

Installing & testing programming tools on macOS machines

Prepared for ECE-231 Spring 2024. This is version 2.0 of this install document

This note provides instructions for downloading the VSCode editor and the AVR toolchain (compiler and other tools) onto a computer running macOS. You should be using an up-to-date version of the mac OS, such as mac OS 12 Monterey, or 13 Ventura or 14 Sonoma. There are many steps involved in downloading, installing, and setting up these tools on your mac. You will need to set some environment PATH variables on your machine and use the Terminal command line app throughout. You should carefully read thru these instructions, twice, before you begin to download anything.

A few words about the terminal and the PATH variable. In ECE-231, we use the Terminal App extensively rather than the graphical user interface on macOS machines. Whenever we start a new coding project, we will create a new folder, or subdirectory, for that project and we will be working within that directory. In order for the binary executable files associated with VS Code, avr-gcc, avrdude, make, etc... to be able to run from different directories, we need to have the directory paths for these files included in the environmental PATH variable. More in this below...

VS Code text editor

Visual Studio Code (aka VS Code) is a no-cost text editor owned by Microsoft. This is a popular editor among programmers because it has a number of features (eg, color coding, indentation, syntax, error checking, code folding, file management, etc..) that help to develop source code. Note that VS Code is not the same as another similarly-named Microsoft product, Visual Studio. Studio is a full blown Integrated Development Environment, while VS Code is simply a text editor that happens to have many features helpful for creating source code.

Installing VS Code

1. Goto <https://code.visualstudio.com/docs/setup/mac> and follow the Installation instructions there to download and install VS Code to your Applications folder and install a shortcut link in your dock. Important: Be sure to follow the instructions for "Launching from the command line".
2. To test your installation, quit VS Code. Then open the Terminal App (found in /Applications/Utilities/Terminal.app) and type `%code .` (Note: in this document, text formatted like `this` refers to text you should type into the Terminal window. The `%` preceeding such test represents the terminal prompt for your input; your prompt might look different from this.) When you hit return, VS Code should open.
3. Under the VS Code file menu, be sure that autosave is checked. If not, check it.

4. Add the Terminal to your dock, since we'll be using it frequently. Quit terminal and quit VS Code.

avr-gcc toolchain

Use the avr-gcc compiler and a set of associated tools (collectively called a tool-chain) on your macOS machine to compile source code into AVR machine language understandable by AVR processors such as the ATmega328p and the ATtiny85. Access these tools by downloading the AVR 8-bit GNU Toolchain freely available from the Microchip web site. (Microchip is the company that manufactures the AVR ATmega328P mcu.)

Installing the avr-gcc toolchain

1. Google "microchip avr gcc compiler" or goto <https://www.microchip.com/en-us/tools-resources/develop/microchip-studio/gcc-compilers>
2. Scroll down and download "AVR 8-bit Toolchain (OSX)" version 3.7.0
3. A compressed tar file will download, probably to your download folder, depending on your machine settings. Move this file into your Applications folder and double-click to open it.
4. You will now have a folder named "avr8-gnu-toolchain-osx-3.7.0.518-darwin.any.x86_64" that contains numerous subfolders. Rename this folder "avr" and delete the downloaded tar file since that is no longer needed.
5. You now need to adjust the environmental PATH variable on your system so that it can find the binary files in the avr toolchain. You need to do two things:

5.a. Create a simlink (symbolic link) to the avr toolchain: open terminal and type the following three lines (remember: % is the prompt. Don't type that.)

```
% cd /usr/local  
  
% sudo ln -s /applications/avr avr  
  
% cd $HOME
```

You will be asked to provide your computer login password after the sudo command.

5.b. Modify your shell startup file so that your PATH variable contains the above link to the avr binary files. Your laptop probably is set up for the "zsh" shell by default, so you will be editing or creating the .zshrc file or the .zprofile in your home directory. If your mac is using the "bash" shell, then you will be editing or creating the .bash_profile in your home directory. You can edit/create these files using VS Code. Type "code ." to open up all the files in your home directory and look for .zshrc or .zprofile (if you're using .zsh) or .bash_profile (if you're using the .bash shell). Either edit the appropriate file or create a new one if needed, adding the line:

```
export PATH=/usr/local/bin:/usr/local/avr/bin:$PATH
```

5.c. Quit VSCode, close Terminal then re-open it for this new setting to take effect.

5.d. Type `% which avr-gcc` and you should see something like `/usr/local/avr/bin/avr-gcc`. Now type `% avr-gcc --version` and you should see something like `avr-gcc (GCC) 7.3.0`. This means you have successfully installed the avr toolchain. So far, so good!

Homebrew

Homebrew is a software package manager for macOS. Once installed, homebrew helps with the installation of other software, handling dependencies, PATH variables, and other things that need to be set up for software to work correctly. Homebrew needs the macOS xcode command line tools in order to operate (not the full xcode system) and will install them during its own installation. Homebrew is similar to MacPorts and Fink, and if you already have one of these installed and know how to use it, feel free to use that instead. We recommend you install Homebrew unless you have an alternate preferred way of installing code on your mac.

Installing Homebrew

goto <https://brew.sh/> You will see a link under the words "Install Homebrew" that looks like this:

```
/bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Copy that link using the copy symbol shown, open Terminal, and past the entire link contents on the command line. Press return, and homebrew will install. You will be asked for your password, then asked to hit return. It will take some time, and it will install xcode command line tools along the way. Be patient. You may be asked to provide your computer login password a few times to give permission.

Important: when Homebrew finishes loading, you will see "Next steps: Run these two comands in your terminal to add Homebrew to your PATH". *Carefully copy those two lines, one at a time, into your Terminal command line and hit return after pasting each line.*

Quit, then reopen Terminal for these path changes to take effect.

avrdude

avrdude (avr downloader/uploader) is the program used to flash compiled code to an AVR microcontroller. You can install it using Homebrew.

Installing avrdude

Once homebrew is installed, you can install avrdude simply by typing `% brew install avrdude` at the Terminal prompt. Homebrew will respond by fetching and downloading numerous files. When it finishes, check the avrdude installation by typing `% which avrdude` you should see `/usr/local/bin/avrdude` (on a mac with intel processor) or `/opt/homebrew/bin/avrdude` (on a mac with Apple silicon processor). Then just type `% avrdude` and you'll see a screen full of menu options. This means avrdude was installed correctly.

make

Make is a program that automates the multi-step process of compiling, linking, copying and flashing a program to a microcontroller. A version of make comes installed as part of macOS, but it needs the xcode command line tools to work, and those tools will have been installed when you installed homebrew. To test you make program, type `% which make` and you should see `/usr/bin/make`. Next, type `% make --version` and you should see something like GNU Make 4.2.1 or GNU Make 3.81. This means you're ready to proceed.

Build & Flash your first program

If everything so far has been successful, you are now able to write a source code file in VS Code then compile it from the command line and download the machine code to the flash ROM of your AVR MCU. Create the classic "blink" project to test everything. The following instructions assume you are coding for an ATmega328p MCU on an Arduino Uno development (dev) board.

1. First, make a directory called "codingprojects" in your home directory. This is where you will be storing all your embedded coding projects. Next, move into this new directory. Then, create a new directory called "blink" for the project that you will now build. Move into that directory. Then open VS Code from that directory. You do all this with the following Terminal commands:

```
% mkdir codingprojects
% cd codingprojects
% mkdir blink
% cd blink
% code .
```

This will open up VS Code in the blink project folder. Create a source code file called "blink.c" by first hovering, then clicking on the + file box toward the top of the window.

2. Next, type in source code for a blinking LED. You can copy it from here: <https://github.com/ProfMcL/ECE231/blob/main/blink/blink.c>
3. Now, compile the code by typing: `% avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -o main.elf blink.c`. If there are any errors, fix them and re-type the line above.

4. Type: `% avr-objcopy -j .text -j .data -O ihex main.elf main.hex`. This will create a text file containing a hexadecimal (base 16) version of the compiled source code that is understandable by the MCU. In the next step, you will upload this hex file to the flash ROM of the MCU. This is called "flashing" the MCU.
5. Connect your Arduino Uno to your computer using the USB cable. Determine which USB port the Uno is connected to by typing: `% ls /dev/tty.*`. You'll see something like `/dev/tty.usbmodem1101`(the numbers may be different). Now flash the code to the ATmega328P mcu by typing `% avrdude -c Arduino -b 115200 -P /dev/tty.usbmodemXYZ -p atmega328p -U flash:w:main.hex:i`. In place of XYZ type the number of your USB port identified in the previous step. You should now see the internal LED on the Arduino board blinking on and off at 1 second each.
6. Now modify the code by changing MYDELAY to 100. Redo the commands `avr-gcc`, `avr-objcopy`, and `avrdude` and the LED should be blinking much faster. Note: you do not need to type in these commands. Position your cursar in the terminal prompt line and use the up and down arrows to scroll through previous commands.
7. You have undoubtedly noted that the three lines we're using to compile and flash are tedious to type. A makefile is a text file that contains a set of instructions for building code without needing to type in all the command line steps. Download the makefile from <https://github.com/ProfMcL/ECE231/blob/main/blink/makefile> and move that file to your blink project folder. (You should have version 2.0 of the makefile created by Prof. McLaughlin for this class). You will notice that this file is now listed along with `blink.c` and other files in the explorer panel of VS Code. Your laptop may have appended `.txt` or another extension to makefile during the download process. If so, right-click on `makefile.txt` and change its name to `makefile` without any extension.
8. Read through the makefile using VSCode and update the PORT variable to the correct USB port for your Arduino Uno. You can find this by typing `ls /dev/tty.*` from terminal.
9. Go back to `blink.c` and change MYDELAY to some other value, such as 3000, corresponding to 3 seconds. From the terminal command line, type `% make`. This will compile your code. Then type `% make flash`. This will flash your code to your device.
10. Unplug your USB cable from your laptop and plug it into a wall outlet via a USB adapter if you have one.

Congratulations on your first embedded coding project!

Document History

Revised on Version Author Description

2/13/23 1.0 D. McLaughlin Initial document creation

2/15/23 1.1 D.McLaughlin Corrected typo page 3 part 2.5.1

2/17/23 1.2 D. McLaughlin added instruction 6.6 about changing USB port in makefile

4/24/24 2.0 D. McLaughlin optimized for ECE-204 Spring 2024