Lab 1 – Introduction to PIC18F4550 Board, MPLABX-IDE, C-compiler and USB downloader.

Objectives

- ☐ To illustrate the procedures to create a Microchip's PIC micro-controller project in MPLABX IDE, and to create, edit and compile a C program using XC8.
- ☐ To show the steps to setup the USB link with the PIC18F4550 micro-controller, and to download a program to the micro-controller and to execute it.

Introduction / Briefing

⇨

- ☐ At the beginning of each lab session, your lab lecturer will go through a short **briefing** before you begin the experiment.
- ☐ The discussion will help you in the MST, the lab test as well as the project. So, please pay attention and participate in the discussion.

⇨

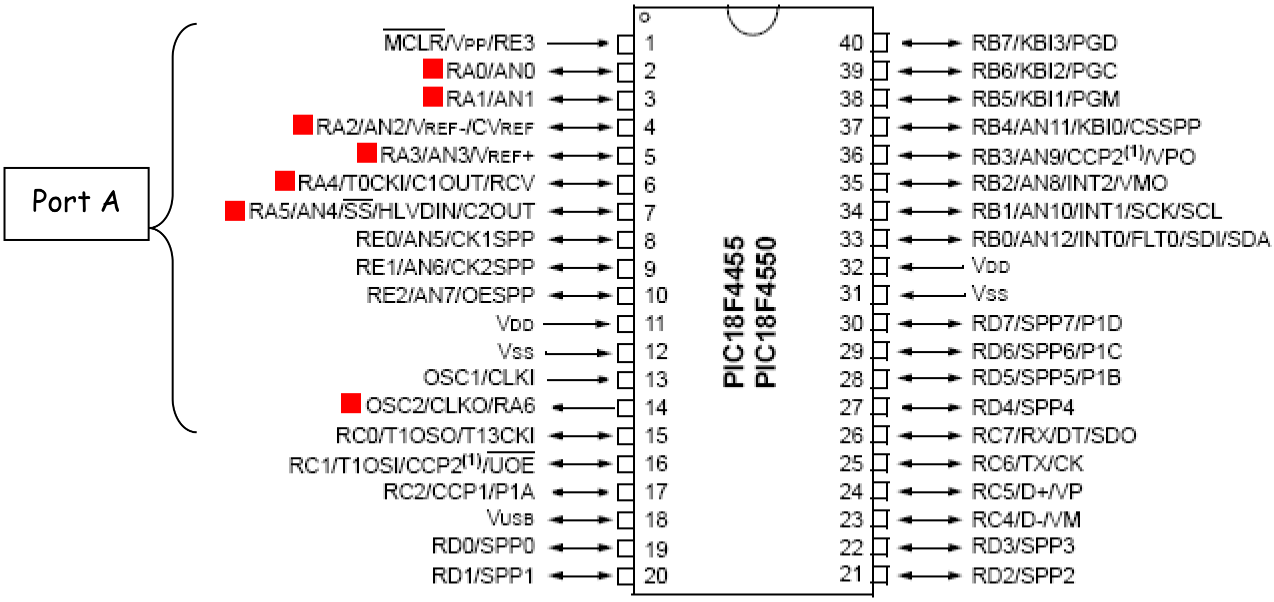
- ☐ This lab sheet contains many **screen captures** to show you how to create a project, how to create, edit and compile a C program, and how to download a program to the micro-controller and run it. In subsequent labs, if you forget certain steps, you should refer to this lab sheet again.

⇨

- ☐ To do this lab, the **software tools** required must already be installed on the PC.

PIC18F4550 I/O ports

- ☐ You will learn more about the I/O ports in Chapter 3. The following is a brief summary.
- ☐ PIC18F4550 has five I/O ports: A to E. Many pins have multiple functions. For instance, pin 14 is RA6 (Port A Pin 6) and also OSC2 (oscillator input 2).



☐ The table below shows which pins can be used as general purpose I/O pins and whether they are, by default (i.e. after power on reset), analogue or digital, input or output.

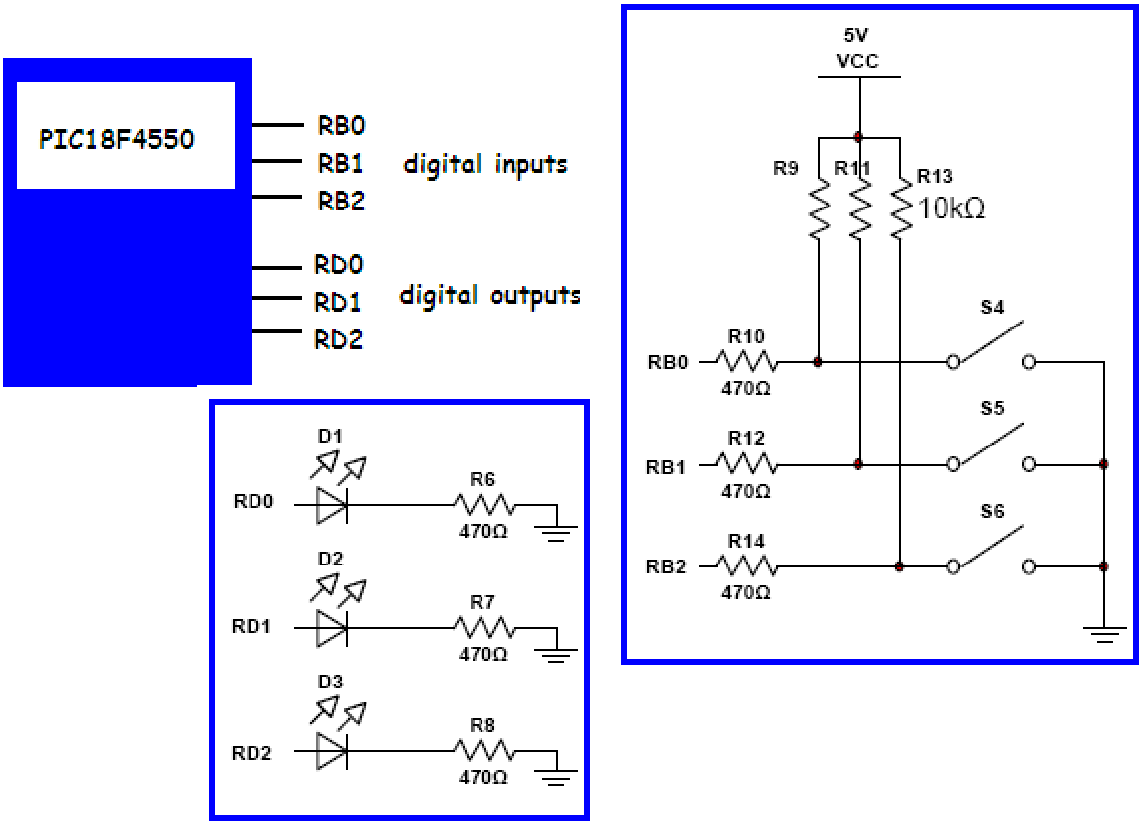
Port	Available pins	Not available as general purpose I/O (- reasons)	After power on reset
A	RA6-0	RA6 (– oscillator)	RA5, 3-0: Analogue inputs (*). RA4: Digital input.
B	RB7-0	RB4 (– "Boot" button)	RB4-0: Digital / Analogue inputs (#). RB7-5: Digital inputs.
C	RC7-4, 2-0	RC5-4 (– USB connector)	RC7-4, 2-0: Digital inputs.
D	RD7-0		RD7-0: Digital inputs.
E	RE3-0	RE3 (– "Reset" button)	RE2-0: Analogue inputs (*). RE3: Digital input.

(*) ADC (Analogue to Digital Conversion) will be discussed in details in the future.
(#) For the PIC18 chips used in the labs, RB4-0 are Digital inputs after power on reset.

☐ This lab will only involve ports B and D.

Port configuration

- ☐ Do you know why the switches connected to RB0-2 are "active low"?
- ☐ Do you know why the LED's connected to RD0-2 are "active high"?



- ☐ To use port B to read the switch status (open or closed), port B must be configured as digital inputs.
- ☐ Referring to the table above, RB4-0 are digital inputs after reset.
- ☐ To use port D to control the LEDs (on or off), port D must be configured as digital outputs.
- ☐ But, RD7-0 are digital inputs after reset.
- ☐ The command below must be added to change them into digital outputs:

```
TRISD = 0b00000000;
```

☐ TRISD is the "data directional register" for Port D.

☐ By writing a 0 into a particular TRISD bit, the corresponding PORTD pin become an Output pin.

	7	6	5	4	3	2	1	0
TRISD	0	0	0	0	0	0	0	0

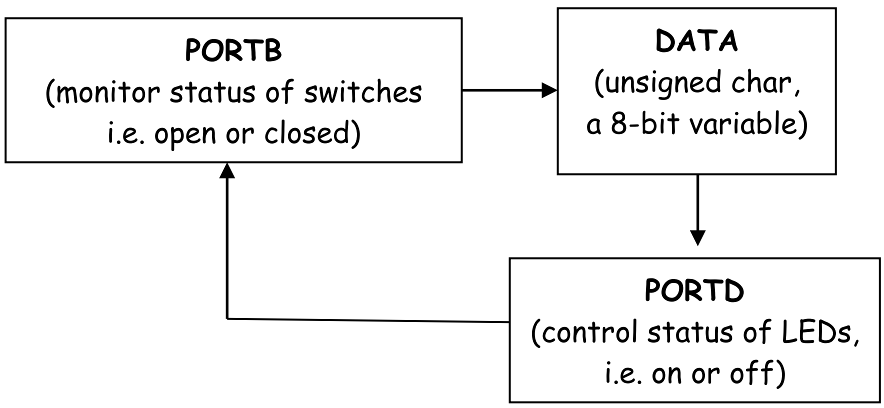
	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
PORTD	Output	Output	Output	Output	Output	Output	Output	Output

☐ Likewise, by writing a 1 into a particular TRISD bit, the corresponding PORTD pin can become an Input pin.

Looping forever

☐ After configuring Port B as digital input and Port D as digital output, the while (1) loop below will be executed over and over:

```
while (1)
 {
   data = PORTB;  // switch status is copied into a variable called DATA
   PORTD = data;  // and used to turn on/off the LEDs
 }
```



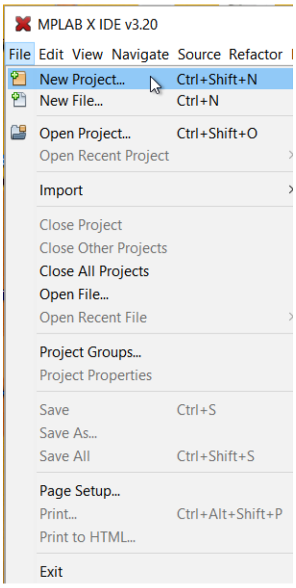
Activites:

1. Before the beginning of each lab lesson, double click on the MAPP icon on the desktop. This will delete the files modified by other students before you and replace them with fresh copies. The (.c & project) files used in all the experiments will be stored in a folder named ProjectX in the D:\ Drive.

2. Double click on the *MPLABX IDE* icon on the desktop to launch the software.

Creating a Microchip's PIC micro-controller project in MPLABX-IDE

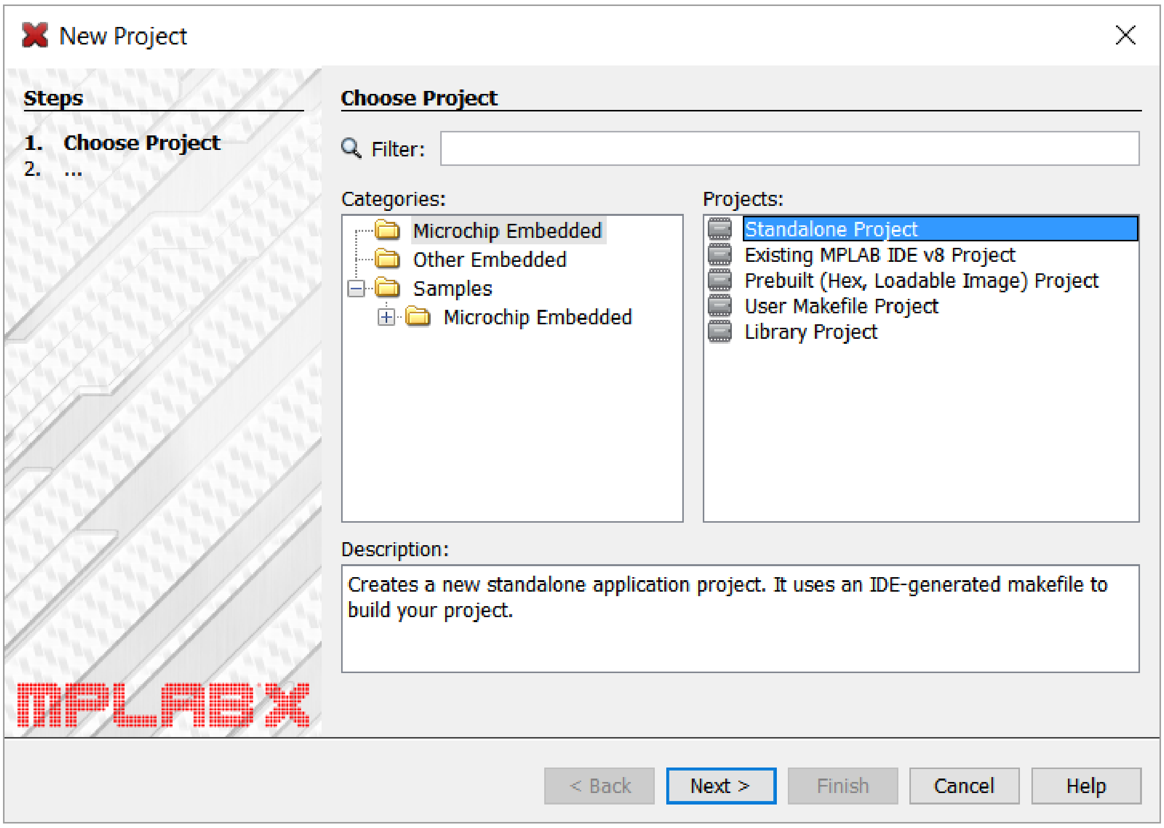
3. Click *File -> New Project …* to create a new project.



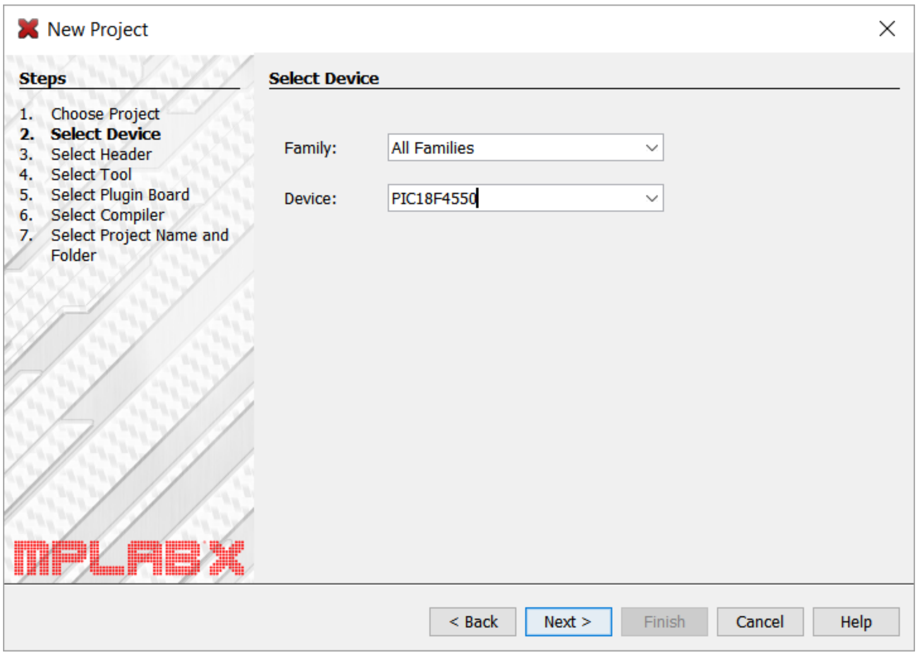
4. You will see the *New Project* window.

   We will use the default setting *Microchip Embedded – Standalone Project*.

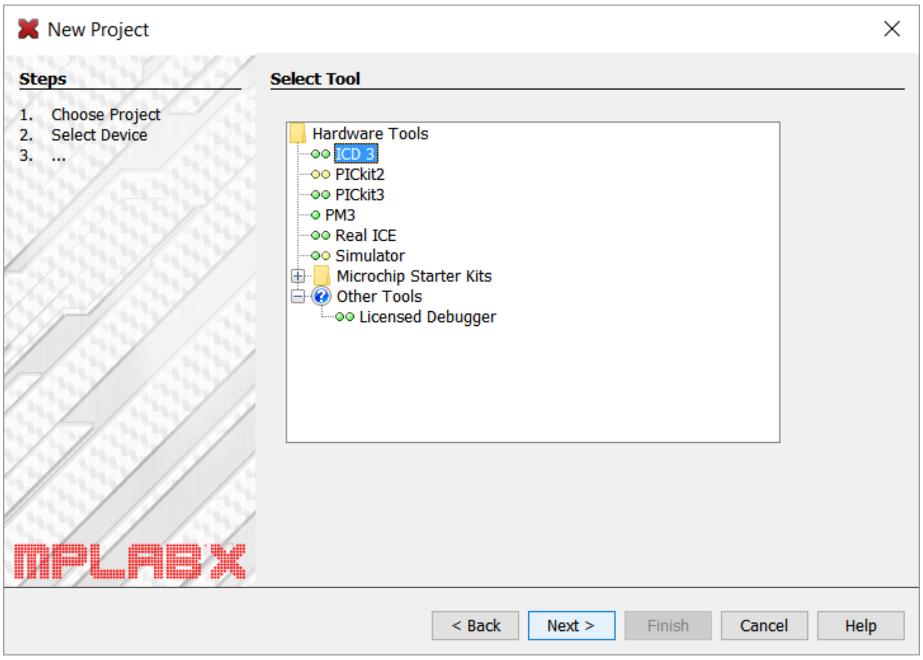
   Click *Next*.



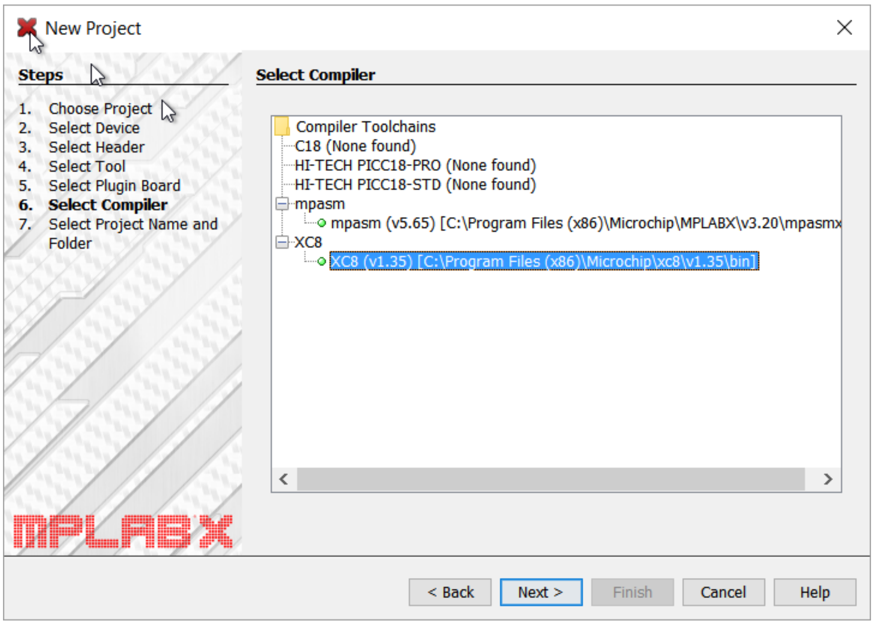
5. Select *PIC18F4550* as the *Device* (i.e. the microcontroller) to use for this project. [Hint: You can type the number 4550 to filter down the list.] Then, click *Next*.



6. We are not using any debugging tools, so just click *Next*.

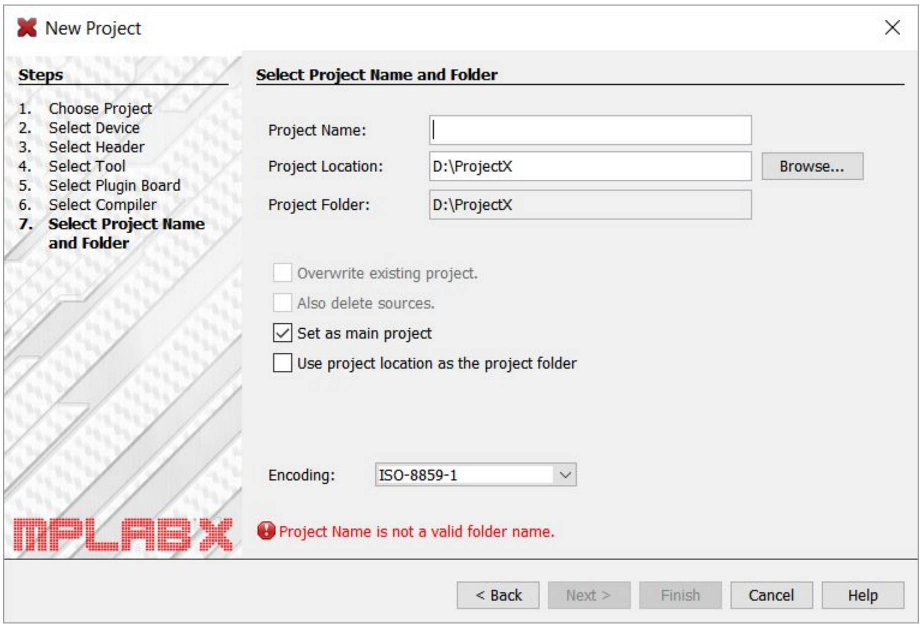


7. Select the *XC8* Compiler as follows and click *Next*.



8. Browse to the *Project Location* as follows and enter the *Project Name*:

Project Name : *Lab1*
Project Location : *D:\ProjectX*

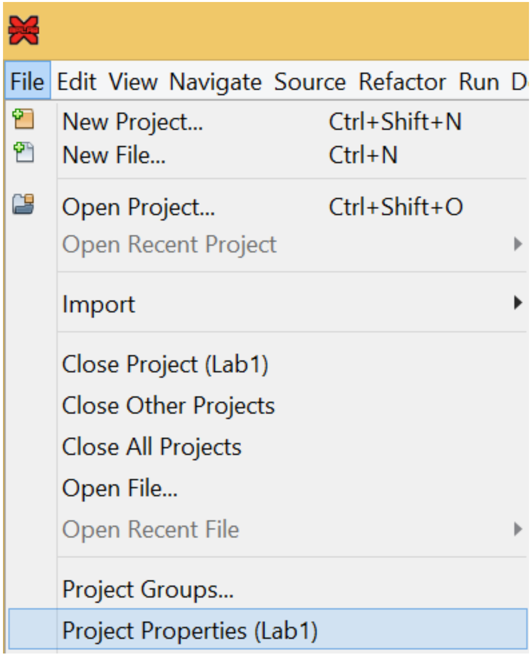


Then, click *Finish*.

9. Set the Codeoffset to 1000 as follows.

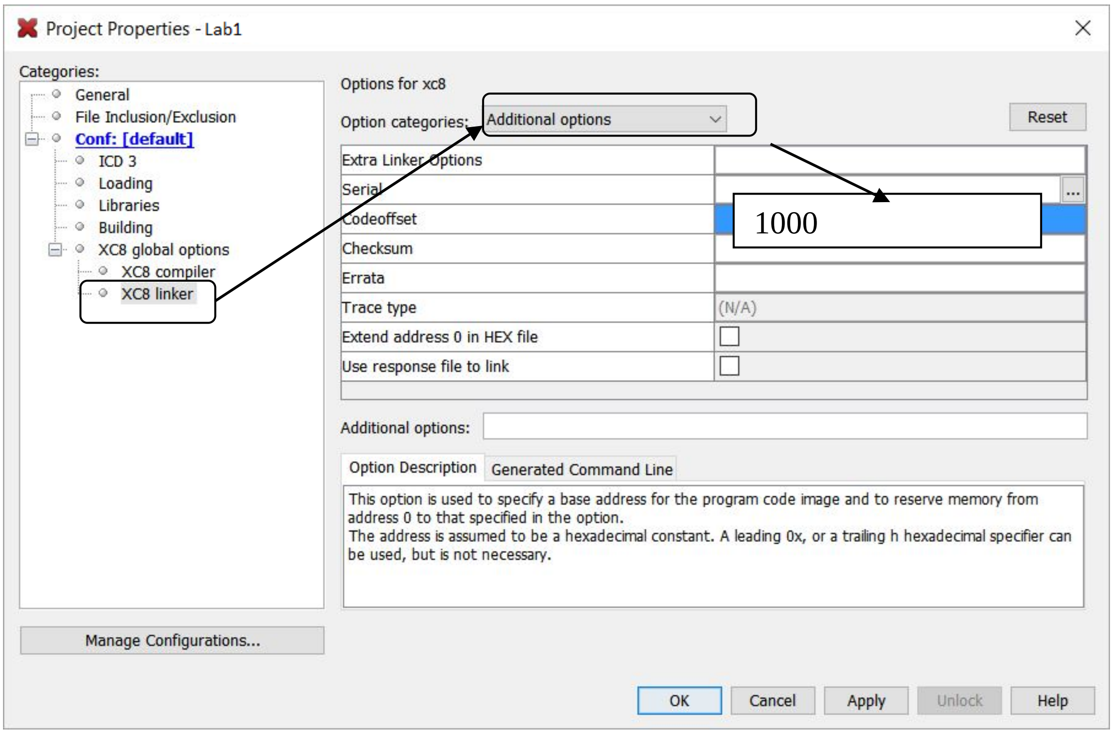
   The microcontroller that we are using already has a program in the ROM called the bootloader. It is used for communicating with a program in the PC using the USB ports. The bootloader in the ROM occupies the locations from 0x0000 to 0x0FFF. The user program for the microcontroller must therefore starts from location 0x1000. To do this, we must set the Codeoffset to 1000.

   Click *File -> Project Properties (Lab1)*:

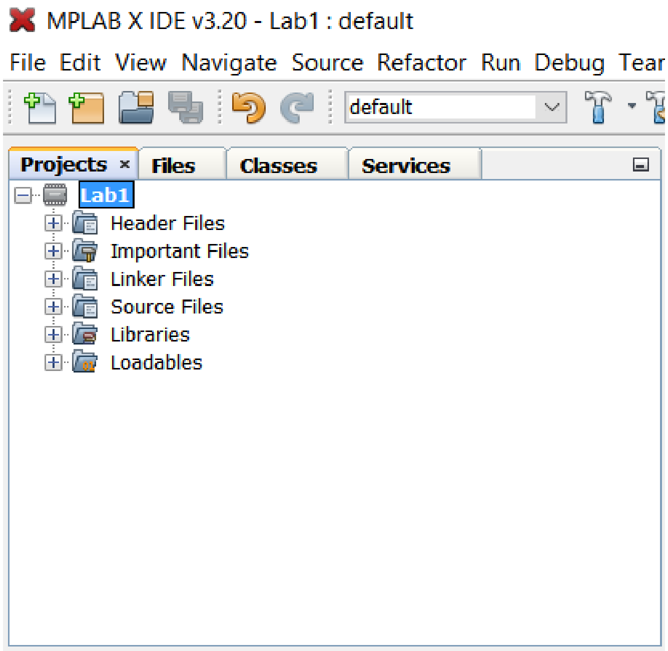


10. You will see the *Project Properties – Lab 1* window:

Click *XC8 linker*. Then in *Option categories*, select *Additional options*. And enter 1000 for *Codeoffset* and then click *OK*.



11. You will see a summary of the project *Lab1* that you have just created.

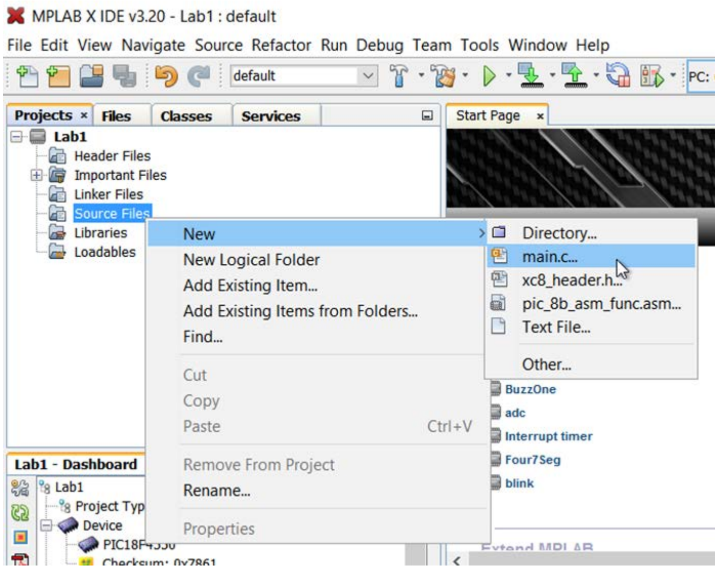


At this point, a new project has been created but there are no files.

Creating, editing and compiling a C-program using XC8

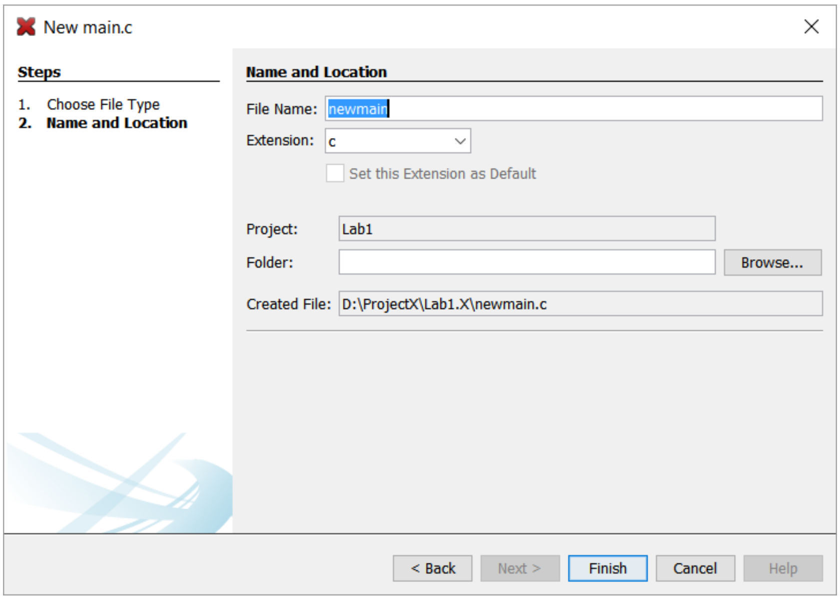
☐ In the next few steps, you will add a new C source file, and edit it. You will then compile the C program.

12. To add a new file, right click on *Source Files -> New -> main.c*

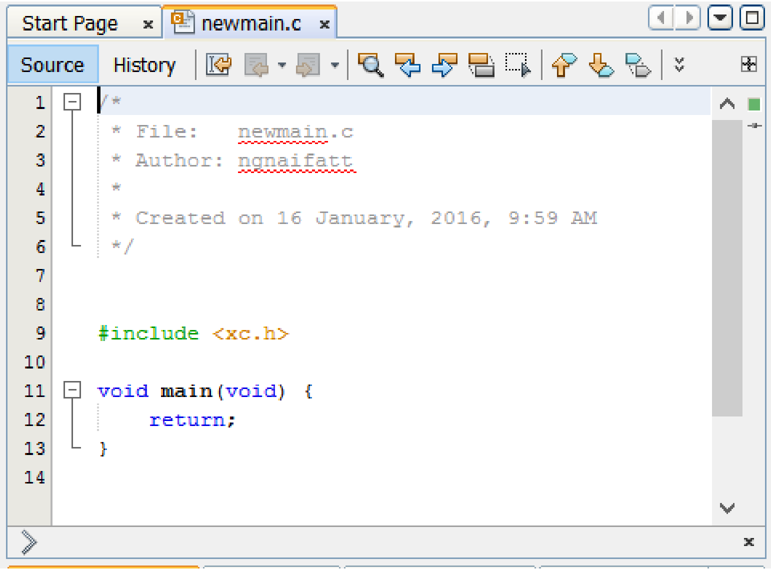


For this lab, we will use the default *File Name*: *newmain* and default *Extension*: *c*.

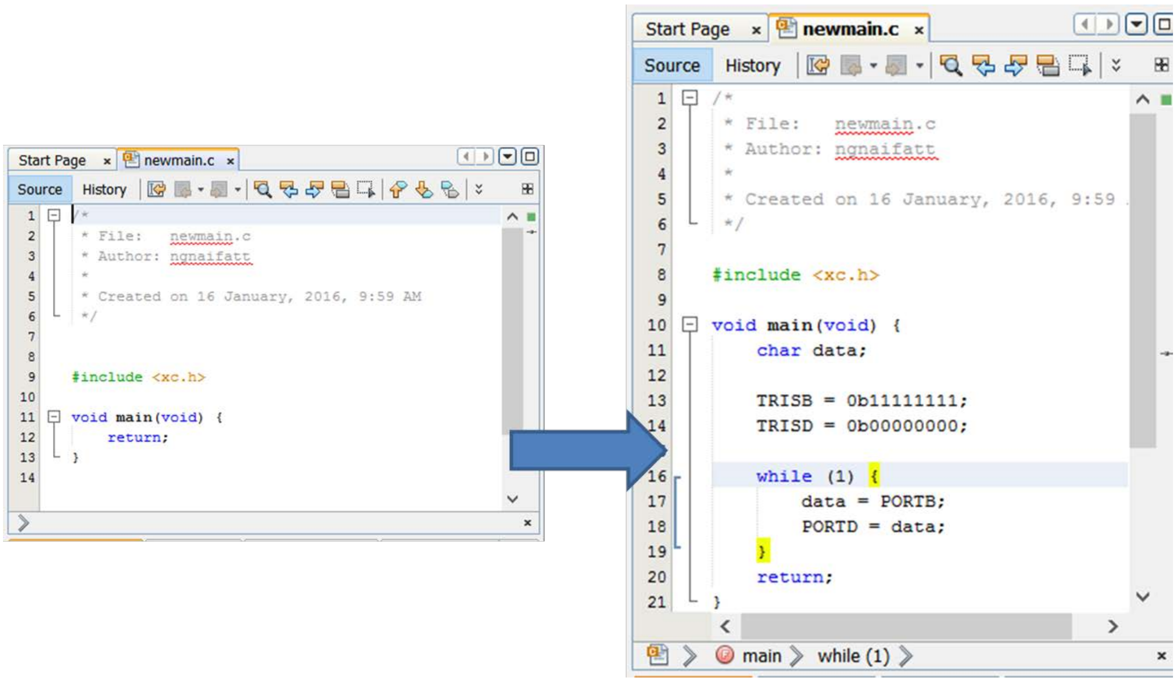
So click *Finish*.



13. The c file generated by the compiler has a brief description and an empty *main()* function:

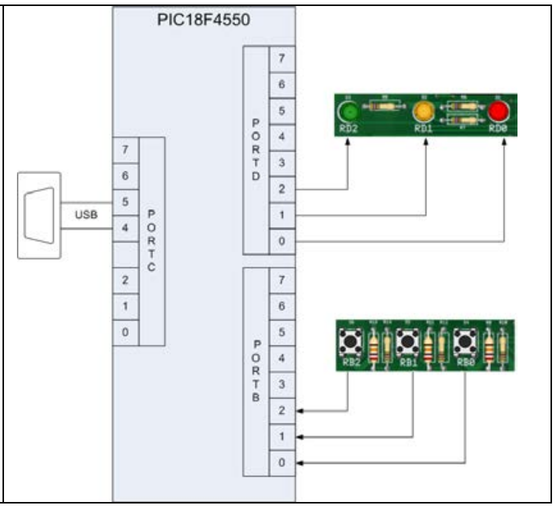


14. Add the lines as shown on the right to *newmain.c*.

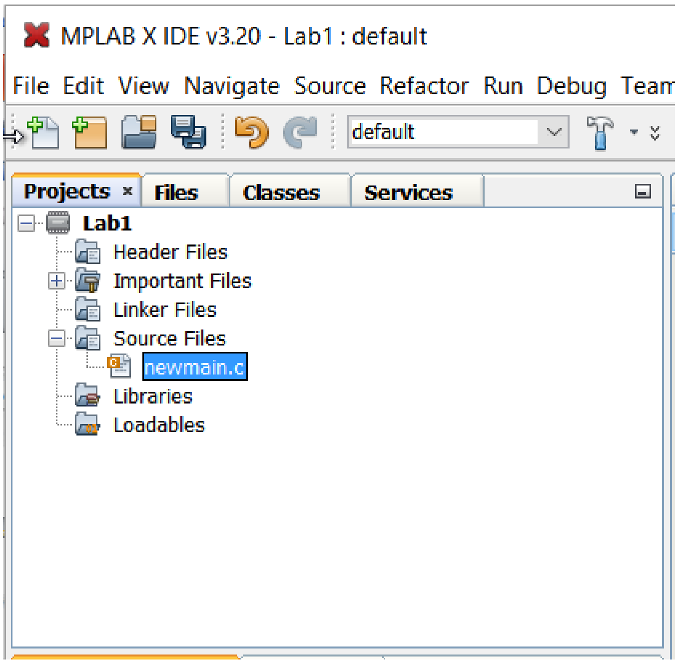


15. What the program is doing? Look at the comments for an explanation.

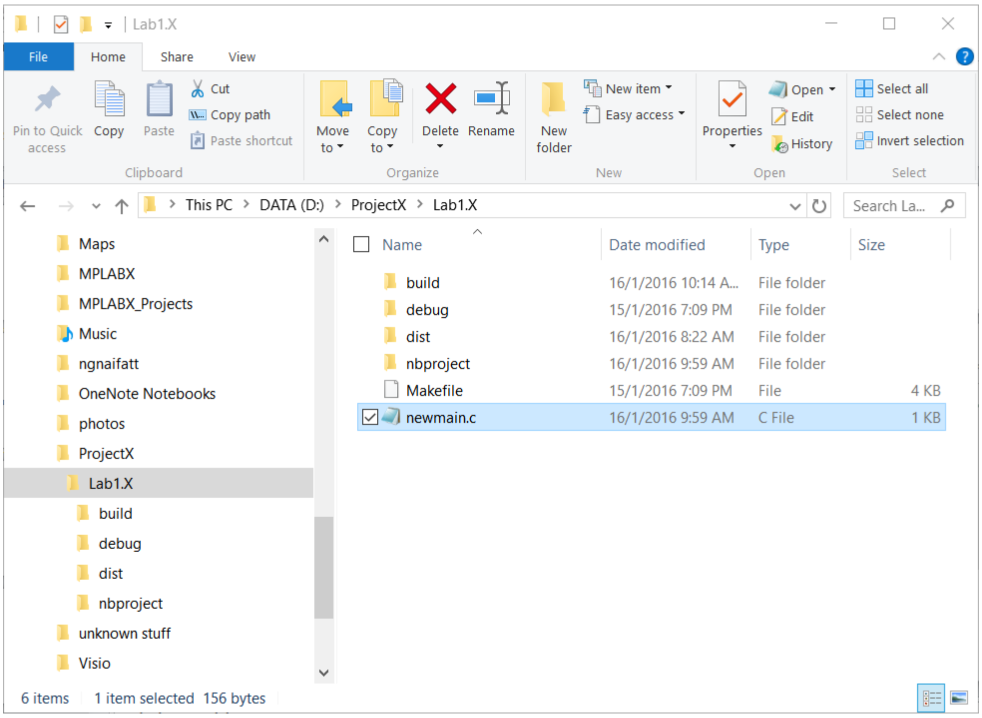
```
TRISB = 0b11111111;// PORTB is configured as all inputs

TRISD = 0b00000000;// PORTD is configured as
                   // all outputs

while (1) {        // repeat forever the code below

    data = PORTB; // read the inputs from PORTB
                  // and save in data

    PORTD = data; // output the data to PORTD
}
```



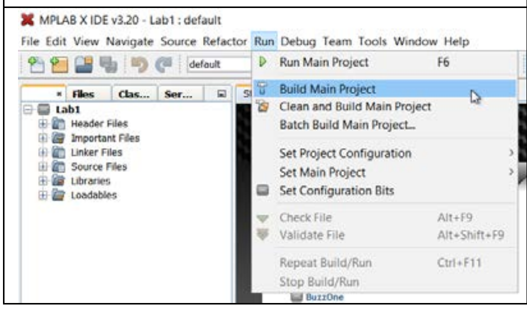
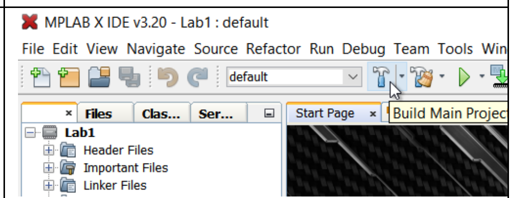
16. In the *Projects* tab, click on the [+] next to *Source Files* to expand it. You should be able to see *newmain.c* under *Source Files*:



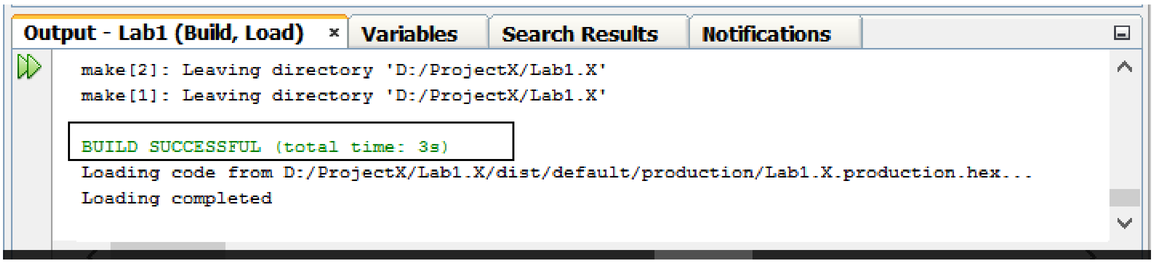
17. The file *newmain.c* is found in the folder *D:\ProjectX\Lab1.X folder*:



18. Let's build the project with a Hammer!

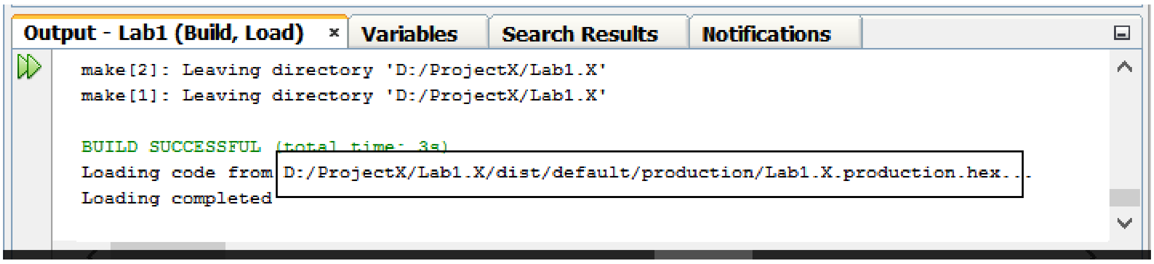
Method 1 : Run -> Build		Method 2: Click Hammer.
		

19. If there are no errors, you will see *BUILD SUCCESSFUL*.

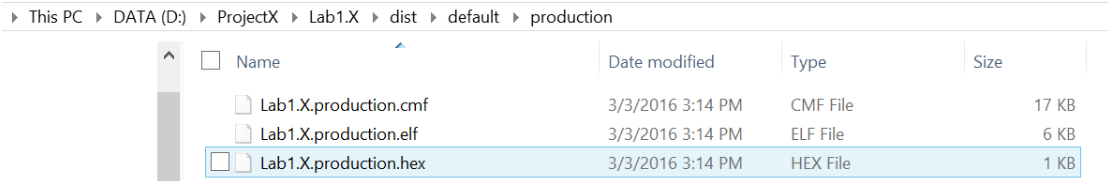


If there are errors, click on the error message and that will lead to the program line that has the error. After correcting the error, try to build again.

20. You can see the machine code/hex file *Lab1.X.production.hex* has been created.



21. Using *window explorer (my computer)*, browse to the folder *D:\ProjectX\Lab1.X\dist\default\production* and you should be able to see that hex file:



At this point, a *C* program (*newmain.c* which was created earlier) has been compiled into the hex code *Lab1.X.production.hex*.

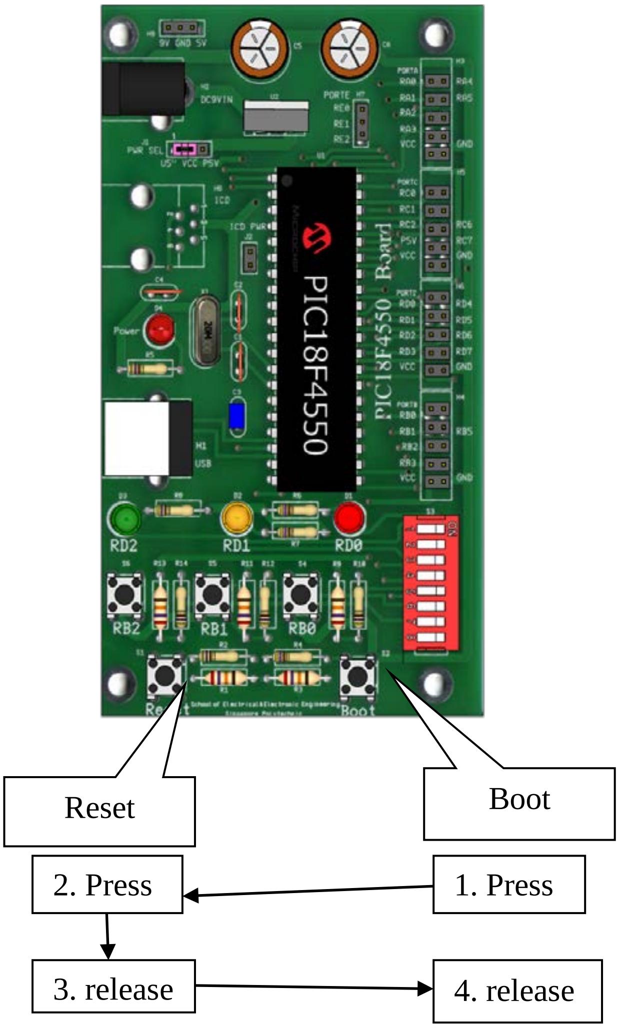
This is a machine code version of the *newmain.c* which must be downloaded to the microcontroller for it to be executed.

<u>Downloading a program (a hex file) to the micro-controller and executing it.</u>

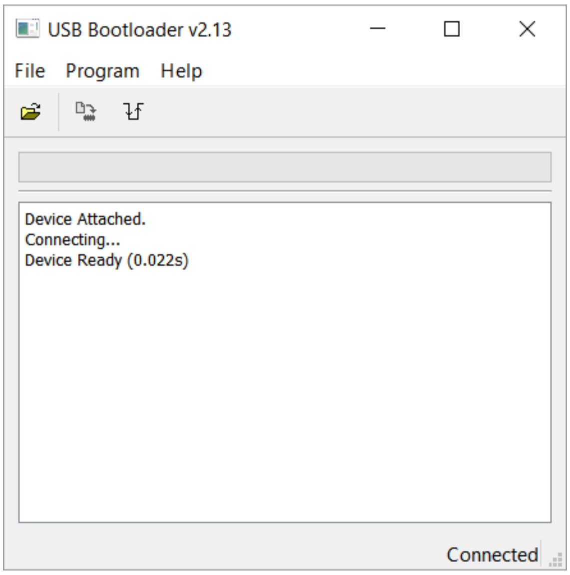
22. Connect the *PIC18F4550 board* to the *PC's USB port* using the *USB cable* provided.

23. Double click on the *HIDBootloader* icon on the desktop to launch the software.

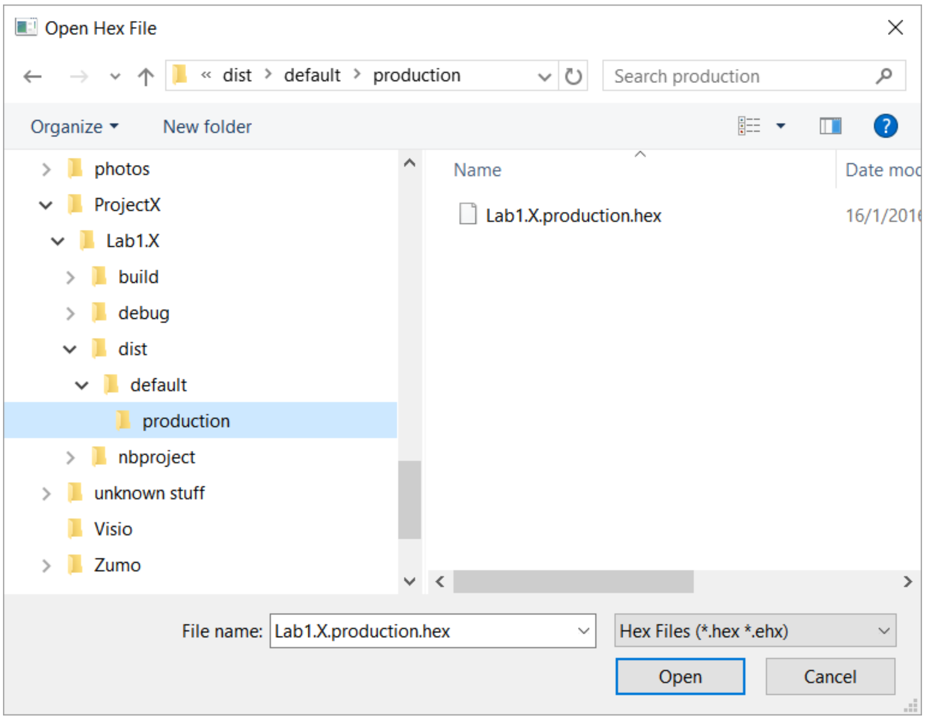
24. On the *PIC18F4550 board*, <u>press & hold</u> the *Boot* button, <u>press</u> and then <u>release</u> the *Reset* button, and finally <u>release</u> the *Boot* button.



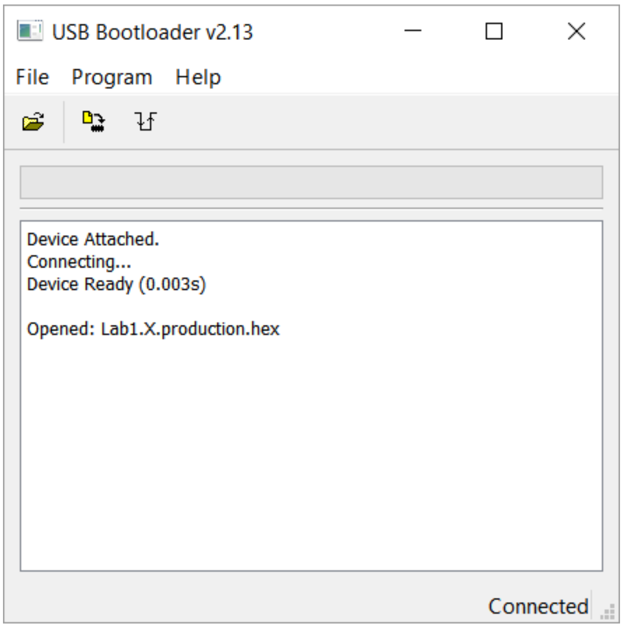
25. The *USB Bootloader* software will automatically detect the *PIC18F4550 board* and the message "*Device Ready*" will be shown:



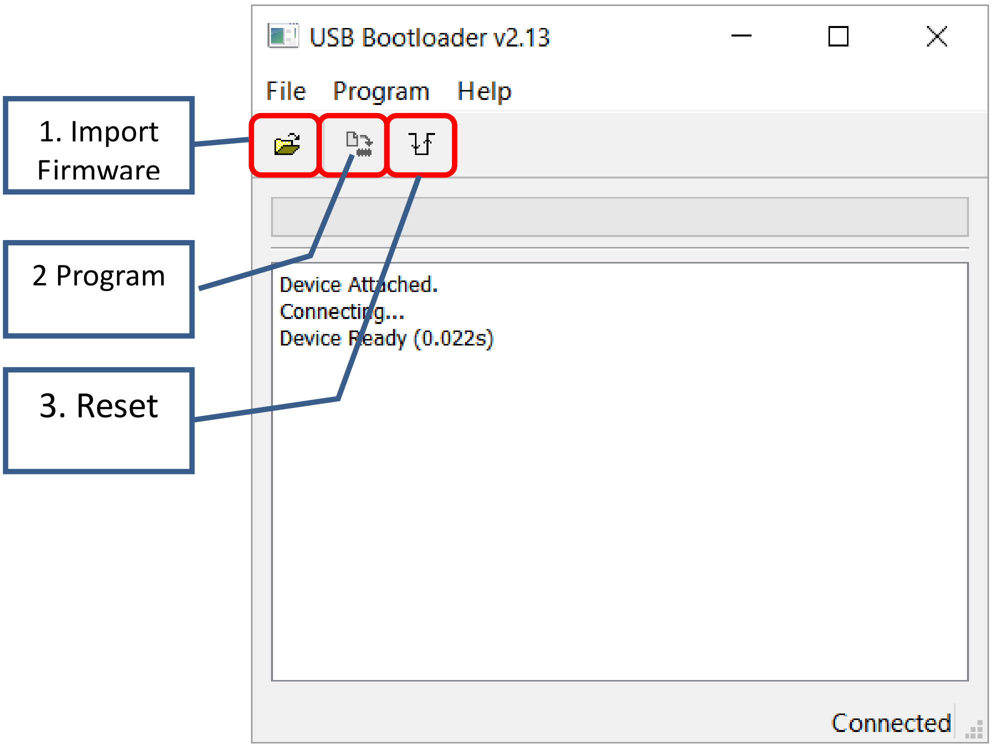
26. Click *File -> Import Firmware Image* and select the HEX file that was just created, *Lab1.X.production.hex*.



27. Click *Open* and the status will be updated as shown.



28. Click *Program Device* to program the micro-controller.



This diagram summaries the three steps needed to download a program.

29. Click *Reset* on the *USB Bootloader* software or pressing the *Reset* button on the *PIC18F4550 board* will run the program in the micro-controller.

30. Can you see the LED's (connected to *RD2*, *RD1* and *RD0*) light up?

31. Press the buttons connected *RB2*, *RB1* and *RB0* one by one. What is the effect of pressing a button?

    Your answer: ____

⇨ ☐ At the end of each lab, there is an **Extra Exercise** for those who finish early. Doing this Extra Exercise allows you to learn more about micro-controller.

Extra Exercise

☐ In this exercise, you will modify the C-program to light up the LED's one by one.

32. Modify the C program to the following.

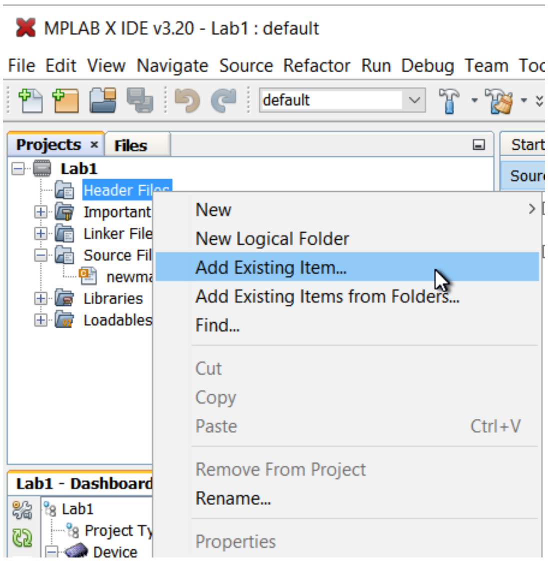
```
#include <xc.h>

void main(void) {

    TRISB = 0b11111111;
    TRISD = 0b00000000;

    while (1) { // repeat
        PORTD = 0b00000001; // turn on LED in RD0
        delay_ms(500);      // delay for 500ms
        PORTD = 0b00000010; // turn on LED in RD1
        delay_ms(500);      // delay for 500ms
        PORTD = 0b00000100; // turn on LED in RD2
        delay_ms(500);      // delay for 500ms
    }

    return;
}
```

33. Build the program and is it successful?

    Your answer: ___________________

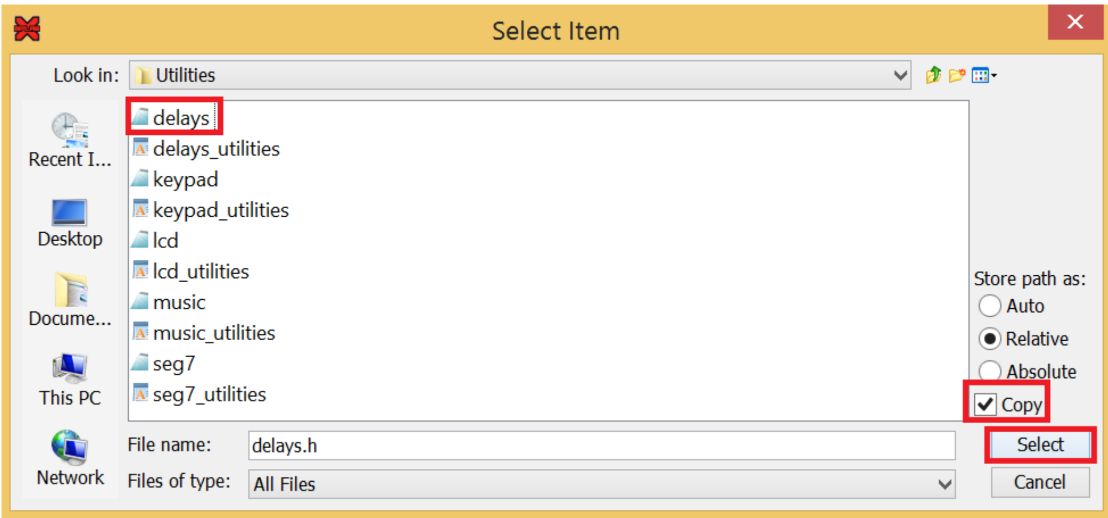
34. The error is because the compiler does not know what the function *delay_ms(500)* is. We need to add in the files *delays.h* to the *Header Files* and *delays_utilities.c* to the *Source files* of the project, as follows.

    Right click on *Header Files -> Add Existing Item…*

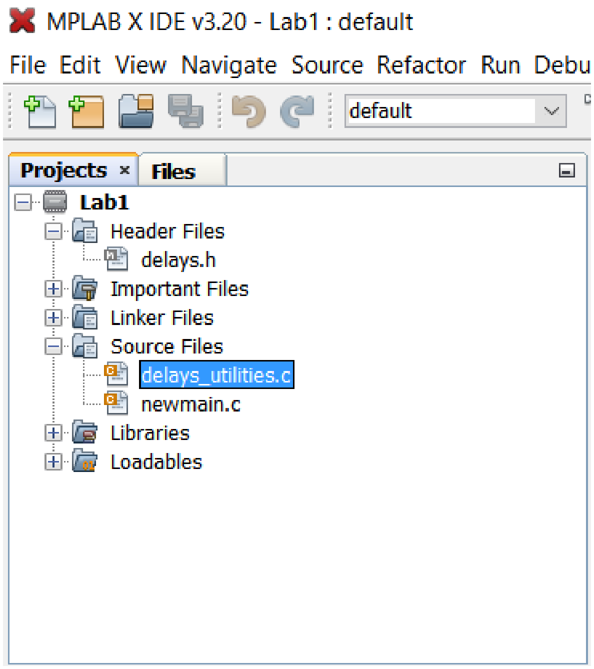


35. In the *Select Item* window that appears, browse to the folder *D:\ProjectX\Utilities* and select the file *delays.h* (the extension may not appear).

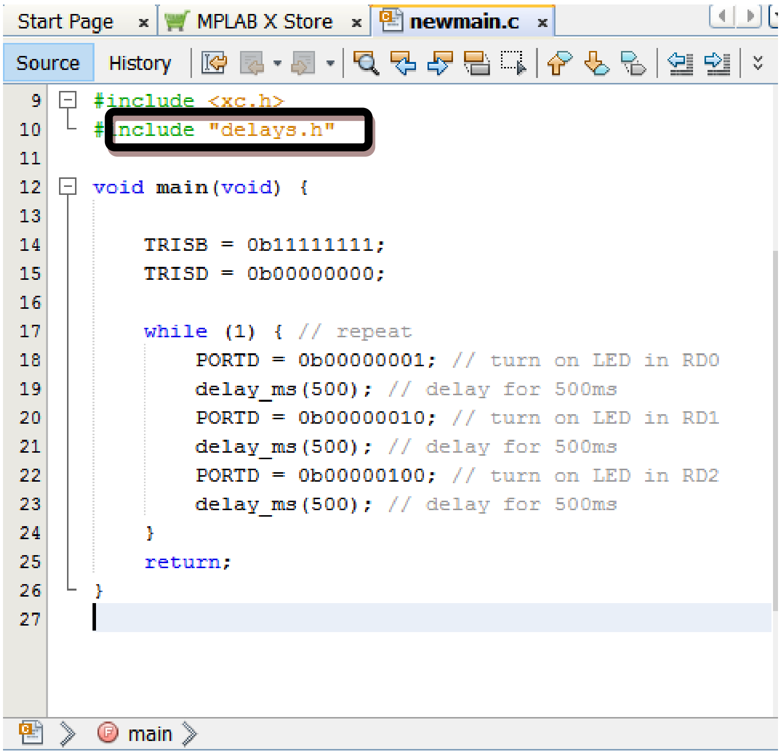
Click *Copy* to get a copy of this file and then click *Select*.



36. The file *delays.h* is now added to the *Header Files*. Next, add the file *delays_utilities.c* to the *Source Files*. After this, you should see:



37. We also need to add the *#include "delays.h"* statement to the C program.



Build the project again and it should be successful. Download this program to the board and the LEDs should be lighting up one at a time with a delay of about half a second.

38. Try to slow down the rate of blinking by increasing the delays. (Hint: change all the values 500 to 1000).

Rebuild and download the program.

39. Then, make the LEDs blink faster and faster until it stops blinking.

Rebuild and download the program.

40. Finally, make the LEDs blink together by turn on and off all three LEDs at the same time. Use any reasonable delay of your choice.

Rebuild and download the program.

```c
// file : delays.h

#define _XTAL_FREQ 48000000

extern void delay_ms(unsigned int i);  // delay in milli-secs, up to
                                        //max 65535 ms

extern void delay_us(unsigned int i);  // delay in micro-secs, up to
                                        //max 65535 us
```

```c
/*
* File:   delays_utilities.c
*
* Created on 13 January, 2016, 1:31 PM
*/

#include <xc.h>
#define _XTAL_FREQ 48000000

void delay_ms(unsigned int i)
{ unsigned int j;
      if(i!=0)  // check for i=0
      for(j=0;j<i;j++)__delay_ms(1); // call __delay_ms(1) x i times
}

// this delay is too short to be accurate - good luck.
void delay_us(unsigned int i)
{
      unsigned int j,lower;
      // for micro sec, the looping takes too long
      // so split into two parts, 20 seems to work fine
      lower = i;
      lower = lower/20;

      if (i< 5)
      {
            return; // too short no delay
      }
      else if (i<10)
      {
            __delay_us(7); // delay is 5 to 9 so just pick 7
      }
      else if (i< 20)
      {
            __delay_us(15); // delay is 10-19
      }
      else
            for(j=0;j<lower;j++)__delay_us(20);

}
```