Milestone 1

Getting Started & "Hello world!"

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

THIS MILESTONE 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 milestones listed here are fundamentally platform-independent. Use your OS, either graphically or via a command line, to perform the milestones 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 milestones 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 Programming Editor

Every good computer professional needs a solid programming editor. I will leave the choice of editor up to you. I personally mostly use *geany* on my linux box. There are also some really good open-source programming editors[1] available across the popular platforms. Some people prefer to use an IDE like Eclipse; however, IDEs are used a lot less in industry than you might think. Every place I have ever worked, the official corporate development process was use programming editors and the product build process was automated with build tools. IDEs simply get in the way – they are constantly changing and often break the corporate build processes, etc. You may use choose to use some IDE for your development and program builds, but no IDEs will be supported by myself or the TA.

1. Install your favorite programming editor

[1] Notepad++, kite, atom, and the venerable vi and emacs.

1.1.2 make build tool

Compiling and linking (building) complex software is usually a non-trivial process. Large software project builds can take appreciable time. To reduce build times, you often want to only compile source files that have change, then link object files together to produce the product. Many build tools can assist in this process. While there are some really nice, more modern build tools, we will use the venerable, if not a bit unfriendly, *make* utility. If you run Linux or MacOSX, a suitable version of *make* is already installed on your computer. You are done. If you run Windows, you will likely need[2] to install a *make* utility. By the end of this milestone, you need to make sure you have *make* and it works.

1.1.3 Python – v.3.10.1

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,[3] 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.[4] 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 Python 3[5] 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.

[2] I *think* the ARM toolchain includes a bundle called binutils that has a *make* utility for the Windows users. Someone will need to verify this as I am not a Windows users.

[3] At least, this little company in Silicon Valley called Google seems to think so. Google uses Python for many of its applications.

[4] 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.

[5] Python 3 makes some fundamental changes to the syntax of the Python language found in revision 2.x and earlier. I think all of the tools we will use in this lab are now compatible with Python 3. If we discover that this is false, you will need to install that latest Python 2 version: 2.7.18.

There are literally thousands of useful Python libraries and applications[6] 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.

1.1.4 PySerial Module – v.3.5

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*.[7]

Since Python offer platform-independent program execution, hardware dependencies[8] 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[9] 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[10], or otherwise.

1. Download and install the PySerial module[11] from the PySerial web site [14]

[6] 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. Of course, IDLE comes built-in with Python. 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.

[7] 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.

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

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

[10] 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.

[11] 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.

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.
3. PySerial should include an example program called "miniterm" that you can use to communicate throught the serial port to your target board. PySerial also has an example application called wxTerminal that uses the wxWidgets library to make a GUI-based terminal program. You will find these apps useful in our work, and we may "modify" them to give customized functions to interact with our target system. If Qt is more your cup of tea in GUI libraries, there are several nice pyqt-based serial communications terminal projects out there.

1.1.5 A serial terminal utility

You will need a method by which you can interact with the serial communications on your PC. As stated above, the pyserial examples contain a simplistic serial terminal. Other good choices on Windows are PuTTY and RealTerm. There are a variety of choice for serial terminal software for Mac and Linux. Whichever one you choose should have the ability to adjust the serial port, baud rate, parity, start/stop bits, and the ability to provide local echo. It is very useful if the utility can also log the serial data stream to disk for archival and post-processing.

1.1.6 Git – v.2.34.1

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 git. git allows you to keep your 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.[12]git is an open-source "source control management" project [17] with clients available for nearly every computer platform on earth. git 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 git client from [17] and install it on your machine.[13]

git is quite easy to use once you understand what Mercurial can do for you. There are several good tutorials on git on the web. Read through this material to get a good idea of how git works. The standard git 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. Several students have recommended the SourceTree [18] client for Windows and MacOX. For Linux users, I recommend GitAhead or GitKraken.

1. Install the git client.
2. If git is new to you, work through the git tutorial[14] to understand what revision control and git can do for you. The tutorial is at *http://mercurial.selenic.com/wiki/Tutorial*.

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

[13]If you run Linux or MacOSX, you likely already have git installed.

[14]This tutorial uses the basic git command-line client. GUI-based clients like tortoise simply provide a "pretty" interface to the underlying commands. If you wish to use a GUI 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.

BitBucket and git-hub are a third-party servers that allows you to create a git repository for free, and large repos for a small fee.

1. After you have been through a git tutorial, you will use git (or a git GUI client) client to "clone" the some repositories for subsquent library installations. While many of the next tool libraries may have a "binary installer", you will want to clone their repo so that updates to your local copy is quick and painless.

1.1.7 ARM Cortex-M tool chain - v.10.3-2021.10

Now, we need a compiler, linker, and other assorted build tools for our 32-bit target processor in the ARM Cortex-M families. You should find the latest toolchain at *https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads*

1.1.8 OpenOCD - v0.11 (optional)

OpenOCD is a community developed, open-source in-circuit debugger. STMicroelectronics and several other silicon vendors participated in its development. It's development appears to have stagnated. We probably will NOT use this utility, but you might find it useful If you decide to try it, install the the latest version v.0.10. Note that this is different from v0.1.0 which is much older.

1.1.9 STM32CubeProgrammer - v.2.9.0

The ARM family from STMicroelectronics is vast, and STMicroelectronics has developed much code support and libraries for their offereings. You may install the STM32CubeMX libraries and study them. However, you are not allowed to use the STM "Cube" libraries and HAL in your designs in ECE 4140/5140.

You will want to install STMicroelectronics' GUI programmer utility, SMT32CubeProgrammer. It is the easiest-to-use STM32 programming utility out there.

You can download from *https://www.st.com/en/development-tools/stm32cubeprog.html*.

1.1.10 libOpenCM3 Library

The libOpenCM3 library is a lightweight hardware abstraction library initially developed for the ARM Cortex-M3 family of MCUs. It has since been ported to the M0, M0+, and M4 families. The ARM Cortex-M4 family port is fairly complete. We will use this library throughout the semester to hide some of the hardware details. This lightweight HAL is being actively developed with new features and fixes being pushed up nearly a daily basis. Install the latest version from the libopencm3 repo at github. You will need to keep this library in its own separate folder as we will likely "pull" an updated version (or two or three) down from the repo before the semester ends.

1.1.11 Update ST-LINK firmware on your target board

The target board you were issued likely has an older ST-LINK firmware running on it. You might need to update it to the latest version.

1.1.12 Texane's stlink utility (optional)

Git contains an open-source command-line utility for using the ST-LINK interface with our Nucleo board at *https://github.com/texane/stlink*. You will need to build the latest or install a bundle appropriate for your OS.

1.2 Toolchain Verification

After you have installed your tools and drivers. You may need to update the ST-LINK firmware on your Nucleo board so that the flash programmer and other tools work properly. The ST-LINK update firmware has easy-to-follow instructions.

1.2.1 Flashing user code

Now, it's time to test our programmer.

- Start up STM32CubeProgrammer.
- Connect to the target board with the STM32CubeProgrammer with "Connect". Verify the chip identifiers on the right.
- Hit the "+" tab, and open the provided code bundle *test0.hex*. It should be available on the course website.
- Press "Download" in the upper-right. (Sometimes it fails on the first press. If so, try pressing it again. If that doesn't work, then hit "X" on your program tab. Reload the hex file and try to program again.)

After the hex file downloads into the ARM processor, you should see the Nucleo LED LD2 flash a pattern:

- LONG-pause-LONG-pause-SHORT-SHORT-LONG
- longer pause

which is "TTU" in Morse code. Identify the location/name[15] of the virtual communications port (VCP) corresponding to your Nucleo target. The sample program communicates at 115.2 kbaud 8/N/1. Connect to your Nucleo board with your serial terminal utility. Demonstrate the ability to interact and control with the board via your screen and keyboard.

[15] Where your board's serial port mounts is OS dependent. On my Linux box, it usually appears as `/dev/ttyACM0` or `/dev/ttyUSB0`. Sometimes, the final character number changes. The `lsusb` command may help you resolve which device is which in the linux USB subsystem. On a Windows box, the serial port is usually COM*x* where *x* is some number. The "Device Manager" in the "Control Panel" may help you figure out where your Nucleo is connected.

1.2.2 Writing some code

The class lab website contains some skeleton code in the m1a directory to get you started. In the "src" folder, you should find m1a.c and user_app.h. You will need to add some additional information about your board and write your main() code. Magic numbers are not allowed. Use the constants in the libOpenCM3 header files. Comment every line of code liberally explaining what each and every item is doing. Follow the coding conventions given in class and the conventions document on the class website.

1.2.3 Compiling

Return to folder above "src". Type "make" and hopefully all compiles cleanly to completion. If not, fix the errors[16] until it does.

1.2.4 Flash the code

Use the CubeProgrammer tool to download your code to the target board. Devise a test (or set of tests) that will verify the program's correct operation.

1.2.5 Your first ARM program

Go back into your code. In the comments at the top of your m1a.c file, answer the following questions.

- What is the period of the blinking LED blink? Be exact. Explain how you measured the period.
- The skeleton code did not do much to configure the MCU main system clock. What is the frequency of the main system clock? How do you know? How could you physically verify that?
- Examine the listing[17] file (`m1a.lst`) created when `m1a.c` is compiled. Explain how to find the exact period of the blink from the listing file contents before you every flash the program to hardware and run it.
- Now, explain how to change the value in `delay_loop()` to get a LED flash period of exactly $\frac{1}{7}$ second. Make the change and verify your effort.

[16] Unfortunately, every computer is a bit different depending on which OS, which version of your OS, the number and ordering of updates applied, number and location of storage devices, file system organization, etc. etc. etc. That means there are no universal instructions that I or anyone can provide to help you troubleshoot your toolchain. You will have to research online, talk to your classmates, the TA, and myself to get your toolchain working correctly. This is a crucial skill. Your employer will expect you as a CmpE or EE to be able to intall, troubleshoot, verify, and deploy industrial-grade software development and EDA tools. In short, you need to develop wickedly-good system software skills Windows, MacOS, or Linux. Better yet, all three.

[17] Listing files – typically with a `.lst` extension – provide valuable insight into what code the compiler creates from your source. THEY ARE IMMENSELY IMPORTANT TO THE EMBEDDED SYSTEMS DESIGNER. You often have the ability to fine-tune the information contained in the listing files. Consult the compiler documentation. Get familiar with listing files and refer to them often to see what instructions your processor is "really" running.

1.3 Your second (and beyond) ARM program

Now that you have writing a few lines of code and gotten proficient with the toolchain and flash programmer, now we will start to really take control of the ARM Cortex MCU. The class website contains another code skeleton `m1b.c`. Place this code in a new "project" folder. You will have to provide the makefile and appropriate file structure.

- Complete the m1b.c skeleton code to setup the processor to run at a 80 MHz system clock and flash the Nucleo user LED. As before, examine the listing file produced when you get your program working. Predict, then measure the LED blink period.

1.4 In-lab Checkoff

- Demonstrate to the TA that all installed software is running properly on you laptop. Individually you may be asked to exercise or demonstrate any or all of the steps in this milestone.
- Demonstrate the correct operation of `test0.hex`., and your ability to connect to the target using the VCP.
- Demonstrate the correct operation of your project `m1a`, and `m1b`.

1.5 Submission

- Sumbit a single archive `imstudent42_m1.zip` file or `imstudent42_m1.tgz` file via iLearn. Your archive should have a consistent directory structure. Apart from changing the specific locations of the libraries in the Makefile, the TA should be able to simply type "make" to build your application to run on his/her hardware. If this is not the case, you will receive zero credit.