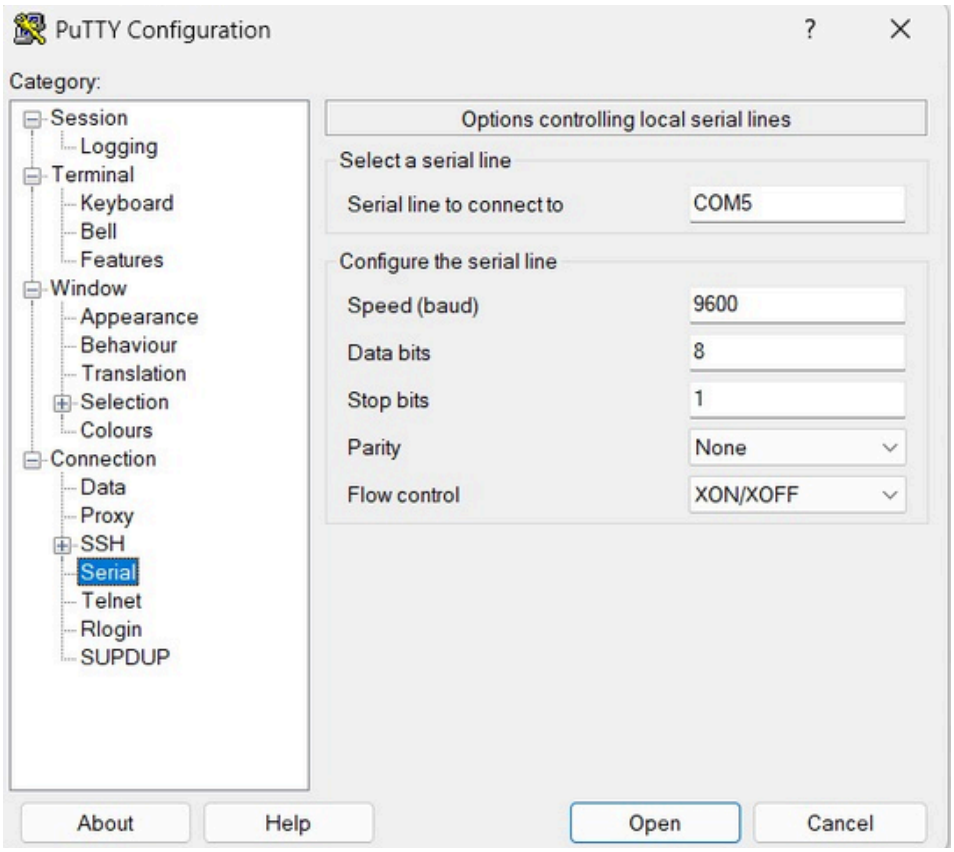
To study and implement UART (Universal Asynchronous Receiver/Transmitter) serial communication between a PIC microcontroller and a PC, and to observe and exchange data using a serial terminal program (Terminal).



Procedure:

1. Connect the PIC development board's UART TX and RX pins (RC6 → RX of USB-serial converter, RC7 ← TX of USB-serial converter) with a USB-to-Serial converter to the PC and common ground.
2. Read the UART program with comments to understand baud rate setup and transmit/receive routines.
3. Open or create the project in MPLAB X IDE, build and generate the Hex file.
4. Use MPLAB IPE to download the Hex file to the PIC microcontroller.
5. Open PuTTY serial terminal software (download link: https://apps.microsoft.com/detail/XPFNZKSKLBP7RJ?hl=en IN&gl=IN&ocid=pdpshare) and configure the COM port and baud rate.
6. Power ON the PIC board and observe data sent from PIC on the Terminal; send data from Terminal to PIC as per program logic.

PuTTY configuration for serial communication



PuTTY Serial monitor to receive data transmitted by the PIC18F / 16F

Procedure:

- When any key is pressed on the keypad, the PIC18F board sends the character to the serial monitor.
- When a character is typed in the serial monitor, the PIC18F receives it and displays it on the LCD.

Program:

```
#include <xc.h>
#include <stdio.h>

#pragma config OSC = HS, WDT = OFF, LVP = OFF, PWRT = ON
#define _XTAL_FREQ 20000000

// --- Hardware Pins (Amotech Schematic) ---
#define RS LATDbits.LATD0
#define EN LATDbits.LATD1
#define LCD_DATA LATD

#define C1 LATBbits.LATB0
#define C2 LATBbits.LATB1
#define C3 LATBbits.LATB2
#define C4 LATBbits.LATB3
#define R1 PORTBbits.RB4
#define R2 PORTBbits.RB5
#define R3 PORTBbits.RB6
#define R4 PORTBbits.RB7

// --- Prototypes ---
void UART_Init(long baud);
void UART_Write(char data);
void LCD_Init(void);
void LCD_Command(unsigned char cmd);
void LCD_Char(unsigned char dat);
char GetKey(void);

// --- UART Functions ---
void UART_Init(long baud) {
    TRISC6 = 0; TRISC7 = 1;
    SPBRG = (unsigned char)((_XTAL_FREQ / (64 * baud)) - 1);
    TXSTA = 0x20; RCSTA = 0x90;
}

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

// --- LCD Functions ---
void LCD_Command(unsigned char cmd) {
    LCD_DATA = (unsigned char)((LCD_DATA & 0x0F) | (cmd & 0xF0));
    RS = 0; EN = 1; __delay_us(10); EN = 0;
    LCD_DATA = (unsigned char)((LCD_DATA & 0x0F) | (cmd << 4));
    EN = 1; __delay_us(10); EN = 0;
    __delay_ms(2);
}
```

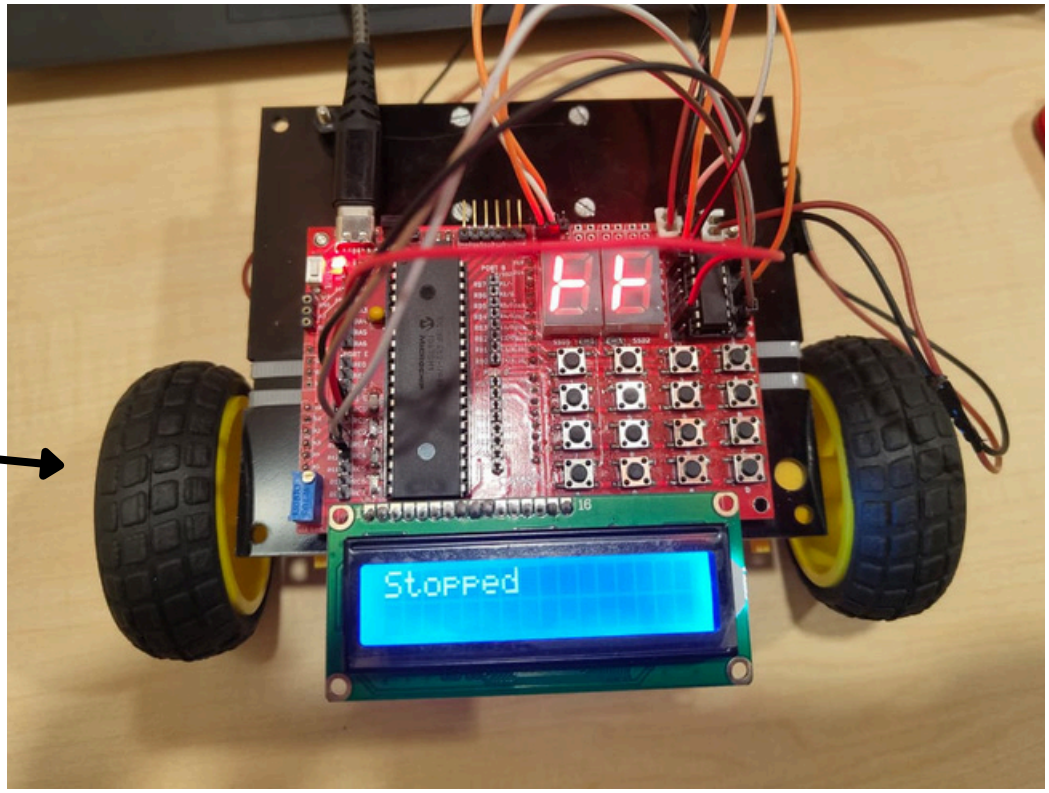
```
void LCD_Char(unsigned char dat) {
    LCD_DATA = (unsigned char)((LCD_DATA & 0x0F) | (dat & 0xF0));
    RS = 1; EN = 1; __delay_us(10); EN = 0;
    LCD_DATA = (unsigned char)((LCD_DATA & 0x0F) | (dat << 4));
    EN = 1; __delay_us(10); EN = 0;
    __delay_ms(2);
}

void LCD_Init(void) {
    TRISD = 0x00;
    __delay_ms(50);
    LCD_Command(0x02); LCD_Command(0x28);
    LCD_Command(0x0C); LCD_Command(0x01);
    LCD_Command(0x06); // Ensure cursor moves right automatically
}

// --- Non-Blocking Keypad Scan ---
char GetKey(void) {
    C1=0; C2=1; C3=1; C4=1;
    if(!R1){ while(!R1); return '1'; } if(!R2){ while(!R2); return '4'; }
    if(!R3){ while(!R3); return '7'; } if(!R4){ while(!R4); return 'C'; }
    C2=0; C1=1; C3=1; C4=1;
    if(!R1){ while(!R1); return '2'; } if(!R2){ while(!R2); return '5'; }
    if(!R3){ while(!R3); return '8'; } if(!R4){ while(!R4); return '0'; }
    C3=0; C1=1; C2=1; C4=1;
    if(!R1){ while(!R1); return '3'; } if(!R2){ while(!R2); return '6'; }
    if(!R3){ while(!R3); return '9'; } if(!R4){ while(!R4); return '='; }
    C4=0; C1=1; C2=1; C3=1;
    if(!R1){ while(!R1); return '+'; } if(!R2){ while(!R2); return '-'; }
    if(!R3){ while(!R3); return '*'; } if(!R4){ while(!R4); return '/'; }
    return 0;
}

void main(void) {
    ADCON1 = 0x07;
    INTCON2bits.RBPU = 0;
    TRISB = 0xF0;

    UART_Init(9600);
    LCD_Init();

    LCD_Command(0x80); // Start at Line 1
```

```
    while(1) {
        // --- 1. RECEIVE: Terminal to LCD ---
        if(PIR1bits.RCIF) {
            char rxData = RCREG;
            LCD_Char(rxData); // Just print the character
        }

        // --- 2. SEND: Keypad to Terminal ---
        char key = GetKey();
        if(key != 0) {
            if(key == 'C') {
                LCD_Command(0x01); // Press 'C' to clear screen
                LCD_Command(0x80);
            } else {
                UART_Write(key);
                // Show sent key on Line 2
                LCD_Command(0xC0);
                LCD_Char('>');
                LCD_Char(key);
            }
        }
    }
}
```

Lab 10 – Interfacing of Two DC Motors using L293D Driver IC

Aim:

To study and implement the interfacing and bidirectional control of two DC motors using the L293D motor driver IC with a PIC18F/PIC16F microcontroller.



Procedure:

1. Connect Motor-A to OUT1 & OUT2 and Motor-B to OUT3 & OUT4 of L293D.
2. Connect PIC output pins to L293D inputs as:
   RC0 → IN1 (Motor-A direction)
   RC1 → IN2 (Motor-A direction)
   RC2 → IN3 (Motor-B direction)
   RC3 → IN4 (Motor-B direction)
3. Connect +5V to L293D Vcc1, and motor supply voltage (e.g., +6V to +12V) to Vcc2 based on motor rating.
4. Connect GND of PIC board and motor supply to L293D GND pins (all grounds common).
5. Read the provided program to understand direction control logic (IN1/IN2 & IN3/IN4).
6. Open or create the project in MPLAB X IDE, build and generate the Hex file.
7. Load the Hex file into MPLAB IPE and program the PIC microcontroller.
8. Power ON the board and verify forward, reverse, and stop operations of both motors as per program.
9. Ensure the programmer's arrow aligns with the RST pin of the ICSP connector during programming.

Program:

```
#include <xc.h>
#include <pic18f452.h>

#pragma config OSC = HS, WDT = OFF, LVP = OFF, BOR = OFF
#define _XTAL_FREQ 20000000UL

// ================= LCD FUNCTIONS (PORTD) =================
void LCD_Command(unsigned char cmd)
{
    LATDbits.LATD0 = 0; // RS = 0
    LATDbits.LATD1 = 1; // EN = 1

    LATD = (LATD & 0x03) | (cmd & 0xF0); // High nibble on RD2-RD5
    __delay_ms(2);
    LATDbits.LATD1 = 0;

    __delay_ms(2);
    LATDbits.LATD1 = 1;
    LATD = (LATD & 0x03) | ((cmd << 4) & 0xF0); // Low nibble
    __delay_ms(2);
    LATDbits.LATD1 = 0;
}

void LCD_Char(unsigned char data)
{
    LATDbits.LATD0 = 1; // RS = 1
    LATDbits.LATD1 = 1; // EN = 1

    LATD = (LATD & 0x03) | (data & 0xF0);
    __delay_ms(2);
    LATDbits.LATD1 = 0;

    __delay_ms(2);
    LATDbits.LATD1 = 1;
    LATD = (LATD & 0x03) | ((data << 4) & 0xF0);
    __delay_ms(2);
    LATDbits.LATD1 = 0;
}

void LCD_Init()
{
    TRISD = 0x00;  // Entire PORTD as OUTPUT for LCD
    __delay_ms(20);

    LCD_Command(0x02); // 4-bit mode
    LCD_Command(0x28); // 2 lines, 5x7
    LCD_Command(0x0C); // Display ON, Cursor OFF
    LCD_Command(0x06); // Auto increment
```

```
    LCD_Command(0x01); // Clear
}

void LCD_String(char *str)
{
    while(*str) LCD_Char(*str++);
}

void LCD_Clear()
{
    LCD_Command(0x01);
}

// ================= MOTOR FUNCTIONS (PORTC) =================
void Motor_Forward()
{
    LATC0 = 1; LATC1 = 0;
    LATC2 = 1; LATC3 = 0;
}

void Motor_Back()
{
    LATC0 = 0; LATC1 = 1;
    LATC2 = 0; LATC3 = 1;
}

void Motor_Left()
{
    LATC0 = 0; LATC1 = 0;
    LATC2 = 1; LATC3 = 0;
}
void Motor_Right()
{
    LATC0 = 1; LATC1 = 0;
    LATC2 = 0; LATC3 = 0;
}

void Motor_Stop()
{
    LATC = 0x00;
}

// ================= KEYPAD FUNCTION (4x4 on PORTB) =================
char Keypad_Read()
{
    INTCON2bits.RBPU = 0; // Enable pull-ups
    TRISB = 0x0F;         // RB0-RB3 Input (COL), RB4-RB7 Output (ROW)

    // Row1 (RB4 low)
    LATB = 0b11101111;
    __delay_ms(3);
    if(!RB0 || !RB1 || !RB2 || !RB3) return '1';
```

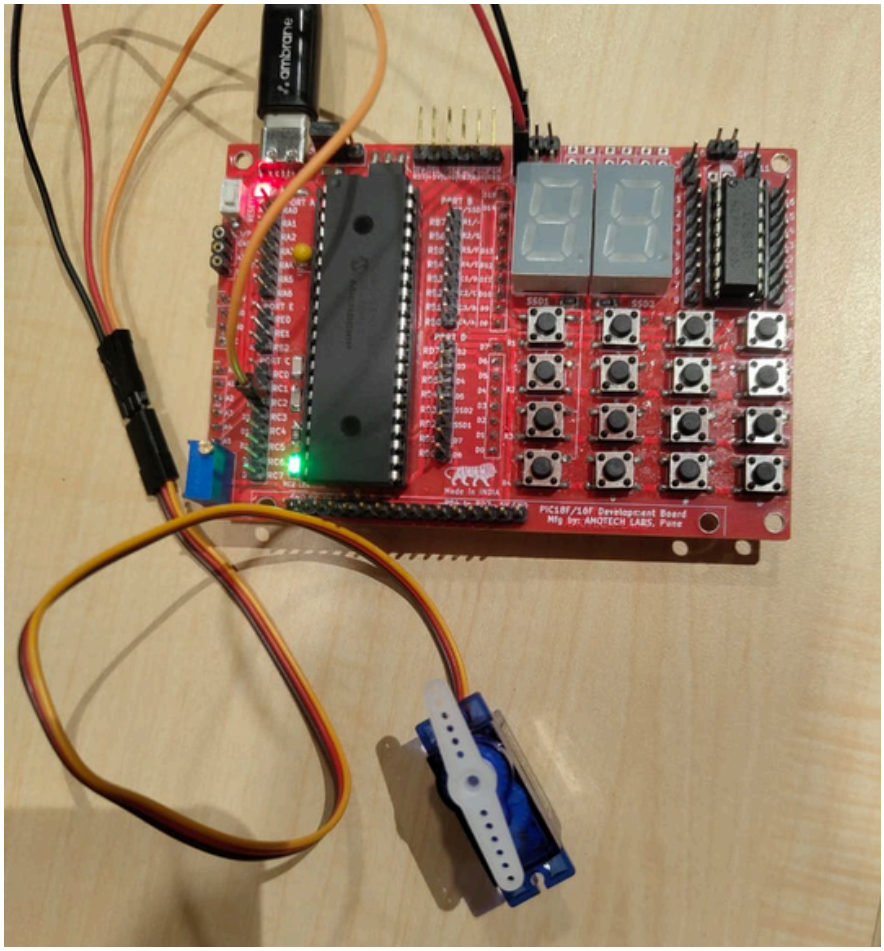
```
    // Row2 (RB5 low)
    LATB = 0b11011111;
    __delay_ms(3);
    if(!RB0 || !RB1 || !RB2 || !RB3) return '2';

    // Row3 (RB6 low)
    LATB = 0b10111111;
    __delay_ms(3);
    if(!RB0 || !RB1 || !RB2 || !RB3) return '3';

    // Row4 (RB7 low)
    LATB = 0b01111111;
    __delay_ms(3);
    if(!RB0 || !RB1 || !RB2 || !RB3) return '4';

    return 0;
}

// ================ MAIN PROGRAM ================
void main()
{
    TRISC = 0x00;    // PORTC output for motors
    LATC = 0x00;     // Motors off initially

    TRISB = 0x0F;    // Keypad RB0-RB3 input
    INTCON2bits.RBPU = 0; // enable internal pullups

    LCD_Init();
    LCD_Clear();
    LCD_String("READY!");
    while(1)
    {
        char key = Keypad_Read();

        switch(key)
        {
            case '1':
                Motor_Forward();
                LCD_Clear();
                LCD_String("Moving Forward");
                break;

            case '2':
                Motor_Back();
                LCD_Clear();
                LCD_String("Moving Backward");
                break;

            case '3':
                Motor_Left();
                LCD_Clear();
                LCD_String("Turning Left");
                break;
```

```
            case '4':
                Motor_Right();
                LCD_Clear();
                LCD_String("Turning Right");
                break;

            default:
                Motor_Stop();
                LCD_Clear();
                LCD_String("Stopped");
                break;
        }

        __delay_ms(150); // debounce
    }
}
```

Lab 11 – Interfacing of Servo Motor

Aim:

To study and implement the interfacing of a servo motor with a PIC18F/PIC16F microcontroller using compare mode control to rotate the shaft to desired angular positions.



Procedure:

1. Connect the servo motor's signal pin to a capable pin of the PIC (e.g., RC2/CCP1), VCC to +5V, and GND to common ground.
2. Read the program to understand CPP configuration and angle control using duty cycle variations.
3. Open or create the project in MPLAB X IDE, build the code, and generate the Hex file.
4. Use MPLAB IPE to program the PIC microcontroller with the generated Hex file.
5. Power ON the development board and observe the servo rotating to different angles as per the program (e.g., 0°, 90°, 180°).
6. Ensure the programmer's arrow is aligned towards the RST pin on the ICSP connector during programming.

Program:

```
#include <xc.h>
#include <pic18f452.h>

// ================== CONFIG BITS ====================
#pragma config OSC = HS, OSCS = OFF
#pragma config PWRT = OFF, BOR = OFF, BORV = 2
#pragma config WDT = OFF, WDTPS = 128
#pragma config STVR = ON, LVP = OFF
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF
#pragma config CPB = OFF, CPD = OFF
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF
#pragma config WRTC = OFF, WRTB = OFF, WRTD = OFF
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF
#pragma config EBTRB = OFF

#define _XTAL_FREQ 20000000UL

// ========================================================
//        SERVO TIMING CALCULATIONS (VERY IMPORTANT)
// ========================================================
//
// CPU Frequency (Fosc) = 20 MHz
// Instruction Clock = Fosc / 4 = 20MHz / 4 = 5 MHz
// ---> Instruction Time = 1 / 5MHz = 0.2 µs per tick
//
// Timer1 Prescaler = 1:1 (chosen in T1CON)
// ---> Timer1 Tick Time = 0.2 µs
//
// Standard analog servo frequency = 50Hz
// ---> Period = 1 / 50Hz = 0.02s = 20,000 µs (20ms)
// ---> Required Timer1 ticks for 20ms:
//      20000µs / 0.2µs = 100000 ticks
//
// Pulse widths for position:
// - 1.0ms (1000µs)  = 1000 / 0.2 =  5000 ticks
// - 1.5ms (1500µs)  = 1500 / 0.2 =  7500 ticks
// - 2.0ms (2000µs)  = 2000 / 0.2 = 10000 ticks
//
// So:
// FRAME_TICKS = 100000 (fixed for 50Hz)
// pulse_ticks = 5000 to 10000 depending on position
//
// CCP1 Compare mode schedules two events:
// 1) Set RC2 HIGH at start of frame
// 2) Set RC2 LOW after pulse_ticks
// ========================================================
```

```
#define FRAME_TICKS 100000UL   // 20ms = 20000µs / 0.2µs = 100000 ticks

volatile unsigned int pulse_ticks = 7500; // default 1500us = 90°
volatile unsigned char phase = 0;         // 0=start pulse, 1=end pulse

// ======================================================
//   INTERRUPT SERVICE (CCP1 compare event)
// ======================================================
void __interrupt() isr(void)
{
    if (PIR1bits.CCP1IF)  // CCP1 compare event happened?
    {
        PIR1bits.CCP1IF = 0;  // clear flag

        if (phase == 0)
        {
            // ============================
            //       START OF PULSE
            // ============================
            PORTCbits.RC2 = 1;    // drive servo signal HIGH

            // schedule pulse end time
            // (current time + pulse_ticks)
            CCPR1 = pulse_ticks;

            phase = 1;
        }
        else
        {
            // ============================
            //        END OF PULSE
            // ============================
            PORTCbits.RC2 = 0;    // drive servo signal LOW

            // schedule start of next 20ms frame:
            // (current time + (FRAME_TICKS - pulse_ticks))
            CCPR1 += (FRAME_TICKS - pulse_ticks);

            phase = 0;
        }
    }
}
```

```
// ======================================================
//                   TIMER1 INIT
// ======================================================
void Timer1_Init(void)
{
    // T1CON bits:
    // T1CKPS1:T1CKPS0 = 00 --> Prescaler = 1:1
    // TMR1CS = 0        --> Internal clock (Fosc/4)
    // TMR1ON = 1        --> Enable Timer1
    //
    // Result: Timer1 tick = 0.2µs
    T1CON = 0b00000001;  // Prescaler 1:1, internal clock, Timer1 ON
}

// ======================================================
//                   CCP1 INIT (COMPARE MODE)
// ======================================================
void CCP1_Init(void)
{
    // CCP1CON bits:
    // 0b00001011 = Compare mode, trigger special event
    CCP1CON = 0b00001011;

    // First event after complete frame:
    // CCPR1 = 100000 ticks (20ms)
    CCPR1 = FRAME_TICKS;

    PIR1bits.CCP1IF = 0;  // clear flag
    PIE1bits.CCP1IE = 1;  // enable CCP1 interrupt
}

// ======================================================
//                       MAIN
// ======================================================
void main(void)
{
    TRISCbits.TRISC2 = 0;  // RC2 as output for servo
    PORTCbits.RC2 = 0;     // start LOW

    Timer1_Init();
    CCP1_Init();

    INTCONbits.PEIE = 1;   // enable peripheral interrupts
    INTCONbits.GIE = 1;    // global interrupt enable
```

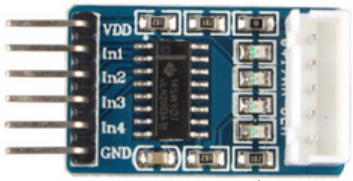
```
    while (1)
    {
        // Test sweep:
        //
        // LEFT = 1.0ms pulse = 5000 ticks
        // CENTER = 1.5ms pulse = 7500 ticks
        // RIGHT = 2.0ms pulse = 10000 ticks

        pulse_ticks = 5000;    // 1000µs
        __delay_ms(1000);

        pulse_ticks = 7500;    // 1500µs
        __delay_ms(1000);

        pulse_ticks = 10000;   // 2000µs
        __delay_ms(1000);
    }
}
```

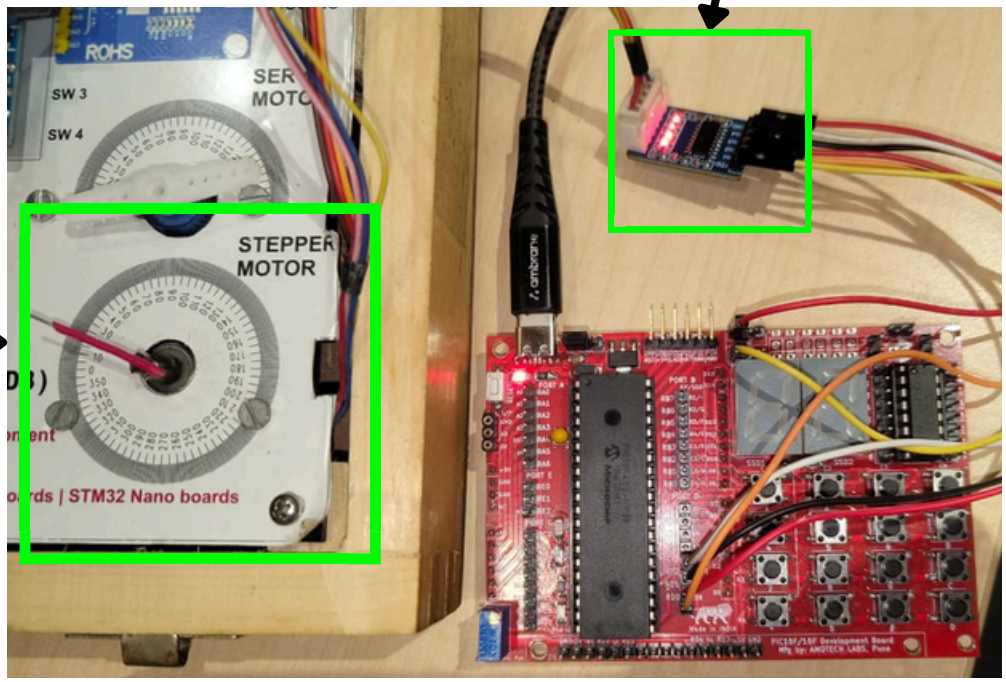
Lab 12 – Interfacing of Stepper Motor

Aim:

To study and implement the interfacing and step-wise rotational control of a stepper motor using a PIC18F/PIC16F microcontroller through a suitable driver circuit (e.g., ULN2003/L293D).

ULN2003A Driver Module
Stepper Motor Driver





28BYJ-48 Stepper Motor DC 5V
Used Inside the Box

Procedure:

1. Connect the stepper motor coils to the driver IC (ULN2003) as per the pin sequence (A–B–C–D).
2. Connect PIC output pins (e.g., RC0–RC3) to the driver input lines to energize the coils in sequence.
3. Connect motor supply voltage to the driver IC Vcc, and connect GND of PIC and driver common.
4. Read the program provided to understand the step sequence (Full-step / Half-step) and delay control.
5. Open or create the project in MPLAB X IDE, build and generate the Hex file.
6. Load the Hex file into MPLAB IPE and program the PIC microcontroller.
7. Power ON the board and observe the stepper motor rotating in clockwise or counter-clockwise direction based on coil energizing sequence.
8. Ensure the programmer's arrow aligns to the RST pin of the ICSP header during programming.

Program:

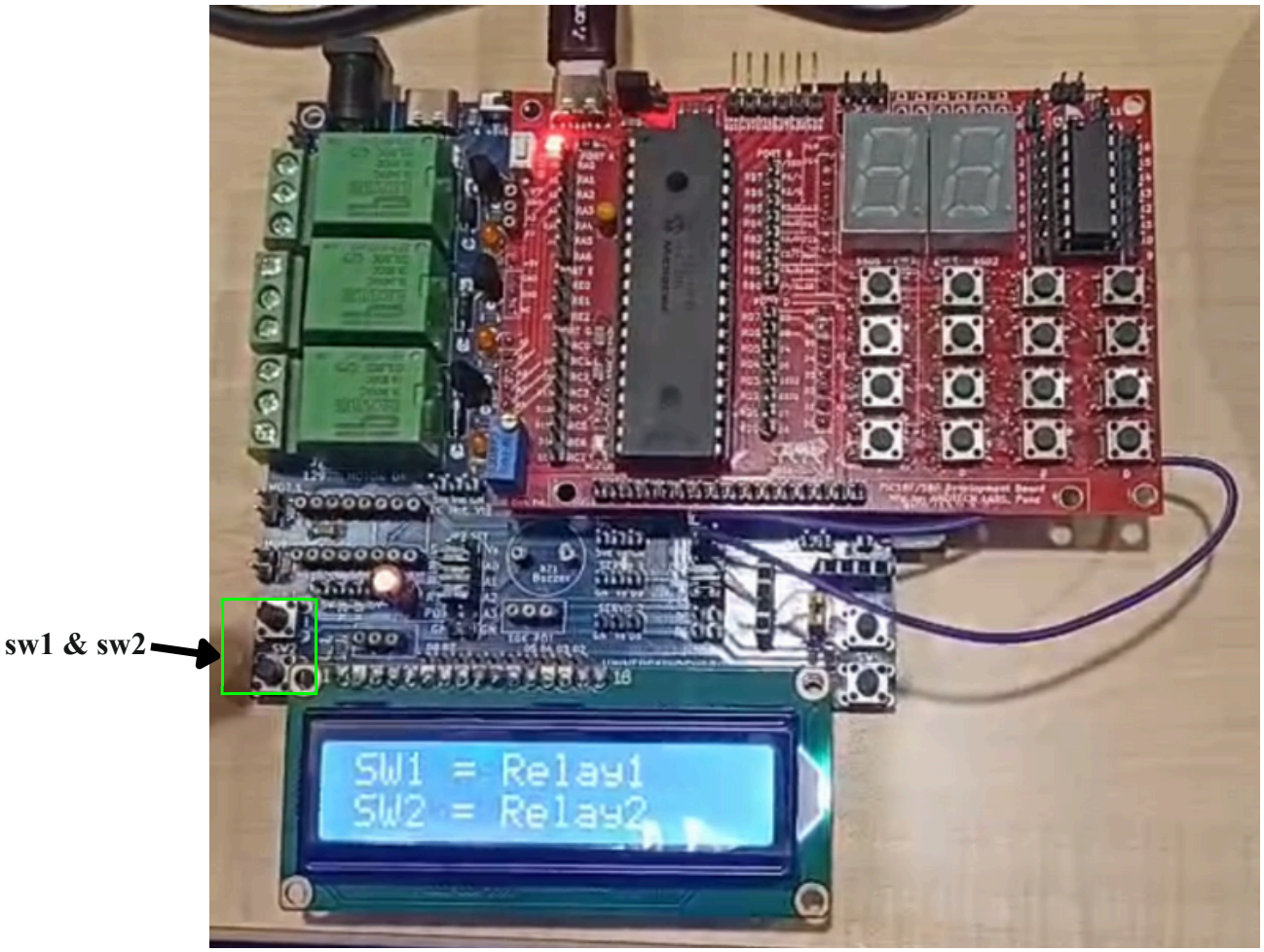
```
#include <xc.h>
#include <pic18f452.h>

// ================== CONFIG BITS ====================
#pragma config OSC = HS, OSCS = OFF
#pragma config PWRT = OFF, BOR = OFF, BORV = 2
#pragma config WDT = OFF, WDTPS = 128
#pragma config STVR = ON, LVP = OFF
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF
#pragma config CPB = OFF, CPD = OFF
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF
#pragma config WRTC = OFF, WRTB = OFF, WRTD = OFF
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF
#pragma config EBTRB = OFF

// ================== CONSTANTS ====================
// For 28BYJ-48 stepper motor:
// 1 revolution = 4096 half-steps = 512 eight-step sequences
#define FULL_ROTATION_STEPS 512

void MSdelay(unsigned int val)
{
    unsigned int i, j;
    for(i = 0; i < val; i++)
        for(j = 0; j < 165; j++);
}

void Stepper_Clockwise(int period)
{
    for(int i = 0; i < FULL_ROTATION_STEPS; i++)
    {
        LATD = 0x09; MSdelay(period); // 1001
        LATD = 0x08; MSdelay(period); // 1000
        LATD = 0x0C; MSdelay(period); // 1100
        LATD = 0x04; MSdelay(period); // 0100
        LATD = 0x06; MSdelay(period); // 0110
        LATD = 0x02; MSdelay(period); // 0010
        LATD = 0x03; MSdelay(period); // 0011
        LATD = 0x01; MSdelay(period); // 0001
    }
}
```

```
void Stepper_Anticlockwise(int period)
{
    for(int i = 0; i < FULL_ROTATION_STEPS; i++)
    {
        LATD = 0x01; MSdelay(period); // 0001
        LATD = 0x03; MSdelay(period); // 0011
        LATD = 0x02; MSdelay(period); // 0010
        LATD = 0x06; MSdelay(period); // 0110
        LATD = 0x04; MSdelay(period); // 0100
        LATD = 0x0C; MSdelay(period); // 1100
        LATD = 0x08; MSdelay(period); // 1000
        LATD = 0x09; MSdelay(period); // 1001
    }
}

void main(void)
{
    int period = 5;          // smaller = faster rotation
    TRISD = 0x00;            // Make PORTD outputs
    LATD = 0x00;             // Initialize outputs off

    while(1)
    {
        // Rotate 360° Clockwise
        Stepper_Clockwise(period);
        MSdelay(1000);

        // Rotate 360° Anticlockwise
        Stepper_Anticlockwise(period);
        MSdelay(1000);
    }
}
```

Lab 13 – Interfacing of Relays using UDB

Aim:

To study and implement relay control using a PIC18F/PIC16F microcontroller, where pressing SW1 turns ON Relay 1 and pressing SW2 turns ON Relay 2 through a relay driver interface.



Procedure:

1. Connect SW1 and SW2 to UDB of D11 &D10 and connect Relay 1 and Relay 2 to A0 & A1.
2. Connect the UDB relay driver board to the PIC.
3. Read the relay control program which assigns SW1 → Relay 1 ON and SW2 → Relay 2 ON.
4. Open or create the project in MPLAB X IDE, build the code, and generate the Hex file.
5. Open MPLAB IPE, load the Hex file, and program the PIC microcontroller.
6. Power ON the system and verify: pressing SW1 turns Relay 1 ON and pressing SW2 turns Relay 2 ON.
7. Ensure the programmer's arrow is aligned towards the RST pin on the ICSP connector during programming.

Program:

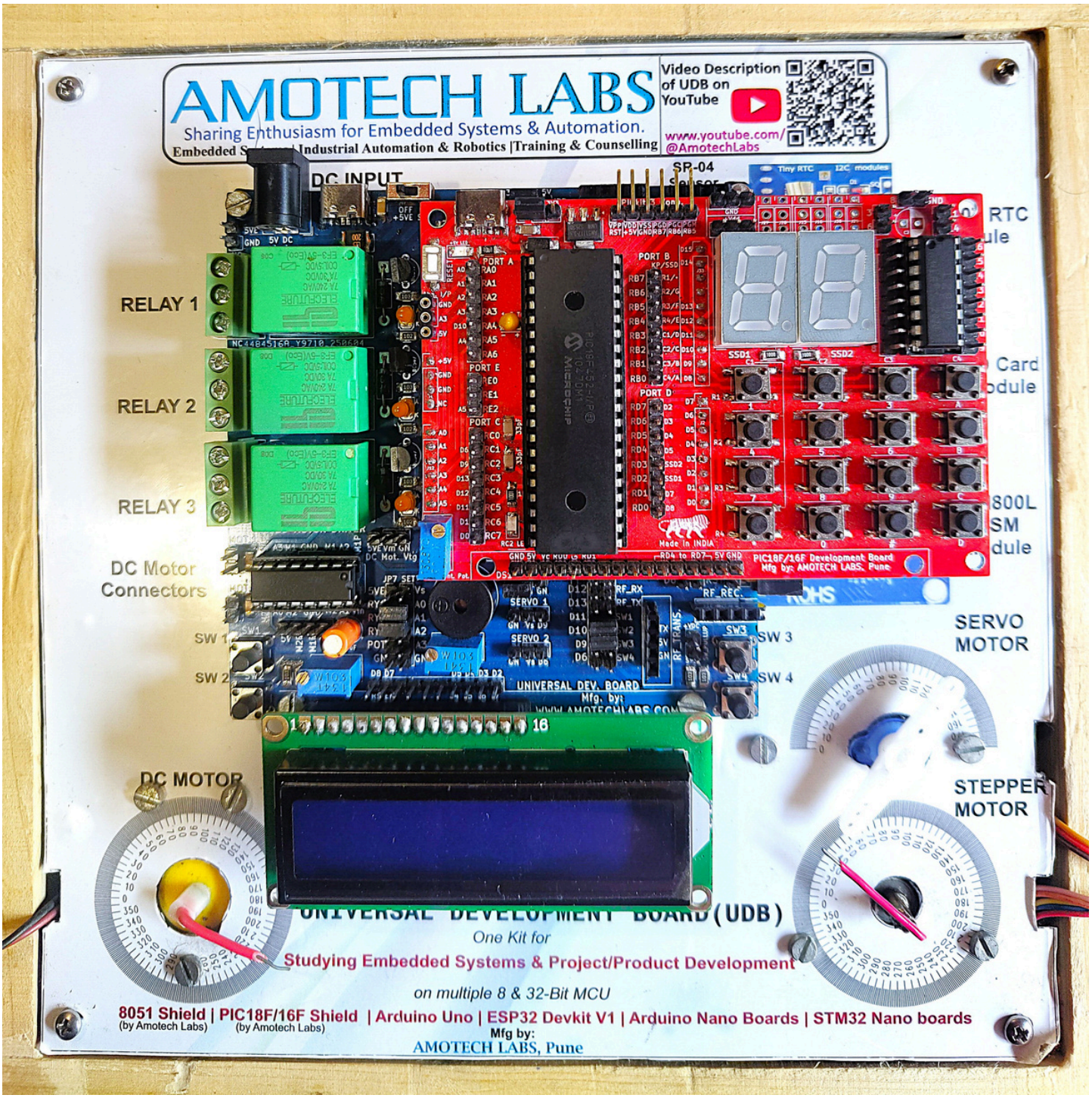
```
#include <xc.h>
#define _XTAL_FREQ 20000000

#pragma config OSC = HS, WDT = OFF, LVP = OFF, DEBUG = OFF, PWRT = ON, BOR = OFF

// LCD pins
#define LCD_RS LATDbits.LATD0
#define LCD_EN LATDbits.LATD1
#define LCD_D4 LATDbits.LATD4
#define LCD_D5 LATDbits.LATD5
#define LCD_D6 LATDbits.LATD6
#define LCD_D7 LATDbits.LATD7

void LCD_Init(void);
void LCD_Cmd(unsigned char cmd);
void LCD_Char(char data);
void LCD_String(const char *txt);
void LCD_Clear(void);
void LCD_Goto(unsigned char r, unsigned char c);
void ShowModes(void);

void main(void)
{
    // Inputs
    TRISCbits.TRISC5 = 1;    // SW1
    TRISAbits.TRISA4 = 1;    // SW2

    // Outputs
    TRISAbits.TRISA0 = 0;    // Relay1
    TRISAbits.TRISA1 = 0;    // Relay2

    LATAbits.LATA0 = 0;
    LATAbits.LATA1 = 0;

    // LCD port
    TRISD = 0x00;

    // Digital mode
    ADCON1 = 0x0F;
```

```
    // 1) SPLASH SCREEN
    LCD_Init();
    LCD_Clear();
    LCD_Goto(1,1); LCD_String("PIC Dev Board");
    LCD_Goto(2,1); LCD_String("With UDB");
    __delay_ms(1500);

    // 2) SHOW MODES
    ShowModes();

    while(1)
    {
        // MODE 1 → Relay1
        if(PORTCbits.RC5 == 0) {
            LCD_Clear();
            LCD_String("Relay1 ON");
            LATAbits.LATA0 = 1;
            while(PORTCbits.RC5 == 0); // wait release
            LATAbits.LATA0 = 0;
            ShowModes();
        }

        // MODE 2 → Relay2
        if(PORTAbits.RA4 == 0) {
            LCD_Clear();
            LCD_String("Relay2 ON");
            LATAbits.LATA1 = 1;
            while(PORTAbits.RA4 == 0); // wait release
            LATAbits.LATA1 = 0;
            ShowModes();
        }
    }
}
void LCD_Enable() {
    LCD_EN = 1; __delay_us(10);
    LCD_EN = 0; __delay_us(10);
}

void LCD_Cmd(unsigned char cmd)
{
    LCD_RS = 0;
    LCD_D4 = (cmd >> 4) & 1;
    LCD_D5 = (cmd >> 5) & 1;
    LCD_D6 = (cmd >> 6) & 1;
    LCD_D7 = (cmd >> 7) & 1;
    LCD_Enable();
```

```
    LCD_D4 = cmd & 1;
    LCD_D5 = (cmd >> 1) & 1;
    LCD_D6 = (cmd >> 2) & 1;
    LCD_D7 = (cmd >> 3) & 1;
    LCD_Enable();
}

void LCD_Char(char data)
{
    LCD_RS = 1;
    LCD_D4 = (data >> 4) & 1;
    LCD_D5 = (data >> 5) & 1;
    LCD_D6 = (data >> 6) & 1;
    LCD_D7 = (data >> 7) & 1;
    LCD_Enable();

    LCD_D4 = data & 1;
    LCD_D5 = (data >> 1) & 1;
    LCD_D6 = (data >> 2) & 1;
    LCD_D7 = (data >> 3) & 1;
    LCD_Enable();
}

void LCD_String(const char *txt)
{
    while(*txt) LCD_Char(*txt++);
}

void LCD_Init()
{
    LCD_RS = 0; LCD_EN = 0;
    __delay_ms(20);

    LCD_Cmd(0x02);
    LCD_Cmd(0x28);
    LCD_Cmd(0x0C);
    LCD_Cmd(0x06);
    LCD_Cmd(0x01);
    __delay_ms(2);
}

void LCD_Clear()
{
    LCD_Cmd(0x01);
    __delay_ms(2);
}

void LCD_Goto(unsigned char r, unsigned char c)
{
    if(r == 1) LCD_Cmd(0x80 + (c - 1));
```

```
        else        LCD_Cmd(0xC0 + (c - 1));
    }

    void ShowModes(void)
    {
        LCD_Clear();
        LCD_Goto(1,1); LCD_String("SW1 = Relay1");
        LCD_Goto(2,1); LCD_String("SW2 = Relay2");
    }
```

‘PIC18F/16F Development Board’ mounted on ‘Universal Development Board’



Description:

This setup shows a PIC18F/16F Development Board mounted on a Universal Development Board (UDB) inside a wooden enclosure. The kit is fitted on an Acrylic Sheet along with peripherals like DC Motor, Stepper Motor, Servo Motor, Keypad, LCD, 7-Segment Display, and Relay Modules.

It also supports expansion peripherals such as GSM Module, RTC Module, SD Card Module, and UART/Serial Communication, enabling a wide range of embedded interfacing studies using PIC microcontrollers.

This hardware was assembled for college laboratory use to help students practically understand motor control, display interfaces, RTC & SD logging, GSM communication, relay switching, and sensor integration with PIC18 MCUs.

For students and hobby projects, the UDB is also available separately along with a Embedded Systems Training & Education like-

- ✓ College Lab Experiments / Practicals
- ✓ Mini & Major Project Development
- ✓ Product Prototyping & Testing
- ✓ Motor Control & Automation Projects
- ✓ RTC + SD Card Data Logging Projects
- ✓ Remote Monitoring using GSM Communication
- ✓ Sensor Interfacing & Real-Time Control