ECE3301L MPLAB and PICKIT Tutorial

The development tool used in this tutorial is

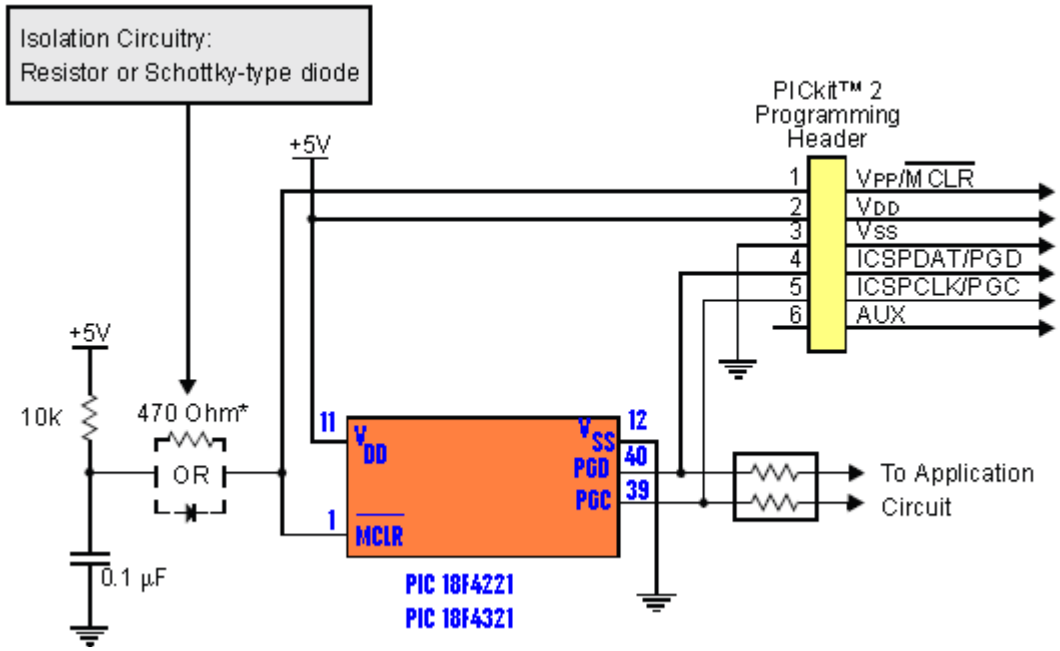
1. PICKit4 or PICKit3

   http://www.microchipdirect.com/productsearch.aspx?Keywords=PG164130

PART 1) Understanding In-Circuit Serial Programming:

The PICkit3/Pickit4 Development Programmer/Debugger can program microcontroller devices that are installed in an application circuit using In-Circuit Serial Programming (ICSP). ICSP requires five signals:

1. VPP . Programming Voltage; when applied, the device goes into Programming mode. (pin 1)
2. ICSPCLK or PGC . Programming Clock; a unidirectional synchronous serial clock line from the programmer to the target. (pin39)
3. ICSPDAT or PGD . Programming Data; a bidirectional synchronous serial data line.(pin40)
4. VDD . Power Supply positive voltage.(pin 11)
5. VSS . Power Supply ground reference. (pin 12)



A normally-open push-button switch can be added to reset the microcontroller.

ISOLATE VPP/MCLR/PORT PIN
When VPP voltage is applied, the application circuit needs to take into consideration that the typical VPP voltage is +12V. This may be an issue in the following situations:

If the VPP pin is used as a MCLR pin

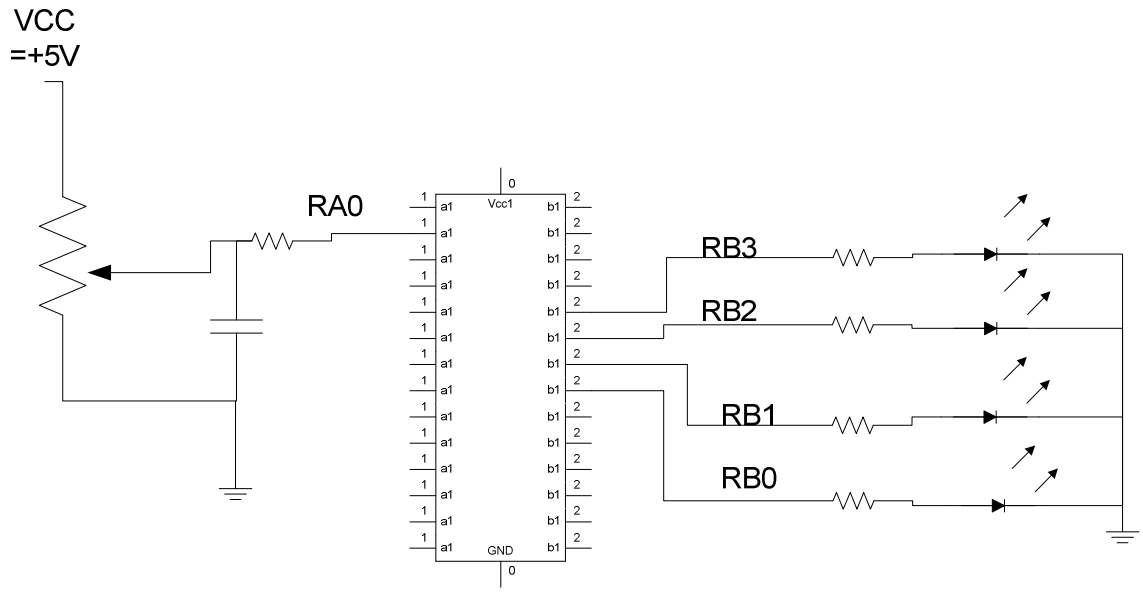
The application circuit is typically connected to a pull up resistor/capacitor circuit, as recommended in the device data sheet. Care must be taken so that the VPP voltage slew rate is not slowed down and exceeds the rise time in the programming specification (typically 1 □s). If a supervisory circuit or a push button is interfaced to the MCLR pin, it is recommended that they be isolated from the VPP voltage by using a Schottky-type diode or limiting resistor.

If the VPP pin is used as an I/O port pin
The application circuit that connects to the I/O pin may not be able to handle the +12V voltage. It is recommended to use a Schottky-type diode or limiting resistor as shown.

ISOLATE ICSPCLK OR PGC AND ICSPDAT OR PGD PINS
The ICSPCLK or PGC and ICSPDAT or PGD pins need to be isolated from the application circuit to prevent the programming signals from being affected by the application circuitry. PGC is a unidirectional synchronous serial programming clock line from the programmer to the target. PGD is a bidirectional synchronous serial programming data line. If the design permits, dedicate these pins for ICSP. However, if the application circuit requires that these pins be used in the application circuit, design the circuitry in a manner that does not alter the signal level and slew rates. Isolation circuitry will vary according to the application.

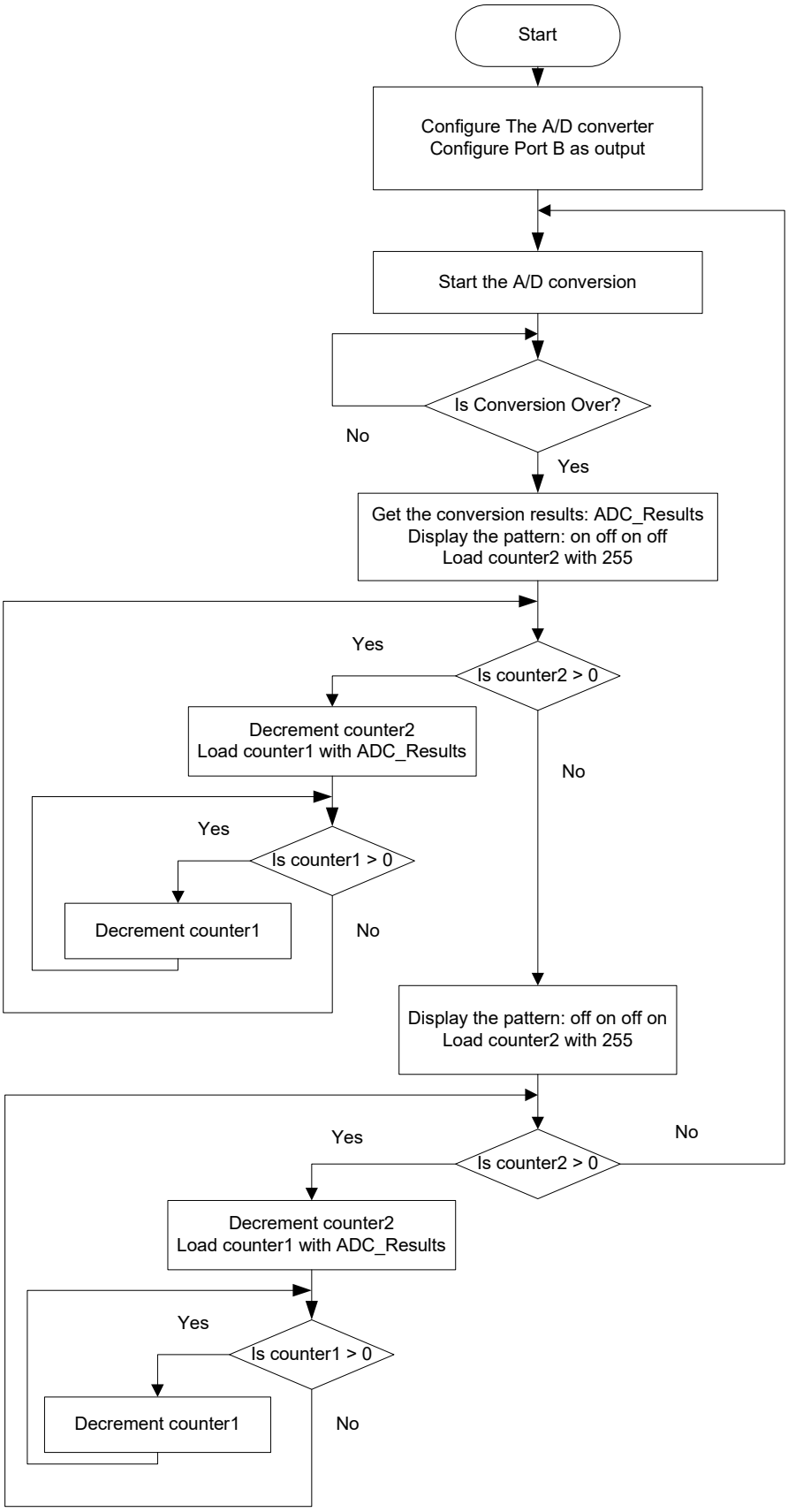
PART 2) Understanding the MPLAB X software and XC8 compiler:

The objective of this tutorial is to use Microchip development tools to design and implement a simple project. An analog voltage will be used to control the speed of flashing four LEDs. The analog voltage is applied to input RA0 and the four LEDs are connected to the least significant four bits of PORT B as shown below.



Note: The picture above is just a quick representation of the circuit to be used on this tutorial. Refer to the real schematics attached to the lab #1 that will show all the proper connections.

The algorithm for the program is described by the following flow chart:


Start
Configure The A/D converter
Configure Port B as output
Start the A/D conversion
Is Conversion Over?
No
Yes
Get the conversion results: ADC_Results
Display the pattern: on off on off
Load counter2 with 255
Yes
Is counter2 > 0
Decrement counter2
Load counter1 with ADC_Results
No
Yes
Is counter1 > 0
Decrement counter1
No
Display the pattern: off on off on
Load counter2 with 255
Yes
Is counter2 > 0
No
Decrement counter2
Load counter1 with ADC_Results
Yes
Is counter1 > 0
Decrement counter1
No

Here is the source code of the software described on the flow chart above: (to be used later)

```
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include <math.h>
#include <p18f4620.h>

#pragma config OSC  =   INTIO67
#pragma config WDT  =   OFF
#pragma config LVP  =   OFF
#pragma config BOREN  =   OFF

#define delay 5
// Prototype Area to place all the references to the routines used in the program
void Init_ADC(void);
unsigned  int Get_Full_ADC(void);
void Flash_LED(unsigned char);

void main(void)
{
 unsigned int ADC_Result;                  // local variable to store the result
     Init_ADC();                           // initialize the A2D converter
    TRISB =0x00;                           // make PORTB as all outputs
    while(1)
    {
        ADC_Result = Get_Full_ADC();       // call routine to measure the A2D port
        Flash_LED(ADC_Result);             // call routine to flash the LED based on the delay
                                           // indicated by ADC_Result
    }
}

void Init_ADC(void)
{
    ADCON0=0x01;                           // select channel AN0, and turn on the A2D subsystem
    ADCON1=0x0E;                           // set pin 2 as analog signal, VDD-VSS as reference voltage
                                           // and right justify the result
    ADCON2=0xA9;                           // Set the bit conversion time (TAD) and acquisition time
}

unsigned int Get_Full_ADC(void)
{
int result;
    ADCON0bits.GO=1;                        // Start Conversion
    while(ADCON0bits.DONE==1);             // Wait for conversion to be completed (DONE=0)
    result = (ADRESH * 0x100) + ADRESL; // Combine result of upper byte and lower byte into
    return result;                         // return the most significant 8- bits of the result.
}
```

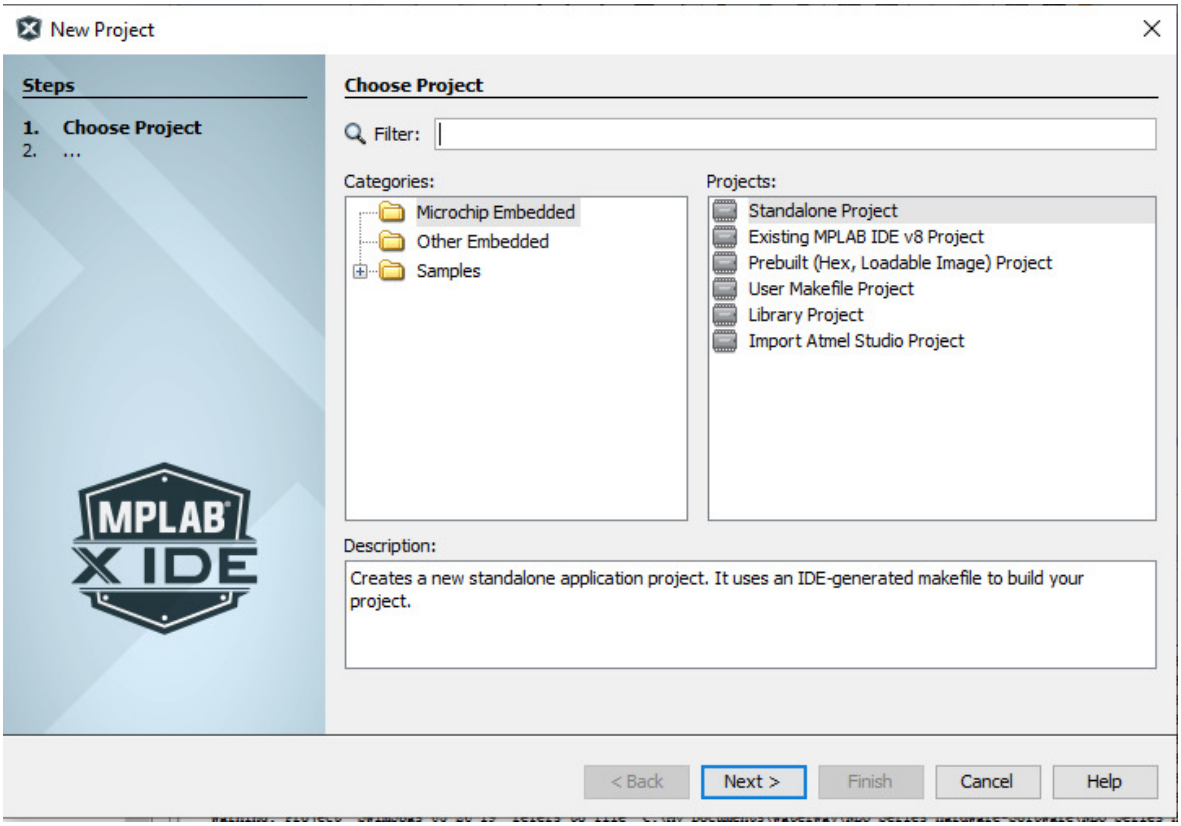
```
void Flash_LED(unsigned int ADC_result)
{
unsigned int counter1, counter2;
    LATB = 0x0A;                        // output to PORTB the pattern 00001010
                                        // delay loop
    for (counter2=delay; counter2>0; --counter2)
    {
       for (counter1=ADC_result ; counter1>0; -- counter1);
    }

    LATB = 0x15                         // output to PORTB the pattern 00010101
                                        // delay loop
    for (counter2=delay; counter2>0; --counter2)
    {
       for (counter1=ADC_result ; counter1>0; -- counter1);
    }
}
```

Let us start the MPLABX IDE software.

From the menu bar, ***File> New Project.***

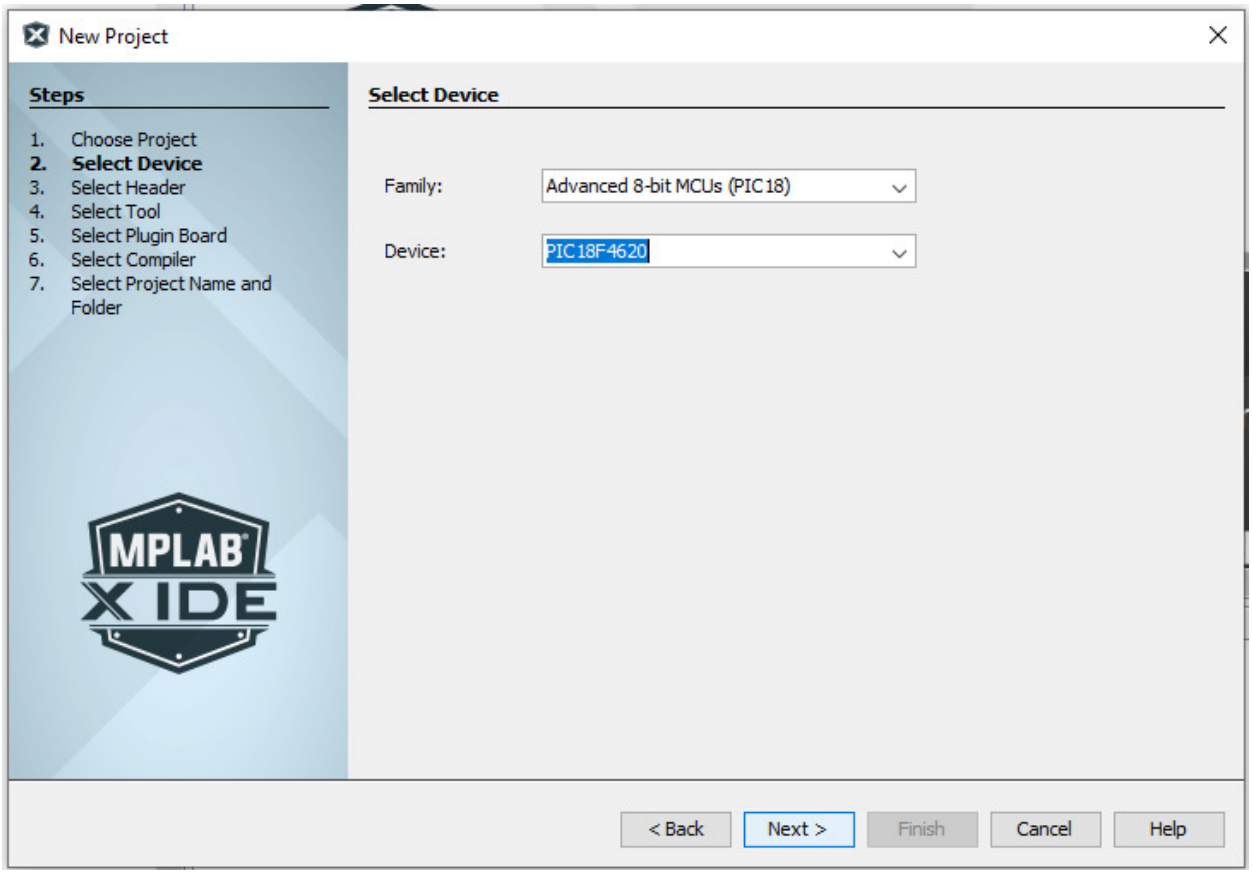
Select "*Microchip Embedded*" from **Categories** and "*Standalone Project*" from **Projects**.



Click ***Next***

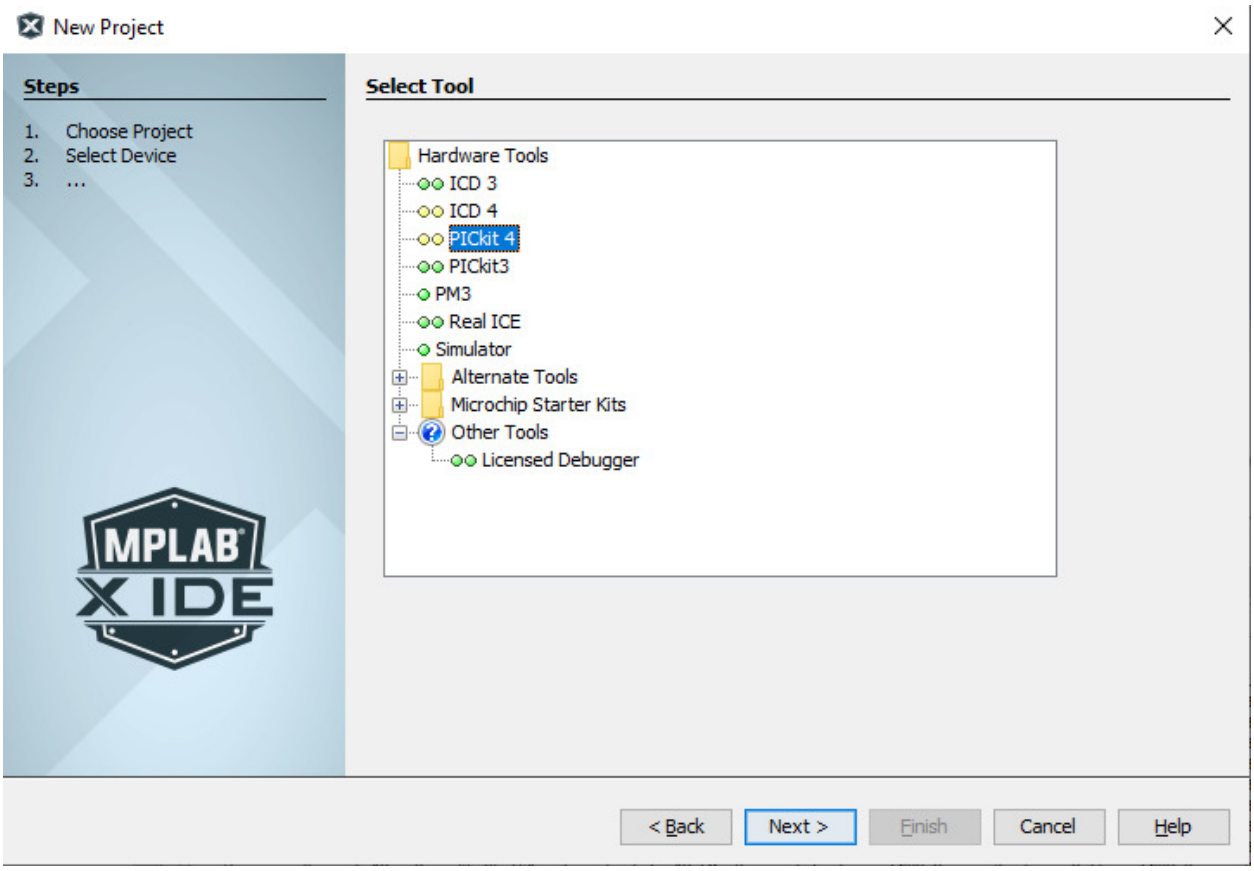
Go the field next to the “Family:”. Select the option “*Advanced 8-bit MCUs (PIC18)*”.

Go the field next to the “Device:”. Scroll down until you find “*PIC18F4620*”.



Click ***Next***

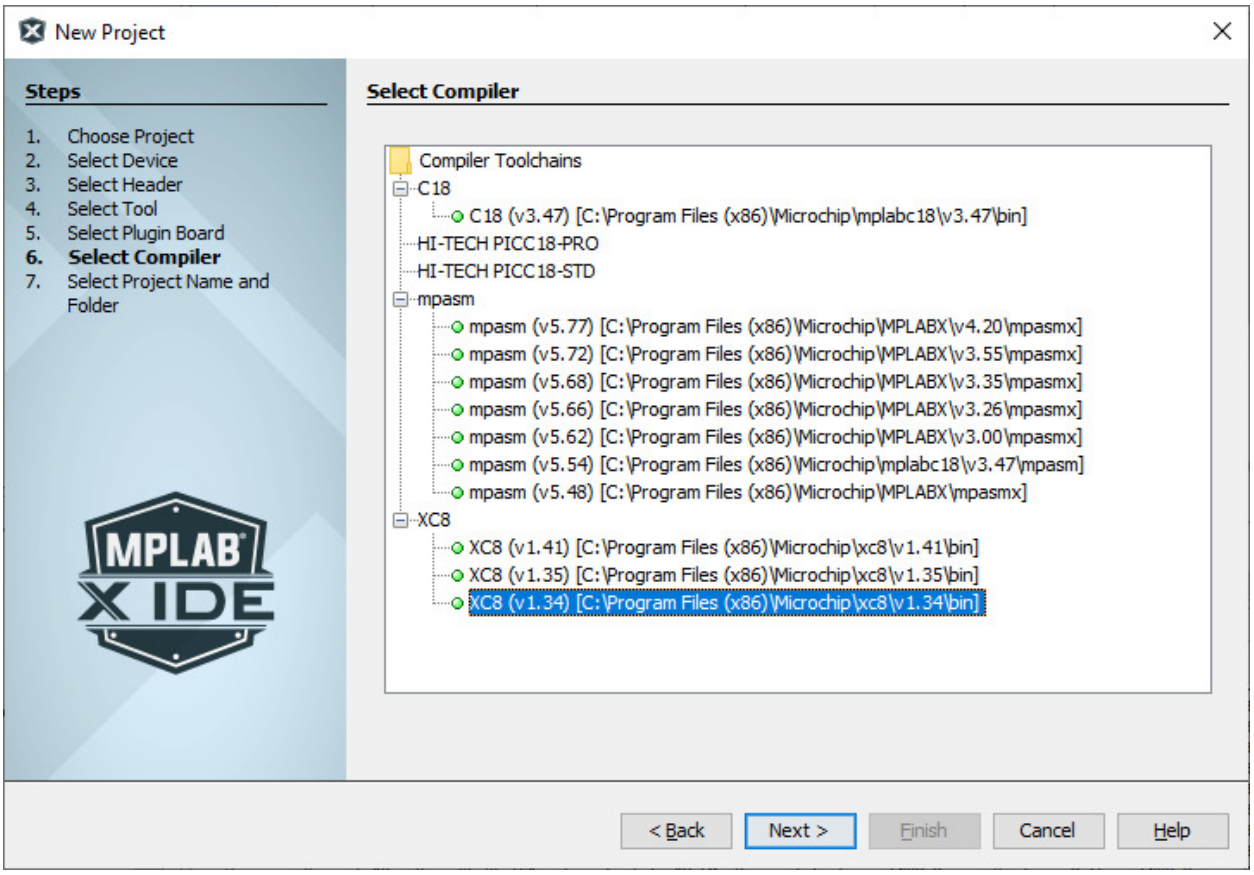
A new page will ask for the hardware tool to be used. Select “PICkit4” or ‘PICkit3’ (based on what programmer you have on-hand) under Select Tool



Click ***Next***

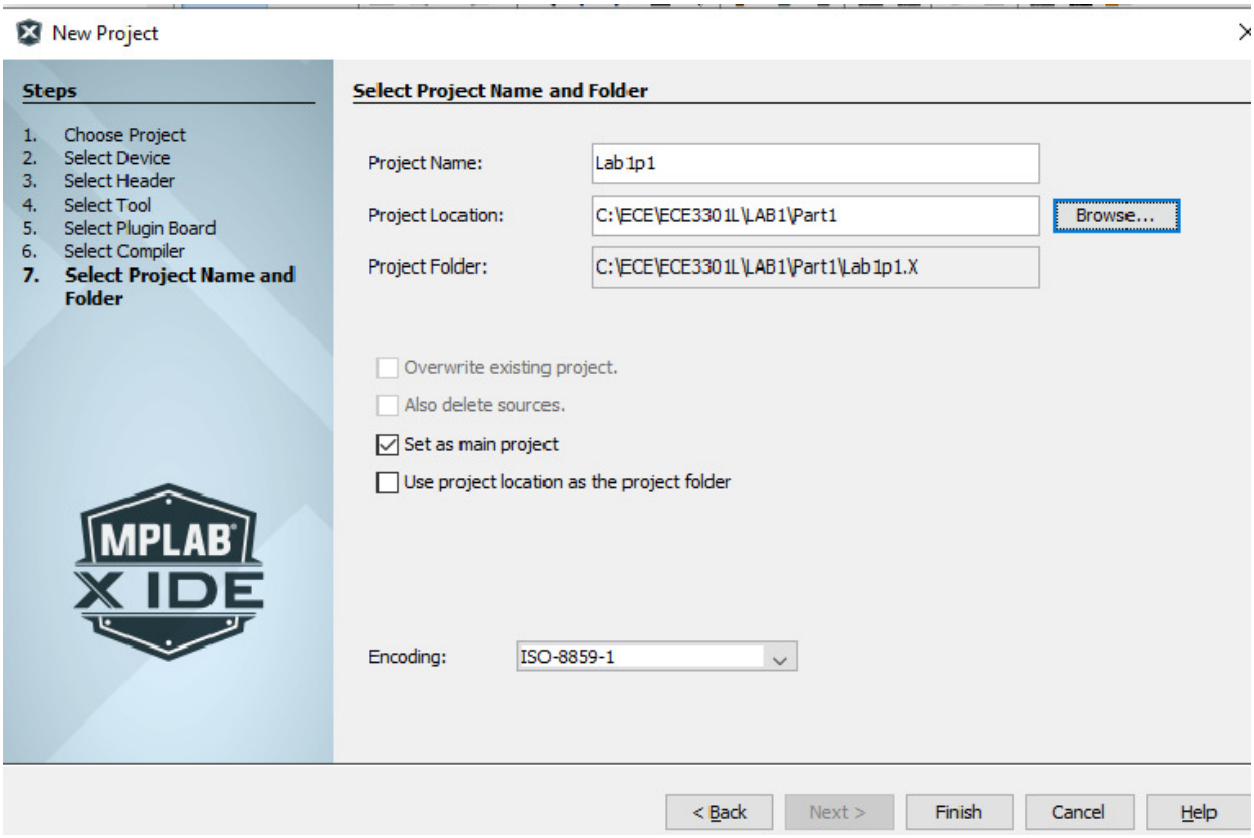
Select under the Compiler Toolchains and XC8 **Tool Suite.** Make sure to select Version 1.34 if there are more versions shown:

Note: You must install the XC8 compiler before the running of this MPLABx software. If not, abort, install the compiler and restart this process again.



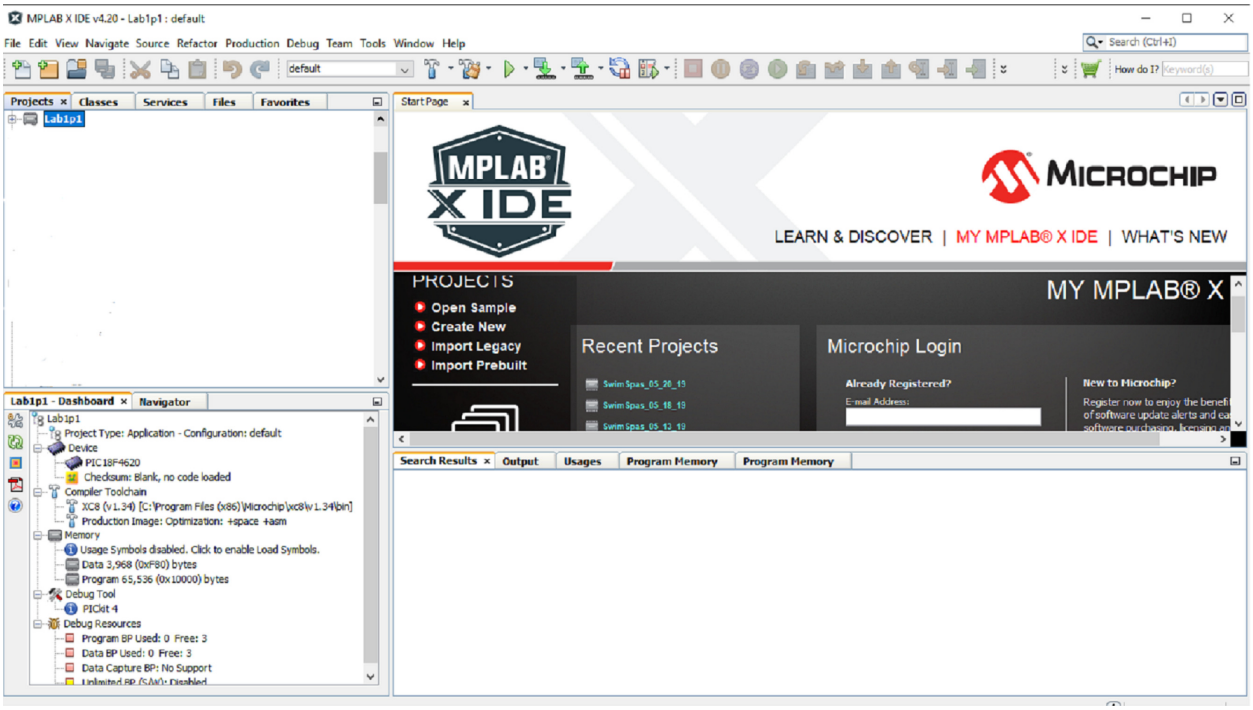
Click ***Next***

Next, type in the project's Name. Call the first project to be 'Lab1p1' for now. **It is recommended to create a top level folder 'ECE' and then a sub-folder called 'ECE3301L'**. Since we are going to make several projects, create a sub-folder 'Lab1'. Then, we might have more than one part in a lab. Next create a new sub-folder called 'Part1'. You should have something like:

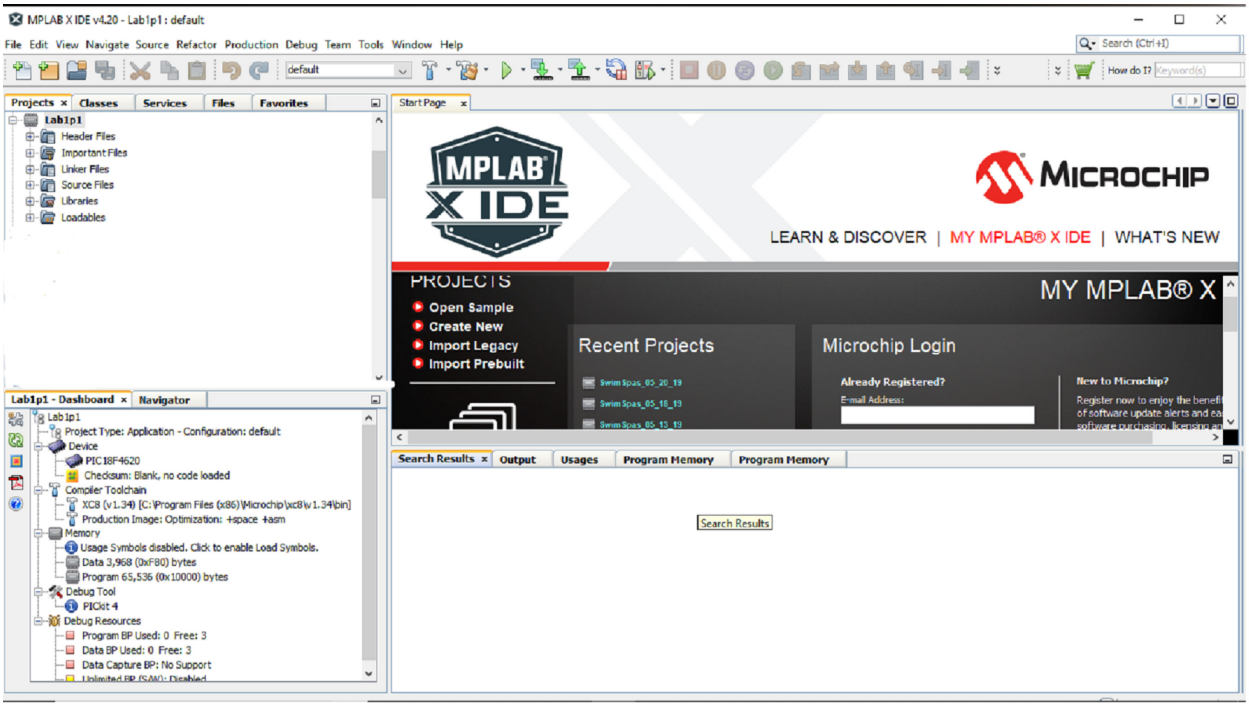


Click ***Finish.***

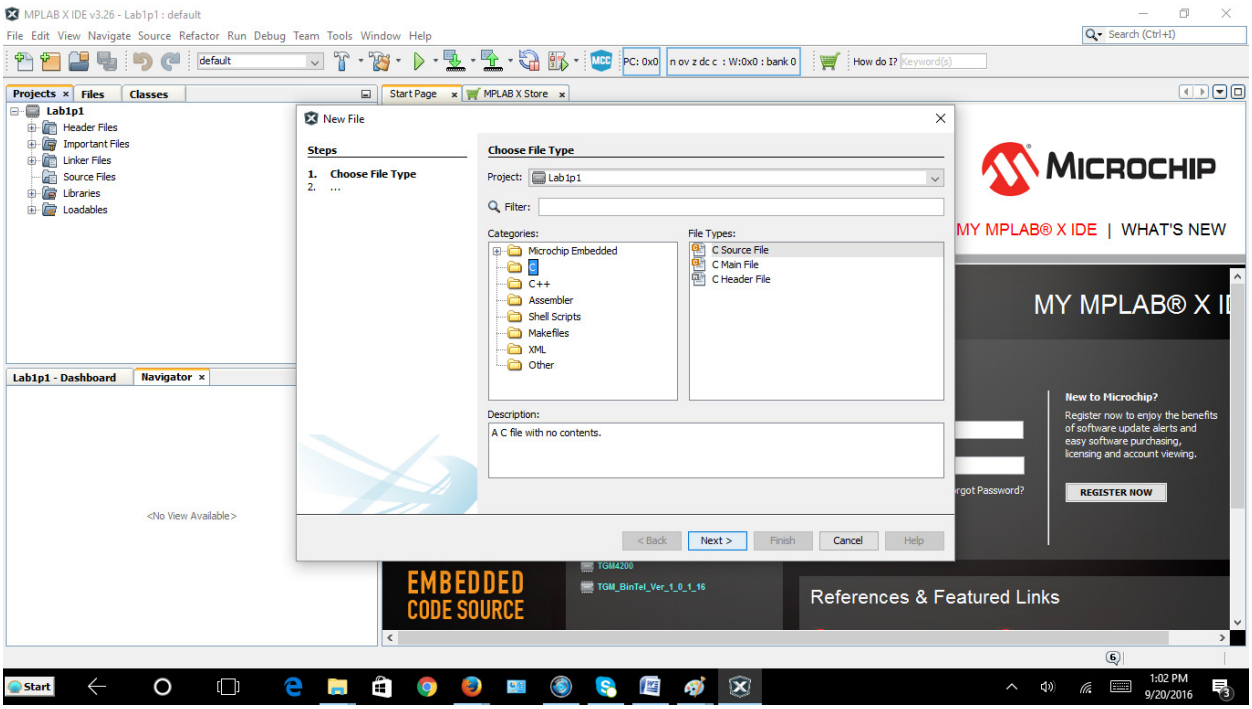
After some processing times, the screen will look like:



Expand the '+' sign at 'Lab1p1' under the Project tab to expose the sub-items:

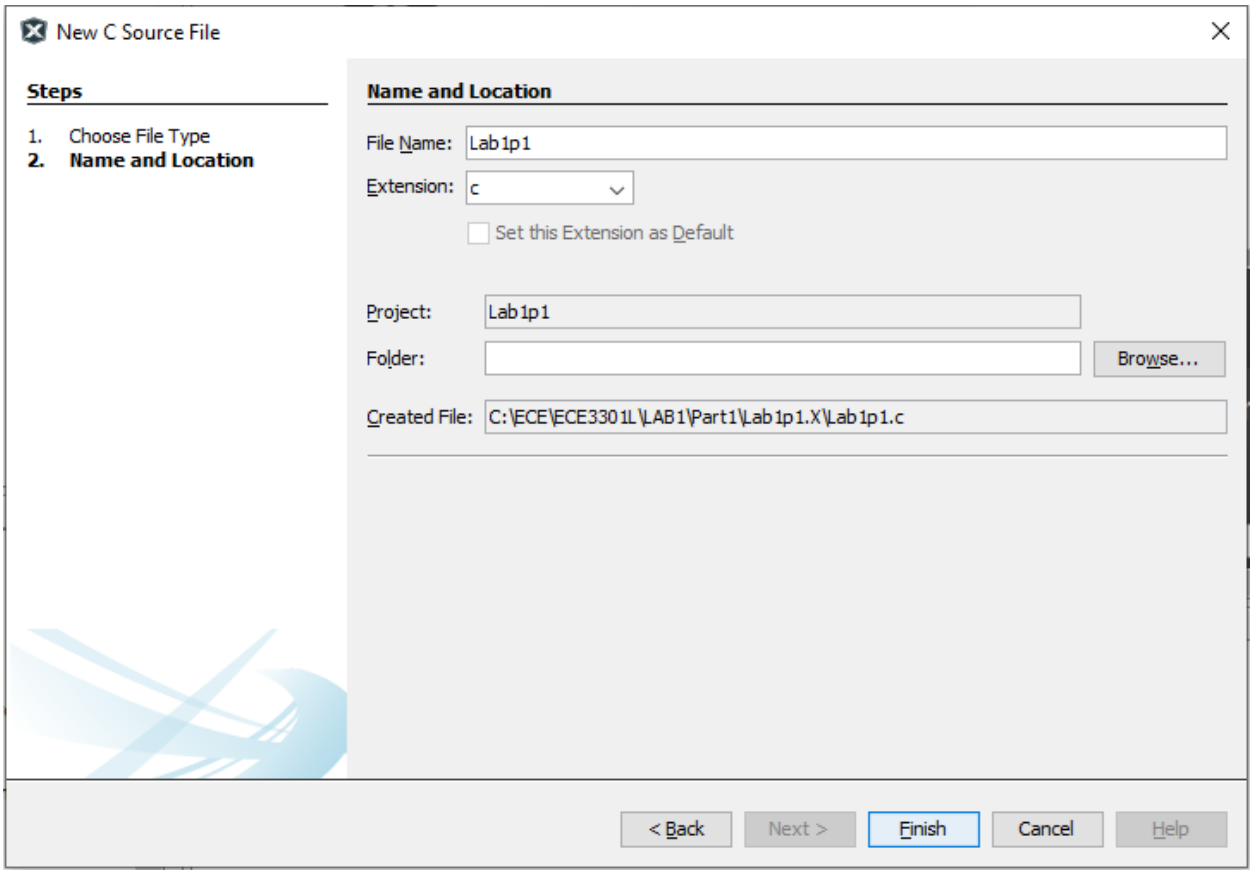


Next step, select *File>New File*. A new window will appear.

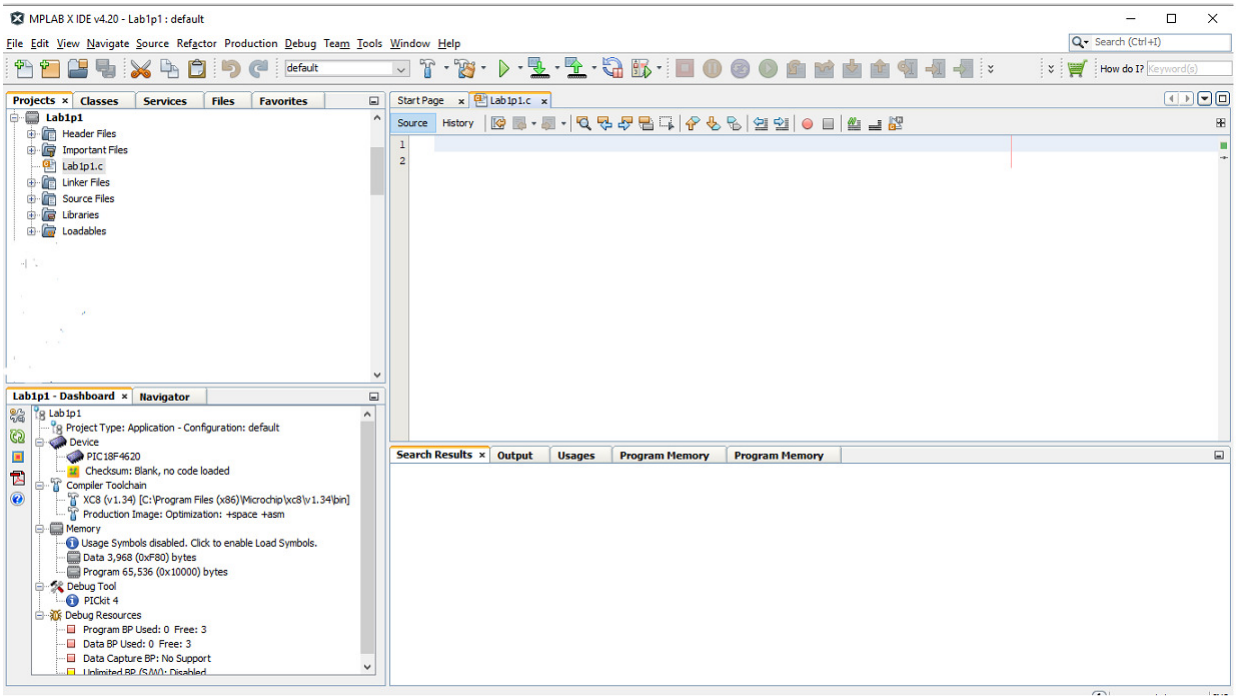


Select ‘C’ under ‘Categories:’ and ‘C Source File’ under ‘File Types:’. Hit *Next*.

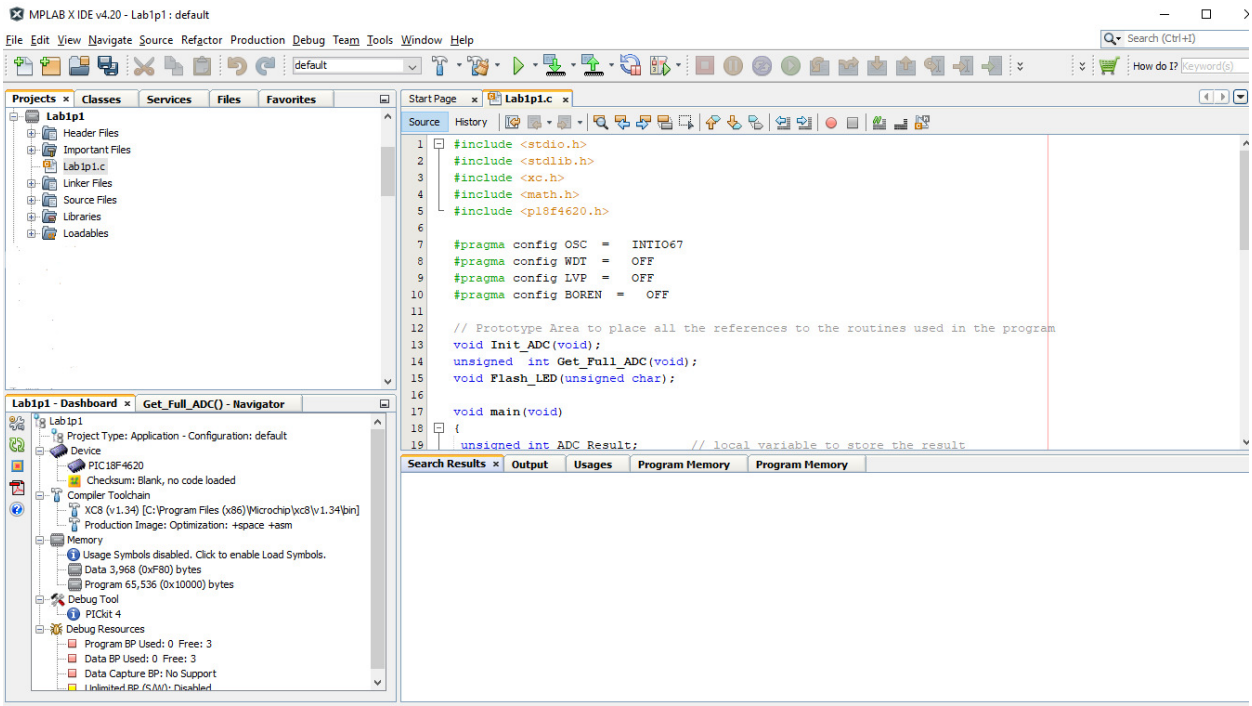
Enter ‘Lab1p1’ in the ‘File Name:’ field and hit *Finish*:



A new screen will show up:

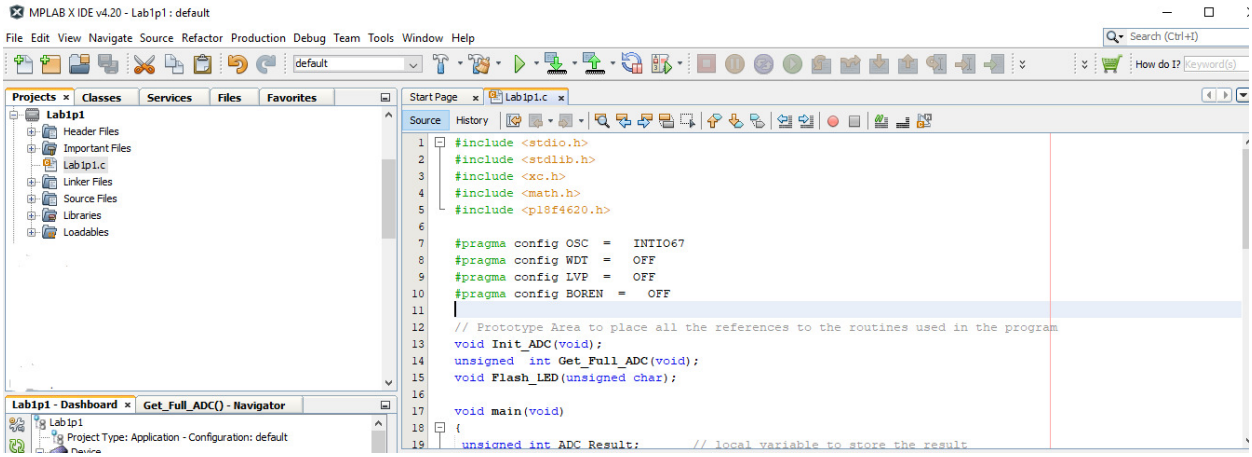


Go back to the top of this document where a display of the c source code was included. Copy the entire source code and paste it into the screen.

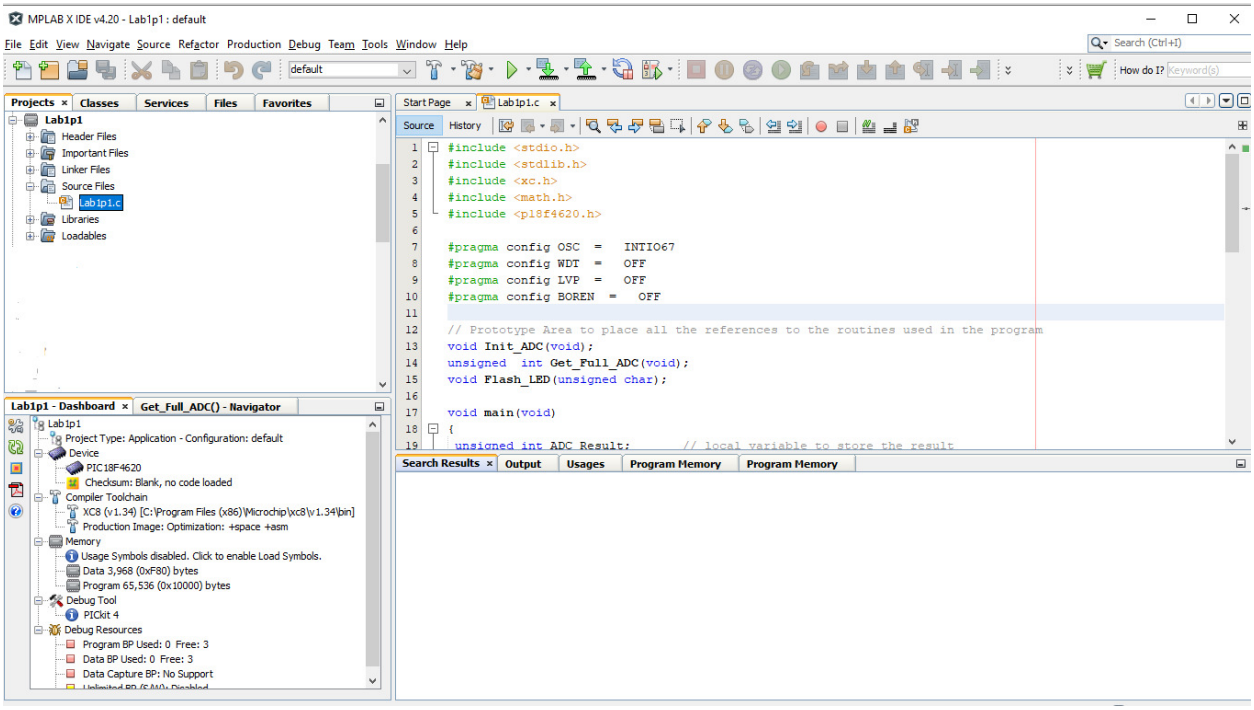


Save the file by executing *File and Save:*

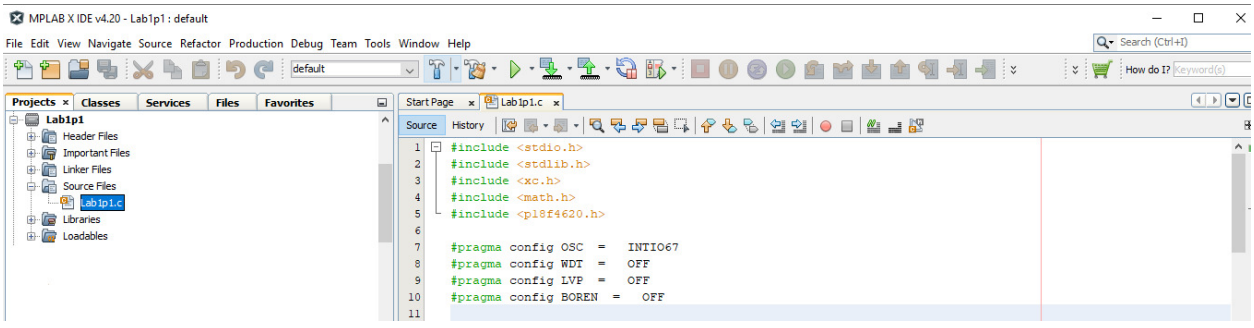
The screen will now look:

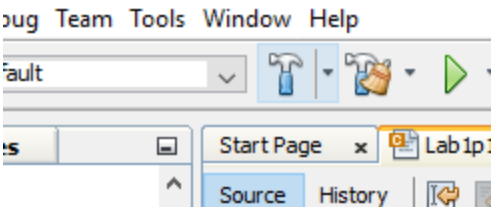


Notice that the file ‘Lab1p1.c’ is under the ‘Important Files’. Use the mouse to select that file and drag it into the ‘Source Files’. The file ‘Lab1p1.c’ will show up under the ‘Source Files’.

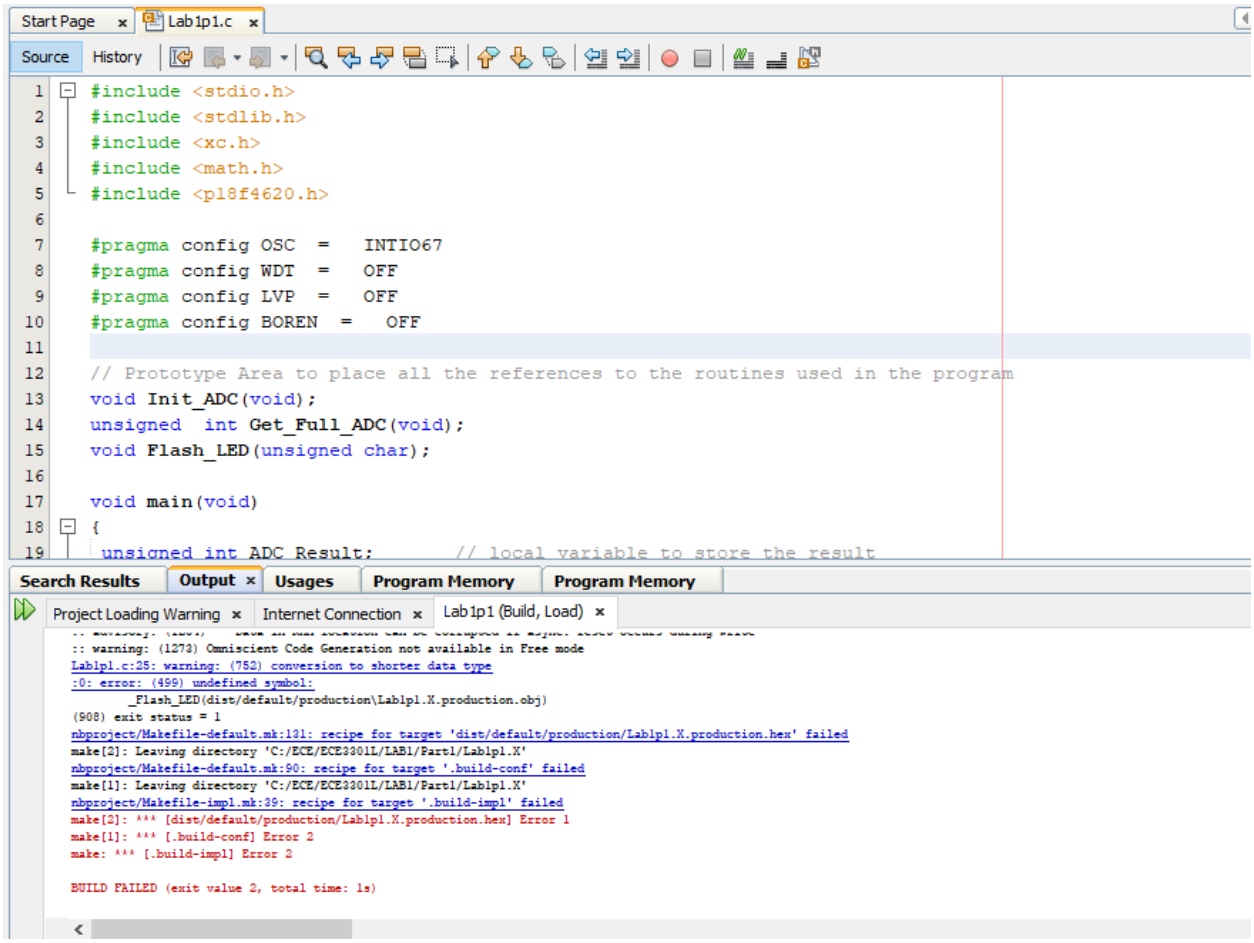


Next, click on the leftmost ‘Hammer’ symbol to build the code (just below the ‘Window’):



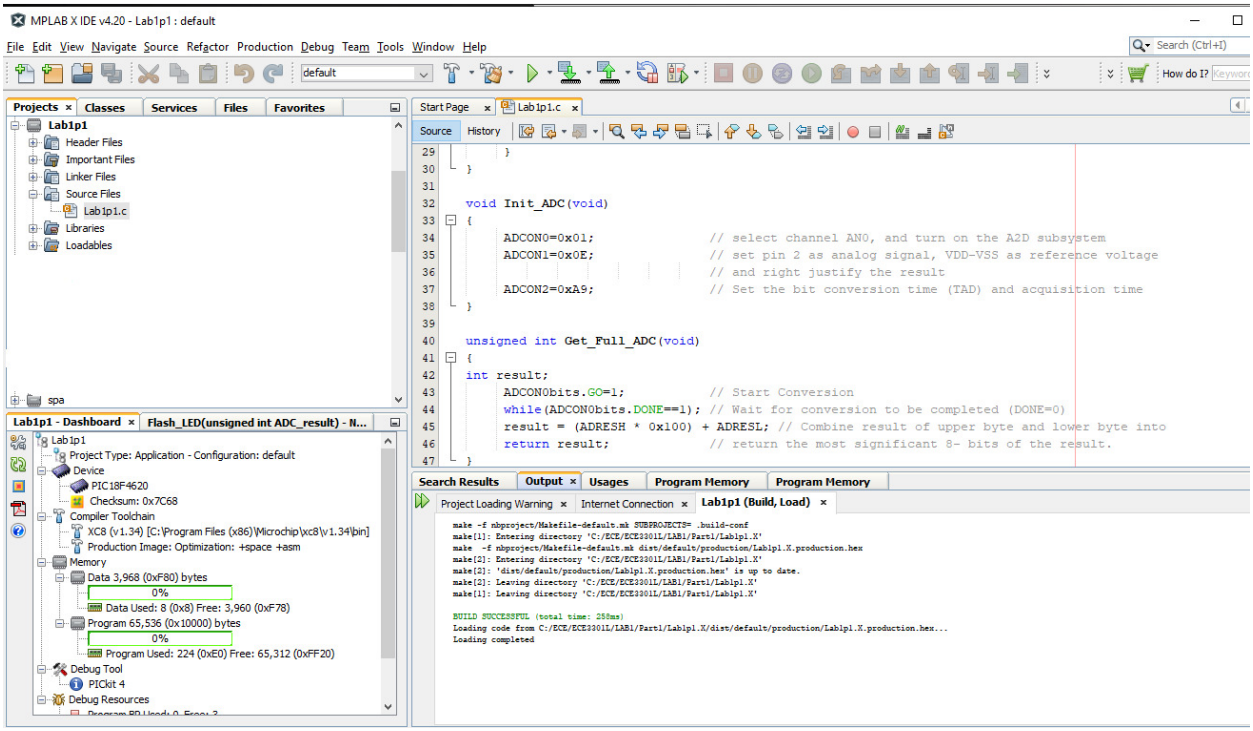


By design, the provided source code has two embedded errors and the compilation will fail:



Fix the errors (2 of them) until the compilation is clean. Click on each line that shows the ‘error:’ message and the program will point to you where the error is.

When ‘BUILD SUCCESSFUL’ message is shown then there is no more error.

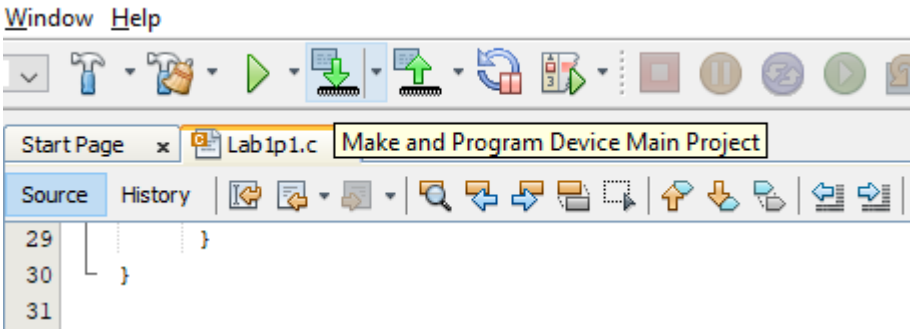


The next part is to program the code onto the prototype board.

You cannot perform the steps below until you have on hand the development board from me.

First, make sure to plug the PICKit3/PICKit4 onto the header of the prototype board. Line up the arrow on the PICKitx to the arrow on the development board provided.

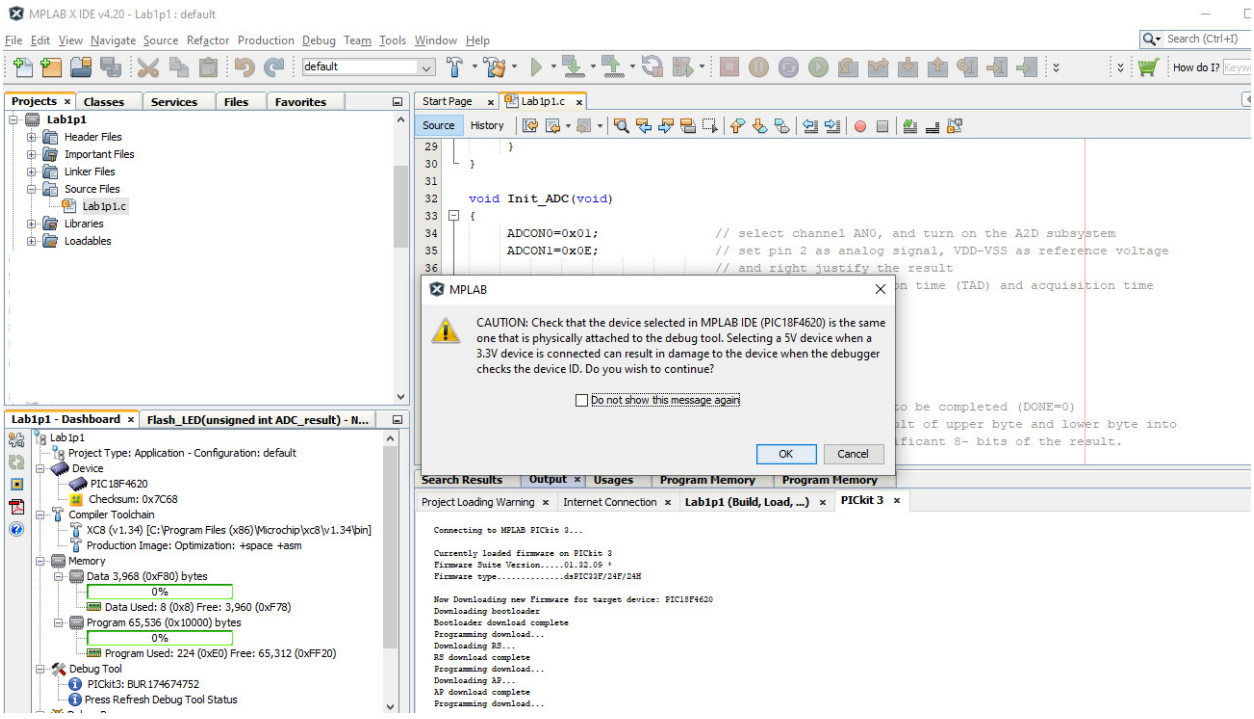
Next, go back to the MPLABx software and move the cursor to the Down arrow dial button which is located a few dial buttons away from the ‘Hammer’ dial (to the right).



Click on that dial. The software will start the programming process.

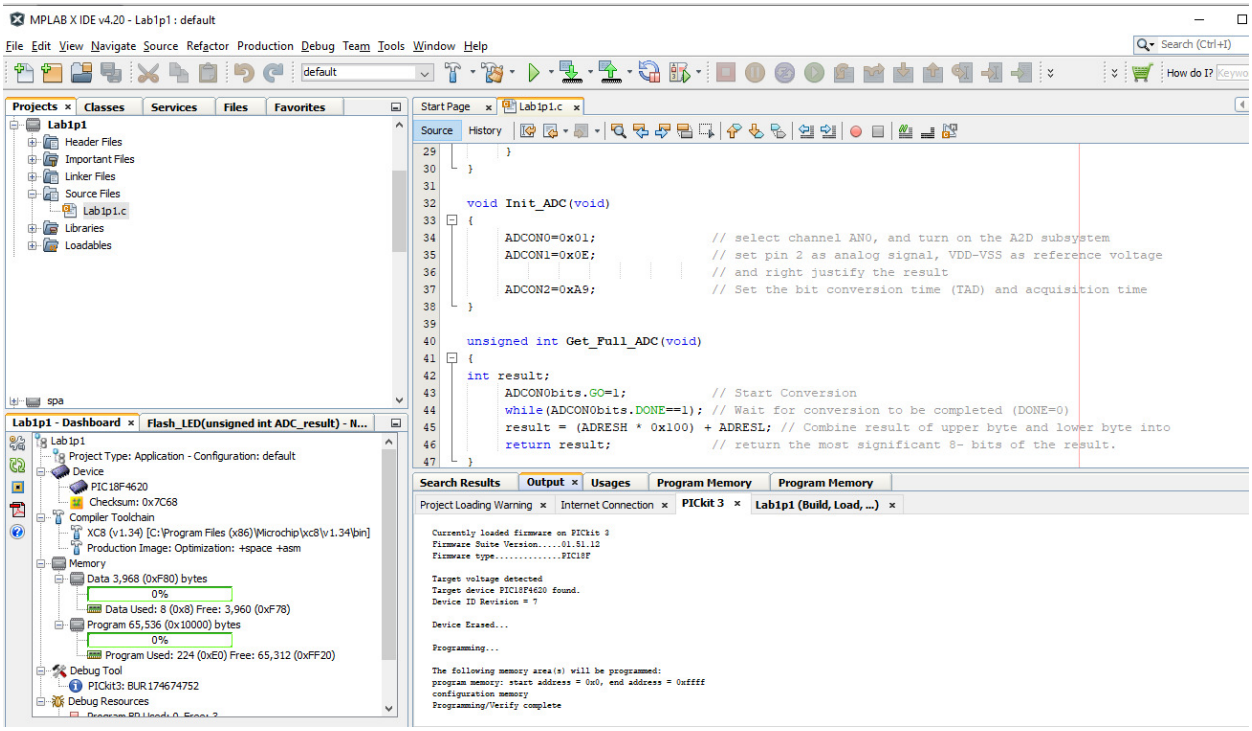
Note: if your PICKit3/PICKit4 is used for the first time, the MPLAB X IDE software will start to download some software to your PICKitx and it will take a few minutes. After the first initialization is done, that process will not be repeated and the direct programming of the prototype board will always be executed.

In addition, the following screen may show up:



Check the ‘Do not show this message again’ and hit ‘OK’. That is just a warning message and it will not show up again if you check off that message (most cases).

The final message should be:



The board is now programmed with the intended software and you should check the output of the LEDs.