Laboratory #1
Introduction to the MPLABX IDE and Microchip PIC24 MCU

Objective: To get familiarization with the MPLABX IDE and the PIC24 MCU, by writing, compiling and simulating a simple program.

Introduction:

This is an introductory lab on the MPLABX IDE (Integrated Development Environment), the 16-bit PIC24 MCU, and the MPLAB XC16 compiler. Please refer to additional reference materials on Microchip's website and the course folder on Blackboard.

This lab assumes the following software components are installed on your PC:

- MPLABX IDE, free Integrated Development Environment
- MPLABX Simulator, which is integrated in the MPLABX IDE
- MPLAB XC16, C Compiler for the 16-bit PIC24 family of MCUs

You can download and install on your personal computer or laptop the software tools you need for the labs from Microchip's website.

1) MPLABX IDE is freely available software. The version we are currently using in our lab is MPLABX IDE v5.25. You can find it at the Microchip MPLABX website, click the "Downloads Archive" on the left, scroll down a bit to find the version we are using.

https://www.microchip.com/mplab/mplab-x-ide

2) The C compilers for the 16-bit MCUs and DSCs (XC16 compiler) and the PIC32 MCU (XC32 compiler) can be found at the following site from the "Downloads Archive" location, scrolling down until you find "Language Tool Archives" then scroll down to see all the available versions. The C compiler versions installed in the lab for the 16-bit MCUs and DSCs is xc16-v1.40. For PIC32 microcontrollers there are two compiles installed, xc32-v1.34 and xc32-v2.20, that we will choose from based on the specific requirements of the projects we will be working on.

https://www.microchip.com/development-tools/pic-and-dspic-downloads-archive

In the CE-420 labs we will be using the free versions of these compilers, which have some limitations on the levels of code optimizations you can achieve. However, it should still be sufficient for the kind of projects we will be working on in this class. With license fees, Microchip also offers standard and professional licenses that achieve better code optimizations. Paying customers also receive technical supports from Microchip.

Tasks:

1) New project set-up

Follow the following steps to create a new project with the MPLABX IDE.

- Start MPLABX IDE by double-clicking its shortcut on the Desktop
- MPLABX automatically opens that last project worked on. Close any open project by right clicking the project name and selecting "**Close"**, or by going to "**File -> Close All Projects"**.

- Select "**File -> New Project…**" or click the "**New Project**" command button under the menu bar. This will guide you through the steps to set up the new project.
- In the New Project window, select "**Microchip Embedded**" from the Categories, and "**Standalone Project**" from the Projects.
- The next step allows you to select the target device or processor. You need to identify the family the device belongs to and the specific device identification. Click the dropdown menu for the family and select "16-bit MCUs (PIC24)" family, and under the device dropdown menu scroll down to find the and select the "PIC24FJ128GA010" part number
- In the Header Sections step, since no header is needed select "**None**". Debug headers are usually only required for some of the 8-bit MCUs with very low pin counts. If you are using 16-bit, 32-bit or 8-bit devices with more than 14 pins, a header might not be needed.
- Next, in the Select Tool, select "**Simulator**" under Hardware Tools. A simulator makes it convenient to develop and debug programs even without requiring the actual target hardware.
- The next step is compiler selection. Under the Compiler Toolchains, select a version of XC16 compiler installed on your computer (most likely v1.34).
- The next step asks you to choose the project location and name.
- After entering your project name and location, make sure the "Set as Main Project" selection is checked, and then click the "Finish" button.
- At this point the project directory and configuration files are automatically created for the project.
- Right now the project does not have any program file in it. So let's go ahead and create a source code to add into the project. Right click the "Source Files" folder and select "**New -> C Main File…**". Give it a filename, such as "main", use the default file extension "c", and click "Finish".
- Default comment, include lines, and an empty main function are automatically added to the main.c file that was just created.
- Delete the default code and add the following program in the main.c file. This is a quick demo program we will use for experimenting with the debugger and simulator. DO NOT enter the sequence numbers shown in the source code, they are used for reference only.

2) Understanding the program

The first line of code is a compiler directive or pseudo-instruction for the pre-processor to read the content of a device-specific file before proceeding any further.

```
1. #include <p24fj128ga010.h>
2.
3. int main()
4. {
5.      int     delay;
6.      TRISA = 0;
7.      PORTA = 0x00;
8.      while(1) {
9.              PORTA ++;
10.             delay = 100;
11.             while(delay--);
12.     }
13. }
```

#include <p24fj128ga010.h>

The content of the device specific ".h" file is nothing more than a long list of names (and sizes) of all the internal special-function registers (SFRs) of the chosen PIC24 model. Those names reflect exactly the ones being used in the device datasheet. You can open the file with the MPLABX editor and take a look inside.

In our simple program we have only the *main()* function. This is the place where the program counter of the microcontroller will go first at power-up or after each subsequent reset.

One caveat – before entering the *main()* function, the microcontroller will execute a short initialization code segment automatically inserted by the linker. This is known as the *c0* code. The *c0* code performs basic housekeeping chores, including the initialization of the microcontroller stack, among other things.

Details of the PIC24 & PIC32 ports and their configuration will be covered in future lectures and lab assignments. For now, assume that line 5 of the *main()* function properly configures PORTA for output to allow sending our counter value to the output pins that we can observe in simulation view.

Note that in C language, by prefixing a literal value with 0x, we indicate the use of the hexadecimal radix. Otherwise the compiler assumes the default decimal radix. Alternatively, the 0b prefix can be used for binary values, while a single 0 prefix is used for the octal notation.

3) Building the project

Using the MPLABX IDE, the compiling and linking operation is applied to transform the program source into a binary executable. This operation is called Project Build. The sequence of events taking place during this process is composed of two steps:

- *Compiling*: The C compiler is invoked and an object code file (.o) is generated. This file is not yet a complete executable code. While most of the code generation is complete, all the addresses of functions and variables are still undefined. In fact the generated object code file is also called a relocatable object code. If there are multiple source files, this step is repeated for each of them.
- *Linking*. The linker is invoked and a proper position in the memory space is found for each

function and each variable. Also any number of precompiled object code files and standard library functions may be added at this time as required. Among the several output files produced by the linker one of them is the actual binary executable file (.hex).

To build the project:

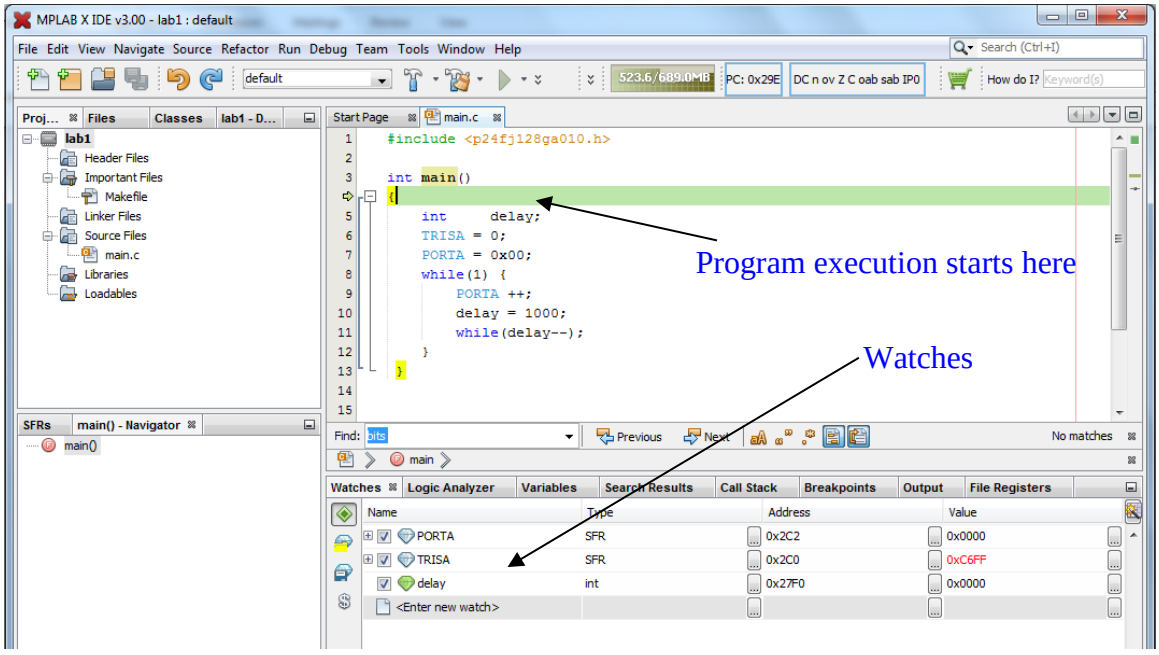
- Select "**Run -> Build Main Project**" or "**Run -> Clean and Build Main Project**" to start the compiler, followed by the linker to generate the executable code (assuming there are no compilation and linking errors, which is confirmed by the BUILD SUCCESSFUL message in the MPLABX IDE Output window).

4) Debugging the project using the Simulator

Before starting the simulator, make sure the simulator is selected as a debug tool, by looking at the project properties (**File - > Project Properties**). Under the Hardware Tools you should see "**Simulator**" selected.

There are some options we can set on whether the program starts running right away when we start debugging. In this project, we would like to stop the program at the beginning of *main()* as we enter into debug mode so that we can start to debug it however we want (such as by single stepping through it, breakpoints, etc.)

To select the debug startup option, go to "**Tools -> Options**", select "Embedded", "Generic Settings" and go to the "Debug startup" line and select "**Halt at Main**" from the pull down choices.



To start the debugger/simulator, run the command "**Debug -> Debug Main Project**" or use the corresponding command button.

The green bar at the first line of code in the main() function shows where the program counter (PC) is at. So the CPU is now ready to start executing the program.

To watch the effect of execution of the program on the variable delay and the TRISA and PORTA special-function registers, follow the following steps.

- Find the "Watches" window in the lower portion of the screen. If it is not open yet, you can open it using "**Window -> Debugging -> Watches**"
- Double click <Enter New Watch> or right click it and select "New Watch" command to type in or select a new watch from a list of Special Function Registers (SFRs) or variables.
- Familiarize yourself with the following debugging related commands. Hover your mouse on each command button to see a brief message about their functions.

- Next reset the processor using "**Debug -> Reset**" or hit the corresponding Rest button, and observe the contents of PORTA and the *delay* variable. Answer the following questions based on what you see right after reset:

Q1. What is the address of PORTA? _____

Q2. What is the address of TRISA? _____

Q3. What is the content of PORTA?_____

Q4. What is the content of TRISA?_____

Q5. What is the address of the variable *delay*?_____

Q6. What is the content of the variable *delay*?_____

- Place your cursor at line number 9 in the C source program code, and select the "Run to Cursor" option on the right-click menu. This will let you run the port initialization code and take the CPU to the first statement inside the *while* loop.
- Next, single-step through the code, using the Step-Over (F8) or Step-Into (F7) Debugger commands to execute the program statements one instruction at a time until you reach the program statement at line 11. Observe the effects of executions of these sequences of program statements on the items listed in the Watch window and answer the following questions.

Q7. What is the content of PORTA?_____

Q8. What is the address of the variable *delay*?_____

Q9. What is the content of the variable *delay*?_____

- Repeat the above two steps (run to cursor by first placing the cursor back to line 9, followed by the single step or step over), while observing the changes on the items in the Watch window.
- Practice with other debugging facilities such as Breakpoint (available by right clicking at the line where you want to set or clear a breakpoint). You may also set and clear a breakpoint by clicking the code line on its line number column.
- Next select "**Window -> Debugging -> Output -> Disassembly Listing File**" to open and view the PIC24 assembly code generated by the Microchip XC16 compiler. In this view, each C source line is shown in a comment that precedes the segment of assembly code it generated. You can even single-step through the assembly listing code and do all the debugging from this view.

Q10. From the disassembly listing file, determine the number of assembly instructions generated by the compiler for the main() function in the program.

_____

- Finally, let's find out how much memory is used by the complete program code and data. To do this go to the program build report tab at the lower portion of your screen. It is where the report about successful compilation of the program was given. Scroll through the messages displayed in that window to find information about the program and data memory usage of your program. Answer the following questions based on your observation:

Q11. How much memory space (in bytes) is used in the program memory?

_____

Q12. Why does program memory usage appear to be too high for the size of our simple program?

_____

_____

Q13. How much memory space (in bytes) is used in the data memory?

_____

Now, in your program source code, move the delay variable declaration from line 5 to line 2. Build the modified program and observe the data memory usage.

Q14. How much memory space (in bytes) is used in the data memory?

Q15. Explain your observation about the data memory usage:

5) Using the Logic Analyzer

The MPLABX Simulator's Logic Analyzer is an effective graphical tool for viewing recorded values of signals on a number of device output pins. Follow the following steps to start using the logic analyzer of the simulator.

- Select the "**Logic Analyzer**" tab in the lower half of the debug screen or select it from "**Window -> Simulator -> Analyzer**".
- At the left side of the Logic Analyzer Window, click the "**Edit Pin Channels Definitions**" command button.
- Then from the available pin list, select the pins RA0, RA1, RA2 for display in the logic analyzer.
- Next run the program for a few seconds and pause. At this point the logic analyzer should display waveforms of the three signals selected above. It uses color codes to identify the different signals.

Q16. From your observation of the waveforms of the signals in the logic analyzer window, explain what you understand about the relationship between the different signals displayed:

Check-off and report:

Demonstrate that you understand the program development process in the Microchip MPLABX environment, by showing to the lab instructor some of your debugging operations and the waveforms from the output of the Logic Analyzer.

Submit, via Blackboard, a lab report that follows the format and structure guidelines given in "Laboratory and Project Report Guidelines" document, which is available on Blackboard. Your report needs to include: Title page, Table of Contents, Objectives, Program Source Code, Disassembly Listing file, Signal waveforms of the Logic Analyzer, Answers to the given questions, and Conclusions.