Overview

This document is a tutorial on programming the PIC16F84A microcontroller using the C programming language and Microchip’s MPLAB IDE. Section I provides an introductory tutorial on MPLAB using the sample C program given for Lab 2.1. Section II provides basic information on simulation in the MPLAB environment followed Section III which discusses programming the micro-controller using the PICSTART hardware device.

I. Building with the MPLAB Integrated Development Environment (IDE)

First of all, what is an IDE? IDE stands for “Integrated Development Environment”. As the name suggests, an IDE is an environment/interface that assists the user in developing something, usually software. The “Integrated” part of the term refers to how the IDE is really just a collection of many different tools working together in order to make programming more streamlined. In our case, we will be using the MPLAB IDE, which consists of such tools as a code editor, a compiler, a debugger, and a programmer.

For more complete information on the MPLAB IDE, please refer to the “MPLAB IDE User's Guide with MPLAB Editor and MPLAB SIM Simulator” at the following URL:
1. Start MPLAB. You will get the window as shown in Figure 1.

![Figure 1. Screenshot of MPLAB.](image)

2. Create a new project using the “Project Wizard” as seen in Figure 2:

   **Project->Project Wizard**

   The chip we will be using is the “PIC16F84A” device, and our compiler, linker, and assembler toolset will be the “HI-TECH PICC Toolsuite”. You may name the project and destination directory as you see fit.

![Figure 2. Screenshot of Project Wizard.](image)
3. Add the Lab2.1 source file “piclab1.c” to the project as shown in Figure 3:

   Right-click Source Files in the Project Window->Add Files…

![Figure 3. Adding Source Files.](image)

4. Build your project:

   **Project->Build All**

   The same process can be done by clicking the button in your toolbar. You should get a screen that looks like Figure 4 if the build is successful. In short, this is the compilation and linking process and is quite similar to the other C compilers that you might have used. If the build is unsuccessful, check for syntax errors and or if there is anything wrong with the build options (Project->Build Options). **Build Options** provides the user with some control parameters for deriving the object files. This is one of the beauties of an IDE; it saves the user the effort of having to create a makefile. We just have to put our files into the project directory and select **Build All**.
5. So, now you’ve done some compiling and linking and you’ve worked out the compile-time and link-time errors. That’s great, but how does your program function to spec? We need to test for any run-time errors or, in other words, debug. MPLAB supplies various tools for debugging; one of which is MPLAB SIM, which we will be discussing in the next section.

II. Simulation

Simulation tools are quite useful in speeding up the design process as shown in Figure 5. Basically simulation provides the user with the ability to check for program-correctness without having to commit to specific hardware. For instance, if you are planning to use a chip that costs hundreds of dollars per chip, you don’t want to buy the chip, load your program, and find out that the chip doesn’t work with your program; that would be a waste of time and money. It also allows you to continue your project without actually being in the presence of your hardware: an example being if you are on a business trip and need to leave the hardware with your colleagues. For our situation, MPLAB SIM allows you to analyze your program by providing tools that allow you to imitate external signals, view internal variables, step through your program one line at a time, and more.
To get started, after obtaining a successful build, we first have to choose MPLAB SIM as our debugging tool as pictured in Figure 6:

**Debugger->Select Tool->MPLAB SIM**
2. So now that you’ve selected the simulation tool, there are many new commands available for you. These commands can be accessed through the **Debugger** menu, or through the toolbar buttons. You can explore these different commands on your own, but the ones to take note of are **Run**, **Animate**, and **Reset**. Try each of these three commands to see what they do.

   - **Run**: The IDE will just run your code all the way through. You won’t know what it’s exactly executing at one time, but your program will execute much faster than **Animate**. You will probably want to use this command when you have to run a mundane process more than 10 times, or else you’ll be sitting and staring for quite a while. This command becomes more useful with breakpoints, which will be discussed in the following step.

   - **Animate**: The IDE executes your C program (actually, the machine code but it keeps track of the association) and highlights which statement of your C program is being currently executed as in Figure 7.

   - **Reset**: The IDE will set the point that you want to begin execution from to the beginning of the program.
3. A good feature of the simulation tool is breakpoints. You can put breakpoints on certain lines of code to indicate that you want the program to stop executing when it gets there. As was mentioned earlier, this is useful with the Run command. A convenient way to set a breakpoint is to double click the line of code that you want the breakpoint on. In MPLAB, a breakpoint is indicated by a !. Breakpoints are very useful for testing if you are stuck in an infinite loop or if you are able to get into the state you expected. Try to add a breakpoint to your code and then select Run. An example of breakpoint can be seen in Figure 8.
4. Since MPLAB SIM simulates the computer hardware that is running the user program, it simulates the various components of the computer such as hardware registers (registers in the register file) and RAM memory (that store variables). While it is running, it these stored values maybe viewed by accessing View in your menu bar. You can explore these views on your own, but some of the views of note are Disassembly, Hardware Stack, Program Memory, File Registers, and Watch.

- **Disassembly**: Recall that an assembly language program can be converted to machine code by an assembler. This view lets you see the results of a disassembler, which is a program that will convert a machine code program into an assembly language program, i.e., mnemonics are used rather than machine instructions and labels are also used.
- **Hardware stack**: This is part of the memory that is used by function calls. It stores local variables of functions among other things.
- **Program memory**: This is where the program is stored.
- **File registers**: Memory cells in the CPU.
- **Watch**: This is a useful view in case you want to track a certain variable or register as you debug your circuit with breakpoints and Run.

5. Once your simulation is complete and you are satisfied with your design, you are ready to download the machine code onto the microcontroller. Earlier, the **Build** process produced a file of
the format called the HEX format, which is a standard from Intel. This will be file is what we will be loading into the PIC16F84A in the following section; however, if you would like to learn more about the features that MPLAB SIM provides a good starting point would be: 
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2123&param=en022520

III. Programming using the PICSTART Plus hardware

1. Connect the programmer hardware, depicted in Figure 9, to the computer. Make sure you place the microcontroller on the programmer hardware correctly, or your code will load improperly. Please refer to the programmer instructions or your TA about the chip’s placement.

Figure 9. PICSTART Plus.

2. Second, select the programmer you will be using as pictured in Figure 10; in our case, the PICSTART Plus:

   Programmer->Select Programmer->PICSTART Plus
3. Third, select the device that we will be using and configuring as illustrated in Figure 11; in our case, the PIC16F84A:

Configure->Select Device…->PIC16F84A

Figure 11. Chip Selection.
4. Fourth, set the configuration bits as shown in Figure 12.

**Configure->Configuration Bits**

These bits set up particular aspects of the chip and can alternatively be set in the program code. For now, we will use MPLAB’s nice graphical interface for this; however, practically, if you want your code to be portable to other compilers, you should eventually learn how to set these bits in your code. Also, you do not need to know what these configuration bits are right now; however, if you plan on working with microcontrollers in the future, these are common bits that you should know. If you are interested in knowing their meaning, please look them up on your own or ask your TA.

![Configuration Bits Window](image)

**Figure 12. Configuration Bits Window.**

5. Now that the chip is has been minimally configured, we can finally program it. So now enable the programmer as in Figure 13:

**Programmer->Enable Programmer**

![Enabling the Programmer](image)

**Figure 13. Enabling the Programmer.**
If the programmer hardware is connected properly, you should be able to execute all the following commands either from the **Programmer** menu or from your toolbar:

- **Blank Check**: This checks if the micro-controller is blank.
- **Program**: This is used to program the PIC microcontroller. The HEX file in the project is downloaded through the cable on to the microcontroller.
- **Verify**: This verifies that the program has been loaded properly into the microcontroller.
- **Read**: This will read the contents of the PIC microcontroller if possible.

6. Select **Program** and wait until the **Active** light on the PICSTART Plus turns off. The computer should also notify you that your program has loaded successfully. Now, you can remove the chip and test it in your circuit to evaluate if it meets your design requirements.