MASTERs 2014

LAB Manual for 18022 XC8

My Class Title Here

Table of Contents

Tools needed for these labs:

PICDEM 2 + with PIC16F1937 ICD3 or Real ICE
9 Volt Power Supply
USB Cable
or
The Proteus VSM simulator can be used to simulate all board function in these labs.

MICROCHIP

MASTERs Conference

LAB 1:
Creating an MPLAB® X C Based Project

Purpose:

The purpose of this lab is to illustrate the steps required to create an MPLAB® X C based project within the MPLAB Integrated Development Environment. You will learn how to select the compiler as the build tool, which files must be included in your project, how to allocate a heap and what code must be included in your source file.

Objective

Create an MPLAB X project for a C program from scratch using the provided source files.

If you want a bigger challenge or you already feel you know this material well enough, then do not continue past this page.

Challenging Lab:

Do not use any of the predefined lab projects or lab manual. Fulfill the above objective on your own.

Procedure

Information

When MPLAB X starts it opens the last project worked on. To avoid confusion this lab asks to you close any open project.

1 Project Creation

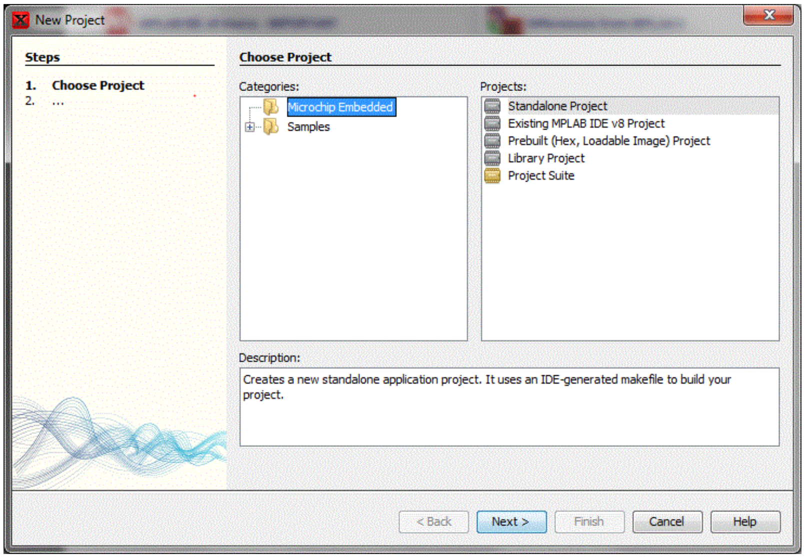
- Open MPLAB X
- Close any open projects in MPLAB X by right clicking on the Project name and selecting “Close”**.**

2 Start Project Creation

- Click the “New Project” icon to start the project creation process

3

- Select “Microchip Embedded” then “Standalone project”**.**



New Project

Steps

1. **Choose Project**
2. ...

Choose Project

Categories:

- Microchip Embedded
- Samples

Projects:

- Standalone Project
- Existing MPLAB IDE v8 Project
- Prebuilt (Hex, Loadable Image) Project
- Library Project
- Project Suite

Description:

Creates a new standalone application project. It uses an IDE-generated makefile to build your project.

Figure 1.1
MPLAB X project creation screen

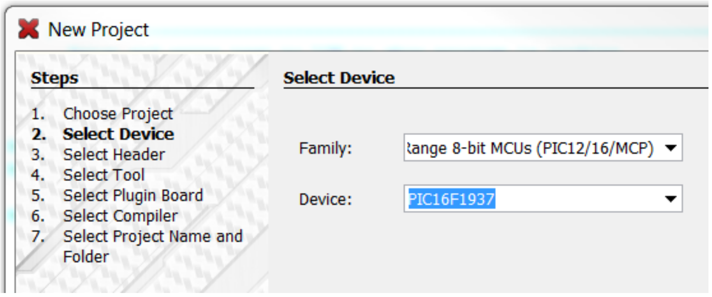
- Select Next **Next >**

Information

We are using the **PIC16F1937** as the microcontroller for this lab.

4 Select the Processor

- Select “Mid-Range 8-bit MCUs” from the ‘Family’ pull down menu then select PIC16F1937 from the “Device” menu.



New Project

Steps

1. Choose Project
2. **Select Device**
3. Select Header
4. Select Tool
5. Select Plugin Board
6. Select Compiler
7. Select Project Name and Folder

Select Device

Family: ʀange 8-bit MCUs (PIC12/16/MCP)

Device: PIC16F1937

Figure 1.2
Selecting the MCU used in a project

Next >

5 Header Selection

- No headers are needed. Click “Next” Next >

Information

Debug Headers are only required for some of the 8 bit controllers with very low pin counts. If you are using an 8-bit, 16-bit, or 32-bit with more than 14 pins a header may not be needed

6 Select the Hardware Tool

- Select “Proteus VSM Viewer” under Hardware tools when asked to select a tool. (You can choose a different debugging hardware option later)

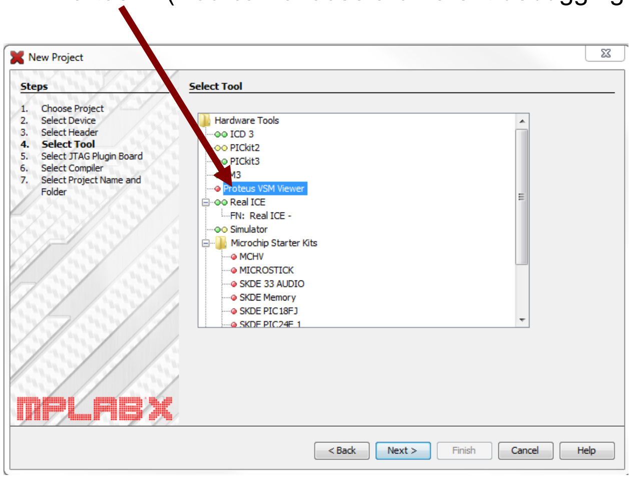


Figure 1.3
Selecting Proteus VSM Viewer as the Hardware tool for a pro-

- Click Next Next >

Tools

- MPLAB XC8 version 1.20 or higher is advised

7 Select the Compiler

- Open the XC8 plus box, if not already open, and select an installed version of the XC8 compiler

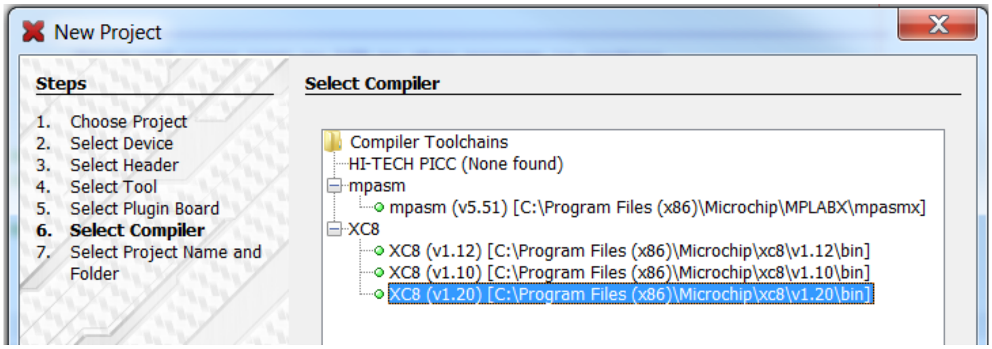
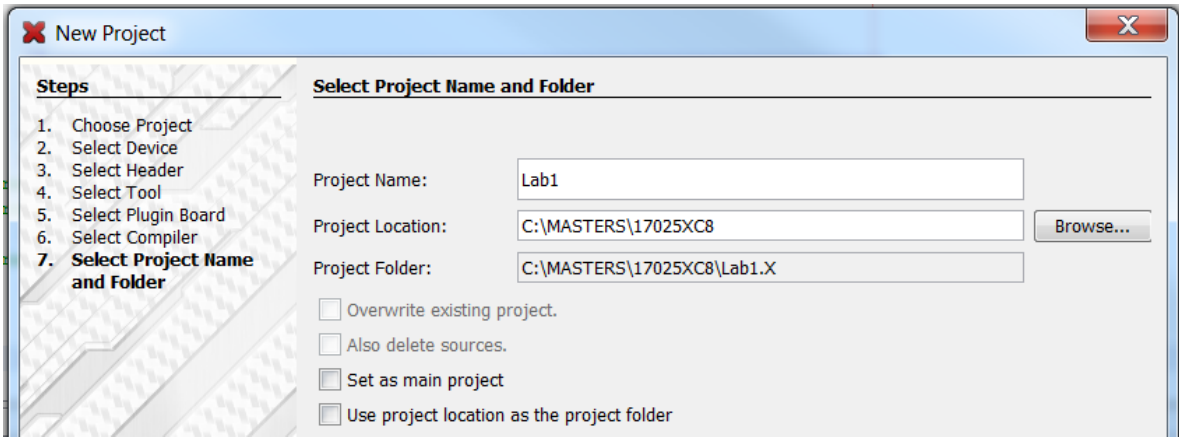

New Project
Steps
1. Choose Project
2. Select Device
3. Select Header
4. Select Tool
5. Select Plugin Board
6. Select Compiler
7. Select Project Name and Folder
Select Compiler
Compiler Toolchains
HI-TECH PICC (None found)
mpasm
mpasm (v5.51) [C:\Program Files (x86)\Microchip\MPLABX\mpasmx]
XC8
XC8 (v1.12) [C:\Program Files (x86)\Microchip\xc8\v1.12\bin]
XC8 (v1.10) [C:\Program Files (x86)\Microchip\xc8\v1.10\bin]
XC8 (v1.20) [C:\Program Files (x86)\Microchip\xc8\v1.20\bin]

Figure 1.4
Selecting the Compiler

- Click Next >

8 Select Project Name and Folder

- Click the “Browse” button and navigate to the folder C:\MASTERS\18022
- On the line marked “Project Name” type in the project name as Lab1
- Notice MPLAB X filling in the Project Folder line with C:\MASTERS\18022\Lab1.X



New Project

Steps

1. Choose Project
2. Select Device
3. Select Header
4. Select Tool
5. Select Plugin Board
6. Select Compiler
7. **Select Project Name and Folder**

Select Project Name and Folder

Project Name: Lab1

Project Location: C:\MASTERS\17025XC8 Browse...

Project Folder: C:\MASTERS\17025XC8\Lab1.X

□ Overwrite existing project.

□ Also delete sources.

□ Set as main project

□ Use project location as the project folder

Figure 1.5
Naming a Project

- Click “Finish” Finish

Congratulations ! You have just created an MPLAB X project

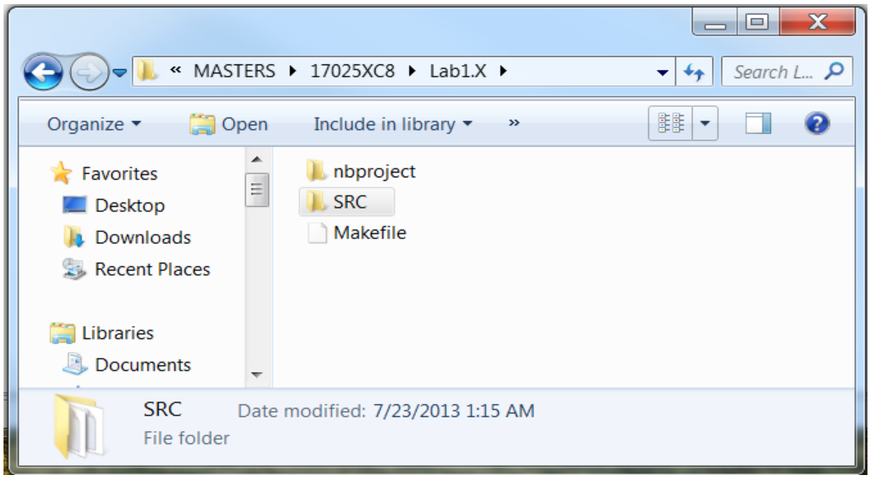
We will now add source code files to the project.

Information

It is **not** necessary to add files to a project folder before adding them to a project. Although, putting all the files into the project folder can make backing up a project easy.

9 Move Source Files into the Project Folder

- If you used the same directories and Project name we gave you, then the three (3) required source files are already in the project folder.
- After this step the Lab1.X folder should contain the subfolders nbproject and SRC (as illustrated below)



« MASTERS ▸ 17025XC8 ▸ Lab1.X ▸

Search L...

nbproject

SRC

Makefile

SRC Date modified: 7/23/2013 1:15 AM

File folder

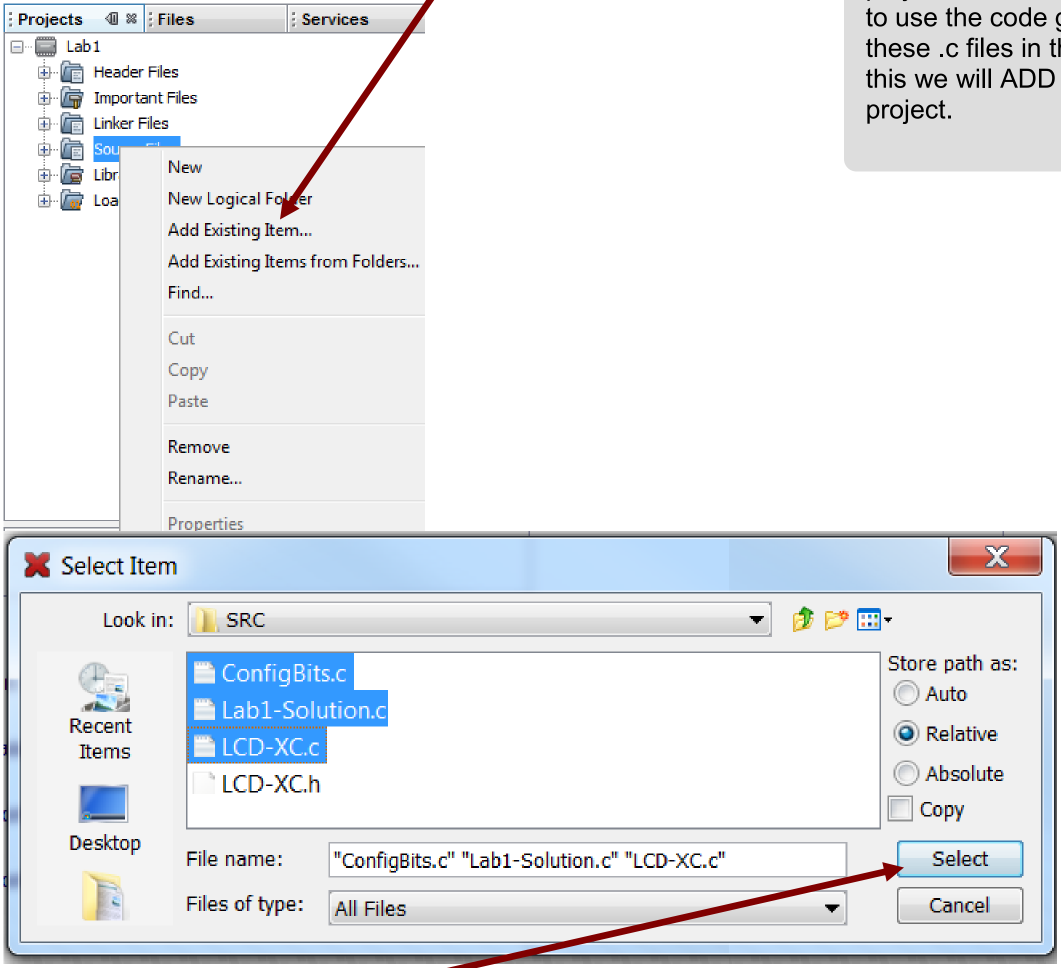
Figure 1.6
View of project folder after the source file Lab1.c has been added

10 Add Source Files to the Project

- Right click on the **Source Files** folder in the project window
- Select **Add Existing Item (open the SRC folder)**

Information

The three files needed for this project are in the folder but not in the project. MPLAB X must be informed to use the code generated by these .c files in the project. To do this we will ADD these files to the project.



Look in: SRC

Store path as: ○ Auto ● Relative ○ Absolute □ Copy

File name: "ConfigBits.c" "Lab1-Solution.c" "LCD-XC.c"

Files of type: All Files

Figure 1.7
Adding existing files to an MPLAB X project

- Highlight the LCD-XC.c, Lab1.c and ConfigBits.c files in the folder **C:\MASTERS\18022\Lab1.X\**
- Ensure the radio button labeled "Relative" in the lower right border of the dialog box is checked
- Click "Select"

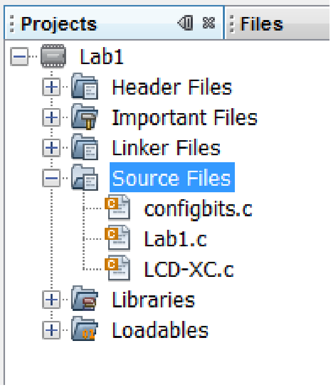


Figure 1.8
Project window after source files have been added to a project

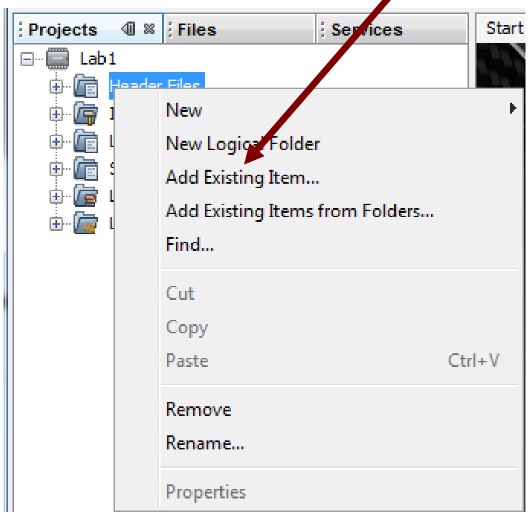
11 Add Header Files to the Project

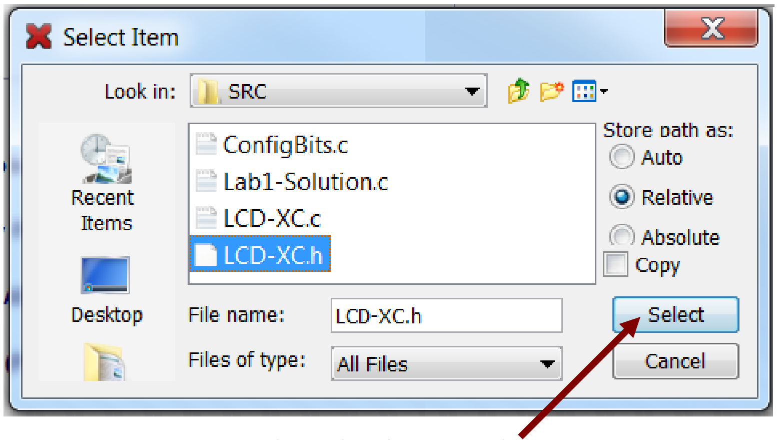
- Right click on the **Header Files** folder in the project window
- Select **Add Existing Item**

Information

Adding header files to the project tree is **optional.** This provides quick access to them from within your project and will enable the IDE to see when a change has been made to a header file. Being aware of header file changes is important when "Building" a project. A "Clean and Build" recompiles all included files automatically.

All header files for the project must be included via the `#include` directive in source files.





Look in: SRC

Store path as: ● Relative

File name: LCD-XC.h

Files of type: All Files

Figure 1.9
Adding an existing file to an MPLAB X project

- Highlight the **LCD-XC.h** file in the folder `C:\MASTERS\18022\Lab1.X\SRC`
- Ensure the radio button labeled "Relative" in the lower right border of the dialog box is checked
- Click "Select"

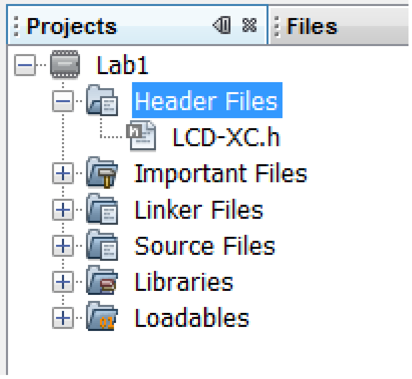
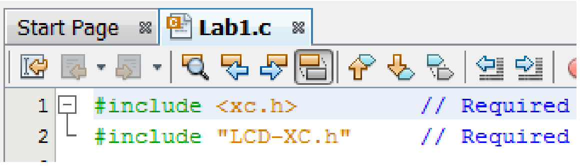


Figure 1.10
Project window after header file has been added to a project

12 Verify header files included in source code

Open “Lab1.c” by double-clicking on it in the “Projects” window.

Verify the generic MPLAB XC8 (xc.h) and “LCD-XC.h” header files have been included in the source code.



```
#include <xc.h>          // Required
#include "LCD-XC.h"      // Required
```

Header files must be included in your source code with **#include**. Adding them to the project tree does not include them in the build process. This is an ANSI C requirement – not an MPLAB X® requirement.

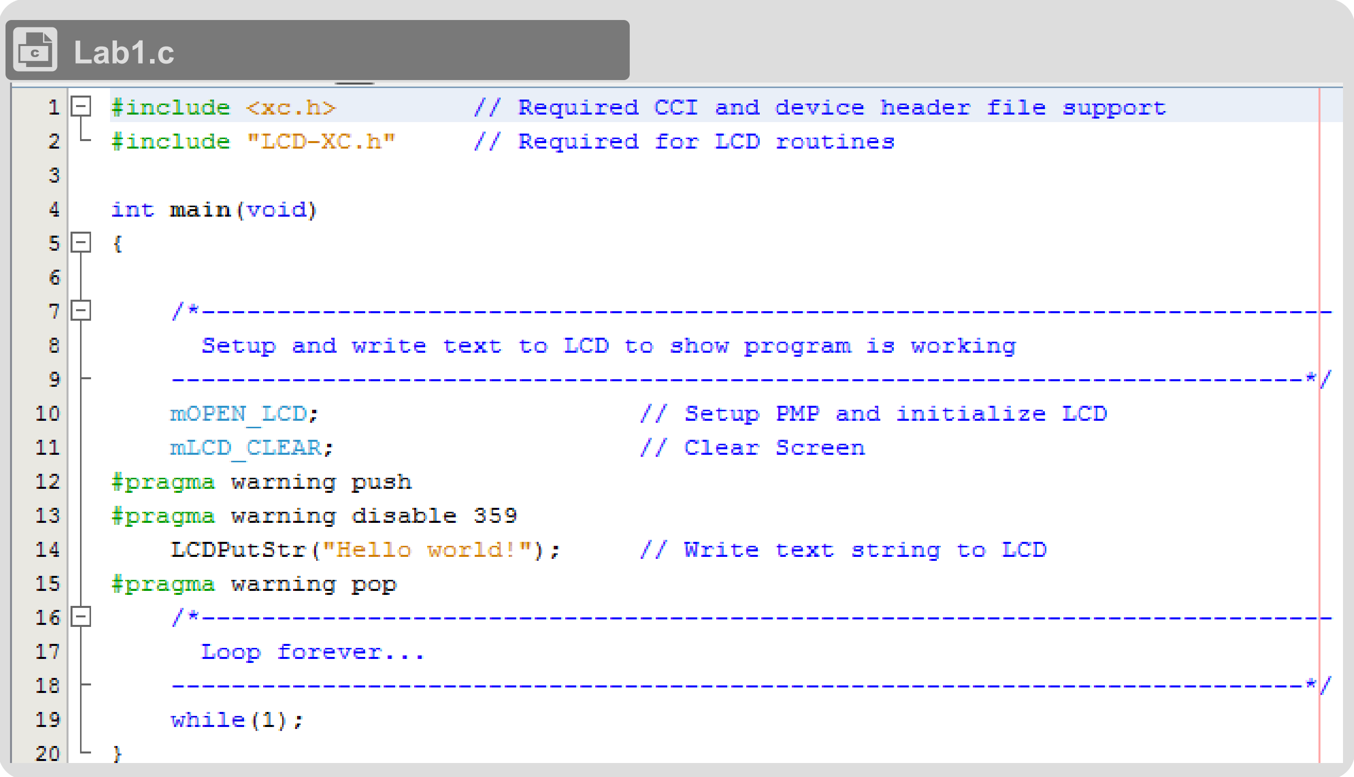
Information

`#include <file>`
Angle brackets indicate that a file exists within the MPLAB XC search path (at or below the MPLAB XC directory). If the file is not found in the compiler’s search path, it will result in a compile error.

`#include “file”`
`#include “C:\...\file”`
Quotes indicate that a file exists within the project directory, unless a fully qualified path is provided. In that case, the compiler will look for the file at the specified location. If the file is not found in the project directory or in the specified location, it will result in a compile error.

13 Finished (with the setup)

The source code below is the program that we will run on the Explorer 16 board in just a moment:



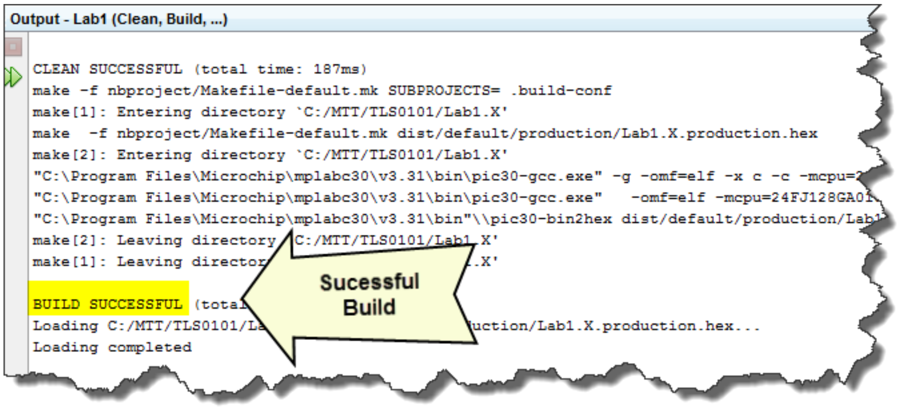
```
#include <xc.h>         // Required CCI and device header file support
#include "LCD-XC.h"     // Required for LCD routines

int main(void)
{

    /*----------------------------------------------------------------------------
      Setup and write text to LCD to show program is working
    ---------------------------------------------------------------------------*/
    mOPEN_LCD;                     // Setup PMP and initialize LCD
    mLCD_CLEAR;                    // Clear Screen
#pragma warning push
#pragma warning disable 359
    LCDPutStr("Hello world!");     // Write text string to LCD
#pragma warning pop
    /*----------------------------------------------------------------------------
      Loop forever...
    ---------------------------------------------------------------------------*/
    while(1);
}
```

There are two macros and a function called within our main program before we go into an infinite loop. First is `mOPEN_LCD` and `mLCD_CLEAR` which sets up the LCD module on the PICDIM 2 PLUS board. Next is `LCDPutStr()` which will write a string of text out to the LCD module at the current cursor position.

The #pragma statements suppress warning messages. These were discussed in class.

14 Build and Debug the Project

- Click the "**Debug and Run**" icon to build and run the project



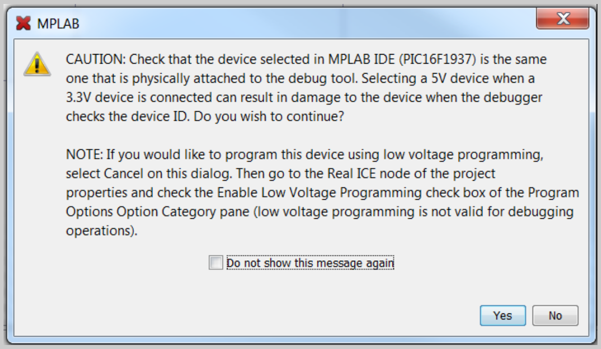
Output - Lab1 (Clean, Build, ...)

```
CLEAN SUCCESSFUL (total time: 187ms)
make -f nbproject/Makefile-default.mk SUBPROJECTS= .build-conf
make[1]: Entering directory `C:/MTT/TLS0101/Lab1.X'
make  -f nbproject/Makefile-default.mk dist/default/production/Lab1.X.production.hex
make[2]: Entering directory `C:/MTT/TLS0101/Lab1.X'
"C:\Program Files\Microchip\mplabc30\v3.31\bin\pic30-gcc.exe" -g -omf=elf -x c -c -mcpu=
"C:\Program Files\Microchip\mplabc30\v3.31\bin\pic30-gcc.exe"   -omf=elf -mcpu=24FJ128GA01
"C:\Program Files\Microchip\mplabc30\v3.31\bin"\\pic30-bin2hex dist/default/production/Lab
make[2]: Leaving directory `C:/MTT/TLS0101/Lab1.X'
make[1]: Leaving director                        .X'

BUILD SUCCESSFUL (total
Loading C:/MTT/TLS0101/La                          uction/Lab1.X.production.hex...
Loading completed
```

Figure 1.11
A project output window after a successful build of a project

High Voltage Programming Warning

You may receive the warning (at right) . This is just warning you that the device you are trying to program is 3.3 volts and if you try to program as a 5 volt device it can be damaged. In these classes we have preselected 3.3 volt programming so this waning can be ignored. You can safely select yes and move on.



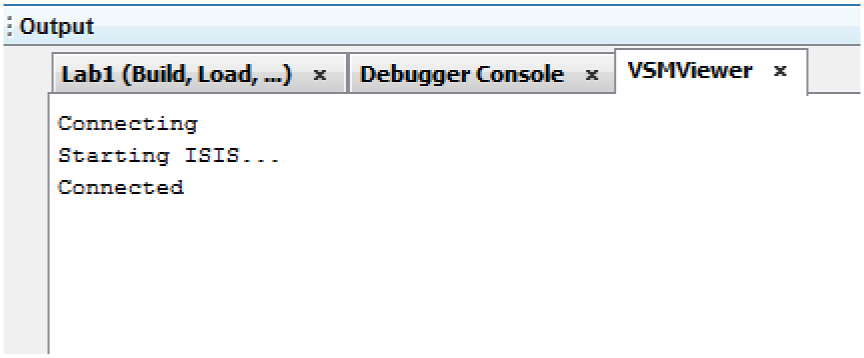
MPLAB

CAUTION: Check that the device selected in MPLAB IDE (PIC16F1937) is the same one that is physically attached to the debug tool. Selecting a 5V device when a 3.3V device is connected can result in damage to the device when the debugger checks the device ID. Do you wish to continue?

NOTE: If you would like to program this device using low voltage programming, select Cancel on this dialog. Then go to the Real ICE node of the project properties and check the Enable Low Voltage Programming check box of the Program Options Option Category pane (low voltage programming is not valid for debugging operations).

□ Do not show this message again

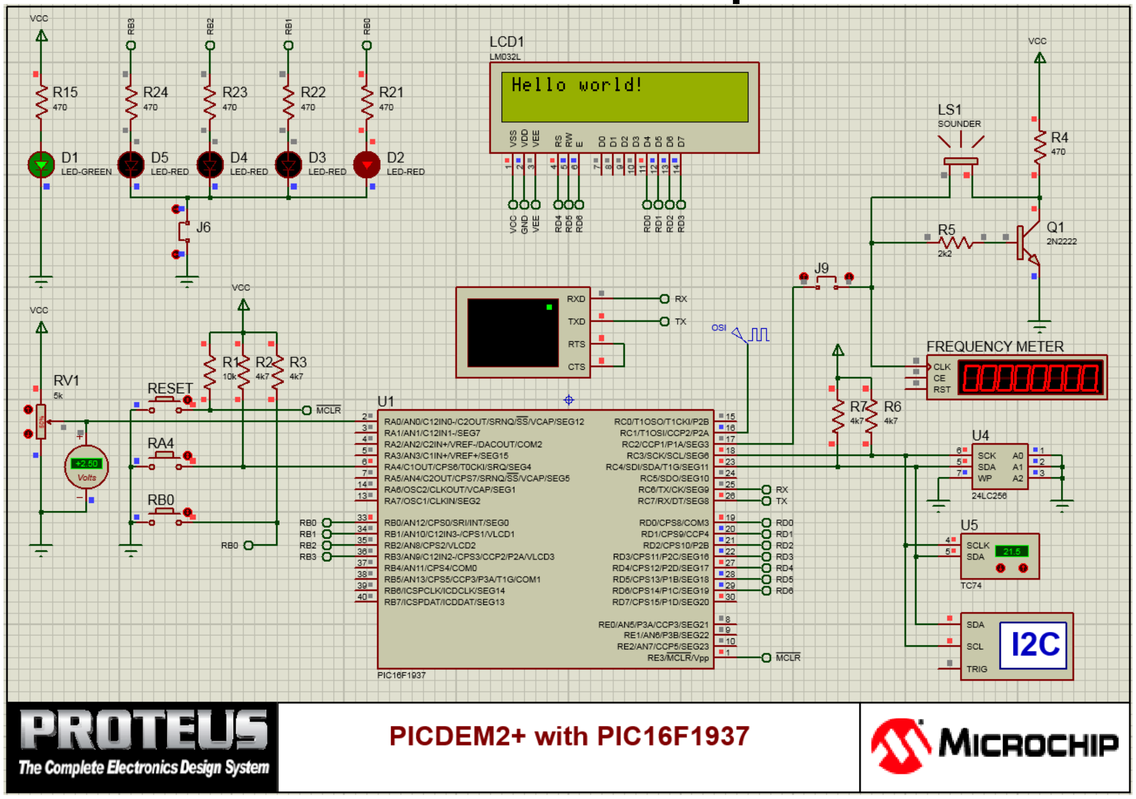
- Once the project builds correctly MPLAB X will automatically:
  - Launch the selected debugger (in this case the Proteus VSM simulator)
  - Program your code into the target device
  - Run your code.



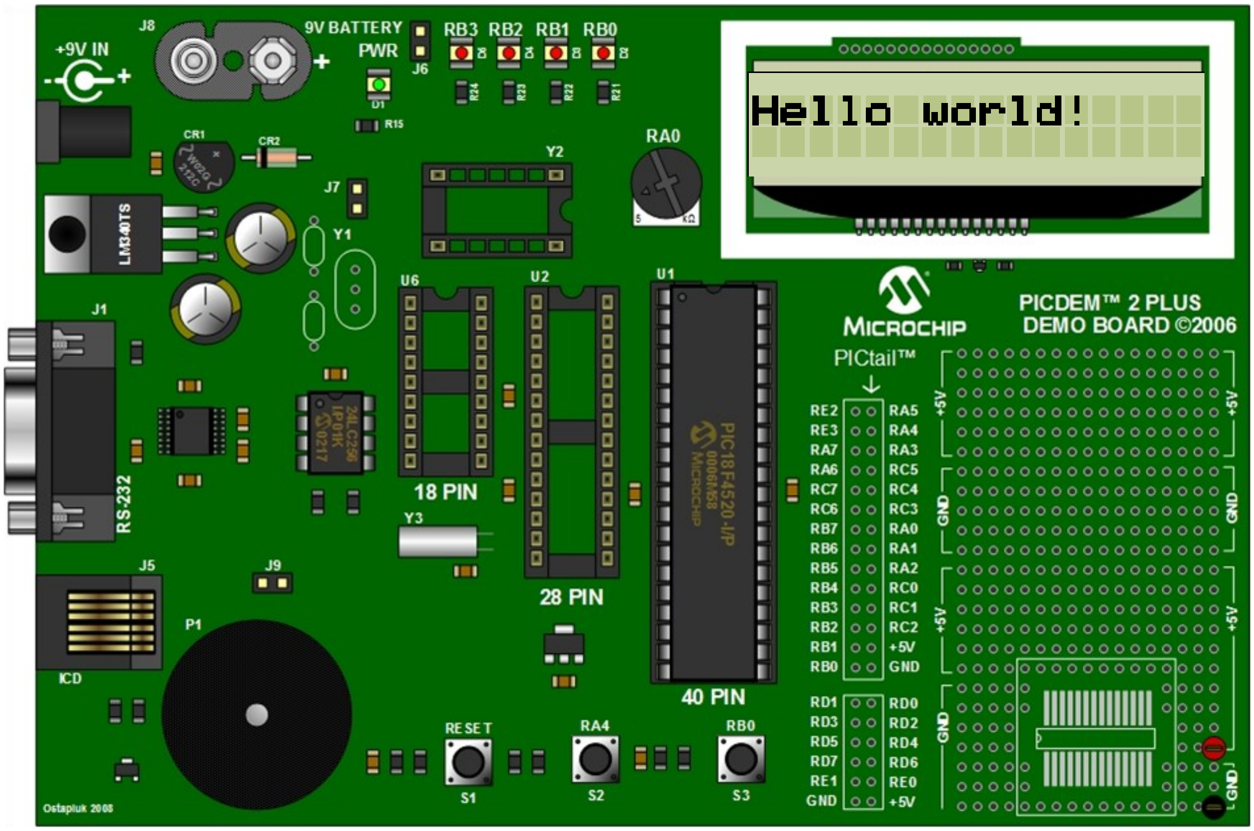
```
Connecting
Starting ISIS...
Connected
```

- The results of the program are list on the next page.

Proteus VSM simulator output



PICDEM 2 PLUS hardware out-



LAB 2: Setting PIC® Configuration Bits in Code C Based Project

Purpose:

The purpose of this lab is to test your understanding of the use of the #pragma to set-up the device configuration bits in code, based on a given set of desired configuration options.

Objective

Based on what you learned in class, setup the PIC® core configuration words in code with the following set of configuration options by using the configuration memory window.

- **Oscillator = *INT OSC***
- **Low Voltage Programming = Off**
- **Powerup Timer = Disabled**
- ***Brownout Reset = Disabled***
- **Watchdog Timer = *Disabled***

If you want a bigger challenge or you already feel you know this material well enough, then do not continue past this page.

Challenging Lab:

Do not use any of the predefined lab projects or lab manual. Fulfill the above objective on your own.

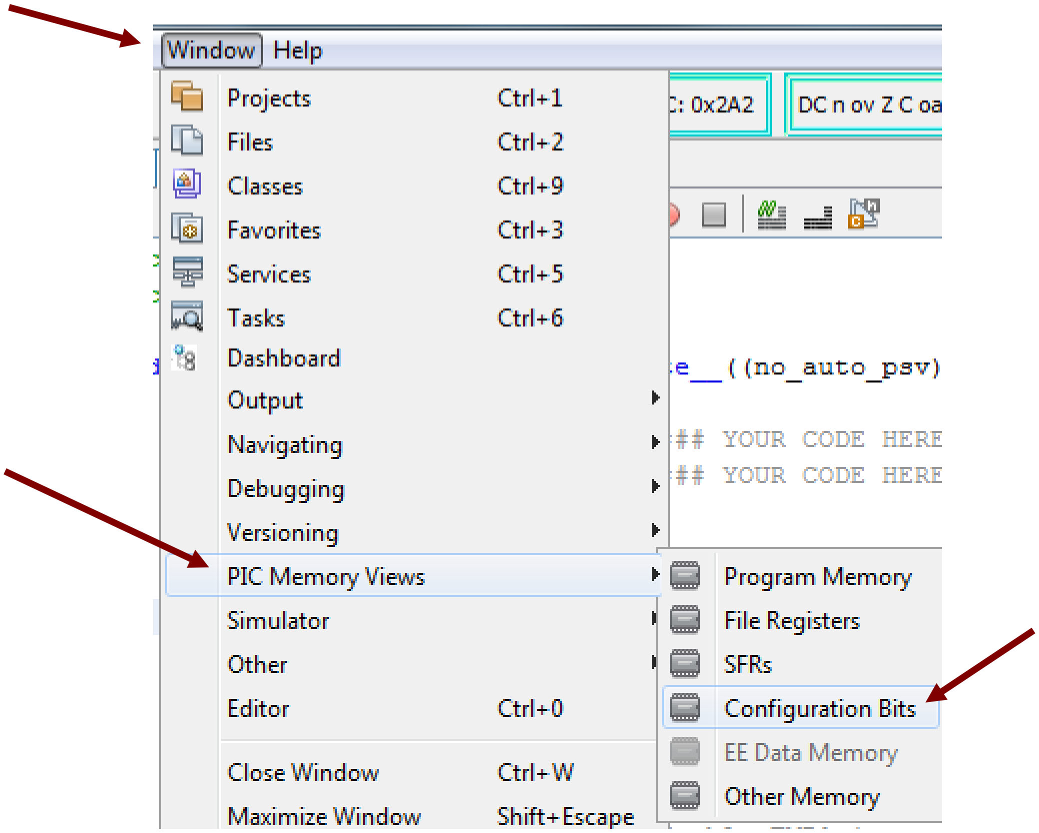
Procedure

1 Open Lab2 Project

- Close any previously opened projects by right clicking on the project name and selecting “Close”
- Click the **Open Project** icon
- Browse to **C:\MASTERS\18022\Lab2.X** and select **“Open Project”**

2 Open the “Configuration Bits” widow

In MPLAB X.



Information

#pragma can be used to setup the CONFIG1 and CONFIG2 registers manually, but the Config Bits window is an easier more intuitive way. For completeness here is the syntax that will be used by the compiler. Read through the syntax and parameters shown in the following tables.

```
#pragma config setting = state|value
```

Single or multiple settings can be included on one #pragma. Each additional setting will be separated by commas. The settings can be made in any order regardless of which CONFIG word the setting is in. Any parameters you leave out will retain their default settings as per the data sheet.

Table 2.1

FOSC	`FOSC = ECH`	External Clock, High Power Mode
	`FOSC = ECM`	External Clock, Medium Power Mode
	`FOSC = ECL`	External Clock, Low Power Mode
	`FOSC = INTOSC`	Internal Oscillator Mode, I/o function on CLKIN pin
	`FOSC = EXTRC`	External RC oscillator on CLKIN pin
	`FOSC = HS`	High-speed crystal oscillator
	`FOSC = XT`	External crystal oscillator
	`FOSC = LP`	Low-power crystal oscillator
Watchdog Timer	`WDTE = OFF`	Disabled
	`WDTE = ON`	Enabled
	`WDTE = NSLEEP`	Disabled in SLEEP mode
	`WDTE = SWDTEN`	**Controlled by the SWDTEN bit in WDTCON**
Power-up Timer	`PWRTE = ON`	ENabled
	`PWRTE= OFF`	Disabled
MCLR pin function	`MCLRE = OFF`	MCLR/Vpp pin function is digital input
	`MCLRE = ON`	MCLR/Vpp pin function is MCLR
FLASH Code Protect	`CP = ON`	Enabled
	`CP = OFF`	Disabled
Data Memory Protect	`CPD = ON`	Enabled
	`CPD = OFF`	Disabled
Brown-out Reset	`BOREN = ON`	Enabled
	`BOREN = OFF`	Disabled
	`BOREN = NSLEEP`	Disabled in SLEEP mode
	`BOREN = SBOREN`	**Controlled by the SBOREN bit in BORCON**
Clock Out Enable	`CLKOUTEN = ON`	Enabled
	`CLKOUTEN = OFF`	Disabled
Internal/External Switchover Mode	`IESO = ON`	Enabled
	`IESO = OFF`	Disabled
Fail Safe Clock Monitoring	`FCMEN = ON`	Enabled
	`FCMEN = OFF`	Disabled

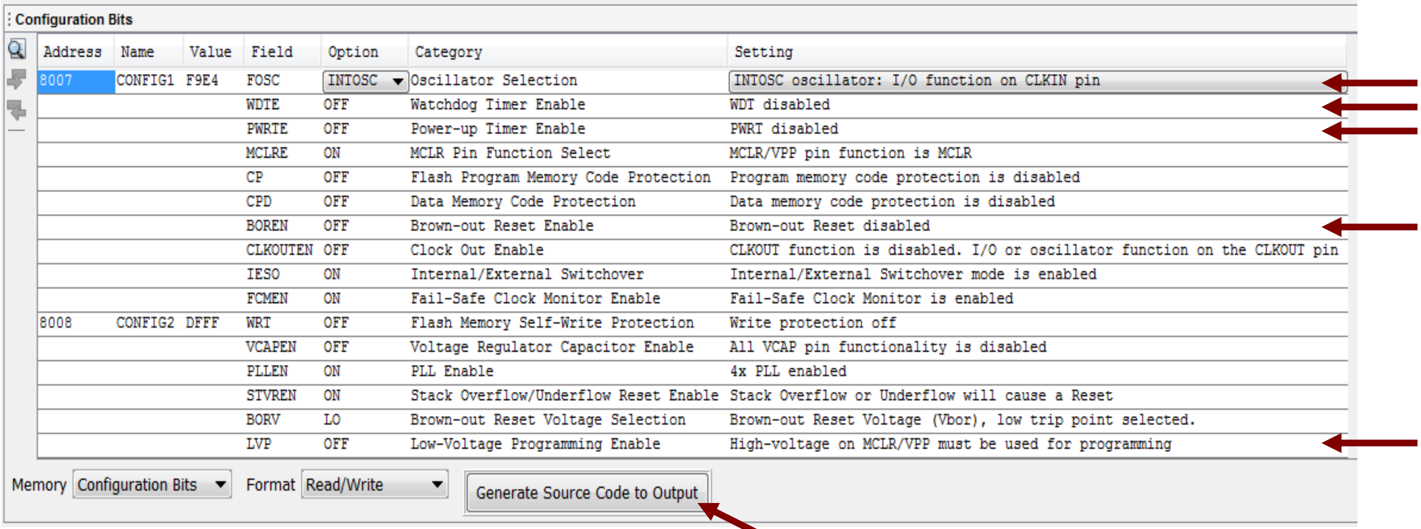
Table 2.2

FLASH Memory Self Write Protect	WRT = OFF	**Write Protection is OFF**
	WRT = BOOT	000h—1FFh write protected
	WRT = HALF	000h—FFFh write protected
	WRT = ALL	000h—1FFFh write protected (All Program Memory)
Watchdog Timer	VCAPEN = OFF	All VCAP pin function disabled
	VCAPEN = RA6	VCAP function on RA6
	VCAPEN = RA5	VCAP function on RA5
	VCAPEN = RA0	VCAP function on RA0
PLL Enable	PLLEN = ON	4x PLL Enabled
	PLLEN = OFF	**4x PLL Disabled**
Stack Overflow Reset	STVREN = OFF	Stack Over/Under flow will not cause Reset
	STVREN = ON	Stack Over/Under flow will cause Reset
Brown-out Voltage Select	BORV = HI	High trip point selected
	BORV = LO	Low trip point selected
Low Voltage Programming	LVP = ON	Enabled
	LVP = OFF	Disabled

Information

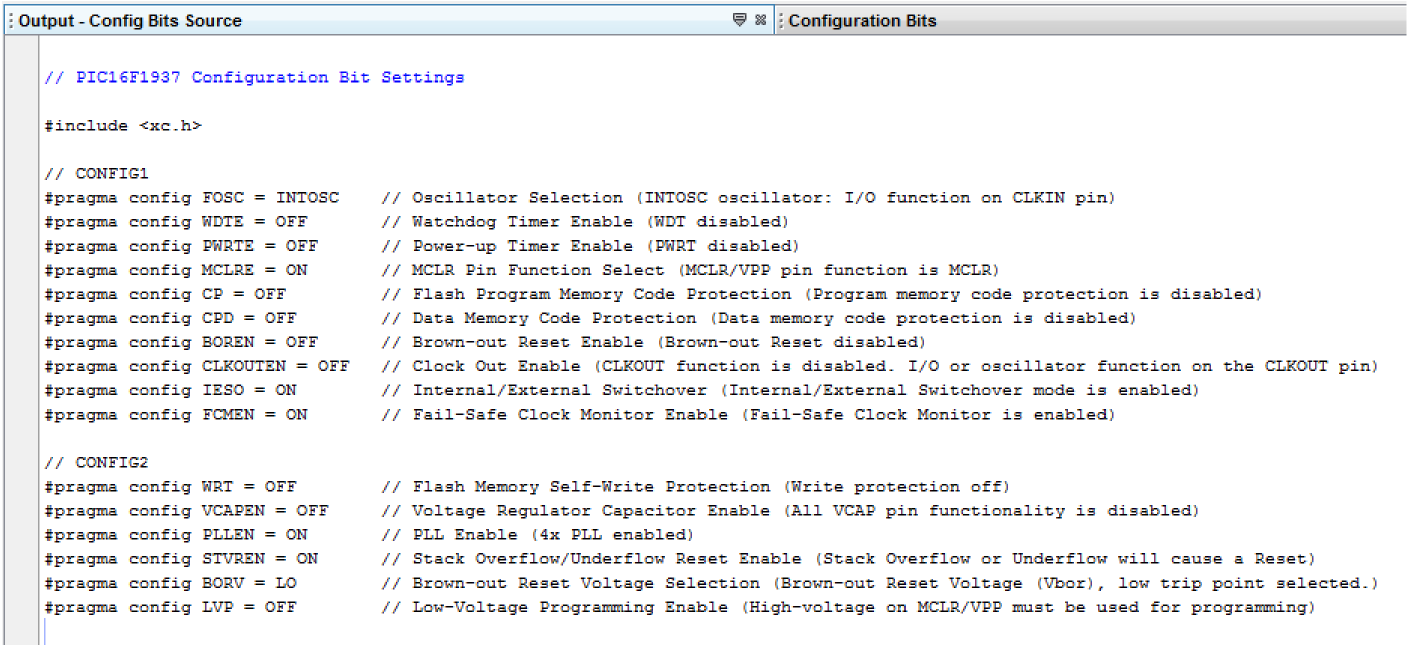
Oscillator = *INT OSC*
Low Voltage Programming = Off
Powerup Timer = Disabled
Brownout Reset = Disabled
Watchdog Timer = *Disabled*

3 Set the Configuration Bit settings (at right). Using the pull down menus.



Address	Name	Value	Field	Option	Category	Setting
8007	CONFIG1	F9E4	FOSC	INTOSC	Oscillator Selection	INTOSC oscillator: I/O function on CLKIN pin
			WDTE	OFF	Watchdog Timer Enable	WDT disabled
			PWRTE	OFF	Power-up Timer Enable	PWRT disabled
			MCLRE	ON	MCLR Pin Function Select	MCLR/VPP pin function is MCLR
			CP	OFF	Flash Program Memory Code Protection	Program memory code protection is disabled
			CPD	OFF	Data Memory Code Protection	Data memory code protection is disabled
			BOREN	OFF	Brown-out Reset Enable	Brown-out Reset disabled
			CLKOUTEN	OFF	Clock Out Enable	CLKOUT function is disabled. I/O or oscillator function on the CLKOUT pin
			IESO	ON	Internal/External Switchover	Internal/External Switchover mode is enabled
			FCMEN	ON	Fail-Safe Clock Monitor Enable	Fail-Safe Clock Monitor is enabled
8008	CONFIG2	DFFF	WRT	OFF	Flash Memory Self-Write Protection	Write protection off
			VCAPEN	OFF	Voltage Regulator Capacitor Enable	All VCAP pin functionality is disabled
			PLLEN	ON	PLL Enable	4x PLL enabled
			STVREN	ON	Stack Overflow/Underflow Reset Enable	Stack Overflow or Underflow will cause a Reset
			BORV	LO	Brown-out Reset Voltage Selection	Brown-out Reset Voltage (Vbor), low trip point selected.
			LVP	OFF	Low-Voltage Programming Enable	High-voltage on MCLR/VPP must be used for programming

4 **Generate Source Code**
MPLAB X will generate the correct source code for the selected device and display it to the output window



```
// PIC16F1937 Configuration Bit Settings

#include <xc.h>

// CONFIG1
#pragma config FOSC = INTOSC    // Oscillator Selection (INTOSC oscillator: I/O function on CLKIN pin)
#pragma config WDTE = OFF       // Watchdog Timer Enable (WDT disabled)
#pragma config PWRTE = OFF      // Power-up Timer Enable (PWRT disabled)
#pragma config MCLRE = ON       // MCLR Pin Function Select (MCLR/VPP pin function is MCLR)
#pragma config CP = OFF         // Flash Program Memory Code Protection (Program memory code protection is disabled)
#pragma config CPD = OFF        // Data Memory Code Protection (Data memory code protection is disabled)
#pragma config BOREN = OFF      // Brown-out Reset Enable (Brown-out Reset disabled)
#pragma config CLKOUTEN = OFF   // Clock Out Enable (CLKOUT function is disabled. I/O or oscillator function on the CLKOUT pin)
#pragma config IESO = ON        // Internal/External Switchover (Internal/External Switchover mode is enabled)
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor Enable (Fail-Safe Clock Monitor is enabled)

// CONFIG2
#pragma config WRT = OFF        // Flash Memory Self-Write Protection (Write protection off)
#pragma config VCAPEN = OFF     // Voltage Regulator Capacitor Enable (All VCAP pin functionality is disabled)
#pragma config PLLEN = ON       // PLL Enable (4x PLL enabled)
#pragma config STVREN = ON      // Stack Overflow/Underflow Reset Enable (Stack Overflow or Underflow will cause a Reset)
#pragma config BORV = LO        // Brown-out Reset Voltage Selection (Brown-out Reset Voltage (Vbor), low trip point selected.)
#pragma config LVP = OFF        // Low-Voltage Programming Enable (High-voltage on MCLR/VPP must be used for programming)
```

5 Right Click on the Source Code Window and select Save As. Browse to the Lab2.X project folder and save the file as:

C:\MASTERS\18022\Lab2.X\ConfigBits.c

6 Right Click on Source Files in the Project tree and add an Existing item :

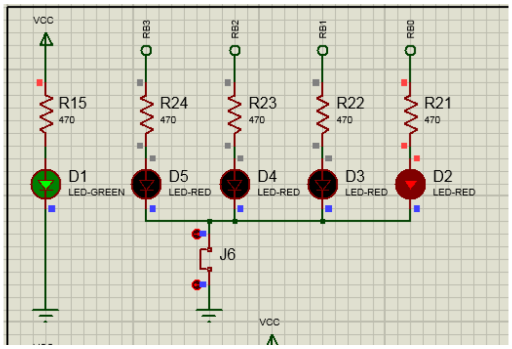
C:\MASTERS\18022\Lab2.X\ConfigBits.c

7 Hit OK, and build the project

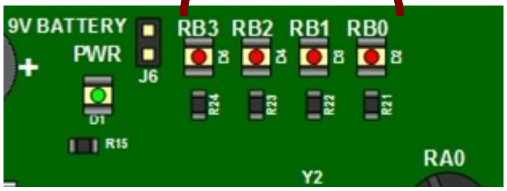
Results:

All four LEDs should be blinking.

Proteus PICDEM 2 +



Hardware PICDEM 2 +



LAB 3:
Managing Memory

Purpose:

The purpose of this lab is to gain insight to the use and allocation of Memory and Objects to control the compilers memory areas allocation.

Objective

While the project will build as is, it is using several different memory areas by using #pragma and keywords to tell the compiler where different variables and constant are to be placed. Here you will experiment with different Memory areas and Objects to change how the compiler allocates the memory space.

If you want a bigger challenge or you already feel you know this material well enough, then do not continue past this page.

Challenging Lab:
Do not use any of the predefined lab projects or lab manual. Fulfill the above objective on your own. By creating a project that places initialized data in each of these areas.

EEPROM, FLASH, SRAM, User ID registers.

Procedure

1 Open Lab3 Project

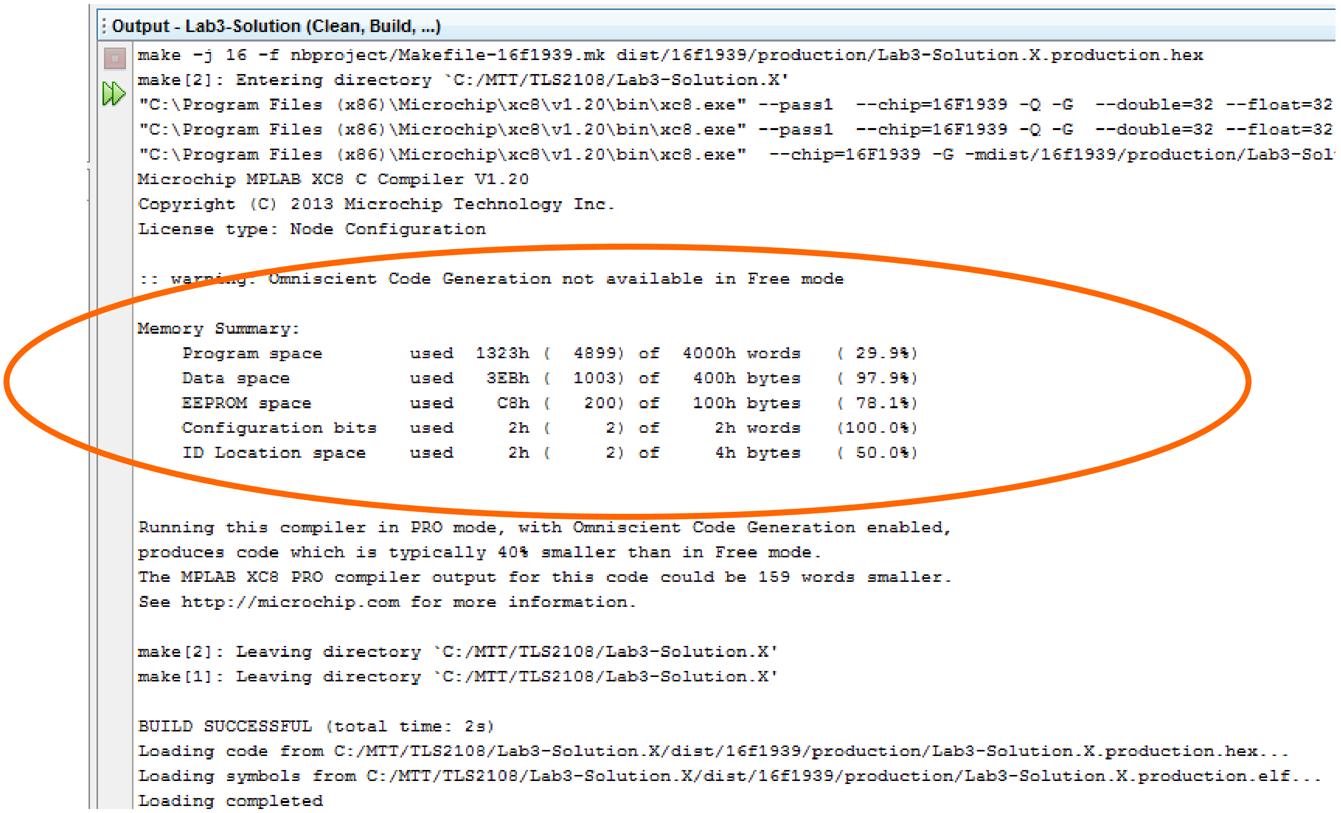
- Close any previously opened projects by right clicking on the project name and selecting “Close”
- Click the **Open Project** icon
- Browse to **C:\MASTERS\18022\Lab3.X** and select **“Open Project”**

This Lab uses a larger memory device than we have on the PICDEM 2 Plus board. This entire lab will be run with the Simulator. All the memory allocation functions can be completed this way.

Lab3.c has large amounts of SRAM and FLASH set up into arrays and scalar variables. Some are initialized and some are not. The names of the arrays are set to give you information about their size and initialization state. Look through the declarations to familiarize yourself with what has been declared.

2 Build the Project

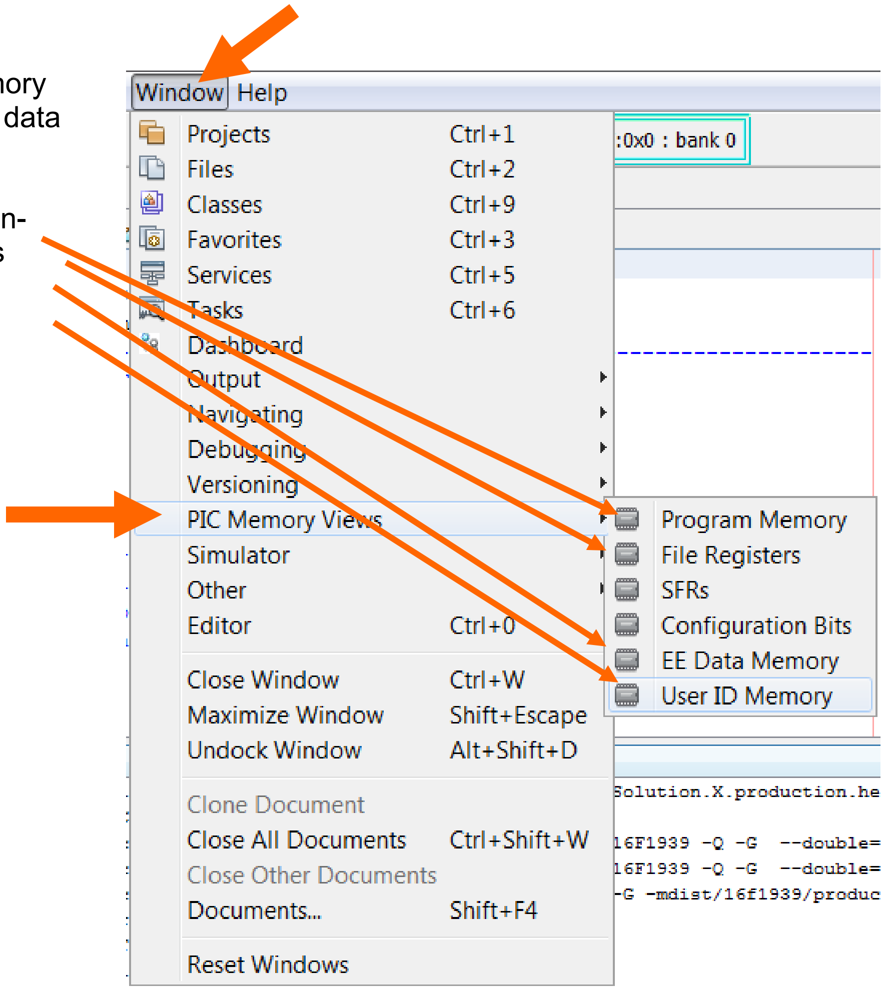
- Look at the Output window and find how much memory is being used of each type
- Play with the keywords used on the declarations to shift some of the declared variables around between different memory areas.
- Be sure to note where they were originally so you can put them back
- Not every declaration can fit in every memory area at the same time so you will likely find some configurations will not build.
- Build again after you have made your modifications.



```
make -j 16 -f nbproject/Makefile-16f1939.mk dist/16f1939/production/Lab3-Solution.X.production.hex
make[2]: Entering directory `C:/MTT/TLS2108/Lab3-Solution.X'
"C:\Program Files (x86)\Microchip\xc8\v1.20\bin\xc8.exe" --pass1  --chip=16F1939 -Q -G  --double=32 --float=32
"C:\Program Files (x86)\Microchip\xc8\v1.20\bin\xc8.exe" --pass1  --chip=16F1939 -Q -G  --double=32 --float=32
"C:\Program Files (x86)\Microchip\xc8\v1.20\bin\xc8.exe"  --chip=16F1939 -G -mdist/16f1939/production/Lab3-Sol
Microchip MPLAB XC8 C Compiler V1.20
Copyright (C) 2013 Microchip Technology Inc.
License type: Node Configuration

:: warning: Omniscient Code Generation not available in Free mode

Memory Summary:
    Program space        used  1323h (  4899) of  4000h words   ( 29.9%)
    Data space           used   3EBh (  1003) of   400h bytes   ( 97.9%)
    EEPROM space         used    C8h (   200) of   100h bytes   ( 78.1%)
    Configuration bits   used     2h (     2) of     2h words   (100.0%)
    ID Location space    used     2h (     2) of     4h bytes   ( 50.0%)


Running this compiler in PRO mode, with Omniscient Code Generation enabled,
produces code which is typically 40% smaller than in Free mode.
The MPLAB XC8 PRO compiler output for this code could be 159 words smaller.
See http://microchip.com for more information.

make[2]: Leaving directory `C:/MTT/TLS2108/Lab3-Solution.X'
make[1]: Leaving directory `C:/MTT/TLS2108/Lab3-Solution.X'

BUILD SUCCESSFUL (total time: 2s)
Loading code from C:/MTT/TLS2108/Lab3-Solution.X/dist/16f1939/production/Lab3-Solution.X.production.hex...
Loading symbols from C:/MTT/TLS2108/Lab3-Solution.X/dist/16f1939/production/Lab3-Solution.X.production.elf...
Loading completed
```

3 After a build Succeeds

- Open some of the PIC Memory Views to inspect where this data resides in the device
- Each Memory area can be inspected with these windows



```
:0x0 : bank 0
Solution.X.production.he
16F1939 -Q -G  --double=
16F1939 -Q -G  --double=
-G -mdist/16f1939/produc
```

Once you feel comfortable with how these memory areas are allocated with the XC8 compiler and you understand how they are stored in each memory area the lab is complete.”

LAB 4: Interrupts

Purpose:

The purpose of this lab is to reinforce the class lessons on how to write an interrupt service routine that will properly handle an event generated by one of the microcontroller's internal peripherals.

Objective

Using the Timer 1 interrupt, make LED D3 (RA0) blink at a rate determined by Timer 1's period with a 1:8 prescaler and Fosc/2 as Timer 1's clock source.
Accomplish the following:

- Turn on Timer 1
- Configure Timer 1's prescaler for a 1:8 ratio
- Select $F_{OSC}/2$ as Timer 1's clock input
- Clear Timer 1's interrupt flag
- Enable the Timer 1 interrupt
-

With a 1:8 postscaler will divide the frequency down from 4MHz. This is the F_{OSC} that should be used in any formulas.
Your task will be to write the Timer 1 interrupt service routine which will toggle the LED and clear the interrupt flag each time the interrupt is triggered by the Timer 1 registers rolling over from 65535 (2^{16}-1) to 0. Blink the LED at approximately .5 to 1 sec intervals.

If you want a bigger challenge or you already feel you know this material well enough, then do not continue past this page.

Challenging Lab:

Do not use any of the predefined lab projects or lab manual. Fulfill the above objective on your own.

Procedure

1 Open Lab4 Project

- Close any previously opened projects by right clicking on the project name and selecting “Close”
- Click the **Open Project** icon
- Browse to **C:\MASTERS\18022\Lab4.X** and select **“Open Project”**

2 Open **Lab4.c** by double clicking on its icon in the project tree. Complete the assigned tasks by adding your code anywhere you see the comments.
`“//### YOUR CODE HERE ###”`
All required reference information is included in this section .
The statement:
void dummy(void) will need to be replaced. This is not a proper interrupt header and is included only so the lab will build before you write the correct code.

The timer has already been set up. You must write the interrupt routine.

Task 4.1: Write the Timer 1 Interrupt Function Header

Write the first line (header) of the interrupt function. Remember that interrupt functions cannot take any parameters nor can they return any data. Also, remember to use the pre-defined function name and that you must tag the interrupt with the appropriate attribute, otherwise it will not be handled properly by the compiler.

Information

Interrupt Attribute Syntax

```
void __interrupt ISRName(void)
{
        <ISR Code Here>
}
```

Section 2.5.10 of MPLAB XC8 User’s Guide

Interrupt Function Name

`Interrupthandler`

Table 11-1 of MPLAB XC8 User’s Guide

Task 4.2: Toggle the LED on RB0 (LATB0)

There are several ways this may be done. The most common method of toggling a bit is to use the exclusive or (XOR) operator (`^`). Any bit that is exclusive or-ed with the value 1 will yield the complement of that bit. In other words: `X ^ 1` → `~X`. Consider using the compound assignment operator `^=`. This isn't the most efficient way to toggle a bit, but it works—we'll cover an optimization trick later in the class.

Task 4.3: Clear the Timer 1 Interrupt Flag (T1IF)

Simply write a value of zero to the appropriate interrupt flag bit.

Syntax: Clear a bit in a register

Syntax to clear a bit named "BITNAME" in a register named "REGISTERNAME":

REGISTERNAME*bits.*BITNAME = 0;

Information

PIR1: PERIPHERAL INTERRUPT REQUEST REGISTER 1

R/W-0/0	R/W-0/0	R-0/0	R-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
TMR1GIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 0 **TMR1IF:** Timer1 Interrupt Flag Status bit
`1` = Interrupt request has occurred
`0` = Interrupt request has not occurred

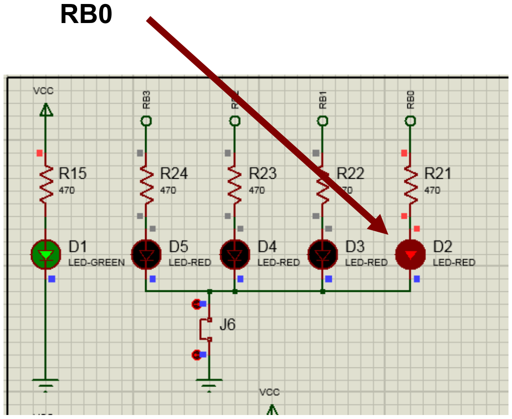
Register 675 of PIC16F19xx Family Data Sheet DS41364E

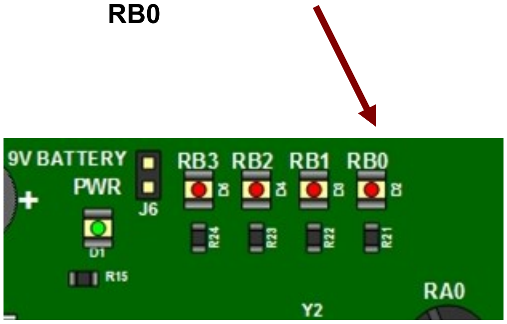
3 Building the Project

- Click the "**Debug and Run**" icon to build and run the project

Proteus PICDEM 2 + RB0

Hardware PICDEM 2 + RB0





LAB 5: *Making and Using Custom Libraries*

Purpose:

The purpose of this lab is to become familiar with the steps required to create a custom library for use in your projects. The Librarian has not been integrated into the MPLAB X environment yet, but it is in the XC8 compiler. We will use it from a command window.

Objective

Using the XC8 Librarian, convert the existing file timer1.c into an object library. Then remove the C file timer1.c from the project and replace it with the newly created library. Rebuild the lab using the library to show everything is working.

If you want a bigger challenge or you already feel you know this material well enough, then do not continue past this page.

Challenging Lab:

Do not use any of the predefined lab projects or lab manual. Fulfill the above objective on your own.

Procedure

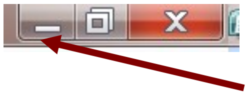
1 Open Lab5 Project

- Close any previously opened projects by right clicking on the project name and selecting “Close”
- Click the **Open Project** icon
- Browse to `C:\MASTERS\18022\Lab5.X` and select **“Open Project”**

2 Build and Debug the Project

- Click the “**Debug and Run**” icon to build and run the project. This will allow you to verify the operation of the lab.
- When you are done remember to stop the debug session

3 Minimize the MPLAB X environment

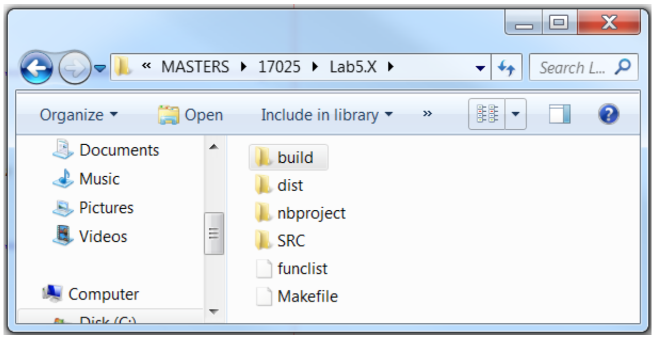


- Use the minimize “underscore” button in the upper right corner.

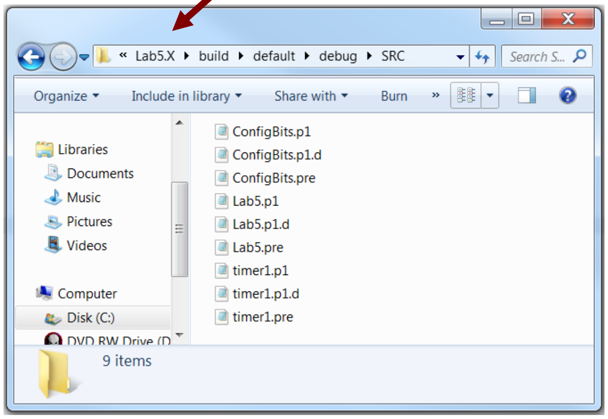
These next sets will be in the windows environment.

4 Open a Windows Explorer window to browse your

- Browse to the Lab5.X folder located at : `C:\MASTERS\18022\Lab5.X`
- Here you will see the **build** and **dist** folders
- Under each folder you will find a default folder, under that a folder named **debug**
- Down here is where we will need to look for the libraries once they are created.

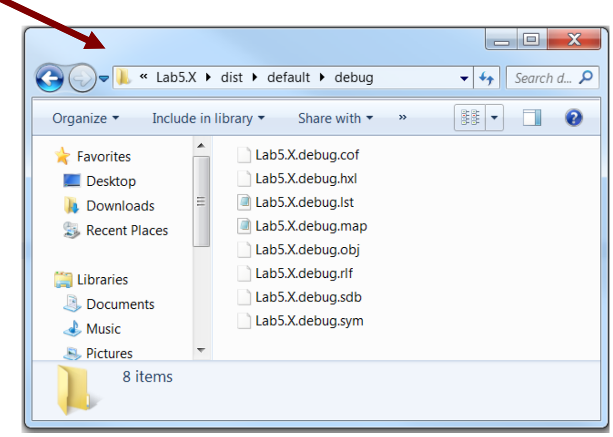


build
dist
nbproject
SRC
funclist
Makefile



ConfigBits.p1
ConfigBits.p1.d
ConfigBits.pre
Lab5.p1
Lab5.p1.d
Lab5.pre
timer1.p1
timer1.p1.d
timer1.pre

9 items

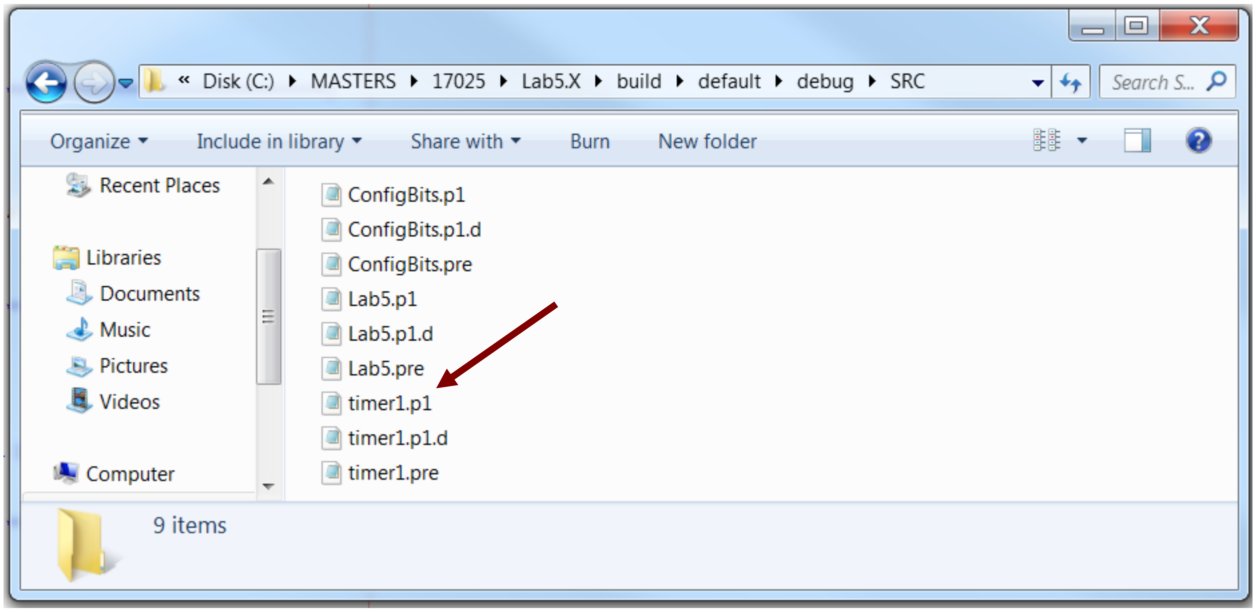


Lab5.X.debug.cof
Lab5.X.debug.hxl
Lab5.X.debug.lst
Lab5.X.debug.map
Lab5.X.debug.obj
Lab5.X.debug.rlf
Lab5.X.debug.sdb
Lab5.X.debug.sym

8 items

5 Find the timer1.p1 file

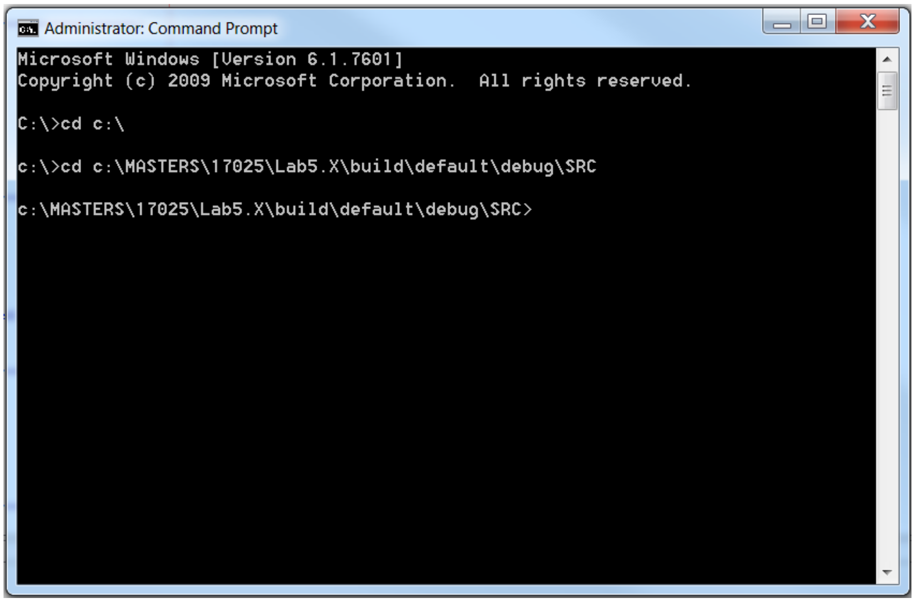
Currently there are .p1 files deep under the build folder, here is what we want to use to create our library.



6 Open a DOS command prompt window

- Make sure you are at the root of your c: drive by typing the command at the prompt
  `cd c:\`
- Next you must move to the local directory where the file that needs to be convert resides by typing
  `cd c:\MASTERS\18022\LAB5.X\build\default\debug\SRC`

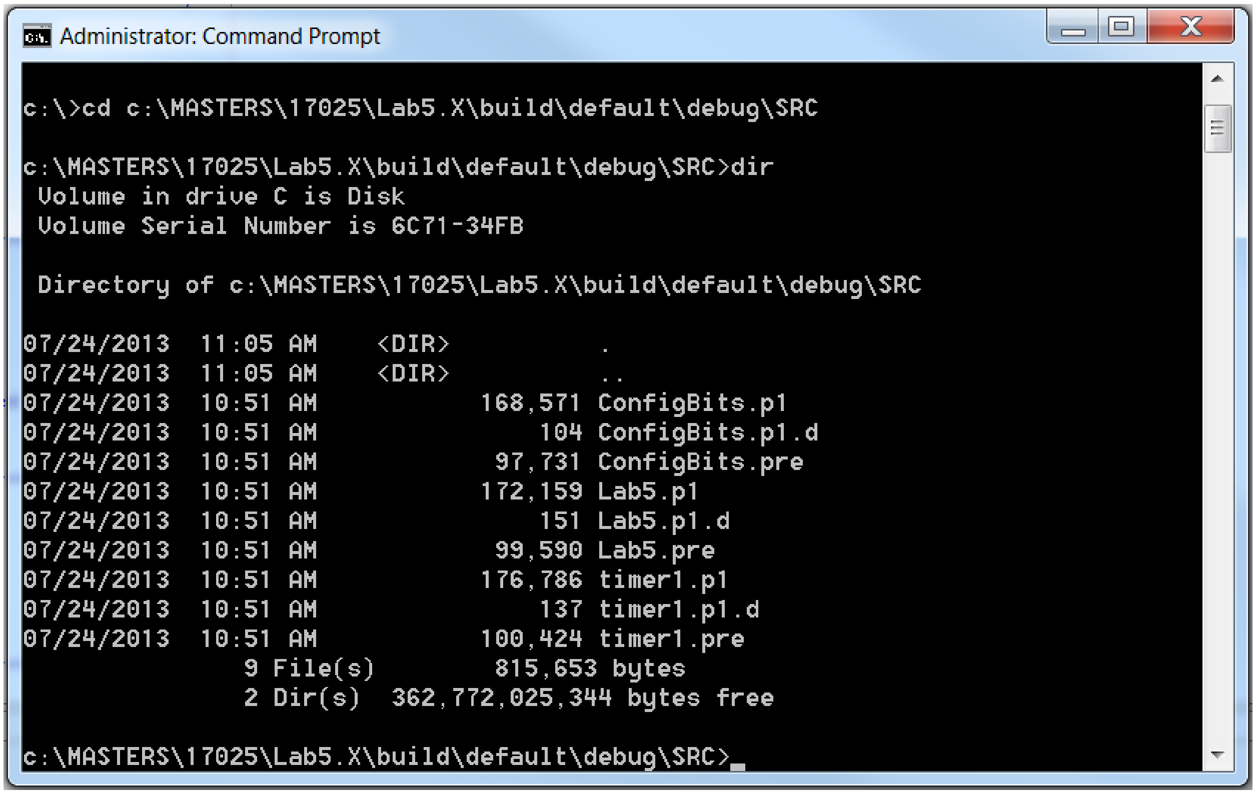
SEE BELOW



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\>cd c:\

c:\>cd c:\MASTERS\17025\Lab5.X\build\default\debug\SRC

c:\MASTERS\17025\Lab5.X\build\default\debug\SRC>
```

7 In the correct directory

- You can now type
  dir
- to verify the files you are looking for are here



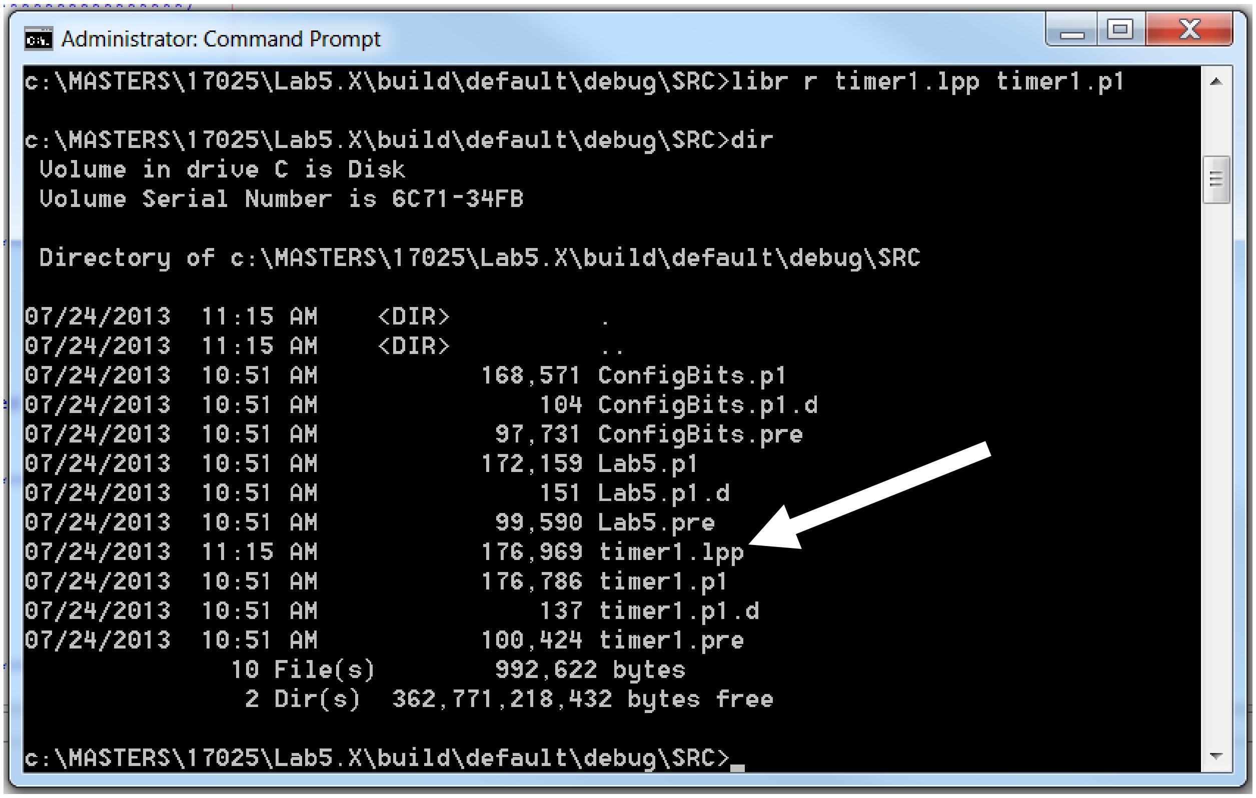
```
c:\>cd c:\MASTERS\17025\Lab5.X\build\default\debug\SRC

c:\MASTERS\17025\Lab5.X\build\default\debug\SRC>dir
 Volume in drive C is Disk
 Volume Serial Number is 6C71-34FB

 Directory of c:\MASTERS\17025\Lab5.X\build\default\debug\SRC

07/24/2013  11:05 AM    <DIR>          .
07/24/2013  11:05 AM    <DIR>          ..
07/24/2013  10:51 AM           168,571 ConfigBits.p1
07/24/2013  10:51 AM               104 ConfigBits.p1.d
07/24/2013  10:51 AM            97,731 ConfigBits.pre
07/24/2013  10:51 AM           172,159 Lab5.p1
07/24/2013  10:51 AM               151 Lab5.p1.d
07/24/2013  10:51 AM            99,590 Lab5.pre
07/24/2013  10:51 AM           176,786 timer1.p1
07/24/2013  10:51 AM               137 timer1.p1.d
07/24/2013  10:51 AM           100,424 timer1.pre
               9 File(s)        815,653 bytes
               2 Dir(s)  362,772,025,344 bytes free

c:\MASTERS\17025\Lab5.X\build\default\debug\SRC>_
```

8 Using Librarian

- At the prompt type:

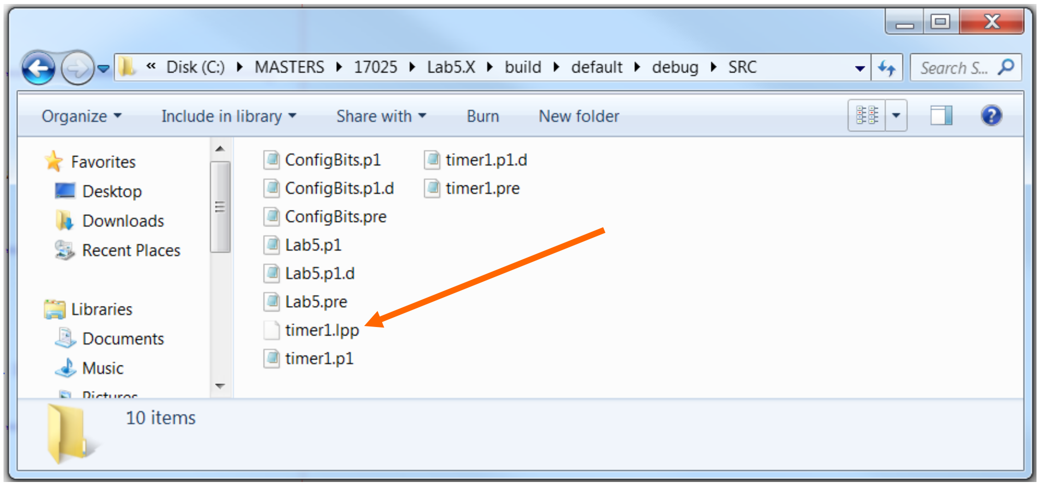
```
libr r timer1.lpp timer1.p1
```

- There is no response from the librarian to the window
- Next type `dir` again
- Now we can see what the librarian has done.
- There is a new file in the directory `timer1.lpp`
- This is our new library
- **Close the command prompt window and we are done with it.**



```
c:\MASTERS\17025\Lab5.X\build\default\debug\SRC>libr r timer1.lpp timer1.p1

c:\MASTERS\17025\Lab5.X\build\default\debug\SRC>dir
 Volume in drive C is Disk
 Volume Serial Number is 6C71-34FB

 Directory of c:\MASTERS\17025\Lab5.X\build\default\debug\SRC

07/24/2013  11:15 AM    <DIR>          .
07/24/2013  11:15 AM    <DIR>          ..
07/24/2013  10:51 AM           168,571 ConfigBits.p1
07/24/2013  10:51 AM               104 ConfigBits.p1.d
07/24/2013  10:51 AM            97,731 ConfigBits.pre
07/24/2013  10:51 AM           172,159 Lab5.p1
07/24/2013  10:51 AM               151 Lab5.p1.d
07/24/2013  10:51 AM            99,590 Lab5.pre
07/24/2013  11:15 AM           176,969 timer1.lpp
07/24/2013  10:51 AM           176,786 timer1.p1
07/24/2013  10:51 AM               137 timer1.p1.d
07/24/2013  10:51 AM           100,424 timer1.pre
              10 File(s)        992,622 bytes
               2 Dir(s)  362,771,218,432 bytes free

c:\MASTERS\17025\Lab5.X\build\default\debug\SRC>_
```

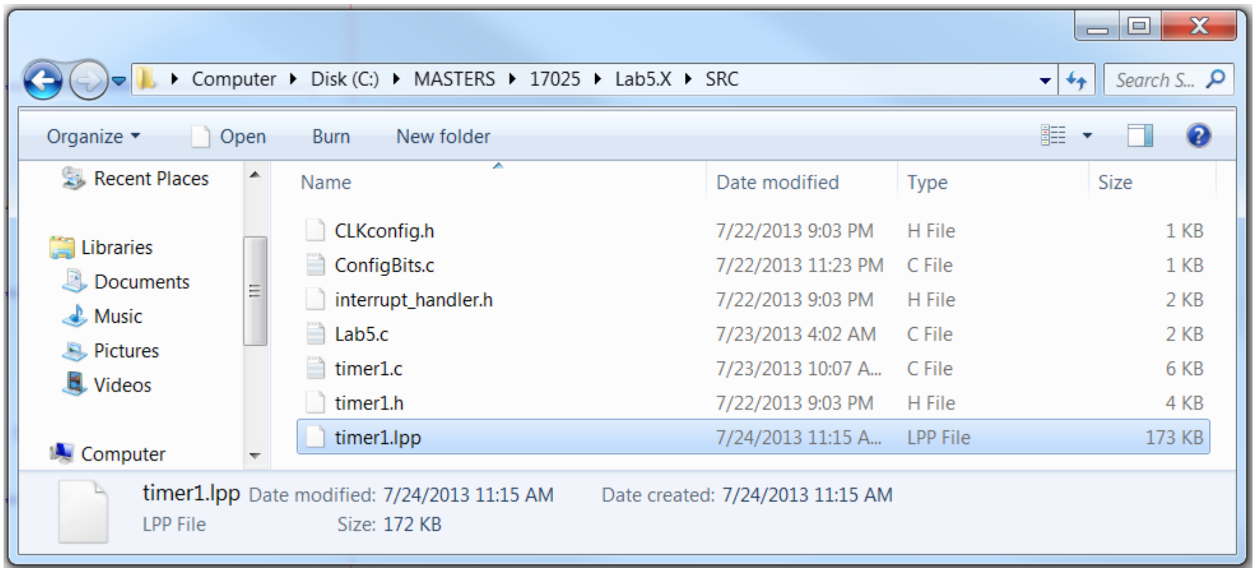
Back in the windows environment we can see the same results.



9 Move the new library into the SRC folder with the other source

- This step is just for convenience the library can reside anywhere and be added to the project
- Right click and copy the file **timer1.lpp**
- Paste it to the folder

```
C:\MASTERS\18022\Lab5.X\SRC
```



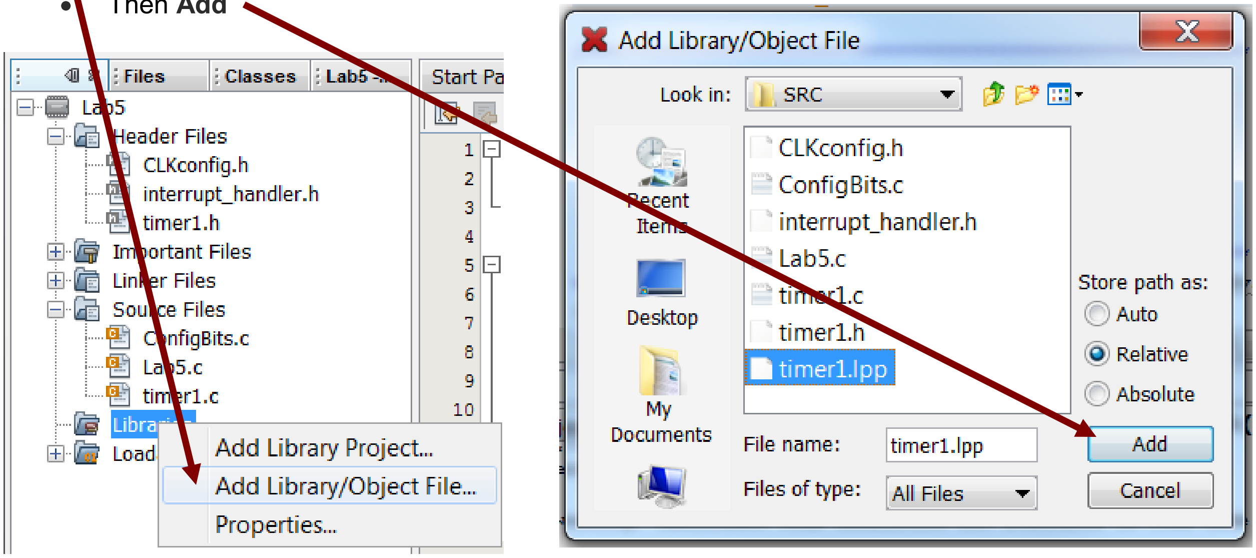
Computer ▸ Disk (C:) ▸ MASTERS ▸ 17025 ▸ Lab5.X ▸ SRC

Name	Date modified	Type	Size
CLKconfig.h	7/22/2013 9:03 PM	H File	1 KB
ConfigBits.c	7/22/2013 11:23 PM	C File	1 KB
interrupt_handler.h	7/22/2013 9:03 PM	H File	2 KB
Lab5.c	7/23/2013 4:02 AM	C File	2 KB
timer1.c	7/23/2013 10:07 A...	C File	6 KB
timer1.h	7/22/2013 9:03 PM	H File	4 KB
timer1.lpp	7/24/2013 11:15 A...	LPP File	173 KB

timer1.lpp Date modified: 7/24/2013 11:15 AM Date created: 7/24/2013 11:15 AM
LPP File Size: 172 KB

10 Add Library Files to the Project

- Un-minimize the MPLAB X environment
- Right click on the **Library Files** folder in the project window
- Select **Add Library/Object File**
- Select the new library file **timer1.lpp**
- Then **Add**



Add Library/Object File

Look in: SRC

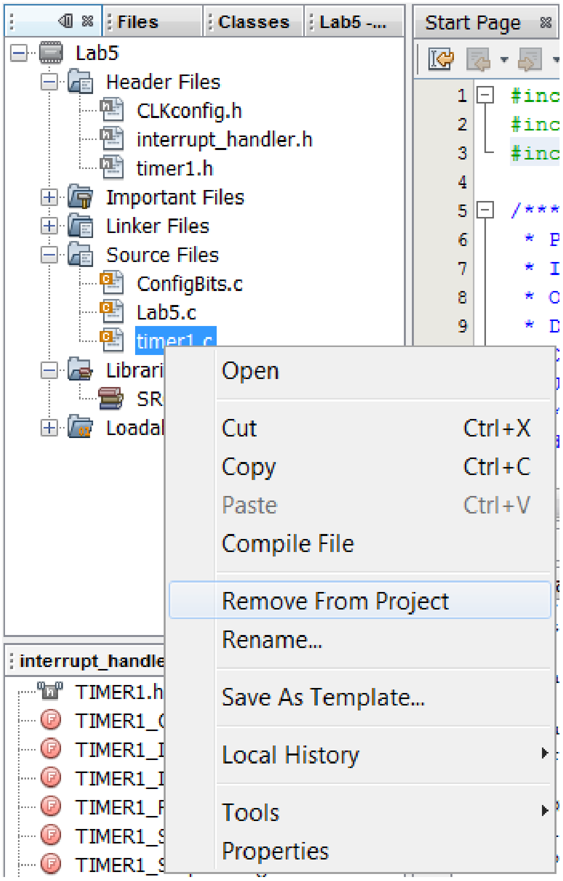
File name: timer1.lpp

Files of type: All Files

Store path as: ○ Auto ● Relative ○ Absolute

11 Remove the old C source file

- Right click on the **timer1.c** in the **Source Files** folder in the project window
- Select **Remove From Project**
- The timer1.c file does not have to be removed physically from the project folder. It will no longer be compiled because it is not in the list of source file within the project window.



12 Build and Debug the Project

- Click the "**Debug and Run**" icon to build and run the project as before
- The LED should again blink as before

LAB 6:
Calling Assembly Functions from C

Purpose:

The purpose of this lab is to illustrate how C and assembly files may be used in the same project, how C code can call assembly functions, and how parameters may be passed between them.

Objective

Given a project that contains an assembly language source file with two subroutines (_incFunction and _decFunction) and a C source file with a main function, add the necessary elements in the C source file to make the assembly subroutines callable from C.

If you want a bigger challenge or you already feel you know this material well enough, then do not continue past this page.

Challenging Lab:

Do not use any of the predefined lab projects or lab manual. Fulfill the above objective on your own project by reusing the assembly source file found in the Lab6 project folder.

Procedure

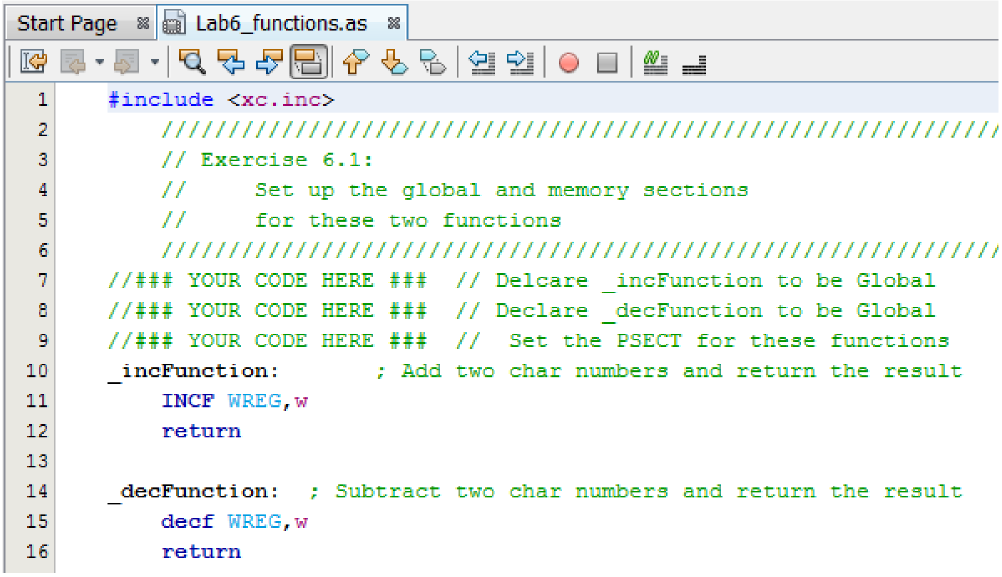
1 Open Lab6 Project

- Close any previously opened projects by right clicking on the project name and selecting “Close”
- Click the **Open Project** icon
- Browse to **C:\MASTERS\18022\Lab6** and select **“Open Project”**

2 The Assembly file

Task 6.1: Write the GLOBAL and PSECT directives

- Here are our two function we wish to call from the main function
- First we must add the global declaration in assembly for these two routines



```
#include <xc.inc>
    ///////////////////////////////////////////////////////////////
    // Exercise 6.1:
    //     Set up the global and memory sections
    //     for these two functions
    ///////////////////////////////////////////////////////////////
//### YOUR CODE HERE ###  // Delcare _incFunction to be Global
//### YOUR CODE HERE ###  // Declare _decFunction to be Global
//### YOUR CODE HERE ###  //  Set the PSECT for these functions
_incFunction:       ; Add two char numbers and return the result
    INCF WREG,w
    return

_decFunction:  ; Subtract two char numbers and return the result
    decf WREG,w
    return
```

Example

```
GLOBAL _function1, _variable2
```

These two directives tell the linker that these function names will have project wide scope. This allows you to call the functions from C just as if they had been written in C.

Task 6.1: Write the GLOBAL and PSECT directives continued

Now we must inform the assembler what this code is, where it belongs and how it will be used with the PSECT directive

PSECT Directive

```
PSECT name,scope,class=name,delta=size
```

- The name can be any unused section name. exp. `Text0`
- Scope defaults to global either can be used in this case but let’s use `local` so we remember the parameter is here.
- Class will be `code` because that is what we are using
- Finally, delta is `2.` This is defined for this device family based on how the FLASH memory is organized.

Task 6.2: Create the C prototypes

In the file Lab6.c

- Add the function prototypes for each function
- These are standard C prototypes for functions with one char parameter and a char return value

Information

Symbol Refresher

Remember that symbols from C have a leading underscore ‘_’ added when used in assembly.

Therefore the C symbol fubar in assembly is referred to as _fubar
And the reverse is true if you are using a symbol _snafu in assembly it is referred to in C as snafu.

Task 6.3: Call the incFunction

In the file Lab6.c

- Use the incFunction to increment the variable a and place the result in c.

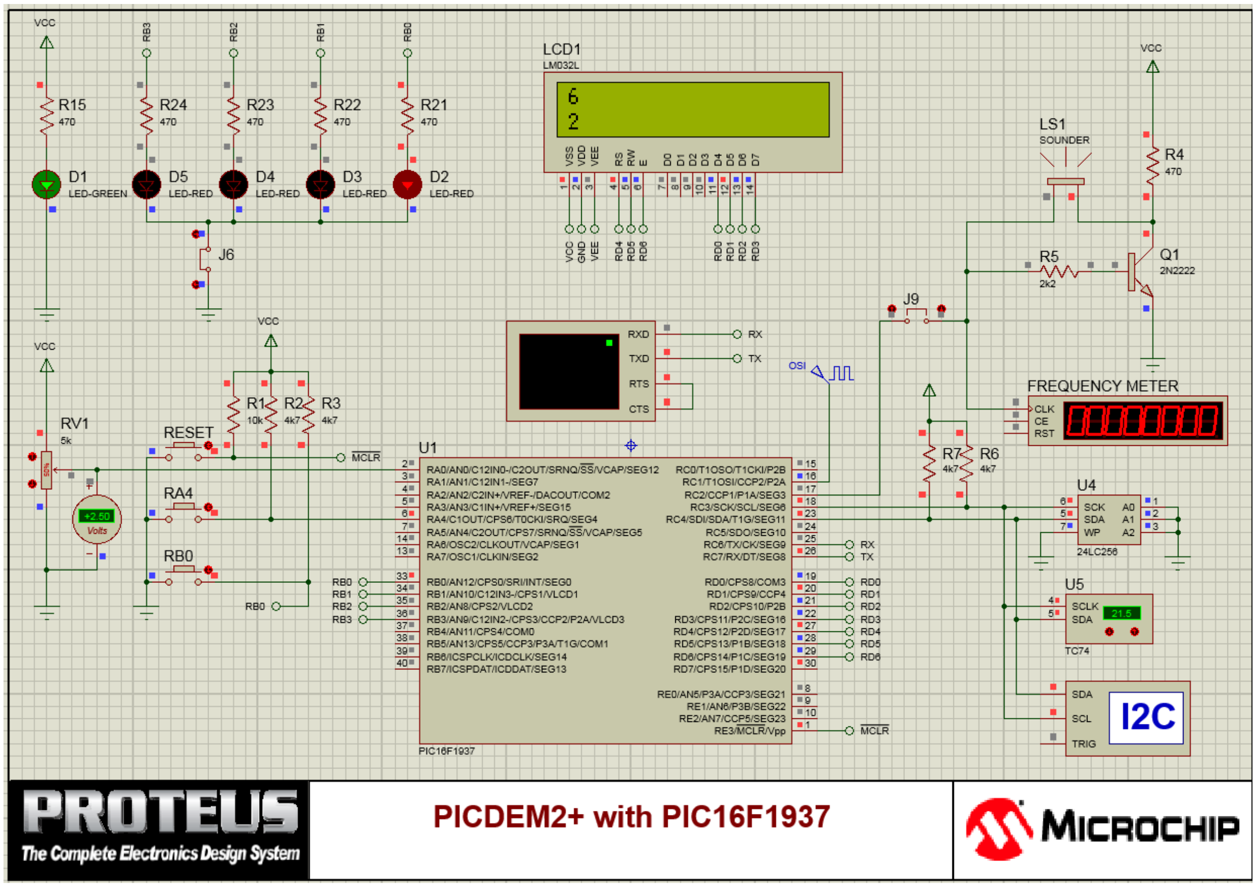
Task 6.4: Call the decFunction

In the file Lab6.c

- Use the decFunction to decrement b and place the result in c.

3 Building the Project

- Click the “**Debug and Run**” icon to build and run the project


VCC
RB3
RB2
RB1
RB0
R15 470
R24 470
R23 470
R22 470
R21 470
D1 LED-GREEN
D5 LED-RED
D4 LED-RED
D3 LED-RED
D2 LED-RED
J6
LCD1
LM032L
6
2
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
VCC GND VEE RD4 RD5 RD6 RD0 RD1 RD2 RD3
LS1
SOUNDER
R4 470
R5 2k2
Q1 2N2222
J9
RXD TXD RTS CTS
RX TX
OSI
FREQUENCY METER
CLK CE RST
RV1 5k
+2.50 Volts
RESET
R1 10k
R2 4k7
R3 4k7
MCLR
RA4
RB0
U1
PIC16F1937
RA0/AN0/C12IN0-/C2OUT/SRNQ/SS/VCAP/SEG12
RA1/AN1/C12IN1-/SEG7
RA2/AN2/C2IN+/VREF-/DACOUT/COM2
RA3/AN3/C1IN+/VREF+/SEG15
RA4/C1OUT/CPS6/T0CKI/SRQ/SEG4
RA5/AN4/C2OUT/CPS7/SRNQ/SS/VCAP/SEG5
RA6/OSC2/CLKOUT/VCAP/SEG1
RA7/OSC1/CLKIN/SEG2
RB0/AN12/CPS0/SRI/INT/SEG0
RB1/AN10/C12IN3-/CPS1/VLCD1
RB2/AN8/CPS2/VLCD2
RB3/AN9/C12IN2-/CPS3/CCP2/P2A/VLCD3
RB4/AN11/CPS4/COM0
RB5/AN13/CPS5/CCP3/P3A/T1G/COM1
RB6/ICSPCLK/ICDCLK/SEG14
RB7/ICSPDAT/ICDDAT/SEG13
RC0/T1OSO/T1CKI/P2B
RC1/T1OSI/CCP2/P2A
RC2/CCP1/P1A/SEG3
RC3/SCK/SCL/SEG6
RC4/SDI/SDA/T1G/SEG11
RC5/SDO/SEG10
RC6/TX/CK/SEG9
RC7/RX/DT/SEG8
RD0/CPS8/COM3
RD1/CPS9/CCP4
RD2/CPS10/P2B
RD3/CPS11/P2C/SEG16
RD4/CPS12/P2D/SEG17
RD5/CPS13/P1B/SEG18
RD6/CPS14/P1C/SEG19
RD7/CPS15/P1D/SEG20
RE0/AN5/P3A/CCP3/SEG21
RE1/AN6/P3B/SEG22
RE2/AN7/CCP5/SEG23
RE3/MCLR/Vpp
R7 4k7
R6 4k7
U4
SCK SDA WP A0 A1 A2
24LC256
U5
SCLK SDA
21.5
TC74
SDA SCL TRIG
I2C
PROTEUS
The Complete Electronics Design System
PICDEM2+ with PIC16F1937
MICROCHIP