CPE 490 Lab 1

## Goals

1. To get the MPLAB X IDE and the MPLAB XC16 C language tool set on everyone’s laptop
2. To write your first C program in this environment
3. To explore and get familiar with the dsPIC33 data sheet

## Overview

This lab will largely be a tutorial. In you lab book record this as Lab 1 and record any information you see fit while you answer the questions. There is no report to turn but please write up the questions with your answers and submit them for your lab 1 work. This lab will be graded on the answers to the questions only.

## Install programs

You will need to install two software applications:

MPLAB X IDE V 2.00 – you can get this from the microchip web site <http://www.microchip.com/pagehandler/en-us/family/mplabx/> or from a thumb drive I will bring to class. You might have an older version installed on your laptop from CPE303; it will be fine to continue to use that version

MPLAB XC16 V1.21 you can get this from the microchip web site you just down loaded MPLAB X form, or you can see instructor for a copy on a thumb drive.

## Write First C program

## Project Wizard

1. Open MPLAB X IDE application
2. Select File -> New Project The new project window should appear.
3. In the category window choose Microchip Embedded, then in the project window select Standalone Project. Click Next >
4. In the Family box select 16-bit DSCs (dsPIC33). In the device box Type dsPIC33FJ256GP710A in the box or partially do so and then use the pull down list. Then click Next >
5. In the Select Tool box select simulator and click Next>.
6. In the select compiler window select XC16 (V1.21)….. then click Next >
7. Create a folder and a name for for you first C code program, then click on next
8. Since we have no existing files to add to the project click, male sure “Set as main project” is checked and then click finished.

## Source Code

1. In the upper left window, with the Projects tab selected, right click on “Source Files” and select new, then C Source File…. Now give your source file a name like FirstProgram.c.
2. Type the following into the new window:

//

// CPE 490 First Program

//

## Include Header File

1. Add the include line to your source file FirstProgram.c:

//

// CPE 490 First Program

//

# include <p33FJ256GP710A.h>

Let’s look at the file we just included.

On the menu bar select File->Open File

To find the file you must go to where the C compiler got installed. On my machine this is C: -> Program Files (X86) -> Microchip -> xc16 -> v1.21 -> support -> dsPIC33F->inc-> p33FJ256GP710A.h

Open this header file in MPLAB and look at it.

**Question 1 -**  What do you think the purpose of this file is? Hint: compare the order in the file to Table 4.1 on page 50 of the data sheet.

## Main Function

1. Add the following to your C source code so it looks like:

//

// CPE 490 First Program

//

# include <p33FJ256GP710A.h>

main()

{

PORTA = 0xFF;

}

## Compile and Link the Code

From the menu bar select Run -> build main project or use the following icon:



The code should build successfully.

## Testing and Debugging the Code

We will use the MPLAB SIM simulator to simulate our code running on the selected targets. This means for now the code is not running in the actual hardware but is simulated on the PC.

Since the code can exit the main subroutine (that is a no no) we want to stop the code so click on the C code line **number** that the code PORTA = 0xFF; is on. A red background color and a red box should appear at the number. This means that you just set a breakpoint.

Next, go into simulation mode by clicking on the Debug main project icon



To allow us to examine register values and port pin values use the watch window to look at the SFR PORTA. To do this, ri**g**ht click on PORTA in your source code and select new watch , click OK and a window should appear in the low right that shows this variable, its address, and its value. We also want to examine the value in Latch A (LATA), show this by going the variable window just made and clicking on the create new watch icon



Select the SFR’s button and find LATA in the list (if you enter the list and hit L it will save time).

To reset the program click on the reset icon



Next click on the step into icon



to execute the one C command and then again to execute the PORTA = 0xFF;

**Question 2 -** did any of the bits in PORTA and LATA change? Describe the changes you observed.

To find out what is going on consult the PIC33 data sheet for the dsPIC33FJ256GP710A. This data sheet you may have already purchased or it can be found at the Microchip website or on blackboard under the menu choice Lab Info-> MicroChip Documentation-> dsPIC33FJXXXGPX10A Data Sheet.pdf. Section 11 of the data sheet will be especially useful. Note that in figure 11.1 that in order to enable a channel as an output you must make sure TRISA buffer is set up correctly. Try this and see if it fixes the problem.

**Question 3 –** Did it fix the problem completely? Describe exactly what occurred.

If you still have a problem with some of the bits you will remember that by default I/O pins are configured be analog inputs. To disable analog inputs see section 21 of the data sheet and note AD1PCFGX on page 250. The problem now is to know what peripherals are assigned to what pin. The diagram on page 17 is helpful. Indentifying the RAx pins that you are still having problems with and what analog channels they represent. If you have problems in finding RAx were x between 0 and 15 just open up an electronic copy and search for it by using ctrl F. This is a way of saving your eyes!

Add to your C source code; such that when the assign statement PORTA = 0xFF is executed all the first 8 pins in PORT A go to the high output state. Verify your fix using breakpoints and the watch window.

**Question 4 –** Explain the behavior of question 2 and 3that is; why did the bits in PORTA and LATA change or not change in the way that they did?

**Question 5 -** What did you need to do to fix this? State what code was added to get the first 8 pins of PORTA to be driven to a high voltage?

## View Disassembly listing

The compiler takes your C code and turns it into hex code that can be disassembled into assembly code. You can view this code by clicking on the menu board Window->Debugging->Disassembly.

**Question 6** – What assembly language is generated from the c statement ‘PORTA = 0xff;’

## View Memory Usage

To access how much memory you have used in your program select from the menu bar Window-> Dashboard, in the upper left portion of the screen you should see the dashboard displayed. The dashboard shows what device your are using, what compiler your using, how much memory you are using, what debugger you are using, and what debugging resources you have used. In the memory section you will see RAM and Flash (ROM) use. Check out how much program memory you have used for just a few C lines.

**Question** 7– How much program memory and data memory have you used?

It might be surprising how much program memory is used; you have seen the Disassembly listing and there are only a few assembly language commands.

**Question 8**-Where is the rest of the instructions coming from and what are the instructions for?

To answer this question reset the program lets examine more of the program memory values. From the menu bar click on the following Window-> PIC memory views -> Program Memory A tab will come up in the lower right window showing the program memory space. Remember that at reset the program counter will be set to zero. Scroll in the newly created window to address 00000. There you will find a goto assembly instruction. Scroll in the window until you find that address that the goto instruction will take you to. Notice in the disassembly window you created a fest steps ago the code in main starts to execute at around 0x298, so there is quite a bit of code running before this point.

If we look at the first few lines we can see loading values into W14, W15 and the SPLIM registers. Think what these registers are used for you can answer question 8.

## Common error

A very common error is to forget to put a semicolon after a statement. Go ahead and remove a semicolon on one of the statements and try to build the code.

**Question 9** What error did the compiler identify?