UNIVERSITY OF CALIFORNIA, SANTA CRUZ
BOARD OF STUDIES IN COMPUTER ENGINEERING

CMPE167/L: INTRODUCTION TO MECHATRONICS

NEW MPLAB-X PROJECT INSTRUCTIONS

PURPOSE:

MPLAB-X is the Integrated Development Environment (IDE) that we use in this class. Making a new project, building it and loading it onto an Uno32 has quite a few steps that need to be completed correctly in order for the whole chain to work. This document is an outline of those steps, and will be a useful reference any time you start a new project. This document will lead you through a "hello world!" application first in the simulator, and then on to the actual hardware. When having difficulty with the tool chain, it is very helpful to recreate a basic "hello world!" to verify that the toolchain works from end to end.

Note that this document does not detail how to set up the IDE and toolchain on your own PC. While this is relatively straightforward, the process is detailed in a different document.

OPEN MPLAB-X AND CREATE A NEW PROJECT:

Launch MPLAB-X (we are currently using version 3.35), and create a new project from File -> New Project, or by clicking on the icon that looks like the folder with a green plus sign on it ( ). This will pop open the new project dialog box, and you want to select the Microchip Embedded and the Standalone Project from the projects box (see *Figure 1*). Click on the Next > button and move onto the selection of the processor for the project.

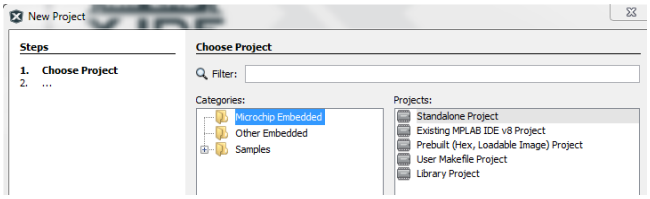

Figure 1: Project Type Selection

For the processor family, choose the 32bit MCUs (PIC32), and for the processor itself, choose the PIC32MX320F128H (see *Figure 2*). Note that you can type in the letters into the processor box and it will filter processors by the letters you type in. The PIC32MX320F128H is the processor on the Uno32 boards we use for the class. You will get very strange behavior from the compiler if you have the wrong MCU selected; make sure you get this correct.

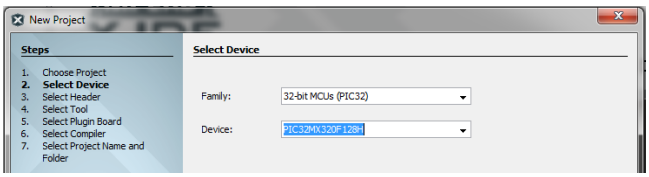

Figure 2: Processor Selection

The next screen is to choose a hardware tool for loading code onto the microcontroller. Select the PICKit3, even though we won't be using it (your PIC32 has firmware for loading code directly from the USB serial connection). It should have a green light next to it. If it doesn't you have the wrong PIC32 chosen in the previous screen (see *Figure 3*).

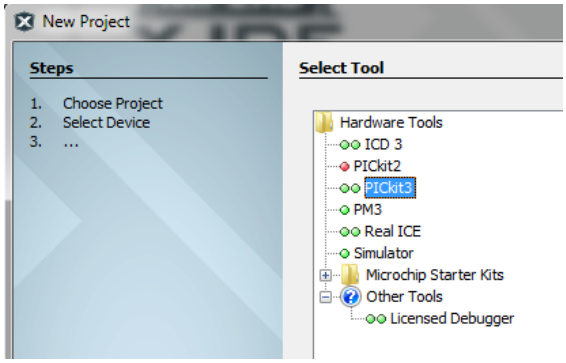

Figure 3: Programmer Selection

Clicking on Next > takes you to the compiler selection screen. For the compiler, there should be only one to select, the XC32 compiler (version 1.42), which should have a green light next to it. The full path to the compiler will be listed as well (see *Figure 4*).

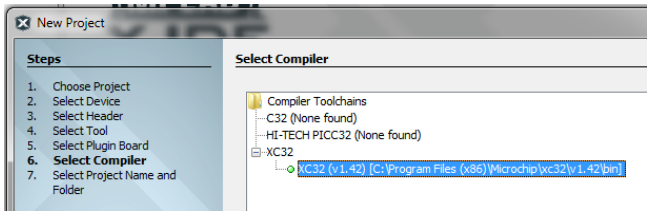

Figure 4: Compiler Selection

The last screen allows you to give your project a name. Use a name that describes what it is that you are doing, such as “HelloWorld” or “Lab0_RoachExploration.” See Figure *5*. We highly recommend that you put your project in Dropbox and use selective sync so that you can share that folder with your lab partner. This also means that you can use any workstation (or even your own laptop) with minimal effort. For those of you familiar with GIT, you can work in GIT as well (we have a GIT server set up for the class).

Figure 5: Naming Project

At this point, you should have an empty project with the correct name showing up and a listing of all of the subfolders that MPLABX uses to organize your project (see *Figure 6*). At this point if you open them up, they will all be empty as nothing has yet been added to the project.

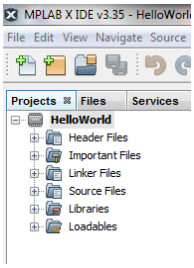

Figure 6: Created Project

In order to get “Hello World!” working on your roach, you will need a few additional steps and files. The first step will be to add a new C-file with the function main() in it. Every C program needs one (and only one) main() function which is used to run the main code. We will add this by right-clicking on the “Source Files” folder and selecting New -> Other which will pop open a New File window. Select Microchip Embedded -> XC16 Compiler and choose mainXC16.c (see *Figure 7*). Note that we are using the XC16 Compiler files even though the target is programmed using the XC32.

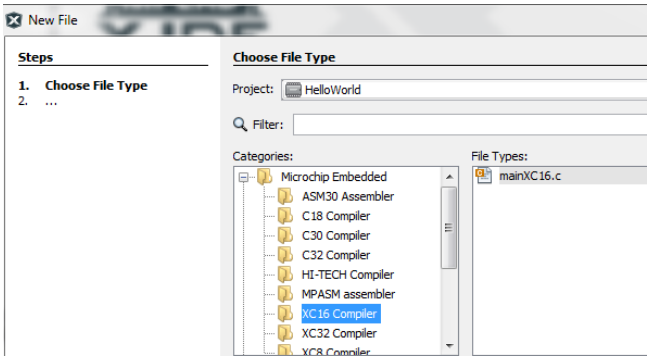

Figure 7: New mainXC16.c

The next step will ask you to name your file. Name it something descriptive, such as HelloWorldMain or something that makes sense to your project (if all of your projects have a HelloWorldMain file, we will be disappointed). The file is added under the Source Files in the project window, and it is opened up in the editor window. You have now successfully created your first project (it is far from complete). You can compile your project now using the menu Run -> Build Main Project or Run -> Clean and Build Main Project. This can also be done by clicking on the hammer (build) or the hammer with the broom (clean and build):  . The difference is that the normal build will keep anything that was previously compiled but not changed, whereas the clean will force the compiler to get rid of all previous work and recompile all files.

If you have done everything correctly to this point, you will get an Output window, and the bottom will have a green `BUILD SUCCESSFUL (total time: XXXms)`. Now you have compiled the program, but it actually does not do anything.

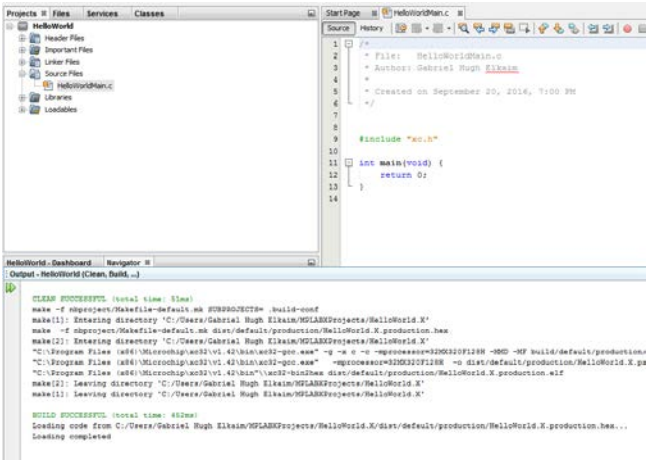

Figure 8: Output from Build

HELLO WORLD!:

In order to get “Hello World!” working, several additional steps need to be taken to initialize the hardware, the bootloader software, and to actually print out the text onto the serial port. Each step is similar to the others, and consists of adding header and code files (.c/.h) to the project.

The first step is to add the bootloader linker script to the file. This tells MPLAB-X where to load your code into the flash so that you don't overwrite the bootloader that we use. Right-click on the Linker Files folder in the project window, and select Add Existing Item …, navigate to the C:/CMPE167 directory and add the booloader320.ld file (see *Figure 9*). For the "Store path as" radio buttons, select Absolute (this means the path to the file won't change if the project is moved).

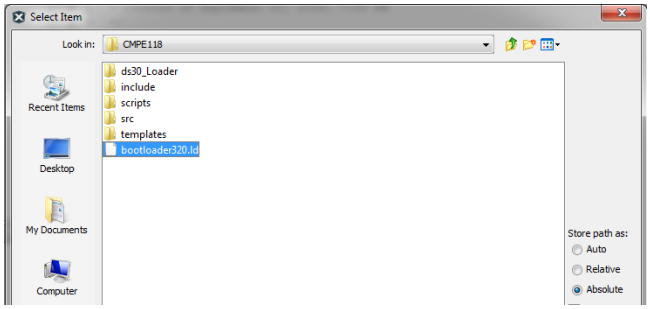

Figure 9: Adding in bootloader file

In a similar fashion, you will add the header files to the project. Right-click on Header Files, select Add Existing Item …, and navigate to the C:/CMPE167/include directory and select the files: BOARD.h, AD.h, and serial.h and add them to the project using Absolute paths (see *Figure 10*).

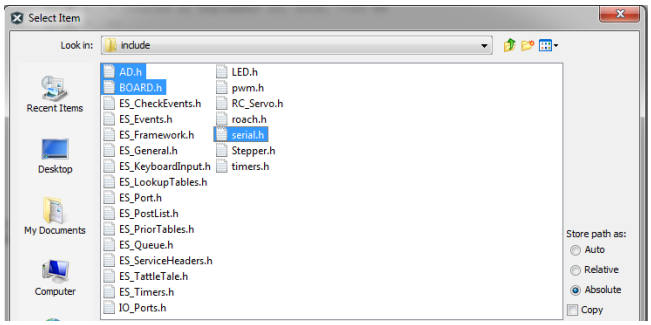

Figure 10: Adding Header Files

In a similar fashion, you will add the source files to the project. Right-click on Source Files, select Add Existing Item …, and navigate to the C:/CMPE167/src directory and select the files: BOARD.c, AD.c, and serial.c and add them to the project using Absolute paths as well.

The next step is to modify your main() to actually print "Hello World!" to the screen. Modify your main.c file to have the following lines (remember that CTRL-space is an autocomplete, it is very handy):

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

int main(void)
{
    BOARD_Init();

    printf("Hello World!");
    while (1) {
        ;
    }
}
```

It is good practice after typing in some code to auto-format the code using ALT-SHIFT-F. Note that if you attempt to compile the program now, it will fail because the compiler cannot find the BOARD.h file. This is fixed using the customize menu where you will add in the paths for the include files so that the compiler can find them. Navigate to Run -> Set Project Configuration -> Customize… (this can also be reached via the pulldown menu on the toolbar). Select xc32-gcc, and select the Preprocessing and Messages button. Click on the "…" next to Include

Directories and add a single "." Next add absolute path to C:/CMPE167/include (see *Figure 11*). This will tell the compiler where the custom 167 library files are. At this point you can compile your code and you should get no errors.

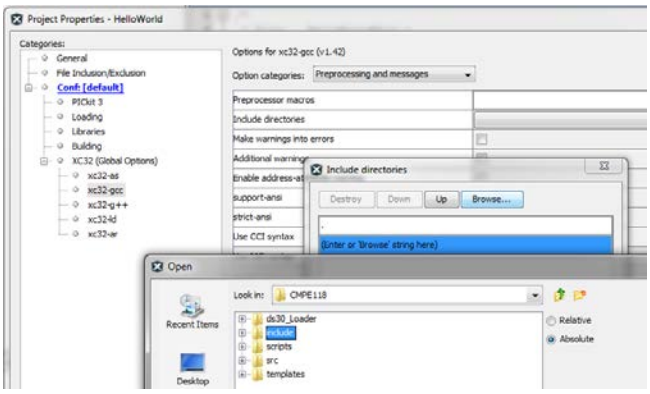

Figure 11: Customize Include Directories

Note that BOARD.h and BOARD_Init() always come first when using the 167 libraries.

USING THE DS30LOADER:

The project you have created is now ready to run on the embedded hardware. The original Uno32 boards come with an Arduino firmware on them that we have removed and replaced with our own custom bootloader that allows us to run C on the bare metal. The Uno32 is flashed via the USB port, using the ds30Loader program.

Launch the ds30Loader. In the View menu, select Advanced Mode. If this is the first time you have launched it, there are some settings that need to be set up correctly for it to communicate to the Uno32. Under the Basic tab, sure the "Write flash" box is checked, and the Baud rate is set to 115200. For the Hex-file, click the … and navigate to your project directory, folder dist -> default -> production and select the .hex file (see *Figure 12*).

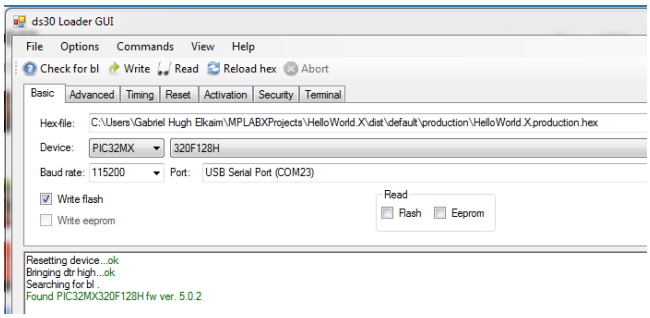

Figure 12: ds30Loader Basic Settings

Under the Reset tab, select the DTR button and set the Reset time [ms]: to 10. Lastly in the Terminal tab, set the Baudrate to 115200, and check the "Switch to after write" checkbox (see *Figure 13*).

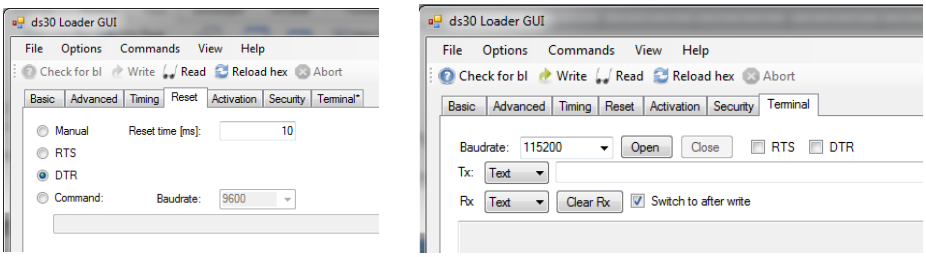

Figure 13: Reset and Terminal Setting

Back in the Basic tab, if you “Check for bl” it will ping your Uno32 and see if it is ready to program (you should see fw ver. 5.0.2). At this point click on the Write button, and your code will get loaded onto your Uno32. The ds30Loader will switch automatically to the serial port, and you should see Hello World! on the output of the serial window (see *Figure 14*).

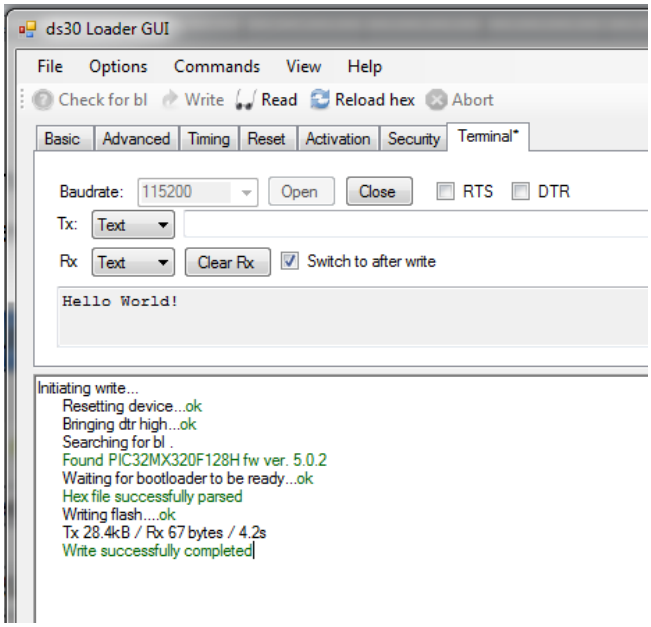

Figure 14: Hello World!

Note that if you want to make a change to your code you will need to close the serial port before you can write the flash again.

CONCLUSIONS:

Congratulations! You have now verified that the entire toolchain works for getting code onto your Uno32 stack or roach, and that you can get output back from the device. If you are having problems, go back and review the steps and make sure you have not omitted any. If at any point you need to verify that you have a working toolchain on a machine, follow the steps outlined in this document to ensure that you have everything set up correctly.