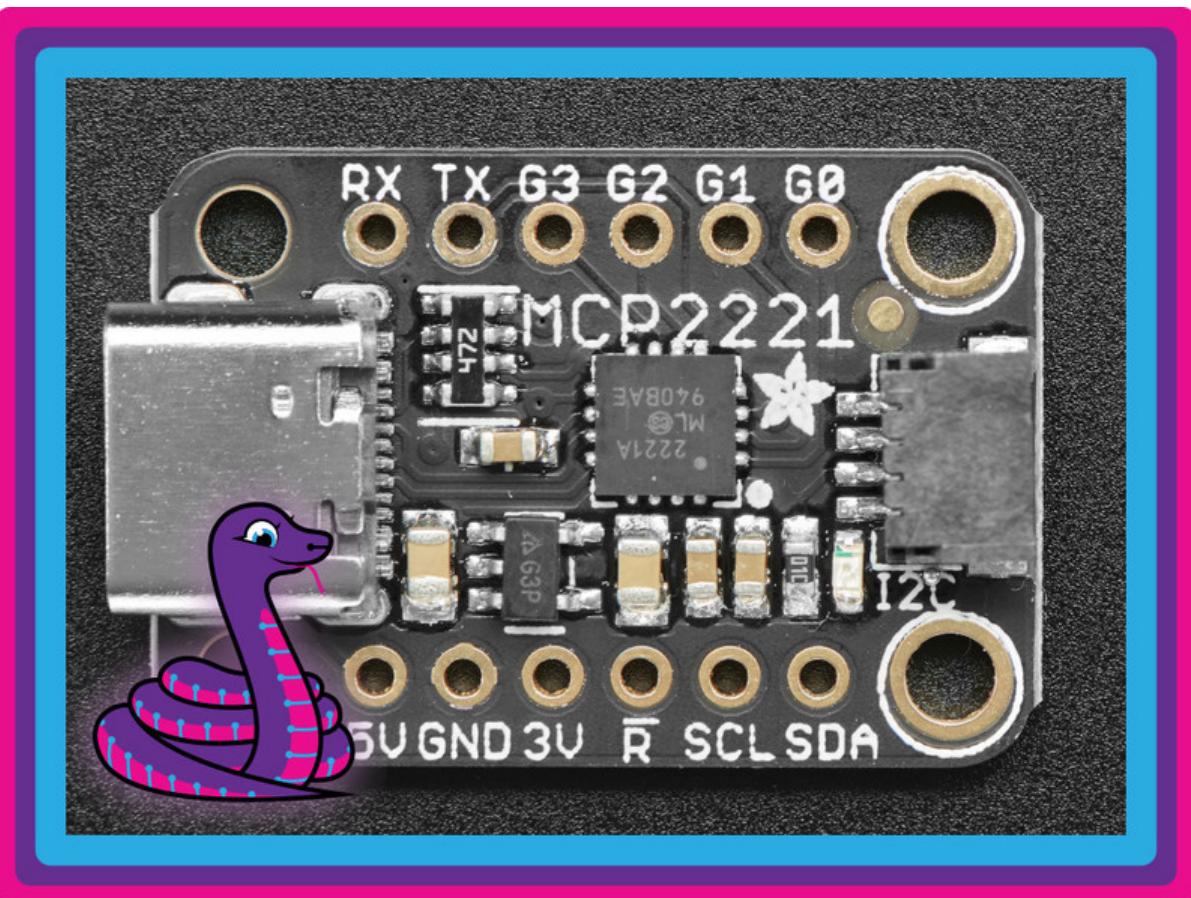




CircuitPython Libraries on any Computer with MCP2221

Created by Carter Nelson



<https://learn.adafruit.com/circuitpython-libraries-on-any-computer-with-mcp2221>

Last updated on 2024-03-08 03:15:10 PM EST

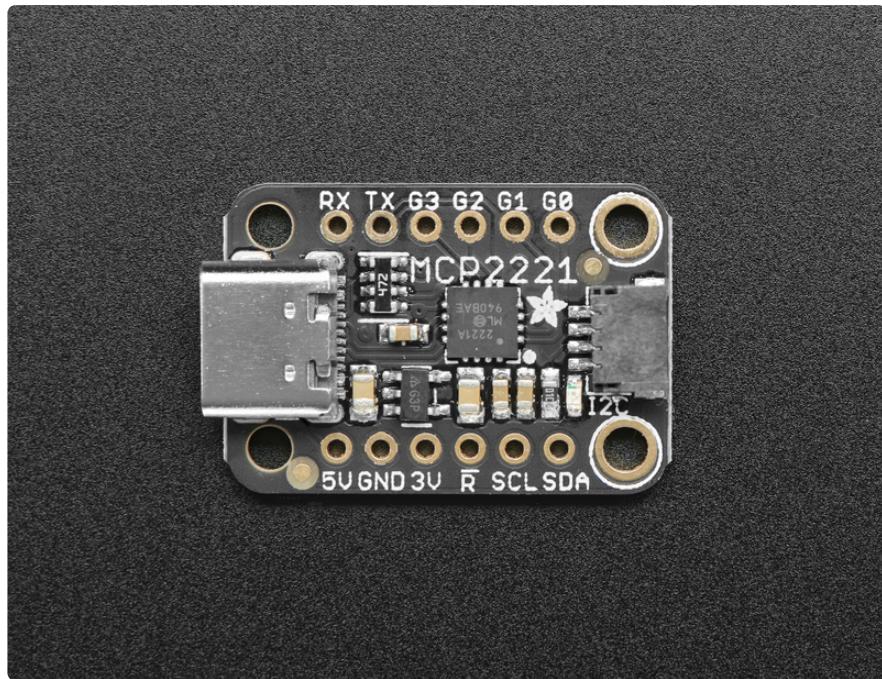
Table of Contents

Overview	5
• CircuitPython and CircuitPython Libraries	
• CircuitPython Libraries on Personal Computers	
Running CircuitPython Code without CircuitPython	7
• Adafruit Blinka: a CircuitPython Compatibility Library	
• Raspberry Pi and Other Single-Board Linux Computers	
• Desktop Computers	
• MicroPython	
• Installing Blinka	
• Installing CircuitPython Libraries	
• Linux Single-Board Computers	
• Desktop Computers using a USB Adapter	
• MicroPython	
Setup	10
• Additional Information	
Windows	10
• Have Python 3 Installed	
• Install hidapi	
• Install Blinka	
• Set Environment Variable	
• Check Platform was detected	
Mac OSX	13
• Python 3 Check	
• Install hidapi	
• Install Blinka	
• Set Environment Variable	
• Check that Platform was detected	
Linux	15
• Install libusb and libudev	
• Setup udev rules	
• Install hidapi	
• Remove Native MCP2221 Driver	
• Install Blinka	
• Set environment variable	
• Run the sanity check.	
Post Install Checks	18
• Check that hidapi is installed correctly	
• Check that MCP2221 can be found	
• Check environment variable within Python	
Pinout	20
• Power Pins	
• GPIO Pins	
• I2C Pins	
• UART Pins	

- ADC Pins
- DAC Pins
- Logic Level

Examples	22
<hr/>	
GPIO	23
<hr/>	
Digital Output	
Digital Input	
Digital Input and Output	
<hr/>	
I2C	26
<hr/>	
Install MSA301 Library	
Example Code	
<hr/>	
ADC	29
<hr/>	
DAC	30
<hr/>	
UART	31
<hr/>	
Install pySerial	
Find COM port in Linux	
Find COM port in Windows	
UART Example with GPS	
<hr/>	
FAQ & Troubleshooting	35
<hr/>	
Update Blinka/Platform Libraries	
<hr/>	
Downloads	43
<hr/>	
Schematic and Fab Print	

Overview



This guide will show you how to use an MCP2221(A) to connect to I2C sensors and breakouts from your desktop PC running Windows, Mac OSX, or Linux. The MCP2221 also allows for general purpose digital input and output (GPIO) for things like buttons and LEDs, analog to digital conversion (ADC), and digital to analog (DAC).

The cool part about this is that you can then use any of the CircuitPython Libraries that have been written for the numerous I2C sensors and breakouts. You can bring that data directly into your PC for any kind of powerful analysis or presentation.

Our breakout uses the MCP2221A chip, but we may refer to it as MCP2221 as the difference are not relevant to using Blinka/CircuitPython libraries and we can use either version of the chip!

CircuitPython and CircuitPython Libraries

As you are going through this guide, keep in mind the difference between CircuitPython and CircuitPython Libraries:

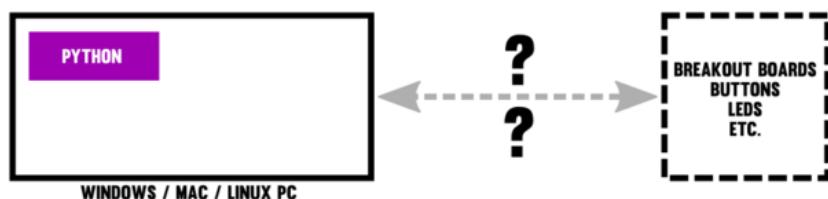
- **CircuitPython** - a microprocessor specific implementation of Python written in C. Here's the [source code](https://adafru.it/tB7) (<https://adafru.it/tB7>). And here's the [Welcome](https://adafru.it/cpy>Welcome main Welcome (Welcome)).

- **CircuitPython Libraries** - sensor and breakout specific code written in Python using the CircuitPython hardware API. There are a lot of these - [check out the bundle \(<https://adafru.it/zdx>\)](https://adafru.it/zdx).

There are various hardware combinations that allow for running CircuitPython and CircuitPython Libraries. In this guide we will **not** be using the actual CircuitPython firmware. But we will be using CircuitPython Libraries.

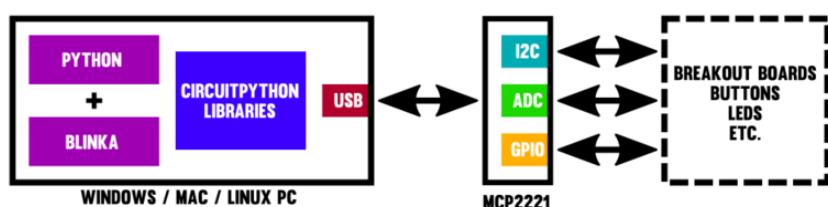
CircuitPython Libraries on Personal Computers

This is essentially the same idea as discussed in the [FT232H Guide \(<https://adafru.it/GNe>\)](https://adafru.it/GNe). How can we directly connect common hardware items like buttons and I2C breakouts to a PC?



The MCP2221 provides another way to do this by utilizing the USB bus. The MCP2221 just makes different trade offs relative to the FT232H. The biggest being **no hardware SPI support**. But you do gain ADC and DAC support. Also, it's much cheaper than the FT232H.

So you end up with something like this:



Great! Let's get everything setup so we can actually do some fun stuff.



Adafruit MCP2221A Breakout - General Purpose USB to GPIO ADC I2C

Wouldn't it be cool to drive a tiny OLED display, read a

<https://www.adafruit.com/product/4471>

Running CircuitPython Code without CircuitPython

There are two parts to the CircuitPython ecosystem:

- **CircuitPython firmware**, written in C and built to run on various microcontroller boards (not PCs). The firmware includes the CircuitPython interpreter, which reads and executes CircuitPython programs, and chip-specific code that controls the hardware peripherals on the microcontroller, including things like USB, I2C, SPI, GPIO pins, and all the rest of the hardware features the chip provides.
- **CircuitPython libraries**, written in Python to use the native (built into the firmware) modules provided by CircuitPython to control the microcontroller peripherals and interact with various breakout boards.

But suppose you'd like to use CircuitPython **libraries** on a board or computer that does not have a native CircuitPython **firmware** build. For example, on a PC running Windows or macOS. Can that be done? The answer is yes, via a separate piece of software called **Blinka**. Details about Blinka follow, however it is important to realize that the **CircuitPython firmware is never used**.

CircuitPython firmware is NOT used when using Blinka.

Adafruit Blinka: a CircuitPython Compatibility Library

Enter **Adafruit Blinka**. Blinka is a software library that emulates the parts of CircuitPython that control hardware. Blinka provides non-CircuitPython implementations for `board`, `busio`, `digitalio`, and other native CircuitPython

modules. You can then write Python code that looks like CircuitPython and uses CircuitPython libraries, without having CircuitPython underneath.

There are multiple ways to use Blinka:

- Linux based Single Board Computers, for example a Raspberry Pi
- Desktop Computers + specialized USB adapters
- Boards running MicroPython

More details on these options follow.

Raspberry Pi and Other Single-Board Linux Computers

On a Raspberry Pi or other single-board Linux computer, you can use Blinka with the regular version of Python supplied with the Linux distribution. Blinka can control the hardware pins these boards provide.

Desktop Computers

On Windows, macOS, or Linux desktop or laptop ("host") computers, you can use special USB adapter boards that provide hardware pins you can control. These boards include [MCP221A](https://adafruit.it/lfV) (<https://adafruit.it/lfV>) and [FT232H](https://adafruit.it/xia) (<https://adafruit.it/xia>) breakout boards, and [Raspberry Pi Pico boards running the u2if software](https://adafruit.it/Sje) (<https://adafruit.it/Sje>). These boards connect via regular USB to your host computer, and let you do GPIO, I2C, SPI, and other hardware operations.

MicroPython

You can also use Blinka with MicroPython, on [MicroPython-supported boards](https://adafruit.it/SBi) (<https://adafruit.it/SBi>). Blinka will allow you to import and use CircuitPython libraries in your MicroPython program, so you don't have to rewrite libraries into native MicroPython code. Fun fact - this is actually the original use case for Blinka.

Installing Blinka

Installing Blinka on your particular platform is covered elsewhere in this guide. The process is different for each platform. Follow the guide section specific to your

platform and make sure Blinka is properly installed before attempting to install any libraries.

Be sure to install Blinka before proceeding.

Installing CircuitPython Libraries

Once Blinka is installed the next step is to install the CircuitPython libraries of interest. How this is done is different for each platform. Here are the details.

Linux Single-Board Computers

On Linux single-board computers, such as Raspberry Pi, you'll use the Python `pip3` program (sometimes named just `pip`) to install a library. The library will be downloaded from pypi.org (<https://adafru.it/19ff>) automatically by `pip3`.

How to install a particular library using `pip3` is covered in the guide page for that library. For example, [here is the `pip3` installation information](https://adafru.it/OkF) (<https://adafru.it/OkF>) for the library for the LIS3DH accelerometer.

The library name you give to `pip3` is usually of the form `adafruit-circuitpython-libraryname`. This is not the name you use with `import`. For example, the LIS3DH sensor library is known by several names:

- The GitHub library repository is [Adafruit_CircuitPython_LIS3DH](https://adafru.it/uBs) (<https://adafru.it/uBs>).
- When you import the library, you write `import adafruit_lis3dh`.
- The name you use with `pip3` is `adafruit-circuitpython-lis3dh`. This is the name used on pypi.org (<https://adafru.it/19ff>).

Libraries often depend on other libraries. When you install a library with `pip3`, it will automatically install other needed libraries.

Desktop Computers using a USB Adapter

When you use a desktop computer with a USB adapter, like the MCP2221A, FT232H, or u2if firmware on an RP2040, you will also use pip3. However, **do not install the library with `sudo pip3`**, as mentioned in some guides. Instead, just install with `pip3`.

MicroPython

For MicroPython, you will not use `pip3`. Instead you can get the library from the CircuitPython bundles. See [this guide page](https://adafru.it/ABU) (<https://adafru.it/ABU>) for more information about the bundles, and also see the [Libraries page on circuitPython.org](https://adafru.it/ENC) (<https://adafru.it/ENC>).

Setup

The support for the MCP2221 in Blinka utilizes the [hidapi library](https://adafru.it/HEI) (<https://adafru.it/HEI>). This in turn relies on a few other things which vary for different OS's. So before we can actually use the MCP2221, we need to get everything setup. See the OS specific sections for what we went through to get things working for each.

Additional Information

Just for reference, here's the README from the hidapi source code repo, which has some install information:

hidapi README

<https://adafru.it/HEJ>

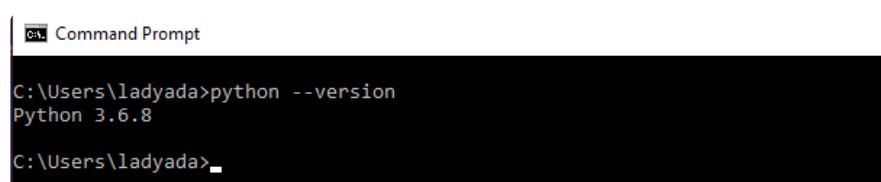
But first try the install instructions on the pages that follow for your OS.

Windows

Have Python 3 Installed

We assume you already have Python 3 installed on your computer. **Note we do not support Python 2** - it's deprecated and no longer supported!

At your command line prompt of choice, check your Python version with `python --version`



```
C:\ Command Prompt
C:\Users\ladyada>python --version
Python 3.6.8
C:\Users\ladyada>
```

Install hidapi

From the command line, manually install **hidapi** with

```
pip3 install hidapi
```

```
C:\Users\ladyada>pip install hidapi
Collecting hidapi
  Using cached https://files.pythonhosted.org/packages/57/81/b8a5b3719ceff5fcc5e
e9029bc7eecd64c13f933e2562e583860d8e64e6a/hidapi-0.7.99.post21-cp36-cp36m-win_am
d64.whl
Requirement already satisfied: setuptools>=19.0 in c:\python36\lib\site-packages
 (from hidapi) (42.0.1)
Installing collected packages: hidapi
Successfully installed hidapi-0.7.99.post21
```

If the install fails with text that ends with something like:

`distutils.errors.DistutilsError: Setup script exited with error: Microsoft Visual C++ 14.0 is required. Get it with "Microsoft Visual C++ Build Tools": https://visualstudio.microsoft.com/downloads/ (https://adafru.it/JBp)`

then you will need to also install the Microsoft Visual C++ Build Tools. Thanks to **@jklem** for pointing this out [in the forums](#) (<https://adafru.it/doW>).

Download it from here (same link as in text):

Microsoft Visual Studio Downloads

<https://adafru.it/JBp>

NOTE: You do not need the full Visual Studio IDE. Just the Build Tools.

All downloads

The screenshot shows the 'All downloads' section of the Microsoft Visual Studio Downloads page. Under the 'Tools for Visual Studio 2019' heading, there is a list of tools: Remote Tools for Visual Studio 2019, Performance Tools for Visual Studio 2019, IntelliTrace Standalone Collector for Visual Studio 2019, Agents for Visual Studio 2019, and Build Tools for Visual Studio 2019. The 'Build Tools for Visual Studio 2019' item is highlighted with a red box. An arrow points to the 'Download' button for the x64 version of the tool.

Scroll down to where it says **Tools for Visual Studio 2019**.
Expand the list to show the sub options.

Click the Download button for **Build Tools for Visual Studio 2019**.

This downloads a .exe file with a name like **vs_BuildTools.exe**. Run that to install the build tools and then try the pip install again.

Install Blinka

To install Blinka and its dependencies, run:

```
pip3 install adafruit-blinka
```

```
C:\Users\ladyada>pip install adafruit-blinka
Collecting adafruit-blinka
  Requirement already satisfied: Adafruit-PureIO in c:\python36\lib\site-packages (0.5.1)
  Requirement already satisfied: Adafruit-PlatformDetect in c:\python36\lib\site-packages (0.5.1)
Installing collected packages: adafruit-blinka
Successfully installed adafruit-blinka-2.5.2
```

Set Environment Variable

You must do this every time before running circuitpython code, you can set it permanently in windows if you like, for now just type into the same cmd window you're using with Python

```
set BLINKA_MCP2221=1
```

```
C:\Users\ladyada>set BLINKA_MCP2221=1
C:\Users\ladyada>
```

If you are using Windows Powershell, the syntax is a little different. In that case do:

```
$env:BLINKA_MCP2221=1
```

NOTE - by default, the terminal window in VSCode uses Powershell.

Check Platform was detected

In the same command window you `set BLINKA_MCP2221=1` env var, run `python` and run

```
import board
dir(board)
```

at the Python REPL. If you get no errors, and you see a list of all the pins available - you're good to go!

```
C:\Users\ladyada>set BLINKA_MCP2221=1
C:\Users\ladyada>python
Python 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import board
>>> dir(board)
['G0', 'G1', 'G2', 'G3', 'I2C', 'SCL', 'SDA', 'SPI', '__builtins__', '__cached__',
 '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
 'ap_board', 'board_id', 'detector', 'pin', 'sys']
>>>
```

Mac OSX

Python 3 Check

We assume you already have Python 3 installed on your computer. **Note we do not support Python 2** - it's deprecated and no longer supported!

At your command line prompt of choice, check your Python version with `python3 --version`

```
1. fish /Users/brentrubell (fish)
~> python --version
Python 3.6.0
```

Install hidapi

From the command line, manually install [hidapi](https://adafruit.io/HIA) (<https://adafruit.io/HIA>) with:

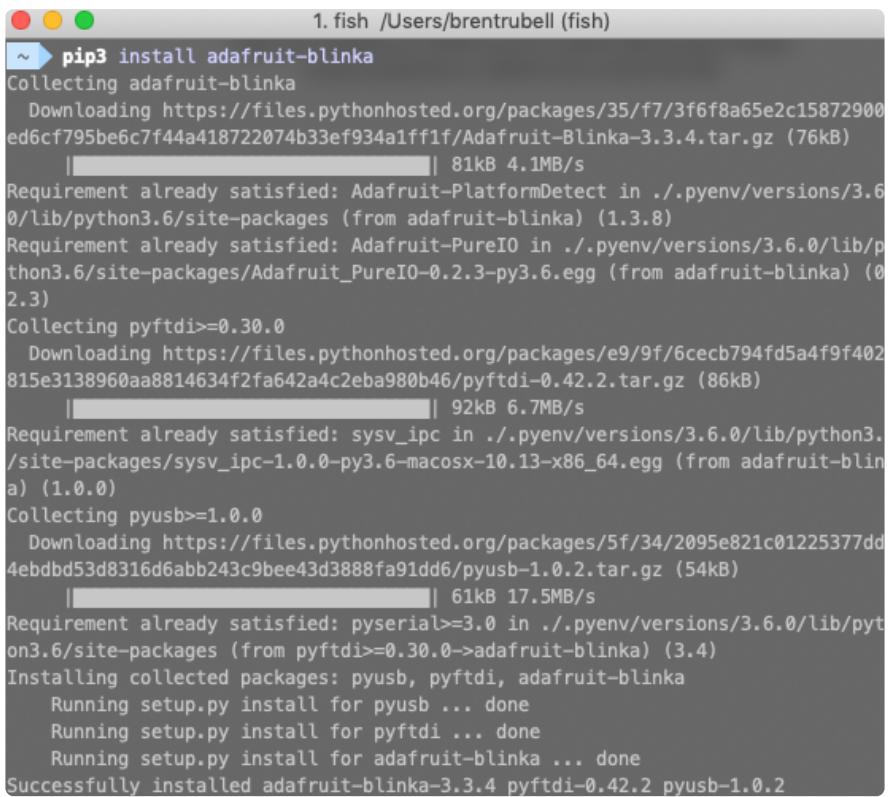
```
pip3 install hidapi
```

```
1. fish /Users/brentrubell (fish)
~> pip3 install hidapi
Collecting hidapi
  Using cached https://files.pythonhosted.org/packages/c1/86/89df0e8890f96eeb5fb
68d4ccb14cb38e2c2d2cf7601ba972206acd9015/hidapi-0.7.99.post21.tar.gz
Requirement already satisfied: setuptools>=19.0 in ./local/lib/python3.6/site-p
ackages (from hidapi) (40.0.0)
Installing collected packages: hidapi
  Running setup.py install for hidapi ... done
Successfully installed hidapi-0.7.99.post21
```

Install Blinka

To install [Blinka](https://adafruit.io/BJX) (<https://adafruit.io/BJX>) and its dependencies, run:

```
pip3 install adafruit-blinka
```



```
1. fish /Users/brentrubell (fish)
~ ➔ pip3 install adafruit-blinka
Collecting adafruit-blinka
  Downloading https://files.pythonhosted.org/packages/35/f7/3f6f8a65e2c15872900ed6cf795be6c7f44a418722074b33ef934a1ff1f/Adafruit-Blinka-3.3.4.tar.gz (76kB)
    |██████████| 81kB 4.1MB/s
Requirement already satisfied: Adafruit-PlatformDetect in ./pyenv/versions/3.6.0/lib/python3.6/site-packages (from adafruit-blinka) (1.3.8)
Requirement already satisfied: Adafruit-PureIO in ./pyenv/versions/3.6.0/lib/python3.6/site-packages/Adafruit_PureIO-0.2.3-py3.6.egg (from adafruit-blinka) (0.2.3)
Collecting pyftdi>=0.30.0
  Downloading https://files.pythonhosted.org/packages/e9/9f/6cecb794fd5a4f9f402815e3138960aa8814634f2fa642a4c2eba980b46/pyftdi-0.42.2.tar.gz (86kB)
    |██████████| 92kB 6.7MB/s
Requirement already satisfied: sysv_ipc in ./pyenv/versions/3.6.0/lib/python3.6/site-packages/sysv_ipc-1.0.0-py3.6-macosx-10.13-x86_64.egg (from adafruit-blinka) (1.0.0)
Collecting pyusb>=1.0.0
  Downloading https://files.pythonhosted.org/packages/5f/34/2095e821c01225377dd4ebdbd53d8316d6abb243c9bee43d3888fa91dd6/pyusb-1.0.2.tar.gz (54kB)
    |██████████| 61kB 17.5MB/s
Requirement already satisfied: pyserial>=3.0 in ./pyenv/versions/3.6.0/lib/python3.6/site-packages (from pyftdi>=0.30.0->adafruit-blinka) (3.4)
Installing collected packages: pyusb, pyftdi, adafruit-blinka
  Running setup.py install for pyusb ... done
  Running setup.py install for pyftdi ... done
  Running setup.py install for adafruit-blinka ... done
Successfully installed adafruit-blinka-3.3.4 pyftdi-0.42.2 pyusb-1.0.2
```

Set Environment Variable

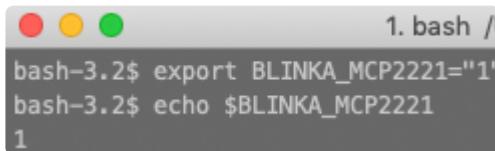
You'll need to set this variable every time before running CircuitPython code. To do this, we set the environment variable to `BLINKA_MCP2221`.

You can set the variable by running:

```
export BLINKA_MCP2221="1"
```

Then, verify that the variable is set by running:

```
echo $BLINKA_MCP2221
```



```
1. bash /l
bash-3.2$ export BLINKA_MCP2221="1"
bash-3.2$ echo $BLINKA_MCP2221
1
```

Don't forget this step. Things won't work unless `BLINKA_MCP2221` is set.

Check that Platform was detected

In the same terminal window you ran `export BLINKA_MCP2221="1"`, run `python3`.

At the REPL, run:

```
import board
```

```
dir(board)
```

```
bash-3.2$ python
Python 3.6.0 (default, Feb 12 2019, 09:59:48)
[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import board
>>> dir(board)
['G0', 'G1', 'G2', 'G3', 'I2C', 'SCL', 'SDA', 'SPI', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'ap_board', 'board_id', 'detector', 'pin', 'sys']
```

If you get no errors and see a list of all the pins available, you're good to go!

Linux

The following shows a typical run through installing and setting things up on Linux.

Install libusb and libudev

Run the following to install required libraries:

```
sudo apt-get install libusb-1.0 libudev-dev
```

and answer **Y** to the prompt. This should install libusb and libudev.

Setup udev rules

Use a text editor to create and edit the file `/etc/udev/rules.d/99-mcp2221.rules` and add the following contents.

```
SUBSYSTEM=="usb", ATTRS{idVendor}=="04d8", ATTR{idProduct}=="00dd", MODE="0666"
```

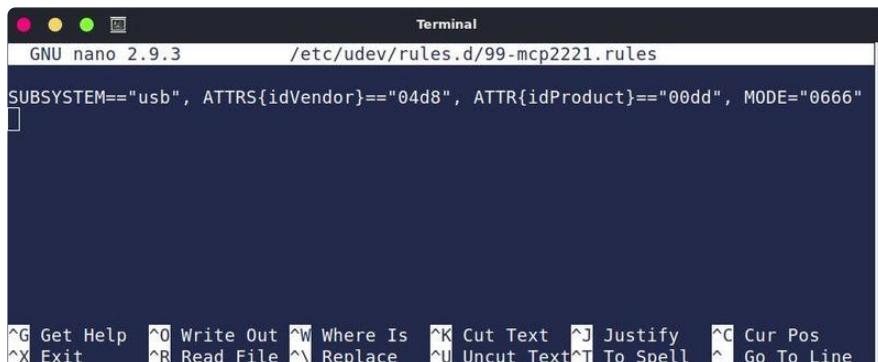
Here we use nano, so run:

like this:



```
Terminal
user:$ sudo nano /etc/udev/rules.d/99-mcp2221.rules
```

and add the contents from above:



```
Terminal
GNU nano 2.9.3          /etc/udev/rules.d/99-mcp2221.rules
SUBSYSTEM=="usb", ATTRS{idVendor}=="04d8", ATTR{idProduct}=="00dd", MODE="0666"

```

and then press **CTRL-X** and **Y** to save and exit.



```
Terminal
GNU nano 2.9.3          /etc/udev/rules.d/99-mcp2221.rules          Modified
SUBSYSTEM=="usb", ATTRS{idVendor}=="04d8", ATTR{idProduct}=="00dd", MODE="0666"

Save modified buffer? (Answering "No" will DISCARD changes.)
Y Yes
N No           ^C Cancel
```

The settings will take effect the next time you plug in the MCP2221.

Install hidapi

To install hidapi, run:

```
pip3 install hidapi
```



```
pi@raspberrypi: ~
user:$ pip3 install hidapi
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting hidapi
  Downloading https://www.piwheels.org/simple/hidapi/hidapi-0.7.99.post21-cp37-cp37m-linux_armv7l.whl (273kB)
    100% |██████████| 276kB 235kB/s
Requirement already satisfied: setuptools>=19.0 in /usr/lib/python3/dist-packages (from hidapi) (40.8.0)
Installing collected packages: hidapi
Successfully installed hidapi-0.7.99.post21
user:$
```

Remove Native MCP2221 Driver

Starting with **Linux Kernel 5.7**, a native MCP2221 driver (`hid_mcp2221`) is included with the installation. This will interfere with the HID generic driver (`hid_generic`) used by Blinka. The native driver can be **temporarily** removed (disabled) with:

```
sudo rmmod hid_mcp2221
```

To prevent the driver from loading again at boot or when reinserting the MCP2221, update the kernel module blacklist configuration. Add this line to `/etc/modprobe.d/blacklist.conf`

```
blacklist hid_mcp2221
```

Then run the following command to also update the configuration in initramfs ([details \(<https://adafruit.com/XDI>\)](https://adafruit.com/XDI)):

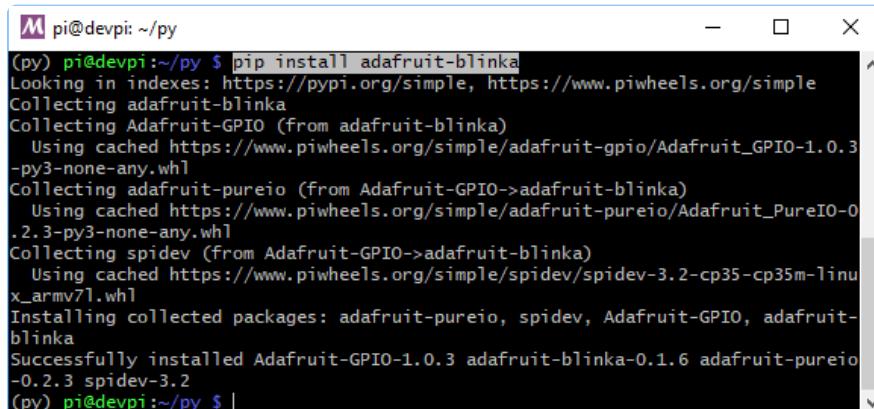
```
sudo update-initramfs -u
```

Reboot the system.

Install Blinka

To install Blinka and its dependencies, run:

```
pip3 install adafruit-blinka
```

A screenshot of a terminal window titled "pi@devpi: ~/py". It shows the command "pip3 install adafruit-blinka" being run and its output. The output includes: Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple; Collecting adafruit-blinka; Collecting Adafruit-GPIO (from adafruit-blinka); Using cached https://www.piwheels.org/simple/adafruit-gpio/Adafruit_GPIO-1.0.3-py3-none-any.whl; Collecting adafruit-pureio (from Adafruit-GPIO->adafruit-blinka); Using cached https://www.piwheels.org/simple/adafruit-pureio/Adafruit_PureIO-0.2.3-py3-none-any.whl; Collecting spidev (from Adafruit-GPIO->adafruit-blinka); Using cached https://www.piwheels.org/simple/spidev/spidev-3.2-cp35-cp35m-linux_armv7l.whl; Installing collected packages: adafruit-pureio, spidev, Adafruit-GPIO, adafruit-blinka; Successfully installed Adafruit-GPIO-1.0.3 adafruit-blinka-0.1.6 adafruit-pureio-0.2.3 spidev-3.2. The terminal prompt "(py) pi@devpi:~/py \$" is visible at the bottom.

Set environment variable

We need to manually signal to Blinka that we have a MCP2221 attached. To do this, we set the environment variable **BLINKA_MCP2221**. The value doesn't matter, just use 1:

```
export BLINKA_MCP2221=1
```

Don't forget this step. Things won't work unless BLINKA_MCP2221 is set.

Run the sanity check.

Now move on to the Post Install Checks section and run the commands there to make sure everything is installed correctly.

Post Install Checks

After going through all the install steps for your OS, run these checks as simple tests to make sure everything is installed correctly. See the rest of the page for some potential hiccups you may run into.

Go ahead and plug in your MCP2221 to a USB port on your PC.

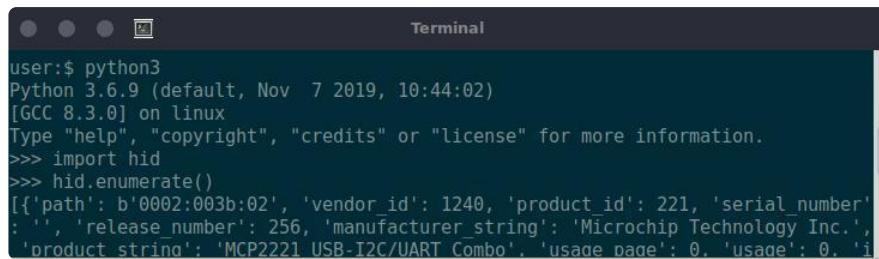
Most of these tests are done via the Python REPL, at the >>> prompt. To get there, simply launch Python:

```
$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Check that hidapi is installed correctly

At the Python REPL, type:

```
import hid
hid.enumerate()
```



```
user:$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import hid
>>> hid.enumerate()
[{'path': b'0002:003b:02', 'vendor_id': 1240, 'product_id': 221, 'serial_number': '',
 'release_number': 256, 'manufacturer_string': 'Microchip Technology Inc.',
 'product_string': 'MCP2221 USB-I2C/UART Combo', 'usage_page': 0, 'usage': 0, 'id': 1}
```

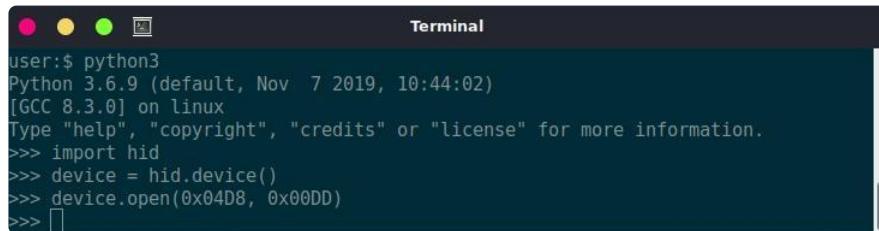
You should get a dump of everything attached to your USB ports.

Check that MCP2221 can be found

At the Python REPL, type:

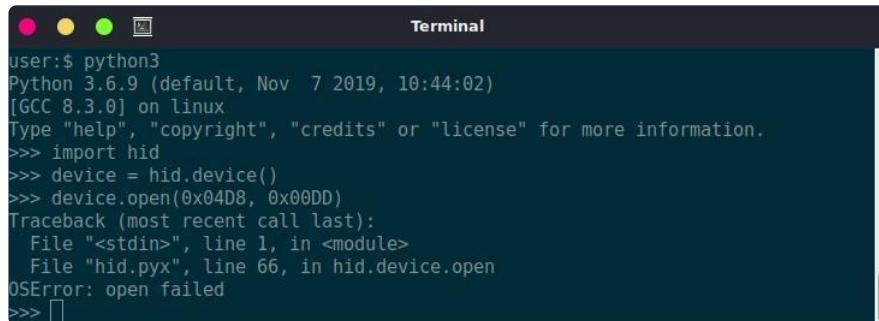
```
import hid
device = hid.device()
device.open(0x04D8, 0x00DD)
```

it should run without any errors:



```
user:$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import hid
>>> device = hid.device()
>>> device.open(0x04D8, 0x00DD)
>>> 
```

If for some reason the MCP2221 can not be found, you might see something like this:



```
user:$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import hid
>>> device = hid.device()
>>> device.open(0x04D8, 0x00DD)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "hid.pyx", line 66, in hid.device.open
OSError: open failed
>>> 
```

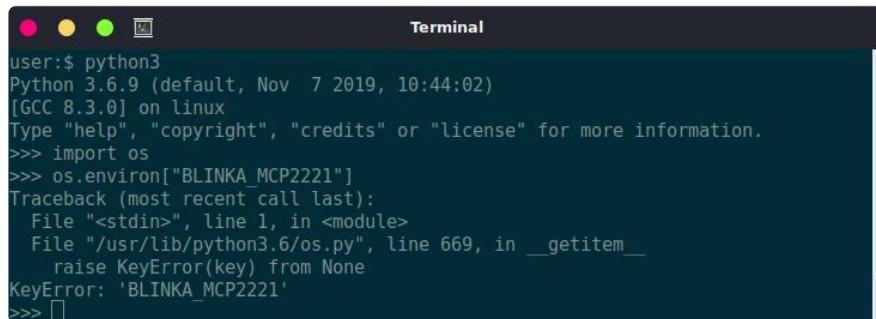
Check your USB cable connection.

Check environment variable within Python

At the Python REPL, type:

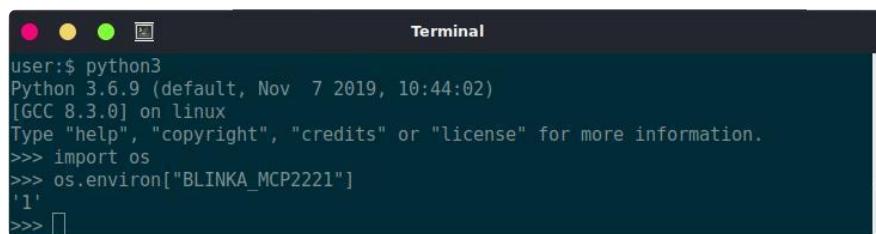
```
import os
os.environ["BLINKA_MCP2221"]
```

If you get a `KeyError` it means you did not set the environment variable right:



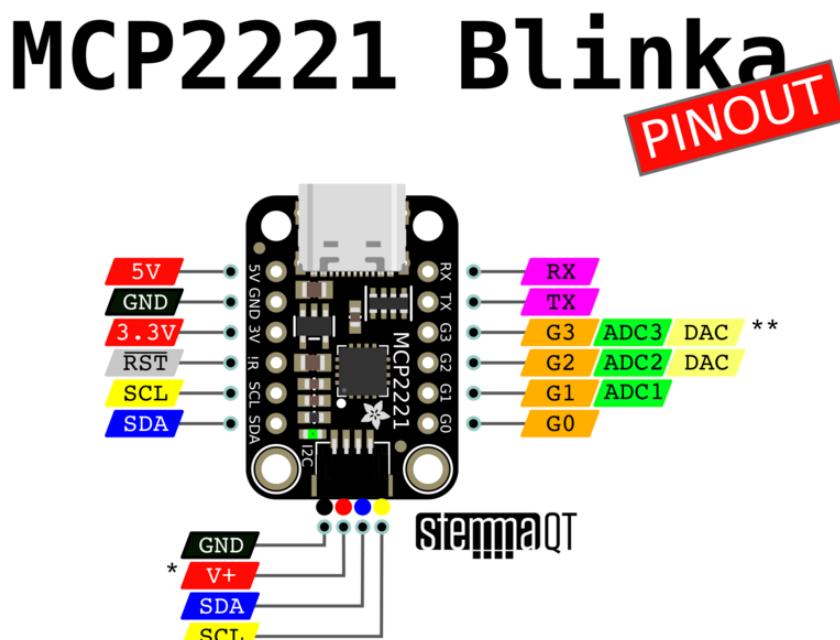
```
Terminal
user:$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.environ["BLINKA_MCP2221"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "/usr/lib/python3.6/os.py", line 669, in __getitem__
      raise KeyError(key) from None
KeyError: 'BLINKA_MCP2221'
>>> 
```

If you have set it correctly, you'll get a value back:



```
Terminal
user:$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.environ["BLINKA_MCP2221"]
'1'
>>> 
```

Pinout



* 3.3V or 5V as set by jumper

** same DAC output on both pins

The logic level is set to 3.3V by default.

Power Pins

- **5V** - this is the 5V power from the USB input.
- **3V** - this is the 3.3V power output from the voltage regulator.
- **GND** - this is the common ground for all power and logic.
- **RST** - reset pin, pulled high internally, set low to reset

GPIO Pins

- **G0 to G3** - can be used as either digital inputs or outputs, logic level is 3.3V by default but can be changed to 5V

The **GPIO pins do not have internal pull-up or pull-down support!** If you need pull resistors, add them externally.

I2C Pins

- **SCL** - the I2C clock signal, there's a 5.1K pullup resistor on this pin to whatever the logic level is (default 3.3V)
- **SDA** - the I2C data signal, there's a 5.1K pullup resistor on this pin to whatever the logic level is (default 3.3V)

UART Pins

- **TX** - transmit (out from board)
- **RX** - receive (in to board)

The uart is totally separate from the GPIO pins, it's controlled as a USB CDC device not as the USB HID interface so it shows up as a serial COM/tty port like any other USB-to-serial converter

ADC Pins

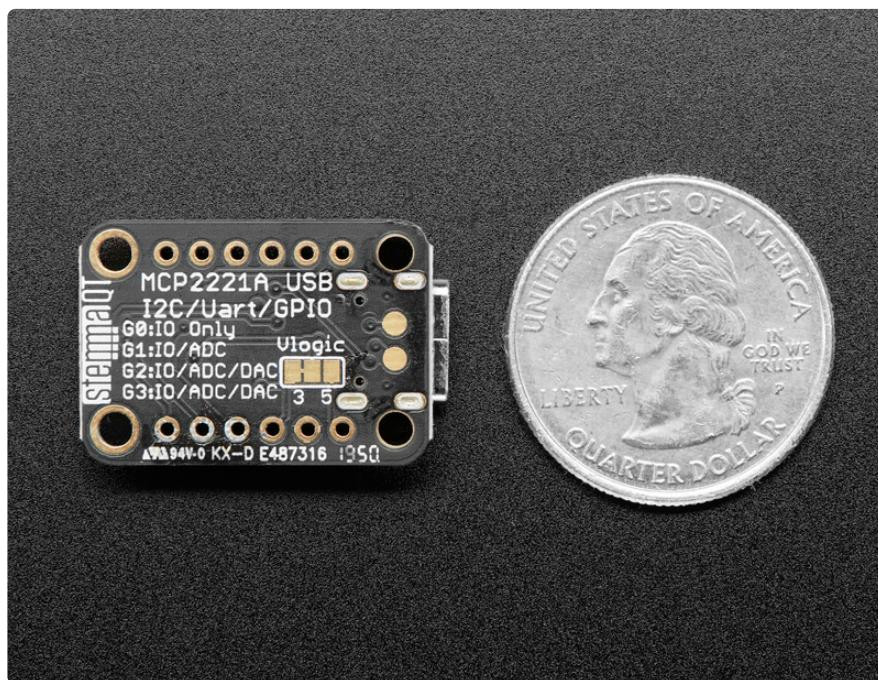
- **ADC1 to ADC3** - can be used as analog inputs (10 bit)

DAC Pins

- **DAC** - there is a single 5 bit DAC which outputs on both of these pins

Logic Level

The default logic level is 3.3V - this is for digital, analog, I2C and UART data. You can change it to 5V by cutting the jumper on the bottom from 3V and soldering it to the 5V side:



Examples

All right, now that all that annoying install stuff is done, let's have some fun.

The following sections will provide some basic examples for the main use cases - GPIO, I2C, ADC, and DAC.

Make sure you've set the `BLINKA_MCP2221` environment variable.

Installing Libraries for Breakouts

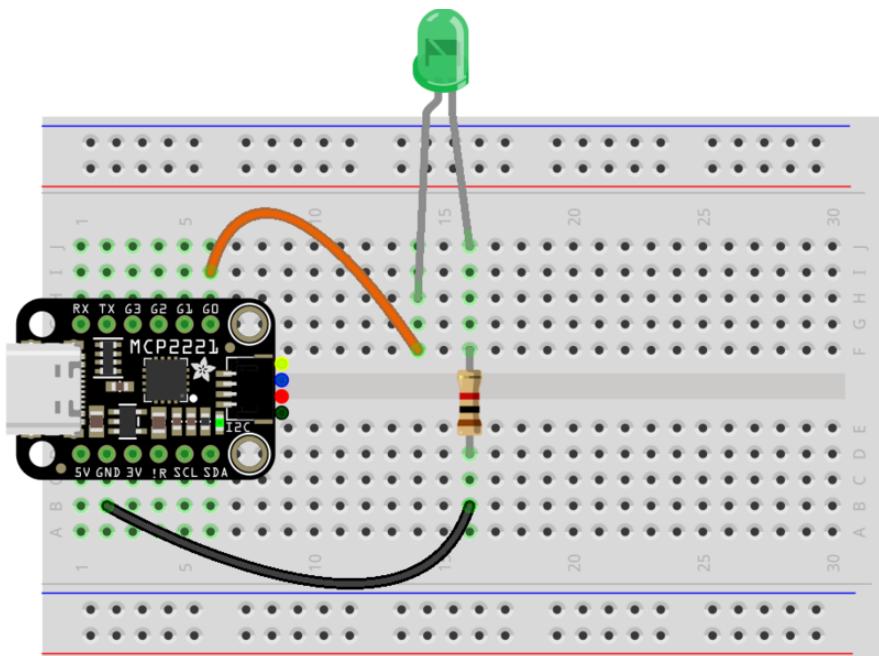
The general process for installing the CircuitPython library you are interested in will be the same as shown in the Python section of the Learn guide for your sensor. Just use `pip3`.

GPIO

Digital Output

Let's blink a LED!

Here's the bread board layout. The resistor can be something around 1kOhm. We don't need to make the LED super bright.



First, let's do things interactively so you can see how it all works one line at a time. Start by launching Python:

```
python3
```

Then, at the Python >>> prompt, enter the following to import the needed modules:

```
import board  
import digitalio
```

Next we'll create our LED digital pin and set the mode to output:

```
led = digitalio.DigitalInOut(board.G0)  
led.direction = digitalio.Direction.OUTPUT
```

And that should be it. You should be able to turn ON the LED with:

```
led.value = True
```

And turn it OFF with:

```
led.value = False
```

And here's a complete blink program you can run to make the LED blink forever.

```
import time
import board
import digitalio

led = digitalio.DigitalInOut(board.G0)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```

Save it as something like **blink.py** and then you can run it with:

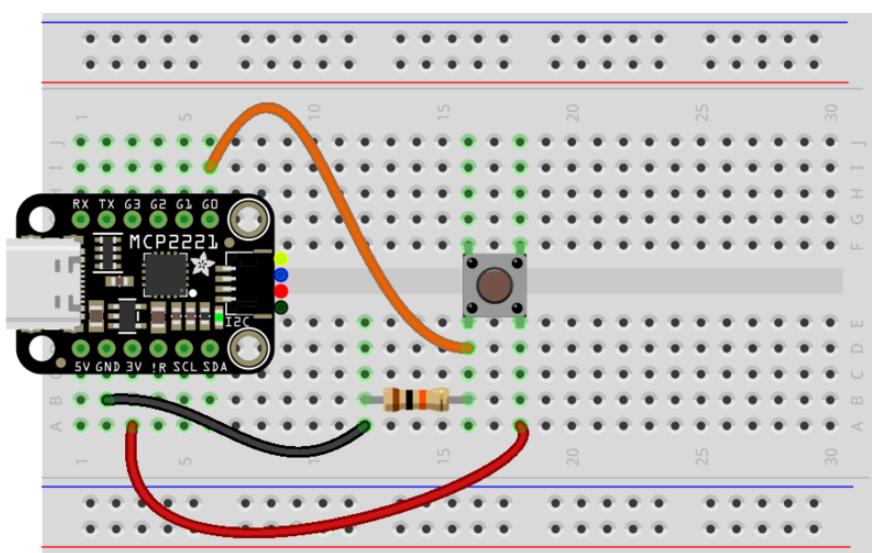
```
python3 blink.py
```

The LED should blink on and off.

Digital Input

Let's read a button!

Here's the bread board layout. Use something like a 10kOhm resistor.



The GPIO pins do not have internal pull-up or pull-down support! If you need pull resistors, add them externally as shown here!

We'll do this interactively also. So launch python:

```
python3
```

Then, at the Python >>> prompt, enter the following to import the needed modules:

```
import board  
import digitalio
```

And now we create our button digital pin and set it to input.

```
button = digitalio.DigitalInOut(board.G0)  
button.direction = digitalio.Direction.INPUT
```

And that's it. To read the current state of the button use:

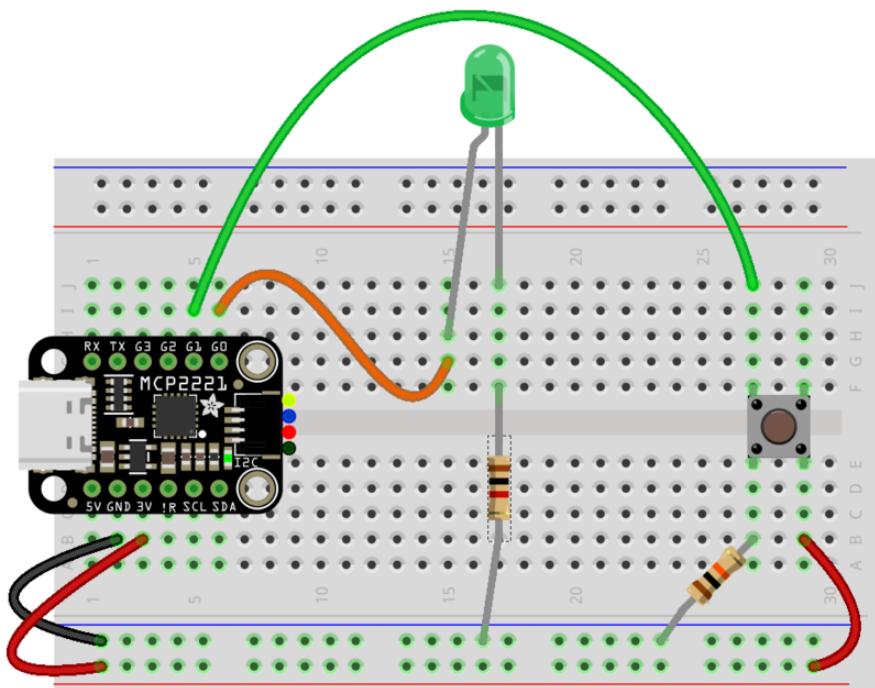
```
button.value
```

This will return **False** when the button is not pressed and **True** when it is pressed.

Digital Input and Output

Ok, let's put those two together and make the button turn on the LED. So we'll use two digital pins - one will be an input (button) and one will be an output (LED).

Here's the bread board layout.



And here's the code.

```
import board
import digitalio

led = digitalio.DigitalInOut(board.G0)
led.direction = digitalio.Direction.OUTPUT

button = digitalio.DigitalInOut(board.G1)
button.direction = digitalio.Direction.INPUT

while True:
    led.value = button.value
```

Save that to a file with a name like **button_and_led.py** and then you can run it with:

```
python3 button_and_led.py
```

and the button should turn on the LED when pressed.

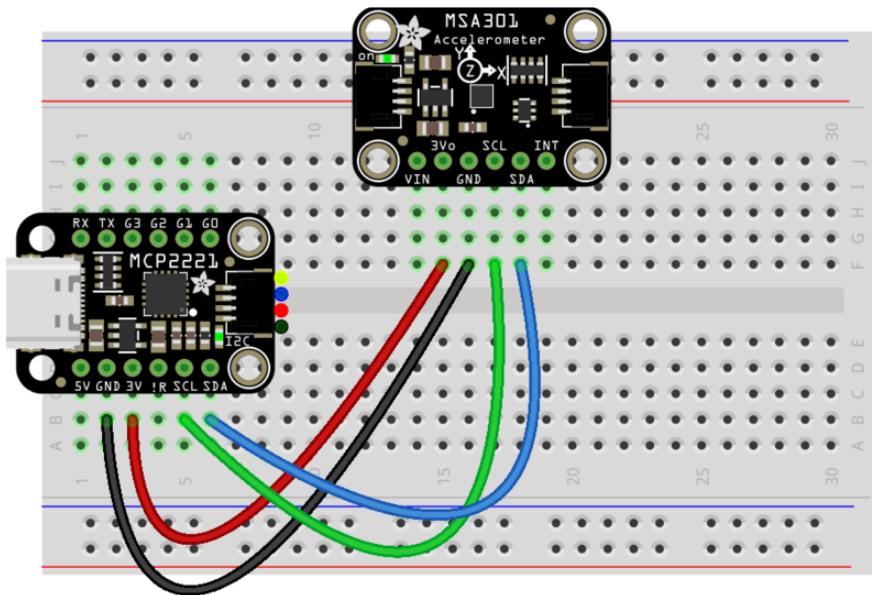
I²C

Try to avoid hot plugging I²C sensors. The MCP2221 doesn't seem to like that.
Remove USB power first.

Let's talk to an I²C sensor.

We'll use the [MSA301 sensor](http://adafru.it/4344) (<http://adafru.it/4344>) which can read acceleration. Since the MCP2221 and the MSA301 both have [STEMMA QT](https://adafru.it/Ft4) (<https://adafru.it/Ft4>) connectors, you have two options for making the connections.

You can either wire everything up on a breadboard, like this:



Or use a STEMMA QT cable:



[STEMMA QT / Qwiic JST SH 4-Pin Cable - 200mm Long](#)

This 4-wire cable is a little over 200mm / 7.8" long and fitted with JST-SH female 4-pin connectors on both ends. Compared with the chunkier JST-PH these are 1mm pitch instead of...

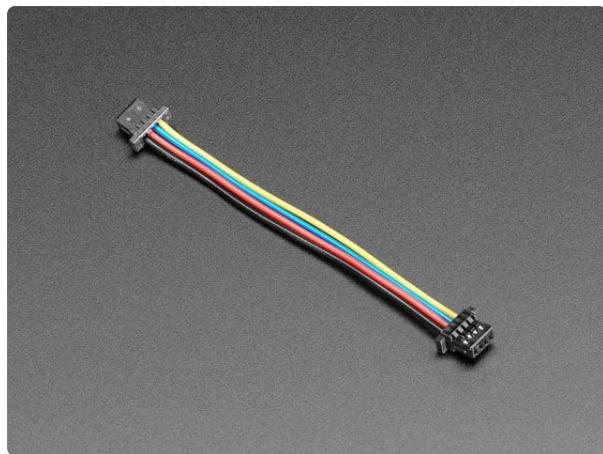
<https://www.adafruit.com/product/4401>



STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long

This 4-wire cable is a little over 100mm / 4" long and fitted with JST-SH female 4-pin connectors on both ends. Compared with the chunkier JST-PH these are 1mm pitch instead of...

<https://www.adafruit.com/product/4210>



STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long

This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...

<https://www.adafruit.com/product/4399>

Install MSA301 Library

To install the MSA301 library, run the following:

```
sudo pip3 install adafruit-circuitpython-msa301
```

Note that this step is the same as shown in the [main MSA301 guide \(https://adafru.it/HEK\)](#). You would do the same general process for any other sensor with a CircuitPython library.

Example Code

And then we can run the example from the library. Download it from here:

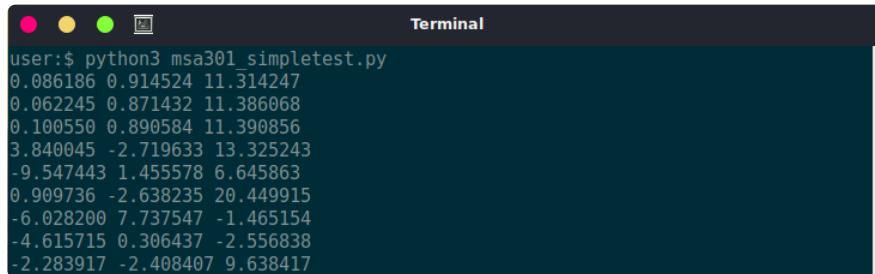
MSA301 Simple Test Example

<https://adafru.it/HEL>

save it as `msa301_simpletest.py` and run it with:

```
python3 msa301_simpletest.py
```

Pick up the board and spin it around. You should see the values change:

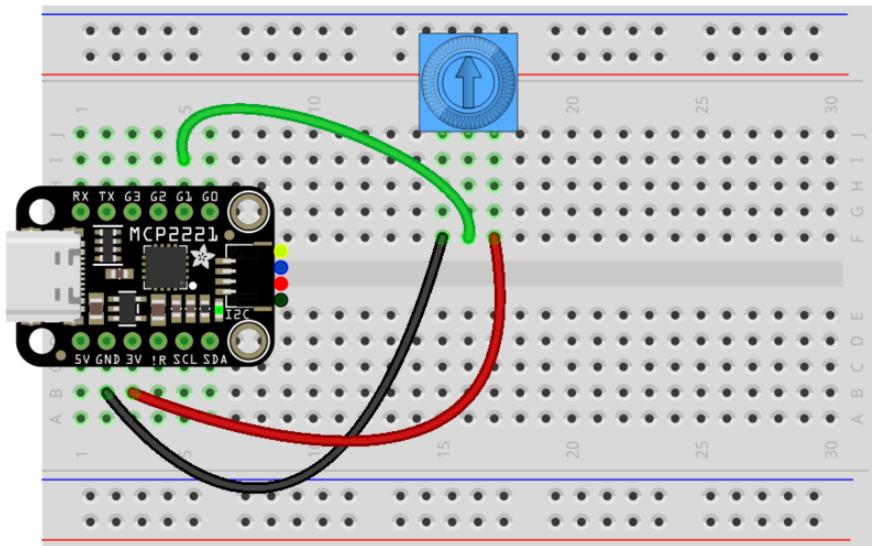


A screenshot of a terminal window titled "Terminal". The command "python3 msa301_simpletest.py" is run, and the output shows a series of floating-point numbers representing sensor data over time. The data includes both positive and negative values, such as 0.086186, 0.914524, 11.314247, 0.062245, 0.871432, 11.386068, 0.100550, 0.890584, 11.390856, 3.840045, -2.719633, 13.325243, -9.547443, 1.455578, 6.645863, 0.909736, -2.638235, 20.449915, -6.028200, 7.737547, -1.465154, -4.615715, 0.306437, -2.556838, -2.283917, -2.408407, 9.638417.

ADC

Let's read an analog signal!

For this, we'll use a small [10k trim pot](http://adafruit.it/356) (<http://adafruit.it/356>) to set up a voltage divider. Here's the wiring diagram:



And here's the code:

```
import time
import board
from analogio import AnalogIn

knob = AnalogIn(board.G1)

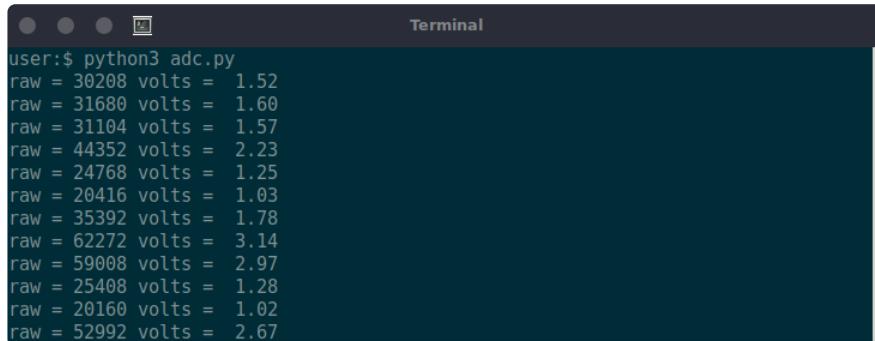
def get_voltage(raw):
    return (raw * 3.3) / 65536

while True:
    raw = knob.value
    volts = get_voltage(raw)
    print("raw = {:5d} volts = {:.5f}".format(raw, volts))
    time.sleep(0.5)
```

Save that as something like **adc.py** and then run it with:

```
python3 adc.py
```

Spin the knob and the values should change.



```
user:$ python3 adc.py
raw = 30208 volts = 1.52
raw = 31680 volts = 1.60
raw = 31104 volts = 1.57
raw = 44352 volts = 2.23
raw = 24768 volts = 1.25
raw = 20416 volts = 1.03
raw = 35392 volts = 1.78
raw = 62272 volts = 3.14
raw = 59008 volts = 2.97
raw = 25408 volts = 1.28
raw = 20160 volts = 1.02
raw = 52992 volts = 2.67
```

Note that even though the MCP2221's ADC is only 10 bits, the value is scaled to 16 bits to comply with the CircuitPython API.

DAC

Let's generate an analog signal!

Well, don't get too excited. There's only a single DAC and it's a whopping 5 bits! So don't expect super awesome high fidelity audio block rockin' beats or anything.

Here's a simple sine wave generator you can try:

```
import time
import math
import board
from analogio import AnalogOut

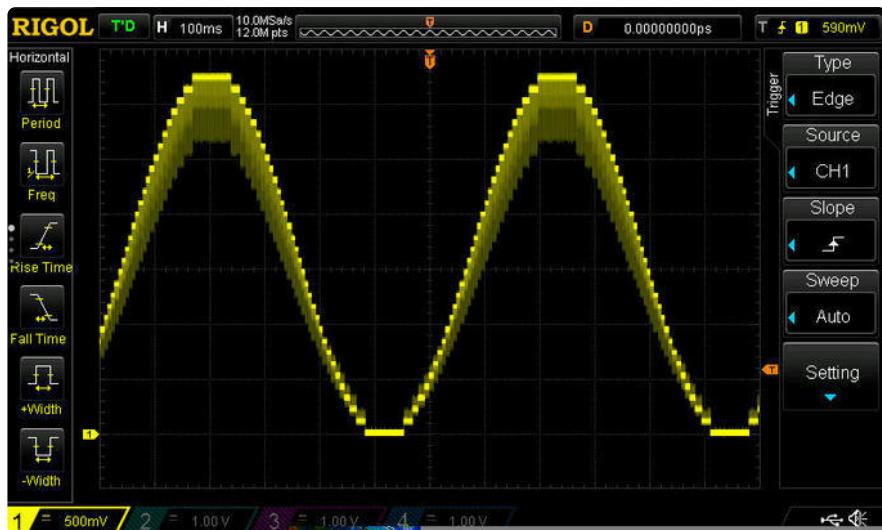
dac = AnalogOut(board.G3)

while True:
    dac.value = int((2**16 - 1) * (0.5 + 0.5 * math.sin(time.monotonic()*10)))
```

Save that as something like **dac.py** and run it with:

```
python3 dac.py
```

And connect the G3 pin to an oscilloscope. You should see something like this:



Note that even though the MCP221's DAC is only 5 bits, you set it using a 16 bit value to comply with the CircuitPython API.

UART

But wait! There's more!

The MCP2221 also provides a USB-to-UART bridge capability. When you plug in the MCP2221, it will show up as two devices:

- **HID Device** - This is what we used for GPIO/I2C/SPI/ADC/DAC
- **CDC Device** - This is how you use the UART

We've already covered all the cool stuff the HID interface provides. So it's this second item were are interested in here. It is how you can use the **TX** and **RX** pins on the MCP2221 breakout.

Install pySerial

We will use the Python library pySerial to access the MCP2221's UART. So, first install that. Here's a link to installation instructions from the libraries homepage:

pySerial Installation

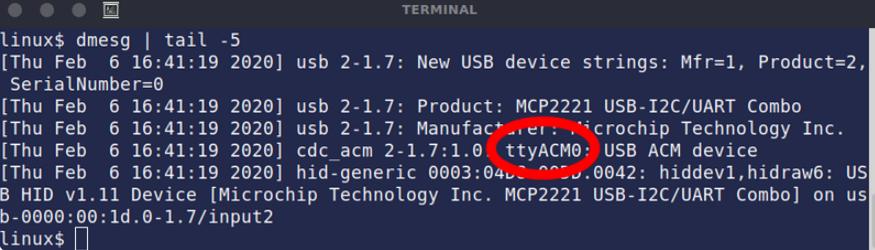
<https://adafru.it/IQA>

Find COM port in Linux

After plugging in the MCP2221, run `dmesg` and look in the output.

```
dmesg | tail
```

Like this:



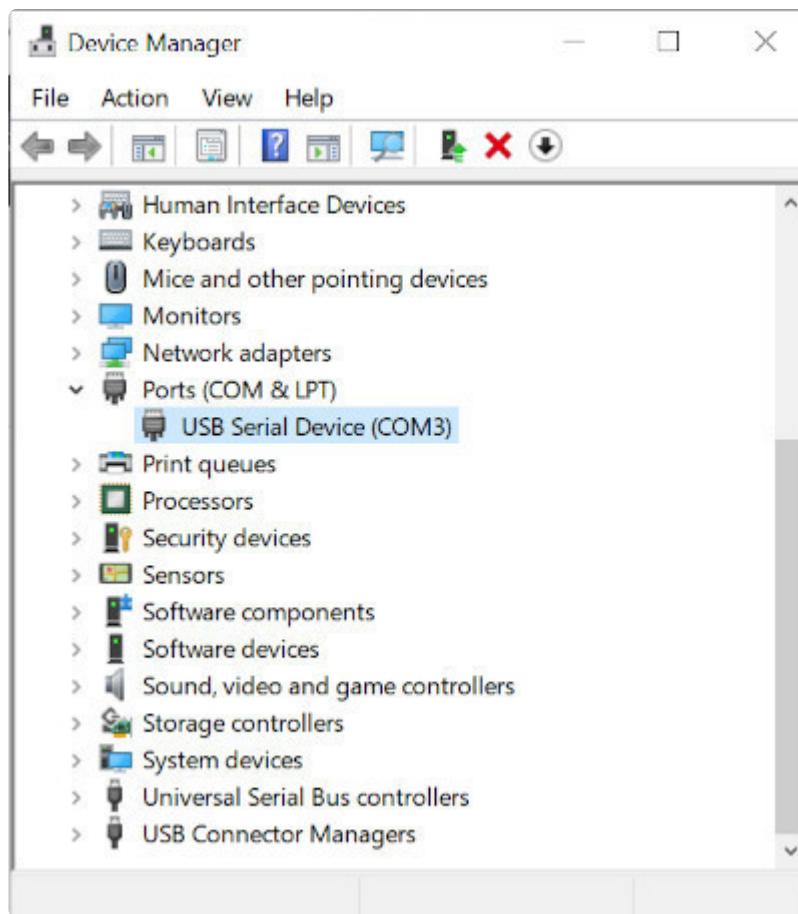
```
linux$ dmesg | tail -5
[Thu Feb  6 16:41:19 2020] usb 2-1.7: New USB device strings: Mfr=1, Product=2,
[Thu Feb  6 16:41:19 2020] usb 2-1.7: Product: MCP2221 USB-I2C/UART Combo
[Thu Feb  6 16:41:19 2020] usb 2-1.7: Manufacturer: Microchip Technology Inc.
[Thu Feb  6 16:41:19 2020] cdc_acm 2-1.7:1.0: ttyACM0: USB ACM device
[Thu Feb  6 16:41:19 2020] hid-generic 0003:045C:0000.0042: hiddev1,hidraw6: USB HID v1.11 Device [Microchip Technology Inc. MCP2221 USB-I2C/UART Combo] on us
b-0000:00:1d.0-1.7/input2
linux$
```

Look for something with a name like `ttyACMx`. An entry for that device will have been created in your `/dev` folder. So in your code, you'll want to use `/dev/ttACMx` (replace x with the number). In the example output above, the MCP2221's UART is available on `/dev/ttyACM0`. It may be slightly different on your specific linux set up.

Find COM port in Windows

After plugging in the MCP2221, look in **Device Manager** under **Ports (COM & LPT)**. If you already have a lot of entries there, it may help to also look before plugging in the MCP2221. That way you can compare and look for the new entry.

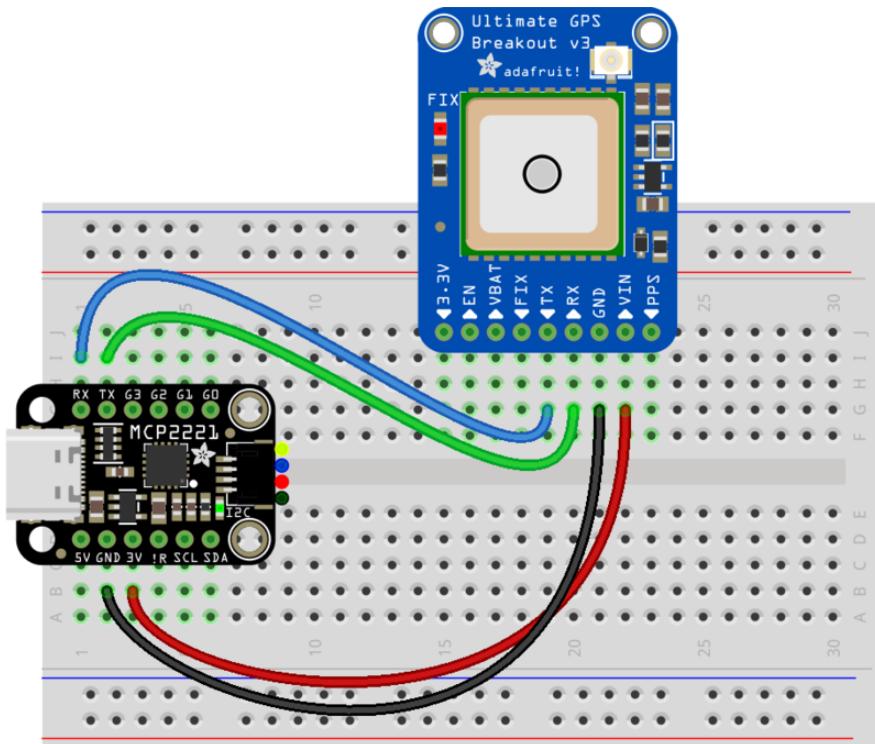
It should show up as something like this:



In the above example, it shows up as `COM3`. That's the value you'll use in your code.

UART Example with GPS

In this example we show how to use the MCP2221's UART to talk to an [Ultimate GPS Breakout Module](#) (<http://adafru.it/746>). Here is how to wire the GPS module to the MCP2221.



Once you've found your COM port and have the MCP2221 and GPS breakout wired as above, you can then follow the example here:

GPS UART Usage

<https://adafru.it/H3E>

NOTE: You will have to make some changes to the code.

You will want to follow the same general information as for the Raspberry Pi. So comment / uncomment the code as described so that you are using pySerial. Then, when you open the serial port, use the information above for your specific COM port location.

For example, on linux it will generally look the same as the Raspberry Pi example:

```
import serial
uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=10)
```

On Windows, you'll want use something like this:

```
import serial
uart = serial.Serial("COM3", baudrate=9600, timeout=10)
```

And that should be it. If you run the example, it should use the MCP2221's UART to talk to the GPS and evenutally, once it gets a fix, you should see GPS output.

FAQ & Troubleshooting

There's a few oddities when running Blinka/CircuitPython on linux. Here's a list of stuff to watch for that we know of!

This FAQ covers all the various platforms and hardware setups you can run Blinka on. Therefore, some of the information may not apply to your specific setup.

Update Blinka/Platform Libraries

Most issues can be solved by forcing Python to upgrade to the latest `blinka` / `platform-detect` libraries. Try running

```
sudo python3 -m pip install --upgrade --force-reinstall adafruit-  
blinka Adafruit-PlatformDetect
```

Getting an error message about "board" not found or "board" has no attribute

Somehow you have ended up with either the wrong `board` module or no `board` module at all.

DO NOT try to fix this by manually installing a library named `board`. There is [one out there \(<https://adafru.it/NCE>\)](#) and it has nothing to do with Blinka. You will break things if you install that library!

The easiest way to recover is to simply force a reinstall of Blinka with:

```
python3 -m pip install --upgrade --force-reinstall adafruit-blinka
```

Mixed SPI mode devices

Due to the way we share an SPI peripheral, you cannot have two SPI devices with different 'mode/polarity' on the same SPI bus - you'll get weird data

95% of SPI devices are mode 0, check the driver to see mode or polarity settings.
For example:

- [LSM9DS1 is mode 1 \(<https://adafru.it/NCF>\)](#), please use in I2C mode instead of SPI

- MAX31865 is phase 1 (<https://adafru.it/NCG>), try using this on a separate SPI device, or read data twice.
-

Why am I getting AttributeError: 'SpiDev' object has no attribute 'writebytes2'?

This is due to having an older version of [spidev](https://adafru.it/JEi) (<https://adafru.it/JEi>). You need at least version 3.4. This should have been [taken care of](https://adafru.it/NCH) (<https://adafru.it/NCH>) when you installed Blinka, but in some cases it does not seem to happen.

To check what version of spidev Python is using:

```
$ python3
Python 3.6.8 (default, Oct 7 2019, 12:59:55)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import spidev
>>> spidev.__version__
'3.4'
>>>
```

If you see a version lower than 3.4 reported, then try a force upgrade of spidev with (back at command line):

```
sudo python3 -m pip install --upgrade --force-reinstall spidev
```

No Pullup/Pulldown support on some linux boards or MCP2221

Some linux boards, for example, AllWinner-based, do not have support to set pull up or pull down on their GPIO. Use an external resistor instead!

Getting OSError: read error with MCP2221

If you are getting a stack trace that ends with something like:

```
return self._hid.read(64)
File "hid.pyx", line 122, in hid.device.read
OSError: read error
```

Try setting an environment variable named **BLINKA_MCP2221_RESET_DELAY** to a value of **0.5 or higher**.

Windows:

```
set BLINKA_MCP2221_RESET_DELAY=0.5
```

Linux:

```
export BLINKA_MCP2221_RESET_DELAY=0.5
```

This is a value in seconds to wait between resetting the MCP2221 and the attempt to reopen it. The reset is seen by the operating system as a hardware disconnect/reconnect. Different operating systems can need different amounts of time to wait after the reconnect before the attempt to reopen. Setting the above environment variable will override the default reset delay time, allowing it to be increased as needed for different setups.

Using FT232H with other FTDI devices.

Blinka uses the libusbk driver to talk to the FT232H directly. If you have other FTDI devices installed that are using the FTDI VCP drivers, you may run into issues. See here for a possible workaround:

<https://forums.adafruit.com/viewtopic.php?f=19&t=166999> (<https://adafru.it/doW>)

Getting "no backend available" with pyusb on Windows

This is probably only an issue for older versions of Windows. If you run into something like this, see this issue thread:

<https://github.com/pyusb/pyusb/issues/120> (<https://adafru.it/Uao>)

which describes copying the 32bit and 64bit DLLs into specific folders. ([example for Win7](#) (<https://adafru.it/Uao>))

Getting "no backend available" or other problems with pyusb on Mac

Check out this issue thread:

<https://github.com/pyusb/pyusb/issues/355> (<https://adafru.it/19fh>)

which has lots of discussion. It is probably worth reading through it all to determine what applies for your setup. Most solutions seem to rely on setting the **DYLD_LIBRARY_PATH** environment variable.

This issue thread has further information:

<https://github.com/orgs/Homebrew/discussions/3424> (<https://adafru.it/19fi>)

I can't get neopixel, analogio, audioio, rotaryio, displayio or pulseio to work!

Some CircuitPython modules like may not be supported.

- Most SBCs do not have analog inputs so there is no `analogio`
- Few SBCs have `neopixel` support so that is only available on Raspberry Pi (and any others that have low level neopixel protocol writing)
- Rotary encoders (`rotaryio`) is handled by interrupts on microcontrollers, and is not supported on SBCs at this time
- Likewise `pulseio` PWM support is not supported on many SBCs, and if it is, it will not support a carrier wave (Infrared transmission)
- For display usage, we suggest using python `Pillow` library or `Pygame`, we do not have `displayio` support

We aim to have, at a minimum, `digitalio` and `busio` (I2C/SPI). This lets you use the vast number of driver libraries

For analog inputs, [the MCP3xxx library](https://adafru.it/CPN) (<https://adafru.it/CPN>) will give you `AnalogIn` objects. For PWM outputs, try [the PCA9685](https://adafru.it/tZF) (<https://adafru.it/tZF>). For audio, use pygame or other Python3 libraries to play audio.

Some libraries, like [Adafruit_CircuitPython_DHT](https://adafru.it/Beq) (<https://adafru.it/Beq>) will try to bit-bang if pulsein isn't available. Slow linux boards (<700MHz) may not be able to read the pins fast enough), you'll just have to try!

Help, I'm getting the message "error while loading shared libraries: libgpiod.so.2: cannot open shared object file: No such file or directory"

It looks like **libgpiod** may not be installed on your board.

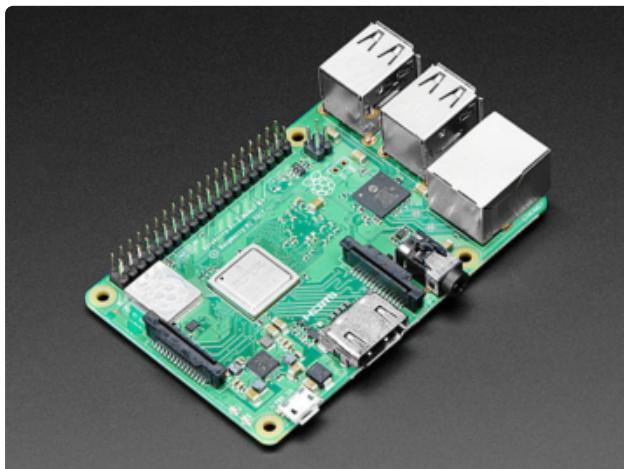
Try running the command: `sudo apt-get install libgpiod2`

= v5.5.0""> When running the libgpiod script, I see the message: configure: error: "libgpiod needs linux headers version >= v5.5.0"

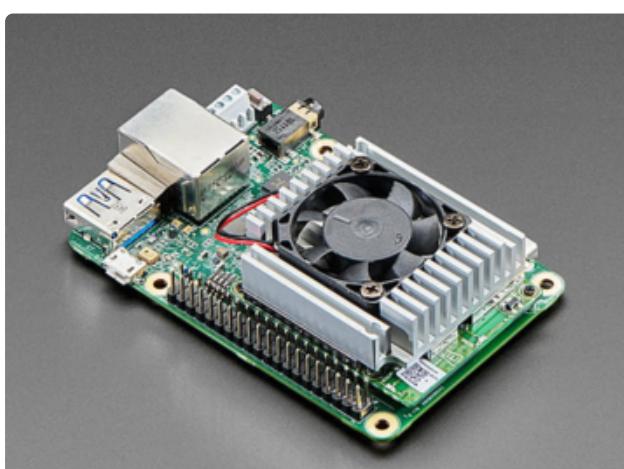
Be sure you have the latest libgpiod.py script and run it with the `-l` or `--legacy` flag:

`sudo python3 libgpiod.py --legacy`

All Raspberry Pi Computers Have:

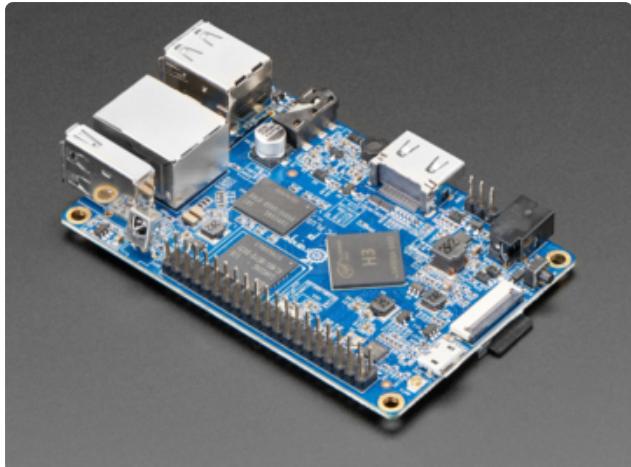


1 x I2C port with **busio** (but clock stretching is not supported in hardware, so you must set the I2C bus speed to 10KHz to 'fix it')
2 x SPI ports with **busio**
1 x UART port with **serial** - note this is shared with the hardware console
pulseio.pulseIn using gpiod
neopixel support on a few pins
No AnalogIn support (Use an MCP3008 or similar to add ADC)
No PWM support (Use a PCA9685 or similar to add PWM)



Google Coral TPU Dev Boards Have:

1 x I2C port with **busio**
1 x SPI ports with **busio**
1 x UART port with **serial** - note this is shared with the hardware console
3 x PWMOut support
No NeoPixel support
No AnalogIn support (Use an MCP3008 or similar to add ADC)



Orange Pi PC Plus Boards Have:

- 1 x I2C port with **busio**
- 1 x SPI ports with **busio**
- 1 x UART port with **serial**
- No NeoPixel support
- No AnalogIn support (Use an MCP3008 or similar to add ADC)
- No PWM support (Use a PCA9685 or similar to add PWM)



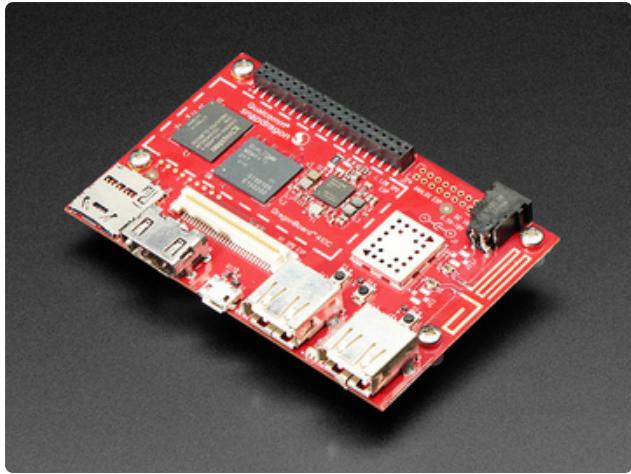
Orange Pi R1 Boards Have:

- 1 x I2C port with **busio**
- 1 x SPI port with **busio**
- 1 x UART port with **serial**
- No NeoPixel support
- No AnalogIn support (Use an MCP3008 or similar to add ADC)
- No PWM support (Use a PCA9685 or similar to add PWM)



ODROID C2 Boards Have:

- 1 x I2C port with **busio**
- No SPI support
- 1 x UART port with **serial** - note this is shared with the hardware console
- No NeoPixel support
- No AnalogIn support (Use an MCP3008 or similar to add ADC)
- No PWM support (Use a PCA9685 or similar to add PWM)



DragonBoard 410c Boards Have:

- 2 x I2C port with **busio**
- 1 x SPI port with **busio**
- 1 x UART port with **serial**
- No NeoPixel support
- No AnalogIn support (Use an MCP3008 or similar to add ADC)
- No PWM support (Use a PCA9685 or similar to add PWM)



NVIDIA Jetson Nano Boards Have:

- 2 x I2C port with **busio**
- 2 x SPI ports with **busio**
- 2 x UART port with **serial** - note one of these is shared with the hardware console
- No NeoPixel support
- No AnalogIn support (Use an MCP3008 or similar to add ADC)
- No PWM support (Use a PCA9685 or similar to add PWM)

FT232H Breakouts Have:

1x I2C port OR SPI port with **busio**
12x GPIO pins with **digitalio**
No UART
No AnalogIn support
No AnalogOut support
No PWM support
If you are using [Blinka in FT232H mode \(<https://adafru.it/FWD>\)](#), then keep in mind these basic limitations.



SPI and I2C can not be used at the same time since they share the same pins.
GPIO speed is not super fast, so trying to do arbitrary bit bang like things may run into speed issues.
There are no ADCs.
There are no DACs.
UART is not available (its a different FTDI mode)

MCP2221 Breakouts Have:

1x I2C port with **busio**
4x GPIO pins with **digitalio**
3x AnalogIn with **analogio**
1x AnalogOut with **analogio**
1x UART with **pyserial**
No PWM support
No hardware SPI support
If you are using Blinka in MCP2221 mode, then keep in mind these basic limitations.



GPIO speed is not super fast, so trying to do arbitrary bit bang like things may run into speed issues.
UART is available via **pyserial**, the serial COM port shows up as a second USB device during enumeration

Downloads

Files:

- [MCP2221 Datasheet \(https://adafru.it/L2C\)](https://adafru.it/L2C)
- [EagleCAD files on GitHub \(https://adafru.it/HFA\)](https://adafru.it/HFA)
- [Fritzing object in Adafruit Fritzing Library \(https://adafru.it/HFB\)](https://adafru.it/HFB)

Schematic and Fab Print

