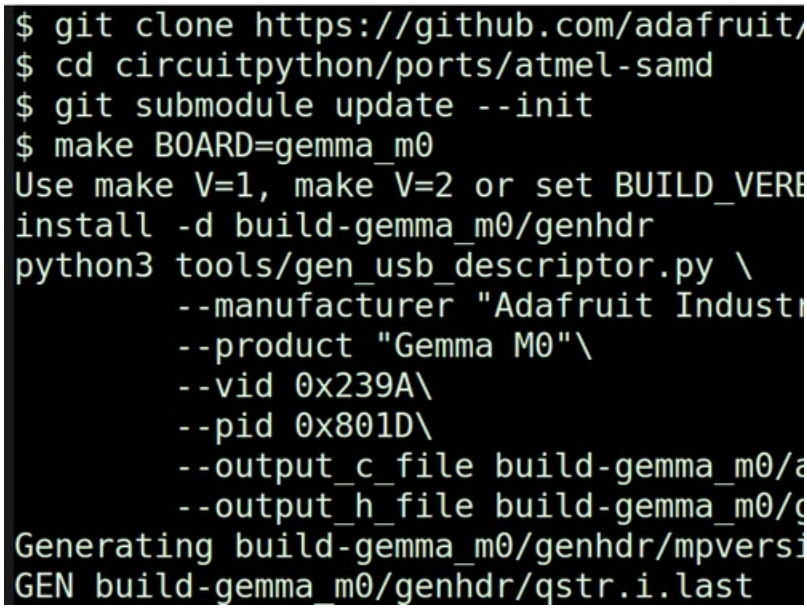
adafruit learning system

Building CircuitPython

Created by Dan Halbert



Last updated on 2020-08-03 11:22:34 PM EDT

Introduction

Adafruit's CircuitPython (https://adafru.it/AIP) is an open-source implementation of Python for microcontrollers. It's derived from (also known as, a "fork" of) MicroPython (https://adafru.it/f9W), a ground-breaking implementation of Python for microcontrollers and constrained environments.

CircuitPython ships on many Adafruit products. We regularly create new releases and make it easy to update your installation with new builds.

However, you might want to build your own version of CircuitPython. You might want to keep up with development versions between releases, adapt it to your own hardware, add or subtract features, or add "frozen" modules to save RAM space. This guide explains how to build CircuitPython yourself.

CircuitPython is meant to be built in a POSIX-style build environment. We'll talk about building it on Linux-style systems or on MacOS. It's possible, but tricky, to build in other environments such as CygWin or MinGW: we may cover how to use these in the future.

Linux Setup

Install a Real or Virtual Linux Machine

If you don't already have a Linux machine, you can set one up in several different ways. You can install a Linux distribution natively, either on its own machine or as a dual-boot system. You can install Linux on a virtual machine on, say, a Windows host machine. You can also use Windows Subsystem for Linux (https://adafru.it/C2z) (WSL), available on Microsoft Windows 10, which allows you to run a Linux distribution with an emulation layer substituting for the Linux kernel.

We recommend using the Ubuntu (https://adafru.it/C2A) distribution of Linux or one of its variants (Kubuntu, Mint, etc.). The instructions here assume you are using Ubuntu. The 18.04 LTS (Long Term Support) version is stable and reliable.

Native Linux

You can install Ubuntu on a bare machine easily. Follow the directions (https://adafru.it/C2B) (this link is for 16.04) on the Ubuntu website. You can also install Ubuntu on a disk shared with your Windows installation, or on a separate disk, and make a dual-boot installation.

Linux on a Virtual Machine

Linux can also be installed easily on a virtual machine. First you install the virtual machine software, and then create a new virtual machine, usually giving the VM software a .iso file of Ubuntu or another distribution. On Windows, VM Workstation Player (https://adafru.it/C2C) and VirtualBox (https://adafru.it/eiS) are both free and easily installed.

Vagrant

Vagrant is software that is designed to make it easy to set up and use pre-configured virtual machines. There are potential stumbling blocks. This guide will include Vagrant information in the future.

Install Build Tools on Ubuntu

The Ubuntu 18.04 LTS Desktop distribution includes most of what you need to build CircuitPython. First, in a terminal window, do:

```
sudo apt update
# Try running `make`. If it's not installed, do:
# sudo apt install build-essential
sudo apt install git
sudo apt install gettext
pip3 install huffman
```

Next you need to download and unpack the ARM gcc toolchain from its download page (https://adafru.it/HCO). ARM is no longer updating its Ubuntu "ppa" (private package archive), so you need to install these manually. For CircuitPython 5 development we are using the 9-2019-q4-major version (https://adafru.it/HCP). If you want to build CircuitPython 4, use the 7-2018q2-update (https://adafru.it/HCQ) version. These downloads are **.bz2** archive files.

```
# Put the unpacked toolchain into an appropriate directory.
# This is an example.
cd ~/bin
tar xvf <name of the .bz2 file you downloaded>
```

Next, add a line to your `.bash_profile` or other startup file to add the unpacked toolchain executables to your `PATH`. For example:

```
export PATH=/home/$USER/bin/gcc-arm-none-eabi-9-2019-q4-major/bin:$PATH
```

Open a new terminal window, and see if you now have the correct executables on your path:

```
$ which arm-none-eabi-gcc
/home/halbert/bin/gcc-arm-none-eabi-9-2019-q4-major/bin/arm-none-eabi-gcc
```

Now move to the Build CircuitPython (https://adafru.it/C2D) section of this guide.

MacOS Setup

To build CircuitPython on MacOS, you need to install `git`, `python3`, and the ARM toolchain. The easiest way to do this is to first install Homebrew (https://adafru.it/wPC), a software package manager for MacOS. Follow the directions on its webpages.

Now install the software you need:

```
brew update
brew install git python3 gettext
brew link gettext --force
pip3 install huffman
```

And finally, install the ARM toolchain (note the `cask`):

```
brew cask install gcc-arm-embedded
```

Once the dependencies are installed, we'll also need to make a disk image to clone CircuitPython into. By default the Mac OSX filesystem is case insensitive. In a few cases, this can confuse the build process. So, we recommend creating a case sensitive disk image with Disk Utility (https://adafru.it/CaT) to store the source code. The disk image is a single file that can be mounted by double clicking it in the Finder. Once it's mounted, it works like a normal folder located under the `/Volumes` directory.

That's it! Now you can move on and actually Build CircuitPython (https://adafru.it/C2D).

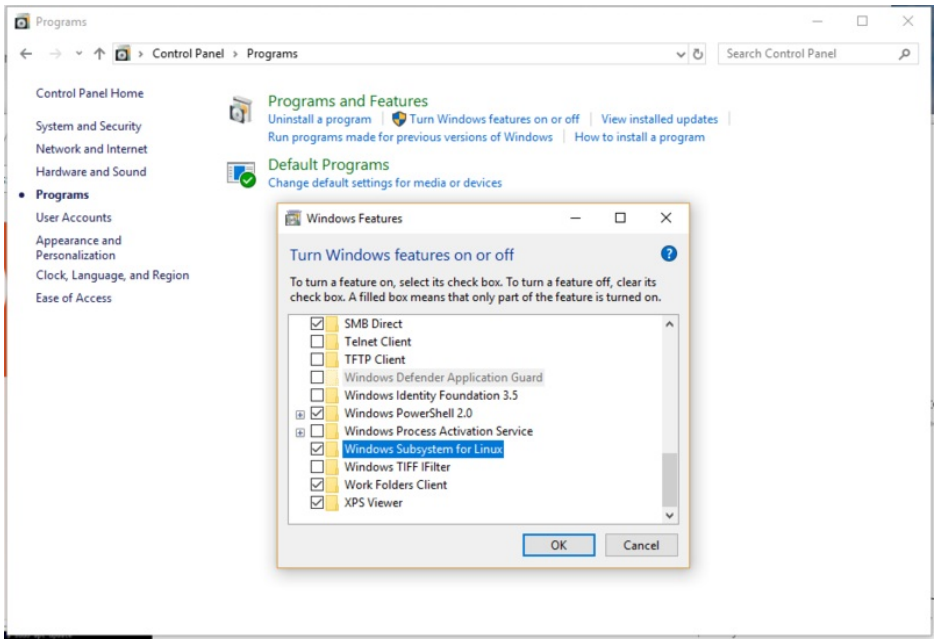
Windows Subsystem for Linux Setup

Windows Subsystem for Linux (WSL) is a new feature of Windows 10 that lets you run Ubuntu and other versions of Linux right in Windows. It's real Ubuntu, without the Linux kernel, but with all the software packages that don't need a graphical interface.

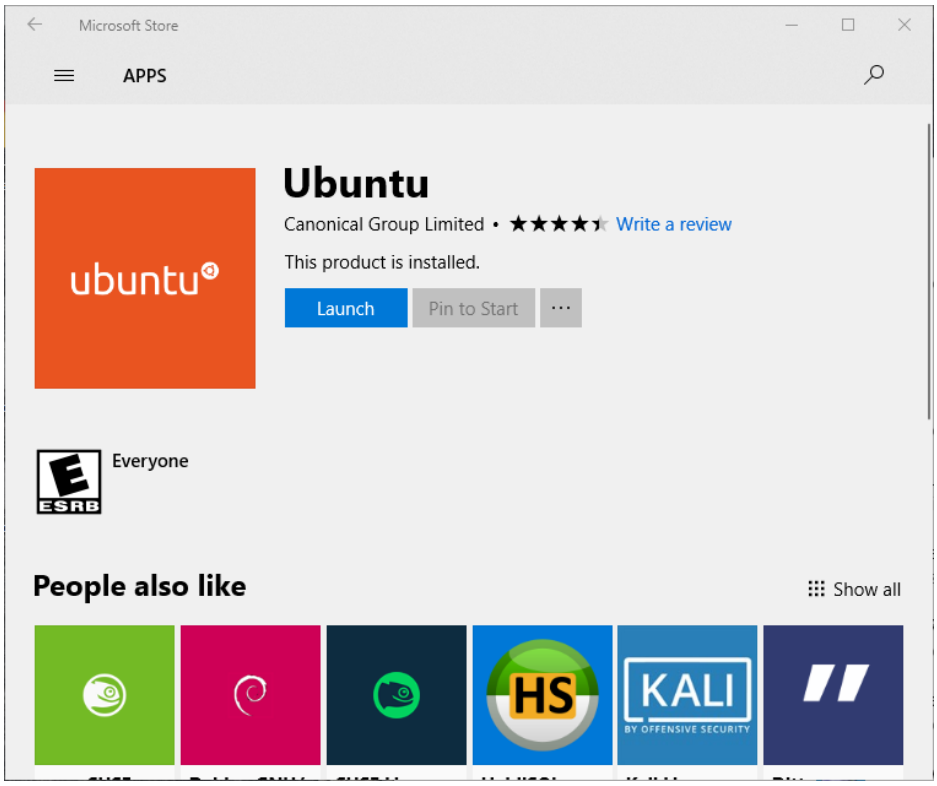
You can build CircuitPython in WSL easily. It's easier to install than a Linux virtual machine. The filesystem access is slower than on regular Linux, so builds will take somewhat longer, but it's quite usable.

Install WSL

To install WSL, you need to be running a recent version of 64-bit Windows 10. Go to Control Panel > Programs >Windows Features, and check the box for Windows Subsystem for Linux.



After that, you'll have to reboot Windows. After it's rebooted, go to the Microsoft Store, search for Ubuntu, and install it.



Then launch Ubuntu. You'll see a terminal console window and you'll be asked some initial setup questions.

Install Build Tools

This step is the same as for regular 18.04 Ubuntu, so refer to that page (https://adafru.it/HCR), and then come back here.

Build CircuitPython

From this point on, you can build CircuitPython just as it's built in regular Ubuntu, described in the Build CircuitPython (https://adafru.it/C2D) section of this guide.

Moving Files to Windows

You can copy files to and from Windows through the /mnt/c. For instance, if you want to copy a CircuitPython build to the desktop, do:

```
cp build-circuitplayground_express/firmware.uf2 /mnt/c/Users/YourName/Desktop
```

Warning: Don't build in a shared folder (in /mnt/c). You'll probably have filename and line-ending problems.

You might be tempted to clone and build CircuitPython in a folder shared with the Windows filesystem (in `/mnt/c` somewhere). That can cause problems, especially if you use `git` commands on the Windows side in that folder. The CircuitPython build assumes case-sensitive filenames, but Windows usually ignores filename case differences. You may also have line-ending problems (CRLF vs. LF). It's better to clone and build inside your home directory in WSL, and copy files over to a shared folder as needed.

Mounting a CircuitPython Board in WSL

You can mount your `...BOOT` or `CIRCUITPY` drive in WSL. Create a mount point and then mount it. Note that you'll have to remount each time the drive goes away, such as when you restart the board or switch between the `BOOT` drive and `CIRCUITPY`. So it's probably more convenient to copy files to the board from Windows instead of WSL.

```
# You only need to do this once.
# Choose the appropriate drive letter.
sudo mkdir /mnt/d
# Now mount the drive.
sudo mount -t drvfs D: /mnt/d
# Now you can look at the contents, copy things, etc.
ls /mnt/d
cp firmware.bin /mnt/d
# etc.
```

Manual Setup

If the setup instructions above don't work for your particular OS setup, for whatever reason, you can get the ball rolling by installing these tools in whatever way you can and then getting them to work with the `Makefile` in `circuitpython/atmel-samd` (2.x branch) or `circuitpython/ports/atmel-samd`. (master branch):

- git
- make
- python2
- python3
- gettext
- ARM gcc toolchain: https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads (https://adafru.it/C2E)

Fetch the Code to Build

Once your build tools are installed, fetch the CircuitPython source code from its GitHub repository ("repo") and also fetch the git "submodules" it needs. The submodules are extra code that you need that's stored in other repos.

In the commands below, you're cloning from Adafruit's CircuitPython repo. But if you want to make changes, you might want to "fork" that repo on GitHub to make a copy for yourself, and clone from there.

```
git clone https://github.com/adafruit/circuitpython.git
cd circuitpython
git submodule sync --quiet --recursive
git submodule update --init
```

Build `mpy-cross`

Build the `mpy-cross` compiler first, which compiles Circuitpython `.py` files into `.mpy` files. It's needed to include library code in certain boards.

```
make -C mpy-cross
```

Build CircuitPython

Now you're all set to build CircuitPython. If you're in the master branch of the repo, you'll be building the latest version. Choose which board you want to build for. The boards available are all the subdirectories in `ports/atmel-samd/boards/` .

```
cd ports/atmel-samd
make BOARD=circuitplayground_express
```

In 4.x we've introduced translated versions of CircuitPython. By default the en_US version will be built. To build for a different language supply a TRANSLATION argument.

```
cd ports/atmel-samd
make BOARD=circuitplayground_express TRANSLATION=es
```

Run your build!

When you've successfully built, you'll see output like:

```
Create build-circuitplayground_express/firmware.bin
Create build-circuitplayground_express/firmware.uf2
python2 ../../tools/uf2/utils/uf2conv.py -b 0x2000 -c -o build-circuitplayground_express/firmware.uf2
build-circuitplayground_express/firmware.bin
Converting to uf2, output size: 485888, start address: 0x2000
Wrote 485888 bytes to build-circuitplayground_express/firmware.uf2.
```

Copy `firmware.uf2` to your board the same way you'd update CircuitPython: Double-click to get the `BOOT` drive, and then just copy the `.uf2` file:

```
# Double-click the reset button, then:
cp build-circuitplayground_express/firmware.uf2 /media/yourname/CPLAYBOOT
```

The board will restart, and your build will start running.

If you're using a board without a UF2 bootloader, you'll need to use `bossac` and the `firmware.bin` file, not the `.uf2` file. Detailed instructions are here (https://adafru.it/Bid).

When to `make clean`

After you make changes to code, normally just doing `make BOARD=...` will be sufficient. The changed files will be recompiled and CircuitPython will be rebuilt.

However, there are some circumstance where you must do:

```
make clean BOARD=...
```

If you have changed the `#include` file structure in certain ways, or if you have defined QSTR's (a way of defining constants strings in the CircuitPython source), then you must `make clean` before rebuilding. If you're not sure, it's always safe to `make clean` and then `make`. It might take a little longer to build, but you'll be sure it was rebuilt properly.

Updating Your Repo

When there are changes in the GitHub repo, you might want to fetch those and then rebuild. Just "pull" the new code (assuming you haven't made changes yourself), update the submodules if necessary, and rebuild:

```
git pull
git submodule sync
git submodule update --init
# Then make again.
```

Those are the basics. There's a lot more to know about how to keep your forked repo up to date, merge "upstream" (Adafruit's) changes into your code, etc. We'll be writing all that up later.

Adding Frozen Modules

Normally, all imported Python modules in CircuitPython are loaded into RAM in compiled form, whether they start as `.mpy` or `.py` files. Especially on M0 boards, a user program can run out of RAM if too much code needs to be loaded.

To ameliorate this problem, a CircuitPython image can include compiled Python code that is stored in the image, in flash memory, and executed directly from there. These are "internal frozen modules". The `circuitplayground_express` and `esp8266` builds use this technique.

If you would like to build a custom image that includes some frozen modules, you can imitate how it's done in the `circuitplayground_express` build. Look at `boards/circuit_playground_express/mpconfigboard.mk`:

```
USB_VID = 0x239A
USB_PID = 0x8019
USB_PRODUCT = "CircuitPlayground Express"
USB_MANUFACTURER = "Adafruit Industries LLC"

CHIP_VARIANT = SAMD21G18A
CHIP_FAMILY = samd21

SPI_FLASH_FILESYSTEM = 1
EXTERNAL_FLASH_DEVICE_COUNT = 2
EXTERNAL_FLASH_DEVICES = "S25FL216K, GD25Q16C"
LONGINT_IMPL = MPZ

# Make room for frozen libs.
CIRCUITPY_DISPLAYIO = 0
CIRCUITPY_FREQUENCYIO = 0
CIRCUITPY_I2CPERIPHERAL = 0

SUPEROPT_GC = 0
CFLAGS_INLINE_LIMIT = 55

# Include these Python libraries in firmware.
FROZEN_MPY_DIRS += $(TOP)/frozen/Adafruit_CircuitPython_BusDevice
FROZEN_MPY_DIRS += $(TOP)/frozen/Adafruit_CircuitPython_CircuitPlayground
FROZEN_MPY_DIRS += $(TOP)/frozen/Adafruit_CircuitPython_HID
FROZEN_MPY_DIRS += $(TOP)/frozen/Adafruit_CircuitPython_LIS3DH
FROZEN_MPY_DIRS += $(TOP)/frozen/Adafruit_CircuitPython_NeoPixel
FROZEN_MPY_DIRS += $(TOP)/frozen/Adafruit_CircuitPython_Thermistor
```

Notice the `FROZEN_MPY_DIRS` lines in the file. Pick the `mpconfigboard.mk` file for the board you are using, and add one or more similar lines. You will need to do add directories for the libraries you want to include. If these are existing libraries in GitHub, you can add them as submodules. For instance, suppose you want to add the `Adafruit_CircuitPython_HID` library to the `feather_m0_express` build. Add this line to `boards/feather_m0_express/mpconfigboard.mk`:

```
FROZEN_MPY_DIRS += $(TOP)/frozen/Adafruit_CircuitPython_HID
```

Then add the library as a submodule:

```
cd circuitpython/frozen
git submodule add https://github.com/adafruit/Adafruit_CircuitPython_HID
```

When you add the submodule it will be cloned into the `frozen/` directory.

Note that there is limited unused space available in the images, especially in the non-Express M0 builds, and you may not be able to fit all the libraries you want to freeze. You can of course try to simplify the library code if necessary to make it fit.

We are no longer supporting CircuitPython on the ESP8266 as of 4.0.0alpha. You can continue to use 3.x but it will no longer be updated.

ESP8266 Build

The esp8266 board build requires a different toolchain. You must install https://github.com/pfalcon/esp-open-sdk (https://adafru.it/f9Y). Follow the directions in the README (https://adafru.it/C2F). Note the caveats for MacOS (https://adafru.it/f9Y) about case-sensitive filesystems.