

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
KHOA ĐIỆN – ĐIỆN TỬ

PGS. TS NGUYỄN VĂN TIỀM
Bộ môn Điều khiển học
ĐT: 0904226592
Email: nguyenvantiem@utc.edu.vn

BÀI GIẢNG
HỆ THỐNG ĐIỀU KHIỂN NHÚNG

Phần 1

Hệ thống điều khiển nhúng PIC16F877a và MPLAB

Phần 1. Giới thiệu phần mềm lập trình MPLAB XC8

Bộ công cụ lập trình IDE MPLABX và MPLAB XC Compilers do hãng Microchip phát triển. Các phần mềm này được phát triển để thay thế bộ công cụ lập trình MPLAB đã lỗi thời. MPLAB IDE sử dụng nền tảng NetBeans IDE của Oracle và hỗ trợ hệ điều hành Windows, Mac và Linux.

MPLAB XC thay thế cho các trình biên dịch MPLABC và Hi-Tech C, hãng microchip khuyến khích các lập trình viên sử dụng MPLAB XC cùng với MPLAB X IDE để phát triển các ứng dụng trên vi điều khiển PIC.

Lập trình với MPLAB XC8

+ Vi điều khiển 16F877A

Các chân I/O (GPIO – General Purpose Input Output) của vi điều khiển PIC được chia ra thành nhiều cổng (Port): PORTA, PORTB, PORTC, PORTD, PORTE.

Các PORTA, PORTB, PORTC và PORTD là 8 bit. Dòng PIC16F mỗi port được quản lý bởi hai thanh ghi (registers): TRIS và PORT, ví dụ TRISA, PORTA, TRISB, PORTB, TRISC, PORTC, TRISD, PORTD.

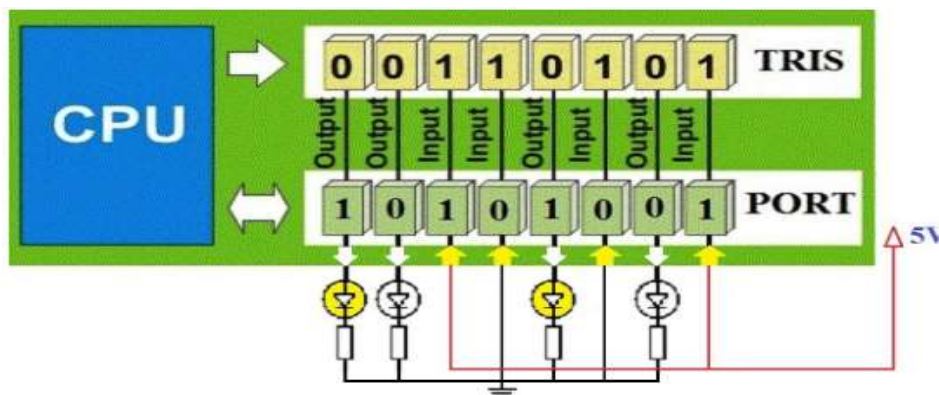
Thanh ghi TRIS đang ở mức logic '1' thì chân đó đang ở trạng thái Input (đầu vào), nếu TRIS ở mức logic '0' thì chân đó đang ở trạng thái Output (đầu ra). Tất cả các chân Input sẽ ở trạng thái cao trở (Hi-Impedance).

Thanh ghi TRIS dùng để định nghĩa từng chân ở trạng thái input (vào) hay ở trạng thái output (ra).

Thanh ghi PORT dùng để đọc hoặc ghi dữ liệu ra chân tương ứng.

Trong trường hợp một chân output (TRIS = 0), nếu thanh ghi PORT ở mức 1 thì chân tương ứng sẽ xuất hiện Logic high (VDD), ngược lại nếu thanh ghi PORT ở mức '0' thì chân tương ứng sẽ xuất hiện mức logic thấp Logic low (VSS).

Với PIC 16F877A sử dụng nguồn 5V thì VDD = 5V và VSS = 0V.



Minh họa trạng thái của các chân liên quan đến nội dung của các thanh ghi TRIS và PORT.

Ghi dữ liệu vào thanh ghi

Có thể ghi dữ liệu vào từng bit hoặc ghi dữ liệu vào tất cả các bit trên thanh ghi (chú ý nội dung của thanh ghi là 8 bit)

Ví dụ:

- 1 TRISC0 = 1; //Makes 0th bit of PORTC Input
- 2 TRISC5 = 0; //Makes 5th bit of PORTC Output
- 3 RB3 = 1; //Makes 3ed bit of PORTB at Logic High
- 4 RB7 = 0; //Makes 7th bit of PORTB at Logic Low

Ghi toàn bộ thanh ghi

Trong ngôn ngữ C quy định ký hiệu cho các dữ liệu với các hệ đếm như sau

- + Một số bắt đầu bằng '0b' thì số đó thuộc hệ đếm nhị phân (binary)
- + Một số bắt đầu bằng '0' thì số đó thuộc hệ đếm bát phân (Octal)
- + Một số bắt đầu bằng '0x' thì số đó thuộc hệ đếm lục phân (Hexadecimal)
- + Một không có ký hiệu đặc biệt nào thì số đó thuộc hệ đếm thập phân (Decimal)

Ví dụ:

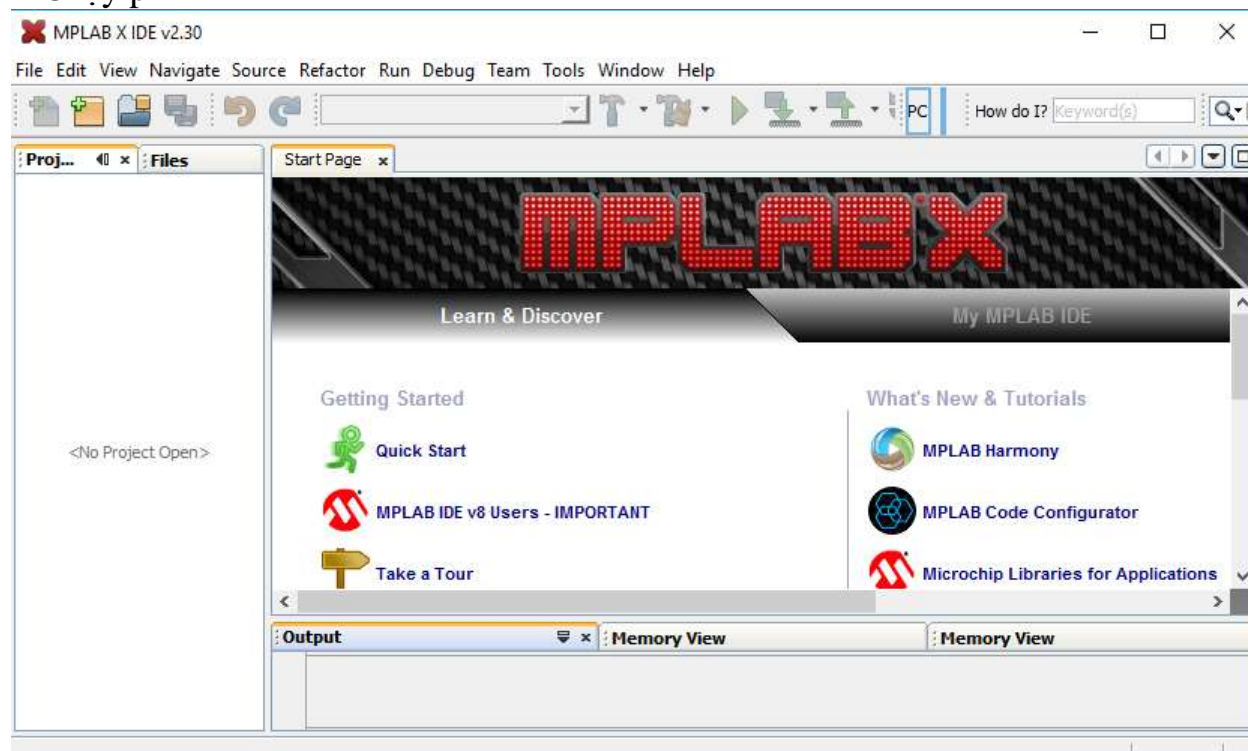
Decimal	Binary	Octal	Hexadecimal
0	0b00000000	00	0x00
1	0b00000011	03	0x03
128	0b10000000	0200	0x80
255	0b11111111	0377	0xFF

Ví dụ:

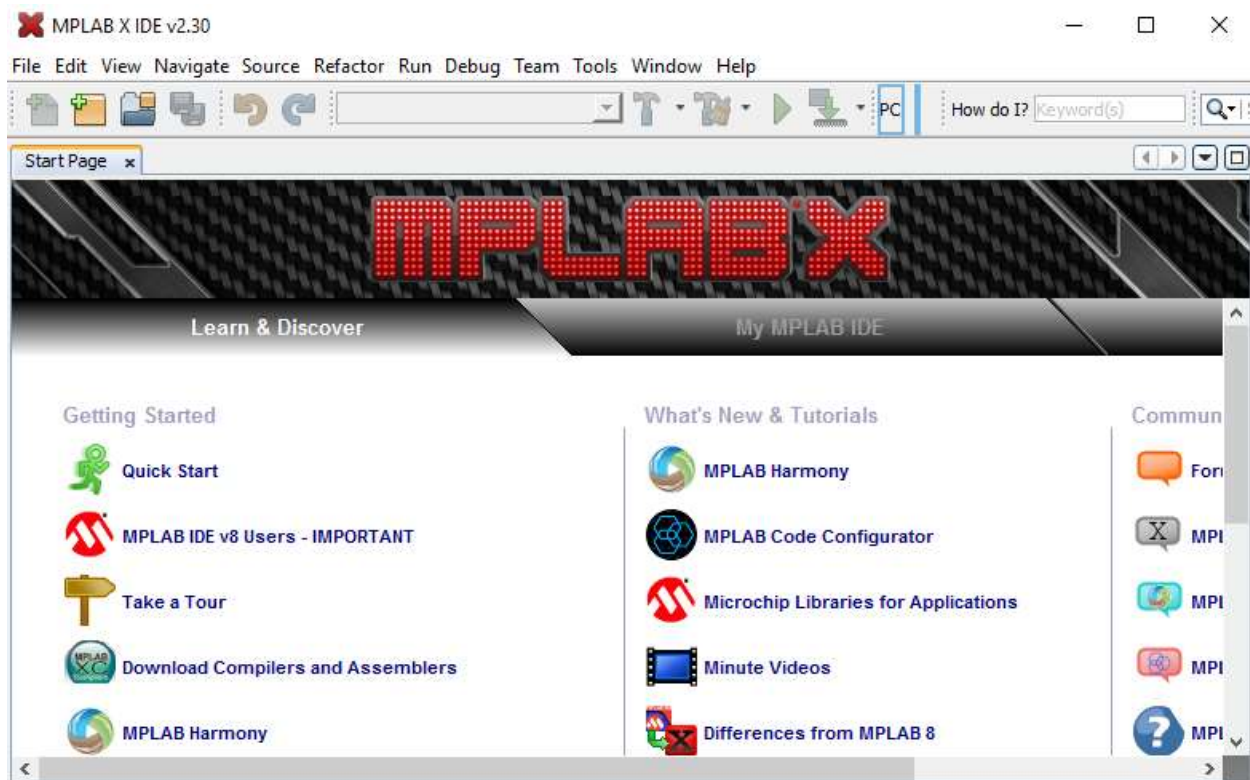
- 1 TRISC0 = 1; //Makes 0th bit of PORTC Input
- 2 PORTB = 0xFF; //Makes all pins of PORTB Logic High
- 3 TRISC = 0x00; //Makes all pins of TRISC Output
- 4 PORTD = 128; //Makes 7th bit of PORTD Logic High

Bài 1. Điều khiển LED

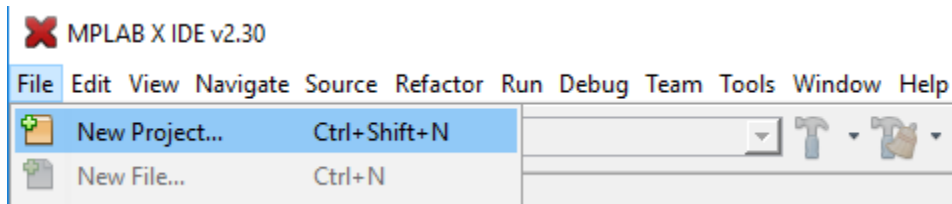
+ Chạy phần mềm MPLAB

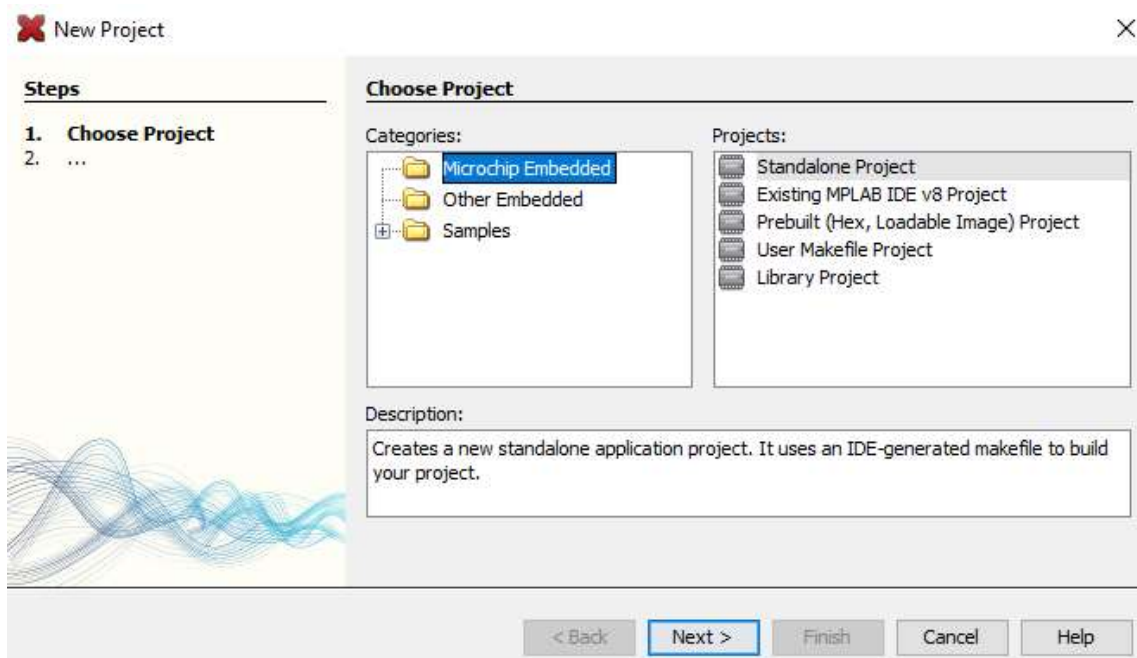


Tạo một project mới
Đưa về cửa sổ làm việc mặc định
Vào Window -> Reset Windows

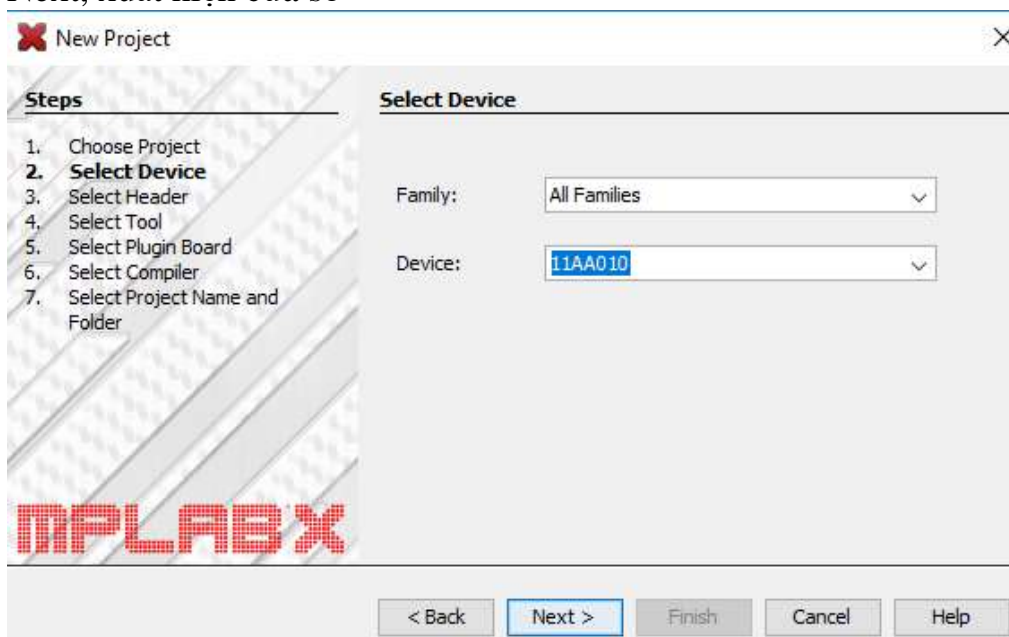


Tạo một project mới

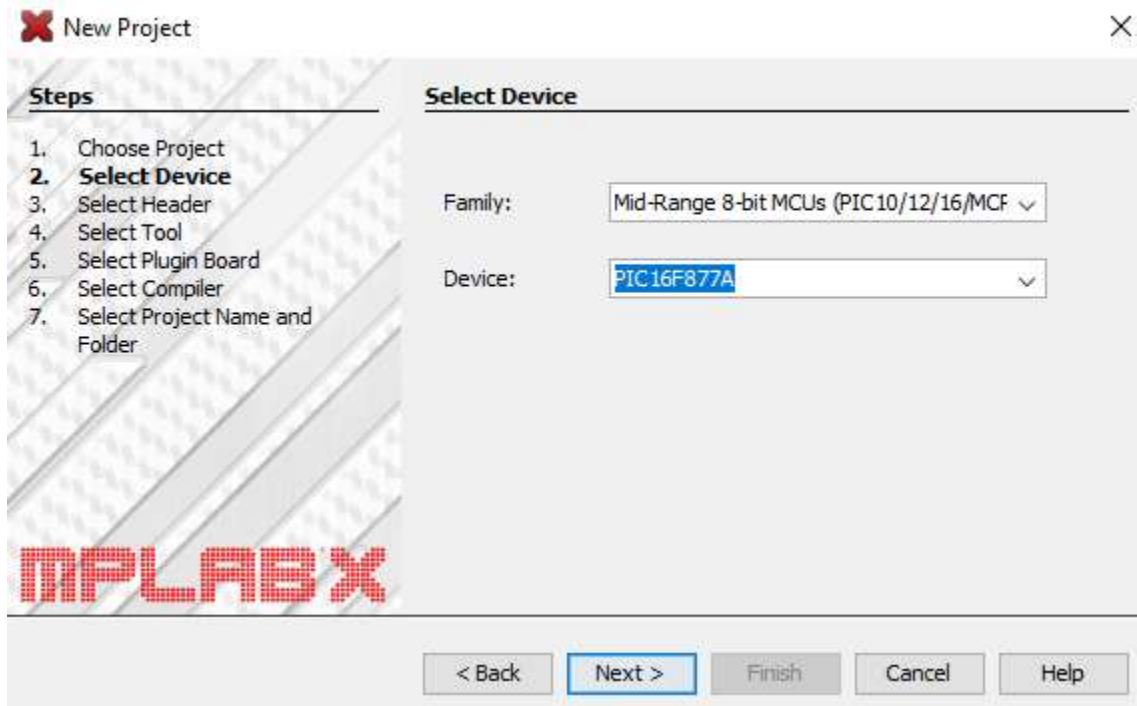




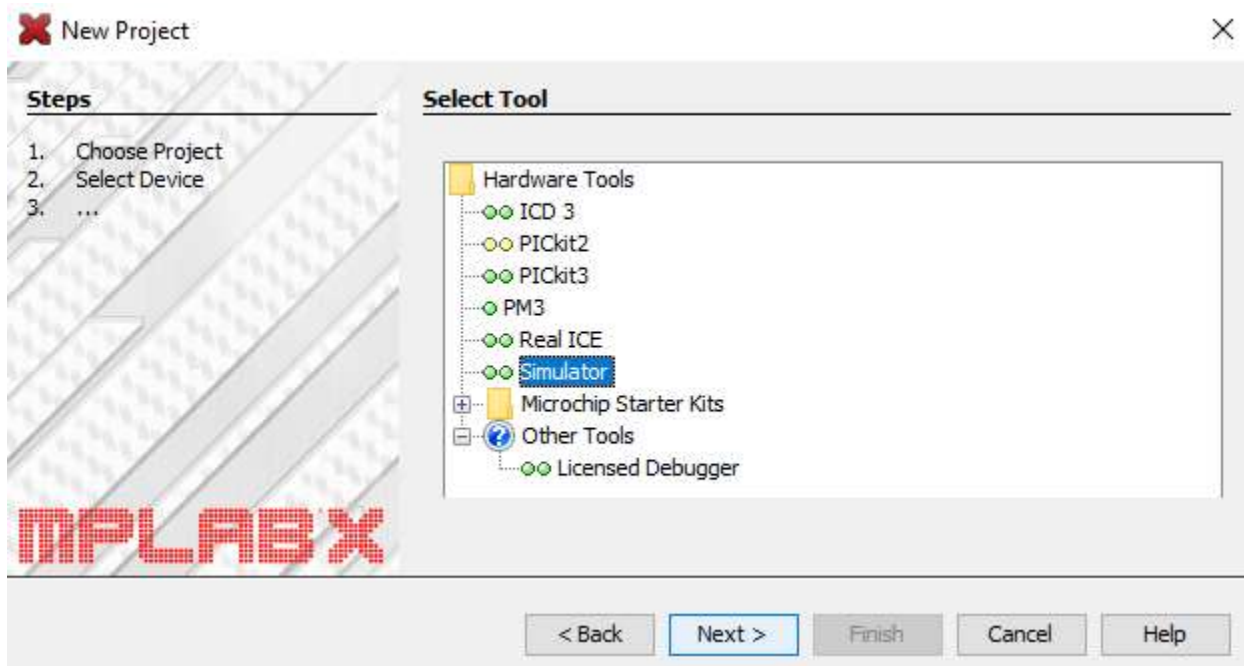
Next, xuất hiện cửa sổ



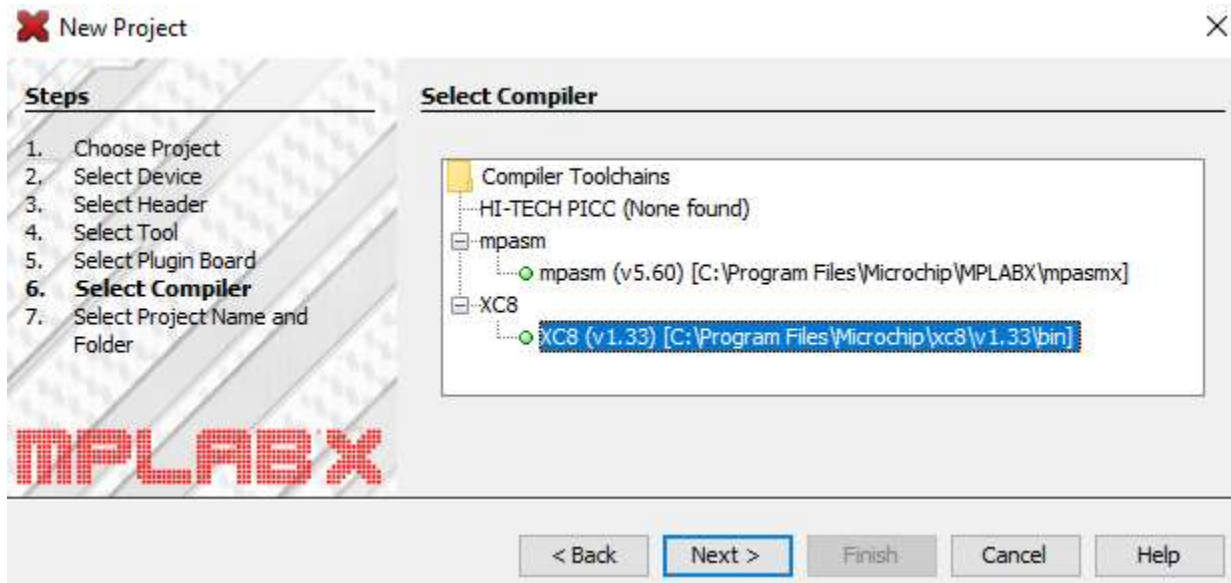
Chọn thiết bị chip



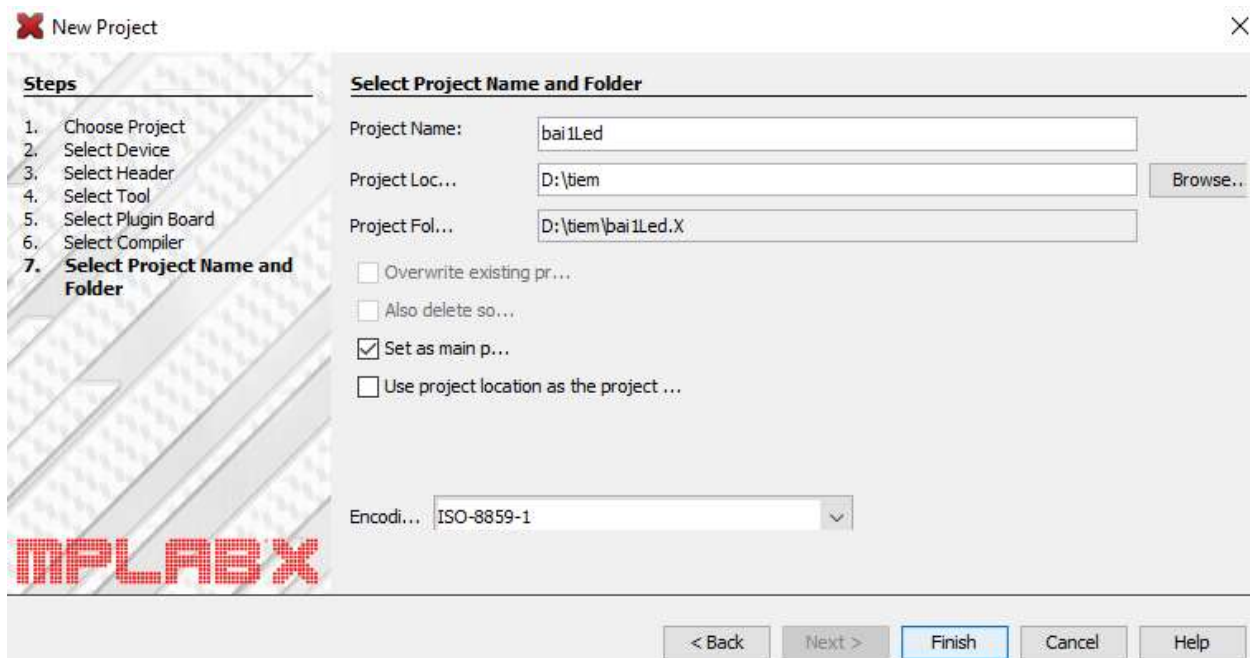
Next, xuất hiện cửa sổ và chọn Simulator



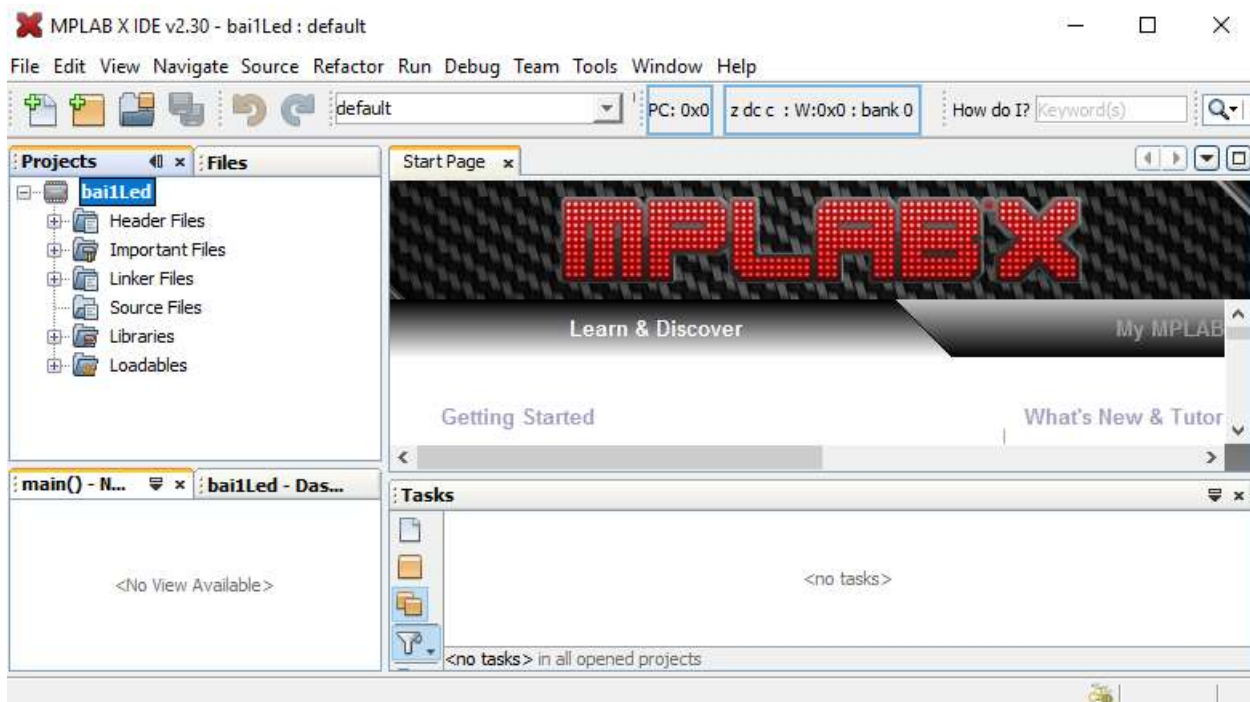
Next, hiện cửa sổ và chọn trình dịch XC8



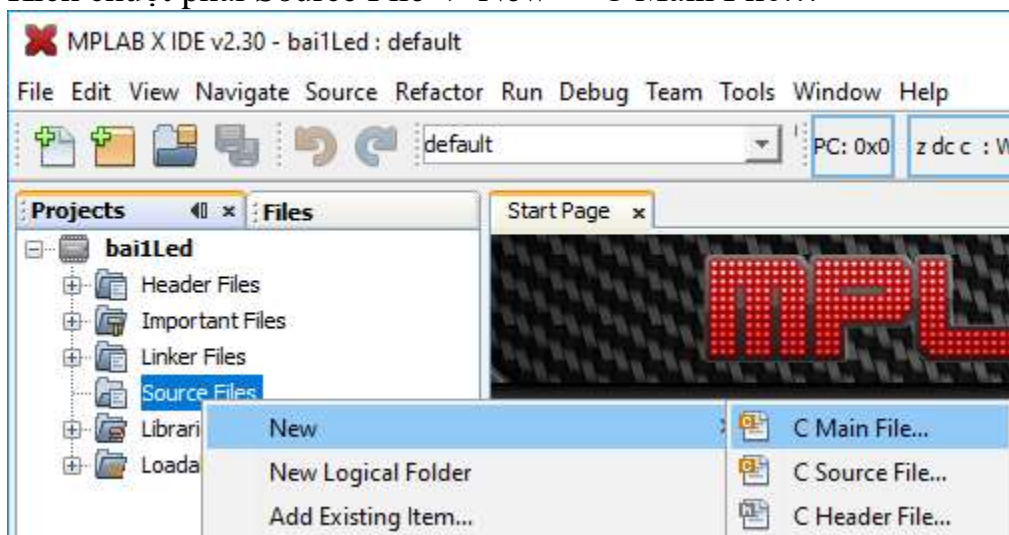
Đặt tên cho project và đặt đường dẫn: bai1Led; D:\tiem; D:\tiem\bai1Led.X



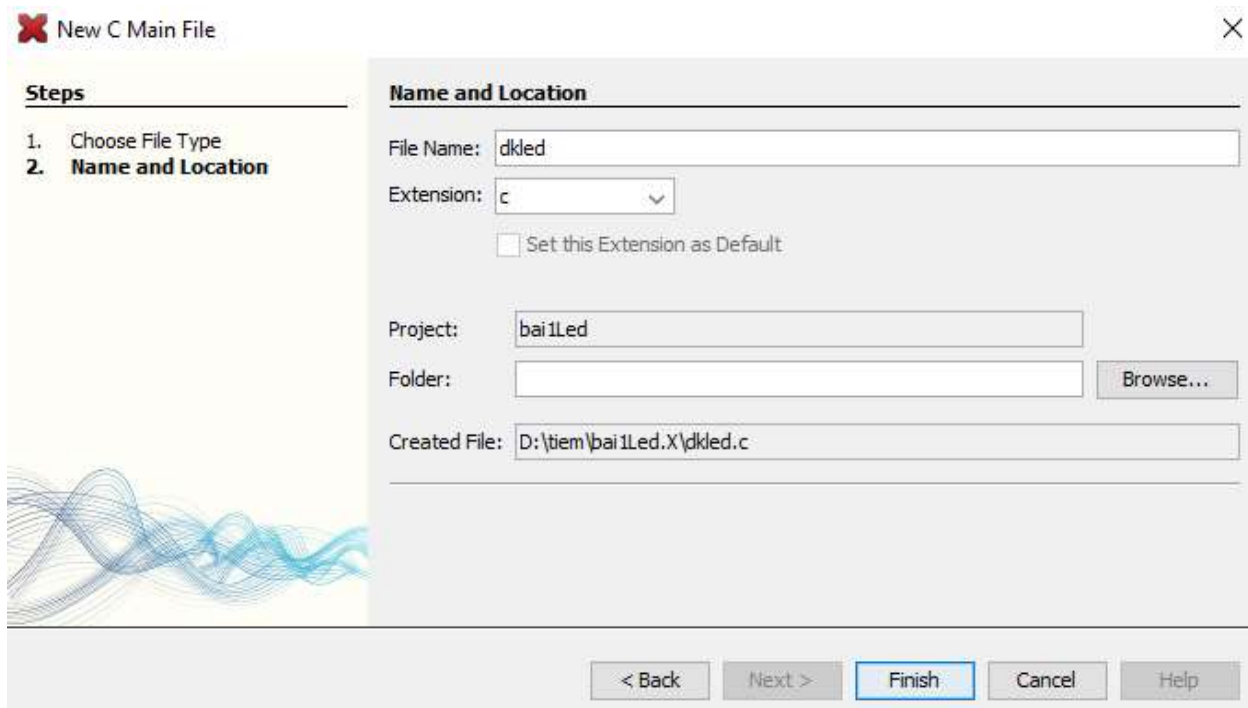
Chọn Finish, một project mới với tên là bai1Led đã được tạo ra.



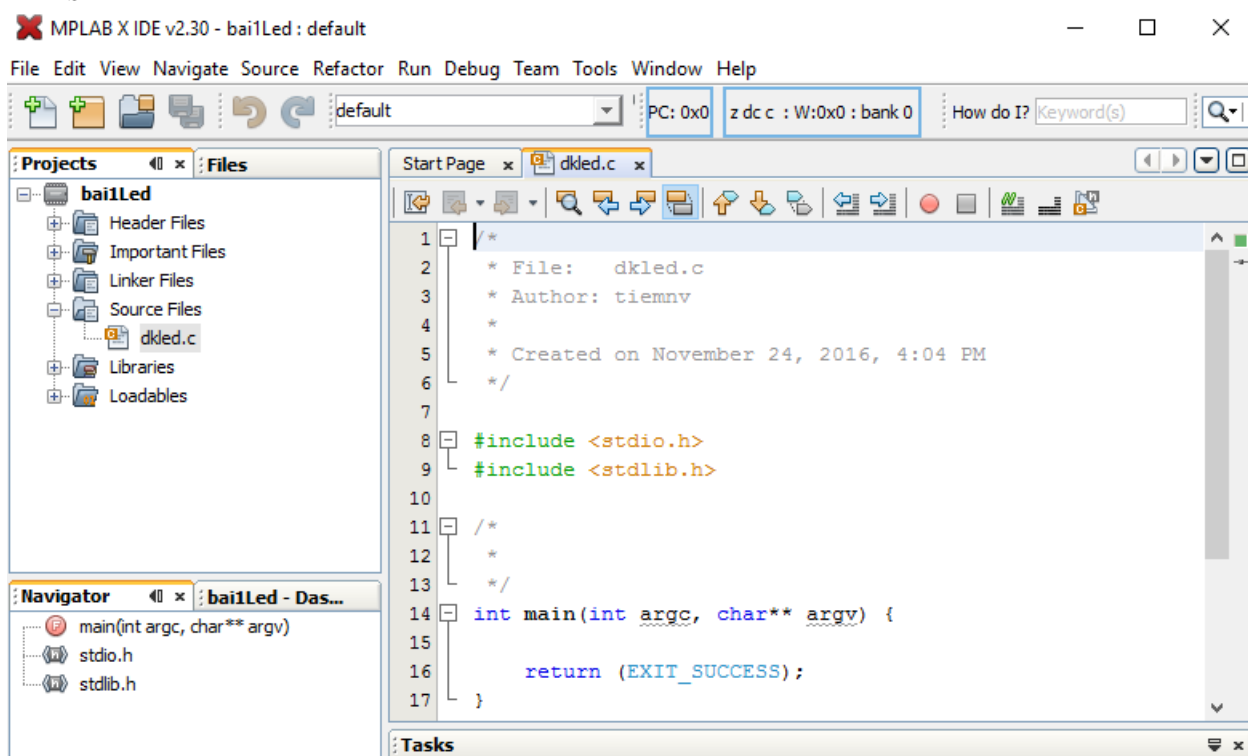
Kích chuột phải Source File -> New -> C Main File...



Gõ tên file và chú ý đường dẫn: dkled



Finish



Thêm các thư viện cần dùng và sửa lại hàm main()

```

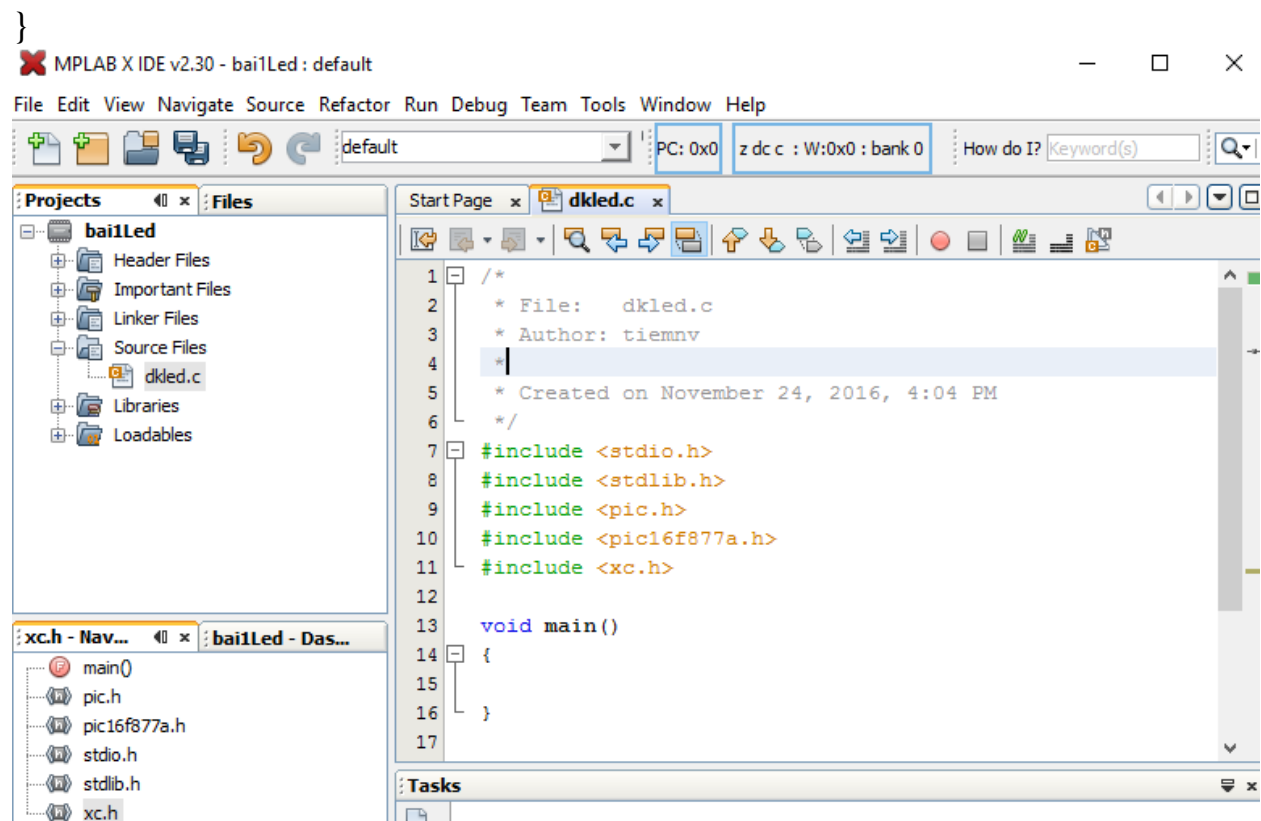
/*
 * File:   dkled.c
 * Author: tiemnv
 *
 * Created on November 24, 2016, 4:04 PM

```

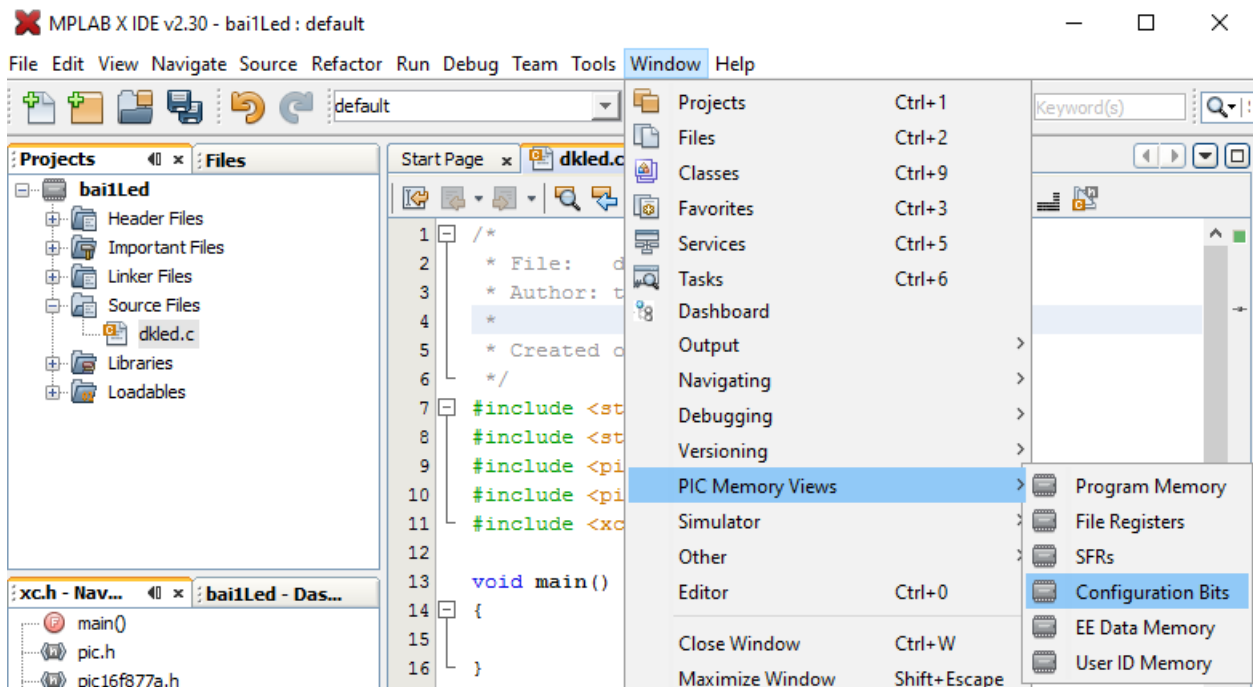
```

*/
#include <stdio.h>
#include <stdlib.h>
#include <pic.h>
#include <pic16f877a.h>
#include <xc.h>
/*
*
*/
void main()
{

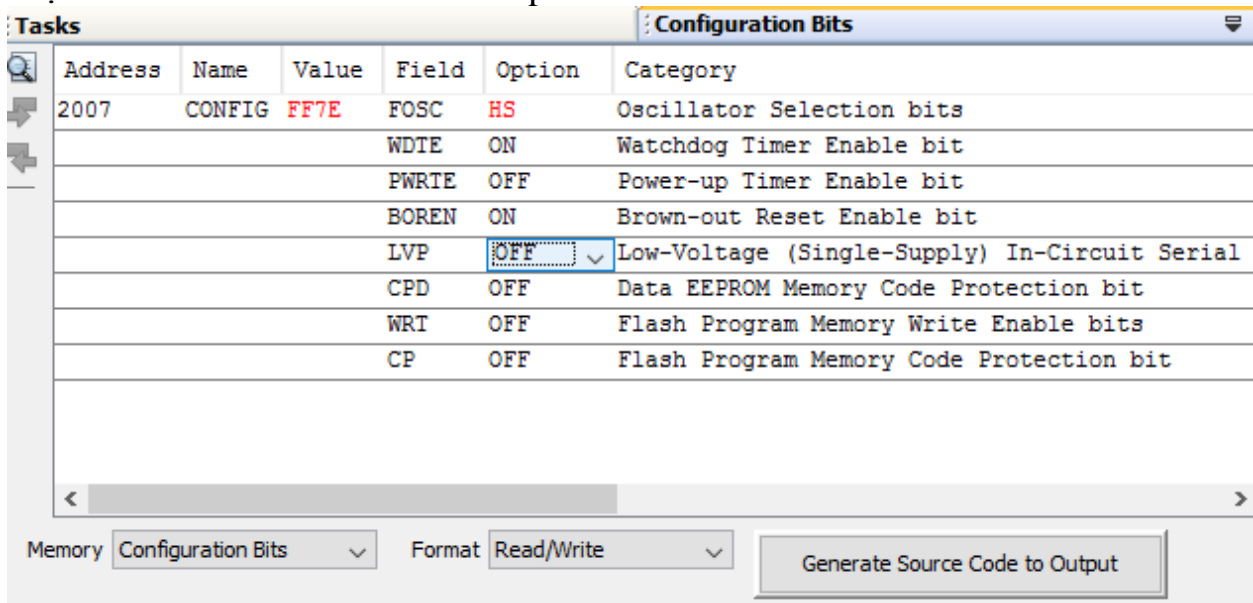
```



Cấu hình cho chip



Chọn tần số dao động thạch anh và các thông số khác, sau thao tác này thì kích chọn Generate Source Code to Output



```

Output - Config Bits Source x Tasks Configuration Bits

#include <xc.h>

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

// CONFIG
#pragma config FOSC = HS      // Oscillator Selection bits (HS 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
#pragma config CPD = OFF      // Data EEPROM Memory Code Protection bit (Data E
#pragma config WRT = OFF      // Flash Program Memory Write Enable bits (Write
#pragma config CP = OFF       // Flash Program Memory Code Protection bit (Code

```

Copy vào Head của hàm main()

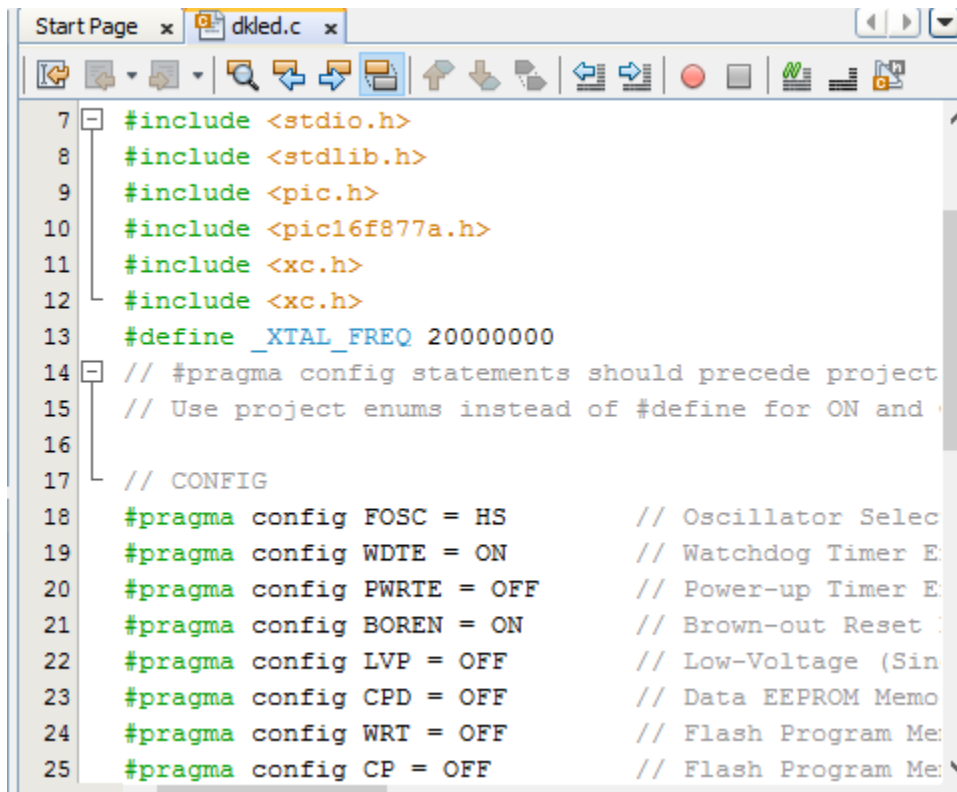
The screenshot shows a code editor with a file explorer on the left and a code window on the right. The file explorer shows a project named 'bailLed' with a file 'dkled.c' selected. The code window shows the following code:

```

10  #include <pic16f877a.h>
11  #include <xc.h>
12  #include <xc.h>
13
14  // #pragma config statements should precede project file includes.
15  // Use project enums instead of #define for ON and OFF.
16
17  // CONFIG
18  #pragma config FOSC = HS      // Oscillator Selection bits (HS o
19  #pragma config WDTE = ON      // Watchdog Timer Enable bit (WDT
20  #pragma config PWRTE = OFF    // Power-up Timer Enable bit (PWRT
21  #pragma config BOREN = ON     // Brown-out Reset Enable bit (BOR
22  #pragma config LVP = OFF      // Low-Voltage (Single-Supply) In-
23  #pragma config CPD = OFF      // Data EEPROM Memory Code Protect
24  #pragma config WRT = OFF      // Flash Program Memory Write Enab
25  #pragma config CP = OFF       // Flash Program Memory Code Prote
26
27  void main()
28  {

```

Bước tiếp theo là lập trình trong main()



```
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <pic.h>
10 #include <pic16f877a.h>
11 #include <xc.h>
12 #include <xc.h>
13 #define _XTAL_FREQ 20000000
14 // #pragma config statements should precede project
15 // Use project enums instead of #define for ON and
16
17 // CONFIG
18 #pragma config FOSC = HS          // Oscillator Selec
19 #pragma config WDTE = ON          // Watchdog Timer E
20 #pragma config PWRTE = OFF        // Power-up Timer E
21 #pragma config BOREN = ON         // Brown-out Reset
22 #pragma config LVP = OFF          // Low-Voltage (Sin
23 #pragma config CPD = OFF          // Data EEPROM Memo
24 #pragma config WRT = OFF          // Flash Program Me
25 #pragma config CP = OFF           // Flash Program Me
```

```
/*
 * File: dkled.c
 * Author: tiemnv
 *
 * Created on November 24, 2016, 4:04 PM
 */
#include <stdio.h>
#include <stdlib.h>
#include <pic.h>
#include <pic16f877a.h>
#include <xc.h>
#include <xc.h>
#define _XTAL_FREQ 20000000
// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

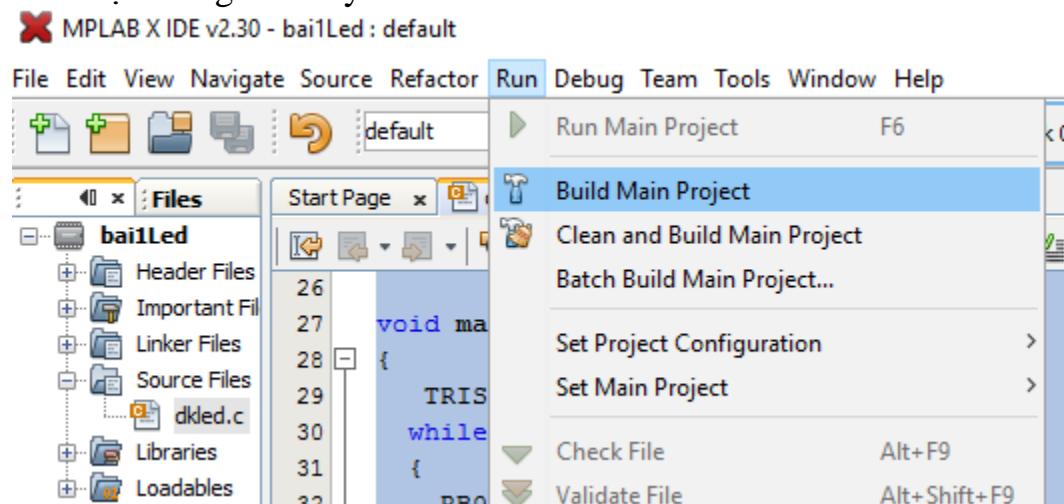
// CONFIG
#pragma config FOSC = HS          // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = ON          // Watchdog Timer Enable bit (WDT enabled)
#pragma config PWRTE = OFF        // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = ON         // Brown-out Reset Enable 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)
```

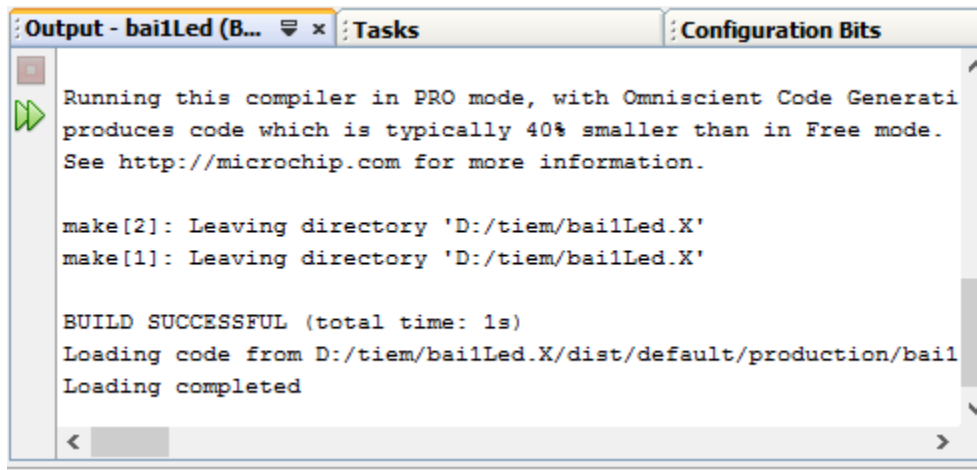


```
#pragma config CPD = OFF      // Data EEPROM Memory Code Protection bit
(Data EEPROM code protection off)
#pragma config WRT = OFF      // Flash Program Memory Write Enable bits
(Write protection off; all program memory may be written to by EECON control)
#pragma config CP = OFF       // Flash Program Memory Code Protection bit
(Code protection off)
```

```
void main()
{
    TRISB0 = 0; //RB0 as Output PIN
    TRISB2 = 0; // RB2 as Output PIN
    while(1)
    {
        RB0 = 1; // LED1 ON
        RB2 = 0; //LED2 OFF
        __delay_ms(500); // 0,5 Second Delay
        RB0 = 0; // LED1 OFF
        RB2 = 1; //LED2 ON
        __delay_ms(500); // 0,5 Second Delay
    }
}
```

Biên dịch sang mã máy





```

make -f nbproject/Makefile-default.mk SUBPROJECTS= .build-conf
make[1]: Entering directory 'D:/tiem/bailLed.X'
make -f nbproject/Makefile-default.mk
dist/default/production/bailLed.X.production.hex
make[2]: Entering directory 'D:/tiem/bailLed.X'
"C:\Program Files\Microchip\xc8\v1.33\bin\xc8.exe" --pass1 --chip=16F877A -Q
-G --double=24 --float=24 --opt=default,+asm,+asmfile,-speed,+space,-debug --
addrqual=ignore --mode=free -P -N255 --warn=0 --asmlist --summary=default,-
psect,-class,+mem,-hex,-file --output=default,-inhx032 --
runtime=default,+clear,+init,-keep,-no_startup,+osccal,-resetbits,-download,-
stackcall,+clib --output=-mcof,+elf:multilocs --stack=compiled:auto:auto "--
errformat=%f:%l: error: (%n) %s" "--warnformat=%f:%l: warning: (%n) %s" "--
msgformat=%f:%l: advisory: (%n) %s" -obuild/default/production/dkled.p1
dkled.c
"C:\Program Files\Microchip\xc8\v1.33\bin\xc8.exe" --chip=16F877A -G -
mdist/default/production/bailLed.X.production.map --double=24 --float=24 --
opt=default,+asm,+asmfile,-speed,+space,-debug --addrqual=ignore --mode=free -
P -N255 --warn=0 --asmlist --summary=default,-psect,-class,+mem,-hex,-file --
output=default,-inhx032 --runtime=default,+clear,+init,-keep,-no_startup,+osccal,-
resetbits,-download,-stackcall,+clib --output=-mcof,+elf:multilocs --
stack=compiled:auto:auto "--errformat=%f:%l: error: (%n) %s" "--
warnformat=%f:%l: warning: (%n) %s" "--msgformat=%f:%l: advisory: (%n) %s"
-odist/default/production/bailLed.X.production.elf
build/default/production/dkled.p1
Microchip MPLAB XC8 C Compiler (Free Mode) V1.33
Part Support Version: 1.33 (A)
Copyright (C) 2014 Microchip Technology Inc.
License type: Node Configuration

```

:: warning: (1273) Omniscient Code Generation not available in Free mode

Memory Summary:

Program space used 2Eh (46) of 2000h words (0.6%)
 Data space used 5h (5) of 170h bytes (1.4%)
 EEPROM space used 0h (0) of 100h bytes (0.0%)
 Data stack space used 0h (0) of 60h bytes (0.0%)
 Configuration bits used 1h (1) of 1h word (100.0%)
 ID Location space used 0h (0) of 4h bytes (0.0%)

Running this compiler in PRO mode, with Omniscent Code Generation enabled,
 produces code which is typically 40% smaller than in Free mode.
 See <http://microchip.com> for more information.

make[2]: Leaving directory 'D:/tiem/bai1Led.X'
 make[1]: Leaving directory 'D:/tiem/bai1Led.X'

BUILD SUCCESSFUL (total time: 1s)

Loading code from

D:/tiem/bai1Led.X/dist/default/production/bai1Led.X.production.hex...

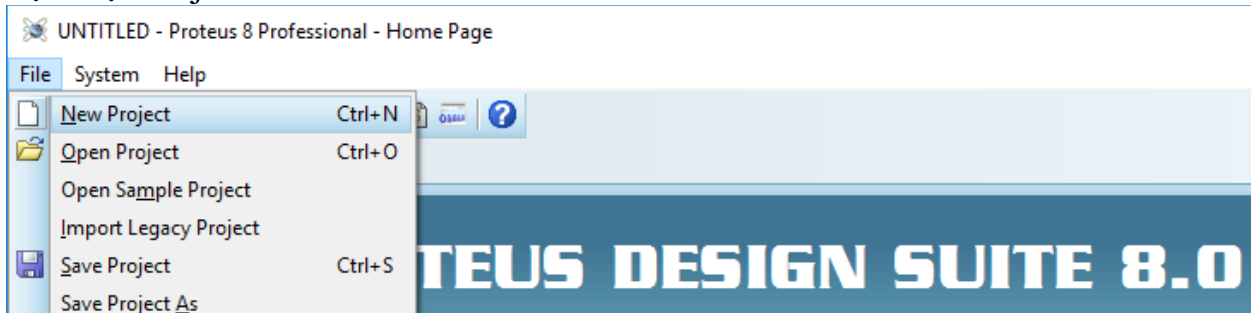
Loading completed

Thiết kế mạch phần cứng và mô phỏng trên proteus

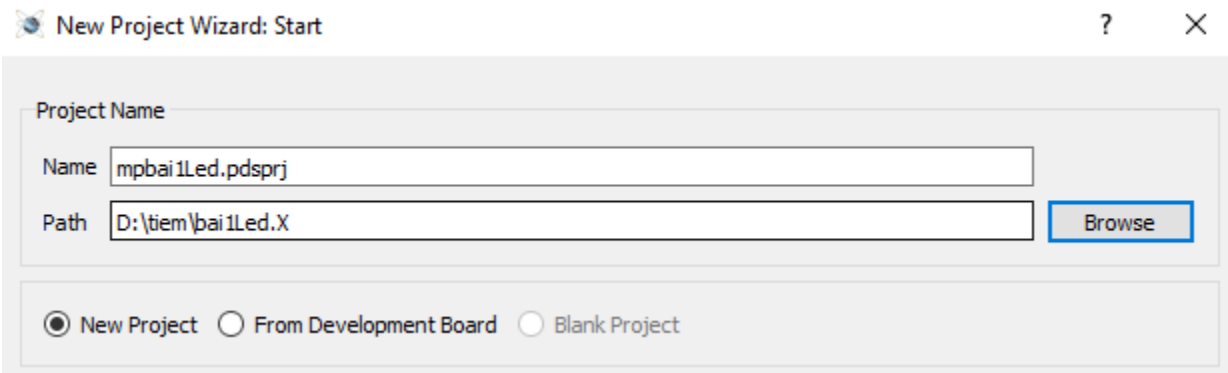
+ Khởi động proteus 8 professional



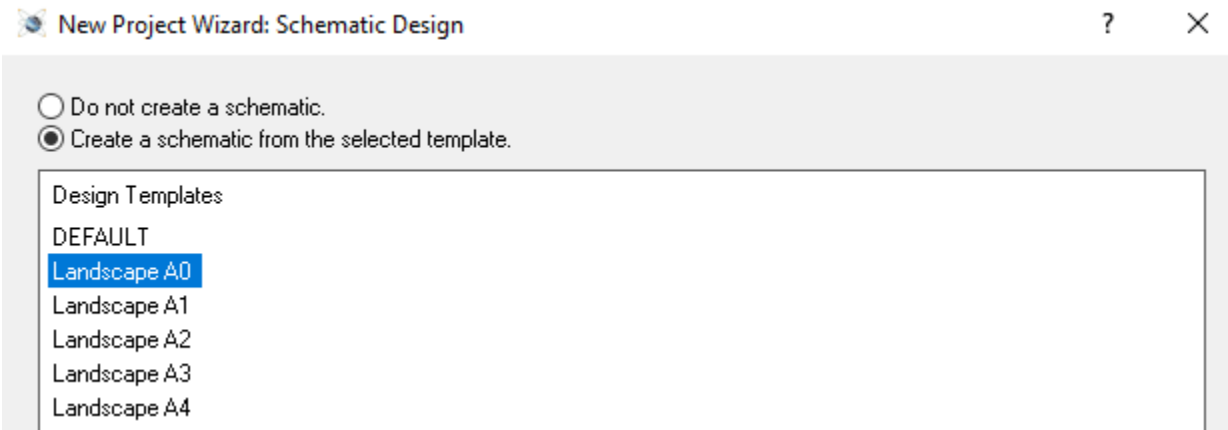
Tạo một Project mới



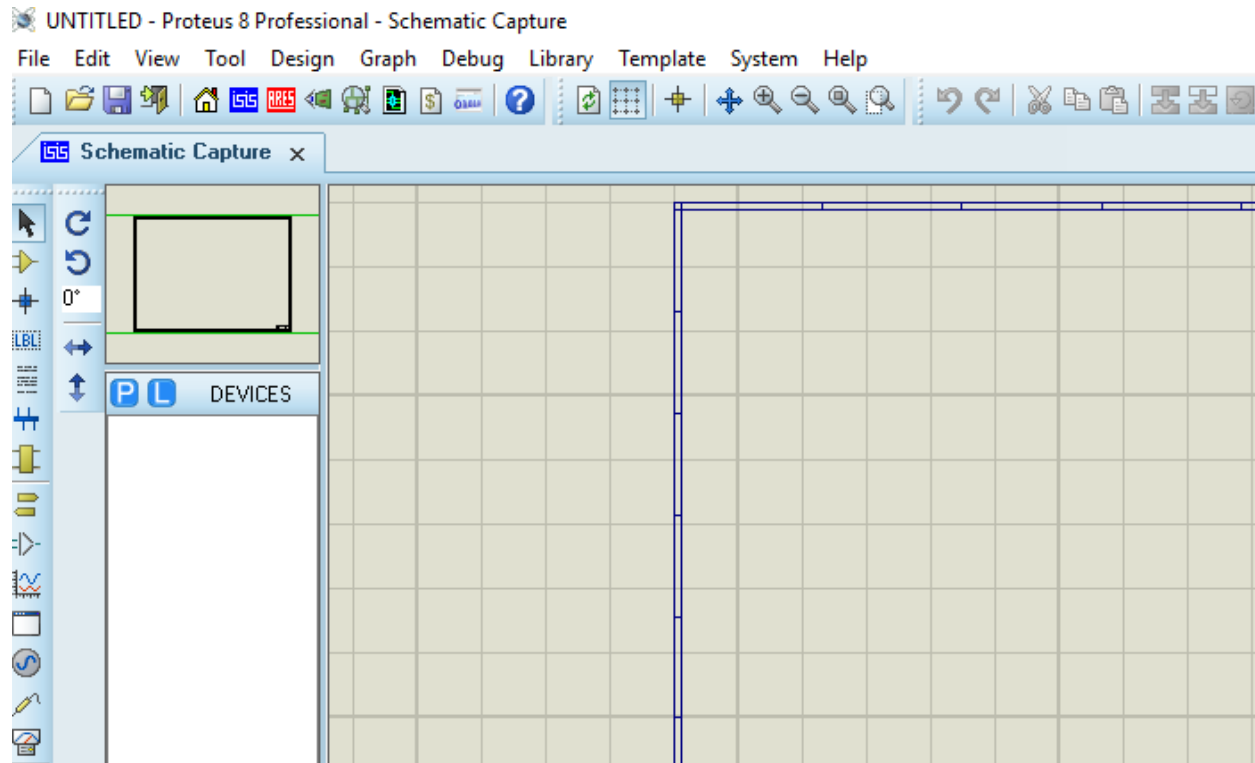
Đặt tên và đường dẫn

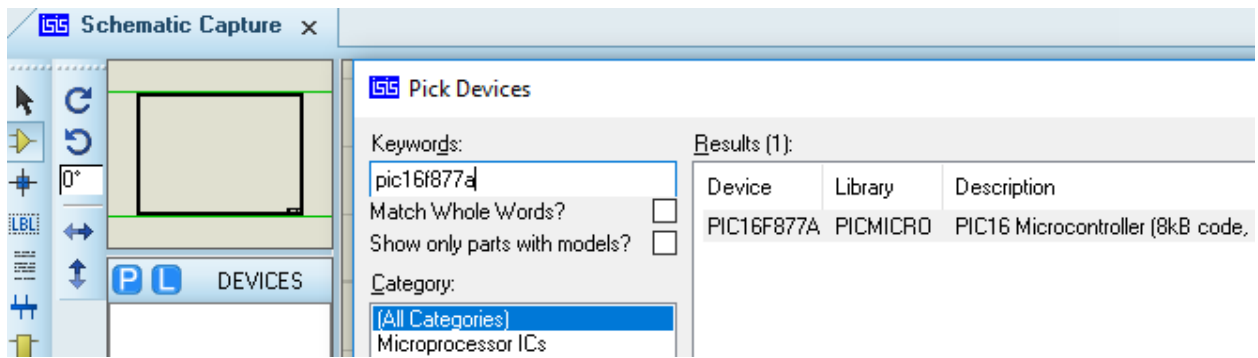


Next

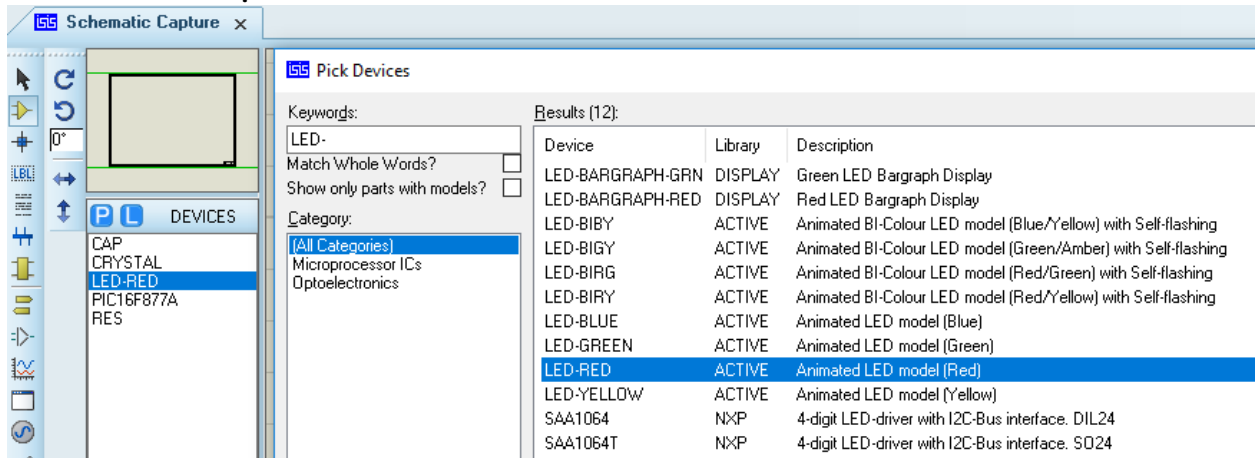


Next

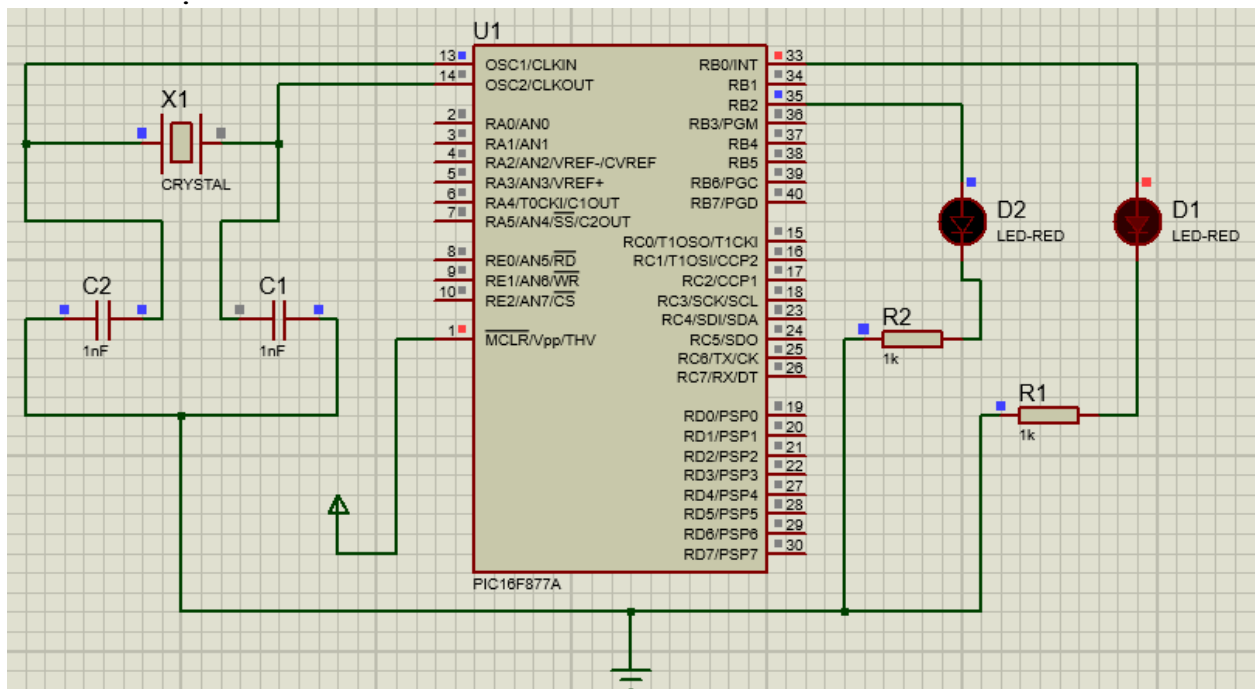




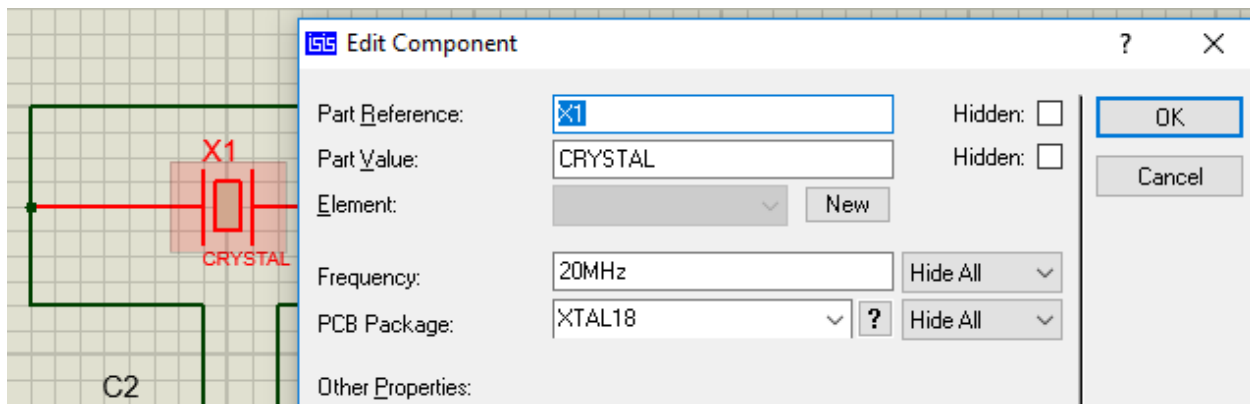
Add các linh kiện



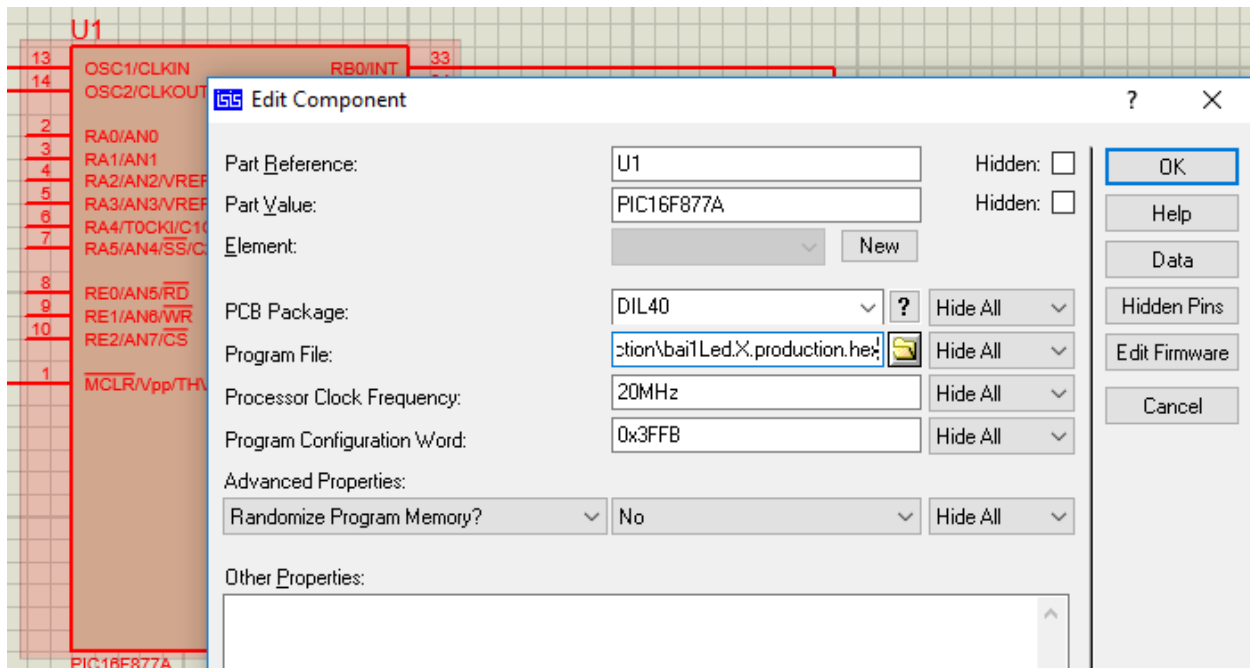
Vẽ sơ đồ mạch điều khiển



Đặt tần số cho thạch anh



Đặt tần số clock cho bộ xử lý và nạp file *bai1LedX.production.hex* cho vi điều khiển.



Tiến hành chạy mô phỏng để kiểm chứng.

Bài 2

Đọc mức logic đầu vào, nếu là mức 1 thì bật đèn, nếu là mức 0 thì tắt đèn

Chương trình main

/*

* File: dknutLed.c

* Author: tiemnv

*

* Created on November 25, 2016, 5:14 PM

*/

#include <stdio.h>

#include <stdlib.h>

#include <pic.h>

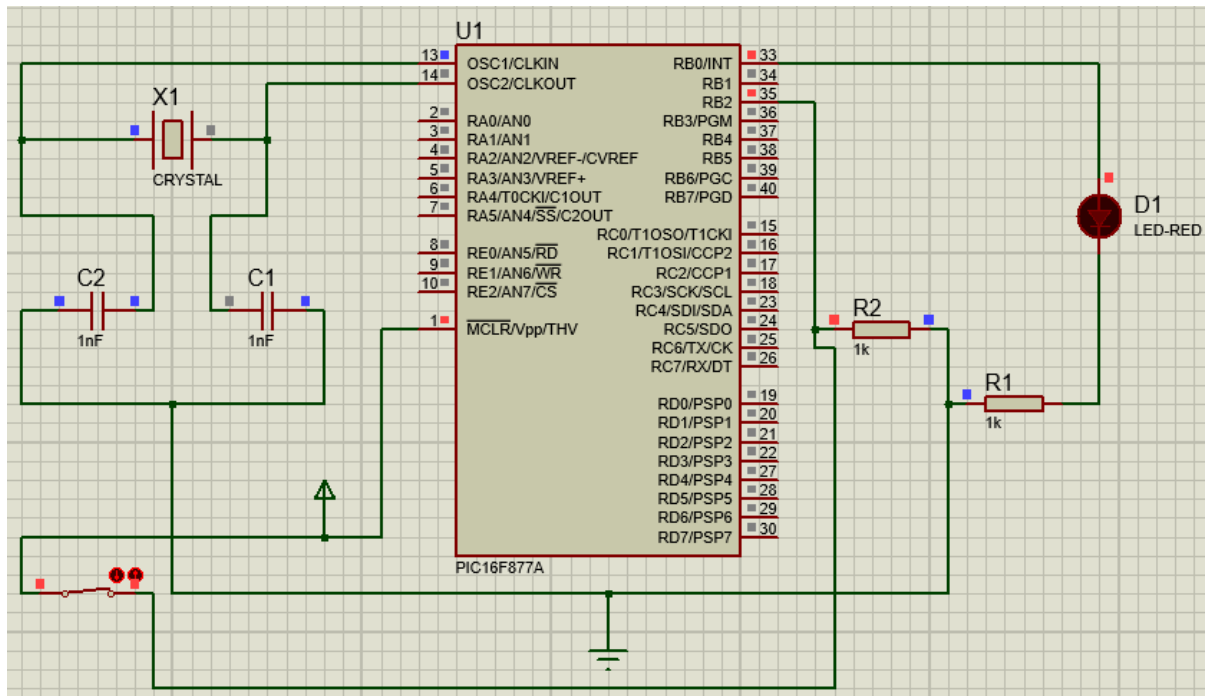
#include <pic16f877a.h>

```

#include <xc.h>
// CONFIG
#pragma config FOSC = HS      // Oscillator Selection bits (HS 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)
#define _XTAL_FREQ 20000000
/*
*
*/
void main() {
    TRISB0 = 0; //RB0 as Output PIN
    TRISB2 = 1; // RB2 as Input PIN
    while(1)
    {
        if(RB2==1)
        {
            RB0 = 1; // LED1 ON
        }
        if (RB2==0){
            RB0=0; // LED1 OFF
        }
    }
}

```

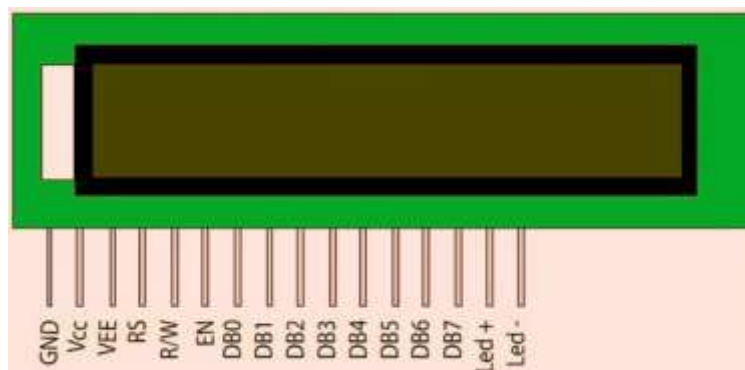
Chương trình mô phỏng trên proteus



Bài 3. Lập trình LCD với PIC16F877A và MPLAB XC8.

Cách lập trình hiển thị LCD 16x2 bằng PIC 16F877A trên MPLAB X IDE dùng MPLAB XC8. LCD 16x2 là một thiết bị hiển thị cơ bản thường được dùng để giao tiếp với người dùng thông qua việc hiển thị. LCD 16x2 gồm 2 dòng, mỗi dòng có thể hiển thị được 16 ký tự, ngoài ra còn có nhiều loại LCD khác nhau như 16x1, 16x4... với LCD này mỗi ký tự được hiển thị bởi một ma trận điểm 5x8 hoặc 5x10.

Để điều khiển được LCD chúng ta cần biết được cấu trúc phần cứng của LCD. Các loại LCD thông thường dùng IC HD44780 để quản lý việc hiển thị. Các bạn có thể xem thêm datasheet của HD44780 để hiểu nguyên lý làm việc của nó. Trong bài này này sẽ cung cấp thư viện LCD bao gồm các lệnh cơ bản giúp việc lập trình trở nên thuận tiện và đơn giản hơn, vì vậy chúng ta không nhất thiết phải tự viết thư viện cho mình.



LCD 16x2.

LCD 16x2:

Chân GND và V_{CC} chính là chân đất và chân nguồn dương cấp cho LCD. Chân thứ 3 – VEE dùng để điều chỉnh độ sáng cho LCD, thông thường chân này được nối

với một biến trở để cho phép điều chỉnh độ sáng của LCD. Vi điều khiển cần truyền dữ liệu cho LCD để LCD có thể hiển thị, dữ liệu truyền xuống gồm 2 phần:

- Data: là mã ASCII của ký tự cần hiển thị lên màn hình LCD
- Command: Lệnh yêu cầu LCD thực thi, ví dụ như xóa màn hình, dịch màn hình, vị trí hiển thị...

Lệnh được truyền xuống LCD qua chân dữ liệu (DB0 – DB7) và được điều khiển bằng chân RS (Register Select). Khi chân RS ở mức ‘1’, LCD hiển thị Data nhận được từ chân DB0 – DB7, ngược lại nếu chân RS ở mức ‘0’, LCD nhận Command từ chân DB0 – DB7. Chân EN (Enable) ở mức ‘1’ (5V) cho phép LCD hoạt động, nếu ở mức ‘0’ (GND) thì LCD không hoạt động. Chân R/W (Read/Write) dùng để cài đặt chế độ đọc (hoặc ghi) dữ liệu của LCD. Thông thường chúng ta chỉ ghi dữ liệu vào LCD nên chân R/W được nối đất (mức ‘0’). R/W = ‘0’, thì LCD đọc.

Giao tiếp với LCD gồm có 2 kiểu truyền: *truyền một lúc 8bit* hoặc *truyền thành từng phần, mỗi phần có 4bit dữ liệu*. Ở chế độ truyền 8 bit, chúng ta sử dụng 8 chân của LCD DB0-DB7, dữ liệu truyền xuống gồm 4 bit data và 4 bit commands. Ở chế độ truyền 4 bit chỉ sử dụng 4 chân của LCD và mỗi lần truyền xuống chỉ có 4 bit dữ liệu, do chỉ dùng 4 chân tín hiệu nên chế độ này thường được sử dụng nhằm tiết kiệm chân của vi điều khiển. Chế độ truyền 4 bit sẽ chậm hơn so với truyền 8 bit nhưng do tốc độ thực thi của LCD thấp hơn so với tốc độ xử lý của vi điều khiển (tầm MHz) và mắt thường của chúng ta chỉ xử lý được 24 hình/s nên sự khác biệt về tốc độ đó không ảnh hưởng đến việc hiển thị lên LCD.

Thư viện LCD

- Lcd_Init() : *Hàm này cấu hình các thông số ban đầu để có thể sử dụng thư viện LCD*

```
1 #define RS RD2
2 #define EN RD3
3 #define D4 RD4
4 #define D5 RD5
5 #define D6 RD6
6 #define D7 RD7
```

- Lcd_Clear() : *Hàm này xóa màn hình LCD*
- Lcd_Set_Cursor(int row, int column) : *Hàm này dùng để di chuyển con trỏ hiển thị đến vị trí mong muốn với row – hàng và column – cột.*
- Lcd_Write_Char(char) : *Hàm này dùng để ghi một ký tự lên LCD.*
- Lcd_Write_String(char *string) : *Hàm này dùng để ghi một chuỗi lên LCD.*
- Lcd_Shift_Right() : *Hàm này dùng để dịch dữ liệu hiển thị trên LCD sang bên phải khi dữ liệu hiển thị dài hơn 16 ký tự*
- Lcd_Shift_Left() : *Hàm này dùng để dịch dữ liệu hiển thị trên LCD sang bên trái khi dữ liệu hiển thị dài hơn 16 ký tự*

Note : The Pins to which LCD is connecting should be configured as Output Pins by writing to TRIS Register.

Cấu hình cho chip

```
// 'C' source line config statements
```

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

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

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

```
// CONFIG
```

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

```
/*
```

```
* File:  bai3lcd1.c
```

```
* Author: tiemnvn
```

```
*
```

```
* Created on November 28, 2016, 3:23 PM
```

```
*/
```

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

```
#include <stdlib.h>
```

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

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

```
#include <pic16f877a.h>
```

```
#define _XTAL_FREQ 8000000
```

```
#define RS RD2
```

```
#define EN RD3
```

```
#define D4 RD4
```



```

#define D5 RD5
#define D6 RD6
#define D7 RD7
// CONFIG
#pragma config FOSC = HS      // Oscillator Selection bits (HS 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)
/* Khai bao va dinh nghia cac ham */
////////////////////////////////////
void Lcd_Port(char a)
{
    if(a & 1)
        D4 = 1;
    else
        D4 = 0;

    if(a & 2)
        D5 = 1;
    else
        D5 = 0;

    if(a & 4)
        D6 = 1;
    else
        D6 = 0;

    if(a & 8)
        D7 = 1;
    else
        D7 = 0;
}
////////////////////////////////////
void Lcd_Cmd(char a)

```

```

{
    RS = 0;          // => RS = 0
    Lcd_Port(a);
    EN = 1;          // => E = 1
    __delay_ms(4);
    EN = 0;          // => E = 0
}
////////////////////////////////////
void Lcd_Init()
{
    Lcd_Port(0x00);
    __delay_ms(20);
    Lcd_Cmd(0x03);
    __delay_ms(5);
    Lcd_Cmd(0x03);
    __delay_ms(11);
    Lcd_Cmd(0x03);
    //////////////////////////////////
    Lcd_Cmd(0x02);
    Lcd_Cmd(0x02);
    Lcd_Cmd(0x08);
    Lcd_Cmd(0x00);
    Lcd_Cmd(0x0C);
    Lcd_Cmd(0x00);
    Lcd_Cmd(0x06);
}
////////////////////////////////////
Lcd_Clear()
{
    Lcd_Cmd(0);
    Lcd_Cmd(1);
}
////////////////////////////////////
void Lcd_Set_Cursor(char a, char b)
{
    char temp,z,y;
    if(a == 1)
    {
        temp = 0x80 + b - 1;
        z = temp>>4;
        y = temp & 0x0F;
        Lcd_Cmd(z);
        Lcd_Cmd(y);
    }
}

```

```

    }
    else if(a == 2)
    {
        temp = 0xC0 + b - 1;
        z = temp>>4;
        y = temp & 0x0F;
        Lcd_Cmd(z);
        Lcd_Cmd(y);
    }
}
////////////////////////////////////
void Lcd_Write_Char(char a)
{
    char temp,y;
    temp = a&0x0F;
    y = a&0xF0;
    RS = 1;          // => RS = 1
    Lcd_Port(y>>4);  //Data transfer
    EN = 1;
    __delay_us(40);
    EN = 0;
    Lcd_Port(temp);
    EN = 1;
    __delay_us(40);
    EN = 0;
}
////////////////////////////////////
void Lcd_Write_String(char *a)
{
    int i;
    for(i=0;a[i]!='\0';i++)
        Lcd_Write_Char(a[i]);
}
////////////////////////////////////
void Lcd_Shift_Right()
{
    Lcd_Cmd(0x01);
    Lcd_Cmd(0x0C);
}
////////////////////////////////////
void Lcd_Shift_Left()
{
    Lcd_Cmd(0x01);

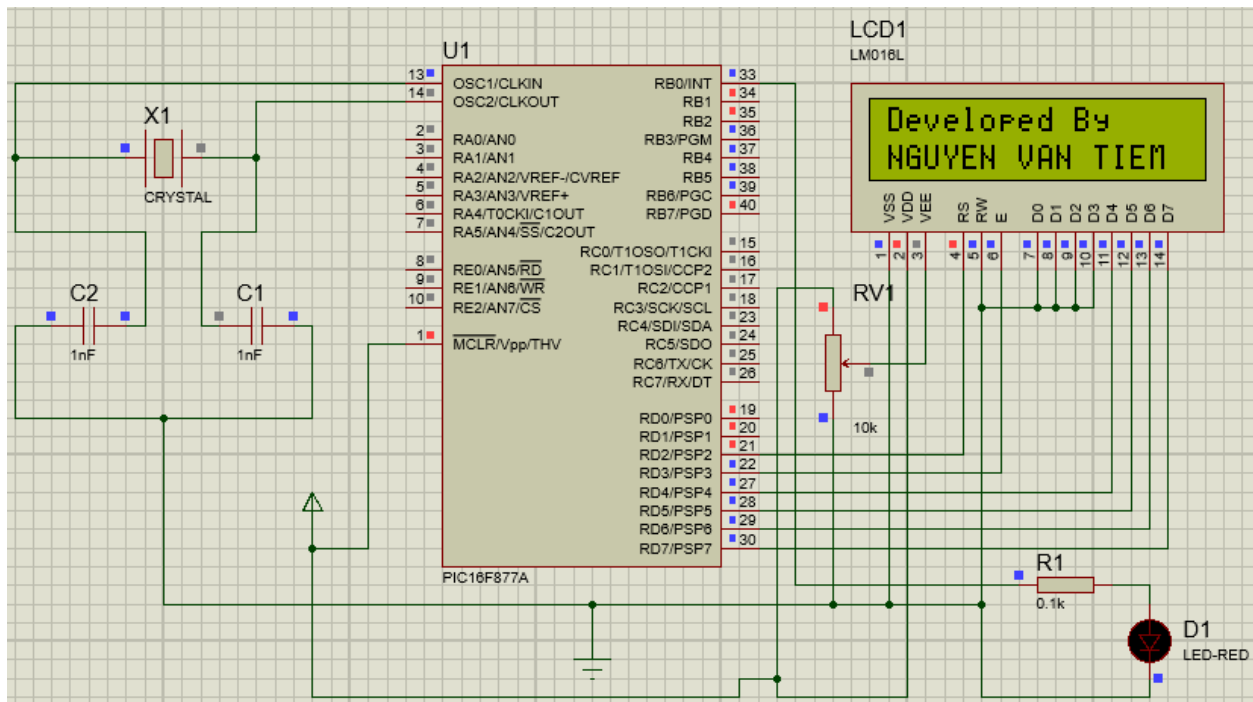
```

```

        Lcd_Cmd(0x08);
    }
    //////////////////////////////////////
void main() {
    unsigned int a;
    TRISD = 0x00;
    TRISB = 0x00; //dieu khien led
    Lcd_Init();
    while(1)
    {
        RB0 = 1; // LED1 ON
        Lcd_Clear();
        Lcd_Set_Cursor(1,1);
        Lcd_Write_String("LCD Display");
        Lcd_Set_Cursor(2,1);
        Lcd_Write_String("MPLAB XC8");
        __delay_ms(2000);
        RB0 = 0; // LED1 OFF

        Lcd_Clear();
        Lcd_Set_Cursor(1,1);
        Lcd_Write_String("Developed By");
        Lcd_Set_Cursor(2,1);
        Lcd_Write_String("NGUYEN VAN TIEM");
        __delay_ms(2000);
        Lcd_Clear();
        for(a=0;a<15;a++)
        {
            __delay_ms(300);
            Lcd_Shift_Left();
        }
        for(a=0;a<15;a++)
        {
            __delay_ms(300);
            Lcd_Shift_Right();
        }
        Lcd_Clear();
        Lcd_Set_Cursor(2,1);
        Lcd_Write_Char('e');
        Lcd_Write_Char('S');
        __delay_ms(2000);
    }
}

```



Cách dùng hàm sprintf()

Hàm sprintf() dùng để ghi giá trị của một biến ra kiểu chuỗi (string). Hàm này được dùng trong thư viện LCD để hiển thị một số dạng interger, float... trên LCD một cách đơn giản

Cú pháp

*sprintf(char * str, const char * format, ...);*

Trong đó:

str là con trỏ kiểu char dùng để lưu chuỗi cần hiển thị.

format là một ký tự cho biết kiểu dữ liệu của biến cần chuyển sang kiểu string.

... (additional arguments) là các cài đặt thêm cho các trường hợp khác

Các kiểu biến dùng cho hàm sprintf()

Bài 4Lcd là ví dụ về cách sử dụng hàm sprintf()

Sử dụng chân RB2 là đầu vào, nếu RB0 là đầu ra; hai biến float f, f1; int a, b=2016; Nếu đầu vào RB2 = 0 thì gán f1=2016; a=b; và RB0=0 để tắt đèn. Nếu RB2=1 thì gán f=0.72; a=2017; và RB0=1 để bật đèn.

Tính f1=f/2 và hiển thị các giá trị f1 và a trong các trường hợp tương ứng với đầu vào RB2.

Cấu hình cho chip giống như Bai3Lcd, và viết lại hàm main() để thực hiện được bài toán trên như sau.

////////////////////////////////////

```
int main() {
    float f,f1;
    int a, b=2016;
    char s[20];
```

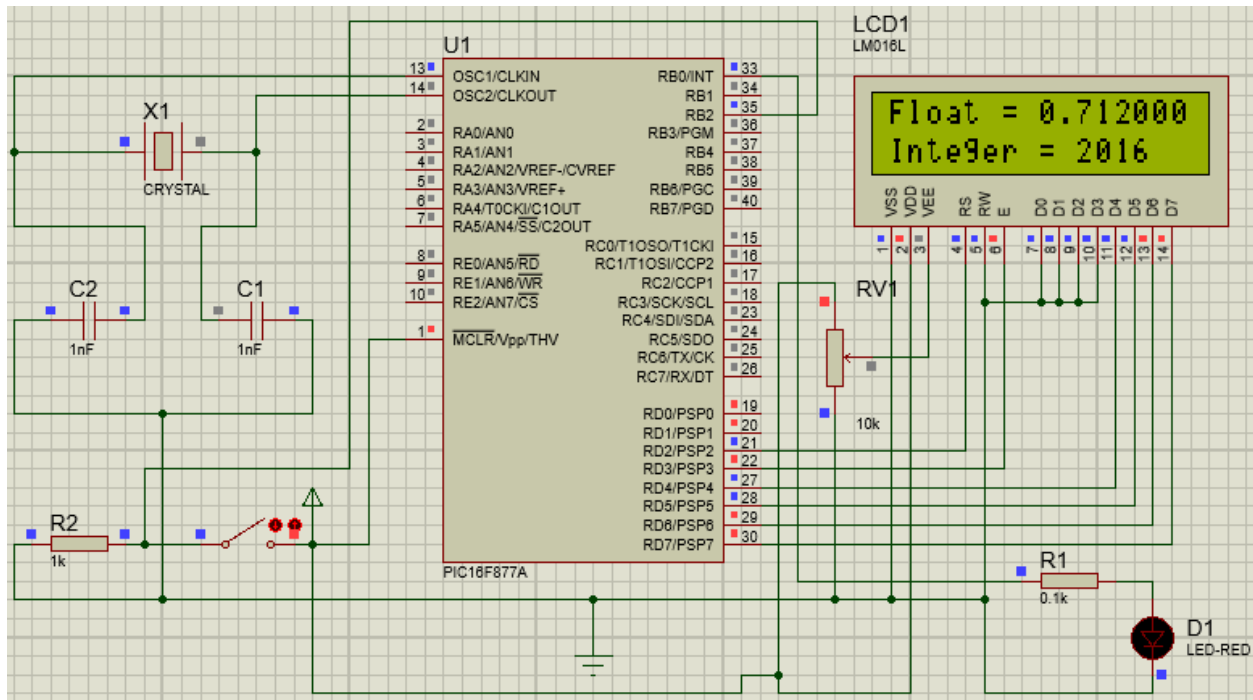


```

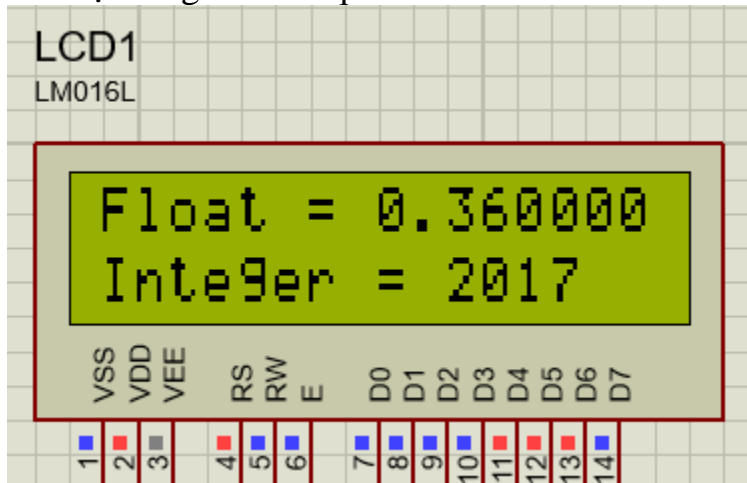
TRISD = 0x00; // PORTD is output
TRISB0 =0; //dieu khien led
TRISB2=1; // input
Lcd_Init();
while(1)
{
    if(RB2==0)
    {
        f=1.424;
        a=b;
        RB0=0; // LED1 OFF
    }
    if(RB2==1)
    {
        f=0.72;
        a=2017;
        RB0=1; // LED1 ON
    }
    f1=f/2;
    sprintf(s,"Float = %f", f1);
    Lcd_Set_Cursor(1,1);
    Lcd_Write_String(s);
    sprintf(s, "Integer = %d",a);
    Lcd_Set_Cursor(2,1);
    Lcd_Write_String(s);
}
return (0);
}

```

Chương trình mô phỏng trên proteus như sau:



Khi bật công tắc để cấp đầu vào RB2 = 1 thì màn hình LCD hiển thị như sau:



Thông qua ví dụ ở Bai4Lcd chúng ta đã biết cách phối hợp vào ra số, tính toán số thực, số nguyên và hiển thị lên LCD.

MPLAB XC8 – Lập trình đọc ADC

Bài 5. Trong bài này chúng ta sẽ học cách lập trình module ADC của vi điều khiển PIC 16F877A dùng trình biên dịch MPLAB XC8.

Tín hiệu của các đại lượng vật lý cần đo nhiệt độ, áp suất, độ ẩm... , thông thường được biến đổi thành tín hiệu điện thông qua các cảm biến và mạch khuếch đại để phù hợp mức và công suất của thiết bị hiển thị. Nếu sử dụng các thiết bị tính toán số (vi xử lý, vi điều khiển, máy tính,...) thì cần phải chuyển đổi tín hiệu tương tự sang tín hiệu số thông qua bộ biến đổi ADC (Analog to Digital Converter). Tức là biến đổi điện áp sang tín hiệu số (rời rạc).

Vi điều khiển PIC16F877A có 8 cổng ADC 10bit và nó sẽ chuyển tín hiệu analog sang giá trị số – digital 10 bit, điện áp tham chiếu thấp – ADC Lower Reference là 0VDC và mức điện áp tham chiếu cao – ADC Higher Reference là 5VDC.

$V_{ref-} = 0V$

$V_{ref+} = 5V$

$n = 10$ bits

Resolution (Độ phân giải) = $(V_{ref+} - V_{ref-}) / (2^n - 1) = 5 / 1023 = 0.004887V$

Vi điều khiển PIC16F877A với bộ ADC 10 bit thì có độ phân giải là 0.004887V.

Ví dụ với mỗi giá trị analog sẽ có giá trị digital tương ứng:

Analog Input	Digital Output		
	Binary	Hex	Decimal
0	0b0000000000	0x000	0
0.004887V	0b0000000001	0x001	1
0.009774V	0b0000000010	0x002	2
0.014661V	0b0000000011	0x003	3
4.999401V	0b1111111111	0x3FF	1023

Module ADC của vi điều khiển PIC16F877A gồm có 4 thanh ghi.

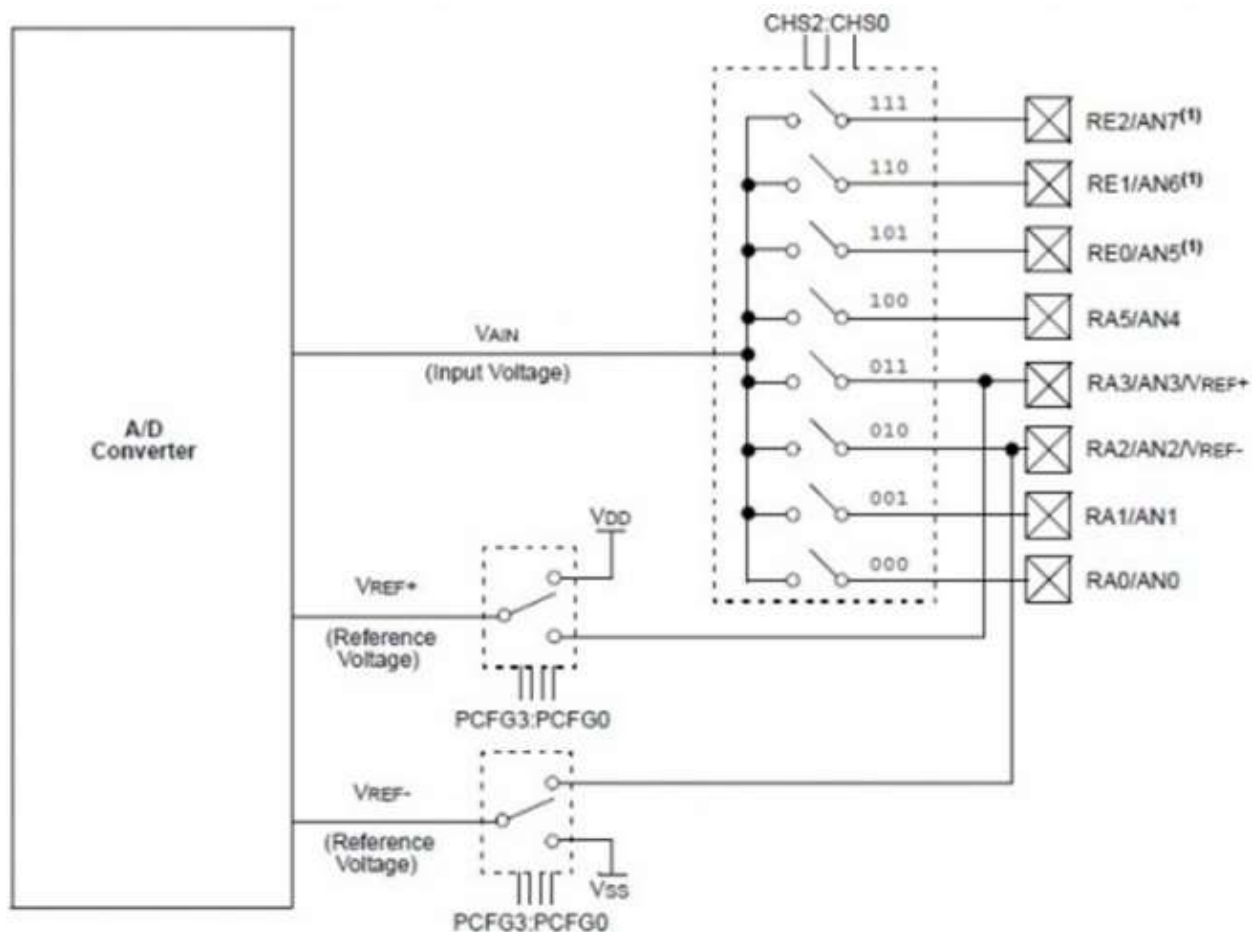
ADRESH – A/D Result High Register: Thanh ghi này lưu giá trị ADC đọc về

ADRESL – A/D Result Low Register: Thanh ghi này lưu giá trị ADC đọc về

ADCON0 – A/D Control Register 0: Thanh ghi này chứa các bit điều khiển module ADC

ADCON1 – A/D Control Register 1: Thanh ghi này chứa các bit điều khiển module ADC

ADC Block Diagram



Từ sơ đồ khối của modul ADC chúng ta có thể dễ dàng hiểu được cách làm việc của ADC

ADCON0 – A/D Control Register 0

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON
bit 7							bit 0

+ Bit 7 ~ 6 : được đặt tên là ADCS1 & ADCS0 (A/D Clock Selection bits). Các bit này kết hợp với bit ADCS2 của thanh ghi ADCON1 dùng để cài đặt tần số chuyển đổi ADC

Bảng 1. Clock conversion

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion (Tần số đọc ADC)
0	00	FOSC/2
0	01	FOSC/8
0	10	FOSC/32
0	11	FRC
1	00	FOSC/4

1	01	FOSC/16
1	10	FOSC/64
1	11	FRC

+ FOSC là tần số của thạch anh

+ FRC là tần số nội của bộ ADC R/C oscillator

+ Bit 5 ~ 3 : được đặt tên là CH2, CH1 & CH0 là các bit lựa chọn các kênh analog của module ADC

000 = Channel 0 (AN0)

001 = Channel 1 (AN1)

010 = Channel 2 (AN2)

011 = Channel 3 (AN3)

100 = Channel 4 (AN4)

101 = Channel 5 (AN5)

110 = Channel 6 (AN6)

111 = Channel 7 (AN7)

+ Bit 2 : Go/Done là bit trạng thái chuyển đổi ADC.

Khi bit ADON = 1:

Go/Done = 1 Quá trình đọc ADC đang tiến hành và khi thực hiện xong bit này bị xóa và bằng 0

Go/Done = 0 Quá trình đọc ADC thực hiện xong.

+ Bit 1 : Không thực thi và có giá trị 0.

+ Bit 0 : ADON, A/D Module On bit. Set bit này ADON = 1 thì module ADC được kích hoạt và khi ADON = 0 thì module ADC bị tắt.

ADCON1 – A/D Control Register 1

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

+ Bit 7 : ADFM, A/D Result Format bit, bit này dùng để cài đặt việc lưu giá trị ADC đọc được trong 2 thanh ghi.

Như đã trình bày ở trên module ADC của vi điều khiển PIC16F877A có 10 và giá trị ADC được lưu ở 2 thanh ghi ADRESH | ADRESL, mỗi thanh ghi là 8 bit vì vậy kết hợp hai thanh ghi chúng ta có tổng cộng 16 bit. Tuy nhiên bộ ADC chỉ lưu giá trị tối đa là 10bit nên sẽ có 6 bit bị bỏ trống. Thanh ghi ADFM sẽ quyết định vị trí của 10 bit ADC sắp xếp như thế nào trong 16 bit của 2 thanh ghi gộp lại.

ADFM = 1 (Right justified) 6 bit cao của thanh ghi ADRESH không dùng, 10bit ADC được lưu vào các bit còn lại trên 2 thanh ghi

ADRESH								ADRESL							
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

ADFM = 0 (Left justified) 6 bit thấp của thanh ghi ADRESL không dùng, 10bit ADC được lưu vào các bit còn lại trên 2 thanh ghi

ADRESH								ADRESL							
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

- + Bit 6 : ADCS2, A/D Conversion clock selection bit. Xem bảng 1
- + Bit 5 ~ 4 : Không dùng và có giá trị mặc định là 0.
- + Bit 3 ~ 0 : PCFG3 ~ PCFG0, A/D Port Configuration Bits. Bit này để cài đặt các kênh ADC

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A= Analog input D=Digital I/O

C/R= # of analog input channels/ # of A/D voltage references

A/D Acquisition Time

TACQ	= Amplifier Settling Time + Hold Capacitor Charging Time + Temperature Coefficient
	= TAMP + Tc + TCOFF
	= 2 μ s + Tc + [(Temperature – 25°C)(0.05 μ s/°C)]
Tc	= CHOLD (RIC + RSS + RS) ln(1/2047)
	= - 120 pF (1 k Ω + 7 k Ω + 10 k Ω) ln(0.0004885)
	= 16.47 μ s
TACQ	= 2 μ s + 16.47 μ s + [(50°C – 25°C)(0.05 μ s/°C)]
	= 19.72 μ s

A/D Clock Selection

A/D clock được lựa chọn để đảm bảo TAD là nhỏ nhất, TAD là thời gian xử lý 1bit ADC và có giá trị là 1.6 μ s. Chúng ta chọn theo trong bảng dưới đây:

TABLE 11-1: TAD vs. MAXIMUM DEVICE OPERATING FREQUENCIES (STANDARD DEVICES (F))

AD Clock Source (TAD)		Maximum Device Frequency
Operation	ADCS2:ADCS1:ADCS0	
2 TOSC	000	1.25 MHz
4 TOSC	100	2.5 MHz
8 TOSC	001	5 MHz
16 TOSC	101	10 MHz
32 TOSC	010	20 MHz
64 TOSC	110	20 MHz
RC ^(1, 2, 3)	x11	(Note 1)

Note 1: The RC source has a typical TAD time of 4 μ s but can vary between 2-6 μ s.

2: When the device frequencies are greater than 1 MHz, the RC A/D conversion clock source is only recommended for Sleep operation.

3: For extended voltage devices (LF), please refer to **Section 17.0 "Electrical Characteristics"**.

Chương trình bài 5 như sau:

/*

* File: bai5adc1.c

* Author: tiemnv

*

* Created on November 29, 2016, 12:53 PM

*/

#include <stdio.h>

#include <stdlib.h>

// 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)

#include <xc.h>

#define _XTAL_FREQ 8000000

void ADC_Init()

{

ADCON0 = 0x81; // kênh 0, tần số/32 và kích hoạt ADC

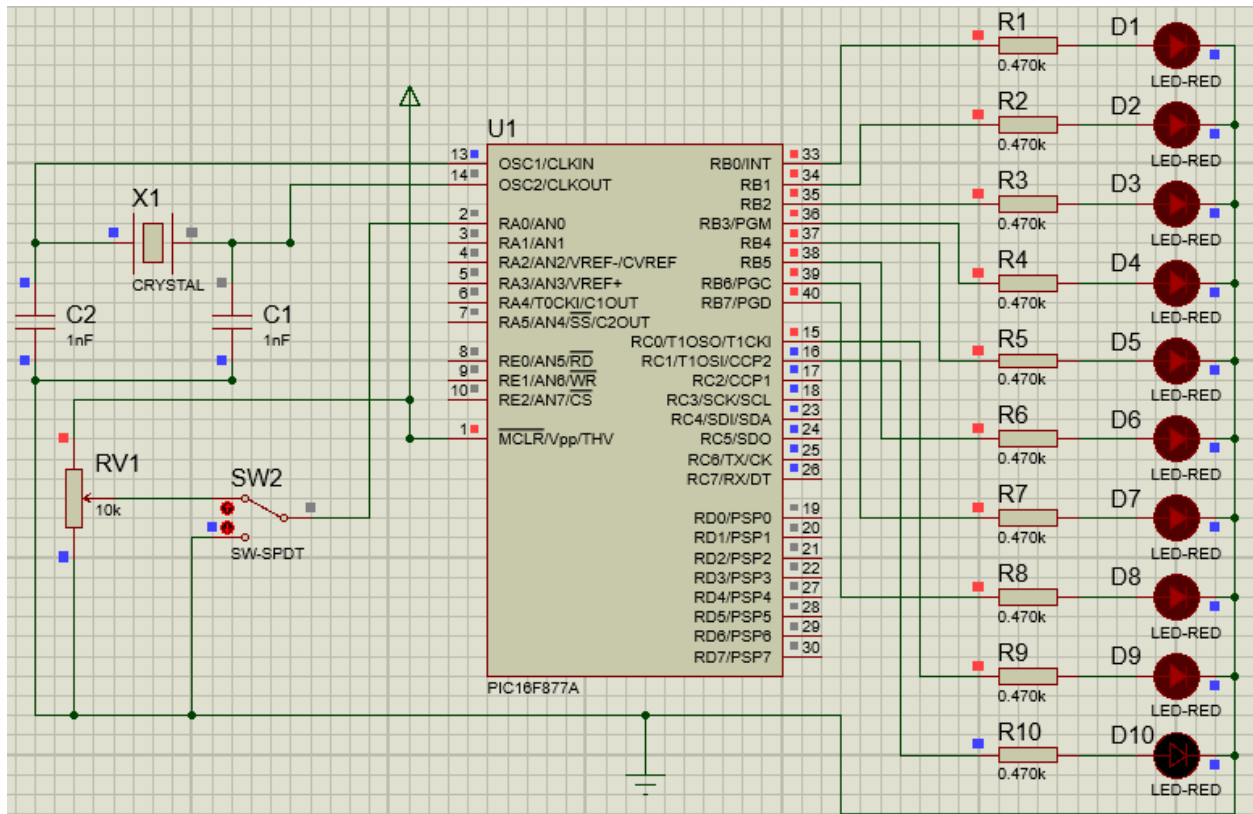

```

    ADCON1 = 0x80; // Định nghĩa kiểu ghi dữ liệu vào thanh ghi chứa dữ liệu 16
bit (theo thứ tự 10bit)
}
unsigned int ADC_Read(unsigned char channel)
{
    if(channel > 7)
        return 0;
    ADCON0 &= 0xC5;
    ADCON0 |= channel<<3;
    __delay_ms(2);
    GO_nDONE = 1;
    while(GO_nDONE);
    return ((ADRESH<<8)+ADRESL);
}

void main() {
    unsigned int a;
    TRISA = 0xFF;
    TRISB = 0x00;
    TRISC = 0x00;
    ADC_Init();
    do{
        a = ADC_Read(0);
        PORTB = a;
        PORTC = a>>8;
        __delay_ms(100);
    }
    while(1);
}

```

Chương trình mô phỏng bài 5

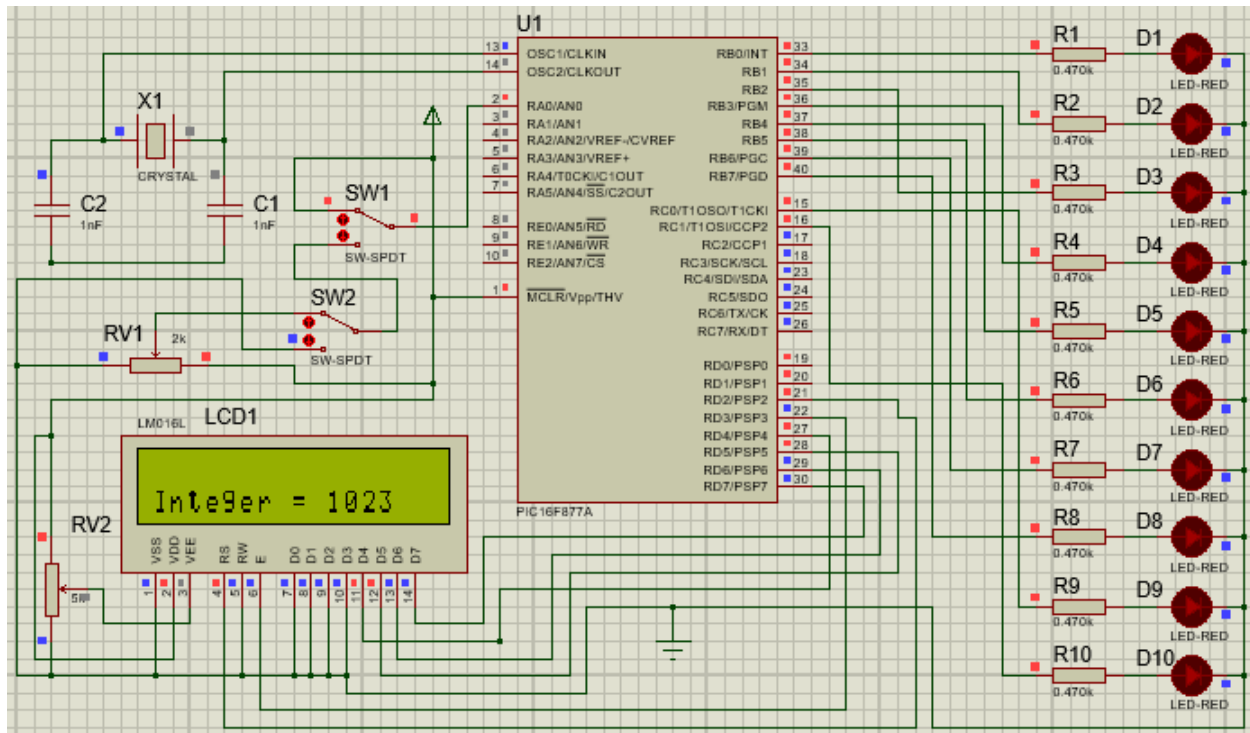


Nếu bật SW2 nối đất thì đầu vào ADC0 = 0V, do đó 10 đèn đều tắt.

Bài 6: Bai6adcLcd

Thiết kế phần cứng và viết chương trình cho hệ nhúng thực hiện nhiệm vụ sau:

+ Đọc giá trị đo lường sử dụng ADC0, có switch chuyển mạch để lựa chọn các giá trị điện áp đầu vào [0V, 2,5V và 5V]. Hiển thị giá trị đo trên LCD 16x2 và hiển thị trên đèn LED dưới dạng các bit của giá trị đo lường.



Chương trình và cấu hình cho chip

/*

* File: bai6adcLcd.c

* Author: tiemnv

*

* Created on November 29, 2016, 2:21 PM

*/

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

```
#include <stdlib.h>
```

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

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

```
#include <pic16f877a.h>
```

```
// 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 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)
```

```
#define _XTAL_FREQ 8000000
```

```
#define RS RD2
```

```
#define EN RD3
```

```
#define D4 RD4
```

```
#define D5 RD5
```

```
#define D6 RD6
```

```
#define D7 RD7
```

```
void Lcd_Port(char a)
```

```
{
    if(a & 1)
        D4 = 1;
    else
        D4 = 0;

    if(a & 2)
        D5 = 1;
    else
        D5 = 0;

    if(a & 4)
        D6 = 1;
    else
        D6 = 0;

    if(a & 8)
        D7 = 1;
    else
        D7 = 0;
}
```

```
////////////////////////////////////
```

```
void Lcd_Cmd(char a)
```

```
{
    RS = 0;          // => RS = 0
    Lcd_Port(a);
    EN = 1;          // => E = 1
    __delay_ms(4);
    EN = 0;          // => E = 0
}
```

```

////////////////////////////////////
void Lcd_Init()
{
    Lcd_Port(0x00);
    __delay_ms(20);
    Lcd_Cmd(0x03);
    __delay_ms(5);
    Lcd_Cmd(0x03);
    __delay_ms(11);
    Lcd_Cmd(0x03);
    //////////////////////////////////
    Lcd_Cmd(0x02);
    Lcd_Cmd(0x02);
    Lcd_Cmd(0x08);
    Lcd_Cmd(0x00);
    Lcd_Cmd(0x0C);
    Lcd_Cmd(0x00);
    Lcd_Cmd(0x06);
}
////////////////////////////////////
Lcd_Clear()
{
    Lcd_Cmd(0);
    Lcd_Cmd(1);
}
////////////////////////////////////
void Lcd_Set_Cursor(char a, char b)
{
    char temp,z,y;
    if(a == 1)
    {
        temp = 0x80 + b - 1;
        z = temp>>4;
        y = temp & 0x0F;
        Lcd_Cmd(z);
        Lcd_Cmd(y);
    }
    else if(a == 2)
    {
        temp = 0xC0 + b - 1;
        z = temp>>4;
        y = temp & 0x0F;
        Lcd_Cmd(z);
    }
}

```

```

        Lcd_Cmd(y);
    }
}
////////////////////////////////////
void Lcd_Write_Char(char a)
{
    char temp,y;
    temp = a&0x0F;
    y = a&0xF0;
    RS = 1;          // => RS = 1
    Lcd_Port(y>>4);  //Data transfer
    EN = 1;
    __delay_us(40);
    EN = 0;
    Lcd_Port(temp);
    EN = 1;
    __delay_us(40);
    EN = 0;
}
////////////////////////////////////
void Lcd_Write_String(char *a)
{
    int i;
    for(i=0;a[i]!='\0';i++)
        Lcd_Write_Char(a[i]);
}
////////////////////////////////////
void Lcd_Shift_Right()
{
    Lcd_Cmd(0x01);
    Lcd_Cmd(0x0C);
}
////////////////////////////////////
void Lcd_Shift_Left()
{
    Lcd_Cmd(0x01);
    Lcd_Cmd(0x08);
}
void ADC_Init()
{
    ADCON0 = 0x81;
    ADCON1 = 0x80;
}

```

```

unsigned int ADC_Read(unsigned char channel)
{
    if(channel > 7)
        return 0;
    ADCON0 &= 0xC5;
    ADCON0 |= channel<<3;
    __delay_ms(2);
    GO_nDONE = 1;
    while(GO_nDONE);
    return ((ADRESH<<8)+ADRESL);
}

```

```

void main() {
    unsigned int aa;
    TRISA = 0xFF;
    TRISB = 0x00;
    TRISC = 0x00;
    TRISD = 0x00; // PORTD is output
    char s[20];
    Lcd_Init();
    ADC_Init();
    do{
        aa = ADC_Read(0);
        PORTB = aa;
        Lcd_Clear();
        /* Lcd_Set_Cursor(1,1);
        Lcd_Write_String("Developed By");
        Lcd_Set_Cursor(2,1);
        Lcd_Write_String("NGUYEN VAN TIEM");
        __delay_ms(2000);*/

        sprintf(s, "Integer = %d",aa);
        Lcd_Set_Cursor(2,1);
        Lcd_Write_String(s);

        PORTC = aa>>8;
        __delay_ms(2000);
    }
    while(1);
}

```

Chạy chương trình:

- + Nếu bật nên 5V ở đầu vào ADC0 thì màn hình hiển thị 1023 và tất cả các đèn đều sáng.
- + Nếu bật nên 0V ở đầu vào ADC0 thì màn hình hiển thị 0 và tất cả các đèn đều tắt.
- + Nếu bật nên 2,5V ở đầu vào ADC0 thì màn hình hiển thị 511 và tất cả các đèn D1 đến D9 sáng, đèn D10 tắt.

MPLAB XC8 – Lập trình PWM

Pulse Width Modulation – PWM (Điều biến độ rộng xung) là một trong những phương pháp đơn giản và phổ biến được sử dụng để tạo ra tín hiệu điện áp analog (tương tự) từ tín hiệu digital (số). Phương pháp này được sử dụng trong nhiều ứng dụng: Digital to Analog Converter (DAC), điều khiển tốc độ động cơ, điều khiển độ sáng...

Tín hiệu PWM có dạng ON – OFF, sau một khoảng thời gian tín hiệu chuyển từ trạng thái ON qua OFF hoặc ngược lại. Khoảng thời gian tín hiệu giữ một trạng thái gọi là Modulation, sự kiện thay đổi trạng thái tín hiệu gọi là Pulse, khoảng thời gian giữa hai tín hiệu ON (hoặc OFF) liên tiếp gọi là Duty Cycle.

Hầu hết các loại vi điều khiển PIC đều có thể xuất xung PWM bằng module CCP (Capture / Compare / PWM). Để sử dụng các module đó chúng ta cần học cách lập trình trên MPLAB XC8, trong ví dụ này chúng ta sử dụng PIC 16F877A.

CCP – Capture / Compare / PWM Module

Micrichip PIC 16F877A có 2 module CCP là CCP1 và CCP2, mỗi module bao gồm 2 thanh ghi 8 bit kết hợp chúng ta sẽ có được:

- + 16 bit Capture Register
- + 16 bit Compare Register
- + PWM Master / Slave Duty Cycle register

Bài hướng dẫn này chỉ sử dụng module CCP để xuất xung PWM với độ phân giải là 10bit.

This tutorial deals only with PWM operation of CCP module. Using this we can generate PWM output having resolution up to 10 bit. Module CCP sử dụng chân

RC1 và RC2 của PORTC nên bit TRISC1 và TRISC2 phải về mức 0 để chân RC1 và RC2 là output.

Sơ đồ khối PWM của module CCP

Timer 2 được sử dụng trong module CCP, giá trị của thanh ghi TMR2 tăng từ 0 cho đến giá trị lớn nhất với mỗi input clock. Input clock được xác định với tần số làm việc của vi điều khiển (F_{osc}) và giá trị bộ chia (prescaler) của timer 2.

Module PWM

+ Prescaler Input Clock : $F_{osc} / 4$

+ Timer 2 Input Clock : $(F_{osc} / 4) / \text{Prescaler Value} = F_{osc} / (4 * \text{Prescaler})$

Time period của xung PWM được xác định thông qua giá trị của thanh ghi PR2.

Bộ Comparator so sánh giá trị của thanh ghi PR2 và TMR2, khi chúng bằng nhau thì bộ Comparator thiết lập xung PWM ở mức High, nó cũng reset giá trị của Timer 2 về 0.

Giá trị duty cycle của PWM được xác định bởi thanh ghi CCPR1L và CCP1CON<5:4> và có thể thay đổi trong quá trình hoạt động. Giá trị duty cycle được lưu để thanh ghi CCPR1H và 2 bit nội chốt lại khi giá trị PR2 và Timer 2 bằng nhau. Điều này nhằm tránh sự mất ổn định của ngõ ra PWM khi thay đổi giá trị duty cycle.

Lưu ý rằng:

Giá trị Duty Cycle nên nhỏ hơn PR2 để phù hợp với việc tạo xung PWM
Không thể sử dụng 2 tần số PWM khác nhau cho cả 2 module CCP do chúng đều dùng Timer 2 để hoạt động.

Độ phân giải PWM

Độ phân giải của PWM được tính bởi công thức

Cách sử dụng

+ Đặt giá trị PWM Period bằng cách set giá trị cho thanh ghi PR2

+ Đặt giá trị PWM Duty Cycle bằng cách set giá trị thanh ghi CCPR1L và CCP1CON<5:4>

+ Đặt chân CCP là output bằng cách set bit TRIS<2> và TRIS<1>

+ Đặt giá trị bộ chia của Timer 2 và enable Timer 2 bằng cách set giá trị cho thanh ghi T2CON

+ Cuối cùng cấu hình module CCP hoạt động ở chế độ PWM bằng cách set giá trị cho thanh ghi CCP.

BÀI 7 : GIAO TIẾP NÚT NHẤN PIC16F877A XC8

1. Nút nhấn - Button

Nói về nút nhấn (button) thì có lẽ chúng ta đã quá quen với nó rồi thậm chí chúng ta chúng ta còn tiếp xúc với nó hằng ngày nữa, ví dụ như : bàn phím máy tính, các thiết bị điều khiển tivi, máy điều hòa nhiệt độ,...



Nút nhấn thường thì có kích thước 6 – 12mm tùy thuộc vào mục đích sử dụng, chúng ta cần lựa chọn loại nút nhấn phù hợp với mục đích cho bài toán ứng dụng cụ thể.

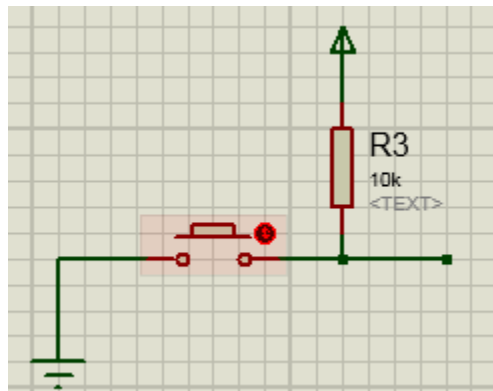


Nút nhấn tuy có 4 chân, theo chức năng của nó thì thực chất chỉ là 2 chân

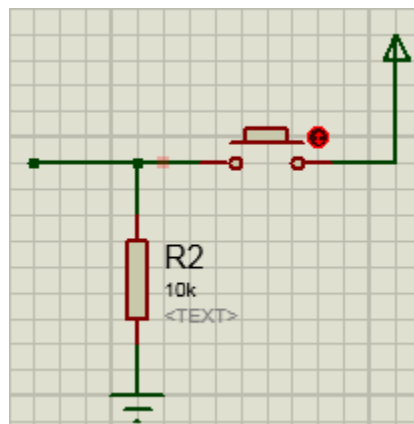


Hiện nay trên thị trường có rất nhiều loại nút nhấn khác nhau và giá thành cũng khác nhau. Độ bền của nút nhấn cũng khá cao.

Nút nhấn trong mô phỏng, tùy theo cách đấu nối mà nó có thể
+ Tích cực mức thấp.

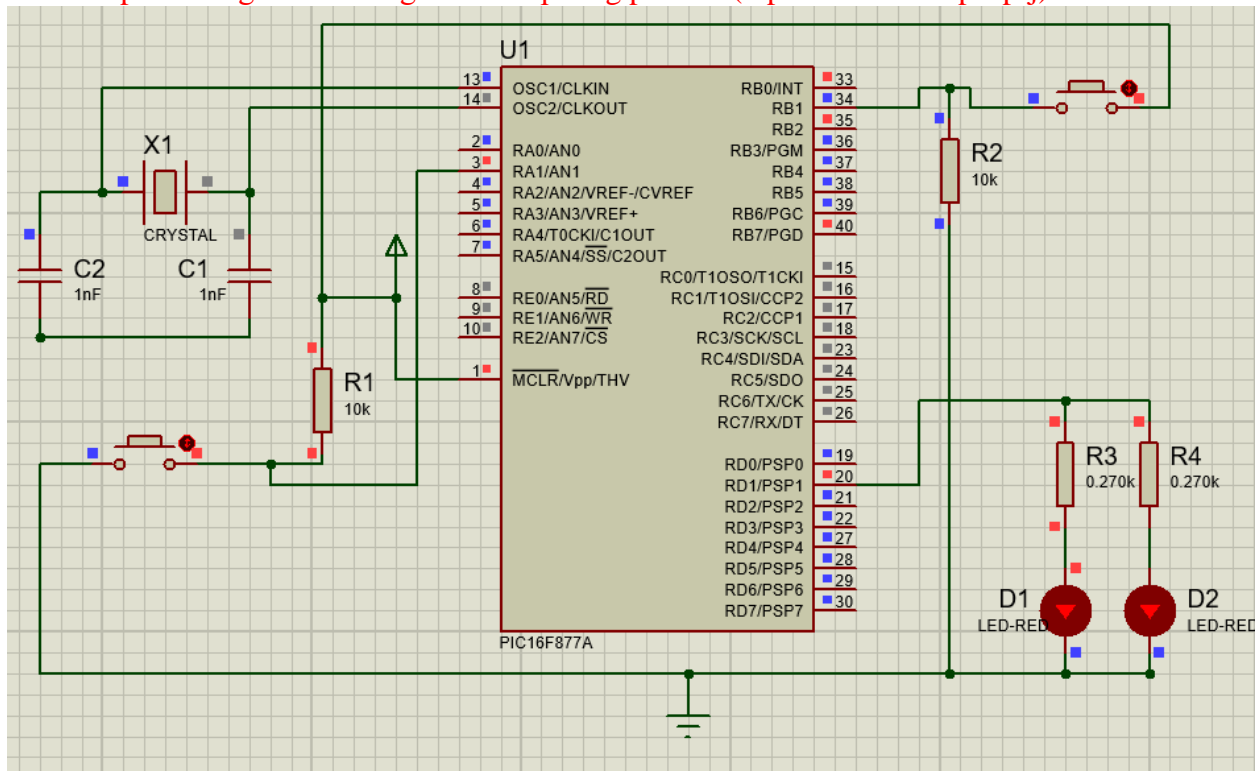


+ Tích cực mức cao.



Giả sử bài toán đặt ra là: Thiết kế mạch điều khiển đèn sử dụng nút nhấn. Nếu nút nhấn 1 nhấn thì đèn sáng, muốn tắt đèn thì phải nhấn nút nhấn 2.

Thiết kế phần cứng như chương trình mô phỏng proteus (mpbai7nutbam1.pdsprj)



Chương trình XC8 (bai7nutbam)

/*

* File: bai7nutbam1.c

* Author: tiemnv

*

* Created on January 1, 2017, 3:09 PM

*/

//

#include <stdio.h>

#include <stdlib.h>

#include <xc.h>

#define _XTAL_FREQ 8000000

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

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

//

// 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)
//-----
void main() {
    TRISA = 0X02; // RA1 is input
    TRISB=0x02; //RB1 is input
    TRISD = 0X00; // PORTD is output
    PORTD = 0X00;
    ADCON1 = 0x07; // off ADC
    while(1)
    {
        if(RA1==0) //or used if(PORTAbits.RA1==0)
        {
            __delay_ms(100);
            if(RA1==0) //or used if(PORTAbits.RA1==0)
            {
                RD1=1; // or used PORTDbits.RD1=1;
            }
        }
        if(RB1==1) // or used if(PORTBbits.RB1==1)
        __delay_ms(100);
        if(RB1==1) // or used if(PORTBbits.RB1==1)
        {
            RD1=0; // or used PORTDbits.RD1=0;
        }
    }
}
//-----

```

Tiến hành mô phỏng:

Khi nhấn button1 lần đầu tiên (đầu vào RA1) thì cả 2 đèn đều sáng, khi muốn tắt 2 đèn thì phải nhấn button2 (đầu vào RB1).

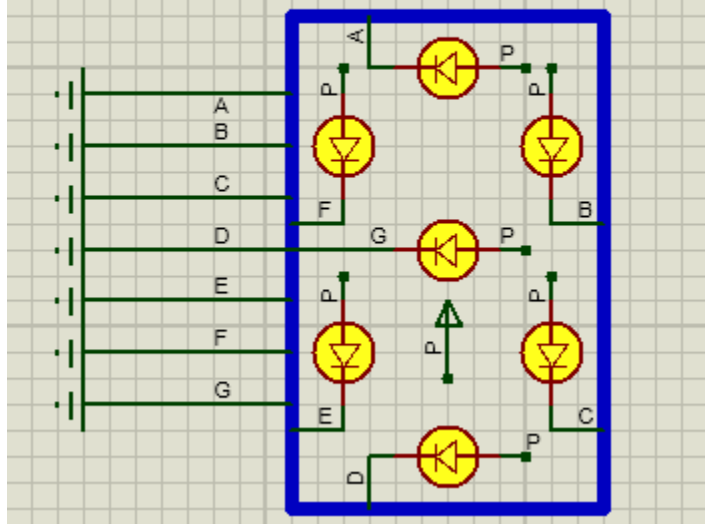
BÀI 5 : HIỂN THỊ SỐ 2015 LÊN LED 7 ĐOẠN PIC16F877A XC8

1.Câu tạo

Thì nó cũng giống LED đơn thôi cũng có 2 loại là Anode chung và Cathode chung.

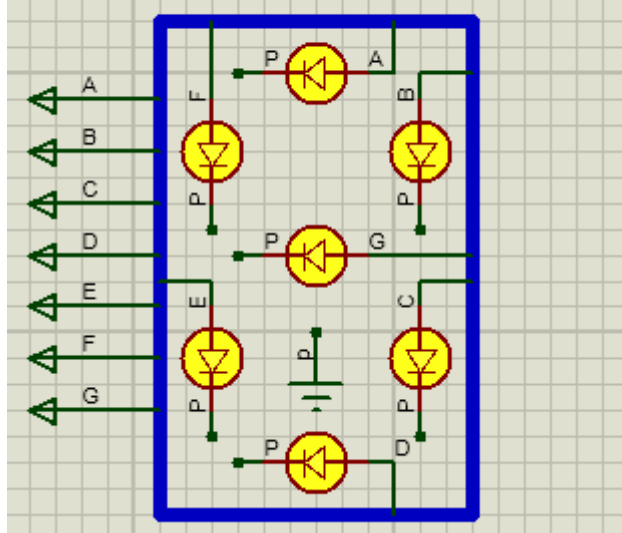
- Anode chung.

Anode Chung



Đối với LED Anode chung thì khi chúng ta cấp nguồn âm vào thì nó sẽ sáng như trên hình.
- Cathode chung.

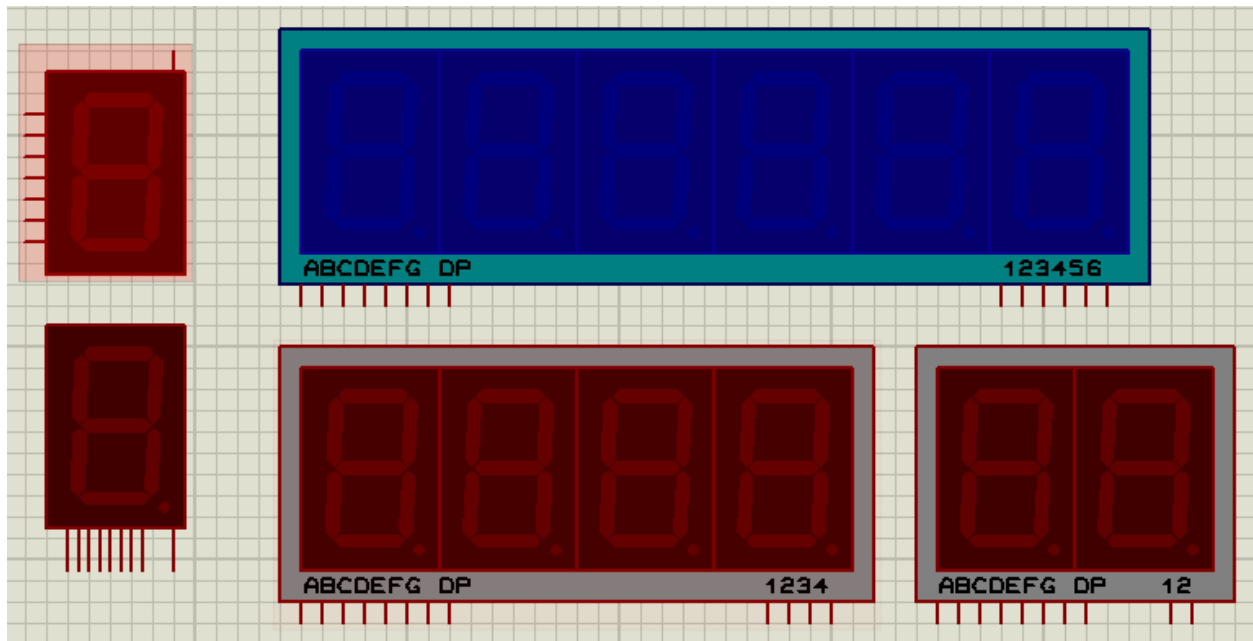
Cathode Chung



Đối với LED Cathode chung thì khi chúng ta cấp nguồn dương cho nó thì nó sẽ sáng như trên hình.

Thực chất nó được cấu tạo từ các LED đơn và được nối chung các chân với nhau giống như hình trên và đưa ra 8 chân cho chúng ta điều khiển là A, B, C, D, E, F, G, H. H là dấu chấm của LED nên chúng ta không cần quan tâm nhiều đến nó. Các chân còn lại là chân dùng để điều khiển LED đối với LED Anode chung thì khi muốn nó sáng thì ta xuất mức 1 cho nó còn LED Cathode chung thì ta xuất mức 0.

- Đây là hình dạng của nó trên protues.



Phần mềm này hỗ trợ cho chúng ta rất nhiều loại LED khác nhau và màu sắc cũng khác nhau các bạn có thể chọn LED phù hợp với sở thích của mình. Nó cũng chia xa làm 2 loại là Anode chung và Cathode chung.

- Còn dưới đây là hình ảnh ngoài thực tế của nó.



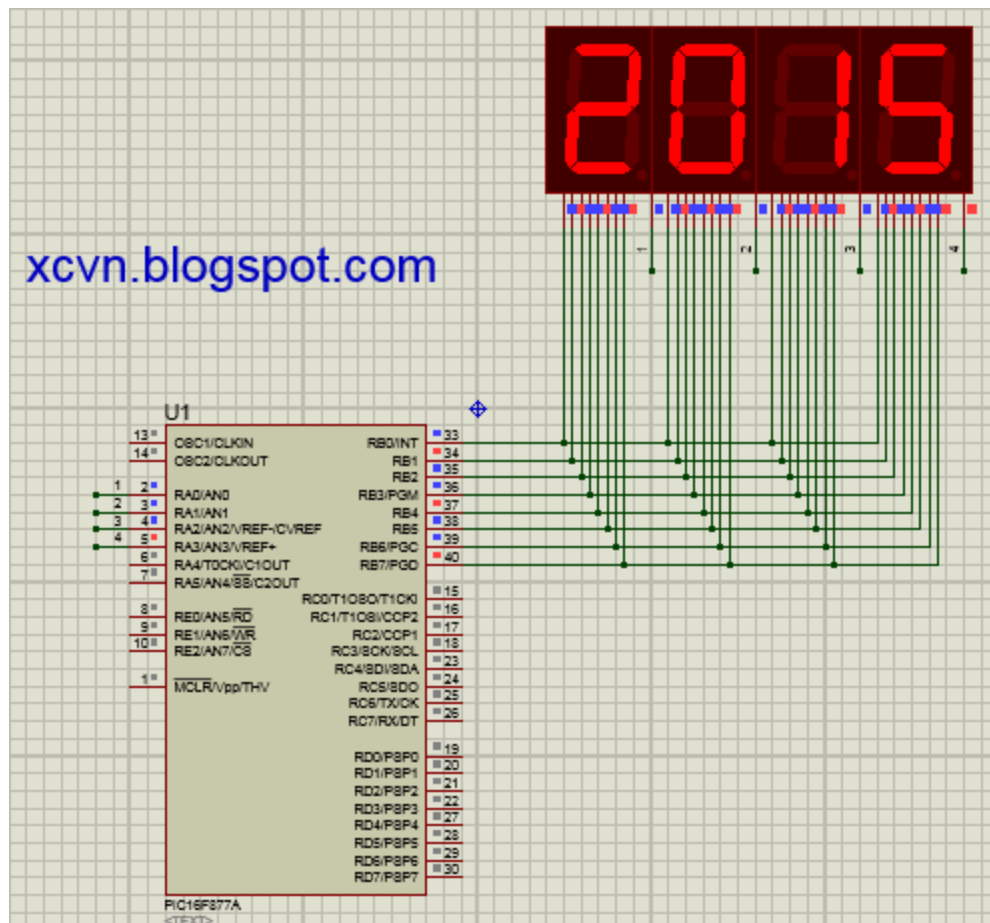
Ứng dụng : Được sử dụng rộng rãi trong đời sống như là : Đồng hồ, Đèn giao thông.....

Ưu điểm : nhỏ gọn, dễ sử dụng, giá thành rẻ, lâu hư....

Nhược điểm : Chỉ hiển thị được số và 1 vài ký tự....

Trong bài này mình sẽ hướng dẫn các bạn hiển thị số 2015 lên 4 LED 7SEG.

- Đây là ảnh mô phỏng protues.



- Đây là code chương trình.

```
#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#define _XTAL_FREQ 8000000
// 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)
const unsigned char Anode[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F,
0x6F};
const unsigned char Cathode[] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80,
0x90};
void main(void)
{
    TRISB = 0X00;
```



```

PORTB = 0X00;
TRISAbits.TRISA0 = 0;
TRISAbits.TRISA1 = 0;
TRISAbits.TRISA2 = 0;
TRISAbits.TRISA3 = 0;
while(1)
{
    // xuất số 2 lên led 7seg
    PORTB = 0XA4;
    PORTAbits.RA0 = 1;
    __delay_ms(4);
    PORTAbits.RA0 = 0;
    // xuất số 0 lên led 7 seg
    PORTB = 0XC0;
    PORTAbits.RA1 = 1;
    __delay_ms(4);
    PORTAbits.RA1 = 0;
    // xuất số 1 lên led 7seg
    PORTB = 0XF9;
    PORTAbits.RA2 = 1;
    __delay_ms(4);
    PORTAbits.RA2 = 0;
    // xuất số 5 lên led 7 seg
    PORTB = 0X92;
    PORTAbits.RA3 = 1;
    __delay_ms(4);
    PORTAbits.RA3 = 0;
}
}

```