

# EE 2361 - Introduction to Microcontrollers

*Laboratory #1*

*Initial software and hardware setup*



## Background

Modern microcontrollers are incredibly powerful and complex. Much of the time you'll find third parties (Adafruit) will have created advanced libraries or Application Programming Interfaces (APIs) that do everything you want. In the case of these libraries, many many people have come before and completed the same or a similar task. Unfortunately, it's hard to get paid for doing a job that many people have done many times before.

This course will teach you how to take the next step toward being a development engineer. You'll be able to read a datasheet, implement your own custom features, and wrap those in externally accessible functions that are easy for future engineers and programmers to work with.

The courses labs are organized into 4 stages.

- 1.) Easy Lab, full walk through, software/hardware online
- 2.) Moderate support, teach the basics
- 3.) Limited Support, work off datasheets, discover on your own
- 4.) Project, do something independant/new

## Purpose

In this lab you will prepare both the hardware and software you will need throughout the rest of this course. You will install the MPLAB X IDE and the XC16 Compiler. Finally, you will load pre-generated code onto your microcontroller in order to test the functionality of your new board.

## Supplemental Resources

PIC24FJ64GA002 [Family Datasheet](#)

PIC24 [Programmer's Reference Manual](#)

## Required Component

1x 10k Ohm Resistor 1x 10uF Caps 2x 0.1uF Caps 6-pin Header Strip PIC24FJ64GA002 Microcontroller Red LED 1x 220 Ohm Resistor PIC programmer
--

## Self-Guided Lab

### Installation of MPLAB X, XC16 Compiler, and MCC

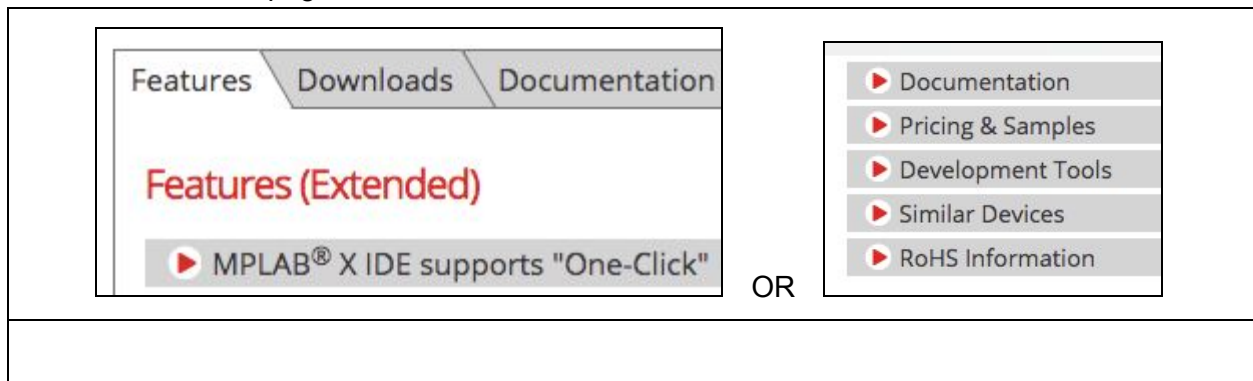
You will need to install two components to develop programs for your PIC24 in EE 2361.

- MPLAB X IDE - an Integrated Development Environment with simulation and debug capabilities
- The XC16 Compiler - a manufacturer specific compiler that converts C (.c files) and Assembly (.s files) into machine code (.hex files)

Both of these applications need to be downloaded and installed separately.

### Install MPLAB X

Start at <http://www.microchip.com/mplab/mplab-x-ide>, you'll notice there is a table with headings at the bottom of the page, click on "Downloads", as show below:

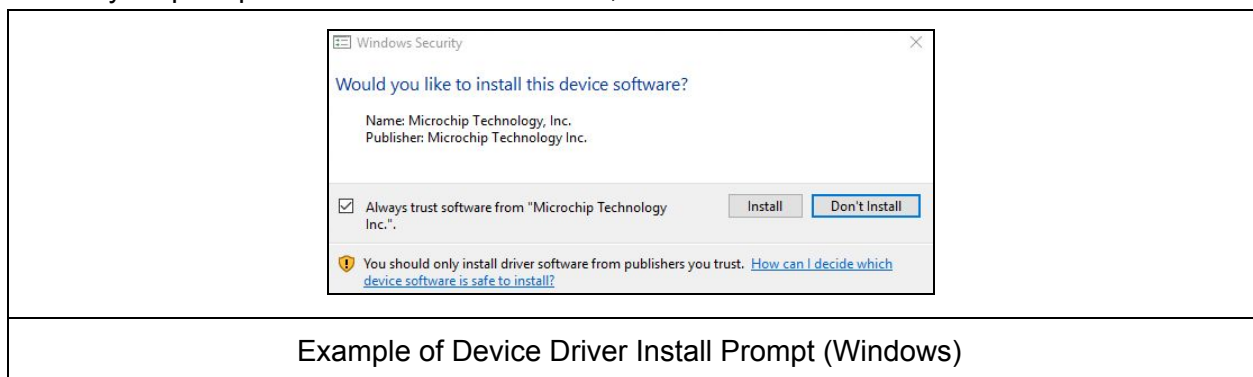


**NOTE:** These tables and "twiddles" are where you will find the vast majority of useful information from Microchip. Try Googling for our microcontroller "PIC24FJ64GA002" and "microchip XC16" and you'll find documentation, downloads, examples, and other useful information.

Once the download is complete:

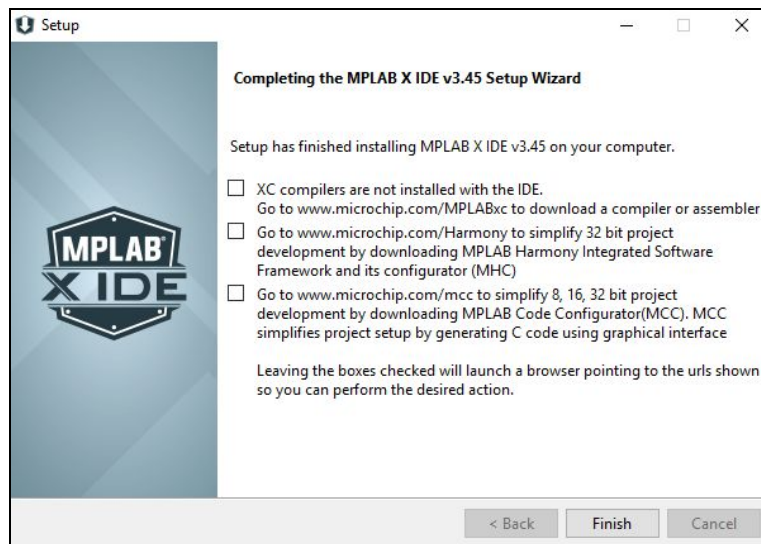
- 1.) Run the install program,
- 2.) Give the installer unlimited admin privileges (and your first born child), when appropriate accept Microchip's EULA (sell your soul)
- 3.) Click through the remaining prompts as appropriate.

You may be prompt to install "device software", like so:



These are drivers for Microchips programmers and debuggers, click “Install”. Leaving “Always trust...” click is ok, but up to you.

- 4.) At the end of the install, Microchip will offer to forward you to several useful websites. We will be manually going to each of these as needed in the labs, uncheck them for now and click “Finish”



Example of end of installer dialog box

## Install the MC16 Compiler

Start at <http://www.microchip.com/mplab/compilers>, click on the “Downloads” tab, and download the latest version of the XC16 compiler for your OS (not the 32-bit or 8-bit versions!). The figure below shows v1.30, if you see a newer version, download that, and not Version 1.30.




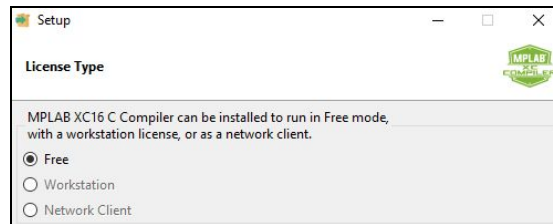
Additional Information Downloads Downloads Archive Documentation Compiler FAQs			
<b>Compilers</b>			
Title	Date Published	Size	D/L
<b>Windows (x86/x64)</b>			
MPLAB® XC8 Compiler v1.38	7/13/16	78.8 MB	
<u>MPLAB® XC16 Compiler v1.30</u>	12/2/16	76.5 MB	
MPLAB® XC32 Compiler v1.42	6/14/16	58.9 MB	
<b>Linux 32-Bit and Linux 64-Bit (Requires 32-Bit Compatibility Libraries)</b>			

Image of the Microchip XC16 compiler download site as of Dec. 2016

- 1.) When the download is complete, run the installer program
- 2.) Agree to anything Microchip asks, give the installer admin privileges, etc.
- 3.) When prompted select the “Free” license



License prompt

- 4.) Defaults should be ok for the remaining prompts, click next a lot and let the install task bar finish
- 5.) When “Installation Complete - Licensing Information” dialog appears, just click Next
- 6.) Click Finish

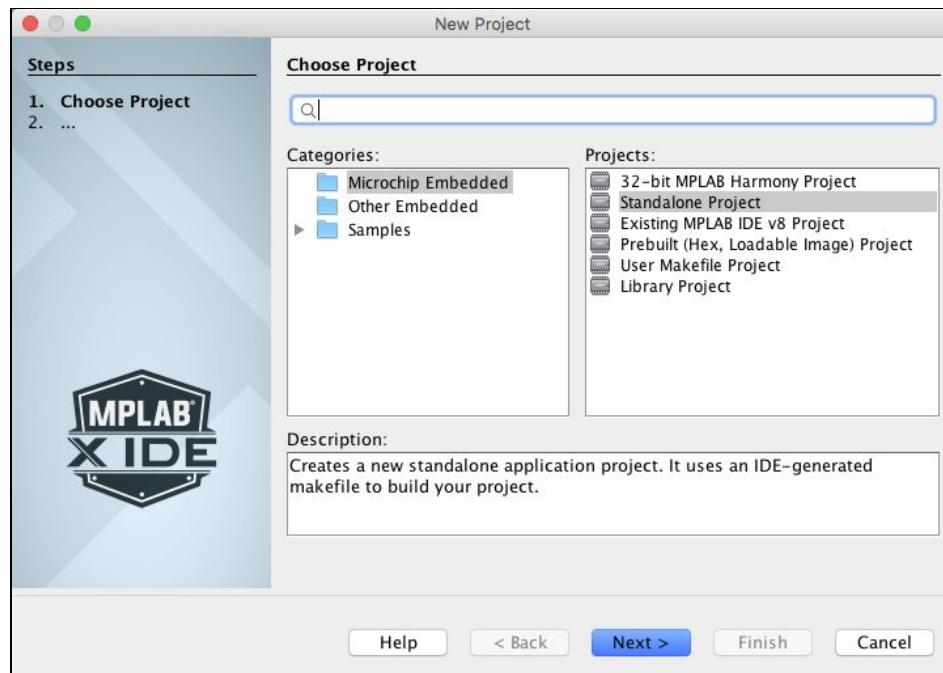
## Creating your first project in MPLAB X

In this section of the lab you will:

- Create a new project
- Create basic boilerplate code
- Toggle an output on your microcontroller
- Simulate your code

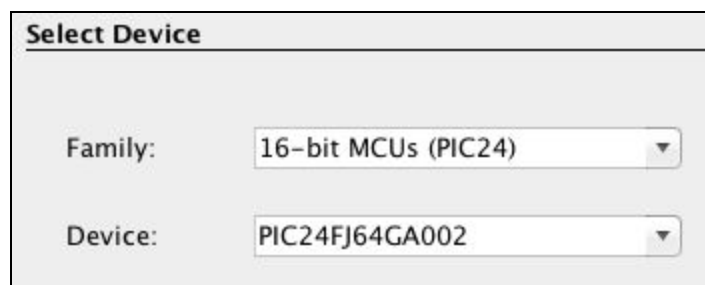
### Create a new project

- 1.) Start MPLAB X
- 2.) Click on “File” → “New Project...”
- 3.) You will be asked to choose a project template. Under Categories:, select “Microchip Embedded” Under Projects:, select “Standalone Project”, click “Next”.

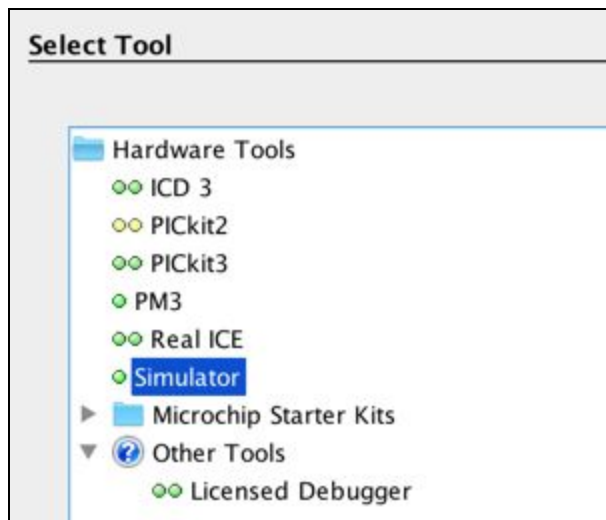


- 4.) You will be prompted to select the device you will be using. We are using the PIC24FJ64GA002 from the 16-bit MCU family as shown below

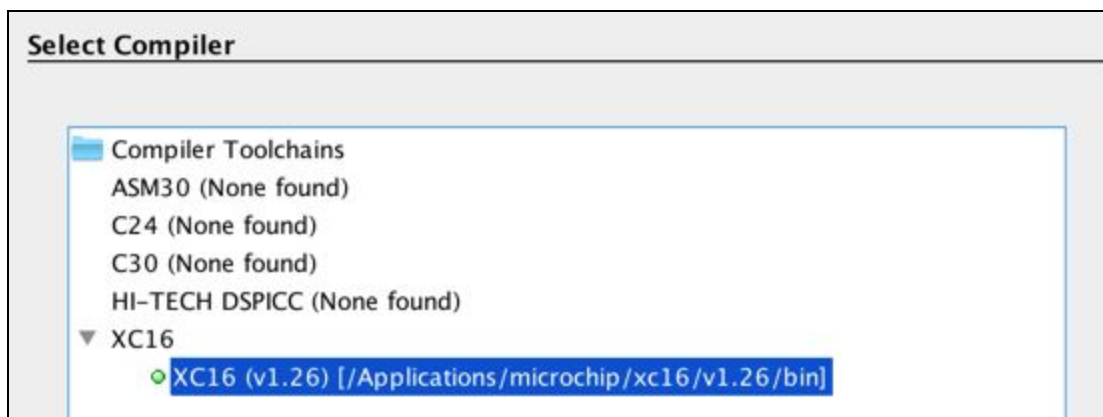
**NOTE:** It is often easier to type “PIC24FJ64” rather than pick it out of the pull down list.



- 5.) Select these options and click “Next”
- 6.) On the next screen, you are asked to “Select Header”. We will not be using a Debug Header, leave “None” selected, and just click “Next”.
- 7.) On the next screen, you will be asked to select a “Tool”. Three options are common, using the PICkit3, Simulator, OR Licensed Debugger (chipKIT). For now, just select the Simulator. It is generally preferable to do initial debug in the simulator. It easy to change later when you want to test your code on hardware.



- 8.) Select “Simulator” and click “Next”.
- 9.) On the next screen, you are asked to select a compiler. For this course, we will always select the XC16 compiler. (If this option is not available, then you didn’t install the compiler correctly. Please go back to section [Install the XC16 Compiler](#).) Your version should be v1.26 or higher.



- 10.) Give your project a name of the form <x500>\_Lab<#>\_v<number>, for example “orser\_Lab1\_v001”



### Select Project Name and Folder

Project Name:

Project Location:

Project Folder:

☐ Overwrite existing project.

11.) Click Finish

Create your first source file and add it to the project

- 1.) Select File → New File...
- 2.) Open the twiddle for Microchip Embedded, select XC16 Compiler, and mainXC16.c, click Next

#### Categories:

- Microchip Embedded
  - ASM30 Assembler
  - C18 Compiler
  - C30 Compiler
  - C32 Compiler
  - HI-TECH Compiler
  - MPASM assembler
  - XC16 Compiler**
  - XC32 Compiler
  - XC8 Compiler
- C
- C++
- Assembler

#### File Types:

- mainXC16.c

#### Description:

A source file with a very simple main function to build with XC16 for 16-bit MCUs.

- 3.) Give the file a unique name! Please name the files with the following format <x500>\_lab##\_<descriptor>\_v### (for example "orser\_lab1\_main\_v001".)





**NOTE:** During this course you may have open multiple files from different labs and projects. If every project has a file named “main.c” you will not be able to tell which project you are editing and you *\*will\** end up editing, compiling, and debugging the wrong file.

**Name and Location**

File Name:

Extension:

☐ Set this Extension as Default

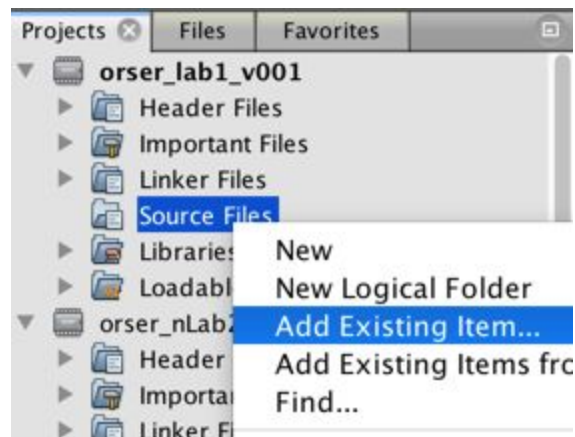
Project:

Location:

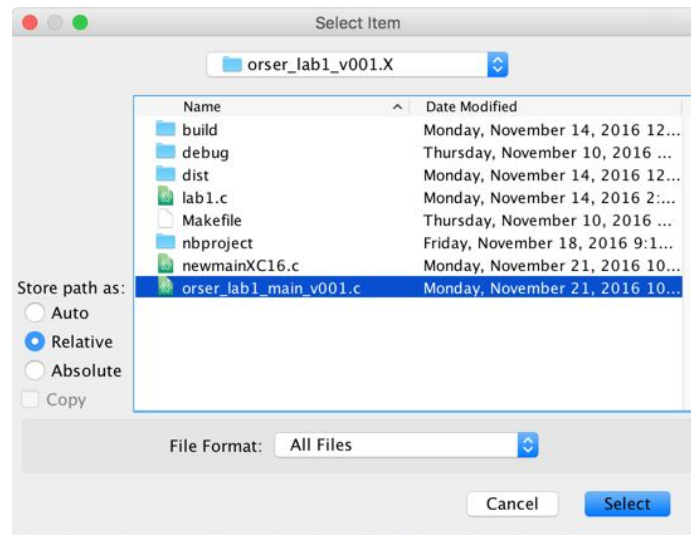
Folder:

Created File:

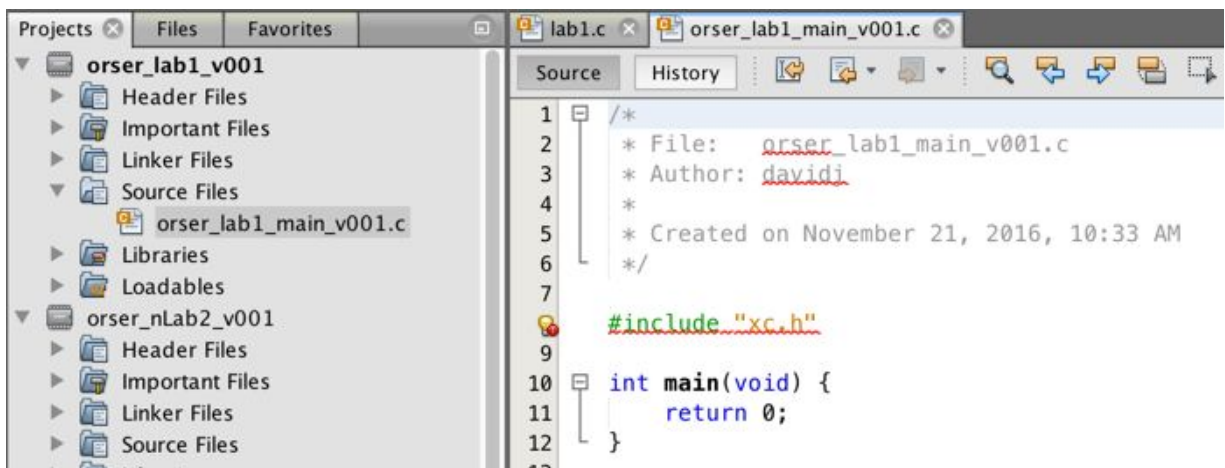
- 4.) (If this does not add the file to your currently open project, do this now, by right-clicking on your project title and selecting “Add Existing Item...”)



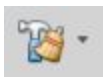
- 5.) The default directory should be the current folder for your project (ie., “/Users/davidj/MPLABXProjects/orser\_lab1\_v001.X” or “C:\Users\k\MPLABXProjects\kia\_lab1\_v001.X” on a Windows machine). If it did not then navigate to the correct directory. Finally, select your newly created file and click “Select”.



6.) When complete your project should look something like this:



7.) You should be able to immediately compile your code using the Clean and Build Project



button. You should see a message in the "Output" tab at the bottom of MPLAB X IDE that says:

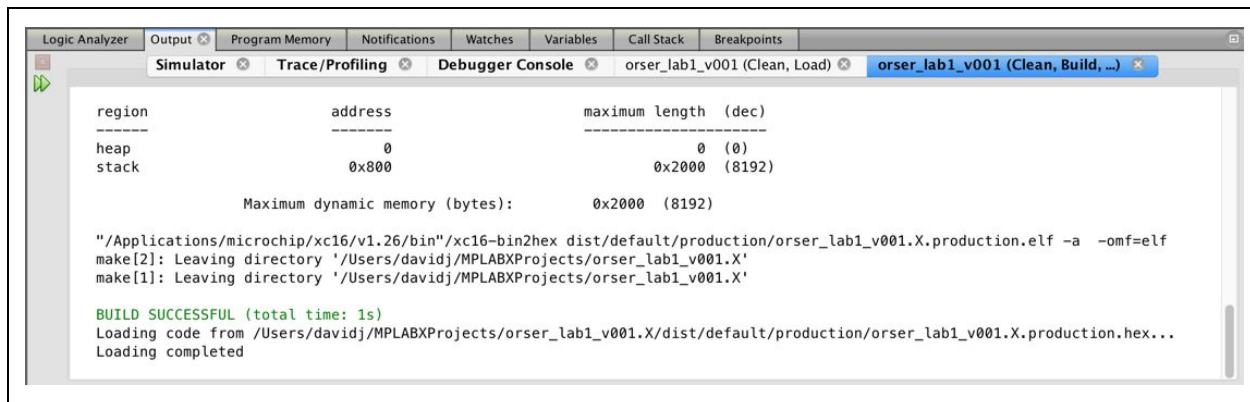


Figure: Clean Build

If you get errors, please double check the installation instructions and then visit any EE2361 TA during the first week of lab. You can view a list of open lab times at CSE Two-Stop → <https://www.aem.umn.edu/cgi-bin/courses/noauth/class-schedule?current=EE>, search for “EE 2361” (note the space).

- 8.) Later we are going to configure the source code in our project so that instructions execute at a frequency of 16MHz. Unfortunately the simulator is not going to be smart enough to deduct the processor speed from configuration code in our .C file. We have to manually specify the “instruction” clock frequency.

Right-click on the project name (click on the “Projects” text on the far-left if the projects window is not open already). Click on “properties” (the last item when you right-click). On the left pane, click on “Simulator”. On the right, set the “Instruction Frequency (Fcyc)” to 16.

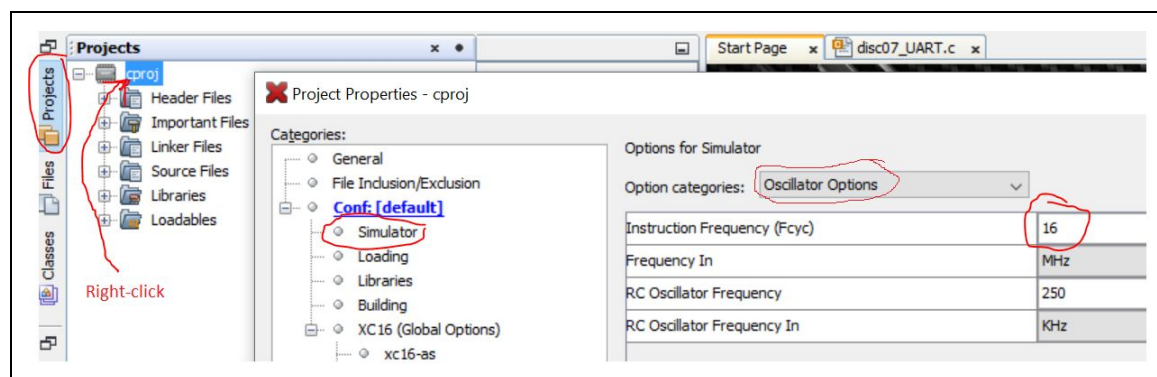


Figure: Setting the Clock Frequency in the Simulator

## Create basic boilerplate code

A “boilerplate” is a generic set of setup code that can be reused across multiple projects.

The first thing we need to include is the header containing the programming constructs we are going to use to make our code readable (for example we want to be able to type “PORTB” instead of the hexadecimal memory location for the port B register.)

Microchip provides these programming constructs in the file “xc.h” (general use constructs, which also includes more controller-specific constructs based on your project configuration). “xc.h” should already be included in your template file.

```
#include "xc.h"
```

Code: Include statements for boilerplate

Microcontrollers are complicated devices and while most of the setting can be configured while the microcontroller is running (ie., in a setup() function), some of the settings must be configured before the microcontroller begins executing code. This is done via the “#pragma config” directive. This directive tells the compiler to set certain bits in the microcontroller's FLASH memory at the time it is programmed. Automatically after reset, those bits are then loaded from FLASH memory into many different registers throughout the MCU Common configuration options that need to be set this way are:

- oscillator control bits
- power-saving wake-up timers
- error-catching watchdog timers.

We need to add the following lines of code to our boilerplate, each line is comment with some hint at what it does. Feel free to read Section 24 of the PIC24 Family Datasheet to learn more.

1.) Add the following #pragma statements to your file under the #include statements:

```
// CW1: FLASH CONFIGURATION WORD 1 (see PIC24 Family Reference Manual 24.1)
#pragma config ICS = PGx1      // Comm Channel Select (Emulator EMUC1/EMUD1 pins are shared with PGC1/PGD1)
#pragma config FWDTEN = OFF    // Watchdog Timer Enable (Watchdog Timer is disabled)
#pragma config GWRP = OFF      // General Code Segment Write Protect (Writes to program memory are allowed)
#pragma config GCP = OFF       // General Code Segment Code Protect (Code protection is disabled)
#pragma config JTAGEN = OFF    // JTAG Port Enable (JTAG port is disabled)

// CW2: FLASH CONFIGURATION WORD 2 (see PIC24 Family Reference Manual 24.1)
#pragma config I2C1SEL = PRI    // I2C1 Pin Location Select (Use default SCL1/SDA1 pins)
#pragma config IOL1WAY = OFF    // IOLOCK Protection (IOLOCK may be changed via unlocking seq)
#pragma config OSCIOFNC = ON    // Primary Oscillator I/O Function (CLKO/RC15 functions as I/O pin)
#pragma config FCKSM = CSECME   // Clock Switching and Monitor (Clock switching is enabled,
                                // Fail-Safe Clock Monitor is enabled)
#pragma config FNOSC = FRCPLL    // Oscillator Select (Fast RC Oscillator with PLL module (FRCPLL))
```

Config Register Boilerplate Code

In the MPLAB X/XC16 programming environment the first function called after reset is main(). This function is defined for you during the creation of the template file. It should look something like this:

```
int main(void)
{
    return 0;
}
```

Initial main() function

Anything that you put inside the main function will get executed one time immediately after the PIC24 resets. This isn't very useful, so we'll add a call to the setup() function and an endless loop within main(), this will make your PIC24 look a lot like the Photon and Arduino you are used to from EE1301 (the only difference being, in Photon and Arduino, main() had an endless loop in which it called loop()). We directly implement the endless loop inside main).

2.) Define your new functions and add them to main(), it should look like this:

```
void setup(void)
{
    // Execute once code goes here
}

int main(void)
{
    setup();
    while(1) {
        // Execute repeatedly forever and ever and ever and ever ...
    }
    return 0;
}
```

Updated main() function

3.) Lastly, we need to add one more line to our template that we will nearly always use. The default speed for the PIC24 is 16 MHz (8 MIPS), however unless you're concerned about power consumption, there is no good reason not to run at the maximum speed of 32 MHz (16 MIPS). Add the following line to your setup() function to change the clock speed to max.

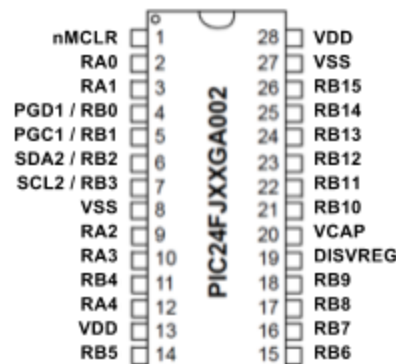
```
void setup(void)
{
    CLKDIVbits.RCDIV = 0; //Set RCDIV=1:1 (default 2:1) 32MHz or FCY/2=16M
}
```

Change to maximum clock rate for the PIC24

## Toggle an output on your microcontroller

On the PIC24, there are no fancy functions like `pinMode(D1, INPUT_PULLUP)` and `digitalRead(D1)`. Instead we will directly manipulate a set of registers that control the I/O Pins.

The PIC24FJ64GA002 has 21 general purpose I/O's. They are split into two groups I/O port A and B pins (due to having only 16-bit registers available). I/O port A and B are notated on the following diagram by "RBx" and "RAx":



Simplified PIC24 pinout diagram

The seven main registers that control these I/O are:

- AD1PCFG - Controls analog (0) or digital (1) pin operation
- TRISx - Sets mode to output (0) or input (1)
- LATx - Set the output driver HIGH (1) or LOW (0)
- PORTx - Reads the input buffer

(Where TRISA controls I/O port A and TRISB controls I/O port B, etc.)

Generally speaking you should write to LATA and read from PORTA, but there are some finer nuances of using these bits that are detailed in PIC24 Family Datasheet - Section 10.1 and in even more detail in [PIC24F FRM - Section 12](#), should you need them in a future lab.

We'll now initialize our I/O Ports. The following code will set all pins to "digital" mode, set I/O port A to inputs, and I/O port B to outputs (except RB0 and RB1 which are reserved to communicate with the chipKIT programmer/debugger), then set all available outputs to HIGH:

```
AD1PCFG = 0x9fff;
TRISA = 0b0000000000001111;
TRISB = 0b0000000000000011;
LATA = 0xffff;
LATB = 0xffff;
```

Digital Port Initialization

1.) Add the above code to the end of your setup() function (after the CLKDIV... line).

2.) Finally declare two variables and add some code to your main() function:

```
int main(void)
{
    unsigned short int count = 0;
    unsigned long int delay;


    setup();
    while(1) {
        // Execute repeatedly forever and ever and ever and ever ...
        count++;
        LATB = count << 2;
        delay = 40;
        while (delay--);
        asm("nop");
        asm("nop");
    }
    return 0;
}
```

Updated main() function.

The above code will increment count (16-bit word), and assign the value to LATB (shifted by 2 digits, since RB0 and RB1 are in use by the programmer). This should result in the outputs RB2 through RB15 toggling each at a different rate.

## Simulate your code

Do the following in order to prepare the IDE to simulate code:

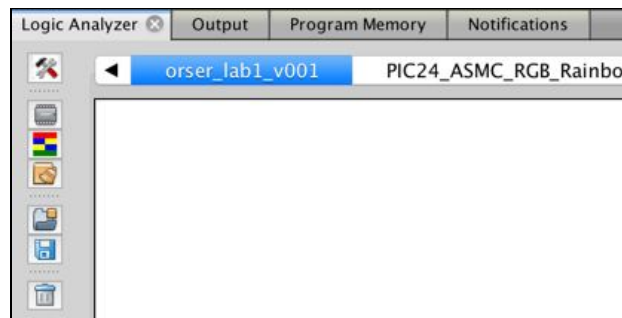
1.) Click on the “Clean and Build Main Project” (  ) Button. Verify that there are new build errors and the message “BUILD SUCCESSFUL”:



```
BUILD SUCCESSFUL (total time: 2s)
Loading code from /Users/davidj/MPLABXProjects/orser_lab1_v001.X/dist/default/production/orser_lab1_v001.X.production.hex...
Loading completed
```

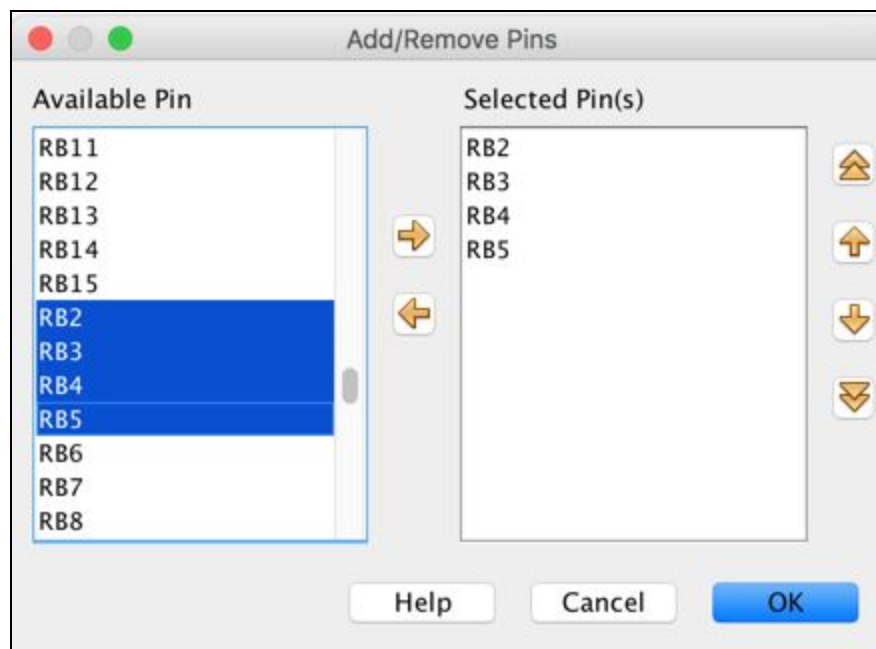
Example “BUILD SUCCESSFUL” message

2.) Open the “Logic Analyzer”: Click on the menu Windows → Simulator → Analyzer  
This will open a window pane at the bottom of your IDE labeled Logic Analyzer. This window should be blank and look something like this:





- 3.) Click on the “Edit pin channel definitions” button (  )
- 4.) In the dialog box that opens up, select RB2 through RB5, and add them using the arrow (  ) button. It should look something like this:



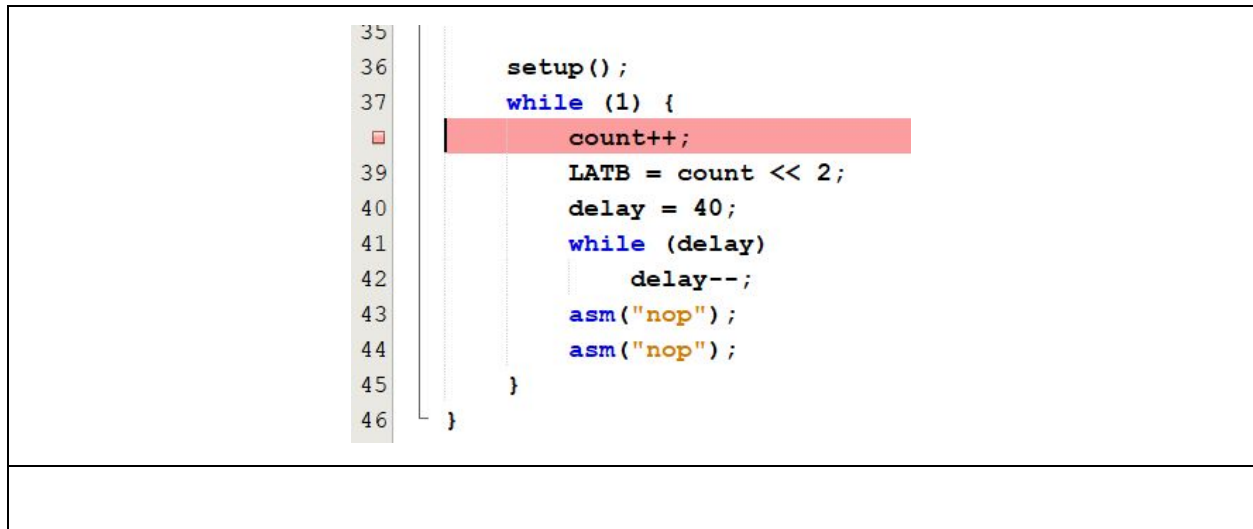
- 5.) Click OK


Now the simulator should be ready to run. Unfortunately, it will just run forever and not show anything interesting. In order for a software simulator to be useful, we have to tell it where to pause the simulation so we can look at the state of the simulation. These stopping points are called “Breakpoints”.

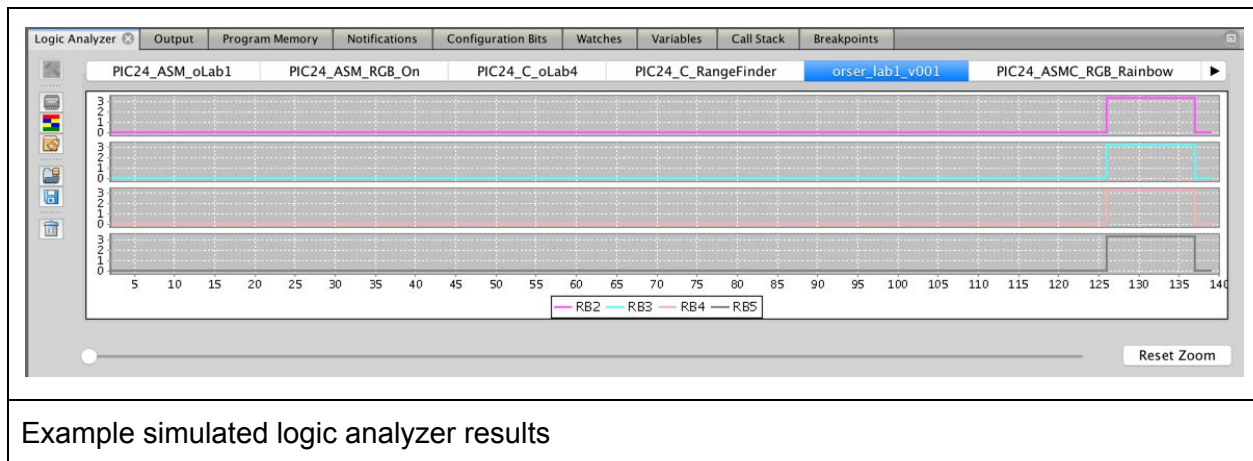





- 6.) Add a breakpoint at the “count++;” line of your loop, by clicking next on the line number, like so:



- 7.) Click the “Debug main project” button (  ) to run the simulator. The IDE will compile your program and run it in the simulator.
- 8.) If the bottom pane tab of the IDE is not set to “Logic Analyzer” select it now. The results should look something like the figure below. Note that the x-axis in the logic analyzer window is the number of instruction cycles.



- 9.) Click the run (  ) button several more times. You will need to determine the period and pattern of the resulting output in your final report.

HINT: It can also be useful to play around with the Window → Debugging → Stopwatch window.

## Prepare the breadboard

Before beginning this section, it is very useful to read through the PIC24 Family Datasheet section 2.1 Guidelines for “Getting Started with 16-bit Microcontrollers: Basic Connection Requirements” (Note: The Family Datasheet contains a general overview for all PIC24FJ devices, including several that have higher pin count packages than our DIP28.)

Of particular note in this section is the list of required pins to operate the PIC24F:

- VDD and VSS - Power and ground
- Our chip does not have AVDD and AVSS (Analog Vdd/Vss pins). You can ignore references to them when you read parts of the data sheet.
- nMCLR (short for “not” MCLR, or active low logic; also notated as  $\overline{\text{MCLR}}$ ) - Reset
- DISVREG - Disable internal core regulator (always connect to ground)
- VCAP - Capacitor for internal core voltage regulator

All of these pins must be connected, then double checked, before powering your breadboard.

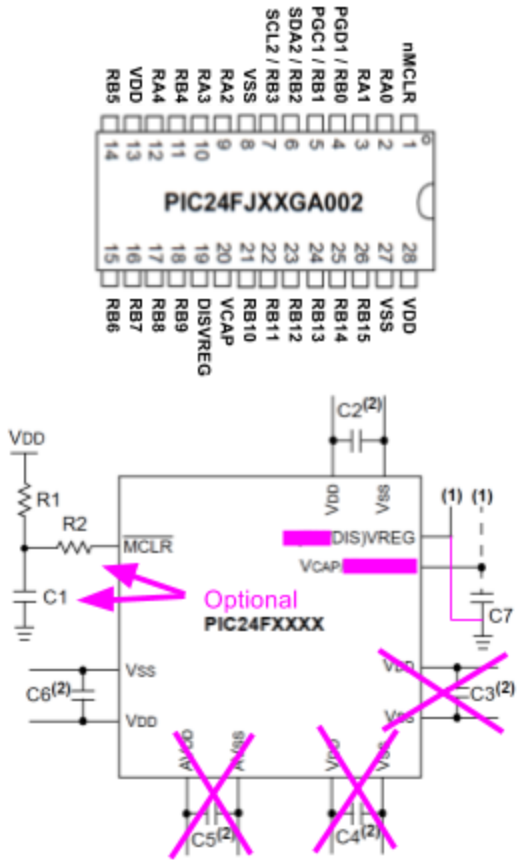
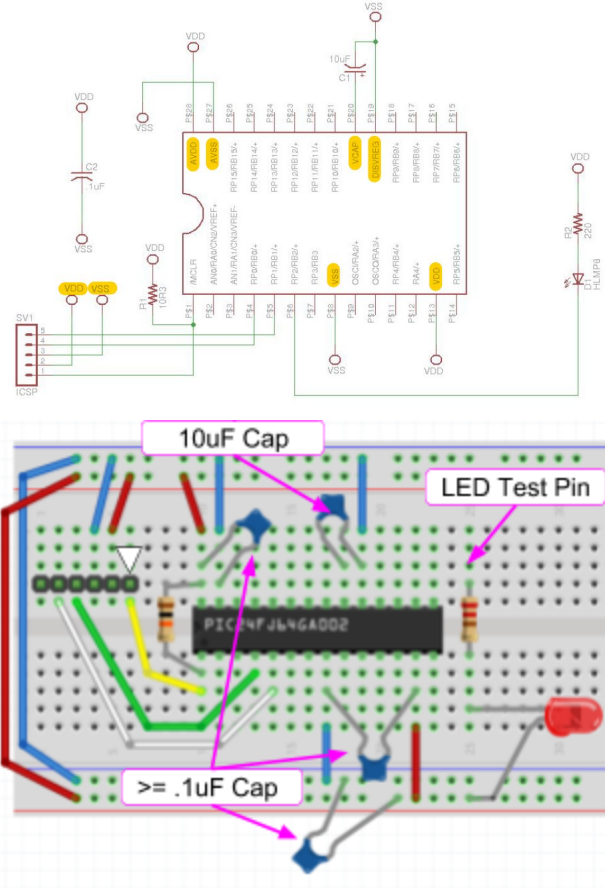
**Failure to connect these pins correctly can result in permanent damage to your PIC24!**


The recommended minimum connections schematic is shown below with pink notes specific for our device (PIC24F in DIP28 package). It is important to note that our 28-pin DIP package only has two sets of digital power supply pins (not four): pins 8 and 13, and pins 27 and 28. The nMCLR pin resets the PIC24 when pulled low, thus needs a pull-up resistor. The capacitor/resistor network will delay the startup of the device until the main power supply is stable. Since we will trigger a reset (after programming) when the supply is stable, R2 and C1 are optional components. However, they should always be included when making a product that operates without the programmer/debugger.

A programming header (ICSP) needs to be implemented as well. ICSP has to access three pins on the PIC24 nMCLR(pin1), PGED1 (pin4), and PGEC1 (pin5). Additionally, the chipKIT (and PICkit3) programmers also supply a weak 3.3V power supply. We will use this power supply to operate the chip during debug.

Finally a test LED is included to view digital outputs. It remains unconnect for now, but should be connected to a long jumper wire so you can probe various outputs.



	
<p>PIC24 Family Datasheet Schematic and Diagram</p>	<p>Improved Schematic and Breadboard Wiring Diagram</p>


<p>chipKIT and PICKit3 programming header pin out</p>

Complete the following steps:

- 1.) Connect your bread board as shown in the Breadboard Wiring Diagram above



- 2.) Have a classmate, TA, or instructor double check your breadboard
  - a.) When double checking, pay careful attention to VDD and VSS and that your horizontal breadboard power rails are connected together.
  - b.) Connecting power backwards \*will\* destroy your PIC!
- 3.) Get out your DMM
- 4.) Connect your chipKIT to your PC via USB
- 5.) Connect your chipKIT to your breadboard via the headers, be certain pin 1 (the white arrow) is closest to your PIC24 (and connected to nMCLR)
- 6.) Quickly check the power supply pins on your chip with your DMM (VSS-VDD and VSSA-VDDA). If you see less than 3.1V disconnect the chipKIT and review your connections. You should also be able to check the VCAP voltage (2.5V).

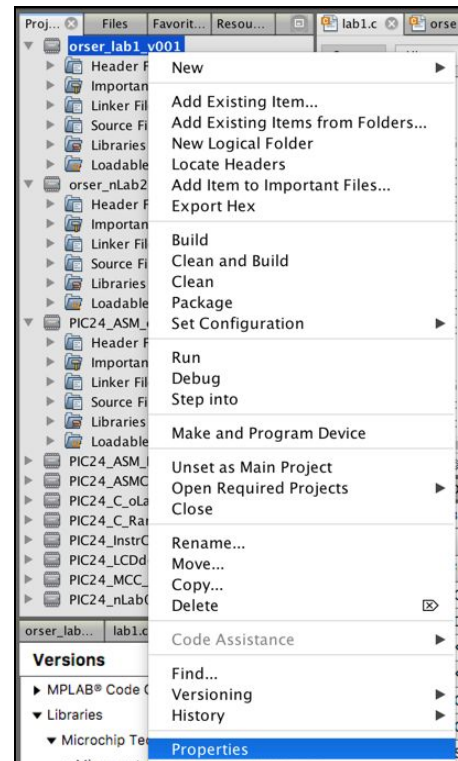
If all when well above, your device should be ready for programming.

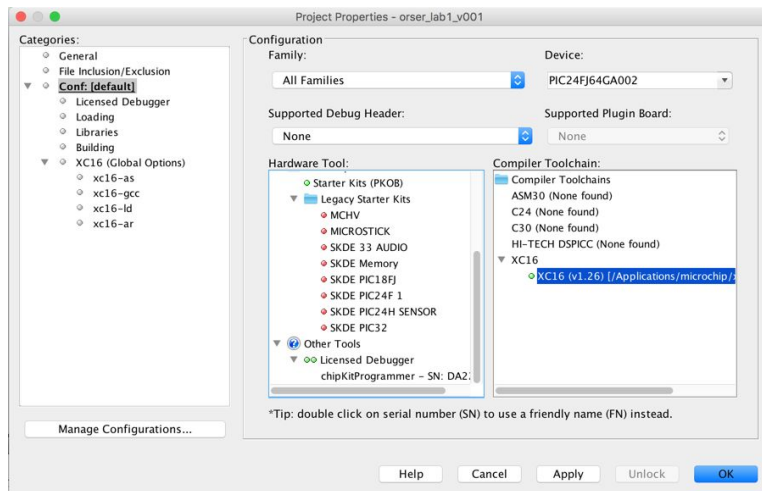
## Programming the PIC24

In order to program the PIC24 the MPLAB X IDE project must be configured with the programmer as the target. Execute the following steps in MPLAB X:

- 1.) Right click your project in the Projects tab of the left pane
- 2.) Select "Properties" (as show on the right)



If your chipKIT has been detected correctly by your operating system it should show up at the bottom of the "Hardware Tool" pane of the project properties dialog box (as shown below.)



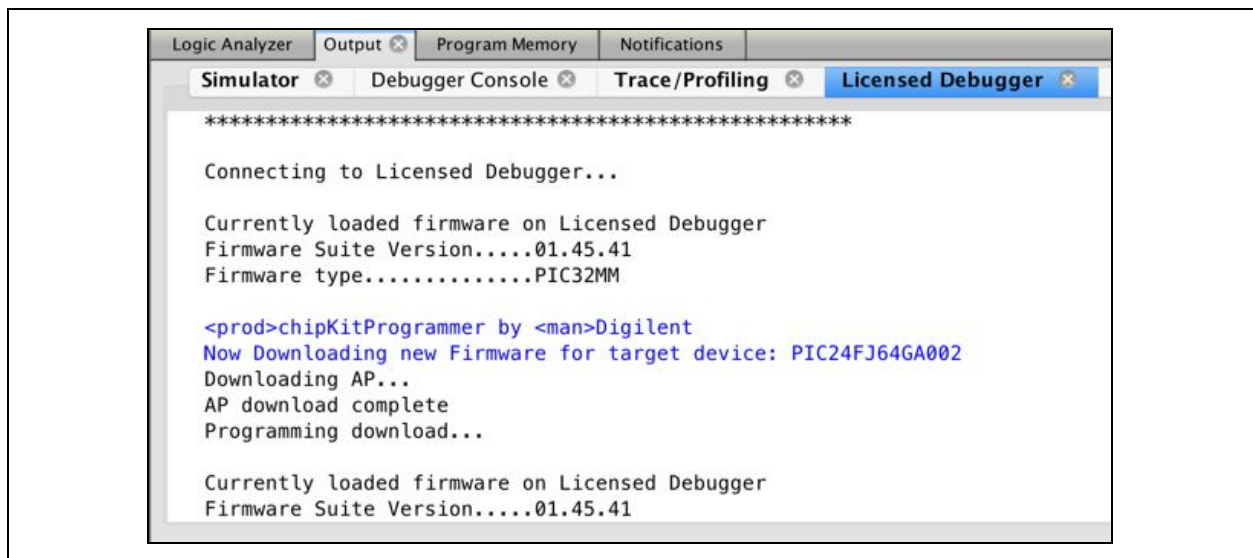


- 3.) Select “Properties” (as show on the right)
- 4.) Click on “chipKitProgrammer” and click OK

Everything should be set now to program you PIC24. However, it's always a good idea (like with the simulator) to compile once before programming, just to be certain you didn't add at characters, etc. on accident.

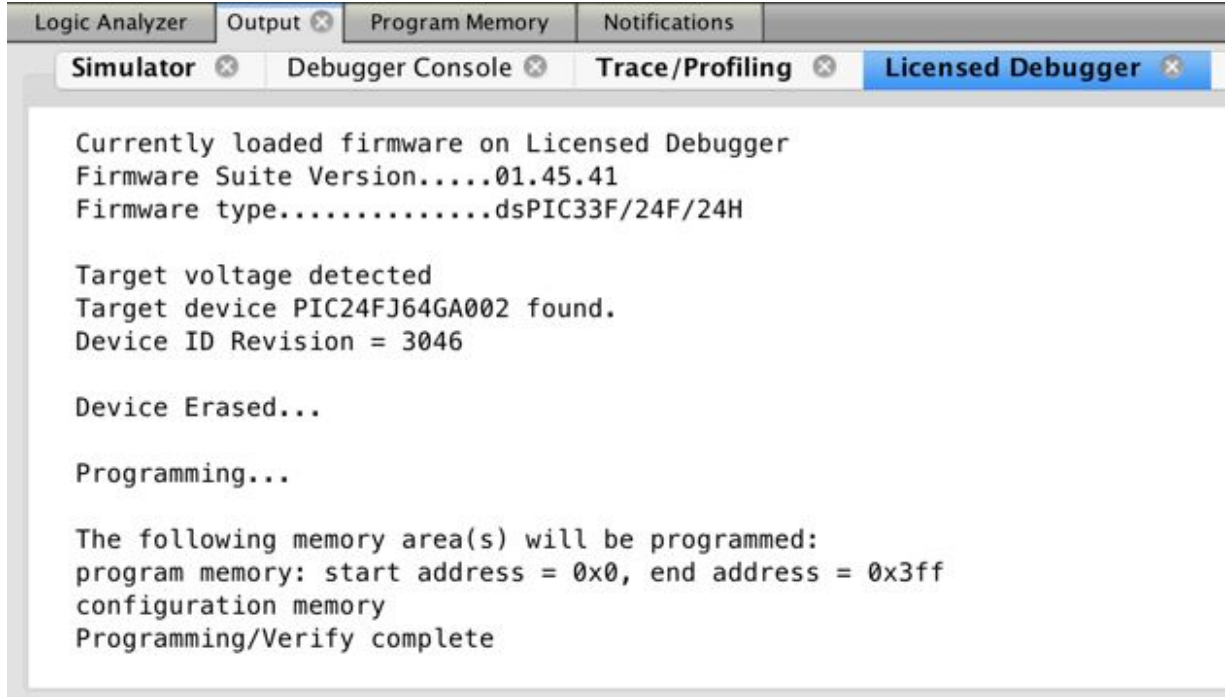
- 5.) Click the Clean and Build button (  )
- 6.) Click the Make and Program button (  )

The first time you program a new device with your chipKIT programmer MPLAB will detect this and update the firmware of your programmer. This process is automatic but will only happen once. The output looks something like this:



### Example of new Firmware re-load on chipKIT programmer

Once the firmware has been loaded on the chipKIT programmer, the chipKIT will immediately start programming your PIC24. The output should look something like this:



```

Logic Analyzer  Output x Program Memory  Notifications
Simulator x  Debugger Console x  Trace/Profiling x  Licensed Debugger x

Currently loaded firmware on Licensed Debugger
Firmware Suite Version.....01.45.41
Firmware type.....dsPIC33F/24F/24H

Target voltage detected
Target device PIC24FJ64GA002 found.
Device ID Revision = 3046

Device Erased...

Programming...


The following memory area(s) will be programmed:
program memory: start address = 0x0, end address = 0x3ff
configuration memory
Programming/Verify complete
    
```


Example output when programming PIC24

You should now be able to connect your “test” LED terminal via jumper wire to each of the output ports RB2 through RB15 and see the LED glow and/or blink. Remember the pinout is at the top of this lab manual ([link](#)).

## Hardware-in-the-Loop Debugging

A useful debugging tool is to use the IDE environment to add breakpoints and single-step code while the code is running on the actual hardware. Connect the test LED to RB2. With the chipKIT programmer selected as the “simulation” tool in MPLAB X, add a breakpoint in the loop (e.g., on

the line that says count++), and this time instead of clicking the program button (  ), click

the Debug Project button (  ). As the program executes and stops at the count++ line, you can click F8 (step-over), and observe how RB2 blinks at the line that says LATB=.... Press the play button to continue running the program and stopping at the breakpoint again. You can see that the program does NOT run at its normal speed, but instead it waits for your to tell it to execute more instructions. This hardware-in-the-loop debugging mode would be useful in



situations that you have tested your software in the simulation and logic analyzer shows that everything should work, but when you program your hardware, the hardware does not do what you wanted. By doing a hardware-in-the-loop debugging, you might for example notice that your wiring for the RB2 connection is wrong.

## Creating New Projects or Reusing One Project?

During the course of the semester, you will be writing many new .c and .asm files. Some people prefer to create brand new projects each time they want to implement something new, and others prefer to work within one project, but switch source codes.

The advantage of creating new projects is that you will have separate folders, each with all the source code and configurations you need. This means you can go back to a previous lab at any time and not worry about shared library files having changed. The disadvantage is that every time you create a new project, you have to go through the steps of choosing the right family of chips, selecting PIC24FJ64GA002 as your chip, and setting the simulator clock frequency to 16MHz. But as we said, some people have a strong preference for this option.

The advantage of reusing your project is that you don't have to go through all those steps at the beginning, and you are not going to forget to set the simulator clock frequency to 16MHz (and e.g., enabling UART simulation). The disadvantage is that you might not be able to organize your files as well. To reuse the project, you can add a new source file to it, and then right-clicking on your old source file and choosing "Exclude file(s) from current configuration". The file will be grayed out and would be excluded from compilation. You can also remove the file from the project to avoid crowding out the "sources" folder in the project view.

## Lab Report

As this is an initial self guided lab there is a very limited report due before the start of the Lab 2. You are primarily responsible for demonstrating that you have working hardware and software and have thought about the relative rates of Human and MCU perception and response.

Put together a simple document (gDocs or Word) with the following information. This is not a formal report, but should include 3-4 complete sentences describing your answers and reasons. Submit your document to the Moodle site in PDF format.

- Screen shot of logic analyser output of the simulation of the counter
- Use the above to estimate the frequency of the counter
- Find the frequency of the actual counter in hardware
  - For example count the number of flashes on pin RB15 in 20secs
  - Alternatively you may wish to use an oscilloscope or hardware logic analyzer
- Compare hardware and simulation.
  - How long is one logic analyser cycle in real time?
  - Do simulation and hardware agree?
- How fast you can perceive a change in the LED?
- How many operations (instruction cycles) the MCU is capable of doing in that time frame?

Before the end of Lab 1, your Lab TA must see a demo of your circuit. Make sure s/he checks off your work before you leave the lab (or, setup a time outside the lab to demo if you don't have time to finish).

## Going further

- Can you wire up the 10-LED light bar with 220 ohm resistors to show a complete 8-bit word at one time?
- Can you slow down the output to update of RB2 to once per ~0.25 seconds?
- Can you change the output pattern to a pseudo random number generator?