ECE 218
EXERCISE 1. LEARNING ABOUT MPLAB X IDE, C, AND STATE MACHINES

This exercise will introduce you to the MPLAB X IDE from Microchip and the X16 compiler. Starting with a pre-written program, we will learn to interface some new components and use a state machine approach to achieve a system with time ordered behavior.

This exercise assumes you are working on one of the lab computers and have brought a USB drive to lab. If you prefer to use your laptop, be sure you have the three items (IDE, compiler, and library) already installed.

Equipment: You will need your Microstick II, a LED, pushbutton switch, and two resistors (300Ω and 10KΩ).

SETTING UP A NEW PROJECT
Before starting a new project, it is important to have your Microstick II kit connected to your computer's USB port, and to have the correct PIC microcontroller (the PIC24HJ128GP502) installed on the Microstick II.

Open the MPLAB X IDE (v5.10) application, and create a new project (File-New Project).

- Choose category "Microchip Embedded", and Project type "Standalone Project".

In the next screen, you will be asked to select a microcontroller family, and a particular device.

- Family: 16-bit MCUs (PIC24)
- Device: PIC24HJ128GP502

For hardware tools, you will choose the Starter Kit (PKOB) - Microstick II starter kit. Note that it will not show up in this window if it is not connected to the USB port.

For the compiler, you will choose the X16 (v1.35).

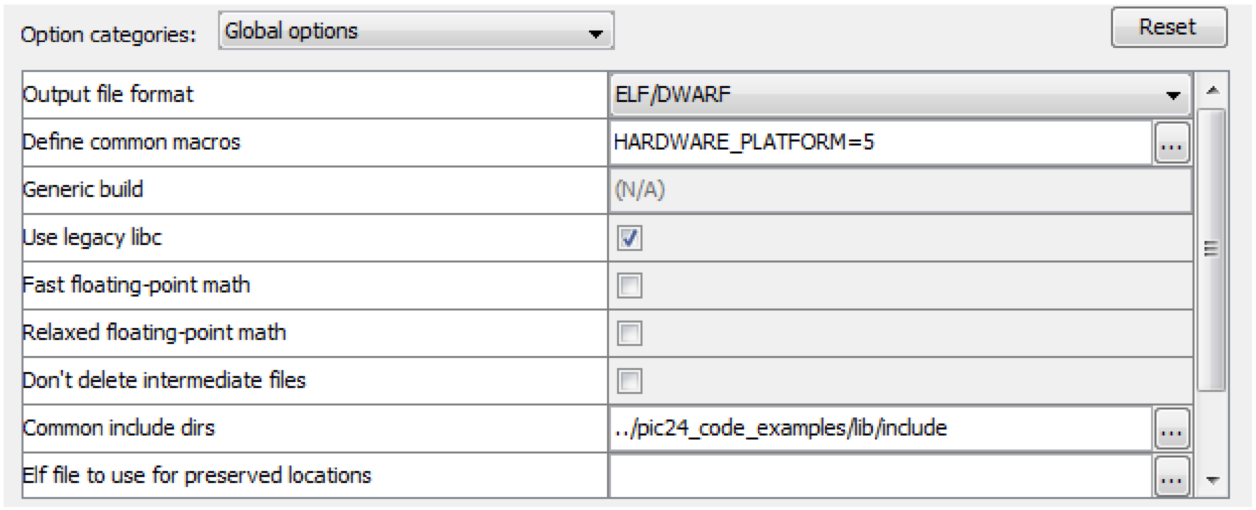
In the next window you sill select a project name, location, and folder. Note that the project location is a top level folder and the project folder is a subfolder and defaults to *project_name.X*. So the files the compiler creates will be in the path: *project_location/project_name.X*. If you check the box for "Use project location as the project folder", then both project location and folder will be *project_location*. Do not check this box.

- Project name*: LEDBlink*
- Project location: Browse to your flashdrive and create a folder for your exercises labeled *Exercises.* [Note: if you forgot a flashdrive, create a folder in the C: drive, the path to your project folder should NOT have spaces.]
- Leave "Use project location as the project folder" and "Set as main project" **unchecked**, leave the default Encoding standard as it is, and select "Finish" to complete the process.

The next steps assume the textbook library folder is installed in your *Exercises* folder.

Extra steps for the textbook library are needed to specify the Microstick II platform and to set the path to the installed library location:

1. Once the project is created, click on the **Project Properties** icon in the dashboard window (lower left).
2. In the Categories window, select XC16 (Global Options).
3. Use the "Define common macros" to set HARDWARE_PLATFORM=5 (click on … and then double-click on the "Enter text here" box and type HARDWARE_PLATFORM=5).
4. Use the "Common include dirs." to specify the path to the downloaded library. For this step you will want to specify the RELATIVE path to the library, from the project folder. Again click on … and then browse to the library *include* folder (the path will be something like: **../pic24_code_examples/lib/include).** When the Project Properties window looks like the image below, press "Apply" and then "OK":



Option categories: Global options

Reset

Output file format	ELF/DWARF
Define common macros	HARDWARE_PLATFORM=5
Generic build	(N/A)
Use legacy libc	✓
Fast floating-point math	☐
Relaxed floating-point math	☐
Don't delete intermediate files	☐
Common include dirs	../pic24_code_examples/lib/include
Elf file to use for preserved locations	

5. Add the library source folder to the project by right clicking on "Source Files" in the Projects window and selecting "Add Existing items from Folder". In the pop-up window, click "Add Folder" and navigate to the pic24_code_examples/lib/src folder and add this to the project.

Identify the following windows in the MPLAB X IDE:

- Project and Files (upper left) – this is a hierarchical list of all the files (source, header, linker, etc) in the projects that are opened. Note that you can have more than one project open at the same time, but only one will be the "main" project.
- Navigator/Dashboard (bottom left) – This is where the IDE settings for your project are displayed. You can find the PIC24 chip that you selected, and the X16 compiler, for example.
- Editor window (upper right)
- Output (bottom right)

C PROGRAM FOR BLINKING LIGHT

The source file for this first example is called "main_blink.c" and you can download it from Nexus. Place it in the *Exercises* folder (NOT in the LEDBlink.X folder). The next step is to add it to your project.

- Right-click (on a PC) the "Source Files" entry in the Project window and select "Add Existing Item". **Be sure to select the "Relative" option for storing the path to the file, so you can move your project folder later if necessary.**
- Navigate to your Exercises folder where your downloaded main_blink.c program is, and select it. It should show up under the Source File heading.

Next we will open the file in the editor (double-click on it) and take a look at all the parts. ***Notice the following types of instructions at the beginning, and briefly describe what they do:***

- /* */ Multiple line comment
- // Single line comment
- #include Declares what reference header files to use
- #define Defines a global variable or association

LED Hardware Setup

Insert your Microstick II board onto the breadboard, and use jumper wires to connect the Vdd and GND pins on the Microstick II board to the + and – rails of your breadboard. Now connect a LED in series with a 300 Ω resistor between pin 3 and the gnd (-) rail as shown in Figure 1.

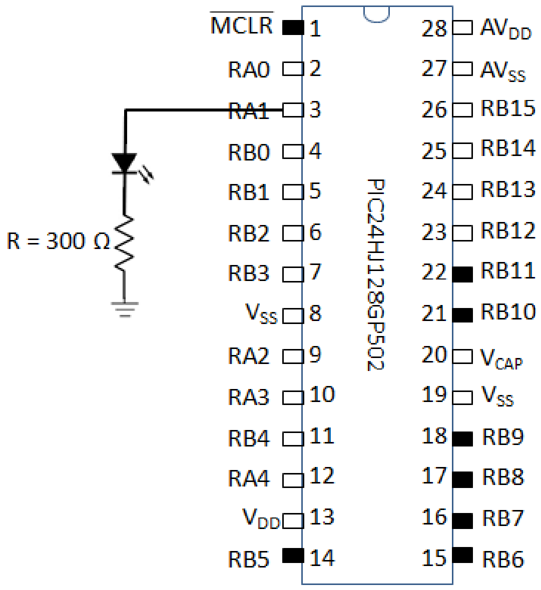


Figure 1. LED connected to RA1

Notice there are two functions called in this program, what are their names?

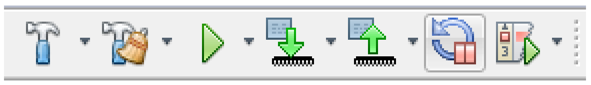
configClock() ________________ DELAY_MS() ________________

Notice the following instructions and describe what they do, both on this sheet and as comments in code.

- LED1 = 0; Turns the LED off ________________
- LED1 = !LED1; Changes the state of the LED ________________
- AD1PCFGL = 0x0002; Sets the pin to analog ________________
- while (1) {} Loops infinitely ________________

Executing Blink Program

Next we will learn about the MPLab X tools for compiling, linking, assembling and downloading machine code to the PIC part on the Microstick II board. These tools are mostly found at the top center of the IDE. Explanations of the tools are given in the attached reference page, so refer to them as you follow the instructions for your blink example.



Build your project to make sure there are no problems with the code or your setup.

Debug Project to download the program to the Microstick II board in debug mode.

Observe that the light blinks. Hooray!

Pause the program and observe where the program counter is. (displayed at top right of tool bar - you may need to mouse over the or widen the window to view it.)

PC = 0x3C1E ________

Put your cursor at the LED1 = !LED1 statement, and run the program to the cursor.

Run and pause again, and then set a breakpoint at the LED1 = !LED1 statement by clicking on the line number so that the line turns pink.

Continue to the breakpoint. Continue repeatedly to manually turn the LED on and off.

Reading Switch Input

Now we will add a switch input and create a modified C program to use this input. You will need a pushbutton switch, some wires and a resistor.

Switch Hardware Setup

Connect the switch to pin 6 (RB2) as show in Figure 2. Use a 10K resistor to pullup the pin to Vdd (3.3V).

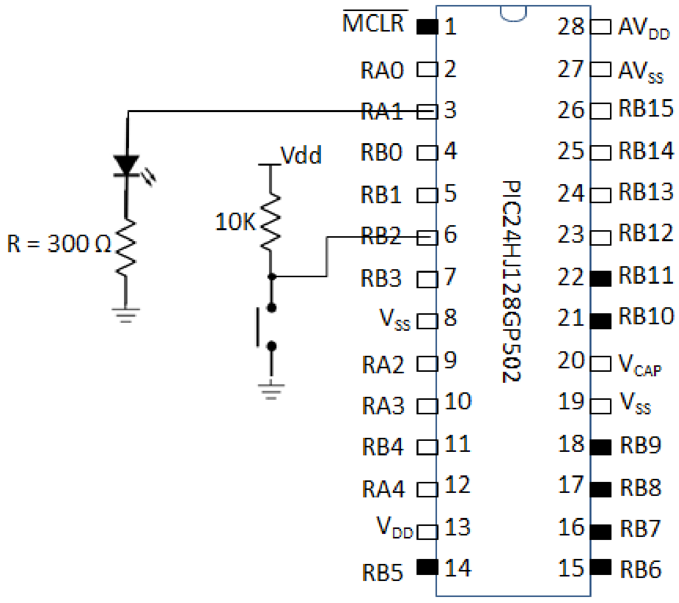


Figure 2. Schematic for Switch Connection to RB2.

<u>What will be the value of the voltage on RB2 when the switch is</u>

pressed? 3.3V   not pressed? 0V

Switch C Program Software Setup

Save the main_blink.c program with the comments that you added. Now use the *File – Save As* menu choice to save this file (in the same location as main_blink.c) as **main_blink_switch.c**. We will add some instructions to this new file to interface with the switch that was added to RB2.

Before changing the actual code, modify the comments at the top (change the file name, and add a note about your modifications. Be sure to specify the pin used for the switch).

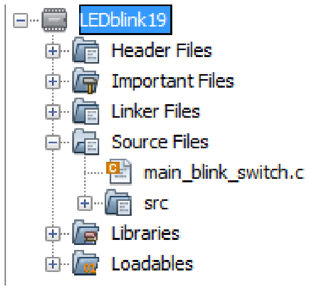
There are two changes to make so that we can read the switch:

1. Use another #define directive to define a constant "sw" to be PORTBbits.RB2. (or _RB2).
2. In the main function, set the pin we are using, RB2 (or AN4) to be digital.
3. Also in the main function, define the pin we are using, RB2, to be an **input** by setting _TRISB2 to be a **1**.

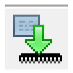
Now, add an additional statement to the main while loop so that the led blinks ONLY when the switch is pressed. Here is the syntax for the "if" statement. Remember that the equality conditional operator is a double "equal" sign: ==.

```
if(boolean_expression) {
   /* statement(s) will execute if the boolean expression is true */
} else {
   /* statement(s) will execute if the boolean expression is false */
}
```

Once the main_blink_switch.c program is completed and saved, we must make it the main program for our project. Do this in the upper left "Projects" window. Right click on the "Source Files" and use the "remove from project" to remove the main_blink.c from the project, and the "add existing" to add the main_blink_switch.c program to the project. When you are done, the Projects window should look something like this:



```
LEDblink19
  Header Files
  Important Files
  Linker Files
  Source Files
    main_blink_switch.c
    src
  Libraries
  Loadables
```

Once modified, use the  Make and Program Device tool to program the modified version to the device. Using this tool, you will not be able to use the debugging tool menu. Test the program, and once it is working, <u>demonstrate it to your instructor.</u>

Time Ordered Behavior Using a State Machine

The previous program used an "if" statement to model an output (LED blinking or not) that is dependent on an input condition (the pushbutton switch). Next we will learn to implement a simple state machine to implement time ordered behavior.

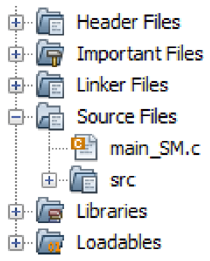
The purpose of the next program is to toggle the LED each time a button is <u>pressed and then released</u>. The hardware setup is the same as Figure 2. The framework for the program has been created, and you will be filling in the next state and output code of the state machine.

Given the purpose, why is this program considered "time ordered"?
<u>The button must be pressed before the LED will power on.</u>

The framework source file is called "main_SM.c" and you can download it from Nexus. Place it in the *Exercises* folder and add it to your project using the following three steps.

- Right click on the *main_blink_switch.c* entry in "Source Files" and use the "remove from project" to remove *main_blink_switch.c* from the project
- Right-click on the "Source Files" entry in the Project window and select "Add Existing Item".

- Navigate to your Exercises folder, select the main_SM.c program that you downloaded, and add it. It should show up under the Source File heading along with the library source folder.



Open the main_SM.c program and review the template code.

1. What constants will be used to refer to the LED output and switch input? LED1 Button
2. What is the name of the function that implements one "tick" of the state machine? Tickfct()
3. What instructions are used to set the pin direction and type? CONFIG_RA1_AS_OUTPUT()

   CONFIG_RB2_AS_DIG_INPUT()

4. How is the state machine initialized in the code? The state machine accounts for button pressing
5. What is switch bounce? Switch bounce is when a secondary pulse is sent after a button is pressed
6. What problem could it cause? Unintended state transitions
7. How does the delay function eliminate the switch bounce problem?

   Waits for the intended signal from the button

Fill in the function that implements the state machine using the Moore style state diagram in Figure 2 as your guide. The init and R0 states are completed for you.

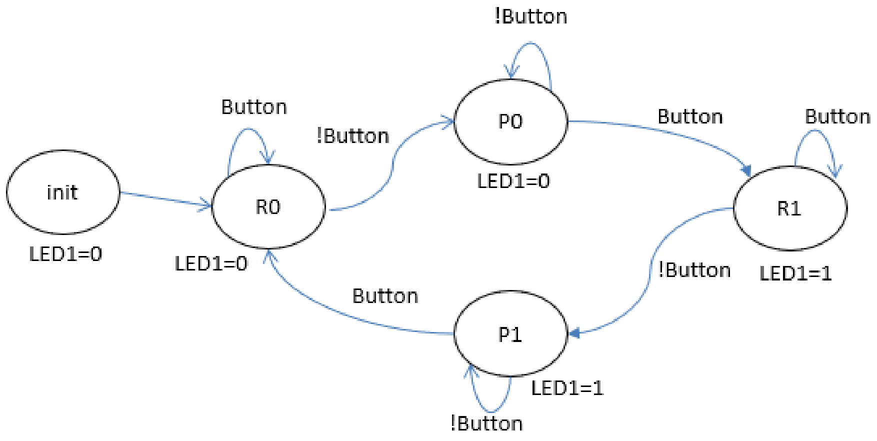
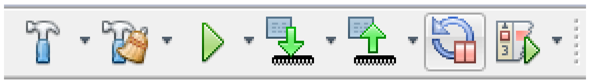


Figure 2. Moore State Machine for LED Toggle System

Testing State Machine

Once you have completed the code we will test it with your hardware interface. We will run this program without the debug mode. Follow the steps below, and refer to the reference on the last page if necessary.



Build your project to make sure there are no problems with the code syntax or your setup. If the build is successful you will see a "BUILD SUCCESSFUL" message in the (Build, Load) tab.

Make and program the device. Your code will be running, but you only have the I/O connected to observe what it is doing. You should see the following message in the "Output" tab when the program is downloaded:

```
Programming/Verify complete
```

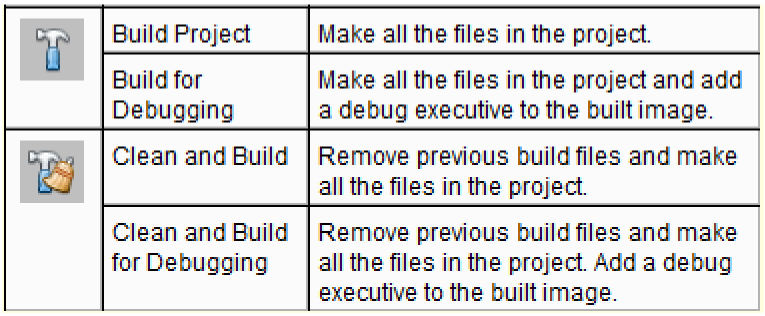
Now test the code by pressing the button and observing the LED. If it works, great! If not, it might be helpful to run the program in debug mode, and put the breakpoint at the delay function and observe the state variable as you press the switch.

To complete the exercise, demonstrate the working state machine program to your instructor. Complete the documentation of your code, and upload the final C program to Nexus.

Save your work!

MPLAB X IDE Tool Menu Icon Reference

The first time you compile a program, it is a good idea to use the Build Project tool.



	Build Project	Make all the files in the project.
	Build for Debugging	Make all the files in the project and add a debug executive to the built image.
	Clean and Build	Remove previous build files and make all the files in the project.
	Clean and Build for Debugging	Remove previous build files and make all the files in the project. Add a debug executive to the built image.

After building, you can "run" your code – note that the "Run Project" function does a "build" for you if you did not do it first, and it does NOT include the debug executive, so it will just run from the beginning, and you cannot set breakpoints, etc.

Run Project

If it makes sense to hold the device in Reset after it is programmed then use "Hold in Reset". Another click on the tool will start the program running.

Hold in Reset

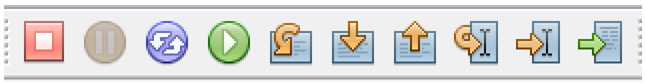
To Build and program the device (without the debug ability), use "Make and Program Device" (this is the same as "Run Project".)

Make and Program Device

When we have programs that need to be debugged, we use the Debug Project tool:

Debug Project

This tool builds your code, and programs the device with your program and the debug executive code, and starts a debug session. Once the debug session starts, the Debug icons will appear on the toolbar. NOTE – if your window is too narrow, the Debug icons are hidden. You can mouse over these tools to learn their names:



Finish Debug Session: Ends the current debug session and closes the connection to the debug tool.

Pause: Pauses debug operation without finishing the session. Watch windows are updated.

Reset: Resets processor

Continue: Resumes debugging until the next breakpoint or the end of the program is reached.

Step over: Executes one source line of the program. If line is a function call, it executes the function and then stops.

Step into: Executes one source line of the program. If line is a function call, it stops at the first statement in the function.

Run to Cursor: Run to the cursors location in the file and stop execution

Set PC at Cursor: Sets the PC value to the line address of the cursor

Focus Cursor at PC: Moves the cursor to the current PC address and centers this address in the window.

Breakpoints: You can also set breakpoints to run the program to a particular line in the code and then halt. Set a breakpoint by clicking on the left margin of the line in the Source Editor. The number of the line will turn to a red square. Click in the same place to clear the breakpoint. To run to the breakpoint, use the "Continue" tool.