Laboratory #1
Familiarization with the MPLAB IDE and Microchip PIC24 MCU

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

Introduction:

This is an introductory lab on the 16-bit PIC24 MCU, the MPLAB IDE (Integrated Development Environment) and the MPLAB C30 language suite. 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:
- MPLAB IDE, free Integrated Development Environment
- MPLAB SIM, software simulator (this is integrated in the MPLAB IDE)
- MPLAB C30, C Compiler for the PIC24 family of MCUs

You can download and install on your computer the latest version of these software tools from Microchip's website.
1) MPLAB IDE is freely available from:
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002.
2) The academic versions of the C compilers for the 16-bit MCUs and DSCs (C30 compiler) and the PIC32 MCU (C-32 compiler) can be downloaded from:
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en536656

For better code efficiency and full technical support Microchip also offers licensed versions of its C compilers. But the academic editions work well enough for this course.

Tasks:

1) New project set-up

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

a) Start MPLAB IDE by double-clicking its shortcut on the Desktop or by selecting it from the Start menu with "Start -> Microchip -> MPLAB IDE v8.20a -> MPLAB IDE"

b) Select "Project -> Project Wizard" to activate the new project wizard, which will guide you to set up a project through the following steps…

c) Select the PIC24FJ128GA010 device and click Next.

d) Select the Microchip C30 Tool Suite and click Next.

e) It is strongly recommended that you use your own storage media (USB flash drive) to save your lab projects in. Since the lab computers have public access you risk losing your work or somebody else copying your work if you don't take proper care.

f) Create a new folder and name it "lab1"; and name the project "helloPIC24" and Click Next.

g) Simply Click Next to the following dialog box – there is no need to copy any source files from any previous projects or directories.

h) Click on Finish to complete the Wizard set-up.

2) *Creating your program source code*

In this task, you are given the source code for a simple demo program that implements a counter on PORTA of the PIC24 MCU. Follow the following steps to create the program source code for your project.

a) Open a new editor window (by Clicking "File-> New").

b) Type the source code for the program given below. Do not enter the numbers on the left margin; they are used for reference purpose only.

```
1. #include <p24fj128ga010.h>
2. main()
3. {
4.      int     delay;
5.      TRISA = 0;
6.      PORTA = 0x00;
7.      while(1) {
8.              PORTA ++;
9.              delay = 1000;
10.             while(delay--);
11.     }
12. }
```

c) Select "File -> Save AS", to save the file as: "helloPIC24.c"

d) Select "Project -> Save" to save the project.

3) Understanding the program

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

```
#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 MPLAB 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 the sequence of C statements properly configure PORTA for output to allow sending our counter values 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.

4) Building the project

Using the MPLAB 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).

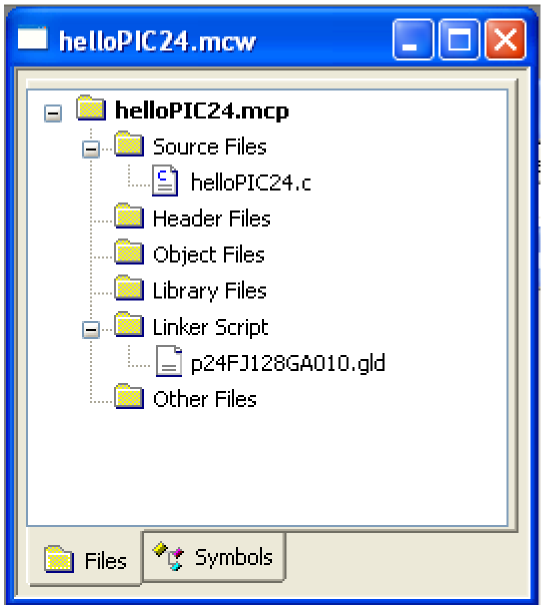
In order for MPLAB to know which file(s) need to be compiled, we will need to add their names (helloPIC24.c in this case) to the Project Source Files List. And in order for the linker to assign the correct addresses to each variable function, we will need to provide MPLAB with the name of a device specific “linker script” file (.gld). Just like the include (.h) file tells the compiler about the names (and sizes) of device-specific, special-function registers (SFRs), the linker script (.gld) file informs the linker about their predefined positions in memory (according to the device datasheet) as well as provides essential memory space information such as: total amount of Flash memory available, total amount of RAM memory available, and their address ranges. The linker script file (in this case p24fj128010.gld) is a simple text file and it can be opened and inspected using the MPLAB editor.

The steps required to build (compile and link) the project is summarized as follows:

a) Add the current source file to the “Project Source Files” List. This can be done following the steps:
   i. Open the Project window, if not already open, by selecting “View -> Project”,
   ii. With the cursor on the editor window, right click to activate the editor pop-up menu,
   iii. Select “Add to project”.

b) Add the PIC24 “linker script” file to the project following the steps:
   i. Right click on the “Linker Scripts” list in the project window
   ii. Select “Add file,” browse and select the “p24fj128ga010.gld” file found in the “Program Files\Microchip\MPLAB C30\support\PIC24F\gld” directory.

Your project window should now look similar to the figure below.



c) Select "Project -> Build All" 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 SUCCEEDED message in the MPLAB IDE Build window).

d) Select "Debugger -> Select Tool -> MPLAB SIM" to select and activate the simulator as our main debugging tool for this project.

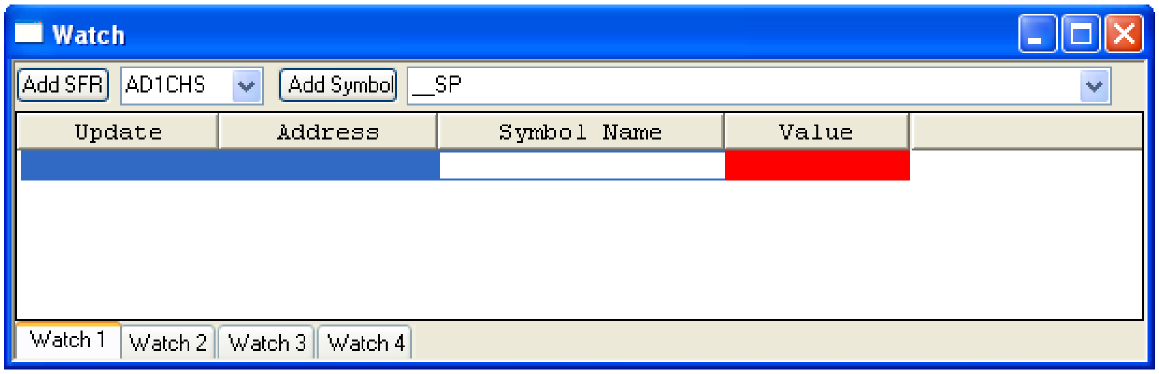
5) Debugging the project using MPLAB SIM

To watch the effect of execution of the program on the variable delay and the PORTA special-function register, follow the following steps:

a) Open a Watch window ("View -> Watch")

b) Type or select PORTA in the SFR combo box, and then click on the "Add SFR" button.

c) Type or select *delay* from the Symbols drop down list and click on the "Add Symbol"



d) Select "Debugger -> Reset" or hit the simulator Rest button and observe the contents of PORTA and the *delay* variable. Observe the address of PORTA and its initial value right after the reset. Answer the following questions:

What is the address of PORTA? ______________________________

What is the address of TRISA? ______________________________

What is the content of PORTA?______________________________

What is the content of TRISA?______________________________

What is the address of the variable *delay*?____________________

What is the content of the variable *delay*?____________________

e) Place your cursor at line number 8 in the C source program code, and select the "Run to Cursor" option on the right-click menu. This will let you run all the C-compiler initialization code (*c0*) and the port initialization to get you right to the first statement inside the *while* loop.

f) Next, single-step (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 in line 10. Observe the effects of executions of these sequences of program statements on the items listed in the Watch window and answer the following questions.

What is the content of PORTA?____________________________

What is the address of the variable *delay*?____________________

What is the content of the variable *delay*?____________________

g) Repeat the above two steps, while observing the changes on the items in the Watch window.

h) 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 double clicking at the line in the program source code window.

i) Next select "View -> Disassembly Listing" to open and view the PIC24 assembly code generated by the MPLAB C30 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.

j) Finally, open the memory usage gauge window by selecting "View -> Memory Usage Gauge". Answer the following questions based on your observation:

How much memory space (in instruction words, 24-bits each) is used in the program memory?____

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

____

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

Explain the result you observe for the data memory usage: ___________________

____

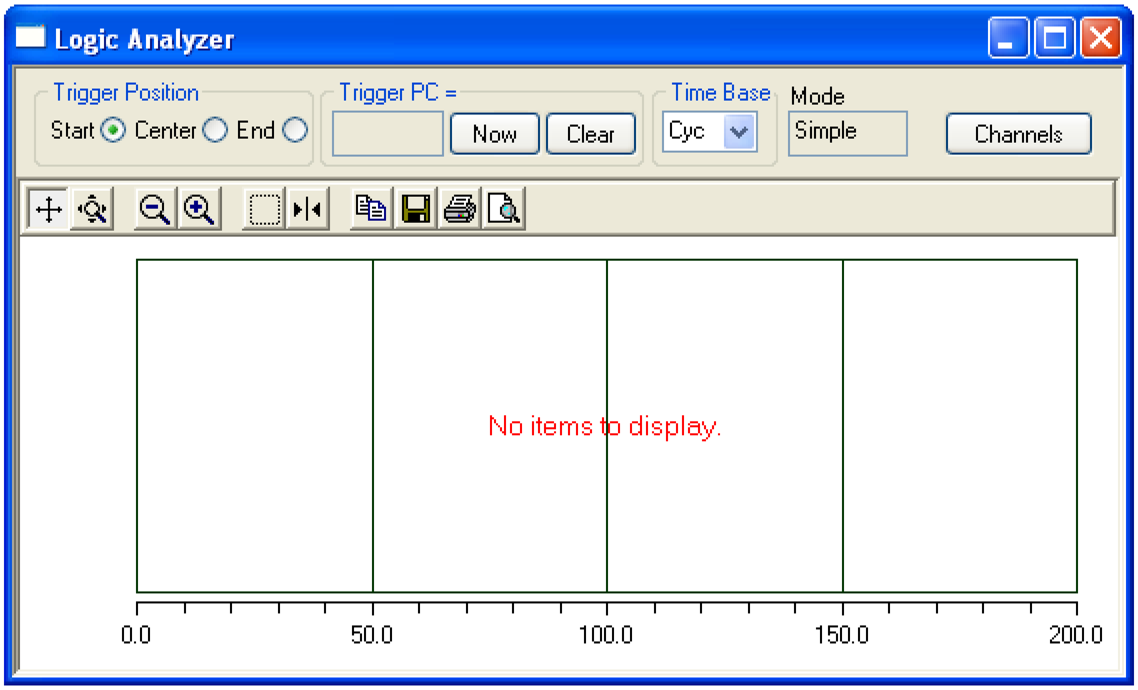
6) Using the Logic Analyzer

The MPLAB 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 turn on the Tracing function of the simulator (which is required before using the Logic Analyzer), and set up the analyzer.

a) Select the "Debugger -> Settings" dialog box and then choose the Osc/Trace tab.

b) In the Tracing options, check the Trace All box.

c) Now you can open the Analyzer window, from the "View Simulator" Logic Analyzer menu.



d) Then click on the Channels button, to bring up the channel-selection dialog box.

e) From here you can select the device output pins you would like to visualize. In our case, select RA0, RA1 and RA2 and click "Add =>".

f) Click on OK to close the channel-selection dialog box.

Run the code for a short while and hit the Halt button. The Logic Analyzer window should produce waveforms for the selected output pins of PORTA. Explain your observation on the relationship between the different signals displayed:

____

____

Checkout and report:

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

Hand in a lab report that includes the following:

- Title page: course #, course title, Lab# & title, date, professor's name, names of group members
- This lab handout with your answers to the given questions.
- Printout of the C program source code.
- Printout of the disassembly listing file.
- Printout of the signal waveforms from the output of the Logic Analyzer.