Task 1

Getting Started

"It's a job that's never started that takes the longest to finish."
– J.R.R. Tolkien (1892-1973)

THIS TASK has you install and configure several of the development tools needed throughout the course. All the software used in this lab is open-source and should provide you with a very feature-rich complement of tools with which you can tackle many significant problems. Learn these tools now, and they will pay big rewards as you go forward in your career.

1.1 Software Installation

The tasks listed here are fundamentally platform-independent. Use your OS, either graphically or via a command line, to perform the tasks below. If you need assistance, your colleagues or the TA can probably give you some guidance. However, you should use them as a reference, not the system administrator of your laptop. The tasks asked of you here are very common actions that any electrical or computer engineer should be capable of performing.

In all cases, you should refer to the included documentation for each application below as you proceed.

1.1.1 MPLAB X – v.3.35 with XC16 compiler – v.1.26

Obviously, we need development tools for the MCU target used in the lab. Microchip updates their tools almost weekly. However, we will standardize on the latest release as of the beginning of this semester. Unless instructed otherwise, do NOT update your tools after the beginning-of-the-semester installation.

1. Install the version of the Microchip IDE and compilers listed above.
2. You will verify the MPLAB installation at a later point in this lab.....

1.1.2 Python – v.3.4.5

Python [15] is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. Python was first released by Dutch compuer scientist Guido van Rossum in 1991 and has grown in popularity very rapidly. It is considered by many to be one of the "major" programming languages in use today. And, Python got to this point without the marketing and resources of a major company pushing it, e.g. Sun's Java/JVM and Microsoft's C#/.NET. Some people view Python solely as a scripting language, and it can be definitely used for that purpose. But, Python is also a valid language and platform for substantial product development,[1] especially internet-based applications.

There are several very nice Python tutorials and beginner books are available online. I recommend [12] and the Python tutorials [9] by Dr. Norm Matloff at UC-Davis' CS department. As we progress to more advanced Python topics, you'll probably want a comprehensive discussion of using Python for complicated tasks. I *strongly recommend* what is considered to be the leading book [7] on using Python in real-world applications. I refer to this book almost everytime I start a new Python project. It is exceptional and very thorough. Buy your copy today. You won't regret it.

You will need a Python installation on your machine.[2] Fortunately, Python is free and available for nearly every computer platform on earth, including many computer platforms that have not been manufactured for decades. In order that everyone in the lab is compatible, you will need to install the most recent "production" version of Python3[3] at the time of writing.

1. Download Python from the Python website [15]
2. Run the appropriate installation program
3. Be sure that `python` is on your `PATH` environment variable, so that you can fire up Python from any folder in your file system. More importantly, Python applications will be able to start/spawn other Python applications from any point in your tree.

There are literally thousands of useful Python libraries and applications[4] out there to do almost anything you can dream up that a computer might do. Feel free to search for them and try them out. However, you may *not* use any third party Python libraries (modules, in Python-ese) without *prior permission* of the instructor.

[1] At least, this little company in Silicon Valley called Google seems to think so. Google uses Python for many of its applications. Furthermore, Google has hired Guido van Rossum away from his previous employer and given him 50% of his work week to work exclusively on Python development.

[2] If you are running Linux or Mac OSX, your computer already has Python. You don't have to install a more recent version of Python, unless you want to. Alas, Microsoft prefers to provide their proprietary .NET framework to its users. Windows users will have to download and install Python. *Python(x,y)* is a third-part amalgam of Python and various modules, libraries, and applications. It is highly regarded as a single-install method of getting started with Python.

[3] Python 3 makes some fundamental changes to the syntax of the Python language found in revision 2.x and earlier. At this point, it is believed that the third party Python modules used in this course are completely compatible or workarounds exist when using Python 3. I reserve the right to "downgrade" back to the $2.7.x$ version of Python if a significant incompatibility is found later in the course.

[4] In particular, I recommend `iPython` as a replacement for the Python command shell. Stani's Python Editor (`SPE`) is a very nice lightweight Python IDE built with Python itself. Spyder is an excellent full-featured Python IDE and is highly recommended. And, the latest version of `winpdb` is a very nice graphical Python debugger. Google these names to find the appropriate project websites for these free Python applications.

1.1.3 PySerial Module – v.3.1.1

Python programs are meant to be platform-independent much like the way Java intends to be. The Python installation you just downloaded includes an embedded Python virtual machine (PVM). Unlike Java though, Python does not publish the specifications for the PVM, reserves the right to change the PVM internals from release to release, and very strongly discourages users from making any assumptions about the underlying Python bytecodes and PVM implementation. When you type Python code, it gets parsed, interpreted, and compiled into Python "bytecode" by the Python application. The Python application then *immediately* runs it on its internal, embedded PVM to produce the results you see at the prompt. Python executes its statements one-by-one as it comes across them. There is no Python compilation step *per se*.[5]

Since Python offer platform-independent program execution, hardware dependencies[6] must be resolved and dealt with somehow. This is the job of many third-party external Python modules. Since Windows, Mac OSX, and Linux all handle the hardware[7] differently, a Python module is needed to create a set of common Python objects for code to use which will be mapped on the hardware in the manner appropriate for the hosting computer's OS.

You will need to download the PySerial module. The purpose of PySerial is do the mapping of platform-specific hardware and OS to the platform-independent Python object for serial port accesses. The module provides a single Python object interface which can be used by the user's (read: your) programs. The PySerial module will then take your user programs requests and make the required OS-level requests to access the hardware. On Windows, Python will make Windows OS calls on your behalf; on Linux, it will call the Linux device drivers; on MacOSX, it will call the Macintosh device drivers, etc. Viola, we can write *one* program to use a Python-view of serial ports and the PySerial module just magically makes it work regardless of the OS that is hosting us. Sweet!

You'll definitely need this PySerial module in your toolbox to communicate via your laptop's serial ports, virtual[8], or otherwise.

1. Download and install the PySerial module[9] from the PySerial web site [14]
2. Verify that PySerial works by running all of its sample programs. Several of these scripts require a serial loop back connector. It is very easily make your own and is strongly encouraged.

1.1.4 Mercurial – v3.9

One way for you (and any development team you're a part of) to share and keep track of your project files is to use a revision control system like Mercurial. Mercurial allows you to keep your

[5] Python does "compile" the code in module `.py` files into `.pyc` files which contain Python bytecode. This does not provide execution time performance improvement, except that it saves Python having to parse the code into Python bytecode everytime you run that module.

[6] Oh, like your computer's serial port, for example.

[7] Again, use your serial ports as an example.

[8] There exists a pyUsb module for Python. But, we will not use that module. If you use a USB "dongle" to communicate with the serial port, Python views your USB device as just a fancy (and fast) virtual communications port, and PySerial is still the appropriate module to use.

[9] If you have ever used the Java Serial Communications API, you will very much appreciate the simplicity of Python here. There is only one PySerial modules for all Python-running computers. No futzing about with goofy third party C-based hardware library "backends", partial installs of the Sun javax.comm library, JRE vs. JDK install locations, etc. Just install PySerial via `python setup.py install` and go. It just works! Batteries are included.

project files in a central location so that all the team members can access it, as well as, keeping track of any file revisions. A complete history of code development can provide invaluable during a rapid design timeframe where lots of changes are happening very quickly. You'll experience lots of design projects like this in your career.[10]Mercurial is an open-source "source control management" project [17] with clients[11] available for nearly every computer platform on earth. Mercurial is used by hundreds of thousands of code development projects around the world and by every major company you can think of. Download the latest Mercurial client from [17] and install it on your machine.[12]

Mercurial is quite easy to use once you understand what Mercurial can do for you. There are several good tutorials on Mercurial on the web, a wiki entry on the MSU ECE wiki server [11], and a very nice free book [4] online. Read through this material to get a good idea of how Mercurial works. The standard Mercurial client at the command line works great and gives you complete control over the repository process.

As a multi-person team commits into a repo, it can become confusing as to what has been committed and by whom. A visual representation of the repo can be helpful. TortoiseHg [18] client is very nice and is highly recommended. TortoiseHg is available for Windows, MacOSX, and Linux platforms.

1. Install the Mercurial client.
2. If Mercurial is new to you, work through the Mercurial tutorial[13] to understand what revision control and Mercurial can do for you. The tutorial is at *http://mercurial.selenic.com/wiki/Tutorial*.

BitBucket is a third-party server that allows you to create a Mercurial repository for free, and large repos for a small fee.

1. After you have been through a Mercurial tutorial, use the `hg` (or tortoisehg) client to "clone" the Mississippi State University PIC24/dsPIC33 hardware libraries[14] written by your illustrious faculty. You can find the repo at *bitbucket.org/bjones/pic24lib_all*
2. Verify MPLAB's operation by compiling the project `chap08/ledflash.X` from the MSU libraries. Generate the "hex" file corresponding to this project. (NOTE: Do not make any changes to the code. Simply compile the target.)

1.1.5 SConstruct – v.2.5.0

Building large software projects can be a complex task, especially when there are multiple deliverables and/or target platforms. Software build systems (often call "make" systems after the early unix build tool) are powerful, but are notoriously cryptic. To make things easier to understand and more flexible, a Python-based build system has been written. It is named "SConstruct" but is typically called `scons`.

The hardware libraries and ESOS written here at MSU uses `scons` as the build system. You will need to install `scons` in order to fully compile several of your targets this semester.

[10]In fact, I predict there will an embedded systems experience over a short 15-week period in your very near future!

[11]The Mercurial client program is usually named `hg` after the symbol for the 80th chemical element mercury.

[12]If you run Linux or MacOSX, you likely already have Mercurial installed.

[13]This tutorial uses the basic Mercurial command-line client. GUI-based clients like tortoise simply provide a "pretty" interface to the underlying commands. If you wish to use the GUI tortoise client, I would recommend going through the tutorial a second time and simply find the appropriate button to press in the GUI to trigger the command-line behavior described in the tutorial.

[14]This repo is still being actively developed. You will need to periodically update your local tree with the latest revision as we progress through the semester. When major changes/fixes are pushed to the repo, you will be alerted.

1. Download and install `scons` from the SConstruct web site [16]
2. Learn more about `scons` operations via the `scons` tutorial on the SConstruct web site.
3. Verify that `scons` works by building the following targets:

```
scons build/33EP512GP806/chap08/echo1.hex
```

1.2 Software Development

Develop a Python module called `weather.py` that contains a temperature class and associated methods. You will develop this module according to a predefined set of unit tests that your module and methods must pass. (Read Chapter 13 in [12] for more details on Python unit testing.) The unit test file `temperature_unittest.py` is located on the lab website. These unit tests will be run against your module by the TA before you can checkoff this task.

Your module should contain the necessary `Exceptions` required to signal when the user requests a temperature that is below absolute zero. Furthermore, your temperature class should have methods that convert the class's temperature value (attribute) to Celsius and to Fahrenheit.[15] Your Python code must adhere to the community Python coding standard PEP 8 located at [15]. Comment your code thoroughly. You might want to consider using a documentation utility like Doxygen [6].

1.3 In-lab Checkoff

- Demonstrate to the TA that all installed software is running properly on you laptop. You may be asked to exercise any PySerial tests of the TA's choosing or *echo1.hex* when connected to the TA's target development board.
- Demonstrate the correct operation of your `temperature.py` module to the TA, including passage of unit tests.

1.4 Submission

- Email the compiled .hex file `chap08/ledflash.hex` to the TA.
- Email the compiled .hex file `chap08/echo1.hex` to the TA.
- Email the python file `weather.py` to the TA.

[15] You may provide other temperature conversion and/or utilities, if you choose. For example, my temperature class stores all temperatures internally as degrees Kelvin and can convert between all three scales.